Model Recycling Framework for Multi-Source Data-Free Supervised Transfer Learning

Anonymous authors
Paper under double-blind review

Abstract

Increasing concerns for data privacy and other difficulties associated with retrieving source data for model training have created the need for *source-free transfer learning*, in which one only has access to pre-trained models instead of data from the original source domains. This setting introduces many challenges, as many existing transfer learning methods typically rely on access to source data, which limits their direct applicability to scenarios where source data is unavailable. Further, practical concerns make it more difficult, for instance efficiently selecting models for transfer without information on source data, and transferring without full access to the source models. So motivated, we propose a model recycling framework for parameter-efficient training of models that identifies subsets of related source models to reuse in both white-box and black-box settings. Consequently, our framework makes it possible for Model as a Service (MaaS) providers to build libraries of efficient pre-trained models, thus creating an opportunity for multi-source data-free supervised transfer learning.

1 Introduction

Many existing methods for transfer learning and domain adaptation heavily rely on access to data from the source domains (Pan et al., 2010; Herath et al., 2017; Kulis et al., 2011; Zhuang et al., 2019; Yao & Doretto, 2010; Sun & Saenko, 2015; Jiang & Zhai, 2007). This situation can give rise to privacy concerns, as organizations may not want to share sensitive information; for instance, healthcare providers may be reluctant to share patient information and security system maintainers may not want to risk sharing facial recognition data for system performance updates. Additionally, there may be issues with obtaining the source data such as when it is hard to retrieve due to technical difficulties or intellectual property restrictions (Li et al., 2020b; Chen et al., 2021; Liang et al., 2020; Ahmed et al., 2021b).

Recent advancements in source-free unsupervised domain adaptation (SFUDA) have presented solutions for a scenario where source data is not accessible (Fang et al., 2022). Purposely, SFUDA utilizes pre-trained source models to improve the generalization of a model on an unlabeled target dataset. Our work is similar to other approaches in the field of SFUDA (Li et al., 2020b; Chen et al., 2021; Liang et al., 2020; Ahmed et al., 2021b), in that it addresses the practical scenario where source data is not available during training. Importantly, a crucial aspect is often overlooked by the majority of SFUDA studies. When it is assumed that source data is not accessible, then it cannot be guaranteed that the available source models have been trained on domains related to the target task. And yet, most of the works only have experimented on classic domain adaptation benchmarks, which are somewhat related by design, *e.g.*, `Digits-Five` (Peng et al., 2019), `Office-31` (Saenko et al., 2010), and `Office-Home` (Venkateswara et al., 2017), *i.e,*, domains that share the same labels but are dissimilar in feature (and ambient) space.

Our approach is unique in that we consider such a *source-free supervised transfer learning* (SFSTL) setting (Lee et al., 2019), where we do not assume source models are trained on tasks with similar feature spaces or labels. Instead, we not only experiment on classic domain adaptation datasets, but also discuss the scenario where source and target tasks have very different datasets or their label sets are partially shared. Another distinguishing factor of our work is that we consider a framework that can search a pool of pre-trained models for those that are most helpful in enhancing the performance of new tasks. Furthermore, it also

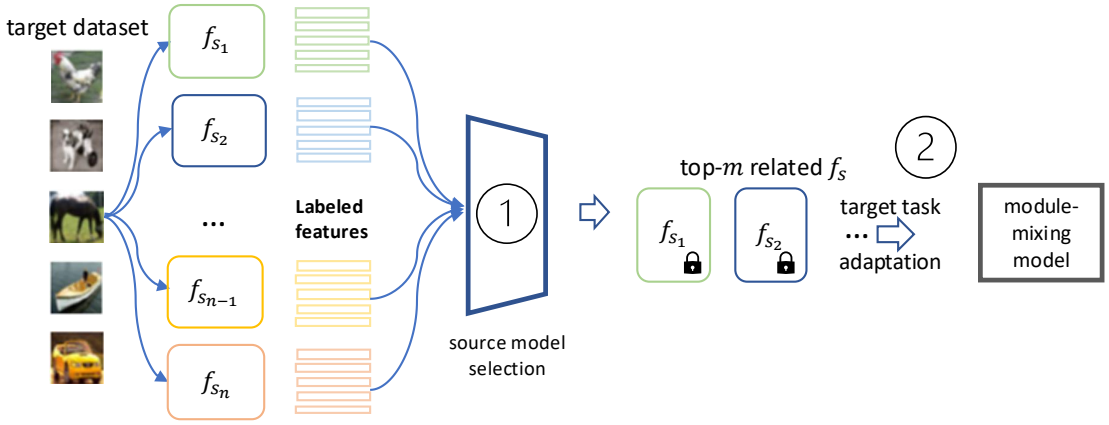

Figure 1: Proposed method in a nutshell: ① For the source model selection phase, we extract features for the target dataset with all the source feature extractors $\{\boldsymbol{f}_{s_n}\}_{n\in S_m}$, and use these features to select the top-m related source feature extractors. ② For the target task adaptation phase, the selected feature extractors are used to build a module-mixing model in either white-box or black-box settings, the details of which are shown in Figure 2.

allows one to do transfer with either white-box or black-box assumptions for the source models. For the *white-box scenario*, we assume many pre-trained source models are already available on a server or service, which defines in advance the model architecture to be used by all models. In the *black-box scenario*, only the extracted features before the classification head are accessible, as large pre-trained models are considered intellectual properties nowadays, thus model details are sometimes unavailable. In either case, users can tweak parameters, such as the number of source models, learning rate, and weight decay, to obtain a better model relative to training an independent model with their own data. This process would presumably require little technical knowledge. Moreover, since many MaaS platforms would accumulate pre-trained source models over time, we also provide a parameter-efficient solution for training source models to control memory requirements.

Our contributions are summarized below: *i*) We study an under-explored source-free supervised transfer learning scenario, where we are given a collection of related and unrelated source models to improve the performance of a new model on a labeled target task. Crucially, we only have access to the target data. *ii*) We propose a framework (shown in Figure 1) that can select several related source models (the number of which is set *a priori*), and utilize their knowledge on new tasks in both *white-box* and *black-box* scenarios. We treat these selected models as 'recycling materials' and reuse them to help learn a better model for the target tasks. *iii*) We perform extensive experiments to highlight the properties of our framework including a specially crafted `CIFAR100`-based experiment and an ablation study.

In the next section, we will discuss related works in relation to the proposed work. In Section 3, we define our problem setting and introduce the parameter-efficient finetune method we adopt as the basic building blocks in our model. In Section 4, we explain in detail each component of our model recycling framework, including the training of source models, the identification of most transferable source models, and the module-mixing model design. We also provide the rationale for the design choices of each component in the ablation study and the Appendix. Experiments are presented in Section 5 including the results for the ablation study. For example, we show that the proposed source model selection module is able to select similar tasks to adapt to with good performance in Figure 5, and compare our method to other model transferability evaluation methods in Table 5. We also analyze the generalization ability of the proposed model and the visualize the features extracted with the learned model. Finally, conclusions are presented in Section 6.

2 Related Work

Multi-source Supervised Transfer Learning Tong et al. (2021) presents a transferability measure that is characterized by the sample sizes, model complexity and the similarities between source and target tasks. Then, they use the transferability measure to form a convex combination of the predictions from different models. Most of the works in this class assume source datasets are available during adaptation (Li et al., 2020a; Jin et al., 2021; Xu et al., 2018; Tong et al., 2021; Li, 2022). Alternatively and most similar to our setting, Lee et al. (2019); Wu et al. (2024) do not assume the availability of source data. Specifically, Lee et al. (2019) avoid finetuning the source models by leveraging (maximal) correlation analysis and conditional expectation operators to build a classifier from the combined weighting of the feature functions from the source networks. The inexpensive weighting and lack of finetuning makes it efficient and effective at adapting to multiple source models in the few-shot regime. Our work is different in that we use a convex combination of feature representations *and* task-specific module parameters.

Source-Free Unsupervised Domain Adaptation Li et al. (2020b); Kurmi et al. (2021); Hou & Zheng (2021) focus on generating data for source domains to accommodate already existing unsupervised domain adaptation methods. For instance, Kurmi et al. (2021) generates proxy source samples by treating the trained source classifier as an energy-based model along with a parametric data generative neural network. Chen et al. (2021); Liu & Zhang (2021); Xiong et al. (2021) leverage self-supervised knowledge distillation to finetune source models by adopting a mean-teaching framework (Tarvainen & Valpola, 2017). Liang et al. (2020) use information maximization and self-supervised pseudo-labeling to adapt the target domain to pre-trained source models. Concerning the potential of utilizing diverse knowledge from multiple domains, Liang et al. (2021b); Ahmed et al. (2021b); Dong et al. (2021); Han et al. (2023) explore the possibility of multi-source data-free adaptation. For example, Ahmed et al. (2021b) extends the work from Liang et al. (2020) by combining the source models with trainable aggregated weights. Dong et al. (2021) introduces a confident-anchor-induced pseudo label generator with multiple source models to provide more reliable pseudo labels for target data. Liang et al. (2021b) studies a challenging scenario where only black-box source models are available during adaptation. Most works in this class assume there is no labeled target data, while we discuss the case when target data is labeled (Fang et al., 2024).

Averaging Model Weights Weighted averaging is widely adopted in optimization approaches. Stochastic Weight Averaging aggregates weights along a single optimization trajectory (Izmailov et al., 2018). Matena & Raffel (2021) merge models that are fine-tuned on different text classification tasks with the same pre-trained initialization. Wortsman et al. (2022) focus on averaging weights across independent runs of models on the same dataset. In their "cross-dataset soup", models are trained on different datasets and adapted to target tasks by learning a set of averaging weights. Another similar work Ram'e et al. (2022) also reuses multiple pretrained foundation models to adapt to a new task. The difference is that they finetune the pretrained foundation models on the new task before averaging their weights to create the final model, and assumes that all the pretrained models have the same architecture. Our work is different in that new modules allocated for target tasks are *trained together* with the mixing weights to learn task-specific features. Further, we also discuss how to mix features from different models when their dimensions are different in a black-box scenario.

Finetuning Models with Task-Specific Modules As deep learning models grow in size (He et al., 2015; Dosovitskiy et al., 2020; Radford et al., 2019), storing and finetuning the whole model becomes exceedingly challenging, due to needing expensive computational resources. Therefore, there is a line of work proposing that instead of finetuning all the parameters in a pre-trained model, one can instead partition the network into a frozen backbone model and task-specific modules for finetuning. The idea has been applied to generative models for synthesizing images (Perez et al., 2018; Cong et al., 2020; Verma et al., 2021), discriminative models for image classification (Verma et al., 2021), and finetuning for downstream tasks with large language models (Houlsby et al., 2019; Li & Liang, 2021). This concept is used in building our white-box parameter-efficient models.

Model Transferability Evaluation There are studies focused on assessing the transferability of trained models for a specific task. For instance, the pioneer work LEEP (Nguyen et al., 2020), measures the log expectation of the empirical predictor by estimating the joint distribution across pretrained labels and the target labels. It features fast selection speed; however, it requires the availability of the source models' classification heads. LogME proposes to estimate the maximum value of label evidence given features extracted by trained models and obtained the Logarithm of Maximum Evidence (LogME) measure (You et al., 2021). It exceeds LEEP and NCE (Tran et al., 2019) in terms of evaluation accuracy, and only uses extracted features and labels to assess source models. However, it still needs extra training to determine the best suited model, which makes this approach impractical when the model pool is too large. Guo et al. (2023) recognizes a similar challenge as in our work, which is how can a user search for useful models in the *learnware* market. They identify useful models simply by comparing user requirements with model specifications, without running models. However, they need the model developers to submit specifications of their model, which involves generating a feature matrix that describes their data with a public feature extractor. In practice, the model specification details might not always be available. Our choice of model selection features a non-parametric k-NN method, which offers a trade-off between selection efficiency and accuracy. We note that the model selection module of our framework can be used interchangeably with the aforementioned methods. Note however that several factors need to be considered when choosing which one to use, including the size of the model pool, the transferability accuracy demand of the users, and also, whether the source model information is limited.

3 Background

3.1 Problem Setting

In this work, we aim to address the challenge of adapting a collection of classification models trained on different source tasks to a new target task. The goal is to optimize the classification accuracy of the target task. Specifically, consider we have N source models $\{\boldsymbol{h}_{s_1}, \boldsymbol{h}_{s_2}, \cdots, \boldsymbol{h}_{s_N}\}$ corresponding to N source tasks $\{\mathcal{T}_{s_1}, \mathcal{T}_{s_2}, \cdots, \mathcal{T}_{s_N}\}$, and that we also have a target task $\mathcal{T}_t$ with a labeled dataset $\{(\mathcal{X}_t, \mathcal{Y}_t)\}$. Importantly, during training for $\mathcal{T}_t$, we only have access to the target task dataset $\{(\mathcal{X}_t, \mathcal{Y}_t)\}$ and source models $\{\boldsymbol{h}_{s_1}, \boldsymbol{h}_{s_2}, \cdots, \boldsymbol{h}_{s_N}\}$. Each source model consists of a feature extractor $\boldsymbol{f}$ and the classification head $\boldsymbol{c}$. We denote the source model as $\boldsymbol{h}_{s_n} := (\boldsymbol{f}_{s_n} \circ \boldsymbol{c}_{s_n})$, with $\boldsymbol{f}_{s_n} : \mathcal{X}_t \to \mathbb{R}^{d_{s_n}}$ being the feature extractor with output feature vector of dimension d_{s_n}, $\boldsymbol{c}_{s_n} : \mathbb{R}^{d_{s_n}} \to \mathbb{R}^{\alpha}$ being a α-way classifier and $\circ$ denoting function composition. The goal is to learn a classification model $\boldsymbol{h}_t$ with all the available knowledge. In practice, source and target tasks do not need to be restricted to having the same number of classes since we only use source models as feature extractors.

3.2 Efficient Feature Transformation for Task-Specific Modules

In the white-box setting, we propose to build models with task-specific modules for source and target tasks because they are memory efficient and less likely to overfit on small datasets. Specifically, we adopt a task-specific module design termed efficient feature transformation (EFT) (Verma et al., 2021) for our source and target models. EFT proposes appending a small convolutional transformation to each convolutional layer's output feature maps. The transformation involves two kinds of convolutional kernels that capture spatial features within groups of channels (*group-wise filter*) and features across channels at every pixel in the feature map (*point-wise filter*). Given the feature maps $F \in \mathbb{R}^{M \times Q \times K}$ of a layer, with K being the number of feature maps, and M and Q being the spatial dimensions of each feature map, to apply a *groupwise filter* $\omega_i^s \in \mathbb{R}^{3 \times 3 \times a}$, we need to first split F into K/a groups of a feature maps. We then convolve a unique filter ω_i^s with each group and get a new group of feature maps $H_i^s \in \mathbb{R}^{M \times Q \times a}$ (*i.e.*, each group of feature maps has a unique spatial filter). Subsequently, we concatenate all the K/a new feature maps H_i^s into $H^s \in \mathbb{R}^{M \times Q \times K}$, which has the same dimension as old feature maps F. To apply a *point-wise filter* $\omega_i^d \in \mathbb{R}^{1 \times 1 \times b}$, the same procedures as above need to be applied to split F into K/b groups of b feature maps to create $H^d \in \mathbb{R}^{M \times Q \times K}$. The final feature maps are $H = H^s + \gamma H^d$, where $\gamma \in \{0, 1\}$ indicates if the point-wise filters are used or not. By setting $a \ll K$ and $b \ll K$, the amount of trainable parameters is substantially reduced. For instance,

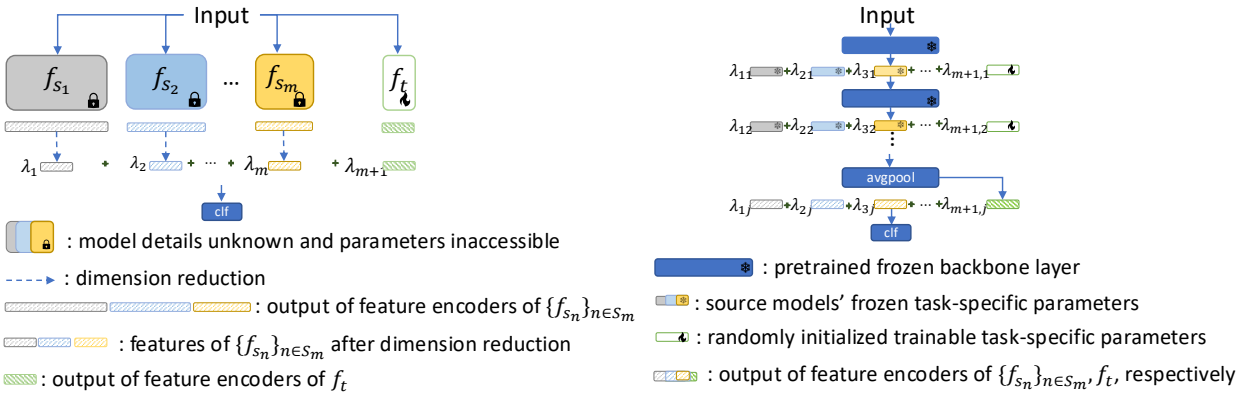

Figure 2: Proposed module-mixing model. Left (black-box): The output features from the source models are first reduced to match the feature dimension of the target model, and then combined with the target features as in equation 4; Right (white-box): For each task-specific layer, modules from source models (frozen during training), and a randomly initialized EFT module are combined as in equation 3. The output features of the source models and that of the new target model are also aggregated as in equation 4.

using a ResNet18 backbone, $a = 8$, and $b = 16$ results in 449k parameters per new task, which is 3.9% the size of the backbone. Additional details can be found in the Appendix A.1 and in Verma et al. (2021).

4 Proposed Method

To highlight the flexibility of the framework, we discuss both white-box and black-box assumptions for our source models. We provide a parameter-efficient solution for source model generation. However, we emphasize that the framework is not restricted to using this design, which is why we provide a more flexible setting for the black-box scenario, where we use different pre-trained APIs as our source models. Within both scenarios, the adaptation model in Figure 2 is obtained through two steps. *i)* Given a large pool of N pre-trained source models, it is not practical nor efficient to learn (transfer) from all of them for a new task $\mathcal{T}_t$. Therefore, we first aim to select a subset of m models that are likely trained on related source domains, from which $\mathcal{T}_t$ can learn to improve performance. *ii)* After the selection process, instead of fine-tuning the pre-trained models directly, we propose a method dubbed *module-mixing* for target task adaptation.

White-box scenario: all models are trained with a modular architecture sharing a common pre-trained backbone. For each task-specific layer, we perform a convex combination on *a)* all the task-specific modules from selected models and a randomly initialized task-specific module; *b)* the output of the selected source feature extractors and that of the new model.

Black-box scenario: we only have access to the output features from source models, and we are transferring to a target model that may have a different (and potentially smaller) architecture from the source models. Specifically, we first reduce the dimensionality of the source model features to match that of the target model, and then aggregate them using a convex combination.

4.1 Source Model Generation

For the white box scenario in which we can control the source model generation, we propose using EFT to build a large library of source models to keep computation and memory requirements under control. Assume there are labeled datasets $\{(\mathcal{X}_{s_n}, \mathcal{Y}_{s_n})\}$ for tasks $\mathcal{T}_{s_n}$, $n \in \{1, \cdots, N\}$, with $\mathcal{X}_{s_n}$ and $\mathcal{Y}_{s_n}$ being the feature and label domains, respectively. The feature extractor $\boldsymbol{f}_{s_n}$ consists of a backbone ϕ_{bb} that is shared amongst all the source models and a set of task-specific modules $\{\boldsymbol{\theta}_{s_n,j}\}_{j=1}^{J}$, with j being the task-specific layer index and J being the total number of task-specific layers in the model. Below we write $\boldsymbol{\theta}_{s_n}$ for simplicity. We

train model $\boldsymbol{h}_{s_n}$ by minimizing a standard cross-entropy loss:

$$\begin{aligned}\boldsymbol{\theta}^*_{s_n}, \boldsymbol{c}^*_{s_n} &= \arg\min_{\boldsymbol{\theta}_{s_n}, \boldsymbol{c}_{s_n}} \mathcal{L}(\boldsymbol{\phi}_{bb}, \boldsymbol{\theta}_{s_n}, \boldsymbol{c}_{s_n}; \mathcal{X}_{s_n}, \mathcal{Y}_{s_n}) \\ &= -\mathbb{E}_{(x,y)\in\mathcal{X}_{s_n}\times\mathcal{Y}_{s_n}} \sum_{i=1}^{\alpha} t_i \log(\delta_i(\boldsymbol{h}_{s_n}(x))),\end{aligned}$$

where $\boldsymbol{\phi}_{bb}$ is frozen during training, $\boldsymbol{\theta}^*_{s_n}$ and $\boldsymbol{c}^*_{s_n}$ are the optimized task-specific modules and classifier, respectively, $t_i = 1$ for the corresponding class i in one-hot encoding (of α classes), and $\delta_i(\boldsymbol{o}) = \frac{exp(\boldsymbol{o}_i)}{\sum_{j=1}^{m} exp(\boldsymbol{o}_j)}$ is i-th element of the softmax activation output vector $\boldsymbol{o} = \boldsymbol{h}_{s_n}(x)$. We also give further training details, such as the learning rate and batch size in Appendix A.2 Hyperparameter settings.

4.2 Selection of Related Source Models

Given the target training dataset $\{(\mathcal{X}_t, \mathcal{Y}_t)\}$, we can extract a dataset of features by collecting the output of the feature extractors $\{\boldsymbol{f}_{s_n}\}_{n=1}^{N}$. Each feature dataset is denoted as $D_{t,s_n} := \{\boldsymbol{f}_{s_n}(\mathcal{X}_t), \mathcal{Y}_t\}$, for $n \in \{1, \cdots, N\}$. The goal is to select the top-m corresponding source models (denoted as S_m) with the best k-NN classifier validation accuracy via data from each D_{t,s_n}. The rationale is that a higher validation accuracy indicates that the feature extractor used for extracting features was trained on a domain more related to the target task. We consider different values of m in the experiments. Below are the details of our k-nearest neighbor (k-NN) (Cover & Hart, 1967) classifier. We denote the set of nearest neighbors of $\boldsymbol{f}_{s_n}(\boldsymbol{x})$ as $S_{\boldsymbol{x},s_n}$. In the experiment we set $k = 5$.

For a given validation point $\boldsymbol{x}$, we obtain $\boldsymbol{f}_{s_n}(\boldsymbol{x}_i{}'') \in S_{\boldsymbol{x},s_n}, i \in \{1, \cdots, k\}$ that satisfies

$$\text{dist}(\boldsymbol{f}_{s_n}(\boldsymbol{x}), \boldsymbol{f}_{s_n}(\boldsymbol{x}')) \geq \max\{\text{dist}(\boldsymbol{f}_{s_n}(\boldsymbol{x}), \boldsymbol{f}_{s_n}(\boldsymbol{x}_i{}''))\}, \quad \forall(\boldsymbol{f}_{s_n}(\boldsymbol{x}'), \boldsymbol{y}') \in D_{t,s_n} \setminus S_{\boldsymbol{x},s_n}, \tag{1}$$

where $\text{dist}(\cdot,\cdot)$ is a distance metric. We use $D_{t,s_n} \setminus S_{\boldsymbol{x},s_n}$ to represent the subset of D_{t,s_n} excluding $S_{\boldsymbol{x},s_n}$. In this work, we employ the Euclidean distance:

$$\text{dist}(\boldsymbol{f}_{s_n}(\boldsymbol{x}), \boldsymbol{f}_{s_n}(\boldsymbol{x}')) = \left(\sum_{i=1}^{d} |\boldsymbol{f}_{s_n}(\boldsymbol{x})_i - \boldsymbol{f}_{s_n}(\boldsymbol{x}')_i|^2\right)^{\frac{1}{2}}.$$

Moreover, the k-NN classifier g_{KNN} is defined as

$$g_{\text{KNN}} = \text{mode}(\boldsymbol{y}'' : (\boldsymbol{f}_{s_n}(\boldsymbol{x}_i{}''), \boldsymbol{y}'') \in S_{\boldsymbol{x},s_n}), \tag{2}$$

where $\text{mode}(\cdot)$ takes the label value that occurs most frequently in the set $S_{\boldsymbol{x},s_n}$. g_{KNN} will predict the label for each validation point $\boldsymbol{x}$ based on a majority voting scheme with the nearest neighbors in $S_{\boldsymbol{x},s_n}$.

Remarks k-NN leverages the majority voting of nearest neighbors when predicting the class labels. A higher k-NN score means most of the nearest neighbors have the same label as the sample's ground truth, which indicates the feature space created by the source feature extractor presents the classes in the target task better. Furthermore, when selecting useful sources, our main concern is to find a solution that is *flexible* to both white-box and black-box scenarios, while *maintaining its efficiency* when having to deal with selecting from a large pool of models. k-NN, with its non-parametric property, offers insights of how transferable the source models are to the target tasks without any training, which makes it an efficient searching method.

4.3 Module-Mixing with Distance Correlation Loss

White-box Scenario In this setting, the architecture for all models is defined in advance for convenience. After selecting the top-m source models, we will use them to build the module-mixing model for the target task t. As mentioned above, each pre-trained feature extractor $\boldsymbol{f}_{s_n}$ has task-specific weights $\{\boldsymbol{\theta}_{s_n,j}\}_{j=1}^{J}$. For each task-specific layer j, the weights for the target task t are

$$\boldsymbol{\theta}_{t,j} = \lambda_{m+1,j}\boldsymbol{\theta}_{new} + \sum_{n\in S_m} \lambda_{n,j}\boldsymbol{\theta}_{s_n,j}, \tag{3}$$

with $\boldsymbol{\theta}_{new}$ being a randomly initialized new module, $\boldsymbol{\theta}_{s_n,j}$ the frozen task-specific module of layer j of a selected source model from S_m, and $\sum_{n=1}^{m+1} \lambda_{n,j} = 1, \forall j \in \{1, \cdots, J\}$. Further, the combined feature representation of the module-mixing model is

$$\boldsymbol{f}(\cdot) = \lambda_{m+1,J+1} \boldsymbol{f}_t(\cdot) + \sum_{n \in S_m} \lambda_{n,J+1} \boldsymbol{f}_{s_n}(\cdot), \tag{4}$$

with $\{\boldsymbol{f}_{s_n}(\cdot)\}_{n \in S_m} \in \mathbb{R}^{\cdot \times d_{s_n}}$ and $\boldsymbol{f}_t(\cdot) \in \mathbb{R}^{\cdot \times d_t}$ being the features (outputs before the classification head) of selected source models and that of the target model, respectively, and $\sum_{n=1}^{m+1} \lambda_{n,J+1} = 1$. Note that $\{\lambda_{n,j}\}_{n=1}^{m+1}, j \in \{1, \cdots, J+1\}$, denoted as $\boldsymbol{\lambda}_t$ for simplicity, are parameters that can be adjusted at initialization and then learned. By setting the initial values of $\boldsymbol{\lambda}_t$, we can control the importance of each task. For instance, we can assume having no prior information on the task importance by initializing $\boldsymbol{\lambda}_t$ in all layers with a uniform distribution as done below in the experiments.

Black-box Scenario We now assume the source models are all black-box APIs, which only produce the features before the classification head. We choose a much smaller architecture for the target model with feature dimensionality being smaller than that extracted from the black-box APIs. The rationale behind this choice is that we want to prevent overfitting provided that the target task usually has a relatively small dataset. Since the extracted feature dimensions from models can be different, we align the dimensions of the APIs' output features to that of the target model via FastICA (Hyvarinen, 1999; Hyvärinen & Oja, 2000), an efficient algorithm for independent component analysis. Then we combine the features as in equation 4. In the experiments, we show that we can still benefit from such a simplified strategy.

Distance Correlation Loss We expect the new modules for the target task can learn knowledge that is not in the mixing modules from the source tasks. Specifically, if we can encourage the learned features from the new modules to be more independent from that of the source models, we could potentially achieve better performance. One way to do so is to train the target module-mixing model with distance correlation loss, which was proposed by Zhen et al. (2022) as a loss function for improving robustness of neural networks against adversarial attacks.

Distance correlation (DC) (Sz'ekely et al., 2007) is a measure of dependence between random vectors. DC between two random variables $X \in \mathbb{R}^p$ and $Y \in \mathbb{R}^q$ satisfies $0 \leq DC \leq 1$, and $DC = 0$ if and only if X and Y are independent. Further, $DC(X, Y)$ is defined for X and Y in arbitrary dimensions. This property allows one to minimize DC between $\{\boldsymbol{f}_{s_n}\}_{n \in S_m}$ and $\boldsymbol{f}_t$ even when their output feature dimensions $d_{s_n} \neq d_t$. Here we follow the notation by Sz'ekely et al. (2007). We use a stochastic estimate of DC by averaging over minibatches $\boldsymbol{x}$ with n samples each. The objective for minimizing the distance correlation is

$$\frac{\langle \boldsymbol{A}(\boldsymbol{f}_{s_n}; \boldsymbol{x}), \boldsymbol{B}(\boldsymbol{f}_t; \boldsymbol{x}) \rangle}{\sqrt{\langle \boldsymbol{A}(\boldsymbol{f}_{s_n}; \boldsymbol{x}), \boldsymbol{A}(\boldsymbol{f}_{s_n}; \boldsymbol{x}) \rangle \langle \boldsymbol{B}(\boldsymbol{f}_t; \boldsymbol{x}), \boldsymbol{B}(\boldsymbol{f}_t; \boldsymbol{x}) \rangle}}, \tag{5}$$

where $\langle \boldsymbol{A}, \boldsymbol{B} \rangle = \sum_{i,j}^{n} (\boldsymbol{A})_{i,j} (\boldsymbol{B})_{i,j}$, $\boldsymbol{A}(\boldsymbol{f}_{s_n}; \boldsymbol{x}) \in \mathbb{R}^{n \times n}$ (simplified as $\boldsymbol{A}$) and $\boldsymbol{B}(\boldsymbol{f}_t; \boldsymbol{x}) \in \mathbb{R}^{n \times n}$ (simplified as $\boldsymbol{B}$) are distance matrices computed with $X := \boldsymbol{f}_{s_n}(\boldsymbol{x}) \in \mathbb{R}^{n \times d_{s_n}}$ and $Y := \boldsymbol{f}_t(\boldsymbol{x}) \in \mathbb{R}^{n \times d_t}$, respectively, with

$$a_{k,l} = ||X_k - X_l||, \quad \bar{a}_{k,\cdot} = \frac{1}{n} \sum_{l=1}^{n} a_{k,l}, \quad \bar{a}_{\cdot,l} = \frac{1}{n} \sum_{k=1}^{n} a_{k,l},$$

$$\bar{a}_{\cdot,\cdot} = \frac{1}{n^2} \sum_{k,l=1}^{n} a_{k,l}, \quad A_{k,l} = a_{k,l} - \bar{a}_{k,\cdot} - \bar{a}_{\cdot,l} + \bar{a}_{\cdot,\cdot},$$

where $k, l \in \{1, \cdots, n\}$, and $A_{k,l}$ is the k^{th} row and l^{th} column of $\boldsymbol{A}$. Similarly, we can define $b_{k,l} = ||Y_k - Y_l||$, and $B_{k,l} = b_{k,l} - \bar{b}_{k,\cdot} - \bar{b}_{\cdot,l} + \bar{b}_{\cdot,\cdot}$. We optimize all the module-mixing models with the cross-entropy loss and DC loss as below, with σ being a constant trade-off parameter:

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \sigma \sum_{n \in S_m} \mathcal{L}_{DC}(\boldsymbol{f}_{s_n}(\cdot), \boldsymbol{f}_t(\cdot)), \tag{6}$$

Algorithm 1 Module-mixing Model.

Data: Training data $D_t = \{\mathcal{X}_t, \mathcal{Y}_t\}$; Selected source models $\{\boldsymbol{f}_{s_n}\}_{n \in S_m}$;
Result: White-box: Module-mixing model $\boldsymbol{f}$ constructed with equation 4 and equation 3; Black-box: Module-mixing model $\boldsymbol{f}$ constructed with only equation 4

for *epoch* $\leftarrow 1$ **to** C **do**
  for *each minibatch* $\boldsymbol{x}$ **do**
    for $n \leftarrow 1$ **to** S_m **do**
      Calculate distance matrices $\boldsymbol{A}(\boldsymbol{f}_{s_n}; \boldsymbol{x})$ and $\boldsymbol{B}(\boldsymbol{f}_t; \boldsymbol{x})$ in equation equation 5
      Calculate $\mathcal{L}_{DC}(\boldsymbol{f}_{s_n}(\boldsymbol{x}), \boldsymbol{f}_t(\boldsymbol{x}))$ as in equation 5
    end
    Calculate $\mathcal{L}_{total}$ as in equation 6
    Update $\boldsymbol{\theta}_{new}, \boldsymbol{c}_t, \boldsymbol{\lambda}_t$ using $\boldsymbol{\theta}^*_{new}, \boldsymbol{c}^*_t, \boldsymbol{\lambda}^*_t = \arg\min \mathcal{L}_{total}(\boldsymbol{h}_t; \mathcal{X}_t, \mathcal{Y}_t)$
  end
end

The objective $\boldsymbol{\theta}^*_{new}, \boldsymbol{c}^*_t, \boldsymbol{\lambda}^*_t = \arg\min \mathcal{L}_{total}(\boldsymbol{h}_t; \mathcal{X}_t, \mathcal{Y}_t)$ indicates that the trainable parameters in $\boldsymbol{h}_t$ are the new modules $\boldsymbol{\theta}_{new}$, the classification head $\boldsymbol{c}_t$ and convex combination parameters $\boldsymbol{\lambda}_t$. The detailed algorithm is presented in Algorithm 1. Overall, our motivation for using distance correlation loss is that it encourages features extracted from previous models to be independent from the features learned from the new tasks; Furthermore, since the output feature dimensions for new and old tasks might not be the same, we need the loss to be able to handle features of different dimensions.

The reasons for using this module-mixing technique are threefold. First, it addresses feature saturation and the issue of plasticity loss that occurs when model weights are fine-tuned over time, which is discussed in Dohare et al. (2021); Ash & Adams (2019). They argue that, in a continual learning setting, if a model is finetuned sequentially on several tasks, the model will likely not benefit from random initialization for later tasks as it will lose plasticity over time. Therefore, our goal is to devise a way to make use of pre-trained models rather than attempting to modify them. Our model ensures that we can learn from previous knowledge while also adding new capacity at the start of training for each new task. Secondly, the flexible nature of the model enables us to investigate whether there is a benefit to learning from more tasks, how to balance training and inference efficiency, and the number of tasks to be reused. The reason is that when the number of selected tasks increases, the amount of trainable parameters of the model grows slowly since only $\boldsymbol{\lambda}_t$ is growing, which is negligible relative to $\boldsymbol{\theta}_{new}$ and $\boldsymbol{c}_t$. Importantly, the last and third reason is that this approach allows us to transfer whether the source models are treated as white- or black-boxes.

5 Experiments

Datasets We create three main tasks with `CIFAR100` (Krizhevsky, 2009), `Office-31` (Saenko et al., 2010), and the S^{long} stream in `CTrL` (Continual Transfer Learning benchmark) (Veniat et al., 2020). Each dataset represents a unique scenario. With a special task creation scheme for `CIFAR100` (detailed below), we study the cases where the classes in each task overlap or are completely different. With `Office-31`, each task has the same labels but different data distributions. With the challenging `CTrL`, we have a large pool of models from which to start, and source and target tasks come from very distinct datasets and have diverse sample sizes. Moreover, an experiment on `Tiny-ImageNet` (Le & Yang, 2015) is shown in the ablation study.

Network Architecture and Implementation Details For k-NN source model selection, we set $k = 5$ for all experiments, then select the source models with the top-m highest k-NN scores based on the results of each task's validation set. One experiment discussing other settings for k is provided in the Appendix A.5. *(i)* In the white-box setting, for source model generation and target task adaptation, we choose a pre-trained ResNet-18 on ImageNet as our frozen backbone and train the EFT modules with the Adam optimizer. We set $a = 8$ and $b = 16$ for EFT for experiments on `CIFAR100` and `Office-31`, and $a = 2$ and $b = 1$ for EFT on `CtrL`. *(ii)* In the black-box setting, a simple LeNet-5 (LeCun et al., 1998) is used as target model. We provide results for when the source models are (a) ResNet-18s with EFT and (b) a

pre-trained MAE (He et al., 2021) (written as API in the tables). The DC loss trade-off is set to $\sigma = 0.05$ in all experiments. Implementation details can be found in the Appendix A.2, and the source code can be found at `https://anonymous.4open.science/r/module_mixing-00C7`.

Baselines (a) **Independent Model:** Add randomly initialized EFT modules to a pre-trained frozen backbone and train the model solely with target task data. (b) **Multi-Source SVM**: Extract features from m randomly selected source models with the target training set, then concatenate the m set of extracted features and use them to train an SVM model (Cortes & Vapnik, 1995). (c) **Maximum Correlation Weighting (MCW)**: Learn the maximum correlation parameters of m randomly selected source feature extractors with one pass of the target training data (Lee et al., 2019). (d) **Finetune Source**: Take the source model selected by k-NN with $m = 1$, and finetune the corresponding task-specific modules. (e) **Cross-dataset Soup**: Train an independent model for the target task first, and then train the mixing weights with selected source models (Wortsman et al., 2022) with top-m k-NN scores. (f) **Model Stacking**: Make predictions with the selected top-m source models and an independent model for the target task, then use those predictions as input features in a logistic regression model. (g) **DECISION**: Freezes the last classification layers of source models and jointly optimize the source feature encoders together with their aggregated weights. The source classification layers are used to generate pseudo labels for target data (Ahmed et al., 2021a). (h) **DATE (SHOT)**: Utilizes a source-similarity transferability module and a proxy discriminability perception module for weight determination for source models (Han et al., 2023). (i) **DINE**: First distills knowledge from the source predictor to a customized target model, then fine-tune the distilled model to further fit to the target domain (Liang et al., 2021a).

MCW, **DECISION**, **DATE**, **DINE** are all previous state-of-the-art (SOTA) methods. However, only **MCW** is designed for the same multi-source data-free supervised transfer learning setting (target task has labels). **DECISION** and **DATE** have the most similar problem setting as ours, which is multi-source data-free unsupervised domain adaptation (target task does not have labels). Furthermore, **DINE** is proposed for adapting to single-source and multi-source black-box models when target task does not have labels. The results of **DECISION**, **DATE** and **DINE** are obtained by strictly conforming to their public source code.

Though **DECISION**, **DATE (SHOT)**, **DINE** achieve great performance under each of their settings, they all assume close-set label spaces between source and target domains. Besides, due to their unsupervised nature, they all use pseudo labels predicted by the source models to train the target models. According to Yi et al. (2023), when the noise rate in pseudo labels is high at the beginning of the training, the target model will quickly remember the noise due to confirmation bias. All the above reasons explain the performance deterioration of these methods under our setting.

5.1 White-box Scenario

CIFAR100 We create a set of 40 tasks with `CIFAR100`. Different tasks in the collection may share some of the same classes. However, we ensure that the overlapping classes do not have the same data samples by splitting data for each class into two even parts and only allowing each class to exist in tasks twice; this allows us to create a scenario where each task is private and has unique data samples. Details of each task are shown in the Appendix A.2.1. We run each experiment 3 times. For each run, we randomize the list of 40 tasks and pick the first ten of the list as the starting source task pool. For the rest of the tasks, $11 - 40$, we proceed as follows. First, for target task 11, we randomly select a subset of size $e = \{40, 80, 160, 320\}$. With each subset, we select $m = \{1, 3, 5\}$ related source tasks from the pool to train our new module-mixing model. Before moving to target task 12, we train an independent model (via EFT) for task 11 with all available data. This is then repeated for all the other tasks, $12 - 40$. Performance is reported for different values of m and e averaged over tasks $11 - 40$. This is done to obtain results for target tasks of different sizes while making sure all source models are trained with sufficient data.

Influence of the Number of Source Models It is important to note that each task has a different optimal selection of m since the number of related tasks with positive transfer is different for each task. However, for the sake of simplification, we report the results with a fixed number of source models (with $m = 1, 3, 5$) for each task. In Figure 3 (Left), we observe that with all of the chosen m, we obtain better

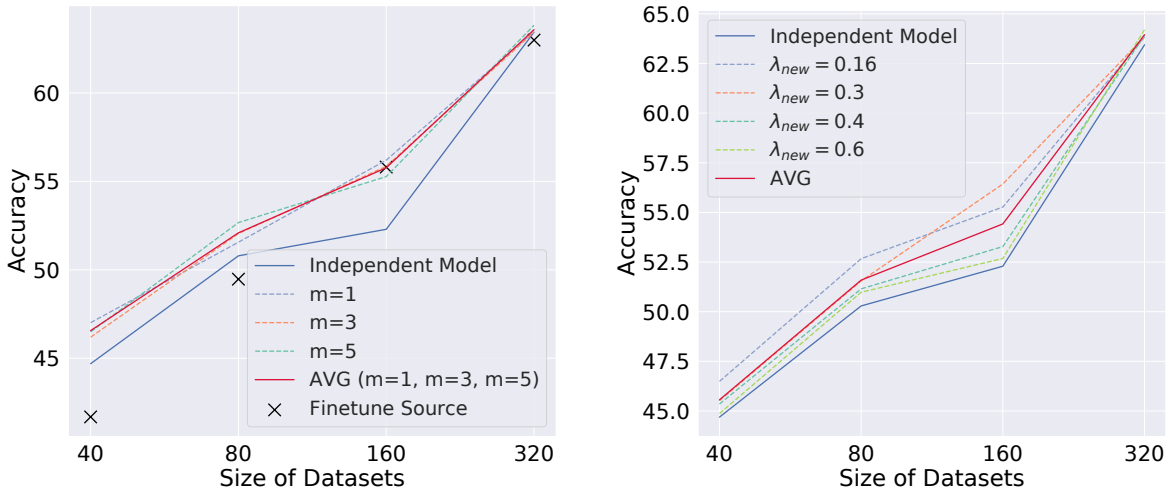

Figure 3: Left: Analysis of the influence of dataset size with respect to the number of source models adapted on `CIFAR100`. Right: Analysis of the influence of dataset size e with respect to different mixing weight initialization on `CIFAR100` ($m = 5$ for all the experiments). λ_{new} on the right panel represents the initial value of the new random initialized modules.

results than directly finetuning a source model and training from scratch. Moreover, a wider gap is observed between the independent model and the average performance of our module-mixing model with varying numbers of source models as the size of the datasets decreases, indicating that one potentially gets better performance gains from our method compared to training an independent model. Further, directly finetuning a source model may result in overfitting, especially when the dataset size is small. Alternatively, our model effectively reduces the chances of overfitting.

Influence of Different Mixing Weight Initialization We also show the effect of the mixing weights' initialization on overall performance. We consider all the source tasks equally, hence no extra information on which one is more important from the start. Hence, we initialize their mixing weights with equal values and only change the weight values of the new random initialized modules, *i.e.*, $\lambda_{new} = \lambda_{m+1,J+1}$. For simplicity, all layers share the same set of weights at initialization. In Figure 3 (Right), for $m = 5$, we observe that the average accuracy for different mixing weight initialization is always better than the independent model, albeit with particular initialization values one may get better performance than the average.

Office-31 `Office-31` consists of 31 categories of images originating from 3 domains: Amazon, DSLR and Webcam, including common objects in everyday office settings. Webcam consists of 795 low-resolution images with noise; the Amazon domain has 2817 images from online merchants; DSLR images are captured with a digital camera, and all 423 images are of high resolution and low noise. With this dataset, we show how our model behaves to covariate shift, when all the tasks have sufficient data and share the same labels.

In this context, Finetune Source becomes a more competitive baseline now that it does not suffer from serious overfitting issues. From Table 1, our model $m = 1$ guarantees that it has better if not equal performance compared to finetuning the same source model. Moreover, we achieve a 7.75% performance increase compared to Finetune Source when adapting to both two source models training with DC loss. We also see that on average, learning from more source models that are trained on a related domain yields better positive transfer, specifically, we observe a 2.78% increase from going from $m = 1$ to $m = 2$ when training with DC loss. Since MCW is designed as a fast-to-adapt and lightweight model that does not introduce too many trainable parameters, it loses its advantage in this setting when compared to other finetuning models, and even multi-source SVM when the source models are trained on closely related domains. As for cross-dataset soup and ensemble methods like model stacking, they work better when all the models are trained on the same datasets (or different splits of the same dataset).

Table 1: Accuracy comparison on `Office-31` in the white-box scenario. The highest accuracy is marked in bold. We also compare the results training with and without distance correlation loss for our model.

Method	$A, D \to W$	$A, W \to D$	$W, D \to A$	AVG
Model stacking	58.75	68.62	64.31	63.89
Cross-dataset soup	11.88	21.78	10.84	14.83
Multi-source SVM	90.11	81.82	84.98	85.64
MCW	64.84	62.12	76.79	67.92
DATE (SHOT)	60.74	52.74	20.43	44.63
DECISION	59.22	50.42	21.35	43.66
Independent	89.24	75.62	71.73	72.34
Finetune Source	91.25	88.24	82.69	87.39
$m = 1$ (w/o $\mathcal{L}_{DC}$)	**97.50**	88.24	91.52	92.42
$m = 2$ (w/o $\mathcal{L}_{DC}$)	95.00	**94.12**	91.17	93.43
$m = 1$ (w/ $\mathcal{L}_{DC}$)	95.00	90.20	91.87	92.36
$m = 2$ (w/ $\mathcal{L}_{DC}$)	**97.50**	**94.12**	**95.05**	**95.14**

Table 2: Accuracy comparison on `CTrL` in the white-box scenario. For our model, besides showing the effect of the distance correlation loss, we also show the importance of a hyperparameter grid search when the model pool is filled with models that are trained on very different source datasets.

Method	$m = 1$	$m = 3$	$m = 5$	AVG
Model stacking	42.53	34.02	34.31	36.95
Cross-dataset soup	47.73	43.83	39.43	43.66
Multi-source SVM	44.34	28.21	26.59	33.08
MCW	42.73	45.02	54.09	47.28
Independent	-	-	-	45.77
Finetune Source	-	-	-	54.26
DATE (SHOT) (knn)	42.45	43.72	42.84	43.00
DECISION (knn)	42.56	45.24	45.68	44.49
Multi-source SVM (knn)	**68.48**	56.96	48.33	57.92
MCW (knn)	66.59	**72.89**	**72.91**	**70.80**
Ours (w/o $\mathcal{L}_{DC}$)	56.40	55.13	52.09	54.54
Ours (w/$\mathcal{L}_{DC}$)	56.48	55.35	52.79	54.87
Ours (w/$\mathcal{L}_{DC}$ grid search)	67.26	70.24	68.71	68.73

CTrL The S^{long} stream from `CTrL` is a collection of 100 tasks created from five well-known computer vision datasets: `CIFAR10`, `CIFAR100`, `SVHN` (Netzer et al., 2011), `MNIST` (LeCun et al., 2010), and `Fashion-MNIST` (short as `FMNIST`) (Xiao et al., 2017). Each task in S^{long} is constructed by first randomly selecting a dataset, then five classes of the chosen dataset, and finally, a large task (containing 5000 training samples) or a small task (containing 25 training samples). During tasks $1 - 33$, the fraction of small tasks is 50%; this increases to 75% for tasks $34 - 66$, and to 100% for tasks $67 - 100$. More details of the dataset can be found in the Appendix A.2.2. In our experiments, we use the first 60 tasks as our starting pool of source models and gradually add newly trained models into the collection. We report the average test accuracy on the last 40 tasks. This is a challenging problem not only because it simulates a realistic setting that starts with a large collection of pre-trained tasks for repurposing, but also because the last 40 tasks have only 25 training samples each, which could cause serious overfitting problems. We perform a simple grid search on hyperparameters (learning rate and weight decay) and compare them to having the same settings for all tasks. Grid search and other experimental details are in the Appendix A.2. Results show that grid search is necessary when tasks come from very different datasets. In Table 2, we also show results of an extension of MCW and multi-source SVM with our k-NN source model sampler. The results show the effectiveness

Table 3: Accuracy comparison on `CIFAR100` in the black-box scenario. For our model, we also compare the results training with and without distance correlation loss.

Method	$m = 1$	$m = 3$	$m = 5$	API	AVG
Model stacking	71.32	70.97	70.36	-	70.88
Cross-dataset soup	39.71	37.17	35.93	-	37.60
Multi-src SVM	54.76	51.48	47.60	-	51.28
MCW	50.46	52.75	59.34	-	54.18
DINE	40.41	41.23	39.76	-	40.47
Independent	-	-	-	-	70.39
Ours (w/o $\mathcal{L}_{DC}$)	71.43	71.74	71.17	69.97	71.08
Ours (w/ $\mathcal{L}_{DC}$)	**72.03**	**71.82**	**71.46**	**71.80**	**71.78**

of our k-NN source model sampler, helping to achieve a 23.52% increase for MCW compared to learning from randomly sampled source models. However, we still suffer from the overfitting problem mainly on tasks created with `SVHN`, which explains the gap between our method and MCW with k-NN sampler.

5.2 Black-box Scenario

Results on `CIFAR100` (with all data in each task) are shown in Table 3. We also used k-NN to select source models for baselines. The results show that we can still benefit from such a simple approach. Additional experiment results on other datasets in this setting can be found in the Appendix A.6.6.

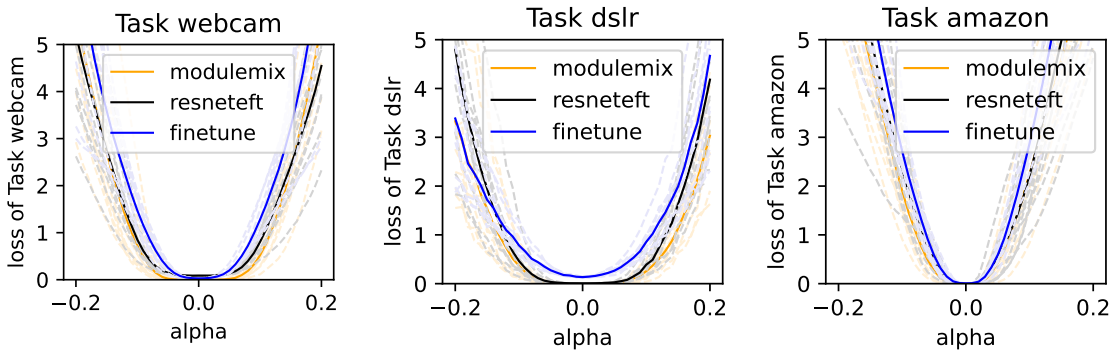

Figure 4: Comparison of weight loss landscapes. The dotted lines shows the loss created with the random directions given the trained model weights.

5.3 Insights Behind Module-Mixing Framework Design

Deng et al. (2021); Liebenwein et al. (2021); Cha et al. (2021); Wortsman et al. (2022) have shown that a model with a flatter loss landscape contributes to the generalization of models in domain adaptation and continual learning, thus is more robust to overfitting compared to directly finetune the source model. A similar idea as ours proposed in Wortsman et al. (2022) showed that by averaging weights of multiple finetuned models with the same initialization, the model's optimum falls in a flatter loss/error landscape with overall lower loss/error. Based on Li et al. (2017), the sharpness of loss landscapes correlates well with generalization error.

To show that the module-mixing framework has a better generalization ability and more robust to overfitting than directly finetuning source models, in Figure 4 we show the weight loss landscape of the independent model (resneteft), finetune, and our module-mixing model ($m = 1$) trained on all data from three domains in `Office31`. We follow the procedure from Deng et al. (2021) and create ten random directions given the

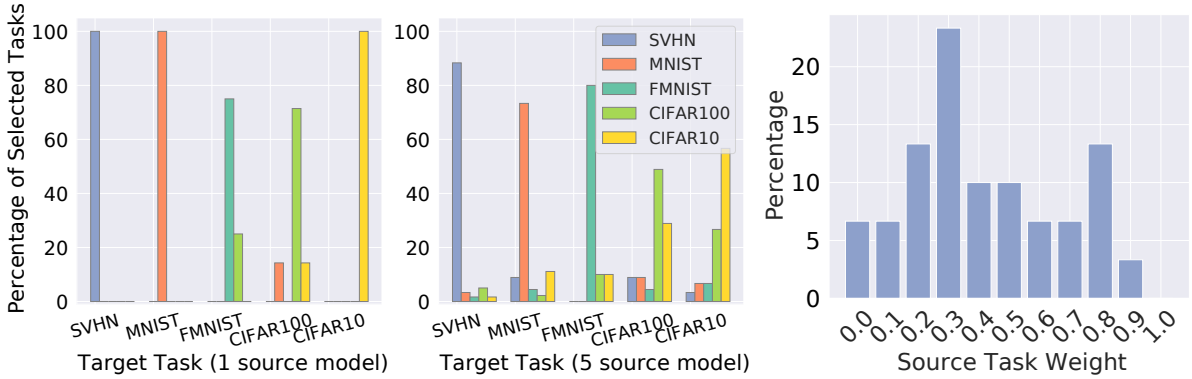

Figure 5: Selection of $m = 1$ (Left) and $m = 5$ (Middle) related tasks via k-NN. The x-axis shows the membership of the target tasks, while the y-axis shows the proportion of selected source tasks for each membership. Right: Module-mixing results for $m = 1$. The x-axis is the mixing weight value set for the source model and the y-axis is the percentage of tasks in `CIFAR100` for which the highest accuracy is obtained at a given source weight setting.

trained model weights. The solid line is the average over the results from the ten random directions. We can see from the plots that module-mixing model has a flatter loss landscape than the other methods, which explains why the model alleviates the effects of potential overfitting more than directly finetuning the model.

We also provide another plot for module-mixing under black-box setting with limited target data in the Appendix A.6.1. In the figure, we compare the module-mixing model with the Finetune Source model trained on webcam with only 20% of its data. Same conclusions can be drawn from this setting.

5.4 Ablation Study

We construct a module-mixing model with the top-1 source model selected via k-NN and a target model using only equation 4. In this setting, the mixing weight λ_1 for the source model features is set as a value in $\{0, 0.1, \ldots, 0.9, 1\}$, and only the target model's modules and the classification head are trained. Thus, an independent model is trained when $\lambda_1 = 0$, whereas only the classification head of a pre-trained source model is trained when $\lambda_1 = 1$. We test on our 30 `CIFAR100` tasks. For each task, one setting for λ_1 will achieve the highest test accuracy. Each bar in Figure 5 (Right) shows the percentage of tasks that reached peak accuracy at the given task weight setting. We see that the optimum results are mostly obtained (93%) when the mixing weight is non-zero for either source and target models, thus justifying our model design.

Table 4: Ablation Study on `CTrL` dataset when $m = 1$.

Method	Accuracy
Independent model	45.77
Random (only equation 4)	45.88
k-nearest neighbors (only 4)	55.68
Random (both equation 4 and equation 3)	46.61
k-nearest neighbors (both equation 4 and equation 3)	**56.40**

In Table 4, we examine the advantages of our source model selection and layer-wise module-mixing design on `CTrL` (with $m = 1$ for adaptation). Alternatively, we consider selecting source models at random (instead of k-NN) and only using the convex combination of features in equation 4 (instead of both features and module parameters). We see that with random selection and only feature aggregation, we get an insignificant gain of 0.11% relative to the independent model, while with k-NN selection and only feature aggregation, the gain is higher at 1%. Importantly, the complete approach using equation 4 and equation 3 with k-nearest neighbors model ewselection yields a substantial 9.79% gain relative to that with random source model selection.

Table 5: `TinyImageNet` in the white-box scenario. Selected task names are colored for clarity.

		Task1	Task2	Task3	Task4	Task5	Task6	AVG
$m = 2$	Independent model	34.55	32.64	35.07	32.47	33.16	33.25	33.52
	Finetune source	34.38	31.77	33.51	32.73	33.59	32.90	33.15
	Model stacking	31.03	31.86	35.33	31.43	30.64	28.91	31.53
	Cross-dataset soup	28.65	32.20	**37.24**	**34.20**	33.51	32.73	33.09
	Multi-source SVM	28.64	25.44	25.76	24.72	27.04	23.76	25.89
	MCW	21.76	22.32	22.40	21.68	22.16	21.68	22.00
	DATE (SHOT)	21.24	22.76	21.49	22.36	21.66	21.90	21.90
	DECISION	21.34	21.93	21.36	21.58	22.46	20.96	21.60
$m = 1$	LogME	35.16	**34.80**	35.36	32.90	35.07	33.28	34.43
	Selected task	Task2	Task1	Task2	Task5	Task4	Task5	
	LEEP	35.16	33.59	35.36	32.90	35.07	34.08	34.36
	Selected task	Task2	Task3	Task2	Task5	Task4	Task3	
	Ours	35.16	33.59	**36.26**	32.90	**35.07**	**35.40**	**34.73**
	Selected task	Task2	Task3	Task4	Task5	Task4	Task4	
$m = 2$	LogME	35.00	34.08	33.68	33.52	34.96	33.36	34.10
	Selected tasks	Task2 Task6	Task1 Task3	Task2 Task4	Task3 Task5	Task4 Task6	Task4 Task5	
	LEEP	35.00	34.08	33.68	33.52	34.96	34.96	34.37
	Selected tasks	Task2 Task6	Task1 Task3	Task2 Task4	Task3 Task5	Task4 Task6	Task3 Task5	
	Ours	**36.40**	34.08	33.68	33.52	34.96	33.84	34.41
	Selected tasks	Task2 Task3	Task1 Task3	Task2 Task4	Task3 Task5	Task4 Task6	Task1 Task4	

To show that our method also applies to more complex tasks, we provide an experiment on `Tiny-ImageNet`. We use a randomly initialized ResNet-18 as the backbone. The purpose of this setting is to show that the improvement in performance of our model does not rely on a powerful and closely related backbone model since `Tiny-ImageNet` is a subset of `ImageNet`. We create 6 tasks with 200 classes in the dataset, with 50 classes for each task. Task 1 through 5 have overlapping classes, while Task 6 has no overlapping with other tasks. Task details are presented in Table 17 and 18, where tasks with overlapping classes have the same color-coding. Since the tasks in this setting all have sufficient training data, the newly initialized task-specific modules for the target task are given higher initial module-mixing weights (0.9) to encourage the learning of target task-specific knowledge. For each task, we search for top-m most similar tasks in the other 5 tasks. In Table 5, we also provide the source model selection results, shown in "selected task". The task names have the same colors as in the header of the table to provide more clarity. We observe that tasks with overlapping classes are always picked for each task. For instance, Task 2 overlaps with Task 1 and 3, so when we build our module-mixing model with the top-2 source models to reuse, these tasks are selected.

Furthermore, we compare with LEEP (Nguyen et al., 2020) and LogME (You et al., 2021) in identifying the most transferable source models. Different model transferability evaluation methods are first used to select the top-m transferable models, and then the selected source models are used to build our module-mixing models to transfer to each target task. As is shown in Table 5, our selection method's performance is on par with the baselines, while having a slight edge over them. We also provided the average results and standard deviation over three runs comparing our method to LogME and LEEP in the Appendix A.6.2.

Moreover, we show how well the k-NN algorithm selects source models with `CTrL`. In Figure 5, we show selection results with $m = 1$ (Left) and $m = 5$ (Middle). Each bar represents the proportion of selected tasks from each dataset for the 40 tasks. When $m = 1$, target tasks created with `SVHN`, `MNIST`, and `CIFAR10` always pick source models trained from the same datasets. Also, we only have 4 `FMNIST` tasks in the experiments, which explains why the selector picks a high 25% of `CIFAR10` source models. As m increases to 5, tasks

with classes from SVHN, MNIST, FMNIST still almost always pick models created from the same dataset. For CIFAR10 and CIFAR100, since they both contain images of natural objects, they are more similar.

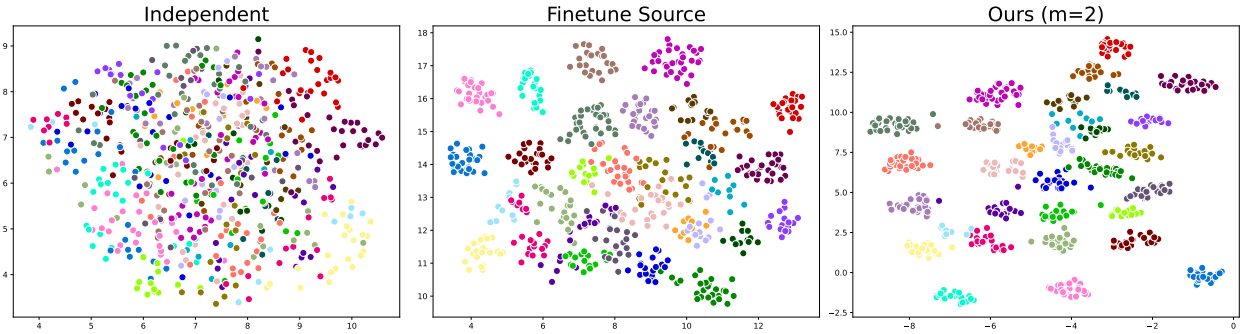

Figure 6: Umap plots on the Webcam domain: each color represents the same label across all three panels.

5.4.1 UMAP Plots of Office-31 Domains

Finally, we validate if our model learned meaningful data representations by visualizing the features of training data extracted from the feature extractor of models Independent, Finetune Source and our Module-mixing model, respectively, using UMAP (McInnes et al., 2018) as shown in Figure 6. Learned features with our model are relatively more clustered and separated compared to that for the other two methods. We also provide quantitative results examining the separation of the features in Appendix A.6.4. For the DSLR domain, the same conclusions can be drawn, as shown in Figure 12 in the Appendix A.6.3.

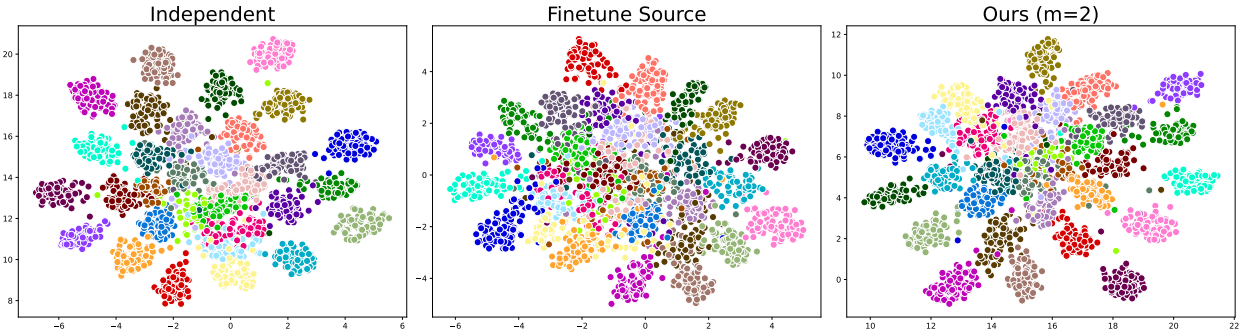

Figure 7: Umap plots on the Amazon Domain with training data. Each unique color represents the same label across all three panels.

For the Amazon domain, in Figure 7, we create scatter plots of the learned UMAP embeddings of the training data, and color samples (points) according to their ground truth labels. All three plots in Figure 7 have good clustering patterns. However, if we color-code the training data points by their respective predicted probabilities of outcomes as in Figure 8, we notice that models Independent and Finetune Source are overconfident about their predictions, while our method also has the effect of preventing the model from predicting the labels too confidently during training. In Figure 9, we visualize the extracted features of the test data after UMAP embedding. We see that the Independent model is less confident about the predictions. While Finetune Source has higher predicted probabilities on the test set, our model has relatively better clustering and is more certain about the predictions compared to Finetune Source. Besides the loss weight landscape analysis, this observation can be seen as another indication that our model has better generalization.

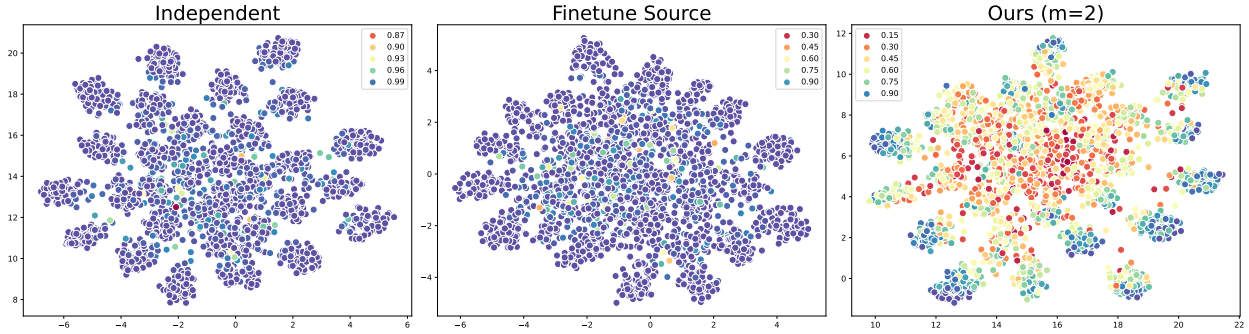

Figure 8: Umap plots on the Amazon domain with training data. The colors represent the predicted probability. Independent and Finetune Source have high overall probability for their predictions, while our method does not. This indicates that our method has better generalization.

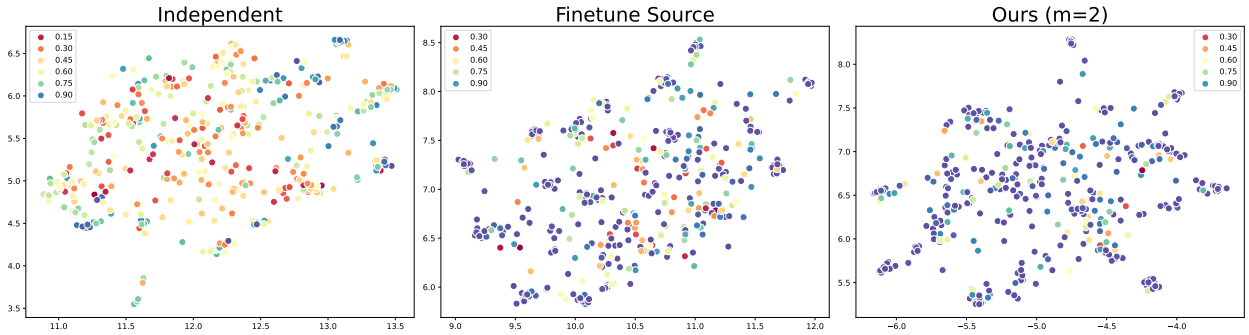

Figure 9: Umap plots on the Amazon domain with testing data. The colors represent the predicted probability. Independent and Finetune Source have lower overall probability for their predictions compared to our method's predictions.

6 Conclusion

In this work, we studied an under-explored source-free supervised transfer learning scenario. The proposed framework can efficiently search for related models to adapt to and also can be applied to white-box and black-box source models. To test our module-mixing model and encourage more work in this field, we not only experimented on a classic domain adaptation dataset `Office-31`, but also crafted new datasets based on existing benchmarks, including `CIFAR100`, `CTrL` and `Tiny-ImageNet`. We showed promising improvements compared to the baselines and examined the properties of each part of our framework.

Nevertheless, we acknowledge some limitations of the proposed method. So far, we initialize the model's mixing weights with equal values or do a grid search for the best mixing weights. However, it will be interesting to study how to initialize the mixing weights according to different source models' transferability to the target task. Moreover, we require all models to have task-specific modules of same sizes for the white-box scenario. Therefore, another interesting future work is to see how to build layer-wise module-mixing modules when the task-specific modules are not of the same sizes.

References

Sk. Miraj Ahmed, Dripta S. Raychaudhuri, S. Paul, Samet Oymak, and Amit K. Roy-Chowdhury. Unsupervised multi-source domain adaptation without access to source data. *2021 IEEE/CVF Confer-*

ence on Computer Vision and Pattern Recognition (CVPR), pp. 10098–10107, 2021a. URL https://api.semanticscholar.org/CorpusID:233025033.

Sk Miraj Ahmed, Dripta S Raychaudhuri, Sujoy Paul, Samet Oymak, and Amit K Roy-Chowdhury. Unsupervised multi-source domain adaptation without access to source data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10103–10112, 2021b.

Jordan T. Ash and Ryan P. Adams. On the difficulty of warm-starting neural network training. *ArXiv*, abs/1910.08475, 2019.

Junbum Cha, Sanghyuk Chun, Kyungjae Lee, Han-Cheol Cho, Seunghyun Park, Yunsung Lee, and Sungrae Park. Swad: Domain generalization by seeking flat minima. In *Neural Information Processing Systems*, 2021. URL https://api.semanticscholar.org/CorpusID:235367622.

Weijie Chen, Luojun Lin, Shicai Yang, Di Xie, Shiliang Pu, Yueting Zhuang, and Wenqi Ren. Self-supervised noisy label learning for source-free unsupervised domain adaptation. *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10185–10192, 2021.

Yulai Cong, Miaoyun Zhao, Jianqiao Li, Sijia Wang, and Lawrence Carin. Gan memory with no forgetting. In *NeurIPS*, volume 33, pp. 16481–16494, 2020.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. doi: 10.1109/TIT.1967.1053964.

Danruo Deng, Guangyong Chen, Jianye Hao, Qiong Wang, and Pheng-Ann Heng. Flattening sharpness for dynamic gradient projection memory benefits continual learning. *Advances in Neural Information Processing Systems*, 34:18710–18721, 2021.

Shibhansh Dohare, A Rupam Mahmood, and Richard S Sutton. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021.

Jiahua Dong, Zhen Fang, Anjin Liu, Gan Sun, and Tongliang Liu. Confident anchor-induced multi-source free domain adaptation. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 2848–2860. Curran Associates, Inc., 2021.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020.

Yuqi Fang, Pew-Thian Yap, Weili Lin, Hongtu Zhu, and Mingxia Liu. Source-free unsupervised domain adaptation: A survey. *arXiv preprint arXiv:2301.00265*, 2022.

Yuqi Fang, Pew-Thian Yap, Weili Lin, Hongtu Zhu, and Mingxia Liu. Source-free unsupervised domain adaptation: A survey. *Neural Networks*, pp. 106230, 2024.

Lan-Zhe Guo, Zhi Zhou, Yu-Feng Li, and Zhi-Hua Zhou. Identifying useful learnwares for heterogeneous label spaces. In *International Conference on Machine Learning*, pp. 12122–12131. PMLR, 2023.

Zhongyi Han, Zhiyan Zhang, Fan Wang, Rundong He, Wan Su, Xiaoming Xi, and Yilong Yin. Discriminability and transferability estimation: a bayesian source importance estimation approach for multi-source-free domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 7811–7820, 2023.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Doll'ar, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15979–15988, 2021.

Samitha Herath, Mehrtash Harandi, and Fatih Porikli. Learning an invariant hilbert space for domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3845–3854, 2017.

Yunzhong Hou and Liang Zheng. Visualizing adapted knowledge in domain transfer. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13819–13828, 2021.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, 2019.

Aapo Hyvarinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE transactions on Neural Networks*, 10(3):626–634, 1999.

Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000.

Pavel Izmailov, Dmitrii Podoprikhin, T. Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in NLP. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 264–271, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `https://aclanthology.org/P07-1034`.

Di Jin, Bunyamin Sisman, Hao Wei, Xin Dong, and Danai Koutra. Deep transfer learning for multi-source entity linkage via domain adaptation. *Proc. VLDB Endow.*, 15:465–477, 2021.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *arXiv*, pp. 32–33, 2009.

Brian Kulis, Kate Saenko, and Trevor Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *CVPR 2011*, pp. 1785–1792. IEEE, 2011.

V. Kurmi, Venkatesh K. Subramanian, and Vinay P. Namboodiri. Domain impression: A source data free domain adaptation method. *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 615–625, 2021.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Joshua Lee, Prasanna Sattigeri, and Gregory Wornell. Learning new tricks from old dogs: Multi-source transfer learning from pre-trained networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *ArXiv*, abs/1712.09913, 2017. URL `https://api.semanticscholar.org/CorpusID:3693334`.

Junnan Li, Ziwei Xu, Yongkang Wong, Qi Zhao, and M. Kankanhalli. Gradmix: Multi-source transfer across domains and tasks. *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 3008–3016, 2020a.

Keqiuyin Li. *Deep Neural Networks for Multi-Source Transfer Learning*. PhD thesis, University of Technology Sydney, 2022.

Rui Li, Qianfen Jiao, Wenming Cao, Hau-San Wong, and Si Wu. Model adaptation: Unsupervised domain adaptation without source data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9641–9650, 2020b.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Jian Liang, D. Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *International Conference on Machine Learning*, 2020.

Jian Liang, Dapeng Hu, Jiashi Feng, and Ran He. Dine: Domain adaptation from single and multiple black-box predictors. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7993–8003, 2021a. URL `https://api.semanticscholar.org/CorpusID:244800743`.

Jian Liang, Dapeng Hu, Ran He, and Jiashi Feng. Distill and fine-tune: Effective adaptation from a black-box source model. *arXiv preprint arXiv:2104.01539*, 1(3), 2021b.

Lucas Liebenwein, Ramin M. Hasani, Alexander Amini, and Daniela Rus. Sparse flows: Pruning continuous-depth models. In *Neural Information Processing Systems*, 2021. URL `https://api.semanticscholar.org/CorpusID:235623962`.

Xiaobing Liu and Shiliang Zhang. Graph consistency based mean-teaching for unsupervised domain adaptive person re-identification. In *International Joint Conference on Artificial Intelligence*, 2021.

Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging. *ArXiv*, abs/2111.09832, 2021.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *NIPS*, 2011.

Cuong Nguyen, Tal Hassner, Matthias Seeger, and Cedric Archambeau. Leep: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning*, pp. 7294–7305. PMLR, 2020.

Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE transactions on neural networks*, 22(2):199–210, 2010.

Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1406–1415, 2019.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *ArXiv*, 2019.

Alexandre Ram'e, Kartik Ahuja, Jianyu Zhang, Matthieu Cord, Léon Bottou, and David Lopez-Paz. Model ratatouille: Recycling diverse models for out-of-distribution generalization. In *International Conference on Machine Learning*, 2022. URL `https://api.semanticscholar.org/CorpusID:254877458`.

Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*, pp. 213–226. Springer, 2010.

Baochen Sun and Kate Saenko. Subspace distribution alignment for unsupervised domain adaptation. In *BMVC*, volume 4, pp. 24–1, 2015.

G'abor J. Sz'ekely, Maria L. Rizzo, and Nail K. Bakirov. Measuring and testing dependence by correlation of distances. *Annals of Statistics*, 35:2769–2794, 2007.

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *NIPS*, 2017.

Xinyi Tong, Xiangxiang Xu, Shao-Lun Huang, and Lizhong Zheng. A mathematical framework for quantifying transferability in multi-source transfer learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021.

Anh T Tran, Cuong V Nguyen, and Tal Hassner. Transferability and hardness of supervised classification tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1395–1405, 2019.

Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. *arXiv preprint arXiv:2012.12631*, 2020.

Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5018–5027, 2017.

Vinay Kumar Verma, Kevin J Liang, Nikhil Mehta, Piyush Rai, and Lawrence Carin. Efficient feature transformations for discriminative and generative continual learning. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13860–13870, 2021.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *ArXiv*, abs/2203.05482, 2022.

Yanru Wu, Jianning Wang, Weida Wang, and Yang Li. H-ensemble: An information theoretic approach to reliable few-shot multi-source-free transfer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 15970–15978, 2024.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Lin Xiong, Mao Ye, Dan Zhang, Yan Gan, Xue Li, and Yingying Zhu. Source data-free domain adaptation of object detector through domain-specific perturbation. *International Journal of Intelligent Systems*, 36 (8):3746–3766, 2021.

Ruijia Xu, Ziliang Chen, Wangmeng Zuo, Junjie Yan, and Liang Lin. Deep cocktail network: Multi-source unsupervised domain adaptation with category shift. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3964–3973, 2018.

Yi Yao and Gianfranco Doretto. Boosting for transfer learning with multiple sources. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pp. 1855–1862. IEEE, 2010.

Li Yi, Gezheng Xu, Pengcheng Xu, Jiaqi Li, Ruizhi Pu, Charles Ling, Allan Mcleod, and Boyu Wang. When source-free domain adaptation meets learning with noisy labels. *ArXiv*, abs/2301.13381, 2023. URL `https://api.semanticscholar.org/CorpusID:256416103`.

Kaichao You, Yong Liu, Jianmin Wang, and Mingsheng Long. Logme: Practical assessment of pre-trained models for transfer learning. In *International Conference on Machine Learning*, pp. 12133–12143. PMLR, 2021.

Xingjian Zhen, Zihang Meng, Rudrasis Chakraborty, and Vikas Singh. On the versatile uses of partial distance correlation in deep learning. *Computer vision - ECCV ... : ... European Conference on Computer Vision : proceedings. European Conference on Computer Vision*, 13686:327–346, 2022.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109:43–76, 2019.

A Appendix

A.1 Efficient Feature Transformation

Verma et al. (2021) proposed efficient feature transformation (EFT) for both 2D convolutional layers and fully connected layers. In the experiments, we apply EFT to convolutional layers in a ResNet18 model. As mentioned in the Background section, given the groupwise convolutional kernels $\omega^s \in \mathbb{R}^{M\times Q\times a}$ and the corresponding $\frac{K}{a}$ groups of feature maps F, each group of feature maps $F^s_{ai:(ai+a-1)}$ is convolved with kernels $[\omega^s_{i,1}, \cdots, \omega^s_{i,a}]$ to form a new group of feature maps $H^s_{ai:(ai+a-1)} \in \mathbb{R}^{M\times Q\times a}$. The formulation of which is shown below:

$$H^s_{ai:(ai+a-1)} = [\omega^s_{i,1} * F_{ai:(ai+a-1)}| \cdots | \omega^s_{i,a} * F_{ai:(ai+a-1)}], i \in \{0, \cdots, \frac{K}{a} - 1\}, \tag{7}$$

where $|$ is the concatenation operation and i indicates the group of feature maps. Then, all the $H^s_{ai:(ai+a-1)}$ are concatenated to form H^s. Same procedures can be applied with the pointwise convolutional kernels $\omega^d \in \mathbb{R}^{M\times Q\times b}$ and the corresponding $\frac{K}{b}$ groups of feature maps F:

$$H^s_{bi:(bi+b-1)} = [\omega^s_{i,1} * F_{bi:(bi+b-1)}| \cdots | \omega^s_{i,b} ss * F_{bi:(bi+b-1)}], i \in \{0, \cdots, \frac{K}{b} - 1\}. \tag{8}$$

Then concatenate $H^d_{ai:(ai+a-1)}$ to form H^d. Combining H^d and H^s produces the final feature maps $H = H^s + \gamma H^d$, where $\gamma \in \{0, 1\}$, and in this paper we set $\gamma = 1$.

By setting $a \ll K$ and $b \ll K$, the amount of trainable parameters per task is substantially reduced. For instance, using a ResNet18 backbone, $a = 8$, and $b = 16$ results in 449k parameters per new task, which is 3.9% the size of the backbone. As empirically demonstrated in Verma et al. (2021), EFT makes model growth efficient in continual learning settings while preserving the remarkable representation power of ResNet.

A.2 Dataset and Implementation Details

Hyperparamter settings For experiments in the *white-box* setting, *i)* with `Office-31`, all the models are trained for 100 epochs with batch size 128; the training starts with a learning rate of 0.001 and weight decay $1e^{-5}$, and the learning rate decays by 0.1 after the first 50 epochs. *ii)* For `CIFAR100`, all the models are trained for 150 epochs with batch size 128, weight decay of $1e^{-5}$, and a learning rate starts at 0.01 and decays by 0.1 after 100 epochs, except for the tasks with dataset size 40, which is trained with weight decay $1e^{-4}$. For tasks with dataset size 40, we also add random horizontal flip data augmentation to all the experiments to further reduce overfitting. *iii)* For `CtrL`, all the experiments are trained for 100 epochs with batch size 16. We perform a grid search on learning rate in $\{0, 0.01, 0.003\}$ and weight decay in $\{0, 1e^{-4}, 1e^{-5}\}$, and compare the results to having the same learning rate of 0.01 and reduced by 0.1 after 50 epochs, with weight decay $1e^{-4}$. We also add random horizontal flip augmentation, as in the original paper Veniat et al. (2020), to alleviate overfitting.

For experiments in the *black-box* setting, the learning rate is set to 0.001 and reduced by 0.1 after 50 epochs, batch size is 128, and weight decay is $1e^{-5}$ for all datasets. We adopt the scikit-learn package implementation of FastICA to reduce the dimensionality of the feature outputs obtained from the source models.

Mixing weights settings In Tables 1, 2, 3, 4, and 5, uniform initialization is set for the mixing weights in each layer. In the Additional Experimental Results section, results with different mixing weight initialization on `Office-31` data can be found.

A.2.1 Details on CIFAR100

`CIFAR100` has 100 classes containing 600 images per class, which is further split into 500 training images and 100 test images for each class. We construct a list of 40 tasks $[1, 2, \cdots, 40]$ as in Table 14 by evenly splitting

each class in `CIFAR100` into 300 images, which means each class in the 40 tasks has 250 training images and 50 test images. For each task, we randomly select a subset of size $e = \{40, 80, 160, 320\}$ out of the training images to train the module-mixing model, with 10% of the selected training images observed as a validation set. However, for training with different subsets, the test set remains the same, which contains 250 images.

A.2.2 Details on CTrL

Each task in the S_{long} dataset consists of training, validation, and test datasets. All images have been rescaled to 32x32 pixels in RGB color format, and normalized per-channel using statistics computed on the training set of each task. Data augmentation is performed by using random crops (4 pixels padding and 32x32 crops) and random horizontal flips.

The details of the tasks used in this paper can be found in Tables 15 and 16. Note that though we used the same random seed (343008097) to generate the dataset as in the original paper, the generated tasks with our hardware have different classes and dataset sizes from the original paper Veniat et al. (2020).

Hyperparameter grid search In the experiment shown in Table 2, learning rates $\{10^{-2}, 10^{-3}\}$, weight decay strengths $\{0, 10^{-5}, 10^{-4}\}$ and weight initializations $\lambda_{new} \in \{0.001, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.997\}$ are considered (0.001 and 0.997 are used to avoid setting weights to zero). For each λ_{new}, other mixing weights in the same layer are set to a same value $(1 - \lambda_{new})/m$, with m being the number of selected source models. We select the hyperparameter combination that produces the best validation accuracy.

A.2.3 Details on Office31

For each domain in Office31, we randomly select 80% of its data as training set, 10% as validation set, and 10% as test set. Data augmentation is performed by using random crops and random horizontal flips. All images have been rescaled to 256×256 and then cropped to 224×224.

A.3 Ablation Study on the a/b Parameter Settings for EFT

a=8 and b=16 are the optimal parameter choices according to the original EFT paper Verma et al. (2021) for `CIFAR100`. We adopted them for our experiments on `CIFAR100` and `Office-31`, where the number of samples for each task is relatively sufficient.

We set $a = 2$ and $b = 1$ for `CtrL` to reduce the overfitting problem on training source models with limited data since most of the source tasks have very limited data, we reduce the number of training parameters accordingly.

We also tried different a/b settings for `CtrL`, and $a = 2/b = 1$ gives the best performance for independent models. Below are some examples for training independent models with different a/b settings on tasks 60 to 100, which all have a limited data size. Weight decay is set to $1e-4$ and learning rate 0.01 for each setting, and all the experiments are run with the same three random seeds.

Table 6: Influence of the number of source models on performance for `CIFAR100`.

	a=1/b=1	a=2/b=1	a=1/b=2	a=2/b=2
Average test accuracy with three random seeds	43.2/44.8/43.9	47.2/46.1/45.9	44.4/45.3/44.9	46.1/45.5/47.2
Avg of three runs	44.0	**46.4**	44.9	46.3

A.4 Computational Complexity Analysis for Source Model Selection

The computational cost of model selection lies in two parts, one comes from extracting features using all the source models, the other comes from k-NN selection.

Time comsumption for feature extraction depends on the complexity of the source models and the number of points used for model selection. Since the source models' complexity is predetermined, we can reduce the number of points used for model selection when the number of candidate models is very large.

Given its non-parametric property, k-NN selection is relatively computational efficient. However, here we provide its' time/space complexity. Given

- n: number of points used for model selection
- d: feature dimensionality
- k: number of neighbors that we consider for voting

The time and space complexity for k-NN are

- Training time complexity: O(1)
- Training space complexity: O(1)
- Prediction time complexity: O(k * n * d)
- Prediction space complexity: O(1)

There is no need for training since all computation is done during prediction. To shorten searching time, two main things can be done. (since feature dimensionality d is predetermined by the source models)

Reduce the value of n We can use a small validation set instead of all the points in the task for model selection. This is also what we did with all the experiments in the paper.

Reduce the value of k Although using a larger k improves the accuracy of model selection, it can increase the time for searching through the model pool. We generally prefer a smaller k value. However, we need to keep in mind the accuracy / efficiency trade-off. Luckily, we found that $k = 5$ (or other relatively smaller k value) can already provide a good performance without sacrificing computational cost.

A.5 Ablation Study on the Setting of k for k-NN

We study how different choices of k for k-NN algorithm influence the selection results for source models using the `CTrL` dataset. We plot the membership selection results in Figure 10. We can see that the selected source models are more related to the target tasks with a larger value for k.

A.6 Additional Experimental Results

A.6.1 Weight Loss Landscape Plot under Limited Data Setting

In Figure 11, we show the weight loss landscape of module-mixing under black-box setting with limited target data. In the figure, we compare the module-mixing model with the Finetune Source model trained on webcam with only 20% of its data.

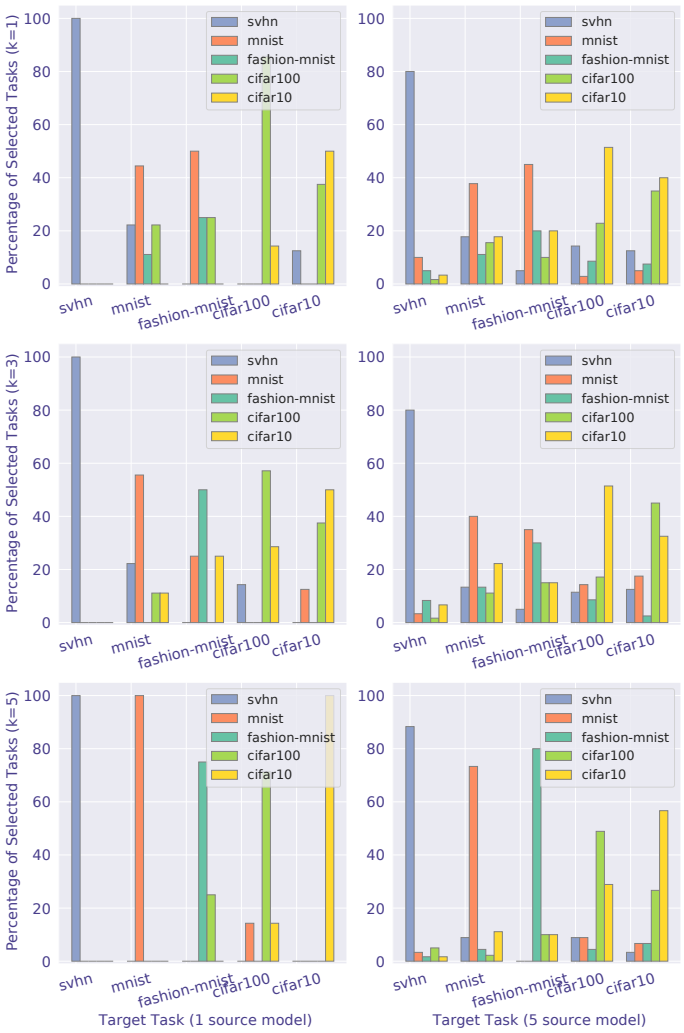

Figure 10: Ablation Study on the different setting of k for k-NN source model selection. From top to bottom, the results are for when $k = 1$, $k = 3$ and $k = 5$.

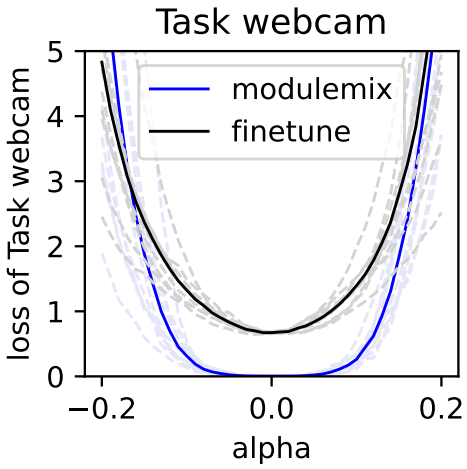

Figure 11: Weight Loss Landscape Plot under Limited Data Setting

A.6.2 Comparison to Other Model Transferability Methods

In Table 7, we show the average and standard deviation over three runs for all three methods (with the same random seeds) to select the most transferable models.

Table 7: `TinyImageNet` in the white-box scenario.

		Task1	Task2	Task3	Task4	Task5	Task6	AVG
$m = 1$	LogME	34.96 ± 0.52	$\mathbf{34.47} \pm 0.50$	35.42 ± 0.31	33.04 ± 0.15	34.79 ± 0.15	33.79 ± 0.54	34.41
	LEEP	35.20 ± 0.33	33.52 ± 0.23	35.42 ± 0.31	33.04 ± 0.15	34.79 ± 0.15	34.12 ± 0.34	34.35
	Ours	35.20 ± 0.33	34.06 ± 0.55	$\mathbf{36.33} \pm 0.27$	33.04 ± 0.15	$\mathbf{35.06} \pm 0.22$	$\mathbf{35.56} \pm 0.25$	**34.88**
$m = 2$	LogME	35.21 ± 0.20	34.28 ± 0.27	33.49 ± 0.24	33.61 ± 0.09	34.94 ± 0.25	33.83 ± 0.43	34.22
	LEEP	35.21 ± 0.20	34.28 ± 0.27	33.49 ± 0.24	33.61 ± 0.09	34.94 ± 0.25	35.02 ± 0.32	34.43
	Ours	$\mathbf{36.22} \pm 0.34$	34.28 ± 0.27	33.49 ± 0.24	33.61 ± 0.09	34.94 ± 0.25	34.06 ± 0.20	34.43

A.6.3 Additional Umap Plots for Office DSLR Domain

In Figure 12, learned features with our model are relatively more clustered and separated compared to that for the other two methods.

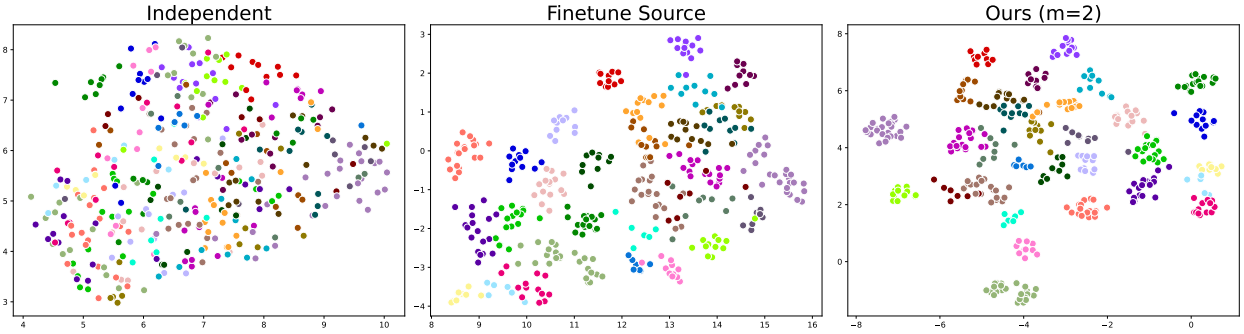

Figure 12: Umap plots on the DSLR domain with training data. Each unique color represents the same label across all three plots.

A.6.4 Quantitative Analysis on Feature Space

With the features of training data of each domain extracted from the feature extractor of model Independent, Finetune Source, and our module-mixing module (when $m = 2$), we fit KNN models respectively for each feature set. Then we acquire the predicting score with the features of testing set with the trained KNN models respectively. Since the KNN scores are all relatively high and close for the three different methods, we run the experiment for three times and report in Table 8 the average and standard deviation. Our model achieves the highest average KNN score on testing feature set, thus showed the quality of learned features from the quantitative perspective.

A.6.5 White-box Scenario

Office-31 We show the results with different mixing weight initialization in Table 9. λ_{new} is initialized with $[0.01, 0.4, 0.5, 0.6, 0.99]$ for both $m = 1$ and $m = 2$.

Table 8: KNN scores for Office domains.

	Independent	Finetune Source	Ours (m=2)
Webcam	0.975 ± 0.010	0.971 ± 0.012	$\mathbf{0.983} \pm 0.012$
Amazon	0.920 ± 0.006	0.909 ± 0.012	$\mathbf{0.940} \pm 0.017$
Dslr	0.961 ± 0.027	0.954 ± 0.009	$\mathbf{0.967} \pm 0.009$

Table 9: `Office-31` in White-box scenario with different mixing weight initialization.

Method	$A, D \to W$	$A, W \to D$	$W, D \to A$	AVG
Independent model	69.61	75.62	71.73	72.34
Finetune Source	91.25	88.24	82.69	87.39
m=1 $\lambda_{new} = 0.01$	78.25	76.47	94.35	83.02
m=1 $\lambda_{new} = 0.4$	96.25	90.20	92.58	**93.01**
m=1 $\lambda_{new} = 0.5$	**97.50**	88.24	91.52	92.42
m=1 $\lambda_{new} = 0.6$	95.00	88.24	85.87	89.70
m=1 $\lambda_{new} = 0.99$	92.50	94.12	68.20	84.94
m=2 $\lambda_{new} = 0.01$	81.25	64.71	**96.47**	80.81
m=2 $\lambda_{new} = 0.4$	93.75	92.16	90.81	92.24
m=2 $\lambda_{new} = 0.5$	92.50	**94.12**	85.16	90.59
m=2 $\lambda_{new} = 0.6$	96.26	92.16	83.75	90.72
m=2 $\lambda_{new} = 0.99$	93.75	90.20	74.56	86.17

CIFAR100 For a closer look at the results shown in Figures 3, we also present the results in Tables 10 and 11, respectively.

Table 10: Influence of the number of source models on performance for `CIFAR100`.

	320	160	80	40
Independent model	63.44±0.54	52.29±0.61	50.79±0.79	44.69±0.93
Finetune Source	62.99±0.60	55.81±0.54	49.47±0.80	41.68±0.99
$m = 1$	63.47±0.55	**56.20**±0.78	51.56±0.72	**47.01**±0.99
$m = 3$	63.44±0.50	55.85±0.63	52.05±0.65	46.19±0.94
$m = 5$	**63.83**±0.43	55.27±0.55	**52.67**±0.76	46.49±1.43
AVG	63.58	55.77	52.09	46.56

A.6.6 Black-box Scenario

In Table 12, we also provide results on `CTrL` under black-box setting for source models. A LeNet-5 model is used as target model. Random horizontal flip and vertical flip data augmentation is used to increase the size of training set to 150 and validation set to 90, a batch size of 128 is used during training in this setting.

A.6.7 Additional Results on OfficeHome Dataset

The Office-Home dataset consists of images from 4 different domains: Artistic images, Clip Art, Product images and Real-World images. Each domain contains images of 65 object categories found typically in Office and Home settings. For each domain in OfficeHome, we randomly select 80% of its data as training set, 10% as validation set, and 10% as test set. Data augmentation is performed by using random crops and random horizontal flips. All images have been rescaled to 256×256 and then cropped to 224×224. The results for OfficeHome can be found in Table 13. Our model achieves the highest performance in most cases.

Table 11: Influence of different mixing weight initialization on `CIFAR100` when $m = 5$.

	320	160	80	40
Independent model	63.44±0.54	52.29±0.61	50.79±0.79	44.69±0.93
$\lambda_{new} = 0.16$	63.83±0.43	55.27±0.55	**52.67**±0.76	**46.49**±1.43
$\lambda_{new} = 0.3$	63.84±0.56	**56.43**±0.72	51.55±0.67	45.52±1.22
$\lambda_{new} = 0.4$	63.91±0.62	53.28±0.66	51.15±0.77	45.35±0.99
$\lambda_{new} = 0.6$	**64.20**±0.58	52.68±0.51	50.98±0.81	44.87±1.15
AVG	63.95	54.42	51.59	45.58

Table 12: `CTrL` in black-box scenario.

Method	$m = 1$	$m = 3$	$m = 5$	API	AVG
Independent	-	-	-	-	45.14
Finetune Source	-	-	-	-	46.31
Ours (w/o $\mathcal{L}_{DC}$)	47.12	47.23	**47.42**	46.40	**47.04**
Ours (w/ $\mathcal{L}_{DC}$)	**47.33**	**47.26**	46.83	**46.41**	46.96

Table 13: `OfficeHome` in white-box scenario.

Method	$A, C, R \to P$	$A, C, P \to R$	$C, R, P \to A$	$A, R, P \to C$	AVG
Model stacking	56.77	36.67	22.56	48.38	41.10
Model soup	15.48	12.80	10.63	14.91	13.46
Multi-src SVM	69.22	42.85	24.68	56.72	48.37
MCW	68.42	43.74	22.37	52.37	46.73
DATE	59.34	38.43	21.47	45.43	41.17
DECISION	53.87	37.21	21.86	42.51	38.86
Independent	71.69	46.22	25.82	57.21	50.24
Finetune Source	71.78	45.08	26.23	**59.95**	50.76
Ours (m=1)	72.58	46.45	25.82	58.35	50.80
Ours (m=3)	**73.03**	**47.44**	**29.51**	58.81	**52.20**

Table 14: The 40 Tasks created with `CIFAR100`.

Task ID	Classes
1	[aquarium_fish, flatfish, ray, shark, trout]
2	[orchid, poppy, rose, sunflower, tulip]
3	[beaver, dolphin, otter, seal, whale]
4	[bottle, bowl, can, cup, plate]
5	[bear, leopard, lion, tiger, wolf]
6	[apple, mushroom, orange, pear, sweet_pepper]
7	[hamster, mouse, rabbit, shrew, squirrel]
8	[baby, boy, girl, man, woman]
9	[maple_tree, oak_tree, palm_tree, pine_tree, willow_tree]
10	[bicycle, bus, motorcycle, pickup_truck, train]
11	[clock, keyboard, lamp, telephone, television]
12	[bed, chair, couch, table, wardrobe]
13	[bee, beetle, butterfly, caterpillar, cockroach]
14	[bridge, castle, house, road, skyscraper]
15	[cloud, forest, mountain, plain, sea]
16	[camel, cattle, chimpanzee, elephant, kangaroo]
17	[fox, porcupine, possum, raccoon, skunk]
18	[crab, lobster, snail, spider, worm]
19	[crocodile, dinosaur, lizard, snake, turtle]
20	[lawn_mower, rocket, streetcar, tank, tractor]
21	[hamster, cattle, maple_tree, squirrel, chimpanzee]
22	[house, bridge, bicycle, baby, lamp]
23	[shark, oak_tree, shrew, beaver, plate]
24	[willow_tree, crocodile, tiger, otter, telephone]
25	[bus, aquarium_fish, camel, skunk, apple]
26	[bee, forest, tank, sweet_pepper, fox]
27	[snail, whale, clock, lion, cockroach]
28	[rabbit, castle, pine_tree, cloud, boy]
29	[butterfly, road, rocket, skyscraper, wardrobe]
30	[spider, crab, tractor, mouse, seal]
31	[bed, elephant, beetle, keyboard, train]
32	[plain, table, ray, worm, sea]
33	[trout, bear, kangaroo, caterpillar, turtle]
34	[motorcycle, bottle, orchid, chair, leopard]
35	[dolphin, can, porcupine, cup, pickup_truck]
36	[poppy, wolf, pear, bowl, man]
37	[snake, tulip, streetwar, palm_tree, girl]
38	[lawn_mower, television, mushroom, lizard, raccoon]
39	[orange, dinosaur, possum, lobster, flatfish]
40	[mountain, couch, rose, woman, sunflower]

Table 15: Details of CTrL S_{long} (tasks 1-50).

Task ID	Dataset	Classes	#Train	# Val	# Test
1	mnsit	[3, 0, 5, 6, 8]	5000	2500	4804
2	svhn	[9, 1, 3, 7, 0]	25	15	5000
3	svhn	[2, 9, 7, 1, 3]	25	15	5000
4	svhn	[5, 3, 0, 2, 9]	5000	2500	5000
5	fashion-mnist	[Sandal, Dress, T-shirt/top, Ankle boot, Pullover]	25	15	5000
6	fashion-mnist	[Ankle boot, Sneaker, Dress, Shirt, Trouser]	25	15	5000
7	svhn	[5, 4, 6, 3, 7]	5000	2500	5000
8	cifar100	[man, squirrel, mouse, keyboard, maple_tree]	2250	250	500
9	cifar10	[horse, cat, bird, frog, dog]	5000	2500	5000
10	fashion-mnist	[Coat, Ankle boot, Shirt, Pullover, Sneaker]	5000	2500	5000
11	mnist	[0, 7, 1, 9, 6]	25	15	4938
12	cifar10	[bird, ship, airplane, dog, frog]	5000	2500	5000
13	cifar100	[hamster, bear, dolphin, bicycle, road]	25	15	500
14	mnist	[9, 5, 0, 8, 3]	5000	2500	4846
15	fashion-mnist	[Sandal, Trouser, Shirt, Bag, Coat]	5000	2500	5000
16	cifar10	[frog, bird, deer, automobile, horse]	5000	2500	5000
17	cifar10	[deer, ship, truck, airplane, frog]	5000	2500	5000
18	svhn	[7, 0, 6, 5, 3]	5000	2500	5000
19	mnist	[8, 5, 7, 6, 4]	25	15	4806
20	mnist	[1, 4, 2, 0, 9]	25	15	4962
21	cifar100	[whale, train, boy, crab, caterpillar]	2250	250	500
22	cifar100	[rabbit, cattle, camel, cockroach, caterpillar]	2250	250	500
23	cifar100	[orchid, road, spider, snake, caterpillar]	25	15	500
24	svhn	[3, 7, 8, 4, 5]	25	15	5000
25	cifar100	[cloud, raccoon, baby, lamp, orange]	2250	250	500
26	cifar100	[dolphin, castle, flatfish, house, whale]	2250	250	500
27	cifar100	[skunk, chimpanzee, tractor, raccoon, lion]	25	15	500
28	svhn	[8, 6, 9, 2, 5]	5000	2500	5000
29	fashion-mnist	[Coat, Sandal, Bag, Ankle boot, Trouser]	5000	2500	5000
30	mnist	[9, 8, 2, 6, 7]	5000	2500	4932
31	cifar10	[truck, ship, deer, bird, automobile]	25	15	5000
32	cifar10	[airplane, horse, dog, bird, cat]	25	15	5000
33	svhn	[7, 8, 4, 5, 2]	25	15	5000
34	cifar100	[castle, fox, shark, skunk, crocodile]	2250	250	500
35	cifar100	[wardrobe, bridge, otter, lawn_mower, telephone]	25	15	500
36	fashion-mnist	[Bag, Coat, T-shirt/top, Dress, Sandal]	25	15	5000
37	cifar10	[deer, airplane, automobile, dog, cat]	25	15	5000
38	cifar10	[cat, deer, frog, horse, truck]	5000	2500	5000
39	svhn	[0, 7, 8, 4, 3]	25	15	5000
40	mnist	[5, 7, 9, 2, 4]	5000	2500	4874
41	cifar100	[wolf, dinosaur, ray, girl, crab]	2250	250	500
42	fashion-mnist	[Sandal, Coat, Sneaker, Trouser, Dress]	25	15	5000
43	cifar100	[snail, rose, sweet_pepper, worm, mushroom]	25	15	500
44	fashion-mnist	[Sneaker, Trouser, Tshirt/top, Dress, Pullover]	25	15	5000
45	fashion-mnist	[Bag, Sandal, Shirt, Sneaker, Dress]	25	15	5000
46	mnist	[1, 4, 9, 6, 8]	25	15	4914
47	cifar10	[frog, dog, truck, horse, bird]	25	15	5000
48	svhn	[0, 2, 4, 1, 7]	5000	2500	5000
49	cifar100	[bed, tiger, lamp, road, caterpillar]	2250	250	500
50	cifar100	[rocket, mouse, butterfly, road, cattle]	2250	250	500

Table 16: Details of CTrL S_{long} (tasks 51-100).

Task ID	Dataset	Classes	#Train	# Val	# Test
51	cifar100	[trout, road, plain, television, wardrobe]	25	15	500
52	cifar100	[ray, pear, lion, whale, wardrobe]	25	15	500
53	cifar10	[bird, dog, deer, frog, truck]	25	15	5000
54	mnist	[0, 7, 4, 9, 6]	25	15	4920
55	fashion-mnist	[Shirt, Sneaker, Pullover, Trouser, Ankle boot]	5000	2500	5000
56	fashion-mnist	[Bag, Pullover, Sneaker, Tshirt/top, Shirt]	25	15	5000
57	mnist	[7, 9, 3, 8, 0]	25	15	4954
58	cifar10	[ship, airplane, frog, automobile, deer]	25	15	5000
59	mnist	[5, 6, 2, 4, 7]	25	15	4832
60	mnist	[5, 4, 2, 6, 0]	25	15	4812
61	svhn	[0, 2, 7, 3, 6]	25	15	5000
62	mnist	[9, 8, 7, 0, 4]	25	15	4936
63	cifar10	[frog, horse, airplane, ship, automobile]	25	15	5000
64	mnist	[9, 1, 2, 8, 6]	25	15	4932
65	svhn	[7, 2, 5, 9, 1]	25	15	5000
66	cifar100	[streetcar, beaver, sea, dolphin, butterfly]	25	15	500
67	svhn	[6, 3, 9, 7, 2]	25	75	5000
68	mnist	[9, 5, 7, 0, 6]	25	15	4830
69	fashion-mnist	[Tshirt/top, Bag, Dress, Shirt, Sneaker]	25	15	5000
70	svhn	[8, 9, 4, 1, 6]	25	15	5000
71	svhn	[5, 1, 0, 7, 2]	25	15	5000
72	mnist	[5, 1, 4, 2, 7]	25	15	4874
73	cifar10	[frog, cat, horse, truck, deer]	25	15	5000
74	mnist	[3, 9, 4, 2, 8]	25	15	4956
75	cifar10	[ship, truck, deer, dog, bird]	25	15	5000
76	cifar100	[pear, bus, fox, cloud, oak_tree]	25	15	500
77	svhn	[1, 0, 2, 3, 7]	25	15	5000
78	svhn	[2, 8, 5, 4, 3]	25	15	5000
79	cifar100	[rabbit, woman, girl, skyscraper, tuplip]	25	15	500
80	cifar100	[trout, road, bridge, bowl, oak_tree]	25	15	500
81	mnist	[5, 4, 8, 0, 6]	25	15	4786
82	fashion-mnist	[Pullover, Dress, Coat, Bag, Tshirt/top]	25	15	5000
83	cifar10	[bird, cat, deer, dog, automobile]	25	15	5000
84	cifar10	[cat, dog, ship, frog, bird]	25	15	5000
85	cifar100	[chimpanzee, camel, palm_tree, leopard, spider]	25	15	500
86	mnist	[3, 4, 5, 9, 6]	25	15	4832
87	svhn	[8, 9, 4, 7, 2]	25	15	5000
88	cifar10	[horse, cat, dog, airplane, automobile]	25	15	5000
89	mnist	[9, 8, 0, 1, 6]	25	15	4912
90	cifar100	[possum, chair, bowl, mountain, cloud]	25	15	500
91	svhn	[7, 1, 5, 9, 6]	25	15	5000
92	cifar10	[bird, cat, ship, frog, deer]	25	15	5000
93	mnist	[2, 8, 7, 5, 6]	25	15	4824
94	svhn	[6, 8, 2, 0, 5]	25	15	5000
95	fashion-mnist	[Sandal, Sneaker, Pullover, Ankle boot, Bag]	25	15	5000
96	svhn	[4, 9, 2, 0, 1]	25	15	5000
97	cifar100	[beetle, motorcycle, skunk, bee, kangaroo]	25	15	500
98	cifar100	[rabbit, butterfly, rose, pear, mushroom]	25	15	500
99	svhn	[8, 7, 4, 2, 3]	25	15	5000
100	fashion-mnist	[Trouser, Sneaker, Bag, Tshirt/top, Pullover]	25	15	5000

Table 17: The 6 tasks created with Tiny-ImageNet. Classes that exist in different tasks are color-coded to highlight the class sharing.

	classes
task1	'n02124075', 'n04067472', 'n04540053', 'n04099969', 'n07749582', 'n01641577', 'n02802426', 'n09246464', 'n07920052', 'n03970156', 'n03891332', 'n02106662', 'n03201208', 'n02279972', 'n02132136', 'n04146614', 'n07873807', 'n02364673', 'n04507155', 'n03854065', 'n03838899', 'n03733131', 'n01443537', 'n07875152', 'n03544143', 'n09428293', 'n03085013', 'n02437312', 'n07614500', 'n03804744', 'n04265275', 'n02963159', 'n02486410', 'n01944390', 'n09256479', 'n02058221', 'n04275548', 'n02321529', 'n02769748', 'n02099712', 'n07695742', 'n02056570', 'n02281406', 'n01774750', 'n02509815', 'n03983396', 'n07753592', 'n04254777', 'n02233338', 'n04008634',
task2	'n09428293', 'n03085013', 'n02437312', 'n07614500', 'n03804744', 'n04265275', 'n02963159', 'n02486410', 'n01944390', 'n09256479', 'n02058221', 'n04275548', 'n02321529', 'n02769748', 'n02099712', 'n07695742', 'n02056570', 'n02281406', 'n01774750', 'n02509815', 'n03983396', 'n07753592', 'n04254777', 'n02233338', 'n04008634', 'n02823428', 'n02236044', 'n03393912', 'n07583066', 'n04074963', 'n01629819', 'n09332890', 'n02481823', 'n03902125', 'n03404251', 'n09193705', 'n03637318', 'n04456115', 'n02666196', 'n03796401', 'n02795169', 'n02123045', 'n01855672', 'n01882714', 'n02917067', 'n02988304', 'n04398044', 'n02843684', 'n02423022', 'n02669723'
task3	'n02823428', 'n02236044', 'n03393912', 'n07583066', 'n04074963', 'n01629819', 'n09332890', 'n02481823', 'n03902125', 'n03404251', 'n09193705', 'n03637318', 'n04456115', 'n02666196', 'n03796401', 'n02795169', 'n02123045', 'n01855672', 'n01882714', 'n02917067', 'n02988304', 'n04398044', 'n02843684', 'n02423022', 'n02669723', 'n04465501', 'n02165456', 'n03770439', 'n02099601', 'n04486054', 'n02950826', 'n03814639', 'n04259630', 'n03424325', 'n02948072', 'n03179701', 'n03400231', 'n02206856', 'n03160309', 'n01984695', 'n03977966', 'n03584254', 'n04023962', 'n02814860', 'n01910747', 'n04596742', 'n03992509', 'n04133789', 'n03937543', 'n02927161'
task4	'n04465501', 'n02165456', 'n03770439', 'n02099601', 'n04486054', 'n02950826', 'n03814639', 'n04259630', 'n03424325', 'n02948072', 'n03179701', 'n03400231', 'n02206856', 'n03160309', 'n01984695', 'n03977966', 'n03584254', 'n04023962', 'n02814860', 'n01910747', 'n04596742', 'n03992509', 'n04133789', 'n03937543', 'n02927161', 'n01945685', 'n02395406', 'n02125311', 'n03126707', 'n04532106', 'n02268443', 'n02977058', 'n07734744', 'n03599486', 'n04562935', 'n03014705', 'n04251144', 'n04356056', 'n02190166', 'n03670208', 'n02002724', 'n02074367', 'n04285008', 'n04560804', 'n04366367', 'n02403003', 'n07615774', 'n04501370', 'n03026506', 'n02906734'

Table 18: (Continued) The 6 tasks created with Tiny-ImageNet. Classes that exist in different tasks are color-coded to highlight the class sharing.

	classes
task5	'n01945685', 'n02395406', 'n02125311', 'n03126707', 'n04532106', 'n02268443', 'n02977058', 'n07734744', 'n03599486', 'n04562935', 'n03014705', 'n04251144', 'n04356056', 'n02190166', 'n03670208', 'n02002724', 'n02074367', 'n04285008', 'n04560804', 'n04366367', 'n02403003', 'n07615774', 'n04501370', 'n03026506', 'n02906734', 'n01770393', 'n04597913', 'n03930313', 'n04118538', 'n04179913', 'n04311004', 'n02123394', 'n04070727', 'n02793495', 'n02730930', 'n02094433', 'n04371430', 'n04328186', 'n03649909', 'n04417672', 'n03388043', 'n01774384', 'n02837789', 'n07579787', 'n04399382', 'n02791270', 'n03089624', 'n02814533', 'n04149813', 'n07747607'
task6	'n03355925', 'n01983481', 'n04487081', 'n03250847', 'n03255030', 'n02892201', 'n02883205', 'n03100240', 'n02415577', 'n02480495', 'n01698640', 'n01784675', 'n04376876', 'n03444034', 'n01917289', 'n01950731', 'n03042490', 'n07711569', 'n04532670', 'n03763968', 'n07768694', 'n02999410', 'n03617480', 'n06596364', 'n01768244', 'n02410509', 'n03976657', 'n01742172', 'n03980874', 'n02808440', 'n02226429', 'n02231487', 'n02085620', 'n01644900', 'n02129165', 'n02699494', 'n03837869', 'n02815834', 'n07720875', 'n02788148', 'n02909870', 'n03706229', 'n07871810', 'n03447447', 'n02113799', 'n12267677', 'n03662601', 'n02841315', 'n07715103', 'n02504458'