IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 4, 2021, accepted February 14, 2021, date of publication February 26, 2021, date of current version March 31, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3062366

ETM: Effective Tuning Method Based on Multi-Objective and Knowledge Transfer in Image Recognition

WEICHUN LIU AND CHENGLIN ZHAO
College of Information Engineering, Shaoyang University, Shaoyang 422000, China
Corresponding author: Chenglin Zhao (chenglin_zhao_sy@163.com)

This work was supported in part by the General Project of Education Department of Hunan Province under Grant 19C1646, and in part by the Key Project of Education Department of Hunan Province under Grant 19A446.

ABSTRACT With the widespread application of machine learning and deep learning, image recognition has been continuously developed. However, there are still huge challenges in the use of machine learning and deep learning. The tuning processes of algorithms are critical and challenging for their performance. Although there have been many previous works to improve the final accuracy of the recognition algorithms through tuning, these works cannot consider some indicators that are also very important in the actual environment (such as latency, central processing unit (cpu) utilization) in the tuning. In this paper, we propose an effective tuning method based on multi-objective and knowledge transfer, which is solved the above limitations in the image recognition. Specifically, we first use an agent to automatically tune the recognition algorithms, and combine the prediction accuracy and the running latency of each episode as a multi-objective reward signal to guide the update of the internal parameters of the agent. In this way, the agent can continuously select the better algorithm configuration to improve prediction performance. In addition, we improve the efficiency of the above tuning process by transferring knowledge. To do that, we can learn the meta parameters from other small-scale tasks to initialize the agent. In the experiments, we apply the proposed method to tune the eXtreme Gradient Boosting and random forest on 57 image recognition tasks and convolutional neural network on 2 tasks. The experimental results verify that the proposed method achieves average accuracy rankings of 1.92, 1.42 and 1.71 on three algorithms to be optimized, respectively. Especially in terms of latency performance, the proposed method performs best on all the tasks (57 data sets) on the three algorithms to be optimized. In addition, we verify the various components of the proposed method through ablation experiments.

INDEX TERMS Image recognition, machine learning, deep learning, tuning, multi-objective, knowledge transfer.

I. INTRODUCTION

So far, machine learning and deep learning has made great progress in many works on the image recognition field [1]–[3]. However, machine learning and deep learning still need many tedious processes in practical applications. The tedious processes include data processing, feature engineering, algorithm selection, hyperparameter optimization and data analysis. Among them, hyperparameter tuning is a particularly important part for the performance of the predictive algorithms, where the hyperparameters refer to the parameters set manually before the training model. In this paper, we mainly solve the hyperparameter optimization problem (HPO problem) to tune the prediction algorithm, so as to improve the prediction performance of the algorithm.

For complex algorithms, the tuning is often a time-consuming and tedious process, which prevents researchers from focusing on the problem that needs to be solved. To solve the above limitation, automatic HPO methods are proposed and used in various fields. This automatic HPO methods automatically select hyperparameter configuration with as little human intervention as possible, and gradually select the optimal hyperparameter configuration by trial and error in the preset ranges ([4]). Subsequently, the idea of

The associate editor coordinating the review of this manuscript and approving it for publication was Ioannis Schizas.

automation is extended to the problem of algorithm selection combined with hyperparameter tuning ([5]). In the field of image recognition, an efficient hyperparameter tuning method can achieve the following goals:

- it greatly reduces the threshold for the use of machine learning and deep learning models, which makes the application of these technologies more popular;
- for researchers, it can pay more attention to the modeling process of problems in specific scenarios, rather than model tuning process;
- compared with traditional manual tuning methods, it can greatly improve optimization efficiency and the prediction performance of model.

The hyperparameter tuning problem of the algorithm is essentially an optimization problem, and its optimization objective is to make the algorithm achieve the best prediction performance by selecting the hyperparameter configuration. However, this optimization problem cannot be solved directly and efficiently due to the following reasons:

- First of all, it is not clear at present the clear functional relationship between the selection of hyperparameters and the performance of the prediction algorithm in different scenarios, so it is not possible to directly perform gradient descent based on the optimization objective to obtain the optimal solution.
- Second, the tuning of each algorithm is a process of constant trial and error, which means that the tuning process needs to be explored in the preset range of each hyperparameter. Obviously, the search space is high-dimensional and as the number of hyperparameters increases exponentially, which makes the entire tuning process very complicated and inefficient.
- Finally, in order to make the prediction performance of the model better, the structure of the model will become very complicated. Importantly, the above situation is very unfavorable for deploying the model on an actual application platform.

To solve the above limitations, many advanced works have been proposed so far. In the algorithm tuning community in the field of image recognition, advanced works mainly includes two categories: tuning algorithms and tuning tools. Tuning algorithms can be roughly divided into basic search methods and sampling-based methods. The typical representatives of basic search methods are grid search and random search, while sampling-based methods mainly include bayesian optimization methods, evolutionary optimization methods, and optimization methods based on reinforcement learning. Tuning tools usually focus on the actual user experience (convenience and flexibility). However, although previous works have proved that the above tuning algorithms and tuning tools can perform well in image recognition tasks, they often only consider the predictive performance of the model and does not pay attention to the indicators (such as latency) of the model in the actual environment. Moreover, most of the previous works cannot carry out the transfer of experience, which actually waste a wealth of tuning knowledge.

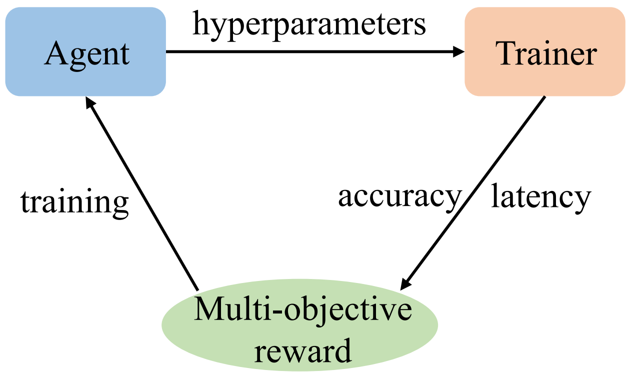


FIGURE 1. The framework of multi-objective optimization.

In this paper, we propose an effective tuning method (ETM) based on multi-objective and knowledge transfer. This method employ an agent to automatically tune the hyperparameters of the recognition algorithms in preset ranges, and combine the prediction accuracy and the running latency of each episode as a multi-objective reward signal to guide the update of the internal parameters of the agent (as shown in Figure 1). In this way, the recognition algorithms can achieve high prediction performance and low actual running latency. In addition, with the development of machine learning, the proposed method can be used for hyperparameter optimization of traditional models, such as prediction tasks and classification tasks.

For the algorithms in the image recognition field, we consider both accuracy and latency to achieve multi-objective optimization. This idea is inspired by the observation: the model has higher predictive performance but may has lower latency. Therefore, we should optimize the prediction performance and latency of the algorithms by hyperparameters tuning. In addition, since image recognition algorithms or models often are deployed in actual environments with resource constraints, they need to meet some specific indicators (such as response time (RT)). In this paper, multi-objective optimization considering predictive performance and latency is feasible and practical.

To further improve the efficiency of the above tuning, this paper uses previous optimization experience to transfer knowledge. Specifically, we perform meta-learning algorithms (model-agnostic meta-learning: MAML [6]) on a number of small-scale tasks to obtain the agent's optimization experience, which represents the agent's meta-parameters and is often used to initialize the agent's internal parameters. In this way, an agent can quickly adapt to new tasks. In the experiments, the proposed method is employed to optimize the hyperparameters of eXtreme Gradient Boosting (XGBoost) [7] and random forest on 57 datasets and convolutional neural network on 2 datasets. In this paper, we focus on the HPO problem in algorithm tuning process. Our main contributions are as follows:

- To solve the problem that the optimized objective function is not clear, we use an agent to automatically select each hyperparameter, and obtain the reward value signal through training and update the agent with reinforcement learning algorithm. In this way, we can get an agent with good decision making.
- Compared with the traditional tuning method, the proposed method can take the prediction accuracy and the running latency as the tuning objective. Importantly, we design an aggregation function that skillfully combines multi-objective optimization with agent updating so that agent decisions can be trade-off accuracy and latency.
- To improve the tuning efficiency, we extend the idea of knowledge transfer to the process of hyperparameters tuning. Specifically, we gain an agent optimization experience (i.e. meta parameters) by performing meta-learning on multiple small tuning tasks and the agent is initialized with the meta-parameters.
- The proposed method is compared with other tuning methods on multiple tuning tasks of image recognition field. The experimental results show that the proposed method is feasible and efficient. Moreover, the effectiveness of each component is verified by ablation experiments.

The remaining of this paper will describe in detail the related work, the specific design and process of the multi-objective tuning method, information on how to use meta-learning to transfer knowledge, experimental results and a conclusion.

II. RELATED WORK

A. MULTI-OBJECTIVE OPTIMIZATION

Multi-objective optimization is an improvement on the basis of single-objective optimization. Most single-objective optimization methods are based on reinforcement learning and optimize the objective continuously by taking the feedback value of the objective as the reward value signal. The single-objective optimization based on reinforcement learning are modeled by an single-objective markov decision process (MDP). The MDP is formed by an agent interacting with the environment and usually expressed as a 5-tuple, which includes a state set S, an action set A, a transition probability function P, a reward function P, and a discount coefficient γ. The state set S mainly includes all the states s that the environment can be in; the action set A includes all the actions a that the environment can execute; the transition probability function represents the probability function of the transition from one state to the next state; the reward function represents the feedback value of the agent's decision; the discount coefficient represents the confidence of the previous actions. During the MDP, the goal of an agent is to obtain a trajectory τ that maximizes the expected reward value, which is formally expressed as follows:

$$J(\pi) = E_{\tau\sim\pi}[R(\tau)] \tag{1}$$

where the trajectory τ is formed by the interaction between the agent and the environment and includes the action, state and reward value of multiple time steps, that is $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$; the expected reward value represents the weighted sum of the reward values of each time step in the trajectory, i.e. $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$.

The methods to solve the single-objective optimization based on reinforcement learning can be roughly divided into value-based optimization and policy-based optimization. The value-based optimization methods first need to calculate the expected reward value of the trajectory and takes it as the value of the state-action pair, that is

$$Q_\pi(s, a) = E_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | S_0 = s, A_0 = a] \tag{2}$$

To solve the single-objective optimization problem, the agent needs the optimal decision policy to maximize the value of the state-action pair, that is $\pi^*(s) \in argmax_a Q^*(s, a)$, where $Q^*(s, a)$ denotes the optimal state-action value. Q-learning [8] is a classical value-based optimization method, which indirectly obtains the optimal decision policy by continuously maximizing the expected reward value of the trajectory. This method satisfies the basic identity of Bellman equation, that is

$$Q^*(s, a) = E_\pi[r + \gamma max_{a'} Q^*(s', a') | S_0 = s, A_0 = a] \tag{3}$$

An important defect of value-based optimization methods similar to Q-learning is the curse of dimension, which makes these methods become very difficult or even ineffective in solving the optimization problem of continuous values. However, another policy-based optimization method can easily address the above limitation. One well-known method is the policy gradient method, which does not require the agent to learn how to maximize the expected reward value but directly optimizes the policy to improve the probability of the optimal action, that is

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}[\nabla_\theta logP(\tau|\theta) R(\tau)] \tag{4}$$

Even so, policy-based optimization methods such as policy gradient have the disadvantage of training instability. Usually in practice, some effective tricks are used to reduce the training variance of the optimization method, such as adding a baseline function and assigning suitable credit. More recently, some advanced research works [9], [10] have been proposed to combine value-based and policy-based optimization methods to achieve complementary advantages.

Based on the single-objective optimization method, the multi-objective optimization method is usually modeled as a multi-objective MDP, which is also represented by a 5-tuple. Different from the single-objective optimization method, the reward value signal is a vector composed of the feedback values of multiple objectives rather than a scalar reward, i.e. $\boldsymbol{r} \in \mathbb{R}^n$. In the actual optimization process, an important challenge of multi-objective optimization method is to find a Pareto optimal solution to trade-off each optimization objective. Since we cannot find an optimal solution to satisfy multiple objectives, we often need to customize an aggregation function to aggregate multiple reward value signals into a scalar value. The aggregation functions can be roughly divided into linear and nonlinear types. The method of weighted sum is a typical linear setting method [11], while the exponential weighting method is a nonlinear one [12]. In this paper, we implement multi-objective optimization by customizing a nonlinear aggregation function.

Up to now, multi-objective optimization methods have made great progress in algorithm tuning, which are mainly used to solve the neural architecture search (NAS) problem in image recognition field. In this paper, we mainly focus on the hyperparameter optimization problem in the algorithm tuning, and strive to realize the performance improvement of the model in many aspects through the hyperparameter tuning.

B. HYPERPARAMETER OPTIMIZATION

Hyperparameter optimization is a part of the algorithm tuning pipeline. The purpose of hyperparameter optimization is to improve the predictive performance of the algorithm by tuning its hyperparameters. To make the hyperparameter optimization method clearer, we first define the commonly used symbols in the hyperparameter optimization:

- $\mathcal{A}$ denotes the algorithm to be tuned;
- $\boldsymbol{\Lambda}$ is the hyperparameter search space of the algorithm to be tuned, which is a high-dimensional and needs to be preset;
- n is the number of hyperparameters to be optimized;
- $\boldsymbol{\lambda}$ denotes the hyperparameter configuration selected by the optimization method, which is represented by a vector composed of n hyperparameter values;
- $\boldsymbol{\lambda}^*$ denotes the optimal hyperparameter configuration;
- $\mathcal{A}_{\boldsymbol{\lambda}}$ represents the algorithm to be optimized that sets the selected hyperparameter configuration;
- $\boldsymbol{D_{train}}$ and $\boldsymbol{D_{valid}}$ represents the training set and the verification set of the target task respectively.

When given a target task, the formal expression of hyperparameter optimization is:

$$\boldsymbol{\lambda}^* = \max_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} E_{(\boldsymbol{D}_{train},\boldsymbol{D}_{valid})\sim\boldsymbol{D}} L(\mathcal{A}_{\boldsymbol{\lambda}}, \boldsymbol{D}_{train}, \boldsymbol{D}_{valid}) \tag{5}$$

where $L(\mathcal{A}_{\boldsymbol{\lambda}}, \boldsymbol{D}_{train}, \boldsymbol{D}_{valid})$ denotes the validation performance of $\mathcal{A}_{\boldsymbol{\lambda}}$ on the target task.

Generally, grid search [13] or random search [4] are widely used for the optimization tasks with small search spaces. Grid search is the simplest hyperparameter tuning methods, and its main idea search the optimal solution by traversing all the combinations of hyperparameters. Obviously, the grid search suffers from the curse of dimensionality, so its optimization process will consume a lot of time when faced with complex tasks. Random search uses random policy instead of traversing all combinations, which mainly idea is to perform hyperparameter tuning by sampling randomly on all possible combinations. Some experiments demonstrate that random search is better than grid search when some hyperparameters are much more crucial than others [4]. Moreover, random search has the advantages of parallelization and flexibility. However, random search cannot achieve the optimal optimization results due to the lack of policy guidance.

Bayesian optimization (BO) is a method cluster, which includes a series of powerful hyperparameter tuning methods. The main idea of bayesian optimization is to use a specific model to fit the functional relationship between hyperparameter configuration and its performance, and to use the acquisition function to obtain the next potential hyperparameter configuration based on the functional relationship. The bayesian optimization methods consist of surrogate model and acquisition function. The specific process of the bayesian optimization methods is as follows: firstly, the most potential hyperparameter configuration is obtained by sampling of the acquisition function; then the performance evaluation is carried out on the target task; and finally the functional relationship is fitted by training the surrogate model on all samples. After many iterations, the acquisition function can finally choose a better hyperparameter configuration. Since the efficiency and accuracy of the surrogate model are important, most of the previous works focused on how to select the surrogate model. At present, two popular surrogate models are Gaussian process and tree model. Spearmint [14] is a bayesian optimization method using Gaussian process as the surrogate model, which is an advanced method for low-dimensional optimization search space. The two disadvantages of using a Gaussian process are time-consuming (cubic time complexity) and poor scalability. The sequential model-based algorithm configuration (SMAC) [15] and the tree Parzen estimators (TPE) [13] are bayesian optimization methods using random forests and a tree of Parzen estimators as the surrogate models respectively. Many studies have shown that bayesian-based optimization methods can achieve higher optimization results [13]. However, the tuning process is inefficient when solving large-scale optimization tasks.

Population-based tuning approach are another competitive HPO methods, which can be roughly divided into genetic algorithms and evolutionary algorithms. The main idea of population-based optimization method is to preserve a series of populations and make them evolve through hybridization and mutation. The covariance matrix adaption evolutionary strategy (CMA-ES [16]) is an improved algorithm based on evolutionary algorithm, which samples configurations from a multivariate Gaussian distribution. More recently, CMA-ES has proved to be a powerful black-box optimization method and is superior to advanced Bayesian methods [17].

The bandit-based methods have been proposed to solve HPO problem recently, such as hyperband [18] and BOHB [19]. Hyperband method uses the idea of the successive halving to allocate resources to each hyperparameter configuration. The researches show that hyperband has a strong performance during the tuning process of the deep learning model. However, because random policy is used for sampling, the optimization efficiency of hyperband is general. To solve the above limitations, a method combining Bayesian optimization and bandit-based is proposed, which is called BOHB [19]. This method has high efficiency at the beginning and has good performance in the long run.

The above mainly describes the single-objective optimization methods. Thus far, multi-objective optimization methods are mainly focus on NAS problems [20]–[22]. The main reason is that the neural network architecture has a great impact on the actual running indicators. For example, due to the limitations of hardware devices and application scenarios, indicators such as computational complexity and resource consumption also need to be optimized. The progress of these multi-objective optimization methods in NAS enables the network architecture to effectively adapt to the actual environment. Therefore, we believe that machine learning models that are widely used in many fields should also be studied in the multi-objectives optimization (response time or resource consumption), so that machine learning model can better adapt to the actual environment in addition to good predictive performance.

C. KNOWLEDGE TRANSFER

Knowledge transfer is an important area of research in the image recognition community. One method of knowledge transfer is transfer learning; for example, [23] uses pretrained weights and data to improve natural language processing (NLP) models. In algorithm tuning, [24] uses transfer learning to learn a generalizable framework that can speed up the search for new tasks. Another important and recent method is meta-learning or learning-to-learn, which has recently received interest [6], [25], [26]. The training of meta-learning is mainly divided into two steps: collecting meta-data of historical learning tasks or previously learned models; extracting useful knowledge from meta-data to guide the completion of new tasks. Meta-data includes hyperparameter configuration, neural network architecture, model evaluation results, model internal parameters, and task attributes (meta-features). Meta-learning can be divided into three categories: meta-representation, meta-objective and meta-optimizer. The meta-objective defines the goal of the meta-learning by selecting meta-objectives and the associated data flow between inner loop events and external optimization. The meta-Optimizer represents the choice for the outer optimizer during meta training. The outer optimizer can take various forms such as gradient descent, reinforcement learning, and evolutionary search. The meta-representation explains what the representation of learning should be. Generally, representations include hyperparameters, network structure, and initial weights.

Meta-learning achieves the goal of fast adaptation to new tasks by learning from other tasks. Auto-sklearn is an advanced tuning tool that applies meta-learning to select a configuration that is likely to perform well on a new task.

III. HYPERPARAMETER TUNING BASED ON MULTI-OBJECTIVE OPTIMIZATION

In this section, we will describe in detail hyperparameter tuning based on multi-objective optimization. First, we illustrate the property of sequential decision making in the HPO problem. Then, the HPO is extended to the multi-objective Markov decision process. Finally, we will introduce the design of the agent and multi-objective optimization in detail.

A. SEQUENTIAL DECISION MAKING IN HPO

For traditional hyperparameter optimization methods, they directly choose a hyperparameter configuration in the preset high-dimensional search space. If the model to be optimized is very complex, the search space of the task will become very large and grow exponentially with the number of hyperparameters. In order to solve the above problems, we consider that there is a natural sequential decision process for the HPO problem. The intuition behind the way to solve the HPO problem is: Any complex high-dimensional action can be selected incrementally, component by component, where each component's probability also depends on components already selected earlier [27]. Specifically, the main idea of the sequential decision process in HPO is: hyperparameters are selected sequentially, and the selection of hyperparameter depends on the selection of previous hyperparameters.

To further illustrate the advantages of the sequential decision process in HPO, we will be compared with the traditional method of directly selecting hyperparameter configuration in high-dimensional spaces. We assume that the model to be optimized has n hyperparameters to be optimized. In each iteration, the traditional optimization method selects a hyperparameter configuration in the search space, where the size of search space is $\boldsymbol{\Lambda} = \boldsymbol{\Lambda}_1 \times \boldsymbol{\Lambda}_2 \times \ldots \boldsymbol{\Lambda}_n$ ($\times$ denotes the Cartesian product; $\boldsymbol{\Lambda}_i$ denote the search space of the i-th hyperparameter). In the case of sequential decision making, hyperparameters are selected sequentially to form the configuration. In this method, each iteration contains n selections, and each selection needs to be conducted in the search space of the corresponding hyperparameter, so the size of the search space is $\boldsymbol{\Lambda}' = \boldsymbol{\Lambda}_1 \cup \boldsymbol{\Lambda}_2 \cup \ldots \boldsymbol{\Lambda}_n$. Obviously, sequential decision making not only reduces the difficulty of tuning but also improve the efficiency of optimization.

In the process of sequential decision making, in addition to sequential selection of hyperparameters, we should also consider the interrelation of hyperparameters selection. In this paper, we use a memorized network structure for implicit association and set the current hyperparameter selection to be dependent on the previous hyperparameter selection for display association.

B. MULTI-OBJECTIVE MARKOV DECISION PROCESS

Based on the formulation of the above sequential decision making, we further defined the HPO problem as a multi-objective Markov decision process. With such the definition, the workflow of using agent to solve HPO problems can be clearly described. First of all, we define the 5-tuple $\langle S, A, P, R, \gamma \rangle$ of the multi-objective Markov decision process in the HPO problem, which are as follows:

- A is a set of all the actions that an environment can perform, that is, the set of all the hyperparameters that the algorithm needs to tune. At each time-step t, the action $a_t = \lambda_t$, and the search space of the action is Λ_t. After n time-steps, the agent can selects n hyperparameters, i.e. $\boldsymbol{\lambda} = a_{1:n}$.
- S is a finite set of states, which includes all the states the environment can be in. For the HPO problem, the environment that interacts with an agent is composed of a dynamic part and a static part, where the algorithm to be optimized and the target task are the static part, and the hyperparameters are the dynamic part. In this paper, we only consider the dynamic part. Specifically, we take the hyperparameter distribution at time $t-1$ as the state of the environment at time t, i.e. $s_t = \mathcal{D}(\lambda_{t-1})$, where the hyperparameter distribution is output by the agent.
- R is the reward function. In multi-objective MDP, the reward value signal is composed of the feedback values of multiple objective. In this paper, we take the accuracy and latency as optimization objectives. Therefore, the vector consisting of the accuracy and latency will be used as a reward value signal. Specifically, $r_t = [0, 0]$ for $t \in [1, n)$ and $r_n = [accuracy, latency]$, where *accuracy* denotes the validation performance of $\mathcal{A}_{\boldsymbol{\lambda}=a_{1:n}}$, *latency* is the latency of $\mathcal{A}_{\boldsymbol{\lambda}=a_{1:n}}$.
- $P : S \times A \rightarrow \mathcal{P}(S)$ is a state transition probability function. We usually do not know the state transition of the environment, otherwise the model-base method will easily solve the problem.
- γ is a discount factor.

As shown in Figure 1, the overall framework consists of three components: an agent to select a hyperparameter configuration, a trainer to obtain the model accuracy and latency with the selected configuration, and multi-objective rewards including accuracy and latency. The multi-objective MDP as follows: for a given task, the agent selects n hyperparameters one by one based on its previous decisions. Then, the machine learning model with the selected hyperparameters is trained on a training set $\boldsymbol{D}_{train}$. The accuracy and latency of a validation set $\boldsymbol{D}_{valid}$ are used as reward signals to update the parameters of the agent by an reinforcement learning algorithm. As a result, the agent learns how to tune hyperparameters over time.

C. DESIGN OF THE AGENT

The agent consists of an input embedding layer, an output embedding layer and a long short-term memory (LSTM) [28], which is the core part of the agent. Specifically, the input state s_t is converted to a high-dimensional representation by an input embedding layer, which allows the agent to better observe the state representation. The output of the input embedding layer is then fed to the core network consisting of three layers of a LSTM. Although it is difficult to train the LSTM network, the LSTM cell has been indicated to be a powerful structure in solving the sequential problem. Finally, the output of the LSTM is converted to a low-dimensional representation by an output embedding layer. The output of the agent is not a hyperparameter value but rather a distribution of the possible values. Following [10], [29], we use the normal distribution to represent the distribution of a hyperparameter (λ_t).

Thus, the output of the agent at t is $\mathcal{N}(\mu_t, \sigma_t)$, and $s_t = \mathcal{N}(\mu_{t-1}, \sigma_{t-1})$, $s_1 = \mathcal{N}(0, 1)$. As described above, the design and workflow of the agent match the sequential decision process very well.

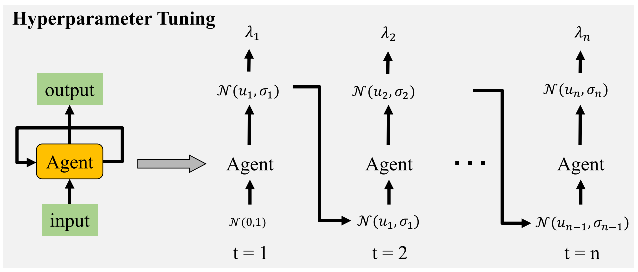

Hyperparameter Tuning
output
Agent
input
λ_1
$\mathcal{N}(u_1, \sigma_1)$
Agent
$\mathcal{N}(0,1)$
t = 1
λ_2
$\mathcal{N}(u_2, \sigma_2)$
Agent
$\mathcal{N}(u_1, \sigma_1)$
t = 2
λ_n
$\mathcal{N}(u_n, \sigma_n)$
Agent
$\mathcal{N}(u_{n-1}, \sigma_{n-1})$
t = n

FIGURE 2. The workflow in which agent selects hyperparameters sequentially.

D. SAMPLING FOR A HYPERPARAMETER

From the above description, it can be seen that the output of the agent is a distribution of possible values of a hyperparameter. Therefore, we need to get the actual hyperparameter value by sampling. A simple sampling method is random sampling. However, due to the significant difference in the preset range of each hyperparameter, random sampling within the preset range of the hyperparameter will make the training of agent very unstable and even ineffective. To solve the above problems, we customize a transformation method to scale the distribution of the hyperparameters. The transformation process is as follows:

- Scale the mean of the distribution μ to μ' by the *tanh* function in the range $(-1, 1)$;
- Sample a value z from the new distribution $\mathcal{N}(\mu', \sigma)$;
- Scale z into the range of hyperparameter $[z_L, z_U]$ by the following method:

$$z' = z_L + (z_L - z_U) \times (1 + z)/2 \tag{6}$$

$$\lambda = clip\_and\_convert(z', z_L, z_U) \tag{7}$$

where z_U and z_L represent the upper and lower bounds, respectively. The *clip_and_convert* function can limit the sampling value z' within the preset range by clipping and make the hyperparameter meet the type requirement by type conversion.

E. MULTI-OBJECTIVE OPTIMIZATION

The internal parameters θ of the agent represent a policy π that can decide which action to choose based on the current

Algorithm 1 Meta-Learning on HPO Tasks

```
Input:
    $\theta$: meta parameters;
    $\alpha, \beta$: step size.
Procedure:
1:  randomly initialize $\theta$
2:  while not done do
3:      Sample a batch of HPO tasks $\mathcal{T}_i$ from source datasets
4:      for all $\mathcal{T}_i$ do
5:          Sample a trajectory using $\pi_\theta$ in $\mathcal{T}_i$: $\tau_i = (s_1, a_1, A_1, LAT_1 \ldots, s_n, a_n, A_n, LAT_n)$
6:          $\theta_i' = \theta - \alpha \nabla_\theta L_{\mathcal{T}_i}(\pi_\theta)$ w.r.t $\tau_i$ and $L_{\mathcal{T}_i}$ defined in Equation (5)
7:          Sample a trajectory using $\pi_{\theta_i'}$ in $\mathcal{T}_i$: $\tau_i' = (s_1, a_1, A_1, LAT_1 \ldots, s_n, a_n, A_n, LAT_n)$
8:      end for
9:      Update $\theta \leftarrow \theta - \beta \nabla_\theta \Sigma_{\mathcal{T}_i} L_{\mathcal{T}_i}(\pi_{\theta_i'})$ using each $\tau_i'$ and $L_{\mathcal{T}_i}$ defined in Equation (5)
10: end while
```

state of the environment. Follow the previous works [30], [31], we use the PPO-clip method [10] to update θ. Compared with the policy gradient method [32], the PPO-clip method implements off-policy based on the important sampling and uses KL divergence to constrain the gradient step, so as to achieve a good training efficiency and stability. The objective function of the PPO-clip method is defined as:

$$\theta_{k+1} = arg \max_{\theta} \mathbb{E}_{s,a \sim \pi_{\theta_k}} [L(s, a, \theta_k, \theta)] \tag{8}$$

where L is given by:

$$\begin{aligned} L(s, a, \theta_k, \theta) = min(&\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \\ &clip(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_k}}(s, a)) \end{aligned} \tag{9}$$

where ϵ is a hyperparameter that controls the change to the new policy θ from the old policy θ_k, $\epsilon = 0.2$. For single-objective RL, the advantage function is defined as $A^{\pi_{\theta_k}} = R(\tau_k) - b$, where the return $R(\tau_k) = \sum_{t=1}^{n} r_t$ is the cumulative reward over the kth sample, and b is an exponential moving average of the returns of the previous samples.

We design an aggregation function that combines the accuracy and latency as a reward signal to achieve the multi-objective optimization. Let $L(s, a, \theta_k, \theta)$ incorporate the latency and be redefined as:

$$\begin{aligned} L(s, a, \theta_k, \theta) = min(&\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \times f_{scalar}, \\ &clip(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon) \times f_{scalar}) \end{aligned} \tag{10}$$

$$f_{scalar} = A^{\pi_{\theta_k}}(s, a) \times \left[\frac{LAT_k}{T}\right]^w \tag{11}$$

where LAT_k denotes the inference latency of the kth sample on the target task, and T is the minimum latency of all configurations searched so far.

We use a customized weighted product method to define the aggregation function. Here, w is the weight factor defined as:

$$w = \begin{cases} \alpha, & \text{if } LAT_k \leq T \\ \beta, & \text{otherwise} \end{cases} \tag{12}$$

where α and β are application-specific constants, where $\alpha \geq 0$ and $\beta < 0$. In fact, we can achieve the accuracy-latency trade-off by tuning the two constants. An empirical rule for determining α and β values is to softly adjust the advantage value $A^{\pi_{\theta_k}}$ by considering the sign of the value. If $A^{\pi_{\theta_k}}$ is positive and $LAT_k \leq T$, which means the selected configuration can achieve high accuracy and uses less inference latency, w is set to a negative value to increase the value of f_{scalar}; otherwise, if $A^{\pi_{\theta_k}}$ is negative and $LAT_k \leq T$, w is set a positive value to increase the value of f_{scalar}, since even $A^{\pi_{\theta_k}}$ is negative, the constraint of latency is satisfied, and this configuration is not too bad. Specifically, we further illustrate the motivation of setting the weight w (α and β) by analyzing the following four cases ($A^{\pi_{\theta_k}}$ is referred to as A for simplicity in the following):

- Case 1: $A \geq 0, LAT_k \leq T$ is the best case, that is, the high accuracy and the low latency. Therefore, we should set w (α) to a negative value to increase the positive advantage value of the action a.
- Case 2: $A \geq 0, LAT_k > T$ is suboptimal case, that is, the accuracy objective is met and the latency objective is ignored. We should set w (β) to a negative value to reduce the original advantage value, thereby reducing the positive effect of the action a.
- Case 3: $A < 0, LAT_k \leq T$ is suboptimal case, that is, the latency objective is met and the accuracy objective is ignored. We should set w (α) to a positive value to increase the original advantage value, thereby reducing the negative effect of the action a.
- Case 4: $A < 0, LAT_k > T$ is the worst case, that is, neither the accuracy objective nor the latency objective is met. We should set w (β) to a positive value to reduce the original advantage value, thereby increasing the negative effect of the action a.

We consider two ways to set values of α and β, hard constraint and soft constraint. If $\alpha = 0$ and $\beta = -1$, we obtain a hard constraint. When $A^{\pi_{\theta_k}}$ is positive and $LAT_k \leq T$, we simply use $A^{\pi_{\theta_k}}$ as the advantage value; otherwise, we sharply penalize the advantage value to discourage models from violating latency constraints. In our experiments, we use a soft constraint that smoothly adjusts the advantage value by setting $\alpha = -0.07$ and $\beta = -0.07$ if $A^{\pi_{\theta_k}}$ is positive; otherwise $\alpha = 0.07$ and $\beta = 0.07$.

IV. KNOWLEDGE TRANSFER IN HPO

For traditional tuning methods, they ignore previous experience of optimizing tasks, which means that each new task is

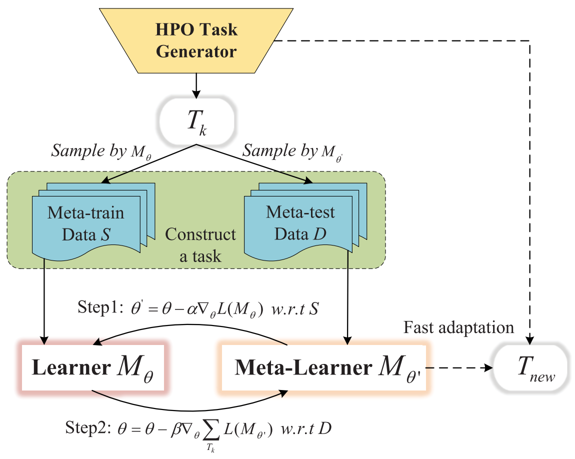

HPO Task Generator
T_k
Sample by M_θ
Sample by $M_{\theta'}$
Meta-train Data S
Construct a task
Meta-test Data D
Step1: $\theta' = \theta - \alpha\nabla_\theta L(M_\theta)$ w.r.t S
Fast adaptation
Learner M_θ
Meta-Learner $M_{\theta'}$
T_{new}
Step2: $\theta = \theta - \beta\nabla_\theta \sum_{T_k} L(M_{\theta'})$ w.r.t D

FIGURE 3. The workflow of meta-learning in the HPO.

solved from scratch. Obviously, such this methods are unnatural and inefficient. In fact, previous experience should be accumulated and used for further exploration, similar to the accumulation of knowledge in human experts [33]. To accelerate learning, we make knowledge transfer from other tuning tasks, i.e., we train the agent on a variety of learning tasks on a small scale to acquire a prior experience and learn the common feature representation. In this way, the agent with prior experience will learn faster. Importantly, many previous works have demonstrated the strong performance of meta-learning in knowledge transfer community. In this paper, we use meta-learning to transfer knowledge in the HPO problem.

Specifically, we use the recently proposed model-agnostic meta-Learning algorithm to transfer knowledge (MAML). The algorithm 1 and figure 3 give an overview and workflow of the training process of meta-learning on different HPO tasks respectively. A hyperparameter tuning task $\mathcal{T}_i$ is defined as the optimization of hyperparameters for a given model $\mathcal{A}$ on a dataset i. Following [6], there are two optimizing steps, namely, the meta-training step (Step 6), in which a task-specific learner θ' learns based on the current parameter θ, and the meta-test step (Step 9), in which the parameter θ updates based on the evaluation of θ', where α and β are the learning rates. In this work, τ_i is sampled by θ and is used for the meta-training process; τ_i' is sampled by θ' and is used for the meta-test. After multiple episodes, the meta parameters θ can be obtained from this meta-learning procedure but are not necessarily a good one for the new task. However, these parameters serve as a good starting point for training a good model using only a few steps of learning.

V. SUMMARY OF THE OVERALL FRAMEWORK

To make the proposed approach clearer, we will integrate all the above details to give a complete description of the tuning approach (see Alg. 2). First, the agent's meta-parameters θ are obtained by using meta-learning on multiple small-scale tasks and used to initialize the agent when solving a new task. Then, the distributions of hyperparameters are output sequentially

Algorithm 2 Tuning Method Based on Multi-Objective and Knowledge Transfer

```
Input:
    $s_1$: The initial state, $s_1 = \mathcal{N}(0, 1)$;
    $n$: The number of the algorithm hyperparameters.
Procedure:
1: The agent is initialized with the meta-parameter obtained
   from the algorithm 1
2: while not done do
3:     for t=1 to $n$ do
4:         The agent outputs $\mathcal{N}(\mu_t, \sigma_t)$ based on $s_t$
5:         Sample $a_t$ ($\lambda_t$) from $\mathcal{N}(\mu_t, \sigma_t)$
6:         Obtain $accuracy_t$ and $LAT_t$ on the validation set
           after training $\mathcal{A}_\lambda$
7:     end for
8:     Use the trajectory $\tau = (s_1, a_1, A_1, LAT_1 \ldots, A_n, LAT_n)$ to update the agent's
       parameters by PPO-clip algorithm
9: end while
```

by the agent, and the actual hyperparameters are obtained by sampling and clipping. After n time steps, a hyperparameter configuration Λ with n hyperparameters λ is obtained. Then, the selected hyperparameter configuration Λ is set to the algorithm to be optimized $\mathcal{A}$ and the multi-objective reward vector (accuracy and latency) is obtained by training $\mathcal{A}_\lambda$ on the target task. Finally, a reinforcement learning algorithm is used to update the agent's internal parameters. In this way, an agent not only quickly adapt to new task but also make hyperparameter tuning take into account multiple objectives.

VI. EXPERIMENTS

In this section, we compare other advanced optimization methods on 57 image recognition datasets to illustrate the performance of the proposed method. The objects of algorithm tuning include two tree-based models (random forest and extreme gradient boosting (XGBoost)) and a deep learning model (convolutional neural network). The experiments consist of two parts: comparison experiments and ablation experiments. Comparison experiments are performed to demonstrate the performance advantages of the proposed method, while ablation experiments are performed to show the feasibility and effectiveness of each component of the proposed method. In the following description, we first describe the relevant details of the experiments, and then conduct comparison experiments and ablation experiments respectively.

A. EXPERIMENTAL SETTINGS

1) DATASETS

In this paper, we focus on datasets in the image recognition field and use them as target tasks. Specifically, we collected a total of 77 datasets from the two public repositories [1] (UCI[2]

[1]https://bit.ly/2LNKPex
[2]http://archive.ics.uci.edu/ml/datasets.php

TABLE 1. The table shows the statistical results of the size of the dataset used in the experiment. We can clearly see that the size of the dataset used in the experiment is wide, so we can verify the robustness of the proposed method to datasets of different sizes.

Datasets Size	100-500	500-1,500	1,500-3,000	3,000-5,000	5,000-10,000	10,000-50,000	50,000-200,000
Number	7	12	14	14	11	11	8

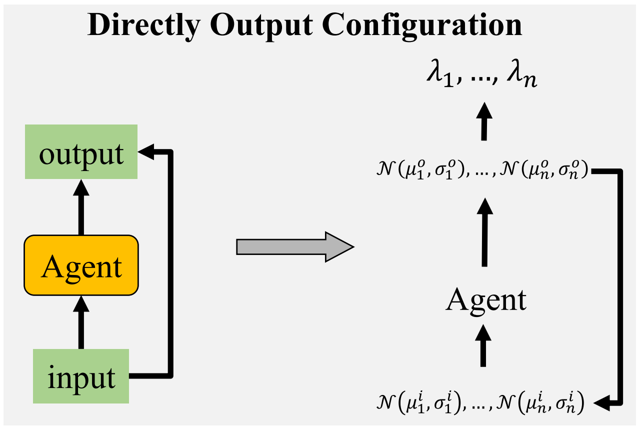

Directly Output Configuration
output
Agent
input
$\lambda_1, \ldots, \lambda_n$
$\mathcal{N}(\mu_1^o, \sigma_1^o), \ldots, \mathcal{N}(\mu_n^o, \sigma_n^o)$
Agent
$\mathcal{N}(\mu_1^i, \sigma_1^i), \ldots, \mathcal{N}(\mu_n^i, \sigma_n^i)$

FIGURE 4. This figure shows the process that the agent directly outputs all hyperparameters at one step. Since the agent makes decisions directly, the horizon of each episode unrolls one step. At each episode, the agent directly outputs μ_i^o and σ_i^o ($i \in [1, n]$) of n hyperparameter distributions, and then the output will be used as input for the next time.

and OpenML[3]). In order to transfer knowledge from the previous task, we selected 20 datasets as the source data sets for meta-learning, and the remaining 57 datasets as the target tasks. Importantly, in order to verify the robustness of the optimization methods, the datasets selected in the experiment include handwritten numbers and letters, cars, animals, and other entities of specific scenes. Moreover, these datasets range in size from thousands to tens of thousands, which can verify the ability of the optimization method to adapt to problems of different sizes (as shown in table 1).

2) COMPARISON METHODS

In this paper, the proposed method is referred to as ETM (effective tuning method), which first initializes the agent through knowledge transfer, then uses the agent select each hyperparameter sequentially, and optimizes accuracy and latency based on multi-objective optimization framework. In the comparison experiment part, we compare the proposed method with the following advanced optimization methods: an evolutionary algorithm-based optimization method CMA-ES [16], three Bayes-based optimization methods TPE [13], Speriment [14] and SMAC [15], and a recent advanced optimization method BOHB [19]. Furthermore, the default hyperparameter configuration of the algorithm to be optimized is used as the baseline.

In order to verify the effectiveness of each component of the proposed method, we propose two variants based on the ETM method: single-ETM and TM. The single-ETM method uses a single-objective optimization framework, which only considers the accuracy performance of the model to be optimized. The other settings are consistent with the ETM method. By comparing the ETM method and the single-ETM method, the effectiveness of multi-objective optimization can be verified, and the advantages of multi-objective optimization for algorithm tuning can also be illustrated. The TM method does not initialize the agent with the knowledge transfer method, while the other settings are the same as the ETM method. By comparing the ETM method and the TM method, the influence of knowledge transfer on optimization efficiency can be explained.

3) EVALUATION CRITERION AND EXPERIMENTAL DETAILS

Following the evaluation criterion of the state-of-the-art papers [15], [30], the three performance indicators of accuracy, time and latency are calculated in each tuning experiment. The above three indicators refer to the performance of the hyperparameter configuration obtained in the training on the test set. In fact, the accuracy performance can be used to illustrate the impact of the tuning of the optimization method on the predictive performance of the optimization model. The time performance can show the optimization efficiency of the optimization method. The latency performance can explain the effect of the tuning of the optimization method on the actual running time of the optimization model. In addition, we calculate the ranking of each optimization algorithm on the above three indicators as well as the average ranking and standard deviation of each optimization algorithm on multiple tasks.

Importantly, we run each tuning method 3 times independently and report the average performance to avoid contingency. Each independent experiment is run 300 times. In the experiment, we use PPO-clip [10] method to update the agent and use Adam algorithm [34] to perform optimization, where we set the learning rate to 0.008. Moreover, we evaluate the hyperparameter configuration by using 5-fold cross validation method. During meta-learning, we sample 5 batches of tasks, and each batch contains 3 different tasks. Afterwards, the Adam algorithm [34] is used to perform 30 meta gradients on each batch of tasks, where $\alpha = 0.0007$ and $\beta = 0.001$. Due to the obvious difference in the size of the data set, the partition ratio is 8 (training set)/2 (test set) for small datasets (the size is less than 10,000), and the partition ratio is 9 (training set)/1 (test set) for big datasets (the size is larger than 10,000). For the trade-off weight w of multi-objective optimization, we use soft constraint to set the weight w. The specific settings and analysis will be described in ablation experiments section. In particular, the proposed method does not introduce parameters of strong sensitivity.

[3]https://www.openml.org/

TABLE 2. Range of hyperparameters of algorithms to be optimized, where *Lower* and *Upper* denote the upper and lower bounds, respectively.

Alg.	Hyperparameter	*Lower*	*Upper*	Alg.	Hyperparameter	*Lower*	*Upper*	Alg.	Hyperparameter	*Lower*	*Upper*
Random Forest Algorithm	n_estimators	100	1200	XGBoost Algorithm	max_depth	1	25	CNN	batch size	24	128
	max_depth	1	35		learning_rate	0.001	0.1		convolution stride	1	5
	min_sample_split	1	100		n_estimators	50	1200		convolution kernel	2	5
	min_samples_leaf	1	100		gamma	0.05	0.9		convolution channel	24	128
	max_features	0.1	0.9		min_child_weight	1	9		pooling kernel	2	5
	bootstrap	0	1		subsample	0.5	1.0		pooling stride	1	5
	-	-	-		colsample_bytree	0.5	1.0		pooling type	0	1
	-	-	-		colsample_bylevel	0.5	1.0		fc layer nodes	128	1100
	-	-	-		reg_alpha	0.1	0.9		learning rate	0.001	0.05
	-	-	-		reg_lambda	0.01	0.1		-	-	-

TABLE 3. The average rank "Rank" and standard deviation "Stdev" of accuracy, time and latency over 101 datasets. "*" and "+" denotes that the statistically significant difference from other values in the same line is $p < 0.01$ and $p < 0.05$, respectively. The best result is in bold font.

Alg.	Measure	ETM(ours)	BOHB	SMAC	TPE	Spearmint	CMA-ES	Baseline
Random Forest	Accuracy Rank	**1.92**	2.44(+)	3.41(*)	5.63(*)	4.74(*)	6.21(*)	8.90(*)
	AR-Stdev	0.31	0.53	0.64	0.61	1.24	0.67	0.53
	Time Rank	**1.79**	3.18(*)	5.42(*)	4.78(*)	6.47(*)	7.29(*)	-
	TR-Stdev	0.51	0.83	0.69	1.32	1.21	0.69	-
	Latency Rank	**1**	4.42(*)	5.01(*)	5.37(*)	5.19(*)	4.87(*)	5.90(*)
	LR-Stdev	0	0.73	0.47	0.69	1.02	1.35	1.14
XGBoost	Accuracy Rank	**1.42**	2.36(+)	4.12(*)	5.45(*)	4.79(*)	6.82(*)	8.94(*)
	AR-Stdev	0.34	0.42	0.53	0.47	1.22	0.44	0.63
	Time Rank	**1.69**	2.98(*)	5.42(*)	4.31(*)	6.36(*)	7.19(*)	-
	TR-Stdev	0.32	1.46	0.17	0.61	0.48	0.56	-
	Latency Rank	**1**	4.31(*)	4.79(*)	5.17(*)	5.42(*)	4.86(*)	5.77(*)
	LR-Stdev	0	0.34	0.18	0.73	0.07	0.90	0.33
CNN	Accuracy Rank	**1.71**	2.52(+)	3.94(*)	5.32(*)	4.47(*)	6.37(*)	8.18(*)
	AR-Stdev	0.31	0.35	0.47	0.58	1.32	0.58	0.52
	Time Rank	**1.79**	3.23(*)	5.40(*)	4.37(*)	6.55(*)	7.58(*)	-
	TR-Stdev	0.34	1.01	0.46	0.65	1.35	0.47	-
	Latency Rank	**1**	4.31(*)	4.25(*)	5.27(*)	5.78(*)	4.79(*)	5.63(*)
	LR-Stdev	0	0.23	0.17	0.43	0.35	0.42	0.57

B. COMPARE WITH OTHER METHODS

In this section, we will verify the performance advantages of the proposed method through running comparison experiments. The comparison experiments mainly take machine learning algorithms and deep learning models in image recognition field as tuning objects. Specifically, we use the tuning algorithms to perform hyperparameters tuning for the random forest, XGBoost and convolutional neural network on 57 tasks. The experimental results and analysis are described below.

1) HYPERPARAMETER TUNING FOR MACHINE LEARNING ALGORITHMS AND DEEP LEARNING MODEL

a: SEARCH SPACE

In this experiment, we chose to optimize the hyperparameters of two advanced machine learning algorithms, the random forest and XGBoost algorithms, based on the following reasons: the random forest algorithm is evaluated by [35] as the best of 179 classifiers arising from 17 families; the XGBoost algorithm contains many more hyperparameters and has recently been dominating the Kaggle competition; the performances of the two algorithms are sensitive to the hyperparameter configuration. The code of two algorithms is based on scikit-learn [36]. Six hyperparameters (continuous) of the random forest algorithm and ten hyperparameters (continuous) of the XGBoost algorithm need to be optimized (Table 2). In recent years, the convolutional neural network has been widely used in the field of image recognition. Therefore, we choose the convolutional neural network in the deep learning models as a more complex optimization object. The architecture of the convolutional neural network is similar to the one proposed by [37], which includes two convolution layers, two pooling layers, and two fully connected layers. We choose 15 hyperparameters to be optimized, including the stride size, kernel size, and channel size in each convolutional layer; the pooling type, kernel size, and stride size in each pooling layer; the number of hidden nodes in each fully connected layer; and the learning rate. The specific information of the hyperparameters is shown in Table 2.

TABLE 4. Number of tasks with the top performance over 101 tasks. The best result is in bold font.

Alg.	Methods	Number of Datasets with Top Accuracy	Number of Datasets with Top Time	Number of Datasets with Top Latency
Random Forest	BOHB	11(19.3%)	3(5.3%)	0(0.0%)
	SMAC	9(15.8%)	11(19.3%)	0(0.0%)
	TPE	0(0.0%)	0(0.0%)	0(0.0%)
	Spearmint	3(5.3%)	0(0.0%)	0(0.0%)
	CMA-ES	0(0.0%)	0(0.0%)	0(0.0%)
	Baseline	0(0.0%)	-	0(0.0%)
	ETM(ours)	**34(60.0%)**	**43(75.4%)**	**57(100.0%)**
XGBoost	BOHB	10(17.5%)	6(10.5%)	0(0.0%)
	SMAC	6(10.5%)	11(19.3%)	0(0.0%)
	TPE	0(0.0%)	0(0.0%)	0(0.0%)
	Spearmint	3(5.3%)	0(0.0%)	0(0.0%)
	CMA-ES	0(0.0%)	1(1.8%)	0(0.0%)
	Baseline	0(0.0%)	-	0(0.0%)
	ETM(ours)	**38(66.7%)**	**39(68.4%)**	**57(100.0%)**
CNN	BOHB	14(24.6%)	7(12.3%)	0(0.0%)
	SMAC	6(10.5%)	11(19.3%)	0(0.0%)
	TPE	0(0.0%)	0(0.0%)	0(0.0%)
	Spearmint	5(8.8%)	5(8.8%)	0(0.0%)
	CMA-ES	2(3.5%)	1(1.8%)	0(0.0%)
	Baseline	0(0.0%)	-	0(0.0%)
	ETM(ours)	**30(52.6%)**	**33(57.9%)**	**57(100.0%)**

b: EXPERIMENTAL RESULTS AND ANALYSIS

Following the previous settings, we run the experiments and report the experimental data. Table 3 shows the optimization performance of each optimization method on three algorithms to be tuned, which includes the average ranking of accuracy, the average ranking of time, the average ranking of latency, their standard deviation and significance level. It is clear from the table 3 that our proposed approach performs strongly in the three optimization scenarios. Specifically, although BOHB and SMAC methods can be competitive in terms of accuracy, the proposed method achieves optimal (i.e., the lowest accuracy ranking) under three optimization scenarios. In terms of time performance, the proposed method can achieve the best performance under three optimization scenarios. It is worth noting that the advantage is more pronounced in the XGBoost and convolutional neural network optimizations scenarios. This indicates that the proposed method can be well adapted to complex optimization tasks. In terms of latency performance, the average ranking of the proposed method is first, that is, it achieved optimal performance on all 57 tasks, which also indicates that considering the multi-objective optimization of accuracy and latency at the same time can improve latency performance while maintaining accuracy performance. Importantly, we use the Friedman statistical test and Wilcoxon post-hoc test to ensure that experimental comparisons are statistically significant [38].

In addition, we compare optimization methods from another perspective, that is, the number of tasks that each optimization method achieves optimal performance on 57 tasks. Through the above statistics and analysis, the performance level of each optimization method on each task can be illustrated. The experimental results are shown in Table 4. Obviously, our method is significantly better than other methods (especially in latency).

To conclude, the proposed approach is superior to other methods in most optimization scenarios and optimization tasks, and the latency performance has obvious advantages.

C. ABLATION EXPERIMENTS

In this section, we demonstrate the effectiveness of each component in the proposed method by performing ablation experiments. In the ablation experiments, We only choose some tuning scenarios and target datasets in the comparison experiments.

1) MULTI-OBJECTIVE OPTIMIZATION VS SINGLE-OBJECTIVE OPTIMIZATION.

In this section, we verify the impact of multi-objective optimization on tuning by comparing the ETM and single-ETM methods. This ablation experiment takes XGBoost as the algorithm to be optimized and 12 datasets as the target task. The experiment is independently executed for 3 times, and each experiment iteration is 300 times. The experiment results are shown in Table 6, and it can be clearly seen that the hyperparameter configuration searched by the ETM can improve the latency performance of the model to be optimized on the premise of ensuring accuracy performance.

To further study of the multi-objective optimization, we chose two sets of values of w to study the effects of soft constraint and hard constraint. The settings of w are as follows:

$$w_{soft} = \begin{cases} \alpha = \begin{cases} -0.07, & \text{if } A^{\pi_\theta} \geq 0 \\ 0.07, & \text{if } A^{\pi_\theta} < 0 \end{cases} & \text{if } LAT \leq T \\ \beta = \begin{cases} -0.07, & \text{if } A^{\pi_\theta} \geq 0 \\ 0.07, & \text{if } A^{\pi_\theta} < 0 \end{cases} & \text{if } LAT > T \end{cases} \tag{13}$$

TABLE 5. Given the XGBoost algorithm, we set three tasks and each task corresponds to a different hyperparameter optimization order.

No.	Hyperparameter Optimization Order
Order 1	$\Rightarrow max\_depth \Rightarrow n\_estimators \Rightarrow min\_child\_weight \Rightarrow colsample\_bytree \Rightarrow reg\_alpha \Rightarrow learning\_rate \Rightarrow gamma \Rightarrow subsample \Rightarrow colsample\_bylevel \Rightarrow reg\_lambda$
Order 2	$\Rightarrow learning\_rate \Rightarrow gamma \Rightarrow subsample \Rightarrow colsample\_bylevel \Rightarrow reg\_lambda \Rightarrow max\_depth \Rightarrow n\_estimators \Rightarrow min\_child\_weight \Rightarrow colsample\_bytree \Rightarrow reg\_alpha$
Order 3	$\Rightarrow colsample\_bytree \Rightarrow reg\_alpha \Rightarrow learning\_rate \Rightarrow max\_depth \Rightarrow n\_estimators \Rightarrow min\_child\_weight \Rightarrow gamma \Rightarrow subsample \Rightarrow colsample\_bylevel \Rightarrow reg\_lambda$

TABLE 6. The test performance, latency and runtime of two variants (ETM and single-ETM). "acc", "latency" and "time" represent accuracy, latency and time performance respectively. We report the mean of the 3 test performances. The best result is in bold font.

Datasets	ETM			single-ETM		
	acc	latency(ms)	time(min)	acc	latency(ms)	time(min)
HAR	0.8036	**10**	**22.5**	**0.8124**	18	27.4
Optdigits	0.9721	**12**	33.4	**0.9722**	22	**27.3**
Fourclass	**0.9641**	**18**	**34.4**	0.9472	37	37.3
SVHN	0.9735	**32**	42.6	**0.9767**	54	**39.5**
Phoneme	**0.9577**	**40**	**54.6**	0.9571	68	56.3
MNIST	**0.9983**	**45**	47.7	0.9981	73	**45.9**
Iris	0.9869	**47**	**37.3**	**0.9871**	70	37.9
Vehicle	**0.9614**	**63**	60.4	0.9610	118	**58.7**
Scene	0.9758	**84**	67.2	**0.9795**	137	**66.9**
Fashion MNIST	**0.9464**	**168**	**134.7**	0.9347	316	139.2
Letter Recognition	**0.9501**	**259**	**209.4**	0.9493	411	213.2
Traffic Sign	**0.6517**	**397**	275.8	0.6474	556	**274.6**

$$w_{hard} = \begin{cases} \alpha = 0, & \text{if } LAT \leq T \\ \beta = \begin{cases} -1, & \text{if } A^{\pi_\theta} \geq 0 \\ 1, & \text{if } A^{\pi_\theta} < 0 \end{cases} & \text{if } LAT > T \end{cases} \tag{14}$$

Figure 5 shows the accuracy and latency of the hyperparameter configuration under soft constraint (w_{soft}) and hard constraint (w_{hard}) settings. When the weight coefficient w is set as the hard constraint, the agent is more inclined to choose a hyperparameter configuration that can improve the latency performance, so as to avoid the severe penalty of the advantage value. However, the hard constraint setting makes the searched hyperparameter configuration to fall into local optimal in terms of accuracy performance. In contrast, when w is set as w_{soft}, the agent can better trade-off accuracy-latency. As shown in Figure 5, although the latency of the hard constraint is lower than soft constraint on most samples, the test set accuracy of the soft constraint is significantly better than the hard constraint on most samples. Importantly, the soft constraint make the agent to explore some configurations that can achieve the best test set accuracy while with less latency.

2) KNOWLEDGE TRANSFER VS TUNING FROM SCRATCH

We compared the performances of the proposed method ETM (which transfer knowledge by the meta-learning) and TM (which does not transfer knowledge) methods on 12 target tasks (Table 7). In terms of test results and runtime, we found that ETM is superior to TM in all datasets, and the latency in the real-world is not affected. Although TM method also employs an agent to sequentially select hyperparameters, it ignores tuning experience of the previous tasks. However, the ETM method uses meta-learning to utilize the experience of previous optimization tasks and uses meta-parameters to initialize the agent, which accelerates the agent's ability to adapt to new tasks. The experimental results demonstrate that the agent can adapt to new tasks quickly by using meta parameters. In addition, it can find better configurations by only a few samples and is not limited by suboptimal values.

TABLE 7. The test performance, latency and runtime of the ETM and a variant TM. "acc", "latency" and "time" represent accuracy, latency and time performance respectively. We report the mean of the 3 test performances. The best result is in bold font.

Datasets	ETM			TM		
	acc	latency(ms)	time(min)	acc	latency(ms)	time(min)
HAR	**0.8036**	**10**	**22.5**	0.7971	16	25.9
Optdigits	**0.9721**	12	**33.4**	0.9708	**11**	35.4
Fourclass	**0.9641**	**18**	**34.4**	0.9411	**18**	43.7
SVHN	**0.9735**	**32**	**42.6**	0.9715	35	50.5
Phoneme	**0.9577**	40	**54.6**	0.9521	**38**	65.2
MNIST	**0.9983**	45	**47.7**	0.9981	**43**	70.5
Iris	**0.9869**	**47**	**37.3**	0.9772	49	57.8
Vehicle	**0.9614**	63	**60.4**	0.9521	**61**	83.6
Scene	**0.9758**	**84**	**67.2**	0.9601	87	101.4
Fashion MNIST	**0.9464**	168	**134.7**	0.9404	**158**	224.9
Letter Recognition	**0.9501**	259	**209.4**	0.9473	**250**	321.6
Traffic Sign	**0.6517**	**397**	**275.8**	0.6453	401	338.1

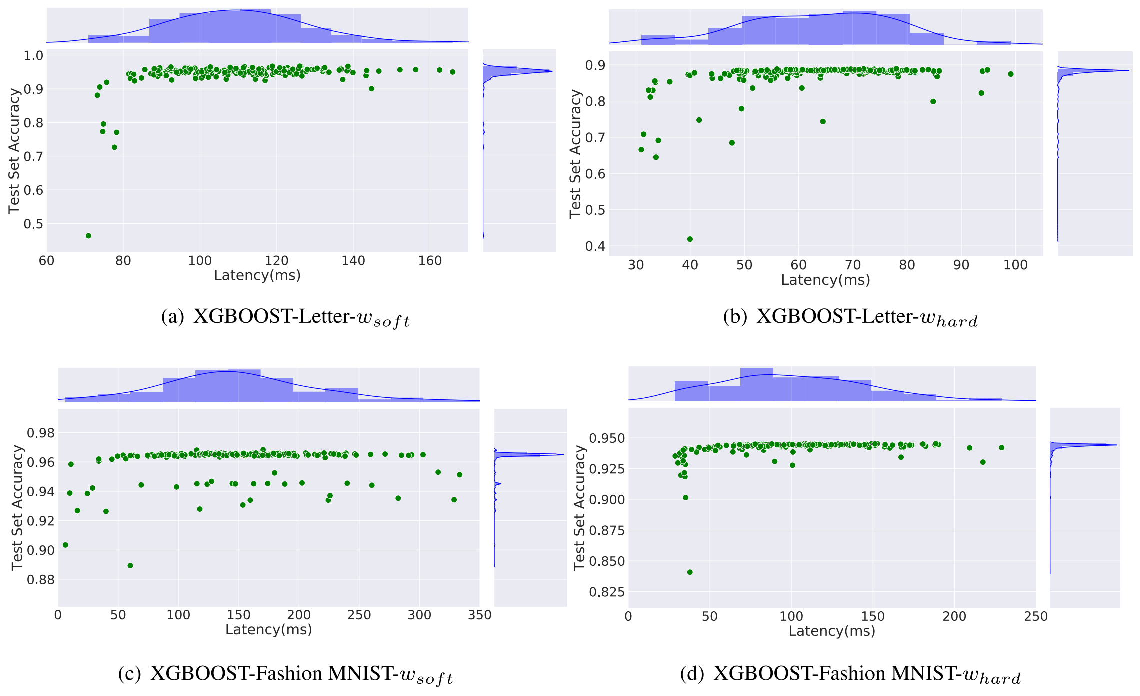
TABLE 8. We set three orders, and the orders of selecting hyperparamters are random and different. "acc", "latency" and "time" represent accuracy, latency and time performance respectively. We report the mean of the 3 test performances. The best result is in bold font.

Datasets	Order1			Order2			Order3		
	acc	latency	time	acc	latency	time	acc	latency	time
HAR	0.8036	10	22.5	0.8062	11	22.1	0.8057	11	23.3
Optdigits	0.9721	12	33.4	0.9720	13	33.2	0.9719	17	35.7
Fourclass	0.9641	18	34.4	0.9644	16	35.9	0.9652	18	36.7
SVHN	0.9735	32	42.6	0.9746	30	43.9	0.9744	28	42.2
Phoneme	0.9577	40	54.6	0.9571	42	53.4	0.9583	42	54.8
MNIST	0.9983	45	47.7	0.9983	45	47.2	0.9985	43	47.9
Iris	0.9869	47	37.3	0.9869	47	35.6	0.9871	44	36.7
Vehicle	0.9614	63	60.4	0.9622	63	59.7	0.9617	68	61.9
Scene	0.9758	84	67.2	0.9753	83	68.7	0.9747	80	69.3
Fashion MNIST	0.9464	168	134.7	0.9460	164	130.4	0.9468	166	136.3
Letter Recognition	0.9501	259	209.4	0.9487	255	197.9	0.9521	257	204.6
Traffic Sign	0.6517	397	275.8	0.6521	401	280.4	0.6497	399	272.3

3) SEQUENTIAL DECISION MAKING VS DIRECTLY OUTPUTS CONFIGURATION

In this part, we tune the XGBoost on 12 tasks and compare ETM-SDM (SDM: sequential decision making) and ETM-DOC (directly outputs configuration) to verify the feasibility of sequential decision making. The experimental results are presented in Figure 9, where each method ran for 200 episodes. We can see that the tuning of method ETM-DOC only works on a few tasks and even fails on some tasks. However, the ETM-SDM method, which treats HPO as a sequential decision problem, achieved better test set performance on all target datasets. We believe that the reasons for this result is as follows: if the search space is very large, it is difficult for ETM-DOC to explore a good policy in such large space. However, ETM-SDM method makes a new decision sequentially, the search space is reduced at each time-step and in this way, it is much easier to handle the problem.

To further explore sequential decision making, we randomly set three different optimization orders for the XGBoost (as shown in Table 5). By comparing the performance of three



(a) XGBOOST-Letter-w_{soft} (b) XGBOOST-Letter-w_{hard}

(c) XGBOOST-Fashion MNIST-w_{soft} (d) XGBOOST-Fashion MNIST-w_{hard}

FIGURE 5. Comparison the test set accuracy and latency of w_{soft} and w_{hard} on the two target tasks. w_{soft} denotes soft constraint, and w_{hard} denotes hard constraint. Each figure shows 200 samples, and the corresponding histograms are shown on the upper side and the right side of the scatter plot respectively.

TABLE 9. The test performance, latency and runtime of two variants (ETM-SDP and ETM-DOC). "acc", "latency" and "time" represent accuracy, latency and time performance respectively. We report the mean of the 3 test performances. The best result is in bold font.

Datasets	ETM-SDP(ours)			ETM-DOC		
	acc	latency(ms)	time(min)	acc	latency(ms)	time(min)
HAR	**0.8036**	**10**	**22.5**	0.7803	20	30.7
Optdigits	**0.9721**	**12**	**33.4**	0.9523	32	40.5
Fourclass	**0.9641**	**18**	**34.4**	0.9379	43	49.2
SVHN	**0.9735**	**32**	**42.6**	0.9537	58	69.2
Phoneme	**0.9577**	**40**	**54.6**	0.9436	57	66.4
MNIST	**0.9983**	**45**	**47.7**	0.9893	71	68.9
Iris	**0.9869**	**47**	**37.3**	0.9710	74	49.3
Vehicle	**0.9614**	**63**	**60.4**	0.9546	84	81.2
Scene	**0.9758**	**84**	**67.2**	0.9601	103	87.6
Fashion MNIST	**0.9464**	**168**	**134.7**	0.9368	185	176.7
Letter Recognition	**0.9501**	**259**	**209.4**	0.9379	358	274.4
Traffic Sign	**0.6517**	**397**	**275.8**	0.6419	473	323.6

different optimization orders, it is shown that the proposed method is insensitive to the optimization order. We can see from Table 8 that the random optimization order does not affect the final tuning performance.

VII. CONCLUSION

In this paper, we focus on algorithm tuning in the field of image recognition and propose an efficient hyperparameter optimization method. This method uses an agent to select hyperparameters sequentially. Compared with the traditional tuning method, this method is based on a multi-objective optimization framework that simultaneously takes accuracy and latency as optimization objectives, and customizes an aggregation function to trade-off accuracy and latency. Finally, we use reinforcement learning algorithm to update the policy. In order to improve the efficiency of tuning, we use meta-learning to obtain the meta-parameters of the agent in the previous optimization tasks and use the meta-parameters to initialize the agent when solving a new task. In the experiments, we compared the proposed method with other advanced tuning methods on 57 datasets of image recognition fields. The experimental results show that the proposed method can perform strongly in time, accuracy, and latency. Specifically, the proposed method achieves average accuracy rankings of 1.92, 1.42 and 1.71 on three algorithms to be optimized, respectively. Especially in terms of latency performance, the proposed method performs best on all the tasks (57 data sets) on the three algorithms to be optimized. In addition, we verify the effectiveness of the proposed component by performing the ablation experiments.

REFERENCES

[1] J. Xu, L. Luo, C. Deng, and H. Huang, "Bilevel distance metric learning for robust image recognition," in *Proc. Adv. Neural Inf. Process. Syst.*, Montreal, QC, Canada, 2018, pp. 4202–4211.
[2] M. S. Alfarzaeai, Q. Niu, J. Zhao, R. M. A. Eshaq, and E. Hu, "Coal/gangue recognition using convolutional neural networks and thermal images," *IEEE Access*, vol. 8, pp. 76780–76789, 2020.
[3] A. A. Chandio, M. Asikuzzaman, and M. R. Pickering, "Cursive character recognition in natural scene images using a multilevel convolutional neural network fusion," *IEEE Access*, vol. 8, pp. 109054–109070, 2020.
[4] J. Bergstra and Y. Bengio, "Random search for hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 2, pp. 281–305, 2017.
[5] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2962–2970.
[6] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 1126–1135.
[7] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, and H. Cho. (2017). *Xgboost: Extreme Gradient Boosting*. [Online]. Available: https://github.com/dmlc/xgboost

[8] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[9] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, Lille, France, Jul. 2015, pp. 1889–1897.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: https://arxiv.org/abs/1707.06347

[11] A. Abels, D. M. Roijers, T. Lenaerts, A. Nowé, and D. Steckelmacher, "Dynamic weights in multi-objective deep reinforcement learning," in *Proc. 36th Int. Conf. Mach. Learn.*, Long Beach, CA, USA, 2019, pp. 11–20.

[12] T. Tajmajer, "Multi-objective deep Q-learning with subsumption architecture," 2017, *arXiv:1704.06676*. [Online]. Available: https://arxiv.org/abs/1704.06676

[13] Y. B. James Bergstra, R. Bardenet, and B. Kegl, "Algorithms for hyperparameter optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 2546–2554.

[14] S. Jasper, L. Hugo, and A. P. Ryan, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Neural Inf. Process. Systems (NIPS)*, Jun. 2012, pp. 2951–2959.

[15] H. H. H. Hutter Frank and L. Kevin, "Sequential model-based optimization for general algorithm configuration," in *Proc. Learn. Intell. Optim.-Int. Conf.*, 2012, pp. 507–523.

[16] N. Hansen, *The CMA Evolution Strategy: A Comparing Review*. Berlin, Germany: Springer, 2006.

[17] L. Ilya and H. Frank, "CMA-ES for hyperparameter optimization of deep neural networks," in *Proc. Int. Conf. Learn. Represent. Workshop*, Feb. 2016, pp. 23–33.

[18] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 185:1–185:52, 2017.

[19] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 1436–1445.

[20] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2820–2828.

[21] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via Lamarckian evolution," in *Proc. 7th Int. Conf. Learn. Represent., ICLR*, New Orleans, LA, USA, May 2019, pp. 1–23.

[22] Z. Chen, F. Zhou, G. Trimponias, and Z. Li, "Multi-objective neural architecture search via non-stationary policy gradient," 2020, *arXiv:2001.08437*. [Online]. Available: https://arxiv.org/abs/2001.08437

[23] M. Hofer, A. Kormilitzin, P. Goldberg, and A. Nevado-Holgado, "Few-shot learning for named entity recognition in medical text," 2018, *arXiv:1811.05468*. [Online]. Available: http://arxiv.org/abs/1811.05468

[24] Z.-Q. Cheng, X. Wu, S. Huang, J.-X. Li, A. G. Hauptmann, and Q. Peng, "Learning to transfer: Generalizable attribute learning with multitask neural model search," in *Proc. 26th ACM Int. Conf. Multimedia*, Oct. 2018, pp. 90–98.

[25] J. Gu, Y. Wang, Y. Chen, V. O. K. Li, and K. Cho, "Meta-learning for low-resource neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2018, pp. 3622–3631.

[26] D. Guo, D. Tang, N. Duan, M. Zhou, and J. Yin, "Coupling retrieval and meta-learning for context-dependent semantic parsing," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 855–866.

[27] J. Schmidhuber, "Reinforcement learning upside down: Don't predict rewards - just map them to actions," 2019, *arXiv:1912.02875*. [Online]. Available: http://arxiv.org/abs/1912.02875

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[29] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," 2018, *arXiv:1812.05905*. [Online]. Available: https://arxiv.org/abs/1812.05905

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

[31] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 2787–2794.

[32] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12, 1999, pp. 1057–1063.

[33] T. Chen, I. Goodfellow, and J. Shlens, "Net2Net: Accelerating learning via knowledge transfer," in *Proc. Int. Conf. Learn. Represent. (15-ICLR)*, 2015, pp. 1–12.

[34] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–15.

[35] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3133–3181, 2014.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, V. D. RonWeiss, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[37] (2018). *Tensorflow*. [Online]. Available: https://github.com/tensorflow/models/

[38] M. Al-Janabi, M. A. Ismail, and V. Mezhuyev, "Improved TLBO-JAYA algorithm for subset feature selection and parameter optimisation in intrusion detection system," *Complex*, vol. 2020, May 2020, Art. no. 5287684.

WEICHUN LIU received the M.S. degree in communication and information system from Zhengzhou University, in 2007, and the Ph.D. degree in information and communication engineering from the National University of Defense Technology, in 2020. He is currently an Associate Professor with the School of Information Engineering, Shaoyang University. His research interests include computer vision, machine learning, and computer graphics.

CHENGLIN ZHAO is currently a Full Professor with the School of Information Engineering, Shaoyang University, where he is also a Master's Supervisor and the Dean. His research interests include image processing, machine vision, artificial intelligence, computer graphics, and scientific visualization.