WealthAdapt: A General Network Adaptation Framework for Small Data Tasks

Bingyan Liu, Yao Guo, Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (MOE),
Dept of Computer Science, School of EECS, Peking University, Beijing, China
{lby_cs,yaoguo,cherry}@pku.edu.cn

ABSTRACT

In this paper, we propose a general network adaptation framework, namely **WealthAdapt**, to effectively adapt a large network for small data tasks, with the assistance of a wealth of related data. While many existing algorithms have proposed network adaptation techniques for resource-constrained systems, they typically implement network adaptation based on a large dataset and do not perform well when facing small data tasks. Because small data have poor feature expression ability, it may result in incorrect filter selection and overfitting during fine-tuning in the network adaptation process. In WealthAdapt, we first expand the target small data task with the wealth of big data, before we perform network adaptation, in order to enrich the features and improve the fine-tuning performance during adaptation. We formally establish network adaptation for small data tasks as an optimization problem and solve it through two main techniques: *model-based fast selection* and *wealth-incorporated iteration adaptation*. Experimental results demonstrate that our framework is applicable to both the vanilla convolutional network VGG-16 and more complex modern architecture ResNet-50, outperforming several state-of-the-art network adaptation pipelines on multiple visual classification tasks including *general object recognition*, *fine-grained object recognition* and *scene recognition*.

CCS CONCEPTS

• **Computing methodologies** → **Computer vision**; • **Computer systems organization** → *Embedded systems*.

KEYWORDS

Network Adaptation; Small Data Tasks; Big Data; Selection; Incorporation

ACM Reference Format:
Bingyan Liu, Yao Guo, Xiangqun Chen . 2019. WealthAdapt: A General Network Adaptation Framework for Small Data Tasks. In *Proceedings of the 27th ACM International Conference on Multimedia (MM'19), Oct. 21–25, 2019, Nice, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3343031.3351035

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MM '19, October 21–25, 2019, Nice, France
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6889-6/19/10…$15.00
https://doi.org/10.1145/3343031.3351035

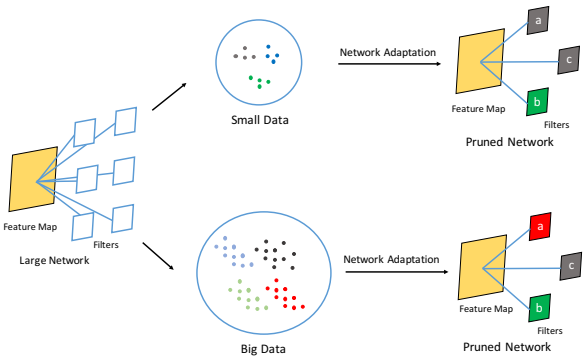

Figure 1: Illustration of two existing network adaptation (i.e., network pruning) methods for small data tasks. The top method implements network adaptation based on a small data task. The bottom method implements network adaptation based on a big data task. Both methods cannot generate a good pruned network for small datasets.

1 INTRODUCTION

Convolutional neural networks (CNNs) have been proved effective in various computer vision tasks such as image classification and object detection [16, 22]. Meanwhile, in order to achieve excellent performance, they typically require sufficient training data, which is hard to satisfy for many small data tasks. In these cases, performance can be improved by fine-tuning a well-designed deep network that was pre-trained on a much larger labeled vision dataset, such as ImageNet. However, the fine-tuned network is still a heavyweight architecture with massive computing power and memory demand, which cannot be executed efficiently on resource-constrained devices and systems.

To address this problem, network compression has been proposed and studied extensively in recent years, resulting in techniques such as low-rank approximation [17, 20, 21, 34], parameter quantization [14, 15] and network pruning [1, 9, 10, 12, 23–25, 35]. Among them *network pruning* has become a promising and widely applicable technique for accelerating and compressing CNNs. During network pruning, one can easily remove the weights, the filters and even the layers of a CNN, without incurring significant performance drop.

In the context of small data tasks, pruning can be roughly divided into two categories: *pruning before fine-tuning* and *pruning after fine-tuning*. For *pruning before fine-tuning*, Liu *et al.* [23] pruned the pre-trained network first before fine-tuning the network with the small data. As examples of *pruning after fine-tuning*, Molchanov *et al.* [25] and Zhong *et al.* [38] directly fine-tuned the pre-trained networks using the small data and then applied pruning.

RIGHTSLINK

However, both categories exhibit some problems when pruning for small data tasks. For *pruning before fine-tuning*, it implements pruning with a large-scale data task, which may not be suitable for the small data task due to the mismatch of feature space in the two tasks. As shown in Figure 1, if we prune the large network with the big data task, the filters *a,b,c* in the pruned network may represent the features of the red, green and gray data respectively. However, the features of red data are useless, or even harmful, to our small data task, while the useful features of blue data cannot be reflected. So fine-tuning this pruned network may result in bad performance.

In the case of *pruning after fine-tuning*, it implements pruning with the small data task. Note that the pruning process also needs fine-tuning for recovering accuracy in every iteration, so severe overfitting may be incurred due to limited labeled data. In addition, limited labels are more likely to have poor feature expression ability, resulting in incorrect filter selection. As a result, unnecessary overlapping features and incorrect features can be largely introduced, which inhibits the effectiveness of pruning. Figure 1 illustrates the network adaptation process with small data tasks. The features of gray and green data can be reflected by the filters. However, the filter *a* and *c* are overlapped and the features of blue data are discarded because the pruning implementation is based on an overfitted network and a small data task.

To the best of our knowledge, there have been no suitable network adaptation approaches specifically designed for small data tasks. Our insight to solve the problem is that it is possible to deal with small data tasks well if the big data such as ImageNet can be utilized effectively. On one hand, incorporating the rich features of big data appropriately contributes to improving the feature expression ability because the data representing useful features can be expanded by our big data. On the other hand, the overfitting problem caused by fine-tuning in each pruning iteration can be largely alleviated by a joint fine-tuning step with the assistance of big data.

In this paper, we attempt to incorporate the wealth of big data into the network adaptation pipeline, in order to generate a better pruned network for small data tasks, targeting resource-constrained systems. To accomplish our goal, we face two important challenges:

-*challenge 1*: The features in the big data are huge and diverse. Which features may be beneficial to our target task and how should we select them efficiently?

-*challenge 2*: The standard network adaptation (pruning) pipeline mainly includes two steps: the pruning metric for defining the importance of weights/filters/layers and the fine-tuning process for recovering the accuracy. How could we incorporate the big data into these steps and make them more effective?

In this paper, we propose a general network adaptation framework, namely **WealthAdapt**, to effectively adapt a large CNN for small data tasks. Specifically, we introduce two key techniques in the framework to address the above challenges:

- *Model-based fast selection.* This technique utilizes the network pre-trained by the big data and puts the target data into it to enforce *class-level distribution*. According to the distribution, we can get a coarse observation to the relation between two tasks. Furthermore, we implement *similarity-level filtering and boosting* based on the class distribution to create a better selection. Note that we do not compare each data with the whole big data, so the processing speed is fast.
- *Wealth-incorporated iteration adaptation.* The wealth of big data is incorporated both in the pruning metric and the fine-tuning process. For the pruning metric, we propose a *wealth&task balanced pruning* according to the Taylor-based ranking. For the fine-tuning process, we break down the single fine-tuning process into two processes: *long task tuning* and *short wealth tuning*. Using these techniques, we prune and fine-tune the network iteratively, such that the big data can be utilized more suitably.

This paper makes the following main contributions:

- We propose **WealthAdapt**, a general network adaptation framework for small data tasks with the assistance of big data. To the best of our knowledge, this is the first work that incorporates the wealth of big data into the network adaptation pipeline.
- We design the pipeline of **WealthAdapt**, which includes two key techniques, *model-based fast selection* and *wealth-incorporated iteration adaptation*, to solve the selection and incorporation problems towards the big data, which make better utilization of the wealth of big data, without incurring significant time overhead.
- Experiments on VGG-16 and ResNet-50 demonstrate that our framework outperforms several state-of-the-art network adaptation pipelines on different visual classification tasks, including *general object recognition*, *fine-grained object recognition* and *scene recognition*.

2 RELATED WORK

2.1 Transfer Learning

Transfer learning aims to apply knowledge learned by the source domain (big data) to similar target domains [26]. It can be roughly divided into two categories.

Fine-tuning. Recently, researchers have found that the representation learned from a large-scale dataset, such as the ImageNet ILSVRC dataset [28], can be effectively transferred to a small but similar dataset [2, 29, 36]. With these observations, fine-tuning the pre-trained networks has become a common method to implement transfer learning [7]. Meanwhile, fine-tuning may introduce the problems of overfitting and negative transfer, due to insufficient training data and the mismatch between two domains. To solve these issues, Ge *et al.* [6] and Cui *et al.* [4] utilized the wealth of big data to implement joint fine-tuning, which improves the performance greatly.

Domain adaptation. Domain adaptation have been proposed to mitigate the burden of manual labeling and cope with different distributions between two domains to avoid the negative transfer problem. The main idea is to add adaptation layers or a subnetwork as domain discriminator leveraging the idea of generative adversarial network (GAN) [3, 8, 31, 32, 37]. Note that domain adaptation assumes that the source and target domains share the same label space, or the target label space is a subspace of the source label space.

RIGHTSLINK

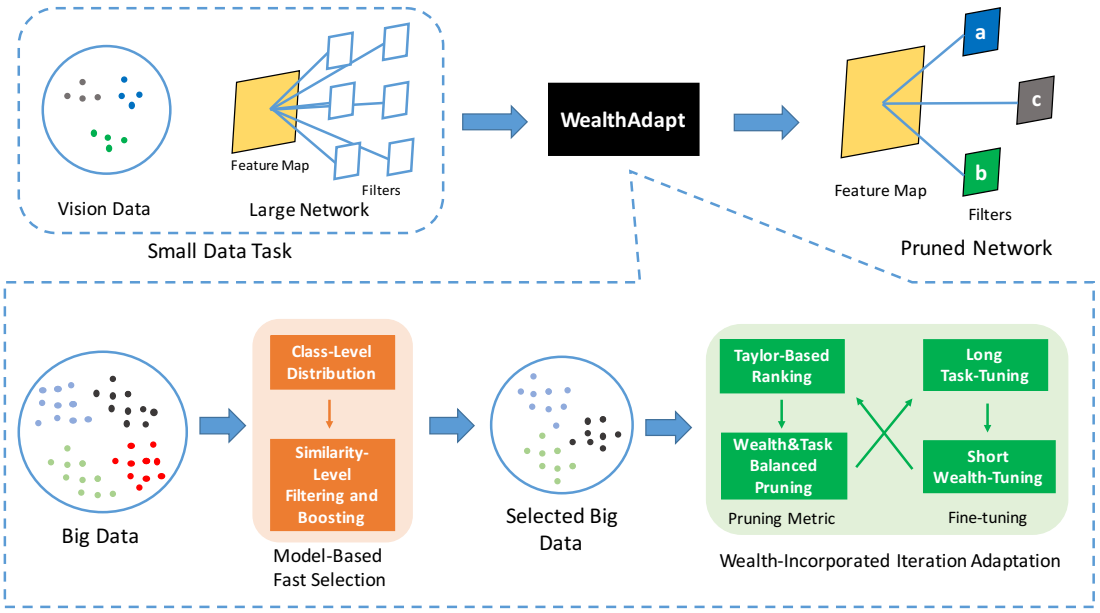

Figure 2: Overview of WealthAdapt. After the processing of *model-based fast selection*, it generates a selected subset of the big data, which is more similar to the target small data. We then use this selected subset of big data and the original vision data to implement *wealth-incorporated iteration adaptation*, which generates a pruned network with more suitable features.

However, these transfer methods typically focus on the training process, which cannot be applied to network adaptation directly.

2.2 Network Pruning

Recently, a large number of efforts have been made in network pruning. By defining various metrics, one can prune the redundant elements without incurring significant performance drop. It mainly includes three categories.

Weight pruning. Weight pruning aims at removing the redundant weights to shrink the network. Han *et al.* [10] proposed an iterative weight pruning method for a compact network. Their insight is that a small weight connectivity below a threshold is redundant and should be discarded. Lebedev and Lempitsky [18] introduced the group-sparsity regularization to the loss function for forcing some weights to zeros. Thus these zero weights can be removed. However, weight pruning may result in unstructured sparsity of networks and needs the support of a special accelerator.

Filter pruning. Filter pruning is a kind of group-wise sparsity method to prune filters for accelerating the CNN networks. Li *et al.* [19] measured the importance of each filter by calculating its absolute weight sum. He *et al.* [13] and Luo *et al.* [24] leveraged LASSO regression and greedy algorithms respectively to prune redundant filters layer by layer. Their methods enabled structured sparsity and off-the-shelf libraries can be used directly.

Transfer pruning. Similar to our work, transfer pruning also focuses on network adaptation for small data tasks. It can be roughly divided into two categories: *pruning before fine-tuning* and *pruning after fine-tuning*. For *pruning before fine-tuning*, Liu *et al.* [23] pruned the pre-trained network first before fine-tuning the network with the small data. As examples of *pruning after fine-tuning*, Molchanov *et al.* [25] and Zhong *et al.* [38] fine-tuned the pre-trained network with small data tasks before pruning, and then pruned unimportant filters and fine-tuned weights iteratively.

However, these pruning methods do not utilize the big data and may result in poor performance when facing small data tasks. Our goal is to overcome this limitation by incorporating the wealth of big data into the pruning pipeline.

3 OUR FRAMEWORK: WEALTHADAPT

We will first give an overview of the WealthAdapt framework, then present a formal definition of the problem and the proposed techniques. The detailed algorithm of the WealthAdapt pipeline is shown in Algorithm 1.

3.1 WealthAdapt Overview

Figure 2 illustrates the pipeline of the proposed WealthAdapt framework, which targets network adaptation for resource-constrained systems, especially when the target task includes a very small dataset, which is typical for resource-constrained systems.

Our main goal is to facilitate these small data tasks with the wealth of big data, first select a subset of the big data that are similar to the target task but with much larger scale (i.e., *model-based fast selection*), and then use the selected big data and the original vision data together to implement *wealth-incorporated iteration adaptation*, generating a pruned network with more suitable features.

More specifically, for the example in Figure 2, which shows the proposed pipeline in **WealthAdapt**, we can generate a pruned network containing blue, gray and green features while eliminating the influence of red features in the big data, which will obviously

RIGHTSLINK

improve the performance for small data tasks. Note that our framework is generalizable to any elements of CNNs, although here we only focus on filter pruning.

3.2 Problem Formulation

We now introduce the symbols and annotations to define the optimization objective. The pruned CNN network can be represented by $(\mathcal{W}^*|\mathcal{D}^*)$, where $\mathcal{W}^* \in \mathbb{R}^{L\times D\times C\times K\times K}$ is the whole weights of the pruned network, which contains L layers, D filters, C channels with $K\times K$ kernel size. $\mathcal{D}^*$ is the data used to prune the network. Towards network adaptation problem for small data tasks, the objective can be formulated as:

$$\begin{aligned} \max_{\mathcal{W}_s^*, \mathcal{D}_s^*} \quad & Acc[\ (\mathcal{W}_p^*|\mathcal{D}_w^*) \leftarrow \mathcal{D}_r\ ] \\ s.t. \quad & (\mathcal{W}_p^*|\mathcal{D}_w^*) = WealthAdapt(\mathcal{W}_o|\mathcal{D}_w \cup \mathcal{D}_r) \\ & \mathcal{W}_p^* \subseteq \mathcal{W}_o \\ & \mathcal{D}_w^* \subseteq (\mathcal{D}_w \cup \mathcal{D}_r) \end{aligned} \tag{1}$$

where $(\mathcal{W}_p^*|\mathcal{D}_w^*)$ represents the network pruned by $\mathcal{D}_w^*$, $\mathcal{D}_w^*$ is a compounded subset of the big data $\mathcal{D}_w$ and the target task data $\mathcal{D}_r$, $\mathcal{W}_o$ represents the large network, *Acc* represents the performance towards the pruned network fine-tuned by the target task $\mathcal{D}_r$. Our goal is to use the proposed **WealthAdapt** framework to generate the pruned network $\mathcal{W}_p^*$ and the subset $\mathcal{D}_w^*$, in order to maximize the *Acc*.

In the following sections, we will describe the details of the two proposed techniques: *model-based fast selection* to get $\mathcal{D}_w^*$ and *wealth-incorporated iteration adaptation* to get $\mathcal{W}_p^*$.

3.3 Model-Based Fast Selection

The goal of the first technique, *model-based fast selection*, is to find a more suitable subset of the big data to assist the network adaptation process. The concept *model-based* indicates that we utilize the model pre-trained by the big data. Based on the pre-trained model, we only need to implement a forward propagation rather than comparing the similarity for each data, which can significantly accelerate the selection process.

The selection process involves mainly two steps: *class-level distribution* and *similarity-level filtering and boosting*.

3.3.1 Class-Level Distribution. At the class level, we implement *class-level distribution* mainly by network forward propagation. First, for each class of the small data task, we sample n data and feed each data x_j $(0 \le j \le n-1)$ into the pre-trained model G_p for forward propagation, to get the corresponding $y_j = G_p(x_j)$. We observe that y_j represents the class-level probability distribution over the big data label space given a x_j, which indicates the relation between two domains. Motivated by this observation, we can describe the *class-level distribution* problem using the following formulas:

$$C_k = \frac{1}{n}\sum_{j=0}^{n-1} mask(G_p(x_j^k)) \tag{2}$$

$$mask(value) = \begin{cases} 1 & value \in top5\% \\ 0 & value \in others \end{cases} \tag{3}$$

Algorithm 1 The WealthAdapt Framework

Input: big data $\mathcal{D}_w$, a large network $\mathcal{W}_o$, target task data $\mathcal{D}_r$
Output: a suitable network $\mathcal{W}_p^*$ pruned by $\mathcal{D}_w^*$

1: Calculate the *class-level distribution* for each target class by Eq. 2 and Eq. 3
2: **for** $i = 1; i \le num(target\_class); i++$ **do**
3: Select the first N classes of the big data for the i_{th} target class by Eq. 4
4: **end for**
5: **for** $i = 1; i \le num(initial\_selected\_classes); i++$ **do**
6: Calculate the *Class_Score* of the i_{th} selected class by Eq. 5
7: **end for**
8: Implement different operations according to Eq. 6 and get the final $\mathcal{D}_w^*$
9: **for** $i = 1, i \le num(filters); i++$ **do**
10: Calculate the Θ_{WA} of i_{th} filter by Eq. 9
11: **end for**
12: Remove the filters with low Θ_{WA}
13: Get the compact and suitable network $\mathcal{W}_p^*$

where C_k represents the big data distribution of the k_{th} class of the small data task. x_j^k represents the j_{th} sample of the k_{th} class. Function *mask*() means that if the value belongs to the *top*5% data of $G_p(x_j^k)$, the result is 1; otherwise the result is 0. By accumulating the results of *mask*(), we can see which classes in the big data are more similar to the target class k given n samples.

3.3.2 Similarity-Level Filtering and Boosting. The results of the previous step can help us get a coarse conclusion about which classes of the big data are more similar to our target task. However, there still exist two problems that may prevent a more accurate selection. Firstly, for some target classes, there are no corresponding features in the big data. So we should not take these classes into consideration when implementing *class-level distribution*. Secondly, for a certain class of the big data, it may benefit several target classes. This class can be regarded as a more important class to our small data task. Therefore, because the *class-level distribution* step regards all these classes equally, it cannot make perfect selection.

We thus propose *similarity-level filtering and boosting* to solve the above mentioned two problems. At first, for each target class, we select the first N max-value classes of the big data according to the *class-level distribution* as initial selected classes. This process can be achieved by a function *SelectTopN*(), which is formulated as:

$$SelectTopN() = Select_{1,2\ldots N}(Sorted(C_k)) \tag{4}$$

where *Sorted*() sorts the distribution C_k and *Select*() picks out the first N corresponding classes of the big data. Note that the initial selected classes may have many overlaps because a certain class of the big data may correspond to several target task classes.

Next, we illustrate *Class_Score*, which is used to measure the similarity-level of the initial selected classes. The *Class_Score* is

RIGHTSLINK

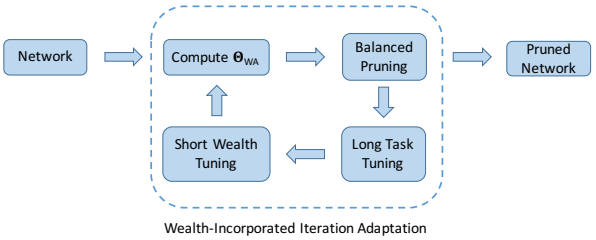

Figure 3: The workflow of wealth-incorporated iteration adaptation.

defined as:

$$Class\_Score(C^j) = \sum_{i=0}^{fre(C^j)} value(C^j)_i \tag{5}$$

where $Class\_Score(C^j)$ represents the score of the j_{th} initial selected class. $fre(C^j)$ represents the frequency of the C^j in the whole selected classes. $value(C^j)$ represents the actual value of the j_{th} class computed by $SelectTopN()$. We accumulate these $value$ to get the final $Class\_Score(C_s^j)$.

According to the score, we break down the initial selected classes into three categories for different operations:

$$C_s^j = \begin{cases} filtering & Class\_Score(C_s^j) < \tau \\ boosting & Class\_Score(C_s^j) \in top30\% \\ remaining & Class\_Score(C_s^j) \in others \end{cases} \tag{6}$$

where *boosting* means we will sample more data in the class for better assistance. *remaining* means we keep the class unchanged and $filtering$ means we will filter out the class due to its poor similarity. τ is the threshold used for filtering.

Through a series of processing in *similarity-level filtering and boosting*, we can get final selected classes of the big data. After combining the selected results with the small data task, we now get the $\mathcal{D}_w^*$.

3.4 Wealth-Incorporated Iteration Adaptation

The next technique in WealthAdapt is called *wealth-incorporated iteration adaptation*, which incorporates the wealth of big data into the network adaptation process. Note that the network adaptation process includes two steps: pruning metric and fine-tuning, which will be modified respectively in our WealthAdapt framework in order to achieve better performance.

3.4.1 Pruning Metric. The aim of the pruning metric is to define the importance of the filters for the network and prune the filters with less importance. There are two main policies to implement pruning: local ranking pruning and global ranking pruning. For local ranking pruning, we need to decide how many filters to prune for each layer, which is typically a manual task requiring extensive experiences. So here we only focus on global ranking pruning for its simplicity and efficiency. Next, we will first introduce *Taylor-based ranking*, and then present *wealth-task balanced pruning* based on the ranking method.

Taylor-Based Ranking. For a pruning problem, we can always formulate it as an optimization problem as follow:

$$\min \quad \Delta\mathcal{L} = |\mathcal{L}(\mathcal{W}_p^*) - \mathcal{L}(\mathcal{W}_o)| \tag{7}$$

where $\mathcal{L}$ represents the loss function. Our goal is to find the $\mathcal{W}_p^*$ to minimize the $\Delta\mathcal{L}$.

Molchanov *et al.* [25] proposed a global ranking method based on Taylor expansion that approximated the change in the cost function induced by pruning network parameters. By a series of derivation and simplification, they define the Θ_{TE} to measure the importance of each filter. The Θ_{TE} is formulated as:

$$\Theta_{TE}(\mathcal{F}) = \left| \frac{\partial \mathcal{L}}{\partial \mathcal{F}(X)} \mathcal{F}(X) \right| \tag{8}$$

where $\mathcal{F}$ represents the filter and X represents the corresponding feature map. The filters with low Θ_{TE} should be pruned.

Motivated by this method, we propose a *wealth-task balanced pruning* method. It can not only prune the filters in a global way, but also utilize the wealth of big data for better performance.

Wealth-Task Balanced Pruning. If we just prune with the task data, the pruned network may not be good enough due to the poor feature expression ability of the very small data. So we incorporate the big data into the pruning metric with the following formula:

$$\Theta_{WA}(\mathcal{F}) = \lambda_1 \left| \frac{\partial \mathcal{L}}{\partial \mathcal{F}(X_w)} \mathcal{F}(X_w) \right| + \lambda_2 \left| \frac{\partial \mathcal{L}}{\partial \mathcal{F}(X_r)} \mathcal{F}(X_r) \right| \tag{9}$$

where Θ_{WA} represents the metric used to measure the importance of filters. X_w and X_r represent the feature map generated by the big data input and small data task input. λ_1 and λ_2 are trade-off weights balancing the proportion of the wealth incorporation.

We regard the selected $\mathcal{D}_w^*$ as the input and prune the filters according to the Θ_{WA}, which contributes to more accurate pruning because the data representing useful features are expanded by our selected big data.

3.4.2 Fine-Tuning. After the pruning process, fine-tuning should be conducted to recover the accuracy lost due to pruning. We incorporate the big data by breaking down the single fine-tuning process into two steps: *long task tuning* and *short wealth tuning*.

Long Task Tuning. Considering our final goal is to enhance the performance for the small data task, the task tuning should dominate the whole fine-tuning process. So we implement *long task tuning* in order to fit the task data as much as possible. The word *long* here indicates that we fine-tune the network with the task data for a long time.

Short Wealth Tuning. Fine-tuning with only the task data will cause overfitting, which introduces negative influence in the next pruning iteration. We incorporate the wealth of big data by *short wealth tuning*, which means we fine-tune the network with the big data for a short time, in order to alleviate too quick overfitting as well as incorporating more useful features. Note that a new FC layer will replace the original FC layer for adapting our big data and we only change the weights of CONV layers because the final output of the original FC layer is not suitable to the labels of the big data.

RIGHTSLINK

Table 1: The results of model-based fast selection.

Model-Based Fast Selection	Caltech101	CUB200	MIT67
# of training classes	102	200	67
# of training data	2,040	1,000	1,340
# of initial selected classes	317	57	103
# of final selected classes	194	44	57
# of final selected data	69,800	15,800	20,500

3.4.3 Workflow. The workflow of *wealth-incorporated iteration adaptation* is shown in Figure 3. It consists of four steps: 1) Compute the Θ_{WA} for measuring the importance of each filter; 2) Implement our balanced pruning according to the computed Θ_{WA}; 3) Long task tuning for recovering the target accuracy; 4) Short wealth tuning for restraining overfitting and incorporating more useful features. We run the four steps iteratively until the result meets the user's requirements and get the final $\mathcal{W}_p^*$.

4 EXPERIMENTS

4.1 Experimental Settings

4.1.1 Networks and Datasets. The proposed framework is implemented on two widely used networks: VGG-16 [30] and ResNet-50 [11]. We consider the ImageNet2012 dataset as the big data for its rich features. The performance is evaluated on three datasets: Caltech101 [5], CUB200 [33] and MIT67 [27]. Caltech101 is a general object recognition dataset with 102 object categories. We randomly sample 20 data for each category to simulate the label constraint. CUB200 is a fine-grained object recognition with 200 different bird species. Because its labels are harder to get, we only sample 5 data for each category. MIT67 is a scene dataset with 67 scene categories and 20 data are sampled for each scene as the training set.

4.1.2 Comparison with State-of-the-Art Methods. We compare our framework with two main pruning pipelines for small data tasks: *pruning before fine-tuning* (*PreAdapt*) [23] and *pruning after fine-tuning* (*TuneAdapt*) [25]. *PreAdapt* prunes the pre-trained network first before fine-tuning for the target task, while *TuneAdapt* fine-tunes the pre-trained network with the target task before applying pruning. We faithfully implement them according to what they are disclosed in the corresponding papers. Both methods use Taylor-based ranking for pruning.

4.1.3 Implementation Details. All our experiments are conducted within PyTorch and every mini-batch includes 32 224×224 images in the training stage. For *model-based fast selection*, the parameter n is set to 20 and N is set to 5. The threshold τ is set to 10 for filtering. For *wealth-incorporated iteration adaptation*, λ_1 and λ_2 are set to 0.2 and 0.8 to balance the importance of filters.

After generating the pruned network, we fine-tune it with 100 epochs on the above mentioned three datasets. For the learning rate, gradually annealing the learning rate over the course of training is adopted. Specifically, the learning rate is set to 0.001 in the first 30 epochs, 0.0008 from the $31st$ epoch to the $70th$ epoch and 0.0006 for the rest. Because the training configuration can have a large impact on the accuracy, we apply the same training configuration to all the methods we compared.

Table 2: Comparison of different methods with three visual tasks on VGG-16. Note that "G" represents the process of global average pooling.

VGG-16	Methods	Caltech101	CUB200	MIT67
G_Pruned_0%	*Original*	83.39	41.34	55.23
G_Pruned_10%	*PreAdapt[23]*	80.38	25.42	48.33
	TuneAdapt[25]	83.90	43.21	55.56
	WealthAdapt	**84.47**	**46.47**	**56.59**
G_Pruned_30%	*PreAdapt[23]*	78.49	19.62	42.65
	TuneAdapt[25]	80.13	38.87	53.49
	WealthAdapt	**82.50**	**46.15**	**54.74**
G_Pruned_50%	*PreAdapt[23]*	73.06	19.51	38.25
	TuneAdapt[25]	76.28	28.04	46.07
	WealthAdapt	**81.57**	**42.00**	**51.17**

Table 3: Comparison of different methods with three visual tasks on ResNet-50.

ResNet-50	Methods	Caltech101	CUB200	MIT67
Pruned_0%	*Original*	91.06	52.93	64.13
Pruned_10%	*PreAdapt[23]*	87.95	36.82	55.99
	TuneAdapt[25]	87.36	43.68	57.33
	WealthAdapt	**88.36**	**45.18**	**58.67**
Pruned_30%	*PreAdapt[23]*	85.85	32.71	52.98
	TuneAdapt[25]	84.64	37.85	54.00
	WealthAdapt	**86.82**	**40.90**	**55.06**
Pruned_50%	*PreAdapt[23]*	83.07	28.80	49.22
	TuneAdapt[25]	81.50	32.51	48.50
	WealthAdapt	**84.22**	**38.52**	**50.28**

4.2 Experimental Results

4.2.1 Model-Based Fast Selection. Table 1 presents the results of our selection technique. We selected 300 data for each *remaining* class and 500 data for each *boosting* class. The partition between the *remaining* class and the *boosting* class is defined by Eq. 6. For all three datasets, we are able to select over 15,000 data to assist the network adaptation process, which will obviously help improve the performance of the final pruned model.

More specifically, we selected more data for Caltech101 (34X compared to the original data) because it is more similar to our big data. For the other two datasets, the size of the selected data is relatively small (15X for both cases) due to their low similarity. Furthermore, we observe that the number of final selected classes is significantly fewer than the initial selected classes, especially for the Caltech101 dataset, which indicates that the filtering process successfully eliminated the influence of low-related classes.

4.2.2 Wealth-Incorporated Iteration Adaptation. Table 2 and Table 3 present a series of pruned models with different pruning rates and their corresponding performance after fine-tuning on three small data tasks.

For VGG-16, a global average pooling layer is created first to replace their original FC layers as an initialization step for reducing the huge number of parameters. We use "G" to represent the process. In addition, the first and the last CONV layers are preserved without

RIGHTSLINK

Table 4: The performance comparison between original networks and their corresponding pruned networks. Note that the "G" in VGG-16 is used to represent the process of global average pooling.

Network	#Flops(G)	#Params(M)	Inference Time(ms)
VGG-16	15.62	138.36	99.8
VGG-16_G_Pruned10%	14.03	12.78	95.0
VGG-16_G_Pruned30%	10.03	7.76	77.6
VGG-16_G_Pruned50%	6.18	4.13	58.7
ResNet-50	4.14	25.56	61.4
ResNet-50_Pruned10%	3.78	21.59	60.3
ResNet-50_Pruned30%	2.91	15.42	55.8
ResNet-50_Pruned50%	2.10	10.68	47.8

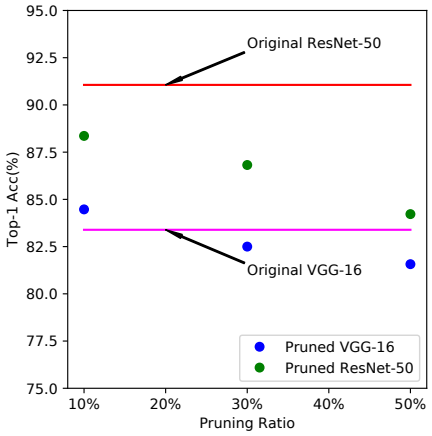

Figure 4: Accuracy comparison between the original networks and their corresponding pruned networks.

implementing pruning. For ResNet-50, we only prune the first and the second CONV layers in each residual block in order to maintain the structure of the original network.

From the two tables, we can observe that: (1) Our framework outperforms the other two state-of-the-art methods in all three datasets no matter what the pruning ratio is. (2) For the fine-grained task CUB200 that contains fewer labels, the assistance of the big data is more effective. The reason may lie in that fewer data are more likely to exhibit poor feature expression ability and cause overfitting in each pruning iteration. So the help from big data becomes very important. (3) For VGG-16, the performance of *G_Pruned*_10% network exceeds the original network in all three datasets. However, for ResNet-50, the accuracy is dropping as the pruning ratio increases. We will analyze the phenomenon detailedly in Section 4.3.1.

4.3 Detailed Analysis

In this subsection, we conducted four detailed analyses, including *pruned network vs. original network*, *sensitivity of pruning ratio*, *sampling number analysis* and *distribution of pruned filters*. Note that the last three analyses are based only on VGG-16 due to limited space.

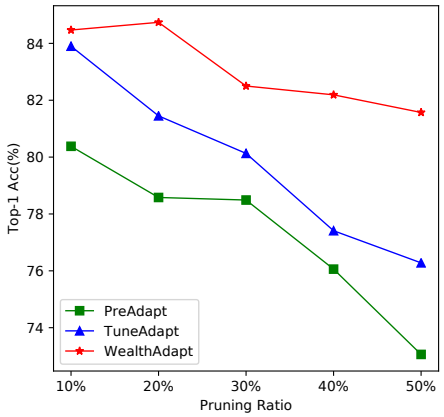

Figure 5: Sensitivity analysis of pruning ratios to different methods.

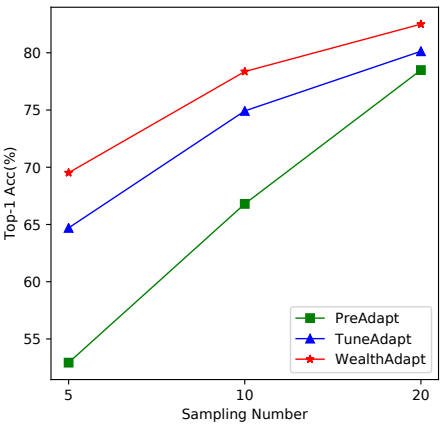

Figure 6: The influence of sampling numbers to different methods.

4.3.1 Pruned Networks vs. Original Networks. We now compare our pruned networks with their corresponding original networks, including *performance comparison* and *accuracy comparison*.

For *performance comparison*, Table 4 demonstrates the $flops$, *params* and $inference\ time$ to measure the performance between the original network and the pruned network. For VGG-16, our framework can shrink the network dramatically, from 138.36M to 4.13M when the pruning ratio is set to 50%. But the $flops$ and $inference\ time$ are relatively large and long because the main computations are in the CONV layers while VGG-16 does not process the CONV layers efficiently. For ResNet-50, we observe that the $flops$ and $inference\ time$ have been improved for its skip trick. However, the network size is relatively large. So if we have the requirement for the size of the network, VGG-16 is recommended. Otherwise ResNet-50 is more suitable.

For *accuracy comparison*, Figure 4 shows the accuracy in pruned networks and their original baselines on the Caltech101 dataset. We can see that: (1) When implementing different pruning ratios, some pruned VGG-16 networks can exceed their baselines, while all the pruned ResNet-50 networks are below their baselines. That may be because the overfitting in VGG-16 is more severe due to its direct

RIGHTSLINK

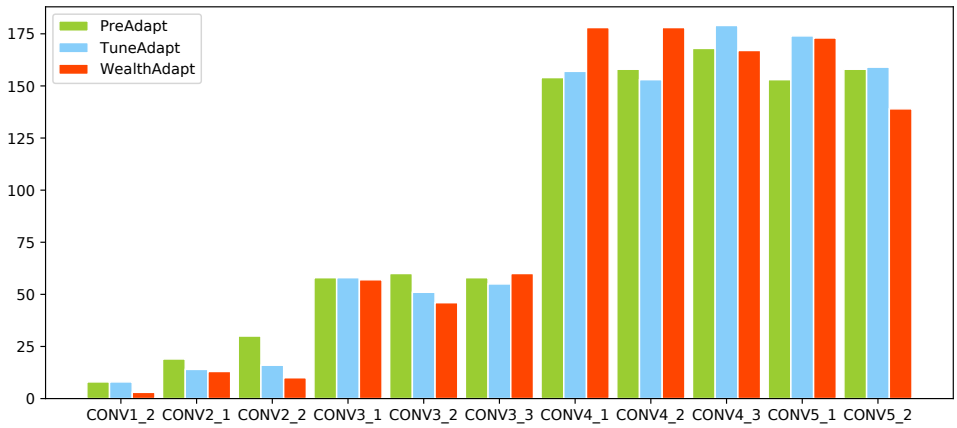

Figure 7: Distribution of pruned filters to different methods. Note that for VGG-16 we do not implement pruning to the first and the last CONV layer.

connection to each CONV layer. So when some filters are pruned, the overfitting becomes slow, which enhances the accuracy. However, the overfitting problem rarely occurs in ResNet-50 because of its massive skip connection. (2) Although pruning cannot enhance the accuracy for ResNet-50, the pruned ResNet-50 network is still superior to the VGG-16 pruned network with the same pruning ratio, which indicates that the architecture of ResNet-50 is better towards this dataset.

4.3.2 Sensitivity of Pruning Ratios. Figure 5 illustrates the pruning sensitivity of VGG-16 on the Caltech101 dataset. When the pruning ratio is small (such as 10%), *PreAdapt* and *TuneAdapt* can reach a relatively comparable accuracy. That may be because at the moment the capacity of the network dominates the performance, which means the potential of the big data cannot be utilized completely. As the pruning ratio increases, the network pruned by our framework can still preserve a good accuracy while the performance of other methods drops quickly. In a word, the more we prune, the more we will see the benefits of our framework.

4.3.3 Sampling Number Analysis. We sample 5, 10, 20 data of each class in Caltech101 and use them as the training data to perform fine-tuning. The networks we picked are VGG-16 with the 30% pruning ratio pruned by three methods, respectively. As shown in Figure 6, fewer number largely destroys the performance to all three methods. However, our framework can still reach higher accuracy no matter how many data are sampled and drops slowly when the sampling number decreases. The reason may be that we incorporate more useful features when implementing *short wealth tuning*, which contributes to the accuracy improvement.

4.3.4 Distribution of Pruned Filters. The number of pruned filters is recorded according to the different CONV layers of VGG-16. Note that we do not implement pruning to the first and the last CONV layers. The pruning input is the Caltech101 dataset. We show the statistics in Figure 7.

We can observe that: (1) From the CONV1_2 to CONV3_3, the number of pruned filters is relatively small with all three methods. More pruned filters lie in the several last CONV layers. That may be because some low-level filters hold some general image features, which should be reserved due to their generalization to any image recognition tasks. So low-level filters are pruned fewer while high-level filters holding some features towards specific tasks are pruned more. (2) Our framework prunes more high-level filters compared with the other methods. The reasons may be two-fold. Firstly, our task is an extremely small task, which only needs a few high-level features. Secondly, too many high-level features will cause feature overlapping, resulting in bad performance for our target task.

5 CONCLUDING REMARKS

We have presented **WealthAdapt**, a framework that incorporates the wealth of big data into the network adaptation pipeline to improve the performance for small data tasks. Given a small data task and the wealth of big data, WealthAdapt can automatically utilize the suitable features in the big data to assist the network adaptation process, generating a better pruned network compared with current pipelines. Experimental results demonstrate that WealthAdapt can be applied to both the vanilla convolutional network VGG-16 and more complex modern architecture ResNet-50, achieving consistently the best performance on multiple visual classification tasks. Moreover, we conduct detailed analyses for an in-depth evaluation of our framework, which further demonstrates the effectiveness and applicability of WealthAdapt.

There are also a number of issues worth exploring further with respect to network adaptation for resource-constrained systems. For example, quantizing the weights and activations to low precision is of vital importance to a specific system. Not only can it shrink the size of the network and save energy consumption, but also it can make the network easier to deploy for a custom low-bits hardware. In the future, we will introduce the quantization technique into our framework for better network adaptation.

ACKNOWLEDGMENTS

This work was partly supported by the National Key Research and Development Program (2017YFB1001904) and the National Natural Science Foundation of China (61772042). Yao Guo is the corresponding author.

RIGHTSLINK

REFERENCES

[1] Akm Ashiquzzaman, Linh Van Ma, SangWoo Kim, Dongsu Lee, Tai-Won Um, and Jinsul Kim. 2019. Compacting Deep Neural Networks for Light Weight IoT & SCADA Based Applications with Node Pruning. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 082–085.

[2] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. 2016. Factors of transferability for a generic convnet representation. *IEEE transactions on pattern analysis and machine intelligence* 38, 9 (2016), 1790–1802.

[3] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Michael I Jordan. 2018. Partial transfer learning with selective adversarial networks. *CVPR* (2018).

[4] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. 2018. Large Scale Fine-Grained Categorization and Domain-Specific Transfer Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4109–4118.

[5] Li Fei-Fei, Rob Fergus, and Pietro Perona. 2007. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer vision and Image understanding* 106, 1 (2007), 59–70.

[6] Weifeng Ge and Yizhou Yu. 2017. Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI*, Vol. 6.

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 580–587.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[9] Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR* (2016).

[10] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[12] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. 2018. Soft filter pruning for accelerating deep convolutional neural networks. *IJCAI* (2018).

[13] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, Vol. 2.

[14] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[15] Soroosh Khoram and Jing Li. 2018. Adaptive quantization of neural networks. *ICLR* (2018).

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[17] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. 2015. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *ICLR* (2015).

[18] Vadim Lebedev and Victor Lempitsky. 2016. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2554–2564.

[19] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient convnets. *ICLR* (2017).

[20] Shaohui Lin, Rongrong Ji, Chao Chen, Dacheng Tao, and Jiebo Luo. 2018. Holistic cnn compression via low-rank decomposition with knowledge transfer. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[21] Shaohui Lin, Rongrong Ji, Xiaowei Guo, Xuelong Li, et al. 2016. Towards Convolutional Neural Networks Compression via Global Error Reconstruction.. In *IJCAI*. 1753–1759.

[22] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.

[23] Jiaming Liu, Yali Wang, and Yu Qiao. 2017. Sparse Deep Transfer Learning for Convolutional Neural Network.. In *AAAI*. 2245–2251.

[24] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. Thinet: A filter level pruning method for deep neural network compression. *ICCV* (2017).

[25] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning convolutional neural networks for resource efficient inference. *ICLR* (2017).

[26] Sinno Jialin Pan, Qiang Yang, et al. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[27] Ariadna Quattoni and Antonio Torralba. 2009. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 413–420.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.

[29] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. 2014. CNN features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 806–813.

[30] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. *ICLR* (2015).

[31] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. 2015. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*. 4068–4076.

[32] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. 2017. Adversarial discriminative domain adaptation. In *Computer Vision and Pattern Recognition (CVPR)*, Vol. 1. 4.

[33] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. 2011. The caltech-ucsd birds-200-2011 dataset. (2011).

[34] Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. Coordinating filters for faster deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 658–666.

[35] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 285–300.

[36] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in neural information processing systems*. 3320–3328.

[37] Jing Zhang, Zewei Ding, Wanqing Li, and Philip Ogunbona. 2018. Importance Weighted Adversarial Nets for Partial Domain Adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8156–8164.

[38] Yang Zhong, Vladimir Li, Ryuzo Okada, and Atsuto Maki. 2018. Target Aware Network Adaptation for Efficient Representation Learning. *ECCV workshop* (2018).

RIGHTSLINK