Performance-Aware Mutual Knowledge Distillation for Improving Neural Architecture Search

Pengtao Xie
University of California, San Diego
La Jolla, CA, United States
p1xie@eng.ucsd.edu

Xuefeng Du
University of Wisconsin-Madison
Madison, WI, United States
xuefengdu1@gmail.com

Abstract

Knowledge distillation has shown great effectiveness for improving neural architecture search (NAS). Mutual knowledge distillation (MKD), where a group of models mutually generate knowledge to train each other, has achieved promising results in many applications. In existing MKD methods, mutual knowledge distillation is performed between models without scrutiny: a worse-performing model is allowed to generate knowledge to train a better-performing model, which may lead to collective failures. To address this problem, we propose a performance-aware MKD (PAMKD) approach for NAS, where knowledge generated by model A is allowed to train model B only if the performance of A is better than B. We propose a three-level optimization framework to formulate PAMKD, where three learning stages are performed end-to-end: 1) each model trains an initial model independently; 2) the initial models are evaluated on a validation set and better-performing models generate knowledge to train worse-performing models; 3) architectures are updated by minimizing a validation loss. Experimental results on a variety of datasets demonstrate that our method is effective.

1. Introduction

Neural architecture search (NAS) [34, 45, 64], which aims to automatically search for high-performance neural architectures, has attracted much research attention recently. Many NAS works [17, 26, 29, 35, 49, 57] propose to leverage knowledge distillation (KD) [3, 21, 52] to improve the quality of searched architectures, by transferring knowledge from human-designed architectures to auto-searched architectures [29, 35, 57], enabling multi-fidelity evaluation of architectures [49], alleviating model capacity gap [26], etc. In KD, a teacher model generates knowledge such as pseudo-labels [21] and a student model is trained using these knowledge. Among various studies on KD, mutual KD [1, 4, 27, 28, 37, 51, 55, 62], where a group of models mutually perform KD (i.e., each model generates pseudo-labels to train other models), has shown promising results. Mutual KD can help models converge to a more robust minima [62], can achieve better generalization to test data [28], can learn multi-scale representations to boost prediction accuracy [55], etc.

In existing mutual KD works, knowledge distillation is performed between any pair of models without scrutiny, which may lead to collective failure. If a model A is not performing well, its generated knowledge is not accurate. Trained using these low-quality knowledge, the performance of the rest models is degraded, which renders their knowledge $\mathcal{L}$ to have low-quality as well. Updated using $\mathcal{L}$, model A becomes worse, which further worsens the rest models. This vicious circle renders all models to fail collectively (empirical justification is in Fig. 1).

In this paper, we aim to address this problem, by proposing a performance-aware mutual KD approach, for improving NAS. In our method, performance scrutiny is performed before transferring knowledge: a learner A is allowed to generate knowledge to train another learner B only if the performance of A is better than B. By doing this, the risk of collective failure can be greatly reduced (empirical justification is in Fig. 1), because a poorly-performing learner is prohibited from generating knowledge.

Existing mutual KD methods [1, 4, 27, 28, 37, 51, 55, 62] are not amenable for performance scrutiny. In these methods, the same model weights are used for measuring performance and are trained at the same time, which will lead to a degenerated solution: all models have the same performance and no KD will be conducted (empirical justification is in Table 6). To address this problem, we propose to learn two sets of model weights sequentially for each learner, use one set of them for measuring performance and generating knowledge, and then train the other set using generated knowledge. The two sets of weights are learned sequentially at different stages instead of simultaneously at the same stage, which can avoid the degenerated solution of

existing methods (empirical justification is in Table 6).

Our method is formulated as a three-level optimization problem, consisting of three learning stages performed end-to-end. In the first stage, each learner k independently trains a predictive model V_k. In the second stage, for each pair of learners k and j, their models V_k and V_j are evaluated on a validation dataset. If the performance of V_k is better than V_j, then V_k generates knowledge which is used to train another model W_j of learner j. In the third stage, models trained in the second stage are further validated and their architectures are updated by minimizing validation losses. Chen et al. [4] learn attentional weights to control how much knowledge is allowed to transfer from one model to another. Attentional weights and model parameters are learned jointly on a training dataset, which may lead to overfitting. In contrast, our method measures performance (during scrutiny) on a validation set and trains models on a training set, which can greatly reduce the risk of overfitting.

Another problem of existing KD methods [1, 4, 27, 28, 37, 51, 55, 58, 62] is that they mutually transfer knowledge on individual examples [1, 4, 27, 28, 51, 55, 62] or on low-order triplets [58], without considering the higher-order (e.g. ≥ 4) relationship between examples, which therefore may not be able to capture complex structure of the entire dataset (empirical justification is in Fig. 2). To address this problem, we propose a new group-wise relative similarity (GRS) based approach to transfer knowledge from model j to k, where learner j uses its model to determine which group of data instances have larger mutual similarities, and learner k trains its model by fitting these GRS relationships. These relationships capture high-order (≥ 4) nonlinear manifold structure [46] in the dataset, which can facilitate more effective knowledge transfer between learners (empirical justification is in Fig. 2).

The major contributions of this paper are:

- We propose a performance-aware mutual knowledge distillation (PAMKD) method for improving neural architecture search. In PAMKD, a model A is allowed to generate knowledge to train another model B only if the performance of A is better than B, which can address the collective failure problem of existing MKD methods. Our framework consists of three learning stages which are performed end-to-end: 1) each learner trains a preliminary model; 2) learners conduct performance-aware mutual knowledge distillation; 3) architectures are updated by minimizing validation losses.
- We propose a group-wise relative similarity based knowledge transfer approach which can capture high-order relationships among data instances.
- Experiments on several datasets show the effectiveness of our method.

2. Related works

Neural architecture search (NAS). NAS [15, 34, 39, 45, 64] aims to automatically identify highly-performing architectures of deep neural networks instead of manually designing them. Various approaches have been proposed for NAS, based on gradient algorithms [2, 34, 53], reinforcement learning [43, 50, 64, 65], and evolutionary algorithms [33, 45]. Our proposed framework is orthogonal to existing NAS approaches and can be applied to improve differentiable ones. Zaidi et al. [59] proposed a neural ensemble architecture search method where an architecture is searched for each baseline classifier in an ensemble. In evolutionary algorithm based NAS approaches [33, 45], a population of architectures are evaluated. Related to these methods, our work also searches for a collection of architectures. Such et al. [47] proposed a meta-learning approach to generate synthetic data for NAS. Related to [47], our work also leverages a three-level optimization framework for architecture search. Different from these works including [33, 45, 47, 59], our work performs mutual KD among architectures during the search process while these works do not. Knowledge distillation for NAS has been broadly explored [17, 26, 29, 49]. In these works, a trained teacher network (with a fixed architecture) is leveraged to generate pseudo-labels, which are used to search the architecture of a student network. Different from these works, architectures of all models in our method are searchable. Peng et al. [41] perform mutual KD among subnetworks within a single model. Different from this work, our method performs mutual KD among an ensemble of models.

Knowledge distillation (KD). KD has broad applications in machine learning, such as model compression [21], achieving adversarial robustness [3], and semi-supervised learning [52], etc. Some KD works [3, 21, 42, 52] are unidirectional: knowledge is distilled from a teacher model to a student model while other works [1, 4, 27, 28, 37, 51, 55, 62] are bi-directional: mutual KD is performed between a collection of models. You et al. [58] distill knowledge from multiple teachers to a student, but there is not mutual KD between teachers. Jonghwan et al. [40] distill knowledge from each base model to its corresponding specialized model in multiple choice learning, but there is no mutual KD between base models or between specialized models. In [62], a group of models are trained together where each model generates pseudo labels to train other models. In [4], a group of models mutually transfer knowledge and the ensemble of these models transfers knowledge to a final model to be deployed. In [55], a group of sub-networks with different widths and input resolutions are trained together to mutually learn multi-scale representations. In [28], mutual KD is performed in an implicit way: each model is trained using pseudo-labels generated by an ensemble of all models.

In [11], an online knowledge distillation method is proposed to mutually transfer the knowledge of feature maps among multiple networks, based on adversarial training. In [18], online knowledge distillation is performed via collaboratively transferring knowledge among multiple models. In these works, pseudo-labeling is performed without scrutiny, which may lead to collective failure. Our work focuses on addressing this problem.

3. Methods

In this section, we propose a performance-aware mutual knowledge distillation method for neural architecture search, based on tri-level optimization. In our framework, there are a set of K learners, all of which learn to solve the same target task. Each learner k has an architecture A_k and two sets of model weights V_k and W_k. V_k is used for measuring performance and generating knowledge, and W_k is trained on knowledge generated by other learners. We use two sets of weights instead of one [1,4,27,28,51,55,62] to prevent a degenerated solution where all learners have the same performance and no mutual knowledge distillation is conducted. All learners share the same training dataset $D^{(\text{tr})}$ and the same validation dataset $D^{(\text{val})}$.

3.1. Three learning stages

The K learners perform learning in three stages. In the first stage, each learner k trains V_k. In the second stage, V_k is evaluated on a validation dataset. For any two learners k and j, if V_k performs better than V_j, then learner k uses V_k to generate knowledge, which is used to train W_j of learner j. In the third stage, each learner k measures the validation performance of its model W_k trained in the second stage and updates its architecture to improve the validation performance. We discuss the details in the sequel.

Stage I. In the first stage, each learner k trains its weights V_k by minimizing a training loss defined on $D^{(\text{tr})}$, with its architecture A_k tentatively fixed:

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

The loss function L is application specific. For example, in image classification, L is a cross-entropy loss. The architecture A_k is used to define the training loss. However, A_k should not be optimized to minimize the training loss. Otherwise, the trained model will overfit the training data and have poor performance on unseen data. The optimally trained weights $V_k^*(A_k)$ depends on A_k since $V_k^*(A_k)$ depends on $L(V_k, A_k, D^{(\text{tr})})$ and $L(V_k, A_k, D^{(\text{tr})})$ is a function of A_k.

Stage II. In the second stage, we evaluate the models $\{V_k^*(A_k)\}_{k=1}^K$ trained in the first stage. Given a validation set $D^{(\text{val})}$, we apply $V_k^*(A_k)$ (optionally, together with A_k) to make predictions on this dataset and obtain a validation loss $L(V_k^*(A_k), A_k, D^{(\text{val})})$. A smaller $L(V_k^*(A_k), A_k, D^{(\text{val})})$ indicates $V_k^*(A_k)$ has a better performance. Then we perform performance-aware mutual knowledge distillation. For each pair of learner k and j, if $L(V_k^*(A_k), A_k, D^{(\text{val})}) > L(V_j^*(A_j), A_j, D^{(\text{val})})$, which indicates that learner j performs better than learner k, then we let learner j generate knowledge using $V_j^*(A_j)$ and leverage this knowledge to train learner k. Different from existing methods [51, 55, 58, 62] which conduct knowledge transfer on individual data examples [51, 55, 62] or lower-order triples [58] without considering the relationship among examples, we propose a group-wise relative similarity based knowledge distillation method which capture high-order (≥ 4) relationship among data instances. Our method transfers knowledge from learner j to k by letting k fit the relative similarity relationship between two groups of data examples, where the relationship is labeled by j. Given two groups of data instances $\mathcal{X} = \{x_i\}_{i=1}^R$ and $\mathcal{Y} = \{y_i\}_{i=1}^R$, each having R instances, we use $V_j^*(A_j)$ to label which group has a relatively larger intra-group instance-similarity. For a group $\mathcal{X}$, its intra-group instance-similarity $s(\mathcal{X}; V_j^*(A_j))$ is defined as the smallest cosine similarity between each pair of instances in $\mathcal{X}$:

$$\begin{aligned} &s(\mathcal{X}; V_j^*(A_j)) = \\ &\min(\{c(e(x; V_j^*(A_j)), e(\hat{x}; V_j^*(A_j)))|x, \hat{x} \in \mathcal{X}\}) \end{aligned} \tag{2}$$

where $c(\cdot,\cdot)$ denotes cosine similarity of two vectors, $e(x; V_j^*(A_j))$ denotes the embedding of x extracted by $V_j^*(A_j)$, and $\min(\cdot)$ denotes the minimum of a set. We use minimum to measure the worst-case similarity. Let $\mathcal{X} \succ \mathcal{Y}|V_j^*(A_j)$ denote $s(\mathcal{X}; V_j^*(A_j)) > s(\mathcal{Y}; V_j^*(A_j))$. Such a group-wise relative similarity (GRS) relationship is labeled by $V_j^*(A_j)$. To transfer knowledge from j to k, we use GRS relationships labeled by learner j to guide the training of learner k. Let $s(\mathcal{X}; W_k) = \min(\{c(e(x; W_k), e(\hat{x}; W_k))|x, \hat{x} \in \mathcal{X}\})$ denote the intra-group instance similarity calculated using W_k, we add the following constraint when training W_k:

$$\forall \mathcal{X} \succ \mathcal{Y}|V_j^*(A_j), s(\mathcal{X}; W_k) > s(\mathcal{Y}; W_k). \tag{3}$$

This constraint encourages W_k to learn representations that are compatible with the GRS relationships specified by learner j. Each GRS relationship involves $2R$ data instances and the minimum value of R is 2. Therefore, the GRS-based knowledge transfer approach can capture data relationships with an order of at least 4. Overall, the second stage solves the following optimization problem:

$$\begin{aligned} &\{W_k^*(A_k, \{V_j^*(A_j), A_j\}_{j\neq k}^K)\}_{k=1}^K = \\ &\min_{\{W_k\}_{k=1}^K} \textstyle\sum_{k=1}^K \big(L(W_k, A_k, D^{(\text{tr})}) + \\ &\lambda \textstyle\sum_{j\neq k}^K -\mathbb{I}(L(V_k^*(A_k), A_k, D^{(\text{val})}) > L(V_j^*(A_j), A_j, D^{(\text{val})})) \\ &\textstyle\sum_{\mathcal{X}\succ\mathcal{Y}|V_j^*(A_j)} \mathbb{I}(s(\mathcal{X}; W_k) > s(\mathcal{Y}; W_k))\big), \end{aligned} \tag{4}$$

where $\mathbb{I}(\cdot)$ is an indicator function. The first loss term $L(W_k, A_k, D^{(\text{tr})})$ is defined on a human-labeled dataset and the second loss term represents knowledge distillation. λ is a tradeoff parameter. $\mathcal{X}$ and $\mathcal{Y}$ are randomly sampled from input data instances (removing labels) from $D^{(\text{val})}$.

Eq.(4) further shows that two different sets of weights (V and W) are needed to conduct performance scrutiny. If the same set of model weights are used for measuring validation performance and are trained at the same time, V_k and V_j in Eq.(4) would be replaced with W_k and W_j. A trivial way to minimize this new loss is to make W_k and W_j to be the same: if $W_k = W_j$, the indicator function $\mathbb{I}(\cdot)$ is 0, and the second loss term is 0. This is a degenerated solution because no knowledge will be transferred among models.

Stage III. In the third stage, each learner validates its $W_k^*(A_k, \{V_j^*(A_j), A_j\}_{j\neq k}^K)$ on the validation set $D^{(\text{val})}$. The learners optimize their architectures by minimizing the validation losses:

$$\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(A_k, \{V_j^*(A_j), A_j\}_{j\neq k}^K), A_k, D^{(\text{val})}). \tag{5}$$

Three-level optimization framework. To this end, we formulate PAMKD as a three-level optimization problem:

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

$$\begin{aligned} s.t.\quad & \{W_k^*(A_k, \{V_j^*(A_j), A_j\}_{j\neq k}^K)\}_{k=1}^K = \\ & \min_{\{W_k\}_{k=1}^K} \sum_{k=1}^K \big(L(W_k, A_k, D^{(\text{tr})}) + \\ & \lambda \sum_{j\neq k}^K -\mathbb{I}(L(V_k^*(A_k), A_k, D^{(\text{val})}) > L(V_j^*(A_j), A_j, D^{(\text{val})})) \\ & \sum_{\mathcal{X}\succ\mathcal{Y}|V_j^*(A_j)} \mathbb{I}(s(\mathcal{X}; W_k) > s(\mathcal{Y}; W_k)) \end{aligned}$$

$$\{V_k^*(A_k)\}_{k=1}^K = \min_{\{V_k\}_{k=1}^K} \sum_{k=1}^K L(V_k, A_k, D^{(\text{tr})})$$

Continuous relaxation. The indicator functions are not differentiable, which prohibits applying gradient-based optimization methods. To address this problem, we use hinge losses to relax indicator functions. We relax $\mathbb{I}(L(V_k^*(A_k), A_k, D^{(\text{val})}) > L(V_j^*(A_j), A_j, D^{(\text{val})}))$ as:

$$\max(0, L(V_k^*(A_k), A_k, D^{(\text{val})}) - L(V_j^*(A_j), A_j, D^{(\text{val})})), \tag{6}$$

and relax $\sum_{\mathcal{X}\succ\mathcal{Y}|V_j^*(A_j)} \mathbb{I}(s(\mathcal{X}; W_k) > s(\mathcal{Y}; W_k)$ as

$$\begin{aligned} & \sum_{\mathcal{X}\succ\mathcal{Y}|V_j^*(A_j)} \max(0, \epsilon - (s(\mathcal{X}; V_j^*(A_j)) - s(\mathcal{Y}; V_j^*(A_j))) \\ & (s(\mathcal{X}; W_k) - s(\mathcal{Y}; W_k))), \end{aligned} \tag{7}$$

where ϵ is a small positive number. The hinge loss $\max(0, d_{kj})$ where $d_{kj} = L(V_k^*(A_k), A_k, D^{(\text{val})}) - L(V_j^*(A_j), A_j, D^{(\text{val})})$ controls whether KD should be performed from learner j to learner k. If $d_{kj} \leq 0$ which indicates that j performs no better than k, the hinge loss is zero and correspondingly the KD loss (second loss term in Eq.(3)) is zero. If $d_{kj} > 0$, the hinge loss is d_{kj}. The larger this difference is, the more valuable it is to transfer knowledge from j to k. Note that V in the hinge loss cannot be replaced with W. Otherwise, a trivial solution in the second step is to make $d_{kj} = 0$, which in fact removes mutual KD. Negative of the multiplication of the two indicator functions is replaced with the multiplication of these two hinge losses.

Algorithm 1 Optimization algorithm for PAMKD

while not converged **do**
1. For learner $1, \cdots, K$, update V_k
2. For learner $1, \cdots, K$, update W_k
3. For learner $1, \cdots, K$, update A_k

Optimization algorithm. To solve the PAMKD problem, we develop a gradient-based optimization algorithm, drawing inspirations from [34]. First of all, we approximate $V_k^*(A_k)$ using

$$V_k' = V_k - \xi_V \nabla_{V_k} L(V_k, A_k, D^{(\text{tr})}), \tag{8}$$

where ξ_V is a learning rate. Plugging $\{V_j'\}_{j=1}^K$ into the loss function at the second stage, we obtain an approximated objective O. Then we approximate $W_k^*(A_k, \{V_j^*(A_j), A_j\}_{j\neq k}^K)$ using one-step gradient descent update of W_k w.r.t the approximated objective:

$$W_k' = W_k - \xi_W \nabla_{W_k} O. \tag{9}$$

Finally, we plug $\{W_k'\}_{k=1}^K$ into the validation loss at the third stage and get an approximated validation loss. Then we update $\{A_k\}_{k=1}^K$ by minimizing the approximated validation loss:

$$\begin{aligned} A_k \leftarrow A_k - \xi_A (& \nabla_{A_k} L(W_k', A_k, D^{(\text{val})}) + \\ & \sum_{j\neq k}^K \nabla_{A_k} L(W_j', A_j, D^{(\text{val})})). \end{aligned} \tag{10}$$

These steps iterate until convergence. The algorithm is summarized in Algorithm 1.

Differentiable representations of architectures. Following [34], we represent architectures using continuous variables which are multiplied to the outputs of candidate building blocks. A larger variable value indicates the corresponding block is more important. Architecture search is formulated as identifying optimal values of these variables using gradient based methods.

4. Experiments

In this section, we present experimental results.

4.1. Datasets

We performed the experiments on three datasets: CIFAR-100, CIFAR-10, and ImageNet [12]. CIFAR-100

Method	Error-C100	Error-C10	Param.	Cost
*ResNet [20]	22.10	6.43	1.7	-
*DenseNet [25]	17.18	3.46	25.6	-
*PNAS [32]	19.53	3.41±0.09	3.2	150
*ENAS [43]	19.43	2.89	4.6	0.5
*AmoebaNet [45]	18.93	2.55±0.05	3.1	3150
*DARTS-2nd [34]	20.58±0.44	2.76±0.09	1.8	1.5
*GDAS [13]	18.38	2.93	3.4	0.2
*R-DARTS [61]	18.01±0.26	2.95±0.21	-	1.6
*DARTS$^-$ [9]	17.51±0.25	2.59±0.08	3.3	0.4
*DropNAS [22]	16.95±0.41	2.58±0.14	4.4	0.7
*DrNAS [7]	-	2.54±0.03	4.0	0.4
*ISTA-NAS [56]	-	2.54±0.05	3.3	0.1
*MiLeNAS [19]	-	2.51±0.11	3.9	0.3
*GAEA [30]	-	2.50±0.06	-	0.1
*PDARTS-ADV [6]	-	2.48±0.02	3.4	1.1
†Darts1st [34]	20.52±0.31	3.00±0.14	1.8	0.4
Darts1st + MKD [62]	18.37±0.16	2.70±0.05	2.1	1.3
Darts1st + NES [59]	18.32±0.19	2.75±0.11	2.2	1.2
Darts1st + OKDDip [4]	18.69±0.21	2.83±0.09	2.1	1.5
Darts1st + KDCL [18]	18.33±0.10	2.68±0.06	2.3	1.7
Darts1st + AFD [11]	18.62±0.17	2.85±0.06	2.2	1.6
Darts1st + ONE [28]	18.93±0.11	2.68±0.07	2.3	1.4
Darts1st + Cream [41]	18.36±0.14	2.95±0.12	2.3	1.8
Darts1st + PAMKD (ours)	**17.61**±0.16	**2.38**±0.03	2.2	1.2
*Pdarts [8]	17.43±0.15	2.54±0.04	3.6	0.3
Pdarts + MKD [62]	16.55±0.13	2.50±0.02	3.7	2.1
Pdarts + NES [59]	16.48±0.16	2.75±0.08	3.6	2.2
Pdarts + OKDDip [4]	16.92±0.20	2.77±0.11	3.8	2.4
Pdarts + KDCL [18]	16.59±0.08	2.52±0.05	3.8	2.3
Pdarts + AFD [11]	16.79±0.15	2.51±0.03	3.7	2.4
Pdarts + ONE [28]	17.04±0.09	2.68±0.07	3.8	2.0
Pdarts + Cream [41]	16.38±0.14	2.65±0.04	3.5	2.5
Pdarts + PAMKD (ours)	**15.82**±0.11	**2.42**±0.06	3.6	2.1
†Pcdarts [54]	17.01±0.06	2.57±0.07	4.0	0.1
Pcdarts + MKD [62]	16.38±0.13	2.56±0.03	4.2	0.6
Pcdarts + NES [59]	16.36±0.11	2.69±0.06	4.2	0.7
Pcdarts + OKDDip [4]	16.59±0.14	2.72±0.09	4.4	0.8
Pcdarts + KDCL [18]	16.33±0.09	2.61±0.08	4.3	0.9
Pcdarts + AFD [11]	16.71±0.16	2.59±0.07	4.3	0.8
Pcdarts + ONE [28]	16.92±0.18	2.66±0.04	4.0	0.8
Pcdarts + Cream [41]	16.63±0.10	2.74±0.05	4.3	1.0
Pcdarts + PAMKD (ours)	**15.89**±0.07	**2.46**±0.05	4.1	0.6
†Prdarts [63]	16.48±0.06	2.37±0.03	3.4	0.2
Prdarts + MKD [62]	16.35±0.08	2.35±0.02	3.3	0.8
Prdarts + NES [59]	16.59±0.06	2.48±0.06	3.6	0.9
Prdarts + OKDDip [4]	16.83±0.04	2.51±0.07	3.5	1.0
Prdarts + KDCL [18]	16.39±0.07	2.42±0.05	3.6	0.8
Prdarts + AFD [11]	16.47±0.06	2.39±0.08	3.6	0.9
Prdarts + ONE [28]	17.03±0.09	2.48±0.09	3.4	1.1
Prdarts + Cream [41]	16.86±0.05	2.36±0.06	3.6	1.5
Prdarts + PAMKD (ours)	**16.05**±0.04	**2.28**±0.03	3.5	0.8

Table 1. Results on CIFAR-100 and CIFRA-10. * indicates that the results are taken from DARTS$^-$ [9]. † indicates that the results were obtained by re-running the methods for 10 times.

is split into a 25K training set, a 25K validation set, and a 10K test set. So is CIFAR-10. The training and validation set is used as $D^{(\text{tr})}$ and $D^{(\text{val})}$ in PAMKD. ImageNet has 1.3M training images and 50K test images. CIFAR-100 and CIFAR-10 have 10 classes and ImageNet has 1000 classes.

4.2. Experimental settings

Following [34], each experiment consists of an architecture search phrase where an architecture A is learned and an architecture evaluation where 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 [34], 2) P-DARTS [8], 3) PC-DARTS [54], and 4) PR-DARTS [63]. In PAMKD, we set the number of learners to 2. λ is set to 0.1 for PCDARTS on CIFAR-100; 0.5 for PR-DARTS on CIFAR-100/10, PCDARTS on CIFAR-10, DARTS1st on CIFAR-10/100; 1 for PDARTS on CIFAR-100/10. ϵ is set to 0.01. R is set to 5. When constructing GRS relationships, instance groups are randomly sampled. For architecture search on CIFAR-100 and CIFAR-10, each architecture consists of a stack of 8 cells and each cell consists of 7 nodes.

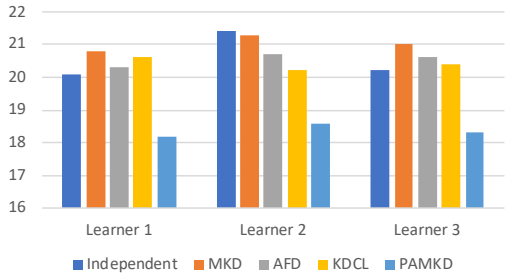

Figure 1. Empirical evidence of cascaded failure.

For architecture search on ImageNet, following [54], we randomly sample 10% images from the 1.3M training set as $D^{(\text{tr})}$ and 2.5% images as $D^{(\text{val})}$ in PAMKD. In PC-DARTS, architectures are directly searched on ImageNet. Search was performed for 50 epochs. After searching, among the K learners, the one achieving the smallest validation loss is retained and its architecture is evaluated. The mean and standard deviation of 10 random runs are reported.

We compare with the following baselines: 1) MKD [62] without performance scrutiny, 2) neural ensemble search (NES) [59], 3) online knowledge distillation with diverse peers (OKDDip) [4], 4) knowledge distillation via collaborative learning (KDCL) [18], 5) adversarial feature distillation (AFD) [11], 6) on-the-fly native ensemble (ONE) [28] for knowledge distillation, and 7) distilling prioritized paths for one-shot neural architecture search (Cream) [41]. We adapted MKD, OKDDip, KDCL, AFD, and ONE to architecture search tasks.

4.3. Results and analysis on CIFAR-100 and CIFAR-10

In Table 1, we compare different methods on CIFAR-100 and CIFAR-10. In PAMKD, we count the number of parameters in a single retained learner (the one yielding the lowest validation loss). We observe the following from this table. **First**, under different settings of search spaces, including those from DARTS, P-DARTS, PC-DARTS, and PR-DARTS, our method performs significantly better than MKD, KDCL, AFD, and ONE. Our method performs scrutiny when performing mutual KD: a learner A is allowed to generate knowledge to train another learner B only when performance of A is better than B.

Data	Space	Darts [34]	DartsES [60]	Darts$^-$ [9]	MKD [62]	OKDDip [4]	Ours
C10	S1	4.66±0.71	3.05±0.07	2.76±0.07	4.38±0.59	4.01±0.36	**2.43**±0.11
C10	S2	4.42±0.40	3.41±0.14	2.79±0.04	4.58±0.25	4.32±0.69	**2.30**±0.07
C10	S3	4.12±0.85	3.71±1.14	2.65±0.04	4.07±0.62	3.89±0.58	**2.27**±0.04
C10	S4	6.95±0.18	4.17±0.21	2.91±0.04	5.96±0.37	5.61±0.15	**2.54**±0.15
C100	S1	29.93±0.41	28.90±0.81	23.26±0.59	29.53±0.66	29.18±0.49	**22.32**±0.23
C100	S2	28.75±0.92	24.68±1.43	22.31±0.65	28.47±0.32	28.83±0.69	**21.47**±0.50
C100	S3	29.01±0.24	26.99±1.79	21.47±0.40	28.58±0.42	27.62±0.23	**20.61**±0.27
C100	S4	24.77±1.51	23.90±2.01	21.75±0.26	23.99±1.37	23.64±1.06	**21.05**±0.64

Table 2. Evaluation of robustness: errors on test sets of CIFAR-10 (C10) and CIFAR-100 (C100).

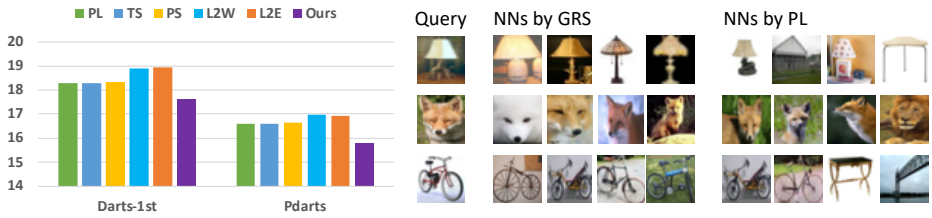

Figure 2. (Left) Comparison of knowledge transfer approaches. (Right) Nearest neighbors retrieved by GRS and PL.

With such a scrutiny mechanism, collective failure can be prevented. In contrast, in MKD, KDCL, AFD, and ONE, each learner is allowed to generate knowledge to train other learners, even when this learner performs worse than others, which incurs high risk of collective failure. Figure 1 provides some empirical evidence. We first train the initial models of three learners independently. Then we train the second set of models of these learners via MKD, KDCL, AFD without scrutiny. As can be seen, after MKD, KDCL, AFD training, performance of learner 1 and 3 are both degraded. This is because learner 2 performs worse than learner 1 and 3. Knowledge generated by learner 2 has poor quality. Trained using such low-quality knowledge, the performance of learner 1 and 3 is degraded. Under PAMKD which only allows KD when one learner outperforms another, the performance of all three learners are improved.

Second, our method works better than OKDDip. While OKDDip learns attentional weights to control how much knowledge is allowed to transfer between one learner to another, it trains these attentional weights together with model parameters on a single training set, which is prone to overfitting. In contrast, our method conducts performance-scrutiny on a validation set and trains model parameters on a training set, which is resilient to overfitting. **Third**, our method achieves significantly lower test errors than vanilla DARTS1st, PDARTS, PCDARTS, PR-DARTS, and NES. These methods do not perform mutual knowledge distillation, which leads to inferior performance. In our method, learners with different architectures collaboratively solve the same task. With different architectures, these learners possess complementary advantages. Via collaboration, each learner can transfer the knowledge in areas it is good at to other learners. Collaboration enables different learners to jointly improve. **Fourth**, parameter number and search cost of our method is similar to those of other differentiable methods, indicating that our method can search more accurate architectures without incurring significant additional costs in memory footprint and inference time.

4.4. Robustness

By dynamically transferring knowledge from better-performing models to worse-performing models where the knowledge is group-wise relative similarity relationships generated by encoders of better-performing models, our method can learn representations that are robust to performance collapse. We empirically verify this by evaluating our method on four architecture search spaces designed by [60]. These spaces are specifically crafted for assessing robustness against performance collapse. Table 2 shows our method achieves lower test errors on CIFAR-100/10 under these four spaces, in comparison with baselines. This demonstrates the capability of our method in avoiding performance degeneration.

4.5. Comparison of knowledge transfer approaches

We compare our group-wise relative similarity based knowledge transfer approach with the following: **1)** pseudo-labeling (PL) [62]; **2)** triple-wise similarity (TS) [58]: a better-performing model (BPM) annotates relative similarity relations (e.g., the similarity between x and y is larger than that between z and y) and a worse-performing model (WPM) fits these relative similarities; **3)** pairwise similarity (PS) [5]: a BPM annotates whether two images are similar or dissimilar and a WPM fits these similarity labels; **4)** L2 regularization on encoder weights (L2W) [44]: encouraging model weights of different learners to have small L2

distance; **5)** L2 regularization on embeddings (L2E) [16]: encouraging embeddings generated by two models to have small L2 distance.

Figure 2(left) shows the results. Our method works better than PL, L2E, and L2W. These baselines transfer knowledge on individual data instances without considering the relationship between instances, which cannot capture global properties of an entire dataset. In contrast, our method takes the group-wise relative similarity relationships among data instances as input and transfers knowledge between learners at the group level instead of individual instance level. Our method works better than PS and TS. PS and TS are limited to capturing second-order and third-order relationships among instances while our method can capture higher order (≥ 4) relationships.

Figure 2(right) shows 4-nearest neighbors retrieved by GRS and PL for some randomly sampled CIRAR-100 test images. As can be seen, nearest neighbors under GRS are more semantically similar to query images than PL. By encouraging different learners to be consistent on group-wise similarities, GRS can more effectively group similar images together, which is good for classification.

4.6. Results on ImageNet

In Table 3, we make a comparison of different methods on ImageNet, in terms of top-1 and top-5 classification errors (%). PAMKD-pcdarts-ImageNet denotes the architecture is searched on ImageNet by applying PAMKD to Pcdarts. Similar meanings hold for other notations like this. Observations made from this table are similar to those in Table 1. Our method outperforms MKD, KDCL, AFD, and ONE, which further demonstrates the effectiveness of conducting performance scrutiny during mutual knowledge transfer. Our method performs better than OKDDip, which further shows the effectiveness of conducting performance scrutiny on a separate validation set. Our method achieves lower errors than NES and vanilla DARTS1st, P-DARTS, PCDARTS, which again demonstrates the effectiveness of performing mutual knowledge transfer.

Method	Top-1	Top-5
*Inception-v1 [48]	30.2	10.1
*ShuffleNet 2× (v2) [38]	25.1	7.6
*PNAS [32]	25.8	8.1
*AKDNet [36]	24.5	6.9
*AmoebaNet-C [45]	24.3	7.6
*DSNAS-ImageNet [23]	25.7	8.1
*SDARTS-ADV-CIFAR10 [6]	25.2	7.8
*PCDARTS-CIFAR10 [54]	25.1	7.8
*ProxylessNAS-ImageNet [2]	24.9	7.5
*FairDARTS-ImageNet [10]	24.4	7.4
*PR-DARTS [63]	24.1	7.3
*DARTS$^+$-CIFAR100 [31]	23.7	7.2
*Darts2nd-cifar10 [34]	26.7	8.7
†Darts1st-cifar10 [34]	26.1	8.3
MKD-darts1st-cifar10 [62]	24.7	7.6
NES-darts1st-cifar10 [59]	24.8	7.7
OKDDip-darts1st-cifar10 [4]	25.0	8.0
KDCL-darts1st-cifar10 [18]	24.9	7.7
AFD-darts1st-cifar10 [11]	24.7	7.5
ONE-darts1st-cifar10 [28]	25.3	8.0
Cream-darts1st-cifar10 [41]	26.0	8.2
PAMKD-darts1st-cifar10 (ours)	**24.3**	**7.2**
*Pdarts-cifar10 [8]	24.4	7.4
MKD-pdarts-cifar10 [62]	24.4	7.3
NES-pdarts-cifar10 [59]	24.3	7.2
OKDDip-pdarts-cifar10 [4]	24.5	7.4
KDCL-pdarts-cifar10 [18]	24.4	7.4
AFD-pdarts-cifar10 [11]	24.5	7.6
ONE-pdarts-cifar100 [28]	24.4	7.4
Cream-pdarts-cifar10 [41]	24.3	7.3
PAMKD-pdarts-cifar10 (ours)	**23.9**	**6.8**
*Pdarts-cifar100 [8]	24.7	7.5
MKD-pdarts-cifar100 [62]	23.8	7.1
NES-pdarts-cifar100 [59]	24.0	7.3
OKDDip-pdarts-cifar100 [4]	24.3	7.5
KDCL-pdarts-cifar100 [18]	23.9	7.2
AFD-pdarts-cifar100 [11]	23.8	7.0
ONE-pdarts-cifar100 [28]	24.6	7.6
Cream-pdarts-cifar100 [41]	24.5	7.5
PAMKD-pdarts-cifar100 (ours)	**23.2**	**6.7**
*Pcdarts-ImageNet [54]	24.2	7.3
MKD-pcdarts-ImageNet [62]	23.2	6.8
NES-pcdarts-ImageNet [59]	23.4	7.0
OKDDip-pcdarts-ImageNet [4]	23.6	7.1
KDCL-pcdarts-ImageNet [18]	23.4	7.1
AFD-pcdarts-ImageNet [11]	23.2	6.9
ONE-pcdarts-ImageNet [28]	23.5	7.1
Cream-pcdarts-ImageNet [41]	23.9	7.2
PAMKD-pcdarts-ImageNet (ours)	**22.8**	**6.4**

Table 3. Top-1 and top-5 test errors on ImageNet. * indicates that the results are taken from DARTS$^-$ [9], DrNAS [7], and AKDNet [36]. † denotes that the result is obtained from our run. The rest notations are the same as those in Table 1.

4.7. Results on NAS-Bench-201

Table 4 shows the results on NAS-Bench-201. Our method outperforms baselines. The analysis of reasons is similar to that for results in Table 1.

4.8. Ablation studies

In the first ablation study, we further investigate the effectiveness of performance scrutiny in mutual knowledge distillation. We experimented with the following ablation settings: in our framework, replacing the objective function at the second stage in Eq.(4) with the objectives of MKD and KDCL (denoted by 2nd-MKD and 2nd-KDCL respectively), where different models are allowed to mutually teach each other without performance scrutiny. Test errors on CIFAR-100 and CIFAR-10 are shown in Table 5. We can see that our method (which uses the performance scrutiny objective function in Eq.(4)) outperforms these two ablation settings. This further demonstrates the necessity of conducting performance scrutiny during mutual knowledge distillation.

Next, we perform an ablation study of "no first stage (No-1st)": the first learning stage is removed. Performance-aware knowledge distillation is performed using the weights W. λ is set to 1. We use the search space of P-DARTS, DARTS1st, and PR-DARTS. Table 6 shows test errors. Re-

	CIFAR-10		CIFAR-100		ImageNet-16-120	
	Validation	Test	Validation	Test	Validation	Test
DARTS2nd [34]	39.77±0.00	54.30±0.00	38.57±0.00	38.97±0.00	18.87±0.00	18.41±0.00
GDAS [14]	90.01±0.46	93.23±0.23	24.05±8.12	24.20±8.08	40.66±0.00	41.02±0.00
SNAS [53]	90.10±1.04	92.77±0.83	69.69±2.39	69.34±1.98	42.84±1.79	43.16±2.64
DSNAS [24]	89.66±0.29	93.08±0.13	30.87±16.40	31.01±16.38	40.61±0.09	41.07±0.09
PC-DARTS [54]	89.96±0.15	93.41±0.30	67.12±0.39	67.48±0.89	40.83±0.08	41.31±0.22
Drnas [7]	91.55±0.00	94.36±0.00	73.49±0.00	73.51±0.00	46.37±0.00	46.34±0.00
MKD+Drnas [62]	88.47±0.82	90.58±0.74	67.44±2.56	68.96±3.58	43.75±0.15	44.59±0.27
NES+Drnas [59]	90.91±0.68	87.55±0.42	65.38±3.72	70.48±3.52	46.27±0.33	45.72±0.13
OKDDip+Drnas [4]	91.04±0.73	89.72±0.44	68.71±1.08	70.73±4.88	44.69±1.36	38.64±0.52
KDCL+Drnas [18]	89.01±0.75	89.36±0.58	67.28±1.94	69.27±3.92	46.26±0.42	39.05±0.26
AFD+Drnas [11]	88.95±0.79	87.99±0.64	66.04±2.51	71.25±2.70	45.84±0.41	41.82±0.77
ONE+Drnas [28]	89.35±0.37	93.50±0.21	68.84±0.38	70.59±2.61	41.64±0.12	40.86±2.49
Cream+Drnas [41]	85.49±0.36	94.37±0.26	70.62±0.99	74.02±0.53	45.98±0.23	40.72±0.31
Ours+Drnas	**92.64**±0.11	**94.83**±0.07	**74.58**±0.10	**74.25**±0.07	**47.73**±0.15	**47.59**±0.09

Table 4. Validation and test accuracy on NAS-Bench-201.

Method	Error-CIFAR100	Error-CIFAR10
2nd-MKD + Darts1st	18.25±0.10	2.66±0.07
2nd-KDCL + Darts1st	18.03±0.14	2.68±0.06
Ours + Darts1st	**17.61**±0.16	**2.38**±0.03
2nd-MKD + Pdarts	16.72±0.09	2.53±0.03
2nd-KDCL + Pdarts	16.51±0.06	2.50±0.06
Ours + Pdarts	**15.82**±0.11	**2.42**±0.06

Table 5. Test errors (%) on CIFAR-100 and CIFAR-10, in the ablation study of performance scrutiny.

	Method	Error (%)
C100	PAMKD + Pdarts	**15.82**±0.11
	No-1st + Pdarts	17.58±0.26
	PAMKD + Prdarts	**16.05**±0.04
	No-1st + Prdarts	16.89±0.06
C10	PAMKD + Darts1st	**2.38**±0.03
	No-1st + Darts1st	2.65±0.07
	PAMKD + Prdarts	**2.28**±0.03
	No-1st + Prdarts	2.50±0.04

Table 6. Classification errors for "no first stage (No-1st)", on test sets of CIFAR-100 (C100) and C10.

moving the first stage renders the errors to increase on both CIFAR-10 and CIFAR-100. The reason is that: in No-1st, for each learner, the same model weights are evaluated and trained simultaneously. To minimize the knowledge distillation loss, a trivial solution is to make all learners have the same validation performance, which makes the knowledge distillation loss become 0. This is a degenerated solution where all learners perform learning independently. In the experiments, we observed that different learners in No-1st reach the same validation loss around epoch 40 and stop mutual KD. Our full method avoids this problem by using two different sets of model weights to conduct performance-aware MKD: one for measuring performance and the other gets trained. In the experiments, we observed that different learners in our method perform mutual KD throughout the entire training process.

Figure 3(Left) shows how classification error of PAMKD+Pdarts changes with the tradeoff parameter λ, on 5K held-out CIFAR-100 data. A λ in the middle ground yields the optimal performance.

We investigate how classification error changes with the number of learners K, in PAMKD+Darts1st. Figure 3(Right) shows results on 5K held-out CIFAR-100 data. 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 via PAMKD. 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 PAMKD.

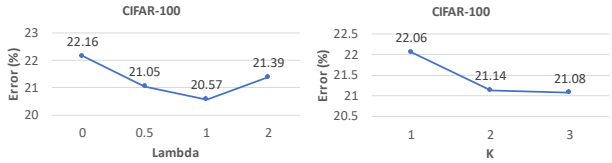

Figure 3. How classification errors change with λ (left) and K (right).

5. Conclusions and discussions

We propose a tri-level optimization based performance-aware mutual knowledge distillation (MKD) approach for neural architecture search, to address the collective failure problem of previous MKD methods. Unlike previous methods where knowledge distillation can happen from any two models, our method adds a scrutiny mechanism: model A is allowed to generate knowledge to train model B only if the performance of A is better than B. We formulate performance-aware MKD as a three-level optimization problem, containing three stages performed end-to-end: 1) each learner trains an initial model independently; 2) learners perform performance-aware MKD; and 3) each learner updates its architecture by minimizing validation losses. Experiments on a variety of datasets demonstrate the effectiveness of our method.

Our method has the following limitations: it learns multiple models instead of one, which incurs more memory consumption and computational cost. We present some studies towards addressing this limitation in the supplements. Our work has the following potential negative societal impact: the increased computational overhead due to training multiple models consumes more power energy, which leads to more greenhouse gas emissions.

References

[1] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Róbert Ormándi, George E. Dahl, and Geoffrey E. Hinton. Large scale distributed neural network training through online distillation. *CoRR*, abs/1804.03235, 2018. 1, 2, 3

[2] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 2, 7

[3] 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. 1, 2

[4] Defang Chen, Jian-Ping Mei, Can Wang, Yan Feng, and Chun Chen. Online knowledge distillation with diverse peers. *CoRR*, abs/1912.00350, 2019. 1, 2, 3, 5, 6, 7, 8

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020. 6

[6] Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. *CoRR*, abs/2002.05283, 2020. 5, 7

[7] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. *CoRR*, abs/2006.10355, 2020. 5, 7, 8

[8] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019. 5, 7

[9] 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, 2020. 5, 6, 7

[10] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: eliminating unfair advantages in differentiable architecture search. *CoRR*, abs/1911.12126, 2019. 7

[11] Inseop Chung, SeongUk Park, Jangho Kim, and Nojun Kwak. Feature-map-level online adversarial knowledge distillation. In *International Conference on Machine Learning*, pages 2006–2015. PMLR, 2020. 3, 5, 7, 8

[12] 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. 4

[13] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *CVPR*, 2019. 5

[14] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours, 2019. 8

[15] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020. 2

[16] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018. 7

[17] Jindong Gu and Volker Tresp. Search for better students to learn distilled knowledge. In *ECAI*, 2020. 1, 2

[18] Qiushan Guo, Xinjiang Wang, Yichao Wu, Zhipeng Yu, Ding Liang, Xiaolin Hu, and Ping Luo. Online knowledge distillation via collaborative learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11020–11029, 2020. 3, 5, 7, 8

[19] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation, 2020. 5

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1, 2

[22] 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. 5

[23] 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. 7

[24] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. Dsnas: Direct neural architecture search without parameter retraining, 2020. 8

[25] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 5

[26] Minsoo Kang, Jonghwan Mun, and Bohyung Han. Towards oracle knowledge distillation with neural architecture search, 2019. 1, 2

[27] Jangho Kim, Minsung Hyun, Inseop Chung, and Nojun Kwak. Feature fusion for online mutual knowledge distillation. *CoRR*, abs/1904.09058, 2019. 1, 2, 3

[28] Xu Lan, Xiatian Zhu, and Shaogang Gong. Knowledge distillation by on-the-fly native ensemble. *CoRR*, abs/1806.04606, 2018. 1, 2, 3, 5, 7, 8

[29] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Blockwisely supervised neural architecture search with knowledge distillation, 2020. 1, 2

[30] Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search, 2021. 5

[31] 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, 2019. 7

[32] 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*, 2018. 5, 7

[33] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018. 2

[34] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019. 1, 2, 4, 5, 6, 7, 8

[35] 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, 2020. 1

[36] 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. 7

[37] Fan Ma, Deyu Meng, Xuanyi Dong, and Yi Yang. Self-paced multi-view co-training. *Journal of Machine Learning Research*, 21(57):1–38, 2020. 1, 2

[38] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, 2018. 7

[39] Abhinav Mehrotra, Alberto Gil Ramos, Sourav Bhattacharya, Łukasz Dudziak, Ravichander Vipperla, Thomas Chau, Mohamed S Abdelfattah, Samin Ishtiaq, and Nicholas D Lane. Nas-bench-asr: Reproducible neural architecture search for speech recognition. In *International Conference on Learning Representations (ICLR)*, 2021. 2

[40] Jonghwan Mun, Kimin Lee, Jinwoo Shin, and Bohyung Han. Learning to specialize with knowledge distillation for visual question answering. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. 2

[41] Houwen Peng, Hao Du, Hongyuan Yu, Qi Li, Jing Liao, and Jianlong Fu. Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. *CoRR*, abs/2010.15821, 2020. 2, 5, 7, 8

[42] Hieu Pham, Zihang Dai, Qizhe Xie, Minh-Thang Luong, and Quoc V. Le. Meta pseudo labels, 2021. 2

[43] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018. 2, 5

[44] Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. *arXiv preprint arXiv:1909.04630*, 2019. 6

[45] 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. 1, 2, 5, 7

[46] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000. 2

[47] 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. 2

[48] 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. 7

[49] Ilya Trofimov, Nikita Klyuchnikov, Mikhail Salnikov, Alexander Filippov, and Evgeny Burnaev. Multi-fidelity neural architecture search with knowledge distillation, 2021. 1, 2

[50] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, and Chunhua Shen. NAS-FCOS: fast neural architecture search for object detection. *CoRR*, abs/1906.04423, 2020. 2

[51] Runmin Wu, Mengyang Feng, Wenlong Guan, Dong Wang, Huchuan Lu, and Errui Ding. A mutual learning method for salient object detection with intertwined multi-supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8150–8159, 2019. 1, 2, 3

[52] 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. 1, 2

[53] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2019. 2, 8

[54] 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. 5, 7, 8

[55] Taojiannan Yang, Sijie Zhu, Chen Chen, Shen Yan, Mi Zhang, and Andrew Willis. Mutualnet: Adaptive convnet via mutual learning from network width and resolution. In *European Conference on Computer Vision*, pages 299–315. Springer, 2020. 1, 2, 3

[56] Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. Ista-nas: Efficient and consistent neural architecture search by sparse coding, 2020. 5

[57] Lewei Yao, Renjie Pi, Hang Xu, Wei Zhang, Zhenguo Li, and Tong Zhang. Joint-detnas: Upgrade your detector with nas, pruning and dynamic distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10175–10184, June 2021. 1

[58] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1285–1294, New York, NY, USA, 2017. Association for Computing Machinery. 2, 3, 6

[59] Sheheryar Zaidi, Arber Zela, Thomas Elsken, Chris Holmes, Frank Hutter, and Yee Whye Teh. Neural ensemble search for performant and calibrated predictions. *CoRR*, abs/2006.08573, 2020. 2, 5, 7, 8

[60] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*, 2019. 6

[61] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020. 5

[62] 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, 2018. 1, 2, 3, 5, 6, 7, 8

[63] Pan Zhou, Caiming Xiong, Richard Socher, and Steven C. H. Hoi. Theory-inspired path-regularized differential network architecture search. *CoRR*, abs/2006.16537, 2020. 5, 7

[64] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 1, 2

[65] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 2