Learning to Weight Filter Groups for Robust Classification

Siyang Yuan[1,†] Yitong Li[2,*] Dong Wang[1] Ke Bai[1] Lawrence Carin[1,3] David Carlson[1]
[1]Duke University, Durham, NC [2]Apple Inc. [3]KAUST, Saudi Arabia
[†]siyang.yuan@duke.edu

Abstract

In many real-world tasks, a canonical "big data" problem is created by combining data from several individual groups or domains. Because test data will likely come from a new group of data, we want to utilize the grouped structure of our training data to enforce generalization between groups of data, not just individual samples. This can be viewed as a multiple-domain generalization problem. Specifically, the goal is to encourage generalization between previously seen labeled source data from multiple domains and unlabeled target domain data. To address this challenge, we introduce Domain-Specific Filter Group (DSFG), where each training domain has a unique filter group and each test data point is predicted by a weighted sum over the outputs of different domain filters. A separate neural network learns to estimate the appropriate filter group weights through a meta-learning strategy. Empirically, experiments on three benchmark datasets demonstrate improved performance compared to current state-of-the-art approaches.

1. Introduction

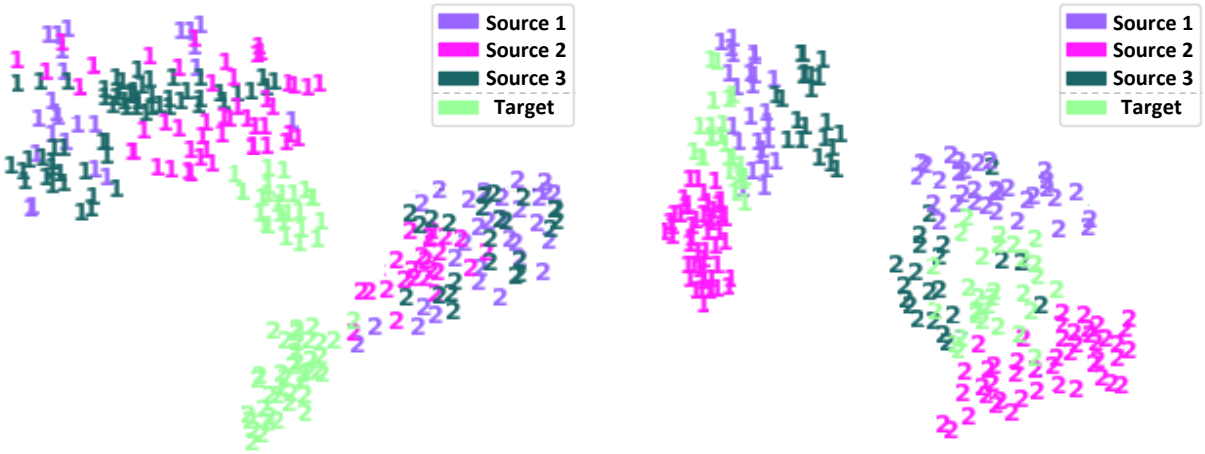
When machine learning algorithms are deployed in the real world, new data often come from a different distribution than the training data. This challenge has motivated development of *unsupervised domain adaption* methods [34, 12] that make use of unlabeled test data to learn procedures that generalize well to new data. Unfortunately, in many cases real test data (new real-world data) are difficult to collect beforehand. Instead, it is desirable for a model to address potential distribution gaps automatically during test time. For instance, consider a clinical situation in which longitudinal data are considered and we desire to either predict events or labels ($e.g.$, a seizure from collected brain activity). Only a small collection of subjects may have expert-labeled data, while the model should be applicable across a wide range of participants or patients. Future individuals will not exactly match the training cohort, which may have many sources of differences (many physiological signals could potentially be used as forensics tools because they are fairly unique between individuals [33]). This data structure has many labeled examples but only a few individuals, and can be thought of as "little big data." Our methods should take advantage of the large sample size while still being cognizant of the fact that the number of observed groups is small.

In contrast to unsupervised domain adaptation, our framework considers each group of samples in the training data as a unique source domain and assumes that *no* test data or target domain information is available during training. In the literature, this situation is called *unsupervised domain generalization* [2, 45]. Unsupervised domain generalization is more feasible in practice for a distributed or deployed system, so the domain generalization framework mimics our motivating situation.

A popular approach to addressing the varying distributions of the source domains is to learn a domain-invariant latent feature space, meaning that the latent feature distribution is similar for all training domains. For example, this can be done by slightly modifying the popular Domain-Adversarial Neural Network (DANN) [13] to operate on multiple source domains. Unfortunately, there is no guarantee that the features of the target samples will fall into this shared representation, and often they do not. A visualization of this issue is shown in Figure 1(a), where the target (light green) falls outside of the shared latent space of three training domains (pink, purple and dark green). This means that test samples appear in a novel part of the feature space, hindering generalization. Furthermore, domain-specific and domain-invariant features are often entangled together [19] and a failure to keep both sources of information harms performance.

Our approach to this challenge is to allow each source domain to have both shared and unique properties, while explicitly training the model to encourage domain generalization. At test time, newly collected data are automatically processed by a weighted combination of a shared feature extractor and source feature extractors. Rather than complete domain invariance, a new sample only needs to be similar to one source domain. This idea is visualized in Figure 1(b),

[*]Work performed while at Duke University.



(a) Source-domain-invariant latent space.

(b) Desired latent space.

Figure 1: Conceptual plots of four domains (MNIST as Source 1, MNISTM as Source 2, synthetic number as Source 3 and SVHN as Target) on two labels ('1' and '2'). Pink, purple and dark green colors represent the three source domains. The target is given by light green. Figure 1(a) plots the shared latent space from a model learning the domain-invariant features. Figure 1(b) is the desired latent space (DSFG), where the target can be adjusted by samples from source domains.

where the training domains are allowed to have distinct feature distributions, and the target (light green) is represented by a superposition of the training domains. To actualize this intuition, two problems need to be solved: (i) retaining both domain-invariant and domain-specific information while (ii) combining different source domains to adapt to unseen targets.

We propose the **Domain Specific Filter Groups** (DSFG) approach to address these problems. It has three major components: (i) we allow for domain differences by giving each source domain unique and shared filter groups; (ii) a weight generator estimates similarity between a new sample and source training domains, allowing DSFG to share strength only between similar domains; and (iii) we train with both in-domain and cross-domain stages, mimicking the test process during training.

The in-domain training learns filters that perform well on the existing domains, whereas the cross-domain training emphasizes learning how to generalize to new samples by mimicking the test procedure. The weight generator is learned in the cross-domain training to heavily weight similar domains by a meta-learning approach, allowing the network to utilize the most relevant information. The model parameters are fixed after training and can be deployed on low-powered machines ($e.g.$, distributed or mobile systems), a major consideration for practical use. Empirically, DSFG improves performance on multiple benchmarks in comparison with state-of-the-art models.

2. Related Work

Unsupervised Domain Adaptation. The developments in unsupervised domain adaptation are highly related to domain generalization. This setup typically focuses on adapting a single source domain to predict well on an unlabeled target domain [34, 12, 44, 9, 13]. Multiple source [32, 49, 27] and multiple target [48, 14] extensions have been proposed. Several extensions have been proposed to address specific issues, including partial label matching [4] and target label shift [39, 28]. In all of these setups, the data are projected to a shared latent space with a shared classifier. The underlying assumption is that the target sample is closely and equally related to all source domains, with slight relaxations such as the weighted similarity of Li *et al.* [27]. However, in our data types, this assumption does not appear to be true in practice ($e.g.$, the example in Figure 1), and it is well-known that adding uncorrelated domains in this framework will harm performance [32].

Incremental Learning. Incremental learning adds and learns new categories to existing models by selectively adapting the parameters from learned categories [40, 38, 42]. This can be achieved by (i) sharing weights of the learned model [42], (ii) adding regularization terms to the new category weights [40], and (iii) using landmarks from each known category to help the newly added task [38]. DSFG has some algorithmic similarities to these, but does not learn on new data and gives predictions in real time.

Channel attention and filter groups. Multiple works have proposed attention maps for adaptive feature refinement [17, 47], including stochastically assigning group filters in

CNN architectures such that task-specific and shared features are encoded with task-specific and shared kernels in convolutional filters [3]. Our weight generator shares some similarities to these ideas, but is its own distinct process.

Domain Generalization and Meta Learning. Domain generalization algorithms try to learn domain-invariant prediction schemes [36]. Data augmentation has been used to modify training samples to make them hard to classify while preserving the label and domain categories [45, 41]. Meta-learning has been used to learn a domain-generalization regularization term [2, 29]. Episodic training has been considered for domain generalization, which includes several learning stages to enhance generalization [26, 11]. Many of these models require test time updates [23]. [8] gives a method to "compute" the model's generalization ability. Finally, self-supervision has been used to extract generalized features [5, 46]. We consider this branch of literature most similar to DSFG, and we compare to many of these approaches in our experiments.

Meta-learning, or learning to learn, is typically thought of as learning the learning update rules for neural models [1, 37], but has been used in few-shot learning [30]. Many domain generalization algorithms relate to meta-learning or ensemble learning, in that the model parameters are adjusted at test time [20, 31]. DSFG uses meta-learning during training to enhance generalization, but is not used at test time.

3. Methods

The training set contains S source domains, where each domain s has samples $\{\boldsymbol{x}_i, y_i, d_i = s\}_{i=1,...,N_s}$; $\boldsymbol{x}_i$ is the raw data input, $y_i \in \{1, ..., L\}$ is the label, and $d_i \in \{1, ..., S\}$ indicates the source domain index. For a test sample, we simply have the raw data input $\boldsymbol{x}_*$. Note that $\boldsymbol{x}_*$ is not available during training. Suppose $\boldsymbol{x}_*$ is sampled from an unknown target distribution $p(\boldsymbol{x}_*)$, which can be *identical* to one of the sources or a *totally new* domain. The goal is to train a generalized model, which can be applied to the unseen target without further updates.

The model framework of DSFG is visualized in Figure 2, which is built upon a convolutional neural network (CNN) with a specifically designed CNN filter structure (F-Conv), which is fully defined in Section 3.1. The newly proposed F-Conv layer contains a shared filter group as well as S domain-specific filter groups. As the names indicate, these filter groups learn shared and specific features, respectively, which we exploit to enhance the domain generalization. A filter weight generator $G(\cdot; \boldsymbol{\theta}_G)$ is proposed to assign filter weights uniquely for each data example (green block in Figure 2), which is fully defined in Section 3.3. The output features of the F-Conv (Filter Convolution) layer are a weighted combination of different filter groups. An F-Conv layer can be used as a substitution of the standard 2d-convolutional layer, and as many F-Conv layers as desired can be used. In

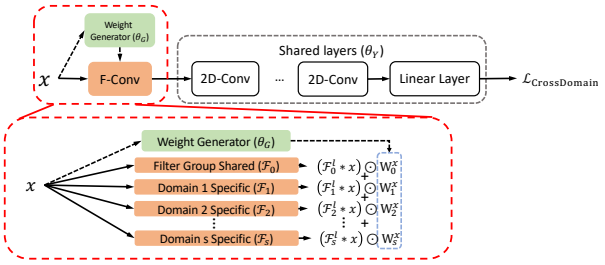


Figure 2: Model Framework with detailed illustrations on the filter group convolutional layer (F-Conv). Black arrows indicate the forward network pass. This structure is also used during the cross-domain training process. The dotted arrows indicates the weight generator $\boldsymbol{\theta}_G$ is used in the forward pass but not updated during cross-domain training step.

the rest of the paper, we discuss the case for which only the first convolutional layer is substituted by the F-Conv layer. Afterwards, the adapted features are input into the remaining shared network layers $f(\cdot; \boldsymbol{\theta}_Y)$ to predict the output. For simplicity, we denote $\boldsymbol{\theta}_Y$ as the parameters used in all shared layers after the F-Conv layer.

3.1. F-Conv Layer

A key component of DSFG is the F-Conv layer, which has both shared and domain-specific filter groups. The shared filter group learns common representations, denoted $\mathcal{F}_0 = \{\mathcal{F}_{01}, \cdots, \mathcal{F}_{0K_0}\}$. Each filter is a standard convolutional filter and is represented as a 4d-cube with dimensions corresponding to the number of input channels, number of output channels, kernel height, and kernel width; K_0 is the total number of the shared filters. The domain-specific filters are $\mathcal{F}_s = \{\mathcal{F}_{s1}, \cdots, \mathcal{F}_{sK_s}\}$ for each domain $s \in \{1, ..., S\}$. Similarly, K_s is the number of domain-specific filters for $\mathcal{D}_s$. In practice we set $K_0 = K_1 = \cdots = K_S$. Figure 2 visualizes the filter assignment at one F-Conv layer in the dashed red box.

While these domain-specific filter groups are focused on their own domains, they should be useful for some of the other domains as well. Given a data sample $\boldsymbol{x}$, at test time these filter groups will be combined based upon their similarity with $\boldsymbol{x}$. This is done by calculating a domain similarity matrix $\boldsymbol{W}^x \in (0, 1)^{(S+1)\times K}$ with column index 0 corresponding to shared filter and column indices , $1, \cdots, S$ corresponding to the source domains. The superscript x denotes that the weights are sample-specific. Each column in $\boldsymbol{W}$ is normalized so that the total weight on each feature is equal to 1. A large value of W_{sk}^x indicates that the sample $\boldsymbol{x}$ is 'close' to the kth feature of $\mathcal{D}_s$, and the filter $\mathcal{F}_{sk}$ is up-weighted. The generated weight W_{sk}^x is then applied to the convolved feature $\mathcal{F}_{sk} * x$ ($*$ is the convolution operation). The first layer's extracted features are given by a weighted

superposition of the filter group outputs, with each feature map $\boldsymbol{h}_k$ given by

$$\boldsymbol{h}_k = \sum_{s=0}^{S} W_{sk}^{x}(\mathcal{F}_{sk} * \boldsymbol{x}), \tag{1}$$

These first-layer feature maps are then input to the next layer. Equation (1) could be viewed as a type of channel attention [47]. The mathematical setup discussed here has focused on 2d convolution to match our experiments, but the same idea can be applied in the context of dense layers or N-d CNNs.

We introduced F-conv only for the first layer, which matches our experimental setup. One can use as many layers of F-conv as desired, as visualized in Supplemental Figure 6. However, since the feature divergence between domains decreases as layer depth increases [35], it is more efficient to substitute shallow layers with F-conv.

There are several considerations for how to effectively learn these filters and weights. We describe our training procedure in Section 3.2 and then the weight generator in Section 3.3.

3.2. In-domain and Cross-domain Training

The goal guiding the training process is to perform well on an unseen test sample without further model parameter updates. This first requires that the shared and domain-specific filter groups perform well on the known source samples. Furthermore, the weight generator (see (3)) should assign filter weights for each data point to enhance generalization. We use *in-domain* and *cross-domain* training processes to encourage these properties.

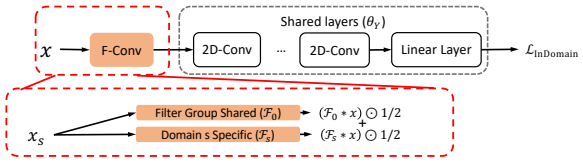
For the in-domain training stage, as illustrated in Figure 3(a), data in domain s only uses the shared filters $\mathcal{F}_0$ and its domain-specific filters $\mathcal{F}_s$. The filter weights $\boldsymbol{W}^x$ are set as constants, where the entries for the shared weights and the domain s weights are set to 0.5 and the others are zero. The loss function is given as

$$\mathcal{L}_{\text{InDomain}} = \sum_s \sum_{\boldsymbol{x}\in\mathcal{D}_s} \text{CE}\left(f(\tfrac{\mathcal{F}_{0k}*\boldsymbol{x}+\mathcal{F}_{sk}*\boldsymbol{x}}{2}|_{k=1}^{K};\boldsymbol{\theta}_Y), y\right), \tag{2}$$

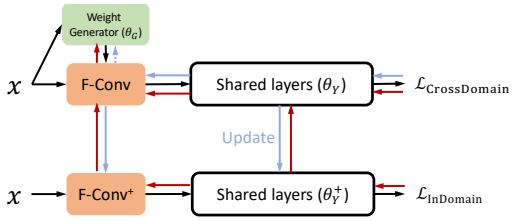
where $\hat{y} = f(\cdot;\boldsymbol{\theta}_Y)$ are the shared layers including feature extraction and prediction network, parameterized by $\boldsymbol{\theta}_Y$, CE is cross entropy loss. The notation $\cdot|_{k=1}^{K}$ represents the 3-d tensor stacking of all feature maps. In this stage the network is trained to achieve good predictions for each training domain domain, which is a prerequisite prior for domain generalization.

We introduce cross-domain training to simulate the test process. During the cross-domain training step (Figure 2), we optimize the classification loss using a weighted combination of all available filter banks. We first calculate filter weights,

$$\boldsymbol{W}^{\boldsymbol{x}} = G(\boldsymbol{x};\boldsymbol{\theta}_G). \tag{3}$$



(a) Illustration of in-domain training process.



(b) Illustration of weight generator update process.

Figure 3: Visualization of the training process. Figure 3(a) illustrates the in-domain training process. Figure 3(b) illustrates the weight generator updates. The blue solid arrow indicates the gradient flow of the one-step gradient descent update in (8) and (9). The blue dotted arrow indicates the one-step gradient descent update does not update weight generator parameters $\boldsymbol{\theta}_G$. The red arrow gives the gradient flow when updating the weight generator. The superscript $+$ indicates the one-step gradient update used in the meta-update.

The loss in the cross-domain training is given as

$$\mathcal{L}_{\text{CrossDomain}} = \sum_s \sum_{x\in\mathcal{D}_s} \text{CE}\left(f(\sum_{s=0}^{S} W_{sk}^{\boldsymbol{x}}(\mathcal{F}_{sk} * \boldsymbol{x})|_k^K;\boldsymbol{\theta}_Y), y\right). \tag{4}$$

Note that the cross-domain weights include a weight on the input data's source domain filters (self-similarity). One could explicitly exclude that; however, we do not know the domain of the test sample and thus do not want to exclude self-similarity. For a given data sample, we expect that the weight generator assigns the majority of weights to the sample's original domain with the remaining weight given to its similar neighbors. This requires the weight generator to accurately calculate weights for each filter group. The gradients from (4) are used to update the $\boldsymbol{\theta}_Y$ and $\{\mathcal{F}_0, \ldots, \mathcal{F}_S\}$ in a standard fashion, but will be used in a unique way for the updates on the weight generator network introduced in the follow section.

3.3. Weight Generator Updates

A good filter weight generator is expected to provide the similarity between a new data point and each of the source domain filters. In DSFG, the weight generator is structured with one convolutional layer followed by one dense layer, with a softmax output over each output k, to produce the normalized weights. Meta-learning is used to learn effective parameters $\boldsymbol{\theta}_G$. This meta-learning gradient strategy is able

to (i) encourage high accuracy from the in-domain loss in (2) and cross-domain loss in (4), and (ii) discover domain similarity. Figure 3(b) visualizes the forward and backward passes of this step.

We first introduce the prior loss, which helps the filter weight generator to discover self-similarity ($e.g.$, samples from domain $\mathcal{D}_s$ should have large weight on filter group $\mathcal{F}_s$). Specifically, we add a prior domain classification loss to encourage self-similarity. This can be written as

$$\mathcal{L}_{\text{prior}} = -\sum_{s}\sum_{\boldsymbol{x}\in\mathcal{D}_s}\sum_{k}\log W^{x}_{sk}. \tag{5}$$

However, (5) alone will cause the filter weight generator to collapse to a specific filter group. We mitigate this problem by introducing an entropy loss that ensures the generated filter weight will not be too sparse,

$$\mathcal{L}_{\text{entropy}} = \sum_{s}\bar{W}^{x}_{s}\log\left(\bar{W}_s{}^{x}\right), \tag{6}$$

where $\bar{W}^{x}_{s} = \sum_k W^{x}_{sk}$ is the averaged mean of filter weight for filter group s. This allows individual samples to have sparse weights, but not a domain as a whole.

These loss terms are combined together as a regularization term for the filter generator:

$$\mathcal{L}_{\text{G-reg}} = \mathcal{L}_{\text{CrossDomain}} + \alpha\mathcal{L}_{\text{prior}} + \beta\mathcal{L}_{\text{entropy}}, \tag{7}$$

where α and β are non-negative hyperparameters, and $\mathcal{L}_{\text{CrossDamain}}$ is the cross-domain loss from (4). Only $\boldsymbol{\theta}_G$ in the cross-domain loss will be updated in this step. In Figure 3(b), the prior loss and the entropy loss are not visualized.

Update: This step is visualized as the red arrow in Figure 3(b). We leverage a meta-learning update scheme to give accurate filter weights and encourage high label classification accuracy. This step bridges the in-domain and cross-domain training losses in the gradient space (Figure 3(b)). We first compute the one-step gradient descent update in the filter groups and classifier parameters,

$$\mathcal{F}^{+} = \mathcal{F} - \delta\frac{\partial L_{\text{CrossDomain}}}{\partial\mathcal{F}}, \tag{8}$$

$$\boldsymbol{\theta}^{+}_{Y} = \boldsymbol{\theta}_Y - \delta\frac{\partial L_{\text{CrossDomain}}}{\partial\boldsymbol{\theta}_Y}, \tag{9}$$

where δ is the step size unique to this step. Note that $\mathcal{F}^{+}$ and $\boldsymbol{\theta}^{+}_{Y}$ are both functions of $\boldsymbol{\theta}_G$. These temporary variables will be used to update $\boldsymbol{\theta}_G$ and then are discarded. This gradient back-propagation is indicated as the blue solid arrow in Figure 3(b).

The parameter update for the filter weight generator uses the in-domain loss based on the temporary parameters and the regularization term in (7),

$$\boldsymbol{\theta}_G \leftarrow \boldsymbol{\theta}_G - \lambda\frac{\partial\mathcal{L}_{\text{InDomain}}(\mathcal{F}^{+},\boldsymbol{\theta}^{+}_{Y},\boldsymbol{\theta}_G)+\partial\mathcal{L}_{\text{G-reg}}(\mathcal{F},\boldsymbol{\theta}_Y,\boldsymbol{\theta}_G)}{\partial\boldsymbol{\theta}_G}. \tag{10}$$

Algorithm 1 DSFG Training Algorithm

Input: Training Data $\{\boldsymbol{x}_i, y_i, d_i\}_{i=1}^{N}$.
Output: Model parameters $\boldsymbol{\theta}_Y$, $\boldsymbol{\theta}_G$, $\mathcal{F}_s$, $s \in \{0, \cdots, S\}$.

Initialize network parameters
for $iter = 1$ to $iter_{max}$ **do**
  In-domain Training
  for $s = 1$ to S **do**
    Sample mini-batch from domain $\mathcal{D}_s$.
    Update filter group weights $\{\mathcal{F}_0, \mathcal{F}_s\}$ and shared weights $\boldsymbol{\theta}_Y$ using in-domain loss $\mathcal{L}_{\text{InDomain}}$ (see (2)).
  end for
  Sample a mini-batch with equal number of samples from each domain.
  Cross-domain Training
  Compute the outputs of weight generator using (3).
  Update all filter groups and $\boldsymbol{\theta}_Y$ using $\mathcal{L}_{\text{CrossDomain}}$ (see (4)).
  Update Weight Generator
  Compute the one-step gradient descent update of $\mathcal{F}^{+}$ and $\boldsymbol{\theta}^{+}_{Y}$ (see (8) and (9)).
  Evaluate the in-domain training loss $\mathcal{L}_{\text{InDomain}}$(see (2)) with $\mathcal{F}^{+}$ and $\boldsymbol{\theta}^{+}_{Y}$.
  Update the weight generator $\boldsymbol{\theta}_G$ with (10).
end for

Algorithm 2 DSFG Test Time Algorithm

Compute the output of weight generator $\boldsymbol{W}^{x} = G(\boldsymbol{x};\boldsymbol{\theta}_G)$(see (3)).
Compute the adapted feature by (1).
Output the label prediction using the adapted feature and the shared classifier.

Here, $\mathcal{L}_{\text{InDomain}}(\mathcal{F}^{+},\boldsymbol{\theta}^{+}_{Y},\boldsymbol{\theta}_G)$ evaluates the in-domain prediction quality at the one-step ahead gradient; λ represents the learning rate, and we set $\lambda = \frac{1}{10}\delta$. Equation (10) helps modify $\boldsymbol{\theta}_G$ so that high classification accuracy on the in-domain training is maintained while reducing the cross-domain error. Without this update, the in-domain error tends to suffer as the cross-domain error decreases.

3.4. Algorithm Outline

During training, each filter group is expected to perform high-quality prediction and generalize well on unseen testing samples. Algorithm 1 gives the pseudo-code for the full training process. We use pretrained parameters when available for initialization. For F-conv, the pretrained weights are replicated over the $S+1$ groups of filters.

Inference on a new target sample requires only a single forward pass, which is computationally efficient, as given in Algorithm 2.

4. Experiments

DSFG is evaluated on three benchmark datasets for domain generalization: digit datasets, VLCS [43] and PACS [24]. We consider different multi-layer CNN feature extractors, including AlexNet [21], ResNet18 and ResNet50 [16]. DSFG's filter group mechanism is only applied on the first CNN layer. The model selection for DSFG is performed by averaging cross-domain classification accuracy on a validation set sampled within source domains. This mimics the well-known nested cross-validation scheme in statistics [22]. To provide a fair comparison, all baseline models have the same feature extractor and classifier structure, unless otherwise noted.

We implemented our framework with the PyTorch library on one NVIDIA TITAN 1080Ti GPU. Our model is optimized with the SGD optimizer, and the model is trained for 100 epochs. The learning rate was set as 0.001 and decreased by a factor of 10 after 50 epochs.

Comparison Approaches: We choose several recent domain generalization works for comparison: **DANN** [13], **CROSSGRAD** [41], and **DATA AUGMEN** [45]. We then considered the episodic training scheme of **Fpi-FCR** [26], as well as the self-supervised approaches of **JiGen** [5] and **EISNet** [46], and the meta-learning approaches of **MetaReg** [2] and **MASF** [11]. We also considered **DMG** [7] in which domain-specific masks are learned to leverage domain-specific features. Finally, **RSC** [18] proposes a training scheme in which the dominant features activated are discarded during training. Here, we compare with reproduced results instead of the reported, ones to ensure that we share the same experimental conditions and model selection method, which is essential for fair comparisons [15].

Ablation Study: We explore comparisons to DSFG with some components missing to elucidate the key contributions. First, in **Baseline**, the feature extractor is fine-tuned on all source domain data without further modifications. Next, since the proposed filter group mechanism involves more parameters, we include a baseline model named **Baseline with F-Conv**, in which multiple filter groups are involved, but only trained with in-domain loss defined in (2). The filter weight generator is replaced with an average over all filter groups at test time. Next, **DSFG (no meta)** denotes the model where we update weight generator parameters $\boldsymbol{\theta}_G$ by optimizing only the regularization loss in (7) and exclude the meta-learning update.

4.1. Digit Datasets

We consider four digit datasets: MNIST, MNIST-M, SVHN and Synthetic numbers (SYN NUM). MNIST is grey-scale hand-written digits from zero to nine. MNIST-M is constructed from MNIST by adding colored backgrounds [13]. SVHN is a color image street number dataset. SYN NUM is generated from Windows fonts to mimic the patterns of

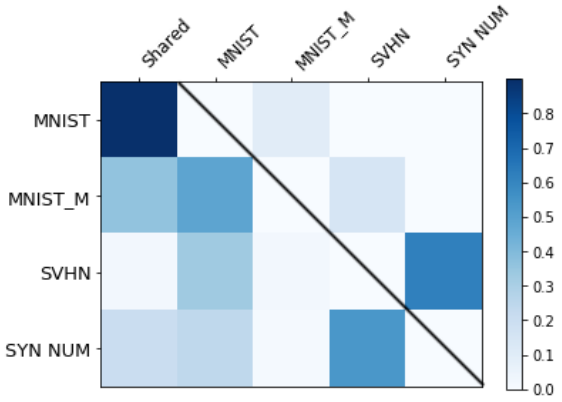


Figure 4: Visualization of averaged generated filter weights on test samples in the digit datasets. The vertical axis represents the current target domain, and the horizontal axis represents the chosen source filter groups, including the shared weights.

the SVHN dataset [13]. The feature extractor in this experiment is configured as a three-layer convolutional neural network with ReLU activation functions. The classifier is a fully connected layer. Through cross-validation search, the hyper-parameters α and β are set as 50 and 0.002. Results are shown in Table 1. The target domain used for each test is shown in the top row, and the other domains are used for training. We use the same feature extractor architecture in all models we train for a direct and fair comparison. The proposed method has strong improvements compared to baselines and competing methods. In our ablation analysis, the largest improvement came from the weight generator added in DSFG (no meta), but including the meta-learning step led to further improvements in all cases.

Many of the cited works typically used a much more complex feature extractor structure. To address this, we also include results from previous works [25, 6]; however, they did not include the results when synthetic numbers were included as a domain. Our baseline result outperforms their results by a large margin. From these results, we can also see that learning a shared latent space harms the performance under several tests ($e.g.$, DANN is worse than baseline).

Filter weight visualization: We visualize the averaged generated filter weights $\boldsymbol{W}^{\boldsymbol{x}}$ in Figure 4. We average all target domain samples across all filter groups to get an averaged target-to-source weight score. Although MNIST-M is composed of color images, it is still matched to MNIST (grey scale). This is expected because MNIST-M is generated based upon images in MNIST. Furthermore, SVHN and SYN NUM are more similar as they mimic similar scenarios. These results support the conclusion that the weight generator is discovering real relationships.

	Backbone	MNIST	MNISTM	SVHN	SYN NUM	Avg ↑
MLDG [25]	3-layer CNN	**99.1**	61.2	69.7	N/A	N/A
ADAGE [6]	3-layer CNN	**99.1**	66.3	76.4	N/A	N/A
Baseline*	3-layer CNN	99.0	82.4	84.3	94.4	90.0
Baseline with F-Conv*	3-layer CNN	99.0	82.5	85.4	94.6	90.4
DANN* [13]	3-layer CNN	99.0	81.7	84.5	94.2	89.9
CROSSGRAD* [41]	3-layer CNN	99.0	84.2	85.8	93.8	90.7
DATA AUGMEN* [45]	3-layer CNN	98.8	80.6	85.2	93.3	89.5
MASF* [11]	3-layer CNN	98.8	81.2	85.2	93.0	89.5
DSFG (no meta)*	3-layer CNN	98.8	82.5	85.6	94.9	90.5
DSFG (full)*	3-layer CNN	98.9	**85.7**	**86.0**	**95.4**	**91.5**

Table 1: Accuracy on digit image classification. All methods are implemented with 3-layer CNN backbone. Methods with a * are our implementation with the same feature extractor structure. Other results are reported from the cited work. In each category, the best results are shown in bold.

VLCS	Backbone	Caltech	Labelme	Pascal	Sun	Avg ↑
Baseline [5]	AlexNet	96.25	59.72	70.58	64.51	72.76
Baseline with F-Conv	AlexNet	98.11	59.60	66.34	61.83	71.47
Epi-FCR [26]	AlexNet	94.10	64.30	67.10	65.90	72.90
JiGen [5]	AlexNet	96.93	60.90	70.62	64.30	73.19
MASF [11]	AlexNet	94.78	**64.90**	69.14	67.64	74.11
EISNet [46]	AlexNet	97.33	63.49	69.83	**68.02**	74.67
DSFG (no meta)	AlexNet	98.03	60.49	69.83	64.57	73.23
DSFG (full)	AlexNet	**98.58**	63.61	**70.68**	67.09	**74.99**

Table 2: Results on multi-source domain generalization on VLCS [43] dataset. All methods are implemented with AlexNet [21]. In each category, the best results are shown in bold.

PACS	Backbone	artpaint	cartoon	sketch	photo	Avg ↑
Baseline [5]	ResNet18	78.96	73.93	70.59	96.28	79.94
Baseline with F-Conv	ResNet18	79.69	74.74	73.71	**96.59**	81.18
MASF [11]	ResNet18	80.29	77.17	71.69	94.99	81.03
Epi-FCR [26]	ResNet18	82.10	77.00	73.00	93.90	81.50
JiGen [5]	ResNet18	79.42	75.25	71.35	96.03	80.51
MetaReg [2]	ResNet18	83.70	77.20	70.30	95.50	81.70
EISNet [46]	ResNet18	81.89	76.44	74.33	95.93	82.15
DMG [7]	ResNet18	76.90	80.38	75.21	93.35	81.46
RSC [18] (reported)	ResNet18	83.43	80.31	80.85	95.99	85.15
RSC [18] (reproduced)	ResNet18	78.9	**76.88**	76.81	94.10	81.67
DSFG (no meta)	ResNet18	81.69	75.51	75.49	95.57	82.07
DSFG (Ours)	ResNet18	**83.89**	76.45	**78.26**	95.09	**83.42**
Baseline [5]	ResNet50	86.20	78.70	70.63	**97.66**	83.29
Baseline (multi)	ResNet50	83.69	76.37	78.6	96.53	83.79
MASF [11]	ResNet50	82.89	80.49	72.29	95.01	82.67
MetaReg [2]	ResNet50	87.20	79.20	70.30	97.60	83.60
EISNet [46]	ResNet50	86.64	81.53	78.07	97.11	85.84
DMG [7]	ResNet50	82.57	78.11	78.32	94.49	83.37
RSC [18] (reported)	ResNet50	87.89	82.16	83.35	97.92	87.83
RSC [18](reproduced)	ResNet50	81.38	80.14	82.31	93.72	84.38
DSFG (no meta)	ResNet50	85.84	79.35	80.05	96.43	85.42
DSFG (full)	ResNet50	**87.30**	**80.93**	**83.43**	96.59	**87.06**

Table 3: Results on multi-source domain generalization on PACS [24] dataset. The top session reports the results for ResNet18 [16] backbone and the bottom session shows the results for ResNet50 backbone. In each category, the best results are marked in bold. To this end, we consider RSC [18] reproduced results instead of the reported ones.

α	β	Avg ↑
60	0.002	**83.42**
60	0.0005	82.20
60	0.008	82.75
40	0.002	82.58
80	0.002	82.51

Table 4: Hyperparameter study on the impact of α and β on average classification accuracy for multi-source domain generalization on PACS [24] dataset. All methods are implemented with ResNet18 [16] backbone.

4.2. VLCS Dataset

VLCS [43] is a classic benchmark for domain generalization, which contains five object classes (bird, car, chair, dog and person) from four domains (PASCAL VOC2007(V), LabelMe(L), Caltech(C) and SUN09(S)). Each dataset is viewed as one domain. We followed the experimental setting from [5] and [46] to train and evaluate our model. The feature extractor structure is AlexNet [21] pretrained on ImageNet [10]. Through cross validation search, the hyperparameters $\alpha = 100$ and $\beta = 0.0005$. Table 2 shows the results comparing DSFG with several domain generalization works in the literature. Our method achieves consistent improvement over state-of-the-art models, suggesting that our method can improve domain generalization. We see the same trends over the ablation models as before, with the meta-learning step again providing improvements.

4.3. PACS Dataset

PACS [24] includes seven object categories (dog, elephant, giraffe, guitar, horse, house, person) in four domains (Photo, Art Paintings, Cartoon and Sketches). We followed the existing experimental protocol [24, 5, 46] and did a leave-one-domain-out test. The domain shifts in PACS are larger than in the previous experiments. The feature extractor was set as ResNet18 and ResNet50 [16], pretrained on ImageNet [10], and the hyperparameters are set as $\alpha = 60$ and $\beta = 0.002$. Table 3 shows the results of all the competing methods with the ResNet18 and ResNet50 networks. We reproduced RSC [18] using the open-source implementation[1] published by the authors with the default configuration. Our method achieves better or comparable performance compared to state-of-the-art models on all of the domains. This suggests that our model is robust and can improve the prediction accuracy even when the domain gap is large and the network is deep. We see the same trends over the ablation models as before, with the meta-learning step again providing improvements.

Hyperparameter study: In order to show model sensitivity to hyper-parameters α and β, results with different hyper-parameters on the PACS dataset are shown in Table 4.

[1] https://github.com/DeLightCMU/RSC

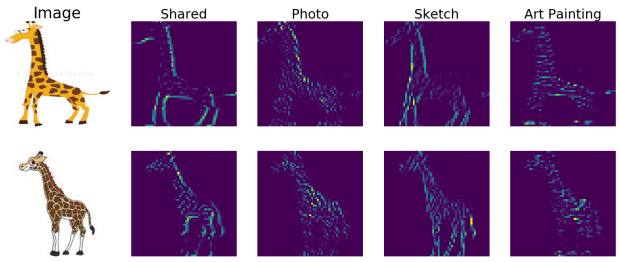


Figure 5: Visualization of selected feature activation maps from the first F-conv layer. The input image is from cartoon domain in PACS dataset with label 'giraffe'. The corresponding domain is shown on top.

We see the model is slightly sensitive to hyper-parameters, but all the results are comparable or better than the performance of state-of-the-art approaches.

Feature map visualization: In Figure 5, we visualize four selected features maps of two images from cartoon domain respectively. The training set includes photo, sketch and art painting domains. From the visualization we can see the unique properties caught by each filter. For example, the convolved output from sketch domain filter focuses on the boundaries (second column to the right), which is expected since sketches have greater information in the the object outlines. In contrast, the photo and art painting filters focus more on the interior textures. The different filters compensate for each other and the final output of F-conv layer is a combination of these features.

5. Conclusions

The proposed approach, DSFG, works well for domain generalization tasks by learning a weight generator that estimates domain similarity. Unlike unsupervised domain adaptation approaches, DSFG can give predictions to new data samples from a previously unobserved domain in real time, without further updates. Empirically, DSFG adapts to unseen test samples even when there is large domain distribution gap. The visualized weights shows that DSFG is effectively capturing and exploiting similarity between domains. Because of these properties, DSFG is effective for domain generalization tasks and can straightforwardly be implemented on many devices at test time.

Acknowledgements

The research was supported in part by DARPA, DOE, NIH, NSF and ONR. DC was supported by the National Institutes of Health under Award Number R01EB026937.

The contents of this manuscript are solely the responsibility of the authors and do not necessarily represent the official views of any of the funding agencies or sponsors.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, pages 3981–3989, 2016.

[2] Yogesh Balaji, Swami Sankaranarayanan, and Rama Chellappa. Metareg: Towards domain generalization using meta-regularization. In *NeurIPS*, pages 1006–1016, 2018.

[3] Felix JS Bragman, Ryutaro Tanno, Sebastien Ourselin, Daniel C Alexander, and Jorge Cardoso. Stochastic filter groups for multi-task cnns: Learning specialist and generalist convolution kernels. In *ICCV*, pages 1385–1394, 2019.

[4] Zhangjie Cao, Lijia Ma, Mingsheng Long, and Jianmin Wang. Partial adversarial domain adaptation. In *ECCV*, 2018.

[5] Fabio M Carlucci, Antonio D'Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. Domain generalization by solving jigsaw puzzles. In *CVPR*, 2019.

[6] Fabio Maria Carlucci, Paolo Russo, Tatiana Tommasi, and Barbara Caputo. Hallucinating agnostic images to generalize across domains. In *ICCV Workshops*, pages 3227–3234, 2019.

[7] Prithvijit Chattopadhyay, Yogesh Balaji, and Judy Hoffman. Learning to balance specificity and invariance for in and out of domain generalization. In *ECCV*, 2020.

[8] Ching-Yao Chuang, Antonio Torralba, and Stefanie Jegelka. Estimating generalization under distribution shifts via domain-invariant representations. In *ICML*, 2020.

[9] Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy. Optimal transport for domain adaptation. *IEEE TPAML*, 39(9):1853–1865, 2016.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[11] Qi Dou, Daniel Coelho de Castro, Konstantinos Kamnitsas, and Ben Glocker. Domain generalization via model-agnostic learning of semantic features. In *NeurIPS*, pages 6447–6458, 2019.

[12] Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. Unsupervised visual domain adaptation using subspace alignment. In *CVPR*, pages 2960–2967, 2013.

[13] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *JMLR*, 2016.

[14] Behnam Gholami, Pritish Sahu, Ognjen Rudovic, Konstantinos Bousmalis, and Vladimir Pavlovic. Unsupervised multi-target domain adaptation: An information theoretic approach. *IEEE TIP*, 29:3993–4002, 2020.

[15] Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. *ICLR*, 2021.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[17] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, pages 7132–7141, 2018.

[18] Zeyi Huang, Haohan Wang, Eric P. Xing, and Dong Huang. Self-challenging improves cross-domain generalization. In *ECCV*, pages 124–140, 2020.

[19] Fredrik D Johansson, Rajesh Ranganath, and David Sontag. Support and invertibility in domain-invariant representations. In *AISTATS*, 2019.

[20] Sameeksha Katoch, Kowshik Thopalli, Jayaraman J Thiagarajan, Pavan Turaga, and Andreas Spanias. Invenio: Discovering hidden relationships between tasks/domains using structured meta learning. *arXiv preprint arXiv:1911.10600*, 2019.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

[22] Damjan Krstajic, Ljubomir J Buturovic, David E Leahy, and Simon Thomas. Cross-validation pitfalls when selecting and assessing regression and classification models. *J. Cheminformatics*, 6(1):1–15, 2014.

[23] Jogendra Nath Kundu, Naveen Venkat, R Venkatesh Babu, et al. Universal source-free domain adaptation. In *CVPR*, pages 4544–4553, 2020.

[24] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *ICCV*, pages 5542–5550, 2017.

[25] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Learning to generalize: Meta-learning for domain generalization. In *AAAI*, 2018.

[26] Da Li, Jianshu Zhang, Yongxin Yang, Cong Liu, Yi-Zhe Song, and Timothy M Hospedales. Episodic training for domain generalization. In *ICCV*, pages 1446–1455, 2019.

[27] Yitong Li, Michael Murias, Samantha Major, Geraldine Dawson, and David E. Carlson. Extracting relationships by multi-domain matching. In *NeurIPS*, pages 6797–6808, 2018.

[28] Yitong Li, Michael Murias, Samantha Major, Geraldine Dawson, and David E Carlson. On target shift in adversarial domain adaptation. In *AISTATS*, 2019.

[29] Yiying Li, Yongxin Yang, Wei Zhou, and Timothy Hospedales. Feature-critic networks for heterogeneous domain generalization. In *ICML*, pages 3915–3924, 2019.

[30] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[31] Shikun Liu, Andrew Davison, and Edward Johns. Self-supervised generalisation with meta auxiliary learning. In *NeurIPS*, pages 1677–1687, 2019.

[32] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation with multiple sources. In *NIPS*, pages 1041–1048, 2009.

[33] Markus Näpflin, Marc Wildi, and Johannes Sarnthein. Test–retest reliability of resting eeg spectra validates a statistical signature of persons. *Clinical Neurophysiology*, 118(11):2519–2524, 2007.

[34] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE TNN*, 22(2):199–210, 2011.

[35] Xingang Pan, Ping Luo, Jianping Shi, and Xiaoou Tang. Two at once: Enhancing learning and generalization capacities via ibn-net. In *ECCV*, pages 464–479, 2018.

[36] Vihari Piratla, Praneeth Netrapalli, and Sunita Sarawagi. Efficient domain generalization via common-specific low-rank decomposition. In *ICML*, 2020.

[37] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2016.

[38] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, 2017.

[39] Ievgen Redko, Nicolas Courty, Rémi Flamary, and Devis Tuia. Optimal transport for multi-source domain adaptation under target shift. In *AISTATS*, pages 849–858, 2019.

[40] Mengye Ren, Renjie Liao, Ethan Fetaya, and Richard Zemel. Incremental few-shot learning with attention attractor networks. In *NeurIPS*, pages 5275–5285, 2019.

[41] Shiv Shankar, Vihari Piratla, Soumen Chakrabarti, Siddhartha Chaudhuri, Preethi Jyothi, and Sunita Sarawagi. Generalizing across domains via cross-gradient training. In *ICLR*, 2018.

[42] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *ICCV*, pages 3400–3409, 2017.

[43] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *CVPR*, pages 1521–1528, 2011.

[44] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *CVPR*, 2017.

[45] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. In *NeurIPS*, 2018.

[46] Shujun Wang, Lequan Yu, Caizi Li, Chi-Wing Fu, and Pheng-Ann Heng. Learning from extrinsic and intrinsic supervisions for domain generalization. In *ECCV*, pages 159–176, 2020.

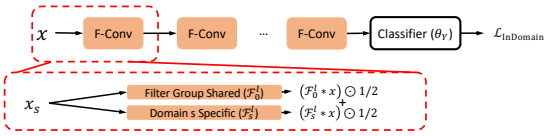
[47] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *ECCV*, pages 3–19, 2018.

[48] Huanhuan Yu, Menglei Hu, and Songcan Chen. Multi-target unsupervised domain adaptation without exactly shared categories. *arXiv preprint arXiv:1809.00852*, 2018.

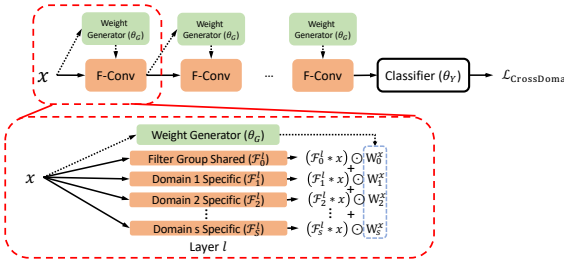
[49] Han Zhao, Shanghang Zhang, Guanhang Wu, José MF Moura, Joao P Costeira, and Geoffrey J Gordon. Adversarial multiple source domain adaptation. In *NeurIPS*, pages 8559–8570, 2018.

A. Generalized Model Form

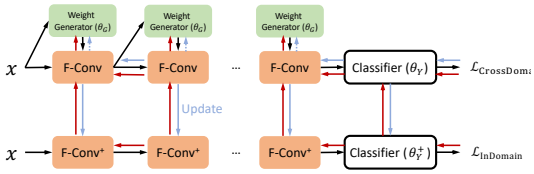
DSFG can be generalized with as many layers of F-Conv as desired. Figure 6 illustrates the general model.



(a) Illustration of in-domain training process for general model.



(b) Illustration of cross-domain training process for general model.



(c) Illustration of weight generator update process for general model.

Figure 6: Visualization of the training process for general case. Figure 6(a) illustrates in-domain training process, where each layer of F-conv follows the same weighting. Figure 6(b) illustrates cross-domain training process. Each sample in the training set will be used as a target in the cross-domain update as with a single layer of F-conv. The solid arrows give the gradient flow. Figure 6(c) illustrates the weight generator updates. The blue solid arrow indicates the gradient flow of the one-step gradient descent update in Eq.(8) and Eq.(9). The blue dotted arrow indicates the one-step gradient descent update does not update weight generator parameter $\boldsymbol{\theta}_G$. And the red arrow gives the gradient flow when updating the weight generator. The superscript $+$ indicates the one step gradient descent variable for meta-update.