Deep Neural Network Parameter Selection via Dataset Similarity under Meta-Learning Framework

Liping Deng, Maziar Raissi, MingQing Xiao

Abstract—Optimizing the performance of deep neural networks (DNNs) remains a significant challenge due to the sensitivity of models to both hyperparameter selection and weight initialization. Existing approaches typically address these two factors independently, which often leads to limiting adaptability and overall effectiveness. In this paper, we present a novel meta-learning framework that jointly recommends hyperparameters and initial weights by leveraging dataset similarity. Our method begins by extracting meta-features from a collection of historical datasets. For a given query dataset, similarity is computed based on distances in the meta-feature space, and the most similar historical datasets are used to recommend the underlying parameter configurations. To capture the diverse characteristics of image datasets, we introduce two complementary types of meta-features. The first, referred to as shallow or visible meta-features, comprises five groups of statistical measures that summarize color and texture information. The second, termed deep or invisible meta-features, consists of 512 descriptors extracted from a convolutional neural network pre-trained on ImageNet. We evaluated our framework in 105 real-world image classification tasks, using 75 datasets for historical modeling and 30 for querying. Experimental results with both vision transformers and convolutional neural networks demonstrate that our approach consistently outperforms state-of-the-art baselines, underscoring the effectiveness of dataset-driven parameter recommendation in deep learning.

Index Terms—Hyperparameter selection, Weight initialization, Image characteristics extraction, Deep learning, Meta-learning

1 Introduction

Deep neural networks have made remarkable strides over the past decade, demonstrating exceptional efficacy across various domains, including image classification [1], speech recognition [2], generative models [3], and natural language processing [4]. Similar to conventional machine-learning algorithms, deep neural networks rely on critical hyperparameters that significantly influence their performance in diverse tasks.

Hyperparameters are values or settings that are independent of model parameters, and they are preset by practitioners before training begins. For instance, in convolutional neural networks (CNNs) [5], the critical hyperparameters related to model training include the number of training epochs, batch size, learning rate, dropout rate, and optimization schemes [6]. These hyperparameters should be carefully selected for a given deep-learning task to optimize desired outcomes. In addition to hyperparameters, the initial weights of neural networks play a vital role in controlling the training process. These weights determine convergence speed and whether the model can reach the global optimum, directly impacting performance in real-world applications [7]. A well-chosen weight initialization should be a few training steps away from the desirable solution for the task [8]. Such effective initialization can significantly reduce training epochs, time budget, and computing resources, ultimately improving the efficacy of deep learning models.

The existing strategies for hyperparameter selection in deep learning encompass three main approaches:

1) Search-based optimization [9], [10]: this approach involves exploring hyperparameter configurations through search algorithms. Examples include random search and Bayesian optimization. However, evaluating each searched configuration on the model can be computationally expensive due to the large scale of model parameters and training data in deep learning.
2) Model-based reinforcement learning [11], [12]: in this method, reinforcement learning techniques guide the search for hyperparameters. Researchers have proposed approaches like dynamic programming and efficient exploration. Despite their effectiveness, these methods also demand substantial high-performance computing resources (such as GPUs and TPUs) and are time-consuming.
3) Gradient-based optimization [13], [14]: unlike the previous two approaches, gradient-based methods focus on differentiable hyperparameters. These methods can handle specific hyperparameters related to optimizers, such as learning rate, momentum, and weight decay. Clearly, their application scope is limited by the requirement for differentiability.

Overall, selecting effective hyperparameters efficiently remains a practical challenge in deep learning, necessitating

- *L. Deng and M. Raissi are with the Department of Mathematics, University of California at Riverside, Riverside, CA, 92521, USA. Email: lipingd@ucr.edu (Liping Deng) and maziarr@ucr.edu (Maziar Raissi)*
- *M. Xiao is with the School of Mathematical and Statistical Sciences, Southern Illinois University Carbondale, Carbondale, IL, 62901, USA. E-mail: mxiao@siu.edu.*

careful consideration of computational resources and the specific hyperparameters involved.

On the other hand, the representative approaches to weight initialization are random initialization, model pretraining [15], and model-agnostic meta-learning (MAML) [16]. The popular schemes in random initialization are Glorot initialization [17] and He initialization [18], which sample weights randomly from a bounded uniform or Gaussian distribution, and have shown promising performance in certain cases. However, Glorot initialization is not suitable for large network architectures and does not work well with ReLU due to its linear activation function assumption, and He initialization is designed for parametric ReLU only. Model pretraining is another widely used method in weight initialization, in which the model parameters trained on very large datasets (e.g., ImageNet) are transferred to new tasks. By fine-tuning the model, we adapt it to specific tasks. However, whether these pre-trained weights are universally suitable for new tasks remains an open question [19], [20], [21], [22]. MAML addresses few-shot learning problems where training data is limited. It effectively avoids overfitting. However, MAML's prohibitive computation burden restricts its applicability to large-scale models.

Existing approaches in deep learning often have limitations: they tend to focus solely on either hyperparameter tuning or weight initialization, resulting in inefficiencies and inflexibility. In this paper, we propose a novel strategy within the framework of meta-learning. Our approach simultaneously recommends fine model hyperparameters and initial weights based on dataset similarity. Unlike existing methods, our approach is guided by the widely accepted heuristic that intrinsically similar datasets are likely to benefit from similar model weights and hyperparameter configurations. Specifically, we pursue the following steps: (i) data collection - we begin by collecting a group of historical datasets[1], and from these datasets, we extract meta-features that characterize the most intrinsic aspects of the data; (ii) similarity evaluation - when a query dataset requires model (hyper)parameters, we compare it to the historical datasets, i.e., quantifying the similarities between the query dataset and historical datasets by calculating distances among their meta-features; (iii) recommendations - both well-performed hyperparameters and well-trained model weights from the historical dataset with the shortest distance are adopted, and these parameters are then recommended to use for this query dataset.

Meta-features serve as a group of hand-crafted descriptors that instantiate the underlying characteristics of datasets [23]. In our approach, meta-features play a critical role in quantifying dataset similarities, and their effectiveness significantly influences the success of both hyperparameter tuning and weight selection tasks. However, existing meta-features have primarily been designed for traditional machine-learning datasets [24]. These datasets can be succinctly summarized as an $n \times p$ matrix, where n represents the number of instances, and p denotes the number of attributes. Therefore, conventional meta-features do not apply to image datasets. To address this limitation, our paper proposes a novel approach: extracting meta-features specifically from image datasets. Our goal is to ensure both efficiency and effectiveness. Ideal meta-features should be easy to obtain and should accurately reflect the most important characteristics of the underlying datasets. Following these principles, we introduce two distinct categories of meta-features. The first category is based on five groups of measures that describe the color and texture features of images, and we refer to these as "visible" shallow meta-features. The second category includes 512 characteristics extracted by a CNN model that is well-trained on the ImageNet image classification dataset. We call these "invisible" deep meta-features. The rationale behind this distinction is that, in addition to the visible features, some critical hidden characteristics exist in images, and they should be meaningful in similarity measurement.

Building upon our proposed approach and meta-features, we have developed a deep neural network hyperparameter and weight recommendation system. In this system, we utilized 75 real-world image classification datasets as historical datasets. To validate the effectiveness of our approach, we employed both a vision transformer (ViT) and a convolutional neural network (CNN) as the backbone models, in which five hyperparameters and a total of 360 configurations were considered. Our empirical results, obtained from 30 real-world datasets, demonstrate the superiority of our approach over the state-of-the-art hyperparameter search algorithms and weight initialization strategies. Specifically, our similarity-based recommendation and meta-feature learning significantly enhance the performance of deep learning models.

In summary, our main contributions to this paper are described as follows:

1) We simultaneously recommend both hyperparameters and initial weights for deep neural networks based on dataset similarity within the framework of meta-learning. This approach overcomes the limitations of existing methods for hyperparameter selection and weight initialization.
2) We extract image meta-features, introducing two types:
   - Visible shallow meta-features, which describe color and texture features of images.
   - Invisible deep meta-features, extracted by a CNN model trained on ImageNet. These hidden features enhance image characterization.
3) Through extensive experiments, we validate our approach on 105 real image datasets, using both vision transformer (ViT) and CNN. Our empirical results demonstrate the effectiveness and superiority of our approach compared to other state-of-the-art baselines.

The remainder of the paper is structured as follows. In Section 2, we provide a brief review of current approaches to hyperparameter optimization and weight initialization for deep neural networks. Next, in Section 3, we introduce our proposed two types of image meta-features. The details of our recommendation framework are outlined in Section 4.

1. Historical datasets are those where the performance of target algorithms is known, meaning no further learning is required on those datasets.

Subsequently, we present the empirical results and discussions in Section 5, and finally, we summarize the conclusions and outline future work in Section 6.

2 Related work

This section is divided into two subsections. The first subsection provides a concise review of hyperparameter optimization algorithms in deep learning. The second subsection delves into the various approaches for weight initialization. The details are as follows.

2.1 Hyperparameter optimization

The representative strategy of hyperparameter tuning is search-based. Grid search is the simplest scheme, greedily evaluating every hyperparameter configuration within the defined search space. While it can find the best ones, its computational complexity grows exponentially as the number of hyperparameters increases. In contrast, random search [9] samples and evaluates points from the hyperparameter space randomly, using a preset stopping criterion. It outperforms grid search when hyperparameters are not uniformly distributed. Bayesian optimization (BO) [10], on the other hand, leverages previous evaluations by utilizing a probabilistic surrogate model (such as a Gaussian process). BO then determines which promising hyperparameter configuration should be evaluated next, guided by an acquisition function. Other search strategies include population-based search (e.g., particle swarm optimization [25] and successive halving [26]). However, due to the need to evaluate each searched configuration on deep neural networks (often with millions of parameters), search algorithms are frequently time-consuming and computationally heavy.

The second strategy in hyperparameter tuning involves gradient-based optimization [27]. As the name suggests, we optimize the hyperparameter selection criterion using gradient descent, which necessitates calculating the hypergradients of hyperparameters. However, since the selection criterion for hyperparameters is typically non-convex and non-smooth, it lacks differentiability, limiting its applicability. Although gradient-based methods have garnered attention in few-shot learning [13], [14], they are primarily effective for specific hyperparameters, such as learning rate, momentum, and weight decay, within the optimizer. Additionally, due to hypergradient degradation and substantial memory costs, gradient-based methods are best suited for simple models and toy datasets with a short horizon—where the learning process can be completed within a few gradient steps.

The third strategy in hyperparameter tuning involves model-based reinforcement learning. In this approach, the agent selects a hyperparameter configuration from the hyperparameter space based on its current policy. It then evaluates the chosen hyperparameter on the model to obtain a score (such as classification accuracy), which serves as the reward for this action. The agent's state is updated accordingly. This process is repeated until a desired hyperparameter configuration is found. Wu et al. [12] proposed modeling the hyperparameter optimization process as a sequential decision problem, akin to a Markov decision process. They designed a reinforcement learning agent specifically to seek the desired hyperparameters. Meanwhile, Dong et al. [11] introduced a dynamical hyperparameter optimization algorithm based on reinforcement learning and continuous deep Q-learning. Their work found applications in visual object tracking. However, similar to search algorithms, this strategy necessitates hyperparameter evaluation during its process, which can be inefficient on large-scale datasets and models.

2.2 Weight Initialization

The widely adopted strategy of weight initialization is random initialization because of its simplicity, where weights are sampled following a Gaussian distribution with a specific mean and variance or uniformly within a relevant interval. Especially, Glorot et al. [17] proposed to initialize weights from a random uniform distribution bounded by $(-\frac{\sqrt{6}}{\sqrt{n_i+n_j}}, \frac{\sqrt{6}}{\sqrt{n_i+n_j}})$, known as Glorot initialization, where n_i and n_j are respectively the number of incoming and outgoing network connections within a layer, which can guarantee the variances of the input and output of a layer stay the same. He et al. [18] proposed to take weights from a normal distribution $\mathcal{N}(-\frac{\sqrt{6}}{\sqrt{n_i(1+a^2)}}, \frac{\sqrt{6}}{\sqrt{n_i(1+a^2)}})$, where n_i is the number of connections of the incoming network in a layer and a is a parameter of the parametric ReLU activation function. This method is shown to outperform Glorot initialization because it is effective in deeper and wider network architectures. Another strategy in random initialization is to use the training data; for example, Gan et al. [28] took patches from each training image to form a matrix on which the principal component analysis algorithm was applied, and the obtained eigenvectors were further leveraged as the weights of convolutional filters in CNNs.

The second strategy for weight initialization is model pretraining, which can be classified as supervised and unsupervised pretraining. In the first case, the weights of the new model are initialized by parameters that are well-trained on other datasets (such as ImageNet [15]). This process, often referred to as transfer learning in deep learning, allows us to fine-tune the model with fewer steps. However, whether this initialization is effective for all new tasks remains an open question. On the other hand, various (convolutional) autoencoder (AE) neural networks [29], [30] are employed in unsupervised pretraining due to their strong feature learning capabilities. An AE has a symmetric structure comprising an encoder that maps input data to a lower-dimensional latent space and a decoder that accurately reconstructs the input data. Once the AE is fully trained on a dataset, the encoder can be further combined with a fully connected layer to perform tasks such as classification or segmentation.

In addition, model-agnostic meta-learning (known as MAML) [16] serves as a powerful tool for learning initial weights in few-shot learning scenarios. When training data is scarce and starting from scratch often leads to overfitting, MAML comes to the rescue. It adopts a bilevel optimization strategy: at the inner level, it optimizes the loss across a range of tasks sampled from a distribution; at the outer level, it optimizes the loss specifically for the task of interest. Initially, MAML initializes the weights randomly, and then iteratively updates them within both the inner and outer

optimization loops. However, MAML's main drawback lies in its high computational complexity, particularly when applied to large network models, rendering it less practical in many scenarios. For a more comprehensive review of weight initialization techniques, readers can refer to a recent survey by Narkhede et al. [7].

3 META-FEATURE EXTRACTION

In this section, we begin by summarizing the general mathematical notations used throughout this paper. Subsequently, we introduce the proposed meta-features for image datasets.

3.1 Notations

In this paper, matrices and vectors are represented by bold uppercase and lowercase letters, respectively, e.g., $\mathbf{X} \in \mathbb{R}^{m\times n}$ and $\mathbf{x} \in \mathbb{R}^{m\times 1}$, and multidimensional tensors are denoted as calligraphic letters, e.g., $\mathcal{I} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_p}$. The entries of a matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ (vector $\mathbf{x} \in \mathbb{R}^{m\times 1}$) are written as X_{ij} (x_i) where i and j are the indexes of each dimension, and the elements of a tensor $\mathcal{I} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_p}$ are written as $\mathcal{I}(i_1, i_2, \cdots, i_p)$ where $i_j, j = 1, 2, \cdots, p$, are the indexes of each dimension.

In addition, an image dataset is denoted as the Ralph Smith formal script $\mathscr{D}$, and a deep learning algorithm is denoted as the Ralph Smith formal script $\mathscr{M}$. The cardinality of a set Ω is represented by $|\Omega|$, and $\mathrm{floor}(x) = \lfloor x \rfloor$ is the floor function. Other notations will be introduced subsequently when necessary.

3.2 Proposed meta-features

Our meta-features encompass six distinct groups of measures, including indexed color histogram, color coherence vectors, color structure descriptors, color autocorrelogram, Haralick, and deep hidden features. In our following discussions, suppose an RGB (red, green, and blue) image $\mathcal{I} \in \mathbb{R}^{n_1\times n_2\times 3}$ is given, where $\mathcal{I}(:,:,i), i = 1, 2, 3$, are corresponding to the red, green, and blue channels, respectively, and the pixels of $\mathcal{I}$ are normalized into $[0, 1]$, i.e., $\mathcal{I}(i, j, k) \in [0, 1]$ for any i, j, and k. Furthermore, the image is converted to an indexed image $\mathbf{I} \in \mathbb{R}^{n_1\times n_2}$ using an universal colormap $\{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_n\}$ where $\mathbf{c}_i \in \mathbb{R}^{3\times 1}, i = 1, 2, \cdots, n$, and each dimension of $\mathbf{c}_i$ is corresponding to a color channel, and n is the number of quantized colors.

3.2.1 Indexed color histogram (ICH)

Color histograms are an effective tool to characterize the color distribution of an image. To create an indexed color histogram (ICH), we classify the pixels in the indexed image $\mathbf{I}$ into n groups based on which color bin they belong to. Denote the group that is corresponding to the color bin $\mathbf{c}_i$ as G_i, then

$$G_i = \{I_{jk} | C(I_{jk}) = \mathbf{c}_i\},$$

where $I_{jk}, j = 1, 2, \cdots, n_1, k = 1, 2, \cdots, n_2$, is a pixel on the indexed image $\mathbf{I}$ and $C(I_{jk}) = \mathbf{c}_i$ indicates the color of pixel I_{jk} is $\mathbf{c}_i$. Finally, we count the number of colors in each bin and obtain an n-dimensional vector which is the ICH meta-features, i.e.,

$$\mathrm{ICH} = [|G_1|, |G_2|, \cdots, |G_n|]. \tag{1}$$

3.2.2 Color coherence vector (CCV)

The previous ICH meta-features primarily focus on the statistical distributions of pixels or colors, while they ignore their spatial information, which is critical to identify the patterns and layouts of colors of an image. Hence, we consider the color coherence vector (CCV). Different from color histograms, CCV classifies the pixels in each bin into coherent and non-coherent. For a particular color bin $\mathbf{c}_i$, we first construct a binary image $\mathbf{B}$ that has the same dimension as $\mathbf{I}$ where

$$B_{jk} = \begin{cases} 1, & \text{if } C(I_{jk}) = \mathbf{c}_i, \\ 0, & \text{if } C(I_{jk}) \neq \mathbf{c}_i. \end{cases}$$

A connected component (CC) in $\mathbf{B}$ is the largest set of pixels such that for any two pixels in the CC, there is always a pathway between them. Then, for each CC in terms of pixel 1 in $\mathbf{B}$, we count the number of pixels R and compare it to a positive integer $\tau = \mathrm{floor}(\frac{n_1 n_2}{100})$, and these pixels are assumed to be coherent if $R \geq \tau$ and non-coherent otherwise. Next, we count the total number of coherent (C_i) and noncoherent (N_i) pixels, denoted as (C_i, N_i). Thus, CCV meta-feature of image $\mathcal{I}$ can be defined as

$$\mathrm{CCV} = \big[(C_1, N_1), (C_2, N_2), \cdots, (C_n, N_n)\big],$$

which is a $2n$-dimensional vector.

3.2.3 Color structure descriptor (CSD)

CSD is another meta-feature that focuses on color spatial distribution. It is defined in the HMMD (Hue-Min-Max-Difference) color space [31], which better represents color relationships based on human perception compared to the RGB color space. For a given RGB image $\mathcal{I}$, we first convert it into an HMMD image $\mathcal{I}'$. Then the image is quantized to n colors $\{\mathbf{c}'_1, \mathbf{c}'_2, \cdots, \mathbf{c}'_n\}$ to obtain an indexed image $\mathbf{I}' \in \mathbb{R}^{n_1\times n_2}$, on which a structure window, e.g., a 3×3 square, is moved throughout the image $\mathbf{I}'$ and we count the total times of each pixel that shown in the window to form an n-dimensional vector, which is the CSD meta-features, i.e.,

$$\mathrm{CSD} = \big[|\mathbf{c}'_1|, |\mathbf{c}'_2|, \cdots, |\mathbf{c}'_n|\big], \tag{2}$$

where $|\mathbf{c}'_i|$ represents the total number of times that color $\mathbf{c}'_i$ is detected in the sliding window. To create a more robust CSD, we adopt 3 square sliding windows with increasing sizes $k \times k$ where $k = 3, 5, 7$, and their concatenation is employed as the CSD meta-features having a length of $3n$.

3.2.4 Color autocorrelogram (CAC)

Color autocorrelogram (CAC) is the diagonal of the color correlogram matrix that focuses on image texture features by counting the probability of a pair of colors with a certain distance d. The elements of CAC are described as follows:

$$\mathrm{CAC}^d(i) = \Pr\{(I_{jk}, I_{j'k'})\}, i = 1, 2, \cdots, n,$$
$$s.t.,\ C(I_{jk}) = C(I_{j'k'}) = \mathbf{c}_i \text{ and } \mathrm{dist}(I_{jk}, I_{j'k'}) = d,$$

where $\Pr(E)$ stands for the probability of event E, I_{jk} and $I_{j'k'}$ are any two pixels of the indexed image $\mathbf{I}$, and $\mathrm{dist}(\cdot,\cdot)$ represents the distance of two pixels. We consider $d \in [1, 3, 5, 7]$ so four CAC vectors can be obtained and their concatenation is taken as the final CAC meta-features, i.e.,

$$\mathrm{CAC} = \big[\mathrm{CAC}^1, \mathrm{CAC}^3, \mathrm{CAC}^5, \mathrm{CAC}^7\big], \tag{3}$$

which has a length of $4n$.

3.2.5 Haralick features (HARA)

Haralick features are the second type of image texture feature that is based on the gray-level co-occurrence matrices (GLCMs). GLCMs are symmetric matrices whose entries are the frequency of a particular pair of grey levels. How to obtain the GLCMs from a color image $\mathcal{I}$ can be found in [32]. Suppose a GLCM $\mathbf{P} \in \mathbb{R}^{n\times n}$ is calculated from image $\mathcal{I}$ where n is the number of grey levels, then it is further normalized by

$$\widetilde{\mathbf{P}} = \frac{\mathbf{P}}{n^2 \sum_{i,j=1}^{n} P_{ij}}.$$

Twenty features are proposed in the literature [33], [34], [35] based on matrix $\widetilde{\mathbf{P}}$, and we choose five of them in this paper and they are cluster shade, correlation, information measure, contrast, and max probability, because most of them are highly correlated. Definitions are as follows.

Cluster shade:

$$f_1 = \frac{1}{n^2}\sum_{i,j=1}^{n}\left(\frac{i}{n}+\frac{j}{n}-2\tilde{\mu}\right)^3 \tilde{P}_{ij}, \tag{4}$$

where $\tilde{\mu} = \frac{1}{n^2}\sum_{i,j=1}^{n}\tilde{P}_{ij}$.

Correlation:

$$f_2 = \frac{1}{n^2}\sum_{i,j=1}^{n}\left(\frac{\frac{i}{n}-\tilde{\mu}_x}{\tilde{\sigma}_x}\right)\left(\frac{\frac{j}{n}-\tilde{\mu}_y}{\tilde{\sigma}_y}\right)\tilde{P}_{ij} \tag{5}$$

where

$$\begin{aligned}
\tilde{\mu}_x &= \frac{1}{n}\sum_{i=1}^{n}\frac{i}{n}\cdot\tilde{p}_x(i),\\
\tilde{\mu}_y &= \frac{1}{n}\sum_{j=1}^{n}\frac{j}{n}\cdot\tilde{p}_y(j),\\
\tilde{\sigma}_x^2 &= \frac{1}{n}\sum_{i=1}^{n}(\frac{i}{n}-\tilde{\mu}_x)^2\cdot\tilde{p}_x(i),\\
\tilde{\sigma}_y^2 &= \frac{1}{n}\sum_{j=1}^{n}(\frac{j}{n}-\tilde{\mu}_y)^2\cdot\tilde{p}_y(j),
\end{aligned}$$

and $\tilde{p}_x(i) = \frac{1}{n}\sum_{j=1}^{n}\tilde{P}_{ij}$ and $\tilde{p}_y(j) = \frac{1}{n}\sum_{i=1}^{n}\tilde{P}_{ij}$.

Information measure:

$$f_3 = \frac{\widetilde{HXY} - \widetilde{HXY1}}{\max\{\widetilde{HX},\widetilde{HY}\}} \tag{6}$$

where

$$\begin{aligned}
\widetilde{HX} &= -\frac{1}{n}\sum_{i=1}^{n}\tilde{p}_x(i)\cdot\log\tilde{p}_x(i),\\
\widetilde{HY} &= -\frac{1}{n}\sum_{j=1}^{n}\tilde{p}_y(j)\cdot\log\tilde{p}_y(j),\\
\widetilde{HXY} &= -\frac{1}{n^2}\sum_{i,j=1}^{n}\tilde{P}_{ij}\cdot\log\tilde{P}_{ij},\\
\widetilde{HXY1} &= -\frac{1}{n^2}\sum_{i,j=1}^{n}\tilde{P}_{ij}\cdot\log[\tilde{p}_x(i)\cdot\tilde{p}_y(j)].
\end{aligned}$$

Contrast:

$$f_4 = \frac{1}{n^2}\sum_{i,j=1}^{n}\left(\frac{i}{n}-\frac{j}{n}\right)^2\tilde{P}_{ij}. \tag{7}$$

Max probability:

$$f_5 = \max_{i,j}\tilde{P}_{ij}. \tag{8}$$

In addition to the existing features, we propose the following three measures that are based on colors' spatial distributions and quantities.

Sum of diagonal:

$$f_6 = \frac{1}{n^2}\sum_{i=1}^{n}\tilde{P}_{ii}. \tag{9}$$

A diagonal entry $\tilde{P}_{ii}$ represents the number of pixels of color i that are aggregated together, so the sum of the diagonal measures the aggregation of colors. A larger value of f_6 implies each color has a more compact layout.

Sum of outliers:

$$f_7 = \frac{1}{n^2}\sum_{i,j=1}^{n}\tilde{P}_{ij}\cdot \mathrm{I}_{\tilde{P}_{ii}<\tilde{P}_{ij}}, \tag{10}$$

where $\mathrm{I}_{a<b}(a,b)$ is an indicator function and $\mathrm{I}(a,b)=1$ if $a<b$ otherwise $\mathrm{I}(a,b)=0$. Based on the definition of GLCM, we call $\tilde{P}_{ij}$ an outlier if $\tilde{P}_{ii}<\tilde{P}_{ij}$, which suggests there are more other colors in the surroundings of color i. Hence, a larger f_7 indicates a more fragmented distribution of colors, or the numbers of various colors are imbalanced.

Sum of zeros:

$$f_8 = \frac{1}{n^2}\sum_{i,j=1}^{n}\mathrm{I}_{\tilde{P}_{ij}=0}, \tag{11}$$

where $\mathrm{I}_{a=0}(a)$ is an indicator function and $\mathrm{I}(a)=1$ if $a=0$ otherwise $\mathrm{I}(a)=0$. A zero entry $\tilde{P}_{ij}$ in $\widetilde{\mathbf{P}}$ means the color i and color j are not close to each other, so a larger f_8 indicates the colors in $\mathcal{I}$ are more likely well-aggregated.

Therefore, our Haralick meta-feature is an 8-dimensional vector, i.e., $\mathrm{Hara} = [f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8]$.

3.2.6 Deeply hidden features (DEEP)

The meta-features mentioned above are based on the image's colors and texture, which are "visible" shallow characteristics. There are more critical underlying features that exist in the images, and extracting those features will be beneficial to dataset similarity measurements. To extract the latent hidden features from images, we propose using convolutional neural networks, which are shown to be powerful in deep feature learning, as meta-feature extractors. Specifically, the backbone of our neural meta-feature extractor adopts the main body of the famous VGG19 [36] architecture with some modifications. It contains 16 convolutional layers, 5 max pooling layers, and 1 adaptive average pooling layer. Suppose the input image has a size of 224×224, then the feature map after the fifth max pooling layer has a dimension of $512\times7\times7$, and it is further fed into the average pooling layer, so the output is a 512-dimensional vector, which is the extracted meta-features. The model parameters of our meta-feature extraction network are transferred from VGG19, which is well-trained on the ImageNet image dataset.

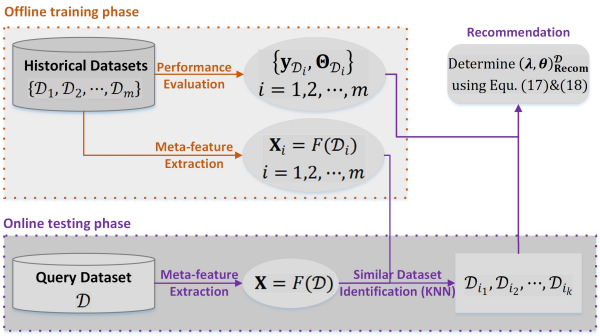

Fig. 1. The workflow of similarity-based hyperparameter and weight recommendation.

4 Similarity-based Parameter Selection

In this section, we initially address the challenges of hyperparameter optimization and weight initialization. Subsequently, we delve into the proposed similarity-based recommendation approach.

4.1 Hyperparameter optimization and weight initialization

Suppose the deep learning model $\mathscr{M}$ of interest has r distinct hyperparameters $\lambda_1, \lambda_2, \cdots, \lambda_r$, then its hyperparameter space is defined as the Cartesian product

$$\boldsymbol{\Lambda}_{\mathscr{M}} = \Lambda_1 \times \Lambda_2 \times \cdots \times \Lambda_r, \tag{12}$$

where $\Lambda_i, i = 1, 2, \cdots, r$, is the search space of hyperparameter $\lambda_i, i = 1, 2, \cdots, r$. Besides, we define the initialization model parameter space as $\boldsymbol{\Theta}_0$ with infinite elements and comprises all possible weights of $\mathscr{M}$. For a given image dataset $\mathscr{D}$, we try to find a hyperparameter configuration $\boldsymbol{\lambda}^\star = [\lambda_1, \lambda_2, \cdots, \lambda_r]$ where $\lambda_i \in \Lambda_i, i = 1, 2, \cdots, r$, as well as a model weight $\boldsymbol{\theta}_0^\star \in \boldsymbol{\Theta}_0$ that can optimize the performance of $\mathscr{M}$ on $\mathscr{D}$. Mathematically, it is formulated as

$$\boldsymbol{\lambda}^\star, \boldsymbol{\theta}_0^\star = \arg\operatorname{opt}_{\boldsymbol{\lambda} \in \Lambda_{\mathscr{M}}, \boldsymbol{\theta}_0 \in \boldsymbol{\Theta}_0} V(\mathcal{P}, \mathscr{M}_{(\boldsymbol{\lambda}, \boldsymbol{\theta}_0)}, \mathscr{D}),$$

where $V(\cdot, \cdot, \cdot)$ is the performance validation strategy, $\mathcal{P}$ is the performance evaluation metric, e.g., classification accuracy, and $\mathscr{M}_{(\boldsymbol{\lambda}, \boldsymbol{\theta}_0)}$ represents the model that is instantiated by the hyperparameters $\boldsymbol{\lambda}$ and weights $\boldsymbol{\theta}_0$.

4.2 Similarity-based recommendation

The process of similarity-based parameter recommendation is divided into two phases, namely, the offline training phase and the online recommendation phase, as shown in Figure 1. We elaborate on the details next.

4.2.1 Offline phase

In the offline phase, we first collect a group of historical image classification datasets $\{\mathscr{D}_1, \mathscr{D}_2, \cdots, \mathscr{D}_m\}$, from which the metadata, including the historical performance of hyperparameters and meta-features, is then extracted.

Performance evaluation: For a given historical dataset $\mathscr{D}_i$, it is first split into 10 folds using the stratified splitting scheme that can eliminate the influence of imbalance classes, then 8 folds are randomly chosen and employed as the training set $\mathscr{D}_i^{\text{tr}}$ and rest of 2 folds are used as the validation set $\mathscr{D}_i^{\text{eval}}$. Then the model $\mathscr{M}$ instantiated with a hyperparameter configuration $\boldsymbol{\lambda}_j \in \boldsymbol{\Lambda}_{\mathscr{M}}$ is trained on $\mathscr{D}_i^{\text{tr}}$. Here, the weights of $\mathscr{M}$ are randomly initialized. When the training process is done, the trained model weights $\boldsymbol{\theta}_{\boldsymbol{\lambda}_j}$ are saved, and we evaluate $\mathscr{M}_{\boldsymbol{\lambda}_j}$ on $\mathscr{D}_i^{\text{eval}}$ to obtain the classification accuracy which is treated as the performance of hyperparameter $\boldsymbol{\lambda}_j$. Mathematically, this process can be expressed as follows:

$$y_{\boldsymbol{\lambda}_j} = \mathcal{P}(\mathscr{M}_{\boldsymbol{\lambda}_j}, \mathscr{D}_i^{\text{tr}}, \mathscr{D}_i^{\text{eval}}), j = 1, 2, \cdots, |\boldsymbol{\Lambda}_{\mathscr{M}}|, \tag{13}$$

where $y_{\boldsymbol{\lambda}_j} \in [0, 1]$ is the obtained performance of hyperparameter configuration $\boldsymbol{\lambda}_j$, $\mathcal{P}(\cdot)$ is the classification accuracy metric, and $\mathcal{P}(\mathscr{M}_{\boldsymbol{\lambda}_j}, \mathscr{D}_i^{\text{tr}}, \mathscr{D}_i^{\text{eval}})$ represents the performance of model $\mathscr{M}_{\boldsymbol{\lambda}_j}$ that is trained on $\mathscr{D}_i^{\text{tr}}$ and evaluated on $\mathscr{D}_i^{\text{eval}}$.

After all candidate hyperparameter configurations in $\boldsymbol{\Lambda}_{\mathscr{M}}$ are evaluated, a performance vector

$$\mathbf{y}_{\mathscr{D}_i} = [y_{\boldsymbol{\lambda}_1}, y_{\boldsymbol{\lambda}_2}, \cdots, y_{\boldsymbol{\lambda}_{|\boldsymbol{\Lambda}_{\mathscr{M}}|}}]^\top \tag{14}$$

and a weight parameter set

$$\boldsymbol{\Theta}_{\mathscr{D}_i} = \{\boldsymbol{\theta}_{\boldsymbol{\lambda}_1}, \boldsymbol{\theta}_{\boldsymbol{\lambda}_2, \cdots, \boldsymbol{\theta}_{\boldsymbol{\lambda}_{|\boldsymbol{\Lambda}_{\mathscr{M}}|}}}}\}$$

on dataset $\mathscr{D}_i$ can be formed where $y_{\boldsymbol{\lambda}_j}$ and $\boldsymbol{\theta}_{\boldsymbol{\lambda}_j}$ are the performance and associated weights of hyperparameter configuration $\boldsymbol{\lambda}_j, j = 1, 2, \cdots, |\boldsymbol{\Lambda}_{\mathscr{M}}|$, respectively.

Meta-feature extraction: Suppose a given historical dataset $\mathscr{D}_i$ has N_i images, i.e., $\mathscr{D}_i = \{\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_{N_i}\}$, we extract the proposed meta-features from each image $\mathcal{I}_j$, denoted as $\mathbf{x}_j = F(\mathcal{I}_j), j = 1, 2, \cdots, N_i$, where $F(\cdot)$ stands for the defined meta-feature extraction function, then the meta-features of the dataset $\mathscr{D}_i$ is defined as

$$\mathbf{X}_{\mathscr{D}_i} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{N_i}] \in \mathbb{R}^{r \times N_i},$$

where each column is the meta-features of an image with a size of r.

When the meta-features of all historical datasets are extracted, we normalize each meta-feature into $[0, 1]$ using the min-max scaling to eliminate the possible influence of scales in our later similarity calculations. Finally, the mean and the covariance of $\mathbf{X}_{\mathscr{D}_i}$ are computed, i.e.,

$$\boldsymbol{\mu}_{\mathscr{D}_i} = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{x}_j, \tag{15}$$

$$\boldsymbol{\Sigma}_{\mathscr{D}_i} = \frac{1}{N_i - 1} \sum_{j=1}^{N_i} (\mathbf{x}_j - \boldsymbol{\mu}_{\mathscr{D}_i})(\mathbf{x}_j - \boldsymbol{\mu}_{\mathscr{D}_i})^\top, \tag{16}$$

and then we assume $\mathbf{X}_{\mathscr{D}_i}$ follows a multivariate Gaussian distribution with mean $\boldsymbol{\mu}_{\mathscr{D}_i}$ and covariance $\boldsymbol{\Sigma}_{\mathscr{D}_i}$, that is $\mathbf{x}_j \sim \mathcal{N}_r(\boldsymbol{\mu}_{\mathscr{D}_i}, \boldsymbol{\Sigma}_{\mathscr{D}_i})$ for $j = 1, 2, \cdots, N_i$.

4.2.2 Online phase

When we are given a new (query) dataset $\mathscr{D}$ with N images, its most suitable hyperparameters and initial weights are required to be determined. We first extract its meta-features, denoted as $\mathbf{X}_{\mathscr{D}} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N]$. Similarly, we assume $\mathbf{X}_{\mathscr{D}}$ follows a multivariate Gaussian distribution $\mathcal{N}_r(\boldsymbol{\mu}_{\mathscr{D}}, \boldsymbol{\Sigma}_{\mathscr{D}})$ where mean $\boldsymbol{\mu}_{\mathscr{D}}$ and covariance $\boldsymbol{\Sigma}_{\mathscr{D}}$ are computed by (15) and (16), respectively. To measure the similarity between $\mathscr{D}$ and a historical dataset $\mathscr{D}_i$, we calculate the

TABLE 1
The search space of the hyperparameters of ViT-B/32. A dropout rate being 0 means there is no dropout during the training.

Hyperparameters	Range
learning rate	{0.0001,0.001,0.01,0.1}
batch size	{32, 128, 256}
optimizer	{Adam, SGD, Rmprop}
training epoch	{10,20,30,40,50}
dropout rate	{0, 0.5}

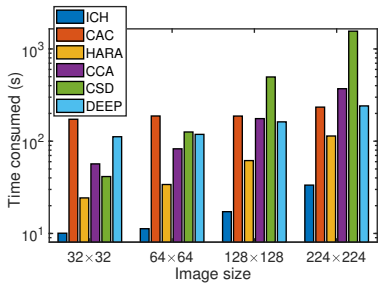

Fig. 2. Comparisons on the time consumed on meta-feature extractions of proposed meta-features on image dataset Lung and Colon Cancer that has 25,000 images, under four different image sizes.

distance between the distributions of their meta-features, and the following Fréchet distance measure [37] is adopted

$$\mathrm{dist}(\mathscr{D},\mathscr{D}_i) = \|\boldsymbol{\mu}_{\mathscr{D}} - \boldsymbol{\mu}_{\mathscr{D}_i}\|_2^2 + \mathrm{Tr}[\boldsymbol{\Sigma}_{\mathscr{D}} + \boldsymbol{\Sigma}_{\mathscr{D}_i} - 2(\boldsymbol{\Sigma}_{\mathscr{D}}\boldsymbol{\Sigma}_{\mathscr{D}_i})^{\frac{1}{2}}],$$

where $\|\cdot\|_2$ stands for the vector L_2 norm, $\mathrm{Tr}(\cdot)$ is the trace of matrices, and $(\boldsymbol{\Sigma}_{\mathscr{D}}\boldsymbol{\Sigma}_{\mathscr{D}_i})^{\frac{1}{2}}$ is the square root of matrix $\boldsymbol{\Sigma}_{\mathscr{D}}\boldsymbol{\Sigma}_{\mathscr{D}_i}$. Therefore, a shorter distance $\mathrm{dist}(\mathscr{D},\mathscr{D}_i)$ indicates a higher degree of similarity between $\mathscr{D}$ and $\mathscr{D}_i$.

We adopt the k-nearest neighbor (kNN) as the recommendation algorithm. Specifically, we first select k historical datasets that have the shortest distance with $\mathscr{D}$ based on the defined distance and then, on each selected dataset, we assign ranks to hyperparameter configurations based on their performance: the best one has rank of 1 while the worst one has rank of $|\Lambda_{\mathscr{M}}|$, where $|\Lambda_{\mathscr{M}}|$ is the total number of hyperparameter configurations. Finally, we pick the top hyperparameter configuration that has the smallest average rank over the k datasets and recommend it to the new dataset $\mathscr{D}$, i.e.,

$$\boldsymbol{\lambda}_{\mathrm{recom}}^{\mathscr{D}} = \arg\min_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}_{\mathscr{M}}} \frac{1}{k}\sum_{j=1}^{k} \mathrm{rank}(\mathbf{y}_{\mathscr{D}_{i_j}}) \tag{17}$$

where $\mathscr{D}_{i_j}, j = 1, 2, \cdots, k$, are the k most similar historical datasets concerning the query dataset $\mathscr{D}$, $\mathrm{rank}(\cdot)$ is ranking strategy described above, and $\mathbf{y}_{\mathscr{D}_{i_j}}$ is the performance vector on dataset $\mathscr{D}_{i_j}$ as defined in (14). The associated model parameters of $\boldsymbol{\lambda}_{\mathrm{recom}}^{\mathscr{D}}$ on these k datasets are therefore recommended to the query dataset $\mathscr{D}$, i.e.,

$$\boldsymbol{\theta}_{\mathrm{recom}}^{\mathscr{D}} = \{\boldsymbol{\theta}_{\boldsymbol{\lambda}_{\mathrm{recom}}}^{\mathscr{D}_{i_1}}, \boldsymbol{\theta}_{\boldsymbol{\lambda}_{\mathrm{recom}}}^{\mathscr{D}_{i_2}}, \cdots, \boldsymbol{\theta}_{\boldsymbol{\lambda}_{\mathrm{recom}}}^{\mathscr{D}_{i_k}}\}, \tag{18}$$

where $\boldsymbol{\theta}_{\boldsymbol{\lambda}_{\mathrm{recom}}}^{\mathscr{D}_{i_j}}$ stands for the model parameters associated to hyperparameter configuration $\boldsymbol{\lambda}_{\mathrm{recom}}^{\mathscr{D}}$ on dataset $\mathscr{D}_{i_j}, j = 1, 2, \cdots, k$. In practice, we can choose the first one $\boldsymbol{\theta}_{\boldsymbol{\lambda}_{\mathrm{recom}}}^{\mathscr{D}_{i_1}}$ only or their average.

5 Experiments

In this section, we evaluate the effectiveness of our proposed approach and meta-features for hyperparameter and weight recommendation. To construct the recommendation system, we curated a dataset pool comprising 105 image classification datasets sourced from Kaggle.com. These datasets exhibit a wide range of characteristics: the number of classes varies from 2 to 211, and the instance counts span from 174 to 57,161. From this collection, we randomly selected 75 historical datasets that represent the aforementioned distributions. The remaining 30 datasets serve as query datasets, upon which our recommendations are applied. For detailed information about the datasets, please refer to the supplementary document.

Our experiment reports are divided into three parts. In the first part, we demonstrate the efficiency of the proposed meta-features in meta-feature extraction. In the second and third parts, we separately present the performance of hyperparameters and weight recommendations.

5.1 Efficiency in meta-feature extraction

To show the efficiency of our proposed meta-features, we report the time consumed on meta-feature extraction on the image dataset Lung and Colon Cancer [32], which has 25,000 images. The settings and configurations employed for each type of meta-feature are explained next, and they are used throughout our experiments. To ensure a fair comparison, meta-features ICH, CCV, and CAC adopt the same RGB colormap with a length of 32, i.e., $n = 32$ in Subsection 3.2, and CSD adopts an HMMD colormap with a length of 32 as well; therefore, ICH, CCV, CAC, and CSD have dimensions of 32, 64, 128, and 96, respectively. To construct a GLCM for HARA meta-features, we consider four directions, namely, horizontal, vertical, left lean diagonal, and right lean diagonal, to obtain four GLCMs, and their sum is leveraged to calculate the HARA meta-features. The shallow meta-features, i.e., ICH, CCV, CAC, CSD, and HARA, are extracted on an AMD Ryzen 7 5800H CPU@3.20 GHz with a RAM of 32.0 GB, and the deep meta-features are extracted by a Kaggle GPU P100 with a RAM of 16.0 GB.

The time comparisons are illustrated in Figure 2, where four different image sizes, i.e., $32 \times 32, 64 \times 64, 128 \times 128$, and 224×224, are investigated to show the influence of image size on the efficiency of meta-feature extractions. The reason we choose the size of 224×224 is that the backbone deep neural network models in our following experiments accept 224×224 images as input. From Figure 2, one can see that: 1) the image size has a significant impact on the time needed for almost all meta-feature extractions except for CAC, which shows a steady time budget. CAC requires converting images from RGB color space to HMMD color space first which is a time-consuming process; 2) ICH consumes the least time and needs about 30 seconds for 25,000 224×224 images, while CSD seems to be the most computationally heavy as it takes more than 1,600 seconds under the same circumstance; 3) although DEEP meta-feature shows an increasing tendency in demand of time budget the increment each time is insignificant. HARA

TABLE 2
Comparisons on ACA (Average Classification Accuracy Rate), ARA (Average Recommendation Accuracy Rate), and HR (Hit Rate) between various meta-features under four different image sizes. The ACA of the real best configurations on the testing problems is 91.28%.

Methods	32×32			64×64			128×128			224×224		
	ACA	ARA	HR	ACA	ARA	HR	ACA	ARA	HR	ACA	ARA	HR
ICH	**88.62**	**96.92**	**100.00**	88.37	96.43	**100.00**	88.34	96.39	**100.00**	**88.65**	**96.92**	**100.00**
CAC	88.50	96.64	**100.00**	**88.66**	**96.88**	**100.00**	88.31	96.20	96.67	88.14	95.81	96.67
HARA	88.22	95.93	96.67	88.39	96.47	96.67	88.39	96.33	96.67	88.07	96.14	96.67
CCV	88.08	95.90	96.67	88.31	96.37	**100.00**	88.41	96.50	**100.00**	88.37	96.41	**100.00**
CSD	88.13	96.03	**100.00**	88.41	96.42	**100.00**	88.42	96.43	**100.00**	88.44	96.47	**100.00**
DEEP	88.38	96.44	96.67	88.32	96.14	96.67	88.42	96.42	96.67	88.47	96.47	96.67
COMB	88.36	96.29	**100.00**	88.29	96.24	**100.00**	88.30	96.25	**100.00**	88.37	96.35	100.00

meta-features can be obtained efficiently since it does not need to slide over an image as CSD and CAC do. Next, we are going to investigate the performance of various meta-features that are extracted from different image sizes.

5.2 Performance in hyperparameter recommendations

In this experiment, we choose the famous vision transformer (ViT-B/32) [38] as the backbone model where five hyperparameters, namely, learning rate, optimizer, batch size, training epoch, and dropout rate, are considered, and their search spaces are listed in Table 1 and we have a total of 360 hyperparameter configurations. We adopted the weights of ViT-B/32 pre-trained on ImageNet, fixed them during training, and just fine-tuned the layer of full connection for a new dataset. All training was run on Kaggle GPU P100 and TPU v3-8, and cosine learning rate decay was deployed.

We employ three metrics to evaluate the goodness of the selected hyperparameters, namely, (average) classification accuracy rate (CA, ACA), (average) recommendation accuracy rate (RA, ARA), and hit rate (HR), and they are widely leveraged in hyperparameter and classifier selection tasks [39], [40]. CA, RA, and HR have a range of $[0, 1]$, and a larger value is preferred and indicates a better recommendation. In addition, we also adopt the normalized discounted cumulative gain (NDCG) [41], which is a popular metric in recommender systems, to measure the similarities between the real rankings of hyperparameter configurations and the rankings of the predictions of recommendation approaches.

5.2.1 Performance of proposed meta-features

The empirical results of our meta-features, under four image sizes, are summarized in Table 2 (ACA, ARA, and HR) and Figure 3 (NDCG) separately. In addition to the performance of every single type of meta-feature, we also investigate the effectiveness of their combinations, denoted as "COMB", where the similarity between two datasets is the summation of the similarities calculated on each type of meta-feature. To eliminate the influence of different dimensions of meta-features, the similarity of each type of meta-feature is divided by its length, i.e., $\mathrm{dist}(\mathscr{D}, \mathscr{D}_i) = \sum_{i=1}^{6} \frac{1}{\mathrm{length}(\mathrm{MFE}_i)} \mathrm{dist}_{\mathrm{MFE}_i}(\mathscr{D}, \mathscr{D}_i)$ where MFE_i stands for a specific meta-feature.

From Table 2, one can observe that: 1) all meta-features have achieved satisfying results, since the highest (lowest) ACA, ARA, and HR are respectively 88.65%, 96.92%, and 100.00% for ICH when image size is 224×224 (88.07%, 95.81%, and 96.67% for HARA when image size is 224×224); 2) the image size has different impacts on the performance

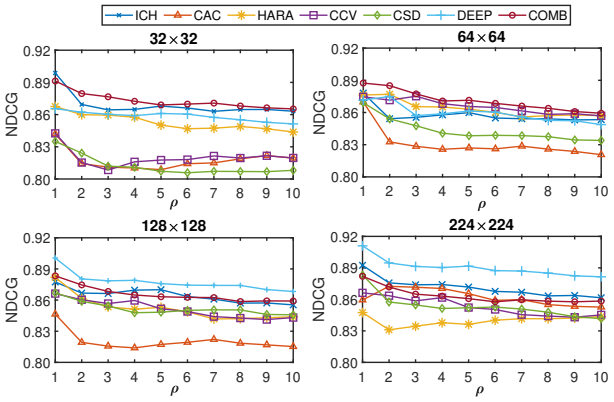

Fig. 3. Comparisons on NDCG@ρ (Normalized Discounted Cumulative Gain), where $\rho = 1, 2, \cdots, 10$, between various meta-features under four different image sizes.

of each type of meta-feature. CCV, CSD, DEEP, as well as COMB, seem to have better performance as the size of images increases, while CAC and HARA present a contrast tendency. In comparison, ICH has the best performance when image sizes are 32×32 and 224×224 and has a similar but lower performance when image sizes are 64×64 and 128×128. DEEP and COMB present a relatively stable performance; 3) among the seven groups of meta-features, ICH achieves the best performance over four image sizes with the highest ACA of 88.65%, ARA of 96.92%, and HR of 100.00%. HARA claims the lowest performance with the highest ACA of 88.39%, ARA of 96.47%, and HR of 96.67%. In particular, COMB does not show a dramatic performance improvement, performing similarly to DEEP.

On the other hand, a higher NDCG@ρ value means the first ρ recommendations made by our approaches are closer to the real top ρ best hyperparameter configurations. From Figure 3, we can see that: 1) the NDCG values under various meta-features mainly show a decreasing tendency as ρ increases, which is a fine property and suggests the ρ-order recommendation can outperform the $(\rho + 1)$-order recommendation; 2) CAC presents a contrast tendency compared to ACA, ARA, and HR, as its best NDCG is found under the image size of 224×224 while the NDCG under other image sizes is mediocre. CCV and CSD have consistent performance as in Table 2 and their NDCG increases as the image size enlarges; 3) ICH, DEEP, and COMB own the best NDCG throughout four image sizes. Especially for DEEP, it outperforms others uniformly under the image sizes of 128×128 and 224×224, which shows its potential.

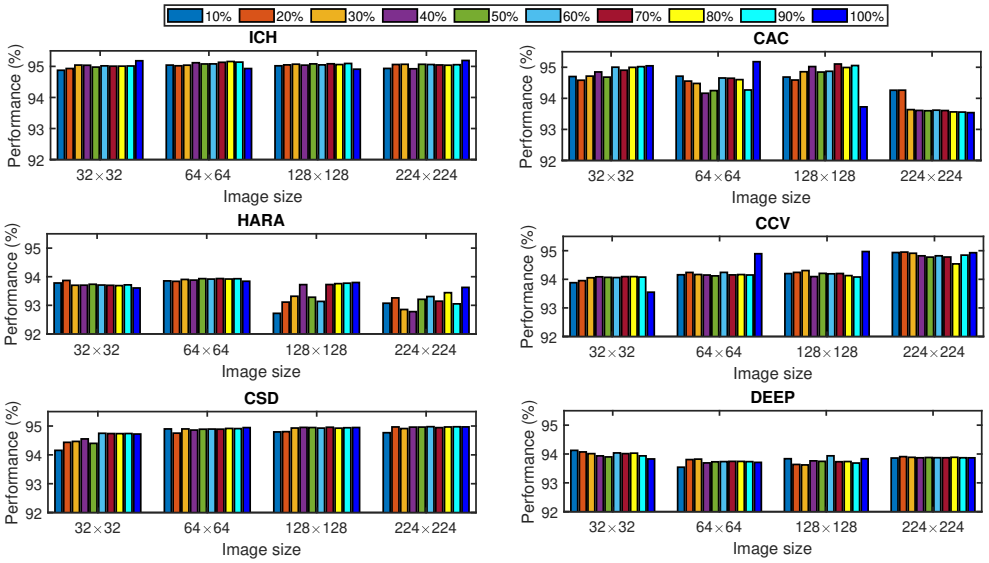

Fig. 4. Performance fluctuation of meta-features when only a portion of the dataset is used for meta-feature extraction. The reported performance is the mean of the three metrics: ACA (Average Classification Accuracy Rate), ARA (Average Recommendation Accuracy Rate), and HR (Hit Rate). The figure is better viewed in color.

5.2.2 Ablation study

In this experiment, we investigate the impact of meta-features on recommendation performance when only a partial dataset is used for meta-feature extraction. In real-world scenarios, dealing with very large-scale datasets can be extremely time-consuming and computationally intensive. To address this, we randomly select ϵN images from each dataset, where $\epsilon \in (0, 1]$ represents the keeping ratio and N is the total number of images. These selected images serve as the basis for extracting meta-features and conducting recommendation tasks. To mitigate the influence of imbalanced classes within a dataset, we adopt a stratified selection strategy. Specifically, we randomly sample images from each class according to the keeping ratio ϵ and combine them to create a new sub-dataset. The empirical results, presented in Figure 4, show the average performance of ACA, ARA, and HR across different values of ϵ, ranging from 0.1 to 1 (i.e., $\epsilon = 0.1, 0.2, \cdots, 1$). We repeat the experiments 10 times for $\epsilon < 1$ to account for randomness. Notably, when $\epsilon = 1$, all images are utilized for meta-feature extraction.

From Figure 4, we can observe that the performance of meta-features under various keeping ratios can remain stable in most cases, and the absence of images does not cause a dramatic performance decline. To be specific, ICH, CSD, and DEEP meta-features show a very minor change in performance under all image sizes and keeping ratios, while there are some stronger fluctuations for CAC (HARA) when image sizes are 64×64 (128×128 and 224×224). On the other hand, there are several obvious performance decreases with smaller keeping ratios for CCV when image sizes are 64×64 and 128×128 and for CAC when image size is 64×64; however, we can also see some cases where the smaller keeping ratios can even produce the better performance, for instance, CAC under image sizes 128×128 and 224×224, CCV under image size 32×32, and DEEP under image size 32×32. Overall, extracting meta-features from partial datasets is a viable approach to reduce computational overhead while ensuring satisfactory performance.

TABLE 3
Comparisons on ACA (Average Classification Accuracy Rate), ARA (Average Recommendation Accuracy Rate), HR (Hit Rate), and NDCG (Normalized Discounted Cumulative Gain) between our approaches and seven search-based algorithms.

Approach	ACA	ARA	HR	NDCG@1	@2	@3
ICH	88.21	96.18	97.78	0.892	0.876	0.874
CAC	**88.32**	**96.32**	96.67	0.852	0.870	0.878
HARA	88.10	96.06	96.67	0.847	0.831	0.834
CCV	88.27	96.16	**98.89**	0.862	0.863	0.864
CSD	88.13	95.96	97.78	0.852	0.853	0.850
DEEP	88.20	96.08	96.67	0.911	**0.895**	**0.891**
COMB	88.21	96.20	**98.89**	0.883	0.873	0.869
RS	72.49	75.82	72.78	0.776	0.676	0.571
BO	76.15	79.97	75.33	0.767	0.664	0.580
PSO	73.71	77.09	72.67	0.760	0.671	0.576
HEBO	68.34	69.72	63.33	0.723	0.593	0.485
AT	86.59	94.13	94.44	**0.918**	0.854	0.800
MTGP	79.15	83.58	83.89	0.838	0.761	0.660
RGPE	85.81	92.63	90.89	0.826	0.778	0.714

5.2.3 Comparisons with search algorithms

In this experiment, we compare our hyperparameter recommendation approaches to seven search algorithms: random search (RS) [9], heteroscedastic evolutionary Bayesian optimization (BO, HEBO) [10], [42], particle swarm optimization (PSO) [25], active testing (AT) [43], multi-task Gaussian process (MTGP) [44], and ranking-weighted Gaussian process ensemble (RGPE) [45]. The last three are meta/transfer learning-based methods.

To obtain fair comparisons, the maximum allowable search time assigned to search algorithms on a dataset should be the same as the time spent in the online recommendation phase of our approach, where the most costly step is the extraction of meta-features. However, we found that merely one search round could be executed on most testing datasets under this time constraint due to the time-

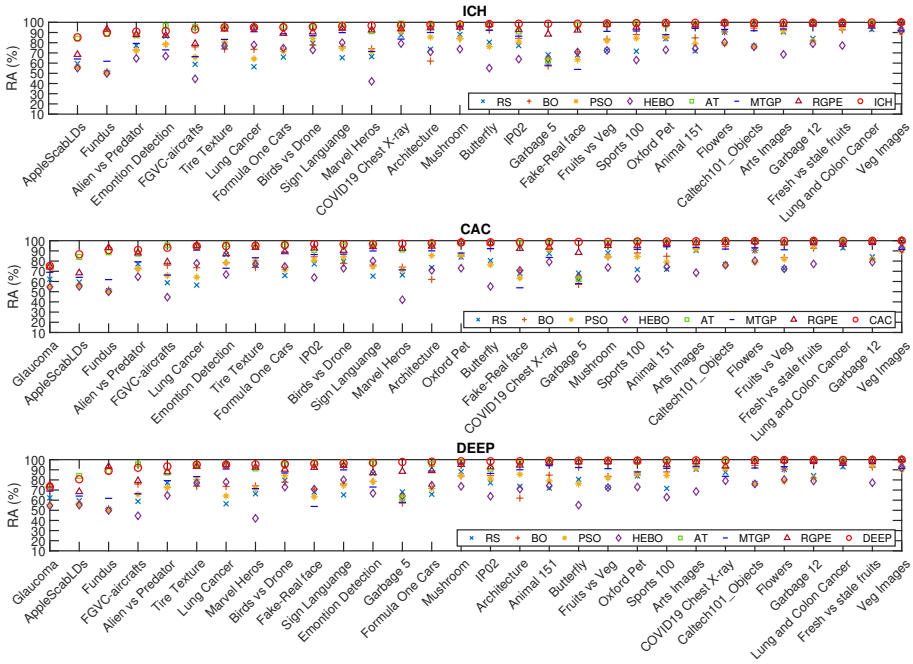

Fig. 5. Comparisons on RA (Recommendation Accuracy Rate) between our approaches and seven search-based algorithms on testing datasets.

consuming hyperparameter evaluations. Therefore, we permitted three hyperparameter configurations to search for all datasets in this experiment.

The empirical results are summarized in Table 3, where we report the average performance of the top-3 recommendations/searches of each baseline. Notably, we only consider the image size of 224×224 for our meta-features. Each search method, except Active Test (AT), was run 10 times independently, and their average performance is displayed.

From Table 3, one can see that: 1) our approaches outperform search algorithms by a significant margin, and obtain the highest ACA of 88.32% (CAC), the highest ARA of 96.32% (CAC), and the highest HR of 98.89% (CCV and COMB), and have the lowest ACA of 88.10% (HARA), the lowest ARA of 95.96% (CSD), and the lowest HR of 96.67% (CAC, HARA, and DEEP); 2) Among the seven search algorithms, Active Test (AT), Multi-Task Gaussian Process (MTGP), and Ranking-weighted Gaussian Process Ensemble (RGPE) outperform other four algorithms evidently, which underscore the effectiveness of meta-learning-based methods. However, it is worth noting that while AT performs well in terms of NDCG@1, its performance drops significantly for other NDCG values. This suggests that the effectiveness of the three searched hyperparameters is uneven; 3) On the other hand, Random Search (RS), Bayesian Optimization (BO), Heteroscedastic Evolutionary Bayesian Optimization (HEBO), and Particle Swarm Optimization (PSO) do not yield satisfactory outcomes. These algorithms struggle to discover promising configurations within the limited search rounds. Additionally, we present per-dataset comparisons (see Figure 5). For brevity, we focus on the results related to ICH, CAC, and DEEP meta-features using the Recommendation Accuracy Rate (RA) metric. Notably, our approaches consistently outperform or perform equivalently to the search algorithms across almost all datasets.

Finally, we employ Wilcoxon signed-rank tests [46] at a 0.05 significance level to verify if the superiority of our approaches over search algorithms is statistically significant. The test null hypothesis is that they have the same performance on average. The p-values of the pair-wise tests are summarized in Table 4, all of which are less than 0.05. Consequently, we reject the null hypothesis and accept the alternative hypothesis: the performance of our approaches outperforms the comparative baselines significantly.

TABLE 4
The p-values of the Wilcoxon signed-rank tests between our approaches and seven search-based algorithms. The null hypothesis is that each pair of baselines performs equivalently the same.

	RS	BO	PSO	HEBO	AT	MTGP	RGPE
ICH	0.000	0.000	0.000	0.000	0.012	0.000	0.000
CAC	0.000	0.000	0.000	0.000	0.001	0.000	0.000
HARA	0.000	0.000	0.000	0.000	0.015	0.000	0.000
CCV	0.000	0.000	0.000	0.000	0.021	0.000	0.000
CSD	0.000	0.000	0.000	0.000	0.025	0.000	0.000
DEEP	0.000	0.000	0.000	0.000	0.008	0.000	0.000
COMB	0.000	0.000	0.000	0.000	0.005	0.000	0.000

5.3 Performance in weight recommendations

In this subsection, we present the performance of our approaches related to weight initialization. To alleviate computational burdens, we devised a compact CNN model consisting of four consecutive convolutional blocks and a fully connected layer. The channel numbers within the four blocks are as follows: 32, 64, 128, and 256, respectively. Within each block, we incorporate three 3×3 convolutional layers, with each layer followed by ReLU activation and batch normalization. Additionally, a max pooling layer is applied at the end of each block. The entire model comprises 1,926,562 trainable parameters. For each of the historical datasets, we train the model on the training set using a fixed hyperparameter configuration: the learning rate is 0.0001,

TABLE 5
Comparisons on CA (Classification Accuracy Rate) between various weight initialization approaches. The average CAs (ACAs) of GlorotU, HeU, RandU, PreTrain, MAML, ICH, ICH_least, DEEP, DEEP_least, COMB, and COMB_least on the testing problems are 68.49%, 65.61%, 69.01%, 68.53%, 64.82%, 70.91%, 68.29%, 70.78%, 68.58%, 70.85%, and 69.05%, respectively. The datasets corresponding to dataset IDs can be found in the supplementary document.

Dataset ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
GlorotU	17.55	24.89	26.33	38.15	39.84	43.21	44.87	52.87	56.30	58.23	62.71	64.49	67.57	71.28	73.75
HeU	11.92	20.55	22.95	36.14	35.55	41.42	41.47	46.00	51.13	45.27	60.73	63.55	69.70	72.82	69.85
RandU	19.15	27.01	26.53	41.77	37.69	45.36	45.69	53.07	58.46	61.90	63.56	64.37	64.54	68.98	74.54
PreTrain	19.10	**32.16**	**31.53**	33.33	37.70	47.50	45.81	**57.27**	**64.54**	58.90	56.21	**65.67**	68.48	61.28	**78.58**
MAML	16.56	28.73	17.00	20.88	**40.50**	47.21	32.91	43.93	54.63	47.77	66.10	63.37	69.39	67.70	69.61
ICH	**26.16**	31.71	26.72	42.17	37.47	48.80	**50.00**	56.40	63.21	62.80	**68.64**	65.27	70.91	**73.85**	74.75
ICH_least	21.03	28.86	26.21	30.12	35.25	46.41	43.36	51.20	62.19	58.20	60.17	**65.67**	68.18	60.00	74.59
DEEP	**26.16**	29.00	26.72	42.17	37.92	**50.00**	**50.00**	56.40	60.71	**64.10**	**68.64**	64.82	**71.82**	**73.85**	74.75
DEEP_least	21.03	28.86	27.56	34.94	34.59	48.20	45.50	51.20	60.93	58.20	52.54	65.27	69.09	68.46	74.75
COMB	**26.16**	31.71	26.72	43.37	39.91	48.80	46.68	56.40	63.21	61.80	**68.64**	64.57	70.91	**73.85**	74.75
COMB_least	21.03	28.86	26.21	**45.78**	35.25	46.41	43.35	51.20	62.91	58.20	60.17	64.07	68.18	73.08	74.59

Dataset ID	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
GlorotU	73.95	77.29	78.10	79.00	79.76	81.16	84.62	86.42	90.95	92.05	94.17	98.02	98.84	98.96	**99.37**
HeU	68.44	72.11	74.06	75.67	77.02	79.23	82.48	85.47	90.58	92.66	89.50	96.84	98.23	98.18	98.73
RandU	77.24	73.71	80.33	78.67	79.35	83.09	85.13	84.43	91.18	95.11	94.00	98.06	99.05	**99.22**	99.21
PreTrain	72.18	69.98	68.20	77.33	**85.09**	82.13	**85.86**	85.47	**92.23**	96.33	87.83	98.07	99.29	98.96	98.99
MAML	67.29	65.21	71.27	77.00	74.35	71.98	83.53	78.16	91.83	92.66	91.33	97.60	98.43	98.44	99.36
ICH	77.33	**77.69**	**80.75**	81.00	79.81	85.51	**85.86**	**87.75**	91.39	92.66	93.00	97.97	**99.32**	**99.22**	99.12
ICH_least	73.07	73.31	75.31	83.00	83.43	78.26	84.23	85.19	90.88	**99.08**	**96.50**	**98.43**	98.98	98.44	99.24
DEEP	**77.60**	77.29	74.90	80.00	79.81	**89.86**	84.46	**87.75**	91.39	93.58	94.00	98.17	99.11	**99.22**	99.12
DEEP_least	73.07	74.90	75.31	**84.00**	83.50	82.61	**85.86**	87.18	91.45	91.74	92.00	**98.43**	98.64	98.44	99.16
COMB	77.33	**77.69**	**80.75**	81.00	79.81	84.06	84.46	**87.75**	91.73	94.50	93.00	98.17	**99.32**	**99.22**	99.32
COMB_least	73.07	73.31	75.31	83.00	83.43	82.61	84.23	85.19	91.68	95.41	90.00	**98.43**	98.98	98.44	99.24

the training epoch is 40, the optimizer is `Adam`, the batch size is 32, and no dropout. These settings ensure consistency across comparisons.

In this experiment, we leverage the nearest-neighbor (NN) recommendation algorithm in our methods. Specifically, for a given query dataset, we identify its most similar historical dataset, whose trained weights are then recommended for the query dataset. Subsequently, we train a CNN model on the training set of the query dataset and evaluate its performance on the validation set. Due to the infinite search space of model weights, we focus solely on classification accuracy rate (CA) as the evaluation metric. Other metrics, such as RA, HR, and NDCG, cannot be precisely defined in this context.

We selected five popular weight initialization strategies as comparative benchmarks: random uniform (RandU), He uniform (HeU) [18], Glorot uniform (GlorotU) [17], model pretraining (PreTrain), and MAML [16]. The implementation details of these methods can be found in the supplement. Additionally, we also compared the weights trained on the least similar historical dataset for each query dataset. According to our similarity assumptions, these weights are expected to perform worse than those generated from the most similar dataset. The empirical results are summarized in Table 5. To save space, our approaches focus on three meta-features: ICH, DEEP, and COMB. Note that ICH_least, DEEP_least, and COMB_least refer to the methods where the weights of the least similar dataset are utilized.

From Table 5, one can observe that: 1) our approaches exhibit similar performance, with an average classification accuracy rate (ACA) of 70.91% (ICH), 70.78% (DEEP), and 70.85% (COMB), respectively. Notably, our proposed weight recommendation approaches outperform the comparative baselines by a significant margin across most datasets (e.g., 1, 6, 7, 10, 11, 13, and 14), demonstrating their effectiveness. Within the five weight initialization strategies, RandU performs the best with an ACA of 69.01%, PreTrain ranks second with an ACA of 68.53%, and GlorotU ranks third with an ACA of 68.49%. It is worth mentioning that PreTrain exhibits best performance on several datasets (e.g., 2, 3, 8, 9, 12, 15, 20, and 24) but has unsatisfactory performance on many datasets (e.g., 1, 4, 10, 11, 14, 16, 17, and 18), suggesting its instability. MAML does not produce satisfying performance in most cases, with an ACA of 64.82%; 3) ICH_least, DEEP_least, and COMB_least have similar performance to RandU, HeU, and GlorotU on average. We noticed that they have higher CA than ICH, DEEP, and COMB on datasets 19, 20, and 27.

TABLE 6
The p-values of the Wilcoxon signed-rank tests between our approaches and comparative baselines. The null hypothesis is that each pair of baselines performs equivalently the same.

	RandU	HeU	GlorotU	PreTrain	MAML	$*_{\text{least}}$
ICH	0.000	0.000	0.000	0.045	0.000	0.002
DEEP	0.000	0.000	0.000	0.040	0.000	0.005
COMB	0.000	0.000	0.000	0.032	0.000	0.000

Similarly, we employ Wilcoxon signed-rank tests at a 0.05 significance level to verify if the superiority of our approaches over other weight initialization methods is statistically significant. The null hypothesis is that the two tested approaches have the same performance on average. The p-values of the pairwise tests are summarized in Table 6, where all p-values are less than 0.05. Consequently, the null hypothesis is rejected, and we can conclude that the performance of our approaches is significantly better than the comparative strategies of weight initialization.

6 Concluding Remarks

Hyperparameter tuning and weight initialization are critical components in the training of deep neural networks. In this work, we introduced a novel meta-learning-based approach that leverages dataset similarity to jointly recommend promising hyperparameters and initial weights. Central to our framework is a set of six meta-feature groups that capture color, texture, and latent characteristics of image datasets, enabling effective similarity measurements across diverse tasks. To evaluate the proposed method, we conducted comprehensive experiments on 105 real-world image classification datasets, employing both Vision Transformer (ViT) and Convolutional Neural Network (CNN) as backbone models. Empirical results, measured against multiple performance metrics and benchmarked against state-of-the-art baselines, consistently demonstrated the effectiveness and robustness of our framework in improving hyperparameter selection and weight initialization. Our analysis further highlights the strong performance of indexed color histograms and deep latent meta-features, with the former offering a compelling balance between simplicity and predictive power. These findings emphasize the potential of meta-feature-driven learning to improve deep learning workflows.

Looking ahead, our future work will explore the extension of this framework to the recommendation of neural architecture. In particular, defining a practical and expressive search space for architectures remains a fundamental and open challenge that we aim to address.

References

[1] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, "Review of image classification algorithms based on convolutional neural networks," *Remote Sensing*, vol. 13, no. 22, p. 4712, 2021.

[2] M. Malik, M. K. Malik, K. Mehmood, and I. Makhdoom, "Automatic speech recognition: A survey," *Multimedia Tools and Applications*, vol. 80, no. 6, pp. 9411–9457, 2021.

[3] L. Regenwetter, A. H. Nobari, and F. Ahmed, "Deep generative models in engineering design: A review," *Journal of Mechanical Design*, vol. 144, no. 7, p. 071704, 2022.

[4] Y. Goldberg, *Neural network methods for natural language processing*. Springer Nature, 2022.

[5] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, 2021.

[6] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," *arXiv preprint arXiv:2003.05689*, 2020.

[7] M. V. Narkhede, P. P. Bartakke, and M. S. Sutaone, "A review on weight initialization strategies for neural networks," *Artificial Intelligence Review*, vol. 55, no. 1, pp. 291–322, 2022.

[8] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5149–5169, 2021.

[9] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.

[10] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 2951–2959.

[11] X. Dong, J. Shen, W. Wang, L. Shao, H. Ling, and F. Porikli, "Dynamical hyperparameter optimization via deep reinforcement learning in tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1515–1529, 2019.

[12] J. Wu, S. Chen, and X. Liu, "Efficient hyperparameter optimization through model-based reinforcement learning," *Neurocomputing*, vol. 409, pp. 381–393, 2020.

[13] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1165–1173.

[14] P. Micaelli and A. J. Storkey, "Gradient-based hyperparameter optimization over long horizons," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10 798–10 809, 2021.

[15] T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor, "ImageNet-21k pretraining for the masses," *arXiv preprint arXiv:2104.10972*, 2021.

[16] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1126–1135.

[17] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on Imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[19] K. He, R. Girshick, and P. Dollár, "Rethinking ImageNet pre-training," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4918–4927.

[20] D. Hendrycks, K. Lee, and M. Mazeika, "Using pre-training can improve model robustness and uncertainty," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2712–2721.

[21] B. Zoph, G. Ghiasi, T.-Y. Lin, Y. Cui, H. Liu, E. D. Cubuk, and Q. Le, "Rethinking pre-training and self-training," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3833–3845, 2020.

[22] P. Agrawal, R. Girshick, and J. Malik, "Analyzing the performance of multilayer neural networks for object recognition," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VII 13*. Springer, 2014, pp. 329–344.

[23] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning*. Springer, 2019.

[24] E. Alcobaça, F. Siqueira, A. Rivolli, L. P. Garcia, J. T. Oliva, A. C. de Carvalho *et al.*, "MFE: Towards reproducible meta-feature extraction," *Journal of Machine Learning Research*, vol. 21, no. 111, pp. 1–5, 2020.

[25] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[26] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[27] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural Computation*, vol. 12, no. 8, pp. 1889–1900, 2000.

[28] Y. Gan, J. Liu, J. Dong, and G. Zhong, "A PCA-based convolutional network," *arXiv preprint arXiv:1505.03703*, 2015.

[29] T. L. Paine, P. Khorrami, W. Han, and T. S. Huang, "An analysis of unsupervised pre-training in light of recent advances," *arXiv preprint arXiv:1412.6597*, 2014.

[30] A. Ruiz-Garcia, M. Elshaw, A. Altahhan, and V. Palade, "Stacked deep convolutional auto-encoders for emotion recognition from facial expressions," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 1586–1593.

[31] B. S. Manjunath, J.-R. Ohm, V. V. Vasudevan, and A. Yamada, "Color and texture descriptors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 6, pp. 703–715, 2001.

[32] T. Löfstedt, P. Brynolfsson, T. Asklund, T. Nyholm, and A. Garpebring, "Gray-level invariant Haralick texture features," *PloS One*, vol. 14, no. 2, p. e0212110, 2019.

[33] R. M. Haralick, K. Shanmugam, and I. H. Dinstein, "Textural features for image classification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 3, no. 6, pp. 610–621, 1973.

[34] L.-K. Soh and C. Tsatsoulis, "Texture analysis of SAR sea ice imagery using gray level co-occurrence matrices," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 37, no. 2, pp. 780–795, 1999.

[35] D. A. Clausi, "An analysis of co-occurrence texture statistics as a function of grey level quantization," *Canadian Journal of Remote Sensing*, vol. 28, no. 1, pp. 45–62, 2002.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015*. Computational and Biological Learning Society, 2015, pp. 1–14.
[37] D. Dowson and B. Landau, "The Fréchet distance between multivariate normal distributions," *Journal of Multivariate Analysis*, vol. 12, no. 3, pp. 450–455, 1982.
[38] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.
[39] Q. Song, G. Wang, and C. Wang, "Automatic recommendation of classification algorithms based on data set characteristics," *Pattern Recognition*, vol. 45, no. 7, pp. 2672–2689, 2012.
[40] L. Deng and M. Xiao, "Latent feature learning via autoencoder training for automatic classification configuration recommendation," *Knowledge-Based Systems*, vol. 261, p. 110218, 2023.
[41] K. Järvelin and J. Kekäläinen, "IR evaluation methods for retrieving highly relevant documents," in *ACM SIGIR Forum*, vol. 51, no. 2. ACM New York, NY, USA, 2017, pp. 243–250.
[42] A. I. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. R. Griffiths, A. M. Maraval, H. Jianye, J. Wang, J. Peters *et al.*, "HEBO: Pushing the limits of sample-efficient hyper-parameter optimisation," *Journal of Artificial Intelligence Research*, vol. 74, pp. 1269–1349, 2022.
[43] R. Leite, P. Brazdil, and J. Vanschoren, "Selecting classification algorithms with active testing," in *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 2012, pp. 117–131.
[44] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task Bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 26, pp. 2004–2012, 2013.
[45] M. Feurer, B. Letham, F. Hutter, and E. Bakshy, "Practical transfer learning for Bayesian optimization," *arXiv preprint arXiv:1802.02219*, 2018.
[46] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

Liping Deng received the B.S. degree in Applied Mathematics from Anshan Normal University, Anshan, China, in 2016, the M.S. degree in Mathematics from Shenzhen University, Shenzhen, China, in 2019, and the Ph.D. degree in Mathematics from Southern Illinois University Carbondale, Carbondale, IL, USA, in 2023.

He is currently a Visiting Assistant Professor with the Department of Mathematics, University of California at Riverside, Riverside, CA, USA. His research interests include machine learning, meta-learning, deep learning, and pattern recognition.

Maziar Raissi is an Assistant Professor of Applied Mathematics at the University of California, Riverside. He earned his Ph.D. in Applied Mathematics, Statistics, and Scientific Computation from the University of Maryland, College Park, followed by postdoctoral research in the Division of Applied Mathematics at Brown University. Dr. Raissi subsequently worked as a Senior Software Engineer at NVIDIA in Silicon Valley before transitioning to academia as an Assistant Professor of Applied Mathematics at the University of Colorado Boulder. His research expertise lies at the intersection of probabilistic machine learning, deep learning, and data-driven scientific computing. He is particularly focused on developing learning algorithms that integrate physical laws and governing equations to identify patterns in high-dimensional experimental data.

MingQing Xiao received his Ph.D. in Mathematics from the University of Illinois at Urbana–Champaign in 1997. He is a Professor in the School of Mathematical and Statistical Sciences at Southern Illinois University Carbondale. He has served on the editorial boards of leading journals, including IEEE Transactions on Automatic Control, Automatica, and the European Journal of Control. His current research interests include big data theory, optimization, and computational science.