Population-based Hyperparameter Tuning with Multi-task Collaboration

Wendi Li, *Student Member, IEEE,* Ting Wang*, *Student Member, IEEE,* Wing W. Y. Ng*, *Senior Member, IEEE*

***Abstract*—Population-based optimization methods are widely used for hyperparameter tuning for a given specific task. In this work, we propose the Population-based Hyperparameter Tuning with Multi-task Collaboration (PHTMC) which is a general multi-task collaborative framework with parallel and sequential phases for population-based hyperparameter tuning methods. In the parallel hyperparameter tuning phase, a shared population for all tasks is kept and the inter-task relatedness is taken into account to both yield a better generalization ability and avoid data bias to a single task. In the sequential hyperparameter tuning phase, a surrogate model is built for each new-added task so that the meta-information from existing tasks can be extracted and used to help the initialization for the new task. Experimental results show significant improvements in generalization abilities yielded by neural networks trained using the PHTMC and better performances achieved by multi-task meta-learning. Moreover, a visualization of the solution distribution and the autoencoder's reconstruction of both the PHTMC and a single-task population-based hyperparameter tuning method is compared to analyze the property with the multi-task collaboration.**

***Index Terms*—Hyperparameter Tuning, Deep Neural Network, Multi-task Learning.**

I. Introduction

THE number of hyperparameters increases significantly when deep neural networks become larger and deeper. Hyperparameter selection is essential to the performance of deep neural networks. Hyperparameters determine the neural network structure and control its training process, whose performance is sensitive to. Hyperparameter tuning aims to find the optimal hyperparameters for a specific deep neural network architecture for a given task. In general, the performance of a hyperparameter configuration acting on a deep neural network is considered a black box. The whole process of hyperparameter tuning can be regarded as a black-box optimization process. Random search and grid search are the first adopted methods which are also the simplest and most direct methods. Although random search methods have been proved to be generally faster than grid search methods [1], both of them are inefficient with the number of hyperparameters increases and the hyperparameter configuration space becomes high-dimensional [2]. For each real-world task, only its valid hyperparameter configurations can make the deep neural network work. Valid hyperparameters are only a small portion of the overall hyperparameter configuration space. Completely random search wastes most of the computational resources outside the valid hyperparameter regions.

Therefore, guided search algorithms are proposed. Researchers have attempted many algorithms including genetic algorithms [3], Particle Swarm Optimization (PSO) [4]–[6], Estimation of Distribution Algorithms (EDA) [7], and Bayesian optimization [8]–[10] in hyperparameter tuning. Population-based evolutionary strategies in hyperparameter tuning mainly include the population-based training (PBT) [11], the covariance matrix adaptation evolutionary strategy (CMA-ES) [12], [13] and the hyperparameter tuning with other evolutionary algorithms [6], [14]–[16]. Such methods use evolutionary mechanisms to explore better-performance points by allocating more deep neural network threads near good-performance ones. Population-based evolutionary strategies have shown good performance in black-box optimization, thus good results have been obtained by applying them to hyperparameter tuning [11], [17], [18].

However, an issue of these population-based evolutionary methods is that their optimization results are task-dependent because all individuals of the population are created for a given task. Therefore, the tuned hyperparameter configurations cannot generalize for other tasks. In practice, when an intelligent device is switched from the original task to another task, completely re-tuning is needed to find a suitable hyperparameter configuration for the new task. This is because that the fine-tuned hyperparameter configuration for a certain task is just catered to the data bias of the task itself. Task-dependent hyperparameter tuning methods may not be able to discover universal hyperparameter configurations, hence making the current task's hyperparameter configuration non-reusable. Therefore, we propose the Population-based Hyperparameter Tuning with Multi-task Collaboration (PHTMC). For tasks in a task pool, the PHTMC aims to find hyperparameter configurations that work for the current task and also work for other tasks after slightly re-tuning. For a new-added task, existing information from previous tasks is utilized to facilitate the initialization of the learning for the new task. Our major contributions are as follows:

1) The PHTMC is the first framework that aims to acceler-

This research was funded by National Natural Science Foundation of China under Grant 61876066, Guangdong Province Science and Technology Plan Project (Collaborative Innovation and Platform Environment Construction) 2019A050510006, and the National Training Program of Innovation and Entrepreneurship for College Students 201910561071. Corresponding authors: Ting Wang and Wing W. Y. Ng

Wendi Li is with the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. (e-mail: cswendyli@mail.scut.edu.cn).

Ting Wang is with the Department of Radiology, Guangzhou First People's Hospital, School of Medicine, South China University of Technology, Guangzhou, Guangdong, China. (e-mail: twang.scut@qq.com).

Wing W. Y. Ng is with the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. (e-mail: wingng@ieee.org).

ate the population-based hyperparameter tuning methods with multi-task collaboration, considering the learning trajectory of each individual in the population. Such that, the final hyperparameter solution also avoids data bias to a certain task because multiple tasks in the task pool are considered during the optimization.

2) The PHTMC provides a two-stage hyperparameter tuning process. The hyperparameter tuning experience in the first stage can be stored permanently (for example, save the hyperparameter configurations and their performances on disk) so that the historical experience can help initialize the hyperparameter tuning process of each new-added task.
3) The PHTMC can extract useful common high-level features among tasks. Therefore, the neural network trained by the PHTMC without extra tuning can be switched to other similar tasks and yields better performance than that of single-task hyperparameter tuning methods'.

This paper is organized as follows. Section II reviews related works. Our framework is proposed in Section III. Section IV presents experimental results and discussions. Finally, we conclude this work in Section V.

II. Related Works

Automated machine learning aims to build models to help humans do chores in machine learning in an automated, data-driven way [19]. This concept has been applied in training or designing deep neural networks. Many open-source AutoML systems are available, such as Auto-WEKA [20], TPOT [21], Hyperband [22], and Auto-Sklearn [23], to use AutoML methods in real applications easily. As the number of hyperparameters in a deep neural network is growing larger, the hyperparameter tuning process becomes complex and time-consuming. There are many different approaches being proposed to deal with this issue, e.g., reinforcement learning [24]–[26], Bayesian optimization [27]–[30], and gradient-based optimization [31]–[33]. However, the range of their applications may be limited because of strong assumptions being imposed by these methods. Thus, universal search methods without any prior information, such as random search methods [1], [34], are proposed because they do not require any assumption and are not limited to a particular application. However, random search methods are very inefficient due to the small portion of valid regions over the whole search space while random search methods search for every point with an equal probability. In this situation, efforts are wasted in invalid regions. Therefore, guided search algorithms are proposed based on assumptions that the performance of hyperparameter configurations should obey the building block hypothesis [35] and schema theorem [36]. Many evolutionary strategies are applied to neural network architecture designing [37]–[41] and hyperparameter tuning [3], [42]–[44].

A. Population-based Evolutionary Strategies in Hyperparameter Tuning

Population-based evolutionary strategies can efficiently deal with the hyperparameter tuning issue. The basic assumption of the evolutionary strategy in hyperparameter tuning is that the process from the hyperparameter input to the performance of a deep neural network can be regarded as a black box. A hyperparameter configuration of a deep neural network defines its architecture and controls the training process. After the training process, the trained deep neural network will have a performance on the task measured by some mechanism, such as getting the accuracy on the testing set or scored by special equipment. Thus, each deep neural network can be regarded as a thread that has the hyperparameter configuration as the input and its performance as the output. During the hyperparameter tuning process, the performance of a deep neural network is often approximately estimated by the performance on a validation set. Moreover, the final performance is not constant for the same hyperparameter configuration because of the stochastic parameter optimization process. Such indeterminacy factors are ignored for their slight influence and simplicity. Hence, the normal form of population-based evolutionary strategies can be built. Each hyperparameter configuration is a point in the hyperparameter space and can be regarded as an individual of the population. The evolutionary strategy keeps a set of individuals as candidate solutions. The goal of the evolutionary strategy is to maximize the performances of candidate solutions which are also called fitness values.

Various evolutionary strategies have been proposed to deal with hyperparameter tuning. They have two major procedures: *exploit* and *explore* [11]. In the *exploit* procedure, algorithms adjust the population by applying crossover or copying current well-behaved individuals to perform the communication of beneficial genes within the population. This procedure only makes use of the acquired information within the population. In the *explore* procedure, algorithms attempt to glean extra information by exploring new regions in the hyperparameter space. In the PBT [11], the *exploit* procedure is overwriting hyperparameters and parameters of other deep neural network threads with well-behaved deep neural network threads to keep multiple copies of individuals that take the useful information. The PBT's *explore* procedure is to probe new information by adding perturbation to the original positions based on its assumption of the information near the proper individuals that may also be useful. In the CMA-ES [12], the *exploit* procedure is copying deep neural network threads while the *explore* procedure is applying covariance matrix adaptation to discover new information. In genetic algorithms [14], [15], [45], the crossover and the mutation are the *exploit* and the *explore* procedures, respectively. Therefore, all the population-based evolutionary strategies can be summarized as Algorithm 1.

Population-based evolutionary strategies aim to optimize a single task by creating many individuals to deal with the task. They can be applied to different tasks, but a new hyperparameter tuning process is needed. Therefore, we propose the PHTMC to overcome the re-tuning issue for multiple tasks and speed up the hyperparameter tuning process. With collaboration from other tasks, shared information can be found and the data bias of individual tasks can be avoided.

Algorithm 1 Population-based Evolutionary Strategies' Normal Form

1: Initialize a population ▷ Each individual is a hyperparameter configuration instance and can be treated as a point in the hyperparameter space
2: **for** each evolutionary iteration **do**
3: Get the fitness value of each individual in the population
4: Apply *exploit* procedure to the population to take advantage of the acquired information
5: Apply *explore* procedure to the population to discover new information
6: **end for**
7: **return** the final population

B. Meta-Learning in Multi-task Collaboration

Shared information exists in related tasks, so there are approaches to allow algorithms to learn multiple tasks simultaneously [46]–[48]. Meta-learning is also called learning to learn. The meta-learning-based multi-task collaboration aims to speed up the learning process by extracting meta-information across different tasks. The meta-learning [19] in our paper refers that we apply hyperparameter tuning methods to learn how to train a neural network. There are two parts to multi-task collaboration. The first part is that we optimize multiple objectives from different tasks, which is inherently a multi-task optimization problem [49]. By achieving the Pareto optimal, the solution can avoid data bias of a specific task, and the generalization capability of the trained model can be enhanced. The second part is that we model the mapping from the performance vector of multiple tasks in the task pool to the performance of the new-added task. There are two goals to apply the meta-learning-based multi-task collaboration. The first one is that general hyperparameter setting information difficult to find in a specific task can be discovered by multi-task collaboration [50], thus the hyperparameter configuration can be generalized even if the task is data biased intensively. The second one is that the overall hyperparameter tuning process can be accelerated by using information obtained from other tasks. Meta-data can be mined, or a mechanism for extracting meta-data can be built by observing the performance of a set of hyperparameters on different tasks.

A difficulty in meta-learning is the design of an algorithm to select useful meta-features from experience and extract meta-information in a data-driven way [51]. Methods have been proposed to select meta-features such as using the tasks' ranks [52], [53], the relative performance [54], [55] and prior evaluations [56]. After selecting meta-features, the meta-model is built to help to choose the hyperparameter configurations. Such meta-models have been applied to binary classification [57] and support vector machines [58]. The meta-model provides a good hyperparameter initialization of tasks.

A few multi-task meta-learning methods are used in population-based evolutionary strategies [59]–[61], yet most of the population-based evolutionary strategies are used for hyperparameter tuning for a single task only. Meta-learning concepts in population-based evolutionary strategies mainly focus on learning from the history of a single task. These learning concepts are similar to the Lamarckian evolution [62] which emphasizes the inheritance of acquired characters because good features will be encouraged by the meta-learning model. It has been used in neural network training [39] and network architecture search [39].

The characteristics compared among typical search strategies and our PHTMC is shown in Table I. We compare them in three aspects. Typical Bayesian optimization, reinforcement learning, and gradient-based optimization methods are sequential model-based optimization, which means each new training procedure requires the entire training procedure of the previous one. The other methods can be parallelized by employing multiple threads. The gradient-based optimization requires additional strong assumptions, so it only works on limited applications. The improvement of applying PHTMC is that it considers the learning trajectory rather than a single, current evaluation of each individual.

In the proposed PHTMC, the population-based evolutionary strategy is combined with the multi-task collaboration. For tasks to be tuned at the beginning, we keep a population to optimize multiple tasks simultaneously in order to collect meta-information among different tasks. For each new-added task, we build a meta-model to predict the performance of the new-added task by the meta-information from previous tasks. Then, the meta-model helps recommend the initial hyperparameter configurations for the new-added task.

III. Population-based Hyperparameter Tuning with Multi-task Collaboration

The proposed multi-task-based hyperparameter tuning framework consists of two phases: parallel hyperparameter tuning and sequential hyperparameter tuning. The first phase is shown in Fig. 1. The algorithm keeps a population for initial tasks. Each task computes the fitness values of all individuals and selects individuals yielding the largest fitness values. An individual is allowed to be selected by different tasks repeatedly, but also may not be selected by any task.

The second phase is shown in Fig. 2 which applies meta-learning to tasks from their historical experience. For each new-added task, a surrogate model is built using the performance from existing multiple tasks to predict the performance on the new-added task. Then, the surrogate model is used to initialize the new-added task's hyperparameter tuning process.

An individual presenting a hyperparameter configuration is defined as $\boldsymbol{\theta} \in \mathbb{R}^n$ where n denotes the number of hyperparameters being tuned. For some non-real-number type hyperparameters in a deep neural network, such as integer (e.g., the number of hidden units) and categorical (e.g., the type of the optimizer) types, hyperparameter tuning algorithms are applied after they are converted to the real numbers. Hyperparameter configurations for all tasks should be normalized to have the same dimensionality so that the population can be applied to all tasks. Each hyperparameter configuration is regarded as a point in the hyperparameter configuration space $\mathbb{R}^n$. For each initial task $t \in \mathcal{T}$ to be tuned, where $\mathcal{T}$ denotes the task pool, $\psi_k(\boldsymbol{\theta})$ denotes the performance of a

TABLE I
THE SEARCH STRATEGIES CHARACTERISTICS COMPARING

Methods	Characteristic		
	Parallelizability	Applicable Scope	Information Used in Exploration
Random Search [1]	Parallel	Not Limited	None.
Bayesian Optimization [27]–[30]	Sequential	Not Limited	Previous evaluated points.
Reinforcement Learning [24]–[26]	Sequential	Not Limited	Generated actions.
Gradient-based Optimization [31]–[33]	Sequential	Limited	The hypergradient of the last evaluation.
Evolutionary Strategy [3], [11], [12]	Parallel	Not Limited	The current evaluations of the individuals in the population.
Evolutionary Strategy with PHTMC	Parallel	Not Limited	The evaluation trajectories of the individuals in the population.

Algorithm 2 Initial Tasks' Hyperparameter Tuning

Initialize the task pool $\mathcal{T}$ with K tasks
Determine the number of individuals I_k for each task t_k
Initialize the initial hyperparameter population group $\mathbf{\Theta}$ with I individuals ▷ Hence $I = \sum_{k=1}^{K} I_k$
Initialize the performance matrix $\mathbf{\Psi}$
for each iteration q in the tuning process **do**
  for each task t_k in the task pool $\mathcal{T}$ **do**
    for each individual $\boldsymbol{\theta}_i$ in the population $\mathbf{\Theta}$ **do**
      $\Psi_{k,i} \leftarrow \psi_k(\boldsymbol{\theta}_i)$ ▷ Train the deep neural network with hyperparameter configuration $\boldsymbol{\theta}_i$ and test the performance on task t_k's validation set as its performance on task t_k
    end for
  end for
  Initialize a temp variable $\mathbf{\Theta}^{(q)}$ for current iteration
  for each task t_k in the task pool $\mathcal{T}$ **do**
    The fitness value for each individual $\boldsymbol{\theta}_i$ on task t_k is $\Phi_i \leftarrow \left(\Psi_{k,i} + \frac{\sum_{j \neq k} \xi_j^k \Psi_{j,i}}{K}\right)$
    Select I_k individuals from the population group $\mathbf{\Theta}$ copied to $\mathbf{\Theta}^{(q)}$ ▷ Apply natural selection to the population group $\mathbf{\Theta}$ due to $\mathbf{\Phi}$
  end for
  $\mathbf{\Theta} \leftarrow \mathbf{\Theta}^{(q)}$ ▷ Assign the new population as the operating variable
  Apply *exploit* and *explore* procedure to $\mathbf{\Theta}$
end for
return the tuned $\mathbf{\Theta}$

hyperparameter configuration $\boldsymbol{\theta}$ on task t_k. The optimization problem for the hyperparameter tuning being solved by the population-based evolution process is as follows:

$$\arg\max_{\boldsymbol{\theta}_i \in \mathbf{\Theta}} \psi_k(\boldsymbol{\theta}_i) \tag{1}$$

where $\boldsymbol{\theta}_i$ and $\mathbf{\Theta}$ denote a specific hyperparameter configuration instance and the population of hyperparameter configuration instances being kept by the algorithm, respectively. We can maintain a matrix $\mathbf{\Psi}$ and its element is defined as follows:

$$\Psi_{k,i} = \psi_k(\boldsymbol{\theta}_i) \tag{2}$$

where the matrix element $\Psi_{k,i}$ is a real value representing the performance of the hyperparameter configuration $\boldsymbol{\theta}_i$ on task t_k. This performance can be obtained from the trained deep neural network with hyperparameter configuration $\boldsymbol{\theta}_i$. Although the evaluation result is the performance with inevitable noises due to the stochastic parameter optimizing process, such fluctuations can be ignored for their slight influences.

A. Parallel Tuning for Initial Tasks

The initial hyperparameter configuration population $\mathbf{\Theta}$ is created for a task pool $\mathcal{T}$ with K tasks. I_k individuals are generated for optimizing task t_k and there are I hyperparameter configuration instances in the population, where $I = \sum_{k=1}^{K} I_k$. The population $\mathbf{\Theta}$ is evaluated to compute the performance matrix $\mathbf{\Psi}$. The goal is to find the suitable hyperparameter configurations for each task while ensuring the generalization of this hyperparameter configuration for other related tasks. Thus for each task t_k $(k = 1, 2, \cdots, K)$, the fitness value for selecting an individual $\boldsymbol{\theta}_i$ in $\mathbf{\Theta}$ is $\left(\Psi_{k,i} + \frac{\sum_{j \neq k} \xi_j^k \Psi_{j,i}}{K}\right)$, where ξ_j^k denotes the degree of relatedness between the target task t_k and other task t_j. This is designed to search hyperparameter configurations that perform the best on the current task and yield good generalization on other related tasks. Given that there is little prior knowledge about the relationship between tasks, it is difficult to measure the degree of relatedness quantitatively. So, the relatedness can only be estimated approximately. For instance, the KullbackLeibler divergence [60] measures the relatedness using the information difference. Spearman's rank correlation coefficient estimates each task pair for their relatedness [63], [64]. The Pareto-dominance relation has also been proposed to estimate the degree of relatedness [59]. In this work, Spearman's rank correlation coefficient is used to estimate the relatedness between tasks because the exact performance values may be

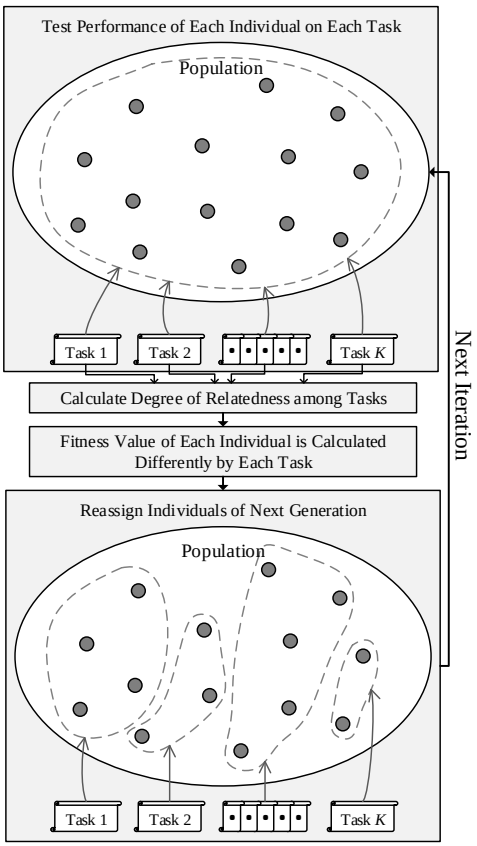


Fig. 1. The parallel hyperparameter tuning framework for initial tasks.

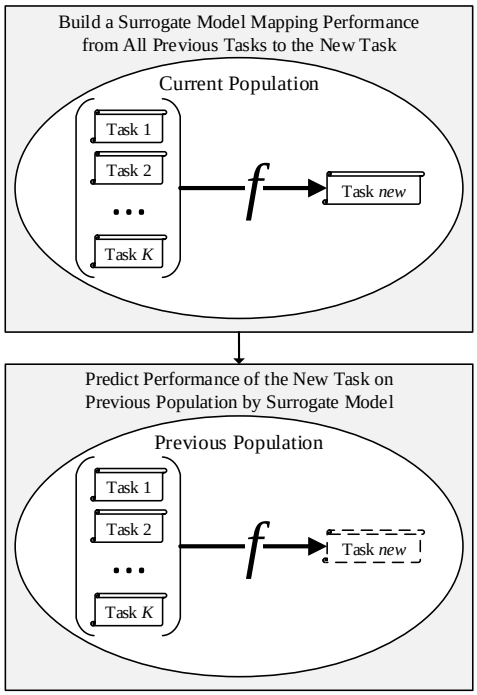


Fig. 2. The sequential hyperparameter tuning framework for new-added tasks.

various due to the different tasks and different metrics. We aim to find the most related tasks, so the non-parametric Spearman's rank correlation is suitable.

Let $\mathbf{\Theta}^{(q)}$ be the evolutionary population at the q^{th} ($q = 1, 2, \cdots$) iteration. The evolutionary strategy in [65] is adapted to generate $\mathbf{\Theta}^{(q)}$ from the previous iteration $\mathbf{\Theta}^{(q-1)}$. Let $\mathbf{\Psi}^{(q)}$ be the performance of $\mathbf{\Theta}^{(q)}$ and this meta-data at each iteration is saved for the new-added tasks. The parallel phase for initial tasks' hyperparameter tuning is shown in Algorithm 2.

B. Sequential Tuning for Subsequent New-added Tasks

For each subsequent new-added task, the meta-data from previous tasks' tuning experience is utilized to initialize the hyperparameter tuning for the new task. We assume that the K previous tasks have found the optimal hyperparameter configurations because the fitness values of their solutions are stable and optimal. With the well-performing hyperparameter configuration of the K previous tasks in the task pool, the sequential phase of tuning uses the existing experience to handle a new-added task t_{new}. Suppose that solution of these K tasks achieved stability using Q iterations, the history data is recorded as $\left\{\mathbf{\Theta}^{(1)}, \mathbf{\Theta}^{(2)}, \cdots, \mathbf{\Theta}^{(Q)}\right\}$ and $\left\{\mathbf{\Psi}^{(1)}, \mathbf{\Psi}^{(2)}, \cdots, \mathbf{\Psi}^{(Q)}\right\}$ for new-add tasks to apply meta-learning, which can be regarded as the learning trajectories of all the individuals in the population.

1) Building a Surrogate Model: The performance of the new task t_{new} is tested using the latest hyperparameter configuration $\mathbf{\Theta}^{(Q)}$ and presented as $\psi_{new}\left(\mathbf{\Theta}^{(Q)}\right)$. There is a set of performances for all individuals of previous tasks' latest population $\mathbf{\Theta}^{(Q)}$. Assume that the new task will be related to some previous tasks in the task pool. We hope that the new task's performance can be estimated through the previous tasks' performance by learning a combination of them. Hence, we build a mapping function f which takes $\mathbf{\Psi}^{(Q)}$ on K previous tasks as the input and is expected to predict the new task's performance. This mapping function is expressed as follows:

$$f\left(\mathbf{\Psi}^{(Q)}\right) = \tilde{\psi}_{new}\left(\mathbf{\Theta}^{(Q)}\right) \approx \psi_{new}\left(\mathbf{\Theta}^{(Q)}\right) \tag{3}$$

Here, the mapping function f is the surrogate model we build. Building surrogate models on meta-level aims to transfer information across domains [66], tasks [67]–[69], and models [70]. In our paper, we build the surrogate model based on other tasks' historical hyperparameter configurations and performances to help initialize the population for the new-added task. This surrogate model itself can be regarded as a meta-information extractor and predictor. There are many surrogate models available in mining the meta-data from the previous tasks and applying it to estimate the new task. In our experiments, three surrogate models (simple weighted sum, support vector regression, and deep neural network) are tried to find the most effective surrogate model (see Fig. 6).

2) Performance Estimation on New-added Tasks using Historical Populations: After building the surrogate model f that fits the mapping from $\mathbf{\Psi}^{(Q)}$ to $\psi_{new}\left(\mathbf{\Theta}^{(Q)}\right)$, the performance of solutions in the history populations on the new tasks are

Algorithm 3 New-added Tasks' Hyperparameter Tuning

Restore the history populations $\left\{\mathbf{\Theta}^{(1)}, \mathbf{\Theta}^{(2)}, \cdots, \mathbf{\Theta}^{(Q)}\right\}$ and their performance $\left\{\mathbf{\Psi}^{(1)}, \mathbf{\Psi}^{(2)}, \cdots, \mathbf{\Psi}^{(Q)}\right\}$ in previous tasks' hyperparameter tuning process
for each new-add task t_{new} **do**
  for each individual $\boldsymbol{\theta}_i$ in $\mathbf{\Theta}^{(Q)}$ **do**
    Get $\psi_{new}\left(\boldsymbol{\theta}_i\right)$ ▷ Train the deep neural network with hyperparameter configuration $\boldsymbol{\theta}_i$ and test the performance on new task's validation set as its performance on the new task
  end for ▷ Present the performance of population $\mathbf{\Theta}^{(Q)}$ on t_{new} as $\psi_{new}\left(\mathbf{\Theta}^{(Q)}\right)$ for simplicity
  Build a surrogate model $f\left(\mathbf{\Psi}^{(Q)}\right) = \tilde{\psi}_{new}\left(\mathbf{\Theta}^{(Q)}\right) \approx \psi_{new}\left(\mathbf{\Theta}^{(Q)}\right)$
  for each history population $\mathbf{\Theta}^{(q)}$ and corresponding performance $\mathbf{\Psi}^{(q)}$ **do**
    Use the surrogate model to get the predicted performance $f\left(\mathbf{\Psi}^{(q)}\right)$ on t_{new}
  end for
  Determine the number for individuals I_{new} for t_{new}
  $\mathbf{\Theta}_{new} = \{\}$ ▷ Initialize the initial hyperparameter population group $\mathbf{\Theta}_{new}$ for t_{new}
  for each iteration in range I_{new} **do**
    $\boldsymbol{\theta}_{temp} \leftarrow \arg\max_{\boldsymbol{\theta}_i} f\left(\mathbf{\Psi}^{(:)}_{:,i}\right)$ *for* $\boldsymbol{\theta}_i$ *not in* $\mathbf{\Theta}_{new}$ ▷ Select the remaining best-predicted individual from all members in all iterations (1 to Q)
    $\mathbf{\Theta}_{new} \leftarrow \mathbf{\Theta}_{new} \cup \{\boldsymbol{\theta}_{temp}\}$
  end for
  Apply original population-based evolutionary method for t_{new}
end for
return $\mathbf{\Theta}_{new}$ for each new-added task

estimated without implementation. For any history population at iteration q, the estimated performance is obtained by $\tilde{\psi}_{new}\left(\mathbf{\Theta}^{(q)}\right) = f\left(\mathbf{\Psi}^{(q)}\right) \approx \psi_{new}\left(\mathbf{\Theta}^{(q)}\right)$. Hence, individuals from previous tasks' populations can be chosen for the new task t_{new}. Then, solutions yielding the largest predicted performances from previous tasks' history populations is used to initialize individuals for the new task. In this way, the hyperparameter tuning process of new tasks is accelerated through multi-task collaboration.

By fitting a mapping relation from the performance of previous tasks to the new task in the same population, the surrogate model learns to extract meta-information about both previous tasks and the new task. Then, when the surrogate model is used to predict the performance of the new task given the performance of the previous tasks in other populations. Similar concepts have been proposed to deal with Bayesian optimization [71], [72] and few-shot learning [73]. In this work, we use it to accelerate the population-based hyperparameter tuning process.

IV. Experiments

In experiments, 10 image classification tasks are used: *CIFAR-10*, *CIFAR-100* [74], *Devanagari Handwritten Character* [75], *FashionMNIST* [76], *MNIST* [77], *notMNIST* [78], *Food-10* [79], *Oxford-IIIT Pet* [80], *STL10* [81], and *SVHN* [82]. Table II shows the information of the datasets, including the number of classes, the number of training and testing samples, and the maximum imbalance ratio. In the experiment, all the images are resized to ensure that the same deep neural network architecture can process the data from different tasks. Two test standards are shown in experiments: the performance

TABLE II
Characteristics of Image Datasets

Datasets	Characteristic			
	#Classes	*#Training*	*#Testing*	*#Imbalance*
CIFAR-10	10	50000	10000	1.00
CIFAR-100	100	50000	10000	1.00
Devanagari	46	78200	13800	1.00
Fashion MNIST	10	60000	10000	1.01
MNIST	10	60000	10000	1.24
notMNIST	10	$\approx 500k$	$\approx 19k$	1.00
Food-101	101	75750	25250	1.00
Oxford-IIIT Pets	37	3680	3669	1.08
STL10	10	$\approx 100k$	8000	1.00
SVHN	10	$\approx 73k$	$\approx 26k$	3.03

of tuned hyperparameter configurations on each task and the transfer ability to the other tasks. To test the transfer ability, the trained deep neural network on the original task will be tested on other tasks after adding additional layers. We add layers on top of the network and freeze those weight values of the original layers. The previously trained deep neural network can be treated as a feature extractor. This is to determine whether the deep neural network that performs well on the original task is a suitable feature extractor for other tasks.

The default deep neural network's architecture in the deep learning framework is shown in Fig. 3. In experiments, the learning rate, leakyReLU rates [83], dropout rates [84], numbers of filters in convolutional layers [85], and numbers of hidden neurons in fully-connected layers are tuned hyperparameters. During the tuning process, the deep neural network's performance on each task is estimated by the corresponding classification accuracy on the validation set and the final

TABLE III
HYPERPARAMETER TUNING PERFORMANCE ON 10 TASKS

Method \ Task	CIFAR-10	CIFAR-100	Devanagari	Fashion MNIST	MNIST	notMNIST	Food-101	Pets	STL10	SVHN
PBT+PHTMC	0.7695 (±0.0062)	**0.5991 (±0.0074)**	**0.9829 (±0.0033)**	0.8602 (±0.0116)	**0.9794 (±0.0039)**	**0.9807 (±0.0027)**	**0.7272 (±0.0114)**	**0.8314 (±0.0104)**	**0.6649 (±0.0091)**	**0.9384 (±0.0076)**
CMA-ES+PHTMC	**0.7815 (±0.0038)**	0.5880 (±0.0025)	0.9805 (±0.0036)	**0.8681 (±0.0099)**	0.9682 (±0.0040)	0.9788 (±0.0063)	0.7111 (±0.0049)	0.8297 (±0.0096)	0.6510 (±0.0117)	0.9329 (±0.0093)
PBT	0.7531 (±0.0078)	0.5864 (±0.0013)	0.9728 (±0.0046)	0.8400 (±0.0081)	0.9583 (±0.0100)	0.9741 (±0.0029)	0.7040 (±0.0042)	0.8112 (±0.0060)	0.6351 (±0.0039)	0.9298 (±0.0012)
CMA-ES	0.7558 (±0.0040)	0.5798 (±0.0073)	0.9710 (±0.0054)	0.8504 (±0.0090)	0.9600 (±0.0093)	0.9691 (±0.0042)	0.7022 (±0.0076)	0.8124 (±0.0067)	0.6410 (±0.0041)	0.9207 (±0.0039)
BO-GP	0.7471 (±0.0082)	0.5853 (±0.0066)	0.9638 (±0.0088)	0.8411 (±0.0229)	0.9558 (±0.0120)	0.9533 (±0.0265)	0.6977 (±0.0074)	0.8059 (±0.0088)	0.6413 (±0.0091)	0.9235 (±0.0114)
TPE	0.7563 (±0.0050)	0.5887 (±0.0039)	0.9704 (±0.0064)	0.8545 (±0.0049)	0.9597 (±0.0083)	0.9629 (±0.0040)	0.7108 (±0.0072)	0.8185 (±0.0179)	0.6499 (±0.0062)	0.9375 (±0.0087)
Hyperband	0.7554 (±0.0038)	0.5639 (±0.0067)	0.9771 (±0.0059)	0.8417 (±0.0044)	0.9679 (±0.0051)	0.9647 (±0.0044)	0.7044 (±0.0039)	0.8109 (±0.0060)	0.6543 (±0.0044)	0.9257 (±0.0026)
BO-HB	0.7567 (±0.0061)	0.5708 (±0.0028)	0.9806 (±0.0066)	0.8547 (±0.0028)	0.9696 (±0.0060)	0.9711 (±0.0065)	0.7141 (±0.0022)	0.8135 (±0.0042)	0.6637 (±0.0060)	0.9317 (±0.0058)

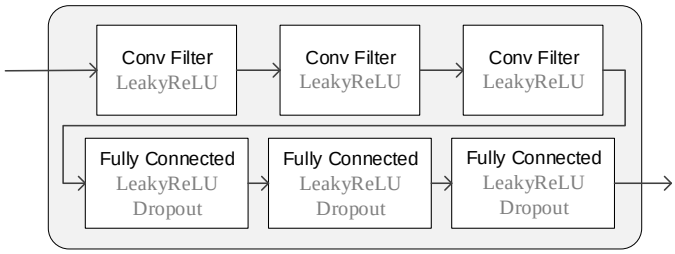


Fig. 3. The deep neural network architecture and corresponding hyperparameters need to be tuned. The input data flows into the deep neural network structure and flows out as the classification label.

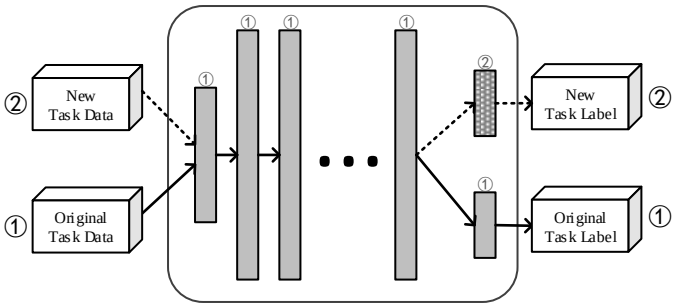


Fig. 4. The transfer ability measurement. When applying other tasks in the trained deep neural network, the original layers' weights are all frozen, which can be regarded as a feature extractor. Only the new layers' weights will be optimized.

performance is evaluated by the corresponding classification accuracy on the testing set. The learning rate (lr) is uniformly distributed between 10^{-1} and 10^{-5} on an exponential scale. Dropout and leakyReLU rates are distributed between 0 and 1. The number of filters in each convolutional (Conv) layer is selected between 4 and 64 while the number of hidden neurons on each fully-connected (FC) layer is selected between 16 and 256. For the natural selection process, for each task t_k, I_k is set to 10, so the total population size I is 100. Each experiment is replicated 10 times, each time the same random seed is applied to different methods. The average accuracy and the mean absolute deviation are recorded.

TABLE IV
THE HYPERPARAMETER (HP) SPACE

HP	Range	Type	Description
lr	$[10^{-5}, 10^{-1}]$	float	Optimization step size.
Dropout	$[0, 1]$	float	Probability of training a neuron.
leakyReLU	$[0, 1]$	float	The negative slope coefficient.
FC	$[16, 256]$	integer	Number of hidden neurons.
Conv	$[4, 64]$	integer	Number of filters.

A. Initial Tasks' Hyperparameter Tuning in Task Pool

In this part, we first compare hyperparameter tuning performances of the state-of-the-art hyperparameter tuning methods, including PBT [11], the CMA-ES [12], [86], the Bayesian optimization with Gaussian process (BO-GP) [87], the Tree-structured Parzen Estimator (TPE) [9], [88], Hyperband [22], and Bayesian Optimization with Hyperband (BO-HB) [89] on the 10 given tasks. For the population-based evolutionary strategies, the population size is set to 100. We combine the PHTMC with the population-based evolutionary strategies to illustrate its improvement. The BO-GP method is an iterative method that adaptively recovers the regret bounds of GP-UCB [90]. Its initial length-scale is set to 1, and the initial norm bound is set to 2, the tradeoff factor is 0.1. The TPE optimizes the expected improvement to help select the hyperparameter configurations. The percentile is set to 15%, and the candidate number is also set to 100. The Hyperband creates buckets with equal hyperparameter configuration numbers and limited resources. After each iteration is finished, it selects the top configurations and continues training them with increased resources. The selectivity is set to 33.3%. The BO-HB is a combination of Bayesian optimization and the original Hyperband. It replaces the random selection with the Bayesian optimization at each iteration. The setting of BO-HB in our experiment is the same as the Hyperband. Table III shows that performances can be improved when the PHTMC is applied. Hyperparameter tuning with multi-task collaboration yields better performance when there is a high degree of relatedness among tasks. Furthermore, Bayesian optimization methods

TABLE V
TRANSFER ABILITY POPULATION-BASED EVOLUTIONARY STRATEGY WITH (UPPER) AND WITHOUT (LOWER) THE PHTMC

From \ To	CIFAR-10	CIFAR-100	Devanagari	Fashion MNIST	MNIST	notMNIST	Food-101	Pets	STL10	SVHN
CIFAR-10	×	**0.2009 (±0.0067)** 0.1393 (±0.0087)	**0.8083 (±0.0224)** 0.7173 (±0.0716)	**0.7367 (±0.0130)** 0.6347 (±0.0218)	**0.7897 (±0.0317)** 0.7179 (±0.0255)	**0.7953 (±0.0214)** 0.7636 (±0.0440)	**0.6306 (±0.0162)** 0.5825 (±0.0374)	**0.07572 (±0.0071)** 0.7216 (±0.0122)	**0.4541 (±0.0089)** 0.4333 (±0.0145)	**0.4858 (±0.0055)** 0.4765 (±0.0020)
CIFAR-100	**0.3815 (±0.0132)** 0.3504 (±0.0152)	×	**0.7816 (±0.0159)** 0.7397 (±0.0204)	0.6643 (±0.0059) **0.6649 (±0.0052)**	**0.6824 (±0.0357)** 0.6242 (±0.0251)	**0.6866 (±0.0170)** 0.6651 (±0.0239)	**0.6585 (±0.0158)** 0.6029 (±0.0182)	**0.7482 (±0.0155)** 0.7140 (±0.0149)	**0.3581 (±0.0119)** 0.3537 (±0.0169)	0.4151 (±0.0120) **0.4167 (±0.0112)**
Devanagari	**0.1847 (±0.0160)** 0.1618 (±0.0328)	**0.1265 (±0.0048)** 0.1186 (±0.0101)	×	**0.7091 (±0.0213)** 0.6371 (±0.0218)	**0.9025 (±0.0044)** 0.8935 (±0.0189)	**0.8329 (±0.0177)** 0.8249 (±0.0082)	**0.5618 (±0.0159)** 0.5342 (±0.0140)	**0.7055 (±0.0104)** 0.6646 (±0.0126)	**0.1875 (±0.0149)** 0.1797 (±0.0191)	**0.2519 (±0.0195)** 0.2502 (±0.0146)
Fashion MNIST	**0.2367 (±0.0070)** 0.2291 (±0.0128)	**0.1365 (±0.0021)** 0.1325 (±0.0042)	**0.6950 (±0.0144)** 0.6394 (±0.0354)	×	**0.7189 (±0.0254)** 0.6857 (±0.0100)	**0.7507 (±0.0149)** 0.7387 (±0.0218)	**0.5586 (±0.0152)** 0.5396 (±0.0187)	**0.6941 (±0.0061)** 0.6581 (±0.0115)	**0.2679 (±0.0092)** 0.2672 (±0.0095)	**0.2496 (±0.169)** 0.2486 (±0.0156)
MNIST	**0.2267 (±0.0248)** 0.1773 (±0.0034)	**0.1409 (±0.0011)** 0.1307 (±0.0036)	**0.8900 (±0.0193)** 0.8758 (±0.0110)	**0.7007 (±0.0104)** 0.6731 (±0.0240)	×	**0.6723 (±0.0107)** 0.5158 (±0.0329)	**0.5573 (±0.0218)** 0.5332 (±0.0264)	**0.7232 (±0.0157)** 0.6828 (±0.0169)	**0.2563 (±0.0176)** 0.2530 (±0.0219)	**0.3372 (±0.0075)** 0.3290 (±0.0087)
notMNIST	**0.2742 (±0.0086)** 0.2461 (±0.0169)	**0.1595 (±0.0046)** 0.1451 (±0.0095)	**0.9428 (±0.0041)** 0.9217 (±0.0195)	**0.7969 (±0.0087)** 0.7823 (±0.0081)	**0.9341 (±0.0144)** 0.8525 (±0.0229)	×	**0.5075 (±0.0273)** 0.4310 (±0.0348)	**0.6531 (±0.0221)** 0.6186 (±0.0207)	**0.2921 (±0.0101)** 0.2820 (±0.0151)	**0.3093 (±0.0340)** 0.2584 (±0.0203)
Food-101	**0.3771 (±0.0038)** 0.3430 (±0.0073)	**0.1973 (±0.0122)** 0.1639 (±0.0126)	**0.8117 (±0.0276)** 0.7866 (±0.0361)	**0.7527 (±0.0099)** 0.7122 (±0.0307)	**0.7502 (±0.0345)** 0.7377 (±0.0296)	**0.7005 (±0.0302)** 0.6929 (±0.0078)	×	**0.7316 (±0.0255)** 0.6441 (±0.0197)	**0.3332 (±0.0105)** 0.3147 (±0.0066)	**0.4031 (±0.0079)** 0.3702 (±0.0031)
Pets	**0.3571 (±0.0203)** 0.3412 (±0.0142)	**0.1437 (±0.0105)** 0.1375 (±0.0132)	**0.7456 (±0.0368)** 0.7179 (±0.0251)	**0.7189 (±0.0268)** 0.6838 (±0.0363)	**0.8505 (±0.0176)** 0.8125 (±0.0133)	**0.7348 (±0.0192)** 0.7264 (±0.0124)	**0.5824 (±0.0310)** 0.5708 (±0.0236)	×	**0.2674 (±0.0280)** 0.2312 (±0.0271)	**0.3535 (±0.0212)** 0.3128 (±0.0156)
STL10	**0.3279 (±0.0109)** 0.3207 (±0.0229)	**0.1544 (±0.0094)** 0.1336 (±0.0069)	**0.7712 (±0.0408)** 0.7348 (±0.0346)	0.7312 (±0.0147) **0.7342 (±0.0311)**	**0.8470 (±0.0245)** 0.8445 (±0.0291)	**0.7678 (±0.0255)** 0.7470 (±0.0282)	**0.6877 (±0.0376)** 0.6443 (±0.0352)	**0.6866 (±0.0255)** 0.6381 (±0.0260)	×	**0.3407 (±0.0211)** 0.3251 (±0.0298)
SVHN	**0.2373 (±0.0105)** 0.2307 (±0.0050)	**0.1450 (±0.0013)** 0.1418 (±0.0029)	**0.7306 (±0.263)** 0.6915 (±0.464)	**0.6594 (±0.0128)** 0.6439 (±0.0095)	**0.7884 (±0.0122)** 0.7741 (±0.0075)	**0.6841 (±0.0333)** 0.6529 (±0.0093)	**0.6695 (±0.0323)** 0.6092 (±0.0412)	**0.7122 (±0.0314)** 0.6643 (±0.0351)	**0.2424 (±0.0098)** 0.2376 (±0.0166)	×

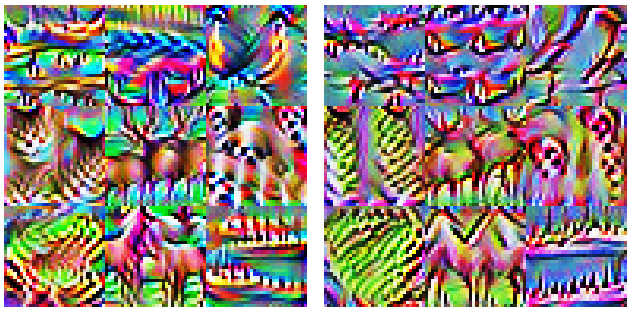
yield considerable performance in a few iterations. However, its further improvement is smaller than population-based hyperparameter tuning methods. This is because the fluctuation noise in deep neural network measurement has a greater impact on such methods. As a result, the population-based hyperparameter tuning methods yield better performances than that of the Bayesian optimization methods. Within population-based hyperparameter tuning methods, the PBT yields better results than the CMA-ES. However, the CMA-ES with the PHTMC outperforms the original PBT, which illustrates that multi-task collaboration further improves the performances of population-based hyperparameter tuning methods.

TABLE VI
SIGNIFICANCE TEST OF THE IMPROVEMENTS

	PBT	CMA-ES
t-test signed rank	10/10	9/10
average p-value	1.1202E-4	6.3489E-4

The significance test of the improvements that the PHTMC brings is shown in Table VI. We carry out one-tailed t-tests to verify whether the performance of the PBT and CMA-ES improves when the PHTMC is applied to them. In the hypothesis tests, the null hypothesis is that the PHTMC does not improve the accuracy for the base hyperparameter tuning method. The alternative hypothesis is that the PHTMC improves the performance of the base hyperparameter tuning method. The ψ_p denotes the classification accuracy of the base hyperparameter tuning method with the PHTMC. The ψ_o denotes the accuracy of the original hyperparameter tuning method. The null hypothesis is defined as: $H_0 : \psi_p \leq \psi_o$, and the alternative hypothesis is: $H_1 : \psi_p > \psi_o$. The ten tasks are reported, and the significance level is set to be 0.01. The first row in Table VI is the number of rejections for H_0, which indicates the number of times that the PHTMC improves the performance of the base hyperparameter tuning method. The p-values on both methods are smaller than 0.001, which means the base methods with the PHTMC outperform the original ones with at least 99.9% confidence.

We also compare the transfer ability of the original single-task population-based hyperparameter tuning method with the PHTMC framework (see Fig. 4). Results of the population-based evolutionary strategy with and without the PHTMC framework are shown in Table V. There are two elements in each cell of Table V, the upper one is the transfer ability of



(a) with PHTMC (b) Original

Fig. 5. The visualization of the trained deep neural network threads. They are the 9 classes in CIFAR-10, including *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, and *ship*.

the hyperparameter configuration with the PHTMC, while the lower one is the transfer ability of the hyperparameter configurations with the original population-based hyperparameter tuning. These results show transfer abilities from one task to another task, which also indicates the generalization abilities of the hyperparameter configuration. The performance of the transferred task is computed using the classification model trained on the source task. Table V shows that the multi-task collaboration framework yields better performances for both the transfer ability and the generalization ability, which demonstrate more general and less data bias of the trained deep neural network structures. The multi-task collaborative hyperparameter tuning yields better performance especially when the collaborative tasks are highly similar.

TABLE VII
THE AVERAGE HYPERPARAMETER CONFIGURATIONS OF THE TWO POPULATIONS

Method \ Type	lr	FC	Conv	Dropout	leakyReLU
with PHTMC	0.0169	160	19	0.7154	0.2935
Original	0.0220	172	23	0.6639	0.2969

In experiments, methods with multi-task collaboration tend to yield simpler networks. We record the average results of different hyperparameter types of the two populations in Table VII. From Table VII, both the number of hidden units in fully-connected layers and the number of filters in convolutional layers are smaller on average compared with single-task population-based hyperparameter tuning. In addition, the average dropout rates are larger with the PHTMC. These phenomena are consistent with findings in [91] that models with better generalization abilities are usually relatively simpler.

Moreover, we visualize the deep neural network threads to compare the tuned hyperparameter configurations [92]. We use Lucid [1] to do the visualization on the output layer on CIFAR-10. Fig. 5 shows the first 9 classes in CIFAR-10. They are *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*. Both the original PBT and PBT with the PHTMC give a reasonable interpretation of the classified neurons. PBT with the PHTMC does it better in the classes, for example, the *bird*, *cat*, and *dog*. It suggests that the deep neural networks tuned by the PHTMC yield better generalization ability.

[1]https://github.com/tensorflow/lucid

B. Subsequent New-added Tasks' Hyperparameter Tuning

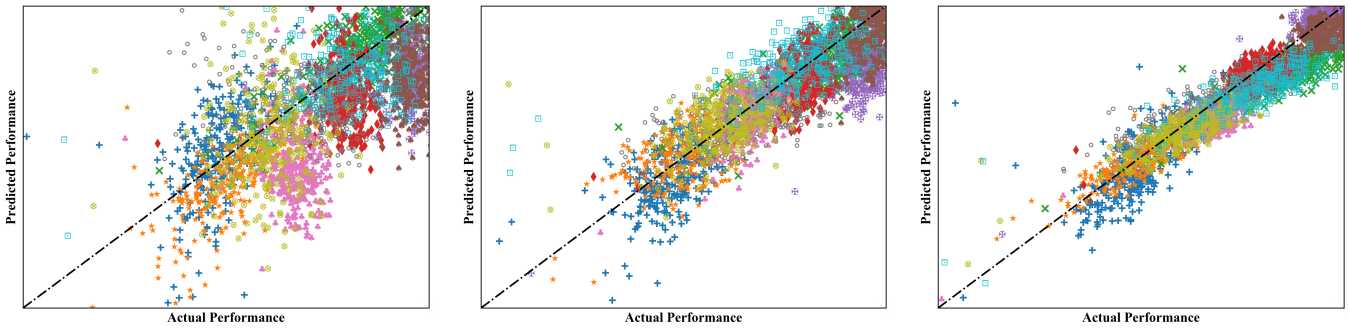
In this part, we test the situation that new-added tasks are being tuned while all initial tasks in the task pool have been tuned. When there are new tasks to be tuned, the previous meta-information is used to initialize new-added tasks. For each new-add task, a surrogate model is built to evaluate the new-add task's performance with all previous tasks' performances. Three types of surrogate models have been compared: the weighted sum model, the support vector regression model (SVR), and the multi-layer perceptron neural network model (MLP). We expect to figure out which model extracts more precise meta-information and helps to initialize the new-added tasks hyperparameter tuning process. For each new-added task, hyperparameter configurations yielding the best results from all historical populations will be used as the initial population for hyperparameter tuning.

Fig. 6 show scatter diagrams of predicted results of three surrogate models. Each task is represented by a corresponding marker. The weighted sum model yields the highest error rate among the three models. It cannot deal with the extreme value as well. It shows that the weighted sum surrogate model is unsuitable to predict the new task's performance. Both the SVR and the MLP accurately predict the performances of new tasks. However, the MLP yields fewer exceptional points, i.e. this model reduces the likelihood of a prediction going wrong.

Hyperparameter tuning results are shown in Table VIII. In this experiment, each task's result is obtained by regarding it as a new-added task after tuning the rest of the 9 tasks as initial tasks in the task pool. Table VIII shows that the MLP is the most suitable surrogate model for extracting and utilizing the meta-information. The SVR shows a slightly worse result than the MLP. In contrast, the simple weighted sum model may not be able to use the meta-information between tasks well and therefore yields the worst performance. Compared with the baseline, which is the same hyperparameter tuning method for the single-task hyperparameter tuning. It is shown that the surrogate models help the new-added task to yield a better performance directly learned together with the other 9 tasks. This shows that there is some common meta-information between tasks and the surrogate model can utilize the meta-information.

C. Visualized Analysis of the Hyperparameter Tuning Results

Visualization of the distribution of hyperparameter configurations for both the multi-task collaboration hyperparameter tuning and the single-task hyperparameter tuning are performed and shown in Fig. 7. Hyperparameter configuration individuals, as presented in Section III, are vectors in a n-dimensional real space. To obtain a 2-dimensional visualization, the t-distributed Stochastic Neighbor Embedding (t-SNE) [93] is adopted to compress n-dimensional hyperparameter configurations to 2-dimensional data representation. 2-dimensional visualization results are shown in Fig. 7.

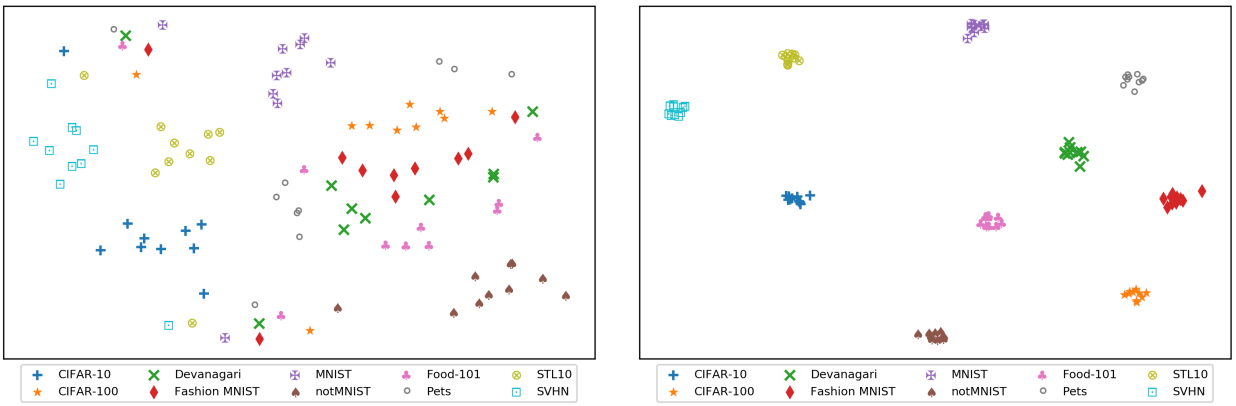


(a) Weighted sum (b) Support vector regression (c) Multi-layer perceptron

Fig. 6. Predicted results of three surrogate models. Different markers represent the corresponding tasks.

TABLE VIII
MULTI-TASK COLLABORATION HYPERPARAMETER TUNING WITH DIFFERENT SURROGATE MODELS

Model \ Task	CIFAR-10	CIFAR-100	Devanagari	Fashion MNIST	MNIST	notMNIST	Food-101	Pets	STL10	SVHN
Weighted	0.7905 (±0.0047)	0.6279 (±0.0069)	0.9886 (±0.0031)	0.8644 (±0.0030)	0.9834 (±0.0046)	0.9852 (±0.0088)	0.7432 (±0.0122)	0.8332 (±0.0090)	0.6687 (±0.0046)	0.9400 (±0.0015)
SVR	0.7973 (±0.0014)	0.6267 (±0.0063)	0.9892 (±0.0032)	0.8706 (±0.0059)	0.9869 (±0.0052)	0.9864 (±0.0049)	0.7350 (±0.0082)	0.8421 (±0.0003)	0.6712 (±0.0102)	0.9478 (±0.0024)
MLP	**0.8041 (±0.0069)**	**0.6350 (±0.0050)**	**0.9913 (±0.0026)**	**0.8771 (±0.0033)**	**0.9881 (±0.0100)**	**0.9902 (±0.0031)**	**0.7554 (±0.0048)**	**0.8514 (±0.0013)**	**0.6766 (±0.0040)**	**0.9557 (±0.0086)**
Baseline	0.7531 (±0.0078)	0.5864 (±0.0013)	0.9728 (±0.0046)	0.8400 (±0.0081)	0.9583 (±0.0100)	0.9741 (±0.0029)	0.7040 (±0.0042)	0.8112 (±0.0060)	0.6351 (±0.0039)	0.9298 (±0.0012)



(a) Visualization of the multi-task collaboration hyperparameter tuning (b) Visualization of the single-task hyperparameter tuning

Fig. 7. The visualization of the hyperparameter configuration distribution in the population-based evolutionary strategy for hyperparameter tuning. Individuals using the original single-task population-based hyperparameter tuning method are more concentrated in each population's own task. In contrast, individuals for the multi-task collaborative population-based hyperparameter tuning method are more scattered over the space and between tasks.

The bias in a certain task may lead to poor overall performance. For example, the initial bias in exploring some regions more than the other regions makes the hyperparameter methods arrange more threads to these explored regions, which in turn promotes the expected improvement of these regions and reinforces the bias of searching these regions. The positive feedback phenomenon leads to the results shown in Fig. 7a. The multi-task collaborative population-based hyperparameter tuning method avoids such bias and gets more scattered results over the whole hyperparameter space (see Fig. 7b). This means that the PHTMC has a better transfer ability which may avoid data bias to a single task and yield better generalization ability.

D. Resource Ratio for PHTMC

The main cost of the whole training process lies in the training of the deep neural network threads. The PHTMC only operates on meta-data, whose size is minimal. For example, one deep neural network thread may have millions of parameters to optimize at each iteration, but its performance record is only a floating-point number. Therefore, the scale of the PHTMC is very tiny.

The PHTMC is an additional structure. It has a constant space occupation and time occupation. Fig. 8 shows that in numbers of deep neural network threads to be tuned, the different computational resource ratios that the PHTMC will

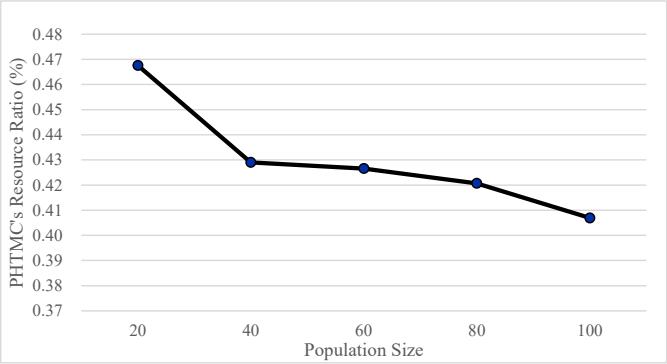


Fig. 8. The computational resource that the PHTMC occupies under different population sizes.

take. The PHTMC only takes up to 0.47% of the overall computational resource in the hyperparameter tuning, but it brings much more benefits.

E. Visualized Results of Autoencoder's Reconstruction

Moreover, we apply autoencoder (AE) to these datasets for image reconstruction. In this experiment, hyperparameter tuning is performed on the AE's architecture with the fully-connected layer's units being selected between 16 and 256 for each layer. Domains of other hyperparameters such as learning rate, leakyReLU rates, and dropout rates are the same as the experiments of the image-classification deep neural network in previous sections.

Fig. 9 shows original images and images being reconstructed by autoencoders with and without multi-task collaborations. The 10 rows in Fig. 9 represent 10 tasks mentioned above, consisting of images of objects, numbers, and characters. Images reconstructed by autoencoders learned with the multi-task collaboration are clearer and sharper. This is because tasks' meta-features can be shared across similar tasks, such as numbers, characters, and objects. As there are similar tasks in the task pool, their meta-information can be shared across tasks and the proposed multi-task collaboration model makes well use of it. The shared meta-features in different tasks help to select more subtle hyperparameter configurations for AE architectures in different tasks to yield higher generalization ability and better image reconstruction.

V. Conclusion

In this paper, we propose a population-based hyperparameter tuning framework combining multi-task collaboration in order to leverage and mine meta-data from hyperparameter tuning processes of similar tasks. This framework includes two parts. In the first part, all initial tasks are tuned together thus each task is with the other tasks' assistance. The second part deals with new tasks that when all previous tasks have been tuned. New-added tasks will be provided with good initialization from previous tasks' hyperparameter tuning experience.

We show the generalization by measuring the effectiveness of extracted features on other tasks. Compared with the original population-based evolutionary strategy which is aimed at hyperparameter tuning tasks one by one, the proposed framework with multi-task collaboration shows significant improvements on the generalization ability without losing the performance on the original task. For new-added tasks, our method makes use of the previous tasks' meta-data in hyperparameter tuning and yields better results.

The new-added tasks' initialization, however, is completely selected from previous tasks' existing hyperparameter configurations. It means that the initialization can only bring limited lift if the previous tasks' experience is not enough to give useful guidance. How to make a more diverse initialization based on previous tasks' limited experience will be the subject of future research. Moreover, an improvement should be considered to make the PHTMC framework efficiently work among low-related tasks.

References

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

[2] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, and A. Talwalkar, "Massively parallel hyperparameter tuning," *arXiv preprint arXiv:1810.05934*, 2018.

[3] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. ACM, 2015, p. 4.

[4] P. R. Lorenzo, J. Nalepa, L. S. Ramos, and J. R. Pastor, "Hyper-parameter selection in deep neural networks using parallel particle swarm optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2017, pp. 1864–1871.

[5] F. Ye, "Particle swarm optimization-based automatic parameter selection for deep neural networks and its applications in large-scale and high-dimensional data," *PloS one*, vol. 12, no. 12, p. e0188746, 2017.

[6] A. Rojas-Domínguez, L. C. Padierna, J. M. C. Valadez, H. J. Puga-Soberanes, and H. J. Fraire, "Optimal hyper-parameter tuning of svm classifiers with application to medical diagnosis," *Ieee Access*, vol. 6, pp. 7164–7176, 2017.

[7] P. Larrañaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Science & Business Media, 2001, vol. 2.

[8] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[9] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[10] M. Feurer, J. Springenberg, and F. Hutter, "Initializing bayesian hyper-parameter optimization via meta-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[11] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan *et al.*, "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017.

[12] I. Loshchilov and F. Hutter, "Cma-es for hyperparameter optimization of deep neural networks," *arXiv preprint arXiv:1604.07269*, 2016.

[13] N. Hansen, Y. Akimoto, and P. Baudis, "CMA-ES/pycma on Github," Zenodo, DOI:10.5281/zenodo.2559634, Feb. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2559634

[14] T. A. Gomes, R. B. Prudêncio, C. Soares, A. L. Rossi, and A. Carvalho, "Combining meta-learning and search techniques to select parameters for support vector machines," *Neurocomputing*, vol. 75, no. 1, pp. 3–13, 2012.

[15] M. Reif, F. Shafait, and A. Dengel, "Meta-learning for evolutionary parameter optimization of classifiers," *Machine learning*, vol. 87, no. 3, pp. 357–380, 2012.

[16] M. López-Ibánez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

(a) Original samples (b) With multi-task collaboration (c) Without multi-task collaboration

Fig. 9. Reconstruction results of autoencoders learned with and without multi-task collaboration.

[17] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2902–2911.

[18] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Denser: deep evolutionary network structured representation," *Genetic Programming and Evolvable Machines*, vol. 20, no. 1, pp. 5–35, 2019.

[19] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[20] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka: Automatic model selection and hyperparameter optimization in weka," in *Automated Machine Learning*. Springer, Cham, 2019, pp. 81–95.

[21] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, J. H. Moore *et al.*, "Automating biomedical data science through tree-based pipeline optimization," in *European conference on the applications of evolutionary computation*. Springer, 2016, pp. 123–137.

[22] 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.

[23] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Auto-sklearn: efficient and robust automated machine learning," in *Automated Machine Learning*. Springer, Cham, 2019, pp. 113–134.

[24] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[25] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 459–468.

[26] C.-H. Chen and C.-B. Liu, "Reinforcement learning-based differential evolution with cooperative coevolution for a compensatory neuro-fuzzy controller," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 10, pp. 4719–4729, 2017.

[27] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *Advances in Neural Information Processing Systems*, 2018, pp. 2016–2025.

[28] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown, "Towards an empirical foundation for assessing bayesian optimization of hyperparameters," in *NIPS workshop on Bayesian Optimization in Theory and Practice*, vol. 10, 2013, p. 3.

[29] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," *arXiv preprint arXiv:1605.07079*, 2016.

[30] Y. Xiao, H. Wang, and W. Xu, "Hyperparameter selection for gaussian process one-class classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 9, pp. 2182–2187, 2015.

[31] S. Xie, H. Zheng, C. Liu, and L. Lin, "Snas: stochastic neural architecture search," *arXiv preprint arXiv:1812.09926*, 2018.

[32] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International Conference on Machine Learning*, 2015, pp. 2113–2122.

[33] F. Pedregosa, "Hyperparameter optimization with approximate gradient," *arXiv preprint arXiv:1602.02355*, 2016.

[34] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," *arXiv preprint arXiv:1902.07638*, 2019.

[35] S. Forrest and M. Mitchell, "Relative building-block fitness and the building-block hypothesis," in *Foundations of genetic algorithms*. Elsevier, 1993, vol. 2, pp. 109–126.

[36] L. Altenberg, "The schema theorem and price's theorem," in *Foundations of genetic algorithms*. Elsevier, 1995, vol. 3, pp. 23–49.

[37] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 497–504.

[38] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[39] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," *arXiv preprint arXiv:1804.09081*, 2018.

[40] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[41] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[42] T. Desell, "Large scale evolution of convolutional neural networks using volunteer computing," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2017, pp. 127–128.

[43] S. Lessmann, R. Stahlbock, and S. F. Crone, "Optimizing hyperparameters of support vector machines by genetic algorithms," in *IC-AI*, 2005, pp. 74–82.

[44] C. Di Francescomarino, M. Dumas, M. Federici, C. Ghidini, F. M. Maggi, W. Rizzi, and L. Simonetto, "Genetic algorithms for hyperparameter optimization in predictive business process monitoring," *Information Systems*, vol. 74, pp. 67–83, 2018.

[45] Y.-J. Gong, J. Zhang, and Y. Zhou, "Learning multimodal parameters: A bare-bones niching differential evolution approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 2944–2959, 2017.

[46] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[47] Y.-L. Xu, X.-X. Li, D.-R. Chen, and H.-X. Li, "Learning rates of regularized regression with multiple gaussian kernels for multi-task learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5408–5418, 2018.

[48] C. Li, F. Wei, J. Yan, X. Zhang, Q. Liu, and H. Zha, "A self-paced regularization framework for multilabel learning," *IEEE transactions on*

neural networks and learning systems, vol. 29, no. 6, pp. 2660–2666, 2017.

[49] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *arXiv preprint arXiv:1810.04650*, 2018.

[50] 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.

[51] J. Vanschoren, "Meta-learning: A survey," *arXiv preprint arXiv:1810.03548*, 2018.

[52] S. Lin, "Rank aggregation methods," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 5, pp. 555–570, 2010.

[53] S. M. Abdulrahman, P. Brazdil, J. N. van Rijn, and J. Vanschoren, "Speeding up algorithm selection using average ranking and active testing by introducing runtime," *Machine learning*, vol. 107, no. 1, pp. 79–108, 2018.

[54] 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.

[55] J. Fürnkranz and J. Petrak, "An evaluation of landmarking variants," in *Working Notes of the ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, 2001, pp. 57–68.

[56] V. Perrone, R. Jenatton, M. Seeger, and C. Archambeau, "Multiple adaptive bayesian linear regression for scalable bayesian optimization with warm start," *arXiv preprint arXiv:1712.02902*, 2017.

[57] E. Nisioti, K. Chatzidimitriou, and A. Symeonidis, "Predicting hyperparameters from meta-features in binary classification problems," in *AutoML Workshop at ICML*, 2018.

[58] S. Ali and K. A. Smith-Miles, "A meta-learning approach to automatic kernel selection for support vector machines," *Neurocomputing*, vol. 70, no. 1-3, pp. 173–186, 2006.

[59] Z. Liang, J. Zhang, L. Feng, and Z. Zhu, "A hybrid of genetic transform and hyper-rectangle search strategies for evolutionary multi-tasking," *Expert Systems with Applications*, 2019.

[60] Y. Chen, J. Zhong, L. Feng, and J. Zhang, "An adaptive archive-based evolutionary framework for many-task optimization," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 3, pp. 369–384, 2019.

[61] W. Li, W. W. Ng, T. Wang, M. Pelillo, and S. Kwong, "Help: An lstm-based approach to hyperparameter exploration in neural network learning," *Neurocomputing*, vol. 442, pp. 161–172, 2021.

[62] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian evolution, the baldwin effect and function optimization," in *International Conference on Parallel Problem Solving from Nature*. Springer, 1994, pp. 5–15.

[63] X. Zheng, A. K. Qin, M. Gong, and D. Zhou, "Self-regulated evolutionary multitask optimization," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 16–28, 2019.

[64] B. Da, Y.-S. Ong, L. Feng, A. K. Qin, A. Gupta, Z. Zhu, C.-K. Ting, K. Tang, and X. Yao, "Evolutionary multitasking for single-objective continuous optimization: Benchmark problems, performance metric, and baseline results," *arXiv preprint arXiv:1706.03470*, 2017.

[65] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[66] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79–111, 2014.

[67] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, "Sequential model-free hyperparameter tuning," in *2015 IEEE international conference on data mining*. IEEE, 2015, pp. 1033–1038.

[68] ——, "Two-stage transfer surrogate model for automatic hyperparameter optimization," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2016, pp. 199–214.

[69] ——, "Scalable gaussian process-based transfer surrogates for hyperparameter optimization," *Machine Learning*, vol. 107, no. 1, pp. 43–78, 2018.

[70] M. Reif, F. Shafait, M. Goldstein, T. Breuel, and A. Dengel, "Automatic classifier selection for non-experts," *Pattern Analysis and Applications*, vol. 17, no. 1, pp. 83–96, 2014.

[71] J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. Ahn, "Bayesian model-agnostic meta-learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 7332–7342.

[72] M. Feurer, B. Letham, and E. Bakshy, "Scalable meta-learning for bayesian optimization," *arXiv preprint arXiv:1802.02219*, 2018.

[73] C. Finn, K. Xu, and S. Levine, "Probabilistic model-agnostic meta-learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 9516–9527.

[74] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[75] S. Acharya, A. K. Pant, and P. K. Gyawali, "Deep learning based large scale handwritten devanagari character recognition," in *2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*. IEEE, 2015, pp. 1–6.

[76] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[77] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[78] Y. Bulatov, "Notmnist dataset," *Google (Books/OCR), Tech. Rep.[Online]. Available: http://yaroslavvb.blogspot.it/2011/09/notmnist-dataset.html*, 2011.

[79] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101–mining discriminative components with random forests," in *European conference on computer vision*. Springer, 2014, pp. 446–461.

[80] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, "Cats and dogs," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3498–3505.

[81] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[82] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[83] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the International Conference on Machine Learning*, vol. 30, no. 1, 2013, p. 3.

[84] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[85] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[86] K. Nishida and Y. Akimoto, "Psa-cma-es: Cma-es with population size adaptation," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 865–872.

[87] F. Berkenkamp, A. P. Schoellig, and A. Krause, "No-regret bayesian optimization with unknown hyperparameters," *arXiv preprint arXiv:1901.03357*, 2019.

[88] M. Zhao and J. Li, "Tuning the hyper-parameters of cma-es with tree-structured parzen estimators," in *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, 2018, pp. 613–618.

[89] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1437–1446.

[90] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009.

[91] H. Wang, N. S. Keskar, C. Xiong, and R. Socher, "Identifying generalization properties in neural networks," *arXiv preprint arXiv:1809.07402*, 2018.

[92] C. B. Azodi, J. Tang, and S.-H. Shiu, "Opening the black box: Interpretable machine learning for geneticists," *Trends in Genetics*, 2020.

[93] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.