Improving Bi-level Optimization Based Methods with Inspiration from Humans' Classroom Study Techniques

Pengtao Xie [1]

Abstract

In humans' classroom learning, many effective study techniques, e.g., the Feynman technique, peer questioning, have been developed to improve learning outcomes. We are interested in investigating whether these techniques can inspire the development of ML training strategies to improve bi-level optimization (BLO) based methods. Towards this goal, we develop a general framework, Skillearn, which consists of basic elements such as learners, interaction functions, learning stages, etc. These elements can be flexibly configured to create various training strategies, each emulating a study technique of humans. In case studies, we apply Skillearn to create new training strategies, by emulating the Feynman technique and peer questioning, which are two broadly adopted techniques in humans' classroom learning. These training strategies are used for improving two BLO based applications including neural architecture search and data weighting. Experiments on various datasets demonstrate the effectiveness of our methods.

1. Introduction

In classroom learning (Van Lier, 1991; Blumenfeld, 1992; Carpenter et al., 2018), humans have developed a lot of effective study techniques, such as the Feynman learning technique (Ambion et al., 2020), peer questioning (King, 1990), interleaving learning (Birnbaum et al., 2013), learning via self-explaining (Ainsworth & Th Loizou, 2003), etc. These techniques enable humans to learn faster and better (Dunlosky et al., 2013; Rovers et al., 2018; Biwer et al., 2020). We are interested in investigating whether humans' classroom study techniques can provide insights for improving bi-level optimization (BLO) based methods (Dempe, 2002; Feurer et al., 2015; Finn et al., 2017; Liu et al., 2019; Shu et al., 2019). These methods consist of two nested levels of optimization problems. At the lower level, model weights are learned by minimizing a training loss. At the upper level, meta parameters such as neural architectures (Liu et al., 2019), importance weights of data examples (Shu et al., 2019), hyperparameters (Feurer et al., 2015) are learned by minimizing a validation loss. BLO-based methods have been widely applied for differentiable neural architecture search (Liu et al., 2019), data reweighting (Shu et al., 2019), few-shot learning (Finn et al., 2017), etc. Existing BLO-based methods suffer from problems such as performance collapse (Zela et al., 2019; Chen & Hsieh, 2020a; Chu et al., 2021; 2020): the learned meta parameters and model weights perform well on validation and training data, but generalize poorly on test data. We aim to address these problems by developing novel training strategies drawing inspiration from humans' classroom study techniques.

Towards this goal, we propose a novel framework, Skillearn, which can create various ML training strategies by emulating humans' study techniques. In Skillearn, there are a set of learner models, each with a collection of weight parameters and meta parameters. Different learners interact with each other through interaction functions. The learning of all learners is organized into multiple stages, each involving a subset of learners. These stages have an order, but they are performed end-to-end in a multi-level optimization framework (Dempe, 2002) where latter stages influence earlier stages and vice versa.

The major contributions of this work are as follows.

- We propose a general framework, Skillearn, which can create training strategies to improve BLO-based methods by drawing inspirations from humans' classroom study techniques.
- We apply Skillearn to emulate two classroom learning techniques of humans including the Feynman technique (Ambion et al., 2020) and peer questioning (King, 1990) and create corresponding training strategies to improve BLO-based applications including neural architecture search and data reweighting.
- We demonstrate the effectiveness of our methods in various experiments.

[1]Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, USA. Correspondence to: Pengtao Xie <p1xie@ucsd.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

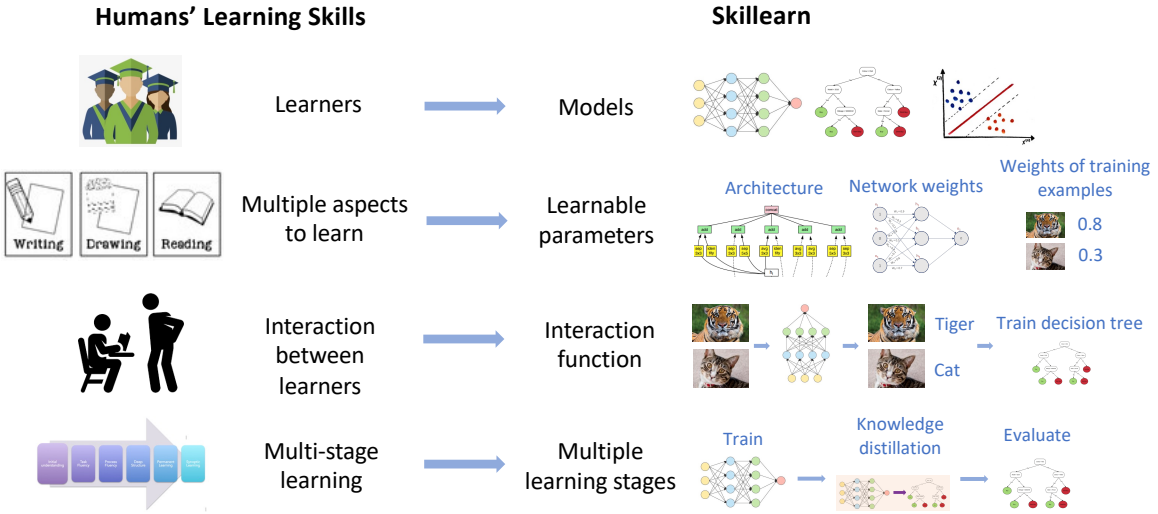

Figure 1. Analogy between humans' classroom studies and Skillearn.

2. Related Works

Bi-level optimization (BLO) and multi-level optimization. BLO (Dempe, 2002; Liu et al., 2021) has been broadly applied for hyperparameter tuning (Feurer et al., 2015), neural architecture search (NAS) (Liu et al., 2019), meta learning (Finn et al., 2017), data reweighting (Shu et al., 2019; Ren et al., 2020; Wang et al., 2020b), learning rate adjustment (Baydin et al., 2018), label denoising (Zheng et al., 2021), data generation (Such et al., 2020), etc. Our framework is orthogonal to existing BLO-based methods and can be leveraged to improve them. Multi-level optimization methods (Sato et al., 2021; Choe et al., 2022), which consist of more than two levels of nested optimization problems, have been developed for saliency-aware NAS (Hosseini & Xie, 2022), end-to-end data augmentation (Somayajula et al., 2022), performance-aware mutual knowledge distillation (Xie & Du, 2022), learning from mistakes (Garg et al., 2022), improving NAS by encouraging transferability (Sheth & Xie, 2023), etc.

Human-inspired learning. Developing ML methods by drawing inspiration from humans has been studied broadly. For example, curiosity-driven learning (Pathak et al.; Burda et al., 2019) is inspired by humans' psychology on curiosity. Few-shot learning (Lake et al., 2015; Sung et al., 2018) draws inspirations from cognitive science on how humans perform concept learning. In this paper, we focus on drawing inspirations from humans' study techniques in classroom learning, which have not been well-explored before. Curriculum learning (Bengio et al., 2009; Kumar et al., 2010; Jiang et al., 2014; Matiisen et al., 2019) is inspired by how human students learn a curriculum in classrooms. Knowledge distillation (Hinton et al., 2015; Tarvainen & Valpola, 2017; Xie et al., 2020) emulates the process of transferring knowledge from teachers to students. Our work aims to provide a unified framework to systematically formalize various classroom study techniques into ML training strategies and leverage them to train better BLO-based models.

3. The Skillearn Framework

We propose a general framework called Skillearn to formalize classroom study techniques of humans and leverage them for improving bi-level optimization based methods. Figure 1 shows an analogy between humans' classroom studies and Skillearn.

3.1. Elements of Skillearn

In Skillearn (Figure 2), we have the following elements.

- **Learners**. There could be one or multiple learners. Each learner is an ML model, such as a deep convolutional network (He et al., 2016), a deep generative model (Goodfellow et al., 2014), a nonparametric density estimator (Gramacki, 2018), etc. For example, in knowledge distillation (Hinton et al., 2015) from a teacher model to a student model, there are two learners: teacher and student. This is analogous to humans' classroom learning which involves one or multiple human learners (Blumenfeld, 1992), such as students, teachers, teaching assistants, etc.
- **Model weights and meta parameters**. Each learner has one or more sets of weight parameters and meta parameters. For example, in neural architecture search (Zoph & Le, 2017), a neural network learns not only model weights, but also architecture variables (which are meta parameters). This is analogous to humans' classroom learning where each learner learns multiple aspects in a learning task (Blumenfeld, 1992), such as comprehending, summarizing, writing articles, etc.
- **Interaction function**, which describes how two or more learners interact with each other. For example, in knowledge distillation (Hinton et al., 2015), given an unlabeled image dataset, a teacher model predicts pseudo labels of

these images; then a student model is trained using pseudo-labeled images. The interaction between the teacher and student is on the pseudo labels. As another example, given a set of texts, two text encoders A and B extract embeddings of the texts; A and B are tied together via distributional matching (Dziugaite et al., 2015): the distribution of embeddings extracted by A is encouraged to have small total-variance with the distribution of embeddings extracted by B. The interaction between A and B is on the distributions of embeddings. This is analogous to humans' classroom learning where multiple human learners interact with each other (Blumenfeld, 1992), such as a teacher teaches a student, a group of students mutually help each other to learn, etc.

- **Learning stages**. The learning of all learners is not conducted at one shot. Instead, learning is performed at multiple stages with an order. Each stage involves a subset of learners. For example, in knowledge distillation (Hinton et al., 2015), there are two stages: 1) a teacher model is trained; 2) the teacher predicts pseudo labels on an unlabeled dataset and the pseudo-labeled dataset is used to train a student model. The first stage involves a single learner, which is the teacher. The second stage involves two learners: teacher and student. This is analogous to humans' classroom learning where a learning process is divided into multiple stages (Blumenfeld, 1992). For example, studying a topic in classroom learning involves three stages: 1) teacher learns this topic; 2) teacher teaches this topic to students; 3) students take a quiz to evaluate how well they have mastered this topic. Mathematically, we formulate the learning at each stage as an optimization problem. Outcomes of one learning stage are passed to another learning stage via the interaction function.

3.2. Formulation

We assume there are K learning stages. The first $K-1$ stages focus on learning model weights on training datasets while the last stage focuses on learning meta parameters on validation datasets. At each stage, a subset of learners (referred to as active learners) are involved. For each active learner at stage 1 to $K-1$, some of its model weights are trained at this stage, which are called active weights. Meanwhile, its meta parameters are used to define a training loss function and interaction function, but are not updated at this stage. Let $\mathcal{W}_k$ and $\mathcal{A}_k$ denote active weights and meta parameters of all active learners at stage k. The learning activity at stage k ($1 \leq k \leq K-1$) is formulated as an optimization problem where optimization variables are active weights $\mathcal{W}_k$. The objective function involves a training loss L_k defined on a collection of training datasets $\mathcal{D}_k^{(\text{tr})}$ and an interaction function I_k that depicts the interaction between active learners. The interaction function is defined on a collection of auxiliary datasets $\mathcal{F}_k$, e.g., unlabeled datasets used for self-supervised pretraining (He et al., 2020b).

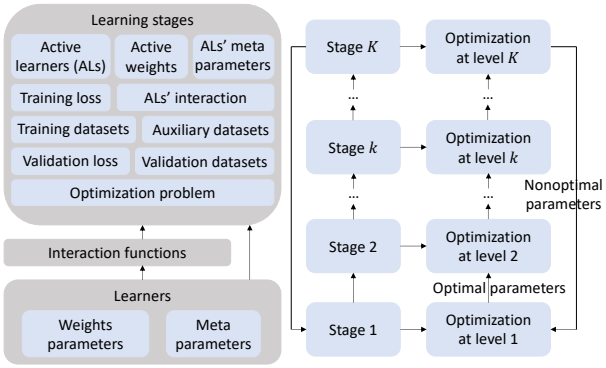

Figure 2. Skillearn consists of key elements including learners, weight and meta parameters, interaction functions, and learning stages. Skillearn is formulated as a multi-level optimization problem, where each learning stage corresponds to one level of optimization problem.

Table 1. Notations in the Skillearn framework.

Notation	Meaning
K	Number of learning stages
$\mathcal{W}_k$	Active weights of all active learners at stage k
$\mathcal{A}_k$	Active meta parameters of all active learners at stage k
L_k	Training loss at stage k
$\mathcal{D}_k^{(\text{tr})}$	Training datasets at stage k
I_k	Interaction function at stage k
$\mathcal{F}_k$	Auxiliary datasets at stage k
L_{val}	Validation loss
$\mathcal{D}^{(\text{val})}$	Validation datasets

The K stages are mutually dependent on each other. After completing the learning at stage k, we obtain optimally-trained active weights $\mathcal{W}_k^*$, which are passed to the next stage $k+1$ for defining a new objective function. At the last stage K, optimally-trained active weights $\{\mathcal{W}_k^*(\{\mathcal{A}_j\}_{j=1}^k)\}_{k=1}^{K-1}$ and meta parameters $\{\mathcal{A}_k\}_{k=1}^{K-1}$ at previous $K-1$ stages are used to define validation losses L_{val} on validation sets $\mathcal{D}^{(\text{val})}$. $\{\mathcal{A}_k\}_{k=1}^{K-1}$ are optimized by minimizing validation losses. There is no interaction function at stage K.

We perform these K stages end-to-end in a multi-level optimization (MLO) (Migdalas et al., 1998) based framework, to enable them mutually influence each other for achieving the overall best performance. An MLO formulation has multiple levels of nested optimization problems, where the optimal parameters of lower-level problems are the inputs of objective functions in upper-level problems, and the nonoptimal parameters of upper-level problems are the inputs of objective functions in lower-level problems. For the K mutually-dependent stages in Skillearn, we design an MLO formulation with K levels of optimization problems where each level corresponds to a stage. The order and relationships between stages are characterized by the dependency structure between levels. The K stages are conducted end-to-end by solving the K levels of nested optimization

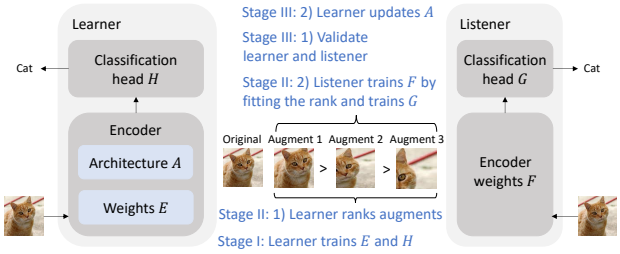

Figure 3. FAS has a learner, a listener, and three learning stages.

problems jointly. As such, the formulation of Skillearn is as follows:

$$\begin{aligned}
&\max_{\{\mathcal{A}_k\}_{k=1}^{K-1}} \quad L_{val}\left(\{\mathcal{W}_k^*(\{\mathcal{A}_j\}_{j=1}^k)\}_{k=1}^{K-1}, \{\mathcal{A}_k\}_{k=1}^{K-1}, \mathcal{D}^{(\text{val})}\right)\\
&s.t. \quad \cdots\\
&\mathcal{W}_k^*\left(\{\mathcal{A}_j\}_{j=1}^k\right) = \underset{\mathcal{W}_k}{\text{argmin}}\, L_k\left(\mathcal{W}_k, \mathcal{A}_k, \mathcal{W}_{k-1}^*\left(\{\mathcal{A}_j\}_{j=1}^{k-1}\right), \mathcal{D}_k^{(\text{tr})}\right)\\
&\qquad +\gamma_k I_k\left(\mathcal{W}_k, \mathcal{A}_k, \mathcal{W}_{k-1}^*\left(\{\mathcal{A}_j\}_{j=1}^{k-1}\right), \mathcal{F}_k\right)\\
&\cdots\\
&\mathcal{W}_1^*\left(\mathcal{A}_1\right) = \text{argmin}_{\mathcal{W}_1}\, L_1\left(\mathcal{W}_1, \mathcal{A}_1, \mathcal{D}_1^{(\text{tr})}\right) + \gamma_1 I_1(\mathcal{W}_1, \mathcal{A}_1, \mathcal{F}_1)
\end{aligned} \tag{1}$$

where $\{\gamma_k\}_{k=1}^{K-1}$ are tradeoff parameters between training losses and interaction functions. From bottom to top, optimization problems correspond to learning stages $1, \cdots, K$. The optimal model weights $\mathcal{W}_k^*(\{\mathcal{A}_j\}_{j=1}^k)$ at stage k depend on $\{\mathcal{A}_j\}_{j=1}^k$ since $\mathcal{W}_k^*$ depends on the objective function and the objective is a function of $\{\mathcal{A}_j\}_{j=1}^k$. The optimal parameters $\mathcal{W}_{k-1}^*$ at level $k-1$ are the inputs of the objective function at level k. The nonoptimal parameters $\{\mathcal{A}_k\}_{k=1}^{K-1}$ at level K are the inputs of the losses at level 1 to $K-1$. Table 1 summarizes the notations in Skillearn. The optimization algorithm is provided in the appendix.

4. Case Study I: Feynman Architecture Search

We instantiate the Skillearn framework to solve a specific BLO-based problem - neural architecture search (NAS) (Zoph & Le, 2017; Liu et al., 2019), motivated by the Feynman technique (Ambion et al., 2020). The Feynman learning technique is a broadly used effective study technique in humans' classroom learning. It consists of three steps: 1) a learner learns a topic; 2) the learner illustrates his/her understanding of this topic to a listener who has little prior knowledge about this topic; and 3) the learner identifies gaps in his/her understanding and re-learns the topic to fill in these gaps. NAS aims to automatically search for well-performing neural architectures. Most existing NAS methods (Zoph & Le, 2017; Pham et al., 2018; Liu et al., 2019) adopt a bi-level optimization based formulation where architectures are learned by minimizing a validation loss.

4.1. Overview

Assume there is a machine learner which aims to search for a neural architecture to perform a target task. Without loss of generality, we assume the target task is learning image representations that are good for classification. The learner has 1) an image encoder (He et al., 2016) which takes an image as input and extracts a visual representation of this image, and 2) a classification head which takes the representation extracted by the encoder as input and predicts the class label of this image. The encoder has a learnable architecture A (Liu et al., 2019) (which are meta parameters) and model weights E. The classification head has a fixed architecture (designed by humans) and model weights H. The learner has access to an image classification dataset D where each image is annotated with a class label.

Table 2. Notations in Feynman architecture search.

Notation	Meaning
A	Architecture of learner's encoder
E	Weight parameters of learner's encoder
H	Learner's classification head
F	Listener's encoder
G	Listener's classification head
$D^{(\text{tr})}$	Training dataset
$D^{(\text{val})}$	Validation dataset

Emulating humans' Feynman technique, the learner model "illustrates" its understanding of images to a listener model and the listener helps the learner to identify gaps in learner's understanding. The listener has an encoder F and a classification head G, whose architectures are fixed (manually designed by humans). To "illustrate" its understanding of an image x to the listener, the learner creates augmented images from x and uses its encoder to "tell" the listener which augmented image is more similar to the original image. To help the learner identify gaps in image understanding, the listener leverages the illustrations presented by the learner to train its encoder and evaluates its trained encoder on a human-annotated validation set. If validation performance is not good, it means that the illustrations are not accurate, which indicates that the learner has large gaps in understanding image contents. Then the learner tries to bridge these gaps by improving its encoder architecture.

We leverage the proposed Skillearn framework to formalize the Feynman technique of humans and obtain a Feynman architecture search (FAS) method (Figure 3). Under the Skillearn terminology, FAS has two learners: a learner and a listener. The learner's model weights include encoder weights and classification head weights. The learner's meta parameters are encoder architecture. The listener has no meta parameters. Its model weights include encoder weights and classification head weights. In an interaction function, the learner "illustrates" its understanding of image contents to the listener. FAS has three learning stages: 1) the learner learns image representations; 2) the learner "illustrates" its understanding of images to the listener; and 3) the learner bridges gaps in its understanding. FAS is formulated as a three-level optimization framework where three learning stages are performed end-to-end. Table 3 shows how to instantiate Skillearn to FAS. Table 2 summarizes notations.

Table 3. Instantiation of Skillearn to FAS.

Skillearn	Feynman architecture search
Learners	1) Learner; 2) Listener
Model weights and meta parameters	1) Architecture A of learner's encoder; 2) Weight parameters E of learner's encoder; 3) Learner's classification head H; 4) Listener's encoder F; 5) Listener's classification head G
Interaction function	The learner ranks augmented images and the listener trains its encoder by fitting the ranking. $\sum_{i=1}^{N} \sum_{1 \leq k < j \leq K} \max\Big(0, -\big(f(a_{ik}, x_i; A, E^*(A)) - f(a_{ij}, x_i; A, E^*(A))\big)\big(f(a_{ik}, x_i; F) - f(a_{ij}, x_i; F)\big)\Big)$.
Learning stages	• Learning stage I: The learner trains its encoder. • Learning stage II: The learner ranks augmented images and the listener trains its encoder by fitting the ranking. • Learning stage III: The learner and listener validate their classification models; the learner's encoder architecture is updated by minimizing validation losses.

4.2. Learning Stages

Next, we discuss the details of the three stages (Figure 5 in the appendix). At the first stage, the learner trains its encoder weights E and classification head H by minimizing a classification loss L (e.g., cross entropy) on the training set $D^{(\text{tr})}$ of D, with its encoder architecture A tentatively fixed. Under the Skillearn terminology, at this stage, the learner is active and the listener is inactive. Active weights include E and H. Meta parameters include A. There is no interaction function. The optimization problem is:

$$E^*(A), H^*(A) = \text{argmin}_{E,H} \; L\left(A, E, H, D^{(\text{tr})}\right). \quad (2)$$

At this stage, A cannot be updated by minimizing the training loss. Otherwise, a degenerate solution will be produced where A has a very large capacity to overfit training examples but will yield poor prediction outcomes on test data. We will discuss how to learn A at a later stage. $E^*(A)$ denotes that E^* depends on A since E^* depends on the loss $L(A, E, H, D^{(\text{tr})})$ which depends on A.

At the second stage, the learner illustrates its understanding of image contents to the listener. The understanding is about the similarity between augmented images and original images. Specifically, the learner ranks augmented images in descending order of their similarities to original images; the listener trains its encoder by fitting the ranking. Given an original image x_i from $D^{(\text{tr})}$, K augmented images $\{a_{ik}\}_{k=1}^K$ are generated from x_i, by applying different augmentation operations (Perez & Wang, 2017) such as rotation, flipping, cropping, etc. The learner uses its trained encoder including A and $E^*(A)$ to rank $\{a_{ik}\}_{k=1}^K$ based on their similarity to x_i. Let $f(a_{ik}, x_i; A, E^*(A))$ denote the cosine similarity between a_{ik} and x_i measured on their representations extracted by the learner's encoder. If $f(a_{ik}, x_i; A, E^*(A)) > f(a_{ij}, x_i; A, E^*(A))$, then a_{ik} is ranked higher than a_{ij}. Let $o_i(1; A, E^*(A)) \succ \cdots \succ o_i(k; A, E^*(A)) \succ \cdots \succ o_i(K; A, E^*(A))$ be the ranking of $\{a_{ik}\}_{k=1}^K$, where $o_i(k; A, E^*(A))$ denotes the augmented image ranked at the k-th position. $o_i(1; A, E^*(A))$ is the augmented image that is the most similar to x_i.

Given this ranking, the listener trains its encoder by fitting the ranking. Let $f(a_{ik}, x_i; F)$ denote the cosine similarity between a_{ik} and x_i calculated on their representations extracted by the listener's encoder F. To fit the ranking, the listener trains F such that $f(o_i(k; A, E^*(A)); F)$ decreases when k increases. In other words, if $o_i(k; A, E^*(A)) \succ o_i(j; A, E^*(A))$ in the ranking, then the listeners makes $f(o_i(k; A, E^*(A)); F)$ larger than $f(o_i(j; A, E^*(A)); F)$. In addition to training its encoder F by fitting the ranking, the listener trains F and classification head G by minimizing a cross-entropy based classification loss L on the training data $D^{(\text{tr})}$. To this end, the second stage amounts to solving the following problem:

$$\begin{aligned} &F^*(A, E^*(A)), G^* = \text{argmin}_{F,G} \; L\left(F, G, D^{(\text{tr})}\right) \\ &s.t. \; \forall i, f(o_i(1; A, E^*(A)), x_i; F) > \cdots > f(o_i(K; A, E^*(A)), x_i; F) \end{aligned} \quad (3)$$

At this stage, the learner and listener are both active. Active weights include F and G. The learner and listener interact on the rankings in the constraints in Eq.(3).

At the third stage, the listener evaluates how its trained classification model including $F^*(A, E^*(A))$ and G^* performs on the validation set $D^{(\text{val})}$ of D. A high validation loss $L(F^*(A, E^*(A)), G^*, D^{(\text{val})})$ indicates the listener's model is not good. Since the model is trained using the rankings generated by the learner, it implies that these rankings are not accurate, i.e., there are gaps in the learner's understanding of image contents. To bridge these gaps, the learner updates its encoder architecture to minimize the listener's validation loss. In addition, the learner minimizes its own validation loss as well. This amounts to solving the following optimization problem, where γ is a tradeoff parameter.

$$\min_A \; L\left(A, E^*(A), H^*(A), D^{(\text{val})}\right) + \gamma L\left(F^*(A, E^*(A)), G^*, D^{(\text{val})}\right). \quad (4)$$

Putting these pieces together, we have the formulation of FAS, which is a special case of Skillearn:

$$\begin{aligned} &\min_A \; L\left(A, E^*(A), H^*(A), D^{(\text{val})}\right) + \gamma L\left(F^*(A, E^*(A)), G^*, D^{(\text{val})}\right) \\ &s.t. \; F^*(A, E^*(A)), G^* = \text{argmin}_{F,G} \; L\left(F, G, D^{(\text{tr})}\right) \\ &\quad s.t. \; \forall i, f(o_i(1; A, E^*(A)), x_i; F) > \cdots > f(o_i(K; A, E^*(A)), x_i; F) \\ &\quad s.t. E^*(A), H^*(A) = \text{argmin}_{E,H} \; L\left(A, E, H, D^{(\text{tr})}\right) \end{aligned} \quad (5)$$

The optimization algorithm is provided in the appendix.

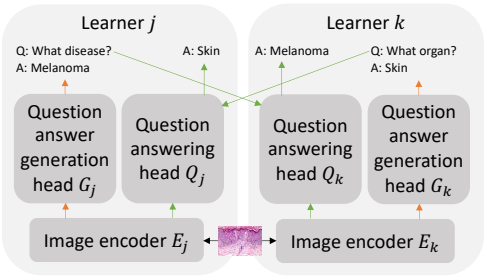

Figure 4. PQDR has K peer learners and three learning stages.

Table 4. Notations in peer-questioning based data reweighting.

Notation	Meaning
K	The number of peer learners
E_k	Image encoder of learner k
G_k	Question-answer generation head of learner k
Q_k	Question-answering head of learner k
D_{qag}	Dataset for training question-answer generation models
A	Weights of data examples in D_{qag}
I_u	Unlabeled images
$D_{qa}^{(val)}$	Validation dataset for question answering

5. Case Study II: Data Reweighting by Peer Questioning

In this section, we instantiate Skillearn to solve another BLO based problem - data reweighting (Ren et al., 2018; Shu et al., 2019), by emulating a peer questioning (King, 1990) study technique of humans. In peer questioning (King, 1990), a group of peer learners collaboratively study a topic and ask each other questions about this topic. By raising questions to peers and answering questions raised by peers, learners can enhance their understanding of this topic thoroughly (King, 1990). Data reweighting aims to automatically learn a weight for each training example where a smaller weight indicates that this example is more noisy. With data reweighting, noisy examples are removed from the training process since their losses are down-weighted to being close to zero. Many data reweighting methods (Ren et al., 2018; Shu et al., 2019) utilize a BLO-based formulation where data weights (which are meta parameters) are learned by minimizing a validation loss in the upper-level problem and model weights are learned by minimizing a training loss in the lower-level problem.

5.1. Overview

Leveraging Skillearn, we propose a peer-questioning based data reweighting (PQDR) method (Figure 4). Table 9 in the appendix shows the instantiation of Skillearn to PQDR. PQDR has K peer learner models, which learn to perform the same task. We assume the task is visual question answering (Fukui et al., 2016): given an image and a textual question asked about the content of this image, predict an answer. Each learner learns to perform two sub-tasks: question-answer generation (QAG) and question answering (QA). In QAG, given an image, the learner generates a question from this image, and then generates an answer for this question. In QA, given an image and a question, the learner predicts an answer. Each learner k has 1) an image encoder E_k which extracts visual representations of images; 2) a QAG head G_k which takes the visual representation of an image as input, generates a question, and then generates an answer for this question; and 3) a QA head (Fukui et al., 2016) Q_k which takes the visual representation of an image and a question about this image as inputs and predicts an answer. The image encoder is shared by the QAG head and the QA head. Given a dataset containing (image, question, answer) triples, we split it into two subsets. The first subset D_{qag} is used for training question-answer generation models where images are inputs and (question, answer) pairs are outputs. The second subset D_{qa} is used for training question answering models where (image, question) pairs are inputs and answers are outputs. D_{qag} contains noisy examples. An (image, question, answer) triple is considered a noisy example if any of the following conditions holds: 1) the image is not about pathology; 2) the image has human annotated artifacts such as arrows, texts, circles, etc.; and 3) the question is not correct in syntax and semantics. Figure 11 shows some examples. For each example in D_{qag}, we learn a weight for it, to identify and down-weight noisy examples using the weights. D_{qa} is manually checked to ensure there is no noisy data. It is split into a validation set $D_{qa}^{(val)}$ and a test set $D_{qa}^{(test)}$. Table 4 shows notations.

5.2. Learning Stages

In PQDR, there are three learning stages. At the first stage, each learner k independently learns to generate question-answer pairs from images, by training its encoder weights E_k and question-answer generation head G_k on D_{qag}. Each example d_i in $D_{qag} = \{d_i\}_{i=1}^N$ is associated with a weight $a_i \in [0, 1]$, which is used to reweight the training loss on d_i. A smaller a_i indicates that d_i is more noisy. Let $L_{qag}(\cdot)$ denote a question-answer generation loss (e.g., negative log-likelihood of questions and answers) and A denote $\{a_i\}_{i=1}^N$. The optimization problem is:

$$\{E_k^*(A), G_k^*(A)\}_{k=1}^K = \underset{\{E_k, G_k\}_{k=1}^K}{\text{argmin}} \sum_{k=1}^K \sum_{i=1}^N a_i L_{qag}(E_k, G_k, d_i). \tag{6}$$

Note that the data weights A cannot be learned at this stage; otherwise a degenerate solution will be yielded where all weights are zero. At this stage, all K learners are active. Active weights include the image encoder and question-answer generation head of each learner. Meta parameters are A. There is no interaction function.

At the second stage, peer questioning is conducted, where each learner k uses its trained question-answer generation model consisting of $E_k^*(A)$ and $G_k^*(A)$ to generate questions $\mathcal{Q}(I_u, E_k^*(A), G_k^*(A))$ from unlabeled images I_u, and

then generate answers $\mathcal{A}(I_u, E_k^*(A), G_k^*(A))$ for these questions. Meanwhile, each learner k uses its QA model to answer the questions $\{\mathcal{Q}(I_u, E_j^*(A), G_j^*(A))\}_{j\neq k}^K$ generated by other learners. Then learner k measures the discrepancy between its predicted answers and the answers $\{\mathcal{A}(I_u, E_j^*(A), G_j^*(A))\}_{j\neq k}^K$ generated by other learners and trains its question-answering head Q_k by minimizing this discrepancy. Let $L_{qa}(\cdot)$ denote a question-answering (QA) loss (e.g., negative log-likelihood of answers). The optimization problem is:

$$\begin{aligned}&\{Q_k^*(\{E_j^*(A), G_j^*(A)\}_{j\neq k}^K, E_k^*(A))\}_{k=1}^K = \\ &\text{argmin}_{\{Q_k\}_{k=1}^K} \sum_{k=1}^K \sum_{j\neq k}^K L_{qa}(E_k^*(A), Q_k, \\ &\mathcal{Q}(I_u, E_j^*(A), G_j^*(A)), \mathcal{A}(I_u, E_j^*(A), G_j^*(A))).\end{aligned} \tag{7}$$

At this stage, all learners are active. Active weights include all learners' QA heads. The above objective is an interaction function where each learner raises questions to other learners and answers questions raised by other learners.

At the third stage, we use $D_{qa}^{(val)}$ as a validation set to evaluate the performance of QA models of all learners. Data weights of D_{qaq} are updated by minimizing validation losses. The corresponding optimization problem is:

$$\min_A \sum_{k=1}^K L_{qa}\left(E_k^*(A), Q_k^*(\{E_j^*(A), G_j^*(A)\}_{j\neq k}^K, E_k^*(A)), D_{qa}^{(val)}\right). \tag{8}$$

Putting these pieces together, we have the following overall formulation for PQDR, which is a special case of the general Skillearn framework:

$$\begin{aligned}&\min_A \ \sum_{k=1}^K L_{qa}\left(E_k^*(A), Q_k^*(\{E_j^*(A), G_j^*(A)\}_{j\neq k}^K, E_k^*(A)), D_{qa}^{(val)}\right) \\ &s.t.\ \{Q_k^*(\{E_j^*(A), G_j^*(A)\}_{j\neq k}^K, E_k^*(A))\}_{k=1}^K = \text{argmin}_{\{Q_k\}_{k=1}^K} \sum_{k=1}^K \sum_{j\neq k}^K \\ &\quad L_{qa}(E_k^*(A), Q_k, \mathcal{Q}(I_u, E_j^*(A), G_j^*(A)), \mathcal{A}(I_u, E_j^*(A), G_j^*(A))) \\ &s.t.\ \{E_k^*(A), G_k^*(A)\}_{k=1}^K = \underset{\{E_k, G_k\}_{k=1}^K}{\text{argmin}} \sum_{k=1}^K \sum_{i=1}^N a_i L_{qag}(E_k, G_k, d_i).\end{aligned} \tag{9}$$

6. Experiments

In this section, we present experimental results for the two case studies. Please refer to the appendix for detailed hyperparameter settings and additional results.

6.1. Experiments on Feynman Architecture Search

We applied FAS to search for architectures in image classification tasks. We followed the experimental protocol in (Liu et al., 2019), consisting of two phases: one for architecture search and the other for architecture evaluation. In the search phase, an optimal cell is searched by minimizing a validation loss. In the evaluation phase, the searched cell is replicated and composed into a large network, which is trained from scratch on training and validation sets. Its performance is reported on a test set. In Section I.1 in the appendix, we applied FAS to six text classification datasets where FAS achieves better text classification accuracy than baselines.

6.1.1. Datasets

We used three image classification datasets: CIFAR-10, CIFAR-100, and ImageNet (Deng et al., 2009), with 10, 100, and 1000 classes respectively. For CIFAR-10 and CIFAR-100, each of them is split into a 25K training set, a 25K validation set, and a 10K test set. ImageNet is split into a training set of 1.2M images and a test set of 50K images. For architecture search on ImageNet, following (Xu et al., 2020), we randomly sample 10% of the 1.2M images as a training set, and randomly sample 2.5% of the 1.2M images as a validation set.

6.1.2. Experimental Setup

For the learner's encoder architecture, we experimented with the search spaces in DARTS (Liu et al., 2019), P-DARTS (Chen et al., 2019), PC-DARTS (Xu et al., 2020), and PR-DARTS (Zhou et al., 2020). The learner's head is a linear classifier. The listener is set to ResNet-50 (He et al., 2016), containing an encoder and a linear classification head. Image encodings generated by the learner and listener have different dimensions. The number K of augmented images is set to 5. Augmentation operations (Perez & Wang, 2017) include random rotation, flipping, cropping, and color jitter. The tradeoff parameters γ and λ are set to 1 and 0.5 respectively.

We compared with the following baselines: 1) multi-task learning (MTL) (Maninis et al., 2019) and 2) shared encoder (SE) (Kokkinos, 2017). MTL solves a bi-level optimization problem. At the lower level, we perform two tasks simultaneously by minimizing the weighted sum of their losses (loss weights are set to 1). The first task is training the weight parameters of the learner by minimizing a classification loss defined on $D^{(tr)}$. The second task is training the weight parameters of the listener by fitting the ranking of augmented images where the ranking is generated by the learner and by minimizing a classification loss defined on $D^{(tr)}$. At the upper level, we update the architecture of the learner by minimizing the validation losses of the learner and listener. The SE baseline is similar to MTL, except that the learner and listener in SE share the same encoder architecture (but with different weight parameters). The formulations of MTL and SE are provided in Section B.6 and B.7 in the appendix.

Each experiment was repeated ten times with random seeds. Mean and standard deviation of the 10 runs are reported. Experiments were conducted on Nvidia 1080Ti GPU.

6.1.3. Results

Table 5 shows results on the test set of CIFAR-100 and CIFAR-10, where we make the following observations. **First**, applying our method to DARTS, P-DARTS, PC-DARTS, and PR-DARTS reduce classification errors significantly, which demonstrates that our method is effective

Table 5. Results on CIFAR-100 (C100) and CIFAR-10 (C10), including classification error (%) on the test set, number of model parameters (millions), and search cost (GPU days). In entries with an X/Y format, X and Y denote results on CIFAR-100 and CIFAR-10 respectively. In SE, MTL, and FAS, parameter number is with respect to the searched architecture in the learner only, not including the listener. FAS-darts2nd represents that FAS' search space is the same as that of DARTS-2nd. Similar meanings hold for other notations in such a format. * denotes that the results are taken from DARTS$^-$ (Chu et al., 2021). † denotes that we re-ran this method for 10 times. Search cost is measured by GPU days on a Nvidia 1080Ti.

Method	Error-C100	Error-C10	Param.	Cost
*ResNet (He et al., 2016)	22.10	6.43	1.7/1.7	-/-
*DenseNet (Huang et al., 2017)	17.18	3.46	25.6/25.6	-/-
*PNAS (Liu et al., 2018a)	19.53	3.41±0.09	3.2/3.2	150/150
*ENAS (Pham et al., 2018)	19.43	2.89	4.6/4.6	0.5/0.5
*AmoebaNet (Real et al., 2019)	18.93	2.55±0.05	3.1/3.1	3150/3150
*GDAS (Dong & Yang, 2019b)	18.38	2.93	3.4/3.4	0.2/0.2
*R-DARTS (Zela et al., 2019)	18.01±0.26	2.95±0.21	-/-	1.6/1.6
*DARTS+PT (Wang et al., 2021)	-	2.61±0.08	-/3.0	-/0.8
*DARTS$^-$ (Chu et al., 2021)	17.51±0.25	2.59±0.08	3.3/3.3	0.4/0.4
*DropNAS (Hong et al., 2020)	16.95±0.41	2.58±0.14	4.4/4.1	0.7/0.6
*DrNAS (Chen et al., 2021a)	-	2.54±0.03	-/4.0	-/0.4
*ISTA-NAS (Yang et al., 2020)	-	2.54±0.05	-/3.3	-/0.1
*AGNAS (Sun et al., 2022)	-	2.53±.003	-/3.6	-/0.4
*MiLeNAS (He et al., 2020a)	-	2.51±0.11	-/3.9	-/0.3
*GAEA (Li et al., 2021)	-	2.50±0.06	-/-	-/0.1
*GAEA (Li et al., 2021)	-	2.50±0.06	-/-	-/0.1
*DOTS (Gu et al., 2021)	16.48±0.13	2.49±0.06	4.1/3.5	0.3/0.3
*β-DARTS (Ye et al., 2022)	16.24±0.22	2.53±0.08	3.8/3.8	0.4/0.4
*PDARTS-ADV (Chen & Hsieh, 2020a)	-	2.48±0.02	-/3.4	-/1.1
*Darts2nd (Liu et al., 2019)	20.58±0.44	2.76±0.09	3.1/3.3	4.0/4.0
SE-darts2nd (Kokkinos, 2017)	19.30±0.38	2.74±0.08	3.4/3.4	5.3/5.3
MTL-darts2nd (Maninis et al., 2019)	18.42±0.27	2.79±0.14	3.2/3.4	5.2/5.2
FAS-darts2nd (ours)	**17.12**±0.18	**2.60**±0.05	3.5/3.4	5.3/5.3
*Pdarts (Chen et al., 2019)	17.42±0.14	2.54±0.04	3.6/3.5	0.3/0.3
SE-pdarts (Kokkinos, 2017)	17.36±0.16	2.62±0.11	3.7/3.7	0.7/0.7
MTL-pdarts (Maninis et al., 2019)	16.95±0.12	2.71±0.08	3.7/3.8	0.7/0.7
FAS-pdarts (ours)	**16.01**±0.09	**2.49**±0.06	3.6/3.5	0.7/0.7
†Pcdarts (Xu et al., 2020)	17.96±0.15	2.57±0.07	3.9/3.6	0.1/0.1
SE-pcdarts (Kokkinos, 2017)	17.81±0.09	2.79±0.08	3.9/3.7	0.3/0.3
MTL-pcdarts (Maninis et al., 2019)	17.73±0.15	2.64±0.05	3.8/3.7	0.3/0.3
FAS-pcdarts (ours)	**16.41**±0.12	**2.51**±0.02	3.8/3.8	0.3/0.3
†Prdarts (Zhou et al., 2020)	16.48±0.06	2.37±0.03	3.4/3.5	0.2/0.2
SE-prdarts (Kokkinos, 2017)	17.04±0.09	2.49±0.06	3.6/3.6	0.4/0.4
MTL-prdarts (Maninis et al., 2019)	16.85±0.15	2.55±0.07	3.4/3.4	0.4/0.4
FAS-prdarts (ours)	**16.12**±0.08	**2.32**±0.03	3.5/3.4	0.4/0.4

in searching for better neural architectures. The reason is: by encouraging the learner to clearly illustrate what it has learned to a listener, the learner can identify its performance gaps and improve itself by bridging these gaps. The learner's architecture is updated by minimizing the listener's validation loss, which indirectly measures the learner's performance. If this loss is high, it indicates that the listener's model is unsatisfactory. Since the listener's model is trained by fitting the ranking generated by the learner, the listener's inferior performance indicates that the generated ranking is not accurate, which implies that the learner's encoder is not accurate. Given this identified performance gap, the learner adjusts its encoder architecture to bridge the gap.

Using the listener's validation loss as a regularizer, the learner's architecture is more robust to performance collapse, because this architecture is required to be effective not only for the learner's classification task, but also for the listener's classification task. To empirically verify this, we evaluate our method on four search spaces (Zela et al., 2019) designed for evaluating architectures' robustness against performance collapse. Following the protocol in RobustDARTS-L2 (Zela et al., 2019), cell number and initial channel number is set to 8 and 16. For each method, the search process runs 4 times with random initialization. For each searched architecture, it is retrained from scratch for several epochs. The architecture (after retraining) with the highest validation accuracy is evaluated on test set. Table 6 shows test results. As can be seen, our method achieves lower classification errors than baseline methods which are specifically developed for alleviating performance collapse. These results demonstrate our method's resilience to performance collapse.

Table 6. Evaluation of robustness against performance collapse: test errors on CIFAR-10 (C10) and C100. We compare with the following baselines: Darts (Liu et al., 2019), RDartsL2 (Zela et al., 2019), DartsES (Zela et al., 2019), Darts- (Chu et al., 2021), SDarts (Chen & Hsieh, 2020a), and MTL (Maninis et al., 2019).

Data	Space	DARTS [1]	RDartsL2 [2]	DartsES [3]	Darts- [4]	SDarts [5]	MTL [6]	Ours
C10	S1	4.69	3.46	3.93	3.34	3.26	4.39	**3.12**
	S2	5.54	3.31	4.07	4.03	3.11	4.11	**2.94**
	S3	3.92	2.51	3.55	2.95	3.07	4.61	**2.44**
	S4	8.33	3.56	4.69	4.14	3.49	6.05	**3.32**
C100	S1	29.46	24.25	28.37	22.41	22.33	27.85	**21.63**
	S2	26.05	22.24	23.25	21.61	20.56	24.25	**19.72**
	S3	28.90	23.99	23.73	21.13	21.08	21.40	**20.35**
	S4	22.85	21.94	21.26	21.55	21.25	22.97	**20.51**

Second, our method performs better than MTL. MTL performs the two tasks of Stage I and Stage II of our method simultaneously without considering their order, where the learner and listener are trained by minimizing the weighted sum of their losses. This incurs a competition between them: more decrease of the learner's loss leads to less decrease of the listener's loss; vice versa. In contrast, our framework trains these two models sequentially in two stages (but still within an end-to-end framework). The listener helps the learner to improve instead of competing with it. **Third**, our method outperforms SE. Similar to MTL, SE breaks the inherent order between tasks. **Fourth**, while our FAS method can help to search for better architectures, it does not substantially increase model size or search cost.

Table 7 shows results on ImageNet, where we make similar observations as in Table 5. Applying FAS to DARTS-2nd, P-DARTS, and PC-DARTS reduces their errors. FAS outperforms MTL and SE. These results further demonstrate the effectiveness of FAS which emulates humans' Feynman technique for improving learning outcomes.

6.2. Peer-Questioning Data Reweighting Experiments

6.2.1. Datasets

We used the PathVQA (He et al., 2021) dataset, which consists of 32,795 question-answer pairs generated from 1,670 pathology images. We split this dataset into D_{qag} and D_{qa} with a ratio of 1:1. The data in D_{qa} is manually checked to ensure there is no noise. D_{qa} is further split into a validation set $D_{qa}^{(val)}$ and a test set with a ratio of 3:1. We

Table 7. Results on ImageNet, including top-1 and top-5 classification errors on the test set, number of model weights (millions), and search cost (GPU days). * denotes that the results are taken from DARTS$^-$ (Chu et al., 2021) and DrNAS (Chen et al., 2021a). The rest notations are the same as those in Table 5. FAS-darts2nd-cifar10 means the architecture is searched using FAS on CIFAR-10, where the search space is the same as that in DARTS-2nd. Similar meanings hold for other notations like this.

Method	Top-1	Top-5	Param.	Cost
*Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	-
*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
*AmoebaNet-C (Real et al., 2019)	24.3	7.6	6.4	3150
*SDARTS-ADV-CIFAR10 (Chen & Hsieh, 2020a)	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-ImageNet (Chu et al., 2020)	24.4	7.4	4.3	3.0
*DOTS (Gu et al., 2021)	24.3	7.4	5.2	0.2
*PR-DARTS-cifar10 (Zhou et al., 2020)	24.1	7.3	5.0	0.2
*β-DARTS (Ye et al., 2022)	23.9	7.0	5.5	0.4
*DARTS$^+$-CIFAR100 (Liang et al., 2019)	23.7	7.2	5.1	0.2
*AGNAS (Sun et al., 2022)	23.4	6.8	6.7	3.3
*Darts2nd-cifar10 (Liu et al., 2019)	26.7	8.7	4.7	4.0
SE-darts2nd-cifar10 (Kokkinos, 2017)	26.3	8.5	4.9	5.3
MTL-darts2nd-cifar10 (Maninis et al., 2019)	26.1	8.2	4.9	5.2
FAS-darts2nd-cifar10 (ours)	**25.3**	**7.7**	4.9	5.3
*Pdarts-cifar10 (Chen et al., 2019)	24.4	7.4	4.9	0.3
SE-pdarts-cifar10 (Kokkinos, 2017)	24.3	7.4	5.1	0.7
MTL-pdarts-cifar10 (Maninis et al., 2019)	24.3	7.3	5.1	0.7
FAS-pdarts-cifar10 (ours)	**23.9**	**7.1**	4.9	0.7
*Pdarts-cifar100 (Chen et al., 2019)	24.7	7.5	5.1	0.3
SE-pdarts-cifar100 (Kokkinos, 2017)	24.7	7.5	5.3	0.7
MTL-pdarts-cifar100 (Maninis et al., 2019)	24.6	7.5	5.2	0.7
FAS-pdarts-cifar100 (ours)	**24.3**	**7.3**	4.9	0.7
*Pcdarts-imagenet (Xu et al., 2020)	24.2	7.3	5.3	3.8
SE-pcdarts-imagenet (Kokkinos, 2017)	24.0	7.1	5.4	5.2
MTL-pcdarts-imagenet (Maninis et al., 2019)	23.7	7.0	5.5	5.2
FAS-pcdarts-imagenet (ours)	**23.2**	**6.5**	5.5	5.2

used input images in $D_{qa}^{(val)}$ as the unlabeled set I_u.

6.2.2. Experimental Settings

We set the number of models K to 3. For the QA model (containing an image encoder and a QA head), we experimented with two choices: LXMERT (Tan & Bansal, 2019) and BAN (Kim et al., 2018). In the QAG model, the image encoder is the same as that in the QA model; the question and answer generation head is set to an LSTM (Hochreiter & Schmidhuber, 1997) with a hidden size of 128. We used three evaluation metrics: 1) accuracy (Malinowski & Fritz, 2014), 2) BLEU (Papineni et al., 2002), and 3) macro-averaged F1 (Goutte & Gaussier, 2005). We compared with the following baselines: 1) multi-task learning (MTL) (Maninis et al., 2019): question-answer generation and question answering are performed jointly by minimizing the sum of their losses; 2) perform QAG and QA separately (Separate): we first train a QAG model, fix it, and use it to generate question-answer pairs; then we use the generated QA pairs to train a QA model; 3) no reweighting of D_{qag} (No-Weight): all examples in D_{qag} are used for training the QAG model.

6.2.3. Results

Table 8 shows the results, where we make the following observations. **First**, our method outperforms vanilla LXMERT and BAN on all metrics, which demonstrates the effectiveness of our proposed peer-questioning mechanism. Through peer-questioning, each learner can effectively identify the weakness of its model and improve itself accordingly. Each learner boosts its QA performance by answering questions raised by other learners and boosts its QAG performance by raising questions to other learners. **Second**, our method performs better than MTL. MTL performs question-answer generation and question-answering simultaneously by minimizing the sum of their training losses, which may incur a conflict between these two tasks. **Third**, our method achieves better performance than Separate. In Separate, the QAG and QA tasks are performed separately where QA cannot guide QAG. In contrast, our method performs these two tasks end-to-end which enables them to synergistically help each other. **Fourth**, our method performs better than No-Weight. Our method can identify noisy examples and remove them from the training process (Figure 11 in the appendix shows some randomly sampled examples identified by our method as noisy). In contrast, No-Weight uses all examples including the noisy ones in D_{qag} for model training, which leads to worse performance.

Table 8. Results on PathVQA. B-n denotes BLEU with n-grams.

		Accuracy	B-1	B-2	B-3	F1
Lxmert	Vanilla (Tan & Bansal, 2019)	57.6	57.4	3.1	1.3	9.9
	Separate	57.8	57.9	3.2	1.5	10.2
	No-Weight	57.9	58.1	3.7	1.6	10.4
	MTL (Maninis et al., 2019)	58.3	58.7	4.0	1.6	10.5
	Ours	**60.4**	**60.1**	**4.5**	**2.8**	**11.7**
BAN	Vanilla (Kim et al., 2018)	55.1	56.2	3.2	1.2	8.4
	Separate	55.3	56.8	3.3	1.4	8.8
	No-Weight	55.9	57.4	3.8	1.7	9.2
	MTL (Maninis et al., 2019)	57.1	57.9	3.6	1.8	10.5
	Ours	**59.3**	**59.7**	**4.1**	**2.5**	**11.3**

7. Conclusions and Discussions

In this paper, we develop a general framework called Skillearn to formalize humans' classroom study techniques into machine-executable training strategies and leverage them to train better BLO-based models. Our framework can flexibly formulate many study techniques of humans, by configuring learners, learning stages, interaction functions, etc. In case studies, we apply Skillearn to formalize two study techniques of humans - the Feynman technique and peer questioning, with applications to neural architecture search and data reweighting. Experiments on various datasets demonstrate the effectiveness of our methods.

Our methods have two major limitations. First, compared with bi-level optimization based methods such as DARTS, our multi-level optimization based formulation incurs additional computational costs due to the increased number of levels. Second, our methods incur additional memory cost due to storing extra models such as the listener in the Feynman architecture search method. Please refer to Section F in the appendix regarding how to address these limitations.

References

Ainsworth, S. and Th Loizou, A. The effects of self-explaining when learning with text or diagrams. *Cognitive science*, 2003.

Ambion, R. I. A., De Leon, R. S. C., Mendoza, A. P. A. R., and Navarro, R. M. The utilization of the feynman technique in paired team teaching towards enhancing grade 10 anhs students' academic achievement in science. In *2020 IEEE Integrated STEM Education Conference (ISEC)*, pp. 1–3. IEEE, 2020.

Axelrod, A., He, X., and Gao, J. Domain adaptation via pseudo in-domain data selection. In *EMNLP*, 2011.

Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., and Wood, F. Online learning rate adaptation with hypergradient descent. *ICLR*, 2018.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *ICML*, 2009.

Birnbaum, M. S., Kornell, N., Bjork, E. L., and Bjork, R. A. Why interleaving enhances inductive learning: The roles of discrimination and retrieval. *Memory & cognition*, 41 (3), 2013.

Biwer, F., oude Egbrink, M. G., Aalten, P., and de Bruin, A. B. Fostering effective learning strategies in higher education–a mixed-methods study. *Journal of Applied Research in Memory and Cognition*, 9(2):186–203, 2020.

Blumenfeld, P. C. Classroom learning and motivation: Clarifying and expanding goal theory. *Journal of Educational psychology*, 84(3):272, 1992.

Borgwardt, K. M. and Kriegel, H.-P. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*, pp. 8–pp. IEEE, 2005.

Bruggemann, D., Kanakis, M., Georgoulis, S., and Van Gool, L. Automated search for resource-efficient branched multi-task networks. *BMVC*, 2020.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. Large-scale study of curiosity-driven learning. *ICLR*, 2019.

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

Carpenter, S. K., Rahman, S., and Perkins, K. The effects of prequestions on classroom learning. *Journal of Experimental Psychology: Applied*, 24(1):34, 2018.

Chen, D., Li, Y., Qiu, M., Wang, Z., Li, B., Ding, B., Deng, H., Huang, J., Lin, W., and Zhou, J. Adabert: Task-adaptive bert compression with differentiable neural architecture search. *arXiv preprint arXiv:2001.04246*, 2020a.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. *ICML*, 2020b.

Chen, X. and Hsieh, C. Stabilizing differentiable architecture search via perturbation-based regularization. *ICML*, 2020a.

Chen, X. and Hsieh, C.-J. Stabilizing differentiable architecture search via perturbation-based regularization. In *International conference on machine learning*, pp. 1554–1565. PMLR, 2020b.

Chen, X., Xie, L., Wu, J., and Tian, Q. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019.

Chen, X., Wang, R., Cheng, M., Tang, X., and Hsieh, C. Drnas: Dirichlet neural architecture search. *ICLR*, 2021a.

Chen, Y., Shen, X., Hu, S. X., and Suykens, J. A. Boosting co-teaching with compression regularization for label noise. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2688–2692, 2021b.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*, 2014.

Choe, S. K., Neiswanger, W., Xie, P., and Xing, E. Betty: An automatic differentiation library for multilevel optimization. *arXiv preprint arXiv:2207.02849*, 2022.

Chu, X., Zhou, T., Zhang, B., and Li, J. Fair DARTS: eliminating unfair advantages in differentiable architecture search. *ECCV*, 2020.

Chu, X., Wang, X., Zhang, B., Lu, S., Wei, X., and Yan, J. DARTS-: robustly stepping out of performance collapse without indicators. *ICLR*, 2021.

Dempe, S. *Foundations of bilevel programming*. Springer Science & Business Media, 2002.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Diehl, F. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019.

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

Dong, X. and Yang, Y. Searching for a robust neural architecture in four GPU hours. In *CVPR*, 2019b.

Dunlosky, J., Rawson, K. A., Marsh, E. J., Nathan, M. J., and Willingham, D. T. Improving students' learning with effective learning techniques: Promising directions from cognitive and educational psychology. *Psychological Science in the Public Interest*, 14(1):4–58, 2013.

Dziugaite, G. K., Roy, D. M., and Ghahramani, Z. Training generative neural networks via maximum mean discrepancy optimization. *UAI*, 2015.

Feurer, M., Springenberg, J., and Hutter, F. Initializing bayesian hyperparameter optimization via meta-learning. In *AAAI*, 2015.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

Foster, G., Goutte, C., and Kuhn, R. Discriminative instance weighting for domain adaptation in statistical machine translation. In *EMNLP*, 2010.

Fukui, A., Park, D. H., Yang, D., Rohrbach, A., Darrell, T., and Rohrbach, M. Multimodal compact bilinear pooling for visual question answering and visual grounding. *EMNLP*, 2016.

Gao, H. and Ji, S. Graph u-nets. *arXiv preprint arXiv:1905.05178*, 2019.

Gao, Y., Bai, H., Jie, Z., Ma, J., Jia, K., and Liu, W. Mtl-nas: Task-agnostic neural architecture search towards general-purpose multi-task learning. In *CVPR*, 2020.

Garg, B., Zhang, L., Sridhara, P., Hosseini, R., Xing, E., and Xie, P. Learning from mistakes-a framework for neural architecture search. *AAAI Conference on Artificial Intelligence*, 2022.

Gidaris, S., Singh, P., and Komodakis, N. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *NeurIPS*, 2014.

Goutte, C. and Gaussier, E. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Conference on Information Retrieval*, 2005.

Gramacki, A. *Nonparametric kernel density estimation and its computational aspects*. Springer, 2018.

Gu, Y.-C., Wang, L.-J., Liu, Y., Yang, Y., Wu, Y.-H., Lu, S.-P., and Cheng, M.-M. Dots: Decoupling operation and topology in differentiable architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12311–12320, 2021.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

He, C., Ye, H., Shen, L., and Zhang, T. Milenas: Efficient neural architecture search via mixed-level reformulation. *CVPR*, 2020a.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. *CVPR*, 2020b.

He, X., Cai, Z., Wei, W., Zhang, Y., Mou, L., Xing, E., and Xie, P. Towards visual question answering on pathology images. In *ACL*, 2021.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 1997.

Hong, W., Li, G., Zhang, W., Tang, R., Wang, Y., Li, Z., and Yu, Y. Dropnas: Grouped operation dropout for differentiable architecture search. In *IJCAI*, 2020.

Hosseini, R. and Xie, P. Saliency-aware neural architecture search. *Conference on Neural Information Processing Systems*, 2022.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR*, 2017.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *ICLR*, 2017.

Jiang, J. and Zhai, C. Instance weighting for domain adaptation in nlp. *ACL*, 2007.

Jiang, L., Meng, D., Yu, S.-I., Lan, Z., Shan, S., and Hauptmann, A. Self-paced learning with diversity. *NeurIPS*, 2014.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

Kim, J.-H., Jun, J., and Zhang, B.-T. Bilinear attention networks. In *NIPS*, 2018.

King, A. Enhancing peer interaction and learning in the classroom through reciprocal questioning. *American Educational Research Journal*, 27(4):664–687, 1990.

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

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *ICLR*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *The International Conference on Learning Representations (ICLR)*, 2017.

Kokkinos, I. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *CVPR*, 2017.

Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *IJCV*, 2017.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Kumar, M. P., Packer, B., and Koller, D. Self-paced learning for latent variable models. In *NeurIPS*, 2010.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Lee, J., Lee, I., and Kang, J. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082*, 2019.

Li, L., Khodak, M., Balcan, M.-F., and Talwalkar, A. Geometry-aware gradient algorithms for architecture search. *ICLR*, 2021.

Liang, H., Zhang, S., Sun, J., He, X., Huang, W., Zhuang, K., and Li, Z. DARTS+: improved differentiable architecture search with early stopping. *CoRR*, abs/1909.06035, 2019.

Liang, J., Meyerson, E., and Miikkulainen, R. Evolutionary architecture search for deep multitask networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A. L., Huang, J., and Murphy, K. Progressive neural architecture search. In *ECCV*, 2018a.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. Hierarchical representations for efficient architecture search. In *ICLR*, 2018b.

Liu, H., Simonyan, K., and Yang, Y. DARTS: differentiable architecture search. In *ICLR*, 2019.

Liu, R., Gao, J., Zhang, J., Meng, D., and Lin, Z. Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond. *arXiv:2101.11517*, 2021.

Liu, S., Zhu, Z., Qu, Q., and You, C. Robust training under label noise by over-parameterization. In *International Conference on Machine Learning*, pp. 14153–14172. PMLR, 2022.

Ma, N., Zhang, X., Zheng, H., and Sun, J. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, 2018.

Ma, Y., Wang, S., Aggarwal, C. C., and Tang, J. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 723–731, 2019.

Malinowski, M. and Fritz, M. A multi-world approach to question answering about real-world scenes based on uncertain input. In *NIPS*, 2014.

Maninis, K.-K., Radosavovic, I., and Kokkinos, I. Attentive single-tasking of multiple tasks. In *CVPR*, 2019.

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. Teacher-student curriculum learning. *TNNLS*, 2019.

Migdalas, A., Pardalos, P. M., and Värbrand, P. *Multilevel optimization: algorithms and applications*, volume 20. Springer Science & Business Media, 1998.

Moore, R. C. and Lewis, W. Intelligent selection of language model training data. *ACL*, 2010.

Muir, R. et al. Text-book of pathology. *Text-Book of Pathology.*, (Fifth Edition), 1941.

Ngiam, J., Peng, D., Vasudevan, V., Kornblith, S., Le, Q. V., and Pang, R. Domain adaptive transfer learning with specialist models. *arXiv preprint arXiv:1811.07056*, 2018.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. Bleu: a method for automatic evaluation of machine translation. In *ACL*, 2002.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *ICML*.

Pennington, J., Socher, R., and Manning, C. Glove: Global vectors for word representation. In *EMNLP*, 2014.

Perez, L. and Wang, J. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

Ren, M., Zeng, W., Yang, B., and Urtasun, R. Learning to reweight examples for robust deep learning. In *International conference on machine learning*, pp. 4334–4343. PMLR, 2018.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

Ren, Z., Yeh, R., and Schwing, A. Not all unlabeled data are equal: Learning to weight data in semi-supervised learning. In *NeurIPS*, 2020.

Robbins, S. L., Angell, M., and Kumar, V. *Basic pathology*. WB Saunders, 1981.

Rovers, S. F., Stalmeijer, R. E., van Merriënboer, J. J., Savelberg, H. H., and De Bruin, A. B. How and why do students use learning strategies? a mixed methods study on learning strategies and desirable difficulties with effective strategy users. *Frontiers in psychology*, 9:2501, 2018.

Ruder, S. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Sato, R., Tanaka, M., and Takeda, A. A gradient method for multilevel optimization. *Advances in Neural Information Processing Systems*, 34, 2021.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pp. 488–495, 2009.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Sheth, P. and Xie, P. Improving differentiable neural architecture search by encouraging transferability. In *International Conference on Learning Representations*, 2023.

Shu, J., Xie, Q., Yi, L., Zhao, Q., Zhou, S., Xu, Z., and Meng, D. Meta-weight-net: Learning an explicit mapping for sample weighting. *Advances in neural information processing systems*, 32, 2019.

Sivasankaran, S., Vincent, E., and Illina, I. Discriminative importance weighting of augmented training data for acoustic model training. In *ICASSP*, 2017.

Somayajula, S. A., Song, L., and Xie, P. A multi-level optimization framework for end-to-end text augmentation. *Transactions of the Association for Computational Linguistics*, 10:343–358, 2022.

Song, H., Kim, M., and Lee, J.-G. Selfie: Refurbishing unclean samples for robust deep learning. In *International Conference on Machine Learning*, pp. 5907–5915. PMLR, 2019.

Such, F. P., Rawal, A., Lehman, J., Stanley, K. O., and Clune, J. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. *ICML*, 2020.

Sun, S., Cheng, Y., Gan, Z., and Liu, J. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019.

Sun, Z., Hu, Y., Lu, S., Yang, L., Mei, J., Han, Y., and Li, X. Agnas: Attention-guided micro and macro-architecture search. In *International Conference on Machine Learning*, pp. 20777–20789. PMLR, 2022.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. Learning to compare: Relation network for few-shot learning. In *CVPR*, 2018.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *CVPR*, 2015.

Tan, H. and Bansal, M. Lxmert: Learning cross-modality encoder representations from transformers. *EMNLP*, 2019.

Tang, R., Lu, Y., Liu, L., Mou, L., Vechtomova, O., and Lin, J. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*, 2019.

Tarvainen, A. and Valpola, H. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *NeurIPS*, 2017.

Van Lier, L. Inside the classroom: Learning processes and teaching procedures. *Applied Language Learning*, 2(1), 1991.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NeurIPS*, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Vinyals, O., Bengio, S., and Kudlur, M. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wang, H., Xiao, R., Dong, Y., Feng, L., and Zhao, J. Promix: Combating label noise via maximizing clean sample utility. *arXiv preprint arXiv:2207.10276*, 2022.

Wang, R., Cheng, M., Chen, X., Tang, X., and Hsieh, C.-J. Rethinking architecture selection in differentiable nas. In *International Conference on Learning Representation*, 2021.

Wang, X., Pham, H., Michel, P., Anastasopoulos, A., Carbonell, J., and Neubig, G. Optimizing data usage via differentiable rewards. In *ICML*, 2020a.

Wang, Y., Guo, J., Song, S., and Huang, G. Meta-semi: A meta-learning approach for semi-supervised learning. *arXiv:2007.02394*, 2020b.

Wei, J. W. and Zou, K. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

Xie, P. and Du, X. Performance-aware mutual knowledge distillation for improving neural architecture search. *CVPR*, 2022.

Xie, Q., Luong, M.-T., Hovy, E., and Le, Q. V. Self-training with noisy student improves imagenet classification. In *CVPR*, 2020.

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS: stochastic neural architecture search. In *ICLR*, 2019.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G., Tian, Q., and Xiong, H. PC-DARTS: partial channel connections for memory-efficient architecture search. In *ICLR*, 2020.

Yang, Y., Li, H., You, S., Wang, F., Qian, C., and Lin, Z. Istanas: Efficient and consistent neural architecture search by sparse coding. *NeurIPS*, 2020.

Ye, P., Li, B., Li, Y., Chen, T., Fan, J., and Ouyang, W. b-darts: Beta-decay regularization for differentiable architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10874–10883, 2022.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, pp. 4800–4810, 2018.

Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*, 2019.

Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020.

Zeng, J. and Xie, P. Contrastive self-supervised learning for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10824–10832, 2021.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Zhang, Y. and Yang, Q. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Zhang, Y., Zheng, S., Wu, P., Goswami, M., and Chen, C. Learning with feature-dependent label noise: A progressive approach. In *International Conference on Learning Representations*.

Zhang, Z., Bu, J., Ester, M., Zhang, J., Yao, C., Yu, Z., and Wang, C. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*, 2019.

Zheng, G., Awadallah, A. H., and Dumais, S. T. Meta label correction for learning with weak supervision. *AAAI*, 2021.

Zhou, P., Xiong, C., Socher, R., and Hoi, S. C. H. Theory-inspired path-regularized differential network architecture search. *NeurIPS*, 2020.

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

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.

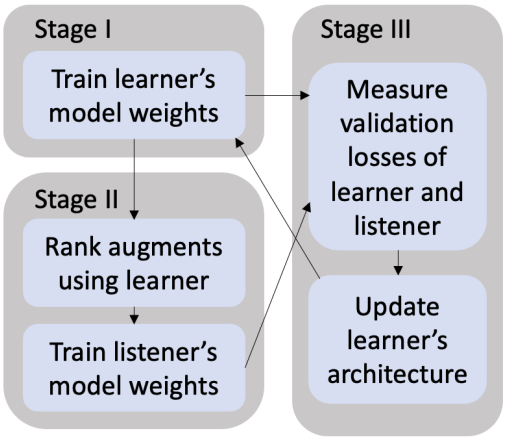

Figure 5. Flowchart of the three stages in FAS.

A. Additional Related Works

A.1. Neural Architecture Search (NAS)

NAS aims to automatically search for high-performance architectures. Existing NAS methods can be categorized into three groups, based on reinforcement learning (Zoph & Le, 2017; Pham et al., 2018; Zoph et al., 2018), gradient methods (Cai et al., 2019; Liu et al., 2019; Xie et al., 2019), and evolutionary algorithms (Liu et al., 2018b; Real et al., 2019). Recently, differentiable NAS methods (Liu et al., 2019; Chen et al., 2019; Xu et al., 2020) have obtained broad attention due to their computational efficiency. These approaches perform search in an overparameterized space composed of a large number of basic building blocks such as convolutions, poolings, etc. A selection variable is learned for each block to represent whether this block should be retained in the final architecture. In the end, architecture search amounts to learning these selection variables, which can be performed using efficient algorithms such as gradient descent. Differentiable NAS methods suffer from performance collapse (Zela et al., 2019; Chu et al., 2020; Chen & Hsieh, 2020a): the searched architectures are degenerate with excessive skip connections and yield poor test performance. For example, Zela et al. (Zela et al., 2019) identified 12 NAS benchmarks based on four search spaces where DARTS (Liu et al., 2019) (a differentiable NAS method) results in degenerate architectures with poor performance on test data. Our framework is orthogonal to existing NAS approaches and can be applied to improve them.

A.2. Data Reweighting

A variety of methods (Jiang & Zhai, 2007; Foster et al., 2010; Moore & Lewis, 2010; Axelrod et al., 2011; Sivasankaran et al., 2017; Ngiam et al., 2018) have been developed for data reweighting. Some of them are based on bi-level optimization (Ren et al., 2018; Shu et al., 2019; Ren et al., 2020; Wang et al., 2020b;a), where a model is trained on reweighted data and data weights are learned by optimizing validation performance of the trained model. Our framework is orthogonal to these methods and can be applied to improve them.

A.3. Multi-task Learning (MTL)

MTL (Ruder, 2017; Zhang & Yang, 2021; Gao et al., 2020; Liang et al., 2018; Bruggemann et al., 2020) learns multiple tasks jointly by minimizing the weighted sum of their losses and transfers knowledge across tasks. In many applications, tasks have an inherent order. For example, distilling knowledge (Hinton et al., 2015) from model A to model B involves three tasks: 1) train model A, 2) use model A to generate pseudo labels, and 3) train model B on pseudo labels. These tasks have a natural order: before generating pseudo labels using model A, we need to train model A first; before training model B on pseudo labels, we need to generate pseudo labels first. MTL performs these tasks simultaneously by minimizing a single objective function, which breaks their inherent order and therefore may lead to worse performance. In contrast, our framework preserves this order using multi-level optimization.

Table 9. Instantiation of Skillearn to PQDR.

Skillearn	Peer-questioning based data reweighting
Learners	K peer learners
Model weights and meta parameters	1) Image encoder E_k of learner k; 2) Question-answer generation head G_k of learner k; 3) Question-answering head Q_k of learner k; 4) Weights A of data examples in D_{qag}.
Interaction function	Each learner k trains its question-answering head Q_k on questions and answers generated by other learners. $\sum_{k=1}^{K}\sum_{j\neq k}^{K} L_{qa}(E_k^*(A), Q_k, \mathcal{Q}(I_u, E_j^*(A), G_j^*(A)), \mathcal{A}(I_u, E_j^*(A), G_j^*(A)))$.
Learning stages	• Learning stage I: Each learner k trains its encoder weights E_k and question-answer generation head G_k on D_{qag}. • Learning stage II: Each learner k generates questions and answers (QAs) from unlabeled images and trains its question-answering head Q_k on QAs generated by other learners. • Learning stage III: Learners evaluate their QA models on $D_{qa}^{(val)}$ and update data weights of D_{qaq} by minimizing validation losses.

B. Additional Details of Methods

B.1. Additional Figures

Figure 5 shows the flowchart of the three stages in FAS.

B.2. Additional Tables

Table 9 shows how to instantiate the Skillearn framework to PQDR.

B.3. A More Detailed Formulation of Skillearn

Below is a more detailed formulation of Skillearn:

$$
\begin{aligned}
&\max_{\{\mathcal{A}_k\}_{k=1}^{K-1}} \quad L_{val}(\{\mathcal{W}_k^*(\{\mathcal{A}_j\}_{j=1}^{k})\}_{k=1}^{K-1}, \{\mathcal{A}_k\}_{k=1}^{K-1}, \mathcal{D}^{(\text{val})}) \\
&s.t.\ \mathcal{W}_{K-1}^*(\{\mathcal{A}_j\}_{j=1}^{K-1}) = \underset{\mathcal{W}_{K-1}}{\text{argmin}}\ L_{K-1}(\mathcal{W}_{K-1}, \mathcal{A}_{K-1}, \mathcal{W}_{K-2}^*(\{\mathcal{A}_j\}_{j=1}^{K-2}), \mathcal{D}_{K-1}^{(\text{tr})}) + \\
&\qquad \gamma_{K-1} I_{K-1}(\mathcal{W}_{K-1}, \mathcal{A}_{K-1}, \mathcal{W}_{K-2}^*(\{\mathcal{A}_j\}_{j=1}^{K-2}), \mathcal{F}_{K-1}) \\
&\quad \cdots \\
&\quad s.t\ \mathcal{W}_k^*(\{\mathcal{A}_j\}_{j=1}^{k}) = \underset{\mathcal{W}_k}{\text{argmin}}\ L_k(\mathcal{W}_k, \mathcal{A}_k, \mathcal{W}_{k-1}^*(\{\mathcal{A}_j\}_{j=1}^{k-1}), \mathcal{D}_k^{(\text{tr})}) + \\
&\qquad \gamma_k I_k(\mathcal{W}_k, \mathcal{A}_k, \mathcal{W}_{k-1}^*(\{\mathcal{A}_j\}_{j=1}^{k-1}), \mathcal{F}_k) \\
&\qquad s.t\ \mathcal{W}_{k-1}^*(\{\mathcal{A}_j\}_{j=1}^{k-1}) = \underset{\mathcal{W}_{k-1}}{\text{argmin}}\ L_{k-1}(\mathcal{W}_{k-1}, \mathcal{A}_{k-1}, \mathcal{W}_{k-2}^*(\{\mathcal{A}_j\}_{j=1}^{k-2}), \mathcal{D}_{k-1}^{(\text{tr})}) + \\
&\qquad\quad \gamma_{k-1} I_{k-1}(\mathcal{W}_{k-1}, \mathcal{A}_{k-1}, \mathcal{W}_{k-2}^*(\{\mathcal{A}_j\}_{j=1}^{k-2}), \mathcal{F}_{k-1}) \\
&\qquad\quad \cdots \\
&\qquad\quad s.t.\ \mathcal{W}_1^*(\mathcal{A}_1) = \text{argmin}_{\mathcal{W}_1}\ L_1(\mathcal{W}_1, \mathcal{A}_1, \mathcal{D}_1^{(\text{tr})}) + \gamma_1 I_1(\mathcal{W}_1, \mathcal{A}_1, \mathcal{F}_1)
\end{aligned}
\tag{10}
$$

B.4. Additional Discussion of Skillearn

The multi-level optimization problem of Skillearn with K levels can be considered as a composition of $K-1$ bi-level optimization problems which are nested. Specifically, each pair of optimization problems at level k and $k+1$ form a bi-level

optimization problem, for $1 \leq k \leq K-1$, as follows:

$$\begin{aligned} &\mathcal{W}_k^*(\{\mathcal{A}_j\}_{j=1}^k) = \underset{\mathcal{W}_k}{\text{argmin}}\ L_k(\mathcal{W}_k, \mathcal{A}_k, \mathcal{W}_{k-1}^*(\{\mathcal{A}_j\}_{j=1}^{k-1}), \mathcal{D}_k^{(\text{tr})}) + \gamma_k I_k(\mathcal{W}_k, \mathcal{A}_k, \mathcal{W}_{k-1}^*(\{\mathcal{A}_j\}_{j=1}^{k-1}), \mathcal{F}_k) \\ &s.t. \quad \mathcal{W}_{k-1}^*(\{\mathcal{A}_j\}_{j=1}^{k-1}) = \underset{\mathcal{W}_{k-1}}{\text{argmin}}\ L_{k-1}(\mathcal{W}_{k-1}, \mathcal{A}_{k-1}, \mathcal{W}_{k-2}^*(\{\mathcal{A}_j\}_{j=1}^{k-2}), \\ &\quad \mathcal{D}_{k-1}^{(\text{tr})}) + \gamma_{k-1} I_{k-1}(\mathcal{W}_{k-1}, \mathcal{A}_{k-1}, \mathcal{W}_{k-2}^*(\{\mathcal{A}_j\}_{j=1}^{k-2}), \mathcal{F}_{k-1}) \end{aligned} \tag{11}$$

The upper-level problem of each bi-level optimization problem is at the same time the lower-level problem of another bi-level optimization problem.

B.5. Continuous Relaxation in FAS

The constraints in Eq.(3) in FAS are not amenable for optimization. To address this problem, we perform a relaxation of these constraints and transform them into loss terms. If an augmented image $a_{i,k}$ ranks higher than $a_{i,j}$, it implies that $f(a_{i,k}, x; A, E^*(A)) > f(a_{i,j}, x; A, E^*(A))$. According to the constraint, we need to make sure $f(a_{i,k}, x; F) > f(a_{i,j}, x; F)$. In other words, we would like $f(a_{i,k}, x; A, E^*(A)) - f(a_{i,j}, x; A, E^*(A))$ and $f(a_{i,k}, x; F) - f(a_{i,j}, x; F)$ to be both positive or negative, which is equivalent to encouraging their product to be positive. This can be achieved by minimizing the following hinge loss:

$$\max(0, -(f(a_{i,k}, x; A, E^*(A)) - f(a_{i,j}, x; A, E^*(A)))(f(a_{i,k}, x; F) - f(a_{i,j}, x; F)). \tag{12}$$

If the product is positive, there is no penalty. The optimization for the discontinuous operator max in the hinge loss is based on the sub-gradient method. Accordingly, the second stage in FAS becomes:

$$\begin{aligned} &F^*(A, E^*(A)), G^* = \text{argmin}_{F,G}\ L(F, G, D^{(\text{tr})}) + \\ &\lambda \textstyle\sum_{i=1}^N \sum_{1 \leq k < j \leq K} \max(0, -(f(a_{i,k}, x_i; A, E^*(A)) - f(a_{i,j}, x_i; A, E^*(A)))(f(a_{i,k}, x_i; F) - f(a_{i,j}, x_i; F))), \end{aligned} \tag{13}$$

where N is the number of training examples and λ is a tradeoff parameter.

The formulation of FAS with continuous relaxation is:

$$\begin{aligned} &\min_A\ L(A, E^*(A), H^*(A), D^{(\text{val})}) + \gamma L(F^*(E^*(A), A), G^*, D^{(\text{val})}) \\ &s.t\ F^*(E^*(A), A), G^* = \text{argmin}_{F,G}\ L(F, G, D^{(\text{tr})}) + \\ &\quad \lambda \sum_{i=1}^N \sum_{1 \leq k < j \leq K} \max(0, -(f(a_{i,k}, x_i; A, E^*(A)) - f(a_{i,j}, x_i; A, E^*(A)))(f(a_{i,k}, x_i; F) - f(a_{i,j}, x_i; F))) \\ &\quad s.t.\ E^*(A), H^*(A) = \text{argmin}_{E,H}\ \ L(A, E, H, D^{(\text{tr})}) \end{aligned} \tag{14}$$

B.6. Formulation for the Multi-task Learning Baseline

For multi-task learning (MTL), the formulation is:

$$\begin{aligned} &\min_A \quad L(A, E^*(A), H^*(A), D^{(\text{val})}) + \gamma L(F^*, G^*, D^{(\text{val})}) \\ &s.t \quad\ E^*(A), H^*(A), F^*, G^* = \text{argmin}_{E,H,F,G}\ L(A, E, H, D^{(\text{tr})}) + L(F, G, D^{(\text{tr})}) + \lambda \sum_{i=1}^N \sum_{1 \leq k < j \leq K} \\ &\quad\ \max(0, -(f(a_{i,k}, x_i; A, E) - f(a_{i,j}, x_i; A, E))(f(a_{i,k}, x_i; F) - f(a_{i,j}, x_i; F))) \end{aligned} \tag{15}$$

B.7. Formulation for the Shared Encoder Baseline

The formulation of Shared Encoder (SE) is similar to MTL, except that in SE, the listener's encoder architecture is the same as that of the learner. But they have different weight parameters. Let A denote the shared encoder architecture. Let F denote the listener's encoder weights.

$$\begin{aligned} &\min_A \quad L(A, E^*(A), H^*(A), D^{(\text{val})}) + \gamma L(A, F^*(A), G^*(A), D^{(\text{val})}) \\ &s.t \quad\ E^*(A), H^*(A), F^*(A), G^*(A) = \text{argmin}_{E,H,F,G}\ L(A, E, H, D^{(\text{tr})}) + L(A, F, G, D^{(\text{tr})}) + \lambda \sum_{i=1}^N \sum_{1 \leq k < j \leq K} \\ &\quad\ \max(0, -(f(a_{i,k}, x_i; A, E) - f(a_{i,j}, x_i; A, E))(f(a_{i,k}, x_i; A, F) - f(a_{i,j}, x_i; A, F))) \end{aligned} \tag{16}$$

C. Optimization Algorithms

C.1. A General Optimization Algorithm for the Skillearn Framework

We develop an algorithm to solve the Skillearn problem, inspired by the algorithm in (Liu et al., 2019). For each learning stage k, we approximate the optimal solution $\mathcal{W}_k^*(\{\mathcal{A}_j\}_{j=1}^k)$ by one-step gradient descent update of the variable $\mathcal{W}_k$:

$$\mathcal{W}_k^*(\{\mathcal{A}_j\}_{j=1}^k) \approx \mathcal{W}_k'(\{\mathcal{A}_j\}_{j=1}^k) = \mathcal{W}_k - \eta \nabla_{\mathcal{W}_k}(L_k(\mathcal{W}_k, \mathcal{A}_k, \mathcal{W}_{k-1}^*(\{\mathcal{A}_j\}_{j=1}^{k-1}), \mathcal{D}_k^{(\text{tr})}) + \gamma_k I_k(\mathcal{W}_k, \mathcal{A}_k, \mathcal{W}_{k-1}^*(\{\mathcal{A}_j\}_{j=1}^{k-1}), \mathcal{F}_k)). \tag{17}$$

At learning stage $k+1$, $\mathcal{W}_k^*(\{\mathcal{A}_j\}_{j=1}^k)$ is used to define the objective function. In the objective at stage $k+1$, we replace $\mathcal{W}_k^*(\{\mathcal{A}_j\}_{j=1}^k)$ with $\mathcal{W}_k'(\{\mathcal{A}_j\}_{j=1}^k)$ and get an approximated objective. When approximating $\mathcal{W}_{k+1}^*(\{\mathcal{A}_j\}_{j=1}^{k+1})$, we use the gradient of the approximated objective:

$$\mathcal{W}_{k+1}^*(\{\mathcal{A}_j\}_{j=1}^{k+1}) \approx \mathcal{W}_{k+1}'(\{\mathcal{A}_j\}_{j=1}^{k+1}) = \mathcal{W}_{k+1} - \eta \nabla_{\mathcal{W}_{k+1}}(L_{k+1}(\mathcal{W}_{k+1}, \mathcal{A}_{k+1}, \mathcal{W}_k'(\{\mathcal{A}_j\}_{j=1}^k), \mathcal{D}_{k+1}^{(\text{tr})}) + \gamma_{k+1} I_{k+1}(\mathcal{W}_{k+1}, \mathcal{A}_{k+1}, \mathcal{W}_k'(\{\mathcal{A}_j\}_{j=1}^k), \mathcal{F}_{k+1})). \tag{18}$$

The objective at the K-th stage can be approximated as:

$$L_{val}(\{\mathcal{W}_k'(\{\mathcal{A}_j\}_{j=1}^k)\}_{k=1}^{K-1}, \{\mathcal{A}_k\}_{k=1}^{K-1}, \mathcal{D}^{(\text{val})}). \tag{19}$$

We update $\{\mathcal{A}_k\}_{k=1}^{K-1}$ by minimizing this approximated objective. These steps iterate until convergence.

C.2. Algorithm for FAS

In this section, we develop a gradient-based algorithm to solve the FAS problem. Let $\frac{d.}{d.}$ and $\frac{\partial.}{\partial.}$ denote ordinary and partial derivative respectively. Let $\nabla_X f(X)$ denote a gradient and $\nabla_{Y,X}^2 f(X,Y)$ denote $\frac{\partial f(X,Y)}{\partial X \partial Y}$. Drawing insights from (Liu et al., 2019), we approximate $E^*(A)$ and $H^*(A)$ using a one-step gradient descent update of E and H:

$$E^*(A) \approx E' = E - \xi \nabla_E L(A, E, H, D^{(\text{tr})}),\ H^*(A) \approx H' = H - \xi \nabla_H L(A, E, H, D^{(\text{tr})}), \tag{20}$$

where ξ is a learning rate. We plug $E^*(A) \approx E'$ into the loss function at Stage 2 and get an approximated loss O:

$$O = L(F, G, D^{(\text{tr})}) + \lambda \sum_{i=1}^{N} \sum_{1 \le k < j \le K} \max(0, -(f(a_{i,k}, x_i; A, E') - f(a_{i,j}, x_i; A, E'))(f(a_{i,k}, x_i; F) - f(a_{i,j}, x_i; F))). \tag{21}$$

Then we approximate $F^*(A, E^*(A))$ and G^* by one-step gradient-descent update of F and G w.r.t the approximated loss O:

$$F^*(A, E^*(A)) \approx F' = F - \xi \nabla_F O,\ G^* \approx G' = G - \xi \nabla_G O. \tag{22}$$

Finally, we plug these approximations into the validation losses at the third stage and update the architecture A using gradient descent (with a learning rate η), by minimizing the approximated validation losses:

$$A \leftarrow A - \eta \nabla_A (L(A, E', H', D^{(\text{val})}) + \gamma L(F', G', D^{(\text{val})})) \tag{23}$$

where

$$\nabla_A L(A, E', H', D^{(\text{val})}) = \frac{\partial L(A, E', H', D^{(\text{val})})}{\partial A} + \frac{dE'}{dA} \frac{\partial L(A, E', H', D^{(\text{val})})}{\partial E'} + \frac{dH'}{dA} \frac{\partial L(A, E', H', D^{(\text{val})})}{\partial H'} \tag{24}$$

$$\nabla_A L(F', G', D^{(\text{val})}) = \frac{dF'}{dA} \frac{\partial L(F', G', D^{(\text{val})})}{\partial F'}. \tag{25}$$

In Eq.(24), $\frac{dE'}{dA} = -\xi \nabla_{A,E}^2 L(A, E, H, D^{(\text{tr})})$ and $\frac{dH'}{dA} = -\xi \nabla_{A,H}^2 L(A, E, H, D^{(\text{tr})})$. In Eq.(25), we have $\frac{dF'}{dA} = -\xi \frac{d\nabla_F O}{dA}$, in which

$$\frac{d\nabla_F O}{dA} = \frac{dE'}{dA} \nabla_{E',F}^2 O + \lambda \sum_{i=1}^{N} \sum_{1 \le k < j \le K} \frac{\partial \nabla_F \max(0, -(f(a_{i,k}, x_i; A, E') - f(a_{i,j}, x_i; A, E'))(f(a_{i,k}, x_i; F) - f(a_{i,j}, x_i; F)))}{\partial A}. \tag{26}$$

These steps iterate until convergence. Similar to (Liu et al., 2019), we approximate matrix-vector multiplications in Eq.(24) and Eq.(25) using finite-difference, which can reduce the quadratic computational complexity (in terms of matrix dimensions) down to linear. The algorithm for solving FAS is in Algorithm 1. Figure 6 shows the dependency between variables and gradients in FAS.

Algorithm 1 Optimization algorithm for FAS

While not converged
1. Update learner's encoder weights E using Eq.(20)
2. Update learner's head H using Eq.(20)
3. Update listener's encoder weights F using Eq.(22)
4. Update listener's head G using Eq.(22)
5. Update learner's encoder architecture A using Eq.(23)

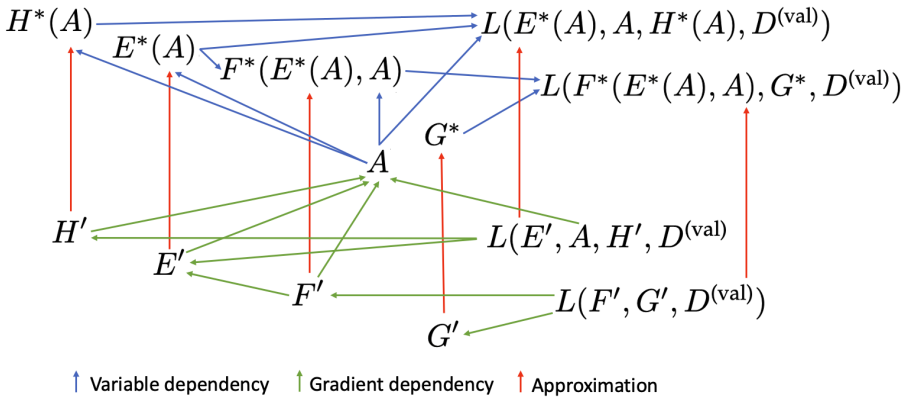

Figure 6. Dependency between variables and gradients in FAS.

C.3. Algorithm for PQDR

We approximate $E_k^*(A)$ and $G_k^*(A)$ using one-step gradient descent update of E_k and G_k w.r.t $\sum_{k=1}^{K}\sum_{i=1}^{N} a_i L_{qag}(E_k, G_k, d_i)$:

$$E_k^*(A) \approx E_k' = E_k - \xi \nabla_{E_k} \sum_{i=1}^{N} a_i L_{qag}(E_k, G_k, d_i), \tag{27}$$

$$G_k^*(A) \approx G_k' = G_k - \xi \nabla_{G_k} \sum_{i=1}^{N} a_i L_{qag}(E_k, G_k, d_i). \tag{28}$$

We plug $E_k^*(A) \approx E_k'$ and $G_k^*(A) \approx G_k'$ into the loss function at the second stage and get an approximated objective:

$$O = \sum_{k=1}^{K}\sum_{j\neq k}^{K} L_{qa}(E_k', Q_k, \mathcal{Q}(I_u, E_j', G_j'), \mathcal{A}(I_u, E_j', G_j')). \tag{29}$$

We approximate $Q_k^*(\{E_j^*(A), G_j^*(A)\}_{j\neq k}^K, E_k^*(A))$ using one-step gradient descent update of Q_k w.r.t the approximated objective:

$$Q_k^*(\{E_j^*(A), G_j^*(A)\}_{j\neq k}^K, E_k^*(A)) \approx Q_k' = Q_k - \xi \nabla_{Q_k} O. \tag{30}$$

Finally, we plug $Q_k^*(\{E_j^*(A), G_j^*(A)\}_{j\neq k}^K, E_k^*(A)) \approx Q_k'$ and $E_k^*(A) \approx E_k'$ into the loss function at the third stage and get an approximated loss function. Then we update A by minimizing the approximated loss using gradient descent:

$$A \leftarrow A - \eta \sum_{k=1}^{K} \nabla_A L_{qa}(Q_k', E_k', D_{qa}), \text{ where} \tag{31}$$

$$\nabla_A L_{qa}(Q_k', E_k', D_{qa}) = \frac{dQ_k'}{dA}\frac{\partial L_{qa}(Q_k', E_k', D_{qa})}{\partial Q_k'} + \frac{dE_k'}{dA}\frac{\partial L_{qa}(Q_k', E_k', D_{qa})}{\partial E_k'}, \tag{32}$$

Algorithm 2 Optimization algorithm for PQDR

While not converged
1. Update E_k using Eq.(15) in the main paper for $k = 1, \cdots, K$
2. Update G_k using Eq.(16) in the main paper for $k = 1, \cdots, K$
3. Update Q_k using Eq.(18) in the main paper for $k = 1, \cdots, K$
4. Update A using Eq.(19) in the main paper

where $\frac{dE'_k}{dA} = -\xi \nabla^2_{A,E_k} \sum_{i=1}^{N} a_i L_{qag}(E_k, G_k, d_i)$ and

$$\frac{dQ'_k}{dA} = -\xi\big(\frac{dE'_k}{dA}\nabla^2_{E'_k,Q_k} O + \sum_{j\neq k}\big(\frac{dE'_j}{dA}\nabla^2_{E'_j,Q_k} O + \frac{dG'_j}{dA}\nabla^2_{G'_j,Q_k} O\big)\big). \tag{33}$$

These steps iterate until convergence. The optimization algorithm for PQDR is summarized in Algorithm 2.

D. Additional Experimental Settings

D.1. Additional Experimental Settings of FAS

For the rest of hyperparameters, our method simply uses their default values provided in baselines (including DARTS, P-DARTS, PC-DARTS, and PR-DARTS) where our method is applied to, without tuning them. For example, when FAS is applied to DARTS, the optimizer, learning rate, learning rate scheduler, batch size, epochs, momentum, weight decay of the weight parameters and architecture variables in FAS' learner are the same as those in DARTS. The values of these hyperparameters for the listener's weight parameters are the same as those in the learner. Similarly, for other experimental setup (e.g., protocols for architecture search and evaluation; network architectures including search spaces and the numbers of cells, nodes, initial channels; parameter initialization methods, etc.), our method follows those in DARTS, P-DARTS, PC-DARTS, and PR-DARTS.

During architecture search on CIFAR-10 and CIFAR-100, the learner's network is a stack of 8 cells, each consisting of 7 nodes, with the initial channel number set to 16. The search algorithm runs for 50 epochs with a batch size of 64. Model weights are optimized using SGD, with an initial learning rate of 0.025 (adjusted using a cosine decay scheduler), a momentum of 0.9, and a weight decay of 3e-4. The architecture variables A are optimized using Adam (Kingma & Ba, 2015) with a learning rate of 0.001, a momentum of $(0.5, 0.999)$, and a weight decay of 0.001. The learning rate is scheduled with cosine scheduling. The architecture variables are initialized with zero initialization.

During architecture evaluation, for CIFAR-10 and CIFAR-100, a larger network of the learner is formed by stacking 20 copies of the searched cell, and is trained on the combination of $D_t^{(\text{tr})}$ and $D_t^{(\text{val})}$. The number of initial channels is set to 36. The network is trained with a batch size of 96, an epoch number of 600. An 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. On ImageNet, we evaluate architectures searched on a subset of ImageNet and those searched on CIFAR-10 or CIFAR-100. 14 copies of searched cells are stacked into a large network, which is trained on the 1.2M training images, with a batch size of 1024, an epoch number of 250, an initial learning rate of 0.5, and a weight decay of 3e-5. The number of initial channels is set to 48. Cutout, path dropout with probability 0.2 and auxiliary towers with weight 0.4 are applied.

We use PyTorch to implement all models. The version of Torch is 1.4.0 (or above). We build our method upon official python packages for different differentiable search approaches, such as "DARTS[1]", "P-DARTS[2]" and "PC-DARTS[3]".

D.1.1. Hyperparameter Tuning Strategy for λ and γ

To tune the hyperparameters γ and λ, we randomly sample 2.5K data from the 25K training set and sample 2.5K data from the 25K validation set. Then we use the 5K sampled data as a hyperparameter tuning set. γ and λ are tuned in $\{0.01, 0.1, 0.5, 1, 2\}$. For each configuration of γ and λ, we use the remaining 22.5K training data and 22.5K validation data to perform architecture search and use their combination to perform architecture evaluation (retraining a larger stacked

[1]https://github.com/quark0/darts
[2]https://github.com/chenxin061/pdarts
[3]https://github.com/yuhuixu1993/PC-DARTS/

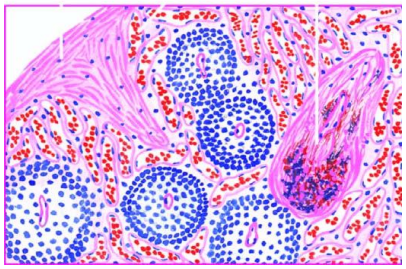

Figure 7. An exemplar image with generated questions from three types: "what", "where", and "yes/no".

Table 10. Frequency of questions in different categories

Question type	Total number and percentage
Yes/No	16,329 (49.8%)
What	13,401 (40.9%)
Where	2,157 (6.6%)
How	595 (1.8%)
How much/many	139 (0.4%)
Why	114 (0.3%)
When	51 (0.2%)
Whose	9 (0.1%)

network from scratch). Then we measure the performance of the stacked network on the 5K sampled data. γ and λ yielding the best performance on the 5K sampled data are selected. For other hyperparameters, they mostly follow those in DARTS (Liu et al., 2019), P-DARTS (Chen et al., 2019), PC-DARTS (Xu et al., 2020), and PR-DARTS (Zhou et al., 2020).

D.2. Additional Details of the PathVQA Dataset

The PathVQA dataset consists of 32,795 question-answer pairs generated from 1,670 pathology images collected from two pathology textbooks: "Textbook of Pathology" (Muir et al., 1941) and "Basic Pathology" (Robbins et al., 1981), and 3,328 pathology images collected from the PEIR[4] digital library. Figure 7 shows an example.

On average, each image has 6.6 questions. The maximum and minimum number of questions for a single image is 14 and 1 respectively. The average number of words per question and per answer is 9.5 and 2.5 respectively. There are eight different categories of questions: what, where, when, whose, how, why, how much/how many, and yes/no. Table 10 shows the number of questions and percentage in each category. The questions in the first 7 categories are open-ended: 16,466 in total and accounting for 50.2% of all questions. The rest are close-ended "yes/no" questions. The questions cover various aspects of visual contents, including color, location, appearance, shape, etc.

D.3. Additional Experimental Settings of PQDR

Data augmentation is applied to images, including shifting, scaling, and shearing. From questions and answers in the PathVQA dataset, we create a vocabulary of 4,631 words that have the highest frequencies.

For the QA model (containing an image encoder and a QA head), we experimented with two choices:

- **LXMERT** (Tan & Bansal, 2019): a Transformer (Vaswani et al., 2017) based model consisting of three encoders: an object relationship encoder, a language encoder, and a cross-modal encoder. The former two are single-modality encoders. The third one learns the relationships between vision and language.
- **BAN** (Kim et al., 2018): using a Gated Recurrent Unit (GRU) (Cho et al., 2014) network and a Faster R-CNN (Ren et al.,

[4] http://peir.path.uab.edu/library/index.php?/category/2

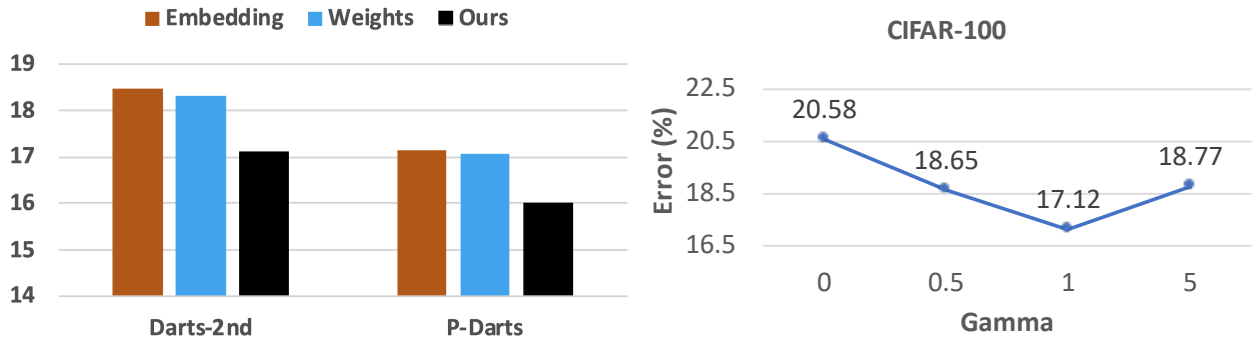

Figure 8. (Left) Comparison of three ways for illustration. (Right) How test error varies as γ increases.

2015) network to embed questions and images. It calculates bilinear attention scores on every pair of multimodal channels.

Adam (Kingma & Ba, 2014) is used to train QAG model weights for 200 epochs, with an initial learning rate of 1e-4 and a batch size of 256. To train the LXMERT QA model, we use the default hyperparameter settings in (Tan & Bansal, 2019). For the text encoder, the hidden size is set to 768. Image features are extracted using Faster-RCNN which is pretrained on BCCD[5] - a medical dataset containing blood cell photos, as well as on Visual Genome (Krishna et al., 2017). LXMERT QA model weights are trained using Adam (Kingma & Ba, 2014) for 200 epochs, with an initial learning rate of 1e-4 and a batch size of 256. In the BAN QA model, words in questions and answers are represented using GloVe (Pennington et al., 2014) vectors pretrained on general-domain corpora such as Wikipedia, Twitter, etc. Image features are extracted using Faster-RCNN pretrained on BCCD and Visual Genome. Dropout (Krizhevsky et al., 2012) rate for linear mapping is set to 0.2. BAN QA model weights are trained using the Adamax optimizer (Kingma & Ba, 2015) for 200 epochs, with an initial learning rate of 0.005 and a batch size of 512. For the data weights A, we optimize them using the Adam optimizer, with an initial learning rate of 0.01. The Gumbel softmax trick (Jang et al., 2017) is used to cope with the non-differentiability of texts.

We use three evaluation metrics: 1) accuracy (Malinowski & Fritz, 2014), measuring the percentage of inferred answers that match exactly with the groundtruth using string matching; 2) BLEU (Papineni et al., 2002), measuring the similarity of predicted answers and groundtruth by matching n-grams; and 3) macro-averaged F1 (Goutte & Gaussier, 2005), measuring the average overlap between predicted answers and groundtruth, where answers are treated as bag of tokens.

We compare with the following baselines: 1) Multi-task learning (MTL) (Maninis et al., 2019): question-answer generation and question answering are performed jointly by minimizing the sum of their losses; 2) Perform QAG and QA separately (Separate): we first train a QAG model, fix it, and use it to generate question-answer pairs; then we use the generated QA pairs to train a QA model; 3) No reweighting of D_{qag} (No-Weight): all examples in D_{qag} are used for training the QAG model.

We implement the methods using PyTorch and perform training on four GTX 1080Ti GPUs.

E. Additional Experimental Results

E.1. Additional Experimental Results for FAS

Ablation study of the ranking-based illustration approach. We compare our proposed ranking-based illustration approach with two baselines: 1) L2 regularization on weights: encouraging the encoder weights of the learner and listener to have small L2 distance; 2) L2 regularization on embeddings: encouraging embeddings generated by the learner and listener to have small L2 distance. Figure 8(left) shows that our method works better than the two baselines. This is because our method does not require learner and listener to have the same encoder architecture or the same embedding dimensions, which is more flexible. In contrast, the two baselines have such a requirement, which is more restrictive. Furthermore, our method is based on ranking K augmented examples, requiring the identification of the global relationship between the K examples. As a result, our method can capture high-order (specifically, Kth-order) relationships among data examples,

[5]https://public.roboflow.ai/object-detection/bccd

Table 11. Analysis of sensitivity to the number of listeners.

Number of listeners	Test error
1	16.41±0.12
2	16.39±0.17
3	16.42±0.14

Table 12. Analysis of sensitivity to the number of learners.

Number of learners	Test error (%)	Search costs (GPU days)
1	16.41±0.12	0.3
2	16.33±0.15	0.5
3	16.28±0.07	0.8

enhancing the effectiveness of "illustration". In contrast, L2 regularization on embeddings is conducted on individual examples, which cannot capture relationships among data examples.

Analysis of sensitivity to tradeoff parameter γ. Figure 8(right) shows how the test error of FAS-darts2nd on CIFAR-100 varies with γ. When γ increases from 0 to 1, the test error decreases. This is because a larger γ enables the listener to give more feedback to the learner regarding whether the learner can accurately understand image contents. As γ continues to increase, the test error starts to increase. This is because the learner relies too much on the listener's feedback while paying insufficient attention to its own validation performance.

Analysis of sensitivity to the number of learners and listeners. We explored how the number of listeners and learners in FAS affects performance. The experiments were conducted on CIFAR-100, with FAS applied to PC-DARTS. In the first experiment, we tested configurations with two listeners (ResNet-50, DenseNet) and three listeners (ResNet-50, DenseNet, EfficientNet-B0), while maintaining a single learner. Table 11 shows the results. Utilizing multiple listeners does not yield significantly better outcomes compared to using a single listener, which indicates that a single listener is sufficient to emulate the Feynman learning mechanism and provide ample feedback to the learner.

In the second experiment, we set the number of learners to 2 and 3, with each learner having a different architecture. In each configuration, after the search is completed, we evaluate the architecture of each learner on a held-out validation set. The best architecture is then selected to report performance on test data. The number of listeners is set to 1. Table 12 presents the results. As the number of learners increases, the test errors decrease slightly; however, the search costs increase considerably. Overall, employing multiple learners does not prove to be advantageous.

Automatically search for the architecture of the listener's encoder. In addition, we perform experiments which automatically search for the architecture of the listener's encoder and perform parameter tying between the weight parameters of the encoders in learner and listener. By doing this, the total number of parameters, computational cost, and memory costs of our method are greatly reduced, without significantly sacrificing classification performance. And we can avoid the burden of manually setting the listener's architecture. Please see Section H for details. We also conduct an ablation study which removes the listener and trains the learner using its own data and using the listener's augmented data. Our FAS method which trains the listener and learner in lower layers works better than this ablation setting. This further demonstrates the effectiveness of our method and the necessity of using a listener. Please see Section J for details.

E.2. Additional Experimental Results for PQDR

In Table 8 of the main paper, the multi-task learning (MTL) baseline is a bi-level optimization (BLO) based data reweighting method. This method is derived by combining the loss functions of the first and second stages in our PQDR framework into a single one, reducing the number of stages from three to two. The MTL baseline is an extension of two strong BLO-based data reweighting methods (Ren et al., 2018; Shu et al., 2019), and our approach significantly outperforms it. The reason is that these methods lack the peer-questioning mechanism featured in our method.

We compared PQDR with two more data reweighting methods (Wang et al., 2022; Liu et al., 2022) which are not based on BLO. The experiments were conducted using the LXMERT model. Table 13 shows the results. Our method outperforms

Table 13. Compare PQDR with two more baselines

Method	Accuracy	B1	B2	B3	F1
(Wang et al., 2022)	58.1	58.5	3.8	1.5	10.1
(Liu et al., 2022)	58.3	58.2	3.7	1.2	9.6
Ours	**60.4**	**60.1**	**4.5**	**2.8**	**11.7**

Table 14. Data reweighting on ANIMAL-10.

Method	Accuracy
(Chen et al., 2021b)	84.1
(Song et al., 2019)	81.8
(Zhang et al.)	83.4
Ours	**87.9**

these two baselines. The reason is that without using BLO, the two baselines are unable to utilize performance on held-out validation data to guide the data reweighting process.

We applied PQDR for another task – image classification with label noise, to reweight images in a classification task (with C classes) where some training images have incorrect class labels and should be down-weighted during training. The class labels of all validation and test images are correct. For each image-label pair (x, c) in the training set, we associate a learnable weight $a \in [0, 1]$ with it. A smaller a indicates that the label c is incorrect. We multiply a to the classification loss defined on this pair. Our PQDR framework has three learners. Each learner has two classifiers. Our framework consists of three end-to-end stages. In the first stage, each learner trains its first classifier by minimizing classification losses reweighted by weights $\{a_i\}$. These data weights are tentatively fixed at this stage and will be updated later on. It the second stage, each learner randomly selects image pairs (excluding labels) from the validation set and "asks" a question for each image pair (x, y): "What is the KL divergence between the probability distribution on the C classes for x and that for y?" The learner generates an answer using its first classifier trained in the first stage. Meanwhile, each learner trains its second classifier by "answering" questions posed by other learners. Additionally, the second classifier is trained by minimizing reweighted losses defined in the first stage. In the third stage, each learner validates its second classifier on the noise-free validation set. The weights $\{a_i\}$ of the training data are updated by minimizing validation losses. We conducted experiments on the ANIMAL-10 dataset. Table 14 shows the results. Our method significantly outperforms state-of-the-art baselines, further demonstrating the effectiveness of the peer-questioning mechanism.

F. Additional Discussion

Our methods have three limitations. First, compared with bi-level optimization based methods such as DARTS, our multi-level optimization based formulation incurs additional computational costs due to the increased number of levels. Second, our methods incur additional memory cost due to storing extra models such as the listener in the Feynman architecture search (FAS) method. Third, in the FAS method, it is needed to decide which convolutional network should be used as the listener model, which incurs additional tuning efforts.

To address these limitations, we improved our methods from the following aspects. To address the third limitation, we automatically search the architecture of the listener model to avoid the overhead of deciding which convolutional network to use as the listener. To address the first and second limitation, we perform parameter tying. For example, in FAS, we let the feature extraction layers of the learner and the listener share the same weight parameters. In Section H, we conducted experiments which automatically search for the architecture of the listener and perform parameter tying between the listener and learner. Experimental results show that with parameter tying (PT), the computational and memory costs of our method (denoted as FAS-PT) is greatly reduced without significantly sacrificing classification performance. With parameter tying, the costs of our FAS-PT method are very close to those of baselines including Darts2nd, Pdarts, Pcdarts, and Prdarts, while the classification errors of our method are much lower than these baselines.

While our method is mainly developed for improving BLO-based methods, it can be extended to improve non-BLO-based methods as well. In Section I.2, we apply our framework to improve graph neural networks (GNNs) for graph classification, where the GNNs do not learn their meta parameters via BLO. Experiments on five datasets demonstrate the effectiveness of

our method.

To further increase the scalability of Skillearn to many levels of nested optimization problems, we will develop a distributed version of the framework which leverages the GPU resources of multiple machines to speed up computation. Section G.4 provides details.

There are two major differences between our FAS method and knowledge distillation (KD). First, our approach concentrates on enhancing the learner by letting it create effective illustrations for the listener. In contrast, KD focuses on improving the student model by training it with pseudo-labels generated by the teacher. Second, we propose a novel ranking-based method for model interaction, where the learner generates rankings that the listener then fits. In KD, knowledge transfer is carried out through pseudo-labeling. Our ranking-based method, which ranks K examples, requires the identification of the global relationship among the K examples. As a result, our method can capture high-order (specifically, K-th order) relationships among data examples, which contributes to improving model performance. In contrast, pseudo-labeling is conducted on individual examples, which cannot capture relationships among data examples.

One instance of performance collapse in BLO is observed in differentiable architecture search methods, such as DARTS, which produce degenerate architectures with notably poor test performance despite successfully minimizing validation and training losses. Figure 3 in (Chen & Hsieh, 2020b) demonstrates this phenomenon. In this figure, at the end of the search process, the test error of the discovered architectures explodes, while their errors on training and validation data consistently decrease.

G. Scalability to many levels of optimization problems

In this section, we discuss the scalability of our method to many levels of nested optimization problems. The key takeaways are:

- First, via finite difference approximation, we reduce the quadratic costs down to linear, making it possible to scale our method to many levels of nested optimization problems.
- Second, in a case study with 10 levels of nested optimization problems, we conducted experiments to show that our method can scale to many levels of nested optimization problems.
- We conducted experiments to show that the accumulation of approximation errors in our method does not hurt final accuracy.
- To further increase the scalability of our framework, we will develop a distributed version of the framework which leverages the GPU resources of multiple machines to speed up computation.

G.1. Computation cost

Following DARTS (Liu et al., 2019), we use finite difference approximation to calculate matrix-vector multiplication, which reduces the quadratic computational complexity (in terms of matrix dimensions) down to linear. Consider a multi-level optimization problem with K levels, where the optimization variables at level 1 to K are $\alpha_1, \cdots, \alpha_K$ respectively. For $\alpha_1, \cdots, \alpha_{K-1}$, their optimal solutions are approximated using one-step gradient descent updates. Their gradients are straightforward to calculate using back propagation, the same as regular deep neural networks. For α_K, we need to calculate a hypergradient, which involves a term like this:

$$M_1 \times M_2 \cdots M_k \times v, \tag{34}$$

where M_1 to M_k denote Hessian matrices and v is a gradient vector. Calculating this term via ordinary matrix-vector multiplication incurs a quadratic cost in terms of matrix dimensions. To reduce the cost, we leverage finite difference (Liu et al., 2019) to recursively approximate matrix-vector multiplication, from right to left. This can reduce the cost down to linear. The procedure is as follows. First, we approximate $M_k \times v$ using finite difference, resulting in a vector u_k. Then we approximate $M_{k-1} \times u_k$ again using finite difference, resulting in another vector u_{k-1}. This procedure repeats from right to left until all matrix-vector multiplications are approximated. Finite difference approximation is conducted in the following way. Suppose the Hessian matrix is $\nabla^2_{\alpha,w} l(w, \alpha)$ where $l(\cdot)$ is a loss function. Let ϵ be a small scalar and $w^+ = w + \epsilon v$, $w^- = w - \epsilon v$. $\nabla^2_{\alpha,w} l(w, \alpha) \times v$ can be approximated as follows:

$$\nabla^2_{\alpha,w} l(w, \alpha) \times v \approx \frac{\nabla_\alpha l(w^+, \alpha) - \nabla_\alpha l(w^-, \alpha)}{2\epsilon}. \tag{35}$$

Table 15. Candidate angles (degrees) in different SSL tasks

Task ID	Angle 1	Angle 2	Angle 3	Angle 4
1	40	80	120	160
2	35	70	105	140
3	30	60	90	120
4	25	50	75	100
5	20	40	60	80
6	15	30	45	60
7	10	20	30	40
8	5	10	15	20

Table 16. Notations in curriculum SSL

Notation	Meaning
W_k	Weight parameters of the k-th cell
A_k	Architecture of the k-th cell
H_k	Head of the k-th SSL task
$D_k^{(\text{ssl})}$	Dataset of the k-th SSL task
$L_k^{(\text{ssl})}$	Loss function of the k-th SSL task
V	Head of the CIFAR-100 classification task
$D_{tr}^{(\text{cls})}$	Training set of CIFAR-100
$D_{val}^{(\text{cls})}$	Validation set of CIFAR-100
$L^{(\text{cls})}$	Classification loss on CIFAR-100

G.2. Experiments on ten levels of nested optimization problems

To test the scalability of our framework, we applied it to a case study which has 10 levels of optimization problems. The application is progressive self-supervised learning (SSL) (He et al., 2019; Chen et al., 2020b) for image classification. SSL learns useful representations by solving pretext tasks without relying on human-provided labels. We design a sequence of self-supervised learning tasks which have increasing levels of difficulty. Then we use these tasks to train different layers of a convolutional network with a searchable architecture. The experiment was conducted on CIFAR-100.

G.2.1. METHOD

The SSL task is rotation prediction (Gidaris et al., 2018). Given an image, it is rotated clockwise with one of a predefined set of angles. A neural layer is trained by predicting which angle the image is rotated with. For the first SSL task (the easiest one), we set the candidate angles to be 40, 80, 120, 160 degrees. For the second SSL task, to make it harder, we set the candidate angles to be 30, 60, 90, 120 which are closer to each other and are hence more difficult to distinguish. For the third to eighth SSL task, we set the angles to those in Table 15. From task 1 to 8, their candidate angles are increasingly closer and therefore are more difficult to distinguish, which makes these tasks increasingly difficult to solve.

For the k-th SSL task, the SSL training dataset $\mathcal{D}_k^{(\text{ssl})}$ is constructed as follows. Let C_k denote the set of four candidate angles of task k, as shown in Table 15. For each image i in CIFAR-100, we randomly sample an angle c from C_k and rotate i with c degrees, resulting in a rotated image i_c. We set the class label of i_c to be c. (i_c, c) is added into $\mathcal{D}_k^{(\text{ssl})}$ as a training example. The task is to predict the rotation-angle class c of the rotated image i_c. The training loss $L_k^{(\text{ssl})}$ for the k-th SSL task is a four-class classification loss. The four classes are the four angles in C_k.

We use these SSL tasks to train feature learning layers of a convolutional network with a searchable architecture. The network is used for image classification on CIFAR-100. Architecture search space and configuration of the network are the same as those in PCDARTS (Xu et al., 2020). The network consists of 1) a stack of 8 searchable cells (layers), and 2) a feedforward layer used as classification head. For layer k where $1 \leq k \leq 8$, we use the k-th SSL task to train it. Given an example (i_c, c) in the dataset $\mathcal{D}_k^{(\text{ssl})}$ of SSL task k, where i_c is a rotated image and c is the corresponding rotation-angle class label, i_c is fed into the sub-network consisting of layer 1 to layer k for extracting a feature representation. The representation

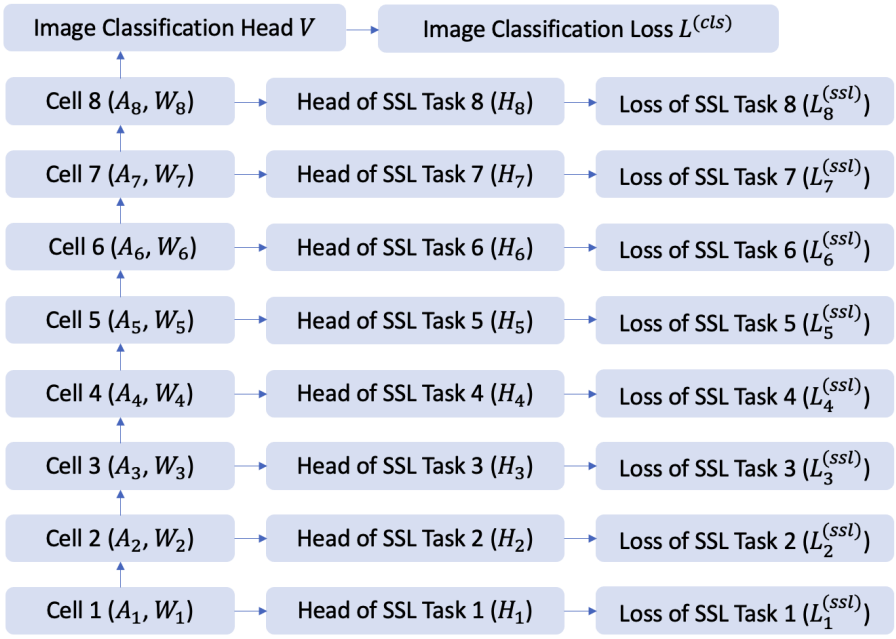

Figure 9. Illustration of the curriculum SSL method.

is then fed into an SSL task head H_k to predict the rotation-angle class label for i_c. A cross-entropy loss is measured on the predicted label and the groundtruth label c. Let W_k and A_k denote the model weights and architecture of layer k. W_k is trained by minimizing the cross-entropy loss.

Next, we describe the corresponding optimization problems. For the first layer, we train its model weights W_1 and the head H_1 of SSL task 1 by minimizing the rotation-angle classification loss $L_1^{(\text{ssl})}$ of SSL task 1 on its dataset $D_1^{(\text{ssl})}$. $L_1^{(\text{ssl})}$ is defined on predicted rotation-angle class labels and groundtruth labels. The prediction is made using 1) W_1 and A_1 (architecture of layer 1) which extract feature representations of rotated images, and 2) H_1 which classifies the extracted representations into rotation-angle classes. The same as PCDARTS, we tentatively fix the architecture A_1 in this optimization problem and will update it later on. The optimization problem is:

$$W_1^*(A_1), H_1^*(A_1) = \text{argmin}_{W_1,H_1}\ L_1^{(\text{ssl})}(W_1, A_1, H_1, D_1^{(\text{ssl})}). \tag{36}$$

For the second layer, we train its model weights W_2 and the head H_2 of SSL task 2 by minimizing the rotation-angle classification loss $L_2^{(\text{ssl})}$ of SSL task 2 on its dataset $D_2^{(\text{ssl})}$. Given a rotated image in $D_2^{(\text{ssl})}$, it is fed into the sub-network consisting of layer 1 and layer 2 to extract a feature representation. Since the model weights $W_1^*(A_1)$ of layer 1 are already trained in the first optimization problem described above, we can directly use them without further training. The variables used for feature extraction include 1) $W_1^*(A_1)$ and A_1 in layer 1, and 2) W_2 and A_2 in layer 2. Similarly, we tentatively fix the architecture A_2 in this optimization problem and will update it later on. The optimization problem is:

$$W_2^*(A_1, A_2), H_2^*(A_1, A_2) = \text{argmin}_{W_2,H_2}\ L_2^{(\text{ssl})}(W_1^*(A_1), A_1, W_2, A_2, H_2, D_2^{(\text{ssl})}). \tag{37}$$

For layer k where $3 \leq k \leq 8$, we train its model weights W_k and the head H_k of SSL task k by minimizing the rotation-angle classification loss $L_k^{(\text{ssl})}$ of SSL task k on its dataset $D_k^{(\text{ssl})}$. Given a rotated image in $D_k^{(\text{ssl})}$, it is fed into the sub-network consisting of layer 1 to layer k to extract a feature representation. Since the model weights $\{W_i^*(\{A_j\}_{j=1}^i)\}_{i=1}^{k-1}$ of layer 1 to $k-1$ are already trained in previous optimization problems, we can directly use them without further training. The variables used for feature extraction include 1) $\{W_i^*(\{A_j\}_{j=1}^i)\}_{i=1}^{k-1}$ and $\{A_i\}_{i=1}^{k-1}$ in layer 1 to $k-1$, and 2) W_k and A_k in layer k. The optimization problem is:

$$W_k^*(\{A_j\}_{j=1}^k), H_k^*(\{A_j\}_{j=1}^k) = \text{argmin}_{W_k,H_k}\ L_k^{(\text{ssl})}(\{W_i^*(\{A_j\}_{j=1}^i), A_i\}_{i=1}^{k-1}, W_k, A_k, H_k, D_k^{(\text{ssl})}). \tag{38}$$

Table 17. Results on CIFAR-100, including classification error (%) on the test set, number of model weights (millions), and search cost (GPU days). * denotes that the results are taken from DARTS$^-$ (Chu et al., 2021). Search cost is measured by GPU days on a Nvidia 1080Ti.

Method	Error on CIFAR-100	Param.	Cost
*ResNet (He et al., 2016)	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
*GDAS (Dong & Yang, 2019a)	18.38	3.4	0.2
*R-DARTS (Zela et al., 2020)	18.01±0.26	-	1.6
*DARTS$^-$ (Chu et al., 2021)	17.51±0.25	3.3	0.4
*DropNAS (Hong et al., 2020)	16.95±0.41	4.4	0.7
*Pcdarts (Xu et al., 2020)	17.96±0.15	3.9	0.14
Curriculum SSL (ours)	**16.27**±0.09	3.8	0.26

After training the eight layers using SSL, we train the CIFAR-100 classification head V by minimizing the classification loss $L^{(\text{cls})}$ on the CIFAR-100 training set $D_{tr}^{(\text{cls})}$. Given an image in $D_{tr}^{(\text{cls})}$, we feed it into the eight layers $\{W_i^*(\{A_j\}_{j=1}^i), A_i\}_{i=1}^8$ to extract a feature representation, which is then fed into V to predict a class label (one of the 100 classes in CIFAR-100). The classification loss is cross-entropy. The optimization problem is:

$$V^*(\{A_j\}_{j=1}^8) = \text{argmin}_V \; L^{(\text{cls})}(\{W_i^*(\{A_j\}_{j=1}^i), A_i\}_{i=1}^8, V, D_{tr}^{(\text{cls})}). \tag{39}$$

Finally, given the feature extraction layers $\{W_i^*(\{A_j\}_{j=1}^i), A_i\}_{i=1}^8$ and head $V^*(\{A_j\}_{j=1}^8)$, we evaluate them on the validation set $D_{val}^{(\text{cls})}$ of CIFAR-100 and optimize the architecture variables $\{A_i\}_{i=1}^8$ by minimizing the validation loss. The optimization problem is:

$$\max_{\{A_i\}_{i=1}^8} \; L^{(\text{cls})}(\{W_i^*(\{A_j\}_{j=1}^i)\}_{i=1}^8, \{A_i\}_{i=1}^8, V^*(\{A_j\}_{j=1}^8), D_{val}^{(\text{cls})}). \tag{40}$$

Putting these pieces together, we have the overall formulation as follows. The notations are summarized in Table 16.

$$\begin{aligned}
&\max_{\{A_i\}_{i=1}^8} \; L^{(\text{cls})}(\{W_i^*(\{A_j\}_{j=1}^i)\}_{i=1}^8, \{A_i\}_{i=1}^8, V^*(\{A_j\}_{j=1}^8), D_{val}^{(\text{cls})}) \\
&s.t.\; V^*(\{A_j\}_{j=1}^8) = \text{argmin}_V \; L^{(\text{cls})}(\{W_i^*(\{A_j\}_{j=1}^i), A_i\}_{i=1}^8, V, D_{tr}^{(\text{cls})}) \\
&\quad s.t.\; W_8^*(\{A_j\}_{j=1}^8), H_8^*(\{A_j\}_{j=1}^8) = \text{argmin}_{W_8,H_8} \; L_8^{(\text{ssl})}(\{W_i^*(\{A_j\}_{j=1}^i), A_i\}_{i=1}^7, W_8, A_8, H_8, D_8^{(\text{ssl})}) \\
&\qquad \cdots \\
&\qquad\quad s.t.\; W_2^*(A_1, A_2), H_2^*(A_1, A_2) = \text{argmin}_{W_2,H_2} \; L_2^{(\text{ssl})}(W_1^*(A_1), A_1, W_2, A_2, H_2, D_2^{(\text{ssl})}) \\
&\qquad\qquad s.t.\; W_1^*(A_1), H_1^*(A_1) = \text{argmin}_{W_1,H_1} \; L_1^{(\text{ssl})}(W_1, A_1, H_1, D_1^{(\text{ssl})})
\end{aligned} \tag{41}$$

This problem is solved using the Algorithm 1 in the main paper. Figure 9 illustrates the curriculum SSL method.

G.2.2. Experimental settings

For the architectures $\{A_i\}_{i=1}^8$, some of them are normal cells while others are reduction cells. The settings of these cells are the same as those in PCDARTS. Please see Section L.2 for details. Architecture tying is performed among these cells. Specifically, all normal cells have the same architecture and all reduction cells have the same architecture.

Next, we describe the experimental settings for architecture search. In vanilla PCDARTS, the number of epochs is 50. Our curriculum SSL method converges faster than PCDARTS. Therefore, we used 30 epochs instead of 50. In each iteration, to train model weights, we randomly sample a mini-batch of input images (batch size is the same as that in PCDARTS) from the CIFAR-100 training set. These images have different output labels in SSL Task 1-8 and in the CIFAR-100 classification task. We use the minibatch of images and their different labels in different tasks to perform gradient-descent updates of model

Table 18. Compare Algorithm 1 and the algorithm in (Sato et al., 2021). They were used to solve the curriculum SSL problem.

Method	Error on CIFAR-100	Param.	Cost
Algorithm 1 (ours)	16.27±0.09	3.8	0.26
Algorithm in (Sato et al., 2021)	16.31±0.05	3.9	0.49

parameters at different layers. To update architectures, we randomly sample a mini-batch of validation examples (batch size is the same as that in PCDARTS) from the CIFAR-100 validation set to calculate hypergradients. Hyerparameter settings in curriculum SSL mostly follow those in PCDARTS. Model weights are optimized using SGD, with an initial learning rate of 0.1 (adjusted using a cosine decay scheduler), a momentum of 0.9, and a weight decay of 3e-4. The architecture variables A are optimized using Adam (Kingma & Ba, 2015) with a fixed learning rate of $6e-4$, a momentum of $(0.5, 0.999)$, and a weight decay of 0.001. The experimental settings for architecture evaluation are the same as those in Section 6.1.2 in the main paper.

G.2.3. RESULTS

Table 17 shows the results. As can be seen, our curriculum SSL method achieves lower test error than baselines while the computational cost of our method is not substantially higher than baselines. Our method finishes training within 0.26 GPU days on a Nvidia 1080Ti. The time cost is not high. This demonstrates that our method can scale to many levels of nested optimization problems.

G.3. Accumulation of errors

The curriculum SSL problem was solved using Algorithm 1 in the main paper, which approximates optimal solutions in lower-level problems using one-step gradient-descent updates and approximates matrix-vector multiplication using finite difference. To investigate how the accumulation of errors incurred by these approximations influences the final accuracy, we solve the curriculum SSL problem using another optimization algorithm proposed by (Sato et al., 2021). The algorithm in (Sato et al., 2021) has smaller approximation errors since the authors use multiple (for example, 10 in their experiments) iterative updates to approximate lower-level problems and prove that the algorithm converges to the exact solution as the number of iterative updates goes to infinity. Table 18 compares the classification error on the CIFAR-100 test set. As can be seen, Algorithm 1 in our method achieves performance similar to the algorithm in (Sato et al., 2021), which demonstrates that the solution found by Algorithm 1 is as good as that found by the more exact algorithm in (Sato et al., 2021). This implies that the accumulation of errors in Algorithm 1 does not significantly sacrifice the final performance.

G.4. Future work for further increasing the scalability of our framework

To further increase the scalability of our framework, we will develop a distributed version of the framework which leverages the GPU resources of multiple machines to speed up computation. Given a multi-level optimization (MLO) problem with many levels of nested optimization problems, we use a dataflow graph to represent the MLO problem. The distributed framework partitions the MLO dataflow graph onto different machines and performs distributed and parallel calculation of hypergradients. Each machine stores one partition and performs computation related to this partition. Machines transfer intermediate results to each other based on the dependency relationships of partitions in the MLO dataflow graph. Pipelining will be used to maximize the throughputs of computations in all machines. Efficient communication mechanisms will be developed to reduce inter-machine communication overhead.

H. Experiments on parameter tying in FAS

To 1) make a fair comparison with baselines in terms of model size, 2) reduce computation and memory costs, and 3) avoid manually setting the listener's architecture, we perform parameter tying between the listener and learner. For the listener, we let its feature extraction layers share the same architecture and weight parameters with the learner. To perform parameter tying, the listener cannot be ResNet-50 anymore and needs to have a learnable architecture similar to the learner. For both the learner and listener, we set them to be a stack of 7 cells, sharing the same architecture A as the first 7 cells in DARTS. Parameter tying is performed by making the weight parameters (denoted as E_s) of the first six cells in the learner and listener be the same. For the seventh cell, the learner and listener have different weight parameters, denoted as E_{le} and E_{li}

Table 19. Notations in parameter-tying FAS

Notation	Meaning
A	Shared architecture of the learner and listener
E_s	Weight parameters of the feature learning layers shared by the learner and listener
E_{le}	Weight parameters of the learner-specific feature learning layer
H_{le}	Classification head of the learner
E_{li}	Weight parameters of the listener-specific feature learning layer
H_{li}	Classification head of the listener

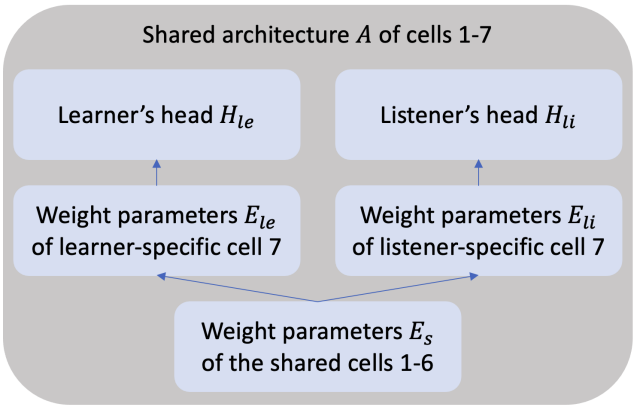

Figure 10. Illustration of FAS with parameter tying.

respectively. Let H_{le} and H_{li} denote the classification head of the learner and listener respectively. Via parameter tying, the number of all parameters (i.e., model size) in our method (including those of the listener) is roughly the same as that in baselines including DARTS, PDARTS, and PCDARTS where the number of cells is 8. After reducing the number of parameters in our method via parameter tying, its computational and memory costs are reduced as well accordingly. Besides, since the listener's architecture is searched automatically, we do not need to manually select its architecture any more.

The formulation is:

$$
\begin{aligned}
&\min_A \; L(A, E_s^*(A), E_{le}^*(A), H_{le}^*(A), D^{(\text{val})}) + \gamma L(A, E_s^*(A), E_{li}^*(A), H_{li}^*(A), D^{(\text{val})}) \\
&\quad s.t.\; E_{li}^*(A), H_{li}^*(A) = \text{argmin}_{E_{li}, H_{li}} \; L(A, E_s^*(A), E_{li}, H_{li}, D^{(\text{tr})}) \\
&\qquad s.t.\; \forall i, f(o_i(1; A, E_s^*(A), E_{le}^*(A)), x_i; A, E_s^*(A), E_{li}) > \cdots \\
&\qquad\quad > f(o_i(K; A, E_s^*(A), E_{le}^*(A)), x_i; A, E_s^*(A), E_{li}) \\
&\quad s.t.\; E_s^*(A), E_{le}^*(A), H_{le}^*(A) = \text{argmin}_{E_s, E_{le}, H_{le}} \; L(A, E_s, E_{le}, H_{le}, D^{(\text{tr})})
\end{aligned}
\tag{42}
$$

Table 19 summarizes the notations. Figure 10 illustrates FAS with parameter tying. The hyperparameter settings of parameter tying FAS are the same as those in Section 6.1.2 in the main paper. For the SE and MTL baselines, we perform parameter tying in the same way.

Table 20 shows the results on CIFAR-100 and CIFAR-10. Table 21 shows the results on ImageNet. As can be seen, our method still achieves lower classification errors than baselines while the number of parameters in our method is approximately the same as that in baselines. In Table 1 and 3 in the main paper, the parameter number is for the searched architecture of the learner model only, which does not include the parameter number of the listener model. In Table 20 and Table 21, the reported parameter number is the total number of parameters in the learner and listener.

Table 22 shows the computational and memory costs of FAS with parameter tying (FAS-PT) on CIFAR-100 and CIFAR-10. Table 23 shows the computational and memory costs of FAS-PT on ImageNet. FAS without PT is denoted as FAS-NoPT. As can be seen, with parameter tying, the computational and memory costs of FAS-PT are much lower those of FAS-NoPT, especially on Darts2nd and Pdarts, with slight increase of classification errors. The costs of FAS-PT are very close to those

Table 20. Results of FAS with parameter tying (PT), on CIFAR-100 and CIFAR-10. PT is performed for SE and MTL as well. The number of parameters in listener models of SE-PT, MTL-PT, and FAS-PT are counted into the total number of parameters. In entries with an X/Y format, X and Y denote results for CIFAR-100 and CIFAR-10 respectively.

Method	Error-C100	Error-C10	# Total Parameters	Cost
*ResNet (He et al., 2016)	22.10	6.43	1.7/1.7	-/-
*DenseNet (Huang et al., 2017)	17.18	3.46	25.6/25.6	-/-
*PNAS (Liu et al., 2018a)	19.53	3.41±0.09	3.2/3.2	150/150
*ENAS (Pham et al., 2018)	19.43	2.89	4.6/4.6	0.5/0.5
*AmoebaNet (Real et al., 2019)	18.93	2.55±0.05	3.1/3.1	3150/3150
*GDAS (Dong & Yang, 2019a)	18.38	2.93	3.4/3.4	0.2/0.2
*R-DARTS (Zela et al., 2020)	18.01±0.26	2.95±0.21	-/-	1.6/1.6
*DARTS$^-$ (Chu et al., 2021)	17.51±0.25	2.59±0.08	3.3/3.3	0.4/0.4
*DropNAS (Hong et al., 2020)	16.95±0.41	2.58±0.14	4.4/4.1	0.7/0.6
*DrNAS (Chen et al., 2021a)	-	2.54±0.03	-/4.0	-/0.4
*ISTA-NAS (Yang et al., 2020)	-	2.54±0.05	-/3.3	-/0.1
*MiLeNAS (He et al., 2020a)	-	2.51±0.11	-/3.9	-/0.3
*GAEA (Li et al., 2021)	-	2.50±0.06	-/-	-/0.1
*PDARTS-ADV (Chen & Hsieh, 2020a)	-	2.48±0.02	-/3.4	-/1.1
*Darts2nd (Liu et al., 2019)	20.58±0.44	2.76±0.09	3.1/3.3	4.0/4.0
SE-PT-darts2nd (Kokkinos, 2017)	19.86±0.32	2.76±0.06	3.2/3.3	4.1/4.1
MTL-PT-darts2nd (Maninis et al., 2019)	18.93±0.23	2.82±0.11	3.1/3.2	4.0/4.0
FAS-PT-darts2nd (ours)	**17.47**±0.14	**2.62**±0.04	3.3/3.3	4.0/4.0
*Pdarts (Chen et al., 2019)	17.42±0.14	2.54±0.04	3.6/3.5	0.3/0.3
SE-PT-pdarts (Kokkinos, 2017)	17.85±0.11	2.66±0.13	3.6/3.6	0.3/0.3
MTL-PT-pdarts (Maninis et al., 2019)	17.38±0.10	2.74±0.06	3.7/3.6	0.4/0.4
FAS-PT-pdarts (ours)	**16.37**±0.06	**2.51**±0.03	3.6/3.6	0.4/0.4
†Pcdarts (Xu et al., 2020)	17.96±0.15	2.57±0.07	3.9/3.6	0.1/0.1
SE-PT-pcdarts (Kokkinos, 2017)	18.39±0.07	2.84±0.12	3.9/3.8	0.2/0.2
MTL-PT-pcdarts (Maninis et al., 2019)	18.21±0.11	2.68±0.07	3.9/3.6	0.2/0.2
FAS-PT-pcdarts (ours)	**16.69**±0.08	**2.53**±0.01	3.9/3.7	0.2/0.2
†Prdarts (Zhou et al., 2020)	16.48±0.06	2.37±0.03	3.4/3.5	0.2/0.2
SE-PT-prdarts (Kokkinos, 2017)	17.67±0.10	2.56±0.09	3.5/3.5	0.2/0.2
MTL-PT-prdarts (Maninis et al., 2019)	17.41±0.12	2.59±0.10	3.4/3.5	0.3/0.3
FAS-PT-prdarts (ours)	**16.20**±0.06	**2.33**±0.02	3.4/3.5	0.3/0.3

of baselines including Darts2nd, Pdarts, Pcdarts, and Prdarts, while the classification errors of FAS-PT are much lower than these baselines.

I. Experiments on other datasets and non-BLO problem

In this section, we apply our FAS method to six text classification datasets and apply our general framework for a graph classification problem which is not based on bi-level optimization.

I.1. Apply FAS to GLUE text datasets

In this section, we apply the proposed FAS method for text classification.

Dataset We applied FAS on six text classification datasets in the GLUE collection (Wang et al., 2018). They are SST-2, MRPC, QQP, MNLI, QNLI and RTE. SST-2 contains (movie review, sentiment label) pairs for sentiment classification. The sentiment label is binary: either positive or negative. On MRPC and QQP, the task is to predict whether two sentences have equivalent semantics. On MNLI, QNLI, and RTE, the task is to predict textual entailment.

Baselines Our method is compared with the following baselines: 1) BERT (Devlin et al., 2018), 2) BERT-PKD (Sun et al., 2019), 3) Distil-BERT (Sanh et al., 2019), 4) TinyBERT (Jiao et al., 2019), 5) BiLSTM$_{SOFT}$ (Tang et al., 2019), 6) AdaBERT (Chen et al., 2020a), 7) SE-AdaBERT (Kokkinos, 2017), and 8) MTL-AdaBERT (Maninis et al., 2019).

Experimental Setup For the learner's encoder architecture, its search space contains candidate operations including dilated convolution, 1D convolution, pooling, identity, and zero. Convolution operations have a sequence of sub-operations including ReLU activation, convolution, batch normalization. The candidate kernel sizes in dilated convolutions are 3, 5, and 7. Candidate pooling operations include max pooling and average pooling. The kernel size of pooling operations is set to 3. For convolution and pooling operations, the "SAME" padding mechanism is used. The learner's classification head is a feedforward layer. We use the text classification model in EDA (Wei & Zou, 2019) as the listener, which has the following

Table 21. Results of parameter-tying (PT) FAS on ImageNet. The number of parameters in listener models of SE-PT, MTL-PT, and FAS-PT are counted into the total number of parameters. FAS-PT-darts2nd-cifar10 means the architecture is searched using FAS-PT on CIFAR-10, where the search space is the same as that in DARTS-2nd. The search cost of FAS-PT-darts2nd-cifar10 is the same as that of FAS-PT-darts2nd on Cifar10. Similar meanings hold for other notations like this.

Method	Top-1	Top-5	# Total Parameters	Cost
*Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	-
*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
*AmoebaNet-C (Real et al., 2019)	24.3	7.6	6.4	3150
*SDARTS-ADV-CIFAR10 (Chen & Hsieh, 2020a)	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-ImageNet (Chu et al., 2020)	24.4	7.4	4.3	3.0
*PR-DARTS-cifar10 (Zhou et al., 2020)	24.1	7.3	5.0	0.2
*DARTS$^+$-CIFAR100 (Liang et al., 2019)	23.7	7.2	5.1	0.2
*Darts2nd-cifar10 (Liu et al., 2019)	26.7	8.7	4.7	4.0
SE-PT-darts2nd-cifar10 (Kokkinos, 2017)	26.3	8.5	4.7	4.1
MTL-PT-darts2nd-cifar10 (Maninis et al., 2019)	26.1	8.2	4.9	4.0
FAS-PT-darts2nd-cifar10 (ours)	**25.5**	**7.8**	4.7	4.0
*Pdarts-cifar10 (Chen et al., 2019)	24.4	7.4	4.9	0.3
SE-PT-pdarts-cifar10 (Kokkinos, 2017)	24.3	7.4	4.9	0.3
MTL-PT-pdarts-cifar10 (Maninis et al., 2019)	24.3	7.3	5.1	0.4
FAS-PT-pdarts-cifar10 (ours)	**24.0**	**7.1**	4.9	0.4
*Pdarts-cifar100 (Chen et al., 2019)	24.7	7.5	5.1	0.3
SE-PT-pdarts-cifar100 (Kokkinos, 2017)	24.7	7.5	5.2	0.3
MTL-PT-pdarts-cifar100 (Maninis et al., 2019)	24.6	7.5	5.1	0.4
FAS-PT-pdarts-cifar100 (ours)	**24.4**	**7.3**	5.2	0.4
*Pcdarts-imagenet (Xu et al., 2020)	24.2	7.3	5.3	3.8
SE-PT-pcdarts-imagenet (Kokkinos, 2017)	24.0	7.1	5.3	3.8
MTL-PT-pcdarts-imagenet (Maninis et al., 2019)	23.7	7.0	5.3	3.9
FAS-PT-pcdarts-imagenet (ours)	**23.4**	**6.6**	5.3	3.9

layers: an input layer, a bi-directional LSTM (Hochreiter & Schmidhuber, 1997) layer with 64 hidden units, dropout where the dropout probability is 0.5, a bi-directional LSTM layer with 32 hidden units, dropout where the dropout probability is 0.5, ReLU activation, a dense layer with 20 hidden units, and a softmax output layer. The layers up to (including) the dense layer are used as the listener's encoder. The softmax output layer is used as a classification head. Text augmentation is performed using EDA (Wei & Zou, 2019). The tradeoff parameter γ is set to 1. The tradeoff parameter λ is set to 2. The number K of augmented images is set to 5. Loss function is cross-entropy. The maximum text length is set to 150. The search algorithm runs for 80 epochs. Batch size is 128. We use Adam (Kingma & Ba, 2014) to optimize architecture variables. Learning rate is set to $3e-4$ and weight decay is set to $1e-3$. We use SGD to optimize weight parameters, with an initial learning rate of $2e-2$, a cosine learning rate scheduler, and a momentum of 0.9.

Main results Table 24 shows the results. Our method works better than AdaBERT, SE, and MTL, which further demonstrates the effectiveness of our method in searching for better-performing neural architectures. Our method has much fewer parameters and much faster inference speed than BERT_{12} and BERT_{12}-T while the classification performance of our method is on par with BERT_{12} and BERT_{12}-T.

I.2. Apply our framework to graph neural network based graph classification

In this section, we apply our framework to an ML problem which is not based on bi-level optimization. The ML problem is graph classification based on graph neural networks. In this problem, there are no meta parameters to learn. Only weight parameters are learned. Therefore, it is not based on bi-level optimization.

Method We make small changes to our FAS method to solve a graph neural network (GNN) based graph classification problem, leveraging the idea of Feynman learning. We refer to our method as Feynman-GNN, which consists of a learner model and a listener model. The learner is a GNN which has a graph encoder E with a manually-designed architecture and a classification head H. The listener is another GNN which has a graph encoder F and a classification head G. There are three stages in Feynman-GNN. At the first stage, we train the graph encoder E of the learner by minimizing a cross-entropy based classification loss L defined on the training set $D^{(\text{tr})}$ of a graph classification dataset, with the learner's classification

Table 22. Computational and memory costs of FAS with parameter tying (FAS-PT) on CIFAR-100 and CIFAR-10. FAS without PT is denoted as FAS-NoPT. Search (computation) cost is measured using GPU days. Memory cost is measured using MiB. In entries with an X/Y format, X and Y denote results for CIFAR-100 and CIFAR-10 respectively.

Method	Error on CIFAR-100	Error on CIFAR-10	Search cost	Memory cost
*Darts2nd (Liu et al., 2019)	20.58±0.44	2.76±0.09	4.0/4.0	11053/11008
SE-PT-Darts2nd (Kokkinos, 2017)	19.86±0.32	2.76±0.06	4.1/4.1	11105/11173
MTL-PT-Darts2nd (Maninis et al., 2019)	18.93±0.23	2.82±0.11	4.0/4.0	11098/11062
FAS-NoPT-Darts2nd (ours)	17.12±0.18	2.60±0.05	5.3/5.3	20159/20037
FAS-PT-Darts2nd (ours)	17.47±0.14	2.62±0.04	4.0/4.0	11117/11085
*Pdarts (Chen et al., 2019)	17.42±0.14	2.54±0.04	0.3/0.3	9659/9721
SE-PT-Pdarts (Kokkinos, 2017)	17.85±0.11	2.66±0.13	0.3/0.3	9714/9766
MTL-PT-Pdarts (Maninis et al., 2019)	17.38±0.10	2.74±0.06	0.4/0.4	9732/9701
FAS-NoPT-Pdarts (ours)	16.01±0.09	2.49±0.06	0.7/0.7	19261/19210
FAS-PT-Pdarts (ours)	16.37±0.06	2.51±0.03	0.4/0.4	9744/9806
†Pcdarts (Xu et al., 2020)	17.96±0.15	2.57±0.07	0.1/0.1	10058/10024
SE-PT-Pcdarts (Kokkinos, 2017)	18.39±0.07	2.84±0.12	0.2/0.2	10092/10136
MTL-PT-Pcdarts (Maninis et al., 2019)	18.21±0.11	2.68±0.07	0.2/0.2	10105/10083
FAS-NoPT-Pcdarts (ours)	16.41±0.12	2.51±0.02	0.3/0.3	19784/19725
FAS-PT-Pcdarts (ours)	16.69±0.08	2.53±0.01	0.2/0.2	10114/10072
†Prdarts (Zhou et al., 2020)	16.48±0.06	2.37±0.03	0.2/0.2	10159/10119
SE-PT-Prdarts (Kokkinos, 2017)	17.67±0.10	2.56±0.09	0.2/0.2	10196/10152
MTL-PT-Prdarts (Maninis et al., 2019)	17.41±0.12	2.59±0.10	0.3/0.3	10217/10195
FAS-NoPT-Prdarts (ours)	16.12±0.08	2.32±0.03	0.4/0.4	19510/19591
FAS-PT-Prdarts (ours)	16.20±0.06	2.33±0.02	0.3/0.3	10235/10207

head H tentatively fixed. This stage amounts to solving the following optimization problem.

$$E^*(H) = \text{argmin}_E \; L(E, H, D^{(\text{tr})}). \tag{43}$$

At the second stage, similarly to FAS, the learner illustrates its understanding of graphs to the listener by ranking augmented graphs. Graph augmentation is performed using the method in (Zeng & Xie, 2021). This stage amounts to solving the following optimization problem.

$$\begin{aligned} &F^*(E^*(H)), G^* = \text{argmin}_{F,G} \; L(F, G, D^{(\text{tr})}) \\ &\quad s.t. \; \forall i, f(o_i(1; E^*(H)), x_i; F) > \cdots > f(o_i(K; E^*(H)), x_i; F) \end{aligned} \tag{44}$$

At the third stage, we validate the learner and listener and update H by minimizing validation losses on a validation set $D^{(\text{val})}$, which amounts to solving the following problem:

$$\min_H \; L(E^*(H), H, D^{(\text{val})}) + \gamma L(F^*(E^*(H)), G^*, D^{(\text{val})}). \tag{45}$$

Putting these pieces together, we have the following multi-level optimization based formulation.

$$\begin{aligned} &\min_H \; L(E^*(H), H, D^{(\text{val})}) + \gamma L(F^*(E^*(H)), G^*, D^{(\text{val})}) \\ &s.t. \; F^*(E^*(H)), G^* = \text{argmin}_{F,G} \; L(F, G, D^{(\text{tr})}) \\ &\quad s.t. \; \forall i, f(o_i(1; E^*(H)), x_i; F) > \cdots > f(o_i(K; E^*(H)), x_i; F) \\ &\quad s.t. \; E^*(H) = \text{argmin}_E \; L(E, H, D^{(\text{tr})}) \end{aligned} \tag{46}$$

We solve this problem using Algorithm 1 in the main paper.

Datasets We performed the experiments on five graph classification datasets[6], including PROTEINS, D&D, NCI1, NCI109, and Mutagenicity. Data examples are (graph, class label) pairs. Graphs in PROTEINS and D&D are protein graphs.

[6]Datasets can be downloaded from https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

Table 23. Computational and memory costs of FAS with parameter tying (FAS-PT) on ImageNet. FAS without PT is denoted as FAS-NoPT. Search (computation) cost is measured using GPU days. Memory cost is measured using MiB. FAS-PT-darts2nd-cifar10 means the architecture is searched using FAS-PT on CIFAR-10, where the search space is the same as that in DARTS-2nd. The search (computation) and memory costs of FAS-PT-darts2nd-cifar10 is the same as that of FAS-PT-darts2nd on Cifar10. Similar meanings hold for other notations like this. Pcdarts-imagenet experiments were performed on 8 GPUs.

Method	Top-1 error	Top-2 error	Search cost	Memory cost
*Darts2nd-cifar10 (Liu et al., 2019)	26.7	8.7	4.0	11008
SE-PT-darts2nd-cifar10 (Kokkinos, 2017)	26.3	8.5	4.1	11173
MTL-PT-darts2nd-cifar10 (Maninis et al., 2019)	26.1	8.2	4.0	11062
FAS-darts2nd-cifar10 (ours)	25.3	7.7	5.3	20037
FAS-PT-darts2nd-cifar10 (ours)	25.5	7.8	4.0	11085
*Pdarts-cifar10 (Chen et al., 2019)	24.4	7.4	0.3	9721
SE-PT-pdarts-cifar10 (Kokkinos, 2017)	24.3	7.4	0.4	9766
MTL-PT-pdarts-cifar10 (Maninis et al., 2019)	24.3	7.3	0.4	9701
FAS-pdarts-cifar10 (ours)	23.9	7.1	0.7	19210
FAS-PT-pdarts-cifar10 (ours)	24.0	7.1	0.4	9806
*Pdarts-cifar100 (Chen et al., 2019)	24.7	7.5	0.3	9659
SE-PT-pdarts-cifar100 (Kokkinos, 2017)	24.7	7.5	0.4	9714
MTL-PT-pdarts-cifar100 (Maninis et al., 2019)	24.6	7.5	0.4	9732
FAS-pdarts-cifar100 (ours)	24.3	7.3	0.7	19261
FAS-PT-pdarts-cifar100 (ours)	24.4	7.3	0.4	9744
*Pcdarts-imagenet (Xu et al., 2020)	24.2	7.3	0.1	154017
SE-PT-pcdarts-imagenet (Kokkinos, 2017)	24.0	7.1	0.1	154358
MTL-PT-pcdarts-imagenet (Maninis et al., 2019)	23.7	7.0	0.1	154511
FAS-pcdarts-imagenet (ours)	23.2	6.5	5.2	307366
FAS-PT-pcdarts-imagenet (ours)	23.4	6.6	0.1	154425

Their binary class labels are about whether the proteins are non-enzyme. Graphs in NCI1, NCI109, and Mutagenicity are chemical compound graphs. Class labels in NCI1 and NCI109 are about whether the graphs can prevent cancer cells from growing. Class labels in Mutagenicity are about whether the graphs are mutagens. Each dataset is randomly split into a train, validation, and test set with a ratio of 8:1:1. We repeat the random split for 10 times, and report the average accuracy together with standard deviations.

Experimental Setup The graph encoder and classification head in the learner GNN are the same as those in HGPSL (Zhang et al., 2019). We set the node representation dimension to 128, the number of HGPSL layers to 3, the dropout ratio to 0.2, and the pooling ratio to 0.4. The listener's classification head is the same as that in HGPSL. The listener's graph encoder is similar to that in HGPSL, with the only difference being that the listener uses 1 HGPSL layer instead of 3. Graph augmentation is performed by performing three consecutive random alteration operations, the same as (Zeng & Xie, 2021). The tradeoff parameter γ is set to 1. The tradeoff parameter λ is set to 2. The number K of augmented images is set to 5. Model weights are optimized using Adam (Kingma & Ba, 2015). Learning rate is set to 0.001. Batch size is set to 16. Weight decay is set to 0.001.

Baselines We compare our method with the following baselines: GRAPHLET (Shervashidze et al., 2009), Shortest-Path (SP) Kernel (Borgwardt & Kriegel, 2005), Weisfeiler-Lehman (WL) Kernel (Shervashidze et al., 2011), GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2017), Set2Set (Vinyals et al., 2015), DGCNN (Zhang et al., 2018), DiffPool (Ying et al., 2018), EigenPool (Ma et al., 2019), gPool (Gao & Ji, 2019), SAGPool (Lee et al., 2019), EdgePool (Diehl, 2019), and HGPSL (Zhang et al., 2019).

Results Table 25 shows graph classification accuracy. Our method outperforms HGPSL and other baselines. This demonstrates that our framework can be applied to improve ML methods that are not based on bi-level optimization.

J. Ablation experiments on using the data of the learner and listener to train the learner

To further evaluate the effectiveness of the proposed Feynman Architecture Search method which trains the listener and learner in lower layers, we conducted an ablation experiment which removes the listener and trains the learner using the data

Table 24. Results on the six text datasets. "Param." denotes model parameter number. "Inference" denotes the speedup of inference time compared with BERT_{12}.

Method	Param.	Inference	SST-2	MRPC	QQP	MNLI	QNLI	RTE	Average
BERT_{12}	109M	1x	93.5	88.9	71.2	84.6	90.5	66.4	82.5
BERT_{12}-T	109M	1x	93.3	88.7	71.1	84.8	90.4	66.1	82.4
BERT_{6}-PKD	67.0M	1.9x	92.0	85.0	70.7	81.5	89.0	65.5	80.6
BERT_{3}-PKD	45.7M	3.7x	87.5	80.7	68.1	76.7	84.7	58.2	76.0
DistilBERT_{4}	52.2M	3.0x	91.4	82.4	68.5	78.9	85.2	54.1	76.8
TinyBert_{4}	14.5M	9.4x	92.6	86.4	71.3	82.5	87.7	62.9	80.6
BiLSTM_{SOFT}	10.1M	7.6x	90.7	-	68.2	73.0	-	-	-
AdaBERT	8.3M	16.1x	91.9	85.3	70.2	81.9	86.9	64.8	80.2
SE-AdaBERT	8.5M	15.4x	91.5	85.3	70.5	81.2	86.4	65.2	80.0
MTL-AdaBERT	8.3M	16.2x	92.1	85.5	70.3	82.0	87.1	64.8	80.3
Ours-AdaBERT	8.2M	16.4x	93.5	87.4	71.9	83.6	88.9	66.8	82.0

Table 25. Graph Classification Accuracy (%).

Method	PROTEINS	D&D	NCI1	NCI109	Mutagenicity
GRAPHLET	72.23±4.49	72.54±3.83	62.48±2.11	60.96±2.37	56.65±1.74
SP	75.71±2.73	78.72±3.89	67.44±2.76	67.72±2.28	71.63±2.19
WL	76.16±3.99	76.44±2.35	76.65±1.99	76.19±2.45	80.32±1.71
GCN	75.17±3.63	73.26±4.46	76.29±1.79	75.91±1.84	79.81±1.58
GraphSAGE	74.01±4.27	75.78±3.91	74.73±1.34	74.17±2.89	78.75±1.18
GAT	74.72±4.01	77.30±3.68	74.90±1.72	75.81±2.68	78.89±2.05
Set2Set	79.33±0.84	70.83±0.84	69.62±1.32	73.66±1.69	80.84±0.67
DGCNN	79.99±0.44	70.06±1.21	74.08±2.19	78.23±1.31	80.41±1.02
DiffPool	79.90±2.95	78.61±1.32	77.73±0.83	77.13±1.49	80.78±1.12
EigenPool	78.84±1.06	78.63±1.36	77.24±0.96	75.99±1.42	80.11±0.73
gPool	80.71±1.75	77.02±1.32	76.25±1.39	76.61±1.39	80.30±1.54
SAGPool	81.72±2.19	78.70±2.29	77.88±1.59	75.74±1.47	79.72±0.79
EdgePool	82.38±0.82	79.20±2.61	76.56±1.01	79.02±1.89	81.41±0.88
HGPSL	84.91±1.62	80.96±1.26	78.45±0.77	80.67±1.16	82.15±0.58
Ours+HGPSL	**85.68**±0.91	**82.01**±1.15	**79.85**±0.92	**81.04**±1.17	**82.47**±0.60

of both the learner and the listener. We denote this ablation setting as Learner-Only. The experiments were conducted on CIFAR-100 and CIFAR-10. The methods were applied to DARTS and PDARTS. The learner's training data is $D^{(\text{tr})}$, which is the training set of CIFAR-100 or CIFAR-10. The listener's training data includes augmented examples and $D^{(\text{tr})}$. For each original example in $D^{(\text{tr})}$, five augmented images are generated via random rotation, flipping, cropping, and color jitter. The class label of an augmented image is set to that of the corresponding original image where the augmented image is generated from. Let $D^{(\text{aug})}$ denote the augmented data, which consists of (augmented image, class label) pairs. We train the learner by minimizing cross-entropy based classification losses on $D^{(\text{aug})}$ and $D^{(\text{tr})}$. The formulation of Learner-Only is:

$$\begin{aligned} &\min_A \ L(A, E^*(A), H^*(A), D^{(\text{val})}) \\ &s.t \qquad E^*(A), H^*(A) = \text{argmin}_{E,H} \ \ L(A, E, H, D^{(\text{tr})}) + \lambda L(A, E, H, D^{(\text{aug})}) \end{aligned} \tag{47}$$

where the tradeoff parameter λ is set to 1. The hyperparameter settings of Learner-Only are the same as those in Section 6.1.2 in the main paper.

Table 26 shows the results, where we make two observations. As can be seen, our FAS method which trains the listener and learner in lower layers works better than Learner-Only. This further demonstrates the effectiveness of our method and the necessity of using a listener. Our method leverages a listener model to provide feedback on the learner's architecture and improves the architecture based on the feedback. Such a mechanism is lacking in Learner-Only.

K. Some noisy examples identified by our PQDR method

Figure 11 shows some randomly-sampled noisy examples identified by our method.

Table 26. Ablation study results on Learner-Only.

Method	Error-C100	Error-C10
Learner-Only-Darts	20.19±0.37	2.74±0.06
FAS-Darts (ours)	**17.12**±0.18	**2.60**±0.05
Learner-Only-Pdarts	17.15±0.10	2.53±0.02
FAS-Pdarts (ours)	**16.01**±0.09	**2.49**±0.06

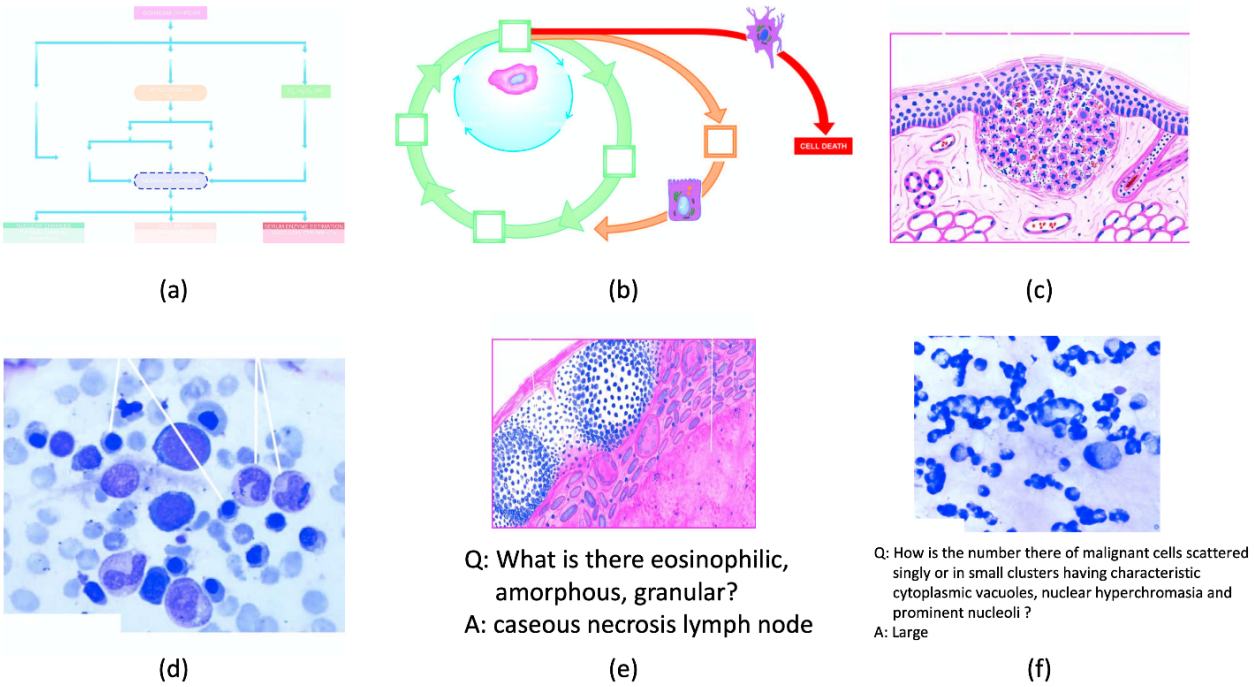

Figure 11. Some randomly-sampled noisy examples identified by our method. In (a) and (b), the images are not about pathology. In (c) and (d), the images contain human-annotated arrows. In (e) and (f), the questions are not correct in syntax and semantics.

L. Experimental details of neural architecture search

L.1. DARTS2nd based experiments

For methods based on DARTS2nd, including FAS-darts2nd (ours), MTL-darts2nd, SE-darts2nd, the experimental settings are similar. λ in Eq.(47) is set to 0.1. In search spaces of DARTS, the candidate operations include: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and zero. The network is a stack of multiple cells, each consisting of 7 nodes. The stride of all operations is set to 1. The convolved feature maps are padded to preserve their spatial resolution. The order for convolutional operations is ReLU-Conv-BN. Each separable convolution is applied twice. The convolutional cell has 7 nodes. The output node is the depthwise concatenation of all intermediate nodes, excluding the input nodes. We create a network by stacking 8 cells. The first and second nodes of cell k are equal to the outputs of cell $k-2$ and cell $k-1$, respectively. 1×1 convolutions are inserted when necessary. Reduction cells are located at the 1/3 and 2/3 of the total depth of the network. In reduction cells, operations adjacent to the input nodes have a stride of 2.

For CIFAR-10 and CIFAR-100, during architecture search, the learner's network is a stack of 8 cells, each consisting of 7 nodes, with the initial channel number set to 16. The search algorithm ran for 50 epochs with a batch size of 64. Network weights are optimized using SGD, with an initial learning rate of 0.025 (adjusted using a cosine decay scheduler),

a momentum of 0.9, and a weight decay of 3e-4. The architecture variables A were optimized using Adam (Kingma & Ba, 2015) with a learning rate of 0.001, a momentum of $(0.5, 0.999)$, and a weight decay of 0.001. The learning rate was scheduled with cosine scheduling. The architecture variables were initialized with zero initialization.

During architecture evaluation, for CIFAR-10 and CIFAR-100, a larger network of the learner is formed by stacking 20 copies of the searched cell. The composed large network is trained on the combination of $D_t^{(\text{tr})}$ and $D_t^{(\text{val})}$. The initial channel number was set to 36. We trained the network with a batch size of 96, an epoch number of 600. 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. On ImageNet, we evaluate two types of architectures: 1) those searched on a subset of ImageNet; 2) those searched on CIFAR-10 or CIFAR-100. In either type, 14 copies of optimally searched cells are stacked into a large network, which was trained on the 1.2M training images, with a batch size of 1024, an epoch number of 250, an initial learning rate of 0.5, and a weight decay of 3e-5. Initial channel number was set to 48. Cutout, path dropout of probability 0.2 and auxiliary towers with weight 0.4 were applied.

L.2. PC-DARTS based experiments

For methods based on PC-DARTS, including FAS-pcdarts (ours), MTL-pcdarts, SE-pcdarts, the experimental settings are similar. λ in Eq.(47) is set to 0.1. The search space of PC-DARTS follows that of DARTS. For architecture search on CIFAR-100 and CIFAR-10, the hyperparameter K was set to 4. The network is a stack of 8 cells. Each cell contains 6 nodes. Initial channel number is set to 16. The architecture variables are trained using the Adam optimizer for 50 epochs. The learning rate is set to $6e-4$, without decay. The weight decay is set to $1e-3$. The momentum is set to $(0.5, 0.999)$. The network weight parameters are trained using SGD for 50 epochs. The initial learning rate is set to 0.1. Cosine scheduling is used to decay the learning rate, down to 0 without restart. The momentum is set to 0.9. The weight decay is set to $3e-4$. The batch size is set to 256. Warm-up is utilized: in the first 15 epochs, architecture variables are frozen and only network weights are optimized.

The settings for architecture evaluation on CIFAR-100 and CIFAR-10 follow those of DARTS. 18 normal cells and 2 reduction cells are stacked into a large network. The initial channel number is set to 36. The stacked network is trained from scratch using SGD for 600 epochs, with batch size 128, initial learning rate 0.025, momentum 0.9, weight decay $3e-4$, norm gradient clipping 5, drop-path rate 0.3, and cutout. The learning rate is decayed to 0 using cosine scheduling without restart.

We combine our method and PC-DARTS to directly search for architectures on ImageNet. The stacked network starts with three convolution layers which reduce the input image resolution from 224×224 to 28×28, using stride 2. After the three convolution layers, 6 normal cells and 2 reduction cells are stacked. Each cell consists of $N = 6$ nodes. The sub-sampling rate was set to 0.5. The network was trained for 50 epochs. Architecture variables are trained using Adam. The learning rate is fixed to $6e-3$. The weight decay is set to $1e-3$. The momentum is set to $(0.5, 0.999)$. In the first 35 epochs, architecture variables are frozen. Network weight parameters are trained using SGD. The initial learning rate is set to 0.5. It is decayed to 0 using cosine scheduling without restart. Momentum is set to 0.9. Weight decay is set to $3e-5$. The batch-size is set to 1024. Epoch number is set to 250. Eight Tesla V100 GPUs were used.

For architecture evaluation on ImageNet, the stacked network starts with three convolution layers which reduce the input image resolution from 224×224 to 28×28, using stride 2. After the three convolution layers, 12 normal cells and 2 reduction cells are stacked. Initial channel number is set to 48. The network is trained from scratch using SGD for 250 epochs, with batch size 1024, initial learning rate 0.5, weight decay $3e-5$, and momentum 0.9. For the first 5 epochs, learning rate warm-up is used. The learning rate is linearly decayed to 0. Label smoothing and auxiliary loss tower is used.

L.3. P-DARTS based experiments

λ in Eq.(47) is set to 0.1. The search process has three stages. At the first stage, the search space and stacked network in P-DARTS are mostly the same as DARTS. The only difference is the number of cells in the stacked network in P-DARTS is set to 5. At the second stage, the number of cells in the stacked network is 11. At the third stage, the cell number is 17. At stage 1, 2, 3, the initial Dropout probability on skip-connect is 0, 0.4, and 0.7 for CIFAR-10, is 0.1, 0.2, and 0.3 for CIFAR-100; the size of operation space is 8, 5, 3, respectively. The final searched cell is limited to have 2 skip-connect operations at maximum. At each stage, the network is trained using the Adam optimizer for 25 epochs. The batch size is set to 96. The learning rate is set to 6e-4. Weight decay is set to 1e-3. Momentum is set to $(0.5, 0.999)$. In the first 10 epochs,

architecture variables are frozen and only network weights are optimized.

For architecture evaluation on CIFAR-100 and CIFAR-10, the stacked network consists of 20 cells. The initial channel number is set to 36. The network is trained from scratch using SGD. The epoch number is set to 600. The batch size is set to 128. The initial learning rate is set to 0.025. The learning rate is decayed to 0 using cosine scheduling. Weight decay is set to 3e-4 for CIFAR-10 and 5e-4 for CIFAR-100. Momentum is set to 0.9. Drop-path probability is set to 0.3. Cutout regularization length is set to 16. Auxiliary towers of weight 0.4 are used.

For architecture evaluation on ImageNet, the settings are similar to those of DARTS. The network consists of 14 cells. The initial channel number is set to 48. The network is trained from scratch using SGD for 250 epochs. Batch size is set to 1024. Initial learning rate is set to 0.5. The learning rate is linearly decayed after each epoch. In the first 5 epochs, learning rate warmup is used. The momentum is set to 0.9. The weight decay is set to $3e-5$. Label smoothing and auxiliary loss tower are used during training. The network was trained on 8 Nvidia Tesla V100 GPUs.

L.4. PR-DARTS based experiments

λ in Eq.(47) is set to 0.1. The operations include: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 average pooling and 3×3 max pooling, zero, and skip connection. The stacked network consists of k cells. The k/3- and $2k$/3-th cells are reduction cells. In reduction cells, all operations have a stride of two. The rest cells are normal cells. Operations in normal cells have a stride of one. Cells of the same type (either reduction or normal) have the same architecture. The inputs of each cell are the outputs of two previous cells. Each cell contains four intermediate nodes and one output node. The output node is a concatenation of all intermediate nodes.

For architecture search on CIFAR-100 and CIFAR-10, the stacked network consists of 8 cells. The initial channel number is set to 16. In PR-DARTS, λ_1, λ_2, and λ_3 are set to 0.01, 0.005, and 0.005 respectively. The network was trained for 200 epochs. The mini-batch size is set to 128. Architecture variables are trained using Adam. The learning rate is set to $3e-4$. The weight decay is set to $1e-3$. Network weights are trained using SGD. The initial learning rate is set to 0.025. The momentum is set to 0.9. The weight decay is set to $3e-4$. The learning rate is decayed to 0 using cosine scheduling. For acceleration, per iteration, only two operations on each edge are randomly selected to update. The temperature τ is set to 10 and is linearly reduced to 0.1; $a=-0.1$ and $b=1.1$. Pruning on each node is conducted by comparing the gate activation probabilities of all non-zero operations collected from all previous nodes and retaining top two operations.

For architecture evaluation on CIFAR10 and CIFAR100, the stacked network consists of 18 normal cells and 2 reduction cells. The initial channel number is set to 36. The network is trained from scratch using SGD. The mini-batch size is set to 128. The epoch number is set to 600. The initial learning rate is set to 0.025. The momentum is set to 0.9. The weight decay is set to $3e-4$. The gradient norm clipping is set to 5. The drop-path probability is set to 0.2. The cutout length is set to 16. The learning rate is decayed to 0 using cosine scheduling.

For architecture evaluation on ImageNet, the input images are resized to 224×224. The stacked network consists of 3 convolutional layers, 12 normal cells, and 2 reduction cells. The channel number is set to 48. The network is trained using SGD for 250 epochs. The batch size is set to 128. The learning rate is set to 0.025. The momentum is set to 0.9. The weight decay is set to $3e-4$. The gradient norm clipping is set to 5. The learning rate is decayed to 0 via cosine scheduling.

M. Experimental details of evaluating robustness against overfitting

The four search spaces $S1-S4$ are designed by (Zela et al., 2020).

- **S1**: In this search space, each edge has only two candidate operations. To identify these operations, operations that have the least importance in the original search space of DARTS are iteratively removed.
- **S2**: For each edge, the candidate operations are 3×3 SepConv and SkipConnect.
- **S3**: For each edge, the candidate operations are: 3×3 SepConv, SkipConnect, and Zero.
- **S4**: For each edge, the candidate operations are: 3×3 SepConv and Noise. In the Noise operation, every value from the input feature map is replaced with random variables sampled from univariate Gaussian distribution.

N. Significance test results

To check whether the performance of our methods are significantly better than that of baselines, we perform statistical significance tests between the result of our methods and the result of the corresponding baselines, using a double-sided T-test. We use the function in the python package "scipy.stats.ttest_1samp" and report the average results over 10 different runs. Table 27 and 28 show the results.

Our method	Baseline	p-value
FAS-darts2nd	MTL-darts2nd	8.35e-7
FAS-darts2nd	SE-darts2nd	1.05e-10
FAS-darts2nd	Darts2nd	4.37e-12
FAS-pdarts	MTL-pdarts	3.72e-6
FAS-pdarts	SE-pdarts	5.39e-7
FAS-pdarts	Pdarts	8.52e-8
FAS-pcdarts	MTL-pcdarts	5.23e-8
FAS-pcdarts	7.14e-3 SE-pcdarts	8.58e-10
FAS-pcdarts	Pcdarts	6.72e-13
FAS-prdarts	MTL-prdarts	9.73e-4
FAS-prdarts	SE-prdarts	3.06e-5
FAS-prdarts	Prdarts	7.14e-3

Table 27. Significance test results on CIFAR-100

Our method	Baseline	p-value
FAS-darts2nd	MTL-darts2nd	8.31e-6
FAS-darts2nd	SE-darts2nd	1.27e-5
FAS-darts2nd	Darts2nd	6.49e-6
FAS-pdarts	MTL-pdarts	3.05e-7
FAS-pdarts	SE-pdarts	7.35e-5
FAS-pdarts	Pdarts	1.04e-3
FAS-pcdarts	MTL-pcdarts	3.73e-4
FAS-pcdarts	SE-pcdarts	8.62e-7
FAS-pcdarts	Pcdarts	4.28e-3
FAS-prdarts	MTL-prdarts	9.57e-4
FAS-prdarts	SE-prdarts	3.30e-4
FAS-prdarts	Prdarts	1.28e-3

Table 28. Significance test results on CIFAR-10

From these two tables, we can see that the p-values are small between baselines methods and our methods, which demonstrate that the errors of our methods are significantly lower than those of baselines.

O. Model parameters, search costs, and FLOPs on ImageNet

Table 29 shows the number of model parameters, search costs, and FLOPs on ImageNet. The parameter numbers, search costs, and FLOPs of our methods are close to those in differentiable baselines.

P. Full lists of hyperparameter settings in NAS experiments

The hyperparameter settings for the learner model such as the optimizer, momentum, weight decay, learning rate, number of layers were the same as those of the original DARTS, P-DARTS, PC-DARTS, and PR-DARTS implementations. The hyperparameter settings for the listener model were also set to the same values as those of the learner model. Tables 30 and 33 show the hyperparameter settings used in architecture search experiments. Tables 34 and 37 show the hyperparameter settings used in architecture evaluation experiments.

Table 29. Top-1 and top-5 classification errors on ImageNet test set, number of model weights (millions), and search cost (GPU days), and FLOPs (M). * denotes that the results are taken from DARTS$^-$ (Chu et al., 2021) and DrNAS (Chen et al., 2021a). FAS-darts2nd-cifar10 means the architecture is searched using FAS on CIFAR-10, where the search space is the same as that in DARTS-2nd. Similar meanings hold for other notations like this. The other notations are the same as those in Table 1 in the main paper.

Method	Top-1 Error (%)	Top-5 Error (%)	Param (M)	Cost (GPU days)	FLOPs (M)
*Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	-	1448
*ShuffleNet 2× (v2) (Ma et al., 2018)	25.1	7.6	7.4	-	299
*NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	1800	564
*AmoebaNet-C (Real et al., 2019)	24.3	7.6	6.4	3150	570
*SDARTS-ADV-CIFAR10 (Chen & Hsieh, 2020a)	25.2	7.8	5.4	1.3	-
*PC-DARTS-CIFAR10 (Xu et al., 2020)	25.1	7.8	5.3	0.1	586
*ProxylessNAS-ImageNet (Cai et al., 2019)	24.9	7.5	7.1	8.3	465
*FairDARTS-ImageNet (Chu et al., 2020)	24.4	7.4	4.3	3.0	440
*DARTS$^+$-CIFAR100 (Liang et al., 2019)	23.7	7.2	5.1	0.2	591
*Darts2nd-cifar10 (Liu et al., 2019)	26.7	8.7	4.7	1.5	574
FAS-darts2nd-cifar10 (ours)	**25.3**	**7.7**	4.9	2.0	535
*Pdarts-cifar10 (Chen et al., 2019)	24.4	7.4	4.9	0.3	557
FAS-pdarts-cifar10 (ours)	**23.9**	**7.1**	4.9	0.7	529
*Pdarts-cifar100 (Chen et al., 2019)	24.7	7.5	5.1	0.3	577
FAS-pdarts-cifar100 (ours)	**24.3**	**7.3**	4.9	0.7	561
*Pcdarts-imagenet (Xu et al., 2020)	24.2	7.3	5.3	0.1	597
FAS-pcdarts-imagenet (ours)	**23.2**	**6.5**	5.5	0.3	574

Name	Value
Architecture optimizer	Adam
Learning rate of architecture variables	3e-4
Epochs	50
Weight decay of architecture variables	1e-3
Batch size	64
Drop path probability	3e-1
Initial channels	16
Network weights optimizer	SGD
Learning rate of network weights	2.5e-2
Number of layers	8
Momentum	9e-1
Weight decay for network weights	3e-4
Gradient clip	5
Lambda	0.1
Train portion	0.5
Unrolled	True
Cutout	False

Table 30. Hyperparameter settings of our FAS method when applied to DARTS, during architecture search on CIFAR-10 and CIFAR-100.

Name	Value
Architecture optimizer	Adam
Learning rate of architecture variables	6e-4
Epochs	50
Weight decay of architecture variables	1e-3
Batch size	96
Drop path probability	3e-1
Initial channels	16
Network weights optimizer	SGD
Learning rate of network weights	1e-1
Number of layers	8
Momentum	9e-1
Weight decay for network weights	3e-4
Grad clip	5
Lambda	0.5
Train portion	0.5
Unrolled	True
Cutout	False

Table 31. Hyperparameter settings of our FAS method when applied to PC-DARTS, during architecture search on CIFAR-10 and CIFAR-100.

Name	Value
Architecture optimizer	Adam
Learning rate of architecture variables	3e-4
Epochs	30
Weight decay of architecture variables	1e-3
Batch size	32
Drop path probability	3e-1
Initial channels	16
Network weights optimizer	SGD
Learning rate of network weights	2.5e-2
Number of layers	8
Momentum	9e-1
Weight decay for network weights	3e-4
Gradient clip	5
Lambda	0.1
Train portion	0.5
Unrolled	True
Cutout	False

Table 32. Hyperparameter settings of our FAS method when applied to PDARTS, during architecture search on CIFAR-10 and CIFAR-100.

Name	Value
Architecture optimizer	Adam
Learning rate of architecture variables	6e-4
Epochs	50
Weight decay of architecture variables	1e-3
Batch size	96
Drop path probability	3e-1
Initial channels	16
Network weights optimizer	SGD
Learning rate of network weights	1e-1
Number of layers	8
Momentum	9e-1
Weight decay for network weights	3e-4
Grad clip	5
Lambda	0.5
Train portion	0.5
Unrolled	True
Cutout	False

Table 33. Hyperparameter settings of our FAS method when applied to PR-DARTS, during architecture search on CIFAR-10 and CIFAR-100.

Name	Value
Optimizer	SGD
Learning rate	2.5e-2
Epochs	600
Weight decay	3e-4
Batch size	96
Momentum	9e-1
Initial channels	36
Number of layers	20
Auxiliary	True
Auxiliary weight	0.4
Cutout	True
Cutout length	16
Drop path probability	0.2
Gradient clip	5

Table 34. Hyperparameter settings of our FAS method when applied to DARTS, during architecture evaluation on CIFAR-10 and CIFAR-100.

Name	Value
Optimizer	SGD
Learning rate	2.5e-2
Epochs	600
Weight decay	3e-4
Batch size	96
Momentum	9e-1
Initial channels	36
Number of layers	20
Auxiliary	True
Auxiliary weight	0.4
Cutout	True
Cutout length	16
Drop path probability	0.3
Gradient clip	5

Table 35. Hyperparameter settings of our FAS method when applied to PC-DARTS, during architecture evaluation on CIFAR-10 and CIFAR-100.

Name	Value
Optimizer	SGD
Learning rate	2.5e-2
Epochs	600
Weight decay	3e-4
Batch size	96
Momentum	9e-1
Initial channels	36
Number of layers	20
Auxiliary	True
Auxiliary weight	0.4
Cutout	True
Cutout length	16
Drop path probability	0.2
Gradient clip	5

Table 36. Hyperparameter settings of our FAS method when applied to PDARTS, during architecture evaluation on CIFAR-10 and CIFAR-100.

Name	Value
Optimizer	SGD
Learning rate	2.5e-2
Epochs	600
Weight decay	3e-4
Batch size	96
Momentum	9e-1
Initial channels	36
Number of layers	20
Auxiliary	True
Auxiliary weight	0.4
Cutout	True
Cutout length	16
Drop path probability	0.3
Gradient clip	5

Table 37. Hyperparameter settings of our FAS method when applied to PR-DARTS, during architecture evaluation on CIFAR-10 and CIFAR-100.

Q. Computing infrastructure, runtime, validation performance, number of weight parameters, and implementation details

The computing infrastructure and runtime (seconds per iteration) are shown in Table 38 and 39. Validation errors are shown in Table 40.

	FAS(DARTS)	FAS(PC-DARTS,unrolled)	FAS(PCDARTS)
	GTX 1080Ti	GTX 1080Ti	GTX 1080Ti
Num. of GPUs	1	1	1
Runtime	185	83	6

Table 38. Computing infrastructure and runtime (seconds per iteration) for architecture search

	FAS(DARTS)	FAS(PC-DARTS)
	GTX 1080Ti	GTX 1080Ti
Num. of GPUs	1	1
Runtime	167	194

Table 39. Computing infrastructure and runtime (seconds per iteration) for architecture evaluation

	Num. of epochs	Validation error (%)
FAS(DARTS,CIFAR-10)	30	2.69
FAS(DARTS,CIFAR100)	30	17.63
FAS(PC-DARTS,CIFAR-10)	50	2.66
FAS(PC-DARTS,CIFAR-100)	50	17.49

Table 40. Validation error in architecture search.

We use PyTorch to implement all models. All experiments were run on a single GTX 1080Ti GPU. We used the codes provided in "DARTS[7]" and "PCDARTS[8]" as baselines.

[7]https://github.com/quark0/darts
[8]https://github.com/yuhuixu1993/PC-DART

R. Visualization of searched architectures

Using the graphviz package, we plot the searched architecture cells in all the experiments. Each architecture consists of a normal cell and reduction cell. Figure 12-19 show the cells searched on CIFAR-10 and CIFAR-100 by our FAS methods.

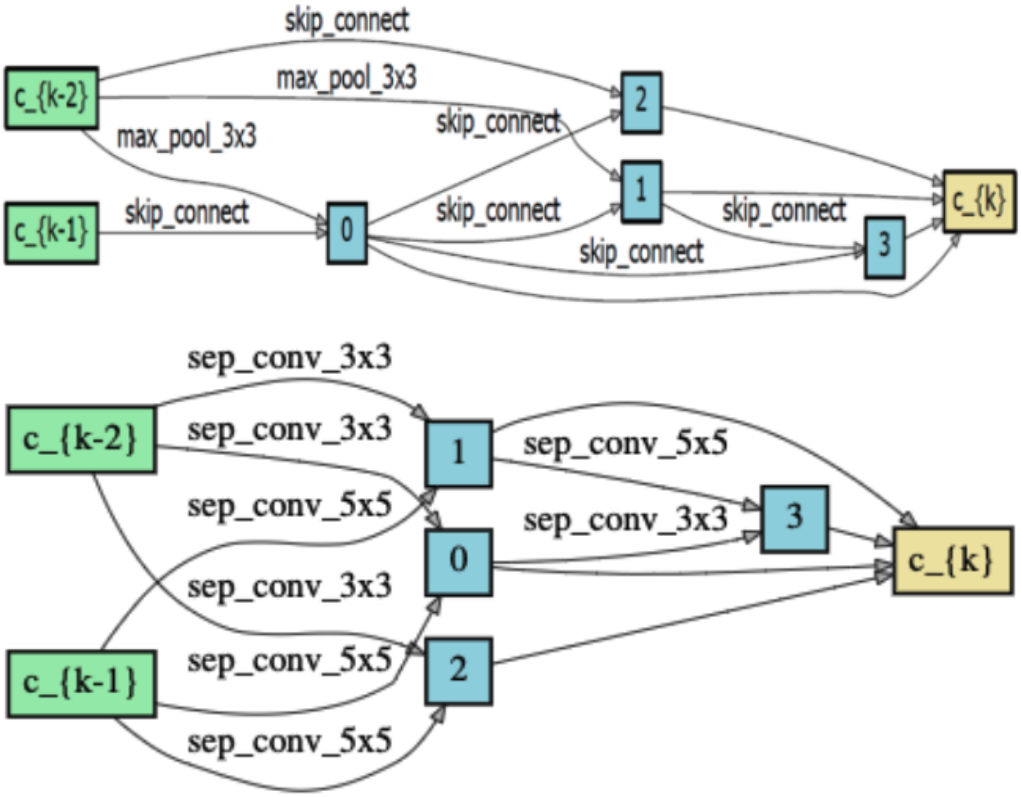

Figure 12. Normal cell searched by FAS(DARTS) on CIFAR-10 and CIFAR-100 respectively.

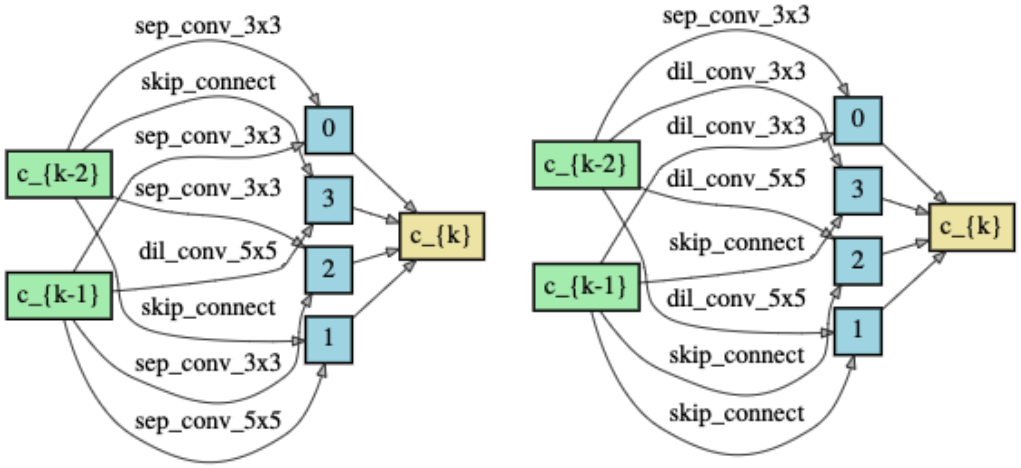

Figure 13. Normal cell searched by FAS(PC-DARTS) on CIFAR-10 and CIFAR-100 respectively

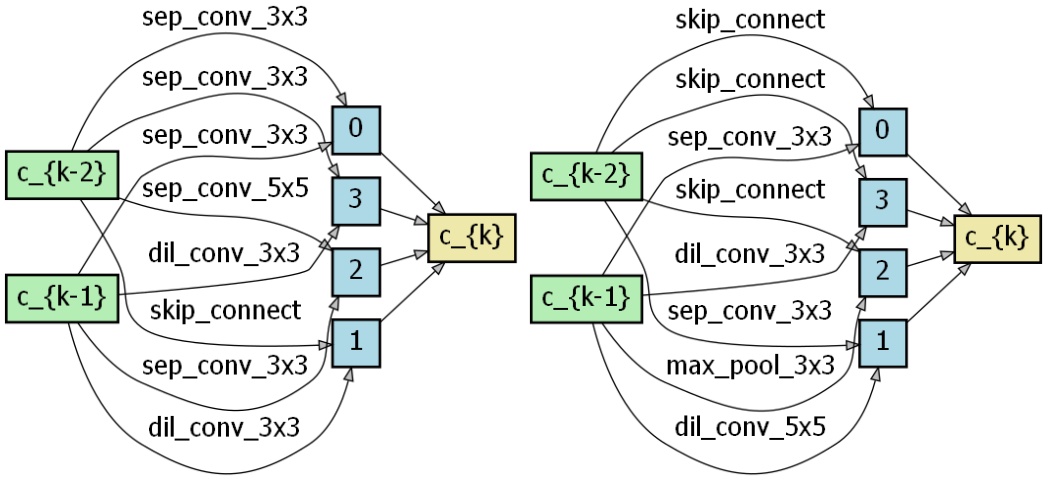

Figure 14. Normal cell searched by FAS(P-DARTS) on CIFAR-10 and CIFAR-100 respectively

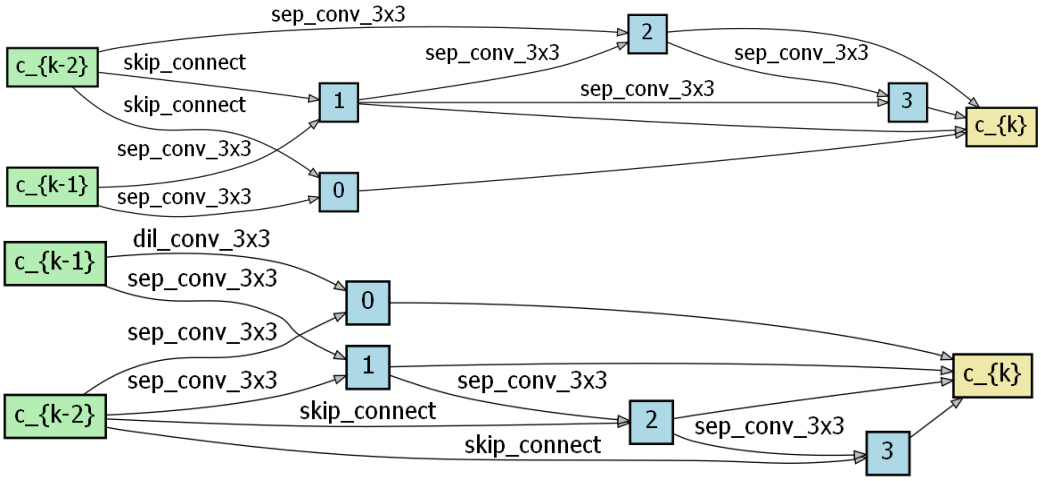

Figure 15. Normal cell searched by FAS(PR-DARTS) on CIFAR-10 and CIFAR-100 respectively

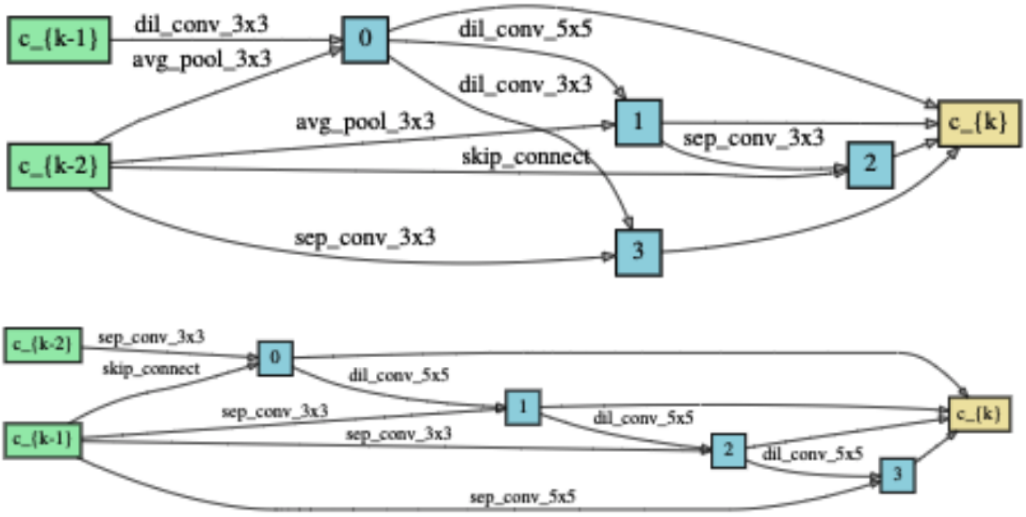

Figure 16. Reduction cell searched by FAS(DARTS) on CIFAR-10 and CIFAR-100 respectively

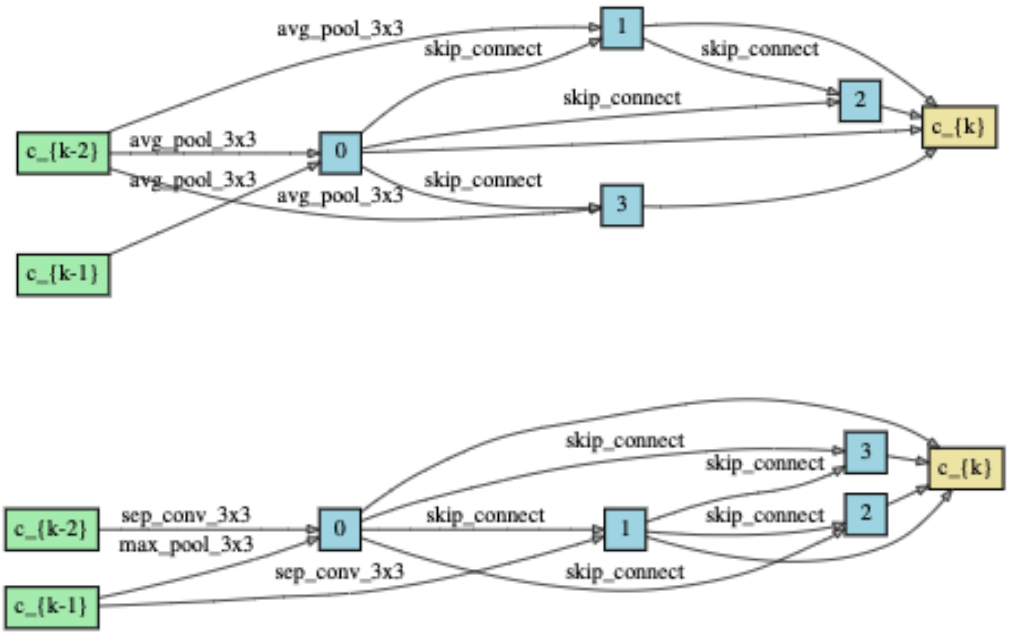

Figure 17. Reduction cell searched by FAS(PC-DARTS) on CIFAR-10 and CIFAR-100 respectively

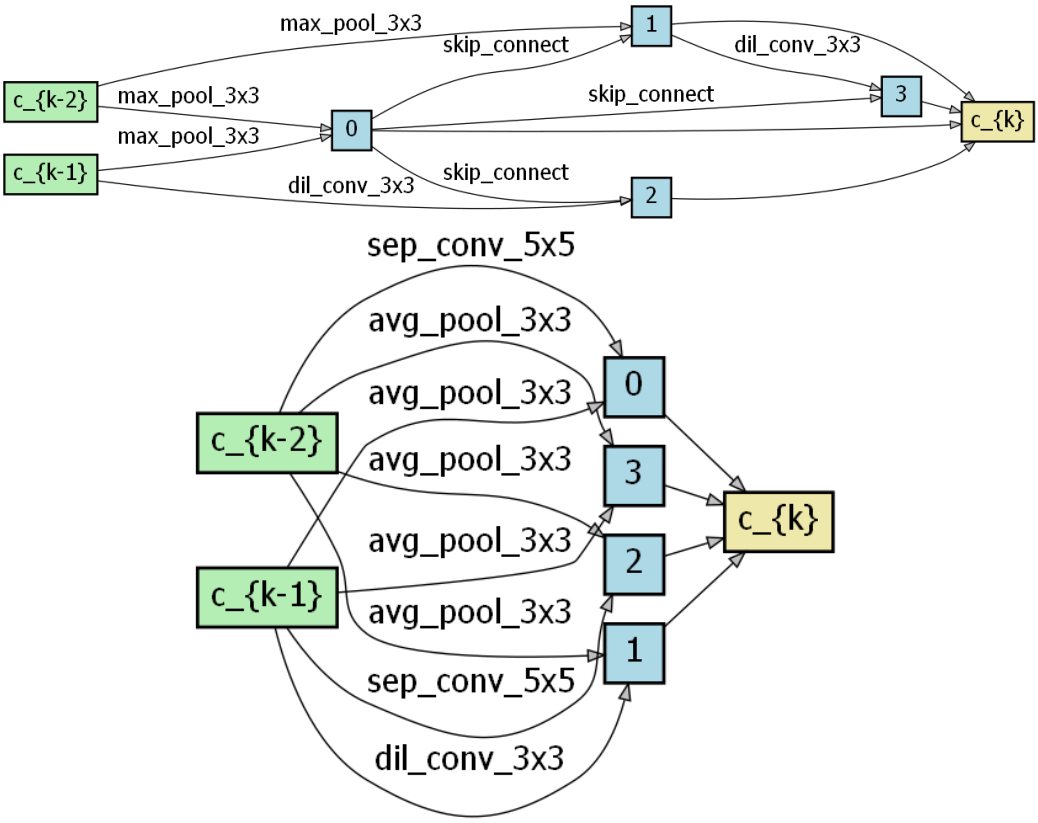

Figure 18. Reduction cell searched by FAS(P-DARTS) on CIFAR-10 and CIFAR-100 respectively

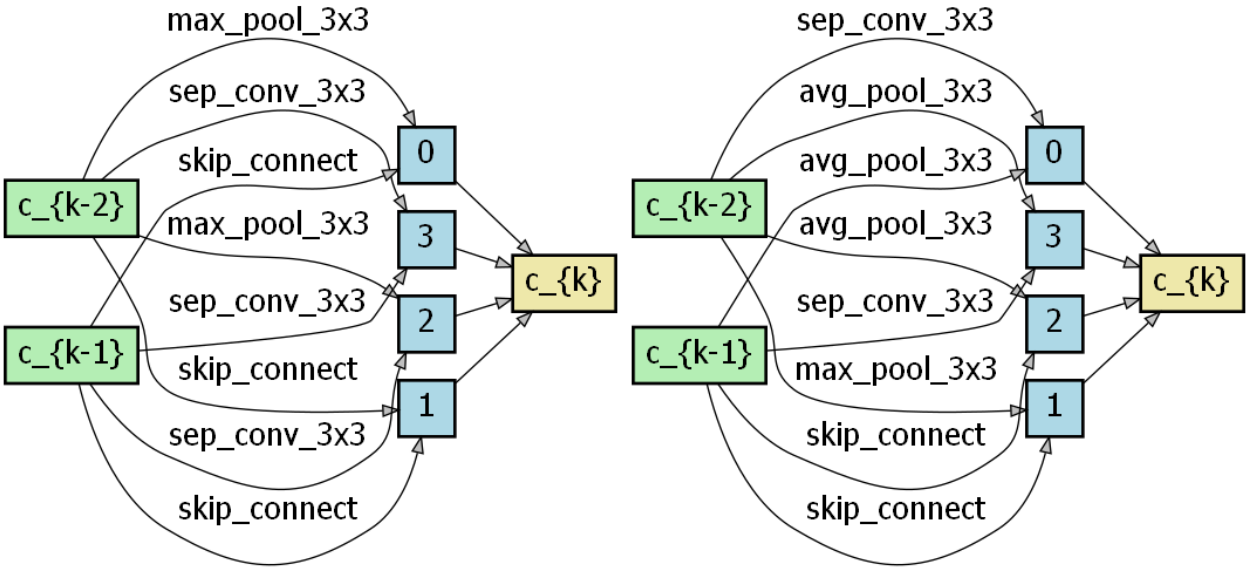

Figure 19. Reduction cell searched by FAS(PR-DARTS) on CIFAR-10 and CIFAR-100 respectively