Skillearn: Machine Learning Inspired by Humans' Learning Skillearn

Pengtao Xie

University of California San Diego

Abstract

Humans, as the most powerful learners on the planet, have accumulated a lot of learning skills, such as learning through tests, interleaving learning, self-explanation, active recalling, to name a few. These learning skills and methodologies enable humans to learn new topics more effectively and efficiently. We are interested in investigating whether humans' learning skills can be borrowed to help machines to learn better. Specifically, we aim to formalize these skills and leverage them to train better machine learning (ML) models. To achieve this goal, we develop a general framework – Skillearn, which provides a principled way to represent humans' learning skills mathematically and use the formally-represented skills to improve the training of ML models. In six case studies, we apply Skillearn to formalize six learning skills of humans: learning by passing tests, interleaving learning, learning by self-explanation, small-group learning, learning by teaching, and learning by ignoring. We use the formalized skills to improve neural architecture search and domain adaptation. Experiments on various datasets show that trained using the skills formalized by Skillearn, ML models achieve significantly better performance.

Contents

Chapter 1

Skillearn: Machine Learning Inspired by Humans' Learning Skills

1.1 Introduction

Given a group of human students, assuming they work equally hard, there are three major factors determining which students learn better than others, including intelligence, learning skills, and learning materials. People with higher intelligence quotient (IQ) are stronger learners. Learning materials, such as textbooks, video lectures, practice questions, etc. are also crucial in determining the quality of learning. Another vital factor impacting learning outcomes is learning skills. Oftentimes, students in the same class have similar IQ and have access to the same learning materials, but their final grades (which measure learning quality) have a large variance. The major differentiating factor is that different students have different levels of mastery of learning skills. Some students have better learning methodologies, which enable them to learn faster and better. In the long history of learning, humans have accumulated a lot of effective learning skills, such as learning through tests, interleaving learning, self-explanation, active recalling, etc.

Similar to human learning, the performance of machine learning (ML) models is also determined by several factors (as compared in Figure 1.1). In the current practice of ML, two dominant factors determining ML performance are the capacity of models and the abundance of data. ML model capacity is analogous to the intelligence of humans. From linear models such as support vector machines to nonlinear models such as deep neural networks, ML researchers have been continuously building more powerful ML models to deal with more complicated tasks. It is like the evolution of humans' brains, which become increasingly intelligent. Data for ML is analogous to learning materials for humans. ML models trained with more labeled data in general perform better.

For intelligence and learning materials in human learning (HL), we identify their counterparts in machine learning as model capacity and data. We are interested in asking: for learning skills in HL, do they have counterparts in ML as well? Can machines be equipped with effective learning skills as humans are? In our work, we aim to address these questions. We propose a general framework – Skillearn, which draws inspiration from humans' learning skills and formulates them into machines' learning skills (MLS). These MLS are leveraged to train better ML models.

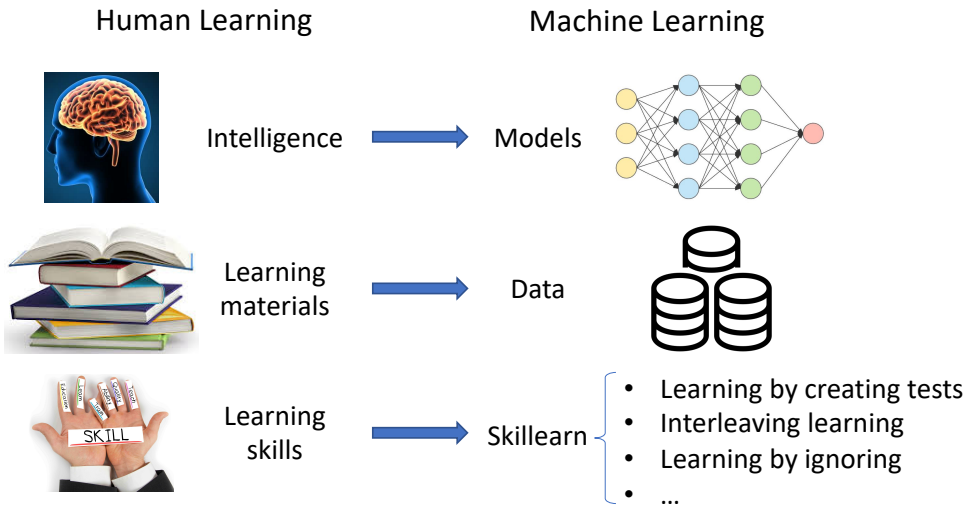

Figure 1.1: Human learning (HL) versus machine learning (ML). Model capacity in ML is analogous to intelligence in HL. Data in ML is analogous to learning materials in HL. Machines' learning skills formulated by our proposed Skillearn framework are analogous to humans' learning skills.

In Skillearn, there are one or multiple learner models, each with one or multiple sets of learnable parameters such as weight parameters, architectures, hyperparameters, etc. 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. The stages have an order, but they are performed end-to-end in a multi-level optimization framework where latter stages influence earlier stages and vice versa. We develop a unified optimization algorithm for solving the multi-level optimization problem in Skillearn.

The major contributions of this work are as follows.

- We propose to leverage broadly-used and effective learning-skills in human learning to develop better machine learning methods.
- We propose Skillearn, a general framework for formulating humans' learning skills into machines' learning skills that can be leveraged by ML models for achieving better learning outcomes.

1.2 Skillearn: Machine Learning Inspired by Human's Learning Skills

In this section, we present a general framework called Skillearn, which draws inspiration from humans' learning skills, formalizes these skills, and leverages them to improve machine learning. We begin with a brief overview of humans' learning skills and summarize their properties. Then we present the Skillearn framework and the optimization algorithm for this framework.

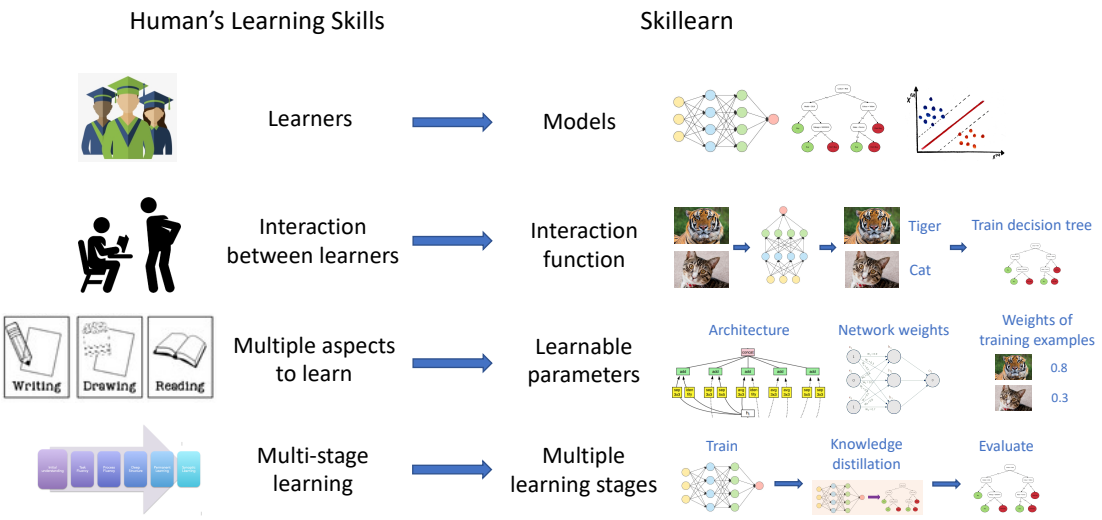

Figure 1.2: The elements of Skillearn and their counterparts in human learning.

1.2.1 Humans' Learning Skills

Humans, as the most powerful learners on the planet, have accumulated a lot of skills and techniques in learning faster and better. Here are some examples.

- **Learning through testing.** After learning a topic, a student can solve some test problems (created or selected by a teacher) about this topic to identify the strong and weak points in his/her understanding of this topic, and re-learns this topic based on the identified strong and weak points. In re-learning, the identified strong and weak points help the student to know what to focus on. The quality of test problems plays a crucial role in effectively evaluating the student. How to create or select high-quality test problems is an important skill that the teacher needs to learn.
- **Interleaving learning** is a learning technique where a learner interleaves the studies of multiple topics: study topic A for a while, then switch to B, subsequently to C; then switch back to A, and so on, forming a pattern of $ABCABCABC\cdots$. Interleaving learning is in contrast to blocked learning, which studies one topic very thoroughly before moving to another topic. Compared with blocked learning, interleaving learning increases long-term retention and improves ability to transfer learned knowledge.
- **Learning by ignoring.** In course learning, given a large collection of practice problems provided in the textbook, the teacher selects a subset of problems as homework for the students to practice instead of using all problems in the textbook. Some practice problems are ignored because 1) they are too difficult which might confuse the students; 2) they are too simple which are not effective in helping the students to practice their knowledge learned during lectures; 3) they are repetitive.

Properties of Humans' Learning Skills

From the above examples of humans' learning skills, we observe the following properties of them.

- A learning event involves multiple learners. For example, in learning through testing, there are two learners: a student and a teacher. The teacher learns how to create test problems and the student learns how to solve these test problems.
- In a learning task, a learner has multiple aspects to learn about this task. For example, in learning by ignoring, to create effective homework problems, the teacher needs to learn: 1) how to solve these problems; 2) which problems are more valuable to use as homework.
- Different learners interact with each other during learning. For example, in learning through testing, the teacher creates test problems and uses them to evaluate the student.
- In a learning task, the learning process is divided into multiple stages. These stages have a certain order. Each stage involves a subset of learners. For example, in learning through testing, there are three stages: 1) the teacher learns a topic; 2) the teacher creates test problems about this topic and uses them to evaluate the student; 3) based on the strong and weak points identified during solving the test problems, the student re-learns this topic. The three stages have a sequential order and cannot be switched. The first stage involves the teacher only; the second stage involves both the teacher and the student; the third stage involves the student only.

1.2.2 General Framework of Skillearn

Based on the properties of humans' learning skills, we propose a framework called Skillearn to formalize the learning skills of humans and incorporate them into machine learning. In Skillearn, we have the following elements (as shown in Figure 1.2).

- **Learners**. There could be one or multiple learners. Each learner is an ML model, such as a deep convolutional network, a deep generative model, a nonparametric kernel density estimator, etc. This is analogous to human learning which involves one or multiple human learners.
- **Learnable parameters**. Each learner has one or more sets of learnable parameters, which could be weight parameters of a network, architecture of a network, importance weights of training examples, hyperparameters, etc. This is analogous to human learning where each human learner learns multiple aspects in a learning task.
- **Interaction function**, which describes how two or more learners interact with others. Some examples of interaction include: 1) in knowledge distillation, given an unlabeled image dataset, model A predicts the pseudo labels of these images; then model B is trained using these images and the pseudo labels generated by model A; 2) 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: the distribution of embeddings extracted by A is encouraged to have small total-variance with the distribution of embeddings extracted by B. This is analogous to human learning where multiple human learners interact with each other.
- **Learning stages**. The learning of all learners is not conducted at one shot simultaneously. The learning is performed at multiple stages with an order. At each stage, a subset of learners

Notation	Meaning
K	Number of learning stages
W_k	Active learnable parameters of all active learners at the k-th learning stage
U_k	Supporting learnable parameters of all active learners at the k-th learning stage
$D_k^{(\text{tr})}$	Training datasets of all active learners at the k-th learning stage
F_k	Auxiliary datasets used at the k-th learning stage
L_k	Training loss at the k-th learning stage
I_k	Interaction function at the k-th learning stage
$D^{(\text{val})}$	Validation datasets of all learners
F	Auxiliary datasets accessible to all learners
L_{val}	Validation loss

Table 1.1: Notations in the Skillearn framework

participate in the learning. For example, in knowledge distillation, there are two stages: 1) a teacher model is trained; 2) the teacher model predicts pseudo labels on an unlabeled dataset and the pseudo-labeled dataset is used to train the student model. The first stage involves a single learner, which is the teacher. The second stage involves two learners: the teacher and the student. This is analogous to human learning where the learning process is divided into multiple stages. Mathematically, we formulate the learning at each stage as an optimization problem. The outcome of one learning stage is passed to another learning stage via the interaction function.

Next, we define the learning stages. Each learning stage performs a focused learning activity which is defined as an optimization problem. The optimization problem involves a training loss and (optionally) an interaction function which describes how the learners involved in this stage interact with each other. A learning stage consists of the following elements:

- **Active learners**. A subset of learners (one or more) are involved at this learning stage. These learners are called active learners.
- **Active learnable parameters**. For each active learner, a sub-collection of its learnable parameter sets are trained at this stage.
- **Supporting learnable parameters**. For each active learner, a sub-collection of its learnable parameter sets are used to define the loss function and interaction function, but they are not updated at this stage.
- **Interaction function**, which depicts the interaction between two or more active learners.

Mathematical Setup

We assume there are a set of learners, each having a training set and optionally a validation set. Meanwhile, all learners share a common collection of auxiliary datasets F, which could be unlabeled datasets used for self-supervised pretraining [38], additional labeled datasets used for validation, and so on. Each learner has one or more sets of learnable parameters, which could be network weights, architectures, hyperparameters, importance weights of training examples,

etc. We assume there are K learning stages. At each of the first $K-1$ stages, a subset of learners are involved in the learning, which are called active learners. For each active learner, a sub-collection of its learnable parameter sets are trained at this stage, which are called active learnable parameters. Let W_k denote the active learnable parameters of all active learners at stage k ($1 \leq k \leq K-1$). Meanwhile, for each active learner, another sub-collection of its learnable parameter sets are used to define the training loss function and interaction function, but they are not updated at this stage. We refer to them as supporting learnable parameters. Let U_k denote the supporting learnable parameters of all active learners, $D_k^{(\text{tr})}$ denote the training datasets of all active learners, and F_k denote the auxiliary datasets used at this stage to define the interaction function. The learning activity at stage k is formulated as an optimization problem where the optimization variables are active learnable parameters and the objective involves a training loss L_k and (optionally) an interaction function I_k depicting the interaction between active learners. In the last learning stage K, the models trained in previous learning stages are evaluated on the validation datasets. Learnable parameters that are not learned in the first $K-1$ stages are updated in stage K by minimizing a validation loss L_{val} and an interaction function I_K. The notations are summarized in Table 1.1.

The Mathematical Framework for Skillearn

The formulation of Skillearn is shown in Eq.(1.1).

$$
\begin{aligned}
&\max_{\{U_i\}_{i=1}^{K-1}} \; L_{val}(\{W_j^*(\{U_i\}_{i=1}^{j})\}_{j=1}^{K-1}, D^{(\text{val})}, F) + \\
&\qquad \gamma_K I_K(\{W_j^*(\{U_i\}_{i=1}^{j})\}_{j=1}^{K-1}, F) \text{ (Learning stage } K) \\
&s.t. \quad \text{Learning stage } K-1\text{: } W_{K-1}^*(\{U_j\}_{j=1}^{K-1}) = \\
&\qquad \min_{W_{K-1}} \; L_{K-1}(W_{K-1}, U_{K-1}, \{W_j^*(\{U_i\}_{i=1}^{j})\}_{j=1}^{K-2}, D_{K-1}^{(\text{tr})}, F_{K-1}) + \\
&\qquad\qquad \gamma_{K-1} I_{K-1}(W_{K-1}, U_{K-1}, \{W_j^*(\{U_i\}_{i=1}^{j})\}_{j=1}^{K-2}, F_{K-1}) \\
&\qquad \vdots \\
&\qquad \text{Learning stage } k\text{: } W_k^*(\{U_j\}_{j=1}^{k}) = \min_{W_k} L_k(W_k, U_k, \{W_j^*(\{U_i\}_{i=1}^{j})\}_{j=1}^{k-1}, D_k^{(\text{tr})}, F_k) + \\
&\qquad\qquad \gamma_k I_k(W_k, U_k, \{W_j^*(\{U_i\}_{i=1}^{j})\}_{j=1}^{k-1}, F_k) \\
&\qquad \vdots \\
&\qquad \text{Learning stage } 1\text{: } W_1^*(U_1) = \min_{W_1} L_1(W_1, U_1, D_1^{(\text{tr})}, F_1) + \gamma_1 I_1(W_1, U_1, F_1).
\end{aligned}
\tag{1.1}
$$

It is a multi-level optimization framework, which involves K optimization problems. On the constraints are $K-1$ optimization problems, each corresponding to a learning stage. The K learning stages are ordered. From bottom to top, the optimization problems correspond to the learning stage $1, 2, \cdots, K$ respectively. In the optimization problem of the learning stage k ($1 \leq k \leq K-1$), the optimization variables are the active learnable parameters W_k of all active learners at this stage. The objective function consists of a training loss $L_k(W_k, U_k, \{W_j^*(\{U_i\}_{i=1}^{j})\}_{j=1}^{k-1}, D_k^{(\text{tr})}, F_k)$ defined on the active learnable parameters W_k, supporting learnable parameters U_k, optimal solutions $\{W_j^*(\{U_i\}_{i=1}^{j})\}_{j=1}^{k-1}$ obtained at previous learning stages, training datasets $D_k^{(\text{tr})}$ of active learners, and auxiliary datasets F_k used in this stage. The other part of the objective function is

the interaction function $I_k(W_k, U_k, \{W_j^*(\{U_i\}_{i=1}^j)\}_{j=1}^{k-1}, F_k)$ which depicts how the active learners interact with each other at this learning stage. γ_k is a tradeoff parameter between the training loss and interaction function. U_k is needed to define the objective, but it is not updated at this stage. After completing the learning at stage k, we obtain the optimal solution $W_k^*(\{U_j\}_{j=1}^k)$. Note that W_k^* is function of $\{U_j\}_{j=1}^k$ since W_k^* is a function of the objective and the objective is a function of $\{U_j\}_{j=1}^k$. $W_k^*(\{U_k\}_{j=1}^k)$ is used to define the objectives at later stages.

At the very top of Eq.(1.1), the optimization problem (outside the constraint block) corresponds to the last learning stage which validates the optimal solutions $\{W_j^*(\{U_i\}_{i=1}^j)\}_{j=1}^{K-1}$ obtained at the first $K-1$ learning stages. The optimization variables are learnable parameters $\{U_i\}_{i=1}^{K-1}$ that have not been learned at the previous $K-1$ stages. The objective function consists of a validation loss and an interaction function. γ_K is a tradeoff parameter. The validation loss $L_{val}(\{W_j^*(\{U_i\}_{i=1}^j)\}_{j=1}^{K-1}, D^{(\text{val})}, F)$ is defined on the validation sets $D^{(\text{val})}$ of all learners, optimal solutions $\{W_j^*(\{U_i\}_{i=1}^j)\}_{j=1}^{K-1}$, and auxiliary datasets F. The interaction function is defined on $\{W_j^*(\{U_i\}_{i=1}^j)\}_{j=1}^{K-1}$ and F.

Remarks:

- Note that for simplicity, we assume the optimization problem at each stage is a minimization problem. The optimization problem can be a more complicated problem such as a min-max problem.
- At a certain stage, a learnable parameter cannot be simultaneously an active parameter and a supporting parameter. For active parameters at stage k, once learned, they cannot be active parameters or supporting parameters at later stages. For supporting parameters at stage k, they can be active parameters or supporting parameters at later stages.
- Supporting parameters must not be learned at previous stages.

1.2.3 Optimization Algorithm for Skillearn

In this section, we develop an algorithm to solve the Skillearn problem in Eq.(1.1), inspired by the algorithm in [58]. For each learning stage k with an optimization problem: $W_k^*(\{U_j\}_{j=1}^k) = \min_{W_k} \; L_k(W_k, U_k, \{W_j^*(\{U_i\}_{i=1}^j)\}_{j=1}^{k-1}, D_k^{(\text{tr})}, F_k) + \gamma_k I_k(W_k,$
$U_k, \{W_j^*(\{U_i\}_{i=1}^j)\}_{j=1}^{k-1}, F_k)$, we approximate the optimal solution $W_k^*(\{U_j\}_{j=1}^k)$ by one-step gradient descent update of the variable W_k:

$$W_k^*(\{U_j\}_{j=1}^k) \approx W_k'(\{U_j\}_{j=1}^k) = \\ W_k - \eta \nabla_{W_k}(L_k(W_k, U_k, \{W_j^*(\{U_i\}_{i=1}^j)\}_{j=1}^{k-1}, D_k^{(\text{tr})}, F_k) + \gamma_k I_k(W_k, U_k, \{W_j^*(\{U_i\}_{i=1}^j)\}_{j=1}^{k-1}, F_k)). \tag{1.2}$$

At learning stages $k+1, \cdots, K$, $W_k^*(\{U_j\}_{j=1}^k)$ may be used to define objective functions. For a stage l where $k < l < K$, if its objective involves $W_k^*(\{U_j\}_{j=1}^k)$, we replace $W_k^*(\{U_j\}_{j=1}^k)$ with $W_k'(\{U_j\}_{j=1}^k)$ and get an approximated objective. When approximating $W_l^*(\{U_j\}_{j=1}^l)$, we use the gradient of the approximated objective:

$$W_l^*(\{U_j\}_{j=1}^l) \approx W_l'(\{U_j\}_{j=1}^l) = \\ W_l - \eta \nabla_{W_l}(L_l(W_l, U_l, \{W_j'(\{U_i\}_{i=1}^j)\}_{j=1}^{l-1}, D_l^{(\text{tr})}, F_l) + \gamma_l I_l(W_l, U_l, \{W_j'(\{U_i\}_{i=1}^j)\}_{j=1}^{l-1}, F_l)). \tag{1.3}$$

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

$$L_{val}(\{W'_j(\{U_i\}_{i=1}^{j})\}_{j=1}^{K-1}, D^{(\text{val})}, F) + \gamma_K I_K(\{W'_j(\{U_i\}_{i=1}^{j})\}_{j=1}^{K-1}, F). \tag{1.4}$$

We update the learnable parameters $\{U_i\}_{i=1}^{K-1}$ that have not been learned at the previous $K-1$ stages by minimizing this approximated objective. The optimization algorithm for Skillearn is summarized in Algorithm 1.

Algorithm 1: Optimization algorithm for Skillearn

while *not converged* **do**

1. For $k = 1 \cdots K-1$, update $W_k^*(\{U_j\}_{j=1}^{k})$ using Eq.(1.3)
2. Update $\{U_i\}_{i=1}^{K-1}$ by minimizing the approximated objective in Eq.(1.4)

end

1.3 Case Studies

In this section, we instantiate Skillearn to several specific cases. In each case study, we leverage Skillearn to formulate a human-learning skill to a machine learning skill and leverage the formulated ML skill to train better ML models.

- **Learning by passing tests**. Learning through tests is a broadly used methodology in human learning and shows great effectiveness in improving learning outcome: a sequence of tests are made with increasing levels of difficulty; the learner takes these tests to identify his/her weak points in learning and continuously addresses these weak points to successfully pass these tests. We are interested in investigating whether this powerful learning technique can be borrowed from humans to improve the learning abilities of machines. We propose a novel learning approach called learning by passing tests (LPT). In our approach, a tester model creates increasingly more-difficult tests to evaluate a learner model. The learner tries to continuously improve its learning ability so that it can successfully pass however difficult tests created by the tester. We propose a multi-level optimization framework to formulate LPT, where the tester learns to create difficult and meaningful tests and the learner learns to pass these tests. We develop an efficient algorithm to solve the LPT problem. Our method is applied for neural architecture search and achieves significant improvement over state-of-the-art baselines on CIFAR-100, CIFAR-10, and ImageNet. Please refer to Chapter 2 for details.
- **Interleaving learning**. Interleaving learning is a human learning technique where a learner interleaves the studies of multiple topics, which increases long-term retention and improves ability to transfer learned knowledge. Inspired by the interleaving learning technique of humans, we explore whether this learning methodology is beneficial for improving the performance of machine learning models as well. We propose a novel machine learning framework referred to as interleaving learning (IL). In our framework, a set of models collaboratively learn a data encoder in an interleaving fashion: the encoder is

trained by model 1 for a while, then passed to model 2 for further training, then model 3, and so on; after trained by all models, the encoder returns back to model 1 and is trained again, then moving to model 2, 3, etc. This process repeats for multiple rounds. Our framework is based on multi-level optimization consisting of multiple inter-connected learning stages. An efficient gradient-based algorithm is developed to solve the multi-level optimization problem. We apply interleaving learning to search neural architectures for image classification on CIFAR-10, CIFAR-100, and ImageNet. The effectiveness of our method is strongly demonstrated by the experimental results. Please refer to Chapter 3 for details.

- **Learning by self-explanation**. Learning by self-explanation is an effective learning technique in human learning, where students explain a learned topic to themselves for deepening their understanding of this topic. It is interesting to investigate whether this explanation-driven learning methodology broadly used by humans is helpful for improving machine learning as well. Based on this inspiration, we propose a novel machine learning method called learning by self-explanation (LeaSE). In our approach, an explainer model improves its learning ability by trying to clearly explain to an audience model regarding how a prediction outcome is made. LeaSE is formulated as a four-level optimization problem involving a sequence of four learning stages which are conducted end-to-end in a unified framework: 1) explainer learns; 2) explainer explains; 3) audience learns; 4) explainer re-learns based on the performance of the audience. We develop an efficient algorithm to solve the LeaSE problem. We apply LeaSE for neural architecture search on CIFAR-100, CIFAR-10, and ImageNet. Experimental results strongly demonstrate the effectiveness of our method. Please refer to Chapter 4 for details.
- **Small-group learning**. In human learning, an effective learning methodology is small-group learning: a small group of students work together towards the same learning objective, where they express their understanding of a topic to their peers, compare their ideas, and help each other to trouble-shoot problems. We aim to investigate whether this human learning method can be borrowed to train better machine learning models, by developing a novel ML framework – small-group learning (SGL). In our framework, a group of learners (ML models) with different model architectures collaboratively help each other to learn by leveraging their complementary advantages. Specifically, each learner uses its intermediately trained model to generate a pseudo-labeled dataset and re-trains its model using pseudo-labeled datasets generated by other learners. SGL is formulated as a multi-level optimization framework consisting of three learning stages: each learner trains a model independently and uses this model to perform pseudo-labeling; each learner trains another model using datasets pseudo-labeled by other learners; learners improve their architectures by minimizing validation losses. An efficient algorithm is developed to solve the multi-level optimization problem. We apply SGL for neural architecture search. Results on CIFAR-100, CIFAR-10, and ImageNet demonstrate the effectiveness of our method. Please refer to Chapter 5 for details.
- Learning by teaching. In human learning, an effective skill in improving learning outcomes is learning by teaching: a learner deepens his/her understanding of a topic by teaching this topic to others. We aim to borrow this teaching-driven learning methodology from humans and leverage it to train more performant machine learning models, by proposing a novel

ML framework referred to as learning by teaching (LBT). In the LBT framework, a teacher model improves itself by teaching a student model to learn well. Specifically, the teacher creates a pseudo-labeled dataset and uses it to train a student model. Based on how the student performs on a validation dataset, the teacher re-learns its model and re-teaches the student until the student achieves great validation performance. Our framework is based on three-level optimization which contains three stages: teacher learns; teacher teaches student; teacher re-learns based on how well the student performs. A simple but efficient algorithm is developed to solve the three-level optimization problem. We apply LBT to search neural architectures on CIFAR-10, CIFAR-100, and ImageNet. The efficacy of our method is demonstrated in various experiments. Please refer to Chapter 6 for details.

- **Learning by ignoring**. Learning by ignoring, which identifies less important things and excludes them from the learning process, is broadly practiced in human learning and has shown ubiquitous effectiveness. There has been psychological studies showing that learning to ignore certain things is a powerful tool for helping people focus. We explore whether this useful human learning methodology can be borrowed to improve machine learning. We propose a novel machine learning framework referred to as learning by ignoring (LBI). Our framework automatically identifies pretraining data examples that have large domain shift from the target distribution by learning an ignoring variable for each example and excludes them from the pretraining process. We formulate LBI as a three-level optimization framework where three learning stages are involved: pretraining by minimizing the losses weighed by ignoring variables; finetuning; updating the ignoring variables by minimizing the validation loss. An gradient-based algorithm is developed to efficiently solve the three-level optimization problem in LBI. Experiments on various datasets demonstrate the effectiveness of our framework. Please refer to Chapter 7 for details.

1.4 Related Works

Developing ML methods by drawing inspirations from humans has been studied for long. For example, neural networks are inspired by human brain neurons. Curiosity-driven learning [9, 75] is inspired by humans' psychology on curiosity. Few-shot concept learning [50] draws inspirations from cognitive science on how humans perform concept learning. In our work, we focus on drawing inspirations from humans' learning skills and methodologies, which have not been well-explored before, though there exist a few previous works touching the surface of this line of research. For example, curriculum learning [4, 46, 49, 64] is inspired by how human students learn a curriculum in classrooms. Human students learn easy courses first to lay a solid foundation and then move to more difficult courses. Based on this easy-to-difficult learning style, curriculum machine learning is proposed where easy ML tasks are learned first before performing more challenging tasks. As another example, meta learning [28**?**] aims to learn a shared model that can be adapted to different tasks. Our work aims to provide a unified framework to systematically formalize a variety of human-learning skills into machine learning skills and leverage them to train better models.

Chapter 2

Learning by Passing Tests

2.1 Introduction

In human learning, an effective and widely used methodology for improving learning outcome is to let the learner take increasingly more-difficult tests. To successfully pass a more challenging test, the learner needs to gain better learning ability. By progressively passing tests that have increasing levels of difficulty, the learner strengthens his/her learning capability gradually.

Inspired by this test-driven learning technique of humans, we are interested in investigating whether this methodology is helpful for improving machine learning as well. We propose a novel machine learning framework called learning by passing tests (LPT). In this framework, there is a "learner" model and a "tester" model. The tester creates a sequence of "tests" with growing levels of difficulty. The learner tries to learn better so that it can pass these increasingly more-challenging tests. Given a large collection of data examples called "test bank", the tester creates a test T by selecting a subset of examples from the test bank. The learner applies its intermediately-trained model M to make predictions on the examples in T. The prediction error rate R reflects how difficult this test is. If the learner can make correct predictions on T, it means that T is not difficult enough. In this case, the tester will create a more challenging test T' by selecting a new set of examples from the test bank such that the new error rate R' achieved by M on T' is larger than R achieved on T. Given this more demanding test T', the learner re-learns its model to pass T', in a way that the newly-learned model M' achieves a new error rate R'' on T' where R'' is smaller than R'. This process (as illustrated in Figure 7.1) iterates until convergence.

In our framework, both the learner and tester perform learning. The learner learns how to best conduct a target task J_1 and the tester learns how to create difficult and meaningful tests. To encourage a created test T to be meaningful, the tester trains a model using T to perform a target task J_2. If the model performs well on J_2, it indicates that T is meaningful. The learner has two sets of learnable parameters: neural architecture and network weights. The tester has three learnable modules: data encoder, test creator, and target-task executor. Learning is organized into three stages. In the first stage, the learner trains its network weights on the training set of task J_1 with the architecture fixed. In the second stage, the tester trains its data encoder and target-task executor on a created test to perform the target task J_2, with the test creator fixed. In the third stage, the learner updates its model architecture by minimizing the predictive loss L on the test

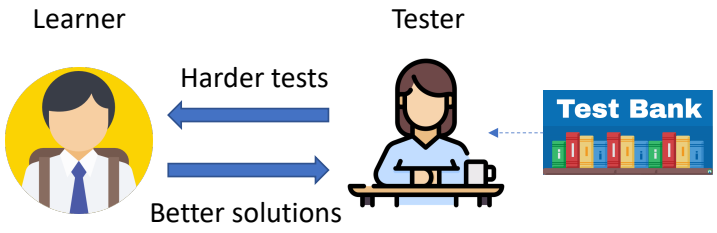

Figure 2.1: Learning by passing tests. A tester model creates tests with increasing levels of difficulty from a test bank to evaluate a learner model. The learner continuously improves its learning ability to deliver better solutions for passing those difficult tests.

created by the tester; the tester updates its test creator by maximizing L and minimizing the loss on the validation set of J_2. The three stages are performed jointly end-to-end in a multi-level optimization framework, where different stages influence each other. We apply our method for neural architecture search [58, 78, 105] in image classification tasks on CIFAR-100, CIFAR-10, and ImageNet [22]. Our method achieves significant improvement over state-of-the-art baselines.

The major contributions of this section are as follows:

- Inspired by the test-driven learning technique of humans, we propose a novel ML approach called learning by passing tests (LPT). In our approach, a tester model creates increasingly more-difficult tests to evaluate a learner model. The learner tries to continuously improve its learning ability so that it can successfully pass however difficult tests created by the tester.
- We propose a multi-level optimization framework to formulate LPT where a learner learns to pass tests and a tester learns to create difficult and meaningful tests.
- We develop an efficient algorithm to solve LPT.
- We apply our approach to neural architecture search and achieve significant improvement on CIFAR-100, CIFAR-10, and ImageNet.

2.2 Related Works

Neural Architecture Search (NAS). NAS has achieved remarkable progress recently, which aims at searching for optimal architectures of neural networks to achieve the best predictive performance. In general, there are three paradigms of methods in NAS: reinforcement learning based approaches [76, 105, 106], evolutionary algorithm based approaches [57, 78], and differentiable approaches [10, 58, 95]. In RL-based approaches, a policy is learned to iteratively generate new architectures by maximizing a reward which is the accuracy on the validation set. Evolutionary learning approaches represent the architectures as individuals in a population. Individuals with high fitness scores (validation accuracy) have the privilege to generate offspring, which replaces individuals with low fitness scores. Differentiable approaches adopt a network pruning strategy. On top of an over-parameterized network, the weights of connections between nodes are learned using gradient descent. Then weights close to zero are pruned later on. There have been many efforts devoted to improving differentiable NAS methods. In P-DARTS [16], the depth of

searched architectures is allowed to grow progressively during the training process. Search space approximation and regularization approaches are developed to reduce computational overheads and improve search stability. PC-DARTS [96] reduces the redundancy in exploring the search space by sampling a small portion of a super network. Operation search is performed in a subset of channels with the held-out part bypassed in a shortcut. Our proposed LPT framework is orthogonal to existing NAS approaches and can be applied to any differentiable NAS methods.

Adversarial Learning. Our formulation involves a min-max optimization problem, which is analogous to that in adversarial learning. Adversarial learning [33] has been widely applied to 1) data generation [33, 98] where a discriminator tries to distinguish between generated images and real images and a generator is trained to generate realistic data by making such a discrimination difficult to achieve; 2) domain adaptation [31] where a discriminator tries to differentiate between source images and target images while the feature learner learns representations which make such a discrimination unachievable; 3) adversarial attack and defence [34] where an attacker adds small perturbations to the input data to alter the prediction outcome and the defender trains the model in a way that the prediction outcome remains the same given perturbed inputs. Different from these existing works, in our work, a tester aims to create harder tests to "fail" the learner while the learner learns to "pass" however hard tests created by the tester. Shu et al. [84] proposed to use an adversarial examiner to identify the weakness of a trained model. Our work differs from this work in that we progressively re-train a learner model based on how it performs on the tests that are created dynamically by a tester model while the learner model in [84] is fixed and not affected by the examination results. Such et al. [87] proposed to learn a generative adversarial network [33] to create synthetic examples which are used to train an NAS model. Our work differs from this work in that we use selected validation examples to validate the model while Such et al. [87] use synthesized example to train the model.

Curriculum Learning. Our work is also related to curriculum learning (CL) [4, 46, 49, 64]. In CL, a sequence of training datasets with increasing levels of difficulty is used for model training, from easy to difficult. Our work differs from these previous works in that: our work dynamically selects more-difficult data examples for model evaluation while previous works select data examples for model training.

2.3 Methods

In this section, we propose a framework to perform learning by passing tests (LPT) (as shown in Figure 7.2) and develop an optimization algorithm for solving the LPT problem.

2.3.1 Learning by Passing Tests

In our framework, there is a learner model and a tester model, where the learner studies how to perform a target task J_1 such as classification, regression, etc. The eventual goal is to make the learner achieve a better learning outcome with help from the tester. There is a collection of data examples called "test bank". The tester creates a test by selecting a subset of examples from

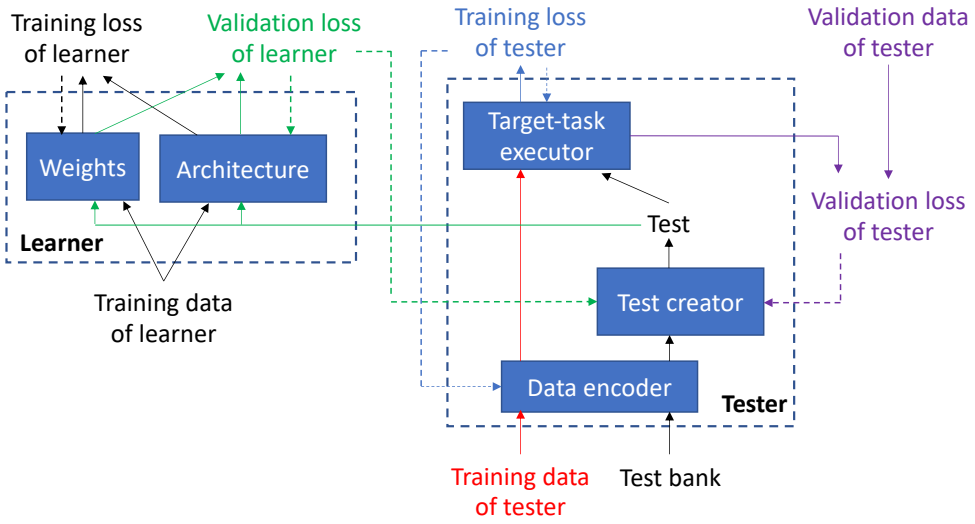

Figure 2.2: Learning by passing tests. The solid arrows denote the process of making predictions and calculating losses. The dotted arrows denote the process of updating learnable parameters by minimizing corresponding losses.

the test bank. Given a test T, the learner applies its intermediately-trained model M to make predictions on T and measures the prediction error rate R. From the perspective of the tester, R indicates how difficult the test T is. If R is small, it means that the learner can easily pass this test. Under such circumstances, the tester will create a more difficult test T' which renders the new error rate R' achieved by M on T' is larger than R. From the learner's perspective, R' indicates how well the learner performs on the test. Given this more difficult test T', the learner refines its model to pass this new test. It aims to learn a new model M' such that the newer error rate R'' achieved by M' on T' is smaller than R'. This process iterates until an equilibrium is reached. In addition to being difficult, the created test should be meaningful as well. It is possible that the test bank contains poor-quality examples where the class labels may be incorrect or the input data instances are outliers. Using an unmeaningful test containing poor-quality examples to guide the learning of the learner may render the learner to overfit these bad-quality examples and generalize poorly on unseen data. To address this problem, we encourage the tester to generate meaningful tests by leveraging the generated tests to perform a target task J_2. Specifically, the tester uses examples in the test to train a model for performing J_2. If the performance (e.g., accuracy) P achieved by this model in conducting J_2 is high, the test is considered to be meaningful. The tester aims to create a test that can yield a high P.

In our framework, both the learner and the tester performs learning. The learner studies how to best fulfill the target task J_1. The tester studies how to create tests that are difficult and meaningful. In the learner' model, there are two sets of learnable parameters: model architecture and network weights. The architecture and weights are both used to make predictions in J_1. The tester's model performs two tasks simultaneously: creating tests and performing another target-task J_2. The model has three learnable modules: data encoder, test creator, and target-task executor, where the test creator performs the task of generating tests and the target-task executor

Notation	Meaning
A	Architecture of the learner
W	Network weights of the learner
E	Data encoder of the tester
C	Test creator of the tester
X	Target-task executor of the tester
$D_{ln}^{(\text{tr})}$	Training data of the learner
$D_{tt}^{(\text{tr})}$	Training data of the tester
$D_{tt}^{(\text{val})}$	Validation data of the tester
D_b	Test bank

Table 2.1: Notations in Learning by Passing Tests

conducts J_2. The test creator and target-task executor share the same data encoder. The data encoder takes a data example d as input and generates a latent representation for this example. Then the representation is fed into the test creator which determines whether d should be selected into the test. The representation is also fed into the target-task executor which performs prediction on d during performing the target task J_2.

In our framework, the learning of the learner and the tester is organized into three stages. In the first stage, the learner learns its network weights W by minimizing the training loss $L(A, W, D_{ln}^{(\text{tr})})$ defined on the training data $D_{ln}^{(\text{tr})}$ in the task J_1. The architecture A is used to define the training loss, but it is not learned at this stage. If A is learned by minimizing this training loss, a trivial solution will be yielded where A is very large and complex that it can perfectly overfit the training data but will generalize poorly on unseen data. Let $W^*(A)$ denotes the optimally learned W at this stage. Note that W^* is a function of A because W^* is a function of the training loss and the training loss is a function of A. In the second stage, the tester learns its data encoder E and target-task executor X by minimizing the training loss $L(E, X, D_{tt}^{(\text{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))$ in the task J_2. The training loss consists of two parts. The first part $L(E, X, D_{tt}^{(\text{tr})})$ is defined on the training dataset $D_{tt}^{(\text{tr})}$ in J_2. The second part $L(E, X, \sigma(C, E, D_b))$ is defined on the test $\sigma(C, E, D_b)$ created by the test creator. To create a test, for each example d in the test bank D_b, it is first fed into the encoder E, then into the test creator C, which outputs a binary value indicating whether d should be selected into the test. $\sigma(C, E, D_b)$ is the collection of examples whose binary value is equal to 1. γ is a tradeoff parameter between these two parts of losses. The creator C is used to define the second-part loss, but it is not learned at this stage. Otherwise, a trivial solution will be yielded where C always sets the binary value to 0 for each test-bank example so that the second-part loss becomes 0. Let $E^*(C)$ and $X^*(C)$ denote the optimally trained E and X at this stage. Note that they are both functions of C since they are functions of the training loss and the training loss is a function of C. In the third stage, the learner learns its architecture by trying to pass the test $\sigma(C, E^*(C), D_b)$ created by the tester. Specifically, the learner aims to minimize its predictive loss on the test:

$$L(A, W^*(A), \sigma(C, E^*(C), D_b)) = \sum_{d \in \sigma(C, E^*(C), D_b)} \ell(A, W^*(A), d), \tag{2.1}$$

where d is an example in the test and $\ell(A, W^*(A), d)$ is the loss defined in this example. A smaller $L(A, W^*(A), \sigma(C, E^*(C), D_b))$ indicates that the learner performs well on this test.

Meanwhile, the tester learns its test creator C in a way that C can create a test with more difficulty and meaningfulness. Difficulty is measured by the learner's predictive loss $L(A, W^*(A), \sigma(C, E^*(C), D_b))$ on the test. Given a model $(A, W^*(A))$ of the learner and two tests of the same size (same number of examples): $\sigma(C_1, E^*(C_1), D_b)$ created by C_1 and $\sigma(C_2, E^*(C_2), D_b)$ created by C_2, if $L(A, W^*(A), \sigma(C_1, E^*(C_1), D_b)) > L(A, W^*(A), \sigma(C_2, E^*(C_2), D_b))$, it means that $\sigma(C_1, E^*(C_1), D_b)$ is more challenging to pass than $\sigma(C_2, E^*(C_2), D_b)$. Therefore, the tester can learn to create a more challenging test by maximizing $L(A, W^*(A), \sigma(C, E^*(C), D_b))$. A trivial solution of increasing $L(A, W^*(A), \sigma(C, E^*(C), D_b))$ is to enlarge the size of the test. But a larger size does not imply more difficulty. To discourage this degenerated solution from happening, we normalize the loss using the size of the test:

$$\frac{1}{|\sigma\left(C, E^*(C), D_b\right)|} L\left(A, W^*\left(A\right), \sigma\left(C, E^*(C), D_b\right)\right), \tag{2.2}$$

where $|\sigma(C, E^*(C), D_b)|$ is the cardinality of the set $\sigma(C, E^*(C), D_b)$. To measure the meaningfulness of a test, we check how well the optimally-trained task executor $X^*(C)$ and data encoder $E^*(C)$ of the tester perform on the validation data $D_{tt}^{(\text{val})}$ of the target task J_2, and the performance is measured by the validation loss: $L(E^*(C), X^*(C), D_{tt}^{(\text{val})})$. $E^*(C)$ and $X^*(C)$ are trained using the test generated by C in the second stage. If the validation loss is small, it means that the created test is helpful in training the task executor and therefore is considered as being meaningful. To create a meaningful test, the tester learns C by minimizing $L(E^*(C), X^*(C), D_{tt}^{(\text{val})})$. In sum, C is learned by maximizing $L(A, W^*(A), \sigma(C, E^*(C), D_b))/|\sigma(C, E^*(C), D_b)| - \lambda L(E^*(C), X^*(C), D_{tt}^{(\text{val})})$, where λ is a tradeoff parameter between these two objectives.

The three stages are mutually dependent: $W^*(A)$ learned in the first stage and $E^*(C)$ and $X^*(C)$ learned in the second stage are used to define the objective function in the third stage; the updated C and A in the third stage in turn change the objective functions in the first and second stage, which subsequently render $W^*(A)$, $E^*(C)$, and $X^*(C)$ to be changed. Putting these pieces together, we formulate LPT as the following multi-level optimization problem.

$$\begin{aligned}
&\max_C \min_A \quad \tfrac{1}{|\sigma(C, E^*(C), D_b)|} L\left(A, W^*\left(A\right), \sigma\left(C, E^*(C), D_b\right)\right) - \lambda L\left(E^*(C), X^*(C), D_{tt}^{(\text{val})}\right) \text{ (III)} \\
&s.t. \quad E^*(C), X^*(C) = \min_{E,X} \quad L\left(E, X, D_{tt}^{(\text{tr})}\right) + \gamma L\left(E, X, \sigma\left(C, E, D_b\right)\right) \text{ (Stage II)} \\
&\qquad W^*\left(A\right) = \min_W \quad L\left(A, W, D_{ln}^{(\text{tr})}\right) \text{ (Stage I)}
\end{aligned} \tag{2.3}$$

This formulation nests three optimization problems. On the constraints of the outer optimization problem are two inner optimization problems corresponding to the first and second learning stage. The objective function of the outer optimization problem corresponds to the third learning stage.

As of now, the test $\sigma(C, E, D_b)$ is represented as a subset, which is highly discrete and therefore difficult for optimization. To address this problem, we perform a continuous relaxation of $\sigma(C, E, D_b)$:

$$\sigma(C, E, D_b) = \{(d, f(d, C, E)) | d \in D_b\}, \tag{2.4}$$

where for each example d in the test bank, the original binary value indicating whether d should be selected is now relaxed to a continuous probability $f(d, C, E)$ representing how likely d should be selected. Under this relaxation, $L(E, X, \sigma(C, E, D_b))$ can be computed as follows:

$$L(E, X, \sigma(C, E, D_b)) = \sum_{d \in D_b} f(d, C, E)\ell(E, X, d), \tag{2.5}$$

where we calculate the loss $\ell(E, X, d)$ on each test-bank example and weigh this loss using $f(d, C, E)$. If $f(d, C, E)$ is small, it means that d is less likely to be selected into the test and its corresponding loss should be down-weighted. Similarly, $L(A, W^*(A), \sigma(C, E^*(C), D_b))$ is calculated as $\sum_{d \in D_b} f(d, C, E^*(C))\ell(A, W^*(A), d)$. And $|\sigma(C, E^*(C), D_b)|$ can be calculated as

$$|\sigma(C, E^*(C), D_b)| = \sum_{d \in D_b} f(d, C, E^*(C)). \tag{2.6}$$

Similar to [58], we represent the architecture A of the learner in a differentiable way. The search space of A is composed of a large number of building blocks. The output of each block is associated with a variable a indicating how important this block is. After learning, blocks whose a is among the largest are retained to form the final architecture. In this end, architecture search amounts to optimizing the set of architecture variables $A = \{a\}$.

2.3.2 Optimization Algorithm

In this section, we derive an optimization algorithm to solve the LPT problem. Inspired by [58], we approximate $E^*(C)$ and $X^*(C)$ using one-step gradient descent update of E and X with respect to $L(E, X, D_{tt}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))$ and approximate $W^*(A)$ using one-step gradient descent update of W with respect to $L(A, W, D_{ln}^{(\mathrm{tr})})$. Then we plug these approximations into

$$L(A, W^*(A), \sigma(C, E^*(C), D_b))/|\sigma(C, E^*(C), D_b)| - \lambda L(E^*(C), X^*(C), D_{tt}^{(\mathrm{val})}), \tag{2.7}$$

and perform gradient-descent update of C and A with respect to this approximated objective. In the sequel, we use $\nabla^2_{Y,X} f(X, Y)$ to denote $\frac{\partial f(X,Y)}{\partial X \partial Y}$.

Approximating $W^*(A)$ using $W' = W - \xi_{ln} \nabla_W L(A, W, D_{ln}^{(\mathrm{tr})})$ where ξ_{ln} is a learning rate and simplifying the notation of $\sigma(C, E^*(C), D_b)$ as σ, we can calculate the approximated gradient of $L\left(A, W^*\left(A\right), \sigma\right)$ w.r.t A as:

$$\begin{aligned}
&\nabla_A L\left(A, W^*\left(A\right), \sigma\right) \approx \\
&\nabla_A L\left(A, W - \xi_{ln} \nabla_W L\left(A, W, D_{ln}^{(\mathrm{tr})}\right), \sigma\right) = \\
&\nabla_A L\left(A, W', \sigma\right) - \xi_{ln} \nabla^2_{A,W} L\left(A, W, D_{ln}^{(\mathrm{tr})}\right) \nabla_{W'} L\left(A, W', \sigma\right).
\end{aligned} \tag{2.8}$$

The second term in the third line involves expensive matrix-vector product, whose computational complexity can be reduced by a finite difference approximation:

$$\nabla^2_{A,W} L\left(A, W, D_{ln}^{(\mathrm{tr})}\right) \nabla_{W'} L\left(A, W', \sigma\right) \approx \frac{1}{2\alpha_{ln}} \left(\nabla_A L\left(A, W^+, D_{ln}^{(\mathrm{tr})}\right) - \nabla_A L\left(A, W^-, D_{ln}^{(\mathrm{tr})}\right)\right), \tag{2.9}$$

where $W^{\pm} = W \pm \alpha_{ln} \nabla_{W'} L\left(A, W', \sigma\right)$ and α_{ln} is a small scalar that equals $0.01/\left\|\nabla_{W'} L\left(A, W', \sigma\right)\right\|_2$. We approximate $E^*(C)$ and $X^*(C)$ using the following one-step gradient descent update of E and C respectively:

$$\begin{aligned} E' &= E - \xi_E \nabla_E [L(E, X, D_{tt}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))] \\ X' &= X - \xi_X \nabla_X [L(E, X, D_{tt}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))] \end{aligned} \tag{2.10}$$

where ξ_E and ξ_X are learning rates. Plugging these approximations into the objective function in Eq.(2.7), we can learn C by maximizing the following objective using gradient methods:

$$L(A, W', \sigma(C, E', D_b))/|\sigma(C, E', D_b)| - \lambda L(E', X', D_{tt}^{(\mathrm{val})}) \tag{2.11}$$

The derivative of the second term in this objective with respect to C can be calculated as:

$$\nabla_C L(E', X', D_{tt}^{(\mathrm{val})}) = \frac{\partial E'}{\partial C} \nabla_{E'} L(E', X', D_{tt}^{(\mathrm{val})}) + \frac{\partial X'}{\partial C} \nabla_{X'} L(E', X', D_{tt}^{(\mathrm{val})}) \tag{2.12}$$

where

$$\begin{aligned} \frac{\partial E'}{\partial C} &= -\xi_E \gamma \nabla_{C,E}^2 L(E, X, \sigma(C, E, D_b)) \\ \frac{\partial X'}{\partial C} &= -\xi_X \gamma \nabla_{C,X}^2 L(E, X, \sigma(C, E, D_b)) \end{aligned} \tag{2.13}$$

Similar to Eq.(6.8), using finite difference approximation to calculate $\nabla_{C,E}^2 L(E, X, \sigma(C, E, D_b))$ $\nabla_{E'} L(E', X', D_{tt}^{(\mathrm{val})})$ and $\nabla_{C,X}^2 L(E, X, \sigma(C, E, D_b)) \nabla_{X'} L(E', X', D_{tt}^{(\mathrm{val})})$, we have:

$$\begin{aligned} &\nabla_C L(E', X', D_{tt}^{(\mathrm{val})}) = \\ &-\gamma \xi_E \frac{\nabla_C L(E^+, X, \sigma(C, E^+, D_b)) - \nabla_C L(E^-, X, \sigma(C, E^-, D_b))}{2\alpha_E} - \gamma \xi_X \frac{\nabla_C L(E, X^+, \sigma(C, E, D_b)) - \nabla_C L(E, X^-, \sigma(C, E, D_b))}{2\alpha_X} \end{aligned} \tag{2.14}$$

where $E^{\pm} = E \pm \alpha_E \nabla_{E'} L(E', X', D_{tt}^{(\mathrm{val})})$ and $X^{\pm} = X \pm \alpha_X \nabla_{X'} L(E', X', D_{tt}^{(\mathrm{val})})$. For the first term $L(A, W', \sigma(C, E', D_b))/|\sigma(C, E', D_b)|$ in the objective, we can use chain rule to calculate its derivative w.r.t C, which involves calculating the derivative of $L(A, W', \sigma(C, E', D_b))$ and $|\sigma(C, E', D_b)|$ w.r.t to C. The derivative of $L(A, W', \sigma(C, E', D_b))$ w.r.t C can be calculated as:

$$\nabla_C L(A, W', \sigma(C, E', D_b)) = \frac{\partial E'}{\partial C} \nabla_{E'} L(A, W', \sigma(C, E', D_b)), \tag{2.15}$$

where $\frac{\partial E'}{\partial C}$ is given in Eq.(2.13) and $\nabla_{C,E}^2 L(E, X, \sigma(C, E, D_b)) \times \nabla_{E'} L(A, W', \sigma(C, E', D_b))$ can be approximated with $\frac{1}{2\alpha_E}(\nabla_C L(E^+, X, \sigma(C, E^+, D_b)) - \nabla_C L(E^-, X, \sigma(C, E^-, D_b)))$, where $E^{\pm}$ is $E \pm \alpha_E \nabla_{E'} L(A, W', \sigma(C, E', D_b))$. The derivative of $|\sigma(C, E', D_b)| = \sum_{d \in D_b} f(d, C, E')$ w.r.t C can be calculated as

$$\sum_{d \in D_b} \nabla_C f(d, C, E') + \frac{\partial E'}{\partial C} \nabla_{E'} f(d, C, E') \tag{2.16}$$

where $\frac{\partial E'}{\partial C}$ is given in Eq.(2.13).

The algorithm for solving LPT is summarized in Algorithm 7.

Algorithm 2: Optimization algorithm for learning by passing tests

while *not converged* **do**

1. Update the architecture of the learner by descending the gradient calculated in Eq.(6.7)
2. Update the test creator of the tester by ascending the gradient calculated in Eq.(2.12-2.16)
3. Update the data encoder and target-task executor of the tester using Eq.(2.10)
4. Update the network weights of the learner by descending $\nabla_W L(A, W, D_{ln}^{(\text{tr})})$

end

2.4 Experiments

We apply LPT for neural architecture search in image classification. Following [58], we first perform architecture search which finds an optimal cell, then perform architecture evaluation which composes multiple copies of the searched cell into a large network, trains it from scratch, and evaluates the trained model on a test set. We let the target tasks of the learner and that of the tester be the same.

2.4.1 Datasets

We used three datasets in the experiments: CIFAR-10, CIFAR-100, and ImageNet [22]. The CIFAR-10 dataset contains 50K training images and 10K testing images, from 10 classes (the number of images in each class is equal). We split the original 50K training set into a 25K training set and a 25K validation set. In the sequel, when we mention "training set", it always refers to the new 25K training set. During architecture search, the training set is used as $D_{ln}^{(\text{tr})}$ of the learner and $D_{tt}^{(\text{tr})}$ of the tester. The validation set is used as the test bank D_b and the validation data $D_{tt}^{(\text{val})}$ of the tester. Under such a setting, the data encoder and target-task executor of the tester are trained on a subset (which is a test) of $D_{tt}^{(\text{val})}$ and validated on the entire set of $D_{tt}^{(\text{val})}$. The interpretation of doing this is: we select a subset of examples from $D_{tt}^{(\text{val})}$ to train a model so that it performs the best on the entire $D_{tt}^{(\text{val})}$. During architecture evaluation, the combination of the training data and validation data is used to train a large network stacking multiple copies of the searched cell. The CIFAR-100 dataset contains 50K training images and 10K testing images, from 100 classes (the number of images in each class is equal). Similar to CIFAR-10, the 50K training images are split into a 25K training set and a 25K validation set. The usage of these subsets is the same as that for CIFAR-10. The ImageNet dataset contains a training set of 1.3M images and a validation set of 50K images, from 1000 object classes. The validation set is used as a test set for architecture evaluation. During architecture search, following [96], 10% of the 1.3M training images are randomly sampled to form a new training set and another 2.5% of the 1.3M training images are randomly sampled to form a new architecture validation set. The usage of the new training set and architecture validation set is the same as that in CIFAR-10. During architecture evaluation, all of the 1.3M training images are used for model training. In addition to searching architectures directly on ImageNet data, following [58], we also evaluate

the architectures searched using CIFAR-10 and CIFAR-100 on ImageNet: given a cell searched using CIFAR-10 and CIFAR-100, multiple copies of it compose a large network, which is then trained on the 1.3M training data of ImageNet and evaluated on the 50K test data.

2.4.2 Experimental Settings

Our framework is a general one that can be used together with any differentiable search method. Specifically, we apply our framework to the following NAS methods: 1) DARTS [58], 2) P-DARTS [16], 3) DARTS$^+$ [54], 4) DARTS$^-$ [18], 5) PC-DARTS [96]. The search space in these methods are similar. 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. In LPT, the network of the learner is a stack of multiple cells, each consisting of 7 nodes. For the data encoder of the tester, we tried ResNet-18 and ResNet-50 [37]. For the test creator and target-task executor, they are set to one feed-forward layer. λ and γ are tuned using a 5k held-out dataset in $\{0.1, 0.5, 1, 2, 3\}$. In most experiments, λ and γ are set to 1 except for P-DARTS and PC-DARTS. For P-DARTS, λ, γ are set to $0.5, 1$ for CIFAR-10 and $1, 0.5$ for CIFAR-100. For PC-DARTS, we use $\lambda = 3, \gamma = 1$ and $\lambda = 0.1, \gamma = 1$ for CIFAR-10 and CIFAR-100, respectively.

For CIFAR-10 and CIFAR-100, during architecture search, the learner's network is a stack of 8 cells, with the initial channel number set to 16. The search is performed for 50 epochs, with a batch size of 64. The hyperparameters for the learner's architecture and weights are set in the same way as DARTS, P-DARTS, DARTS$^+$, and DARTS$^-$. The data encoder and target-task executor of the tester are optimized using SGD with a momentum of 0.9 and a weight decay of 3e-4. The initial learning rate is set to 0.025 with a cosine decay scheduler. The test creator is optimized with the Adam optimizer [47] with a learning rate of 3e-4 and a weight decay of 1e-3.

During architecture evaluation, 20 copies of the searched cell are stacked to form the learner's network, with the initial channel number set to 36. The network is trained for 600 epochs with a batch size of 96 (for both CIFAR-10 and CIFAR-100). The experiments are performed on a single Tesla v100. For ImageNet, following [58], we take the architecture searched on CIFAR-10 and evaluate it on ImageNet. We stack 14 cells (searched on CIFAR-10) to form a large network and set the initial channel number as 48. The network is trained for 250 epochs with a batch size of 1024 on 8 Tesla v100s. Each experiment on LPT is repeated for ten times with the random seed to be from 1 to 10. We report the mean and standard deviation of results obtained from the 10 runs.

2.4.3 Results

Table 6.2 shows the classification error (%), number of weight parameters (millions), and search cost (GPU days) of different NAS methods on CIFAR-100. From this table, we make the following observations. **First**, when our method LPT is applied to different NAS baselines including DARTS-1st (first order approximation), DARTS-2nd (second order approximation), DARTS$^-$ (our run), DARTS$^+$, PC-DARTS, and P-DARTS, the classification errors of these baselines can be significantly reduced. For example, applying our method to P-DARTS, the error reduces from 17.49% to 16.28%. Applying our method to DARTS-2nd, the error reduces from 20.58% to

Method	Error(%)	Param(M)	Cost
*ResNet [36]	22.10	1.7	-
*DenseNet [45]	17.18	25.6	-
*PNAS [56]	19.53	3.2	150
*ENAS [76]	19.43	4.6	0.5
*AmoebaNet [78]	18.93	3.1	3150
*GDAS [25]	18.38	3.4	0.2
*R-DARTS [100]	18.01±0.26	-	1.6
*DropNAS [42]	16.39	4.4	0.7
†DARTS-1st [58]	20.52±0.31	1.8	0.4
LPT-R18-DARTS-1st (ours)	**19.11**±0.11	2.1	0.6
*DARTS-2nd [58]	20.58±0.44	1.8	1.5
LPT-R18-DARTS-2nd (ours)	19.47±0.20	2.1	1.8
LPT-R50-DARTS-2nd (ours)	**18.40**±0.16	2.5	2.0
*DARTS$^-$ [18]	17.51±0.25	3.3	0.4
†DARTS$^-$ [18]	18.97±0.16	3.1	0.4
LPT-R18-DARTS$^-$ (ours)	18.28±0.14	3.4	0.6
$^\Delta$DARTS$^+$ [53]	17.11±0.43	3.8	0.2
LPT-R18-DARTS$^+$ (ours)	**16.58**±0.19	3.7	0.3
†PC-DARTS [96]	17.96±0.15	3.9	0.1
LPT-R18-PC-DARTS (ours)	17.04±0.05	3.6	0.1
LPT-R50-PC-DARTS (ours)	**16.97**±0.21	4.0	0.1
*P-DARTS [16]	17.49	3.6	0.3
LPT-R18-P-DARTS (ours)	**16.28±0.10**	3.8	0.5
LPT-R50-P-DARTS (ours)	16.38±0.07	3.6	0.5

Table 2.2: Results on CIFAR-100, including classification error (%) on the test set, number of parameters (millions) in the searched architecture, and search cost (GPU days). LPT-R18-DARTS-1st denotes that our method LPT is applied to the search space of DARTS. Similar meanings hold for other notations in such a format. R18 and R50 denote that the data encoder of the tester in LPT is set to ResNet-18 and ResNet-50 respectively. DARTS-1st and DARTS-2nd denotes that first order and second order approximation is used in DARTS. * means the results are taken from DARTS$^-$ [18]. † means we re-ran this method for 10 times. Δ means the algorithm ran for 600 epochs instead of 2000 epochs in the architecture evaluation stage, to ensure a fair comparison with other methods (where the epoch number is 600). The search cost is measured by GPU days on a Tesla v100.

18.40%. This demonstrates the effectiveness of our method in searching for a better architecture. In our method, the learner continuously improves its architecture by passing the tests created by the tester with increasing levels of difficulty. These tests can help the learner to identify the weakness of its architecture and provide guidance on how to improve it. Our method creates a new test on the fly based on how the learner performs in the previous round. From the test bank, the tester selects a subset of difficult examples to evaluate the learner. This new test poses a greater challenge to the learner and encourages the learner to improve its architecture so that it can overcome the new challenge. In contrast, in baseline NAS approaches, a single fixed validation set is used to evaluate the learner. The learner can achieve a good performance via "cheating": focusing on performing well on the majority of easy examples and ignoring the minority of difficult examples. As a result, the learner's architecture does not have the ability to deal with challenging cases in the unseen data. **Second**, LPT-R50-DARTS-2nd outperforms LPT-R18-DARTS-2nd,

Method	Error(%)	Param(M)	Cost
*DenseNet [45]	3.46	25.6	-
*HierEvol [57]	3.75±0.12	15.7	300
*NAONet-WS [62]	3.53	3.1	0.4
*PNAS [56]	3.41±0.09	3.2	225
*ENAS [76]	2.89	4.6	0.5
*NASNet-A [106]	2.65	3.3	1800
*AmoebaNet-B [78]	2.55±0.05	2.8	3150
*R-DARTS [100]	2.95±0.21	-	1.6
*GDAS [25]	2.93	3.4	0.2
*GTN [87]	2.92±0.06	8.2	0.67
*SNAS [95]	2.85	2.8	1.5
*BayesNAS [103]	2.81±0.04	3.4	0.2
*MergeNAS [93]	2.73±0.02	2.9	0.2
*NoisyDARTS [19]	2.70±0.23	3.3	0.4
*ASAP [69]	2.68±0.11	2.5	0.2
*SDARTS [14]	2.61±0.02	3.3	1.3
*DropNAS [42]	2.58±0.14	4.1	0.6
*FairDARTS [17]	2.54	3.3	0.4
*DrNAS [15]	2.54±0.03	4.0	0.4
*DARTS-1st [58]	3.00±0.14	3.3	0.4
LPT-R18-DARTS-1st (ours)	**2.85**±0.09	2.7	0.6
*DARTS-2nd [58]	2.76±0.09	3.3	1.5
LPT-R18-DARTS-2nd (ours)	2.72±0.07	3.4	1.8
LPT-R50-DARTS-2nd (ours)	**2.68**±0.02	3.4	2.0
*DARTS$^-$ [18]	2.59±0.08	3.5	0.4
†DARTS$^-$ [18]	2.97±0.04	3.3	0.4
LPT-R18-DARTS$^-$ (ours)	2.74±0.07	3.4	0.6
$^\Delta$DARTS$^+$ [53]	2.83±0.05	3.7	0.4
LPT-R18-DARTS$^+$ (ours)	**2.69**±0.05	3.6	0.5
*PC-DARTS [96]	**2.57**±0.07	3.6	0.1
LPT-R18-PC-DARTS (ours)	2.65±0.17	3.7	0.1
*P-DARTS [16]	2.50	3.4	0.3
LPT-R18-P-DARTS (ours)	2.58±0.14	3.3	0.5

Table 2.3: Results on CIFAR-10. * means the results are taken from DARTS$^-$ [18], Noisy-DARTS [19], and DrNAS [15]. The rest notations are the same as those in Table 6.2.

where the former uses ResNet-50 as the data encoder in the tester while the latter uses ResNet-18. ResNet-50 has a better ability of learning representations than ResNet-18 since it is "deeper": 50 layers versus 18 layers. This shows that a "stronger" tester can help the learner to learn better. With a more powerful data encoder, the tester can better understand examples in the test bank and can make better decisions in creating difficult and meaningful tests. Tests with better quality can evaluate the learner more effectively and help to improve the learner's learning capability. When our method is applied to PC-DARTS and P-DARTS, the performance difference resulting from ResNet-18 and ResNet-50 is not statistically significant. **Third**, our method LPT-R18-P-DARTS achieves the best performance among all methods, which further demonstrates the effectiveness of LPT in driving the frontiers of neural architecture search forward. **Fourth**, the number of weight parameters and search costs corresponding to our methods are on par with those in differentiable NAS baselines. This shows that LPT is able to search better-performing architectures without significantly increasing network size and search cost. A few additional remarks: 1) On CIFAR-100, DARTS-2nd with second-order approximation in the optimization algorithm is not

Method	Top-1 Error (%)	Top-5 Error (%)	Param (M)	Cost (GPU days)
*Inception-v1 [88]	30.2	10.1	6.6	-
*MobileNet [43]	29.4	10.5	4.2	-
*ShuffleNet 2× (v1) [101]	26.4	10.2	5.4	-
*ShuffleNet 2× (v2) [63]	25.1	7.6	7.4	-
*NASNet-A [106]	26.0	8.4	5.3	1800
*PNAS [56]	25.8	8.1	5.1	225
*MnasNet-92 [89]	25.2	8.0	4.4	1667
*AmoebaNet-C [78]	24.3	7.6	6.4	3150
*SNAS-CIFAR10 [95]	27.3	9.2	4.3	1.5
*BayesNAS-CIFAR10 [103]	26.5	8.9	3.9	0.2
*PARSEC-CIFAR10 [12]	26.0	8.4	5.6	1.0
*GDAS-CIFAR10 [25]	26.0	8.5	5.3	0.2
*DSNAS-ImageNet [44]	25.7	8.1	-	-
*SDARTS-ADV-CIFAR10 [14]	25.2	7.8	5.4	1.3
*PC-DARTS-CIFAR10 [96]	25.1	7.8	5.3	0.1
*ProxylessNAS-ImageNet [10]	24.9	7.5	7.1	8.3
*FairDARTS-CIFAR10 [17]	24.9	7.5	4.8	0.4
*FairDARTS-ImageNet [17]	24.4	7.4	4.3	3.0
*DrNAS-ImageNet [15]	24.2	7.3	5.2	3.9
*DARTS$^+$-ImageNet [53]	23.9	7.4	5.1	6.8
*DARTS$^-$-ImageNet [18]	23.8	7.0	4.9	4.5
*DARTS$^+$-CIFAR100 [53]	23.7	7.2	5.1	0.2
*DARTS-2nd-CIFAR10 [58]	26.7	8.7	4.7	1.5
LPT-R18-DARTS-2nd-CIFAR10 (ours)	25.3	7.9	4.7	1.8
*P-DARTS (CIFAR10) [16]	24.4	7.4	4.9	0.3
‡LPT-R18-P-DARTS-CIFAR10 (ours)	24.2	7.3	4.9	0.5
*P-DARTS (CIFAR100) [16]	24.7	7.5	5.1	0.3
‡LPT-R18-P-DARTS-CIFAR100 (ours)	24.0	7.1	5.3	0.5
*PC-DARTS-ImageNet [96]	24.2	7.3	5.3	3.8
‡LPT-R18-PC-DARTS-ImageNet (ours)	**23.4**	**6.8**	5.7	4.0

Table 2.4: Results on ImageNet, including top-1 and top-5 classification errors on test set. * means the results are taken from DARTS$^-$ [18] and DrNAS [15]. The rest notations are the same as those in Table 6.2. The first row block shows networks designed by humans manually. The second block shows non-gradient based methods. The third block shows gradient-based methods. ‡ means the results are obtained by following the hyperparameters selected for CIFAR10/100. The hyperparameters for CIFAR-100 are used when directly searching on ImageNet.

advantageous compared with DARTS-1st which uses first-order approximation; 2) In our run of DARTS$^-$, we were not able to achieve the performance reported in [18]; 3) In our run of DARTS$^+$, in the architecture evaluation stage, we set the number of epochs to 600 instead of 2000 as in [53], to ensure a fair comparison with other methods (where the epoch number is 600).

Table 6.3 shows the classification error (%), number of weight parameters (millions), and search cost (GPU days) of different NAS methods on CIFAR-10. As can be seen, applying our proposed LPT to DARTS-1st, DARTS-2nd, DARTS$^-$ (our run), and DARTS$^+$ significantly reduces the errors of these baselines. For example, with the usage of LPT, the error of DARTS-2nd is reduced from 2.76% to 2.68%. This further demonstrates the efficacy of our method in searching better-performing architectures, by creating tests with increasing levels of difficulty

and improving the learner through taking these tests. On PC-DARTS and P-DARTS, applying our method does not yield better performance.

Table 6.4 shows the results on ImageNet, including top-1 and top-5 classification errors on the test set. In our proposed LPT-R18-PC-DARTS-ImageNet, the architecture is searched on ImageNet, where our method performs much better than PC-DARTS-ImageNet and achieves the lowest error (23.4% top-1 error and 6.8% top-5 error) among all methods in Table 6.4. In our methods including LPT-R18-P-DARTS-CIFAR100, LPT-R18-P-DARTS-CIFAR10, and LPT-R18-DARTS-2nd-CIFAR10, the architectures are searched on CIFAR-10 or CIFAR-100 and evaluated on ImageNet, where these methods outperform their corresponding baselines P-DARTS-CIFAR100, P-DARTS-CIFAR10, and DARTS-2nd-CIFAR10. These results further demonstrate the effectiveness of our method.

2.4.4 Ablation Studies

In order to evaluate the effectiveness of individual modules in LPT, we compare the full LPT framework with the following ablation settings.

- **Ablation setting 1**. In this setting, the tester creates tests solely by maximizing their level of difficulty, without considering their meaningfulness. Accordingly, the second stage in LPT where the tester learns to perform a target-task by leveraging the created tests is removed. The tester directly learns a selection scalar $s(d) \in [0, 1]$ for each example d in the test bank without going through a data encoder or test creator. The corresponding formulation is:

$$\begin{aligned} &\max_S \min_A \ \ \frac{1}{\sum_{d\in D_b} s(d)} \sum_{d\in D_b} s(d)\ell(A, W^*(A), d) \\ &s.t. \ \ W^*(A) = \min_W \ \ L\left(A, W, D_{ln}^{(\mathrm{tr})}\right) \end{aligned} \tag{2.17}$$

  where $S = \{s(d)|d \in D_b\}$. In this study, λ and γ are both set to 1. The data encoder of the tester is ResNet-18. For CIFAR-100, LPT is applied to P-DARTS and DARTS-2nd. For CIFAR-10, LPT is applied to DARTS-2nd.

- **Ablation setting 2**. In this setting, in the second stage of LPT, the tester is trained solely based on the created test, without using the training data of the target task. The corresponding formulation is:

$$\begin{aligned} &\max_C \min_A \ \ \frac{1}{|\sigma(C,E^*(C),D_b)|} L\left(A, W^*\left(A\right), \sigma\left(C, E^*(C), D_b\right)\right) - \lambda L\left(E^*(C), X^*(C), D_{tt}^{(\mathrm{val})}\right) \\ &s.t. \ \ E^*(C), X^*(C) = \min_{E,X} \ \ L\left(E, X, \sigma\left(C, E, D_b\right)\right) \\ &\qquad W^*\left(A\right) = \min_W \ \ L\left(A, W, D_{ln}^{(\mathrm{tr})}\right) \end{aligned} \tag{2.18}$$

  In this study, λ and γ are both set to 1. The data encoder of the tester is ResNet-18. For CIFAR-100, LPT is applied to P-DARTS and DARTS-2nd. For CIFAR-10, LPT is applied to DARTS-2nd.

- Ablation study on λ. We are interested in how the learner's performance varies as the tradeoff parameter λ in Eq.(2.3) increases. In this study, the other tradeoff parameter γ in Eq.(2.3) is set to 1. For both CIFAR-100 and CIFAR-10, we randomly sample 5K data from the 25K training and 25K validation data, and use it as a test set to report performance in this ablation

Method	Error (%)
Difficult only (DARTS-2nd, CIFAR-100)	20.38±0.17
Difficult + meaningful (DARTS-2nd, CIFAR-100)	**19.47**±0.20
Difficult only (P-DARTS, CIFAR-100)	18.12±0.11
Difficult + meaningful (P-DARTS, CIFAR-100)	**16.28**±0.10
Difficult only (DARTS-2nd, CIFAR-10)	2.79±0.06
Difficult + meaningful (DARTS-2nd, CIFAR-10)	**2.72**±0.07

Table 2.5: Results for ablation setting 1. "Difficult only" denotes that the tester creates tests solely by maximizing their level of difficulty, without considering their meaningfulness, i.e., the tester does not use the tests for learning to perform the target task. "Difficult + meaningful" denotes the full LPT framework where the tester creates tests by maximizing both difficulty and meaningfulness.

Method	Error (%)
Test only (DARTS-2nd, CIFAR-100)	19.81±0.06
Test + training (DARTS-2nd, CIFAR-100)	**19.47**±0.20
Test only (P-DARTS, CIFAR-100)	17.54±0.07
Test + training (P-DARTS, CIFAR-100)	**16.28**±0.10
Test only (DARTS-2nd, CIFAR-10)	2.75±0.03
Test + training (DARTS-2nd, CIFAR-10)	**2.72**±0.07

Table 2.6: Results for ablation setting 2. "Test only" denotes that the tester is trained only using the created test to perform the target task. "Test + training" denotes that the tester is trained using both the test and the training data of the target task.

study. The rest 45K data is used as before. Tester's data encoder is ResNe-18. LPT is applied to P-DARTS.

- Ablation study on γ. We investigate how the learner's performance varies as c increases. In this study, the other tradeoff parameter λ is set to 1. Similar to the ablation study on λ, on 5K randomly-sampled test data, we report performance of architectures searched under different values of γ. Tester's data encoder is ResNe-18. LPT is applied to P-DARTS.

Table 6.5 shows the results for ablation setting 1. As can be seen, on both CIFAR-10 and CIFAR-100, creating tests that are both difficult and meaningful is better than creating tests solely by maximizing difficulty. The reason is that a difficult test could be composed of bad-quality examples such as outliers and incorrectly-labeled examples. Even a highly-accurate model cannot achieve good performance on such erratic examples. To address this problem, it is necessary to make the created tests meaningful. LPT achieves meaningfulness of the tests by making the tester leverage the created tests to perform the target task. The results demonstrate that this is an effective way of improving meaningfulness.

Table 6.6 shows the results for ablation setting 2. As can be seen, for both CIFAR-100 and CIFAR-10, using both the created test and the training data of the target task to train the tester performs better than using the test only. By leveraging the training data, the data encoder can be better trained. And a better encoder can help to create higher-quality tests.

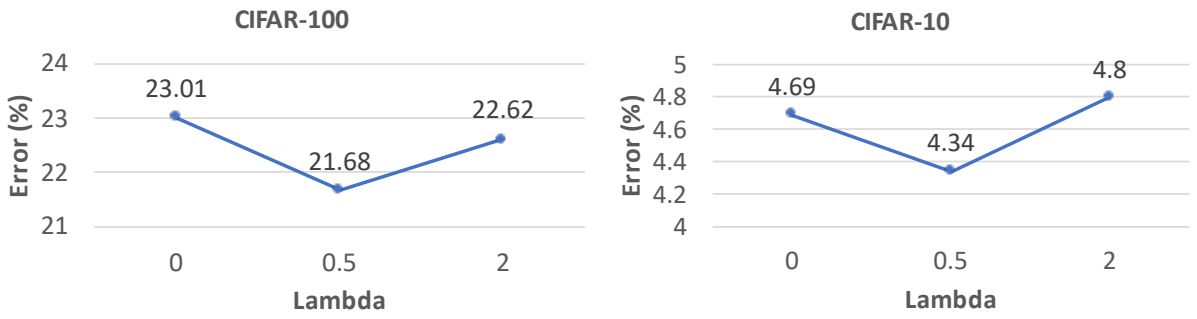

Figure 2.3: How errors change as λ increases.

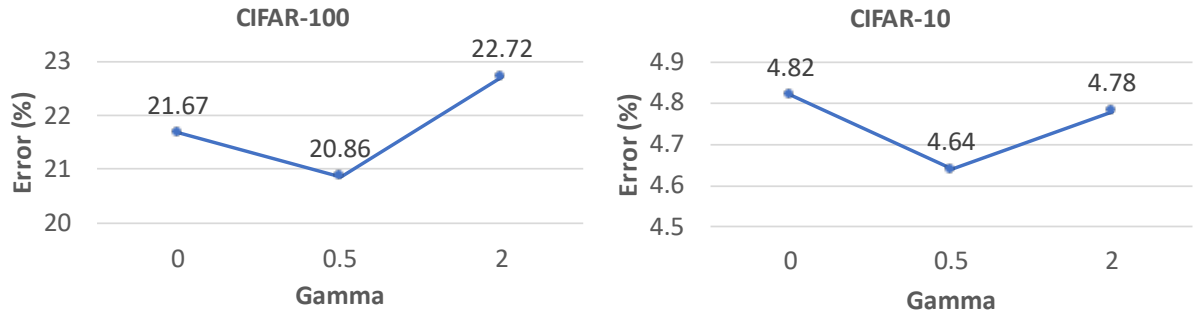

Figure 2.4: How errors change as γ increases.

Figure 7.3 shows how classification errors change as λ increases. As can be seen, on both CIFAR-100 and CIFAR-10, when λ increases from 0 to 0.5, the error decreases. However, further increasing λ renders the error to increase. From the tester's perspective, λ explores a tradeoff between difficulty and meaningfulness of the tests. Increasing λ encourages the tester to create tests that are more meaningful. Tests with more meaningfulness can more reliably evaluate the learner. However, if λ is too large, the tests are biased to be more meaningful but less difficult. Lacking enough difficulty, the tests may not be compelling enough to drive the learner for improvement. Such a tradeoff effect is observed in the results on CIFAR-10 as well.

Figure 6.4 shows how classification errors change as γ increases. As can be seen, on both CIFAR-100 and CIFAR-10, when γ increases from 0 to 0.5, the error decreases. However, further increasing γ renders the error to increase. Under a larger γ, the created test plays a larger role in training the tester to perform the target task. This implicitly encourages the test creator to generate tests that are more meaningful. However, if γ is too large, training is dominated by the created test which incurs the following risk: if the test is not meaningful, it will result in a poor-quality data-encoder which degrades the quality of created tests.

2.5 Conclusions

In this section, we propose a new machine learning approach – learning by passing tests (LPT), inspired by the test-driven learning technique of humans. In LPT, a tester model creates a se-

quence of tests with growing levels of difficulty. A learner model continuously improves its learning ability by striving to pass these increasingly more-challenging tests. We propose a multi-level optimization framework to formalize LPT where the tester learns to select hard validation examples that render the learner to make large prediction errors and the learner refines its model to rectify these prediction errors. Our framework is applied for neural architecture search and achieves significant improvement on CIFAR-100, CIFAR-10, and ImageNet.

Chapter 3

Interleaving Learning

3.1 Introduction

Interleaving learning is a learning technique where a learner interleaves the studies of multiple topics: study topic A for a while, then switch to B, subsequently to C; then switch back to A, and so on, forming a pattern of $ABCABCABC\cdots$. Interleaving learning is in contrast to blocked learning, which studies one topic very thoroughly before moving to another topic. Compared with blocked learning, interleaving learning increases long-term retention and improves ability to transfer learned knowledge. Figure 7.1 illustrates the difference between interleaving learning and block learning.

Motivated by humans' interleaving learning methodology, we are intrigued to explore whether machine learning can be benefited from this learning methodology as well. We propose a novel multi-level optimization framework to formalize the idea of learning multiple topics in an interleaving way. In this framework, we assume there are K learning tasks, each performed by a learner model. Each learner has a data encoder and a task-specific head. The data encoders of all learners share the same architecture, but may have different weight parameters. The K learners perform M rounds of interleaving learning with the following order:

$$\underbrace{l_1, l_2, \cdots, l_K}_{\text{Round 1}}\underbrace{l_1, l_2, \cdots, l_K}_{\text{Round 2}}\cdots\underbrace{l_1, l_2, \cdots, l_K}_{\text{Round } m}\cdots\underbrace{l_1, l_2, \cdots, l_K}_{\text{Round } M} \tag{3.1}$$

where l_k denotes that the k-th learner performs learning. In the first round, we first learn l_1, then learn l_2, and so on. At the end of the first round, l_K is learned. Then we move to the second round, which starts with learning l_1, then learns l_2, and so on. This pattern repeats until the M rounds of learning are finished. Between two consecutive learners $l_k l_{k+1}$, the encoder weights of the latter learner l_{k+1} are encouraged to be close to the optimally learned encoder weights of the former learner l_k. In the interleaving process, the K learners help each other to learn better. Each learner transfers the knowledge learned in its task to the next learner by using its trained encoder to initialize the encoder of the next learner. Meanwhile, each learner leverages the knowledge shared by the previous learner to better train its own model. Via knowledge sharing, in one round of learning, l_1 helps l_2 to learn better, l_2 helps l_3 to learn better, and so on. Then moving into the next round, l_K learned in the previous round helps l_1 to re-learn for achieving

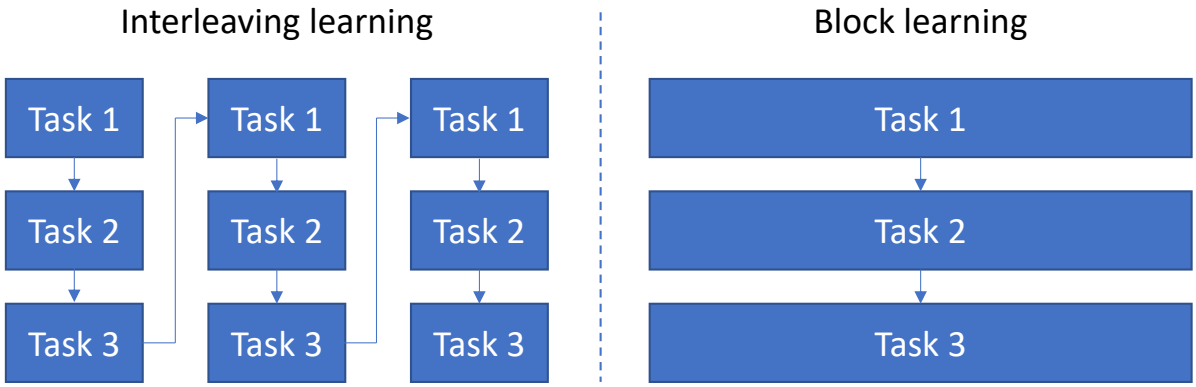

Figure 3.1: Comparison between interleaving learning and block learning. In interleaving learning, we perform task 1 for a short while, then move to task 2, then task 3. Afterwards, we move from task 3 back to task 1. This process iterates where each task is performed for a short time period before switching to another task. In contrast, in block learning, we perform task 1 to the very end, then move to task 2, and later task 3. Each task is performed for a long time period until it is completed. Once a task is finished, it will not be performed again.

a better learning outcome, then a better l_1 further helps l_2 to learn better, and so on. After M rounds of learning, each learner uses its model trained in the final round to make predictions on a validation dataset and updates their shared encoder architecture by minimizing the validation losses. Our interleaving learning framework is applied to search neural architectures for image classification on CIFAR-10, CIFAR-100, and ImageNet, where experimental results demonstrate the effectiveness of our method.

The major contributions of this chapter are as follows:

- Drawing insights from a human learning methodology – interleaving learning, we propose a novel machine learning framework which enables a set of models to cooperatively train a data encoder in an interleaving way: model 1 trains this encoder for a short time, then hands it over to model 2 to continue the training, then to model 3, etc. When the encoder is trained by all models in one pass, it returns to model 1 and starts the second round of training sequentially by each model. This cyclic training process iterates until convergence. During the interleaving process, each model transfers its knowledge to the next model and leverages the knowledge shared by the previous model to learn better.
- We formulate interleaving machine learning as a multi-level optimization problem.
- We develop an efficient differentiable algorithm to solve the interleaving learning problem.
- We utilize our interleaving learning framework for neural architecture search on CIFAR-100, CIFAR-10, and ImageNet. Experimental results strongly demonstrate the effectiveness of our method.

3.2 Related Works

The goal of neural architecture search (NAS) is to automatically identify highly-performing neural architectures that can potentially surpass human-designed ones. NAS research has made considerable progress in the past few years. Early NAS [76, 105, 106] approaches are based on reinforcement learning (RL), where a policy network learns to generate high-quality architectures by maximizing the validation accuracy (as reward). These approaches are conceptually simple and can flexibly perform search in any search spaces. However, they are computationally very demanding. To calculate the reward of a candidate architecture, this architecture needs to be trained on a training dataset, which is very time-consuming. To address this issue, differentiable search methods [10, 58, 95] have been proposed. In these methods, each candidate architecture is a combination of many building blocks. The combination coefficients represent the importance of building blocks. Architecture search amounts to learning these differentiable coefficients, which can be done using differentiable optimization algorithms such as gradient descent, with much higher computational efficiency than RL-based approaches. Differentiable NAS methods started with DARTS [58] and have been improved rapidly since then. For example, P-DARTS [16] allows the architecture depth to increase progressively during searching. It also performs search space regularization and approximation to improve stability of searching algorithms and reduce search cost. In PC-DARTS [96], the redundancy of search space exploration is reduced by sampling sub-networks from a super network. It also performs operation search in a subset of channels via bypassing the held-out subset in a shortcut. Another paradigm of NAS methods [57, 78] are based on evolutionary algorithms (EA). In these approaches, architectures are considered as individuals in a population. Each architecture is associated with a fitness score representing how good this architecture is. Architectures with higher fitness scores have higher odds of generating offspring (new architectures), which replace architectures that have low-fitness scores. Similar to RL-based methods, EA-based methods are computationally heavy since evaluating the fitness score of an architecture needs to train this architecture. Our proposed interleaving learning framework in principle can be applied to any NAS methods. In our experiments, for simplicity and computational efficiency, we choose to work on differentiable NAS methods.

3.3 Method

In this section, we present the details of the interleaving learning framework. There are K learners. Each learner learns to perform a task. These tasks could be the same, e.g., image classification on CIFAR-10; or different, e.g., image classification on CIFAR-10, image classification on ImageNet [22], object detection on MS-COCO [55], etc. Each learner k has a training dataset $D_k^{(\text{tr})}$ and a validation dataset $D_k^{(\text{val})}$. Each learner has a data encoder and a task-specific head performing the target task. For example, if the task is image classification, the data encoder could be a convolutional neural network extracting visual features of the input images and the task-specific head could be a multi-layer perceptron which takes the visual features of an image extracted by the data encoder as input and predicts the class label of this image. We assume the architecture of the data encoder in each learner is learnable. The data encoders of all learners

Notation	Meaning
K	Number of learners
M	Number of rounds
$D_k^{(\text{tr})}$	Training dataset of the k-th learner
$D_k^{(\text{val})}$	Validation dataset of the k-th learner
A	Encoder architecture shared by all learners
$W_k^{(m)}$	Weight parameters in the data encoder of the k-th learner in the m-th round
$H_k^{(m)}$	Weight parameters in the task-specific head of the k-th learner in the m-th round
$\widetilde{W}_k^{(m)}$	The optimal encoder weights of the k-th learner in the m-th round
$\widetilde{H}_k^{(m)}$	The optimal weight parameters of the task-specific head in the k-th learner in the m-th round
γ	Tradeoff parameter

Table 3.1: Notations in interleaving learning

share the same architecture, but their weight parameters could be different in different learners. The architectures of task-specific heads are manually designed by humans and they could be different in different learners. The K learners perform M rounds of interleaving learning with the following order:

$$\underbrace{l_1, l_2, \cdots, l_K}_{\text{Round 1}} \underbrace{l_1, l_2, \cdots, l_K}_{\text{Round 2}} \cdots \underbrace{l_1, l_2, \cdots, l_K}_{\text{Round } m} \cdots \underbrace{l_1, l_2, \cdots, l_K}_{\text{Round } M} \tag{3.2}$$

where l_k denotes that the k-th learner performs learning. In the first round, we first learn l_1, then learn l_2, and so on. At the end of the first round, l_K is learned. Then we move to the second round, which starts with learning l_1, then learns l_2, and so on. This pattern repeats until the M rounds of learning are finished. Between two consecutive learners $l_k l_{k+1}$, the weight parameters of the latter learner l_{k+1} are encouraged to be close to the optimally learned encoder weights of the former learner l_k. For each learner, the architecture of its encoder remains the same across all rounds; the network weights of the encoder and head can be different in different rounds.

Each learner k has the following learnable parameter sets: 1) architecture A of the encoder; 2) in each round m, the learner's encoder has a set of weight parameters $W_k^{(m)}$ specific to this round; 3) in each round m, the learner's task-specific head has a set of weight parameters $H_k^{(m)}$ specific to this round. The encoders of all learners share the same architecture and this architecture remains the same in different rounds. The encoders of different learners have different weight parameters. The weight parameters of a learner's encoder are different in different rounds. Different learners have different task-specific heads in terms of both architectures and weight parameters. In the interleaving process, the learning of the k-th learner is assisted by the $(k-1)$-th learner. Specifically, during learning, the encoder weights W_k of the k-th learner are encouraged to be close to the optimal encoder weights $\widetilde{W}_{k-1}$ of the $(k-1)$-th learner. This is achieved by minimizing the following regularizer : $\|W_k - \widetilde{W}_{k-1}\|_2^2$.

There are $M \times K$ learning stages: in each of the M rounds, each of the K learners is learned in a stage. In the very first learning stage, the first learner in the first round is learned. It trains

the weight parameters of its data encoder and the weight parameters of its task-specific head on its training dataset. The optimization problem is:

$$\widetilde{W}_1^{(1)}(A) = \min_{W_1^{(1)}, H_1^{(1)}} L(A, W_1^{(1)}, H_1^{(1)}, D_1^{(\text{tr})}). \tag{3.3}$$

In this optimization problem, A is not learned. Otherwise, a trivial solution of A will be resulted in. In this trivial solution, A would be excessively large and expressive, and can perfectly overfit the training data, but will have poor generalization capability on unseen data. After learning, the optimal head is discarded. The optimal encoder weights $\widetilde{W}_1^{(1)}(A)$ are a function of A since the training loss is a function of A and $\widetilde{W}_1$ is a function of the training loss. $\widetilde{W}_1^{(1)}(A)$ is passed to the next learning stage to help with the learning of the second learner.

In any other learning stage, e.g., the l-th stage where the learner is k and the round of interleaving is m, the optimization problem is:

$$\widetilde{W}_k^{(m)}(A) = \min_{W_k^{(m)}, H_k^{(m)}} L(A, W_k^{(m)}, H_k^{(m)}, D_k^{(\text{tr})}) + \lambda \|W_k^{(m)} - \widetilde{W}_{l-1}(A)\|_2^2,$$

where $\|W_k^{(m)} - \widetilde{W}_{l-1}\|_2^2$ encourages the encoder weights $W_k^{(m)}$ at this stage to be close to the optimal encoder weights $\widetilde{W}_{l-1}$ learned in the previous stage and λ is a tradeoff parameter. The optimal encoder weights $\widetilde{W}_k^{(m)}(A)$ are a function of the encoder architecture A. The encoder architecture is not updated at this learning stage, for the same reason described above. In the round of 1 to $M-1$, the optimal heads are discarded after learning. In the round of M, the optimal heads $\{\widetilde{H}_k^{(M)}(A)\}_{k=1}^K$ are retained and will be used in the final learning stage. In the final stage, each learner evaluates its model learned in the final round M on the validation set. The encoder architecture A is learned by minimizing the validation losses of all learners. The corresponding optimization problem is:

$$\min_A \sum_{k=1}^{K} L(A, \widetilde{W}_k^{(M)}(A), \widetilde{H}_k^{(M)}(A), D_k^{(\text{val})}). \tag{3.4}$$

To this end, we are ready to formulate the interleaving learning problem using a multi-level optimization framework, as shown in Eq.(3.5). From bottom to top, the K learners perform M rounds of interleaving learning. Learners in adjacent learning stages are coupled via $\|W_k - \widetilde{W}_{k-1}\|_2^2$. The architecture A is learned by minimizing the validation loss. Similar to [58], we represent A in a differentiable way. A is a weighted combination of multiple layers of basic building blocks such as convolution, pooling, normalization, etc. The output of each building block is multiplied with a weight indicating how important this block is. During architecture search, these differentiable weights are learned. After the search process, blocks with large weights are retained to form the final architecture.

$$
\begin{aligned}
\min_A \quad & \textstyle\sum_{k=1}^{K} L(A, \widetilde{W}_k^{(M)}(A), \widetilde{H}_k^{(M)}(A), D_k^{(\text{val})}) \\
s.t. \quad & \textbf{Round M}: \\
& \widetilde{W}_K^{(M)}(A), \widetilde{H}_K^{(M)}(A) = \min_{W_K^{(M)}, H_K^{(M)}} L(A, W_K^{(M)}, H_K^{(M)}, D_K^{(\text{tr})}) + \lambda \|W_K^{(M)} - \widetilde{W}_{K-1}^{(M)}(A)\|_2^2 \\
& \cdots \\
& \widetilde{W}_1^{(M)}(A), \widetilde{H}_1^{(M)}(A) = \min_{W_1^{(M)}, H_1^{(M)}} L(A, W_1^{(M)}, H_1^{(M)}, D_1^{(\text{tr})}) + \lambda \|W_1^{(M)} - \widetilde{W}_K^{(M-1)}(A)\|_2^2 \\
& \cdots \\
& \textbf{Round 2}: \\
& \widetilde{W}_K^{(2)}(A) = \min_{W_K^{(2)}, H_K^{(2)}} \quad L(A, W_K^{(2)}, H_K^{(2)}, D_K^{(\text{tr})}) + \lambda \|W_K^{(2)} - \widetilde{W}_{K-1}^{(2)}(A)\|_2^2 \\
& \cdots \\
& \widetilde{W}_1^{(2)}(A) = \min_{W_1^{(2)}, H_1^{(2)}} \quad L(A, W_1^{(2)}, H_1^{(2)}, D_1^{(\text{tr})}) + \lambda \|W_1^{(2)} - \widetilde{W}_K^{(1)}(A)\|_2^2 \\
& \textbf{Round 1}: \\
& \widetilde{W}_K^{(1)}(A) = \min_{W_K^{(1)}, H_K^{(1)}} \quad L(A, W_K^{(1)}, H_K^{(1)}, D_K^{(\text{tr})}) + \lambda \|W_K^{(1)} - \widetilde{W}_{K-1}^{(1)}(A)\|_2^2 \\
& \cdots \\
& \widetilde{W}_k^{(1)}(A) = \min_{W_k^{(1)}, H_k^{(1)}} \quad L(A, W_k^{(1)}, H_k^{(1)}, D_k^{(\text{tr})}) + \lambda \|W_k^{(1)} - \widetilde{W}_{k-1}^{(1)}(A)\|_2^2 \\
& \cdots \\
& \widetilde{W}_2^{(1)}(A) = \min_{W_2^{(1)}, H_2^{(1)}} \quad L(A, W_2^{(1)}, H_2^{(1)}, D_2^{(\text{tr})}) + \lambda \|W_2^{(1)} - \widetilde{W}_1^{(1)}(A)\|_2^2 \\
& \widetilde{W}_1^{(1)}(A) = \min_{W_1^{(1)}, H_1^{(1)}} \quad L(A, W_1^{(1)}, H_1^{(1)}, D_1^{(\text{tr})})
\end{aligned}
\tag{3.5}
$$

3.3.1 Optimization Algorithm

In this section, we develop an optimization algorithm for interleaving learning. For each optimization problem $\widetilde{W}_k^{(m)}(A) = \min_{W_k^{(m)}, H_k^{(m)}} \quad L(A, W_k^{(m)}, H_k^{(m)}, D_k^{(\text{tr})}) + \lambda \|W_k^{(m)} - \widetilde{W}_{k-1}^{(m)}(A)\|_2^2$ in a learning stage, we approximate the optimal solution $\widetilde{W}_k^{(m)}(A)$ by one-step gradient descent update of the optimization variable $W_k^{(m)}$:

$$
\widetilde{W}_k^{(m)}(A) \approx \overline{W}_k^{(m)}(A) = W_k^{(m)} - \eta \nabla_{W_k^{(m)}} (L(A, W_k^{(m)}, H_k^{(m)}, D_k^{(\text{tr})}) + \lambda \|W_k^{(m)} - \widetilde{W}_{k-1}^{(m)}(A)\|_2^2).
$$

For $\widetilde{W}_1^{(1)}(A)$, the approximation is:

$$
\widetilde{W}_1^{(1)}(A) \approx \overline{W}_1^{(1)}(A) = W_1^{(1)} - \eta \nabla_{W_1^{(1)}} L(A, W_1^{(1)}, H_1^{(1)}, D_1^{(\text{tr})}). \tag{3.6}
$$

For $\widetilde{W}_k^{(m)}(A)$, the approximation is:

$$
\overline{W}_k^{(m)}(A) = W_k^{(m)} - \eta \nabla_{W_k^{(m)}} L(A, W_k^{(m)}, H_k^{(m)}, D_k^{(\text{tr})}) - 2\eta\lambda (W_k^{(m)} - \overline{W}_{k-1}^{(m)}(A)), \tag{3.7}
$$

where $\overline{W}_{k-1}^{(m)}(A)$ is the approximation of $\widetilde{W}_{k-1}^{(m)}(A)$. Note that $\{\overline{W}_k^{(m)}(A)\}_{k,m=1}^{K,M}$ are calculated recursively, where $\overline{W}_k^{(m)}(A)$ is a function of $\overline{W}_{k-1}^{(m)}(A)$, $\overline{W}_{k-1}^{(m)}(A)$ is a function of $\overline{W}_{k-2}^{(m)}(A)$, and

so on. When $m > 1$ and $k = 1$, $\overline{W}_{k-1}^{(m)}(A) = \overline{W}_{K}^{(m-1)}(A)$. For $\widetilde{H}_k^{(M)}(A)$, the approximation is:

$$\overline{H}_k^{(M)}(A) = H_k^{(M)}(A) - \eta \nabla_{H_k^{(M)}(A)} L(A, W_k^{(M)}, H_k^{(M)}, D_k^{(\text{tr})}). \tag{3.8}$$

In the validation stage, we plug the approximations of $\{\widetilde{W}_k^{(M)}(A)\}_{k=1}^K$ and $\{\widetilde{H}_k^{(M)}(A)\}_{k=1}^K$ into the validation loss function, calculate the gradient of the approximated objective w.r.t the encoder architecture A, then update A via:

$$A \leftarrow A - \eta \sum_{k=1}^{K} \nabla_A L(A, \overline{W}_k^{(M)}(A), \overline{H}_k^{(M)}(A), D_k^{(\text{val})}). \tag{3.9}$$

The update steps from Eq.(3.6) to Eq.(3.9) iterate until convergence. The entire algorithm is summarized in Algorithm 3.

Algorithm 3: Optimization algorithm for interleaving learning

while *not converged* **do**

1. Update $\widetilde{W}_1^{(1)}(A)$ using Eq.(3.6)
2. For $k = 2 \cdots K$, update $\widetilde{W}_k^{(1)}(A)$ using Eq.(3.7)
3. For $k = 1 \cdots K$ and $m = 2 \cdots M$, update $\widetilde{W}_k^{(m)}(A)$ using Eq.(3.7)
4. For $k = 1 \cdots K$, update $\widetilde{H}_k^{(M)}(A)$ using Eq.(3.8)
5. Update A using Eq.(3.9)

end

3.4 Experiments

In this section, we apply the proposed interleaving ML framework for neural architecture search in image classification tasks. Following the experimental protocol in [58], each experiment consists of an architecture search phrase and an architecture evaluation phrase. In the search phrase, an optimal architecture cell is searched by minimizing the validation loss. In the evaluation phrase, a larger network is created by stacking multiple copies of the optimally searched cell. This new network is re-trained from scratch and evaluated on the test set. Please refer to the supplements for more details of hyperparameters, results, and significance tests of results.

3.4.1 Datasets

Three popular image classification datasets are involved in the experiments: CIFAR-10, CIFAR-100, and ImageNet [22]. CIFAR-10 contains 60K images from 10 classes. CIFAR-100 contains 60K images from 100 classes. ImageNet contains 1.25 million images from 1000 classes. For CIFAR-10 and CIFAR-100, each of them is split into train/validation/test sets with 25K/25K/10K images respectively. For ImageNet, it has 1.2M training images and 50K test images.

3.4.2 Baselines

Our IL framework can be generally used together with any differentiable NAS method. In the experiments, we apply IL to three widely-used NAS methods: DARTS [58], P-DARTS [16], and PC-DARTS [96]. The search space of these methods are similar, where the building blocks include 3×3 and 5×5 (dilated) separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and zero. We compare our interleaving framework with a multi-task learning framework where a shared encoder architecture is searched simultaneously on CIFAR-10 and CIFAR-100: The formulation is:

$$\begin{array}{ll} \min_A & L(A, \widetilde{W}_{100}(A), \widetilde{H}_{100}(A), D_{100}^{(\text{val})}) + \alpha L(A, \widetilde{W}_{10}(A), \widetilde{H}_{10}(A), D_{10}^{(\text{val})}) \\ s.t. & \widetilde{W}_{100}(A), \widetilde{H}_{100}(A), \widetilde{W}_{10}(A), \widetilde{H}_{10}(A) = \\ & \min_{W_{100},H_{100},W_{10},H_{10}} \quad L(A, W_{100}, H_{100}, D_{100}^{(\text{tr})}) + \beta L(A, W_{10}, H_{10}, D_{10}^{(\text{tr})}) \end{array} \tag{3.10}$$

where W_{100} and H_{100} are the encoder weights and classification head for CIFAR-100. W_{10} and H_{10} are the encoder weights and classification head for CIFAR-10. $D_{100}^{(\text{tr})}$ and $D_{100}^{(\text{val})}$ are the training and validation sets of CIFAR-100. $D_{10}^{(\text{tr})}$ and $D_{10}^{(\text{val})}$ are the training and validation sets of CIFAR-10. A is the encoder architecture shared by CIFAR-100 and CIFAR-10. α and β in Eq.(3.10) are both set to 1.

3.4.3 Experimental Settings

In the interleaving learning framework, we set two learners: one learns to classify CIFAR-10 images and the other learns to classify CIFAR-100 images. Each learner has an image encoder and a classification head. Encoders of these two learners share the same architecture, whose search space is the same as that in DARTS/P-DARTS/PC-DARTS. The encoder is a stack of 8 cells, each consisting of 7 nodes. The initial channel number was set to 16. For the learner on CIFAR-10, the classification head is a 10-way linear classifier. The training and validation set of CIFAR-10 is used as $D_1^{(\text{tr})}$ and $D_1^{(\text{val})}$ respectively. For the learner on CIFAR-100, the classification head is a 100-way linear classifier. The training and validation set of CIFAR-100 is used as $D_2^{(\text{tr})}$ and $D_2^{(\text{val})}$ respectively. We set the number of interleaving rounds to 2. The tradeoff parameter λ in Eq.(3.5) is set to 100. The order of tasks in the interleaving process is: CIFAR-100, CIFAR-10, CIFAR-100, CIFAR-10.

During architecture search, network weights were optimized using the SGD optimizer with a batch size of 64, an initial learning rate of 0.025, a learning rate scheduler of cosine decay, a weight decay of 3e-4, a momentum of 0.9, and an epoch number of 50. The architecture variables were optimized using the Adam [47] optimizer with a learning rate of 3e-4 and a weight decay of 1e-3. The rest hyperparameters follow those in DARTS, P-DARTS, and PC-DARTS.

Given the optimally searched architecture cell, we evaluate it individually on CIFAR-10, CIFAR-100, and ImageNet. For CIFAR-10 and CIFAR-100, we stack 20 copies of the searched cell into a larger network as the image encoder. The initial channel number was set to 36. We trained the network for 600 epochs on the combination of the training and validation datasets where the mini-batch size was set to 96. The experiments were conducted on one Tesla v100 GPU. For ImageNet, similar to [58], we evaluate the architecture cells searched on CIFAR10/100.

Method	Error(%)	Param(M)	Cost
*ResNet [36]	22.10	1.7	-
*DenseNet [45]	17.18	25.6	-
*PNAS [56]	19.53	3.2	150
*ENAS [76]	19.43	4.6	0.5
*AmoebaNet [78]	18.93	3.1	3150
†DARTS-1st [58]	20.52±0.31	1.8	0.4
*GDAS [25]	18.38	3.4	0.2
*R-DARTS [100]	18.01±0.26	-	1.6
*DARTS$^-$ [18]	17.51±0.25	3.3	0.4
†DARTS$^-$ [18]	18.97±0.16	3.1	0.4
$^\Delta$DARTS$^+$ [53]	17.11±0.43	3.8	0.2
*DropNAS [42]	16.39	4.4	0.7
*DARTS-2nd [58]	20.58±0.44	1.8	1.5
MTL-DARTS2nd	18.92±0.17	2.4	3.1
IL-DARTS2nd (ours)	**17.12**±0.08	2.6	3.2
*P-DARTS [16]	17.49	3.6	0.3
MTL-PDARTS	17.67±0.31	3.5	0.6
IL-PDARTS (ours)	**16.14**±0.17	3.6	0.6
†PC-DARTS [96]	17.96±0.15	3.9	0.1
MTL-PCDARTS	18.11±0.27	3.9	0.2
IL-PCDARTS (ours)	17.83±0.14	3.8	0.3

Table 3.2: Classification errors on the test set of CIFAR-100, number of model parameters, and search cost (GPU days). IL-DARTS-2nd denotes that our proposed interleaving learning (IL) framework is applied to the search space of DARTS-2nd. DARTS-1st and DARTS-2nd means that first order and second order approximation is used in DARTS' optimization procedure. Results marked with * are taken from DARTS$^-$ [18]. Methods marked with † were re-run for 10 times with different random initialization. Δ denotes this algorithm ran for 600 epochs instead of 2000 epochs in the architecture evaluation stage, to ensure the comparison with other methods (which all ran for 600 epochs) is fair. Search cost is measured by GPU days on a Tesla v100.

A larger network is formed by stacking 14 copies of the searched cell. The initial channel number was set to 48. We trained the network for 250 epochs on the 1.2M training images using eight Tesla v100 GPUs where the batch size was set to 1024. Each IL experiment was repeated for ten times with different random initialization. Mean and standard deviation of classification errors obtained from the 10 runs are reported.

3.4.4 Results

Table 3.2 and Table 3.3 show the classification errors on the test sets of CIFAR-100 and CIFAR-10 respectively, together with the number of model parameters and search costs (GPU days) of different NAS methods. From these two tables, we make the following observations. **First**, when our proposed interleaving learning (IL) framework is applied to different differentiable

Method	Error(%)	Param(M)	Cost
*DenseNet [45]	3.46	25.6	-
*HierEvol [57]	3.75±0.12	15.7	300
*NAONet-WS [62]	3.53	3.1	0.4
*PNAS [56]	3.41±0.09	3.2	225
*ENAS [76]	2.89	4.6	0.5
*NASNet-A [106]	2.65	3.3	1800
*AmoebaNet-B [78]	2.55±0.05	2.8	3150
*DARTS-1st [58]	3.00±0.14	3.3	0.4
*R-DARTS [100]	2.95±0.21	-	1.6
*GDAS [25]	2.93	3.4	0.2
*SNAS [95]	2.85	2.8	1.5
$^{\Delta}$DARTS^{+} [53]	2.83±0.05	3.7	0.4
*BayesNAS [103]	2.81±0.04	3.4	0.2
*MergeNAS [93]	2.73±0.02	2.9	0.2
*NoisyDARTS [19]	2.70±0.23	3.3	0.4
*ASAP [69]	2.68±0.11	2.5	0.2
*SDARTS [14]	2.61±0.02	3.3	1.3
*DARTS^{-} [18]	2.59±0.08	3.5	0.4
†DARTS^{-} [18]	2.97±0.04	3.3	0.4
*DropNAS [42]	2.58±0.14	4.1	0.6
*FairDARTS [17]	2.54	3.3	0.4
*DrNAS [15]	2.54±0.03	4.0	0.4
*DARTS-2nd [58]	2.76±0.09	3.3	1.5
MTL-DARTS2nd	2.91±0.12	2.4	3.1
IL-DARTS2nd (ours)	**2.62**±0.04	2.6	3.2
*PC-DARTS [96]	2.57±0.07	3.6	0.1
MTL-PCDARTS	2.63±0.05	3.9	0.2
IL-PCDARTS (ours)	2.55±0.11	3.8	0.3
*P-DARTS [16]	2.50	3.4	0.3
MTL-PDARTS	2.63±0.12	3.5	0.6
IL-PDARTS (ours)	2.51±0.10	3.6	0.6

Table 3.3: Classification errors on the test set of CIFAR-10, number of model parameters, and search cost. Results marked with * are taken from DARTS^{-} [18], NoisyDARTS [19], and DrNAS [15]. The rest notations are the same as those in Table 3.2.

NAS methods, the errors of these methods can be greatly reduced. For example, on CIFAR-100, IL-DARTS2nd (applying IL to DARTS) achieves an average error of 17.12%, which is significantly lower than the error of vanilla DARTS-2nd, which is 20.58%. As another example, the error of P-DARTS on CIFAR-100 is 17.49%; applying IL to P-DARTS, this error is reduced to 16.14%. On CIFAR-10, applying IL to DARTS-2nd reduces the error from 2.76% to 2.62%. These results demonstrate the effectiveness of interleaving learning. In IL, the encoder trained on CIFAR-100 is used to initialize the encoder for CIFAR-10. Likewise, the encoder trained on CIFAR-10 is used to help with the learning of the encoder on CIFAR-100. These two procedures iterates, which enables the learning tasks on CIFAR-100 and CIFAR-10 to mutually benefit each other. In contrast, in baselines including DARTS-2nd, P-DARTS, and PC-DARTS, the encoders for CIFAR-100 and CIFAR-10 are learned separately without interleaving; there is no mechanism to let the learning on CIFAR-100 benefit the learning on CIFAR-10 and vice versa.

Method	Top-1 Error (%)	Top-5 Error (%)	Param (M)	Cost (GPU days)
*Inception-v1 [88]	30.2	10.1	6.6	-
*MobileNet [43]	29.4	10.5	4.2	-
*ShuffleNet 2× (v1) [101]	26.4	10.2	5.4	-
*ShuffleNet 2× (v2) [63]	25.1	7.6	7.4	-
*NASNet-A [106]	26.0	8.4	5.3	1800
*PNAS [56]	25.8	8.1	5.1	225
*MnasNet-92 [89]	25.2	8.0	4.4	1667
*AmoebaNet-C [78]	24.3	7.6	6.4	3150
*SNAS [95]	27.3	9.2	4.3	1.5
*BayesNAS [103]	26.5	8.9	3.9	0.2
*PARSEC [12]	26.0	8.4	5.6	1.0
*GDAS [25]	26.0	8.5	5.3	0.2
*DSNAS [44]	25.7	8.1	-	-
*SDARTS-ADV [14]	25.2	7.8	5.4	1.3
*PC-DARTS [96]	25.1	7.8	5.3	0.1
*ProxylessNAS [10]	24.9	7.5	7.1	8.3
*FairDARTS (CIFAR-10) [17]	24.9	7.5	4.8	0.4
*FairDARTS (ImageNet) [17]	24.4	7.4	4.3	3.0
*DrNAS [15]	24.2	7.3	5.2	3.9
*DARTS$^+$ (ImageNet) [53]	23.9	7.4	5.1	6.8
*DARTS$^-$ [18]	23.8	7.0	4.9	4.5
*DARTS$^+$ (CIFAR-100) [53]	23.7	7.2	5.1	0.2
*DARTS2nd-CIFAR10 [58]	26.7	8.7	4.7	1.5
MTL-DARTS2nd-CIFAR10/100	26.4	8.5	3.5	3.1
IL-DARTS2nd-CIFAR10/100 (ours)	**25.5**	**8.0**	3.8	3.2
*PDARTS-CIFAR10 [16]	24.4	7.4	4.9	0.3
*PDARTS-CIFAR100 [16]	24.7	7.5	5.1	0.3
MTL-PDARTS-CIFAR10/100	25.0	7.9	5.1	0.6
IL-PDARTS-CIFAR10/100 (ours)	**24.1**	**7.1**	5.3	0.6

Table 3.4: Top-1 and top-5 classification errors on the test set of ImageNet, number of model parameters, and search cost (GPU days). Results marked with * were taken from DARTS$^-$ [18] and DrNAS [15]. The rest notations are the same as those in Table 3.2. From top to bottom, on the first, second, and third block are: 1) networks manually designed by humans; 2) non-differentiable architecture search methods; and 3) differentiable search methods.

Overall, the improvement achieved by our method on CIFAR-100 is more significant than that on CIFAR-10. This is probably because CIFAR-10 is a relatively easy dataset for classification (with 10 classes only), which leaves smaller room for improvement. With 100 classes, CIFAR-100 is more challenging for classification and can better differentiate the capabilities of different methods. **Second**, interleaving learning (IL) performs better than multi-task learning (MTL). For example, on CIFAR-100, when applied to DARTS-2nd, the error of IL is lower than that of MTL; this is also the case when applied to P-DARTS and PC-DARTS. On CIFAR-10, when applied to DARTS-2nd, P-DARTS, and PC-DARTS, IL outperforms MTL as well. In the inner optimization problem of the MTL formulation, the encoder weights W_{100} for CIFAR-100 and the encoder weights W_{10} for CIFAR-10 are trained independently without a mechanism of mutually benefiting each other. In contrast, IL enables W_{100} and W_{10} to help each other for better training via the interleaving mechanism. **Third**, among all the methods in Table 3.2, our IL-PDARTS

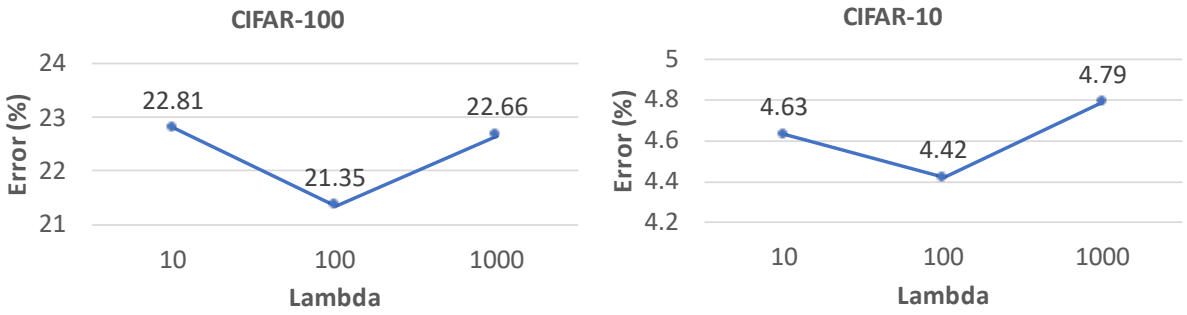

Figure 3.2: How classification errors on CIFAR-100 and CIFAR-10 change as λ increases.

method achieves the lowest error, which shows that our IL method is highly competitive in pushing the limit of the state-of-the-art. **Fourth**, while our IL method achieves better accuracy, it does not substantially increase model size (number of parameters) or search cost.

Table 3.4 shows the top-1 and top-5 classification errors on the test set of ImageNet, number of model parameters, and search cost (GPU days). Similar to the observations made from Table 3.2 and Table 3.3, the results on ImageNet show the following. **First**, when applying our IL framework to DARTS and P-DARTS, the errors of these methods can be greatly reduced. For example, IL-DARTS2nd-CIFAR10/100 (applying IL to DARTS-2nd and searching the architecture on CIFAR-10 and CIFAR-100) achieves a top-1 error of 25.5% and top-5 error of 8.0%; without IL, the top-1 and top-5 error of DARTS2nd-CIFAR10 is 26.7% and 8.7%. As another example, the errors achieved by IL-PDARTS-CIFAR10/100 are much lower than those of PDARTS-CIFAR100 and PDARTS-CIFAR10. These results further demonstrate the effectiveness of interleaving learning which enables different tasks to mutually help each other. **Second**, interleaving learning (IL) outperforms multi-task learning (MTL). For example, IL-DARTS2nd-CIFAR10/100 achieves lower errors than MTL-DARTS2nd-CIFAR10/100; IL-PDARTS-CIFAR10/100 performs better than MTL-PDARTS-CIFAR10/100. These results further show that making different tasks help each other in an interleaving and cyclic way is more advantageous than performing them jointly and simultaneously. **Third**, while our IL framework can greatly improve classification accuracy, it does not increase the parameter number and search cost substantially.

3.4.5 Ablation Studies

We perform ablation studies to check the effectiveness of individual modules in our framework. In each ablation study, the ablation setting is compared with the full interleaving learning framework.

- Ablation study on the tradeoff parameter λ. We explore how the learners' performance varies as the tradeoff parameter λ in Eq.(3.3) increases. For both CIFAR-100 and CIFAR-10, we randomly sample 5K data from the 25K training and 25K validation data, and use it as a test set to report performance in this ablation study. The rest 45K data (22.5K training data and 22.5K validation data) is used for architecture search and evaluation. IL is applied to DARTS-2nd. The number of rounds is set to 2.

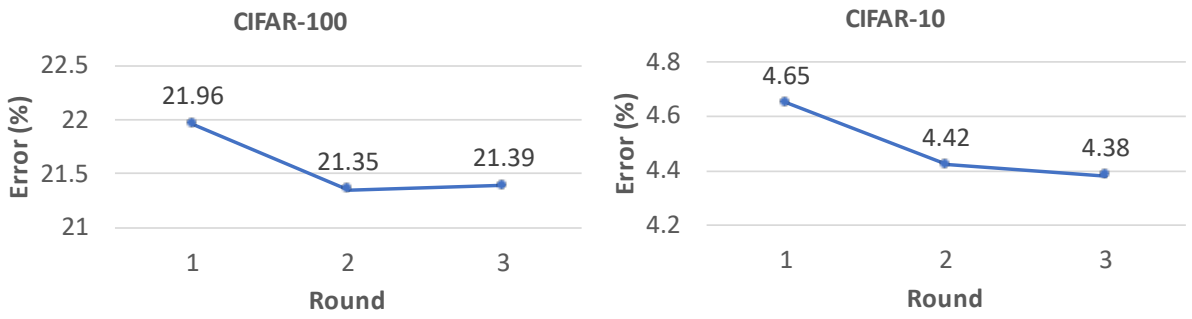

Figure 3.3: How classification errors on CIFAR-100 and CIFAR-10 change as the number of interleaving rounds M increases.

Method	Error (%)
Order 1 (CIFAR-100)	17.12±0.08
Order 2 (CIFAR-100)	17.19±0.14
Order 1 (CIFAR-10)	2.73±0.04
Order 2 (CIFAR-10)	2.79±0.11

Table 3.5: Results for ablation study on the order of tasks. "Order 1" denotes "CIFRA-100, CIFAR-10, CIFAR-100, CIFAR-10". "Order 2" denotes "CIFRA-10, CIFAR-100, CIFAR-10, CIFAR-100".

- Ablation study on the number of rounds. In this study, we explore how the test error changes as we increase the number of interleaving rounds M from 1 to 3. The results are reported on the 5K sampled data. In this experiment, the tradeoff parameter λ is set to 100. IL is applied to DRATS-2nd.
- Ablation study on the order of tasks. In this study, we explore whether the order of tasks affects the test error. We experimented two orders (with the number of rounds set to 2): 1) CIFAR-100, CIFAR-10, CIFAR-100, CIFAR-10; 2) CIFAR-10, CIFAR-100, CIFAR-10, CIFAR-100. In order 1, classification on CIFAR-100 is performed first; in order 2, classification on CIFAR-10 is performed first. In this experiment, the tradeoff parameter λ is set to 100.

Figure 7.3 shows how the classification errors on the test sets of CIFAR-100 and CIFAR-10 vary as the tradeoff parameter λ increases. As can be seen, for both datasets, when λ increases from 10 to 100, the errors decrease. A larger λ encourages a stronger knowledge transfer effect: the learning of the current learner C is sufficiently influenced by the previous learner P; the well-trained data encoder of P can effectively help to train the encoder of C, which results in better classification performance. However, further increasing λ renders the errors to increase. This is because an excessively large λ will make the encoder of C strongly biased to the encoder of P while ignoring the specific data patterns in C's own training data. Since P's encoder may not be suitable for representing C's data, such a bias leads to inferior classification performance.

Figure 3.3 shows how the classification errors on the test sets of CIFAR-100 and CIFAR-10 vary as the number of rounds M increases. For CIFAR-100, when M increases from 1 to 2, the error is reduced. When $M = 1$, the interleaving effect is weak: classification on CIFAR-

100 influences classification on CIFAR-10, but not the other way around. When $M = 2$, the interleaving effect is strong: CIFAR-100 influences CIFAR-10 and CIFAR-10 in turn influences CIFAR-100. This further demonstrates the effectiveness of interleaving learning. Increasing M from 2 to 3 does not significantly reduce the error further. This is probably because 2 rounds of interleaving have brought in sufficient interleaving effect. Similar trend is observed in the plot of CIFAR-10.

Table 3.5 shows the test errors on CIFAR-100 and CIFAR-10 under two different orders. In order 1, the starting task is classification on CIFAR-100. In order 2, the starting task is classification on CIFAR-10. As can be seen, the errors are not affected by the task order significantly. The reason is that: via interleaving, each task influences the other task at some point in the interleaving sequence; therefore, it does not matter too much regarding which task should be performed first.

3.5 Conclusions

In this chapter, we propose a novel machine learning framework called interleaving learning (IL). In IL, multiple tasks are performed in an interleaving fashion where task 1 is performed for a short while, then task 2 is conducted, then task 3, etc. After all tasks are learned in one round, the learning goes back to task 1 and the cyclic procedure starts over. These tasks share a data encoder, whose network weights are trained successively by different tasks in the interleaving process. Via interleaving, different models transfer their learned knowledge to each other to better represent data and avoid being stuck in bad local optimums. We propose a multi-level optimization framework to formulate interleaving learning, where different learning stages are performed end-to-end. An efficient gradient-based algorithm is developed to solve the multi-level optimization problem. Experiments of neural architecture search on CIFAR-100 and CIFAR-10 demonstrate the effectiveness of interleaving learning.

Chapter 4

Learning by Self-Explanation

4.1 Introduction

In humans' learning practice, a broadly adopted learning skill is self-explanation where students explain to themselves a learned topic to achieve better understanding of this topic. Self-explanation encourages a student to actively digest and integrate prior knowledge and new information, which helps to fill in the gaps in understanding a topic. It has shown considerable effectiveness in improving learning outcomes.

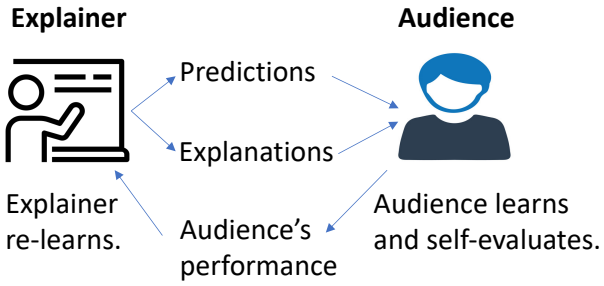

Figure 4.1: Illustration of learning by self-explanation. An explainer explains to the audience regarding how predictions are made. The audience leverages the explanations to learn and evaluates itself. Performance of the audience acts as feedback to guide the explainer to re-learn for giving better explanations. This process repeats until the audience's performance is good enough.

We are interested in asking: can the explanation-driven learning method be borrowed from humans to help machines to learn better? Motivated by this inspiration, we propose a novel machine learning framework called learning by self-explanation (LeaSE) (as illustrated in Figure 7.1). In this framework, there is an explainer model and an audience model. They both learn to undertake the same ML task, such as image classification. The explainer has a learnable architecture and a set of learnable network weights. The audience has a predefined architecture by human experts and a set of learnable network weights. The goal is to help the explainer to learn well on the target task. The way to achieve this goal is to encourage the explainer to give clear explanations to the audience regarding how predictions are made. Intuitively, if a model

can explain prediction outcomes well, it must have a deep understanding of the prediction task and can learn better based on this understanding. The learning is organized into four stages. In the first stage, the explainer trains its network weights by minimizing the prediction loss on its training dataset, with its architecture fixed. In the second stage, the explainer uses its model trained in the first stage to make predictions on the training data examples of the audience and leverages an adversarial attack [27, 34] approach to explain the prediction outcomes. In the third stage, the audience model combines its training examples and the explainer-made explanations of prediction outcomes on these examples to train its network weights. In the fourth stage, the explainer updates its neural architecture by minimizing its validation loss and the audience's validation loss. The fours stages are synthesized into a unified four-level optimization framework where they are performed jointly in an end-to-end manner. Each learning stage has an influence on other stages. We apply our method for neural architecture search in image classification tasks. Our method achieves significant improvement on CIFAR-100, CIFAR-10, and ImageNet [22].

The major contributions of this section are as follows:

- Drawing inspiration from the explanation-driven learning technique of humans, we propose a novel machine learning approach called learning by self-explanation (LeaSE). In our approach, an explainer model improves its learning ability by trying to clearly explain to an audience model regarding how the prediction outcomes are made.
- We develop a multi-level optimization framework to formulate LeaSE which involves four stages of learning: explainer learns; explainer explains; audience learns; explainer relearns based on the audience's performance.
- We develop an efficient algorithm to solve the LeaSE problem.
- We apply LeaSE for neural architecture search on CIFAR-100, CIFAR-10, and ImageNet, where the results demonstrate the effectiveness of our method.

4.2 Related Works

4.2.1 Neural Architecture Search (NAS)

In the past few years, a wide variety of NAS methods have been proposed and achieved considerable success in automatically identifying highly-performing architectures of neural networks for the sake of reducing the reliance on human experts. Early NAS approaches [76, 105, 106] are mostly based on reinforcement learning (RL) which use a policy network to generate architectures and evaluate these architectures on validation set. The validation loss is used as a reward to optimize the policy network and train it to produce high-quality architectures. While RL-based approaches achieve the first wave of success in NAS research, they are computationally very expensive since evaluating the architectures requires a heavy-duty training process. This limitation renders RL-based approaches not applicable for most users who do not have enough computational resources. To address this issue, differentiable search methods [10, 58, 95] have been proposed, which parameterize architectures as differentiable functions and perform search using efficient gradient-based methods. In these methods, the search space of architectures is composed of a large set of building blocks where the output of each block is multiplied with a

smooth variable indicating how important this block is. Under such a formulation, search becomes solving a mathematical optimization problem defined on the importance variables where the objective is to find out an optimal set of variables that yield the lowest validation loss. This optimization problem can be solved efficiently using gradient-based methods. Differentiable NAS research is initiated by DARTS [58] and further improved by subsequent works such as P-DARTS [16], PC-DARTS [96], etc. P-DARTS [16] grows the depth of architectures progressively in the search process. PC-DARTS [96] samples sub-architectures from a super network to reduce redundancy during search. Our proposed LeaSE framework is orthogonal to existing NAS methods and can be used in combination with any differentiable NAS method to further improve these methods. Such et al. [87] proposed to learn a generative model to generate synthetic examples which are used to search the architecture of an auxiliary model. Our work differs from this one in that: 1) we focus on searching the architecture of a primary model (the explainer) by letting it explain to an auxiliary model (the audience) while [87] focuses on searching the architecture of the auxiliary model; 2) our primary model produces explanations via adversarial attack while the generative model in [87] generates synthetic examples. Besides RL-based approaches and differentiable NAS, another paradigm of NAS methods [57, 78] are based on the evolutionary algorithm. In these methods, architectures are formulated as individuals in a population. High-quality architectures produce offspring to replace low-quality architectures, where the quality is measured using fitness scores. Similar to RL-based approaches, these methods also require considerable computing resources.

4.2.2 Interpretable Machine Learning

The explainability and transparency of machine learning models is crucial for mission-critical applications. Many approaches have been proposed for understanding the predictions made by black-box models. Many prior approaches for interpretable ML focus on finding out key evidence from the input data (such as phrases in texts and regions in images) that is most relevant to a prediction, then using these evidence to justify the meaningfulness of the prediction. In [99] and [104], the authors perform perturbation on the input data elements (e.g., pixels) and check which perturbed elements cause more changes of the output. Such elements are considered to be more relevant to the output and are used as explanations. In [3, 85] and [86], the authors identify the contribution of each input feature to the output prediction by propagating the contribution through layers of a deep neural network. Another body of works [51, 67, 97] are based on the idea of attention. While training the prediction model, an attention network is trained to calculate an attention score for each input data element. Elements with large attention scores are used as explanations. In [97], attention mechanisms are used to select important words and sentences for interpreting hierarchical recurrent networks. Mullenbach et al. [67] leverage attention networks for explaining convolution networks. In addition, another widely adopted idea is to use simple but more interpretable models to interpret expressive black-box models. For example, in LIME [80], an interpretable linear model is used to approximate the decision boundary of a black-box model at an interested instance and interpretation is performed by checking dominant features in the linear model. Our work differs from these existing works in that: existing works focus on interpreting a trained model while our method focuses on improving the training of a model by letting it explain. In other words, the goal of existing works is explaining while that of

Notation	Meaning
A	Architecture of the explainer
E	Network weights of the explainer
W	Network weights of the audience
δ	Explanations
$D_e^{(\text{tr})}$	Training data of the explainer
$D_a^{(\text{tr})}$	Training data of the audience
$D_e^{(\text{val})}$	Validation data of the explainer
$D_a^{(\text{val})}$	Validation data of the audience

Table 4.1: Notations in Learning by Self-Explanation

our work is learning. The interpretation module (in the second stage) in our framework can be any model-interpretation method.

The concept of self-explanation was investigated in [26], which calculates confidence levels for decisions and explanations based on mutual information. Different from our work, this work does not leverage self-explanation to improve model training. Alvarez-Melis and Jaakkola [1] proposed a self-explaining network (SEN) which simultaneously outputs predictions and explanations. Our work differs from this one in that: our work uses the explanations generated by model A to train model B where B's validation performance reflects how good A's explanations are; the training of A is continuously improved so that it can generate good explanations. Leveraging another model to evaluate the quality of explanations made by one model is more robust and overfitting-resilient. In contrast, SEN learns to make predictions and explains prediction results in a single model, where the explanations may not be able to generalize in other models. Explanation-based learning [21, 65] has been investigated in logic-based AI systems. These approaches require manual design of logic rules, which are not scalable.

4.3 Methods

In this section, we propose a four-level optimization framework to formulate learning by self-explanation (LeaSE) (as shown in Figure 7.2) and develop an optimization algorithm to solve the four-level optimization problem.

4.3.1 Learning by Self-Explanation

In the LeaSE framework, there is an explainer model and an audience model, both of which learn to perform the same target task. The primary goal of our framework is to help the explainer to learn the target task very well. The way to achieve this goal is to let the explainer make meaningful explanations of the prediction outcomes in the target task. The intuition behind LeaSE is: to correctly explain prediction results, a model needs to learn to understand the target task very well. The explainer has a learnable architecture A and a set of learnable network weights E. The audience has a pre-defined neural architecture (by human experts) and a set of

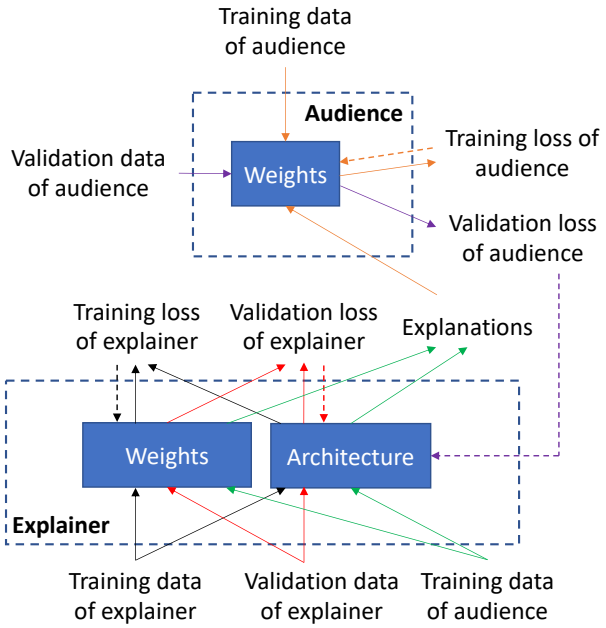

Figure 4.2: Learning by self-explanation. Following the solid arrows, predictions are made and training/validation losses are calculated. Following the dotted arrows, gradient updates of network weights and architecture variables are calculated and weights/architecture are updated.

learnable network weights W. The learning is organized into four stages. In the first stage, the explainer trains its network weights E on its training dataset $D_e^{(\text{tr})}$, with the architecture fixed:

$$E^*(A) = \min_E L(E, A, D_e^{(\text{tr})}). \tag{4.1}$$

To define the training loss, we need to use the architecture A together with network weights W to make predictions on training examples. However, A cannot be updated by minimizing the training loss. Otherwise, a trivial solution of A will be yielded: A is very large and complex that it can perfectly overfit the training data but will make largely incorrect predictions on novel data examples. Note that $E^*(A)$ is a function of A for that $L(E, A, D_e^{(\text{tr})})$ is a function of A and $E^*(A)$ depends on $L(E, A, D_e^{(\text{tr})})$. In the second stage, the explainer uses the trained model $E^*(A)$ to make predictions on the input training examples $D_a^{(\text{tr})}$ of the audience and explains the prediction outcomes. Specifically, given an input data example x (without loss of generality, we assume it is an image) and the predicted label y, the explainer aims to find out a subset of image patches P in x that are mostly correlated with y and uses P as explanations for y. We leverage an adversarial attack approach [27, 34] to achieve this goal. Adversarial attack adds small random perturbations δ to pixels in x so that the prediction outcome on the perturbed image $x + \delta$ is no longer y. Pixels that are perturbed more have higher correlations with the prediction outcome y and can be used as explanations. This process amounts to solving the following optimization problem:

$$\Delta^*(E^*(A)) = \max_\Delta \sum_{i=1}^{N} \ell(f(x_i + \delta_i; E^*(A)), f(x_i; E^*(A))) \tag{4.2}$$

where $\Delta = \{\delta_i\}_{i=1}^N$ and δ_i is the perturbation added to image x_i. $f(x_i + \delta_i; E^*(A))$ and $f(x_i; E^*(A))$ are the prediction outcomes of the explainer's network $f(\cdot; E^*(A))$ on $x_i + \delta_i$ and x_i. Without loss of generality, we assume the task is image classification (with K classes). Then $f(x_i + \delta_i; E^*(A))$ and $f(x_i; E^*(A))$ are both K-dimensional vectors containing prediction probabilities on the K classes. $\ell(\cdot, \cdot)$ is the cross-entropy loss with $\ell(\mathbf{a}, \mathbf{b}) = -\sum_{k=1}^K b_i \log a_i$. In this optimization problem, the explainer aims to find out perturbations for each image so that the predicted outcome on the perturbed image is largely different from that on the original image. The learned optimal perturbations are used as explanations and those with larger values indicate that the corresponding pixels are more important in decision-making. $\Delta^*(E^*(A))$ is a function of $E^*(A)$ since $\Delta^*(E^*(A))$ is a function of the objective in Eq.(4.2) and the objective is a function of $E^*(A)$. In the third stage, given the explanations $\Delta^*(E^*(A))$ made by the explainer, the audience leverages them to learn the target task. Since the perturbations indicate how important the input pixels are, the audience uses them to reweigh the pixels: $x \odot \delta$, where $\odot$ denotes element-wise multiplication. Pixels that are more important are given more weights. Then the audience trains its network weights on these weighted images:

$$W^*(\Delta^*(E^*(A))) = \min_W \sum_{i=1}^N \ell(f(\delta_i^*(E^*(A)) \odot x_i; W), t_i), \tag{4.3}$$

where $f(\delta_i^*(E^*(A)) \odot x_i; W)$ is the prediction outcome of the audience's network $f(\cdot; W)$ on the weighted image $\delta_i^*(E^*(A)) \odot x_i$ and t_i is the class label. $W^*(\Delta^*(E^*(A)))$ is a function of $\Delta^*(E^*(A))$ since $W^*(\Delta^*(E^*(A)))$ is a function of the objective in Eq.(4.3) and the objective is a function of $\Delta^*(E^*(A))$. In the fourth stage, the explainer validates its network weights $E^*(A)$ on its validation set $D_e^{(\text{val})}$ and the audience validates its network weights $W^*(\Delta^*(E^*(A)))$ on its validation set $D_a^{(\text{val})}$. The explainer optimizes its architecture by minimizing its validation loss and the audience's validation loss:

$$\min_A L(E^*(A), A, D_e^{(\text{val})}) + \gamma L(W^*(\Delta^*(E^*(A))), D_a^{(\text{val})}), \tag{4.4}$$

where γ is a tradeoff parameter.

We integrate the four stages in a unified four-level optimization framework and obtain the following formulation of LeaSE:

$$\begin{aligned}
\min_A \quad & L(E^*(A), A, D_e^{(\text{val})}) + \gamma L(W^*(\Delta^*(E^*(A))), D_a^{(\text{val})}) \\
s.t. \quad & W^*(\Delta^*(E^*(A))) = \min_W \sum_{i=1}^N \ell(f(\delta_i^*(E^*(A)) \odot x_i; W), t_i) \\
& \Delta^*(E^*(A)) = \max_\Delta \sum_{i=1}^N \ell(f(x_i + \delta_i; E^*(A)), f(x_i; E^*(A))) \\
& E^*(A) = \min_E L(E, A, D_e^{(\text{tr})}).
\end{aligned} \tag{4.5}$$

In this framework, there are four optimization problems, each corresponding to a learning stage. From bottom to up, the optimization problems correspond to learning stage 1, 2, 3, and 4 respectively. The first three optimization problems are nested on the constraint of the fourth optimization problem. These four stages are conducted end-to-end in this unified framework. The

solution $E^*(A)$ obtained in the first stage is used to perform explanation in the second stage. The explanations $\Delta^*(E^*(A))$ obtained in the second stage are used to train the model in the third stage. The solutions obtained in the first and third stage are used to make predictions on the fourth stage. The architecture A updated in the fourth stage changes the training loss in the first stage and consequently changes the solution $E^*(A)$, which subsequently changes $\Delta^*(E^*(A))$ and $W^*(\Delta^*(E^*(A)))$. Following [58], we perform differentiable search on A in a search space composed of candidate building blocks. Searching amounts to selecting a subset of candidate blocks by learning a selection variable for each block. The selection variables indicate the importance of individual blocks and are differentiable.

4.3.2 Optimization Algorithm

We develop an efficient algorithm to solve the LeaSE problem. Getting insights from [58], we calculate the gradient of $L(E, A, D_e^{(\text{tr})})$ w.r.t E and approximate $E^*(A)$ using one-step gradient descent update of E. We plug the approximation E' of $E^*(A)$ into $\sum_{i=1}^N \ell(f(x_i+\delta_i; E^*(A)), f(x_i; E^*(A)))$ and obtain an approximated objective denoted by O_Δ. Then we approximate $\Delta^*(E^*(A))$ using one-step gradient descent update of Δ based on the gradient of O_Δ. Next, we plug the approximation Δ' of $\Delta^*(E^*(A))$ into $\sum_{i=1}^N \ell(f(\delta_i^*(E^*(A)) \odot x_i; W), t_i)$ and get another approximated objective denoted by O_W. Then we approximate $W^*(\Delta^*(E^*(A)))$ using one-step gradient descent update of W based on the gradient of O_W. Finally, we plug the approximation E' of $E^*(A)$ and the approximation W' of $W^*(\Delta^*(E^*(A)))$ into $L(E^*(A), A, D_e^{(\text{val})}) + \gamma L(W^*(\Delta^*(E^*(A))), D_a^{(\text{val})})$ and get the third approximated objective denoted by O_A. A is updated by descending the gradient of O_A. In the sequel, we use $\nabla^2_{Y,X} f(X, Y)$ to denote $\frac{\partial f(X,Y)}{\partial X \partial Y}$.

First of all, we approximate $E^*(A)$ using

$$E' = E - \xi_e \nabla_E L(E, A, D_e^{(\text{tr})}) \tag{4.6}$$

where ξ_e is a learning rate. Plugging E' into $\sum_{i=1}^N \ell(f(x_i + \delta_i; E^*(A)), f(x_i; E^*(A)))$, we obtain an approximated objective $O_\Delta = \sum_{i=1}^N \ell(f(x_i + \delta_i; E'), f(x_i; E'))$. Then we approximate $\Delta^*(E^*(A))$ using one-step gradient descent update of Δ with respect to O_Δ:

$$\Delta' = \Delta - \xi_\Delta \nabla_\Delta (\sum_{i=1}^N \ell(f(x_i + \delta_i; E'), f(x_i; E'))). \tag{4.7}$$

Plugging Δ' into $\sum_{i=1}^N \ell(f(\delta_i^*(E^*(A)) \odot x_i; W), t_i)$, we obtain an approximated objective $O_W = \sum_{i=1}^N \ell(f(\delta_i' \odot x_i; W), t_i)$. Then we approximate $W^*(\Delta^*(E^*(A)))$ using one-step gradient descent update of W with respect to O_W:

$$W' = W - \xi_W \nabla_w (\sum_{i=1}^N \ell(f(\delta_i' \odot x_i; W), t_i)). \tag{4.8}$$

Finally, we plug E' and W' into $L(E^*(A), D_e^{(\text{val})}) + \gamma L(W^*(\Delta^*(E^*(A))), D_a^{(\text{val})})$ and get $O_A = L(E', D_e^{(\text{val})}) + \gamma L(W', D_a^{(\text{val})})$. We can update the explainer's architecture A by descending the

gradient of O_A w.r.t A:

$$A \leftarrow A - \eta(\nabla_A L(E', D_e^{(\text{val})}) + \gamma \nabla_A L(W', D_a^{(\text{val})})) \tag{4.9}$$

where

$$\begin{aligned} &\nabla_A L(E', A, D_e^{(\text{val})}) = \\ &\nabla_A L(E - \xi_e \nabla_E L(E, A, D_e^{(\text{tr})}), A, D_e^{(\text{val})}) = \\ &-\xi_e \nabla^2_{A,E} L(E, A, D_e^{(\text{tr})}) \nabla_{E'} L(E', A, D_e^{(\text{val})}) + \nabla_A L(E', A, D_e^{(\text{val})}) \end{aligned} \tag{4.10}$$

The first term in the third line involves expensive matrix-vector product, whose computational complexity can be reduced by a finite difference approximation:

$$\nabla^2_{A,E} L(E, A, D_e^{(\text{tr})}) \nabla_{E'} L(E', A, D_e^{(\text{val})}) \approx \tfrac{1}{2\alpha}(\nabla_A L(E^+, A, D_e^{(\text{tr})}) - \nabla_A L(E^-, A, D_e^{(\text{tr})})), \tag{4.11}$$

where $E^{\pm} = E \pm \alpha \nabla_{E'} L(E', A, D_e^{(\text{val})})$ and α is a small scalar that equals $0.01/\|\nabla_{E'} L(E', A, D_e^{(\text{val})})\|_2$.

For $\nabla_A L(W', D_a^{(\text{val})})$ in Eq.(7.10), it can be calculated as $\frac{\partial E'}{\partial A} \frac{\partial \Delta'}{\partial E'} \frac{\partial W'}{\partial \Delta'} \nabla_{W'} L(W', D_a^{(\text{val})})$ according to the chain rule, where

$$\frac{\partial W'}{\partial \Delta'} = \frac{\partial (W - \xi_w \nabla_W (\sum_{i=1}^N \ell(f(\delta_i' \odot x_i; W), t_i)))}{\partial \Delta'} \tag{4.12}$$

$$= -\xi_W \nabla^2_{\Delta', W} (\sum_{i=1}^N \ell(f(\delta_i' \odot x_i; W), t_i)), \tag{4.13}$$

$$\frac{\partial \Delta'}{\partial E'} = \frac{\partial (\Delta - \xi_\Delta \nabla_\Delta (\sum_{i=1}^N \ell(f(x_i + \delta_i; E'), f(x_i; E'))))}{\partial E'} \tag{4.14}$$

$$= -\xi_\Delta \nabla^2_{E', \Delta} (\sum_{i=1}^N \ell(f(x_i + \delta_i; E'), f(x_i; E'))), \tag{4.15}$$

and

$$\frac{\partial E'}{\partial A} = \frac{\partial (E - \xi_e \nabla_E L(E, A, D_e^{(\text{tr})}))}{\partial A} \tag{4.16}$$

$$= -\xi_e \nabla^2_{A,E} L(E, A, D_e^{(\text{tr})}). \tag{4.17}$$

This algorithm is summarized in Algorithm 7.

4.4 Experiments

In the experiments, we apply our proposed LeaSE framework to perform neural architecture search for image classification. Each experiment consists of two phrases: architecture search and evaluation. In the search phrase, an optimal cell is identified. In the evaluation phrase, multiple copies of the optimal cell are stacked into a larger network, which is retrained from scratch. Please refer to the supplements for more hyperparameter settings, additional results, and significant tests of results.

Algorithm 4: Optimization algorithm for learning by self-explanation

while *not converged* **do**
1. Update the explainer's weights E
2. Update the explanations Δ
2. Update the audience's weights W
3. Update the explainer's architecture

end

4.4.1 Datasets

The experiments are performed on three datasets, including CIFAR-10, CIFAR-100, and ImageNet [22]. Both CIFAR-10 and CIFAR-100 contain 60K images from 10 classes (each class has the same number of images). For each of them, we split it into a training set with 25K images, a validation set with 25K images, and a test set with 10K images. During architecture search in LeaSE, the training set is used as $D_e^{(\text{tr})}$ and $D_a^{(\text{tr})}$ and the validation set is used as $D_e^{(\text{val})}$ and $D_a^{(\text{val})}$. During architecture evaluation, the composed large network is trained on the combination of $D_e^{(\text{tr})}$ and $D_a^{(\text{tr})}$. ImageNet contains 1.2M training images and 50K test images, coming from 1000 objective classes. Performing architecture search on the 1.2M images is computationally too costly. To address this issue, following [96], we randomly sample 10% images from the 1.2M images to form a new training set and another 2.5% images to form a validation set, then perform search on them. During architecture evaluation, the composed large network is trained on the entire set of 1.2M images.

4.4.2 Experimental Settings

Our framework is orthogonal to existing NAS approaches and can be applied to any differentiable NAS method. In the experiments, LeaSE was applied to DARTS [58], P-DARTS [16], and PC-DARTS [96]. The search spaces of these methods are composed of (dilated) separable convolutions with sizes of 3×3 and 5×5, max pooling with size of 3×3, average pooling with size of 3×3, identity, and zero. Each LeaSE experiment was repeated for ten times with different random seeds. The mean and standard deviation of classification errors obtained from the 10 runs are reported.

During architecture search, for CIFAR-10 and CIFAR-100, the architecture of the explainer is a stack of 8 cells. Each cell consists of 7 nodes. We set the initial channel number to 16. For the architecture of the audience model, we experimented with ResNet-18 and ResNet-50 [37]. We set the tradeoff parameter γ to 1. The search algorithm was based on SGD, with a batch size of 64, an initial learning rate of 0.025 (reduced in later epochs using a cosine decay scheduler), an epoch number of 50, a weight decay of 3e-4, and a momentum of 0.9. The rest of hyperparameters mostly follow those in DARTS, P-DARTS, and PC-DARTS.

During architecture evaluation, for CIFAR-10 and CIFAR-100, a larger network of the explainer is formed by stacking 20 copies of the searched cell. The initial channel number was set to 36. We trained the network with a batch size of 96, an epoch number of 600, on a single Tesla v100 GPU. On ImageNet, we evaluate two types of architectures: 1) those searched on a

Method	Error(%)	Param(M)	Cost
*ResNet [36]	22.10	1.7	-
*DenseNet [45]	17.18	25.6	-
*PNAS [56]	19.53	3.2	150
*ENAS [76]	19.43	4.6	0.5
*AmoebaNet [78]	18.93	3.1	3150
*GDAS [25]	18.38	3.4	0.2
*R-DARTS [100]	18.01±0.26	-	1.6
*DARTS$^-$ [18]	17.51±0.25	3.3	0.4
†DARTS$^-$ [18]	18.97±0.16	3.1	0.4
$^\Delta$DARTS$^+$ [53]	17.11±0.43	3.8	0.2
*DropNAS [42]	16.39	4.4	0.7
†DARTS-1st [58]	20.52±0.31	3.5	1.0
LeaSE-RN18-DARTS1st (ours)	17.04±0.10	3.6	1.1
LeaSE-RN50-DARTS1st (ours)	**16.87**±0.08	3.6	1.2
*DARTS-2nd [58]	20.58±0.44	3.5	1.5
LeaSE-RN18-DARTS2nd (ours)	16.80±0.17	3.7	1.7
LeaSE-RN50-DARTS2nd (ours)	**16.39**±0.07	3.6	1.9
†PC-DARTS [96]	17.96±0.15	3.9	0.1
LeaSE-RN18-PCDARTS (ours)	16.39±0.21	4.0	0.5
LeaSE-RN50-PCDARTS (ours)	**16.17**±0.05	3.9	0.7
*P-DARTS [16]	17.49	3.6	0.3
LeaSE-RN18-PDARTS (ours)	15.23±0.11	3.7	0.8
LeaSE-RN50-PDARTS (ours)	**15.13**±0.07	3.6	1.0

Table 4.2: Test error on CIFAR-100, number of model weights (millions), and search cost (GPU days on a Tesla v100). DARTS-1st and DARTS-2nd represents that first-order and second-order approximation is used in DARTS' optimization algorithm. LeaSE-R18-DARTS1st represents that the manually-designed architecture in the audience model is ResNet-18 and the search space is the same as that in DARTS-1st. Similar meanings hold for other notations in such a format. R50 denotes ResNet-50. Results marked with * are obtained from DARTS$^-$ [18]. Methods marked with † were re-run for 10 times. For DARTS$^+$ marked with Δ, we ran it for 600 epochs instead of 2000 epochs (used in [53]) in the architecture evaluation stage, to ensure the comparison with other methods (running 600 epochs) is fair.

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 using eight Tesla v100 GPUs on the 1.2M training images, with a batch size of 1024 and an epoch number of 250. Initial channel number was set to 48.

4.4.3 Results

Table 6.2 shows the results on CIFAR-100, including classification errors on the test set, number of model parameters, and search cost. By comparing different methods, we make the following observations. **First**, applying LeaSE to different NAS methods, including DARTS, P-DARTS, and PC-DARTS, the classification errors of these methods are greatly reduced. For example, the original error of DARTS-2nd is 20.58%; when LeaSE is applied, this error is significantly re-

Method	Error(%)	Param(M)	Cost
*DenseNet [45]	3.46	25.6	-
*HierEvol [57]	3.75±0.12	15.7	300
*NAONet-WS [62]	3.53	3.1	0.4
*PNAS [56]	3.41±0.09	3.2	225
*ENAS [76]	2.89	4.6	0.5
*NASNet-A [106]	2.65	3.3	1800
*AmoebaNet-B [78]	2.55±0.05	2.8	3150
*R-DARTS [100]	2.95±0.21	-	1.6
*GDAS [25]	2.93	3.4	0.2
*GTN [87]	2.92±0.06	8.2	0.67
*SNAS [95]	2.85	2.8	1.5
$^{\Delta}$DARTS^{+} [53]	2.83±0.05	3.7	0.4
*BayesNAS [103]	2.81±0.04	3.4	0.2
*MergeNAS [93]	2.73±0.02	2.9	0.2
*NoisyDARTS [19]	2.70±0.23	3.3	0.4
*ASAP [69]	2.68±0.11	2.5	0.2
*SDARTS [14]	2.61±0.02	3.3	1.3
*DARTS^{-} [18]	2.59±0.08	3.5	0.4
†DARTS^{-} [18]	2.97±0.04	3.3	0.4
*DropNAS [42]	2.58±0.14	4.1	0.6
*PC-DARTS [96]	2.57±0.07	3.6	0.1
*FairDARTS [17]	2.54	3.3	0.4
*DrNAS [15]	2.54±0.03	4.0	0.4
*DARTS-1st [58]	3.00±0.14	3.3	0.4
LeaSE-R18-DARTS1st (ours)	2.85±0.09	3.4	0.6
LeaSE-R50-DARTS1st (ours)	**2.76**±0.03	3.3	0.7
*DARTS-2nd [58]	2.76±0.09	3.3	1.5
LeaSE-R18-DARTS2nd (ours)	2.59±0.06	3.3	1.5
LeaSE-R50-DARTS2nd (ours)	**2.52**±0.04	3.4	1.7
*PC-DARTS [96]	2.57±0.07	3.6	0.1
LeaSE-R18-PC-DARTS (ours)	2.50±0.04	3.7	0.4
LeaSE-R50-PC-DARTS (ours)	**2.48**±0.02	3.7	0.5
*P-DARTS [16]	2.50	3.4	0.3
LeaSE-R18-PDARTS (ours)	2.45±0.03	3.4	0.8
LeaSE-R50-PDARTS (ours)	**2.44**±0.03	3.4	1.0

Table 4.3: Test error on CIFAR-10, number of model weights (millions), and search cost (GPU days on a Tesla v100). Results marked with * are obtained from DARTS^{-} [18], NoisyDARTS [19], and DrNAS [15]. The rest notations are the same as those in Table 6.2.

duced to 16.39%. As another example, after applying LeaSE to PC-DARTS, the error is reduced from 17.96% to 16.17%. Similarly, with the help of LeaSE, the error of P-DARTS is decreased from 17.49% to 15.13%. These results strongly demonstrate the broad effectiveness of our framework in searching better neural architectures. The reason behind this is: in our framework, the explanations made by the explainer are used to train the audience model; the validation performance of the audience reflects how good the explanations are; to make good explanations, the explainer's model has to be trained well; driven by the goal of helping the audience to learn well, the explainer continuously improves the training of itself. Such an explanation-driven learning mechanism is lacking in baseline methods, which are hence inferior to our method. **Second**, an audience model with a more expressive architecture can help the explainer to learn better. We experimented with two architectures for the audience model: ResNet with 18 layers (RN18) and ResNet with 50 layers (RN50), where RN50 is more expressive than RN18 since it has more layers. As can be seen, in LeaSE applied to DARTS, PC-DARTS, and P-DARTS, using RN50 as the

Method	Top-1 Error (%)	Top-5 Error (%)	Param (M)	Cost (GPU days)
*Inception-v1 [88]	30.2	10.1	6.6	-
*MobileNet [43]	29.4	10.5	4.2	-
*ShuffleNet 2× (v1) [101]	26.4	10.2	5.4	-
*ShuffleNet 2× (v2) [63]	25.1	7.6	7.4	-
*NASNet-A [106]	26.0	8.4	5.3	1800
*PNAS [56]	25.8	8.1	5.1	225
*MnasNet-92 [89]	25.2	8.0	4.4	1667
*AmoebaNet-C [78]	24.3	7.6	6.4	3150
*SNAS-CIFAR10 [95]	27.3	9.2	4.3	1.5
*BayesNAS-CIFAR10 [103]	26.5	8.9	3.9	0.2
*PARSEC-CIFAR10 [12]	26.0	8.4	5.6	1.0
*GDAS-CIFAR10 [25]	26.0	8.5	5.3	0.2
*DSNAS-ImageNet [44]	25.7	8.1	-	-
*SDARTS-ADV-CIFAR10 [14]	25.2	7.8	5.4	1.3
*PC-DARTS-CIFAR10 [96]	25.1	7.8	5.3	0.1
*ProxylessNAS-ImageNet [10]	24.9	7.5	7.1	8.3
*FairDARTS-CIFAR10 [17]	24.9	7.5	4.8	0.4
*FairDARTS-ImageNet [17]	24.4	7.4	4.3	3.0
*DrNAS-ImageNet [15]	24.2	7.3	5.2	3.9
*DARTS$^+$-ImageNet [53]	23.9	7.4	5.1	6.8
*DARTS$^-$-ImageNet [18]	23.8	7.0	4.9	4.5
*DARTS$^+$-CIFAR100 [53]	23.7	7.2	5.1	0.2
*DARTS2nd-CIFAR10 [58]	26.7	8.7	4.7	1.5
LeaSE-R18-DARTS2nd-CIFAR10 (ours)	**24.7**	**8.3**	4.8	1.5
*PDARTS (CIFAR10) [16]	24.4	7.4	4.9	0.3
LeaSE-R18-PDARTS-CIFAR10 (ours)	**23.8**	**6.7**	5.0	0.8
*PDARTS (CIFAR100) [16]	24.7	7.5	5.1	0.3
LeaSE-R18-PDARTS-CIFAR100 (ours)	**23.9**	**6.7**	5.1	0.8
*PCDARTS-ImageNet [96]	24.2	7.3	5.3	3.8
LeaSE-R18-PCDARTS-ImageNet (ours)	**22.1**	**6.0**	5.5	4.0

Table 4.4: Top-1 and top-5 classification errors on the test set of ImageNet, number of model parameters (millions) and search cost (GPU days). Results marked with * are obtained from DARTS$^-$ [18] and DrNAS [15]. The rest notations are the same as those in Table 6.2. From top to bottom, on the first three blocks are 1) networks manually designed by humans; 2) non-gradient based NAS methods; and 3) gradient-based NAS methods.

audience achieves better performance than using RN18. For example, LeaSE-R50-DARTS2nd achieves an error of 16.39%, which is lower than the 16.80% error of LeaSE-R18-DARTS2nd. When replacing the audience's architecture from RN18 to RN50, the error of LeaSE-DARTS1st is reduced from 17.04% to 16.87%, the error of LeaSE-PCDARTS is reduced from 16.39% to 16.17%, and the error of LeaSE-PDARTS is reduced from 15.23% to 15.13%. The reason is that to help a stronger audience to learn better, the explainer has to be even stronger. For a stronger audience model, it already has great capability in achieving excellent classification performance. To further improve this audience, the explanations used to train this audience need to be very sensible and informative. To generate such explanations, the explainer has to force itself to learn very well. **Third**, our LeaSE-RN50-PDARTS method achieves the lowest error among all methods listed in this table, which indicates that our method is very competitive in driving the field of NAS research to a new state-of-the-art. **Fourth**, the performance gain of our method does

not come at a cost of substantially increasing model size and search cost: the number of model parameters in architectures searched by our methods are at a similar level compared with those by other methods; so are the search costs.

In Table 6.3, we show the results on CIFAR-10, including classification errors on the test set, number of model parameters, and search cost. The observations made from this table are similar to those from Table 6.2. **First**, with the help of our LeaSE framework, the classification errors of DARTS, PC-DARTS, and P-DARTS are all reduced. For example, applying LeaSE to DARTS-2nd manages to reduce the error of DARTS-2nd from 2.76% to 2.52%. As another example, applying LeaSE to P-DARTS decreases the error from 2.50% to 2.44%. This further demonstrates the effectiveness of explanation-driven learning. **Second**, an audience with a stronger architecture helps the explainer to learn better. For example, in LeaSE-DARTS, LeaSE-PDARTS, and LeaSE-PCDARTS, when the audience is set to RN50, the performance is better, compared with setting the audience to RN18. **Third**, our method LeaSE-R50-PDARTS achieves the lowest error among all methods in this table, which further demonstrates its great potential in continuously pushing the limit of NAS research. **Fourth**, while the architectures searched by our framework yield better performance, their model size and search cost are not substantially increased compared with baselines.

In Table 6.4, we compare different methods on ImageNet, in terms of top-1 and top-5 classification errors on the test set, number of model parameters, and search cost. In experiments based on PC-DARTS, the architectures are searched on a subset of ImageNet. In other experiments, the architectures are searched on CIFAR-10 and CIFAR-100. LeaSE-R18-DARTS2nd-CIFAR10 denotes that LeaSE is applied to DARTS-2nd and performs search on CIFAR10, with the audience model set to ResNet-18. Similar meanings hold for other notations in such a format. The observations made from these results are consistent with those made from Table 6.2 and Table 6.3. The architectures searched using our methods are consistently better than those searched by corresponding baselines. For example, LeaSE-R18-DARTS2nd-CIFAR10 achieves lower top-1 and top-5 errors than DARTS2nd-CIFAR10. LeaSE-R18-PDARTS-CIFAR10 outperforms PDARTS-CIFAR10. These results again show that by explaining well, a model can gain better predictive performance. The model size and search cost of architectures searched by our methods are on par with those in other methods, which demonstrates that the performance gain of our framework is obtained without sacrificing compactness of architectures or computational efficiency substantially. Among all the methods in this table, our method LeaSE-R18-PCDARTS-ImageNet achieves the lowest top-1 and top-5 errors, which further demonstrates the great effectiveness of our method.

4.4.4 Ablation Studies

In this section, we perform ablation studies to investigate the importance of individual components in our framework. In each ablation study, we compare the ablation setting with the full framework. Specifically, we study the following ablation settings.

- **Ablation setting 1**. In this setting, the explainer updates its architecture by minimizing the validation loss of the audience only, without considering the validation loss of itself. The

Method	Error (%)
Audience only (CIFAR-100)	16.08±0.15
Audience + explainer (CIFAR-100)	**15.23**±0.11
Audience only (CIFAR-10)	2.72±0.07
Audience + explainer (CIFAR-10)	**2.59**±0.06

Table 4.5: Results for ablation setting 1. "Audience only" means that only the audience's validation loss is minimized to update the architecture of the explainer. "Audience + explainer" means that both the validation loss of the audience and the validation loss of the explainer itself are minimized to learn the explainer's architecture.

corresponding formulation is:

$$\begin{aligned}
&\min_{A} \quad L(W^*(\Delta^*(E^*(A))), D_a^{(\text{val})}) \\
&s.t.\ W^*(\Delta^*(E^*(A))) = \min_{W} \ \sum_{i=1}^{N} \ell(f(\delta_i^*(E^*(A)) \odot x_i; W), t_i) \\
&\quad \Delta^*(E^*(A)) = \max_{\Delta} \ \sum_{i=1}^{N} \ell(f(x_i + \delta_i; E^*(A)), f(x_i; E^*(A))) \\
&\quad E^*(A) = \min_{E} L(E, A, D_e^{(\text{tr})}).
\end{aligned}$$

During this study, we set the architecture of the audience to ResNet-18. On CIFAR-100, LeaSE is applied to P-DARTS. On CIFAR-10, LeaSE is applied to DARTS-2nd.

- **Ablation study on** γ. We investigate how the tradeoff parameter γ in Eq.(4.5) affects the classification errors of the explainer. For both CIFAR-100 and CIFAR-10, 5K images are uniformly sampled from the 50K training and validation examples. The 5K images are used as a test set for reporting the architecture evaluation performance, where the architecture is searched on the rest 45K images. We applied LeaSE to P-DARTS and chose ResNe-18 as the architecture of the audience's model.

In Table 6.5, we show the results on CIFAR-10 and CIFAR-100 under the ablation setting 1. On both datasets, "audience + explainer" where the validation losses of both the audience model and explainer itself are minimized to update the explainer's architecture works better than "audience only" where only the audience's validation loss is used to learn the architecture. Audience's validation loss reflects how good the explanations made by the explainer are. Explainer's validation loss reflects how strong the explainer's prediction ability is. Combining these two losses provides more useful feedback to the explainer than using one loss only, which hence can help the explainer to learn better.

In Figure 6.4, we show how LeaSE's classification errors on the test sets of CIFAR-10 and CIFAR-100 vary as we increase the tradeoff parameter γ. The curve on CIFAR-100 shows that the error decreases when we increase γ from 0.1 to 0.5. The reason is that a larger γ enables the audience to provide stronger feedback to the explainer regarding how good the explanations are. Such feedback can guide the explainer to refine its architecture for generating better explanations. However, if γ is further increased, the error becomes worse. Under such circumstances, too much emphasis is put on evaluating how good the explanations are and less attention is paid to

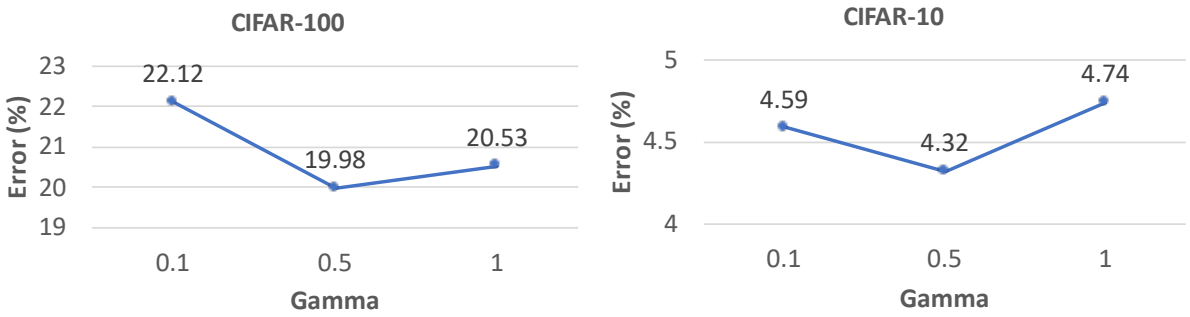

Figure 4.3: How errors change as γ increases.

the predictive ability of the explainer. The architecture is biased to generating good explanations with predictive performance compromised, which leads to inferior performance. A similar trend is shown in the curve on CIFAR-10.

4.5 Conclusions

Motivated by humans' explanation-driven learning skill, we develop a novel machine learning framework referred to as learning by self-explanation (LeaSE). In LeaSE, the primary goal is to help an explainer model learn how to well perform a target task. The way to achieve this goal is to let the explainer make sensible explanations. The intuition behind LeaSE is that a model has to learn to understand a topic very well before it can explain this topic clearly. A four-level optimization framework is developed to formalize LeaSE, where the learning is organized into four stages: the explainer learns a topic; the explainer explains this topic; the audience learns this topic based on the explanations given by the explainer; the explainer re-learns this topic based on the learning outcome of the audience. We apply LeaSE for neural architecture search on image classification datasets including CIFAR-100, CIFAR-10, and ImageNet. Experimental results strongly demonstrate the effectiveness of our proposed method.

Chapter 5

Small-Group Learning

5.1 Introduction

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

We are interested in asking whether this human learning skill can be translated into a machine learning skill to train better ML models. In a machine learning task (e.g., multi-class classification), it is typically the case that models with different architectures have complementary advantages. On a certain class c, model A (e.g., ResNet [37]) performs better than model B (e.g., DenseNet [45]). On another class d, model B performs better than model A. It is desirable to let these two models help each other during learning so that both of them eventually achieve great performance on both classes: specifically, model A helps model B to improve its performance on class c and model B helps model A to improve its performance on class d. Based on this idea, we propose a novel learning framework – small-group learning (SGL) (as illustrated in Figure 7.1). SGL involves a set of ML learners which solve the same learning task T. Without loss of generality, we assume T is classification while noting that our framework can be broadly applied to other tasks as well. Each learner k has a learnable neural architecture A_k and two sets of network weights W_k and V_k. The architectures and network weights of different learners are different. All learners share the same training dataset and validation dataset. These learners help each other to learn, in the following way: given an unlabeled dataset, each learner uses its intermediately trained model (including architecture and network weights) to generate pseudo-labels on this dataset and each learner leverages the pseudo-labeled datasets generated by other peers to re-train its model. On each class c, models performing well on c can generate high-quality pseudo-labeled datasets w.r.t c. Trained on these datasets, models that originally do not perform well on c can improve their performance on c. In this framework, there are three learning stages. In the first stage, each learner k trains its network weights V_k on the training dataset, with its architecture A_k fixed. In the second stage, each learner k uses its optimally trained V_k^* in the first stage to make predictions on an unlabeled dataset and generates a pseudo-labeled dataset;

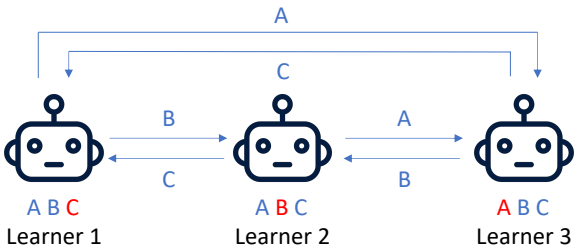

Figure 5.1: Illustration of small-group learning. There are three learners performing a classification task with three classes: A, B, and C. Blue letters denote classes that learners are good at (e.g., learner 1 performs well on class A and B) and red letters denote classes that learners are not good at. For each class, strong learners help weak learners to learn. For example, learner 2 and 3 are strong learners on class C; they help learner 1 (which is a weak learner on C) to improve the performance on C.

then each learner k trains its second set of network weights W_k using the pseudo-labeled datasets generated by other learners and using a human-labeled training dataset. In the third stage, each learner k updates its architecture A_k by minimizing prediction losses on the validation dataset. A multi-level optimization framework is developed to organize the three learning stages and enable them to be performed in a joint manner. Our method is applied for neural architecture search tasks on CIFAR-100, CIFAR-10, and ImageNet [22]. Experimental results demonstrate the broad effectiveness of SGL.

The major contributions of this chapter are:

- Drawing inspiration from the small-group learning (SGL) methodology in human learning, we develop a novel ML framework which formalizes SGL into a machine learning skill. In our framework, each learner uses its intermediately trained model to generate a pseudo-labeled dataset and re-trains its model using pseudo-labeled datasets generated by other learners.
- To formalize SGL, we develop a multi-level optimization (MLO) framework consisting of three learning stages: learners learn independently; learners learn collaboratively; learners validate themselves.
- An efficient optimization algorithm is developed to solve the MLO problem.
- We apply SGL for neural architecture search. The effectiveness of SGL is demonstrated by the experimental results on CIFAR-100, CIFAR-10, and ImageNet.

5.2 Related Works

Neural Architecture Search (NAS). NAS aims to automatically identify highly-performing architectures of deep neural networks instead of manually designing them by humans. Various approaches have been proposed for NAS, including differentiable search methods [10, 58, 95] and those based on reinforcement learning [76, 105, 106] and evolutionary algorithms [57, 78]. In RL-based approaches, a policy is learned to iteratively generate new architectures by max-

imizing a reward which is the accuracy on the validation set. Evolutionary algorithm based approaches represent architectures as individuals in a population. Individuals with high fitness scores (validation accuracy) have the privilege to generate offspring, which replaces individuals with low fitness scores. Differentiable approaches adopt a network pruning strategy. On top of an over-parameterized network, importance weights of building blocks are learned using gradient descent. After learning, blocks whose weights are close to zero are pruned. There have been many efforts devoted to improving differentiable NAS methods. In P-DARTS [16], the depth of searched architectures is allowed to grow progressively during the training process. Search space approximation and regularization approaches are developed to reduce computational overheads and improve search stability. PC-DARTS [96] reduces the redundancy in exploring the search space by sampling a small portion of a super network. Operation search is performed in a subset of channels with the held-out part bypassed in a shortcut. Our proposed SGL framework is applicable to any differentiable search approach.

Pseudo Labeling. Pseudo labeling has been broadly applied to different applications including knowledge distillation [40], adversarial robustness [11], self-supervised learning [94], etc. Zhang et al. [102] proposed a deep mutual learning (DML) approach, which applies pseudo labeling for mutually learning multiple learners. Our proposed method differs from this one in two aspects. First, our method proposes a multi-level optimization framework to perform pretraining, pseudo-labeling, and finetuning (based on pseudo-labels) jointly end-to-end, where the models are pretrained before they are used for generating pseudo labels. In contrast, DML directly uses untrained models to perform pseudo-labeling, which may incur collective failures as discussed in Section 5.4.4. Second, our method focuses on searching neural architectures while in DML the architectures are manually designed by humans. In several works [35, 52, 91], a trained teacher network (with a fixed architecture) is leveraged to generate pseudo-labels, which are used to search the architecture of a student network. Such et al. [87] proposed a meta-learning approach to learn a generative model which generates synthetic data and uses the generated data to search neural architectures. In these works, pseudo-labeling is unidirectional: a network A with a fixed architecture generates pseudo-labels which are used to search the architecture of B; B does not generate pseudo-labels to search the architecture of A. Different from these works, in our work pseudo-labeling is bi-directional: each model generates pseudo-labels which are used to search the architectures of other models; meanwhile, each model uses pseudo-labels generated by other models to search its own architecture.

Ensemble Learning. Our work is also related to ensemble learning, such as boosting [30], bagging [8], etc. In ensemble learning, a set of models are learned to perform the same task. During training, these models do not necessarily collaborate with each other. During testing, the trained models work together to make a single prediction. The individual models in ensemble learning are preferred to be different from each other so that their comparative advantages can be combined together when making predictions. Our work differs from ensemble learning in that the learners in our method help with each other during training; after training, different models achieve similar performance as a result of mutual teaching; during testing, only the single best model is retained for making predictions. Our method is more computationally efficient and

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

Table 5.1: Notations in small-group learning

memory efficient than ensemble learning methods during testing time since a single model is used for making predictions while ensemble learning utilizes all models to perform predictions.

Cooperative Learning. Previously, cooperative learning, where multiple models collaborate to perform a task, has been investigated in multi-agent learning [72], co-training [5], federated learning [6], etc. Our method differs from these existing methods in that we focus on collaborative neural architecture search via mutual pseudo-labeling.

5.3 Methods

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

5.3.1 Small-Group Learning

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

$$V_k^*(A_k) = \min_{V_k} L(V_k, A_k, D^{(\text{tr})}). \tag{5.1}$$

The architecture A_k is used to make predictions on training examples and define the training loss. However, A_k should not be optimized to minimize the training loss. Otherwise, the training loss can be easily minimized to zero by making A_k a very large (hence highly expressive) architecture, which will overfit the training data and have poor performance on unseen data. The optimally

trained weights $V_k^*(A_k)$ is a function of A_k since $V_k^*(A_k)$ is a function of $L(V_k, A_k, D^{(\text{tr})})$ and $L(V_k, A_k, D^{(\text{tr})})$ is a function of A_k.

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

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

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

$$\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K), A_k, D^{(\text{val})}). \tag{5.2}$$

Performed in a unified framework, the three stages mutually influence each other: $\{V_k^*(A_k)\}_{k=1}^K$ trained in the first stage are leveraged to calculate the training loss in the second stage; $\{W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K)\}_{k=1}^K$ trained in the second stage are leveraged to calculate the validation loss in the third stage; after $\{A_k\}_{k=1}^K$ are optimized in the third stage, they will render the training loss in the first stage to be changed, which changes $\{V_k^*(A_k)\}_{k=1}^K$ in the second stage accordingly. Figure 7.2 illustrates the three stages. To this end, we formulate SGL as a three-level optimization problem:

$$\begin{aligned}
&\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K), A_k, D^{(\text{val})}) \\
&s.t. \quad \{W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K)\}_{k=1}^K = \min_{\{W_k\}_{k=1}^K} \sum_{k=1}^K L(W_k, A_k, D^{(\text{tr})}) \\
&\qquad +\lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j^*(A_j))) \\
&\qquad \{V_k^*(A_k)\}_{k=1}^K = \min_{\{V_k\}_{k=1}^K} \sum_{k=1}^K L(V_k, A_k, D^{(\text{tr})})
\end{aligned} \tag{5.3}$$

This formulation consists of three optimization problems including two inner-optimization problems and one outer-optimization problem. The two inner-optimization problems are on the constraints of the outer optimization problem. From bottom to top, the three optimization problems

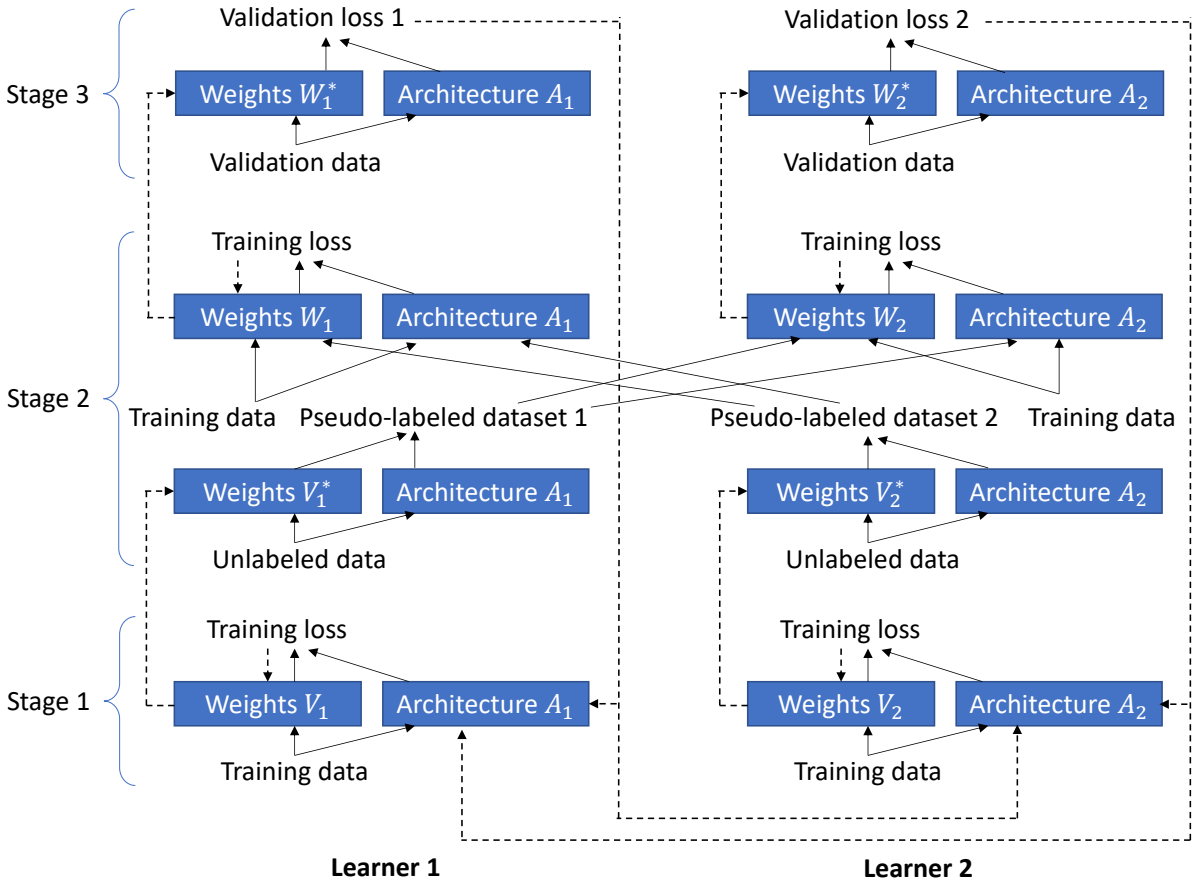

Figure 5.2: Three learning stages in small-group learning. Solid arrows denote the forward pass where predictions are made and losses are calculated. Dotted arrows represent the backward pass of calculating gradients of model architectures and weights and performing gradient-based updates. For simplicity, we assume there are two learners in the group. Extension to multiple learners is straightforward.

correspond to the first, second, and third learning stage respectively. Inspired by [58], we parameterize the architecture A using continuous variables which allow the search to be conducted using efficient gradient-based methods. The search space of A is overparameterized with many building blocks. For each block, a continuous variable is used to represent whether this block should be selected to form the final architecture. Search amounts to learning these variables.

5.3.2 Optimization Algorithm

To solve the SGL problem, we develop an efficient gradient-based optimization algorithm, drawing inspirations from [58]. In the following, $\nabla^2_{Y,X} f(X,Y)$ denotes $\frac{\partial f(X,Y)}{\partial X \partial Y}$. First of all, we approximate $V_k^*(A_k)$ using

$$V_k' = V_k - \xi_V \nabla_{V_k} L(V_k, A_k, D^{(\mathrm{tr})}), \tag{5.4}$$

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

$$W'_k = W_k - \xi_W \nabla_{W_k}(L(W_k, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V'_j))). \tag{5.5}$$

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

$$A_k \leftarrow A_k - \eta_A(\nabla_{A_k} L(W'_k, A_k, D^{(\text{val})}) + \sum_{j\neq k}^K \nabla_{A_k} L(W'_j, A_j, D^{(\text{val})})). \tag{5.6}$$

where

$$\begin{aligned}
&\nabla_{A_k} L(W'_k, A_k, D^{(\text{val})}) = \\
&\nabla_{A_k} L(W_k - \xi_W \nabla_{W_k}(L(W_k, A_k, D^{(\text{tr})}) + \lambda \textstyle\sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V'_j))), A_k, D^{(\text{val})}) = \\
&\nabla_{A_k} L(W'_k, A_k, D^{(\text{val})}) - \xi_W \nabla^2_{A_k, W_k}(L(W_k, A_k, D^{(\text{tr})}) + \\
&\lambda \textstyle\sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V'_j)))\nabla_{W'_k} L(W'_k, A_k, D^{(\text{val})}).
\end{aligned} \tag{5.7}$$

Calculating matrix-vector product $\nabla^2_{A_k, W_k}(L(W_k, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V'_j)))\nabla_{W'_k} L(W'_k, A_k, D^{(\text{val})})$ incurs a lot of computational cost. This cost can be reduced by a finite difference approximation [58]:

$$\begin{aligned}
&\nabla^2_{A_k, W_k}(L(W_k, A_k, D^{(\text{tr})}) + \lambda \textstyle\sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V'_j)))\nabla_{W'_k} L(W'_k, A_k, D^{(\text{val})}) \approx \\
&\tfrac{1}{2\alpha}(\nabla_{A_k}(L(W_k^+, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k^+, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V'_j))) \\
&-\nabla_{A_k}(L(W_k^-, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k^-, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V'_j)))),
\end{aligned} \tag{5.8}$$

where $W_k^{\pm} = W_k \pm \alpha \nabla_{W'_k} L(W'_k, A_k, D^{(\text{val})})$ and α is a small scalar $0.01/\|\nabla_{W'_k} L(W'_k, A_k, D^{(\text{val})})\|_2$. For $\nabla_{A_k} L(W'_j, A_j, D^{(\text{val})})(j \neq k)$ in Eq.(7.10), it can be calculated as:

$$\begin{aligned}
&\nabla_{A_k} L(W'_j, A_j, D^{(\text{val})}) = \\
&\tfrac{\partial W'_j}{\partial A_k} \nabla_{W'_j} L(W'_j, A_j, D^{(\text{val})}) = \\
&-\xi_W \lambda \tfrac{\partial \nabla_{W_j} L(W_j, A_j, D_k^{(\text{pl})}(D^{(\text{u})}, V'_k))}{\partial A_k} \nabla_{W'_j} L(W'_j, A_j, D^{(\text{val})}) = \\
&-\xi_W \lambda \tfrac{\partial V'_k}{\partial A_k} \nabla^2_{V'_k, W_j} L(W_j, A_j, D_k^{(\text{pl})}(D^{(\text{u})}, V'_k)) \nabla_{W'_j} L(W'_j, A_j, D^{(\text{val})})
\end{aligned} \tag{5.9}$$

where

$$\frac{\partial V'_k}{\partial A_k} = \frac{\partial (V_k - \xi_V \nabla_{V_k} L(V_k, A_k, D^{(\text{tr})}))}{\partial A_k} \tag{5.10}$$

$$= -\xi_V \nabla^2_{A_k, V_k} L(V_k, A_k, D^{(\text{tr})}) \tag{5.11}$$

The algorithm for solving SGL is in Algorithm 7.

Algorithm 5: Optimization algorithm for SGL

while *not converged* **do**
1. For learner $1, \cdots, K$, update V_k using Eq.(7.6)
2. For learner $1, \cdots, K$, update W_k using Eq.(7.8)
3. For learner $1, \cdots, K$, update A_k using Eq.(7.10)

end

5.4 Experiments

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

5.4.1 Datasets

We performed the experiments on three datasets: CIFAR-10, CIFAR-100, and ImageNet [22]. CIFAR-10 is split into a 25K training set, a 25K validation set, and a 10K test set. So is CIFAR-100. The training and validation set is used as $D^{(\text{tr})}$ and $D^{(\text{val})}$ in SGL. ImageNet has 1.3M training images and 50K test images. CIFAR-10 and CIFAR-100 have 10 classes and ImageNet has 1000 classes.

5.4.2 Experimental Settings

Following the protocol in [58], each experiment consists of an architecture search phrase and an architecture evaluation phrase. In the search phrase, an architecture A is learned by solving Eq.(5.14). In the evaluation phrase, multiple copies of A are composed into a larger network, which is then trained from scratch and tested on the test set. For the search space of A, we used the ones proposed in DARTS [58], 2) DARTS$^-$ [18], 3) P-DARTS [16], and 4) PC-DARTS [96], where the building blocks include 3×3 and 5×5 (dilated) separable convolutions, 3×3 max/average pooling, identity operation, and zero operation. In SGL, we set the number of learners to 2. The tradeoff parameter λ was tuned in $\{0.1, 0.5, 1, 2, 3\}$ on a held-out 5k dataset. The best performing value of λ is 1. When searching architectures on CIFAR-10, the input images (without labels) in CIFAR-100 were used as the unlabeled dataset; vice versa.

For architecture search on CIFAR-10 and CIFAR-100, each architecture consists of a stack of 8 cells and each cell consists of 7 nodes. The initial channel number was set to 16. The rest hyperparameters for architectures and network weights mostly follow those in DARTS, P-DARTS, PC-DARTS, and DARTS$^-$. During architecture search, the Adam optimizer [47] was used to optimize architecture variables, with a learning rate of 3e-4, a batch size of 50, and a weight decay of 1e-3. The SGD optimizer was used to optimize network weights, with an initial learning rate of 0.025, a cosine decay scheduler, a batch size of 50, a weight decay of 3e-4, and a momentum of 0.9. The search algorithm ran for 50 epochs. For architecture search on ImageNet, following [96], we randomly sample 10% images from the 1.3M training set as $D^{(\text{tr})}$ and 2.5% images as $D^{(\text{val})}$ in SGL. Another randomly sampled 10% images (excluding labels) are used as

the unlabeled dataset. The hyperparameters in the ImageNet experiments mostly follow those in the CIFAR-10 experiments. These experiments were conducted using a single Tesla v100 GPU.

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

5.4.3 Results

In Table 6.2, we compare the classification error on the test set, number of weight parameters, and search cost (GPU days) of different methods on CIFAR-100. We observe the following from this table. **First**, under different settings of search spaces, including those from DARTS, DARTS$^-$, P-DARTS, and PC-DARTS, our method achieves significantly lower test errors than the original baselines. For example, DARTS(1st)+SGL achieves an error of 18.54% which is greatly lower than the 20.52% error of DARTS(1st). As another example, PDARTS+SGL achieves a significantly lower error of 16.58%, compared with the 17.49% error of P-DARTS. These results show that our proposed small-group learning is an effective approach in searching better-performing architectures. In our method, learners with different architectures collaboratively solve the same task. Since having different architectures, these learners possess complementary advantages: for a certain class x, some learners perform better in classifying examples belonging to x than other learners; for another class y, another set of learners have better classification abilities than the rest of learners; and so on. Via pseudo-labeling, each learner can transfer the knowledge in areas it is good at to other learners; equivalently, each learner can mitigate its deficiency in certain areas by leveraging the wisdom from other learners which are excellent in those areas. This collaboration mechanism enables different learners to jointly improve. In single-learner NAS, such a mechanism is lacking, which renders the performance to be inferior. **Second**, our method is broadly applicable and effective, as evidenced by the fact that our method improves a variety of baselines (including DARTS, DARTS$^-$, P-DARTS, and PC-DARTS) when applied to these methods. Our method is designed as a general framework which is agnostic to how the search space is crafted, which paves a way for leveraging our method to improve most (if not all) existing differentiable NAS methods. **Third**, among all methods, PCDARTS+SGL achieves the best performance. This shows that our proposed SGL approach is very competitive in pushing the limit of NAS research. **Fourth**, parameter number and search cost of our method is similar to those of other differentiable methods. This indicates that our method can search more accurate

Method	Error(%)	Param(M)	Cost
*ResNet [36]	22.10	1.7	-
*DenseNet [45]	17.18	25.6	-
*PNAS [56]	19.53	3.2	150
*ENAS [76]	19.43	4.6	0.5
*AmoebaNet [78]	18.93	3.1	3150
*DARTS-2nd [58]	20.58±0.44	1.8	1.5
*GDAS [25]	18.38	3.4	0.2
*R-DARTS [100]	18.01±0.26	-	1.6
$^{\Delta}$DARTS^{+} [53]	17.11±0.43	3.8	0.2
*DropNAS [42]	16.39	4.4	0.7
†DARTS(1st) [58]	20.52±0.31	1.8	0.4
DARTS(1st) + SGL (ours)	**18.54**±0.21	2.2	1.2
*DARTS^{-} [18]	17.51±0.25	3.3	0.4
†DARTS^{-} [18]	18.97±0.16	3.1	0.4
DARTS^{-} + SGL (ours)	**16.90**±0.10	3.5	2.0
*P-DARTS [16]	17.49	3.6	0.3
P-DARTS + SGL (ours)	**16.58**±0.18	3.6	2.1
†PC-DARTS [96]	17.01±0.06	4.0	0.1
PC-DARTS + SGL (ours)	**16.34**±0.11	4.1	0.5

Table 5.2: Test error, parameter number, and search cost (GPU days) on CIFAR-100. Search cost is measured by GPU days on a Tesla v100. In SGL, we count the number of parameters in the single retained learner (the one yielding the lowest validation loss). DARTS(1st) and DARTS(2nd) denotes that the first-order and second-order approximation is used in DARTS. On CIFAR-100, DARTS(2nd) is no better than DARTS(1st) despite using a much more computationally-heavy and memory-inefficient second-order approximation. * indicates that the results are taken from DARTS^{-} [18]. † indicates that the results were obtained by re-running the methods for 10 times. In our run of DARTS^{-}, we were not able to achieve the error reported in [18]. Δ indicates that during architecture evaluation, we ran for 600 epochs instead of 2000 as in [54] to make sure the comparison with other approaches (which use 600 epochs) is fair.

architectures without incurring significant additional costs such as memory footprint, training time, inference time, etc.

In Table 6.3, we compare different methods on CIFAR-10, in terms of classification error on test set, number of network weights, and search cost (GPU days). When applied to DARTS(1st), our method improves the performance of this baseline very significantly, reducing the 3.00% error of DARTS(1st) to 2.41%, which surpasses all other methods in this table. This again demonstrates that our method is effective in searching highly-performant architectures, thanks to its mechanism of enabling models with complementary advantages to collaboratively learn from each other and mitigate weakness by taking in knowledge from stronger learners. The performance of GTN is close to ours. However, this method leads to a very large architecture with 97.9M parameters, which is about 26 times larger than ours. When applied to other methods including †DARTS^{-}, PC-DARTS, and P-DARTS, our method does not achieve a significant

Method	Error(%)	Param(M)	Cost
*DenseNet [45]	3.46	25.6	-
*HierEvol [57]	3.75±0.12	15.7	300
*NAONet-WS [62]	3.53	3.1	0.4
*PNAS [56]	3.41±0.09	3.2	225
*ENAS [76]	2.89	4.6	0.5
*NASNet-A [106]	2.65	3.3	1800
*AmoebaNet-B [78]	2.55±0.05	2.8	3150
*R-DARTS [100]	2.95±0.21	-	1.6
*GDAS [25]	2.93	3.4	0.2
*SNAS [95]	2.85	2.8	1.5
$^{\Delta}$DARTS^{+} [53]	2.83±0.05	3.7	0.4
*BayesNAS [103]	2.81±0.04	3.4	0.2
*DARTS-2nd [58]	2.76±0.09	3.3	1.5
*MergeNAS [93]	2.73±0.02	2.9	0.2
*NoisyDARTS [19]	2.70±0.23	3.3	0.4
*ASAP [69]	2.68±0.11	2.5	0.2
*SDARTS [14]	2.61±0.02	3.3	1.3
*DropNAS [42]	2.58±0.14	4.1	0.6
*FairDARTS [17]	2.54	3.3	0.4
*DrNAS [15]	2.54±0.03	4.0	0.4
*GTN [87]	2.42±0.03	97.9	0.67
*DARTS(1st) [58]	3.00±0.14	3.3	0.4
DARTS(1st) + SGL (ours)	**2.41**±0.06	3.7	1.2
*DARTS^{-} [18]	2.59±0.08	3.5	0.4
†DARTS^{-} [18]	2.97±0.04	3.3	0.6
DARTS^{-} + SGL (ours)	2.60±0.07	3.1	2.0
*PC-DARTS [96]	2.57±0.07	3.6	0.1
PC-DARTS + SGL (ours)	2.60±0.12	3.5	0.5
*P-DARTS [16]	2.50	3.4	0.3
P-DARTS + SGL (ours)	2.47±0.10	3.6	2.1

Table 5.3: Test error, parameter number, and search cost (GPU days) on CIFAR-10. * indicates the results are taken from DARTS^{-} [18], NoisyDARTS [19], DrNAS [15], and GTN [87]. Other notations are the same as those in Table 6.2.

improvement, which is probably due to the fact that CIFAR-10 is a much easier classification dataset than CIFAR-100 and therefore different methods tend to achieve very close performance.

In Table 6.4, we make a comparison of different methods on ImageNet, in terms of top-1 and top-5 classification errors. In SGL-PC-DARTS-ImageNet, our proposed SGL is applied to PC-DARTS and the search is performed on ImageNet. Our method achieves a top-1 error of 23.3% and a top-5 error of 6.7%, which are the lowest among all methods in this table and are much lower than those of PC-DARTS-ImageNet. This demonstrates that our method has strong capability in searching high-quality architectures not only for small datasets like CIFAR10/100, but also for large-scale real-world datasets such as ImageNet. In addition, on ImageNet we evaluated the architectures searched on CIFAR10/100, including SGL-DARTS-1st-CIFAR10, SGL-P-DARTS-CIFAR10, and SGL-P-DARTS-CIFAR100. As can be seen, these architectures searched by our SGL method achieve better performance than those searched by corresponding baselines. For example, applying SGL to DARTS-1st-CIFAR10, we reduce the 26.1% error of DARTS-1st-CIFAR10 down to 24.9%. This further demonstrates the effectiveness of our method.

Method	Top-1 Error (%)	Top-5 Error (%)	Param (M)	Runtime (GPU days)
*Inception-v1 [88]	30.2	10.1	6.6	-
*MobileNet [43]	29.4	10.5	4.2	-
*ShuffleNet 2× (v1) [101]	26.4	10.2	5.4	-
*ShuffleNet 2× (v2) [63]	25.1	7.6	7.4	-
*NASNet-A [106]	26.0	8.4	5.3	1800
*PNAS [56]	25.8	8.1	5.1	225
*MnasNet-92 [89]	25.2	8.0	4.4	1667
*AmoebaNet-C [78]	24.3	7.6	6.4	3150
*SNAS-CIFAR10 [95]	27.3	9.2	4.3	1.5
*BayesNAS-CIFAR10 [103]	26.5	8.9	3.9	0.2
*PARSEC-CIFAR10 [12]	26.0	8.4	5.6	1.0
*GDAS-CIFAR10 [25]	26.0	8.5	5.3	0.2
*DSNAS-ImageNet [44]	25.7	8.1	-	-
*SDARTS-ADV-CIFAR10 [14]	25.2	7.8	5.4	1.3
*PC-DARTS-CIFAR10 [96]	25.1	7.8	5.3	0.1
*ProxylessNAS-ImageNet [10]	24.9	7.5	7.1	8.3
*FairDARTS-CIFAR10 [17]	24.9	7.5	4.8	0.4
*FairDARTS-ImageNet [17]	24.4	7.4	4.3	3.0
*DrNAS-ImageNet [15]	24.2	7.3	5.2	3.9
*DARTS$^+$-ImageNet [53]	23.9	7.4	5.1	6.8
*DARTS$^-$-ImageNet [18]	23.8	7.0	4.9	4.5
*DARTS$^+$-CIFAR100 [53]	23.7	7.2	5.1	0.2
*DARTS-2nd-CIFAR10 [58]	26.7	8.7	4.7	0.4
†DARTS-1st-CIFAR10 [58]	26.1	8.3	4.5	0.4
SGL-DARTS-1st-CIFAR10 (ours)	**24.9**	**7.7**	5.2	2.1
*P-DARTS-CIFAR10 [16]	24.4	7.4	4.9	0.3
SGL-P-DARTS-CIFAR10 (ours)	**24.3**	**7.2**	5.1	2.1
*P-DARTS-CIFAR100 [16]	24.7	7.5	5.1	0.3
SGL-P-DARTS-CIFAR100 (ours)	**23.9**	**7.2**	5.3	2.1
*PC-DARTS-ImageNet [96]	24.2	7.3	5.3	3.8
SGL-PC-DARTS-ImageNet (ours)	**23.3**	**6.7**	6.4	4.0

Table 5.4: Top-1 and top-5 test errors (%) on ImageNet. * indicates that the results are taken from DARTS$^-$ [18], DrNAS [15], and AKDNet [59]. † denotes that the result is obtained from our run. The rest notations are the same as those in Table 6.2. From top to bottom, different panels show manually-designed networks, non-differentiable search methods, and differentiable search methods.

5.4.4 Ablation Studies

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

- **Ablation setting 1**. In this setting, the first learning stage in Eq.(5.14) is removed. Pseudo-

labeling is performed using the weights W. The corresponding formulation is:

$$\begin{aligned}
&\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(\{A_j\}_{j=1}^K), A_k, D^{(\text{val})}) \\
&s.t. \quad \{W_k^*(\{A_j\}_{j=1}^K)\}_{k=1}^K = \min_{\{W_k\}_{k=1}^K} \sum_{k=1}^K L(W_k, A_k, D^{(\text{tr})}) \\
&\qquad +\lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, W_j, A_j))
\end{aligned} \tag{5.12}$$

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

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

$$\begin{aligned}
&\min_{\{B_k\}_{k=1}^K} \sum_{k=1}^K L(V_k^*(B_k), B_k, D^{(\text{val})}) \\
&s.t. \quad \{V_k^*(B_k)\}_{k=1}^K = \min_{\{V_k\}_{k=1}^K} \sum_{k=1}^K L(V_k, B_k, D^{(\text{tr})})
\end{aligned} \tag{5.13}$$

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

$$\begin{aligned}
&\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(A_k), A_k, D^{(\text{val})}) \\
&s.t. \quad \{W_k^*(A_k)\}_{k=1}^K = \min_{\{W_k\}_{k=1}^K} \sum_{k=1}^K L(W_k, A_k, D^{(\text{tr})}) + \lambda \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j^*, B_j^*))
\end{aligned} \tag{5.14}$$

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

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

$$\begin{aligned}
&\min_{\{A_k\}_{k=1}^K} \sum_{k=1}^K L(W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K), A_k, D^{(\text{val})}) \\
&s.t. \quad \{W_k^*(A_k, \{V_j^*(A_j)\}_{j\neq k}^K)\}_{k=1}^K = \min_{\{W_k\}_{k=1}^K} \sum_{k=1}^K \sum_{j\neq k}^K L(W_k, A_k, D_j^{(\text{pl})}(D^{(\text{u})}, V_j^*(A_j))) \\
&\qquad \{V_k^*(A_k)\}_{k=1}^K = \min_{\{V_k\}_{k=1}^K} \sum_{k=1}^K L(V_k, A_k, D^{(\text{tr})})
\end{aligned} \tag{5.15}$$

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

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

Table 5.5: Results for ablation setting 1. "With first stage" corresponds to the full SGL framework. "Without first stage" corresponds to the formulation in Eq.(6.4.4) where the first learning stage is removed. The results are classification errors on the test sets of CIFAR-10 and CIFAR-100.

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

Table 5.6: Results for ablation setting 2. "Separate" means the first stage in SGL is conducted separately from the second stage. "Joint" corresponds to the full SGL framework where the first and second stages are conducted jointly. The results are classification errors on the test sets of CIFAR-10 and CIFAR-100.

- Ablation study on the tradeoff parameter λ. We investigate how the classification error changes with λ in Eq.(5.14). For either CIFAR-10 or CIFAR-100, 5K examples are uniformly sampled from the 25K training set and 25K validation set. Performance is reported on the 5K sampled data. The rest data is used as before. SGL is applied to P-DARTS.
- Ablation study on the number of learners K. We investigate how the classification error changes with the number of learners K. Performance is reported on the 5K randomly sampled data. SGL is applied to DARTS(1st).

Table 6.5 shows the classification errors on the test sets of CIFAR-10 and CIFAR-100, under the settings of "with first stage" and "without first stage" in the first ablation study. As can be seen, removing the first stage renders the errors to increase on both CIFAR-10 and CIFAR-100, compared with using the first stage. The reason is that: for "without first stage", different learners directly use untrained models to generate pseudo-labels. The classification performance of untrained models is not very satisfactory. As a result, the quality of pseudo-labels has no guarantee. Training models on poorly-labeled datasets will degrade the quality of these models. In contrast, in full SGL which performs model pretraining in the first stage then using pretrained models to perform pseudo-labeling, the risk of generating poor-quality pseudo-labels is significantly reduced.

Table 5.6 shows the classification errors on the test sets of CIFAR-10 and CIFAR-100, under the settings of "Separate" and "Joint" in the second ablation study. As can be seen, conducting the first and second stage separately leads to worse performance than performing them jointly, on both datasets. The reason is that: when performed separately, the second stage cannot influence the first stage; in the first stage, once the architectures are searched and network weights are trained, they will remain fixed. In contrast, in the full SGL framework where the first and second

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

Table 5.7: Results for ablation setting 3. "With human labels" corresponds to the full SGL framework. "Without human labels" corresponds to the formulation in Eq.(6.4.4) where in the second stage each learner is trained only on pseudo-labeled datasets without using human-labeled datasets. The results are classification errors on the test sets of CIFAR-10 and CIFAR-100.

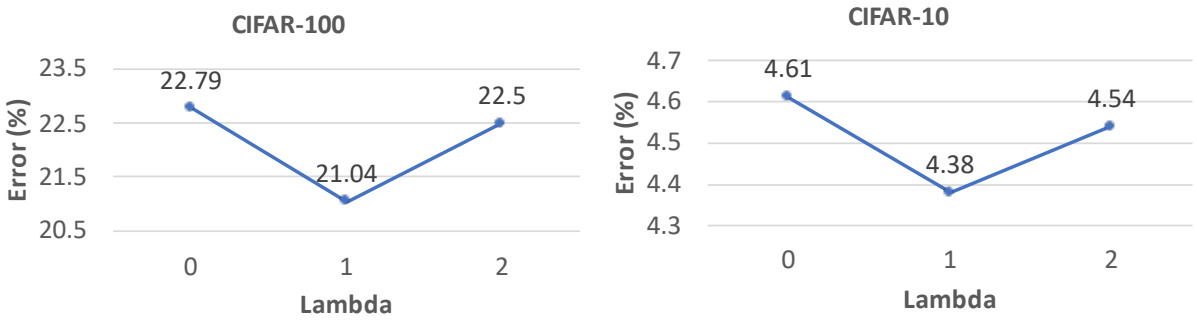

Figure 5.3: How classification errors change as the tradeoff parameter λ increases. The errors are reported on the 5K randomly sampled data.

stage are performed jointly and share the same architecture, models trained in the second stage affect the validation loss, which subsequently affects the architecture; the changed architecture renders the models in the first stage to change as well. This can potentially bring in a synergistic effect: better models in the second stage results in better architectures in the third stage; better architectures improve the models in the first stage. Performing the first and second stage separately makes such a synergistic effect impossible to happen, which hence leads to inferior performance.

Table 6.6 shows the classification errors on the test sets of CIFAR-10 and CIFAR-100 under the settings of "with human labels" and "without human labels" in the third ablation study. As can be seen, in the second stage of SGL, using only pseudo-labeled datasets for model training without leveraging human-labeled dataset yields worse performance. The reason is that mutual training solely based on pseudo-labeled datasets has a high risk of collective failure: if one model performs poorly, it will generate erroneous pseudo-labels, which renders other models trained using the pseudo-labels to fail as well. By leveraging an external human-labeled dataset where the labels are mostly correct, such a risk can be greatly reduced.

Figure 7.3 shows the classification errors on CIFAR-100 and CIFAR-10 change as λ increases. As can be seen, on CIFAR-100, when λ increases from 0 to 1, the error decreases. This is because a larger λ results in a stronger collaboration between learners where each learner actively relies on the pseudo-labeled datasets generated by other learners to train itself. A stronger collaboration enables different learners to help each other to improve. However, if λ further increases to 2, the error increases. This is because too much emphasis is put on the pseudo-labeled

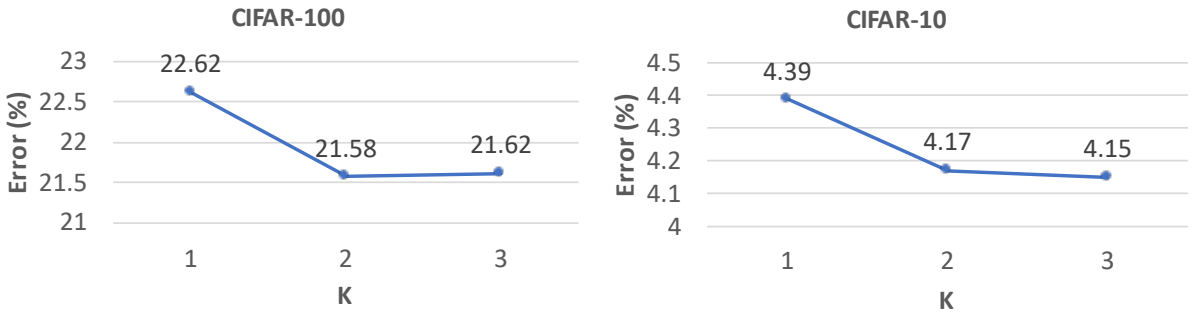

Figure 5.4: How classification errors change as the number of learners K increases. The errors are reported on the 5K randomly sampled data.

datasets which are less reliable than the human-labeled training set. Ignoring the human-labeled training set degrades the quality of trained models. Similar trend is observed in the CIFAR-10 results as well.

Figure 5.4 shows how the classification errors on CIFAR-100 and CIFAR-10 vary as the number of learners K increase from 1 to 3. As can be seen, when increasing K from 1 to 2, the error decreases. Under $K = 1$, there is no collaboration. When $K = 2$, two learners collaboratively help each other to improve, hence achieving better performance. When K increases from 2 to 3, the performance does not change significantly. This indicates that two learners are sufficient for exploring the benefit of collaboration.

5.5 Conclusions

In this chapter, drawing inspirations from the small-group learning (SGL) skill of humans, we propose a new ML framework to formalize SGL into a machine learning skill and leverage it to train better ML models. In SGL, a set of learners help each other to learn better: each learner uses its intermediately trained model to make predictions on unlabeled data examples and generates a pseudo-labeled dataset; meanwhile, each learner uses pseudo-labeled datasets generated by other learners to retrain and improve its model. To formalize SGL, a multi-level optimization framework is proposed, which consists of three learning stages: each learner trains an initial model separately; all learners retrain their models collaboratively by mutually performing pseudo-labeling; all learners improve their neural architectures based on the feedback obtained during model validation. We apply SGL to search neural architectures on CIFAR-100, CIFAR-10, and ImageNet. Experimental results demonstrate the effectiveness of our method.

Chapter 6

Learning by Teaching

6.1 Introduction

As the saying goes, a good learner is also a good teacher. In human learning, a commonly adopted strategy is to learn by teaching others. In the process of explaining a topic to others, the learner can further enhance his/her understanding of this topic. The efficacy of teaching in helping improve learning has been demonstrated in many studies. In [29], the studies showed that students who teach what they have learned to their peers achieve better understanding and knowledge retention than students spending the same time re-studying. In [48], the study shows that teaching improves the teacher's learning because it encourages the teacher to retrieve what they have previously learned.

This teaching-driven learning methodology of humans motivates us to think about whether it can benefit machine learning as well. Toward this goal, we propose a novel ML framework called learning by teaching (LBT) (as illustrated in Figure 7.1), which improves the learning outcome of a model by encouraging this model to teach other models to perform well. In our framework, there is a teacher model and a student model, which perform the same target task (e.g., text classification, time-series forecasting, etc.). The eventual goal is to make the teacher perform well on the target task. The way to achieve that is to let the teacher teach the student and use the student's performance as a feedback to guide the teacher to improve its learning capability and outcome. The teacher model consists of a learnable architecture and a set of learnable network weights. The student model consists of a predefined architecture (by human experts) and a set of learnable network weights. Similar to [41], teaching is conducted via pseudo-labeling: given an unlabeled dataset U, the teacher uses its intermediately trained model to make predictions on the input data examples in U; then the student model is trained on these pseudo-labeled data examples. Teacher-student learning based on pseudo-labeling has been studied in many previous works [73, 90, 94]. Our work differs from previous ones in that we aim to improve the learning ability of the teacher by letting it teach a student while previous works focus on improving the learning ability of a student model by letting it be taught by a fixed teacher model. In other words, our work focuses on learning the teacher while previous works focus on learning the student.

In our framework, the learning of the teacher and student are organized into three stages. In the first stage, the teacher learns its network weights on a training dataset while temporarily

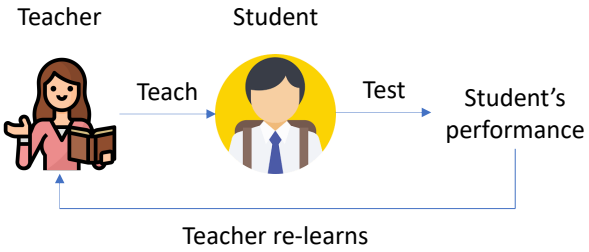

Figure 6.1: Illustration of learning by teaching. The teacher first learns a topic. Then the teacher teaches this topic to the student and the student learns this topic. The student performs a test to check how well he/she masters this topic. Based on the student's performance on the test, the teacher re-learns this topic to improve his/her understanding.

fixing its architecture. In the second stage, the teacher performs pseudo-labeling on an unlabeled dataset and uses the pseudo-labeled dataset to train the student model. Specifically, the teacher applies its model trained in the first stage to make predictions on unlabeled data examples and the student model is trained to fit these predictions. In the third stage, the student's model is evaluated on a validation set and the teacher adjusts its architecture based on the student's validation performance. The three stages are organized into a three-level optimization framework and are performed end-to-end in a unified manner, where earlier stages influence later stages and vice versa. We apply LBT for neural architecture search. Experiments on CIFAR-100, CIFAR-10, and ImageNet [22] demonstrate the effectiveness of our method.

The major contributions of this paper include:

- Motivated by the teaching-driven learning methodology of humans, we develop a novel machine learning framework called learning by teaching (LBT). In our approach, a teacher creates a pseudo-labeled dataset and uses it to train a student model. Based on how the student performs on the validation dataset, the teacher re-learns its model and re-teaches the student until the student achieves great validation performance.
- To formulate LBT, we develop a three-level optimization framework. This framework consists of three learning stages: 1) teacher performs learning; 2) teacher teaches what it has learned to a student; 3) teacher re-learns based on the performance of the student.
- An efficient algorithm is developed to solve the three-level optimization problem.
- We apply LBT for neural architecture search on CIFAR-100, CIFAR-10, and ImageNet, where the results show that our method is very effective in searching highly-performing neural architectures.

6.2 Related Works

6.2.1 Neural Architecture Search

Neural architecture search (NAS) is the task of developing algorithms to automatically find out architectures that can yield high ML-performance. Existing NAS methods can be categorized

into three groups. Methods in the first group [76, 105, 106] are based on reinforcement learning, where an architecture generation policy is learned by maximizing ML performance on validation data. Methods in the second group [10, 58, 95] are gradient-based and differentiable. These methods adopt a network pruning strategy where an overparameterized network with many building blocks is pruned into the final architecture and the optimal pruning is achieved by minimizing the validation loss. There have been many efforts devoted to improving differentiable NAS methods. In P-DARTS [16], the depth of searched architectures is allowed to grow progressively during the training process. Search space approximation and regularization approaches are developed to reduce computational overheads and improve search stability. PC-DARTS [96] reduces the redundancy in exploring the search space by sampling a small portion of a super network. Operation search is performed in a subset of channels with the held-out part bypassed in a shortcut. Methods in the third group [57, 78] are based on evolutionary algorithms where architectures are represented as a population. Highly-performing architectures are allowed to generate offspring while poorly-performing architectures are eliminated.

6.2.2 Teacher-Student Learning

Teacher-student learning has been investigated in knowledge distillation [40], adversarial robustness [11], self-supervised learning [94], etc. Most of these methods are based on pseudo-labeling. In these existing methods, the focus is to learn a student model with the help of a trained and fixed teacher model. In these works, the teacher model is not updated. In contrast, our method focuses on learning a teacher model, by letting it teach a student model. The teacher model constantly updates itself based on the teaching outcome. Teacher-student learning has been investigated in several neural architecture search works as well [35, 52, 91]. In these works, when searching the architecture of a student model, pseudo-labels generated by a trained teacher model whose architecture is fixed are leveraged. Our work differs from these works in that we focus on searching the architecture of a teacher model by letting it teach a student model where the student's architecture is fixed, whereas the existing works focus on searching the architecture of a student model by letting it be taught by a teacher where the teacher's architecture is fixed. In a recent work [77] which was conducted independently of and in parallel to our work, the teacher model is updated based on student's performance. Our work differs from this one in that our work is based on a three-level optimization framework which searches teacher's architecture by minimizing student's validation loss and trains teacher's network weights before using teacher to generate pseudo-labels, whereas in [77] the framework is based on two-level optimization which has no architecture search and does not train the teacher before using it to perform pseudo-labeling. In [87], a meta-learning method is developed to learn a deep generative model which generates synthetic labeled-data. A student model leverages the synthesized data to search its architecture. Our work differs from this method in that we focus on searching the teacher's architecture via three-level optimization while [87] focuses on searching the student's architecture via meta-learning.

Notation	Meaning
A	Architecture of the teacher
T	Network weights of the teacher
S	Network weights of the student
$D_t^{(\text{tr})}$	Training data of the teacher
$D_t^{(\text{val})}$	Validation data of the teacher
$D_s^{(\text{tr})}$	Training data of the student
$D_s^{(\text{val})}$	Validation data of the student
D_u	Unlabeled dataset

Table 6.1: Notations in Learning by Teaching

6.3 Methods

In this section, we propose a three-level optimization framework to formalize learning-by-teaching (LBT) (as shown in Figure 7.2) and develop an efficient optimization algorithm for solving the LBT problem.

6.3.1 Learning by Teaching

In our framework, there is a teacher model and a student model, which both study how to perform the same target task. Without loss of generality, we assume the target task is classification. The eventual goal is to make the teacher achieve better learning outcomes. The way to achieve this goal to let the teacher teach the student to perform well on the target task. The intuition behind LBT is that a teacher needs to learn a topic very well in order to teach this topic to a student clearly. Teaching is performed based on pseudo-labeling [40]: the teacher uses its model to generate a pseudo-labeled dataset; the student is trained on the pseudo-labeled dataset. The teacher has a learnable neural architecture A and a set of learnable network weights T. The student has a predefined architecture (by humans) and a set of learnable network weights S. The teacher has a training dataset $D_t^{(\text{tr})}$ and a validation dataset $D_t^{(\text{val})}$. The student has a training dataset $D_s^{(\text{tr})}$ and a validation dataset $D_s^{(\text{val})}$. There is an unlabeled dataset D_u where pseudo labeling is performed. In our framework, both the teacher and student perform learning, which is organized into three stages. In the first stage, the teacher fixes its architecture and trains its network weights by minimizing the training loss defined on $D_t^{(\text{tr})}$:

$$T^*(A) = \min_T L(T, A, D_t^{(\text{tr})}). \tag{6.1}$$

The architecture A is needed to calculate the loss on training examples. However, it cannot be updated by minimizing the training loss. Otherwise, a degenerated solution will be produced where A has very large capacity to overfit the training examples but will yield poor prediction outcomes on unseen examples. $T^*(A)$ is a function of A: a different A will result in a different training loss $L(A, T, D_t^{(\text{tr})})$; T trained by minimizing $L(A, T, D_t^{(\text{tr})})$ will be different as well. In the second stage, the teacher teaches a student via pseudo-labeling. Given

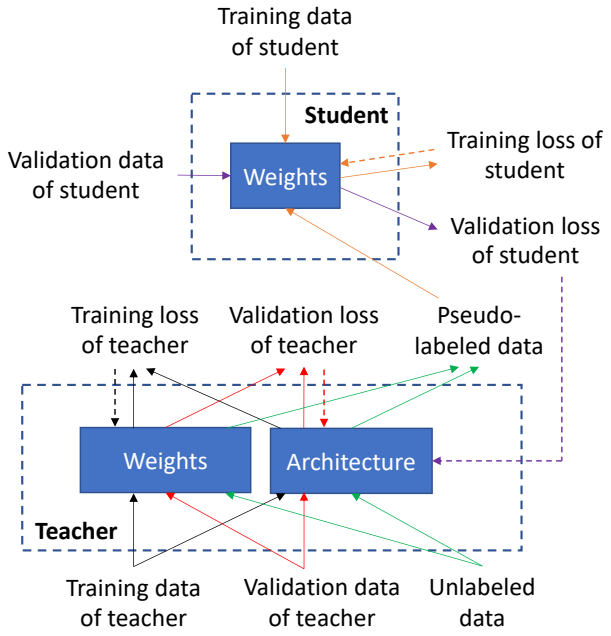

Figure 6.2: Learning by teaching. The solid arrows denote a forward pass where predictions are made and training/validation losses are defined. The dotted arrows denote a backward process where gradients of losses are calculated and parameters are updated.

an unlabeled dataset $D_u = \{x_i\}_{i=1}^N$, the teacher uses its model $T^*(A)$ trained in the first stage to make predictions on D_u. Assuming the task is classification with K classes, the prediction $f(x_i; T^*(A))$ on x_i would be a K-dimensional vector, where the k-th element indicates the probability that x_i belongs to the k-th class and the sum of elements in $f(x_i; T^*(A))$ is one. Let $D_{pl}(D_u, T^*(A)) = \{(x_i, f(x_i; T^*(A)))\}_{i=1}^N$ denote the pseudo-labeled dataset. The network weights S of the student is trained on $D_{pl}(D_u, T^*(A))$ and a human-labeled training set $D_s^{(\text{tr})}$:

$$S^*(T^*(A)) = \min_S L(S, D_s^{(\text{tr})}) + \lambda L(S, D_{pl}(D_u, T^*(A))).$$

where $L(\cdot)$ denotes a cross-entropy loss and λ is a tradeoff parameter. $S^*(T^*(A))$ is a function of $T^*(A)$: a different $T^*(A)$ will result in a different pseudo-labeled dataset $D_{pl}(D_u, T^*(A))$ which will render the training loss to be different; a different training loss will result in a different $S^*(T^*(A))$. In the third stage, the student's model $S^*(T^*(A))$ trained in the second stage is validated on $D_s^{(\text{val})}$. Besides, we also validate the teacher's model $T^*(A)$ trained in the first stage on $D_t^{(\text{val})}$. The validation performances provide feedback on how good the teacher's architecture A is. At this stage, A is optimized by minimizing the validation losses:

$$\min_A L(T^*(A), A, D_t^{(\text{val})}) + \gamma L(S^*(T^*(A)), D_s^{(\text{val})}), \tag{6.2}$$

where γ is a tradeoff parameter.

Given the three learning stages, we propose a three-level optimization framework to stitch

them together:

$$\begin{aligned} \min_{A} \quad & L(T^*(A), A, D_t^{(\text{val})})) + \gamma L(S^*(T^*(A)), D_s^{(\text{val})}) \\ s.t. \quad & S^*(T^*(A)) = \min_{S} \; L(S, D_s^{(\text{tr})}) + \lambda L(S, D_{pl}(D_u, T^*(A))) \\ & T^*(A) = \min_{T} L(A, T, D_t^{(\text{tr})}) \end{aligned} \tag{6.3}$$

From bottom to top, the three optimization problems correspond to the first, second, and third stage respectively. The first two optimization problems are on the constraints of the third optimization problem. The three stages are performed end-to-end in a joint manner where different stages mutually influence each other. $T^*(A)$ trained in the first stage is used to generate pseudo-labeled dataset in the second stage; $T^*(A)$ and $S^*(T^*(A))$ trained in the first two stages are validated in the third stage; after A is updated in the third stage, it will render the training loss in the first stage to be changed, which accordingly results in a new $T^*(A)$. For computational efficiency, we search A in a differentiable way [58]: given an overparameterized network, a sub-network is carved out as the final architecture. The overparameterized network contains a large number of basic building blocks such as convolution operations, pooling operations, etc. The output of each building block is multiplied with a scalar. The search algorithm optimizes these scalars by minimizing validation losses. In the end, building blocks with largest scalars form the final architecture.

6.3.2 Optimization Algorithm

In this section, we develop a gradient-based algorithm to solve the learning by teaching (LBT) problem. Drawing insights from [58], we calculate the gradient of $L(A, T, D_t^{(\text{tr})})$ w.r.t T, update T by one-step gradient descent and get T', which is used as an approximation of $T^*(A)$. We substitute this approximation into $L(S, D_s^{(\text{tr})}) + \lambda L(S, D_{pl}(D_u, T^*(A)))$, yielding an approximated objective O_s. Similarly, we approximate $S^*(T^*(A))$ using one-step gradient descent update of S using the gradient of O_s. Lastly, we substitute the approximations of $T^*(A)$ and $S^*(T^*(A))$ into $L(T^*(A), D_t^{(\text{val})})) + \gamma L(S^*(T^*(A)), D_s^{(\text{val})})$ and perform gradient-descent update of A by minimizing the approximated validation loss. Let $\nabla^2_{Y,X} f(X, Y)$ denote $\frac{\partial f(X,Y)}{\partial X \partial Y}$.

First, we approximate $T^*(A)$ using

$$T' = T - \xi_t \nabla_T L(T, A, D_t^{(\text{tr})}) \tag{6.4}$$

where ξ_t is a learning rate. Substituting T' into $L(S, D_s^{(\text{tr})}) + \lambda L(S, D_{pl}(D_u, T^*(A)))$ results in an approximated objective $O_s = L(S, D_s^{(\text{tr})}) + \lambda L(S, D_{pl}(D_u, T'))$. Next, we approximate $S^*(T^*(A))$ using one-step gradient descent update of S w.r.t O_s:

$$S' = S - \xi_s \nabla_S (L(S, D_s^{(\text{tr})}) + \lambda L(S, D_{pl}(D_u, T'))). \tag{6.5}$$

Finally, we plug T' and S' into $L(T^*(A),$
$D_t^{(\text{val})})) + \gamma L(S^*(T^*(A)), D_s^{(\text{val})})$ and get $O_v = L(T', D_t^{(\text{val})}) + \gamma L(S', D_s^{(\text{val})})$. We can update the teacher's architecture A by descending the gradient of O_v w.r.t A:

$$A \leftarrow A - \eta(\nabla_A L(T', A, D_t^{(\text{val})}) + \gamma \nabla_A L(S', D_s^{(\text{val})})) \tag{6.6}$$

where

$$\begin{aligned}&\nabla_A L(T', A, D_t^{(\text{val})}) = \\ &\nabla_A L(T - \xi_t \nabla_T L(T, A, D_t^{(\text{tr})}), A, D_t^{(\text{val})}) = \\ &-\xi_t \nabla^2_{A,T} L(T, A, D_t^{(\text{tr})}) \nabla_{T'} L(T', A, D_t^{(\text{val})}) + \nabla_A L(T', A, D_t^{(\text{val})})\end{aligned} \tag{6.7}$$

The matrix-vector multiplication in the first term on the third line is computationally expensive. To reduce computational cost, following [58], we approximate the multiplication using a finite difference:

$$\nabla^2_{A,T} L(T, A, D_t^{(\text{tr})}) \nabla_{T'} L(T', A, D_t^{(\text{val})}) \approx \tfrac{1}{2\alpha}(\nabla_A L(T^+, A, D_t^{(\text{tr})}) - \nabla_A L(T^-, A, D_t^{(\text{tr})})), \tag{6.8}$$

where $T^{\pm} = T \pm \alpha \nabla_{T'} L(T', A, D_t^{(\text{val})})$ and α is $0.01/\|\nabla_{T'} L(T', A, D_t^{(\text{val})})\|_2$.

For $\nabla_A L(S', D_s^{(\text{val})})$ in Eq.(7.10), it can be calculated as $\frac{\partial S'}{\partial A} \nabla_{S'} L(S', D_s^{(\text{val})})$ according to the chain rule, where

$$\frac{\partial S'}{\partial A} = \frac{\partial (S - \xi_s \nabla_S (L(S, D_s^{(\text{tr})}) + \lambda L(S, D_{pl}(D_u, T'))))}{\partial A} \tag{6.9}$$

$$= \frac{\partial (-\xi_s \lambda \nabla_S L(S, D_{pl}(D_u, T')))}{\partial A} \tag{6.10}$$

$$= -\xi_s \lambda \frac{\partial T'}{\partial A} \nabla^2_{T',S} L(S, D_{pl}(D_u, T')) \tag{6.11}$$

For $\frac{\partial T'}{\partial A}$, it can be calculated as:

$$\frac{\partial T'}{\partial A} = \frac{\partial (T - \xi_t \nabla_T L(T, A, D_t^{(\text{tr})}))}{\partial A} = -\xi_t \nabla^2_{A,T} L(T, A, D_t^{(\text{tr})}) \tag{6.12}$$

The algorithm for solving LBT is presented in Algorithm 7.

Algorithm 6: Optimization algorithm for learning by teaching

while *not converged* **do**
1. Update the teacher's network weights T using Eq.(6.4)
2. Update the student's network weights S using Eq.(6.5)
3. Update the teacher's architecture A using Eq.(7.10)

end

6.4 Experiments

In this section, we apply learning-by-teaching (LBT) to search neural architectures in image classification tasks. We follow the experimental protocol in [58], consisting of two phrases: one for architecture search and the other for architecture evaluation. In the search phrase, an optimal cell is searched by minimizing the validation loss. In the evaluation phrase, 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 the test set. More hyperparameter settings, additional results, and significance tests of results are deferred to the supplements.

6.4.1 Datasets

The experiments were conducted on three image classification datasets: CIFAR-10, CIFAR-100, and ImageNet [22], 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 5K test set. The training set is used as $D_t^{(\text{tr})}$ of the teacher and $D_s^{(\text{tr})}$ of the student. The validation set is used as $D_t^{(\text{val})}$ of the teacher and $D_s^{(\text{val})}$ of the student. For experiments on CIFAR-10, input images in CIFAR-100 (removing labels) are used as the unlabeled dataset D_u. For experiments on CIFAR-100, input images in CIFAR-10 (removing labels) are used as the unlabeled dataset D_u. For ImageNet, it is split into a training set of 1.2M images and a test set of 50K images.

6.4.2 Experimental Settings

In LBT, for the search space of A, we experimented the spaces in DARTS [58], P-DARTS [16], and PC-DARTS [96]. These search spaces are similar, with the following candidate operations: 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. For the student's architecture, we experimented with ResNet-18 and ResNet-50 [37]. λ and γ are both set to 1.

During architecture search, for CIFAR-10 and CIFAR-100, the teacher's architecture is a stack of 8 cells, each consisting of 7 nodes. The initial channel number was set to 16. The rest hyperparameters for the teacher's architecture and network weights follow those in DARTS, P-DARTS, and PC-DARTS. The search algorithm ran for 30 epochs in LBT-DARTS with a batch size of 32 and ran for 50 epochs in LBT-PC-DARTS, with a batch size of 96. Network weights are optimized using SGD, with an initial learning rate of 0.025 (adjusted using a cosine decay scheduler) for LBT-DARTS and an initial learning rate of 0.1 for LBT-PC-DARTS, a momentum of 0.9, and a weight decay of 3e-4. For architecture search on ImageNet, following [96], we randomly sample 10% of the 1.2M ImageNet images as $D_t^{(\text{tr})}$ and $D_s^{(\text{tr})}$ in LBT, randomly sample 2.5% of the 1.2M images as $D_t^{(\text{val})}$ and $D_s^{(\text{val})}$, and randomly sample another 10% of the 1.2M images as D_u.

During architecture evaluation, for CIFAR-10 and CIFAR-100, 20 copies of the optimal cell searched in the search phrase are stacked into a large network, which is trained using the combined training and validation datasets. The initial channel number was set to 36. 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, a weight decay of 3e-4, and epoch number of 600. These experiments were conducted on a Tesla v100 GPU.

For ImageNet, we evaluate the architectures searched by PC-DARTS on the subset of ImageNet and architectures searched by DARTS-2nd and P-DARTS on CIFAR-10 and CIFAR-100, by stacking 14 searched cells into a large network and training it on the 1.2M training images and reporting its performance on the 50K test images. The initial channel number was set to 48. The network was trained for 250 epochs with a batch size of 1024, an initial learning rate of 0.5, and a weight decay of 3e-5, on 8 Tesla v100s. Each LBT experiment was repeated for ten times with different random seeds. Mean and standard deviation of the 10 runs are reported.

6.4.3 Results

In Table 6.2, we compare different NAS methods in terms of classification error on the test set of CIFAR-100, number of network weights, and search cost measured using GPU days. From these results, we observe the following. **First**, with the help of our proposed learning-by-teaching (LBT) framework, the architectures searched by various methods including DARTS, P-DARTS, and PC-DARTS can be greatly improved. For example, applying LBT to DARTS-2nd, the error is reduced from 20.58% to 17.06%. With the assistance of LBT, the error of P-DARTS decreases from 17.49% to 16.29%. Without using LBT, the error of PC-DARTS is 17.96%; adding LBT reduces this error to 16.88%. These results strongly demonstrate that LBT is an effective learning framework that helps to improve a wide variety of NAS methods. In our method, the teacher model improves its learning ability by teaching a student model to perform well on the classification task. The student is trained on the pseudo-labeled dataset created by the teacher. If the student does not perform well on the validation set, that means the pseudo labels are not correct, which indicates the teacher's model is not accurate. To avoid such an outcome, the teacher enforces itself to learn better to generate correct pseudo labels. **Second**, a stronger student helps the teacher to learn better. Here we consider a student model is stronger if its architecture (manually designed) is more powerful and expressive. For example, ResNet with 50 layers (RN50) is generally considered to have stronger representation learning power than ResNet with 18 layers (RN18). In LBT applied to DARTS, P-DARTS, and PC-DARTS, we experimented with two student models with RN50 and RN18 as their architectures respectively. As can be seen, LBT with RN50 as the student achieves better performance than LBT with RN18 as the student. For example, on DARTS-2nd, LBT with RN50 achieves an error of 17.06% while LBT with RN18 achieves an error of 17.93%. As another example, on P-DARTS, the error achieved by LBT(RN50) is 16.29%, which is lower than the 16.53% error achieved by LBT(RN18). The reason is that to teach a stronger student to learn better, the teacher has to be even stronger. For example, given a relatively weak student such as a CNN with only three layers, since there is a large room for the student to improve, an ordinary teacher such as a CNN with ten layers would be sufficient to teach the student to perform better. However, if the student (e.g., ResNet with 50 layers) is very strong whose performance is already high, it is very challenging to teach the student to improve unless the teacher is even stronger. To better train a strong student, the pseudo-labels generated by the teacher are required to be highly accurate. To achieve this goal, the teacher is forced to escalate the performance of its model to an excellent level. **Third**, our method LBT(RN50,P-DARTS) achieves the lowest error among all methods in this table. This shows that our method is very competitive in bringing the NAS performance to a state-of-the-art level. **Fourth**, while our LBT framework can greatly help to improve the quality of searched architectures, it does not substantially increase the number of model parameters or search cost. As shown in the third column and fourth column, the model size and search cost of our methods are similar to those of baselines.

In Table 6.3, we show the results on CIFAR-10, including the classification error on the test set, number of model parameters, and search cost measured by GPU days. From this table, we make similar observations as those in Table 6.2. **First**, our proposed LBT framework is widely effective in helping different NAS methods to improve the quality of searched architectures. For example, applying LBT to DARTS-2nd reduces the error from 2.76% to 2.61%. This fur-

Method	Error(%)	Param(M)	Cost
*ResNet [36]	22.10	1.7	-
*DenseNet [45]	17.18	25.6	-
*PNAS [56]	19.53	3.2	150
*ENAS [76]	19.43	4.6	0.5
*AmoebaNet [78]	18.93	3.1	3150
*GDAS [25]	18.38	3.4	0.2
*R-DARTS [100]	18.01±0.26	-	1.6
*DARTS$^-$ [18]	17.51±0.25	3.3	0.4
*DropNAS [42]	16.39	4.4	0.7
†DARTS-1st [58]	20.52±0.31	1.8	0.4
LBT(RN18,DARTS-1st) (ours)	19.28±0.13	1.9	0.5
LBT(RN50,DARTS-1st) (ours)	**18.74**±0.09	1.9	0.6
*DARTS-2nd [58]	20.58±0.44	1.8	1.5
LBT(RN18,DARTS-2nd) (ours)	17.93±0.18	2.0	1.9
LBT(RN50,DARTS-2nd) (ours)	**17.06**±0.22	2.1	2.2
*P-DARTS [16]	17.49	3.6	0.3
LBT(RN18,P-DARTS) (ours)	16.53±0.16	3.6	0.5
LBT(RN50,P-DARTS) (ours)	**16.29**±0.07	3.7	0.6
†PC-DARTS [96]	17.96±0.15	3.9	0.1
LBT(RN18,PC-DARTS) (ours)	17.21±0.13	3.8	0.1
LBT(RN50,PC-DARTS) (ours)	**16.88**±0.09	3.8	0.2

Table 6.2: Results on CIFAR-100, including classification error (%) on the test set, number of model weights (millions), and search cost (GPU days). LBT(RN18,DARTS-1st) represents that in LBT the search space is the same as that of DARTS-1st and the architecture of the student is ResNet-18. Similar meanings hold for other notations in such a format. RN50 denotes ResNet-50. DARTS-1st and DARTS-2nd indicates that first-order and second-order approximation is used in DARTS' optimization algorithm. * denotes that the results are taken from DARTS$^-$ [18]. † denotes that we re-ran this method for 10 times. The search cost is measured by GPU days on a Tesla v100.

ther demonstrates that by teaching a student model to learn well, the teacher can improve itself greatly. In the GTN [87] baseline approach where the architecture of a student is searched by leveraging the synthetic data generated by a trainable generative model, the classification error is 2.92%, which is much worse than those achieved by our methods, while the number of parameters in GTN is twice more than ours. Setting the network width to 128, GTN achieves an error of 2.42%; however, the resulting number of parameters in GTN is about 30 times more than ours. GTN focuses on searching the student's architecture while our method focuses on searching the teacher's architecture. These results demonstrate that searching teacher's architecture is more advantageous than searching student's architecture. **Second**, using ResNet with 50 layers (RN50) as the student results in better performance than using ResNet-18 (RN18), which further demonstrates that teaching a stronger student can drive the teacher to learn better. **Third**, while achieving better classification accuracy than baselines, our method does not substantially

Method	Error(%)	Param(M)	Cost
*DenseNet [45]	3.46	25.6	-
*HierEvol [57]	3.75±0.12	15.7	300
*NAONet-WS [62]	3.53	3.1	0.4
*PNAS [56]	3.41±0.09	3.2	225
*ENAS [76]	2.89	4.6	0.5
*NASNet-A [106]	2.65	3.3	1800
*AmoebaNet-B [78]	2.55±0.05	2.8	3150
*R-DARTS [100]	2.95±0.21	-	1.6
*GDAS [25]	2.93	3.4	0.2
*SNAS [95]	2.85	2.8	1.5
*BayesNAS [103]	2.81±0.04	3.4	0.2
*MergeNAS [93]	2.73±0.02	2.9	0.2
*NoisyDARTS [19]	2.70±0.23	3.3	0.4
*ASAP [69]	2.68±0.11	2.5	0.2
*SDARTS [14]	2.61±0.02	3.3	1.3
*DropNAS [42]	2.58±0.14	4.1	0.6
*PC-DARTS [96]	2.57±0.07	3.6	0.1
*FairDARTS [17]	2.54	3.3	0.4
*DrNAS [15]	2.54±0.03	4.0	0.4
*GTN [87]	2.92±0.06	8.2	0.67
*GTN(F=128) [87]	2.42±0.03	97.9	0.67
*DARTS-1st [58]	3.00±0.14	3.3	0.4
LBT(RN18,DARTS-1st) (ours)	2.87±0.05	3.2	0.6
LBT(RN50,DARTS-1st) (ours)	**2.79**±0.07	3.3	0.7
*DARTS-2nd [58]	2.76±0.09	3.3	1.5
LBT(RN18,DARTS-2nd) (ours)	2.65±0.03	3.4	1.9
LBT(RN50,DARTS-2nd) (ours)	**2.61**±0.05	3.4	2.1
*P-DARTS [16]	2.50	3.4	0.3
LBT(RN18,P-DARTS) (ours)	2.64±0.11	3.4	0.4
LBT(RN50,P-DARTS) (ours)	2.57±0.15	3.4	0.5
*PC-DARTS [96]	2.57±0.07	3.6	0.1
LBT(RN18,PC-DARTS) (ours)	2.59±0.03	3.7	0.1
LBT(RN50,PC-DARTS) (ours)	2.56±0.04	3.7	0.2

Table 6.3: Results on CIFAR-10, 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$^-$ [18], NoisyDARTS [19], and DrNAS [15]. The rest notations are the same as those in Table 6.2.

increase the model size or search cost compared with baselines.

The results on ImageNet are shown in Table 6.4, including top-1 and top-5 classification errors on the test set, number of weight parameters (millions), and search costs (GPU days). LBT(RN50,DARTS-2nd,CIFAR10), which is the architecture searched by applying LBT to DARTS-2nd on CIFAR10 with ResNet-50 as the student, achieves lower error than DARTS-2nd(CIFAR10) which does not use LBT. Similarly, LBT(RN50,P-DARTS,CIFAR10) outperforms P-DARTS(CIFAR10), LBT(RN50,P-DARTS,CIFAR100) outperforms P-DARTS(CIFAR100), and LBT(RN50,PC-DARTS,ImageNet) outperforms PC-DARTS(ImageNet). These results again demonstrate the effectiveness of our method in improving a model by letting it teach another model to learn well.

Method	Top-1 Error (%)	Top-5 Error (%)	Param (M)	Cost (GPU days)
*Inception-v1 [88]	30.2	10.1	6.6	-
*MobileNet [43]	29.4	10.5	4.2	-
*ShuffleNet 2× (v2) [63]	25.1	7.6	7.4	-
*NASNet-A [106]	26.0	8.4	5.3	1800
*PNAS [56]	25.8	8.1	5.1	225
*MnasNet-92 [89]	25.2	8.0	4.4	1667
*AmoebaNet-C [78]	24.3	7.6	6.4	3150
*SNAS-CIFAR10 [95]	27.3	9.2	4.3	1.5
*BayesNAS-CIFAR10 [103]	26.5	8.9	3.9	0.2
*PARSEC-CIFAR10 [12]	26.0	8.4	5.6	1.0
*GDAS-CIFAR10 [25]	26.0	8.5	5.3	0.2
*DSNAS-ImageNet [44]	25.7	8.1	-	-
*SDARTS-ADV-CIFAR10 [14]	25.2	7.8	5.4	1.3
*PC-DARTS-CIFAR10 [96]	25.1	7.8	5.3	0.1
*ProxylessNAS-ImageNet [10]	24.9	7.5	7.1	8.3
*FairDARTS-CIFAR10 [17]	24.9	7.5	4.8	0.4
*FairDARTS-ImageNet [17]	24.4	7.4	4.3	3.0
*DrNAS-ImageNet [15]	24.2	7.3	5.2	3.9
*DARTS$^-$-ImageNet [18]	23.8	7.0	4.9	4.5
*DARTS$^+$-CIFAR100 [53]	23.7	7.2	5.1	0.2
*DARTS-2nd(CIFAR10) [58]	26.7	8.7	4.7	1.5
LBT(RN50,DARTS-2nd,CIFAR10) (ours)	**25.5**	**7.9**	4.8	2.1
*P-DARTS(CIFAR10) [16]	24.4	7.4	4.9	0.3
LBT(RN50,P-DARTS,CIFAR10) (ours)	**24.1**	**7.2**	4.9	0.5
*P-DARTS(CIFAR100) [16]	24.7	7.5	5.1	0.3
LBT(RN50,P-DARTS,CIFAR100) (ours)	**24.2**	**7.1**	5.3	0.6
*PC-DARTS(ImageNet) [96]	24.2	7.3	5.3	3.8
LBT(RN50,PC-DARTS,ImageNet) (ours)	**23.5**	**6.8**	5.4	4.1

Table 6.4: 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$^-$ [18] and DrNAS [15]. The rest notations are the same as those in Table 6.2. From top to bottom, on the first, second, and third panel are manually-designed networks, non-differentiable search methods, and differentiable search methods. LBT(RN50,DARTS-2nd,CIFAR10) means the architecture is searched using LBT on CIFAR-10 with ResNet-50 as the student, where the search space is the same as that in DARTS-2nd. Similar meanings hold for other notations like this.

6.4.4 Ablation Studies

In this section, we perform several ablation studies to better understand the individual learning stages in LBT. For each ablation setting, we compare it with the full LBT framework.

- **Ablation setting 1**. In this setting, the teacher updates its architecture by minimizing the validation loss of the student only, without considering the validation loss of itself. The corresponding formulation is:

$$\begin{aligned} &\min_{A} \;\; L(S^*(T^*(A)), D_s^{(\text{val})}) \\ &s.t. \;\; S^*(T^*(A)) = \min_{S} \;\; L(S, D_s^{(\text{tr})}) + \lambda L(S, D_{pl}(D_u, T^*(A))) \\ &\qquad T^*(A) = \min_{T} L(A, T, D_t^{(\text{tr})}) \end{aligned}$$

In this study, λ is set to 1. The student's architecture is ResNet-18. LBT is applied to DARTS-2nd.

- **Ablation setting 2**. In this setting, in the second stage of LBT, only the pseudo-labeled dataset is used to train the student; $D_s^{(\text{tr})}$ labeled by humans is not used. The corresponding formulation is:

$$\begin{aligned} \min_{A} \quad & L(T^*(A), A, D_t^{(\text{val})})) + \gamma L(S^*(T^*(A)), D_s^{(\text{val})}) \\ s.t. \quad & S^*(T^*(A)) = \min_{S} \; L(S, D_{pl}(D_u, T^*(A))) \\ & T^*(A) = \min_{T} L(A, T, D_t^{(\text{tr})}) \end{aligned}$$

  In this study, γ is set to 1. The student's architecture is ResNet-18. LBT is applied to DARTS-2nd.

- Ablation study on λ. We investigate how the teacher's test error changes with the tradeoff parameter λ in Eq.(6.3). In this study, the other tradeoff parameter γ in Eq.(6.3) is set to 1. For either CIFAR-100 or CIFAR-10, from their training set and validation set, 5K examples are uniformly sampled to form a new test set. Architecture search is performed on the remaining training and validation sets. Architecture evaluation result is reported on the 5K new test set. Student's architecture is ResNet-18. LBT is applied to DARTS-2nd.

- Ablation study on γ. We investigate how the teacher's test error changes with the tradeoff parameter γ in Eq.(6.3). In this study, the other tradeoff parameter λ is set to 1. Similar to the ablation study on λ, the test error is reported on the 5K dataset. Student's architecture is ResNet-18. LBT is applied to DARTS-2nd.

In Table 6.5, we show the classification errors on the test set of CIFAR-10 and CIFAR-100 in ablation setting 1. As can be seen, on both datasets, minimizing both student's validation loss and teacher's validation loss results in better architectures for the teacher than minimizing student's validation only. The reason is that student's validation loss indirectly measures the quality of the teacher's architecture. How well the student performs depends on not only how well the teacher teaches the student but also how strong the student itself is. If the student is a very strong learner, its validation loss may be largely determined by the student itself and less influenced by the teacher. In this case, student's validation would be a relatively weak signal for guiding the learning of the teacher. In contrast, the validation loss of the teacher directly depends on its architecture and can serve as a direct (hence strong) signal to guide the teacher to learn. In the end, combining the direct signal (teacher's validation loss) and indirect signal (student's validation loss) together is more beneficial than using the indirect signal only.

In Table 6.6, we show the classification errors on the test sets of CIFAR-10 and CIFAR-100 in ablation setting 2. We can see that using both the pseudo-labeled dataset and human-labeled dataset to train the student yields better performance than using the pseudo-labeled dataset only. The reason is that since the pseudo-labels are automatically generated by a model, they are not entirely reliable. Trained on less reliable labels, the student's model may have low quality and a poorly-performing student cannot drive the teacher to learn better. This risk can be reduced by incorporating human-provided labels which are more reliable. As a result, using human labels and pseudo-labels jointly yields better performance than solely using pseudo-labels.

In Figure 7.3, we show how the classification errors of LBT on CIFAR-10 and CIFAR-100

Method	Error (%)
Student only (CIFAR-100)	20.27±0.24
Student + Teacher (CIFAR-100)	**17.93**±0.18
Student only (CIFAR-10)	3.01±0.08
Student + teacher (CIFAR-10)	**2.61**±0.05

Table 6.5: Classification errors in ablation setting 1. "Student only" means that only the student's validation loss is used to update the teacher's architecture. "Student + teacher" means that both the student's validation loss and teacher's validation loss are minimized to update the teacher's architecture.

Method	Error (%)
Pseudo labels only (CIFAR-100)	19.82±0.31
Pseudo labels + human labels (CIFAR-100)	**17.93**±0.18
Pseudo labels only (CIFAR-10)	2.93±0.07
Pseudo labels + human labels (CIFAR-10)	**2.61**±0.05

Table 6.6: Classification errors in ablation setting 2. "Pseudo labels only" means in the second learning stage, only the pseudo-labeled dataset is used to train the student. "Pseudo labels + human labels" means both the pseudo-labeled dataset and a human-labeled dataset $D_s^{(\text{tr})}$ are used to train the student.

vary as λ increases. From the plot on CIFAR-100, we observe the following. First, when λ increases from 0.5 to 1, the error decreases. This is because a larger λ incurs a stronger effect of teaching, where the training of the student relies more on the pseudo-labeled dataset created by the teacher. When the teaching effect is strong, the teacher can gain more feedback from the student's performance, which helps the teacher to learn better. Second, if we continue to increase λ, the performance becomes worse. The reason is that if λ is too large, the teaching effect would be excessively strong. Under such circumstances, the student is mainly trained on the pseudo labels which are less reliable than human-provided labels and consequently its model may be of low quality. A mediocre student will not be very helpful in driving the teacher to improve. Similar phenomenon are observed from the plot on CIFAR-10.

In Figure 6.4, we show how the classification errors of LBT vary as we increase γ. As can be seen, on CIFAR-100, when we increase γ from 0.1 to 1, the error decreases. This is because a larger γ encourages the teacher to pay more attention to the feedback obtained from the student. This feedback is valuable because the validation performance of the student reflects the correctness of the pseudo-labels generated by the teacher and the quality of pseudo-labels reflects the quality of the teacher's architecture. Paying more attention to such feedback enables the teacher to identify its weakness and strive for improvement. However, if γ is too large, the learning of the teacher's architecture would be guided excessively by student's validation loss which is an indirect (hence weaker signal) but inadequately influenced the validation loss of the teacher itself which is a direct (hence stronger signal). Similar observations can be made from the plot on CIFAR-10 as well.

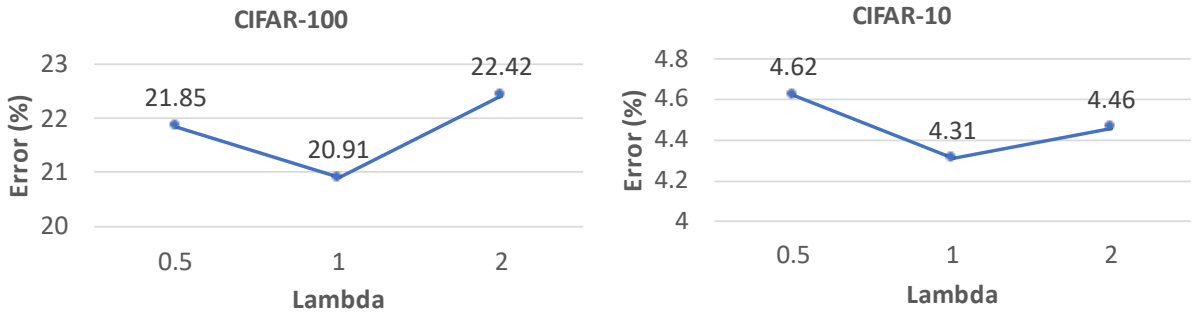

Figure 6.3: How errors change as λ increases.

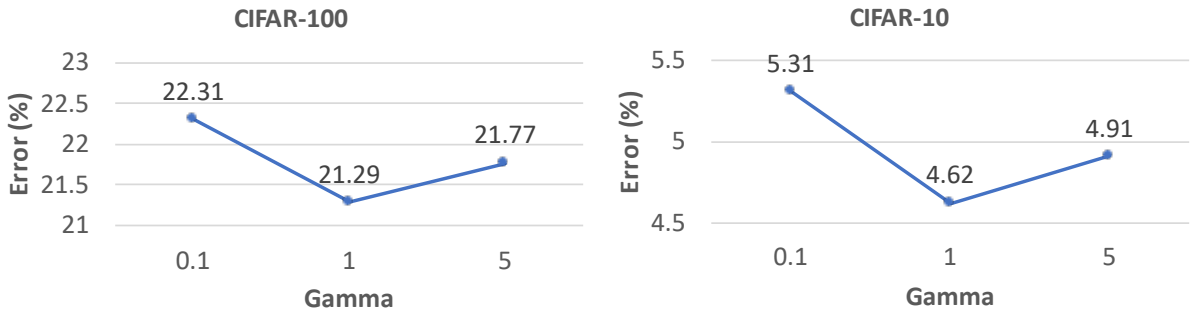

Figure 6.4: How errors change as γ increases.

6.5 Conclusions

Motivated by the teaching-driven learning methodology of humans, we propose a novel machine learning framework called learning by teaching (LBT). LBT improves the training of a teacher model by encouraging the teacher to teach what it has learned to a student model. Based on how the student performs on a validation set, the teacher refines its model. The intuition behind LBT is that to be able to teach others to well accomplish a task, the teacher itself needs to thoroughly understand this task and know how to perform it excellently. We develop a multi-level optimization framework to formulate LBT and the framework consists of three learning stages: the teacher learns; the teacher teaches the student by pseudo-labeling; the teacher improves its architecture based on the validation results of itself and of the student. Our framework is applied for neural architecture search on CIFAR-100, CIFAR-10, and ImageNet. The results demonstrate the effectiveness of our method.

Chapter 7

Learning by Ignoring

7.1 Introduction

In human learning, a widely-practiced effective learning methodology is learning by ignoring. For example, in course learning, given a large collection of practice problems provided in the textbook, the teacher selects a subset of problems as homework for the students to practice instead of using all problems in the textbook. Some practice problems are ignored because 1) they are too difficult which might confuse the students; 2) they are too simple which are not effective in helping the students to practice their knowledge learned during lectures; 3) they are repetitive. The study in [20] shows that learning to ignore certain things is powerful for helping people focus.

Drawing inspirations from this effective human-learning method, we are intrigued in exploring whether this method is helpful for training better machine learning models as well. We propose a novel machine learning framework called learning by ignoring (LBI) (as illustrated in Figure 7.1). In this framework, a model is trained to perform a target task. The model consists of a data encoder and a task-specific head. The encoder is trained in two phrases: pretraining and finetuning, conducted on a pretraining dataset and a finetuning dataset. Pretraining is a commonly used technique in deep learning to learn more effective representations for alleviating overfitting. Given a target task where the amount of training data is limited, training deep neural networks on this small-sized dataset has high risk of overfitting. To address this problem, one can pretrain the feature extraction layers in the network on large-sized external data from some source domain, then finetune these layers on the target data. The abundance of source data enables the network to learn powerful representations that are robust to overfitting. And such representation power can be leveraged to assist in the learning of the target task with more resilience to overfitting.

Some pretraining data examples have a domain difference with the finetuning dataset, rendering them not suitable to pretrain the data encoder that will be used for performing the target task. We would like to identify such out-of-domain data examples and exclude them from the pretraining process. To achieve this goal, we associate each pretraining example with an ignoring variable $a \in [0, 1]$. If a is close to 0, it means that this example should be ignored. We develop a three-level framework (involving three learning stages) to automatically learn these ignoring

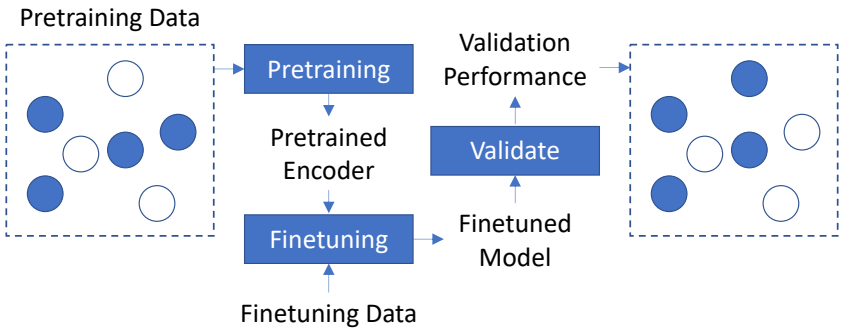

Figure 7.1: Illustration of learning by ignoring. Void circles denote ignored pretraining data examples. Given a set of intermediately selected pretraining examples, they are used to pretrain a model, which is then finetuned and validated. The validation performance provides guidance on what pretraining examples should be ignored in the next round of learning. This process iterates until convergence.

variables. In the first learning stage, we pretrain a data encoder V by minimizing a weighted pretraining loss: the loss defined on each pretraining example is multiplied with the ignoring variable of this example. If an example x should be ignored, its a is close to 0; multiplying a to the loss of x makes this loss close to 0, which effectively excludes x from the pretraining process. In this stage, these ignoring variables $A = \{a\}$ are fixed. They will be updated at a later stage. Note that the optimally pretrained encoder $V^*(A)$ is a function of A since $V^*(A)$ is a function of the weighted loss and the weighted loss is a function of A. In the second stage, we train another data encoder W on the finetuning dataset. During the training of W, we encourage it to be close to the optimal encoder $V^*(A)$ trained in the first stage by minimizing the squared L2 distance between W and $V^*(A)$. $V^*(A)$ contains information distilled from non-ignored pretraining examples. Encouraging W to be close to $V^*(A)$ effectively achieves the goal of pretraining W on these non-ignored examples. Note that the optimally trained encoder $W^*(V^*(A))$ is a function of $V^*(A)$. In the third stage, we apply $W^*(V^*(A))$ to make predictions on the validation set of the target task and update A by minimizing the validation loss. To summarize, we aim to learn an ignoring variable for each pretraining example so that the data encoder pretrained on non-ignored examples achieves the optimal performance on the validation set after finetuning. The three stages are conducted end-to-end in a joint manner. Experiments on various datasets demonstrate the effectiveness of our method.

The major contributions of this chapter is as follows:

- Drawing inspirations from the ignoring-driven learning methodology of humans, we propose a novel machine learning framework called learning by ignoring (LBI). Our framework automatically identifies pretraining data examples that have large domain shift from the target distribution by learning an ignoring variable for each example and excludes them from the pretraining process.
- We formulate LBT as a multi-level optimization framework involving three learning stages: pretraining by minimizing the losses weighed by ignoring variables; finetuning; updating the ignoring variables by minimizing the validation loss.

- An efficient gradient-based algorithm is developed to solve the LBI problem.
- Experiments on various datasets demonstrate the effectiveness of our method.

7.2 Related Works

Due to space limit, we defer the review of domain adaptation works to the supplements.

Data Re-weighting and Selection. Several approaches have been proposed for data selection. Matrix column subset selection [7, 23] aims to select a subset of data examples that can best reconstruct the entire dataset. Similarly, coreset selection [2] chooses representative training examples in a way that models trained on the selected examples have comparable performance with those trained on all training examples. These methods perform data selection and model training separately. As a result, the validation performance of the model cannot be used to guide data selection. Ren et al. [79] proposes a meta learning method to learn the weights of training examples by performing a meta gradient descent step on the weights of the current mini-batch of examples. Shu et al. [83] propose a method which can adaptively learn an explicit weighting function directly from data. These works focus on selecting target examples using an one-step meta-learning framework while our work focuses on selecting source examples using a three-level optimization framework.

Pretraining. Arguably, the most popular pretraining approach is supervised pretraining (SP) [71], which learns the weight parameters of a representation network by solving a supervised source task, i.e., correctly mapping input data examples to their labels (e.g., classes, segmentation masks, etc.). Recently, self-supervised learning [13, 66, 70**?**], as an unsupervised pretraining approach, has achieved promising success and outperforms supervised pretraining in a wide range of applications. Similar to SP, self-supervised pretraining (SSP) also solves predictive tasks. But the output labels in SSP are constructed from the input data, rather than annotated by humans as in SP. The auxiliary predictive tasks in SSP could be predicting whether two augmented data examples originate from the same original data example [13, 39, 66**?**], inpainting masked regions in images [74], etc.

7.3 Methods

Our framework aims to learn a machine learning model for accomplishing a target task T. The ML model is composed of a data encoder W and a task-specific head H. For instance, in a text classification task, the data encoder can be a BERT [24] model which produces an embedding of the input text and the classification head can be a multi-layer perceptron which predicts the class label of this text based on its embedding. The learning is performed in two phrases: pretraining and finetuning. In the pretraining phrase, we pretrain the data encoder W on a pretraining dataset $D^{(\text{pre})} = \{d_i^{(\text{pre})}\}_{i=1}^{M}$ by solving a pretraining task P. P could be the same as T. In this case, W and the task-specific head H are trained jointly on $D^{(\text{pre})}$. P could be different from T as well. Under such circumstances, P has its own task-specific head J while sharing the same encoder W

Notation	Meaning
M	Number of pretraining examples
N	Number of finetuning examples
O	Number of validation examples
$d_i^{(\text{pre})}$	The i-th pretraining data example
a_i	Ignoring variable of $d_i^{(\text{pre})}$ in pretraining
b_i	Ignoring variable of $d_i^{(\text{pre})}$ in finetuning
$d_i^{(\text{tr})}$	The i-th finetuning data example
$d_i^{(\text{val})}$	The i-th validation example
W	Encoder in finetuning
V	Encoder in pretraining
H	Head of the target task
J	Head of the pretraining task

Table 7.1: Notations in learning by ignoring

with T. J and W are trained jointly on $D^{(\text{pre})}$. Oftentimes, some pretraining data examples have a domain shift with the finetuning dataset. This domain shift renders these examples not suitable for pretraining the encoder. We aim to automatically identify such examples using ignoring variables and exclude them from the pretraining process. To achieve this goal, we multiply the loss of a pretraining example x with an ignoring variable $a \in [0, 1]$. If a is close to 0, it means that this example should be ignored; accordingly, the loss (after multiplied with a) is made close to 0, which effectively excludes x from the pretraining process. We aim to automatically learn the values of these ignoring variables, which will be detailed later. After pretraining, the encoder is finetuned on the training dataset $D^{(\text{tr})} = \{d_i^{(\text{tr})}\}_{i=1}^N$. For mathematical convenience, we formulate "pretraining" in the following way: in the pretraining phrase, we train another encoder V; in the finetuning phrase, we encourage the encoder W to be close to the optimally pretrained encoder V^* where the closeness is measured using squared L2 distance $\|W - V^*\|_2^2$.

Overall, the learning is performed in three stages. In the first stage, the model trains the encoder V and the head J specific to the pretraining task P on the pretraining dataset $D^{(\text{pre})} = \{d_i^{(\text{pre})}\}_{i=1}^M$, with the ignoring variables $A = \{a_i\}_{i=1}^M$ fixed:

$$V^*(A) = \min_{V,J} \sum_{i=1}^{M} a_i L(V, J, d_i^{(\text{pre})}). \tag{7.1}$$

After training, the optimal head is discarded. The optimal encoder $V^*(A)$ is retained for future use. The ignoring variables A are needed to make predictions and calculate training losses. But they should not be updated in this stage. Otherwise, the values of A will all be zero. Note that $V^*(A)$ is a function of A since it is a function of $\sum_{i=1}^M a_i L(V, J, d_i^{(\text{pre})})$ and $\sum_{i=1}^M a_i L(V, J, d_i^{(\text{pre})})$ is a function of A.

In the second stage, the model trains its data encoder W and task-specific head H by minimizing the training loss of the target task T. During training, W is encouraged to be close to $V^*(A)$ trained in the pretraining phrase by minimizing the squared L2 distance $\|W - V^*(A)\|_2^2$.

This implicitly achieves the effect of pretraining W on the non-ignored pretraining examples. The optimization problem in the second stage is:

$$W^*(V^*(A)), H^* = \min_{W,H} \sum_{i=1}^{N} L(W, H, d_i^{(\text{tr})}) + \lambda \|W - V^*(A)\|_2^2, \tag{7.2}$$

where λ is a tradeoff parameter. Note that $W^*(V^*(A))$ is a function of $V^*(A)$ since it is a function of $\|W - V^*(A)\|_2^2$, which is a function of $V^*(A)$.

In the third stage, we use the trained model consisting of $W^*(V^*(A))$ and H^* to make predictions on the validation dataset $D^{(\text{val})}$ of the target task T. We update A by minimizing the validation loss.

$$\min_A \sum_{i=1}^{O} L(W^*(V^*(A)), H^*, d_i^{(\text{val})}). \tag{7.3}$$

The three stages mutually influence each other: $V^*(A)$ trained in the first stage is needed to calculate the loss function in the second stage; $W^*(V^*(A))$ trained in the second stage is needed to calculate the objective function in the third stage; the ignoring variables A updated in the third stage alter the loss function in the first stage, which subsequently changes $V^*(A)$ and $W^*(V^*(A))$ as well.

Putting the three learning stages together, we formulate LBI as the following three-level optimization problem:

$$\begin{array}{ll} \min_A & \sum_{i=1}^{O} L(W^*(V^*(A)), H^*, d_i^{(\text{val})}) \\ s.t. & W^*(V^*(A)), H^* = \min_{W,H} \sum_{i=1}^{N} L(W, H, d_i^{(\text{tr})}) + \lambda \|W - V^*(A)\|_2^2 \\ & V^*(A) = \min_{V,J} \sum_{i=1}^{M} a_i L(V, J, d_i^{(\text{pre})}) \end{array} \tag{7.4}$$

This formulation consists of three optimization problems. The two inner optimization problems (on the constraints) represent the first and second learning stage respectively. The outer optimization problem represents the third learning stage. The three stages are illustrated in Figure 7.2.

If the pretaining task and target task are the same, in the second stage we can use the pretraining data to train W as well. Due to domain difference, not all pretraining examples are suitable for training W. To exclude such examples, we associate each pretraining example $d_i^{(\text{pre})}$ with an ignoring variable $b_i \in [0, 1]$. Note that b_i is different from a_i. b_i is used to determine whether $d_i^{(\text{pre})}$ should be ignored during training W and a_i is used to determine whether $d_i^{(\text{pre})}$ should be ignored during training V. The corresponding formulation is:

$$\begin{array}{ll} \min\limits_{A,B} & \sum_{i=1}^{O} L(W^*(V^*(A), B), H^*(B), d_i^{(\text{val})}) \\ s.t. & W^*(V^*(A), B), H^*(B) = \min\limits_{W,H} \sum\limits_{i=1}^{N} L(W, H, d_i^{(\text{tr})}) + \lambda \|W - V^*(A)\|_2^2 + \gamma \sum\limits_{i=1}^{M} b_i L(W, H, d_i^{(\text{pre})}) \\ & V^*(A) = \min\limits_{V,J} \sum_{i=1}^{M} a_i L(V, J, d_i^{(\text{pre})}) \end{array} \tag{7.5}$$

where γ is a tradeoff parameter and $B = \{b_i\}_{i=1}^{M}$. Note that W^* and H^* are both functions of B.

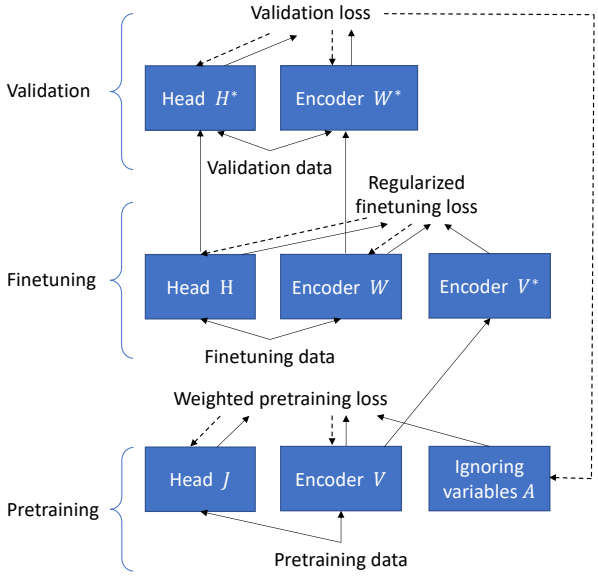

Figure 7.2: Learning by ignoring. Along the solid arrows, predictions are made and train/validation losses are calculated. Along the dotted arrows, gradients are calculated and parameters (including ignoring variables and network weights) are updated.

7.3.1 Optimization Algorithm

In this section, we develop a gradient-based optimization algorithm to solve the three-level optimization problem in Eq.(7.4). Drawing inspirations from [58], we approximate $V^*(A)$ using:

$$V' = V - \xi_V \nabla_V \sum_{i=1}^{M} a_i L(V, J, d_i^{(\text{pre})}), \tag{7.6}$$

where ξ_V is a learning rate. We update J as:

$$J \leftarrow J - \xi_J \sum_{i=1}^{M} a_i \nabla_J L(V, J, d_i^{(\text{pre})}). \tag{7.7}$$

Substituting V' into $\sum_{i=1}^{N} L(W, H, d_i^{(\text{tr})}) + \lambda \|W - V^*(A)\|_2^2$, we obtain an approximated objective $\sum_{i=1}^{N} L(W, H, d_i^{(\text{tr})}) + \lambda \|W - V'\|_2^2$. Then we calculate the gradient of O_W w.r.t to W and approximate $W^*(V^*(A))$ using one-step gradient descent update of W:

$$W' = W - \xi_W \nabla_W (\textstyle\sum_{i=1}^{N} L(W, H, d_i^{(\text{tr})}) + \lambda \|W - V'\|_2^2) \tag{7.8}$$

Similarly, we approximate H^* with:

$$H' = H - \xi_H \nabla_H (\textstyle\sum_{i=1}^{N} L(W, H, d_i^{(\text{tr})}) + \lambda \|W - V'\|_2^2) \tag{7.9}$$

Finally, we substitute W' and H' into $\sum_{i=1}^{O} L(W^*(V^*(A)), H^*, d_i^{(\text{val})})$ and get $O_A = \sum_{i=1}^{O} L(W', H', d_i^{(\text{val})})$. We calculate the gradient of O_A w.r.t A and update A by descending the gradient:

$$A \leftarrow A - \xi_A \nabla_A \textstyle\sum_{i=1}^{O} L(W', H', d_i^{(\text{val})}) \tag{7.10}$$

			A→W	A→D	D→W	D→A	W→D	W→A	Average
DAN-1 [60]			78.11	67.27	91.12	50.66	98.18	53.85	73.20
DAN-2 [60]			95.86	98.18	96.45	86.12	97.27	87.05	93.49
DAN-3 [60]			97.04	96.36	97.04	**88.74**	**100.0**	87.24	94.40
DANN-1 [32]			78.70	68.18	87.57	46.72	94.55	50.28	71.00
DANN-2 [32]			94.08	93.64	94.67	85.55	95.45	84.99	91.40
DANN-3 [32]			96.45	94.55	**97.63**	86.68	**100.0**	85.37	93.45
CDAN-1 [61]			83.84	71.82	94.08	56.47	97.27	60.98	77.41
CDAN-2 [61]			94.67	95.45	96.45	86.68	97.27	84.80	92.55
CDAN-3 [61]			97.04	95.45	**97.63**	85.93	99.09	87.24	93.73
MME-1 [82]			60.95	50.91	88.17	40.71	95.45	41.28	62.91
MME-2 [82]			95.86	96.36	96.45	84.05	99.09	85.74	92.93
MME-3 [82]			94.67	90.90	**97.63**	87.24	**100.0**	84.43	92.48
	Pretrain	Finetune							
Ablation 1	No	No source	96.45	97.48	**97.63**	85.46	99.09	85.37	93.58
Ablation 2	No	Full source	96.45	93.64	**97.63**	87.43	99.09	86.12	93.39
Ablation 3	No	Weighted source	98.22	97.27	**97.63**	86.68	98.18	86.87	94.14
Ablation 4	Full source	No source	97.04	97.27	**97.63**	85.74	98.18	86.38	93.71
Ablation 5	Full source	Full source	95.86	95.45	97.04	87.05	**100.0**	86.30	93.62
Ablation 6	Full source	Weighted source	97.63	96.36	**97.63**	88.18	99.09	**87.99**	94.48
Ablation 7	Weighted source	No source	96.45	**99.09**	97.04	86.49	99.09	86.49	94.11
Ablation 8	Weighted source	Full source	95.86	91.82	**97.63**	88.37	**100.0**	87.05	93.46
Full LBI (ours)	Weighted source	Weighted source	**98.82**	**99.09**	97.04	87.80	99.09	86.87	**94.79**

Table 7.2: Accuracy (%) on the Office31 dataset. In the $A \rightarrow B$ notion, A denotes the source data and B denotes the target data. 1, 2, 3 in DAN, DANN, CDAN, MME denote unsupervised, semi-supervised, and supervised domain adaptation settings respectively.

Algorithm 7: Optimization algorithm for LBI

while *not converged* **do**
1. Update encoder V in pretraining phrase using Eq.(7.6)
2. Update head J in pretraining using Eq.(7.7)
3. Update encoder W in finetuning phrase using Eq.(7.8)
4. Update head H in finetuning using Eq.(7.9)
5. Update ignoring variables A using Eq.(7.10)

end

7.4 Experiments

In this section, we present experimental results. Please refer to the supplements for more hyper-parameter settings and additional results.

7.4.1 Datasets

We perform experiments on three datasets: OfficeHome [92], Office31 [81], and ImageCLEF [68]. OfficeHome consists of 15,500 images of daily objects from 65 categories and 4 domains, including Art (Ar), Clipart (Cl), Product (Pr), and Real-World (Rw). Each category has an average

			C→P	C→I	P→C	P→I	I→C	I→P	Average
DAN-1 [60]			68.15	76.40	89.19	78.88	88.51	66.88	78.00
DAN-2 [60]			61.78	83.85	93.92	83.85	92.57	63.06	79.84
DAN-3 [60]			64.33	81.37	93.25	85.71	93.24	68.79	81.12
DANN-1 [32]			61.78	73.29	84.46	77.02	75.68	66.24	73.08
DANN-2 [32]			63.06	83.23	93.92	85.71	93.24	64.33	80.58
DANN-3 [32]			68.43	83.85	95.27	88.20	91.89	69.26	82.82
CDAN-1 [61]			64.97	70.81	70.27	78.26	90.54	64.33	73.20
CDAN-2 [61]			64.42	85.71	95.27	85.09	95.27	67.52	82.21
CDAN-3 [61]			64.97	87.58	93.24	**89.44**	94.59	67.52	82.89
MME-1 [82]			57.96	68.32	75.00	72.05	85.14	64.97	70.57
MME-2 [82]			61.15	80.12	93.92	81.99	93.92	59.97	78.51
MME-3 [82]			64.97	80.75	88.51	84.47	91.22	61.78	78.62
	Pretrain	Finetune							
Ablation 1	No	No source	59.87	86.34	91.89	84.47	91.22	65.61	79.90
Ablation 2	No	Full source	68.02	86.34	93.24	86.04	90.54	67.39	81.93
Ablation 3	No	Weighted source	67.52	88.20	94.59	84.47	95.95	66.88	82.94
Ablation 4	Full source	No source	62.42	87.28	93.87	83.79	93.72	63.43	80.75
Ablation 5	Full source	Full source	68.79	86.96	94.59	85.09	93.92	68.15	82.92
Ablation 6	Full source	Weighted source	68.15	**89.44**	**95.45**	86.95	**97.30**	68.15	84.24
Ablation 7	Weighted source	No source	63.06	87.58	93.23	86.34	93.24	64.33	81.30
Ablation 8	Weighted source	Full source	66.88	85.09	93.24	85.71	94.59	66.88	82.07
Full LBI (ours)	Weighted source	Weighted source	**70.70**	**89.44**	93.24	87.58	95.95	**70.06**	**84.50**

Table 7.3: Accuracy (%) on the ImageCLEF dataset. In the $A \rightarrow B$ notion, A denotes the source data and B denotes the target data. 1, 2, 3 in DAN, DANN, CDAN, MME denote unsupervised, semi-supervised, and supervised domain adaptation settings respectively.

of about 70 images and a maximum of 99 images. Office31 contains 4,110 images belonging to 31 classes and 3 domains, including Amazon website (A), web camera (W), and digital SLR camera (D). The number of images in domain A, W, and D is 2817, 795, and 498 respectively. ImageCLEF consists of 1,800 images from 3 datasets (domains) with 12 classes, including Caltech-256 (C), ILSVRC2012 (I), and Pascal VOC2012 (P). We split OfficeHome, Office31, and ImageCLEF into train/validation/test sets with a ratio of 5:3:2, 6:2:2, and 6:2:2 respectively. For each dataset, we perform domain adaptation [60] studies: one domain is selected as source and another domain is selected as target; data in the source domain is leveraged to help with model training in the target domain. In the learning-by-ignoring framework, source data is used as pretraining data and target data is used as finetuning data.

7.4.2 Baselines

We compare with the following domain adaptation (DA) baselines. For each baseline, we experimented with three DA settings: 1) the labels of source examples are used for DA and the labels of target examples are not; 2) the labels of target examples are used for DA and the labels of source examples are not; 3) both the labels of source examples and target examples are used for DA.

- **DAN** [60], which matches mean embeddings of different domain distributions in a reproducing kernel Hilbert space.

			Ar→Cl	Ar→Pr	Ar→Rw	Cl→Ar	Cl→Pr	Cl→Rw	Pr→Ar
DAN-1 [60]			30.86	46.25	54.19	33.54	41.22	45.03	33.33
DAN-2 [60]			67.66	86.77	72.74	57.51	87.59	69.50	62.14
DAN-3 [60]			68.46	**87.94**	75.53	59.05	85.13	72.29	60.29
DANN-1 [32]			33.37	43.56	55.75	40.95	47.66	50.50	37.24
DANN-2 [32]			67.09	83.37	73.63	55.56	82.79	69.83	58.64
DANN-3 [32]			68.80	85.60	72.96	60.08	86.53	72.74	59.88
CDAN-1 [61]			37.49	49.77	59.55	41.56	51.99	54.41	41.77
CDAN-2 [61]			68.23	84.43	73.41	60.49	84.66	71.84	59.67
CDAN-3 [61]			67.31	86.65	73.52	61.93	84.66	71.06	61.93
MME-1 [82]			28.11	40.40	52.40	29.84	35.71	40.34	29.22
MME-2 [82]			68.23	84.66	70.73	55.35	85.01	66.37	58.44
MME-3 [82]			69.60	85.01	71.96	56.97	84.66	67.93	52.47
	Pretrain	Finetune							
Ablation 1	No	No source	66.40	87.47	73.18	65.23	87.24	72.96	64.40
Ablation 2	No	Full source	68.69	85.36	74.75	60.08	86.07	72.29	59.47
Ablation 3	No	Weighted source	68.00	86.42	75.08	62.14	86.53	73.52	63.37
Ablation 4	Full source	No source	69.71	**87.94**	**76.65**	67.28	88.17	76.54	65.02
Ablation 5	Full source	Full source	67.54	85.13	74.75	60.08	84.54	71.17	61.11
Ablation 6	Full source	Weighted source	68.00	84.66	74.64	61.73	86.3	71.84	63.17
Ablation 7	Weighted source	No source	**70.51**	87.70	76.20	**69.75**	**89.34**	**77.65**	**68.52**
Ablation 8	Weighted source	Full source	67.66	85.25	74.41	61.52	86.77	73.30	61.52
Full LBI (ours)	Weighted source	Weighted source	69.03	84.66	75.42	61.32	85.83	73.52	62.76

			Pr→Cl	Pr→Rw	Rw→Ar	Rw→Cl	Rw→Pr	Average
DAN-1 [60]			28.69	52.96	51.03	33.37	65.69	43.01
DAN-2 [60]			70.40	74.19	65.23	68.11	86.07	72.33
DAN-3 [60]			70.51	75.75	63.79	69.14	86.89	72.90
DANN-1 [32]			33.37	56.54	52.67	42.74	70.73	47.09
DANN-2 [32]			65.60	72.29	60.70	68.00	86.07	70.30
DANN-3 [32]			68.80	73.18	64.61	68.46	87.70	72.45
CDAN-1 [61]			41.14	58.21	56.58	41.83	73.54	50.65
CDAN-2 [61]			69.49	73.41	63.58	67.54	88.06	72.07
CDAN-3 [61]			**70.74**	72.63	65.23	69.94	87.35	72.75
MME-1 [82]			26.97	48.04	46.71	33.71	61.71	39.43
MME-2 [82]			67.31	69.50	59.26	66.51	86.18	69.80
MME-3 [82]			69.14	73.07	62.96	68.80	85.83	70.70
	Pretrain	Finetune						
Ablation 1	No	No source	68.91	73.85	60.70	69.14	86.77	73.02
Ablation 2	No	Full source	67.77	74.75	66.16	69.60	87.47	72.71
Ablation 3	No	Weighted source	69.94	74.53	65.02	**70.17**	89.23	73.66
Ablation 4	Full source	No source	69.71	76.20	**68.96**	69.37	88.06	75.30
Ablation 5	Full source	Full source	66.74	71.84	64.61	66.63	87.00	71.76
Ablation 6	Full source	Weighted source	66.97	74.19	65.43	67.54	88.41	72.74
Ablation 7	Weighted source	No source	70.63	**77.65**	67.90	69.83	**90.52**	**76.35**
Ablation 8	Weighted source	Full source	67.43	74.19	68.31	68.34	86.18	72.91
Full LBI (ours)	Weighted source	Weighted source	69.94	74.41	67.28	67.89	86.18	73.19

Table 7.4: Accuracy (%) on the OfficeHome dataset.

- **DANN** [32], which uses adversarial learning to learn representations so that source domain and target domain are not distinguishable.
- **CDAN** [61], which conditions adversarial adaptation on discriminative information.
- **MME** [82], which is a semi-supervised domain adaptation method based on minimax entropy.

We also compared with data re-weighting methods including [79, 83]. These methods focus on re-weighting target data while our methods focus on re-weighting source data. Please refer to the supplements for details.

7.4.3 Experimental Settings

We use ResNet-34 [37] pretrained on ImageNet [22] as the backbone. Our models were trained using the Adam [47] optimizer, with a batch size of 64, a learning rate of 1e-4 for the feature extractor, a learning rate of 1e-3 for the classifier, a weight decay of 5e-4, for 50 epochs. The learning rate was decreased by a factor of 10 after 40 epochs. In learning by ignoring, γ is set to 1e-2 for all datasets; λ is set to 7e-3 for OfficeHome, 5e-4 for Office31, and 3e-3 for ImageCLEF.

7.4.4 Results

Table 7.2 shows the results on the Office31 dataset. In the $A \rightarrow B$ notion, A denotes the source dataset and B denotes the target dataset. As can be seen, our proposed LBI method achieves better averaged accuracy than the domain adaptation (DA) baselines including DAN, DANN, CDAN, and MME. The major reason is that our method automatically identifies source examples that are not suitable for pretraining due to source-target domain shift and excludes them from the pretraining process. In contrast, these DA baselines adapt all source examples into the target domain and lack the ability of explicitly identifying source examples that are not suitable for domain adaptation. Table 7.3 shows the results on the ImageCLEF dataset. Our proposed LBI achieves better average accuracy than the DA baselines, thanks to its mechanism of ignoring source examples that are not suitable for helping with the learning on the target domain. Table 7.4 shows the results on the OfficeHome dataset. LBI outperforms all DA methods, which further demonstrates the effectiveness of our proposed learning-by-ignoring framework. On this dataset, using source dataset for finetuning leads to worse performance. This is probably because the source domains in this dataset have large domain shifts with the target domains. Therefore, using source data directly for finetuning may not be proper. However, using non-ignored source data for pretraining is helpful.

7.4.5 Ablation Studies

We perform ablation studies to check the effectiveness of individual modules in our framework. Unless otherwise notified, γ was set to 1e-2 for all three datasets; λ was set to 7e-3 for OfficeHome, 5e-4 for Office31, and 3e-3 for ImageCLEF.

- Ablation setting 1: no pretraining, finetune on target data only. We ignore the source dataset and directly train a model on the target dataset. This is equivalent to setting both λ and γ in LBI (Eq.(7.5)) to 0.
- Ablation setting 2: no pretraining, finetune on target data and all source examples. There is no pretraining; we combine the source dataset and the target dataset as a single dataset, then train a model on the combined dataset. This is equivalent to setting λ to 0 and setting all ignoring variables in B to 1 in LBI (Eq.(7.5)).
- Ablation setting 3: no pretraining, finetune on target data and weighted source examples. There is no pretraining; the model is directly trained on all target examples and selected source examples. This is equivalent to setting the tradeoff parameter λ to 0 in the LBI framework (Eq.(7.5)).

- Ablation setting 4: pretrain on all source examples, finetune on target data only. We first train a model M_1 on the full source dataset. Then we train a model M_2 on the target dataset. When training M_2, we encourage its weights to be close to the optimally trained weights of M_1 by minimizing their squared L2 distance. This is equivalent to setting γ to 0 and setting all ignoring variables in A to 1 in LBI (Eq.(7.5)).
- Ablation setting 5: pretrain on all source examples, finetune on target and all source examples. This setting is similar to ablation 4, except that M_2 is trained on target and full source dataset. This is equivalent to setting ignoring variables in A and B to 1 in LBI (Eq.(7.5)).
- Ablation setting 6: pretrain on all source examples, finetune on target data and weighted source examples. This setting is similar to ablation 4, except that M_2 is trained on the target and weighted source dataset. This is equivalent to setting all ignoring variables in A to 1 in LBI (Eq.(7.5)).
- Ablation setting 7: pretrain on weighted source examples, finetune on target data only. This is equivalent to setting the tradeoff parameter γ to 0 in the LBI framework (Eq.(7.5)).
- Ablation setting 8: pretrain on weighted source examples, finetune on target data and all source examples. In this setting, all source examples are used for finetuning, without ignoring any of them. This is equivalent to setting all ignoring variables in B to 1 in the LBI framework (Eq.(7.5)).
- Ablation study on λ. We explore how the performance varies as the tradeoff parameter λ in Eq.(7.5) increases. In this study, the other tradeoff parameter γ in Eq.(7.5) is set to 1e-2. We report the average accuracy on the validation set. The experiments are conducted on ImageCLEF and OfficeHome. Please refer to the supplements for results on OfficeHome.
- Ablation study on γ. We explore how the performance varies as γ increases. We report the average accuracy on the validation set. The experiments are conducted on ImageCLEF and OfficeHome. In this study, the other tradeoff parameter λ is set to 3e-3 for ImageCLEF and 7e-3 for OfficeHome. Please refer to the supplements for results on OfficeHome.

Table 7.2 shows the ablation study results on the Office31 dataset. From this table, we make the following observations. **First**, during pretraining, ignoring certain source examples is better than using all source examples. For instance, ablation 7 achieves better average accuracy than ablation 4 where the former performs pretraining on weighted source data and the latter uses all source examples for pretraining. The rest settings of ablation 7 and 4 are the same. As another example, our proposed full LBI framework achieves higher average accuracy than ablation 6 where the former performs ignoring of certain source examples during pretraining while the latter does not. The rest settings of full LBI are the same as those of ablation 6. The reason is that due to domain shift between source and target, some source examples are largely different from the target examples; if using such source examples for pretraining, the pretrained encoder may be biased to the source distribution and cannot well represent target examples. Our proposed approaches automatically identify source examples that have large domain discrepancy with the target and remove them from the pretraining process. Consequently, the pretrained encoder with ignoring is better than the encoder pretrained using all source examples without ignoring. **Second**, when using source examples to finetune the encoder (together with target examples), it is beneficial to ignore some source examples in the finetuning process. As can

be seen, in terms of average accuracy, the full LBI framework performs better than ablation 8 where the former ignores certain source examples during finetuning while the latter does not. The rest settings of these two methods are the same. Similarly, ablation 6 outperforms ablation 5; ablation 3 achieves higher average accuracy than ablation 2. Again, the reason is that due to domain shift, some source examples are not suitable for finetuning the encoder of the target dataset. Our proposed ignoring approach excludes such source examples from finetuning, hence achieving better performance than not using ignoring. **Third**, without the ignoring mechanism, using source examples for finetuning is harmful. For example, ablation 2 which uses all source samples for finetuning achieves lower average accuracy than ablation 1 which uses no source example for finetuning. Similarly, ablation 5 achieves worse average accuracy than ablation 4; ablation 8 performs less well than ablation 7. However, once we add the ignoring mechanism and use non-ignored source examples for finetuning, the resulting average accuracy is higher than not using any source example for finetuning, as can be seen from the average accuracy results that ablation 3 outperforms ablation 1; ablation 6 outperform ablation 4; and the full LBI method outperforms ablation 7. This further demonstrates the effectiveness of our proposed ignoring mechanism, which can 1) identify sources examples that are close to the target domain and use them for finetuning; 2) recognize source example having large domain shifts from the target and exclude them from finetuning. In contrast, the baseline methods can only do the black-or-white choice: either using all source examples for finetuning or not using any source example at all, which hence leads to inferior performance.

Table 7.3 shows ablation results on ImageCLEF. From this table, we make similar observations as in Table 7.2. **First**, in pretraining, the ignoring mechanism which identifies source examples that have large domain shifts from the target and excludes them from the pretraining process achieves better performance than not using ignoring. This is evidenced from the average accuracy results that ablation 7 outperforms ablation 4; our full LBI framework achieves better accuracy than ablation 6. **Second**, during finetuning, ignoring certain source examples works better than using all source examples. This is demonstrated by the average accuracy results that the full LBI outperforms ablation 8; ablation 6 outperforms ablation 5; and ablation 3 outperforms ablation 2.

Table 7.4 shows the ablation results on OfficeHome. The observations from these results are similar to those made in Table 7.2 and 7.3. **First**, performing ignoring on source during pretraining is beneficial, as seen from the average accuracy results that ablation 7 outperforms ablation 4; ablation 8 outperforms ablation 5; and the full LBI outperforms ablation 6. **Second**, performing ignoring on source during finetuning is advantageous than not using ignoring, as demonstrated by the average accuracy results that ablation 3 performs better than ablation 2; ablation 6 outperforms ablation 5; and the full LBI framework outperforms ablation 8.

Figure 7.3(Left) shows how the average accuracy on ImageCLEF varies as the tradeoff parameter λ increases. As can be seen, when λ increases from 0.001 to 0.007, the accuracy increases. This is because a larger λ results in a stronger effect of pretraining. The pretrained encoder captures information of non-ignored source examples and such information is valuable for finetuning. However, if λ continues to increase, the accuracy starts to decrease. The reason is that an excessively large λ leads to too much emphasis on the pretrained encoder and pays less attention to the finetuning on the target data. Figure 7.3(Right) shows how the average accuracy varies as the tradeoff parameter γ increases. When γ increases from 0.001 to 0.01, the accuracy

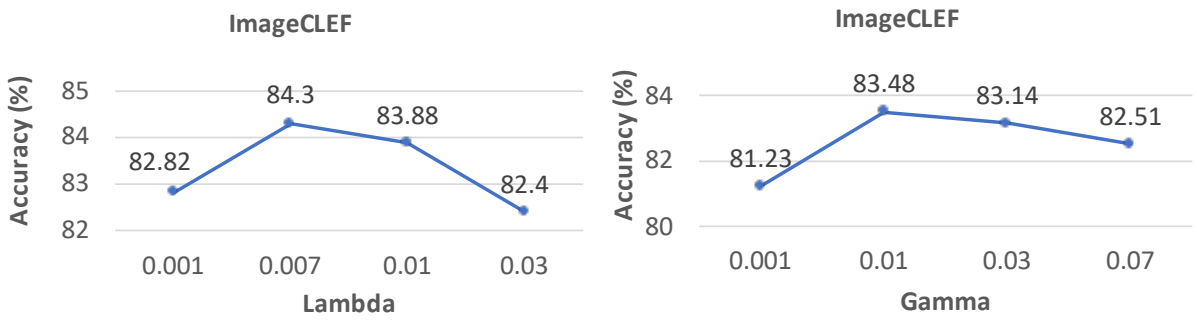

Figure 7.3: (Left) How accuracy changes as λ increases. (Right) How accuracy changes as γ increases.

increases. This is because a larger γ imposes a stronger utilization of non-ignored source examples as additional data for finetuning. However, further increasing γ leads to a worse accuracy. The reason is that if γ is too large, the source data will dominate target data.

7.5 Conclusions

In this chapter, we propose a novel machine learning framework – learning by ignoring (LBI), motivated by the ignoring-driven learning methodology of humans. In LBI, an ignoring variable is learned for each pretraining data example to identify examples that have significant domain difference with the target distribution. We formulate LBI as a three-level optimization problem which consists of three learning stages: an encoder is trained by minimizing a weighted pretraining loss where the loss of each data example is weighted by its ignoring variable; another encoder is finetuned where during the finetuning this encoder is encouraged to be close to the previously pretrained encoder; the ignoring variables are updated by minimizing the validation loss calculated using the finetuned encoder. We conduct experiments on various datasets where the results demonstrate the effectiveness of our method.

Bibliography

[1] David Alvarez-Melis and Tommi S Jaakkola. Towards robust interpretability with self-explaining neural networks. *arXiv preprint arXiv:1806.07538*, 2018. 4.2.2

[2] Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. 7.2

[3] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(Jun):1803–1831, 2010. 4.2.2

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 1.4, 2.2

[5] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100, 1998. 5.2

[6] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019. 5.2

[7] Christos Boutsidis, Michael W Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 968–977. SIAM, 2009. 7.2

[8] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 5.2

[9] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018. 1.4

[10] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 2.2, 2.4.3, 3.2, 3.4.4, 4.2.1, 4.4.3, 5.2, 5.4.3, 6.2.1, 6.4.3

[11] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017. 5.2, 6.2.2

[12] Francesco Paolo Casale, Jonathan Gordon, and Nicoló Fusi. Probabilistic neural architec-

ture search. *CoRR*, abs/1902.05116, 2019. 2.4.3, 3.4.4, 4.4.3, 5.4.3, 6.4.3

[13] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020. 7.2

[14] Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. *CoRR*, abs/2002.05283, 2020. 2.4.3, 2.4.3, 3.4.4, 3.4.4, 4.4.3, 4.4.3, 5.4.3, 5.4.3, 6.4.3, 6.4.3

[15] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. *CoRR*, abs/2006.10355, 2020. 2.4.3, 2.3, 2.4.3, 2.4, 3.4.4, 3.3, 3.4.4, 3.4, 4.4.3, 4.3, 4.4.3, 4.4, 5.4.3, 5.3, 5.4.3, 5.4, 6.4.3, 6.3, 6.4.3, 6.4

[16] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019. 2.2, 2.4.2, 2.4.3, 2.4.3, 2.4.3, 3.2, 3.4.2, 3.4.4, 3.4.4, 3.4.4, 4.2.1, 4.4.2, 4.4.3, 4.4.3, 4.4.3, 5.2, 5.4.2, 5.4.3, 5.4.3, 5.4.3, 6.2.1, 6.4.2, 6.4.3, 6.4.3, 6.4.3

[17] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: eliminating unfair advantages in differentiable architecture search. *CoRR*, abs/1911.12126, 2019. 2.4.3, 2.4.3, 3.4.4, 3.4.4, 4.4.3, 4.4.3, 5.4.3, 5.4.3, 6.4.3, 6.4.3

[18] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. DARTS-: robustly stepping out of performance collapse without indicators. *CoRR*, abs/2009.01027, 2020. 2.4.2, 2.4.3, 2.2, 2.4.3, 2.3, 2.4.3, 2.4, 2.4.3, 3.4.4, 3.2, 3.4.4, 3.3, 3.4.4, 3.4, 4.4.3, 4.2, 4.4.3, 4.3, 4.4.3, 4.4, 5.4.2, 5.4.3, 5.2, 5.4.3, 5.3, 5.4.3, 5.4, 6.4.3, 6.2, 6.3, 6.4.3, 6.4

[19] Xiangxiang Chu, Bo Zhang, and Xudong Li. Noisy differentiable architecture search. *CoRR*, abs/2005.03566, 2020. 2.4.3, 2.3, 3.4.4, 3.3, 4.4.3, 4.3, 5.4.3, 5.3, 6.4.3, 6.3

[20] Corbin A Cunningham and Howard E Egeth. Taming the white bear: Initial costs and eventual benefits of distractor inhibition. *Psychological science*, 27(4):476–485, 2016. 7.1

[21] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine learning*, 1(2):145–176, 1986. 4.2.2

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 2.1, 2.4.1, 3.3, 3.4.1, 4.1, 4.4.1, 5.1, 5.4.1, 6.1, 6.4.1, 7.4.3

[23] Amit Deshpande and Luis Rademacher. Efficient volume sampling for row/column subset selection. In *2010 ieee 51st annual symposium on foundations of computer science*, pages 329–338. IEEE, 2010. 7.2

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 7.3

[25] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *CVPR*, 2019. 2.4.3, 2.4.3, 2.4.3, 3.4.4, 3.4.4, 3.4.4, 4.4.3, 4.4.3, 4.4.3, 5.4.3, 5.4.3, 5.4.3, 6.4.3, 6.4.3, 6.4.3

[26] Daniel C Elton. Self-explaining ai as an alternative to interpretable ai. In *International Conference on Artificial General Intelligence*, pages 95–106. Springer, 2020. 4.2.2

[27] Christian Etmann, Sebastian Lunz, Peter Maass, and Carola-Bibiane Schönlieb. On the connection between adversarial robustness and saliency map interpretability. *arXiv preprint arXiv:1905.04172*, 2019. 4.1, 4.3.1

[28] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017. 1.4

[29] Logan Fiorella and Richard E Mayer. The relative benefits of learning by teaching and teaching expectancy. *Contemporary Educational Psychology*, 38(4):281–288, 2013. 6.1

[30] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997. 5.2

[31] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning*, pages 1180–1189, 2015. 2.2

[32] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016. 7.3.1, 7.3.1, 7.3.1, 7.4.2

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2.2

[34] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 2.2, 4.1, 4.3.1

[35] Jindong Gu and Volker Tresp. Search for better students to learn distilled knowledge. In *ECAI*, 2020. 5.2, 6.2.2

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2.4.3, 3.4.4, 4.4.3, 5.4.3, 6.4.3

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2.4.2, 4.4.2, 5.1, 6.4.2, 7.4.3

[38] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019. 1.2.2

[39] Olivier J Hénaff, Ali Razavi, Carl Doersch, SM Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019. 7.2

[40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 5.2, 6.2.2, 6.3.1

[41] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. 6.1

[42] Weijun Hong, Guilin Li, Weinan Zhang, Ruiming Tang, Yunhe Wang, Zhenguo Li, and Yong Yu. Dropnas: Grouped operation dropout for differentiable architecture search. In *IJCAI*, 2020. 2.4.3, 2.4.3, 3.4.4, 3.4.4, 4.4.3, 4.4.3, 5.4.3, 5.4.3, 6.4.3, 6.4.3

[43] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 2.4.3, 3.4.4, 4.4.3, 5.4.3, 6.4.3

[44] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. DSNAS: direct neural architecture search without parameter retraining. In *CVPR*, 2020. 2.4.3, 3.4.4, 4.4.3, 5.4.3, 6.4.3

[45] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 2.4.3, 2.4.3, 3.4.4, 3.4.4, 4.4.3, 4.4.3, 5.1, 5.4.3, 5.4.3, 6.4.3, 6.4.3

[46] Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. Self-paced learning with diversity. *Advances in Neural Information Processing Systems*, 27:2078–2086, 2014. 1.4, 2.2

[47] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. 2.4.2, 3.4.3, 5.4.2, 7.4.3

[48] Aloysius Wei Lun Koh, Sze Chi Lee, and Stephen Wee Hun Lim. The learning benefits of teaching: A retrieval practice hypothesis. *Applied Cognitive Psychology*, 32(3):401–410, 2018. 6.1

[49] M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in neural information processing systems*, pages 1189–1197, 2010. 1.4, 2.2

[50] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 1.4

[51] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016. 4.2.2

[52] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. In *CVPR*, 2020. 5.2, 6.2.2

[53] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. DARTS+: improved differentiable architecture search with early stopping. *CoRR*, abs/1909.06035, 2019. 2.4.3, 2.4.3, 2.4.3, 2.4.3, 3.4.4, 3.4.4, 3.4.4,

4.4.3, 4.2, 4.4.3, 4.4.3, 5.4.3, 5.4.3, 5.4.3, 6.4.3

[54] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019. 2.4.2, 5.2

[55] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 3.3

[56] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018. 2.4.3, 2.4.3, 2.4.3, 3.4.4, 3.4.4, 3.4.4, 4.4.3, 4.4.3, 4.4.3, 5.4.3, 5.4.3, 5.4.3, 6.4.3, 6.4.3, 6.4.3

[57] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018. 2.2, 2.4.3, 3.2, 3.4.4, 4.2.1, 4.4.3, 5.2, 5.4.3, 6.2.1, 6.4.3

[58] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019. 1.2.3, 2.1, 2.2, 2.3.1, 2.3.2, 2.4, 2.4.1, 2.4.2, 2.4.3, 2.4.3, 2.4.3, 3.2, 3.3, 3.4, 3.4.2, 3.4.3, 3.4.4, 3.4.4, 3.4.4, 4.2.1, 4.3.1, 4.3.2, 4.4.2, 4.4.3, 4.4.3, 4.4.3, 5.2, 5.3.1, 5.3.2, 5.3.2, 5.4.2, 5.4.3, 5.4.3, 5.4.3, 6.2.1, 6.3.1, 6.3.2, 6.3.2, 6.4, 6.4.2, 6.4.3, 6.4.3, 6.4.3, 7.3.1

[59] Yu Liu, Xuhui Jia, Mingxing Tan, Raviteja Vemulapalli, Yukun Zhu, Bradley Green, and Xiaogang Wang. Search to distill: Pearls are everywhere but not the eyes. In *CVPR*, 2020. 5.4

[60] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015. 7.3.1, 7.3.1, 7.4.1, 7.4.2, 7.3.1

[61] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. *arXiv preprint arXiv:1705.10667*, 2017. 7.3.1, 7.3.1, 7.3.1, 7.4.2

[62] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018. 2.4.3, 3.4.4, 4.4.3, 5.4.3, 6.4.3

[63] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, 2018. 2.4.3, 3.4.4, 4.4.3, 5.4.3, 6.4.3

[64] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*, 2019. 1.4, 2.2

[65] Steven Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2-3):363–391, 1990. 4.2.2

[66] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and*

Pattern Recognition, pages 6707–6717, 2020. 7.2

[67] James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. Explainable prediction of medical codes from clinical text. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1101–1111, 2018. 4.2.2

[68] Henning Müller, Paul Clough, Thomas Deselaers, and Barbara Caputo. *ImageCLEF: experimental evaluation in visual information retrieval*, volume 32. Springer Science & Business Media, 2010. 7.4.1

[69] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. ASAP: architecture search, anneal and prune. In *AISTATS*, 2020. 2.4.3, 3.4.4, 4.4.3, 5.4.3, 6.4.3

[70] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. 7.2

[71] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009. 7.2

[72] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005. 5.2

[73] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016. 6.1

[74] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016. 7.2

[75] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR, 2017. 1.4

[76] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018. 2.2, 2.4.3, 2.4.3, 3.2, 3.4.4, 3.4.4, 4.2.1, 4.4.3, 4.4.3, 5.2, 5.4.3, 5.4.3, 6.2.1, 6.4.3, 6.4.3

[77] Hieu Pham, Qizhe Xie, Zihang Dai, and Quoc V Le. Meta pseudo labels. *arXiv preprint arXiv:2003.10580*, 2020. 6.2.2

[78] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 2.1, 2.2, 2.4.3, 2.4.3, 2.4.3, 3.2, 3.4.4, 3.4.4, 3.4.4, 4.2.1, 4.4.3, 4.4.3, 4.4.3, 5.2, 5.4.3, 5.4.3, 5.4.3, 6.2.1, 6.4.3, 6.4.3, 6.4.3

[79] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. *arXiv preprint arXiv:1803.09050*, 2018. 7.2, 7.4.2

[80] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM,

2016. 4.2.2

[81] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European conference on computer vision*, pages 213–226. Springer, 2010. 7.4.1

[82] Kuniaki Saito, Donghyun Kim, Stan Sclaroff, Trevor Darrell, and Kate Saenko. Semi-supervised domain adaptation via minimax entropy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8050–8058, 2019. 7.3.1, 7.3.1, 7.3.1, 7.4.2

[83] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in Neural Information Processing Systems*, pages 1919–1930, 2019. 7.2, 7.4.2

[84] Michelle Shu, Chenxi Liu, Weichao Qiu, and Alan Yuille. Identifying model weakness with adversarial examiner. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11998–12006, 2020. 2.2

[85] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013. 4.2.2

[86] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017. 4.2.2

[87] Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. *CoRR*, abs/1912.07768, 2019. 2.2, 2.4.3, 4.2.1, 4.4.3, 5.2, 5.4.3, 5.3, 6.2.2, 6.4.3, 6.4.3

[88] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 2.4.3, 3.4.4, 4.4.3, 5.4.3, 6.4.3

[89] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019. 2.4.3, 3.4.4, 4.4.3, 5.4.3, 6.4.3

[90] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in neural information processing systems*, pages 1195–1204, 2017. 6.1

[91] Ilya Trofimov, Nikita Klyuchnikov, Mikhail Salnikov, Alexander Filippov, and Evgeny Burnaev. Multi-fidelity neural architecture search with knowledge distillation. *CoRR*, abs/2006.08341, 2020. 5.2, 6.2.2

[92] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5018–5027, 2017. 7.4.1

[93] Xiaoxing Wang, Chao Xue, Junchi Yan, Xiaokang Yang, Yonggang Hu, and Kewei Sun. Mergenas: Merge operations into one for differentiable architecture search. In *IJCAI*, 2020. 2.4.3, 3.4.4, 4.4.3, 5.4.3, 6.4.3

[94] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10687–10698, 2020. 5.2, 6.1, 6.2.2

[95] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2019. 2.2, 2.4.3, 2.4.3, 3.2, 3.4.4, 3.4.4, 4.2.1, 4.4.3, 4.4.3, 5.2, 5.4.3, 5.4.3, 6.2.1, 6.4.3, 6.4.3

[96] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: partial channel connections for memory-efficient architecture search. In *ICLR*, 2020. 2.2, 2.4.1, 2.4.2, 2.4.3, 2.4.3, 2.4.3, 3.2, 3.4.2, 3.4.4, 3.4.4, 3.4.4, 4.2.1, 4.4.1, 4.4.2, 4.4.3, 4.4.3, 4.4.3, 5.2, 5.4.2, 5.4.3, 5.4.3, 5.4.3, 6.2.1, 6.4.2, 6.4.3, 6.4.3, 6.4.3

[97] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016. 4.2.2

[98] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, 2017. 2.2

[99] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014. 4.2.2

[100] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020. 2.4.3, 2.4.3, 3.4.4, 3.4.4, 4.4.3, 4.4.3, 5.4.3, 5.4.3, 6.4.3, 6.4.3

[101] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018. 2.4.3, 3.4.4, 4.4.3, 5.4.3

[102] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4320–4328, 2018. 5.2

[103] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *ICML*, 2019. 2.4.3, 2.4.3, 3.4.4, 3.4.4, 4.4.3, 4.4.3, 5.4.3, 5.4.3, 6.4.3, 6.4.3

[104] Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. *International Conference on Learning Representations*, 2017. 4.2.2

[105] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2.1, 2.2, 3.2, 4.2.1, 5.2, 6.2.1

[106] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable

architectures for scalable image recognition. In *CVPR*, 2018. 2.2, 2.4.3, 2.4.3, 3.2, 3.4.4, 3.4.4, 4.2.1, 4.4.3, 4.4.3, 5.2, 5.4.3, 5.4.3, 6.2.1, 6.4.3, 6.4.3