Skillearn: Machine Learning Inspired by Humans' Learning Skills

Pengtao Xie [1], Xuefeng Du [1], and Hao Ban [1]

[1]Affiliation not available

October 30, 2023

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 two case studies, we apply Skillearn to formalize two learning skills of humans: learning by passing tests and interleaving learning, and use the formalized skills to improve neural architecture search. Experiments on various datasets show that trained using the skills formalized by Skillearn, ML models achieve significantly better performance.

Skillearn: Machine Learning Inspired by Humans' Learning Skills

Pengtao Xie p1xie@eng.ucsd.edu
Xuefeng Du xuefengdu1@gmail.com
Hao Ban bhsimon0810@gmail.com
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 two case studies, we apply Skillearn to formalize two learning skills of humans: learning by passing tests and interleaving learning, and use the formalized skills to improve neural architecture search. Experiments on various datasets show that trained using the skills formalized by Skillearn, ML models achieve significantly better performance.

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. 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

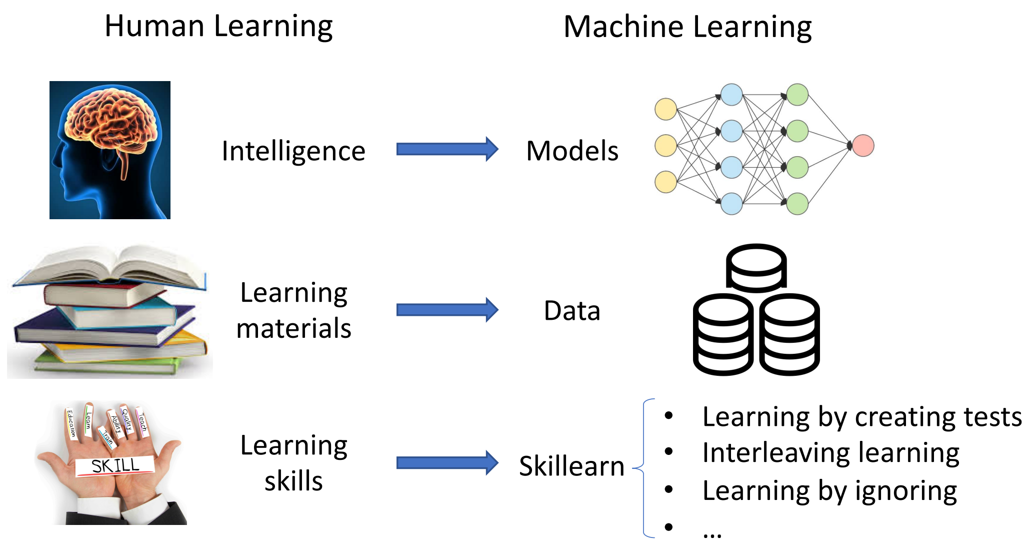

Figure 1: Human learning (HL) versus machine learning (ML). Model capacity in ML is analogous to intelligence in HL. Data in ML is analogous to learning materials in HL. The machines' learning skills formulated by our proposed Skillearn framework are analogous to humans' learning skills.

vector machine 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 this paper, 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. 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. In two case studies, we apply Skillearn to formalize two learning skills of humans – learning by passing tests (LPT) and interleaving learning (IL) – into machines' learning skills (MLS) and leverage these MLS for neural architecture search (Zoph and Le, 2017; Real et al., 2019; Liu et al., 2019). In LPT, a tester model dynamically creates tests with increasing levels of difficulty to evaluate a testee model; the testee continuously improves its architecture by passing however difficult tests created by the tester. In IL, 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. Experiments on various datasets demonstrate that ML models trained by these two learning skills achieve significantly better performance.

The major contributions of this work are as follows.

- We propose to leverage the 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.
- We apply Skillearn to formalize two skills in human learning – learning by passing tests (LPT) and interleaving learning (IL), and apply them to improve neural architecture search.
- On various datasets, we demonstrate the effectiveness of the two skills – LPT and IL formalized by Skillearn – in learning better neural architectures.

The rest of the paper is organized as follows. Section 2 presents the general Skillearn framework. In Section 3 and 4, we present two case studies, where Skillearn is applied to formalize two skills in human learning: learning by passing tests and interleaving learning. Section 5 reviews related works and Section 6 concludes the paper.

2. Skillearn: Machine Learning Inspired by Human's Learning Skills

In this section, we present a general framework called Skillearn, which gets inspiration from humans' learning skills, formalize these skills, and leverage 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.

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-learn the 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.

2.1.1. 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.
- Testing and validation are widely used to evaluate the outcome of learning and provide feedback for improving learning. For example, in learning through testing, the student takes a test to identify the strong and weak points in his/her learning of a topic.
- Learning is performed on various learning materials, including textbooks used for initial learning, homeworks used for enhancing the understanding of knowledge learned from textbooks, tests used for evaluating the outcome of learning, etc.

2.2. General Framework of Skillearn

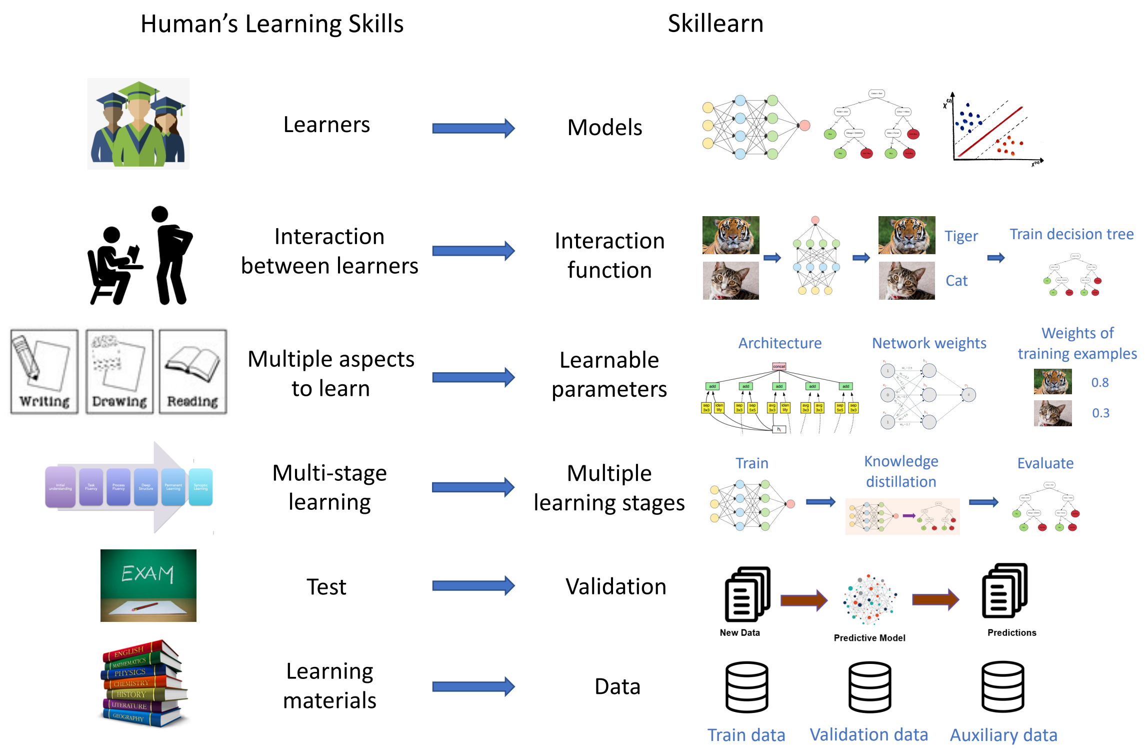

Figure 2: The elements of Skillearn and their counterparts in human learning. The goal of Skillearn is to learn one or a set of ML models which are analogous to learners in human learning. The models can be of any type, such as deep neural network, decisions tree, support vector machine, etc. In a human learning event, a learner has multiple aspects to learn, such as how to read, how to write, how to draw, etc. Analogously, a model in Skillearn has multiple sets of parameters that are learnable, such as architectures, network weights, weights of training examples, etc. In human learning, different learners interact with each other. For example, a teacher can teach a student. An examiner can evaluate an examinee. Likewise, the models in Skillearn can interact with each other. For example, in knowledge distillation, a teacher model (e.g., a deep neural network) can "teach" a student model (e.g., a decision tree) where the teacher predicts pseudo labels on unlabeled data, then these pseudo-labeled data examples are used to train the student model. In a human learning event, there are multiple stages of learning events. For example, in classroom learning, there could be three learning stages: 1) a teacher learns the course materials; 2) the teacher teaches these materials to students; 3) the students take tests to evaluate how well they learn. Analogously, in Skillearn, the learning involves multiple stages. For example, in knowledge distillation, there could be three learning stages: 1) a teacher model is trained; 2) the teacher performs knowledge distillation to "teach" a student model as described above; 3) the performance of the student model is evaluated. In human learning, tests are widely used to evaluate the learners and provide feedback for improving the learners. Analogously, ML models are validated for further improvement. In human learning, the learners learn from learning materials such as textbooks, lecture notes, homework, etc. Likewise, ML models are learned on various datasets, such as training data, validation data, and other auxiliary data.

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.

- **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, 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 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.

- **Validation stage.** This stage evaluates the outcome of learning and provides feedback to improve the learning at the learning stages. This is analogous to the testing and validation in human learning. The validation stage is formulated as an optimization problem as well. The learning outcomes produced in the learning stages are passed to the validation stage.

- **Datasets.** Datasets in ML are analogous to learning materials in human learning. Each learner has a training dataset and a validation dataset. The training dataset is used in the learning stages and the validation dataset is used in the validation stage. Besides, there are auxiliary datasets (labeled or unlabeled) on which the learners interact with each other.

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 in 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.
- **Active training datasets**, which include the training dataset of every active learner.
- **Active auxiliary datasets**, which include the auxiliary datasets where the interaction function in this learning stage is defined on.
- **Training loss**, which is defined on the active training data collection, active learnable parameters, and supporting learnable parameters.
- **Interaction function**, which depicts the interaction between two or more active learners. It is defined on the active auxiliary datasets, active learnable parameters, and supporting learnable parameters.

In Skillearn, there is a single validation stage where an optimization problem is defined. The optimization problem involves one or more validation losses and (optionally) an interaction function which describes how the learners in the validation stage interact with each other. The validation stage consists of the following elements.

- **Active learners**, which are the learners to validate.
- **Remaining learnable parameters**. At each learning stage, a subset of parameters are learned. After all learning stages, the parameters that have not been learned are called remaining parameters. The remaining parameters are updated in the validation stage.
- **Validation datasets**: validation datasets of all active learners.
- **Active auxiliary datasets**, which include the auxiliary datasets where the interaction function in the validation stage is defined on.
- **Validation losses**, which are defined on remaining learnable parameters, validation datasets, and (optionally) active auxiliary datasets.
- **Interaction function**, which depicts the interaction between two or more active learners. It is defined on remaining learnable parameters and active auxiliary datasets.

Notation	Meaning
M	Number of learners
$D_m^{(\text{tr})}$	Training data of the m-th learner
$D_m^{(\text{val})}$	Validation data of the m-th learner
$\mathcal{F}$	Auxiliary datasets accessible to all learners
N_m	Number of learnable parameter sets belonging to the m-th learner
$W_i^{(m)}$	The i-th learnable parameter set of the m-th learner
K	Number of learning stages
M_k	Number of active learners in the k-th learning stage
$\mathcal{A}_k$	The set of active learners in the k-th learning stage
$a_i^{(k)}$	The i-th active learner in the k-th learning stage
O_{ki}	The number of active parameter sets of the i-th active learner in the k-th learning stage
W_{kij}	The j-th active parameter set of the i-th learner in the k-th learning stage
$\mathcal{W}_{ki}$	The collection of active parameter sets of the i-th active learner in the k-th learning stage
$\mathcal{W}_k$	All active parameter sets in the k-th learning stage
P_{ki}	The number of supporting parameter sets of the i-th active learner in the k-th learning stage
U_{kij}	The j-th supporting parameter set of the i-th active learner in the k-th learning stage
$\mathcal{U}_{ki}$	The collection of supporting parameter sets of the i-th active learner in the k-th learning stage
$\mathcal{U}_k$	All supporting parameter sets in the k-th learning stage
$D_{ki}^{(\text{tr})}$	Training dataset of the i-th active learner in the k-th learning stage
$\mathcal{D}_k^{(\text{tr})}$	Active training datasets in the k-th learning stage
$\mathcal{F}_k$	Active auxiliary datasets in the k-th learning stage
L_k	Training loss in the k-th learning stage
I_k	Interaction function in the k-th learning stage

Table 1: Notations in the Skillearn framework

2.2.1. Mathematical Setup

We assume there are M learners in total. For each learner m, it has a training set $D_m^{(\text{tr})}$ and optionally a validation set $D_m^{(\text{val})}$. Meanwhile, all learners share a common collection of auxiliary datasets $\mathcal{F}$, which could be unlabeled datasets used for self-supervised pretraining (He et al., 2019), additional labeled datasets used for validation, and so on. The learner m has one or more sets of learnable parameters $\{W_i^{(m)}\}_{i=1}^{N_m}$. The learnable parameters could be network weights, architectures, hyperparameters, weights of training examples, etc.

We assume there are K learning stages. At each stage k, a subset of M_k learners $\mathcal{A}_k = \{a_i^{(k)}\}_{i=1}^{M_k}$ are involved in the learning, which are called active learners. For each active learner $a_i^{(k)}$, a sub-collection of its learnable parameter sets $\mathcal{W}_{ki} = \{W_{kij}\}_{j=1}^{O_{ki}}$ are trained at this stage, which are called active learnable parameters. Let $\mathcal{W}_k = \{\mathcal{W}_{ki} | i = 1, \cdots, M_k\}$ denote the active learnable parameters for all active learners. Meanwhile, another sub-collection of its learnable parameter sets $\mathcal{U}_{ki} = \{U_{kij}\}_{j=1}^{P_{ki}}$ are used to define the training loss function and interaction function. But $\mathcal{U}_{ki}$ are not updated at this stage. They are called supporting learnable parameters. Let $\mathcal{U}_k = \{\mathcal{U}_{ki} | i = 1, \cdots, M_k\}$ denote the supporting learnable parameters for all active learners. Let $\mathcal{D}_k$ denote the active training datasets, consisting of the training dataset of each active learner in $\mathcal{A}_k$. Let $\mathcal{F}_k$ denote the active auxiliary datasets used in 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 1) a training loss L_k defined on the active training datasets, active learnable parameters, and supporting learnable parameters; 2) (optionally) an interaction function I_k that depicts the interaction between learners in $\mathcal{A}_k$. The notations are summarized in Table 1.

2.2.2. The Mathematical Framework for Skillearn

The formulation of Skillearn is shown in Eq.(1).

$$\begin{array}{ll}
\max_{\{\mathcal{U}_i\}_{i=1}^K} & L_{val}(\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^K, \mathcal{D}^{(\text{val})}, \mathcal{F}) + \gamma_{val} I_{val}(\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^K, \mathcal{F}) \quad \text{(Validation stage)} \\
s.t. & \text{Learning stage } K\text{:} \\
& \mathcal{W}_K^*(\{\mathcal{U}_j\}_{j=1}^K) = \min\limits_{\mathcal{W}_K} L_K(\mathcal{W}_K, \mathcal{U}_K, \{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{K-1}, \mathcal{D}_K^{(\text{tr})}, \mathcal{F}_K) + \\
& \qquad\qquad \gamma_K I_K(\mathcal{W}_K, \mathcal{U}_K, \{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{K-1}, \mathcal{F}_K) \\
& \vdots \\
& \text{Learning stage } k\text{:} \\
& \mathcal{W}_k^*(\{\mathcal{U}_j\}_{j=1}^k) = \min\limits_{\mathcal{W}_k} L_k(\mathcal{W}_k, \mathcal{U}_k, \{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1}, \mathcal{D}_k^{(\text{tr})}, \mathcal{F}_k) + \\
& \qquad\qquad \gamma_k I_k(\mathcal{W}_k, \mathcal{U}_k, \{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1}, \mathcal{F}_k) \\
& \vdots \\
& \text{Learning stage 1:} \\
& \mathcal{W}_1^*(\mathcal{U}_1) = \min\limits_{\mathcal{W}_1} L_1(\mathcal{W}_1, \mathcal{U}_1, \mathcal{D}_1^{(\text{tr})}, \mathcal{F}_1) + \gamma_1 I_1(\mathcal{W}_1, \mathcal{U}_1, \mathcal{F}_1)
\end{array} \tag{1}$$

It is a multi-level optimization framework, which involves $K + 1$ optimization problems. On the constraints are K 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, the optimization variables are the active learnable parameters $\mathcal{W}_k$ of all active learners in this stage. The objective function consists of a training loss $L_k(\mathcal{W}_k, \mathcal{U}_k, \{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1}, \mathcal{D}_k^{(\text{tr})}, \mathcal{F}_k)$ defined on the active learnable parameters $\mathcal{W}_k$, supporting learnable parameters $\mathcal{U}_k$, optimal solutions $\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1}$ obtained in previous learning stages, active training datasets $\mathcal{D}_k^{(\text{tr})}$, and active auxiliary datasets $\mathcal{F}_k$. Typically, $L_k(\mathcal{W}_k, \mathcal{U}_k, \{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1}, \mathcal{D}_k^{(\text{tr})}, \mathcal{F}_k)$ can be decomposed into a summation of active learners' individual training losses:

$$L_k(\mathcal{W}_k,\mathcal{U}_k,\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1},\mathcal{D}_k^{(\mathrm{tr})},\mathcal{F}_k)=\sum_{i=1}^{M_k}L_{ki}(\mathcal{W}_{ki},\mathcal{U}_{ki},\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1},D_{ki}^{(\mathrm{tr})},\mathcal{F}_k) \tag{2}$$

where $L_{ki}(\mathcal{W}_{ki},\mathcal{U}_{ki},\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1},D_{ki}^{(\mathrm{tr})},\mathcal{F}_k)$ is the training loss of the active learner i defined on its active parameters $\mathcal{W}_{ki}$, supporting parameters $\mathcal{U}_{ki}$, and training dataset $D_{ki}^{(\mathrm{tr})}$. The M_k learners do not interact in the training loss. The other part of the objective function is the interaction function $I_k(\mathcal{W}_k,\mathcal{U}_k,\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1},\mathcal{F}_k)$ which depicts how the M_k active learners interact with each other in this learning stage. It is defined on the active learnable parameters $\mathcal{W}_k$, supporting learnable parameters $\mathcal{U}_k$, optimal solutions $\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1}$ in previous stages, and active auxiliary datasets $\mathcal{F}_k$. γ_k is a tradeoff parameter between the training loss and interaction function. $\mathcal{U}_k$ 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 $\mathcal{W}_k^*(\{\mathcal{U}_j\}_{j=1}^k)$. Note that $\mathcal{W}_k^*$ is function of $\{\mathcal{U}_j\}_{j=1}^k$ since $\mathcal{W}_k^*$ is a function of the objective and the objective is a function of $\{\mathcal{U}_j\}_{j=1}^k$. $\mathcal{W}_k^*(\{\mathcal{U}_k\}_{j=1}^k)$ is used to define the objectives in later stages.

At the very top of Eq.(1), the optimization problem (outside the constraint block) corresponds to the validation stage which validates the optimal solutions $\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^K$ obtained in the K learning stages. The optimization variables are remaining learnable parameters $\{\mathcal{U}_i\}_{i=1}^K$ that have not been learned in the K learning stages. The objective function consists of a validation loss and an interaction function. γ_{val} is a tradeoff parameter. The validation loss $L_{val}(\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^K,\mathcal{D}^{(\mathrm{val})},\mathcal{F})$ is defined on the validation sets of all learners $\mathcal{D}^{(\mathrm{val})}=\{\mathcal{D}_i^{(\mathrm{val})}\}_{i=1}^M$, the optimal solutions $\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^K$, and the auxiliary datasets $\mathcal{F}$. The interaction function is defined on $\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^K$ and $\mathcal{F}$.

Remarks:

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

2.3. Optimization Algorithm for Skillearn

In this section, we develop an algorithm to solve the Skillearn problem in Eq.(1), inspired by the algorithm in (Liu et al., 2019). For each learning stage k with an optimization problem: $\mathcal{W}_k^*(\{\mathcal{U}_j\}_{j=1}^k)=\min_{\mathcal{W}_k}L_k(\mathcal{W}_k,\mathcal{U}_k,\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1},\mathcal{D}_k^{(\mathrm{tr})},\mathcal{F}_k)+\gamma_k I_k(\mathcal{W}_k,\mathcal{U}_k,\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1},\mathcal{F}_k)$, we approximate the optimal solution $\mathcal{W}_k^*(\{\mathcal{U}_j\}_{j=1}^k)$ by one-step gradient descent update of the variable $\mathcal{W}_k$:

$$\begin{aligned}&\mathcal{W}_k^*(\{\mathcal{U}_j\}_{j=1}^k) \approx \mathcal{W}_k'(\{\mathcal{U}_j\}_{j=1}^k) = \\ &\mathcal{W}_k - \eta\nabla_{\mathcal{W}_k}(L_k(\mathcal{W}_k,\mathcal{U}_k,\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1},\mathcal{D}_k^{(\mathrm{tr})},\mathcal{F}_k) + \gamma_k I_k(\mathcal{W}_k,\mathcal{U}_k,\{\mathcal{W}_j^*(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{k-1},\mathcal{F}_k)).\end{aligned} \tag{3}$$

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

$$\begin{aligned}&\mathcal{W}_l^*(\{\mathcal{U}_j\}_{j=1}^l) \approx \mathcal{W}_l'(\{\mathcal{U}_j\}_{j=1}^l) = \\ &\mathcal{W}_l - \eta\nabla_{\mathcal{W}_l}(L_l(\mathcal{W}_l,\mathcal{U}_l,\{\mathcal{W}_j'(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{l-1},\mathcal{D}_l^{(\mathrm{tr})},\mathcal{F}_l) + \gamma_l I_l(\mathcal{W}_l,\mathcal{U}_l,\{\mathcal{W}_j'(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^{l-1},\mathcal{F}_l)).\end{aligned} \tag{4}$$

For the objective in the validation stage, it can be approximated as:

$$L_{val}(\{\mathcal{W}_j'(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^K,\mathcal{D}^{(\mathrm{val})},\mathcal{F}) + \gamma_{val} I_{val}(\{\mathcal{W}_j'(\{\mathcal{U}_i\}_{i=1}^j)\}_{j=1}^K,\mathcal{F}). \tag{5}$$

We update the remaining learnable parameters $\{\mathcal{U}_i\}_{i=1}^K$ 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 $k = 1 \cdots K$, update $\mathcal{W}_k^*(\{\mathcal{U}_j\}_{j=1}^k)$ using Eq.(4)
    2. Update $\{\mathcal{U}_i\}_{i=1}^K$ by minimizing the approximated objective in Eq.(5)
end
```

3. Case Study I: Learning by Passing Tests

In this section, we apply our general Skillearn framework to formalize a human learning technique – learning by passing tests, and apply it to improve machine learning. 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 use the Skillearn framework to formalize this human learning technique, which results in a novel machine learning framework called learning by passing tests (LPT). In this framework, there are two learners: a "testee" model and a "tester" model. The tester creates a sequence of "tests" with growing levels of difficulty. The testee 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 testee 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 testee can make correct predictions on T, it means that T is not difficult enough. The tester will create a more challenging test T' by selecting a new set of examples from the test bank in a way that the new error rate R' achieved by M is larger than R. Given this more demanding test T', the testee 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 iterates until convergence.

In our framework, both the testee and tester perform learning. The testee 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 testee 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. The learning is organized into three stages. In the first stage, the testee 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 testee updates its model architecture by minimizing the predictive loss L on the test 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 testee and tester interact on the loss function L in an adversarial manner, where the testee minimizes this loss while the tester maximizes this loss. The three stages are performed jointly end-to-end in a multi-level optimization framework, where a latter stage influences an earlier stage and vice versa. We apply our method for neural architecture search (Zoph and Le, 2017; Liu et al., 2019; Real et al., 2019) in image classification tasks on CIFAR-100, CIFAR-10, and ImageNet (Deng et al., 2009). Our method achieves significant improvement over state-of-the-art baselines.

3.1. Method

In this section, we describe how to instantiate the general Skillearn framework to the LPT framework, and how to instantiate the general optimization procedure of Skillearn to a specialized optimization algorithm for LPT.

3.1.1. Learning by Passing Tests

In the learning by passing tests (LPT) framework, there are two learners: a testee model and a tester model, where the testee studies how to perform a target task J_1 such as classification, regression, etc. The eventual goal is to make the testee achieve a better learning outcome with the help of the tester. There is a collection of data examples called "test bank". The tester creates a test by selecting a subset of examples from the test bank. Given a test T, the testee 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 testee 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 testee's perspective, R' indicates how well the testee performs on the test. Given this more difficult test T', the testee refines its model to pass this new test. It aims to learn a new model M' in a way that the error

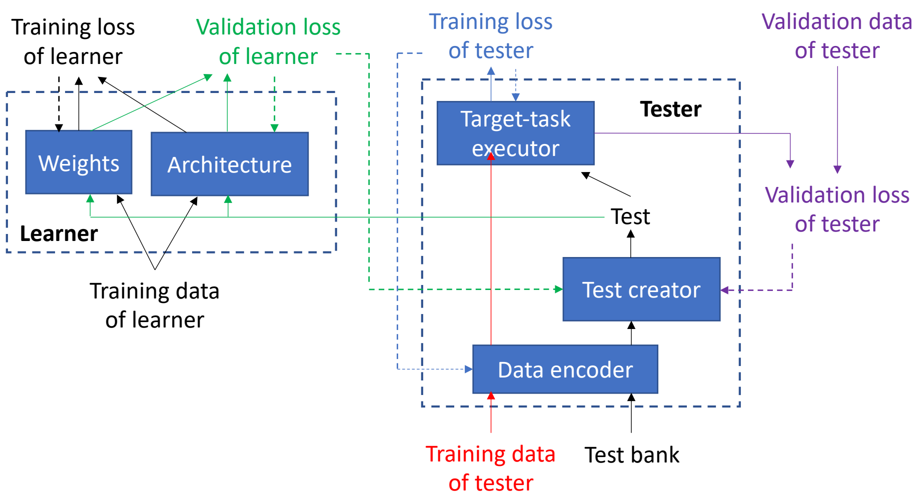

Figure 3: Illustration of learning by passing tests. The solid arrows denote the process of making predictions and calculating losses. The dotted arrows denote the process of updating learnable parameters by minimizing corresponding losses.

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

Table 2: Notations in Learning by Passing Tests

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 testee may render the testee 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 (e.g., classification). 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.

Active learners	Testee
Active learnable parameters	Network weights of the testee
Supporting learnable parameters	Architecture of the testee
Active training datasets	Training dataset of target-task J_1 performed by the testee
Active auxiliary datasets	–
Training loss	Training loss of target-task J_1: $L(A, W, D_{ee}^{(\mathrm{tr})})$
Interaction function	–
Optimization problem	$W^*(A) = \min_W L(A, W, D_{ee}^{(\mathrm{tr})})$

Table 3: Learning Stage I in LPT

Active learners	Examiner
Active learnable parameters	1) Data encoder of the tester; 2) Target-task executor of the tester.
Supporting learnable parameters	Test creator of the tester
Active training datasets	Training data of target-task J_2 performed by the tester
Active auxiliary datasets	Test bank
Training loss	$L(E, X, D_{er}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))$
Interaction function	–
Optimization problem	$E^*(C), X^*(C) = \min_{E,X} \quad L(E, X, D_{er}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))$.

Table 4: Learning Stage II in LPT

In our framework, both the testee and the tester performs learning. The testee studies how to best fulfill the target task J_1. The tester studies how to create tests that are difficult and meaningful. In the testee' 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 target-task J_1. The model has three modules with learnable parameters: data encoder, test creator, and target-task executor, where the test creator performs the task of generating tests and the target-task executor conducts J_1. 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 testee and the tester is organized into three stages. In the first stage, the testee learns its network weights W by minimizing the training loss $L(A, W, D_{ee}^{(\mathrm{tr})})$ defined on the training data $D_{ee}^{(\mathrm{tr})}$ in the task J_1. The architecture A is used to define the training loss, but it is not learned in 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 in 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. Table 3 shows the key elements of this learning stage under the Skillearn terminology. The testee is the active learner, which performs learning in this stage. Network weights of the testee are the active learnable parameters, which are updated at this stage. The architecture variables of the testee are the supporting learnable parameters, which are used to define the loss function, but are not updated at this stage. Active training datasets include the training data of the task J_1 performed by the testee. There are no active auxiliary datasets. Training loss is $L(A, W, D_{ee}^{(\mathrm{tr})})$. There is no interaction function at this stage. The optimization problem is:

$$W^*(A) = \min_{W} L\left(A, W, D_{ee}^{(\mathrm{tr})}\right). \tag{6}$$

In the second stage, the tester learns its data encoder E and target-task executor X by minimizing the training loss $L(E, X, D_{er}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))$ in the task J_2. The training loss consists of two parts. The first part $L(E, X, D_{er}^{(\mathrm{tr})})$ is defined on the training dataset $D_{er}^{(\mathrm{tr})}$ 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. For each example d in the test bank D_b, it is first fed into the encoder E, then the 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 in 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 in 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. Table 4 shows the key elements of this learning stage under the Skillearn terminology. The tester is the active learner. The active learnable parameters include the data encoder and target-task executor of the tester. The supporting learnable parameters include the test creator. The active training datasets include the training data of target-task J_2 performed by the tester. The active auxiliary datasets include the test bank. The training loss is $L(E, X, D_{er}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))$. There is no interaction function at this stage. The optimization problem is:

$$E^*(C), X^*(C) = \min_{E,X} \; L\left(E, X, D_{er}^{(\mathrm{tr})}\right) + \gamma L\left(E, X, \sigma\left(C, E, D_b\right)\right). \tag{7}$$

In the third stage, the testee learns its architecture by trying to pass the test $\sigma(C, E^*(C), D_b)$ created by the tester. Specifically, the testee aims to minimizes the predictive loss of its model on the test:

$$L(A, W^*(A), \sigma(C, E^*(C), D_b)) = \sum_{d \in \sigma(C, E^*(C), D_b)} \ell(A, W^*(A), d) \tag{8}$$

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 testee performs well on this test. Meanwhile, the tester learns its test creator C in a way that C can create

Active learners	Testee, tester
Remaining learnable parameters	1) Architecture of the testee; 2) Test creator of the tester
Validation datasets	Validation dataset of the tester
Active auxiliary datasets	Test bank
Validation loss	$L(E^*(C), X^*(C), D_{er}^{(\text{val})})$
Interaction function	Testee's prediction loss defined on the test created by the tester: $L(A, W^*(A), \sigma(C, E^*(C), D_b))/\lvert\sigma(C, E^*(C), D_b)\rvert$
Optimization problem	$\max_C \min_A \quad L(A, W^*(A), \sigma(C, E^*(C), D_b))/\lvert\sigma(C, E^*(C), D_b)\rvert - \lambda L(E^*(C), X^*(C), D_{er}^{(\text{val})})$

Table 5: Validation Stage in LPT

a test with more difficulty and meaningfulness. Difficulty is measured by the testee's predictive loss $L(A, W^*(A), \sigma(C, E^*(C), D_b))$ on the test. Given a model $(A, W^*(A))$ of the testee 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:

$$\frac{1}{\lvert\sigma\left(C, E^*(C), D_b\right)\rvert} L\left(A, W^*\left(A\right), \sigma\left(C, E^*(C), D_b\right)\right) \tag{9}$$

where $\lvert\sigma(C, E^*(C), D_b)\rvert$ is the cardinality of the set $\sigma(C, E^*(C), D_b)$. Under the Skillearn terminologies, the loss in Eq.(9) is the interaction function where the testee and tester interact. The testee aims to minimize this loss to "pass" the testee and the tester aims to maximize this loss to "fail" the testee. To measure the meaningfulness of a test, we check how well the optimally-trained task executor $E^*(C)$ and data encoder $X^*(C)$ of the tester perform on the validation data $D_{er}^{(\text{val})}$ in the target task J_2, and the performance is measured by the validation loss: $L(E^*(C), X^*(C), D_{er}^{(\text{val})})$. $E^*(C)$ 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 $L(E^*(C), X^*(C), D_{er}^{(\text{val})})$. In sum, C is learned by maximizing $L(A, W^*(A), \sigma(C, E^*(C), D_b))/\lvert\sigma(C, E^*(C), D_b)\rvert - \lambda L(E^*(C), X^*(C), D_{er}^{(\text{val})})$, where λ is a tradeoff parameter between these two objectives. Under the Skillearn terminology, this stage is a validation stage. Table 5 summarizes the key elements of this stage. The active learners include both the testee and the tester. The remaining learnable parameters include the architecture of the testee and the test creator of the tester. The validation datasets include the validation data in the target-task J_2 performed by the tester. The active auxiliary

Skillearn	Learning by Passing Tests
Learners	1) Testee; 2) Tester
Learnable parameters	1) Architecture of testee; 2) Network weights of testee; 3) Data encoder of tester; 4) Target-task executor of tester; 5) Test creator of tester.
Interaction function	Testee's prediction loss defined on the test created by the tester: $L(A, W^*(A), \sigma(C, E^*(C), D_b))/\lvert\sigma(C, E^*(C), D_b)\rvert$
Learning stages	Learning stage I: the testee learns its network weights on its training data: $W^*(A) = \min_W L(A, W, D_{ee}^{(\text{tr})})$ Learning stage II: the tester uses its test creator to select a subset of examples from the test bank, then it learns its data encoder and target-task executor on its training data and on the selected examples from the test bank: $E^*(C), X^*(C) = \min_{E,X} \quad L(E, X, D_{er}^{(\text{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))$.
Validation stage	1) The testee updates its architecture to minimize the prediction loss on the test created by the tester; 2) The tester updates its test creator to maximize the testee's prediction loss and minimize its own validation loss. $\max_C \min_A \quad \frac{1}{\lvert\sigma(C,E^*(C),D_b)\rvert} L(A, W^*(A), \sigma(C, E^*(C), D_b)) - \lambda L(E^*(C), X^*(C), D_{er}^{(\text{val})})$.
Datasets	1) Training data of the testee; 2) Training data of the tester; 3) Validation data of the tester; 4) Test bank.

Table 6: Instantiation of Skillearn to LPT

datasets include the test bank. The validation loss is $L(E^*(C), X^*(C), D_{er}^{(\text{val})})$. The interaction function is $\frac{1}{\lvert\sigma(C,E^*(C),D_b)\rvert} L(A, W^*(A), \sigma(C, E^*(C), D_b))$. The optimization problem is:

$$\max_C \min_A \quad \frac{1}{\lvert\sigma(C, E^*(C), D_b)\rvert} L\left(A, W^*(A), \sigma(C, E^*(C), D_b)\right) - \lambda L\left(E^*(C), X^*(C), D_{er}^{(\text{val})}\right). \tag{10}$$

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 instantiate the Skillearn framework into the following LPT formulation:

$$\begin{aligned}
&\max_C \min_A \quad \tfrac{1}{\lvert\sigma(C,E^*(C),D_b)\rvert} L\left(A, W^*(A), \sigma(C, E^*(C), D_b)\right) - \lambda L\left(E^*(C), X^*(C), D_{er}^{(\text{val})}\right) \text{(III)}\\
&s.t. \quad E^*(C), X^*(C) = \min_{E,X} \quad L\left(E, X, D_{er}^{(\text{tr})}\right) + \gamma L\left(E, X, \sigma(C, E, D_b)\right) \text{(Stage II)}\\
&\qquad W^*(A) = \min_W \quad L\left(A, W, D_{ee}^{(\text{tr})}\right) \text{(Stage I)}
\end{aligned} \tag{11}$$

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 respectively. The objective function of the outer optimization problem corresponds to the validation stage. Table 6 summarizes the instantiation of Skillearn to LPT.

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 $\sigma(C, E, D_b)$:

$$\sigma(C, E, D_b) = \{(d, f(d, C, E))|d \in D_b\} \tag{12}$$

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:

$$L(E, X, \sigma(C, E, D_b)) = \sum_{d \in D_b} f(d, C, E)\ell(E, X, d) \tag{13}$$

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

$$|\sigma(C, E^*(C), D_b)| = \sum_{d \in D_b} f(d, C, E^*(C)) \tag{14}$$

Similar to (Liu et al., 2019), we represent the architecture A of the testee 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 $A = \{a\}$.

3.1.2. Optimization Algorithm

In this section, we instantiate the general optimization framework in Section 2.3 to derive an optimization algorithm for LPT. We approximate $E^*(C)$ and $X^*(C)$ using one-step gradient descent update of E and X with respect to $L(E, X, D_{er}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))$ and approximate $W^*(A)$ using one-step gradient descent update of W with respect to $L(A, W, D_{ee}^{(\mathrm{tr})})$. Then we plug in these approximations into

$$L(A, W^*(A), \sigma(C, E^*(C), D_b))/|\sigma(C, E^*(C), D_b)| - \lambda L(E^*(C), X^*(C), D_{er}^{(\mathrm{val})}), \tag{15}$$

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

$$\begin{aligned}
&\nabla_A L\left(A, W^*(A), \sigma\right) \approx \\
&\nabla_A L\left(A, W - \xi_{ee}\nabla_W L\left(A, W, D_{ee}^{(\mathrm{tr})}\right), \sigma\right) = \\
&\nabla_A L\left(A, W', \sigma\right) - \xi_{ee}\nabla^2_{A,W} L\left(A, W, D_{ee}^{(\mathrm{tr})}\right) \nabla_{W'} L\left(A, W', \sigma\right),
\end{aligned} \tag{16}$$

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

$$\nabla^2_{A,W} L\left(A, W, D_{ee}^{(\mathrm{tr})}\right) \nabla_{W'} L\left(A, W', \sigma\right) \approx \tfrac{1}{2\alpha_{ee}} \left(\nabla_A L\left(A, W^+, D_{ee}^{(\mathrm{tr})}\right) - \nabla_A L\left(A, W^-, D_{ee}^{(\mathrm{tr})}\right)\right), \tag{17}$$

where $W^{\pm} = W \pm \alpha_{ee} \nabla_{W'} L\left(A, W', \sigma\right)$ and α_{ee} 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:

$$\begin{array}{l} E' = E - \xi_E \nabla_E [L(E, X, D_{er}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))] \\ X' = X - \xi_X \nabla_X [L(E, X, D_{er}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))] \end{array} \tag{18}$$

where ξ_E and ξ_X are learning rates. Plugging in these approximations into the objective function in Eq.(15), we can learn C by maximizing the following objective using gradient methods:

$$L(A, W', \sigma(C, E', D_b))/|\sigma(C, E', D_b)| - \lambda L(E', X', D_{er}^{(\mathrm{val})}) \tag{19}$$

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

$$\nabla_C L(E', X', D_{er}^{(\mathrm{val})}) = \tfrac{\partial E'}{\partial C} \nabla_{E'} L(E', X', D_{er}^{(\mathrm{val})}) + \tfrac{\partial X'}{\partial C} \nabla_{X'} L(E', X', D_{er}^{(\mathrm{val})}) \tag{20}$$

where

$$\begin{array}{l} \tfrac{\partial E'}{\partial C} = -\xi_E \gamma \nabla^2_{C,E} L(E, X, \sigma(C, E, D_b)) \\ \tfrac{\partial X'}{\partial C} = -\xi_X \gamma \nabla^2_{C,X} L(E, X, \sigma(C, E, D_b)) \end{array} \tag{21}$$

Similar to Eq.(17), using finite difference approximation to calculate $\nabla^2_{C,E} L(E, X, \sigma(C, E, D_b))$ $\nabla_{E'} L(E', X', D_{er}^{(\mathrm{val})})$ and $\nabla^2_{C,X} L(E, X, \sigma(C, E, D_b)) \nabla_{X'} L(E', X', D_{er}^{(\mathrm{val})})$, we have:

$$\begin{array}{l} \nabla_C L(E', X', D_{er}^{(\mathrm{val})}) = \\ -\gamma \xi_E \frac{\nabla_C L(E^+, X, \sigma(C, E^+, D_b)) - \nabla_C L(E^-, X, \sigma(C, E^-, D_b))}{2\alpha_E} - \gamma \xi_X \frac{\nabla_C L(E, X^+, \sigma(C, E, D_b)) - \nabla_C L(E, X^-, \sigma(C, E, D_b))}{2\alpha_X} \end{array} \tag{22}$$

where $E^{\pm} = E \pm \alpha_E \nabla_{E'} L(E', X', D_{er}^{(\mathrm{val})})$ and $X^{\pm} = X \pm \alpha_X \nabla_{X'} L(E', X', D_{er}^{(\mathrm{val})})$. 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:

$$\nabla_C L(A, W', \sigma(C, E', D_b)) = \tfrac{\partial E'}{\partial C} \nabla_{E'} L(A, W', \sigma(C, E', D_b)), \tag{23}$$

where $\frac{\partial E'}{\partial C}$ is given in Eq.(21) and $\nabla^2_{C,E} L(E, X, \sigma(C, E, D_b)) \times \nabla_{E'} L(A, W', \sigma(C, E', D_b))$ 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

$$\sum_{d \in D_b} \nabla_C f(d, C, E') + \frac{\partial E'}{\partial C} \nabla_{E'} f(d, C, E') \tag{24}$$

where $\frac{\partial E'}{\partial C}$ is given in Eq.(21). The algorithm for solving LPT is summarized in Algorithm 2.

Algorithm 2 Optimization algorithm for learning by passing tests

```
while not converged do
    1. Update the architecture of the testee by descending the gradient calculated in Eq.(16)
    2. Update the test creator of the tester by ascending the gradient calculated in Eq.(20-24)
    3. Update the data encoder and target-task executor of the tester using Eq.(18)
    4. Update the weights of the testee by descending $\nabla_W L(A, W, D_{ee}^{(\mathrm{tr})})$
end
```

3.2. Experiments

We apply LPT for neural architecture search in image classification tasks. Following (Liu et al., 2019), we first perform architecture search which finds out 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 the test set. We let the target task of the learner and that of the tester be the same.

3.2.1. Datasets

We used three datasets in the experiments: CIFAR-10, CIFAR-100, and ImageNet (Deng et al., 2009). The CIFAR-10 dataset contains 50K training images and 10K testing images, from 10 classes (the number of images in each class is equal). Following (Liu et al., 2019), we split the original 50K training set into a new 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 the training data $D_{ee}^{(\mathrm{tr})}$ of the learner and the training data $D_{er}^{(\mathrm{tr})}$ of the tester. The validation set is used as the test bank D_b and the validation data $D_{er}^{(\mathrm{val})}$ of the tester. During architecture evaluation, the combination of the training data and validation data is used to train the 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-100, the 50K training images are split into a 25K training set and 25K validation set. The usage of the new training set and validation set is the same as that for CIFAR-10. The ImageNet dataset contains a training set of 1.2M images and a validation set of 50K images, from 1000 object classes. The validation set is used as a test set for architecture evaluation. Following (Liu et al., 2019), we 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 1.2M training data of ImageNet and evaluated on the 50K test data.

3.2.2. 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 (Liu et al., 2019), 2) P-DARTS (Chen et al., 2019), 3) DARTS$^+$ (Liang et al., 2019b), 4)

DARTS$^-$ (Chu et al., 2020a). 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 (He et al., 2016b). For the test creator and target-task executor, they are set to one feed-forward layer. λ and γ are both set to 1.

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 (Kingma and Ba, 2014) optimizer 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 (Liu et al., 2019), 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.

3.2.3. Results

Table 7 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$^-$, 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 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

Method	Error(%)	Param(M)	Cost
*ResNet (He et al., 2016a)	22.10	1.7	-
*DenseNet (Huang et al., 2017)	17.18	25.6	-
*PNAS (Liu et al., 2018a)	19.53	3.2	150
*ENAS (Pham et al., 2018)	19.43	4.6	0.5
*AmoebaNet (Real et al., 2019)	18.93	3.1	3150
*GDAS (Dong and Yang, 2019)	18.38	3.4	0.2
*R-DARTS (Zela et al., 2020)	18.01±0.26	-	1.6
*DropNAS (Hong et al., 2020)	16.39	4.4	0.7
†DARTS-1st (Liu et al., 2019)	20.52±0.31	1.8	0.4
LPT-R18-DARTS-1st (ours)	19.11±0.11	2.1	0.6
*DARTS-2nd (Liu et al., 2019)	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$^-$ (Chu et al., 2020a)	17.51±0.25	3.3	0.4
†DARTS$^-$ (Chu et al., 2020a)	18.97±0.16	3.1	0.4
LPT-R18-DARTS$^-$ (ours)	18.28±0.14	3.4	0.6
$^\Delta$DARTS$^+$ (Liang et al., 2019a)	17.11±0.43	3.8	0.2
LPT-R18-DARTS$^+$ (ours)	16.58±0.19	3.7	0.3
*P-DARTS (Chen et al., 2019)	17.49	3.6	0.3
LPT-R18-P-DARTS (ours)	**16.28±0.10**	3.8	0.5

Table 7: 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$^-$ (Chu et al., 2020a). † 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.

with challenging cases in the unseen data. **Second**, LPT-R50-DARTS-2nd outperforms LPT-R18-DARTS-2nd, 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 more effectively evaluate the learner and promote its learning capability. **Third**, our method LPT-R18-P-DARTS

Method	Error(%)	Param(M)	Cost
*DenseNet (Huang et al., 2017)	3.46	25.6	-
*HierEvol (Liu et al., 2018b)	3.75±0.12	15.7	300
*NAONet-WS (Luo et al., 2018)	3.53	3.1	0.4
*PNAS (Liu et al., 2018a)	3.41±0.09	3.2	225
*ENAS (Pham et al., 2018)	2.89	4.6	0.5
*NASNet-A (Zoph et al., 2018)	2.65	3.3	1800
*AmoebaNet-B (Real et al., 2019)	2.55±0.05	2.8	3150
*R-DARTS (Zela et al., 2020)	2.95±0.21	-	1.6
*GDAS (Dong and Yang, 2019)	2.93	3.4	0.2
*SNAS (Xie et al., 2019)	2.85	2.8	1.5
*BayesNAS (Zhou et al., 2019)	2.81±0.04	3.4	0.2
*MergeNAS (Wang et al., 2020)	2.73±0.02	2.9	0.2
*NoisyDARTS (Chu et al., 2020b)	2.70±0.23	3.3	0.4
*ASAP (Noy et al., 2020)	2.68±0.11	2.5	0.2
*SDARTS (Chen and Hsieh, 2020)	2.61±0.02	3.3	1.3
*DropNAS (Hong et al., 2020)	2.58±0.14	4.1	0.6
*PC-DARTS (Xu et al., 2020)	2.57±0.07	3.6	0.1
*FairDARTS (Chu et al., 2019)	2.54	3.3	0.4
*DrNAS (Chen et al., 2020)	2.54±0.03	4.0	0.4
*P-DARTS (Chen et al., 2019)	2.50	3.4	0.3
*DARTS-1st (Liu et al., 2019)	3.00±0.14	3.3	0.4
LPT-R18-DARTS-1st (ours)	2.85±0.09	2.7	0.6
*DARTS-2nd (Liu et al., 2019)	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$^-$ (Chu et al., 2020a)	2.59±0.08	3.5	0.4
†DARTS$^-$ (Chu et al., 2020a)	2.97±0.04	3.3	0.4
LPT-R18-DARTS$^-$ (ours)	2.74±0.07	3.4	0.6
$^\Delta$DARTS$^+$ (Liang et al., 2019a)	2.83±0.05	3.7	0.4
LPT-R18-DARTS$^+$ (ours)	2.69±0.05	3.6	0.5

Table 8: Results on CIFAR-10. * means the results are taken from DARTS$^-$ (Chu et al., 2020a), NoisyDARTS (Chu et al., 2020b), and DrNAS (Chen et al., 2020). The rest notations are the same as those in Table 7.

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 advantageous compared with DARTS-1st which uses

Method	Top-1 Error (%)	Top-5 Error (%)	Param (M)	Cost (GPU days)
*Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	-
*MobileNet (Howard et al., 2017)	29.4	10.5	4.2	-
*ShuffleNet 2× (v1) (Zhang et al., 2018)	26.4	10.2	5.4	-
*ShuffleNet 2× (v2) (Ma et al., 2018)	25.1	7.6	7.4	-
*NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	1800
*PNAS (Liu et al., 2018a)	25.8	8.1	5.1	225
*MnasNet-92 (Tan et al., 2019)	25.2	8.0	4.4	1667
*AmoebaNet-C (Real et al., 2019)	24.3	7.6	6.4	3150
*SNAS (Xie et al., 2019)	27.3	9.2	4.3	1.5
*BayesNAS (Zhou et al., 2019)	26.5	8.9	3.9	0.2
*PARSEC (Casale et al., 2019)	26.0	8.4	5.6	1.0
*GDAS (Dong and Yang, 2019)	26.0	8.5	5.3	0.2
*DSNAS (Hu et al., 2020)	25.7	8.1	-	-
*SDARTS-ADV (Chen and Hsieh, 2020)	25.2	7.8	5.4	1.3
*PC-DARTS (Xu et al., 2020)	25.1	7.8	5.3	0.1
*ProxylessNAS (Cai et al., 2019)	24.9	7.5	7.1	8.3
*FairDARTS (Chu et al., 2019)	24.9	7.5	4.8	0.4
*P-DARTS (CIFAR-100) (Chen et al., 2019)	24.7	7.5	5.1	0.3
*P-DARTS (CIFAR-10) (Chen et al., 2019)	24.4	7.4	4.9	0.3
*FairDARTS (Chu et al., 2019)	24.4	7.4	4.3	3.0
*DrNAS (Chen et al., 2020)	24.2	7.3	5.2	3.9
*PC-DARTS (Xu et al., 2020)	24.2	7.3	5.3	3.8
*DARTS$^+$ (Liang et al., 2019a)	23.9	7.4	5.1	6.8
*DARTS$^-$ (Chu et al., 2020a)	23.8	7.0	4.9	4.5
*DARTS$^+$ (CIFAR-100) (Liang et al., 2019a)	23.7	7.2	5.1	0.2
*DARTS-2nd-CIFAR-10 (Liu et al., 2019)	26.7	8.7	4.7	4.0
LPT-R18-DARTS-2nd-CIFAR-10 (ours)	25.3	7.9	4.7	4.0

Table 9: Results on ImageNet, including top-1 and top-5 classification errors on the test set, number of weight parameters (millions), and search cost (GPU days). * means the results are taken from DARTS$^-$ (Chu et al., 2020a) and DrNAS (Chen et al., 2020). The rest notations are the same as those in Table 7. The first row block shows networks designed by humans manually. The second row block shows non-gradient based search methods. The third block shows gradient-based methods.

first-order approximation; 2) In our run of DARTS$^-$, the performance reported in (Chu et al., 2020a) cannot be achieved; 3) In our run of DARTS$^+$, in the architecture evaluation stage, we set the number of epochs to 600 instead of 2000 as used in (Liang et al., 2019a), to ensure a fair comparison with other methods (where the epoch number is 600).

Table 8 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$^-$, 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.

Table 9 shows the results on ImageNet, including top-1 and top-5 classification errors on the test set, number of weight parameters (millions), and search costs (GPU days). Following (Liu et al., 2019), we take the architecture searched by LPT-R18-DARTS-2nd on CIFAR-10 and evaluate it on ImageNet. As can be seen, applying our LPT method to DARTS-2nd, the top-1 error reduces from 26.7% to 25.3% and the top-5 error reduces from 8.7% to 7.9%, without increasing the search cost and parameter number. This further demonstrates the effectiveness of our method.

3.2.4. 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 a test creator. The corresponding formulation is:

$$\begin{array}{l} \max_S \min_A \;\; \frac{1}{\sum_{d\in D_b} s(d)} \sum_{d\in D_b} s(d)\ell(A, W^*(A), d) \\ s.t.\;\; W^*(A) = \min_W \;\; L\left(A, W, D_{ee}^{(\mathrm{tr})}\right) \end{array} \tag{25}$$

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, to avoid performance collapse because of skip connections, LPT is applied to P-DARTS. 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 create test, without using the training data of the target task. The corresponding formulation is:

$$\begin{array}{l} \max_C \min_A \;\; \frac{1}{|\sigma(C,E^*(C),D_b)|} L\left(A, W^*\left(A\right), \sigma\left(C, E^*(C), D_b\right)\right) - \lambda L\left(E^*(C), X^*(C), D_{er}^{(\mathrm{val})}\right) \\ s.t.\;\; E^*(C), X^*(C) = \min_{E,X} \;\; L\left(E, X, \sigma\left(C, E, D_b\right)\right) \\ \qquad W^*\left(A\right) = \min_W \;\; L\left(A, W, D_{ee}^{(\mathrm{tr})}\right) \end{array} \tag{26}$$

In this study, λ and γ are both set to 1. The data encoder of the tester is ResNet-18. For CIFAR-100, to avoid performance collapse because of skip connections, LPT is applied to P-DARTS. 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.(11) increases. In this study, the other tradeoff parameter γ in Eq.(11) 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 study. The rest 45K data (22.5K training data and 22.5K

Method	Error (%)
Difficulty only (CIFAR-100)	18.12±0.11
Difficulty + meaningfulness (CIFAR-100)	**17.18**±0.12
Difficulty only (CIFAR-10)	2.79±0.06
Difficulty + meaningfulness (CIFAR-10)	**2.72**±0.07

Table 10: Results for ablation setting 1. "Difficulty 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 to learn to perform the target task. "Difficulty + meaningfulness" denotes the full LPT framework where the tester creates tests by maximizing both difficulty and meaningfulness.

Method	Error (%)
Test only (CIFAR-100)	17.54±0.07
Test + Training data (CIFAR-100)	**17.18**±0.12
Test only (CIFAR-10)	2.75±0.03
Test + Training data (CIFAR-10)	**2.72**±0.07

Table 11: Results for ablation setting 2. "Test only" denotes that the tester is trained only using the create test to perform the target task. "Test + Training data" denotes that the tester is trained using both the test and the training data of the target task.

validation data) is used for architecture search and evaluation. 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 γ 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 and evaluated on 45K data. Tester's data encoder is ResNe-18. LPT is applied to P-DARTS.

Table 10 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 learner 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 11 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 4: How errors change as increases.

Figure 5: How errors change as increases.

Figure 4 shows how classi cation errors change as increases. As can be seen, on both CIFAR-100 and CIFAR-10, when increases from 0.1 to 0.5, the error decreases. However, further increasing renders the error to increase. From the tester's perspective, explores a tradeo between di culty and meaningfulness of the tests. Increasing encourages the tester to create tests that are more meaningful. Tests with more meaningfulness can more reliably evaluate the learner. However, if is too large, the tests are biased to be more meaningful and less di cult. Lacking enough di culty, the tests may not be compelling enough to drive the learner for improvement. Such a tradeo e ect is observed in the results on CIFAR-10 as well.

Figure 5 shows how classi cation errors change as increases. As can be seen, on both CIFAR-100 and CIFAR-10, when increases from 0.1 to 0.5, the error decreases. However, further increasing renders the error to increase. Under a larger , 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 is too large, the 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 further degrades the quality of test creation.

3.3. Summary

In this section, we apply Skillearn to formalize a skill in human learning { learning by passing tests (LPT) and use it for neural architecture search. In LPT, a tester model problem is:

$$\min_A \sum_{k=1}^{K} L(A, \widetilde{W}_k^{(M)}(A), \widetilde{H}_k^{(M)}(A), D_k^{(\text{val})}). \tag{31}$$

Putting all these pieces together, we instantiate the Skillearn framework to an interleaving learning framework, as shown in Eq.(32). From bottom to top, the K learners perform M rounds of interleaving learning. Learners in adjacent learning stages are coupled via the interaction function. The architecture A is not updated in the learning stages. It is learned by minimizing the validation loss. Table 16 summarizes the key elements of interleaving learning under the Skillearn terminology.

$$\begin{array}{ll}
\min_A & \sum_{k=1}^{K} L(A, \widetilde{W}_k^{(M)}(A), \widetilde{H}_k^{(M)}(A), D_k^{(\text{val})}) \\
s.t. & \textbf{Round M}: \\
& \widetilde{W}_K^{(M)}(A), \widetilde{H}_K^{(M)}(A) = \min_{W_K^{(M)}, H_K^{(M)}} \quad L(A, W_K^{(M)}, H_K^{(M)}, D_K^{(\text{tr})}) + \lambda \| W_K^{(M)} - \widetilde{W}_{K-1}^{(M)}(A) \|_2^2 \\
& \dots \\
& \widetilde{W}_1^{(M)}(A), \widetilde{H}_1^{(M)}(A) = \min_{W_1^{(M)}, H_1^{(M)}} \quad L(A, W_1^{(M)}, H_1^{(M)}, D_1^{(\text{tr})}) + \lambda \| W_1^{(M)} - \widetilde{W}_K^{(M-1)}(A) \|_2^2 \\
& \dots \\
& \textbf{Round 2}: \\
& \widetilde{W}_K^{(2)}(A) = \min_{W_K^{(2)}, H_K^{(2)}} \quad L(A, W_K^{(2)}, H_K^{(2)}, D_K^{(\text{tr})}) + \lambda \| W_K^{(2)} - \widetilde{W}_{K-1}^{(2)}(A) \|_2^2 \\
& \dots \\
& \widetilde{W}_1^{(2)}(A) = \min_{W_1^{(2)}, H_1^{(2)}} \quad L(A, W_1^{(2)}, H_1^{(2)}, D_1^{(\text{tr})}) + \lambda \| W_1^{(2)} - \widetilde{W}_K^{(1)}(A) \|_2^2 \\
& \textbf{Round 1}: \\
& \widetilde{W}_K^{(1)}(A) = \min_{W_K^{(1)}, H_K^{(1)}} \quad L(A, W_K^{(1)}, H_K^{(1)}, D_K^{(\text{tr})}) + \lambda \| W_K^{(1)} - \widetilde{W}_{K-1}^{(1)}(A) \|_2^2 \\
& \dots \\
& \widetilde{W}_k^{(1)}(A) = \min_{W_k^{(1)}, H_k^{(1)}} \quad L(A, W_k^{(1)}, H_k^{(1)}, D_k^{(\text{tr})}) + \lambda \| W_k^{(1)} - \widetilde{W}_{k-1}^{(1)}(A) \|_2^2 \\
& \dots \\
& \widetilde{W}_2^{(1)}(A) = \min_{W_2^{(1)}, H_2^{(1)}} \quad L(A, W_2^{(1)}, H_2^{(1)}, D_2^{(\text{tr})}) + \lambda \| W_2^{(1)} - \widetilde{W}_1^{(1)}(A) \|_2^2 \\
& \widetilde{W}_1^{(1)}(A) = \min_{W_1^{(1)}, H_1^{(1)}} \quad L(A, W_1^{(1)}, H_1^{(1)}, D_1^{(\text{tr})})
\end{array} \tag{32}$$

4.2. Optimization Algorithm

In this section, we develop an optimization algorithm for interleaving learning by instantiating the general optimization framework of Skillearn in Section 2.3. 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 $W_k^{(m)}$:

$$\widetilde{W}_k^{(m)}(A) \approx \overline{W}_k^{(m)}(A) = W_k^{(m)} - \eta \nabla_{W_k^{(m)}} (L(A, W_k^{(m)}, H_k^{(m)}, D_k^{(\text{tr})}) + \lambda \| W_k^{(m)} - \widetilde{W}_{k-1}^{(m)}(A) \|_2^2) \tag{33}$$

Skillearn	Interleaving Learning
Learners	K learners
Learnable parameters	1) Encoder architecture shared by all learners; 2) In each round, each learner has weight parameters for the data encoder and weight parameters for the task-specific head.
Interaction function	The encoder weights W_l at learning stage l are encouraged to be close to the optimal encoder weights $\widetilde{W}_{l-1}$ at stage $l-1$: $\|W_l - \widetilde{W}_{l-1}\|_2^2$.
Learning stages	1) In the first learning stage (the first learner in the first round), the learner trains the weights of its data encoder and the weights of its task-specific head on its training dataset: $\widetilde{W}_1^{(1)}(A) = \min_{W_1^{(1)}, H_1^{(1)}} \ L(A, W_1^{(1)}, H_1^{(1)}, D_1^{(\mathrm{tr})})$; 2) In other learning stages, the learner trains the weights of its data encoder and the weights of its task-specific head on its training dataset where the encoder weights are encouraged to be close to the optimal encoder weights trained in the previous stage: $\widetilde{W}_k^{(m)}(A) = \min_{W_k^{(m)}, H_k^{(m)}} \ L(A, W_k^{(m)}, H_k^{(m)}, D_k^{(\mathrm{tr})}) + \lambda\|W_k^{(m)} - \widetilde{W}_{k-1}^{(m)}(A)\|_2^2$.
Validation stage	Each learner validates its optimal data encoder and task-specific head learned in the last round on its validation dataset.
Datasets	Each learner has a training dataset and a validation dataset.

Table 16: Mapping from Skillearn to Interleaving Learning

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

$$\widetilde{W}_1^{(1)}(A) \approx \overline{W}_1^{(1)}(A) = W_1^{(1)} - \eta\nabla_{W_1^{(1)}} L(A, W_1^{(1)}, H_1^{(1)}, D_1^{(\mathrm{tr})}) \tag{34}$$

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

$$\widetilde{W}_k^{(m)}(A) \approx \overline{W}_k^{(m)}(A) = W_k^{(m)} - \eta\nabla_{W_k^{(m)}} L(A, W_k^{(m)}, H_k^{(m)}, D_k^{(\mathrm{tr})}) - 2\eta\lambda(W_k^{(m)} - \overline{W}_{k-1}^{(m)}(A)) \tag{35}$$

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:

$$\widetilde{H}_k^{(M)}(A) \approx \overline{H}_k^{(M)}(A) = H_k^{(M)}(A) - \eta\nabla_{H_k^{(M)}(A)} L(A, W_k^{(M)}, H_k^{(M)}, D_k^{(\mathrm{tr})}) \tag{36}$$

In the validation stage, we plug in 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:

$$A \leftarrow A - \eta \sum_{k=1}^{K} \nabla_A L(A, \overline{W}_k^{(M)}(A), \overline{H}_k^{(M)}(A), D_k^{(\mathrm{val})}) \tag{37}$$

The update steps from Eq.(34) to Eq.(37) 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.(34)
    2. For $k = 2 \cdots K$, update $\widetilde{W}_k^{(1)}(A)$ using Eq.(35)
    3. For $k = 1 \cdots K$ and $m = 2 \cdots M$, update $\widetilde{W}_k^{(m)}(A)$ using Eq.(35)
    4. For $k = 1 \cdots K$, update $\widetilde{H}_k^{(M)}(A)$ using Eq.(36)
    5. Update $A$ using Eq.(37)
end
```

4.3. Experiments

We apply interleaving learning for neural architecture search in image classification tasks. Two tasks are interleaved: image classification on CIFAR-10 and image classification on CIFAR-100. We search the shared architecture of data encoders in these two tasks. The search space is the same as that in DARTS (Liu et al., 2019). For CIFAR-10 which has 10 classes, the task-specific head is a 10-way linear classifier. For CIFAR-100 which has 100 classes, the head is a 100-way linear classifier. Similar to Section 3.2, following (Liu et al., 2019), we first perform architecture search which finds out 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 the test set. Architecture search is performed jointly on CIFAR-10 and CIFAR-100 via interleaving. Architecture evaluation is performed separately on CIFAR-10 and CIFAR-100. In the interleaving process, we set the number of rounds to 2. The training and validation datasets of CIFAR-10 and CIFAR-100 are the same as those described in Section 3.2.1. The tradeoff parameter γ is set to 1. The rest of experimental settings are the same as those described in Section 3.2.2.

4.3.1. Results

Table 17 and Table 18 shows the classification error (%), number of weight parameters (millions), and search cost (GPU days) of different NAS methods on CIFAR-100 and CIFAR-10 respectively. As can be seen, when applied to DARTS-2nd, our interleaving learning (IL) method achieves great improvement on CIFAR-100 and slight improvement on CIFAR-10. On CIFAR-100, our proposed IL-DARTS-2nd achieves an average error of 17.46%, which is significantly lower than the 20.58% error of DARTS-2nd. 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,

Method	Error(%)	Param(M)	Cost
*ResNet (He et al., 2016a)	22.10	1.7	-
*DenseNet (Huang et al., 2017)	17.18	25.6	-
*PNAS (Liu et al., 2018a)	19.53	3.2	150
*ENAS (Pham et al., 2018)	19.43	4.6	0.5
*AmoebaNet (Real et al., 2019)	18.93	3.1	3150
†DARTS-1st (Liu et al., 2019)	20.52±0.31	1.8	0.4
*GDAS (Dong and Yang, 2019)	18.38	3.4	0.2
*R-DARTS (Zela et al., 2020)	18.01±0.26	-	1.6
*DARTS$^-$ (Chu et al., 2020a)	17.51±0.25	3.3	0.4
†DARTS$^-$ (Chu et al., 2020a)	18.97±0.16	3.1	0.4
*P-DARTS (Chen et al., 2019)	17.49	3.6	0.3
$^\Delta$DARTS$^+$ (Liang et al., 2019a)	17.11±0.43	3.8	0.2
*DropNAS (Hong et al., 2020)	16.39	4.4	0.7
*DARTS-2nd (Liu et al., 2019)	20.58±0.44	1.8	1.5
IL-DARTS-2nd (ours)	17.46±0.25	2.1	1.8

Table 17: Results on CIFAR-100, including classification error (%) on the test set, number of parameters (millions) in the searched architecture, and search cost (GPU days). IL-DARTS-2nd denotes that our proposed interleaving learning (IL) framework is applied to the search space of DARTS. DARTS-1st and DARTS-2nd denotes that first order and second order approximation is used in DARTS. * means the results are taken from DARTS$^-$ (Chu et al., 2020a). † 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.

which enables the learning tasks on CIFAR-100 and CIFAR-10 to mutually benefit each other. In contrast, in the DARTS-2nd baseline, 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.

One may wonder whether the performance gain in IL is simply due to more data being used (CIFAR-100 plus CIFAR-10), rather than because of the interleaving mechanism. To answer this question, we perform an ablation study where the encoder is learned in a multi-task learning framework with one task on CIFAR-100 and the other task on CIFAR-10. The formulation is:

$$\begin{array}{ll} \min_A & L(A, \widetilde{W}_{100}(A), \widetilde{H}_{100}(A), D_{100}^{(\mathrm{val})}) + \lambda L(A, \widetilde{W}_{10}(A), \widetilde{H}_{10}(A), D_{10}^{(\mathrm{val})}) \\ s.t. & \widetilde{W}_{100}(A), \widetilde{H}_{100}(A), \widetilde{W}_{10}(A), \widetilde{H}_{10}(A) = \\ & \min_{W_{100},H_{100},W_{10},H_{10}} \quad L(A, W_{100}, H_{100}, D_{100}^{(\mathrm{tr})}) + \gamma L(A, W_{10}, H_{10}, D_{10}^{(\mathrm{tr})}) \end{array} \tag{38}$$

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. $D_{100}^{(\mathrm{tr})}$ and $D_{100}^{(\mathrm{val})}$

Method	Error(%)	Param(M)	Cost
*DenseNet (Huang et al., 2017)	3.46	25.6	-
*HierEvol (Liu et al., 2018b)	3.75±0.12	15.7	300
*NAONet-WS (Luo et al., 2018)	3.53	3.1	0.4
*PNAS (Liu et al., 2018a)	3.41±0.09	3.2	225
*ENAS (Pham et al., 2018)	2.89	4.6	0.5
*NASNet-A (Zoph et al., 2018)	2.65	3.3	1800
*AmoebaNet-B (Real et al., 2019)	2.55±0.05	2.8	3150
*DARTS-1st (Liu et al., 2019)	3.00±0.14	3.3	0.4
*R-DARTS (Zela et al., 2020)	2.95±0.21	-	1.6
*GDAS (Dong and Yang, 2019)	2.93	3.4	0.2
*SNAS (Xie et al., 2019)	2.85	2.8	1.5
$^\triangle$DARTS$^+$ (Liang et al., 2019a)	2.83±0.05	3.7	0.4
*BayesNAS (Zhou et al., 2019)	2.81±0.04	3.4	0.2
*MergeNAS (Wang et al., 2020)	2.73±0.02	2.9	0.2
*NoisyDARTS (Chu et al., 2020b)	2.70±0.23	3.3	0.4
*ASAP (Noy et al., 2020)	2.68±0.11	2.5	0.2
*SDARTS (Chen and Hsieh, 2020)	2.61±0.02	3.3	1.3
*DARTS$^-$ (Chu et al., 2020a)	2.59±0.08	3.5	0.4
†DARTS$^-$ (Chu et al., 2020a)	2.97±0.04	3.3	0.4
*DropNAS (Hong et al., 2020)	2.58±0.14	4.1	0.6
*PC-DARTS (Xu et al., 2020)	2.57±0.07	3.6	0.1
*FairDARTS (Chu et al., 2019)	2.54	3.3	0.4
*DrNAS (Chen et al., 2020)	2.54±0.03	4.0	0.4
*P-DARTS (Chen et al., 2019)	2.50	3.4	0.3
*DARTS-2nd (Liu et al., 2019)	2.76±0.09	3.3	1.5
LPT-R18-DARTS-2nd (ours)	2.74±0.07	3.4	1.8

Table 18: Results on CIFAR-10. * means the results are taken from DARTS$^-$ (Chu et al., 2020a), NoisyDARTS (Chu et al., 2020b), and DrNAS (Chen et al., 2020). The rest notations are the same as those in Table 17.

are the training and validation sets of CIFAR-100. $D_{10}^{(\mathrm{tr})}$ and $D_{10}^{(\mathrm{val})}$ are the training and validation sets of CIFAR-10. A is the encoder architecture shared by CIFAR-100 and CIFAR-10. λ and γ are both set to 1. Table 19 compares the classification errors achieved by this multi-task learning method and our proposed interleaving learning method. As can be seen, interleaving learning (IL) performs better than multi-task learning (MTL). 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. These results further demonstrate the effectiveness of interleaving.

Method	Error (%) on CIFAR-100	Error (%) on CIFAR-10
Multi-task learning	18.88	2.93
Interleaving learning	**17.46**	**2.74**

Table 19: Classification errors achieved by multi-task learning and interleaving learning on CIFAR-100 and CIFAR-10.

4.4. Summary

In this section, we apply Skillearn to formalize the interleaving learning (IL) skill of humans. In IL, 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. Via interleaving, different models transfer their learned knowledge to each other to better represent data and avoid being stuck in bad local optimums. Experiments of neural architecture search on CIFAR-100 and CIFAR-10 demonstrate the effectiveness of interleaving learning.

4.5. Related Works

4.5.1. Neural Architecture Search

Neural architecture search (NAS) has achieved remarkable progress recently, which aims at searching for the optimal architecture of neural networks to achieve the best predictive performance. In general, there are three paradigms of methods in NAS: reinforcement learning (RL) approaches (Zoph and Le, 2017; Pham et al., 2018; Zoph et al., 2018), evolutionary learning approaches (Liu et al., 2018b; Real et al., 2019), and differentiable approaches (Cai et al., 2019; Liu et al., 2019; Xie et al., 2019). In RL-based approaches, a policy is learned to iteratively generate new architectures by maximizing a reward which is the accuracy on the validation set. Evolutionary 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 (Chen et al., 2019), the depth of searched architectures is allowed to grow progressively during the training process. Search space approximation and regularization approaches are developed to reduce computational overheads and improve search stability. PC-DARTS (Xu et al., 2020) reduces the redundancy in exploring the search space by sampling a small portion of a super network. Operation search is performed in a subset of channels with the held-out part bypassed in a shortcut. Our proposed LCT framework can be applied to any differentiable NAS methods.

4.5.2. Adversarial Learning

Our proposed LPT involves a min-max optimization problem, which is analogous to that in adversarial learning. Adversarial learning (Goodfellow et al., 2014a) has been widely applied to 1) data generation (Goodfellow et al., 2014a; Yu et al., 2017) 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 (Ganin and Lempitsky, 2015) 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 (Goodfellow et al., 2014b) 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. (2020) proposed to use an adversarial examiner to identify the weakness of a trained model. Our work differs from this one in that we progressively re-train a learner model based on how it performs on the tests dynamically created by a tester model while the learner model in (Shu et al., 2020) is fixed and not affected by the examination results.

5. Conclusions

In this paper, we develop a general framework called Skillearn to formalize humans' learning skills into machine-executable learning skills and leverage them to train better machine learning models. Our framework can flexibly formulate many learning skills of humans, by mapping from learners, learnable parameters, interaction functions, learning stages, etc. in Skillearn to their counterparts in human learning. The formulated machine-executable learning skills can be applied to improve any ML model. In two case studies, we apply Skillearn to formalize two learning skills of humans – learning by passing tests (LPT) and interleaving learning (IL). In LPT, a tester model dynamically creates tests with increasing levels of difficulty to evaluate a testee model; the testee continuously improves its architecture by passing however difficult tests created by the tester. In IL, 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. Experiments on various datasets demonstrate that ML models trained by these two learning skills achieve significantly better performance.

References

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

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

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

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

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

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

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

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

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

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

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

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, 2014a.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

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

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

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

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

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

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

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

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

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

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

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.

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

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

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

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

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

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

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

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

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.

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

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

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.

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

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

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

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

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

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

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

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