Small-Group Learning, with Application to Neural Architecture Search

Xuefeng Du XUEFENGDU1@GMAIL.COM
Pengtao Xie* P1XIE@ENG.UCSD.EDU
University of California San Diego

arXiv:2012.12502v2 [cs.LG] 11 Mar 2021

Abstract

In human learning, an effective learning methodology is small-group learning: a small group of students work together towards the same learning objective, where they express their understanding of a topic to their peers, compare their ideas, and help each other to trouble-shoot problems. In this paper, we aim to investigate whether this human learning method can be borrowed to train better machine learning models, by developing a novel ML framework – small-group learning (SGL). In our framework, a group of learners (ML models) with different model architectures collaboratively help each other to learn by leveraging their complementary advantages. Specifically, each learner uses its intermediately trained model to generate a pseudo-labeled dataset and re-trains its model using pseudo-labeled datasets generated by other learners. SGL is formulated as a multi-level optimization framework consisting of three learning stages: each learner trains a model independently and uses this model to perform pseudo-labeling; each learner trains another model using datasets pseudo-labeled by other learners; learners improve their architectures by minimizing validation losses. An efficient algorithm is developed to solve the multi-level optimization problem. We apply SGL for neural architecture search. Results on CIFAR-100, CIFAR-10, and ImageNet demonstrate the effectiveness of our method.

1. Introduction

Small-group learning is a broadly practiced learning methodology in humans' learning activities with high efficacy. In SGL, a small group of students collaborate with each other to study the same topic. Each student conveys his/her understanding of this topic to others and deepens the understanding by referring to others' thoughts. SGL can effectively facilitate more profound and meaningful learning.

We are interested in asking whether this human learning skill can be translated into a machine learning skill to train better ML models. In a machine learning task (e.g., multi-class classification), it is typically the case that models with different architectures have complementary advantages. On a certain class c, model A (e.g., ResNet (He et al., 2016b)) performs better than model B (e.g., DenseNet (Huang et al., 2017)). On another class d, model B performs better than model A. It is desirable to let these two models help each other during learning so that both of them eventually achieve great performance on both classes: specifically, model A helps model B to improve its performance on class c and model B helps model A to improve its performance on class d. Based on this idea, we propose a

. *Corresponding author.

© X. Du & P. Xie*.

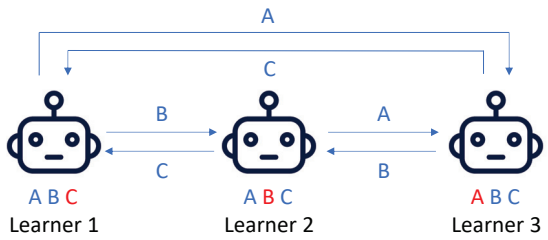


Figure 1: Illustration of small-group learning. There are three learners performing a classification task with three classes: A, B, and C. Blue letters denote classes that learners are good at (e.g., learner 1 performs well on class A and B) and red letters denote classes that learners are not good at. For each class, strong learners help weak learners to learn. For example, learner 2 and 3 are strong learners on class C; they help learner 1 (which is a weak learner on C) to improve the performance on C.

novel learning framework – small-group learning (SGL) (as illustrated in Figure 1). SGL involves a set of ML learners which solve the same learning task T. Without loss of generality, we assume T is classification while noting that our framework can be broadly applied to other tasks as well. Each learner k has a learnable neural architecture A_k and two sets of network weights W_k and V_k. The architectures and network weights of different learners are different. All learners share the same training dataset and validation dataset. These learners help each other to learn, in the following way: given an unlabeled dataset, each learner uses its intermediately trained model (including architecture and network weights) to generate pseudo-labels on this dataset and each learner leverages the pseudo-labeled datasets generated by other peers to re-train its model. On each class c, models performing well on c can generate high-quality pseudo-labeled datasets w.r.t c. Trained on these datasets, models that originally do not perform well on c can improve their performance on c. In this framework, there are three learning stages. In the first stage, each learner k trains its network weights V_k on the training dataset, with its architecture A_k fixed. In the second stage, each learner k uses its optimally trained V_k^* in the first stage to make predictions on an unlabeled dataset and generates a pseudo-labeled dataset; then each learner k trains its second set of network weights W_k using the pseudo-labeled datasets generated by other learners and using a human-labeled training dataset. In the third stage, each learner k updates its architecture A_k by minimizing prediction losses on the validation dataset. A multi-level optimization framework is developed to organize the three learning stages and enable them to be performed in a joint manner. Our method is applied for neural architecture search tasks on CIFAR-100, CIFAR-10, and ImageNet (Deng et al., 2009). Experimental results demonstrate the broad effectiveness of SGL.

The major contributions of this paper are:

- Drawing inspiration from the small-group learning (SGL) methodology in human learning, we develop a novel ML framework which formalizes SGL into a machine learning skill. In our framework, each learner uses its intermediately trained model

to generate a pseudo-labeled dataset and re-trains its model using pseudo-labeled datasets generated by other learners.

- To formalize SGL, we develop a multi-level optimization (MLO) framework consisting of three learning stages: learners learn independently; learners learn collaboratively; learners validate themselves.
- An efficient optimization algorithm is developed to solve the MLO problem.
- We apply SGL for neural architecture search. The effectiveness of SGL is demonstrated by the experimental results on CIFAR-100, CIFAR-10, and ImageNet.

2. Related Works

Neural Architecture Search (NAS). NAS aims to identify highly-performing architectures of deep neural networks automatically instead of manually designing them by humans. Various approaches have been proposed for NAS, including differentiable search methods (Cai et al., 2019; Liu et al., 2019; Xie et al., 2019) and those based on reinforcement learning (Zoph and Le, 2017; Pham et al., 2018; Zoph et al., 2018) and evolutionary algorithms (Liu et al., 2018b; Real et al., 2019). In RL-based approaches, a policy is learned to iteratively generate new architectures by maximizing a reward which is the accuracy on the validation set. Evolutionary algorithm based approaches represent architectures as individuals in a population. Individuals with high fitness scores (validation accuracy) have the privilege to generate offspring, which replaces individuals with low fitness scores. Differentiable approaches adopt a network pruning strategy. On top of an over-parameterized network, importance weights of building blocks are learned using gradient descent. After learning, blocks whose weights are close to zero are pruned. There have been many efforts devoted to improving differentiable NAS methods. In P-DARTS (Chen et al., 2019), the depth of searched architectures is allowed to grow progressively during the training process. Search space approximation and regularization approaches are developed to reduce computational overheads and improve search stability. PC-DARTS (Xu et al., 2020) reduces the redundancy in exploring the search space by sampling a small portion of a super network. Operation search is performed in a subset of channels with the held-out part bypassed in a shortcut. Our proposed SGL framework is applicable to any differentiable search approach.

Pseudo Labeling. Pseudo labeling has been broadly applied to different applications including knowledge distillation (Hinton et al., 2015), adversarial robustness (Carlini and Wagner, 2017), self-supervised learning (Xie et al., 2020), etc. Zhang et al. (2018b) proposed a deep mutual learning (DML) approach, which applies pseudo labeling for mutually learning multiple learners. Our proposed method differs from this one in two aspects. First, our method proposes a multi-level optimization framework to perform pretraining, pseudo-labeling, and finetuning (based on pseudo-labels) jointly end-to-end, where the models are pretrained before they are used for generating pseudo labels. In contrast, DML directly uses untrained models to perform pseudo-labeling, which may incur collective failures as discussed in Section 4.4. Second, our method focuses on searching neural architectures while in DML the architectures are manually designed by humans. In several works (Gu and Tresp, 2020; Li et al., 2020; Trofimov et al., 2020), a trained teacher network (with

Table 1: Notations in small-group learning

Notation	Meaning
A_k	Architecture of learner k
V_k	The first set of weights of learner k
W_k	The second set of weights of learner k
$D^{(\text{tr})}$	Training dataset
$D^{(\text{val})}$	Validation dataset
$D^{(\text{u})}$	Unlabeled dataset

a fixed architecture) is leveraged to generate pseudo-labels, which are used to search the architecture of a student network. Such et al. (2019) proposed a meta-learning approach to learn a generative model which generates synthetic data and uses the generated data to search neural architectures. In these works, pseudo-labeling is unidirectional: a network A with a fixed architecture generates pseudo-labels which are used to search the architecture of B; B does not generate pseudo-labels to search the architecture of A. Different from these works, in our work pseudo-labeling is bi-directional: each model generates pseudo-labels which are used to search the architectures of other models; meanwhile, each model uses pseudo-labels generated by other models to search its own architecture.

Ensemble Learning. Our work is also related to ensemble learning, such as boosting (Freund and Schapire, 1997), bagging (Breiman, 2001), etc. In ensemble learning, a set of models are learned to perform the same task. During training, these models do not necessarily collaborate with each other. During testing, the trained models work together to make a single prediction. The individual models in ensemble learning are preferred to be different from each other so that their comparative advantages can be combined together when making predictions. Our work differs from ensemble learning in that the learners in our method help with each other during training; after training, different models achieve similar performance as a result of mutual teaching; during testing, only the single best model is retained for making predictions. Our method is more computationally efficient and memory efficient than ensemble learning methods during testing time since a single model is used for making predictions while ensemble learning utilizes all models to perform predictions.

Cooperative Learning. Previously, cooperative learning, where multiple models collaborate to perform a task, has been investigated in multi-agent learning (Panait and Luke, 2005), co-training (Blum and Mitchell, 1998), federated learning (Bonawitz et al., 2019), etc. Our method differs from these existing methods in that we focus on collaborative neural architecture search via mutual pseudo-labeling.

3. Methods

In this section, we propose a small-group learning (SGL) framework and develop an optimization algorithm.

3.1. Small-Group Learning

In our framework, there are a set of K learners, all of which learn to solve the same target task. Without loss of generality, we assume the task is classification. Each learner k has a learnable architecture A_k and two sets of learnable network weights V_k and W_k. All learners share the same training dataset $D^{(\text{tr})}$, the same validation dataset $D^{(\text{val})}$, and an unlabeled dataset $D^{(\text{u})}$. The K learners perform learning in three stages. In the first stage, each learner trains a preliminary model. In the second stage, each learner uses its model trained in the first stage to perform pseudo-labeling. Then each learner re-trains its model using pseudo-labeled datasets generated by other learners. In the third stage, each learner measures the validation performance of its model trained in the second stage and updates its architecture to improve the validation performance. We discuss the details in the sequel. In the first stage, each learner k trains its weights V_k, with its architecture A_k fixed:

$$V_k^*(A_k) = \min_{V_k} L(V_k, A_k, D^{(\text{tr})}). \tag{1}$$

The architecture A_k is used to make predictions on training examples and define the training loss. However, A_k should not be optimized to minimize the training loss. Otherwise, the training loss can be easily minimized to zero by making A_k a very large (hence highly expressive) architecture, which will overfit the training data and have poor performance on unseen data. The optimally trained weights $V_k^*(A_k)$ is a function of A_k since $V_k^*(A_k)$ is a function of $L(V_k, A_k, D^{(\text{tr})})$ and $L(V_k, A_k, D^{(\text{tr})})$ is a function of A_k.

In the second stage, each learner k uses $V_k^*(A_k)$ learned in the first stage to make predictions on an unlabeled dataset $D^{(\text{u})} = \{x_i\}_{i=1}^N$ and generates a pseudo-labeled dataset $D_k^{(\text{pl})}(D^{(\text{u})}, V_k^*(A_k)) = \{(x_i, f(x_i; V_k^*(A_k)))\}_{i=1}^N$, where $f(\cdot; V_k^*(A_k))$ is the network parameterized by $V_k^*(A_k)$. $f(x_i; V_k^*(A_k))$ is a J-dimensional vector where J is the number of classes and the j-th element of $f(x_i; V_k^*(A_k))$ indicates the probability that x_i belongs to the j-th class. The sum of all elements in $f(x_i; V_k^*(A_k))$ is one. Meanwhile, each learner k uses pseudo-labeled datasets $\{D_j^{(\text{pl})}\}_{j\neq k}^K$ produced by other learners as well as a human-labeled dataset $D^{(\text{tr})}$ to train its other set of network weights W_k:

$$W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K) = \min_{W_k} L(W_k, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j^*(A_j))),$$

where the first loss term $L(W_k, A_k, D^{(\text{tr})})$ in the objective is defined on the human-labeled training dataset and the second loss term is defined on the pseudo-labeled datasets produced by other learners. Both losses are cross-entropy losses. λ is a tradeoff parameter. Due to the same reason mentioned above, A_k should not be updated at this stage either. Note that $W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K)$ is a function of A_k and $\{V_j^*(A_j)\}_{j\neq k}^K$ since $W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K)$ is a function of $L(W_k, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j^*(A_j)))$ which is a function of A_k and $\{V_j^*(A_j)\}_{j\neq k}^K$. In the third stage, each learner validates its $W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K)$ on the validation set $D^{(\text{val})}$. The learners optimize their architectures by minimizing the validation losses:

$$\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K), A_k, D^{(\text{val})}). \tag{2}$$

Performed in a unified framework, the three stages mutually influence each other: $\{V_k^*(A_k)\}_{k=1}^K$ trained in the first stage are leveraged to calculate the training loss in the second stage; $\{W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K)\}_{k=1}^K$ trained in the second stage are leveraged to calculate the validation loss in the third stage; after $\{A_k\}_{k=1}^K$ are optimized in the third stage, they will render the training loss in the first stage to be changed, which changes $\{V_k^*(A_k)\}_{k=1}^K$ in the second stage accordingly. Figure 2 illustrates the three stages. To this end, we formulate SGL as a three-level optimization problem:

$$\begin{aligned}
&\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K), A_k, D^{(\text{val})}) \\
&s.t. \quad \{W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K)\}_{k=1}^K = \\
&\qquad \min_{\{W_k\}_{k=1}^K} \sum_{k=1}^K L(W_k, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j^*(A_j))) \\
&\qquad \{V_k^*(A_k)\}_{k=1}^K = \min_{\{V_k\}_{k=1}^K} \sum_{k=1}^K L(V_k, A_k, D^{(\text{tr})})
\end{aligned} \tag{3}$$

This formulation consists of three optimization problems including two inner-optimization problems and one outer-optimization problem. The two inner-optimization problems are on the constraints of the outer optimization problem. From bottom to top, the three optimization problems correspond to the first, second, and third learning stage respectively. Inspired by (Liu et al., 2019), we parameterize the architecture A using continuous variables which allow the search to be conducted using efficient gradient-based methods. The search space of A is overparameterized with many building blocks. For each block, a continuous variable is used to represent whether this block should be selected to form the final architecture. Search amounts to learning these variables.

3.2. Optimization Algorithm

To solve the SGL problem, we develop an efficient gradient-based optimization algorithm, drawing inspirations from (Liu et al., 2019). In the following, $\nabla^2_{Y,X} f(X, Y)$ denotes $\frac{\partial f(X,Y)}{\partial X \partial Y}$. First of all, we approximate $V_k^*(A_k)$ using

$$V_k' = V_k - \xi_V \nabla_{V_k} L(V_k, A_k, D^{(\text{tr})}), \tag{4}$$

where ξ_V is a learning rate. Plugging $\{V_j'\}_{j=1}^K$ into $\sum_{k=1}^K (L(W_k, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j^*(A_j))))$, we obtain an approximated objective $\sum_{k=1}^K O_k^W$ where $O_k^W = L(W_k, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j'))$. Then we approximate $W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K)$ using one-step gradient descent update of W_k w.r.t O_k^W:

$$W_k' = W_k - \xi_W \nabla_{W_k} (L(W_k, A_k, D^{(\text{tr})}) + \lambda \textstyle\sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j'))). \tag{5}$$

Finally, we plug $\{W_k'\}_{k=1}^K$ into $\sum_{k=1}^K L(W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K), A_k, D^{(\text{val})})$ and get $\sum_{k=1}^K L(W_k', A_k, D^{(\text{val})})$. We can update the architecture A_k of learner k by descending the gradient of $\sum_{j=1}^K L(W_j', A_j, D^{(\text{val})})$ w.r.t A_k:

$$A_k \leftarrow A_k - \eta_A (\nabla_{A_k} L(W_k', A_k, D^{(\text{val})}) + \sum_{j\neq k}^K \nabla_{A_k} L(W_j', A_j, D^{(\text{val})})). \tag{6}$$

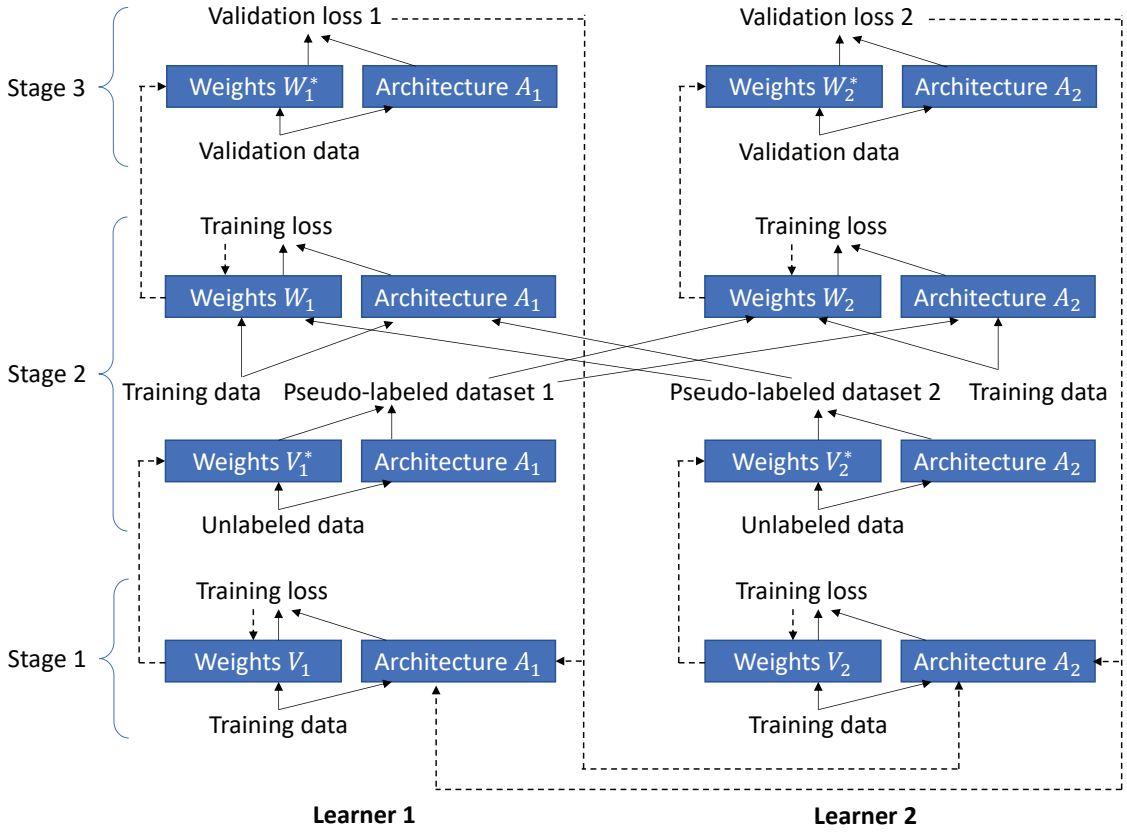


Figure 2: Three learning stages in small-group learning. Solid arrows denote the forward pass where predictions are made and losses are calculated. Dotted arrows represent the backward pass of calculating gradients of model architectures and weights and performing gradient-based updates. For simplicity, we assume there are two learners in the group. Extension to multiple learners is straightforward.

where

$$\begin{aligned}
&\nabla_{A_k} L(W_k', A_k, D^{(\mathrm{val})}) = \\
&\nabla_{A_k} L(W_k - \xi_W \nabla_{W_k}(L(W_k, A_k, D^{(\mathrm{tr})}) + \lambda \textstyle\sum_{j\neq k}^{K} L(W_k, A_k, D_j^{(\mathrm{pl})}(D^{(\mathrm{u})}, V_j'))), A_k, D^{(\mathrm{val})}) = \\
&\nabla_{A_k} L(W_k', A_k, D^{(\mathrm{val})}) - \xi_W \nabla^2_{A_k, W_k}(L(W_k, A_k, D^{(\mathrm{tr})}) + \\
&\lambda \textstyle\sum_{j\neq k}^{K} L(W_k, A_k, D_j^{(\mathrm{pl})}(D^{(\mathrm{u})}, V_j'))) \nabla_{W_k'} L(W_k', A_k, D^{(\mathrm{val})}).
\end{aligned} \tag{7}$$

Calculating matrix-vector product $\nabla^2_{A_k, W_k}(L(W_k, A_k, D^{(\mathrm{tr})}) + \lambda \sum_{j\neq k}^{K} L(W_k, A_k, D_j^{(\mathrm{pl})}(D^{(\mathrm{u})}, V_j'))) \nabla_{W_k'} L(W_k', A_k, D^{(\mathrm{val})})$ incurs a lot of computational cost. This cost can be reduced

by a finite difference approximation (Liu et al., 2019):

$$\begin{aligned}
&\nabla^2_{A_k,W_k}(L(W_k,A_k,D^{(\mathrm{tr})})+\lambda\textstyle\sum_{j\neq k}^{K}L(W_k,A_k,D_j^{(\mathrm{pl})}(D^{(\mathrm{u})},V_j')))\nabla_{W_k'}L(W_k',A_k,D^{(\mathrm{val})})\approx\\
&\tfrac{1}{2\alpha}(\nabla_{A_k}(L(W_k^+,A_k,D^{(\mathrm{tr})})+\lambda\sum_{j\neq k}^{K}L(W_k^+,A_k,D_j^{(\mathrm{pl})}(D^{(\mathrm{u})},V_j')))\\
&-\nabla_{A_k}(L(W_k^-,A_k,D^{(\mathrm{tr})})+\lambda\sum_{j\neq k}^{K}L(W_k^-,A_k,D_j^{(\mathrm{pl})}(D^{(\mathrm{u})},V_j')))),
\end{aligned}\tag{8}$$

where $W_k^{\pm}=W_k\pm\alpha\nabla_{W_k'}L(W_k',A_k,D^{(\mathrm{val})})$ and α is a small scalar $0.01/\|\nabla_{W_k'}L(W_k',A_k,D^{(\mathrm{val})})\|_2$. For $\nabla_{A_k}L(W_j',A_j,D^{(\mathrm{val})})(j\neq k)$ in Eq.(6), it can be calculated as:

$$\begin{aligned}
&\nabla_{A_k}L(W_j',A_j,D^{(\mathrm{val})})=\\
&\tfrac{\partial W_j'}{\partial A_k}\nabla_{W_j'}L(W_j',A_j,D^{(\mathrm{val})})=\\
&-\xi_W\lambda\tfrac{\partial\nabla_{W_j}L(W_j,A_j,D_k^{(\mathrm{pl})}(D^{(\mathrm{u})},V_k'))}{\partial A_k}\nabla_{W_j'}L(W_j',A_j,D^{(\mathrm{val})})=\\
&-\xi_W\lambda\tfrac{\partial V_k'}{\partial A_k}\nabla^2_{V_k',W_j}L(W_j,A_j,D_k^{(\mathrm{pl})}(D^{(\mathrm{u})},V_k'))\nabla_{W_j'}L(W_j',A_j,D^{(\mathrm{val})})
\end{aligned}\tag{9}$$

where

$$\frac{\partial V_k'}{\partial A_k}=\frac{\partial(V_k-\xi_V\nabla_{V_k}L(V_k,A_k,D^{(\mathrm{tr})}))}{\partial A_k}\tag{10}$$

$$=-\xi_V\nabla^2_{A_k,V_k}L(V_k,A_k,D^{(\mathrm{tr})})\tag{11}$$

The algorithm for solving SGL is in Algorithm 1.

Algorithm 1 Optimization algorithm for SGL

while *not converged* **do**
1. For learner $1,\cdots,K$, update V_k using Eq.(4)
2. For learner $1,\cdots,K$, update W_k using Eq.(5)
3. For learner $1,\cdots,K$, update A_k using Eq.(6)

end

4. Experiments

In this section, our proposed SGL is applied to search neural architectures for image classification. Please refer to the supplements for more hyperparameter settings, results, and significance tests of results.

4.1. Datasets

We performed the experiments on three datasets: CIFAR-10, CIFAR-100, and ImageNet (Deng et al., 2009). CIFAR-10 is split into a 25K training set, a 25K validation set, and a 10K test set. So is CIFAR-100. The training and validation set is used as $D^{(\mathrm{tr})}$ and $D^{(\mathrm{val})}$ in SGL. ImageNet has 1.3M training images and 50K test images. CIFAR-10 and CIFAR-100 have 10 classes and ImageNet has 1000 classes.

4.2. Experimental Settings

Following the protocol in (Liu et al., 2019), each experiment consists of an architecture search phrase and an architecture evaluation phrase. In the search phrase, an architecture A is learned by solving Eq.(14). In the evaluation phrase, multiple copies of A are composed into a larger network, which is then trained from scratch and tested on the test set. For the search space of A, we used the ones proposed in DARTS (Liu et al., 2019), 2) DARTS$^-$ (Chu et al., 2020a), 3) P-DARTS (Chen et al., 2019), and 4) PC-DARTS (Xu et al., 2020), where the building blocks include 3×3 and 5×5 (dilated) separable convolutions, 3×3 max/average pooling, identity operation, and zero operation. In SGL, we set the number of learners to 2. The tradeoff parameter λ was tuned in $\{0.1, 0.5, 1, 2, 3\}$ on a held-out 5k dataset. The best performing value of λ is 1. When searching architectures on CIFAR-10, the input images (without labels) in CIFAR-100 were used as the unlabeled dataset; vice versa.

For architecture search on CIFAR-10 and CIFAR-100, each architecture consists of a stack of 8 cells and each cell consists of 7 nodes. The initial channel number was set to 16. The rest hyperparameters for architectures and network weights mostly follow those in DARTS, P-DARTS, PC-DARTS, and DARTS$^-$. The Adam optimizer (Kingma and Ba, 2014) was used to optimize architecture variables, with a learning rate of 3e-4, a batch size of 50, and a weight decay of 1e-3. The SGD optimizer was used to optimize network weights, with an initial learning rate of 0.025, a cosine decay scheduler, a batch size of 50, a weight decay of 3e-4, and a momentum of 0.9. The search algorithm ran for 50 epochs. For architecture search on ImageNet, following (Xu et al., 2020), we randomly sample 10% images from the 1.3M training set as $D^{(\text{tr})}$ and 2.5% images as $D^{(\text{val})}$ in SGL. Another randomly sampled 10% images (excluding labels) are used as the unlabeled dataset. The hyperparameters in the ImageNet experiments mostly follow those in the CIFAR-10 experiments. These experiments were conducted using a single Tesla v100 GPU.

After searching, among the K learners, the one achieving the smallest validation loss is retained and the rest are discarded. The architecture A^* of the retained learner is evaluated. For CIFAR-10 and CIFAR-100, the optimal cell in A^* is duplicated 20 times, which are then stacked into a large network. The initial channel number was set to 36. This large network was trained on the combination of training set and validation set for 600 epochs. The SGD optimizer is used for weights training, with an initial learning rate of 0.025, a cosine decay scheduler, a batch size of 96, a momentum of 0.9, and a weight decay of 3e-4. For architecture evaluation on ImageNet, the large network is a stack of 14 cells, with an initial channel number of 48. The large network was trained on the 1.3M training images using SGD on 8 Tesla v100 GPUs. for 250 epochs, with an initial learning rate of 0.5, a batch size of 1024, and a weight decay of 3e-5. On ImageNet, we also evaluated the architectures searched on CIFAR-10 and CIFAR-100, following the same evaluation protocol. We repeat each SGL experiment 10 times with different random seeds (between 1 and 10). The mean and standard deviation of test results in the 10 runs are reported.

4.3. Results

In Table 2, we compare the classification error on the test set, number of weight parameters, and search cost (GPU days) of different methods on CIFAR-100. We observe the follow-

Table 2: Test error, parameter number, and search cost (GPU days) on CIFAR-100. Search cost is measured by GPU days on a Tesla v100. In SGL, we count the number of parameters in the single retained learner (the one yielding the lowest validation loss). DARTS(1st) and DARTS(2nd) denotes that the first-order and second-order approximation is used in DARTS. On CIFAR-100, DARTS(2nd) is no better than DARTS(1st) despite using a much more computationally-heavy and memory-inefficient second-order approximation. * indicates that the results are taken from DARTS$^-$ (Chu et al., 2020a). † indicates that the results were obtained by re-running the methods for 10 times. In our run of DARTS$^-$, we were not able to achieve the error reported in (Chu et al., 2020a). Δ indicates that during architecture evaluation, we ran for 600 epochs instead of 2000 as in (Liang et al., 2019b) to make sure the comparison with other approaches (which use 600 epochs) is fair.

Method	Error(%)	Param(M)	Cost
*ResNet (He et al., 2016a)	22.10	1.7	-
*DenseNet (Huang et al., 2017)	17.18	25.6	-
*PNAS (Liu et al., 2018a)	19.53	3.2	150
*ENAS (Pham et al., 2018)	19.43	4.6	0.5
*AmoebaNet (Real et al., 2019)	18.93	3.1	3150
*DARTS-2nd (Liu et al., 2019)	20.58±0.44	1.8	1.5
*GDAS (Dong and Yang, 2019)	18.38	3.4	0.2
*R-DARTS (Zela et al., 2020)	18.01±0.26	-	1.6
$^{\Delta}$DARTS$^+$ (Liang et al., 2019a)	17.11±0.43	3.8	0.2
*DropNAS (Hong et al., 2020)	16.39	4.4	0.7
†DARTS(1st) (Liu et al., 2019)	20.52±0.31	1.8	0.4
DARTS(1st) + SGL (ours)	**18.54**±0.21	2.2	1.2
*DARTS$^-$ (Chu et al., 2020a)	17.51±0.25	3.3	0.4
†DARTS$^-$ (Chu et al., 2020a)	18.97±0.16	3.1	0.4
DARTS$^-$ + SGL (ours)	**16.90**±0.10	3.5	2.0
*P-DARTS (Chen et al., 2019)	17.49	3.6	0.3
P-DARTS + SGL (ours)	**16.58**±0.18	3.6	2.1
†PC-DARTS (Xu et al., 2020)	17.01±0.06	4.0	0.1
PC-DARTS + SGL (ours)	**16.34**±0.11	4.1	0.5

ing from this table. **First**, under different settings of search spaces, including those from DARTS, DARTS$^-$, P-DARTS, and PC-DARTS, our method achieves significantly lower test errors than the original baselines. For example, DARTS(1st)+SGL achieves an error of 18.54% which is greatly lower than the 20.52% error of DARTS(1st). As another example, PDARTS+SGL achieves a significantly lower error of 16.58%, compared with the 17.49% error of P-DARTS. These results show that our proposed small-group learning is an effective approach in searching better-performing architectures. In our method, learners with different architectures collaboratively solve the same task. Since having different architec-

Table 3: Test error, parameter number, and search cost (GPU days) on CIFAR-10. * indicates the results are taken from DARTS$^-$ (Chu et al., 2020a), NoisyDARTS (Chu et al., 2020b), DrNAS (Chen et al., 2020), and GTN (Such et al., 2019). Other notations are the same as those in Table 2.

Method	Error(%)	Param(M)	Cost
*DenseNet (Huang et al., 2017)	3.46	25.6	-
*HierEvol (Liu et al., 2018b)	3.75±0.12	15.7	300
*NAONet-WS (Luo et al., 2018)	3.53	3.1	0.4
*PNAS (Liu et al., 2018a)	3.41±0.09	3.2	225
*ENAS (Pham et al., 2018)	2.89	4.6	0.5
*NASNet-A (Zoph et al., 2018)	2.65	3.3	1800
*AmoebaNet-B (Real et al., 2019)	2.55±0.05	2.8	3150
*R-DARTS (Zela et al., 2020)	2.95±0.21	-	1.6
*GDAS (Dong and Yang, 2019)	2.93	3.4	0.2
*SNAS (Xie et al., 2019)	2.85	2.8	1.5
$^{\Delta}$DARTS$^+$ (Liang et al., 2019a)	2.83±0.05	3.7	0.4
*BayesNAS (Zhou et al., 2019)	2.81±0.04	3.4	0.2
*DARTS-2nd (Liu et al., 2019)	2.76±0.09	3.3	1.5
*MergeNAS (Wang et al., 2020)	2.73±0.02	2.9	0.2
*NoisyDARTS (Chu et al., 2020b)	2.70±0.23	3.3	0.4
*ASAP (Noy et al., 2020)	2.68±0.11	2.5	0.2
*SDARTS (Chen and Hsieh, 2020)	2.61±0.02	3.3	1.3
*DropNAS (Hong et al., 2020)	2.58±0.14	4.1	0.6
*FairDARTS (Chu et al., 2019)	2.54	3.3	0.4
*DrNAS (Chen et al., 2020)	2.54±0.03	4.0	0.4
*GTN (Such et al., 2019)	2.42±0.03	97.9	0.67
*DARTS(1st) (Liu et al., 2019)	3.00±0.14	3.3	0.4
DARTS(1st) + SGL (ours)	**2.41**±0.06	3.7	1.2
*DARTS$^-$ (Chu et al., 2020a)	2.59±0.08	3.5	0.4
†DARTS$^-$ (Chu et al., 2020a)	2.97±0.04	3.3	0.6
DARTS$^-$ + SGL (ours)	2.60±0.07	3.1	2.0
*PC-DARTS (Xu et al., 2020)	2.57±0.07	3.6	0.1
PC-DARTS + SGL (ours)	2.60±0.12	3.5	0.5
*P-DARTS (Chen et al., 2019)	2.50	3.4	0.3
P-DARTS + SGL (ours)	2.47±0.10	3.6	2.1

tures, these learners possess complementary advantages: for a certain class x, some learners perform better in classifying examples belonging to x than other learners; for another class y, another set of learners have better classification abilities than the rest of learners; and so on. Via pseudo-labeling, each learner can transfer the knowledge in areas it is good at to other learners; equivalently, each learner can mitigate its deficiency in certain areas by

Table 4: Top-1 and top-5 test errors (%) on ImageNet. * indicates that the results are taken from DARTS$^-$ (Chu et al., 2020a), DrNAS (Chen et al., 2020), and AKDNet (Liu et al., 2020). † denotes that the result is obtained from our run. The rest notations are the same as those in Table 2. From top to bottom, different panels show manually-designed networks, non-differentiable search methods, and differentiable search methods.

Method	Top-1 Error (%)	Top-5 Error (%)	Param (M)	Runtime (GPU days)
*Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	-
*MobileNet (Howard et al., 2017)	29.4	10.5	4.2	-
*ShuffleNet 2× (v1) (Zhang et al., 2018a)	26.4	10.2	5.4	-
*ShuffleNet 2× (v2) (Ma et al., 2018)	25.1	7.6	7.4	-
*NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	1800
*PNAS (Liu et al., 2018a)	25.8	8.1	5.1	225
*MnasNet-92 (Tan et al., 2019)	25.2	8.0	4.4	1667
*AmoebaNet-C (Real et al., 2019)	24.3	7.6	6.4	3150
*SNAS-CIFAR10 (Xie et al., 2019)	27.3	9.2	4.3	1.5
*BayesNAS-CIFAR10 (Zhou et al., 2019)	26.5	8.9	3.9	0.2
*PARSEC-CIFAR10 (Casale et al., 2019)	26.0	8.4	5.6	1.0
*GDAS-CIFAR10 (Dong and Yang, 2019)	26.0	8.5	5.3	0.2
*DSNAS-ImageNet (Hu et al., 2020)	25.7	8.1	-	-
*SDARTS-ADV-CIFAR10 (Chen and Hsieh, 2020)	25.2	7.8	5.4	1.3
*PC-DARTS-CIFAR10 (Xu et al., 2020)	25.1	7.8	5.3	0.1
*ProxylessNAS-ImageNet (Cai et al., 2019)	24.9	7.5	7.1	8.3
*FairDARTS-CIFAR10 (Chu et al., 2019)	24.9	7.5	4.8	0.4
*FairDARTS-ImageNet (Chu et al., 2019)	24.4	7.4	4.3	3.0
*DrNAS-ImageNet (Chen et al., 2020)	24.2	7.3	5.2	3.9
*DARTS$^+$-ImageNet (Liang et al., 2019a)	23.9	7.4	5.1	6.8
*DARTS$^-$-ImageNet (Chu et al., 2020a)	23.8	7.0	4.9	4.5
*DARTS$^+$-CIFAR100 (Liang et al., 2019a)	23.7	7.2	5.1	0.2
*DARTS-2nd-CIFAR10 (Liu et al., 2019)	26.7	8.7	4.7	0.4
†DARTS-1st-CIFAR10 (Liu et al., 2019)	26.1	8.3	4.5	0.4
SGL-DARTS-1st-CIFAR10 (ours)	**24.9**	**7.7**	5.2	2.1
*P-DARTS-CIFAR10 (Chen et al., 2019)	24.4	7.4	4.9	0.3
SGL-P-DARTS-CIFAR10 (ours)	**24.3**	**7.2**	5.1	2.1
*P-DARTS-CIFAR100 (Chen et al., 2019)	24.7	7.5	5.1	0.3
SGL-P-DARTS-CIFAR100 (ours)	**23.9**	**7.2**	5.3	2.1
*PC-DARTS-ImageNet (Xu et al., 2020)	24.2	7.3	5.3	3.8
SGL-PC-DARTS-ImageNet (ours)	**23.3**	**6.7**	6.4	4.0

leveraging the wisdom from other learners which are excellent in those areas. This collaboration mechanism enables different learners to jointly improve. In single-learner NAS, such a mechanism is lacking, which renders the performance to be inferior. **Second**, our method is broadly applicable and effective, as evidenced by the fact that our method improves a variety of baselines (including DARTS, DARTS$^-$, P-DARTS, and PC-DARTS) when applied to these methods. Our method is designed as a general framework which is

agnostic to how the search space is crafted, which paves a way for leveraging our method to improve most (if not all) existing differentiable NAS methods. **Third**, among all methods, PCDARTS+SGL achieves the best performance. This shows that our proposed SGL approach is very competitive in pushing the limit of NAS research. **Fourth**, parameter number and search cost of our method is similar to those of other differentiable methods. This indicates that our method can search more accurate architectures without incurring significant additional costs such as memory footprint, training time, inference time, etc.

In Table 3, we compare different methods on CIFAR-10, in terms of classification error on test set, number of network weights, and search cost (GPU days). When applied to DARTS(1st), our method improves the performance of this baseline very significantly, reducing the 3.00% error of DARTS(1st) to 2.41%, which surpasses all other methods in this table. This again demonstrates that our method is effective in searching highly-performant architectures, thanks to its mechanism of enabling models with complementary advantages to collaboratively learn from each other and mitigate weakness by taking in knowledge from stronger learners. The performance of GTN is close to ours. However, this method leads to a very large architecture with 97.9M parameters, which is about 26 times larger than ours. When applied to other methods including †DARTS^{-}, PC-DARTS, and P-DARTS, our method does not achieve a significant improvement, which is probably due to the fact that CIFAR-10 is a much easier classification dataset than CIFAR-100 and therefore different methods tend to achieve very close performance.

In Table 4, we make a comparison of different methods on ImageNet, in terms of top-1 and top-5 classification errors. In SGL-PC-DARTS-ImageNet, our proposed SGL is applied to PC-DARTS and the search is performed on ImageNet. Our method achieves a top-1 error of 23.3% and a top-5 error of 6.7%, which are the lowest among all methods in this table and are much lower than those of PC-DARTS-ImageNet. This demonstrates that our method has strong capability in searching high-quality architectures not only for small datasets like CIFAR10/100, but also for large-scale real-world datasets such as ImageNet. In addition, on ImageNet we evaluated the architectures searched on CIFAR10/100, including SGL-DARTS-1st-CIFAR10, SGL-P-DARTS-CIFAR10, and SGL-P-DARTS-CIFAR100. As can be seen, these architectures searched by our SGL method achieve better performance than those searched by corresponding baselines. For example, applying SGL to DARTS-1st-CIFAR10, we reduce the 26.1% error of DARTS-1st-CIFAR10 down to 24.9%. This further demonstrates the effectiveness of our method.

4.4. Ablation Studies

In this section, we perform ablation studies to evaluate the importance of individual components in our SGL framework, by comparing the full SGL method with the following ablation settings.

- **Ablation setting 1**. In this setting, the first learning stage in Eq.(14) is removed. Pseudo-labeling is performed using the weights W. The corresponding formulation is:

$$\begin{aligned}
&\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(\{A_j\}_{j=1}^K), A_k, D^{(\text{val})}) \\
&s.t. \quad \{W_k^*(\{A_j\}_{j=1}^K)\}_{k=1}^K = \min_{\{W_k\}_{k=1}^K} \sum_{k=1}^K L(W_k, A_k, D^{(\text{tr})}) \\
&\qquad +\lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, W_j, A_j))
\end{aligned} \tag{12}$$

  In this experiment, λ is set to 2 for CIFAR-10 and to 1 for CIFAR-100. In terms of the search space, it is set to the space used in P-DARTS for CIFAR-100, and set to the space used in DARTS(1st) for CIFAR-10.

- **Ablation setting 2**. In this setting, we separate the first stage in SGL from the second stage. We first run the first stage to search an architecture B_k for each learner k independently, by solving the following problem:

$$\begin{aligned}
&\min_{\{B_k\}_{k=1}^K} \sum_{k=1}^K L(V_k^*(B_k), B_k, D^{(\text{val})}) \\
&s.t. \quad \{V_k^*(B_k)\}_{k=1}^K = \min_{\{V_k\}_{k=1}^K} \sum_{k=1}^K L(V_k, B_k, D^{(\text{tr})})
\end{aligned} \tag{13}$$

  Then we use the searched architectures $\{B_k^*\}_{k=1}^K$ together with their optimally trained network weights $\{V_k^*\}_{k=1}^K$ to perform pseudo-labeling and use the pseudo-labeled datasets $\{D_j^{(\text{pl})}(D^{(\text{u})}, V_j^*, B_j^*)\}_{j=1}^K$ to perform the second stage, which amounts to solving the following problem:

$$\begin{aligned}
&\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(A_k), A_k, D^{(\text{val})}) \\
&s.t. \quad \{W_k^*(A_k)\}_{k=1}^K = \min_{\{W_k\}_{k=1}^K} \sum_{k=1}^K L(W_k, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j^*, B_j^*))
\end{aligned} \tag{14}$$

  In this experiment, λ is set to 2 for CIFAR-10 and to 1 for CIFAR-100. In terms of the search space, it is set to the space used in P-DARTS for CIFAR-100, and set to the space used in DARTS(1st) for CIFAR-10.

- **Ablation setting 3**. In this setting, in the second stage of SGL, each learner is trained solely based on pseudo-labeled datasets by other learners, without using human-labeled training data. The corresponding formulation is:

$$\begin{aligned}
&\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K), A_k, D^{(\text{val})}) \\
&s.t. \quad \{W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K)\}_{k=1}^K = \min_{\{W_k\}_{k=1}^K} \sum_{k=1}^K \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j^*(A_j))) \\
&\qquad \{V_k^*(A_k)\}_{k=1}^K = \min_{\{V_k\}_{k=1}^K} \sum_{k=1}^K L(V_k, A_k, D^{(\text{tr})})
\end{aligned} \tag{15}$$

Table 5: Results for ablation setting 1. "With first stage" corresponds to the full SGL framework. "Without first stage" corresponds to the formulation in Eq.(12) where the first learning stage is removed. The results are classification errors on the test sets of CIFAR-10 and CIFAR-100.

Method	Error (%)
With first stage (CIFAR-100)	**16.58**±0.18
Without first stage (CIFAR-100)	17.33±0.20
With first stage (CIFAR-10)	**2.41**±0.06
Without first stage (CIFAR-10)	2.60±0.05

For CIFAR-100, SGL is applied to P-DARTS. For CIFAR-10, SGL is applied to DARTS-1st.

- Ablation study on the tradeoff parameter λ. We investigate how the classification error changes with λ in Eq.(14). For either CIFAR-10 or CIFAR-100, 5K examples are uniformly sampled from the 25K training set and 25K validation set. Performance is reported on the 5K sampled data. The rest data is used as before. SGL is applied to P-DARTS.

- Ablation study on the number of learners K. We investigate how the classification error changes with the number of learners K. Performance is reported on the 5K randomly sampled data. SGL is applied to DARTS(1st).

Table 5 shows the classification errors on the test sets of CIFAR-10 and CIFAR-100, under the settings of "with first stage" and "without first stage" in the first ablation study. As can be seen, removing the first stage renders the errors to increase on both CIFAR-10 and CIFAR-100, compared with using the first stage. The reason is that: for "without first stage", different learners directly use untrained models to generate pseudo-labels. The classification performance of untrained models is not very satisfactory. As a result, the quality of pseudo-labels has no guarantee. Training models on poorly-labeled datasets will degrade the quality of these models. In contrast, in full SGL which performs model pretraining in the first stage then using pretrained models to perform pseudo-labeling, the risk of generating poor-quality pseudo-labels is significantly reduced.

Table 6 shows the classification errors on the test sets of CIFAR-10 and CIFAR-100, under the settings of "Separate" and "Joint" in the second ablation study. As can be seen, conducting the first and second stage separately leads to worse performance than performing them jointly, on both datasets. The reason is that: when performed separately, the second stage cannot influence the first stage; in the first stage, once the architectures are searched and network weights are trained, they will remain fixed. In contrast, in the full SGL framework where the first and second stage are performed jointly and share the same architecture, models trained in the second stage affect the validation loss, which subsequently affects the architecture; the changed architecture renders the models in the first stage to change as well. This can potentially bring in a synergistic effect: better models in the second stage results in better architectures in the third stage; better architectures

Table 6: Results for ablation setting 2. "Separate" means the first stage in SGL is conducted separately from the second stage. "Joint" corresponds to the full SGL framework where the first and second stages are conducted jointly. The results are classification errors on the test sets of CIFAR-10 and CIFAR-100.

Method	Error (%)
Joint (CIFAR-100)	**16.58**±0.18
Separate (CIFAR-100)	17.80±0.15
Joint (CIFAR-10)	**2.41**±0.06
Separate (CIFAR-10)	2.55±0.13

Table 7: Results for ablation setting 3. "With human labels" corresponds to the full SGL framework. "Without human labels" corresponds to the formulation in Eq.(15) where in the second stage each learner is trained only on pseudo-labeled datasets without using human-labeled datasets. The results are classification errors on the test sets of CIFAR-10 and CIFAR-100.

Method	Error (%)
With human labels (CIFAR-100)	**16.58**±0.18
Without human labels (CIFAR-100)	16.96±0.17
With human labels (CIFAR-10)	**2.41**±0.06
Without human labels (CIFAR-10)	2.74±0.15

improve the models in the first stage. Performing the first and second stage separately makes such a synergistic effect impossible to happen, which hence leads to inferior performance.

Table 7 shows the classification errors on the test sets of CIFAR-10 and CIFAR-100 under the settings of "with human labels" and "without human labels" in the third ablation study. As can be seen, in the second stage of SGL, using only pseudo-labeled datasets for model training without leveraging human-labeled dataset yields worse performance. The reason is that mutual training solely based on pseudo-labeled datasets has a high risk of collective failure: if one model performs poorly, it will generate erroneous pseudo-labels, which renders other models trained using the pseudo-labels to fail as well. By leveraging an external human-labeled dataset where the labels are mostly correct, such a risk can be greatly reduced.

Figure 3 shows the classification errors on CIFAR-100 and CIFAR-10 change as λ increases. As can be seen, on CIFAR-100, when λ increases from 0 to 1, the error decreases. This is because a larger λ results in a stronger collaboration between learners where each learner actively relies on the pseudo-labeled datasets generated by other learners to train itself. A stronger collaboration enables different learners to help each other to improve. However, if λ further increases to 2, the error increases. This is because too much emphasis is put on the pseudo-labeled datasets which are less reliable than the human-labeled training set. Ignoring the human-labeled training set degrades the quality of trained models. Similar trend is observed in the CIFAR-10 results as well.

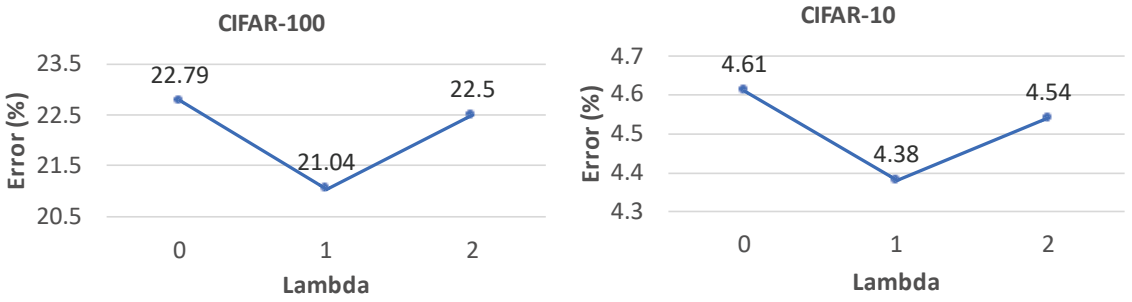


Figure 3: How classification errors change as the tradeoff parameter λ increases. The errors are reported on the 5K randomly sampled data.

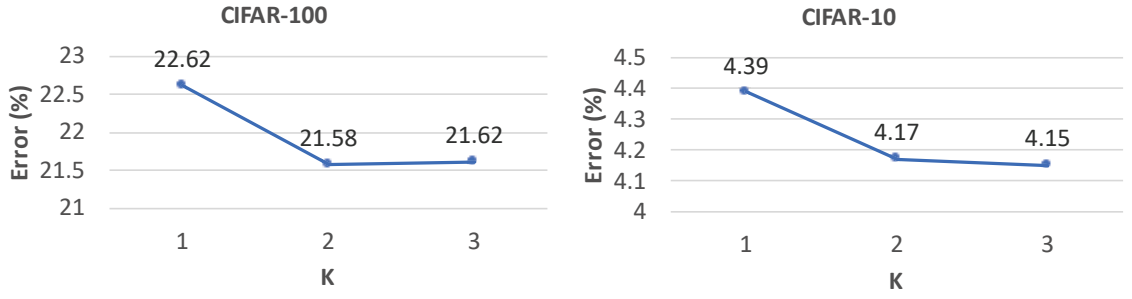


Figure 4: How classification errors change as the number of learners K increases. The errors are reported on the 5K randomly sampled data.

Figure 4 shows how the classification errors on CIFAR-100 and CIFAR-10 vary as the number of learners K increase from 1 to 3. As can be seen, when increasing K from 1 to 2, the error decreases. Under $K = 1$, there is no collaboration. When $K = 2$, two learners collaboratively help each other to improve, hence achieving better performance. When K increases from 2 to 3, the performance does not change significantly. This indicates that two learners are sufficient for exploring the benefit of collaboration.

5. Conclusions

In this paper, drawing inspirations from the small-group learning (SGL) skill of humans, we propose a new ML framework to formalize SGL into a machine learning skill and leverage it to train better ML models. In SGL, a set of learners help each other to learn better: each learner uses its intermediately trained model to make predictions on unlabeled data examples and generates a pseudo-labeled dataset; meanwhile, each learner uses pseudo-labeled datasets generated by other learners to retrain and improve its model. To formalize SGL, a multi-level optimization framework is proposed, which consists of three learning stages: each learner trains an initial model separately; all learners retrain their models collaboratively by mutually performing pseudo-labeling; all learners improve their neural

architectures based on the feedback obtained during model validation. We apply SGL to search neural architectures on CIFAR-100, CIFAR-10, and ImageNet. Experimental results demonstrate the effectiveness of our method.

References

Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100, 1998.

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

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

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

Francesco Paolo Casale, Jonathan Gordon, and Nicoló Fusi. Probabilistic neural architecture search. *CoRR*, abs/1902.05116, 2019.

Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. *CoRR*, abs/2002.05283, 2020.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. *CoRR*, abs/2006.10355, 2020.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019.

Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: eliminating unfair advantages in differentiable architecture search. *CoRR*, abs/1911.12126, 2019.

Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. DARTS-: robustly stepping out of performance collapse without indicators. *CoRR*, abs/2009.01027, 2020a.

Xiangxiang Chu, Bo Zhang, and Xudong Li. Noisy differentiable architecture search. *CoRR*, abs/2005.03566, 2020b.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *CVPR*, 2019.

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.

Jindong Gu and Volker Tresp. Search for better students to learn distilled knowledge. In *ECAI*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016b.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Weijun Hong, Guilin Li, Weinan Zhang, Ruiming Tang, Yunhe Wang, Zhenguo Li, and Yong Yu. Dropnas: Grouped operation dropout for differentiable architecture search. In *IJCAI*, 2020.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. DSNAS: direct neural architecture search without parameter retraining. In *CVPR*, 2020.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. In *CVPR*, 2020.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. DARTS+: improved differentiable architecture search with early stopping. *CoRR*, abs/1909.06035, 2019a.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019b.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018a.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018b.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019.

Yu Liu, Xuhui Jia, Mingxing Tan, Raviteja Vemulapalli, Yukun Zhu, Bradley Green, and Xiaogang Wang. Search to distill: Pearls are everywhere but not the eyes. In *CVPR*, 2020.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, 2018.

Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. ASAP: architecture search, anneal and prune. In *AISTATS*, 2020.

Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. *CoRR*, abs/1912.07768, 2019.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

Ilya Trofimov, Nikita Klyuchnikov, Mikhail Salnikov, Alexander Filippov, and Evgeny Burnaev. Multi-fidelity neural architecture search with knowledge distillation. *CoRR*, abs/2006.08341, 2020.

Xiaoxing Wang, Chao Xue, Junchi Yan, Xiaokang Yang, Yonggang Hu, and Kewei Sun. Mergenas: Merge operations into one for differentiable architecture search. In *IJCAI*, 2020.

Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10687–10698, 2020.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2019.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: partial channel connections for memory-efficient architecture search. In *ICLR*, 2020.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018a.

Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4320–4328, 2018b.

Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *ICML*, 2019.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.