

Joint Learning of Neural Transfer and Architecture Adaptation for Image Recognition

Guangrun Wang, Liang Lin, Rongcong Chen, Guangcong Wang, and Jiqi Zhang

Abstract—Current state-of-the-art visual recognition systems usually rely on the following pipeline: (a) pretraining a neural network on a large-scale dataset (e.g., ImageNet) and (b) finetuning the network weights on a smaller, task-specific dataset. Such a pipeline assumes the sole weight adaptation is able to transfer the network capability from one domain to another domain, based on a strong assumption that a fixed architecture is appropriate for all domains. However, each domain with a distinct recognition target may need different levels/paths of feature hierarchy, where some neurons may become redundant, and some others are re-activated to form new network structures. In this work, we prove that dynamically adapting network architectures tailored for each domain task along with weight finetuning benefits in both efficiency and effectiveness, compared to the existing image recognition pipeline that only tunes the weights regardless of the architecture. Our method can be easily generalized to an unsupervised paradigm by replacing supernet training with self-supervised learning in the source domain tasks and performing linear evaluation in the downstream tasks. This further improves the search efficiency of our method. Moreover, we also provide principled and empirical analysis to explain why our approach works by investigating the ineffectiveness of existing neural architecture search. We find that preserving the joint distribution of the network architecture and weights is of importance. This analysis not only benefits image recognition but also provides insights for crafting neural networks. Experiments on five representative image recognition tasks such as person re-identification, age estimation, gender recognition, image classification, and unsupervised domain adaptation demonstrate the effectiveness of our method.

Index Terms—Neural Architecture Adaptation, Structured Learning, Deep Neural Networks; Image Recognition; Weight Pretraining and Finetuning

I. INTRODUCTION

The success of ImageNet has enabled a standard paradigm of image recognition. Specifically, neural networks are often first pretrained on ImageNet to obtain a set of pretrained weights (e.g., w_1 in Fig. 1 (a)). Then, these pretrained network weights are further finetuned on a smaller, task-specific dataset to obtain the final optimal weights (e.g., w_2, w_3, w_4 in Fig. 1 (a)). Such a paradigm has led to state-of-the-art performance in almost all computer vision tasks, including person re-identification (re-ID) [1], human attribute recognition (e.g., age estimation and gender recognition) [2], and image classification [3].

However, this paradigm of weight pretraining and finetuning is not always effective, especially when the gap between the

source and target domain tasks is large. The reason behind that is three-fold. **First**, different domains with distinct recognition targets may need different levels/paths of feature hierarchy and different network topological connectivity. For example, as is known to all, DenseNet [4] achieves good performance on CIFAR-10 while having poor performance on ImageNet. In contrast, ResNets [5] obtain high top-1 accuracy on ImageNet but have high error on CIFAR-10. **Second**, transferred to the smaller dataset or simpler task, the architectures may be towards shallower; on the contrary, transferred to challenging tasks, the transferred architecture may be deeper. **Third**, different adaptation tasks prefer different new operations. For instance, the CIFAR-10 task likes operations that can perform feature augmentation while the person re-identification task prefers operations that capture global dependencies. In summary, the weight adaptation with the fixed network architecture suffers from the limited transfer capability from one domain into another. Usually, adapting specific neural network structures for different tasks is necessary to achieve state-of-the-arts, as Fig. 1 (b) illustrates.

To guarantee each task a personalized network structure, recently, many efforts have been made to manually [6], [7] or automatically [8]–[10] design neural networks for different tasks. Among these works, neural architecture search (NAS) is well-known for searching for neural structures in an automated way [11]. However, the scheme of using NAS is quite inefficient and time-consuming [8], [9], [12], which contains three indispensable stages. The three phases include:

- Searching for a different architecture for each target task.
- Pretraining these architectures on ImageNet one by one.
- Finetuning them on the target tasks one by one.

These redundant phrases are caused by the isolated optimization of the network architecture and network weights in NAS.

In this work, we propose a new transfer learning¹ framework called neural transfer and architecture adaptation (NTAA), to address the above problems. We find that dynamically tuning the network architectures tailored for each domain task along with fine-tuning weight leads to more efficient and effective transfer learning, compared to the existing adaptation pipelines that only tune the weights regardless of the architecture backbone. Given a network architecture α_0 whose network weights have been pretrained on ImageNet, our NTAA comprises of three main steps. **i)** We design a search space of network architectures \mathcal{A} such that the given architecture α_0 can be seen

¹Generally, transfer learning refers to reusing a model developed for a task as the starting point for a model on a second task. In this work, transfer learning specially represents using pre-trained neural networks on a large image classification dataset for benefiting different image recognition tasks.

G. Wang, L. Lin, R. Chen, G. Wang, J. Zhang are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, P. R. China. L. Lin is also with DarkMatter AI Research, China. Email: wang-grun@mail2.sysu.edu.cn.; linliang@ieee.org; Corresponding author: Liang Lin.

as an instance in the search space \mathcal{A} (i.e., $\alpha_0 \in \mathcal{A}$). **ii)** We start with α_0 , and jointly finetune the network weights and network architecture within the space \mathcal{A} . **iii)** The target architecture α^* is obtained after the optimization, based on which we further finetune the network weights for some epochs. Our method can be easily generalized to an unsupervised paradigm by training the supernet in the source domain task in a self-supervised manner and performing the linear evaluation in the downstream tasks to search for the best architecture. Since the supernet needs no fine-tuning during searching, and since the self-supervised learning features have good generalization ability [13], the unsupervised version’s search efficiency can be significantly higher than the supervised one.

We further discuss the reason behind the effectiveness of our NTAA and compare it with existing NAS approaches [11], [14]. In addition to the aforementioned search inefficiency [8]–[10], current NAS methods are also challenged with search ineffectiveness [15]–[17]. In fact, NAS is far away from being broadly practical in general image recognition. Some recently proposed works [16], [17] suggest that the performances of some NAS methods in certain scenarios are even worse than random architecture selection. In this work, we reveal the limitation of applying current NAS solutions within the transfer learning from both the principled and empirical perspectives. Solving a NAS model, essentially, is to find a network architecture with the highest train-from-scratch accuracy. The problem is very hard due to the intolerable cost of training each architecture in the search space from scratch. To simplify the problem, existing NAS solutions often use under-trained or shared network weights to evaluate an architecture’s performance by implicitly assuming the network architecture is independent of the network weights. The inappropriate evaluation has led to the unreproducible results of some existing NAS methods. On the contrary, our NTAA formulates the optimization of the network weights and architecture as a joint entity and solves the two parts synchronously. In addition to providing support to NTAA, the conducted analysis also benefits the future research of NAS.

Overall, the paper makes the following three contributions.

- We propose a general framework to adjust the network weights along with the network architecture adaptation synchronously. In a defined search space of network architecture, our method searching for an optimal architecture reduces to selecting an appropriate operation from the candidate operation set for each layer.
- Our method can be easily generalized to an unsupervised paradigm, which further improves the search efficiency of our method.
- We provide principled analysis to explain why our framework works by investigating the ineffectiveness of existing solutions to NAS. We find that preserving the joint distribution of the network architecture and weights is of importance. This analysis not only benefits transfer learning but also provides insights for NAS.
- We conduct extensive experiments on a variety of tasks, i.e., person re-ID, age estimation, gender recognition, image classification, and unsupervised domain adaptation. On these tasks, we achieve state-of-the-art performance,

demonstrating the superiority of our learning framework.

The remainder of this work is organized as follows. We review the previous works relevant to our method in Section II and introduce the NTAA learning framework and its extension in Section III. In Section IV, We provide the analysis to explain the reasons of the effectiveness of NTAA in a principled way. Section V presents the experimental results and comparisons. Section VI concludes this paper.

II. RELATED WORK

Weight Pretraining and Finetuning (WP&F). There is an overwhelming amount of deep-learning-based methods borrowing the powers from pre-training neural networks on large-scale datasets. Decades ago, Hinton et al., introduced transfer learning into training neural networks, especially under unsupervised learning scenarios [18]. Transfer learning techniques attracted much interest since 2012 when large-scale datasets such as ImageNet [19] were utilized in many image recognition tasks. Pre-training models on ImageNet is a key to achieve state-of-the-art performances in various tasks such as object detection [20], semantic segmentation [21], video recognition [22], person re-identification [1], human attribute recognition [2], and image classification [3], [23]–[25]. Moreover, transferring from ImageNet not only benefits the accuracies of the target tasks but also speeds up the learning process [21], [23]–[25]. In addition to computer vision, weight pretraining and finetuning is also used in other domains, such as natural language processing (NLP). Language model pretraining has shown remarkable improvement for many NLP tasks, such as natural language inference, paraphrasing, named entity recognition, and question answering. Notable works include ELMo [26], OpenAI GPT [27], and BERT [28]. ELMo [26] uses the pretrained representation as an additional feature for ensembling. The way of transfer learning in OpenAI GPT [27] and BERT [28] are more similar to that of the standard transfer learning in computer vision, i.e., a backbone is pretrained in the source task whose network weights are further finetuned for the target task. Compared with the feature transferring in previous works, our NTAA jointly optimizes the network architecture and neural weights. Moreover, our framework can be also combined with existing transfer learning methods.

The idea of using confidence values for operation shares the merit of adaptation factors in [29]. There are two differences between our confidence values for operation and the adaptation factors in [29]. **First**, our confidence values are associated with the selection for candidate operations, which result in different network architectures, i.e., both the network architecture and the network weights can be evolved with the confidence values changing. In contrast, in [29], only the model weights can be adapted with respect to the adaptation factors. Therefore, the adaptation factor used in [29] is only a tool in standard transfer learning, while our confidence value is a formulation for neural architecture adaptation. **Second**, a low confidence value in our method can lead to a straightforward removal of an operation of a predefined architecture; however, a low adaptation factor still assigns a small weight for a pretrained model weight.

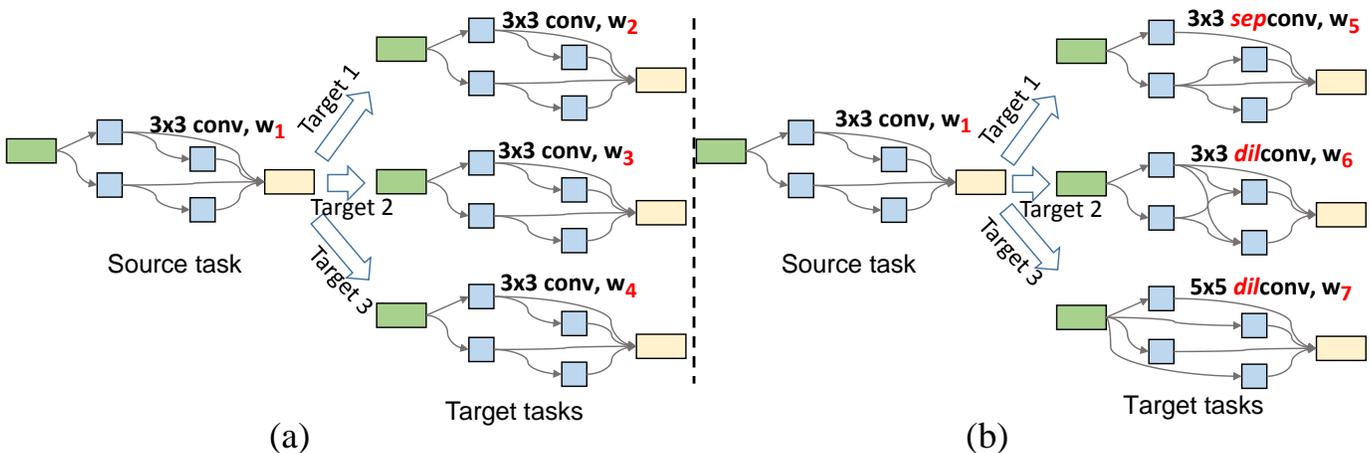


Fig. 1: Comparison between (a) weight pretraining and finetuning (WP&F) and (b) the proposed framework of neural transfer and architecture adaptation (NTAA). In WP&F, only network weights are transferred to the downstream tasks, e.g., from w_1 in a source task to w_2, w_3, w_4 in the target tasks. While in our NTAA, both the network weights and architecture are transferred to the downstream tasks, e.g., from $\langle 3 \times 3 \text{ conv}, w_1 \rangle$ in a source task to $\langle 3 \times 3 \text{ sepconv}, w_2 \rangle$, $\langle 3 \times 3 \text{ dilconv}, w_3 \rangle$, $\langle 5 \times 5 \text{ dilconv}, w_4 \rangle$ in the target tasks.

Neural Architecture Search (NAS). Recently, there has been a growing interest in developing algorithmic solutions to automate the manual process of designing a machine learning algorithm. In particular, neural architecture search (NAS) is expected to reduce the effort of human experts in network architecture design [11], [30]–[33], [33]–[38]. However, there is still an unsolved problem in NAS, i.e., how to efficiently solve a NAS model. The most mathematically accurate solution is to train each of the candidate architectures within the search space from scratch and compare their performance. The architecture with the most satisfactory performance is regarded as the target architecture. However, this solution is so extremely time-consuming as the search space is usually quite large (e.g., $> 1e^{20}$ [39]). To cope with this problem, many works have explored to train only the candidate architectures in a search *sub-space* by using reinforcement learning (RL) [11], [40] or evolution learning [14] to guide the search direction. For example, in the RL-based NAS, only the most potential candidate architectures with the largest rewards are trained because they are assumed to contain the target architecture. These NAS algorithms have achieved remarkable performance. However, they are still computationally demanding. For example, obtaining a state-of-the-art architecture for CIFAR-10 required 1,800 GPU days of reinforcement learning [11] and 3,150 GPU days of evolution [14]. This indicates that training the candidate architectures in the search *sub-space* (e.g., 1 million architectures) is still impractical, as training even *one* architecture costs a long time (e.g., more than 10 GPU days for a ResNet [5] on ImageNet).

To speed up NAS, the current methods have given up the mathematically accurate solution. They propose not to train each of the candidate architectures from scratch. Instead, they propose to train different candidate architectures by sharing the network weights [41]. Notable works include the weight-sharing reinforcement learning methods [36], attention-based differentiable methods (e.g., DARTS [42] and ProxylessNAS

[33]), and supernet-based methods [38], [43] in which a supernet is built to represent the full search space and each path is a stand-alone model. However, there is no theoretical guarantee that weight-sharing methods should work. Actually, as is suggested by [16], [17], many existing solutions to NAS are not better than random architecture selection. In our work, we explain the ineffectiveness of existing solutions to NAS in a principled way.

Self-Supervised Learning. Recently, unsupervised learning has recently shown remarkable progress in representation learning, especially in natural language understanding and computer vision. Notable works in natural language understanding include GPT [27] and BERT [28]. Among unsupervised learning, the results of self-supervised learning are most promising in the computer vision task. Specifically, self-supervised learning in computer vision can be divided into three groups, including low-level methods, mid-level methods, and high-level methods. Low-level methods include de-noising auto-encoders [44], context auto-encoders [45], or colorization [46]. Mid-level methods include patch orderings [47], [48]. However, both low-level methods and mid-level methods have poor performance in learning universal features. The most successful methods are high-level methods, i.e., contrastive learning methods. Notable works include MoCo [13], [49], simCLR [50], and BYOL [51]. For example, on ImageNet, the top-1 accuracy of BYOL is 74.3%, which is close to that of supervised learning, i.e., 76.4%.

Despite the promising accuracy and high expectation, the learning efficiency of self-supervised learning is low. Self-supervised learning usually costs ten times longer time for optimization than supervised learning. Specifically, for the task of training a ResNet50 on ImageNet, the supervised method costs about 100 epochs, while simCLR and BYOL cost 1,000 epochs, and CoCo v2 costs 800 epochs. The inefficiency of existing contrastive learning is the unreliability of the momentum encode. Specifically, the momentum encoder in existing

methods is an Exponential Moving Average (EMA) networks of the encoder networks. However, the EMA networks are not reliable in the earlier stage. Therefore, the learning efficiency of existing self-supervised learning is low. On the contrary, in our method, there is a reliable pretrained model. Using the pretrained model to replace the EMA networks in self-supervised learning significantly improves the learning efficiency.

III. METHODOLOGY

We start by briefly reviewing the technical background related to our work and then introduce our learning framework in detail. An extension of our framework to an unsupervised paradigm is further discussed, followed by more implementation details.

A. Preliminaries

Deep Neural Networks A deep model is written as:

$$\begin{aligned} \Phi(X, W^{(1)}, \dots, W^{(i)}, \dots, W^{(K)}) \\ = \psi_K(\dots \psi_i(\dots \psi_1(X \otimes W^{(1)}) \dots \otimes W^{(i)}) \dots \otimes W^{(K)}), \end{aligned} \quad (1)$$

where the whole neural network is formulated as a complicatedly nonlinear function $\Phi(\cdot)$. X is the input. $W = \{W^{(1)}, \dots, W^{(i)}, \dots, W^{(K)}\}$ denotes the network weights. K is the depth of the network. \otimes denotes a convolution operation (e.g., vanilla convolution, separate convolution, dilated convolution). $\psi(\cdot)$ denotes a nonlinear activation function (e.g., batch normalization + ReLU). For presentation simplification, Eqn. (1) is re-written as:

$$\Phi_{W,\alpha}(X) = (C_{W^{(K)}} \dots \circ C_{W^{(i)}} \dots \circ C_{W^{(1)}})(X), \quad (2)$$

where $C_{W^{(i)}}$ denotes a convolution operation using the network weight $W^{(i)}$ as convolutional filters and \circ denotes a sequence of operations (e.g., convolution) in a network architecture α . In the following, we use C_i to denote $C_{W^{(i)}}$ for notation simplification. Here, i denotes the i -th layer. In other words, the network architecture α can be represented as:

$$\alpha = C_K \dots \circ C_i \dots \circ C_1. \quad (3)$$

Finally, a deep learning problem is an optimization problem in the form of:

$$\min_w \mathcal{L}(Y, \Phi_{W,\alpha}(X)) + \lambda \|W\|_2 \quad (4)$$

where \mathcal{L} denotes the loss function. $\lambda \|W\|_2$ is a weight decay term that regularizes the model to prevent overfitting. λ is a hyper-parameter which is usually set as $1e^{-4}$.

Weight Pretraining and Finetuning Given a network α_0 , pretraining is to train α_0 from scratch on a source domain task. After the pretraining, we obtained the pretrained network weights W_0 . Then, a pretrained deep model can be written as $\alpha_0 = (C_{W_0^{(K)}}^0 \dots \circ C_{W_0^{(i)}}^0 \dots \circ C_{W_0^{(1)}}^0)$. Remarkably, in computer vision, ImageNet has been proved to be transferable to many other tasks. Therefore, ImageNet is considered as the most commonly used source domain task. Moreover, the pretrained ImageNet models are available online in the model zoos of almost all the deep learning frameworks (e.g., Caffe, Tensorflow, and Pytorch). Hence, the pretraining stage can be omitted in both standard transfer learning and our method.

In the paradigm of weight pretraining and finetuning, the network weights are tuned while the network architecture is

fixed when transferred to a target task. The goal of weight pretraining and finetuning is to find the optimal weights $W^* = W_0 + \Delta W^*$ for a fixed architecture α_0 which is subject to the condition that W is initialized as W_0 , i.e.,

$$\begin{aligned} \Delta W^* = \arg \min_{\Delta W} \mathcal{L}(Y, \Phi_{W_0 + \Delta W, \alpha_0}(X)) + \lambda \|W_0 + \Delta W\|_2 \\ s.t. \quad \begin{cases} W_0 \text{ is pretrained in the source task} \\ \alpha_0 \text{ is kept fixed} \end{cases} \end{aligned} \quad (5)$$

Note that α_0 keep unchanged before and after the optimization, indicating that the network architecture is unlearnable during the process of weight pretraining and finetuning.

Neural Architecture Search Neural architecture search (NAS) is an AutoML technique, which aims at automatically designing neural architectures for different tasks. NAS is not defined for a transfer learning problem, i.e., NAS does not exploit the pretrained weights obtained on a source task. Formally, NAS is formulated as:

$$\alpha^* = \arg \min_{\alpha} (\min_{W} (\mathcal{L}(Y, \Phi_{W,\alpha}(X)) + \lambda \|W\|_2)) \quad (6)$$

As shown, for each architecture α , we should train its network weights from scratch to convergence.

B. Neural Transfer and Architecture Adaptation

Different from weight pretraining and finetuning, our method is supposed to synchronously adjust the network weights as well as network architecture. The formulation of our method is as follows:

$$\begin{aligned} \Delta W^*, \Delta \alpha^* = \arg \min_{\Delta W, \Delta \alpha} \mathcal{L}(Y, \Phi_{W_0 + \Delta W, \alpha_0 + \Delta \alpha}(X)) \\ + \lambda \|W_0 + \Delta W\|_2 \\ s.t. \quad \begin{cases} W_0 \text{ is pretrained in the source task} \\ \alpha_0 \text{ is predefined for the source task} \end{cases} \end{aligned} \quad (7)$$

Here, we use $\Delta \alpha$ to denote the modification between the searched architecture and the architecture predefined for the source task. We use $\alpha_0 + \Delta \alpha$ to denote making a modification $\Delta \alpha$ for the architecture α_0 . Our goal is to find optimal weights $W^* = W_0 + \Delta W^*$ and an optimal architecture $\alpha^* = \alpha_0 + \Delta \alpha^*$ which is subject to the conditions that W is initialized as W_0 and α is initialized as α_0 .

As tuning the network weight is easy to understand, in the following, we only focus on the adaptation of the network architecture for notation simplification. Eqn. (7) can be simplified as follows:

$$\begin{aligned} \Delta \alpha^* = \arg \min_{\Delta \alpha} \mathcal{L}(Y, \Phi_{\alpha_0 + \Delta \alpha}(X)) \\ s.t. \quad \alpha_0 \text{ is predefined for the source task} \end{aligned} \quad (8)$$

Integrating Eqn. (3) into Eqn. (8), adapting the architecture from an initial architecture α_0 to the target architecture α^* can be finalized as follows:

$$\begin{aligned} \Delta C_K^*, \dots, \Delta C_K^*, \dots, \Delta C_1^* = \arg \min_{\Delta C_K, \dots, \Delta C_i, \dots, \Delta C_1} \mathcal{L}(Y, ((C_K^0 + \Delta C_K) \\ \circ \dots \circ (C_i^0 + \Delta C_i) \circ \dots \circ (C_1^0 + \Delta C_1))(X)) \\ s.t. \quad C_K^0, \dots, C_i^0, \dots, C_1^0 \text{ are predefined for the source task} \end{aligned} \quad (9)$$

Search Space Design. As discussed, our method is to make a set of modification $\{\Delta C_K, \dots, \Delta C_i, \dots, \Delta C_1\}$ for the predefined convolution operations $\{C_K^0, \dots, C_i^0, \dots, C_1^0\}$.

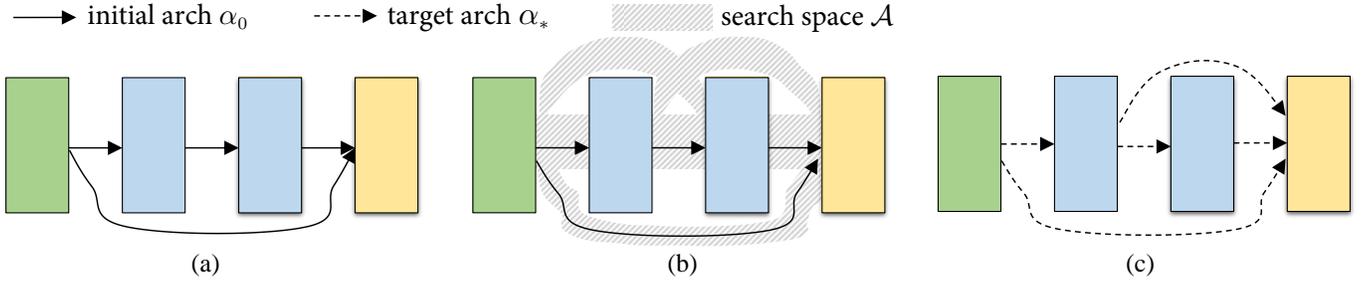


Fig. 2: An illustration of our framework: (a) the initial architecture; (b) the search space; (c) the target architecture.

Therefore, we need to design a search space for the modification $\{\Delta C_K, \dots, \Delta C_i, \dots, \Delta C_1\}$. That is, we need to design a search space $\mathcal{A} = \{\mathcal{O}_1, \dots, \mathcal{O}_i, \dots, \mathcal{O}_K\}$ such that:

$$\Delta C_K \in \mathcal{O}_K, \dots, \Delta C_i \in \mathcal{O}_i, \dots, \Delta C_1 \in \mathcal{O}_1. \quad (10)$$

How to form the architecture search space in a reasonable way is an active problem in NAS. Conventionally, in NAS, the search space is formed by defining candidate operations for each layer. Inspired by NAS, we define the modification C_i ($0 \leq i \leq K$) as replacing the predefined convolution operation C_i^0 by another operations selected from a candidate operation set \mathcal{O}_i . By definition, each \mathcal{O}^i consists of three types of operations:

- **Convolution:** 5×5 , 3×3 , and 1×1 convolution, denoted as o_1^i , o_2^i , and o_3^i , respectively.
- **Pooling:** 3×3 max pooling, 3×3 average pooling, globalization operation, denoted as o_4^i , o_5^i , and o_6^i , respectively.
- **Others:** identity, and noise disturbing, denoted as o_7^i and o_8^i , respectively.

where globalization indicates broadcasted global average pooling, and noise disturbing indicates adding a gaussian noise to the features. Here i denotes the i -th layer. Note that the operations in *pooling* and *others* are weight-free, indicating that choosing these two types of operations makes the network shallower.

It is important to note that, different from NAS, some of the predefined convolution operations may stay unchanged after the architecture transfer. Therefore, we must ensure that $\alpha_0 \in \mathcal{A}$, i.e., all \mathcal{O}_i s ($0 \leq i \leq K$) should meet the condition of $C_i^0 \in \mathcal{O}_i$ ($0 \leq i \leq K$).

Finally, searching for an optimal architecture reduces to selecting an appropriate operation from the candidate operation set for each layer. With the definition of the search space, our method is defined as an optimization problem as follows:

$$\begin{aligned} \Delta C_K^*, \dots, \Delta C_i^*, \dots, \Delta C_1^* &= \arg \min_{\Delta C_K, \dots, \Delta C_i, \dots, \Delta C_1} \mathcal{L}(Y, ((C_K^0 + \Delta C_K) \\ &\circ \dots \circ (C_i^0 + \Delta C_i) \circ \dots \circ (C_1^0 + \Delta C_1))(X)) \\ \text{s.t.} \quad &\begin{cases} C_K^0, \dots, C_i^0, \dots, C_1^0 \text{ are predefined for the source task} \\ \Delta C_i \in \mathcal{O}_i \end{cases} \end{aligned} \quad (11)$$

Fig. 2 shows the relationship of the initial architecture α_0 , the target architecture α^* , and the search space \mathcal{A} , in which the initial architecture α_0 can be seen as an instance in the search space \mathcal{A} (i.e., $\alpha_0 \in \mathcal{A}$).

(Rule of thumb) How to design a reasonable search space is an active problem in NAS. Conventionally, in NAS, the search space is created by defining candidate operations for each layer

and considering the topological connectivity of the network. These two aspects have been taken into account in our method. Among these aspects, predefined candidate operations is preferred in most of the existing NAS methods [30]–[43]. There is no instruction or guide for how to select the proper types of candidate operations. But spontaneously, current works on NAS (e.g., [30]–[43]) all consider the following three types of operations as candidates, i.e., convolution, pooling, and identity. Our search space is consistent with existing methods. Defining different search spaces can result in different possible consequences in search cost and model efficiency. **First**, defining too many candidate operations can increase the search cost. For example, if we double the number of candidate operations in our search space, the required GPU memory can be beyond what our machine can offer. **Second**, too large search space can lead to lousy architecture rating, which randomizes the performance of architecture search, as is suggested by [15], [52] below. **Third**, having different types of convolution can lead to different efficiency. For example, replacing vanilla convolution with MBConv can improve model efficiency but reduces the model accuracy. One possible **Rule of Thumb** on search space design may be searching for a search space, which is an unexplored / rarely-explored area in AutoML. For example, we can first define a vast search space consisting of almost all possible spaces. Then, we divide the search space into some search sub-spaces (i.e., 1,000 sub-spaces). Next, we sample architectures from these sub-spaces and pre-evaluate their performance. Last, the combination of the sub-spaces achieving higher pre-evaluation accuracies is considered as our final search space.

Formulation of Architecture Optimization. As discussed in Eqn. (11), searching for an optimal architecture reduces to selecting an appropriate operation from the candidate operation set for each layer. However, selecting an operation from the candidate set is discrete and non-differentiable, which cannot be optimized by DNNs. To address this problem, we relax the hard selection problem into a soft one. Specifically, we associate each candidate operation in \mathcal{O} with a confidence value $\mathcal{P} \in [0, 1]$, with $\mathcal{P} = 1$ indicating that the corresponding operation is definitely adopted. We assume that p can be learned in a data-driven manner. For example, for the i -th layer ($0 \leq i \leq K$), the probability of selecting a 3×3 convolution is defined as:

$$\mathcal{P}(C_i = o_2^i) = \frac{\exp(\theta_{o_2^i})}{\exp(\theta_{o_1^i}) + \exp(\theta_{o_2^i}) + \dots + \exp(\theta_{o_8^i})}. \quad (12)$$

where $\theta_{o_2^i}$ denotes a learnable parameter measuring the probability of selecting a 3×3 convolution for the i -th layer. The

probability of selecting other operations are similarly defined. Then, the neural network as a complicated function in the searching process is defined as:

$$\Phi_\theta(X) = (\mathbb{E}[o^K] \circ \dots \circ \mathbb{E}[o^i] \circ \mathbb{E}[o^1])(X). \quad (13)$$

where $\mathbb{E}[o^i]$ denotes an expectation of multiple choice, which is further defined as:

$$\mathbb{E}[o^i] = \sum_{t=1}^8 \mathcal{P}(C_i = o_t^i) [o_t^i] \quad (14)$$

Moreover, in Eqn. (12), as C_i^0 has been pretrained to have a stronger capacity than other operation in \mathcal{O}_i at the beginning, we initialize the $\theta_{C_i^0}$ to be ones while initializing the other θ s in \mathcal{O}_i to be zeros. So far, all the learnable parameters have been introduced, including W associated with the network weights and θ associated with the network architecture. Both W and θ are optimized by:

$$W^*, \theta^* = \arg \min_w \mathcal{L}(Y, \Phi_{W, \theta}(X)) + \lambda \|W\|_2, \quad (15)$$

where $\Phi_{W, \theta}(X)$ is calculated by:

$$\Phi_{W, \theta}(X) = (\mathbb{E}[o_{W_{o^K}}^K] \circ \dots \circ \mathbb{E}[o_{W_{o^i}}^i] \circ \mathbb{E}[o_{W_{o^1}}^1])(X). \quad (16)$$

where $o_{W_{o^i}}^i$ has the same meaning as o^i , indicating the network weights of the operation o^i is denoted as W_{o^i} . Inspired by the weight decay term, we include a operation regularization term ($\lambda \sum_{\theta \in \alpha_0} \theta - \lambda \sum_{\theta \in \mathcal{A} \setminus \alpha_0} \theta$) to regularize the network capacity. Here \setminus denotes the set difference. In particular, $\lambda \sum_{\theta \in \alpha_0} \theta$ discourages initial architectures, while $-\lambda \sum_{\theta \in \mathcal{A} \setminus \alpha_0} \theta$ encourage the newly-introduced operations, especially the identity and the noise disturbing operations that reduce the network complexity. Finally, our learning framework is cast to an classical optimization problem:

$$W^*, \theta^* = \arg \min_w \mathcal{L}(Y, \Phi_{W, \theta}(X)) + \lambda (\|W\|_2 + \sum_{\theta \in \alpha_0} \theta - \sum_{\theta \in \mathcal{A} \setminus \alpha_0} \theta), \quad (17)$$

which can be optimized using stochastic gradient descend.

Obtaining the target architecture α^* reduces to selecting an operation C_i from the candidate operation set \mathcal{O}_i for each layer. After searching, we have obtained θ s that measures the probability of selecting an operation for the i -th layer. Hence, the optimal operation is the one with the maximum probability:

$$C_i^* = \arg \max_{\forall 1 \leq j \leq 8} \theta_{o_j^i}. \quad (18)$$

The target architecture α^* is obtained, based on which we further finetune the network weights for some epochs.

C. Extension to Unsupervised Learning

Our framework can be easily generalized to a unsupervised paradigm. This needs minimal adjustment. For the presentation simplicity, we first introduce the definition of a supernet, which is a general concept widely used in the NAS community [31]–[33], [36], [38], [42], [53]. Formally, a supernet is a directed acyclic super-graph covering a whole search space with each node representing the feature maps and each edge representing a connection between the nodes with a particular operation (e.g., a convolution). Each subnet in the supernet represents a candidate architecture in the search space.

With the supernet definition, our supervised NTAA described in the previous section can be summarized into three steps. **Step 1:** A search space \mathcal{A} is designed such that the given architecture α_0 can be seen as an instance in the search space \mathcal{A} (i.e., $\alpha_0 \in \mathcal{A}$). This indicates we design a supernet containing α_0 as its subnet. **Step 2:** We start with α_0 , and jointly finetune the network weights and network architecture within the space \mathcal{A} . This indicates we train the supernet in which the subnet α_0 has been pretrained in the source task. **Step 3:** The target architecture α^* is obtained after the optimization, based on which we further finetune the network weights for some epochs. This indicates we obtain an optimal subnet from the supernet and finetune its network weights.

We can extend our supervised architecture search to a fast architecture search for different tasks by slightly modifying Step 2. The **NEW Step 2** is as follows: Before jointly optimizing the network architecture and network weights as supervised NTAA does, we first train the supernet in a unsupervised manner to obtain universal representations [13]. This universal supernet, called *Super* α_0 , is used to replace the original α_0 . It is saved and is ready for all the downstream tasks without re-training in the future. Then, we perform the linear evaluation in the downstream tasks to jointly optimize the network architecture and the head weights, i.e., we initialize all θ s in Eqn. (12) to be ones and freeze the backbone network weights, only learning θ s and the network weights in the last layer (e.g., the ten-class classifier weights for CIFAR-10).

Except for Step 2, Steps 1 and 3 in the unsupervised NTAA are the same as those in the supervised NTAA without modification. For example, we perform Step 3 to search for the best architecture α^* and further finetune the network weights for some epochs.

Our method of training the supernet in a self-supervised manner is novel and is different from all of the existing self-supervised learning methods. Our self-supervised learning method fits our NTAA perfectly. Specifically, the momentum encoder in existing methods, such as MoCo v1 [13], MoCo v2 [49], and BYOL [51] are all Exponential Moving Average (EMA) networks of the encoder networks. However, the EMA networks are not reliable in the earlier stage (see Figure 3). Therefore, the learning efficiency of existing self-supervised learning is low. On the contrary, in transfer learning, there is a reliable pretrained model. Using the pretrained model to replace the EMA networks in self-supervised learning significantly improves the learning efficiency. Following [13] and [49], our loss function is also defined as InfoNCE:

$$\mathcal{L}_{q, k^+, \{k^-\}} = -\log \frac{\exp(q \cdot k^+ / \tau)}{\exp(q \cdot k^+ / \tau) + \sum_{k^-} \exp(q \cdot k^- / \tau)}, \quad (19)$$

where q is a query feature, k^+ is the feature of a positive sample, and $\{k^-\}$ are the features of the negative samples.

Remark 1: Our unsupervised NTAA can improve search efficiency due to two reasons. First, Our Super α_0 is saved and is ready for all the downstream tasks without re-training. That is, when performing architecture search for a specific task, we can reuse the trained supernet. As most of the search time of NAS lies in the supernet training, our method significantly reduces the time's cost of architecture search for different downstream tasks. Second, the self-supervised

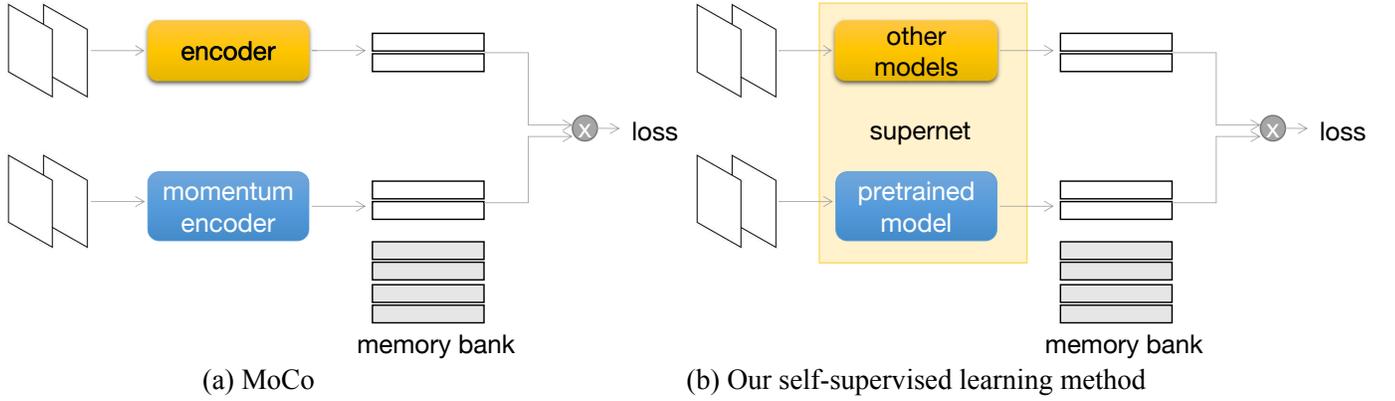


Fig. 3: Training $Super\ \alpha$ in a self-supervised manner. (a) MoCo v1 [13] and v2 [49]; (b) Our self-supervised learning.

learning features has good generalization ability [13], which accelerates the linear evaluation and the final finetuning.

D. Implementation Details

Conventionally, in NAS, the search space is formed by defining candidate operations for each layer and considering the topological connectivity of the network. These two aspects have been taken into account in our method. In section III-B, we have discussed the details of defining candidate operations for each layer. For ease of understanding, we have omitted the topological network connectivity in the search space \mathcal{A} . Actually, the number of layers and connection between units have been taken into account in our method.

- There is a candidate skip connection in our search space. Specifically, as Fig. 2 (b) shows, in our search space \mathcal{A} , a skip layer operation is also allowed to enrich the search space. Actually, each layer is supposed to be connected with preceding three layers, i.e., both $\alpha = C_4 \circ C_3 \circ C_2 \circ C_1$ and $\alpha = C_4 \circ C_1$, $\alpha = C_3 \circ C_1$ are allowed. there is a candidate skip connection in our search space (Fig. 2 (b)). Therefore, the topological connectivity and the number of layers have also been taken into account in our architecture search.
- Our candidate operation sets includes *Pooling*, *identity*, and *noise disturbing*. Note that these operations are weight-free, indicating that choosing these operations can reduce the depth of the searched architecture. Therefore, the depth of our searched architecture can be further made multifarious.

IV. PRINCIPLED ANALYSIS ON THE EFFECTIVENESS

In the following, we provide the principled analysis to explain the reason behind the effectiveness of our NTAA by comparing it with the existing NAS approaches. Here, we start by discussing the defect of NAS and the ineffectiveness of directly applying NAS methods in the transfer learning.

A. Ineffectiveness of NAS

We have presented the formulation of NAS in Eqn. (6), from which we can find that the challenge of NAS lies in the inner term of Eqn. (6), i.e., $\min_{\forall W} (\mathcal{L}(Y, \Phi_{W, \alpha}(X)) + \lambda \|W\|_2)$,

which requires us to train each of the candidate architectures in the search space \mathcal{A} from scratch to convergence to obtain the optimal W^* . However, this is infeasible as training even one ResNet costs 10 GPU days, not to mention there are more than $1e^{20}$ architectures in the search space [39]. To cope with this problem, under-training [11], [40] and co-training [15], [31]–[33], [36], [38], [42], [53] have been proposed. Under-training is to train each architecture for a few epochs (e.g., five epochs) and use the performance at the 5th epoch to evaluate the architecture. Such architecture rating is inaccurate due to the insufficient training. Therefore, co-training is proposed. Co-training is to train different candidate architectures by sharing the network weights in a supernet. Currently, co-training is the most commonly-used method in NAS. However, co-training is still ineffective because the architecture rating based on the subnet disengaged from supernet is inaccurate [15]–[17], [54].

In the following, we analyze the inaccurate rating of weight-sharing NAS by using the tool of generalization boundedness.

Theorem 1: (Generalization boundedness). Let η be the learning rate, \mathcal{S} be the number of the candidate operations for each layer, and T be the training iterations. Let \mathcal{R} be the misclassification error, L be the loss function, and \mathbf{W} be the learnable parameters. Then, for any candidate architecture j , the misclassification error is upper bounded by:

$$\min_{t \in [T]} \mathbb{E} [\mathcal{R}(\mathbf{W}_{t,j})] \leq \mathcal{O}\left(\frac{\ln^2(T) + \ln(ST)}{T}\right). \quad (20)$$

Proof sketch. Since the expected loss is independent from sampled data, we have:

$$\mathbb{E}_{\mathcal{D}_{t+1}} [\mathcal{L}(\mathbf{W}_t)] = \mathbb{E}_{\mathcal{D}_t, \mathcal{B}_t} [\mathcal{L}(\mathbf{W}_t)] = \mathbb{E}_{\mathcal{D}_t} [\mathcal{L}(\mathbf{W}_t)]. \quad (21)$$

As the loss upper-bounds the error rate [55], we have:

$$\begin{aligned} \mathbb{E}_{\mathbf{w}_1, \mathcal{D}_T} \left[\frac{1}{T} \sum_{t=1}^T \mathcal{R}(\mathbf{w}_t) \right] &\leq \mathbb{E}_{\mathbf{w}_t, \mathcal{D}_T} \left[\frac{1}{T} \sum_{t=1}^T \mathcal{L}(\mathbf{w}_t) \right] \\ &\leq \mathbb{E}_{\mathbf{w}_t, \mathcal{D}_T, \mathcal{B}_T} \left[\frac{1}{T} \sum_{t=1}^T \mathcal{L}_t(\mathbf{w}_t) \right] \quad (\text{by Eqn. (21)}) \\ &\leq \frac{\mathbb{E}_{\mathbf{w}_1} [\|\mathbf{W}_1 - U\|_F^2]}{\eta T} + \frac{2}{T} \sum_{t=1}^T \mathbb{E}_{\mathbf{w}_1, \mathcal{D}_T, \mathcal{B}_T} [\mathcal{L}_t^{(t)}(U)]. \end{aligned} \quad (22)$$

Let $U = \mathbf{W}_1 - \lambda V$ with $\|V\|_F \leq 1$ and $\lambda = c_1 \ln(2\eta T) + \sqrt{c_2 \ln(c_3 \eta \sqrt{ST^2})}$ with c_1, c_2 , and c_3 being constants. Since $\|\mathbf{W}_1 - U\|_F^2 = \|\mathbf{W}_1 - \mathbf{W}_1 - \lambda V\|_F^2 = \lambda^2 \|V\|_F^2 \leq \lambda^2$, we have: $\frac{\mathbb{E}_{\mathbf{w}_1} [\|\mathbf{W}_1 - U\|_F^2]}{\eta T} \leq \frac{\lambda^2}{\eta T}$. Since $\mathcal{L}_t^{(t)}(U) \leq \frac{\lambda^2}{2\eta T}$ and $\mathcal{L}_t^{(t)}(U) \leq c_4 \sqrt{S}$, we have: $\mathbb{E}[\mathcal{L}_t^{(t)}(U)] \leq (1 - \sigma) \frac{\lambda^2}{2\eta T} + \sigma c_4 \sqrt{S} \leq \frac{\lambda^2}{2\eta T} + \sigma c_4 \sqrt{S} \leq \frac{\lambda^2}{2\eta T} + \frac{1}{2\eta T} \leq \frac{\lambda^2}{\eta T}$, where $\sigma \triangleq \frac{1}{2c_4 \sqrt{S} \eta T}$. This finally yields:

$$\min_{t \in [T]} \mathcal{R}(\mathbf{W}_{t,j}) \leq \frac{3\lambda^2}{\eta T} \leq \mathcal{O}\left(\frac{\ln^2(T) + \ln(ST)}{T}\right), \quad (23)$$

which completes the proof.

Remark 2: (Inaccurate rating in existing NAS). *Theorem 1 shows that the misclassification error (usually measuring the generalization ability) of the weight-sharing solution has an upper bound related to the search space size (i.e., S). This might provide an explanation for the inaccurate rating problem in existing NAS. Specifically, for the models learned by the weight-sharing solution, increasing the search space size (i.e., S) would lead to a larger upper bound of the misclassification error and thus the poorer generalization ability.*

A poor generalization ability implies that the architecture rating based on the weight-sharing solution might not be predictive of the true rating of a neural architecture. Suppose an architecture has a good ground-truth rating, but its generalization ability based on the weight-sharing solution is poor, and thus its validation accuracy is low. Then, this architecture might have a bad predicted rating.

In summary, the large search space might be the cause of the ineffectiveness of existing NAS because a large search space would result in a poor generalization ability of the weight-sharing solution and further would lead to the inaccurate rating of the network architecture, which finally leads to the ineffectiveness of NAS.

B. Effectiveness of NTAA

By comparing Eqn. (6) and Eqn. (7), we reveal that a NAS model aims at searching for an optimal architecture that can be trained from scratch. Therefore, good architecture rating is necessary to ensure the ability of a network to be trained from scratch. Differently, our NTAA synchronously optimizes the network weights and network architecture, which needs no architecture rating.

Generally, NTAA is better compatible with the transfer learning scenarios in a principled way compared with traditional NAS solutions. In particular, Eqn. (7) shows that our NTAA combines the network weights and architecture learning with a joint distribution, and optimizes them synchronously. After the joint optimization, an optimal pair $\{\text{architecture}(\alpha_*), \text{weights}(W_*)\}$ indicating a good adaptation for transfer learning tasks is obtained. Note that obtaining $\{\alpha_*, W_*\}$ in Eqn. (7) does not necessarily indicate obtaining an optimal solution to NAS in Eqn. (6). Specifically, although $\{\alpha_*, W_*\}$ has good accuracy in transfer learning using our NTAA, it may not be obtained by NAS method, because W_* may be inaccessible when the architecture α_* is trained from scratch.

Actually, training from scratch is required by the definition of NAS as Eqn. (6) suggests. The inaccessibility of W_* has also been discussed in the theory of lottery ticket hypothesis [56]–[58].

C. Advantage of NTAA from the Practical Perspective

The above analysis has shown the advantage of our NTAA over existing NAS methods. From the practical perspective, our NTAA aims at improving the fundamental and universal problem of standard transfer learning, while NAS aims at searching for an effective architecture for different tasks. Limited by the number of training examples and the huge cost of annotation, many tasks (e.g., person re-ID, age estimation, gender recognition, and image classification) should be first trained on large-scale datasets (e.g., ImageNet) to obtain transferable representations and subsequently fine-tuned on the specific task. Directly searching architectures for these tasks lead to a suboptimal solution. On the contrary, our NTAA exploits both the power of standard transfer learning and NAS, achieving a better performance.

V. EXPERIMENTS

To justify the effectiveness of NTAA, we have conducted extensive experiments on a wide range of computer vision tasks such as person re-ID, age estimation, gender recognition, image classification, and domain adaptation. Our goal is to tune the standard backbone, including ResNet50 [5], ResNet101 [5], InceptionV2 [59], VGG [60], AlexNet [19], and the recently proposed EfficientNet [3].

A. Comparison with Weight Pretraining and Finetuning in Person Re-Identification

We first evaluate the effectiveness of our NTAA in the well-known transfer learning task, i.e., re-ID, which has been extensively studied in recent years [61], [62]. It refers to the problem of re-identifying individuals across cameras. Solving re-ID problems is very challenging but has many applications in video surveillance for public safety. Recent state-of-the-art re-ID models are all built upon standard transfer learning techniques [1], [63]–[69]. Specifically, a standard ResNet-50 is first pretrained on ImageNet, and then the network weights are tuned to the re-ID domain for adaptation. For the evaluation, the test set is further divided into a gallery set of images and a probe set. We use the Rank-1, Rank-5, Rank-10, and mAP as the evaluation metric, which are standard metrics in re-ID. Note that all the compared methods only reported the Rank-1 and mAP metric on CUHK03. We are not able to access to their Rank-5 and Rank-10 results. Therefore, we only compare the Rank-1 and mAP results on CUHK03.

Market-1501 We first conduct experiments on the Market-1501 [84], which is one of the largest databases for re-ID. This database contains 32,668 images of 1,501 pedestrians captured from 6 different cameras. The dataset is split into two parts: 12,936 images with 751 identities for training and 19,732 images with 750 identities for testing. In testing, 3,368 hand-drawn images with 750 identities are used as probe set to identify the correct identities on the testing set.

TABLE I: Comparison on a Person Re-identification task (Market-1501). (**WP&F**: weight pretraining and finetuning; **SOTA**: state of the art; **R-1,5,10**: Rank-1,5,10; **+E**: random erasing; **+R**: re-ranking; **UNTAA**: unsupervised NTAA)

| Transfer type | Methods | R-1 | R-5 | R-10 | mAP |
|----------------|-----------------|-------------|-------------|-------------|-------------|
| WP&F (SOTA) | MSCAN [63] | 80.31 | n/a | n/a | 57.53 |
| | DF [64] | 81.0 | n/a | n/a | 63.4 |
| | SSM [65] | 82.21 | n/a | n/a | 68.80 |
| | SVDNet [66] | 82.3 | n/a | n/a | 62.1 |
| | GAN [67] | 83.97 | n/a | n/a | 66.07 |
| | PDF [68] | 84.14 | n/a | n/a | 63.41 |
| | TriNet+E+R [69] | 86.67 | 93.38 | n/a | 81.07 |
| | Omin [70] | 94.8 | n/a | n/a | 84.9 |
| | JointDG [71] | 94.8 | n/a | n/a | 86.0 |
| | IANet [72] | 94.4 | n/a | n/a | 83.1 |
| | CASN+PCB [73] | 94.4 | n/a | n/a | 82.8 |
| | CAMA [74] | 94.7 | 98.1 | n/a | 84.5 |
| | MHN-6 [75] | 95.1 | 98.1 | n/a | 85.0 |
| | AANet [76] | 93.9 | n/a | n/a | 83.4 |
| | P^2 -Net [77] | 95.2 | 98.2 | n/a | 85.6 |
| | PGFA [78] | 91.2 | n/a | n/a | 76.8 |
| | ISP [79] | 95.3 | 98.6 | n/a | 88.6 |
| | CBN [80] | 94.3 | 97.9 | 98.7 | 83.6 |
| | SNR [81] | 94.4 | n/a | n/a | 84.7 |
| | M^3 +R50 [82] | 95.4 | n/a | n/a | 82.6 |
| | PCB [83] | 91.2 | 96.5 | 97.3 | 76.7 |
| NTAA | PCB + NTAA | 94.5 | 97.9 | 98.3 | 84.8 |
| | PCB + UNTAA | 94.5 | 98.0 | 98.4 | 85.0 |
| | NTAA + SE | 95.6 | 98.2 | 98.9 | 85.6 |

Comparison with weight pretraining and finetuning. In Table I, we compare with 21 representative methods of weight pretraining and finetuning, which is also the current best models on Market-1501, including MSCAN [63], DF [64], SSM [65], SVDNet [66], GAN [67], PDF [68], TriNet [69], TriNet + Era. + Re-ranking [1], PCB [83], Omin [70], JointDG [71], IANet [72], CASN+PCB [73], CAMA [74], MHN-6 [75], AANet [76], P^2 -Net [77], PGFA [78], ISP [79], CBN [80], SNR [81], and M^3 +ResNet50 [82]. All settings of the above methods are consistent with the common training settings. Our NTAA achieves a rank-1 accuracy of 94.5% with an improvement of 3.3% over its baseline. It also surpasses other competitors by a clear margin. As Omin [70] introduces an Omni-Scale features which contains multi-scale feature and SE global dependencies [85], for fair comparison, we also equip our search architecture with global SE features. The results in Table I show that our NTAA outperforms OSNet by a large margin. This comparison verifies the effectiveness of NTAA on re-ID tasks.

CUHK03 We further conduct experiments on the CUHK03 dataset [86], which is another of the largest databases for re-ID. This database contains 14,096 images of 1,467 pedestrians. Each person is observed by two disjoint camera views and is shown in 4.8 images on average in each view. We follow the 767/700-split setting of using CUHK03 [1], where 767 individuals are regarded as the training set, and another 700 individuals are considered as the testing set without overlap.

Comparison with weight pretraining and finetuning. Actually, all the state-of-the-art models in the re-ID domain uses the technique of weight pretraining and finetuning. Therefore, comparing with the methods of weight pretraining and fine-

TABLE II: Comparison on a Person Re-identification task (CUHK03) (**bs**: batch size; **WP&F**: weight pretraining and finetuning; **SOTA**: state of the art; **R-1**: Rank-1; **+E**: random erasing; **+R**: re-ranking; **UNTAA**: unsupervised NTAA)

| Transfer Type | Methods | R-1 | mAP |
|----------------|------------------------------------|-------------|-------------|
| WP&F (SOTA) | SVDNet [66] | 41.5 | 37.3 |
| | Omin [70] | 72.3 | 67.8 |
| | TriNet + Era. [1] | 55.5 | 50.74 |
| | Omin [70] | 72.3 | 67.8 |
| | Auto-ReID [9] | 73.3 | 69.3 |
| | CASN+PCB [73] | 71.5 | 64.4 |
| | CAMA [74] | 66.6 | 64.2 |
| | MHN-6 [75] | 71.7 | 65.4 |
| | M^3 +R50 [82] | 66.9 | 60.7 |
| | TriNet+E (Our reproduction) | 62.0 | 57.6 |
| | TriNet+E+R (bs = 32) | 61.2 | 55.4 |
| NTAA | TriNet+E+NTAA | 64.6 | 61.5 |
| | TriNet+E+UNTAA | 66.1 | 63.2 |
| | TriNet+E+R+NTAA(bs = 32) | 65.0 | 61.8 |
| | TriNet+E+R+NTAA + longer NTAA + SE | 71.6 | 66.1 |
| | | 74.8 | 69.8 |

tuning is equivalent to compare with the current best models on CUHK03. In Table II, we compare with ten representative standard transfer learning methods, including Omni [70], SVDNet [66], TriNet + Era. [1], TriNet + Era. + Reranking [1], Auto-ReID [9], CASN+PCB [73], CAMA [74], MHN-6 [75], and M^3 +ResNet50 [82]. All the settings of the above methods are consistent with the common training settings. Our NTAA has achieved a new benchmarking state of the art. Specifically, NTAA achieves a rank-1 accuracy of 74.8%. We also highlight NTAA surpasses its baseline by a clear margin (e.g., 65.0% vs. 61.2%). This verifies the effectiveness of NTAA on re-ID tasks.

Results of Unsupervised Paradigm We also report the performance of our unsupervised NTAA transfer on both Market-1501 and CUHK03. As shown in Table I and Table II, our unsupervised NTAA obtains slightly higher accuracy than the supervised NTAA. For example, the rank-1 accuracies of the unsupervised NTAA and supervised NTAA on CUHK03 are 66.1% and 64.6%, respectively. The improvement may be attributed to the universal features learned by self-supervised learning. This comparison verifies the effectiveness of our unsupervised NTAA in addressing transfer learning problem.

Computational Complexity There are two rounds of running in our framework, each of which contains 100 epochs. In the first round, our goal is to tune the architecture. For the supervised paradigm, the search takes 5 hours on a sole GPU on Market-1501. In the second round, our goal is to finetune the network weights based on the optimized architecture. This round takes 3 hours on a single GPU.

For the unsupervised paradigm, the search stage takes 2 hours on a sole GPU on Market-1501, which cuts the search time of the supervised paradigm by half. This is due to two reasons. **First**, Our *Super* pretrained model is saved and is ready for all the downstream tasks without retraining. That is, when performing architecture search for a specific task, we can reuse the trained supernet. As most of the search time lies in the supernet training, our method significantly reduces the time's cost of architecture search for the downstream tasks.

TABLE III: Effectiveness / ineffectiveness of NAS on Market-1501. (**R-1,5,10**: Rank-1,5,10)

| Type | Methods | R-1 | R-5 | R-10 | mAP |
|------------------|---------|-------------|-------------|-------------|-------------|
| NAS | DARTS | 91.7 | 96.9 | 97.5 | 77.2 |
| | SPOS | 91.9 | 96.9 | 97.7 | 77.3 |
| Random Selection | - | 91.6 | 96.8 | 97.5 | 77.2 |

Second, the self-supervised learning features has good generalization ability [13], which accelerates the linear evaluation.

B. Effectiveness Analysis in Person Re-Identification

In the following, we conduct empirical studies to verify the effectiveness of our NTAA compared with existing NAS approaches.

We use two representative NAS methods as the competitors, including DARTS [42] and single-path-one-shot (SPOS) method [38]. These two methods are the current state-of-the-art methods, which reports the best performance on ImageNet with a considerably acceptable computational cost. DARTS is an attention-based and differentiable method that densely connects the candidate operations with attention scores during the search process. Then the edges with weak attention are removed after searching, forming the target architecture. SPOS uses a supernet to represent the full search space, and each path is a stand-alone model. All the experiments in this section are conducted on Market-1501. We have the following three comparisons.

Effectiveness / ineffectiveness of NAS. We evaluate the effectiveness of NAS in the same search space as our NTAA mentioned in Section III-B. We evaluate our method and others with the following setting. i) We randomly sample four architectures from search space with a uniform distribution and train them from scratch. The accuracies are averaged to represent the capacity of random architecture selection. ii) We use DARTS and SPOS to search for the target architectures in the search space, respectively. Then, the search architectures are trained from scratch. Note that here, our goal is to analyze the effectiveness/ineffectiveness of NAS but NOT to push the state-of-the-art. Therefore, all models have not been pretrained by ImageNet.

The results in Table III provides three insights. First, the accuracies of the NAS models are not significantly better than that of random architecture selection, indicating that the solution to NAS maybe not effective. In particular, the accuracies of DARTS and random selection are similar, while SPOS obtains slightly better results than both DARTS and random architecture selection. Considering that NAS models have spent lots of computational costs to search for the architecture, we regard the solution to NAS in this problem as ineffective. Second, although SPOS has made several efforts to improve the shared weights, the accuracy of SPOS is just slightly better than that of DARTS. This indicates that improving the solution to NAS has a long way to go. Third, the accuracies in Table III are significantly lower than that in Table I. The degradation is due to the lack of ImageNet pretraining, verifying the importance of exploiting both the network weights and network architecture as our NTAA does.

TABLE IV: NAS v.s. NTAA for transfer learning on Market-1501. (**R-1,5,10**: Rank-1,5,10)

| Type | Methods | R-1 | R-5 | R-10 | mAP |
|----------------|---------|-------------|-------------|-------------|-------------|
| NAS + transfer | DARTS | 93.2 | 97.4 | 98.2 | 81.1 |
| | SPOS | 93.4 | 97.6 | 98.3 | 81.6 |
| NTAA | - | 94.5 | 97.9 | 98.3 | 84.8 |

TABLE V: Can the optimal weights of NTAA searched by NAS? (Market-1501) (**R-1,5,10**: Rank-1,5,10)

| Method | R-1 | R-5 | R-10 | mAP |
|-----------------------------|-------------|-------------|-------------|-------------|
| NTAA | 94.5 | 97.9 | 98.3 | 84.8 |
| Retrain NTAA's architecture | 92.0 | 96.9 | 97.9 | 77.4 |

Superiority of NTAA over NAS. In addition, we compare the accuracies of NAS and NTAA on Market-1501. In our method, the architecture has first exploited the ImageNet pretraining, and then the architecture and the weights are jointly optimized in the downstream task. However, ImageNet pretraining is absent in the NAS counterparts. In the NAS counterparts, the architecture is searched in the downstream task, and then the weights are trained in the downstream task. For a fair comparison, after the architecture search in the NAS counterparts, we perform ImageNet pretraining for the searched architecture before finetuning the weights in the downstream task (i.e., Market-1501) following using the standard transfer learning pipeline [1], [63]–[69], [83]. The results are reported in Table IV. As shown, our NTAA obtains a 1.1% higher accuracy than NAS models (94.5% vs. 93.4% for NTAA vs. SPOS). There are two reasons behind the improvement of our method over others. First, as explained Section IV, our NTAA better accords with the goal and setting of transfer learning. Second, there is a gap between existing NAS solutions and transfer learning. In particular, the target architecture is searched for on the target task (i.e., person re-ID). But the target architecture can not be directly used on the target task until it has been pretrained on the source task (i.e., ImageNet classification).

According to the above principled analysis, our NTAA can produce the optimal pair of $\{\alpha_*, W_*\}$ that can not be searched by existing NAS methods. Accordingly, we provide some empirical studies for demonstrating the superiority of our NTAA. We use the optimal pair of $\{\alpha_*, W_*\}$ (i.e., the state-of-the-art model) in Table I and Table IV as our reference. Specifically, we employ the architecture α_* while discarding the network weights W_* . The network weights are reset as randomly initialized weights. Then we retrain the architecture α_* from scratch. The converged accuracy is compared with our NTAA in Table V. We can observe a significant performance degradation of the retrained model compared with our NTAA. The inaccessibility of W_* has also be noticed by [56]–[58], proving the importance of our NTAA in synchronously learning the network weights and the architecture in the same way as our NTAA.

C. Image Classification

We further evaluate our NTAA on the CIFAR-10 [87], which consists of 10 categories. The models are trained on the 50,000 training images and evaluated on the testing 10,000 images. CIFAR-10 is chosen for the following reasons. First, the data

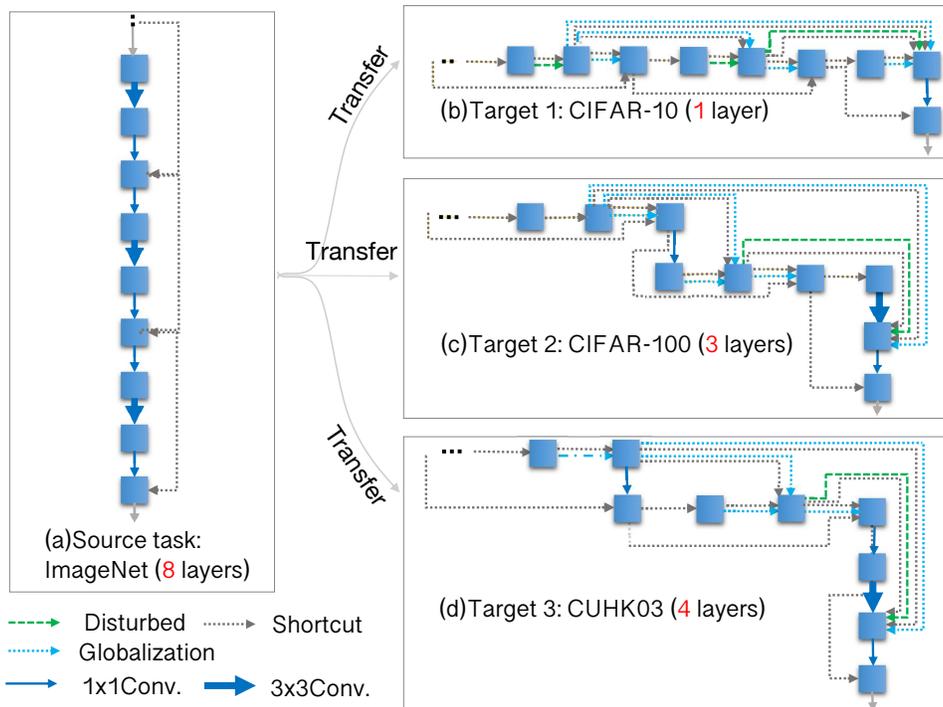


Fig. 4: An example of visualization of the neural architecture adaptation from the pre-training on ImageNet to different image recognition tasks (best view in color).

volumes of CIFAR-10 are far less than that of ImageNet. Pretraining on ImageNet may provide performance gains. Second, CIFAR-10 has fewer categories than ImageNet, which consists of 1,000 categories. Learning on more complicated tasks may enable more transferable representations. Finally, the resolution of CIFAR-10 is less than that of ImageNet. Specifically, CIFAR Images are 32×32 pixels, while ImageNet images are $500 \times X$ pixels, where X denotes the width of a rectangle. There is a pattern gap between large images and tiny images, making the finetuning problem more challenging. We examine the error rate. The representative residual networks (e.g., ResNet50 [5] / ResNet101 [5] / ResNet152 [5] / InceptionV2 [59] / VGG [60]) are chosen as our backbones. NTAA is performed by employing the standard protocol [42].

To demonstrate the effectiveness of our NTAA, we explore two representative standard transfer learning methods, i.e., **full** standard transfer learning and **partial** standard transfer learning. Specifically, a network is first pretrained on the ImageNet dataset. Then, for the full standard transfer learning, we tune all the network weights on CIFAR-10; for the partial standard transfer learning, only the network weights in the high-level layers are tuned. Note that the architectures are kept fixed for standard transfer learning, which is different from our NTAA. Besides, another baseline is also introduced, i.e., training from scratch on CIFAR-10 without any pretraining steps. The performances of the three baseline models are presented in Table VI.

We have four observations from Table VI for all the backbones. First, both the full and partial standard transfer learning brings a clear gain when compared with training from scratch (e.g., 3.90% / 3.72% vs. 6.61% for ResNet50). This reflects the importance of pretraining network weights.

Second, the partial standard transfer learning has a slightly better performance than the full standard transfer learning. We believe this observation is of importance because it presents insights into transfer learning. **1)** the data volume of the target domain is usually smaller than that of the source domain. Consequently, the large network architecture may lead to overfitting. **2)** The network architecture can be divided into a *general* part and a *specific* part. Loosely, the lower layers may be general, while the high-level layers may be specific. Therefore, fixing the low-level layers while finetuning the high-level layers (i.e., partial standard transfer) leads to an improvement.

Third, our NTAA surpasses both the full and partial standard transfer learning. For example, in ResNet50, our NTAA achieves an error rate of 2.43%, which is 1.47% and 1.29% lower than the full and partial standard transfer learning, respectively. The superiority of our NTAA over the standard transfer learning is attributed to the architecture adaptation. Specifically, in our method, the *general* and *specific* part are adaptively learned, while in partial standard transfer learning, they are handcrafted and fixed.

Fourth, the depth of our NTAA is significantly less than all baseline models (e.g., 33 vs. 50 in ResNet50), indicating that our NTAA has a more efficient neural architecture. The above observations verify the superiority of our NTAA over the standard transfer learning pipelines.

Computational Complexity Both the search & training process contain 600 epochs. The search takes 1.3 days on 4 GPUs. The training takes 1.9 days on a single GPU.

TABLE VI: The comparison of NTAA and the standard transfer learning scheme.

| Method | | ResNet50 | | ResNet101 | | InceptionV2 | | VGG19 | |
|--------------------------------------|---------|-------------------------------------|-----------|-------------------------------------|-----------|-------------------------------------|-------|-------------------------------------|-----------|
| | | Error | Depth | Error | Depth | Error | Depth | Error | Depth |
| Train from scratch | | 6.61 | 50 | 6.53 | 101 | 4.27 | - | 7.09 | 19 |
| weight pretraining and finetuning | full | 3.90 | 50 | 3.85 | 101 | 3.79 | - | 6.58 | 19 |
| | partial | 3.72 | 50 | 3.71 | 101 | 3.63 | - | 6.55 | 19 |
| NTAA | | 2.43 (± 0.04) | 33 | 2.39 (± 0.05) | 65 | 2.30 (± 0.05) | - | 6.33 (± 0.18) | 13 |

TABLE VII: Age estimation & gender recognition on Adience.

| Method | Age Accuracy | Gender Accuracy |
|----------------------------|--------------|-----------------|
| Train from scratch | 53.5% | 93.8% |
| pretraining and finetuning | 56.4% | 94.1% |
| Levi Hassner [90] | 44.1% | 82.5% |
| LMTCNN [89] | 44.3% | 85.2% |
| WRN [91] | 57.4% | 93.9% |
| WRN* [92] | 59.7% | 94.6% |
| NTAA | 64.3% | 95.5% |

D. Age Estimation & Gender Recognition

To investigate the effectiveness of our NTAA in more complicated tasks, we adopt age estimation and gender recognition tasks on Adience [88], which is very challenging due to the extreme variations, e.g., low resolution, occlusions, pose and expression variations. Adience consists of 26k unconstrained images of 2,284 person IDs, whose ages range from 0 to 60+. Images in the Adience dataset are taken in the wild, making the evaluation on Adience meaningful.

We use the standard evaluation protocol on Adience to measure the top-1 accuracy [89]. Also, following the standard settings, we do not use an additional face alignment technique. In contrast, the official aligned version of faces are used in our experiment. The images are resized to 256×256 , and a random crop or a center crop of 224×224 pixels are applied during the training and testing, respectively.

We use the recently proposed EfficientNet-B4 [3] as the initial architecture α_0 . As EfficientNet-B4 [3] uses MBConv as its cell, which does not contain a vanilla convolution, we revise our search space \mathcal{A} by replacing the 5×5 and 3×3 vanilla convolution in the candidate operation sets with 5×5 and 3×3 separate convolution. We train the EfficientNet-B4 [3] for 150 epochs with a cosine learning rate schedule. The batch size is set as 32. Two baselines are adopted, including training from scratch and standard transfer learning. For training from scratch, the initial learning rate is set as 0.1. For standard transfer learning and our NTAA, the initial learning rate is set as $3e-3$. Experimental results in Table VII shows that our NTAA has a large gain over the two baselines, especially on age estimation (64.3% vs. 56.4% for NTAA vs. weight pretraining and finetuning), verifying the effectiveness of our NTAA. We further compare our method with four state-of-the-art methods, including Levi Hassner [90], LMTCNN [89], WRN [91], and WRN* [92]. We can see that our NTAA outperforms the competitors by a large margin (e.g., 4.6% improvement on age estimation). These comparisons demonstrate the effectiveness of our NTAA.

E. Unsupervised Domain Adaptation

We finally validate the effectiveness of our method on the unsupervised domain adaptation task on DomainNet [93],

TABLE VIII: Domain adaptation on DomainNet.

| Metric | Models | Accuracy | p-value |
|--------------------------|-------------|-------------------------------------|---------|
| Single Best + NTAA | Source Only | 26.4 (± 0.70) | 0.0066 |
| | Source Only | 28.4 (± 0.68) | |
| Source Combine + NTAA | Source Only | 32.9 (± 0.54) | 0.0002 |
| | Source Only | 34.5 (± 0.60) | |
| Oracle + NTAA | AlexNet | 58.0 (± 0.53) | 0.0079 |
| | AlexNet | 61.1 (± 0.54) | |

which is currently the largest benchmark for multi-source unsupervised domain adaptation. Specifically, it contains about 0.6 million images belonging to 6 domains and 345 categories. The problem definition of standard transfer learning is clearly different from that of domain adaptation. Standard transfer learning means that a backbone of a model is trained on a large source dataset and is finetuned on a small target dataset. Note that the source task is different from the target task in standard transfer learning. For example, the source task is image classification, while the target task is age recognition. Both the source domain and the target domain have training data and testing data. In contrast, domain adaptation means that a model is trained on the source domain and is tested on the target domain. In domain adaptation, the source task is the same as the target task (e.g., both are 1000-category classification). Moreover, the source domain only contains the training set, and the target domain only contains test set.

The difference between standard transfer learning and domain adaptation makes it infeasible to compare our method and domain adaptation method on DomainNet. Although we cannot compare our method with domain adaptation methods, we can compare our method with the single-source method, single-best method, and the oracle method on Domain. As shown in Table VIII, our method outperforms the baselines remarkably. The present study confirms that our method is promising in closing the domain gap. This indicates a promising direction in searching for architecture in unsupervised domain adaptation tasks, which is our future work.

F. Visualization of Architecture Adaptation.

To investigate how the same architecture is transferred to different domain tasks, we visualize the target architectures α^* in the three tasks (CIFAR-10, CIFAR-100, and CUHK03), respectively. These tasks share the same initial architecture α_0 , which is a ResNet50 pretrained on ImageNet. Due to the space limitation, only the 3rd to 10th layers are exhibited in Fig. 4, where we have two observations. First, transferred to the smaller dataset or simpler task, the architectures are towards shallower. For example, CIFAR-10 is the simplest task among the above benchmarks, which has only ten categories. Therefore the transferred architecture in CIFAR-10 is shallowest (only *one* layer). On the contrary, the more challenging

person re-identification task on CUHK03 with 767 training categories has a deeper transferred architecture (4 layers). Second, in addition to removing the redundant depth, different adaptation tasks prefer different new operations. For instance, the CIFAR-10 task enables more noise disturbing operations than the others. This may attribute to the tiny volume of CIFAR-10 dataset, which requires noise disturbing as data augmentation. The above observations confirm the fact that each domain task with a distinct recognition target may need different levels/paths of feature hierarchy, as our NTAA does.

G. Analysis of Stableness and Robustness

To validate whether our results are statistically significant, we perform the architecture adaptation for four times and report the means and the error bars. The results are reported in Table VI and VIII. We can find that our method is stably better than the competing methods. For example, using ResNet50 on CIFAR-10, our NTAA outperforms the method of weight pretraining and finetuning by 1.3%; but the error bar of our method is only 0.04%. This statistical significance verifies the robustness and effectiveness of our method.

Apart from the means and error bars, another useful tool to eliminate the randomness in our conclusion is a statistical test. Although no previous work reports a statistical test, a statistical test really does provide new insight into the NAS. It can serve as a new metric to evaluate NAS's effectiveness, which is quite important to the NAS community. To conduct a statistical test, we perform the following five steps. **Step 1:** State the null hypothesis, i.e., we have H_0 : There is no significant difference between our method's performance and the competitors. **Step 2:** State the alternate hypothesis, i.e., we have H_1 : There is a significant difference between our method's performance and the competitors. **Step 3:** State the α value. We use the default 0.05 as the α value. **Step 4:** Perform variance test in the domain adaptation task on DomainNet. Although the mean accuracy of our NTAA is significantly higher than the competitors, we still challenge whether the superiority of our NTAA is caused by randomness. Therefore, we decide to perform a variance test. We run the architecture transfer experiments four times and use the results to calculate the p-values of the variance test². The statistical test results are reported in Table VIII. As shown, the p-values are 0.0066, 0.0002, and 0.0079. **Step 5:** Since the p-values are less than the α level 0.05, we have to reject the null hypothesis and accept the alternate hypothesis. This indicates that our NTAA is robust and effective.

VI. CONCLUSION

In this paper, we have proposed a novel learning framework called neural transfer and architecture adaptation (i.e., NTAA), which is capable of jointly adapting network weights and network architecture into new domains. The principled analysis has been discussed to explain the reason behind the effectiveness of NTAA by comparing it with the existing solutions to NAS. We have highlighted that preserving the

joint distribution of the network architecture and weights is of importance. Our experiments show that the proposed NTAA outperforms state-of-the-art methods on four computer vision tasks while achieving remarkable accuracy improvements.

In future work, a promising direction is to develop a more powerful inference such as the stochastic Monte Carlo algorithms for exploring the optimal neural network architectures.

ACKNOWLEDGEMENT

This work was supported in part by the National Key Research and Development Program of China under Grant No. 2018YFC0830103, in part by Major Project of Guangzhou Science and Technology of Collaborative Innovation and Industry under Grant 201605122151511, in part by the National Natural Science Foundation of China (NSFC) under Grants 61876045 and 61836012, and in part by Zhujiang Science and Technology NewStar Project of Guangzhou under Grants 201906010057.

REFERENCES

- [1] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, 2020, pp. 13 001–13 008.
- [2] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *ICCV*, 2015, pp. 3730–3738.
- [3] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, 2019, pp. 6105–6114.
- [4] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 2261–2269.
- [5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
- [6] M. Tan and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, 2020, pp. 10 778–10 787.
- [7] Z. Zhang, C. Lan, W. Zeng, X. Jin, and Z. Chen, "Relation-aware global attention for person re-identification," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, 2020, pp. 3183–3192.
- [8] Y. Chen, T. Yang, X. Zhang, G. Meng, C. Pan, and J. Sun, "Detnas: Neural architecture search on object detection," *CoRR*, vol. abs/1903.10979, 2019.
- [9] R. Quan, X. Dong, Y. Wu, L. Zhu, and Y. Yang, "Auto-reid: Searching for a part-aware convnet for person re-identification," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019, pp. 3749–3758.
- [10] C. Liu, L. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and F. Li, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 82–92.
- [11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.
- [12] G. Wang, J. Lai, W. Liang, and G. Wang, "Heterogeneous model transfer between different neural networks," 2021. [Online]. Available: https://openreview.net/forum?id=7xArdn_FkTV
- [13] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, "Momentum contrast for unsupervised visual representation learning," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, 2020, pp. 9726–9735.
- [14] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI*, 2019, pp. 4780–4789.
- [15] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, "Block-wisely supervised neural architecture search with knowledge distillation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, 2020, pp. 1986–1995.

²This can be simply done by using the *anova1* function in MATLAB.

- [16] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," in *ICLR*, 2020, pp. 7132–7141.
- [17] A. Yang, P. M. Esperança, and F. M. Carlucci, "NAS evaluation is frustratingly hard," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.
- [18] G. E. Hinton, T. J. Sejnowski *et al.*, "Learning and relearning in boltzmann machines," *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, pp. 282–317, 1986.
- [19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of Conference on Neural Information Processing Systems 2012, December 3-6, 2012, Lake Tahoe, Nevada, United States*, 2012, pp. 1106–1114.
- [20] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *NIPS*, 2015, pp. 91–99.
- [21] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *TPAMI*, vol. 40, no. 4, pp. 834–848, 2018.
- [22] Y. Wu and K. He, "Group normalization," in *ECCV*, 2018, pp. 3–19.
- [23] X. Shu, J. Tang, G. Qi, Z. Li, Y. Jiang, and S. Yan, "Image classification with tailored fine-grained dictionaries," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 2, pp. 454–467, 2018.
- [24] J. Tang, X. Shu, Z. Li, G. Qi, and J. Wang, "Generalized deep transfer networks for knowledge propagation in heterogeneous domains," *ACM Trans. Multimed. Comput. Commun. Appl.*, vol. 12, no. 4s, pp. 68:1–68:22, 2016.
- [25] X. Shu, G. Qi, J. Tang, and J. Wang, "Weakly-shared deep transfer networks for heterogeneous-domain knowledge propagation," in *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference, MM '15, Brisbane, Australia, October 26 - 30, 2015*, 2015, pp. 35–44.
- [26] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, 2018, pp. 2227–2237.
- [27] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding with unsupervised learning," in *Technical report, OpenAI*, 2018.
- [28] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.
- [29] S. N. Tran and A. S. d'Avila Garcez, "Adaptive transferred-profile likelihood learning," in *2016 International Joint Conference on Neural Networks, IJCNN 2016, Vancouver, BC, Canada, July 24-29, 2016*, 2016, pp. 2687–2692.
- [30] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," *arXiv preprint arXiv:1712.00559*, 2017.
- [31] S. Xie, H. Zheng, C. Liu, and L. Lin, "Snas: stochastic neural architecture search," *ICLR*, 2019.
- [32] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," *arXiv preprint arXiv:1812.03443*, 2018.
- [33] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *ICLR*, 2019.
- [34] K. Maziarz, A. Khorlin, Q. de Laroussilhe, and A. Gesmundo, "Evolutionary-neural hybrid agents for architecture search," *CoRR*, vol. abs/1811.09828, 2018. [Online]. Available: <http://arxiv.org/abs/1811.09828>
- [35] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *CoRR*, vol. abs/1807.11626, 2018. [Online]. Available: <http://arxiv.org/abs/1807.11626>
- [36] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *ICML*, 2018.
- [37] C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," *ICML*, 2019.
- [38] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," *CoRR*, vol. abs/1904.00420, 2019.
- [39] C. Jiang, H. Xu, W. Zhang, X. Liang, and Z. Li, "SP-NAS: serial-to-parallel backbone search for object detection," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, 2020, pp. 11 860–11 869.
- [40] B. Baker, G. Otkrist, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *ICLR*, 2017.
- [41] H. Cai, C. Gan, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *NIPS*, 2019.
- [42] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *ICLR*, 2019.
- [43] X. Chu, B. Zhang, J. Li, Q. Li, and R. Xu, "Scarletnas: Bridging the gap between scalability and fairness in neural architecture search," *CoRR*, vol. abs/1908.06022, 2019.
- [44] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, 2008, pp. 1096–1103.
- [45] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 2536–2544.
- [46] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III*, 2016, pp. 649–666.
- [47] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 1422–1430.
- [48] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI*, 2016, pp. 69–84.
- [49] X. Chen, H. Fan, R. B. Girshick, and K. He, "Improved baselines with momentum contrastive learning," *CoRR*, vol. abs/2003.04297, 2020.
- [50] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, "A simple framework for contrastive learning of visual representations," *CoRR*, vol. abs/2002.05709, 2020. [Online]. Available: <https://arxiv.org/abs/2002.05709>
- [51] J. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. Á. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko, "Bootstrap your own latent: A new approach to self-supervised learning," *CoRR*, vol. abs/2006.07733, 2020. [Online]. Available: <https://arxiv.org/abs/2006.07733>
- [52] X. Chu, B. Zhang, R. Xu, and J. Li, "Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search," *CoRR*, vol. abs/1907.01845, 2019.
- [53] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," in *ICLR*, 2018.
- [54] Anonymous, "Exploring single-path architecture search ranking correlations," in *Submitted to International Conference on Learning Representations*, 2021, under review. [Online]. Available: <https://openreview.net/forum?id=J40FkblddTX>
- [55] P. Mianjy and R. Arora, "On convergence and generalization of dropout training," in *Advances in Neural Information Processing Systems 33 pre-proceedings (NeurIPS 2020)*, 2020.
- [56] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.
- [57] R. V. Soelen and J. W. Sheppard, "Using winning lottery tickets in transfer learning for convolutional neural networks," in *International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019*, 2019, pp. 1–8.
- [58] A. S. Morcos, H. Yu, M. Paganini, and Y. Tian, "One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers," *CoRR*, vol. abs/1906.02773, 2019.
- [59] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015, pp. 1–9.
- [60] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

- [61] M. Farenzena, L. Bazzani, A. Perina, V. Murino, and M. Cristani, "Person re-identification by symmetry-driven accumulation of local features," in *CVPR*. IEEE, 2010, pp. 2360–2367.
- [62] D. Gray and H. Tao, "Viewpoint invariant pedestrian recognition with an ensemble of localized features," in *ECCV*. Springer, 2008, pp. 262–275.
- [63] D. Li, X. Chen, Z. Zhang, and K. Huang, "Learning deep context-aware features over body and latent parts for person re-identification," in *CVPR*, 2017, pp. 384–393.
- [64] L. Zhao, X. Li, Y. Zhuang, and J. Wang, "Deeply-learned part-aligned representations for person re-identification." in *ICCV*, 2017, pp. 3239–3248.
- [65] S. Bai, X. Bai, and Q. Tian, "Scalable person re-identification on supervised smoothed manifold," in *CVPR*, 2017.
- [66] Y. Sun, L. Zheng, W. Deng, and S. Wang, "Svdnet for pedestrian retrieval," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, 2017, pp. 3820–3828.
- [67] Z. Zheng, L. Zheng, and Y. Yang, "Unlabeled samples generated by GAN improve the person re-identification baseline in vitro," in *ICCV*, 2017, pp. 3774–3782.
- [68] C. Su, J. Li, S. Zhang, J. Xing, W. Gao, and Q. Tian, "Pose-driven deep convolutional model for person re-identification," in *ICCV*. IEEE, 2017, pp. 3980–3989.
- [69] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017.
- [70] K. Zhou, Y. Yang, A. Cavallaro, and T. Xiang, "Omni-scale feature learning for person re-identification," in *CVPR*, 2019.
- [71] Z. Zheng, X. Yang, Z. Yu, L. Zheng, Y. Yang, and J. Kautz, "Joint discriminative and generative learning for person re-identification," in *CVPR*, 2019, pp. 2138–2147.
- [72] R. Hou, B. Ma, H. Chang, X. Gu, S. Shan, and X. Chen, "Interaction-and-aggregation network for person re-identification," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 9317–9326.
- [73] M. Zheng, S. Karanam, Z. Wu, and R. J. Radke, "Re-identification with consistent attentive siamese networks," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 5735–5744.
- [74] W. Yang, H. Huang, Z. Zhang, X. Chen, K. Huang, and S. Zhang, "Towards rich feature discovery with class activation maps augmentation for person re-identification," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 1389–1398.
- [75] B. Chen, W. Deng, and J. Hu, "Mixed high-order attention network for person re-identification," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019, pp. 371–381.
- [76] C. Tay, S. Roy, and K. Yap, "Aanet: Attribute attention network for person re-identifications," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 7134–7143.
- [77] J. Guo, Y. Yuan, L. Huang, C. Zhang, J. Yao, and K. Han, "Beyond human parts: Dual part-aligned representations for person re-identification," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019, pp. 3641–3650.
- [78] J. Miao, Y. Wu, P. Liu, Y. Ding, and Y. Yang, "Pose-guided feature alignment for occluded person re-identification," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019, pp. 542–551.
- [79] K. Zhu, H. Guo, Z. Liu, M. Tang, and J. Wang, "Identity-guided human semantic parsing for person re-identification," in *Proc. of European Conference on Computer Vision (ECCV)*, 2020.
- [80] Z. Zhuang, L. Wei, L. Xie, T. Zhang, H. Zhang, H. Wu, H. Ai, and Q. Tian, "Rethinking the distribution gap of person re-identification with camera-based batch normalization," in *Proc. of European Conference on Computer Vision (ECCV)*, 2020.
- [81] X. Jin, C. Lan, W. Zeng, Z. Chen, and L. Zhang, "Style normalization and restitution for generalizable person re-identification," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, 2020.
- [82] J. Zhou, B. Su, and Y. Wu, "Online joint multi-metric adaptation from frequent sharing-subset mining for person re-identification," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, 2020.
- [83] Y. Sun, L. Zheng, Y. Yang, Q. Tian, and S. Wang, "Beyond part models: Person retrieval with refined part pooling (and a strong convolutional baseline)," in *ECCV*, 2018, pp. 480–496.
- [84] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification: A benchmark," in *ICCV*, 2015.
- [85] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *CVPR*, 2018, pp. 7132–7141.
- [86] W. Li, R. Zhao, T. Xiao, and X. Wang, "Deepreid: Deep filter pairing neural network for person re-identification," in *CVPR*, 2014, pp. 152–159.
- [87] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," , 2009.
- [88] E. Eidinger, R. Enbar, and T. Hassner, "Age and gender estimation of unfiltered faces," *IEEE TIFS*, vol. 9, no. 12, pp. 2170–2179, 2014.
- [89] J. Lee, Y. Chan, T. Chen, and C. Chen, "Joint estimation of age and gender from unconstrained face images using lightweight multi-task CNN for mobile applications," in *IEEE 1st Conference on Multimedia Information Processing and Retrieval, MIPR 2018, Miami, FL, USA, April 10-12, 2018*, 2018, pp. 162–165.
- [90] G. Levi and T. Hassner, "Age and gender classification using convolutional neural networks," in *CVPR Workshops*, 2015, pp. 34–42.
- [91] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *BMVC*, 2016.
- [92] P. R. López, D. V. Dorta, G. Cucurull, J. M. Gonfau, F. X. Roca, and J. G. Sabaté, "Pay attention to the activations: a modular attention mechanism for fine-grained image recognition," *CoRR*, vol. abs/1907.13075, 2019.
- [93] X. Peng, Q. Bai, X. Xia, Z. Huang, K. Saenko, and B. Wang, "Moment matching for multi-source domain adaptation," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019, pp. 1406–1415.



Guangrun Wang is currently a Ph.D. candidate in the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. He received the B.E. degree from Sun Yat-sen University in 2014. From 2015 to 2017, he was a visiting scholar with the Department of Information Engineering, The Chinese University of Hong Kong. His research interests include machine learning and computer vision. He is the recipient of the 2018 Pattern Recognition Best Paper Award and one ESI Highly Cited Paper.



Liang Lin (M'09-SM'15) is a Full Professor at Sun Yat-sen University. He served as the Executive R&D Director and Distinguished Scientist of Sense-Time Group from 2016 to 2018, taking charge of cutting-edge technology transfer into products. He has authored or co-authored more than 200 papers in leading academic journals and conferences (e.g., TPAMI/IJCV, CVPR/ICCV/NIPS/ICML/AAAI). He is an associate editor of IEEE Trans, Human-Machine Systems, and IET Computer Vision. He served as Area Chairs for numerous conferences such as CVPR and ICCV. He is the recipient of numerous awards and honors including Wu Wen-Jun Artificial Intelligence Award for Natural Science, ICCV Best Paper Nomination in 2019, Annual Best Paper Award by Pattern Recognition (Elsevier) in 2018, Best Paper Dimond Award in IEEE ICME 2017, Google Faculty Award in 2012, and Hong Kong Scholars Award in 2014. He is a Fellow of IET.



Rongcong Chen received his B.E. degree from the School of Mathematics, Sun Yat-sen University, Guangzhou, China, in 2017, where he is currently pursuing his Master's Degree in computer science with the School of Data and Computer Science. His current research interests include computer vision and machine learning.



Guangcong Wang is pursuing a Ph.D. degree in the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. He received the B.E. degree in communication engineering from Jilin University (JLU), Changchun, China, in 2015. His research interests are computer vision and machine learning. He has published several works on person re-identification, weakly supervised learning, semi-supervised learning, and deep learning.



Jiqi Zhang is currently an undergraduate student in the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include machine learning and computer vision.