Learning From Mistakes: A Multilevel Optimization Framework

Li Zhang, Bhanu Garg, Pradyumna Sridhara, Ramtin Hosseini, and Pengtao Xie

***Abstract*—Bi-level optimization methods in machine learning are popularly effective in subdomains of neural architecture search, data reweighting, etc. However, most of these methods do not factor in variations in learning difficulty, which limits their performance in real-world applications. To address the above problems, we propose a framework that imitates the learning process of humans. In human learning, learners usually focus more on the topics where mistakes have been made in the past to deepen their understanding and master the knowledge. Inspired by this effective human learning technique, we propose a multilevel optimization framework, learning from mistakes (LFM), for machine learning. We formulate LFM as a three-stage optimization problem: 1) the learner learns, 2) the learner relearns based on the mistakes made before, and 3) the learner validates his learning. We develop an efficient algorithm to solve the optimization problem. We further apply our method to differentiable neural architecture search and data reweighting. Extensive experiments on CIFAR-10, CIFAR-100, ImageNet, and other related datasets powerfully demonstrate the effectiveness of our approach. The code of LFM is available at: https://github.com/importZL/LFM.**

***Impact Statement*—Bi-level optimization (BLO) has emerged as a compelling approach in machine learning, offering a hierarchical solution to complex optimization challenges. However, conventional BLO methods often struggle with learning difficulty variations present in real-world applications. To this end, we introduce learning from mistakes (LFM), a novel framework inspired by human learning. LFM automatically adjusts training example weights based on learning difficulties, significantly enhancing model robustness. Integrated into neural architecture search (NAS) and data reweighting (DR), LFM demonstrates remarkable improvements in adaptability and reliability across scenarios like class imbalance and noisy labels. This work marks a pivotal step towards more effective optimization of machine learning models, crucial for addressing complex real-world challenges.**

Received 7 May 2024; revised 13 December 2024; accepted 20 January 2025. Date of publication 27 January 2025; date of current version 30 May 2025. This work was supported in part by NSF under Grant IIS2405974 and Grant IIS2339216. This article was recommended for publication by Associate Editor Diego Oliva upon evaluation of the reviewers' comments. *(Corresponding author: Pengtao Xie.)*

The authors are with the Department of Electrical and Computer Engineering, University of California, San Diego, CA 92093 USA (e-mail: liz042@ucsd.edu; bgarg@ucsd.edu; prsridha@ucsd.edu; rhossein@eng.ucsd.edu; p1xie@ucsd.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TAI.2025.3534151, provided by the authors.

Digital Object Identifier 10.1109/TAI.2025.3534151

***Index Terms*—Data reweighting (DR), learning from mistakes (LFM), multilevel optimization, neural architecture search (NAS).**

I. Introduction

BI-LEVEL optimization (BLO) is a hierarchical optimization problem of two or more layers [1] and is recently gaining popularity in machine learning (ML). In BLO, the outer optimization problem (upper-level problem) is restricted by the solution set mapping of the inner-level optimization problem (lower-level problem) [2]. Common BLO-based methods include neural architecture search (NAS) [3], [4], [5] and data reweighting (DR) [6], [7], [8], etc. Most BLO-based methods update the model weights by minimizing the training loss. In contrast, the meta parameters (architecture parameters, weights of data examples, etc.) are learned by minimizing the validation loss. This approach [9] has shown success in tasks such as image classification, object detection, etc.

Most traditional BLO approaches do not factor in variations in learning difficulty. Consider the case of challenging images obtained by driving the vehicle in harsh weather conditions, complex backgrounds, etc., in deep learning-based self-driving applications [10]; or deep learning in healthcare, where the training data are highly heterogeneous, ambiguous, noisy, and with an imbalanced distribution [11]—the BLO-based methods can easily overfit and result in instability of prediction results [12], [13].

To address the above issues, researchers propose various DR strategies like AdaBoost [14], Focal Loss [15], and Active Bias [16], which monotonically increase the weights of samples with larger loss because they may be samples whose features are difficult to learn or samples with class imbalance. Another paradigm—Self-paced Learning [17], MentorNet [18], and Iterative Reweighting [19] aims to emphasize samples with a smaller loss. The rationale is that samples with larger losses are likely to have corrupted labels. However, both paradigms design a specific form of the reweighting function to weigh samples based on expert opinion because it's hard to determine the relationship between data weights and loss values. Moreover, the specificity of the weighting function limits the applicability of the method to other situations.

Standing around the above problems, this article proposes an original multi-level framework, learning from mistakes (LFM), inspired by the practical learning technique of humans to calculate the weights for training samples automatically. Over the

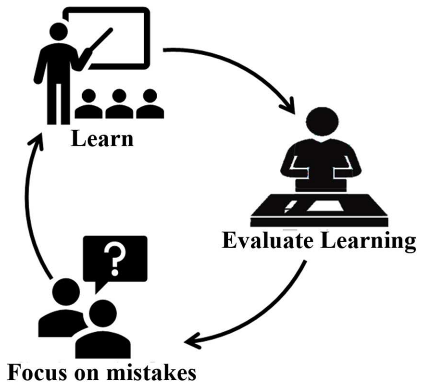

Fig. 1. The simple process flow of human learning.

years, humans have accumulated a lot of valuable learning techniques. One such effective learning method is to learn from previous mistakes. As shown in Fig. 1, initially, the learner learns a concept and evaluates themselves through a test to measure their level of understanding. The topics in the idea where the learner makes more mistakes are identified as not having been learned well by the learner. Therefore, the learner will restudy the issue while focusing on the topics where the learner made mistakes before. The above learning process can prevent the repetition of similar errors in the future while also strengthening previously well-learned issues. Inspired by this human learning technique, we propose a methodology that can be applied to the training process of machine learning to improve its performance.

The major contributions of this article are as follows.

1) Inspired by the human learning process, we propose a novel optimization framework, Learning From Mistakes (LFM), which can apply to the most regular model training process of machine learning.
2) We formulate LFM as a multi-level optimization framework that includes three steps: learner learns; learner relearns to correct its mistakes; learner validates its performance.
3) We applied LFM to NAS and DR, aiming to verify the effectiveness of our method. We conducted a series of experiments, including experiments about NAS, experiments under class imbalance, and noisy label cases. The results demonstrate the effectiveness of LFM in improving the robustness of a learning algorithm on biased training samples.

II. Related Works

A. BLO

BLO derives from the area of economic game theory [2] and has been introduced in model optimization. BLO can solve problems that involve two levels of optimization tasks, of which one task is usually nested inside the other, including hyperparameter optimization [20], meta-learning [21], NAS [22], DR [6], etc. These methods optimize meta-parameters (e.g., neural architectures, data weights, etc.) by minimizing average validation loss in the upper-level tasks. Model weights are updated by minimizing average training loss in the lower-level functions. For example, Franceschi et al. [23] presented a novel BLO framework that uses average validation loss in hyperparameter optimization. These methods can achieve tremendous average-case performance but are more likely to perform poorly in worst-case scenarios. To address this problem, Shu et al. [24] dynamically selected a sequence of validations based on adversarial examination. While this work does not focus on average-case performance, it also does not leverage the evaluation results to retrain the model for further improvement.

To address the limitations, our methods can automatically calculate a training weight for each training sample based on the evaluation results of the validation set.

B. NAS

Recently, NAS has come to the forefront of deep learning techniques due to its success in discovering neural architectures that can substantially outperform manually designed ones. Early versions of NAS such as [3], [25], [26] used computationally intensive approaches like reinforcement learning—where the accuracy of the validation set was defined as the reward and a policy network was trained to generate architectures that can maximize these rewards. Another contemporary approach [4], [27] was using evolutionary learning techniques—where the set of all architectures represents a population, and the fitness score is the validation accuracy of each architecture. The architectures with lower fitness scores would be replaced with higher fitness score architectures. However, even this approach is computationally intensive. To address this problem, differentiable architecture search techniques were explored [5], [28], [29] and their results are much more promising because of the use of weight-sharing techniques and the application of gradient descent in a continuous architecture search space.

Differentiable architecture search (DARTS) [5] made the first breakthrough in Differentiable NAS. Several other DARTS-based techniques [30], [31], [32], [33] have been explored to reduce the cost of computation for differentiable NAS. Some of the approaches include—Progressive differentiable architecture search (PDARTS) [30] increases the depth of architectures progressively during the search, Partial channel connections for memory-efficient architecture search (PC-DARTS [31]) evaluates only a subset of channels, thereby reducing the search space's redundancy. The LFM framework proposed in this article can be applied to any differentiable NAS method for further enhancement.

C. DR

DR has been well-studied in the literature. The broad paradigms include methods that reweight the samples based on specific prior knowledge of the task or data. For example, synthetic minority over-sampling technique (SMOTE) [6] randomly synthesizes data to supplement categories with small samples after analyzing the data distribution first. Additionally, Zadrozny et al. [7] presented a bias correction method for handling different distributions between training data and test examples.

Furthermore, several works [8], [34] proposed designing a weighting function mapping that can assign weights to samples based on training loss. There are two different principles of the weighting function. One principle is to increase the weights of samples with higher losses. For example, AdaBoost [35] trains subsequent classifiers based on data selected from more difficult training samples, while hard example mining [36] trains Exemplar-support vector machines (SVMs) [37] to exploit challenging training samples and downsample the majority category. Focal loss [15] emphasizes more difficult training samples based on a soft weighting scheme. These methods heavily weigh samples with higher losses, making them suitable for datasets with class imbalance. On the other hand, another paradigm of methods imposes higher weights on samples with smaller loss values. For example, self-paced learning (SPL) [17] prioritizes training easier samples first, MentorNet [18] uses a meta-learning long short-term memory [38] to calculate weights for data with potentially corrupted labels, and curriculum learning [39] prioritizes easier training sets. This strategy is effective for datasets with label noise. However, practical datasets often exhibit both class imbalance and label noise simultaneously, making it challenging to strictly increase or decrease weights based solely on loss values. Instead, a balanced approach is necessary based on the specific dataset characteristics. Currently, most methods require manual design of a specific weighting function based on domain knowledge.

Unlike the mentioned methods, our method, LFM, focuses on the samples likely to make mistakes rather than simply increasing or decreasing the weights of training samples based on the losses. For data with class imbalance, the data loss corresponding to the category with a smaller sample size must be higher, so using label similarity, the samples' weights of this category are increased. And for data with label noise, even if two training samples have the same label, their visual similarity is likely to be smaller, so we can avoid increasing the weight of the data with a corrupted label.

III. Methods

In this section, we propose a framework that can imitate human learning in the form of LFM and present an optimization algorithm for solving the problem of LFM when applying to NAS and DR with the fixed network.

A. Overview

Inspired by an effective human learning technique, LFM, where the learners focus more on the topics where they made mistakes, to deepen their understanding, we investigate if machine learning methods can apply this human learning strategy. We propose a novel machine learning framework called LFM, wherein the learner improves his ability to learn by focusing more on the mistakes during revision.

The framework contains two sets of network weights W_1 and W_2—that are two parts of the same learner and are trying to learn to perform the same target task, assuming the classification task in this article. The primary goal of W_2 here is to help the first learner, W_1, correct the mistakes (made when studying for the first time) during the revision. Further, to help map the topics in the test to issues in the syllabus, there is an encoder with predefined neural architecture (by human experts) with learnable network weights V, a coefficient vector r, and a learnable weight set B.

We begin by training the first network's weights on a training dataset. We then see what mistakes our model makes while predicting the validation set. Then, for each training example, we assign specific weights based on the errors made by the model and the similarity of this example to an incorrectly predicted validation example. More specifically, these weights are computed based on a combination of validation performance and the similarity between training and validation examples. The second set of network weights is trained on these weighted examples, making the model learn from its mistakes and correct them. In this way, each training sample automatically assigns a weight based on the model's performance on the validation set. Finally, the learnable weight set, encoder, and coefficient vectors are updated based on the second model's validation performance.

B. The Multilevel Optimization Framework

We organize our framework into three stages.

Stage I: In the first stage, we train the first set of network weights W_1 by minimizing the loss on the training dataset $D^{(tr)}$. The optimal weights $W_1^*(B)$ is a function of hyperparameter B, which at this stage is fixed, and hence

$$W_1^*(B) = \arg\min_{W_1} L(B, W_1, D^{tr}). \tag{1}$$

The hyperparameter B could be an architecture parameter of the network or a weight parameter for training samples, which is used to define the training loss but is not updated at this stage. If we were to directly learn B by minimizing this training loss, a trivial solution would be yielded where B is very large and complex that would perfectly overfit the training data but generalize poorly on unseen data.

Stage II: In the second stage, the goal is to reweight the training samples based on LFM for training the learner's second set of network weights W_2. We apply $W_1^*(B)$ to the validation dataset $D^{(val)}$ and check its performance on the validation examples. To make the model relearn while paying more attention to mistakes in the validation examples by $W_1^*(B)$, we reweight each training example $d_i^{(tr)}$ based on the following metrics.

1) Visual similarity between $d_i^{(tr)}$ and $d_j^{(val)}$, as x_{ij}
2) Label similarity of $d_i^{(tr)}$ and $d_j^{(val)}$, as z_{ij}
3) Validation performance of $W_1^*(B)$ on $d_j^{(val)}$, as u_j.

In human learning, a question incorrectly learned previously can be corrected during the relearning stage by focusing more on examples similar to the wrongly learned question. Here, the metric x_{ij} tries to measure how similar a previously incorrectly predicted $d_j^{(val)}$ is to a training example $d_i^{(tr)}$ and z_{ij} depicts whether they describe the same topic. The term u_j measures how much $W_1^*(B)$ is wrong for each validation example j.

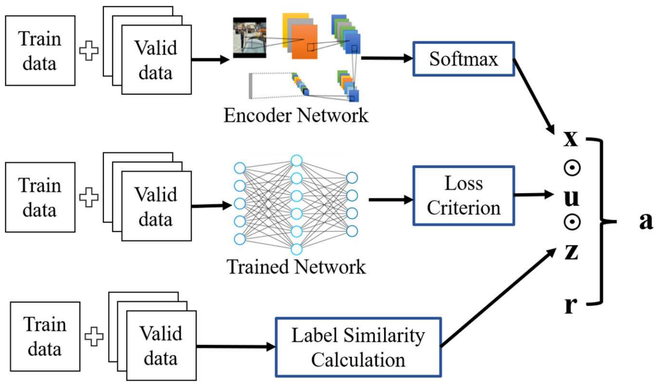

Fig. 2. The process flow to calculate the weights a_i for training example i.

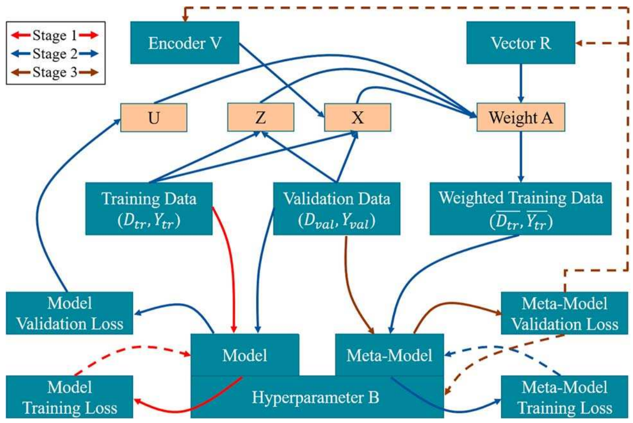
Stage 1
Stage 2
Stage 3
Encoder V
Vector R
U
Z
X
Weight A
Training Data (D_{tr}, Y_{tr})
Validation Data (D_{val}, Y_{val})
Weighted Training Data ($\overline{D_{tr}}, \overline{Y_{tr}}$)
Model Validation Loss
Meta-Model Validation Loss
Model
Meta-Model
Hyperparameter B
Model Training Loss
Meta-Model Training Loss

Fig. 3. The overall process flow of our method. The red arrows indicate stage 1 processes, the blue arrows indicate stage 2 processes, and the black arrows indicate stage 3 processes. In each stage, the solid lines starting from data blocks represent the beginning of each stage, indicating feeding the corresponding data into the model, and the dashed lines represent the end of each stage, indicating updating the corresponding model components.

We use these reweighted training examples to train W_2. This allows W_2 to focus on the topics that W_1, after training, could not get right.

Visual similarity measures how similar the training example $d_i^{(tr)}$ is to each validation example $d_j^{(val)}$. Let V denote an image encoder. For each validation example $d_j^{(val)}$, its similarity with $d_i^{(tr)}$ is defined as the dot-product attention [40] as

$$x_{ij} = \text{align}(V(d_i^{(tr)}), V(d_j^{(val)})) = \frac{exp(V(d_i^{(tr)}).V(d_j^{(val)}))}{\sum_{k=1}^{N^{(val)}} exp(V(d_i^{(tr)}).V(d_k^{(val)}))} \quad (2)$$

where $N^{(val)}$ is the number of validation examples. $V(d)$ denotes the K-dimensional visual representation of the data example d.

Label similarity measures the similarity between the label of the training example and the label of each validation example. Let z_{ij} denote the label similarity between a validation example $d_j^{(val)}$ and a training example $d_j^{(val)}$. We define z_{ij} as

$$z_{ij} = \mathbb{I}\{y_i^{(tr)} = y_j^{(val)}\} \quad (3)$$

where y is the label of the corresponding data example, and $\mathbb{I}\{\}$ is the indicator function on the condition being true or not.

The validation performance u_j of $W_1^*(B)$ on a validation example $d_j^{(val)}$ is the cross entropy loss on this example

$$u_j = \text{crossentropy}(f(d_j^{(val)}; W_1^*(B)), y_j^{(val)}) \quad (4)$$

where $f(d_j^{(val)}; W_1^*(B))$ is the predicted probabilities of $W_1^*(B)$ on $d_j^{(val)}$ and $y_j^{(val)}$ is the class label of $d_j^{(val)}$.

Let x_i, z_i, and u be $N^{(val)}$-dimensional vectors where the jth element is x_{ij}, z_{ij}, and u_j defined before. We calculate the weight a_i of the training example $d_i^{(tr)}$ as

$$a_i = \text{sigmoid}((x_i \odot z_i \odot u)^T r) \quad (5)$$

where $\odot$ denotes element-wise multiplication, and r is a coefficient vector. Note a_i is a function of V, $W_1^*(B)$, and r. We summarise the calculation of a_i in the process flow diagram Fig. 2.

Given the weight a_i of each training example, we train the second set of network weights W_2 by minimizing the weighted training loss, with the architecture, encoder V, and r fixed.

$$W_2^*(W_1^*(B), V, r) = \arg\min_{W_2} \sum_{i=1}^{N^{tr}} a_i \ell(W_2, d_i^{(tr)}, y_i^{(tr)}). \quad (6)$$

Stage III: In the third and final stage, update the encoder V, coefficient vector r, and the hyperparameter B by minimizing the validation loss of $W_2^*(W_1^*(B), V, r)$.

$$B, V, r = \arg\min_{B,V,r} L(W_2^*(W_1^*(B), V, r), D^{(val)}). \quad (7)$$

We summarise the above equations in the process flow diagram Fig. 3. It shows the input data and output data in every stage in detail.

We represent the hyperparameter B of the learner in a differentiable way. In stages 1 and 2, B is fixed and updated in stage 3. In this way, we turn this problem into a tri-level optimization. As shown above, after the weights W_2 of model 2 are trained by correcting the mistakes made by W_1, the parameter B will be updated accordingly since the gradient of B depends on W_2; an updated B will also render W_1 to change as well since the gradient of W_1 depends on B. Along with this $W_1 \to W_2 \to B$ chain, W_1 is indirectly influenced by W_2, since W_2 corrects the mistakes, W_1 will avoid these mistakes in the next round of training as well. All in all, in our end-to-end framework, model 1 will learn from its previous mistakes and avoid making the same mistakes indirectly.

The overall algorithm of LFM is summarised in Algorithm 1.

C. Applications

We apply our framework for two applications: differentiable NAS and DR with fixed human-designed networks.

Differentiable NAS aims to search for high-performance network architecture in a differentiable way. To apply our framework for Differentiable NAS, we set B in (1) to be neural

Algorithm 1: Optimization Algorithm for LFM.

```
1: Input: Sub-datasets $D_{tr}$, $D_{val}$. Parameter initialization $W_1$, $W_2$, $V$, $r$, and $B$.
2: for $t = 1, 2, 3, \cdots$ do
3:     Sample a batch from $D_{tr}$. Update $W_1$ via (1).
4:     Sample a batch from $D_{tr}$. Update $W_2$ via (6.
5:     Sample a batch from $D_{val}$. Update $V$, $r$, $B$ via (7).
6: end for
```

architectures. Similar to DARTS [5], the search space of B is composed of a large number of building blocks, where the output of each block is associated with weight b indicating the importance of the block. Similar to the above, the framework contains two sets of network weights W_1 and W_2. They share a learnable architecture parameter B. The primary goal here is to learn an architecture that performs better. We also organized the learning into three stages. The overall optimization problem with learnable architecture is as follows:

$$\begin{aligned} &B, V, r = \underset{B,V,r}{\arg\min}\, L(B, W_2^*(W_1^*(B), V, r), D^{(val)}) \\ &s.t. \quad W_2^*(B) = \underset{W_2}{\arg\min} \sum_{i=1}^{N^{tr}} a_i \ell(A, W_2(B), d_i^{(tr)}) \\ &\qquad W_1^*(B) = \underset{W_1}{\arg\min}\, L(B, W_1, D^{tr}). \end{aligned} \tag{8}$$

DR aims to identify and remove the influence of train examples, which limits the improvement of models' performance. To apply our framework for DR we specialize (1) to the following:

$$W_1^*(B) = \arg\min_{W_1} \sum_{i=1}^{N^{tr}} b_i L(W_1, d^{tr})$$

where $B = \{b_i\}_{i=1}^{N}$. b_i is the weight of a data sample $d_i^{(tr)}$ and is used to reweight the training loss of each sample. After applying our framework to DR, the model can automatically pay more attention to the examples that would be more difficult to classify during the training process. The overall optimization problem with DR can be summarized as follows:

$$\begin{aligned} &B, V, r = \arg \min_{B,V,r} L(W_2^*(W_1^*(B), V, r), D^{(val)}) \\ &s.t. \quad W_2^*(B) = \arg\min_{W_2} \sum_{i=1}^{N^{tr}} a_i \ell(W_2(B), d_i^{(tr)}) \\ &\qquad W_1^*(B) = \arg\min_{W_1} \sum_{i=1}^{N^{tr}} b_i L(W_1, d^{tr}). \end{aligned} \tag{9}$$

D. Optimization Algorithm

We promote an efficient algorithm to solve the LFM problems when applying to NAS and DR, as described in (8) and (9). Inspired by DARTS [5], we approximate W_1^* and W_2^* by one-step gradient descent updates for the inner optimization equations to reduce the computational complexity. For Stage 1, we approximate $W_1^*(B)$ using one step descent for the loss on training data $L(B, W_1, D^{tr})$, where the hyperparameter B keeps fixed. For Stage 2, we use W_1' from the previous update to get u_j. To get the weights a_i for training samples, we compute x_i and z_i for each training sample d_i^{tr}. Then based on the reweighted training set, we use one-step gradient descent to approximate $W_2^*(B, W_1^*(B), V, r)$, where the hyperparameter B keeps fixed. For Stage 3, we plug W_2' to learn hyperparameter B, encoder V, and coefficient vector r from the validation loss $L(B, W_2'(W_1'(B), V, r), D^{(val)})$.

To save space, the complete derivations of these two applications can be found in Appendix A and Appendix B of the supplement file, respectively.

IV. Experiments

In this section, we investigate the effectiveness of our proposed LFM framework. We explore the performance of LFM with searchable architectures in image classifications. Further, we designed both class imbalance and noisy label settings experiments on standard CIFAR-10 and CIFAR-100 benchmarks with fixed human-designed networks to verify the effectiveness of our method when applied to DR.

A. Differentiable NAS

In this section, we applied our method to differentiable NAS for image classification tasks. Following DARTS [5], the approaches consist of architecture search and evaluation stages, where the optimal cell obtained from the search stage is stacked several times into a more extensive composite network. We then train the resultant composite network from scratch in the evaluation stage.

1) Datasets: The experiments were performed on three popular NAS datasets, namely CIFAR-10, CIFAR-100, and ImageNet. We conducted architecture searching on CIFAR-10 and CIFAR-100 datasets. For ImageNet dataset, we conduct experiments with the model architectures searched on CIFAR-10 and CIFAR-100 datasets. Both CIFAR-10 and CIFAR-100 datasets contain 60K images, with each class having the same number of images. We split each dataset into a training set with 25K images, a validation set with 25K images, and a test set with 10K images. During the architecture search, the training set is notated as D^{tr}, and the validation set is notated as D^{val}. During architecture evaluation, the learned network is trained on the combination of D^{tr} and D^{val}. Further, ImageNet contains 1.2M training images and 50K test images with 1000 objective classes.

2) Experimental Settings: Our framework is orthogonal to existing NAS approaches and can be applied to any differentiable NAS method. In our experiments, we applied LFM to DARTS [5] and PDARTS [30]. The search spaces of these methods are kept the same as their backbone. For the encoder V, ResNets that pretrained on Imagenet were used. Each LFM experiment was repeated three times with different random seeds. The mean and standard deviation of classification errors obtained from the three runs are reported.

During the architecture search for CIFAR-10 and CIFAR-100, the architectures of W_1 and W_2 are a stack of 8 cells. Each

TABLE I
TEST ERROR ON CIFAR-100

Method	Error (%)
*ResNet [41]	22.10
*DenseNet [43]	17.18
*PNAS [44]	19.53
*ENAS [25]	19.43
*AmoebaNet [27]	18.93
*GDAS [45]	18.38
*R-DARTS [46]	18.01±0.26
*DARTS$^-$ [33]	17.51±0.25
*DARTS$^+$ [32]	17.11±0.43
*DropNAS [47]	16.39
Random search	21.92±0.34
Random sampling	21.37±0.48
*DARTS-2nd [5]	20.58±0.44
LFM-DARTS-2nd-R18 (ours)	**17.65±0.45**
*PDARTS [30]	17.49
LFM-PDARTS-R18 (ours)	**16.44±0.11**
LFM-PDARTS-R34 (ours)	**15.69±0.15**

Note: LFM-PDARTS-R18 denotes applying LFM to PDARTS with ResNet-18 as encoder; such format apply to other results. Results marked with * are from Skillearn [42].

TABLE II
TEST ERROR ON CIFAR-10

Method	Error (%)
*DenseNet [43]	3.46
*HierEvol [48]	3.75±0.12
*PNAS [44]	3.41±0.09
*ENAS [25]	2.89±0.13
*NASNet-A [26]	2.65±0.05
*AmoebaNet-B [27]	2.55±0.05
*R-DARTS [46]	2.95±0.21
*GTN [49]	2.92±0.06
*BayesNAS [50]	2.81±0.04
*MergeNAS [51]	2.73±0.02
*NoisyDARTS [52]	2.70±0.23
*ASAP [53]	2.68±0.11
*SDARTS [54]	2.61±0.02
*DropNAS [47]	2.58±0.14
*DrNAS [55]	2.54±0.03
Random search	3.07±0.17
Random sampling	2.75±0.09
*DARTS-2nd [5]	2.76±0.09
LFM-DARTS-2nd-R18 (ours)	**2.70±0.06**
*PDARTS [30]	2.50
LFM-PDARTS-R18 (ours)	**2.46±0.04**

Note: Results marked with * are obtained from Skillearn [42]. The other notations are the same as described in Table I.

cell consists of 7 nodes. We set the initial channel number to 16. For the architecture of the encoder model, we experimented with ResNet-18 and ResNet-34 [41]. The search algorithm was based on stochastic gradient descent (SGD) optimizer, and the hyperparameters of epochs, initial learning rate, and momentum follow the original implementation of the respective DARTS [5] and PDARTS [30]. During architecture evaluation for CIFAR-10 and CIFAR-100, a more extensive network of each category-specific model is formed by stacking 20 copies of the searched cell.

The LFM method is used to learn the architecture B, while the weights W_1, W_2, V, and r learned during the LFM search are discarded during the architecture evaluation. All the architecture evaluations are run using the same standardized setup. This results in a fair comparison between architectures learned from different methods. All the models during evaluation have the same number of parameters and hyper-parameters, such as epochs, learning rate, and batch size. More detailed experimental settings can be found in Appendix C.

3) Results and Analysis: The experiments are performed on three popular NAS datasets, namely CIFAR-10, CIFAR-100 [64], and ImageNet [65]. More detailed results can be found in the Appendix, including information about parameters, search cost, more compared methods, and other ablation studies. The results of the classification error (%) of different NAS methods on CIFAR-100 are shown in Table I. We make the following observations from this table.

1) When LFM is applied to DARTS-2nd (second-order approximation) and PDARTS, significant reductions in classification errors are observed. For example, when LFM is applied to DARTS-2nd, the error rate decreases from 20.58 to 17.70%. Similarly, in PDARTS, the error rate decreases from 17.49 to 16.44% (when using ResNet-18 as encoder V). These results demonstrate the effectiveness of our method in improving the performance of architecture search. In baseline NAS approaches, all training examples are assigned equal weights. In contrast, our method dynamically reweights training examples at each stage based on the current learning capability of the model, assigning more weight to samples that are challenging to learn. This approach reflects a more realistic learning scenario.
2) LFM-PDARTS-R34 outperforms LFM-PDARTS-R18 by 0.75%, where the former uses ResNet-34 as the image encoder, while the latter uses ResNet-18. ResNet-34 is a deeper and more robust data encoder than ResNet-18. This shows that mapping the validation examples to similar training examples is a core component contributing to the effectiveness of our proposed LFM method.
3) LFM-PDARTS-R34 achieves the best performance among all methods, which demonstrates the effectiveness of applying LFM to differentiable NAS methods and improving their performance.

The results of the classification error (%) of different NAS methods on CIFAR-10 are shown in Table II. As can be seen, LFM applied to DARTS-2nd and PDARTS reduces the errors of these baselines by roughly 0.05%. This further demonstrates the effectiveness of our method.

The results of the classification error (%) for top-1 and top-5 of different NAS methods on ImageNet are presented in Table III. In the methods LFM-DARTS-2nd-CIFAR10 and LFM-PDARTS-CIFAR10, the architectures searched on CIFAR-10 are evaluated on ImageNet, while in LFM-PDARTS-CIFAR100, the architecture searched on CIFAR-100

TABLE III
TEST ERRORS ON IMAGENET

Method	Top-1 Error (%)	Top-5 Error (%)
*Inception-v1 [56]	30.2	10.1
*MobileNet [57]	29.4	10.5
*ShuffleNet 2× (v1) [58]	26.4	10.2
*ShuffleNet 2× (v2) [59]	25.1	7.6
*NASNet-A [26]	26.0	8.4
*PNAS [44]	25.8	8.1
*MnasNet-92 [60]	25.2	8.0
*AmoebaNet-C [27]	24.3	7.6
*SNAS-CIFAR10 [29]	27.3	9.2
*PARSEC-CIFAR10 [61]	26.0	8.4
*DSNAS-ImageNet [62]	25.7	8.1
*SDARTS-ADV-CIFAR10 [54]	25.2	7.8
*FairDARTS-ImageNet [63]	24.4	7.4
*DrNAS-ImageNet [55]	24.2	7.3
*ProxylessNAS-ImageNet [28]	24.9	7.5
*GDAS-CIFAR10 [45]	26.0	8.5
*DARTS2nd-CIFAR10 [5]	26.7	8.7
LFM-DARTS-2nd-CIFAR10 (ours)	**25.1**	**7.6**
*PDARTS (CIFAR10) [30]	24.4	7.4
LFM-PDARTS-CIFAR10 (ours)	**24.1**	**6.8**
*PDARTS (CIFAR100) [30]	24.7	7.5
LFM-PDARTS-CIFAR100 (ours)	**24.1**	**6.7**

Note: The first three blocks represent: 1) manually designed networks, 2) non-gradient-based NAS methods, and 3) gradient-based NAS methods. The rest of the notations follow the Table I.

is evaluated on ImageNet. Specifically, LFM-DARTS-2nd-CIFAR10 outperforms the baseline DARTS-2nd-CIFAR10 by 1.6%, whereas LFM-PDARTS-CIFAR100 outperforms its corresponding baseline by 0.6%, and LFM-PDARTS-CIFAR10 by 0.3%. These results demonstrate that the LFM methods consistently outperform their corresponding baselines, highlighting the effectiveness of our approach.

4) Ablation Studies on LFM-NAS:

Ablation 1: Our method introduces three important components to the reweighting parameter a_i: x, u, and z. In this study, we demonstrate the effect of ablating each component. We conducted experiments on CIFAR-100 using ResNet-18 as the encoder, with other details matching the base experiments described in earlier sections. The performances of the ablated models are shown in Fig. 4. These results highlight the effectiveness and necessity of incorporating measurements of mistakes (u) on validation examples, calculating visual similarity (x), and evaluating label similarity (z) between training and validation examples within our method.

Ablation 2: For visual similarity, we utilize dot product attention, which has demonstrated effectiveness in various applications and is straightforward to implement. To showcase its efficacy within LFM, we compared it with other metrics such as cosine similarity and L2 distance. The experimental setup closely follows the base experiments described in earlier sections. The results, depicted in Fig. 5, show that dot product attention used in our framework performs better than the other two metrics.

Ablation 3: We experiment to evaluate the necessity of the second network. A degenerated way of learning a single set of weights is to discard the end-to-end tri-level framework and

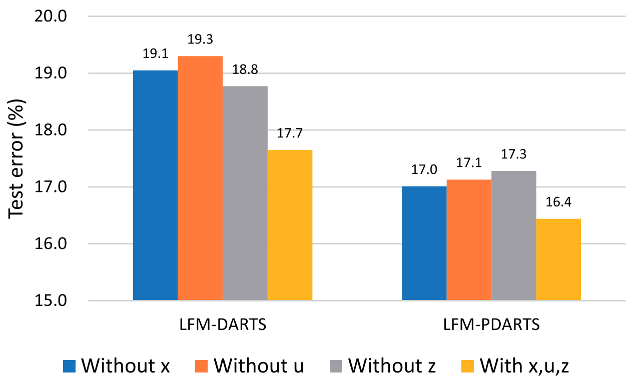

Fig. 4. Ablation on components of a_i. The left bar column shows the comparison of ablated models 1) without x, 2) without u, 3) without z, and 4) the full models.

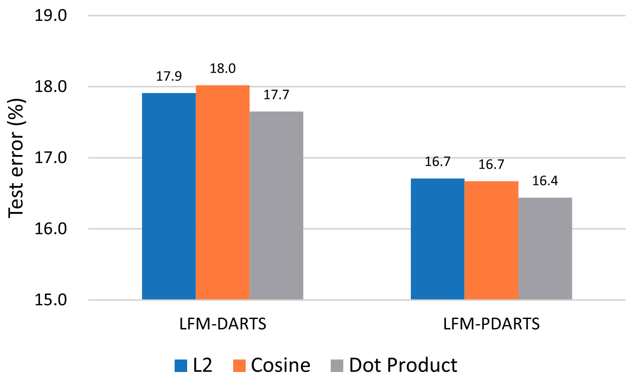

Fig. 5. Comparison of different visual similarity metrics.

TABLE IV
TEST ERROR COMPARISON OF SINGLE NETWORK (SN) AND TWO NETWORKS (TN) ON CIFAR-100. DARTS DEFAULTS THE DARTS OPTIMIZED IN THE SECOND ORDER

Method	Test error (%)
SN+DARTS	19.28±0.31
TN+DARTS	17.65±0.45
SN+PDARTS	17.37±0.23
TN+PDARTS	16.44±0.11

learn the weights in two separate stages: train the weights, use them to reweight training examples, and then retrain the weights on reweighted examples. However, in this case, the method is no longer an end-to-end framework, which leads to inferior performance. We experimented with this degenerated method and its performance is worse than our tri-level end-to-end framework. The results are shown in Table IV.

Ablation 4: We conduct experiments to evaluate the impact of the image encoder used in the second stage, where larger models (ResNet-101 and ResNet-152) are applied to compute visual similarity. Specifically, we aim to assess how the capacity of the image encoder influences the overall model's ability to extract meaningful representations. The model architecture is searched

TABLE V
TEST ERROR ON CIFAR-100 AND IMAGENET TEST SETS WITH DIFFERENT SIZE OF IMAGE ENCODERS

Dataset	Encoder	Test error (%)
CIFAR-100	LFM-PDARTS-R18	16.44
	LFM-PDARTS-R101	15.45
	LFM-PDARTS-R152	14.88
ImageNet	LFM-PDARTS-R18	24.10
	LFM-PDARTS-R101	23.52
	LFM-PDARTS-R152	23.02

by PDARTS on the CIFAR-100 dataset, and the discovered architecture is then applied to both the CIFAR-100 and ImageNet datasets. As shown in Table V, increasing the size of the image encoder enhances model performance. A more powerful encoder not only enhances feature extraction but also improves the accuracy of identifying training examples that significantly impact model capability. This precision in example selection ensures that the model learns from instances that maximize its generalization, thereby reducing overfitting and improving robustness across datasets.

B. DR Under Class Imbalance

1) Datasets: We apply LFM-DR for image classification, specifically using long-tailed datasets based on CIFAR-10 and CIFAR-100 to evaluate the performance of LFM-DR. Following the approach described in [66] for creating long-tailed datasets, we reduce the number of training samples per class according to an exponential function $n = n_i\mu^i$, where i is the class index, n_i is the original number of samples in class i, and $\mu \in (0, 1)$. In this experiment, the imbalance factor represents the degree of data imbalance and is defined as the number of training samples in the largest class divided by the number in the smallest class.

2) Experimental Settings: Same as the article [12], we applied LFM to ResNet-32 [41]. About the settings specified in the LFM-DR, the ResNet-18 is used as the data encoder when calculating the weight a_i in stage 2. Finally, the learning rate of ResNet-32 is divided by 10 after 80 and 90 epochs (for a total of 100 epochs). The compared methods include 1) Base Model, which uses a softmax cross-entropy loss to train ResNet-32 on the training set; 2) Focal loss [15], Class-Balanced [66], Learning to reweight (L2RW) [67], and Meta-Weight-Net [12] represent the state-of-the-art of the DR techniques. And LFM-DR represents our means of applying LFM to DR. More detailed experimental settings can be found in the Appendix.

3) Results and Analysis: The classification accuracy results of LFM-DR applied to ResNet-32 on long-tailed datasets of CIFAR-10 are shown in Table VI. As shown in the tables that

1) After applying our LFM framework to the base model, the classification accuracy is improved significantly across all imbalance factors. For example, on an imbalance factor equal to 200, the test accuracy improves from 65.68 to 73.72%, by more than 8%. Our method significantly enhances the test accuracy of the model, demonstrating the effectiveness of LFM in addressing class imbalance challenges within the training set. By integrating LFM into the base model, each training sample autonomously learns a specific weight, enabling the model to focus more on challenging examples. This adaptive weighting mechanism equips the learner with enhanced capabilities to tackle complex tasks effectively. The observed improvement in test accuracy underscores LFM's ability to optimize model performance amidst class imbalance, showcasing its potential to bolster machine learning systems for real-world applications.
2) For imbalance factors between 200 and 10, our method achieves the best performance among all baselines, demonstrating the effectiveness of applying LFM to the base model and improving its performance. It also shows the superiority of our method over other comparison methods.
3) When the imbalance factor is 1, which means all the classes have the same number of training samples, after applying our method, the model attains a comparable performance with the base model, showing its robustness in different situations.
4) For a long-tailed CIFAR-10 dataset with imbalance factor 200, there are only 24 images in the training set of the class with the least amount of data (class 9). While there are 4990 images in the training set of the class with the most amount of data (class 0). The great bias of training set is the reason leading to an imbalance results. However, as shown in Fig. 6, comparing to the base model, after applying LFM, the problem about imbalance prediction results on test set have been greatly mitigated. Specifically, we improve the performance of the category with the least amount of training data from 14.4 to 47.3%.

The classification accuracy results of ResNet-32 that are trained in a standard way and trained in our framework on long-tailed datasets of CIFAR-100 are also shown in Table VI. It can be seen that our method, LFM-DR, has shown better performance in any case, from the imbalance factor equal from 200 to 1. The results on long-tailed CIFAR-100 further demonstrate the effectiveness of our strategy that, after applying our method, the robustness of the model can be improved significantly.

C. DR Under Corrupted Labels

1) Datasets: We evaluated our method on datasets with corrupted labels in the training set, and two different types of noises were applied to the original samples. One is the uniform noise, which means the label of each training sample can uniformly change to another random class with probability p following the instruction in [72] and is the most common phenomenon in the literature. Another type is the flip noise: the label of each sample is independently flipped to similar classes with total probability p [12]. Here, two specific classes are selected as similar classes. These two types of noise are employed for CIFAR-10 and CIFAR-100 [73].

2) Experimental Settings: We use the Wide ResNet-28-10 (WRN-28-10) [74] for uniform noise, and ResNet-32 [41] for flip noise as their base models. We use different classifier

TABLE VI
TEST ACCURACY (%) OF RESNET-32 ON LONG-TAILED CIFAR-10 AND CIFAR-100

Dataset name	Long-tailed CIFAR-10						Long-tailed CIFAR-100					
Imbalance Factor	200	100	50	20	10	1	200	100	50	20	10	1
*Base Model	65.68	70.36	74.81	82.23	86.39	92.89	34.84	38.32	43.85	51.44	55.71	70.50
*Focal Loss [15]	65.29	70.38	76.71	82.76	86.66	**93.03**	35.62	38.41	44.32	51.95	55.78	***70.52***
*Class-Balance [66]	68.89	74.57	79.27	84.36	87.49	92.89	36.23	39.60	45.32	52.59	57.99	70.50
*L2RW [67]	66.51	74.16	78.93	82.12	85.19	89.25	33.38	40.23	44.44	51.64	53.73	64.11
*Meta-Net [12]	***68.91***	***75.21***	***80.06***	***84.94***	***87.84***	92.66	***37.91***	***42.09***	***46.74***	***54.37***	***58.46***	70.37
LFM-DR	**73.72**	**78.89**	**83.46**	**86.98**	**89.24**	***92.92***	**39.08**	**43.27**	**47.42**	**56.10**	**59.70**	**70.58**

Note: The best and the second best results are highlighted in bold and ***italic bold***, respectively. LFM-DR refers to applying LFM to the base model. Results marked with * are obtained from [12].

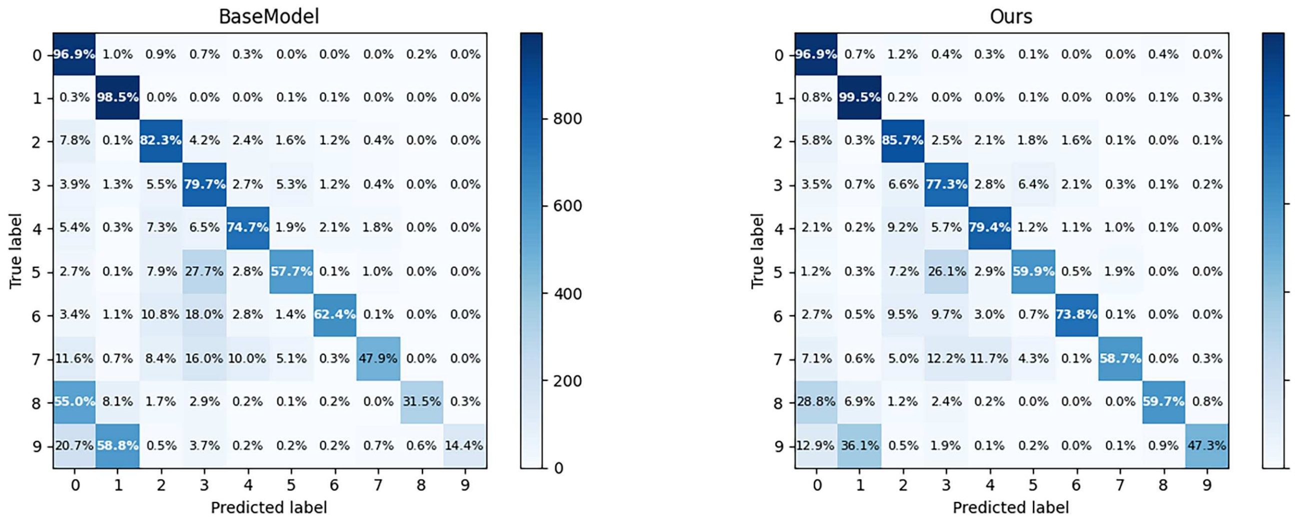

Fig. 6. Confusion matrices for the BaseModel and Ours on long-tailed CIFAR-10 dataset with imbalance factor 200.

TABLE VII
TEST ACCURACY (%) COMPARISON ON CIFAR-10 AND CIFAR-100 OF RESNET-32 WITH VARYING NOISE RATES UNDER FLIP NOISE

Dataset name	Corrupted CIFAR-10			Corrupted CIFAR-100		
Noise Rate	0%	20%	40%	0%	20%	40%
*Base Model	***92.89±0.32***	76.83±2.30	70.77±2.31	70.50±0.12	50.86±0.27	43.01±1.16
*Reed-Hard [68]	92.31±0.25	88.28±0.36	81.06±0.76	69.02±0.32	60.27±0.76	50.40±1.01
*Self-paced [17]	88.52±0.21	87.03±0.34	81.63±0.52	67.55±0.27	63.63±0.30	53.51±0.53
*Focal Loss [15]	93.03±0.16	86.45±0.19	80.45±0.97	70.02±0.53	61.87±0.30	54.13±0.40
*Co-teaching [69]	89.87±0.10	82.83±0.85	75.41±0.21	63.31±0.05	54.13±0.55	44.85±0.81
*D2L [70]	92.02±0.14	87.66±0.40	83.89±0.46	68.11±0.26	63.48±0.53	51.83±0.33
*Fine-tining	**93.23±0.23**	82.47±3.64	74.07±1.65	***70.72±0.22***	56.98±0.50	46.37±0.25
*MentorNet [18]	92.13±0.30	86.36±0.31	81.76±0.28	70.24±0.21	61.97±0.47	52.66±0.56
*L2RW [67]	89.25±0.37	87.86±0.36	85.66±0.51	64.11±1.09	57.47±1.16	50.98±1.55
*GLC [71]	91.02±0.20	89.68±0.33	***88.92±0.24***	65.42±0.23	63.07±0.53	62.22±0.62
*Meta-Net [12]	92.04±0.15	***90.33±0.61***	87.54±0.23	70.11±0.33	***64.22±0.28***	***58.64±0.47***
LFM-DR	92.85±0.11	**91.28±0.58**	**89.06±0.17**	**70.93±0.44**	**65.88±0.27**	**60.73±0.51**

Note: The baselines include base model, reed-hard, self-paced, focal loss, co-teaching, d2l, fine-tining, mentronet, l2rw, glc, and meta-net.

networks as base models that aim to show that networks with different architectures can adapt our strategy to make an improvement. We use the same hyperparameter settings as in class imbalance experiments. The results of each competing method are an average of 5 trials.

3) Results and Analysis: The classification accuracy results of LFM-DR applied to ResNet-32 on datasets with flip noise of CIFAR-10 and CIFAR-100 are shown in Table VII. It can be seen from these tables that:

1) After adopting our strategy, the performance of the base model has shown significant improvement, particularly evident under flip noise scenarios. The integration of LFM with DR (LFM-DR) resulted in notable enhancements across most datasets and noise rates, with

TABLE VIII
TEST ACCURACY (%) COMPARISON ON CIFAR-10 AND CIFAR-100 OF WRN-28-10 WITH VARYING NOISE RATES UNDER UNIFORM NOISE. OTHERS ARE THE SAME AS TABLE VII

Dataset name	Corrupted CIFAR-10			Corrupted CIFAR-100		
Noise Rate	0%	20%	40%	0%	20%	40%
*Base Model	95.60±0.22	68.07±1.23	53.12±3.03	79.95±1.26	51.11±0.42	30.92±0.33
*Reed-Hard [68]	94.38±0.14	81.26±0.51	73.53±1.54	64.45±1.02	51.27±1.18	26.95±0.98
*Self-paced [17]	90.81±0.34	86.41±0.29	53.10±1.78	59.79±0.46	46.31±2.45	19.08±0.57
*Focal Loss [15]	**95.70±0.15**	75.96±1.31	51.87±1.19	**81.04±0.24**	51.19±0.46	27.70±3.77
*Co-teaching [69]	88.67±0.25	74.81±0.34	73.06±0.25	61.80±0.25	46.20±0.15	35.67±1.25
*D2L [70]	94.64±0.33	85.60±0.13	68.02±0.41	66.17±1.42	52.10±0.97	41.11±0.30
*Fine-tining	***95.65±0.15***	80.47±0.25	78.75±2.40	80.88±0.21	52.49±0.74	38.16±0.38
*MentorNet [18]	94.35±0.42	87.33±0.22	82.80±1.35	73.26±1.23	61.39±3.99	36.87±1.47
*L2RW [67]	92.38±0.10	86.92±0.19	82.24±0.36	72.99±0.58	60.79±0.91	48.15±0.34
*GLC [71]	94.30±0.10	88.28±0.03	83.49±0.24	73.75±0.51	61.31±0.22	50.81±1.00
*Meta-Net [12]	94.52±0.25	***89.27±0.28***	***84.07±0.33***	78.76±0.24	***67.73±0.26***	***58.75±0.11***
LFM-DR	95.56±0.11	**89.66±0.58**	**84.78±0.17**	*80.64±0.44*	**68.71±0.27**	**60.74±0.51**

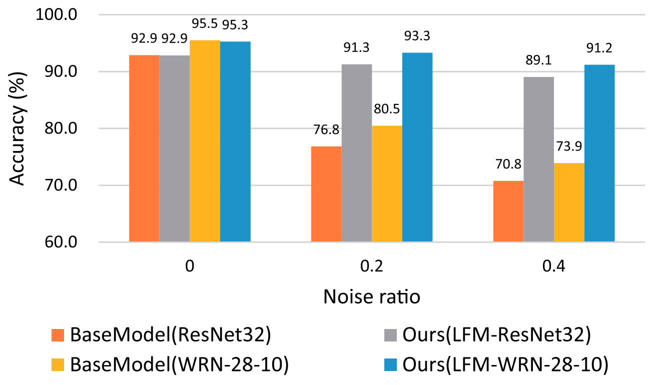

Fig. 7. Performance of comparison for different classifier networks (WRN-28-10 and ResNet32) under CIFAR-10 flip noise.

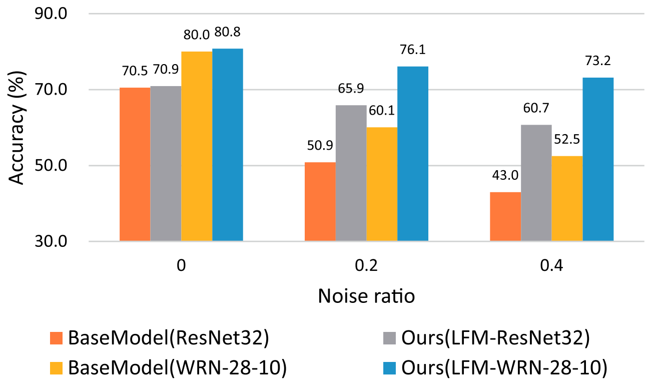

Fig. 8. Performance of comparison for different classifier networks (WRN-28-10 and ResNet32) under CIFAR-100 flip noise.

the exception of the corrupted CIFAR-10 dataset at 0% noise rate. Notably, compared to the base model, LFM-DR improved accuracy from 70.77 to 89.06% on corrupted CIFAR-10 and from 43.01 to 60.73% on CIFAR-100 at a 40% noise rate. These results highlight the effectiveness of our method in empowering the base model to effectively manage label noise challenges. LFM-DR's ability to substantially boost accuracy under noisy conditions underscores its practical utility in enhancing model robustness and reliability in real-world scenarios.

2) Compared to other mainstream methods, the robustness of our approach can be seen in almost all situations. Our method outperforms the method ranked second by more than 2% for a 40% noise rate on the corrupted CIFAR-100 dataset. In other cases, our method also obtains a relatively higher classification accuracy. This further demonstrates the effectiveness of our approach.

3) Not only for ResNet-32 but also for WRN-28-10 under uniform noise, whose results are shown in Table VIII, our method shows significant performance. After applying LFM, the base models became more robust means that they can handle more difficult tasks in various cases. This demonstrates the applicability of our method to different machine learning methods.

To further demonstrate the effectiveness of our method, we arranged experiments to compare the performance of WRN-28-10 and ResNet32 under flip noise and the improvements after implying our approach to the base models. As shown in Fig. 7 and Fig. 8, we can observe that, after applying our method to the base model, the performance all increased significantly, and performance gains for our method and base model between two different networks take almost the same value. The results imply that the performance improvement of LFM is available for other network architectures.

D. DR on Real Dataset

To further verify the effectiveness of LFM, we conduct experiments on the ANIMAL-10 dataset [75]. ANIMAL-10 dataset is a noisy dataset with human-labeled images. ANIMAL-10 contains 5 pairs of confusing animals with a total of 55 000 images, of which 50 000 are training samples, and 5000 are test samples. The five pairs are as follows: (cat, lynx), (hamster, guinea pig), (wolf, coyote), (jaguar, cheetah), (chimpanzee, orangutan), where two animals in each pair look very similar, as shown in Fig. 9. The overall noise rate of ANIMAL-10 is about 8%.

In this experiment, we use Vgg19-BN [76] as the base model and apply our LFM framework to it. The base model was trained

Fig. 9. Image samples from ANIMAL-10.

TABLE IX
CLASSIFICATION ACCURACY (%) ON THE ANIMAL-10 TEST SET

Method	Accuracy	Method	Accuracy
NCT [77]	84.1	PLC [78]	83.4
SELFIE [75]	81.8	CE Dropout [77]	81.3
Vgg19-BN [76]	79.4	**LFM**	**86.4**

using SGD with a momentum of 0.9, a weight decay of $5e-4$, and its initial learning rate is $1e-1$. The settings of the LFM hyper-parameters are the same as in the last experiment.

The results are summarized in Table IX. All the methods of comparison used Vgg19-BN as the baseline network. To summarize, our method achieves better performance in relation to the current state-of-the-art.

V. CONCLUSION

In this article, we proposed a novel optimization framework, LFM, which is inspired by the practical human learning skill of learning from the mistakes corresponding to the topics the learner learns currently. To formalize the idea of LFM, we design a multilevel optimization framework to solve the problem. Compared with other prevailing methods, LFM can develop the weighting function without prior knowledge. It can modulate the weights of different training samples automatically for the degree of difficulty of its task. In our method, three metrics have been used to measure the extent of mistakes the learner made. Our experiments show the effectiveness of the proposed method in generic data bias cases.

Our method requires the use of two learners who have similar learning capabilities so that one can learn from the mistakes of others. This increases the memory requirements and makes the learning slow compared to the traditional approaches. In future work, we explore reducing memory cost during architecture search by parameter-sharing between the three models W_1, W_2, and V. For W_1 and W_2, we let them share the same convolutional layers but have different classification heads. For V, we replace ResNet-18 with W_1. As shown in Table X that via parameter sharing (PS), the memory and computation costs of our method are reduced to a level similar to traditional DARTS and PDARTS, while our method still achieves significantly lower test errors than DARTS and PDARTS. A future work direction is to improve memory usage while keeping the full performance of the LFM method. Another direction is to extend the applicability of LFM to other meta-learning tasks such as tasks like semantic segmentation. LFM can also be extended to language modeling tasks as well. Further, recent theoretical work on phase transitions in time-varying complex networks (TVCNs) by Znaidi et al. [79] highlights how local changes can trigger abrupt shifts in global properties. This mirrors the behavior seen in NAS, where small architecture adjustments can lead to significant performance changes. The Forman–Ricci curvature framework used in TVCNs could also be applied to NAS to identify critical performance shifts, potentially enhancing the LFM framework by targeting such transition points.

TABLE X
TEST ERROR (%), MEMORY COST (MIB) AND COMPUTATION COST (GPU DAYS) OF DIFFERENT MODELS ON CIFAR-100

Method	Test error (%)	Memory (MiB)	Cost (days)
LFM+DARTS, no PS	17.65±0.45	23 702	5.4
LFM+DARTS+PS	18.77±0.31	12 138	1.6
DARTS	20.58±0.44	11 053	1.5
LFM+PDARTS, no PS	16.44±0.11	20 744	2.0
LFM+PDARTS+PS	16.83±0.08	10 582	0.3
PDARTS	17.49	9659	0.3

REFERENCES

[1] S. Dempe et al., "Bilevel optimization: Theory, algorithms, applications and a bibliography," in *Bilevel Optimization. Springer Optimization and Its Applications*, Cham: Springer, 2020, pp. 581–672.
[2] R. Liu, J. Gao, J. Zhang, D. Meng, and Z. Lin, "Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 12, pp. 10045–10067, Dec. 2022.
[3] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.
[4] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. Int. Conf. Learn. Represent.*, vol. 7, 2018, pp. 5008–5020.
[5] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2018.
[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.
[7] B. Zadrozny, "Learning and evaluating classifiers under sample selection bias," in *Proc. 21st Int. Conf. Mach. Learn.*, 2004, p. 114.
[8] C. Elkan, "The foundations of cost-sensitive learning," in *Proc. Int. Joint Conf. Artif. Intell.*, vol. 17, no. 1, 2001, pp. 973–978.
[9] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1997–2017, 2019.
[10] J. Ni, Y. Chen, Y. Chen, J. Zhu, D. Ali, and W. Cao, "A survey on theories and applications for self-driving cars based on deep learning methods," *Appl. Sci.*, vol. 10, no. 8, p. 2749, 2020.
[11] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Brief. Bioinf.*, vol. 19, no. 6, pp. 1236–1246, 2018.
[12] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng, "Meta-weight-net: Learning an explicit mapping for sample weighting," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 1919–1930.
[13] Z. Shen, P. Cui, T. Zhang, and K. Kunag, "Stable learning via sample reweighting," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 04, 2020, pp. 5692–5699.
[14] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognit.*, vol. 40, no. 12, pp. 3358–3378, 2007.
[15] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vision*, 2017, pp. 2980–2988.
[16] H.-S. Chang, E. Learned-Miller, and A. McCallum, "Active bias: Training more accurate neural networks by emphasizing high variance samples," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, pp. 1002–1012, 2017.

[17] M. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 23, pp. 1189–1197, 2010.

[18] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, "Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels," in *Proc. Int. Conf. Mach. Learn.*. PMLR, 2018, pp. 2304–2313.

[19] Z. Zhang and M. R. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Proc. 32nd Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2018.

[20] M. Feurer, J. Springenberg, and F. Hutter, "Initializing Bayesian hyperparameter optimization via meta-learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 29, no. 1, 2015.

[21] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[22] Y. Hu, X. Wu, and R. He, "TF-NAS: Rethinking three search freedoms of latency-constrained differentiable neural architecture search," in *Proc. Eur. Conf. Comput. Vision*, 2020, pp. 123–139.

[23] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1568–1577.

[24] M. Shu, C. Liu, W. Qiu, and A. Yuille, "Identifying model weakness with adversarial examiner," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 07, 2020, pp. 11 998–12 006.

[25] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.

[26] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 8697–8710.

[27] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 01, 2019, pp. 4780–4789.

[28] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Represent.*, 2018.

[29] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2018.

[30] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE/CVF Int. Conf. Comput. Vision*, 2019, pp. 1294–1303.

[31] Y. Xu et al., "PC-DARTS: Partial channel connections for memory-efficient architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[32] H. Liang et al., "Darts+: Improved differentiable architecture search with early stopping," 2019, *arXiv:1909.06035*.

[33] X. Chu, X. Wang, B. Zhang, S. Lu, X. Wei, and J. Yan, "Darts-: Robustly stepping out of performance collapse without indicators," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[34] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, "Cost-sensitive learning of deep feature representations from imbalanced data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3573–3587, Aug. 2017.

[35] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.

[36] T. Malisiewicz, A. Gupta, and A. A. Efros, "Ensemble of exemplar-svms for object detection and beyond," in *Proc. 2011 Int. Conf. Comput. Vision*. IEEE, 2011, pp. 89–96.

[37] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intell. Syst. their Appl.*, vol. 13, no. 4, pp. 18–28, Jul./Aug. 1998.

[38] S. Hochreiter, "Long short-term memory," *Neural Comput.,* vol. 9, no. 9, pp. 1735–1780, 1997.

[39] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 41–48.

[40] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. 2015 Conf. Empir. Methods Natural Lang. Process.*, 2015, pp. 1412–1421.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 770–778.

[42] P. Xie, X. Du, and H. Ban, "Skillearn: Machine learning inspired by humans' learning skills," 2020, *arXiv:2012.04863*.

[43] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vision Recognit.*, 2017, pp. 4700–4708.

[44] C. Liu et al., "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vision*, 2018, pp. 19–35.

[45] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 1761–1770.

[46] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, "Understanding and robustifying differentiable architecture search," in *Int. Conf. Learn. Represent.*, 2019.

[47] W. Hong et al., "Dropnas: grouped operation dropout for differentiable architecture search," in *Proc. 29th Int. Conf. Int. Joint Conf. Artif. Intell.*, 2021, pp. 2326–2332.

[48] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," 2018, *arXiv:1711.00436*.

[49] F. P. Such, A. Rawal, J. Lehman, K. Stanley, and J. Clune, "Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9206–9216.

[50] H. Zhou, M. Yang, J. Wang, and W. Pan, "Bayesnas: A Bayesian approach for neural architecture search," in *Int. Conf. Mach. Learn.*, 2019, pp. 7603–7613.

[51] X. Wang, C. Xue, J. Yan, X. Yang, Y. Hu, and K. Sun, "Mergenas: Merge operations into one for differentiable architecture search," in *Proc. 29th Int. Conf. Int. Joint Conferences Artif. Intell.*, 2021, pp. 3065–3072.

[52] X. Chu and B. Zhang, "Noisy differentiable architecture search," 2020, *arXiv:2005.03566*.

[53] A. Noy et al., "Asap: Architecture search, anneal and prune," in *Proc. Int. Conf. Artif. Intell. Stat.*, 2020, pp. 493–503.

[54] X. Chen and C.-J. Hsieh, "Stabilizing differentiable architecture search via perturbation-based regularization," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 1554–1565.

[55] X. Chen, R. Wang, M. Cheng, X. Tang, and C.-J. Hsieh, "DrNAS: Dirichlet neural architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[56] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 1–9.

[57] A. G. Howard et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[58] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. vision pattern Recognit.*, 2018, pp. 6848–6856.

[59] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vision*, 2018, pp. 116–131.

[60] M. Tan et al., "Mnasnet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 2820–2828.

[61] F. P. Casale, J. Gordon, and N. Fusi, "Probabilistic neural architecture search," 2019, *arXiv:1902.05116*.

[62] S. Hu, S. Xie, H. Zheng, C. Liu, J. Shi, X. Liu, and D. Lin, "DSNAS: Direct neural architecture search without parameter retraining," in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit.*, 2020, pp. 12084–12092.

[63] X. Chu, T. Zhou, B. Zhang, and J. Li, "Fair darts: Eliminating unfair advantages in differentiable architecture search," in *Eur. Conf. Comput. Vision*, 2020, pp. 465–480.

[64] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," *Citeseer*, p. 60, 2009.

[65] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. 2009 IEEE Conf. Comput. Vision Pattern Recognit.*, 2009, pp. 248–255.

[66] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-balanced loss based on effective number of samples," in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 9268–9277.

[67] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," in *Int. Conf. Mach. Learn.*, 2018, pp. 4334–4343.

[68] S. E. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich, "Training deep neural networks on noisy labels with bootstrapping," in *Proc. ICLR (Workshop)*, 2015.

[69] B. Han et al., “Co-teaching: Robust training of deep neural networks with extremely noisy labels,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.

[70] X. Ma et al., “Dimensionality-driven learning with noisy labels,” in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3355–3364.

[71] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel, “Using trusted data to train deep networks on labels corrupted by severe noise,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.

[72] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, “Understanding deep learning (still) requires rethinking generalization,” *Commun. ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[73] A. Krizhevsky et al., “Learning multiple layers of features from tiny images,” *Citeseer*, 2009.

[74] S. Zagoruyko and N. Komodakis, “Wide residual networks,” in *Brit. Mach. Vision Conf. 2016*, 2016.

[75] H. Song, M. Kim, and J.-G. Lee, “Selfie: Refurbishing unclean samples for robust deep learning,” in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5907–5915.

[76] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, *arXiv:1409.1556*.

[77] Y. Chen, X. Shen, S. X. Hu, and J. A. Suykens, “Boosting co-teaching with compression regularization for label noise,” in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit.*, 2021, pp. 2688–2692.

[78] Y. Zhang, S. Zheng, P. Wu, M. Goswami, and C. Chen, “Learning with feature-dependent label noise: A progressive approach,” in *Int. Conf. Learn. Represent.*, 2020.

[79] M. R. Znaidi, J. Sia, S. Ronquist, I. Rajapakse, E. Jonckheere, and P. Bogdan, “A unified approach of detecting phase transition in time-varying complex networks,” *Sci. Reports*, vol. 13, no. 1, p. 17948, 2023.