International Journal of Machine Learning and Cybernetics
https://doi.org/10.1007/s13042-019-01004-6

ORIGINAL ARTICLE

Transfer channel pruning for compressing deep domain adaptation models

Chaohui Yu[1,2] · Jindong Wang[3] · Yiqiang Chen[1,2] · Xin Qin[1,2]

Received: 29 March 2019 / Accepted: 19 August 2019

Abstract
Deep unsupervised domain adaptation has recently received increasing attention from researchers. However, existing methods are computationally intensive due to the computational cost of convolutional neural networks (CNN) adopted by most work. There is no effective network compression method for such problem. In this paper, we propose a unified transfer channel pruning (TCP) method for accelerating deep unsupervised domain adaptation (UDA) models. TCP method is capable of compressing the deep UDA model by pruning less important channels while simultaneously learning transferable features by reducing the cross-domain distribution divergence. Therefore, it reduces the impact of negative transfer and maintains competitive performance on the target task. To the best of our knowledge, TCP method is the *first* approach that aims at accelerating deep unsupervised domain adaptation models. TCP method is validated on two main kinds of UDA methods: the discrepancy-based methods and the adversarial-based methods. In addition, it is validated on two benchmark datasets: Office-31 and ImageCLEF-DA with two common backbone networks - VGG16 and ResNet50. Experimental results demonstrate that our TCP method achieves comparable or better classification accuracy than other comparison methods while significantly reducing the computational cost. To be more specific, in VGG16, we get even higher accuracy after pruning 26% floating point operations (FLOPs); in ResNet50, we also get higher accuracy on half of the tasks after pruning 12% FLOPs for both discrepancy-based methods and adversarial-based methods.

Keywords Unsupervised domain adaptation · Transfer channel pruning · Accelerating

1 Introduction

Deep neural networks have significantly improved the performance of diverse machine learning applications. However, in order to avoid overfitting and achieve better performance, a large amount of labeled data is needed to train a deep network. Since the manual labeling of massive training data is usually expensive in terms of money and time, it is urgent to develop effective algorithms to reduce the labeling workload on the domain to be learned (i.e. *target* domain).

A popular solution to solve the above problem is called *transfer learning*, or *domain adaptation* [32, 45, 46], which tries to transfer knowledge from well-labeled domains (i.e. *source* domains) to the target domain. Specifically, unsupervised domain adaptation (UDA) is considered more challenging since the target domain has no labels. The key is to learn a discriminative model to reduce the distribution divergence between domains.

Traditional methods perform adaptation by reweighting samples from the source domain [2, 19], or seeking an explicit feature space transformation that transforms the source and target samples into the same feature space [31, 46]. Recent studies have indicated that deep networks can learn more transferable features for domain adaptation [5, 48]. The latest advances have been achieved by embedding domain adaptation modules in the pipeline of deep feature learning to extract domain-invariant representations [8, 23, 25, 35, 41]. This is because that they take advantages of CNN (convolutional neural networks) to learn more transferable representations [23] compared to traditional methods.

✉ Yiqiang Chen
yqchen@ict.ac.cn

[1] Beijing Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

[2] University of Chinese Academy of Sciences, Beijing, China

[3] Microsoft Research Asia, Beijing, China

Published online: 11 September 2019

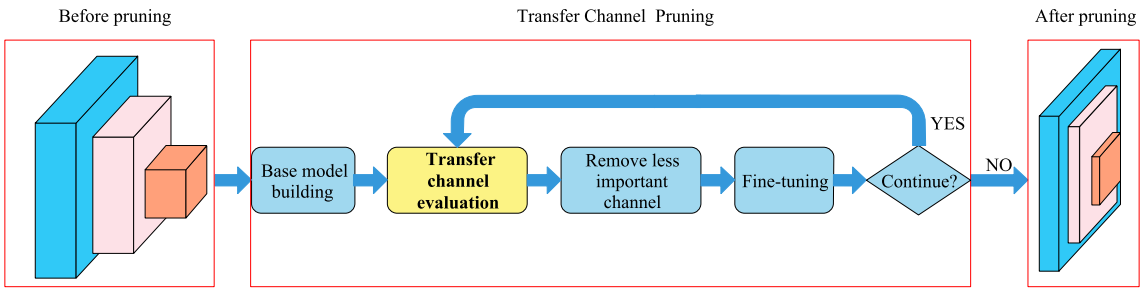


Fig. 1 The framework of the proposed transfer channel pruning (TCP) method

Popular CNN architectures such as AlexNet [21], VGGNet [38], and ResNet [14] are widely adopted as the backbone networks for deep unsupervised domain adaptation methods. Then, knowledge can be transferred to the target domain by reducing the cross-domain distance such as maximum mean discrepancy (MMD) [31] or KL divergence [31].

Recently, adversarial learning [11] has been successfully embedded into deep networks to reduce distribution discrepancy between the source and target domains. Prior advanced adversarial adaptation methods [8, 29, 35, 43] have shown promising results in several domain adaptation tasks. Existing adversarial domain adaptation methods either learn a single domain discriminator to align the global source and target distributions, or pay attention to align subdomains based on multiple discriminators. For instance, domain-adversarial Neural network (DANN) [8, 9] focuses on the global adversarial learning, while multi-adversarial domain adaptation (MADA) [35] pays attention to the subdomain adaptation by training several domain classifiers.

Unfortunately, it is still challenging to deploy these deep UDA models on resource constrained devices such as mobile phones since there is a huge computational cost required by these methods. In order to reduce resource requirement and accelerate the inference process, a common solution is *network compression*. Network compression methods mainly include network quantization [36, 49], weight pruning [13, 15, 28], and low-rank approximation [4, 12]. Especially channel pruning [15, 28], which is a type of weight pruning and compared to other methods, it does not need special hardware or software implementations. In addition, it can reduce negative transfer by pruning some redundant channels, therefore it is a good choice for compressing deep UDA models.

However, it is not feasible to apply the above network compression methods directly to the UDA problems. The reasons are two-fold. Firstly, these compression methods are proposed to solve *supervised* learning problems, which is not suitable for the UDA settings since there are no labels in the target domain. Secondly, even if we can acquire some labels manually, applying these compression methods directly to UDA will result in *negative transfer* [32], since they fail to consider the distribution discrepancy between the source and target domains. Currently, there is no effective network compression method for UDA.

In this paper, we propose a unified network compression method called transfer channel pruning (TCP) for accelerating deep unsupervised domain adaptation models. The general framework of our TCP method is shown in Fig. 1. Starting from a deep unsupervised domain adaptation base model, TCP method iteratively evaluates the importance of channels with the transfer channel evaluation module and remove less important channels for both source and target domains. TCP method is capable of compressing the deep UDA model by pruning less important channels while simultaneously learning transferable features by reducing the cross-domain distribution divergence. Experimental results demonstrate that TCP method achieves better classification accuracy than other comparison pruning methods while significantly reducing the computational cost. To the best of our knowledge, our TCP method is the *first* approach to accelerate the deep UDA models.

To summarize, the contributions of this paper are as follows:

(1) We present our TCP method as a unified approach for accelerating deep unsupervised domain adaptation models. TCP method is a generic, accurate, and efficient compression method that can be easily implemented by most deep learning libraries.
(2) TCP method is able to reduce negative transfer by considering the cross-domain distribution discrepancy using the proposed transfer channel evaluation module.
(3) Extensive experiments on two public UDA datasets demonstrate the significant superiority of our TCP method.

2 Related work

Our work is mainly related to unsupervised domain adaptation and network compression.

2.1 Unsupervised domain adaptation

Unsupervised domain adaptation (UDA) is a specific area of transfer learning [32, 45], which is to learn a discriminative model in the presence of the domain-shifts between domains. The main problem of UDA is how to reduce the domain shift between the source and target domains. There are many methods to tackle this problem: traditional (shallow) learning and deep learning.

Traditional (shallow) learning methods have several aspects: (1) Subspace learning. Subspace alignment (SA) [7] aligns the base vectors of both domains and subspace distribution alignment (SDA) [40] extends SA by adding the subspace variance adaptation. Gong et al. proposed the Geodesic Flow Kernel (GFK) [10] to sample indefinite points along the geodesic flow between domains. CORAL [39] aligns subspaces in second-order statistics. (2) Distribution alignment. Pan et al. proposed the transfer component analysis (TCA) method to align the marginal distributions between domains. Based on TCA, joint distribution adaptation (JDA) [24] is proposed to match both marginal and conditional distributions. Later works extend JDA by adding regularization, structural consistency [17] and domain invariant clustering [42]. But these works treat the two distributions equally and fail to leverage the different importance of distributions. Recently, Wang et al. proposed the manifold embedded distribution alignment (MEDA) [45, 46] approach to dynamically evaluate the different effect of marginal and conditional distributions and achieved the state-of-the-art results on domain adaptation.

As for deep learning methods, CNN can learn nonlinear deep representations and capture underlying factors of variation between different tasks [1]. These deep representations can disentangle the factors of variation, which enables the transfer of knowledge between tasks.

Recent works on deep UDA approaches can be mainly summarized into two cases [34]. The first case is the discrepancy-based deep UDA approach, which assumes that fine-tuning the deep network model with labeled or unlabeled target data can diminish the shift between the two domains. The most commonly used methods for aligning the distribution shift between the source and target domains are maximum mean discrepancy (MMD), correlation alignment (CORAL) [41], Kullback-Leibler (KL) divergence [50], among others. The second case can be referred to as the adversarial-based approach. In this case, a domain discriminator that classifies whether a data point is drawn from the source or target domain is used to encourage domain confusion through an adversarial objective to minimize the distance between the source and target distributions [8]. Adversarial learning has been explored in generative adversarial networks (GANs) [11]. In addition, generative multi adversarial network (GMAN) [6] extends GANs to multiple discriminators including formidable adversary and forgiving teacher, which significantly eases model training.

Recent works on deep UDA embed domain-adaptation modules into deep networks to improve transfer performance [9], where significant performance gains have been obtained. UDA has wide applications in computer vision [16, 23] and natural language processing and is receiving increasing attention from researchers. In this paper, we concentrate on accelerating both the discrepancy-based and adversarial-based deep UDA approaches.

As far as we know, no previous UDA approach has focused on the acceleration of the network.

2.2 Network compression

These years, for better accuracy, designing deeper and wider CNN models has become a general trend, such as VGGNet [38] and ResNet [14]. However, as the CNN grow bigger, it is harder to deploy these deep models on resource constrained devices. Network compression becomes an efficient way to solve this problem. Network compression methods mainly include network quantization, low-rank approximation and weight pruning. Network quantization is good at decreasing the presentation precision of parameters so as to reduce the storage space. Low-rank approximation reduces the storage space by low-rank matrix techniques, which is not efficient for point-wise convolution [3]. Weight pruning mainly includes two methods, neural pruning [13, 22] and channel pruning [15, 27, 28].

Channel pruning methods prune the whole channel each time, therefore it is fast and efficient than neural pruning which removes a single neuron connection each time. It is a structured pruning method, compared to network quantization and low-rank approximation, it does not introduce sparsity to the original network structure and also does not require special software or hardware implementations. It has demonstrated superior performance compared to other methods and many works [15, 27, 28] have been proposed to perform channel pruning on pre-trained models with different kinds of criteria.

These above pruning methods mainly aim at supervised learning problems, by contrast, there have been few studies for compressing unsupervised domain adaptation models. As far as we know, we are the first to study how to do channel pruning for deep unsupervised domain adaptation.

The TCP method is primarily motivated by [28], while our work is different from it. TCP method is presented for pruning unsupervised domain adaptation models. To be more specific, we take the discrepancy between the source and target domains into consideration, therefore we can prune the less important channels not just for the source domain but also for the unlabeled target domain. We call

this Transfer Channel Evaluation, which is highlighted in yellow in Fig. 1.

3 Transfer channel pruning

In this section, we introduce the proposed Transfer Channel Pruning (TCP) method.

3.1 Problem definition

Definition 1 *In unsupervised domain adaptation, we are given a source domain* $\mathcal{D}_s = \{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^{n_s}$ *of* n_s *labeled examples and a target domain* $\mathcal{D}_t = \{\mathbf{x}_j^t\}_{j=1}^{n_t}$ *of* n_t *unlabeled examples.* $\mathcal{D}_s$ *and* $\mathcal{D}_t$ *have the same label space, i.e. The marginal distributions between two domains are different, i.e.* $P_s(\mathbf{x}_s) \neq P_t(\mathbf{x}_t)$. *The goal of deep UDA is to design a deep neural network that enables learning of transfer classifiers* $y = f_s(\mathbf{x})$ *and* $y = f_t(\mathbf{x})$ *to close the source-target discrepancy and can achieve the best performance on the target dataset.*

For a pre-trained deep UDA model, its parameters can be denoted as $\mathbf{W}$. Here we assume the l_{th} convolutional layer has an output activation tensor $\mathbf{a}_l$ of size of $h_l \times w_l \times k_l$, where k_l represents the number of output channels of the l_{th} layer, and h_l and w_l stand for the height and width of feature maps of the l_{th} layer, respectively. Therefore, the goal of TCP method is to prune a UDA model in order to accelerate it with comparable or even better performance on the target domain. In this way, we can obtain smaller models that require less computation complexity and memory consumption, which can be deployed on resource constrained devices.

3.2 Motivation

We compress the deep UDA model using model pruning methods for their efficiency. A straightforward model pruning technique is a *two-stage* method, which first prunes the model on the source domain with supervised learning and then fine-tune the model on the target domain.

It is worth noting that there is not prior work to compress UDA models, as far as we know, we are the first to study how to do channel pruning for UDA models and the splitting two-stage method is probably the easiest method to think of and be applied to compress UDA models. Because existing channel pruning methods are usually for compressing supervised learning models, thus it is natural to think that channel pruning method can be applied to the source domain first with supervised learning and then fine-tune for the unlabelled target domain with the pruned model by UDA methods. However, negative transfer [32] is likely to happen during this pruning process since the discrepancy between the source and target domains is ignored.

In this work, we propose a unified transfer channel pruning (TCP) method to tackle such challenge. TCP method is capable of compressing the deep UDA model by pruning less important channels while simultaneously learning transferable features by reducing the cross-domain distribution divergence. Therefore, TCP method reduces the impact of negative transfer and maintains competitive performance on the target task. In short, TCP method is a generic, accurate, and efficient compression method that can be easily implemented by most deep learning libraries.

To be more specific, Fig. 1 illustrates the main idea of our TCP method. There are mainly three steps. Firstly, TCP method learns the base deep UDA model through base model building. The base model is fine-tuned with the standard UDA criteria. Secondly, TCP method evaluates the importance of channels of all layers with the transfer channel evaluation and performs further fine-tuning. Specifically, the convolutional layers, which usually dominate the computation complexity, are pruned in this step. Thirdly, TCP method iteratively refines the pruning results and stops after reaching the trade-off between accuracy and FLOPs (i.e. computational cost) or parameter size.

3.3 Base model building

In this step, we build the base UDA models with deep neural networks.

3.3.1 Deep adaptation networks

Deep neural networks have been successfully used in UDA with state-of-the-art algorithms [9, 23, 43] in recent years. Previous studies [43, 48] have shown that the features extracted by deep networks are general at lower layers, while specific at the higher layers since they are more task-specific. Therefore, more transferable representations can be learned by transferring the features at lower layers and then fine-tune the task-specific layers. During fine-tuning, the cross-domain discrepancy can be reduced by certain adaptation distance. Since our main contribution is not designing new deep UDA networks, we build the base model like most discrepancy-based deep UDA approaches [23, 41, 44]. In the following, we will briefly introduce the main idea of the base model, and more details can be found in the original paper of these discrepancy-based deep UDA approaches.

As shown in Fig. 2, we learn transferable features via several convolutional and pooling layers (the blue and purple blocks). Then, the classification task can be accomplished with the fully-connected layers (the yellow blocks). MMD (maximum mean discrepancy) [31] is adopted as the adaptation loss in order to reduce domain shift. MMD has

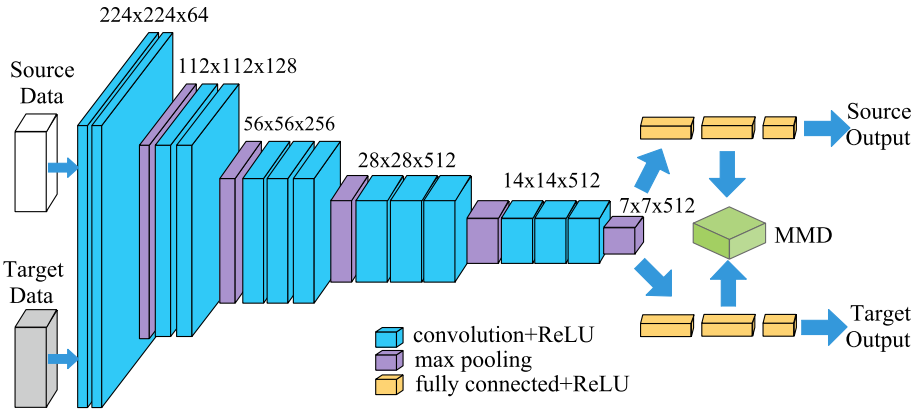


Fig. 2 The basic architecture of deep adaptation networks

been proposed to provide the distribution difference between the source and target datasets. In addition, it has been widely utilized in many UDA methods [23, 25, 30]. The MMD loss between two domains can be computed as

$$L_{mmd} = \left\| \frac{1}{n_s} \sum_{\mathbf{x}_i \in \mathcal{D}_s} \phi(\mathbf{x}_i) - \frac{1}{n_t} \sum_{\mathbf{x}_j \in \mathcal{D}_t} \phi(\mathbf{x}_j) \right\|_{\mathcal{H}}^2, \tag{1}$$

where $\mathcal{H}$ denotes RKHS (Reproducing Kernel Hilbert Space) and $\phi(\cdot)$ denotes some feature map to map the original samples to RKHS.

3.3.2 Domain-adversarial neural networks

In this case, a domain discriminator that classifies whether a data point is drawn from the source or target domain is used to encourage domain confusion through an adversarial objective to minimize the distance between the source and target distributions [8]. Domain adversarial adaptation methods borrow the idea of GAN to help learn transferable features.

As shown in Fig. 3, we learn transferable features via domain adversarial learning. Our basic domain-adversarial neural network consists of three parts: the feature extractor G_f (the blue blocks), the label classifier G_y (the green blocks) and the domain discriminator G_d (the orange blocks). The adversarial learning procedure is a two-player game, where the first player is the domain discriminator G_d trained to distinguish the source domain from the target domain, and the second player is the feature extractor G_f that tries to confuse the domain discriminator by extracting domain-invariant features. The two players are trained adversarially: the parameters θ_f of feature extractor G_f are learned by maximizing the loss of domain discriminator G_d, while the parameters θ_d of G_d are trained by minimizing the loss of the domain discriminator. In addition, the loss of the label classifier G_y is also minimized. The loss function can be formalized as:

$$\begin{aligned} L(\theta_f, \theta_y, \theta_d) = & \frac{1}{n_s} \sum_{\mathbf{x}_i \in \mathcal{D}_s} L_y\big(G_y\big(G_f(\mathbf{x}_i)\big), y_i\big) \\ & - \frac{\lambda}{n_s + n_t} \sum_{\mathbf{x}_i \in (\mathcal{D}_s \cup \mathcal{D}_t)} L_d\big(G_d\big(G_f(\mathbf{x}_i)\big), d_i\big), \end{aligned} \tag{2}$$

where λ is a trade-off parameter. G_y and L_y are the label classifier and its loss, G_d and L_d are the domain classifier and its loss, and d_i is the domain label of the input sample $\mathbf{x}_i$.

Several popular architectures can serve as the backbone network of the base model, such as AlexNet [21], VGGNet [38], and ResNet [14]. After obtaining the base model, we can perform channel pruning to accelerate the model.

3.4 Transfer channel evaluation

The goal of transfer channel evaluation is to iteratively evaluate the importance of output channels of layers in order to prune the $\mathcal{K}$ least important channels. Here $\mathcal{K}$ is controlled by users. In the pruning process, we want to preserve and refine a set of parameters $\mathbf{W}'$, which represents those important parameters for both source and target domains. Let $L(\mathcal{D}_s, \mathcal{D}_t, \mathbf{W})$ be the cost function for UDA and $\mathbf{W}' = \mathbf{W}$ at the starting time. For a better set of parameters $\mathbf{W}'$, we want to minimize the loss change after pruning a channel $\mathbf{a}_{l,i}$. This can be considered as an optimization problem. Here we introduce the absolute difference of loss:

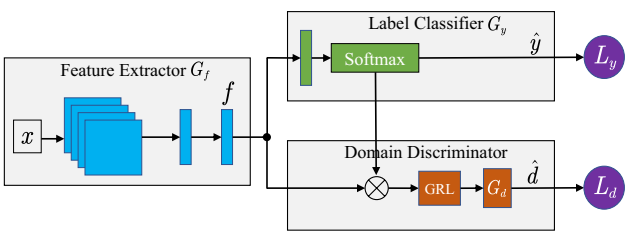


Fig. 3 The basic architecture of deep domain-adversarial neural networks

$$|\Delta L(\mathbf{a}_{l,i})| = |L(\mathcal{D}_s, \mathcal{D}_t, \mathbf{a}_{l,i}) - L(\mathcal{D}_s, \mathcal{D}_t, \mathbf{a}_{l,i} = 0)|, \tag{3}$$

which means the loss change after pruning the i_{th} channel of the l_{th} convolutional layer. In addition, we want to minimize $|\Delta L(\mathbf{a}_{l,i})|$ by selecting the appropriate channel $\mathbf{a}_{l,i}$. Pruning will stop until a trade-off between accuracy and pruning object (FLOPs or parameter size) has been achieved.

However, it is hard to find a set of optimal parameters in one go, because the search space is $2^{|\mathbf{W}|}$ which is too huge to compute and try every combination. Inspired by [28], our TCP method solves this problem with a greedy algorithm by iteratively removing the $\mathcal{K}$ least important channels at each time.

3.4.1 Criteria

Criteria is the criterion for judging the importance of channels. Since the key to channel pruning is to select the least important channel, especially for UDA, we design the criteria of TCP method carefully. There are many heuristic criteria, including the L_2-norm of filter weights, the activation statistics of feature maps, mutual information between activations and predictions and Taylor expansion, etc. Here we choose the *first-order Taylor expansion* as the base criteria since its efficiency and performance has been verified in [28] for pruning supervised learning models. Compared with our TCP method, we also take pruning as an optimization problem, however, the objective we want to optimize is the final performance on the unlabeled target dataset. Therefore we design our criteria in a different way which is better for pruning deep UDA models.

According to Taylor's theorem, the Taylor expansion at point $x = a$ can be computed as:

$$f(x) = \sum_{p=0}^{P} \frac{f^{(p)}(a)}{p!}(x-a)^p + R_p(x), \tag{4}$$

where p denotes the p_{th} derivative of $f(x)$ at point $x = a$ and the last item $R_p(x)$ represents the p_{th} remainder. To approximate $|\Delta L(\mathbf{a}_{l,i})|$, we can use the first-order Taylor expansion near $\mathbf{a}_{l,i} = 0$ which means the loss change after removing $\mathbf{a}_{l,i}$, then we can get:

$$f(\mathbf{a}_{l,i} = 0) = f(\mathbf{a}_{l,i}) - f'(\mathbf{a}_{l,i}) \cdot \mathbf{a}_{l,i} + \frac{|\mathbf{a}_{l,i}|^2}{2} \cdot f''(\xi), \tag{5}$$

where ξ is a value between 0 and $\mathbf{a}_{l,i}$, and $\frac{|\mathbf{a}_{l,i}|^2}{2} \cdot f''(\xi)$ is a Lagrange form remainder which requires too much computation, therefore we abandon this item for accelerating the pruning process. Then back to Eq. (3), we can get:

$$L(\mathcal{D}_s, \mathcal{D}_t, \mathbf{a}_{l,i} = 0) = L(\mathcal{D}_s, \mathcal{D}_t, \mathbf{a}_{l,i}) - \frac{\partial L}{\partial \mathbf{a}_{l,i}} \cdot \mathbf{a}_{l,i}. \tag{6}$$

Then, we combine Eqs. (3) and (6) and get the criteria G of TCP method:

$$G(\mathbf{a}_{l,i}) = |\Delta L(\mathbf{a}_{l,i})| = \left| \frac{\partial L}{\partial \mathbf{a}_{l,i}} \cdot \mathbf{a}_{l,i} \right|, \tag{7}$$

which means the absolute value of product of the activation and the gradient of the cost function, and $\mathbf{a}_{l,i}$ can be calculated as:

$$\mathbf{a}_{l,i} = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h_l \times w_l} \sum_{p=1}^{h_l} \sum_{q=1}^{w_l} \mathbf{a}_{l,i}^{p,q}, \tag{8}$$

where N is the number of batch size.

3.4.2 Loss function of TCP method

To make our TCP method focus on pruning UDA models, we simultaneously take the source domain and the unlabeled target domain into consideration. The loss function of TCP method consists of two parts, $L_{cls}(\mathcal{D}_s, \mathbf{W})$ and $L_{mmd}(\mathcal{D}_s, \mathcal{D}_t, \mathbf{W})$. Here, $L_{cls}(\mathcal{D}_s, \mathbf{W})$ is a cross-entropy loss which denotes the classification loss on source domain and can be computed as:

$$L_{cls}(\mathcal{D}_s, \mathbf{W}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} P_{i,c} log(h_c(x_i^s)) \tag{9}$$

where C is the number of classes of source dataset, $P_{i,c}$ is the probability of x_i^s belonging to class c, and $h_c(x_i^s)$ denotes the probability that the model predicts x_i^s as class c. In addition, $L_{mmd}(\mathcal{D}_s, \mathcal{D}_t, \mathbf{W})$ denotes the MMD loss between the source and target domains that presented in Eq. (1). The total loss function can be computed as:

$$L(\mathcal{D}_s, \mathcal{D}_t, \mathbf{W}) = L_{cls}(\mathcal{D}_s, \mathbf{W}) + \beta L_{mmd}(\mathcal{D}_s, \mathcal{D}_t, \mathbf{W}), \tag{10}$$

where

$$\beta = \frac{4}{1 + e^{-1 \cdot \frac{i}{ITER}}} - 2. \tag{11}$$

Here, β is a dynamic value which takes values in (0, 1). $i \in (0, ITER)$ where $ITER$ is the number of pruning iterations. We design β in this way for two main reasons, on the one hand, during the early stage of pruning, the weights have not converged and keep unstable therefore the L_{mmd} is too large and makes the pruned model hard to converge. On the other hand, in the rest of the pruning process, the L_{mmd} becomes more important that can guide the pruned model to focus more on the target domain. Therefore the criteria of TCP method can be computed as:

$$G(\mathbf{a}_{l,i}) = \left| \frac{\partial L_{cls}(\mathcal{D}_s, \mathbf{W})}{\partial \mathbf{a}^s_{l,i}} \cdot \mathbf{a}^s_{l,i} + \beta \frac{\partial L_{mmd}(\mathcal{D}_s, \mathcal{D}_t, \mathbf{W})}{\partial \mathbf{a}^t_{l,i}} \cdot \mathbf{a}^t_{l,i} \right|, \tag{12}$$

where $\mathbf{a}^s_{l,i}$ and $\mathbf{a}^t_{l,i}$ denote the activation with source data and target data respectively.

3.5 Iterative refinement

After the transfer channel evaluation, each channel is sorted according to $G(\mathbf{a}_{l,i})$ and the $\mathcal{K}$ least important channels are removed after each pruning iteration. Then, a short-term fine-tuning is adopted to the pruned model for 5 epochs to help the model to converge and the pruning is done after a trade-off between accuracy and FLOPs or parameter size has been achieved. This step is done *iteratively* to prune the network.

The learning procedure of TCP method is described in Algorithm 1. Specifically, for a given UDA baseline $\mathbf{W}$, firstly fine-tune the UDA baseline until the best performance achieved on the target dataset. Then, sort the channels according to their importance which is measured by criteria Eq. (12) and remove the $\mathcal{K}$ least important channels. Next, perform short-term fine-tuning to the pruned model for 5 epochs to help the model to converge. These steps are done iteratively to prune the network. Finally, a long-term fine-tuning is adopted to increase the performance of the final pruned model $\mathbf{W}'$.

4 Experimental analysis

In this section, we evaluate the performance of TCP method via experiments on pruning deep unsupervised domain adaptation models. We evaluate our approaches for VGGNet [38] and ResNet [14] on two popular datasets—Office-31 [37] and ImageCLEF-DA.[1] All our methods are implemented based on the PyTorch [33] framework and the code will be released soon at [47].

4.1 Datasets

4.1.1 Office-31

This dataset is a standard and maybe the most popular benchmark for unsupervised domain adaptation. It consists of 4110 images within 31 categories collected from everyday objects in an office environment. It consists of three domains: *Amazon* (**A**), which contains images downloaded from https://www.amazon.com/, *Webcam* (**W**) and *DSLR* (**D**), which contain images respectively taken by web camera and digital SLR camera under different settings. We evaluate all our methods across six transfer tasks on all the three domains **A**→ **W**, **W**→ **A**, **A**→ **D**, **D**→ **A**, **D**→ **W** and **W**→ **D**.

4.1.2 ImageCLEF-DA

This dataset is a benchmark dataset for ImageCLEF 2014 domain adaptation challenge, and it is collected by selecting the 12 common categories shared by the following

Algorithm 1 TCP: Transfer Channel Pruning

Input: Source domain $\mathcal{D}_s = \{(\mathbf{x}^s_i, y^s_i)\}^{n_s}_{i=1}$, target domain $\mathcal{D}_t = \{\mathbf{x}^t_j\}^{n_t}_{j=1}$, the baseline $\mathbf{W}$.

Output: A pruned model $\mathbf{W}'$ for deep unsupervised domain adaptation.

```
Fine-tune the unsupervised domain adaptation baseline until the best performance achieved on the unlabeled target dataset;
for iteration i do
    Sort the importance of channels by criteria Eq. (12) and identify less significant channels;
    Remove the K least important channels of the layers;
    Short-term fine-tune;
    if the trade-off between accuracy and FLOPs or parameter size has achieved then
        break
    end if
end for
Long-term fine-tune;
return pruned model W'.
```

[1] http://imageclef.org/2014/adaptation

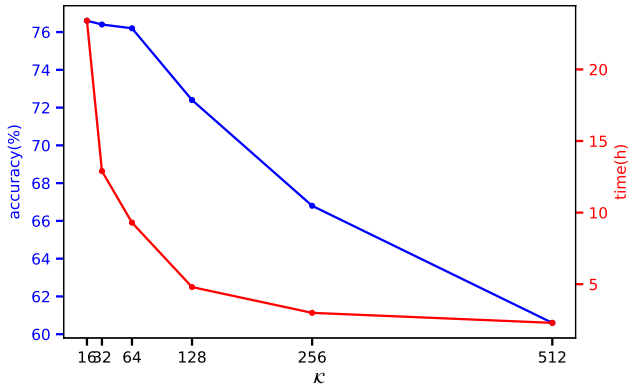


Fig. 4 The sensitivity analysis of the effect of $\mathcal{K}$

public datasets and each of them is considered as a domain: *Caltech* − 256 (**C**), *ImageNet ILSVRC* 2012 (**I**), *Pascal VOC* 2012 (**P**) and *Bing* (**B**). There are 50 images in each category and 600 images in each domain. We evaluate all methods across six transfer tasks following existing work [8, 23, 46]: **I**→ **P**, **P**→ **I**, **I**→ **C**, **C**→ **I**, **P**→ **C** and **C**→ **P**. Compared with Office-31, this dataset is more balanced and can be a good comparable dataset to Office-31.

4.2 Implementation details

We mainly compare three methods: (1) Two_stage: which is the most straightforward method that applies channel pruning to the source domain task first, then fine-tune for the target domain task with the pruned model. (2) TCP_w/o_DA: Our TCP method without the MMD loss, here we call it domain adaptation (DA) loss. Which also means $\beta = 0$ all the time in Eq. (10). (3) TCP: Our full TCP method with DA loss.

We evaluate all the methods on two popular backbone networks: VGG16 [38] and ResNet50 [14]. As baselines, VGG16-based and ResNet50-based are the original models that are not pruned. As for VGG16-based model, it has 13 convolutional layers and 3 fully-connected layers. We prune all the convolutional layers and the first fully-connected layer and we only use the activations of the second fully-connected layer as image representation and build the MMD loss which is shown in Fig. 2. In addition, as for ResNet50-based model, we use similar settings as VGG16-based model with a few differences. Because of the shortcut and residual branch structure, we only prune the inside convolutional layers of each bottleneck block. The MMD loss is built with the only fully-connected layer. Moreover, we also take the Batch Normalization (BN) [20] layers into consideration and reconstruct the whole model during pruning.

In practice, all the input images are cropped to a fixed size 224×224 and randomly sampled from the resized image with horizontal flip and mean-std normalization. At first, we fine-tune all the UDA models on each unsupervised domain adaptation tasks for 200 epochs with learning rate from 0.01 to 0.0001 and the batch size = 32. During pruning, we set $\mathcal{K} = 64$ which means 64 channels will be removed after each pruning iteration. Here we set $\mathcal{K} = 64$ because we want to speed up the pruning process while maintaining high accuracy. In fact, this is a tradeoff between accuracy and pruning time. We analyze the effect of $\mathcal{K}$ value on the accuracy and pruning time of the model and the result is shown in Fig. 4.

As can be seen, if we set $\mathcal{K}$ to a smaller value, we may achieve better performance on the target domain, but we also need more time to do the pruning and fine-tuning procedure to get the same FLOPs reduction. In addition, if we set $\mathcal{K}$ to a larger value, the pruning process can be accelerated while the accuracy on the target domain can not be guaranteed.

After that, extra 5 epochs are adopted to help the pruned model to converge. In addition, we follow [18, 26] to prune the baseline with different compression rate. The VGG16-based baseline is pruned with 26% and 70% FLOPs reduced while the ResNet50-based baseline is pruned with 12% and 46% FLOPs reduced. ResNet50 has lower compression rate since the bottleneck structure stops some layers from being pruned. As for the VGG16-based and ResNet50-based baselines, they are the original well fine-tuned UDA models without pruning. To be more specific, as for the VGG16-based baseline, we build the UDA model as Fig. 2. The backbone netowrk is VGG16 and MMD loss is built on the second fully-connected layer to compare and align the distribution between source and target domains. Then we fine-tune the UDA model for 200 epochs with learning rate from 0.01 to 0.0001 and batch size = 32. In addition, as for the ResNet50-based baseline, the structure is similar to the VGG16-based baseline, we use ResNet50 as the backbone network and MMD loss is built on the only fully-connected layer.

We follow standard evaluation protocol for UDA and use all source examples with labels and all target examples without labels [10]. The labels for the target domain are only used for evaluation. We adopt *classification accuracy* on the target domain and *parameter reduction* as the evaluation metrics: higher accuracy and fewer parameters indicate better performance.

4.3 Results and analysis

Discrepancy-based UDA model pruning Firstly, we evaluate our TCP method on the discrepancy-based base model. we first evaluate all the tasks on Office-31 dataset. The results are shown in Table 1. As can be seen, the full TCP method outperforms other methods in accuracy and F1 score under the same compression rate (FLOPs reduction) and can reduce more parameters, and the bold indicates the best results under the same compression rate. It is important and interesting that TCP method achieves even better performance than the baseline model (which is not pruned).

Table 1 The performance on Office-31 dataset (VGG16-based and ResNet50-based)

Models	FLOPs↓	A→W			D→W			W→D			A→D			D→A			W→A			Average		
		Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓
VGG16-based (baseline)	26%	74.0	0.714	–	94.0	0.949	–	97.5	0.976	–	72.3	0.734	–	54.1	0.528	–	55.2	0.541	–	74.5	0.740	–
Two_stage		69.4	0.684	29.4%	94.5	0.949	31.2%	99.0	0.993	30.6%	69.8	0.694	29.3%	42.7	0.392	32.5%	47.2	0.429	28.3%	70.4	0.576	30.2%
TCP_w/o_DA		73.0	0.714	29.7%	95.5	0.964	32.7%	99.3	0.996	29.2%	75.8	0.758	25.8%	45.9	0.453	29.3%	50.4	0.482	29.6%	73.3	0.728	29.4%
TCP	70%	**76.1**	**0.757**	**36.8%**	**96.1**	**0.965**	**36.2%**	**99.8**	**0.999**	**32.6%**	**76.2**	**0.764**	**35.9%**	**47.9**	**0.483**	**37.1%**	**51.2**	**0.512**	**39.8%**	**74.5**	**0.748**	**36.4%**
Two_stage		57.1	0.563	63.3%	88.1	0.869	68.0%	96.3	0.966	64.5%	55.0	0.530	61.0%	31.8	0.304	66.2%	32.7	0.314	63.1%	60.2	0.497	64.3%
TCP_w/o_DA		53.5	0.528	62.5%	89.2	0.894	61.3%	97.9	0.975	57.5%	61.8	0.617	54.8%	35.3	0.341	62.6%	34.5	0.323	58.8%	62.0	0.613	59.6%
TCP		**74.1**	**0.651**	**69.3%**	**89.5**	**0.896**	**69.8%**	**98.8**	**0.986**	**65.2%**	**65.9**	**0.652**	**66.2%**	**35.6**	**0.345**	**68.5%**	**38.5**	**0.365**	**68.2%**	**67.1**	**0.649**	**67.9%**
ResNet50-based (baseline)	12%	80.3	0.790	–	97.1	0.969	–	99.2	0.993	–	78.9	0.801	–	64.3	0.639	–	62.3	0.603	–	80.3	0.667	–
Two_stage		75.8	0.734	32.4%	96.7	0.968	31.4%	99.5	0.997	35.5%	76.0	0.776	30.4%	48.0	0.461	28.3%	50.1	0.482	29.4%	74.4	0.736	31.2%
TCP_w/o_DA		79.8	0.783	33.5%	97.0	0.974	35.5%	**100**	**1.000**	34.5%	77.1	**0.796**	36.2%	47.8	0.471	34.5%	52.6	0.511	33.1%	75.7	0.755	34.5%
TCP		**81.8**	**0.801**	**37.7%**	**98.2**	**0.983**	**36.2%**	99.8	0999	**37.0%**	**77.9**	0.785	**36.9%**	**50.0**	**0.494**	**35.0%**	**55.5**	**0.531**	**36.9%**	**77.2**	**0.766**	**36.7%**
Two_stage	46%	65.5	0.622	56.2%	93.0	0.928	56.3%	98.7	0.991	57.3%	64.9	0.642	56.0%	34.0	0.338	57.2%	38.9	0.373	57.3%	65.8	0.649	56.7%
TCP_w/o_DA		75.1	0.732	56.4%	95.8	0.969	56.4%	99.2	0.992	56.6%	70.8	0.705	55.8%	34.2	0.343	56.4%	41.5	0.376	56.6%	69.4	0.686	56.4%
TCP		**77.4**	**0.754**	**58.4%**	**96.3**	**0.970**	**58.0%**	**100**	**1.000**	**57.1%**	**72.0**	**0.716**	**59.0%**	**36.1**	**0.363**	**57.8%**	**46.3**	**0.443**	**58.5%**	**71.3**	**0.708**	**58.1%**

Here, *FLOPs* ↓ and *Param* ↓ denote the decrement of FLOPs and parameter size compared with the baseline, *F1* and *Acc* are the F1 score and accuracy on the target domain. The base model is a discrepancy-based model shown in Fig. 2

The bold values denote the best performance under the same compression rate of different tasks

Table 2 The performance on ImageCLEF-DA dataset (VGG16-based and ResNet50-based)

Models	FLOPs↓	I→P			P→I			I→C			C→I			C→P			P→C			Average		
		Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓
VGG16-based (baseline)	26%	71.3	0.709		80.0	0.791		88.5	0.879		77.0	0.756		61.1	0.587		87.2	0.875		77.5	0.766	
Two_stage		68.0	0.669	29.0%	77.7	0.765	29.9%	88.5	0.878	27.5%	64.5	0.605	29.0%	56.0	0.536	29.3%	83.3	0.822	26.9%	73.0	0.712	28.6%
TCP_w/o_DA		70.5	0.689	33.6%	79.5	0.789	**34.2%**	89.0	0.885	33.1%	77.1	0.771	34.5%	62.2	0.608	35.0%	85.1	0.842	33.1%	77.2	0.764	33.9%
TCP		**72.0**	**0.720**	**39.0%**	**80.5**	**0.837**	32.2%	**90.5**	**0.893**	**35.1%**	**77.8**	**0.775**	**36.3%**	**64.8**	**0.646**	**36.6%**	**87.5**	**0.893**	**34.5%**	**78.9**	**0.794**	**35.6%**
Two_stage	70%	58.6	0.564	65.8%	69.2	0.679	62.9%	80.6	0.798	64.3%	57.7	0.544	57.0%	43.2	0.401	67.8%	74.5	0.743	61.5%	63.9	0.621	63.2%
TCP_w/o_DA		61.0	0.598	55.4%	69.1	0.673	65.7%	80.5	0.798	**66.5%**	55.1	0.516	63.3%	47.0	0.457	66.5%	70.9	0.705	65.8%	63.9	0.624	63.9%
TCP		**61.9**	**0.605**	**66.7%**	**69.5**	**0.680**	**66.0%**	**81.8**	**0.808**	65.7%	**59.8**	**0.579**	**68.7%**	**49.7**	**0.491**	**68.8%**	**75.9**	**0.753**	**67.2%**	**66.4**	**0.653**	**67.1%**
ResNet50-based (baseline)	12%	74.8	0.749		82.2	0.863		92.3	0.930		83.3	0.825		70.0	0.721		89.8	0.924		82.1	0.835	
Two_stage		71.8	0.706	29.4%	81.3	0.807	31.5%	92.1	0.915	34.4%	76.5	0.729	32.4%	64.0	0.621	29.2%	84.0	0.835	30.8%	78.2	0.769	31.2%
TCP_w/o_DA		73.0	0.715	33.5%	80.5	0.788	34.6%	92.0	**0.922**	33.8%	76.1	0.729	31.2%	64.3	0.621	30.1%	86.3	0.853	36.3%	78.7	0.771	33.2%
TCP		**75.0**	**0.750**	**37.5%**	**82.6**	**0.818**	**36.5%**	**92.5**	0.919	**35.5%**	**80.8**	**0.788**	**36.7%**	**66.2**	**0.640**	**36.6%**	**86.5**	**0.857**	**37.6%**	**80.6**	**0.795**	**36.7%**
Two_stage	46%	65.6	0.638	53.2%	71.8	0.691	56.5%	85.2	0.841	54.2%	68.2	0.645	54.0%	57.4	0.541	51.5%	78.1	0.775	54.0%	71.1	0.688	53.9%
TCP_w/o_DA		66.6	0.651	55.4%	73.0	0.719	57.4%	85.5	0.848	55.4%	67.7	0.640	55.5%	55.5	0.529	53.6%	77.0	0.768	57.1%	70.8	0.692	55.7%
TCP		**67.8**	**0.657**	**57.2%**	**77.5**	**0.765**	**58.0%**	**88.6**	**0.881**	**56.2%**	**71.6**	**0.692**	**58.5%**	**57.7**	**0.555**	**55.7%**	**79.5**	**0.786**	**58.2%**	**73.8**	**0.723**	**57.3%**

The base model is a discrepancy-based model

The bold values denote the best performance under the same compression rate of different tasks

Table 3 The performance on Office-31 dataset (VGG16-based and ResNet50-based)

Models	FLOPs↓	A→W			D→W			W→D			A→D			D→A			W→A			Average		
		Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓
VGG16-based (baseline)	26%	76.8	0.750		94.8	0.945		98.3	0.979		73.8	0.743		56.2	0.566		57.4	0.564		76.4	0.757	
Two_stage		71.2	0.704	30.2%	95.3	0.951	31.9%	99.1	0.990	31.7%	71.5	0.708	30.3%	44.3	0.433	33.6%	49.7	0.492	29.6%	71.9	0.713	31.7%
TCP_w/o_DA		75.8	0.761	30.8%	96.2	0.957	33.5%	99.5	0.994	30.5%	76.3	0.716	26.8%	47.5	0.480	29.9%	52.5	0.507	30.3%	74.2	0.735	29.8%
TCP		**77.6**	**0.765**	**38.7%**	**97.5**	**0.976**	**37.9%**	**100**	**1.0**	**33.9%**	**77.8**	**0.795**	**36.7%**	**54.4**	**0.528**	**38.2%**	**53.6**	**0.528**	**40.8%**	**76.8**	**0.765**	**37.2%**
Two_stage	70%	59.8	0.578	65.6%	89.1	0.880	70.0%	97.7	0.957	66.7%	57.9	0.572	63.3%	33.5	0.310	68.6%	33.7	0.322	65.6%	62.2	0.603	65.9%
TCP_w/o_DA		56.2	0.559	65.2%	89.6	0.905	63.9%	98.6	0.981	59.3%	63.7	0.658	56.8%	37.2	0.364	64.4%	36.5	0.351	61.5%	64.3	0.636	62.1%
TCP		**75.7**	**0.734**	**71.1%**	**91.8**	**0.925**	**71.9%**	**99.5**	**0.993**	**67.3%**	**71.3**	**0.721**	**67.9%**	**38.5**	**0.375**	**69.8%**	**38.4**	**0.364**	**70.3%**	**69.2**	**0.685**	**69.7%**
ResNet50-based (baseline)	12%	82.0	0.815		96.9	0.966		99.1	0.978		79.7	0.783		68.2	0.680		67.4	0.650		82.2	0.812	
Two_stage		77.3	0.782	35.1%	96.4	0.960	34.5%	99.6	0.980	37.8%	77.1	0.758	33.4%	49.3	0.503	29.8%	51.8	0.509	30.3%	75.6	0.749	32.9%
TCP_w/o_DA		81.2	0.809	36.3%	96.7	0.957	37.3%	**99.9**	**0.997**	36.9%	78.3	0.776	37.2%	48.2	0.478	35.7%	53.7	0.532	34.6%	76.7	0.758	35.5%
TCP		**84.1**	**0.846**	**40.5%**	**98.1**	**0.985**	**38.2%**	99.8	0.995	**39.2%**	**78.8**	**0.793**	**38.1%**	**51.4**	**0.510**	**36.8%**	**56.9**	**0.562**	**37.2%**	**78.8**	**0.782**	**37.6%**
Two_stage	46%	67.4	0.669	59.7%	90.6	0.897	59.3%	98.9	0.979	**60.7%**	65.6	0.644	57.3%	35.5	0.340	58.9%	39.7	0.399	59.0%	66.7	0.655	58.3%
TCP_w/o_DA		78.1	0.773	60.1%	93.2	0.916	59.7%	99.6	0.995	59.4%	71.1	0.709	56.5%	36.1	0.355	58.1%	43.1	0.429	58.3%	70.3	0.696	58.1%
TCP		**80.5**	**0.810**	**62.3%**	**93.9**	**0.931**	**61.8%**	**100**	**1.000**	60.3%	**73.0**	**0.728**	**61.0%**	**37.8**	**0.369**	**59.8%**	**47.7**	**0.458**	**60.2%**	**72.8**	**0.716**	**60.7%**

Here, *FLOPs* ↓ and *Param* ↓ denote the decrement of FLOPs and parameter size compared with the baseline, *F1* and *Acc* are the F1 score and accuracy on the target domain

The base model is an adversarial-based model shown in Fig. 3

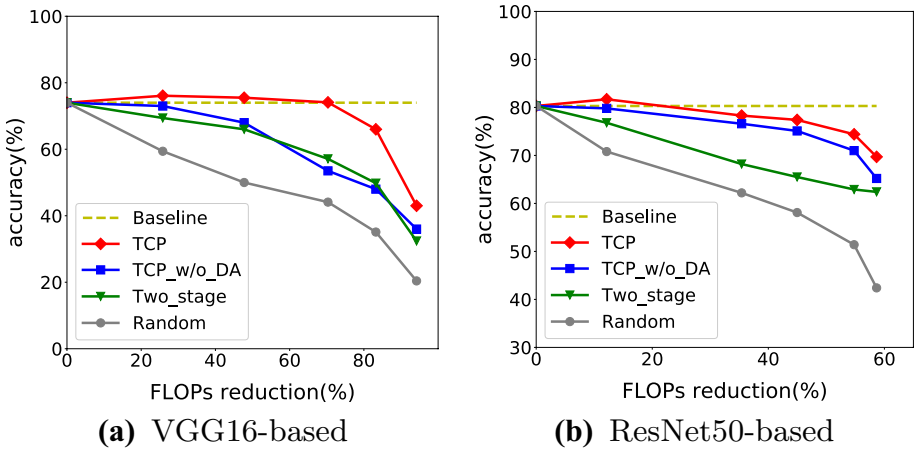
The bold values denote the best performance under the same compression rate of different tasks

Table 4 The performance on ImageCLEF-DA dataset (VGG16-based and ResNet50-based)

Models	FLOPs↓	I→P			P→I			I→C			C→I			C→P			P→C			Average		
		Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓	Acc	F1	Param↓
VGG16-based (baseline)	26%	73.0	0.727		81.9	0.825		90.2	0.897		79.5	0.802		63.6	0.626		89.9	0.891		79.8	0.794	
Two_stage		70.2	0.695	30.0%	79.6	0.778	31.3%	90.1	0.887	29.5%	66.6	0.647	31.6%	57.8	0.558	31.3%	86.1	0.847	27.6%	75.2	0.735	30.2%
TCP_w/o_DA		72.4	0.718	34.6%	80.8	0.801	**35.8%**	91.5	0.910	35.1%	80.2	0.788	36.6%	65.2	0.653	36.5%	88.2	0.876	34.7%	79.6	0.791	34.2%
TCP		**74.7**	**0.738**	**39.9%**	**82.3**	**0.813**	33.5%	**92.9**	**0.927**	**37.3%**	**82.5**	**0.819**	**38.0%**	**67.3**	**0.672**	**37.7%**	**89.9**	**0.895**	**35.6%**	**81.6**	**0.811**	**36.8%**
Two_stage	70%	58.6	0.580	66.7	72.2	0.719	64.8%	81.8	0.805	66.5%	59.5	0.587	59.4%	46.0	0.455	69.5%	76.3	0.766	63.0%	65.7	0.652	65.8%
TCP_w/o_DA		61.0	0.602	56.4%	71.9	0.708	67.0%	81.6	0.807	**68.0%**	57.7	0.568	65.6%	50.1	0.491	68.6%	71.8	0.709	67.7%	65.7	0.647	65.9%
TCP		**61.9**	**0.607**	**67.9%**	**72.8**	**0.717**	**67.8%**	**82.9**	**0.825**	67.7%	**61.5**	**0.614**	**69.9%**	**52.3**	**0.513**	**70.8%**	**79.5**	**0.782**	**69.6%**	**68.5**	**0.676**	**69.2%**
ResNet50-based (baseline)	12%	75.0	0.741		86.0	0.844		96.2	0.955		87.0	0.862		74.3	0.731		91.5	0.902		85.0	0.839	
Two_stage		72.1	0.717	30.8%	85.8	0.850	33.5%	96.3	0.962	36.6%	79.8	0.786	34.4%	66.9	0.653	31.6%	86.3	0.844	32.0%	80.8	0.802	33.3%
TCP_w/o_DA		73.2	0.722	34.9%	84.8	0.837	36.0%	95.9	0.947	35.4%	79.0	0.793	33.8%	70.3	0.692	32.3%	88.0	0.867	38.1%	81.2	0.809	35.6%
TCP		**75.8**	**0.745**	**38.8%**	**87.1**	**0.863**	**38.2%**	**96.9**	**0.972**	**37.8%**	**84.5**	**0.855**	**38.6%**	**72.6**	**0.722**	**38.1%**	**88.9**	**0.873**	**39.0%**	**82.5**	**0.838**	**37.9%**
Two_stage	46%	66.7	0.653	54.8%	73.6	0.721	57.3%	89.0	0.864	56.5%	70.0	0.683	56.0%	61.5	0.605	53.3%	83.1	0.822	56.4%	73.9	0.725	54.6%
TCP_w/o_DA		67.5	0.667	56.5%	75.2	0.743	58.4%	89.4	0.882	57.4%	69.6	0.686	57.3%	58.2	0.575	55.2%	82.3	0.819	59.0%	73.7	0.729	57.6%
TCP		**68.6**	**0.682**	**58.3%**	**79.3**	**0.791**	**59.3%**	**91.8**	**0.909**	**58.6%**	**72.8**	**0.719**	**59.9%**	**58.9**	**0.582**	**57.2%**	**84.8**	**0.840**	**60.5%**	**76.0**	**0.754**	**59.4%**

The base model is an adversarial-based model

The bold values denote the best performance under the same compression rate of different tasks



(a) VGG16-based **(b)** ResNet50-based

Fig. 5 The pruning result on task A→W with more compression rate

This is probably because some redundant channels in the base model are removed thus negative transfer is reduced. Especially for the results of ResNet50-based models, our ResNet50-based baseline achieves comparable or better performance compared with the discrepancy-based deep UDA approaches result of DDC [44] and DAN [23] in [25]. However, we can get better performance on half of the tasks and we even get 100% on task W→D after 46% FLOPs have been reduced.

Then, we evaluate our methods on ImageCLEF-DA dataset and the results are shown in Table 2. We can draw the same conclusion that our TCP method performs better on all tasks on ImageCLEF-DA dataset. We get higher accuracy than the baseline on all the VGG16-based experiments after 26% FLOPs have been reduced, and we also get higher accuracy on the target dataset on half of the tasks on ResNet50-based experiments after 12% FLOPs have been reduced, compared with the baseline which is almost the same as the discrepancy-based deep UDA approaches results of DAN in [25].

Adversarial-based UDA model pruning As for the adversarial-based base model, we also evaluate all the transfer learning tasks on Office-31 and ImageCLEF-DA datasets. The results are shown in Tables 3 and 4, which indicates that our TCP method outperforms other methods under the same compression rate (FLOPs reduction) and can reduce more parameters while performing pruning on the adversarial-based model. It is worth noting that TCP method achieves even better performance than the VGG16-based baseline model (which is not pruned). In addition, we also get higher accuracy on the target dataset on most of the tasks on ResNet50-based experiments after 12% FLOPs have been reduced. This is probably because some redundant channels in the base model are removed thus negative transfer is reduced.

Apart from Tables 1, 2, 3 and 4, Fig. 5 shows the comparison for all methods. Moreover, we also add a Random method which randomly removes a certain number of channels to achieve the same reduction of FLOPs. Combining these results, more conclusions can be made. (1) Compared with Two_stage, TCP method is more efficient because it is a unified framework and treat the pruning as a single optimization problem, while Two_stage is a split method and it does not take the target domain into consideration while pruning. (2) Compared with TCP_w/o_DA, the full TCP method uses the transfer channel evaluation to represent the discrepancy between the source and target domains. We try to remove those less important channels for both source and target domains and reduce negative transfer by reducing domain discrepancy. (3) As can be seen from Fig. 5, our TCP method outperforms other methods on unlabeled target dataset under different compression rate. (4) Our TCP method outperforms other methods on pruning two mainly kinds of UDA models: discrepancy-based models and adversarial-based models. This indicates that TCP method is generic, accurate, and efficient, which can dramatically reduce the computational cost of a deep UDA model without sacrificing the performance.

4.4 Acceleration analysis

Our proposed TCP method is capable of solving the acceleration problem of UDA models. In this section we show more

Table 5 The acceleration analysis on task A→D

Models	FLOPs↓ (%)	Time (ms)	Speed-up (%)
VGG16-base (baseline)		0.263	
TCP	26	0.187	28.90
TCP	70	0.168	36.10
ResNet50-base (baseline)		1.260	
TCP	12	0.981	22.10
TCP	46	0.888	29.50

Here, *FLOPs* ↓ denotes the decrement of FLOPs compared with the baseline, *time* means the average processing time of one image during inference. *speed-up* is the speed-up ratio compared with the baseline model. The base model is a discrepancy-based model shown in Fig. 2

Table 6 The acceleration analysis on task A→D

Models	FLOPs↓ (%)	Time (ms)	Speed-up (%)
VGG16-base (baseline)		0.244	
TCP	26	0.200	18.00
TCP	70	0.181	25.80
ResNet50-base (baseline)		1.350	
TCP	12	1.200	11.10
TCP	46	0.912	32.40

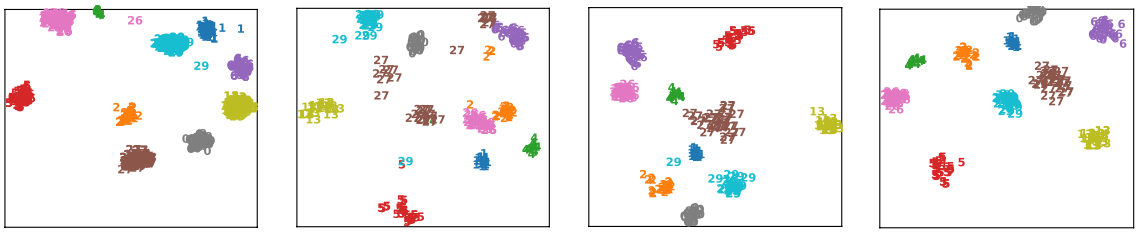
Here, the base model is an adversarial-based model shown in Fig. 3

details to present the acceleration degree of our TCP method compared with the baselines. Here we perform acceleration analysis on task A→D, and other tasks are the same. We evaluate our TCP method on the discrepancy-based base model and record the inference time to calculate the average inference time and speed-up ratio. Our experiments run on a NVIDIA TITAN XP GPU and the result is shown in Table 5. As can be seen, as for VGG16-based baseline, it can be accelerated by 28.90% and 36.10% after 26% and 70% FLOPs have been reduced with TCP respectively. In addition, as for ResNet50-based baseline, it can be accelerated by 11.10% and 32.40% after 12% and 46% FLOPs have been reduced respectively. Our TCP method is able to accelerate the discrepancy-based UDA models.

We can draw the same conclusion that TCP method is capable of accelerating the adversarial-based UDA models, and the results are shown in Table 6.

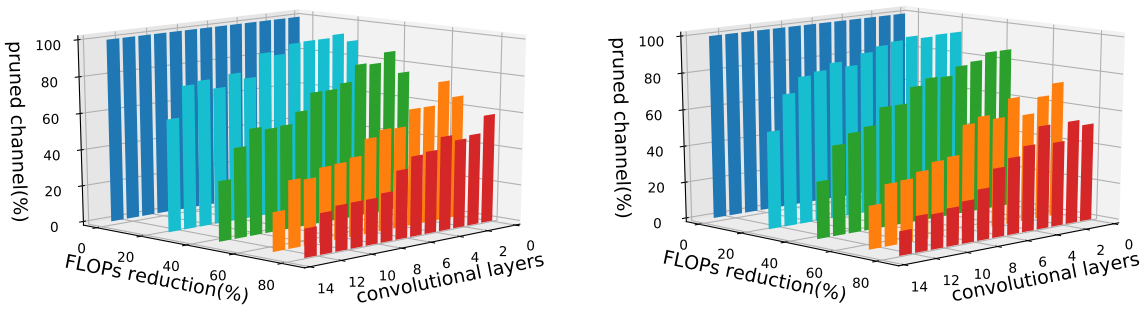
4.5 Visualization analysis

To evaluate the effectiveness of TCP method in reducing negative transfer, in Fig. 6, we follow [5] to visualize the model activations of task A→W pruned by different methods using t-SNE [5]. Fig. 6a shows the results of ResNet50-based baseline without pruning on the source domain. Figure 6b–d denote the result of ResNet50-based models on the target domain, which have been pruned by 12% FLOPs with our three methods Two_stage, TCP_w/o_DA and TCP respectively. The colored digits represent the ground truth of the examples, therefore the number is from 0 to 30, which denotes target dataset has 31 categories. Here we randomly pick 10 categories to visualize. As can be seen, the target categories are discriminated much more clearly with the model pruned by our TCP method. This suggests that our



(a) Baseline **(b)** Two_stage **(c)** TCP_w/o_DA **(d)** TCP

Fig. 6 The t-SNE visualization of network activations. **a** Is generated by ResNet50-based baseline without pruning on source domain. **b–d** Are generated by ResNet50-based (with 12% FLOPs pruned) with our three methods on target domain respectively. Best view in color



(a) Office31: A→W **(b)** ImageCLEF-DA: I→P

Fig. 7 The pruned structure of all the 13 convolutional layers of VGG16-based network on different dataset for deep unsupervised domain adaptation. Best view in color

TCP method is effective in learning more transferable features by reducing the cross-domain divergence.

To explore if there is any pattern in the structure of the pruned models, we show the structure of pruned models on task A→W and I→P in Fig. 7 with TCP method, and different color denotes different compression rate (FLOPs reduction). The dark blue bars represent channels of the 13 convolutional layers of the VGG16-based baseline before pruning. In addition, the light blue, green, orange and red bars represent the channels of pruned model under different compression rate respectively. As can be seen, higher layers have more redundancy than lower layers in VGG16-based models, and our TCP method prefer pruning the channels of higher layers. This is reasonable for unsupervised domain adaptation because the lower layers usually encode many common and important features for both source and target domains. Moreover, with more parameters in higher layers, our TCP method thus can prune more parameters under the same compression rate. The same result can be observed on ResNet50-based models.

5 Conclusion and future work

In this paper, we propose a unified Transfer Channel Pruning (TCP) method for accelerating deep unsupervised domain adaptation models. TCP method is capable of compressing the deep UDA model by pruning less important channels while simultaneously learning transferable features by reducing the cross-domain distribution divergence. Therefore, it reduces the impact of negative transfer and maintains competitive performance on the target task. TCP method is a generic, accurate, and efficient compression method that can be easily implemented by most deep learning libraries. Experimental results on two main kinds of UDA methods (the discrepancy-based methods and the adversarial-based methods) demonstrate the significant superiority of our TCP method over other methods.

In the future, we plan to apply our TCP method to heterogeneous UDA problems.

Acknowledgements This work is supported in part by National Key Research & Development Plan of China (No. 2017YFB1002802), NSFC (No. 61572471), and Beijing Municipal Science & Technology Commission (No. Z171100000117017).

References

1. Bengio Y, Courville A, Vincent P (2013) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828
2. Borgwardt KM, Gretton A, Rasch MJ, Kriegel HP, Schölkopf B, Smola AJ (2006) Integrating structured biological data by kernel maximum mean discrepancy. Bioinformatics 22(14):e49–e57
3. Chollet F (2017) Xception: deep learning with depthwise separable convolutions. arXiv preprint pp. 1610–02357
4. Denton EL, Zaremba W, Bruna J, LeCun Y, Fergus R (2014) Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in neural information processing systems, pp 1269–1277
5. Donahue J, Jia Y, Vinyals O, Hoffman J, Zhang N, Tzeng E, Darrell T (2014) Decaf: a deep convolutional activation feature for generic visual recognition. In: ICML. pp 647–655
6. Durugkar I, Gemp I, Mahadevan S (2016) Generative multi-adversarial networks. arXiv preprint arXiv:1611.01673
7. Fernando B, Habrard A, Sebban M, Tuytelaars T (2013) Unsupervised visual domain adaptation using subspace alignment. In: Proceedings of the IEEE international conference on computer vision. Sydney, Australia, pp 2960–2967. http://www.iccv2013.org/
8. Ganin Y, Lempitsky V (2014) Unsupervised domain adaptation by backpropagation. arXiv preprint arXiv:1409.7495
9. Ganin Y, Ustinova E, Ajakan H, Germain P, Larochelle H, Laviolette F, Marchand M, Lempitsky V (2016) Domain-adversarial training of neural networks. J Mach Learn Res 17(1):2096-30
10. Gong B, Grauman K, Sha F (2013) Connecting the dots with landmarks: discriminatively learning domain-invariant features for unsupervised domain adaptation. In: ICML. pp 222–230
11. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y: Generative adversarial nets. In: Advances in neural information processing systems. pp 2672–2680 (2014)
12. Han S, Mao H, Dally WJ (2015) Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149
13. Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems. pp 1135–1143
14. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. Las Vegas, USA, pp 770–778. http://cvpr2016.thecvf.com/
15. He Y, Zhang X, Sun J (2017) Channel pruning for accelerating very deep neural networks. In: International conference on computer vision (ICCV), vol. 2. Venice, Italy. http://iccv2017.thecvf.com/
16. Hoffman J, Tzeng E, Park T, Zhu JY, Isola P, Saenko K, Efros AA, Darrell T (2018) Cycada: cycle-consistent adversarial domain adaptation. In: ICML
17. Hou CA, Tsai YHH, Yeh YR, Wang YCF (2016) Unsupervised domain adaptation with label and structural consistency. IEEE Trans Image Process 25(12):5552–5562
18. Hu Y, Sun S, Li J, Wang X, Gu Q (2018) A novel channel pruning method for deep neural network compression. arXiv preprint arXiv:1805.11394
19. Huang J, Gretton A, Borgwardt K, Schölkopf B, Smola AJ (2007) Correcting sample selection bias by unlabeled data. In: Advances in neural information processing systems. pp 601–608
20. Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML
21. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp 1097–1105
22. Lin J, Rao Y, Lu J, Zhou J (2017) Runtime neural pruning. In: Advances in neural information processing systems. pp 2181–2191
23. Long M, Cao Y, Wang J, Jordan MI (2015) Learning transferable features with deep adaptation networks. In: ICML
24. Long M, Wang J, Ding G, Sun J, Yu PS (2013) Transfer feature learning with joint distribution adaptation. In: Proceedings of the

IEEE international conference on computer vision. Sydney, Australia, pp 2200–2207. http://www.iccv2013.org/
25. Long M, Zhu H, Wang J, Jordan MI (2017) Deep transfer learning with joint adaptation networks. In: ICML
26. Luo JH, Wu J (2018) Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. arXiv preprint arXiv:1805.08941
27. Luo JH, Wu J, Lin W (2017) Thinet: A filter level pruning method for deep neural network compression. arXiv preprint arXiv :1707.06342
28. Molchanov P, Tyree S, Karras T, Aila T, Kautz J (2017) Pruning convolutional neural networks for resource efficient inference. In: ICLR
29. Motiian S, Jones Q, Iranmanesh S, Doretto G (2017) Few-shot adversarial domain adaptation. In: Advances in neural information processing systems. pp 6670–6680
30. Pan SJ, Kwok JT, Yang Q (2008) Transfer learning via dimensionality reduction. AAAI 8:677–682
31. Pan SJ, Tsang IW, Kwok JT, Yang Q (2011) Domain adaptation via transfer component analysis. IEEE Trans Neural Netw 22(2):199–210
32. Pan SJ, Yang Q et al (2010) A survey on transfer learning. IEEE Trans Knowl Data Eng 22(10):1345–1359
33. Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in pytorch
34. Patel VM, Gopalan R, Li R, Chellappa R (2015) Visual domain adaptation: a survey of recent advances. IEEE Signal Process Mag 32(3):53–69
35. Pei Z, Cao Z, Long M, Wang J (2018) Multi-adversarial domain adaptation. In: Thirty-second AAAI conference on artificial intelligence. New Orleans, Louisiana, USA. https://aaai.org/Conferences/AAAI-18/
36. Rastegari M, Ordonez V, Redmon J, Farhadi A (2016) Xnor-net: imagenet classification using binary convolutional neural networks. In: ECCV. Springer, pp 525–542
37. Saenko K, Kulis B, Fritz M, Darrell T (2010) Adapting visual category models to new domains. In: European conference on computer vision. Springer, Hersonissos, Heraklion, Crete, Greece, pp 213–226. https://www.ics.forth.gr/eccv2010/intro.php/
38. Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. In: ICLR
39. Sun B, Feng J, Saenko K (2016) Return of frustratingly easy domain adaptation. AAAI 6:8
40. Sun B, Saenko K (2015) Subspace distribution alignment for unsupervised domain adaptation. In: BMVC. pp 24–1
41. Sun B, Saenko K (2016) Deep coral: Correlation alignment for deep domain adaptation. In: European conference on computer vision. Springer, Amsterdam, The Netherlands, pp 443–450. http://www.eccv2016.org/
42. Tahmoresnezhad J, Hashemi S (2017) Visual domain adaptation via transfer feature learning. Knowl Inf Syst 50(2):585–605
43. Tzeng E, Hoffman J, Saenko K, Darrell T (2017) Adversarial discriminative domain adaptation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. Honolulu, Hawaii, USA, pp 7167–7176. http://cvpr2017.thecvf.com/
44. Tzeng E, Hoffman J, Zhang N, Saenko K, Darrell T (2014) Deep domain confusion: maximizing for domain invariance. arXiv preprint arXiv:1412.3474
45. Wang J, Chen Y, Hao S, Feng W, Shen Z (2017) Balanced distribution adaptation for transfer learning. In: Data mining (ICDM), 2017 IEEE International Conference on IEEE. New Orleans, USA, pp 1129–1134. http://icdm2017.bigke.org/
46. Wang J, Feng W, Chen Y, Yu H, Huang M, Yu PS (2018) Visual domain adaptation with manifold embedded distribution alignment. In: 2018 ACM Multimedia Conference on Multimedia Conference. Seoul, Korea, pp 402–410. https://acmmm.org/2018/
47. Wang J, et al.: Everything about transfer learning and domain adapation. http://transferlearning.xyz
48. Yosinski J, Clune J, Bengio Y, Lipson H (2014) How transferable are features in deep neural networks? In: Advances in neural information processing systems. pp 3320–3328
49. Zhou A, Yao A, Guo Y, Xu L, Chen Y (2017) Incremental network quantization: Towards lossless cnns with low-precision weights. arXiv preprint arXiv:1702.03044
50. Zhuang F, Cheng X, Luo P, Pan SJ, He Q (2015) Supervised representation learning: transfer learning with deep autoencoders. In: IJCAI. pp 4119–4125