PAPER

Model-agnostic Multi-Domain Learning with Domain-Specific Adapters for Action Recognition

Kazuki OMI[†], **Jun KIMATA**[†], ***Nonmembers*, *and* Toru TAMAKI**[†], ***Member***

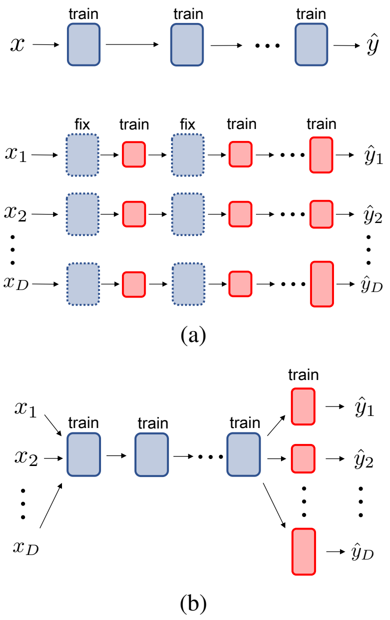
SUMMARY In this paper, we propose a multi-domain learning model for action recognition. The proposed method inserts domain-specific adapters between layers of domain-independent layers of a backbone network. Unlike a multi-head network that switches classification heads only, our model switches not only the heads, but also the adapters for facilitating to learn feature representations universal to multiple domains. Unlike prior works, the proposed method is model-agnostic and doesn't assume model structures unlike prior works. Experimental results on three popular action recognition datasets (HMDB51, UCF101, and Kinetics-400) demonstrate that the proposed method is more effective than a multi-head architecture and more efficient than separately training models for each domain.

key words: *multi domain learning, action recognition, domain-specific adapters, domain-independent layers, multi-head*

1. Introduction

Video recognition tasks [1], especially recognition of human actions, has become important in various real-world applications, and therefore many methods have been proposed. In order to train deep models, it is necessary to collect a variety of videos of human actions in various situations, therefore many datasets have been proposed [2]–[4]. The distribution of a training dataset is called *domain*, and the difference in distribution between two domains is called *domain shift* [5]–[7]. A domain is greatly characterized by the process of collecting the dataset of the domain, therefore, it is necessary to collect training samples in several different domains for recognizing actions in various situations. Usually recognition models are trained on a single given dataset (or domain) for performance evaluation, but they often face to the difficulty of performing well in a cross-dataset situation, which means that they perform well on samples of the same domain but don't well generalize on samples of other domains.

A possible approach might be domain adaptation (DA) [5]–[7]. DA approaches adapt a model trained on samples of a source domain to samples of a target domain in order to cope with situations where training and test domains are different. However, when there are more than two domains, it would be better to use Multi-Domain Learning (MDL) [8], [9], which built a single model that can be used in multiple domains. Recently, many-domain problems have been attracted their attention for images (such as Visual Decathlon [10] and Medical Segmentation Decathlon [11]) as well as videos (Video Pentathlon [12]). In these cases, MDL has advantages over pair-wise domain adaptation approaches as the number of domains increases.



(a)

(b)

Fig. 1: Two types of multi-domain learning architectures. (a) Adapter type: after pre-training of domain-independent parameters (blue), they are fixed and domain-specific parameters (red) are trained for each domain separately. (b) Multi-head type: domain-independent (blue) and domain-specific parameters (red) are trained simultaneously for all domains. Note that $(x_d, \hat{y}_d)$ are input and prediction of the sample from domain d.

MDL models have two types of trainable parameters; one is domain-*independent* parameters that are shared by all domains, and the other is domain-*specific* parameters such that different domains have different ones. A model with fewer domain-specific parameters will be computationally less expensive even when more domains are added, while more domain-independent parameters are expected to improve the ability to represent features common for different domains. There are two main architectures of MDL as shown in Figure 1; domain-specific and independent parameters are trained separately [9], [10], or simultaneously [13]. In the former, domain-independent parameters are fixed after pre-training and domain-specific parameters are trained on each domain separately. In the latter, all parameters are trained on multiple domains at once.

Manuscript received April 31, 2022.
Manuscript revised April 31, 2022.

[†] The author is with the Nagoya Institute of Technology, Gokisocho, Showa, Nagoya 466–0061, Japan

DOI: 10.1587/trans.E0.??.1

Action recognition involves a variety of domains, however, the development of MDL models has received less attention than image recognition tasks so far, although some DA methods for action recognition have been proposed [14]–[17]. It is important to develop MDL models for video recognition tasks because the computation cost of action recognition models often become large, and a single MDL model would be more efficient than using different models for different domains. In this paper, we propose a new MDL model for action recognition. The proposed method, inspired by the prior work [9], inserts adapters with domain-specific parameters between domain-independent layers. The contributions of this work are as follows;

- We propose a method of multi-domain learning for action recognition. To the best of the authors' knowledge, this is the first attempt at MDL for action recognition.
- Our proposed method uses adapters between layers, which can be applicable to many of existing action recognition models, unlike prior works [8], [10] that restrict the model to be a ResNet with resblocks.
- The proposed adapter has (2+1)D convolutions that processes temporal and spatial information jointly while reducing parameters.
- We show experimental results with three different datasets (HMDB51, UCF101, and Kinetics400) demonstrating the effectiveness of the proposed method.

2. Related Work

2.1 Action recognition and domain adaptation

Action recognition has been an actively studied topic [1] over the last two decades, and various models have been devised to capture the temporal information, such as X3D [18] with 3D CNN, as well as recent models [19] based on Vision Transformer [20]. However, they all require one model per domain and usually each dataset is used to train and validate models separately for performance evaluation.

Domain adaptation (DA) for action recognition has been studied to capture the difference of the appearance information as well as the temporal dynamics, which makes recognizing videos difficult compared to images. For example, TA^{3}N [14] introduces a domain discriminator to achieves an effective domain alignment with adversarial training. TCoN [15] uses a cross-domain attention module to avoid frames with low information content and focus on frames commonly important both in the source and target domains. SAVA [16] is a model that responds to human actions rather than the background for adapting domains with different backgrounds. MM-SADA [17] performs adaptation for each of RGB and optical flow domains. These DA approaches however don't handle more than two domains.

2.2 Multi-domain learning

To handle multiple domains, an approach similar to multi-task learning would be taken, that is, using multi-heads [13]. As shown in Fig.1(b), the model has a single feature extractor used for all domains and multiple classification heads for each domain. In this case, the feature extractor has domain-independent parameters, while each head has its own domain-specific parameters. However, as more domains are involved, it will become more difficult for a single extractor to extract universal features for multiple domains, particularly for complex video domains.

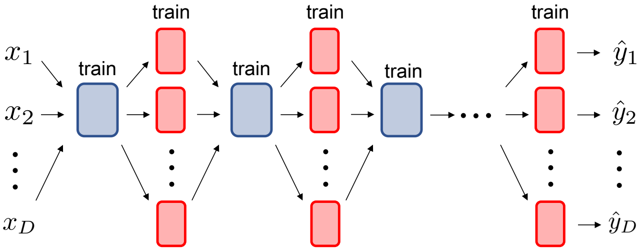


Fig. 2: An overview of our proposed method, which trains domain-independent backbone layers (blue) and domain-specific adapters (red) for all domains simultaneously. Unlike other multi-head and adapter models, this model has adapters for each domain between backbone layers.

Another approach is to insert adapters in a backbone network [8]–[10] as shown in Fig.1(a). First, the backbone model is pre-trained to fix the domain-independent parameters. Then adapters, which are domain-specific parameters, are inserted to the backbone network. Finally, the modified network is trained on each domain. One drawback of this approach is that the backbone network is assume to have a ResNet structure to insert adapters in parallel or series inside the resblocks [8], [10]. Hence it is difficult to apply the adapter approach to other models, even though a variety of pre-trained models [18], [21] are currently available. To alleviate this issue, CovNorm [9] doesn't assume model structures and inserts model-agnostic adapters between layers. However, the training is not end-to-end because adapters need the dimensionality reduction of features offline by principal component analysis.

In contrast, our method doesn't assume the model structure, like as [9], while the training is done in an end-to-end manner. In addition, the proposed method fine-tunes all the parameters during the training with multiple domains, whereas prior works using adapters [8], [10] have fixed pre-trained domain-independent parameters (of the backbone network) during the training with multiple domains.

3. Method

3.1 Architecture

Figure 2 shows the overview of the proposed method. The core idea is the use of adapters between layers like as CovNorm [9], but different adapters are used for different domains like as classification heads in a multi-head network [13]. First, we pre-train a backbone model that has N

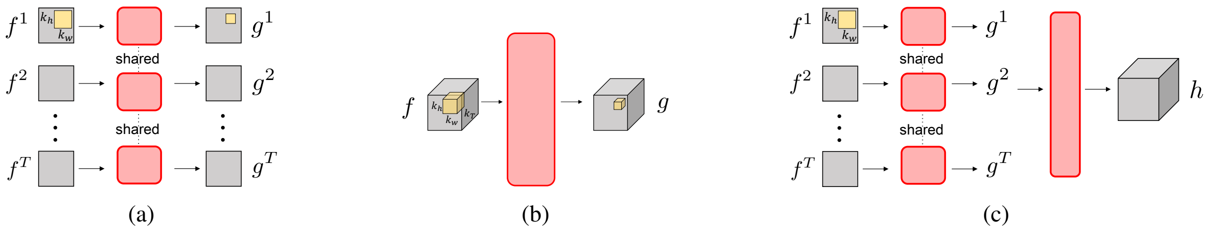


Fig. 3: Three types of adapters; (a) frame-wise 2D convolutions, (b) 3D convolution, and (c) (2+1)D convolution.

layers (or stages, blocks), each of which is shown as blue modules in Fig.2. This is the same with the top of Fig.1(a) where only the backbone model is shown.

Let M^ℓ be the ℓ-th layer in the backbone, that takes input $f^\ell \in \mathbb{R}^{T\times C^\ell \times H^\ell \times W^\ell}$ and output $f^{\ell+1} \in \mathbb{R}^{T\times C^{\ell+1} \times H^{\ell+1} \times W^{\ell+1}}$. Here H^ℓ and W^ℓ are spatial dimensions (width and height) of f^ℓ with C^ℓ channels. The first layer takes an input video clip $x = f^1 \in \mathbb{R}^{T\times 3\times H^1 \times W^1}$, where T is the number of frames in the clip, assuming that the layers doesn't change the temporal extent of the input. The last layer M^L predicts the softmax score $\hat{y} \in [0,1]^N$ of N categories. Using these notations, the backbone network is assume to be a type of stacking layers;

$$\hat{y} = M^L(M^{L-1}(\cdots M^2(M^1(x))\cdots)). \tag{1}$$

Note that this type is widely used in many architectures, such as 3D ResNet [22] and X3D [18].

Next, we insert adapter A_d^ℓ between layers M^ℓ and $M^{\ell+1}$ for $\ell = 1, \dots, L-2$. Thus the adapter takes $f^{\ell+1}$ and output a transformed feature $g^{\ell+1}$ of the same shape, which is then passed to the next layer $M^{\ell+1}$. Here d is the index of domains $d \in \{1, \dots, D\} = \mathcal{D}$. This means that we use different adapters A_d^ℓ for different domain d;

$$\hat{y}_d = M_d^L(M^{L-1}(A_d^{L-2}\cdots M^2(A_d^1(M^1(x_d)))\cdots))). \tag{2}$$

Note that we don't insert adapters just before the head M_d^L because the head itself is domain-specific.

As shown in Fig.2, when the network input is a sample $x_d = f_d^1$ from domain d, then the data passes through domain-specific adapters $A_d^1, A_d^2, \dots, A_d^{L-2}$ between the domain-independent backbone layers $M^1, M^2, \dots, M^{L-1}$ during the forward and backward computations. At the end of the network, there are multiple heads M_d^L, each for domain d, predicting scores $\hat{y}_d \in [0,1]^{N_d}$ where N_d is the number of categories in domain d. This is the same as the multi-head architecture (Fig.1(b)), but our method switches not only the heads but also the adapters for each domain depending on from which domain the sample comes.

3.2 Loss

Then, we train the whole network, that is, all of the domain-specific parameters (adapters A_d^ℓ and classification heads M_d^L) as well as the domain-independent parameters (backbone layers M_ℓ). Let $(x_{i,d}, y_{i,d})$ is i-th input-output pair of domain d. Note that domain d of each sample is given. Then, we minimize the following cross entropy loss;

$$L = E_{d\sim\mathcal{D}} E_{x,y\sim d}[L_{CE,d}(\hat{y}, y)] \tag{3}$$

$$\approx \sum_d \sum_i L_{CE,d}(\hat{y}_{i,d}, y_{i,d}), \tag{4}$$

assuming that the domain is sampled from a discrete uniform distribution.

Naively implementing this empirical loss is however inefficient when different samples come from different domains, causing the network to switch adapters for each sample. Instead, it would be more efficient if all samples in a batch come from the same domain because the forward computation of the batch uses adapters of the same domain without adapter switching. Therefore, we introduce the following loss to minimize for a sampled batch $\{x_i, y_i\}_{i=1}^B$;

$$L = E_{d\sim\mathcal{D}} E_{\{x_i,y_i\}_{i=1}^B \sim d}\left[\sum_{i=1}^B L_{CE,d}(\hat{y}_i, y_i)\right], \tag{5}$$

where B is the batch size.

In our implementation, a domain is selected sequentially (rather than randomly), and a batch is sampled from the domain, then the loss of the domain is computed. The gradient is updated only after batches sampled from all domains have been used for backward computations. In other words, parameters are only updated once after every D backward computations.

3.3 Spatio-temporal adapters

We proposed to use the following three types of adapters (i.e., 2D, 3D, and (2+1)D) that spatially and temporally transform features.

Frame-wise 2D conv. The 2D adapter performs convolutions for each frame separately. Let $f \in R^{T\times C\times H\times W}$ be the input feature, and $f^t \in R^{C\times H\times W}$ be the feature of t-th frame for $t = 1, \dots, T$. 2D adapters perform 2D convolution A_{2D} to each frame separately;

$$g^t = A_{2D} \otimes f^t, \tag{6}$$

where $\otimes$ represent convolutions. This is implemented by 3D convolutions A_{3D} with the kernel of size $R^{C\times 1\times k_h \times k_w}$;

$$g = A_{3D} \otimes f, \tag{7}$$

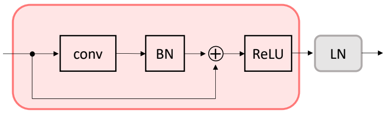


Fig. 4: The structure of adapters. The "conv" layer is either 2D, 3D, or (2+1)D convolutions.

to produce the output g.

3D conv. Unlike the 2D adapter that doesn't transform features temporally, the 3D adapter uses 3D convolution on the 3D video volume (Figure 3(b)). An adapter A_{3D} is applied as in the same with Eq.(7) with the kernel of size $R^{C \times k_t \times k_h \times k_w}$.

(2+1)D conv. 3D convolution is expected to model the temporal information of actions because it considers both spatial and temporal dimensions simultaneously. However, as the number of adapters increases with the number of domains, adapters are required having fewer parameters. Inspired by separable convolution [23]–[25], we introduce (2+1)D convolution adapters that use two convolutions in series; one for spatial and the other for temporal. First, frame-wise 2D convolutions with the kernel of size size $R^{C \times 1 \times k_h \times k_w}$ are applied;

$$g^t = A_{\mathrm{2D}} \otimes f^t, \text{ for } t = 1, \dots, T, \tag{8}$$

then a 1D convolution with the kernel of size $R^{C \times k_t \times 1 \times 1}$ aggregates the outputs of T frames along the temporal direction;

$$g = A_{\mathrm{1D}} \otimes [g^1, g^2, \dots, g^T]. \tag{9}$$

3.4 Adapter structure

Figure 4 shows the structure of adapters. Each adapter has a batch normalization (BN) after either of 2D, 3D, or (2+1)D convolutions described above, followed by skip connection and ReLU. In Fig.4, the red plate represents an adapter A_d^ℓ, that is switched for each domain d. In addition, we place a layer normalization (LN) as additional domain-independent parameters after the output of these adapters. Adapters output domain-specific features, which may differ for each domain. We expect LN to make the domain-specific adapter outputs more domain-independent for facilitating the training of the next layer.

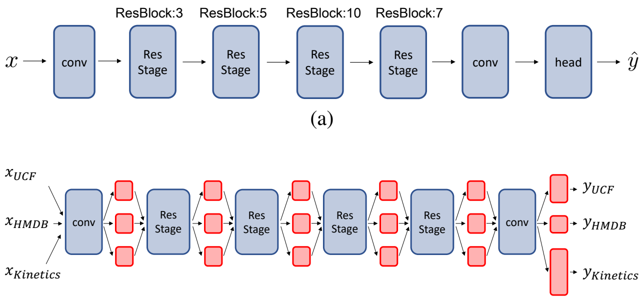
4. Experiments

We show experimental results using three domains, and compare the proposed method with multi-head and non-MDL approaches.

4.1 Setting

(1) Datasets

HMDB51 [4] consists of 3.6k videos in the training set and 1.5k videos in the validation set, with 51 human action categories. Each video is collected from movies, web, Youtube, etc., and the shortest video is less than 1 second and the longest is about 35 seconds, while most videos are between 1 and 5 seconds long, with an average length of 3.15 seconds. The first split was used in this experiment.



(a)

(b)

Fig. 5: Structures of (a) the backbone X3D-M, and (b) our model with adapters and heads for each domain.

UCF101 [3] consists of 9.5k videos in the training set and 3.5k videos in the validation set, with 101 human action categories. Each video was collected from Youtube, and the video length is 1 second for the shortest and 30 seconds for the longest, while most videos are between 3 and 10 seconds in length, with an average length of 7.21 seconds. There are three splits for training and validation, and we report the performance of the first split as it is usually used.

Kinetics400 [2] consists of 22k videos in the training set, 18k videos in the validation set, and 35k videos in the test set, with 400 human action categories. Each video was collected from Youtube and trimmed to a 10 second long segment corresponding to one of the action categories.

(2) Model

We used X3D-M [18] pre-trained on Kinetics400 as the backbone network. Figure 5(a) shows the structure of X3D-M, which has the stem conv block M^1, followed by four ResBlock stages $M^2, \dots, M^5$ and a conv block M^6, and finally a classification head M^7 (hence $L = 7$). The proposed model used in the experiments is shown in Fig.5(b). This model has $D = 3$ classification heads M_d^7 at the end of the network, and five adapters per domain A_d^ℓ (for $\ell = 1, \dots, 5$) between the backbone modules.

We used the following adapter parameters. For frame-wise 2D conv, the kernel size was $k_h \times k_w = 3 \times 3$. For 3D conv, the kernel size was $k_t \times k_h \times k_w = 3 \times 3 \times 3$. For (2+1)D conv, the kernel size for spatial convolution was $k_h \times k_w = 3 \times 3$ and for temporal convolution $k_t = 3$.

In the following experiments, unless otherwise denoted, we used the (2+1)D adapters, which were inserted to all five locations (the "all" row in Tab.3), for the proposed method (Fig.2, the "train&train" row in Tab.2) with X3D-M as a backbone.

(3) Training

Training video files of different datasets differ in fps. Therefore, we used the following protocol, following `pytorchvideo` [26]. From one video in the training set, we randomly extracted consecutive frames corresponding to a specified duration starting from a randomly decided position, and created a clip by sampling 16 frames uniformly from the extracted frames. We used the duration of about 2.67 seconds (corresponding to 80 frames in 30 fps) because of the setting of X3D-M (using 16 frames with a stride of 5 frames). The short sides of extracted frames were randomly resized to [224, 320] pixels and resized while maintaining the aspect ratio. Then we randomly cropped 224×224 pixels and flipped them horizontally with a probability of 50%.

The backbone X3D-M model were pre-trained on Kinetics400, so were the domain-independent parameters. We trained from scratch the adapters and heads (domain-specific parameters), as well as LN layers (added as domain-independent parameters).

The term "epoch" doesn't make sense because we train the models on three datasets simultaneously and different datasets have different number of samples. Therefore, in the experiments, we trained models for 42,000 iterations, corresponding to 14,000 iterations for each dataset. The batch size was set to 32, therefore the effective numbers of training epochs were about 2 for Kinetics400, 48 for UCF101, and 128 for HMDB51. The input clips for training were taken from the three datasets in turn for each batch. In other words, the first batch of 32 clips was taken from the first dataset, the second batch was taken from the second dataset, the third batch was taken from the third dataset, and so on, for 42,000 batches. When training a batch of dataset d, the batch is passed through adapters A_d^ℓ and head M_d^L, as well as domain-independent layers M^ℓ, to compute the loss $L_{CE,d}$. The gradient is back-propagated using layers and adapters only used in the forward computation. However, parameters are not updated until the gradients of batches of three datasets have been back-propagated. In this experiment, parameters were updated once every three batches, each from three different datasets.

We used an SGD optimizer with momentum of 0.9. The initial learning rate was set to 0.001 and reduced to 1/10 at 8,000 and 12,000 iterations.

(4) Inference

In validation, we performed a multi-view test as in prior works [27]. For each video in the validation set, we repeated the clip sampling 10 times to sample 10 clips. Then we resized the frames while maintaining the aspect ratio so that the short side was 256 pixels, and cropped to 224×224 at the right, center, and left. This generated 30 clips (30 views), and we averaged these results to compute a single prediction score.

Table 1: The top-1 performance with different adapter types. Number of parameters are also shown for the backbone model (base), heads, and adapters.

					params (M)			
adapter	HMDB	UCF	K400	average	total	base	head	adap.
2D	73.07	95.93	69.94	79.65	5.45	2.97	1.13	1.34
(2+1)D	74.77	96.25	69.84	80.29	5.89	2.97	1.13	1.79
3D	75.03	95.77	70.08	80.29	8.12	2.97	1.13	4.02

Table 2: The top-1 performances by fixing or fine-tuning domain-independent layers. Note that the number of trainable parameters are shown. The first row corresponds to the architecture shown in Fig.1(a), and the second row to Fig.2.

						params (M)			
M^ℓ	A_d^ℓ, M_d^L	HMDB	UCF	K400	average	total	base	head	adap.
fix	train	73.07	95.19	67.54	78.60	2.91	—	1.13	1.79
train	train	74.77	96.25	69.84	80.29	5.89	2.97	1.13	1.79

4.2 Results

4.2.1 Adapter types

First we compare three types of adapters. Table 1 shows the performances for each adapter type. As expected, 3D and (2+1)D adapters performed better than 2D adapters because of the ability to model the temporal information. In the following experiments, we used (2+1)D conv because it has fewer parameters while both 3D and (2+1)D performed similarly.

4.2.2 Fixing or fine-tuning domain-independent parameters

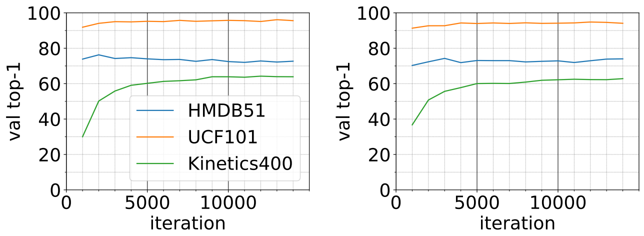
In the prior works with adapters [8]–[10], the domain-independent parameters of the backbone were pre-trained on some domain, then fixed during training with multiple domains. In contrast, our model fine-tunes those parameters to jointly train with adapters. Table 2 shows the performance comparison of these two settings. The first row shows the performance of our model with adapters inserted, but the domain-independent backbone layers were not trained during multi-domain learning. As expected, the performance is better when all parameters are trained jointly, indicating that training adapters only is insufficient to support multiple domains. The backbone layer should extracts more generic domain-independent features, which makes the feature transformation with adapters more effective.

4.2.3 Adapter locations in the backbone

Here we investigate the different configurations of adapter insertion. Table 3 shows the performances by changing positions where we insert adapters in the backbone model. "Early-x" used adapters $A_d^1, \dots, A_d^x$ between the early layers of the backbone, while "late-x" inserted adapters

Table 3: The top-1 performance with different adapter configurations for the validation sets. The row "multi head" corresponds to the architecture shown in Fig.1(b).

					params (M)			
# config	HMDB	UCF	K400	average	total	base	head	adap.
early-1	74.77	96.38	71.00	80.72	4.13	2.97	1.13	0.02
early-3	74.64	96.19	70.75	80.53	4.23	2.97	1.13	0.13
late-3	74.90	95.90	70.45	80.42	5.85	2.97	1.13	1.75
late-1	73.27	96.03	70.86	80.29	5.44	2.97	1.13	1.33
multi-head	73.99	96.25	70.62	79.98	4.11	2.97	1.13	—
all	74.77	96.25	69.84	80.29	5.89	2.97	1.13	1.79



(a)

(b)

Fig. 6: Performance over training epochs of using (a) a single domain, or (b) three domains.

$A_d^{\ell-2-(x-1)}, \ldots, A_d^{\ell-2}$ between the late layers. These configurations also have domain-specific heads M_d^L, but "multi-head" is the case using only the heads but no adapters. "All" is the full model that uses all of the adapters.

On average, the multi-head type shows the least performance, indicating that that domain-specific parameters are needed not only at the final heads, but also between layers as adapters. The best performance was obtained by early-1, which has the first adapter A_d^1 only in addition to the heads as domain-specific parameters. As the positions of adapters inserted in the backbone becomes deeper, the performance deteriorates gradually, which is consistent with the fact that the multi-head has domain-specific parameters only at very the end of the network.

The prior work [8] has reported that better performances were obtained when adapters were inserted in the late layers rather than the early layers. The differences between our work and theirs are that videos come from similar datasets, all the parameters are trained jointly, and a specific backbone model is not assumed. Three datasets in this experiments have similar categories, and most videos were taken from third-person views. Therefore adapters in the early layers might be enough to transform low-level temporal information of videos in these datasets. We would have different results with other datasets of first-person views, such as SSv2 [28] and Epic-Kitchen [29], which are significantly different domains. Another factor may be the use of X3D pre-trained on Kinetics as the backbone. Its structure was explored in a greedy way, so adding adaptors and heads for multiple domains may be suboptimal.

4.2.4 Number of domains

In MDL, the number of domains is the important factor. Table 4(a) shows the results when different numbers of domains were involved for the "all" configuration in Tab.3, and Table 4(b) for the "early-1" configuration. Rows of "# domains 1" are the cases using a single domain, which means that the network have adapters between layers and a single head, and is trained on the domain only. The performance of HMDB increases as more domains are used, demonstrating that MDL is beneficial for smaller datasets by leveraging information from other larger datasets. In contrast, performances of UCF and Kinetics decreases when other datasets were used. In particular, performances dropped significantly when HMDB, the smallest one, was used jointly as shown in rows of "# domains 2". This issue of dataset sizes may caused by several factors. Currently we assume that the domain was sampled from a uniform distribution, regardless of the dataset size, as in Eq.(4). Also we minimize the sum of losses of different datasets without any weights. We would investigate the effects of these factors in future, by introducing non-uniform domain distributions or importance sampling.

Figure 6 shows the performance of the validation sets of three datasets when the network was trained on a single domain ("# domains 1" in Tab.4(a)) or on three domains ("# domains 3"). Note that the validation performance is of a single view (not 30 views as mentioned before), and horizontal axes Fig.6(a) and (b) should be interpreted differently as in Fig.6(b) a single iteration refers to a single gradient update after back-propagation of three domains. The performance of HMDB deteriorates as training progresses when trained on a single domain, but this is not the case when trained on multiple domains. This is in agreement with the observation in Tab.4(a) above.

Note that Tab.4(a) also shows the performances of the backbone network without any adapters in rows with "—" in the domain column. This shows that our model with adapters doesn't work better than the backbone itself, even for a single domain. This might be due to the increase of the parameters to be trained while fixing the training iterations. But we should note that three backbone networks are needed for three domains to train separately and have more parameters (about 10M in total), whereas our method requires a single model of fewer parameters (about 5.8M).

4.3 Using another backbone model

Our proposed method is model-agnostic and applicable to any models that have a structure like Eq.(1). We have used X3D-M [18] in the experiments above, and here we show results of 3D ResNet [22] pre-trained on Kinetics400. It has the same structure with X3D-M; it has the stem conv block M^1, four ResBlock stages $M^2, \ldots, M^5$, a conv block M^6, and a head M^7 (hence $L = 7$). As like in Fig.5(b), this model has a head and five adapters per domain for the "all" configuration.

Results shown in Tab.5(a) show that the all configu-

Table 4: Effect of the number of domains on the top-1 performance with the X3D backbone of (a) the all and (b) early-1 configurations. Note that "—" in the left-most column shows the cases using no adapters.

				params (M)			
# dom	HMDB	UCF	K400	total	base	head	adap.
—	75.62	—	—	3.08	2.97	0.10	—
—	—	97.28	—	3.18	2.97	0.21	—
—	—	—	72.43	3.79	2.97	0.82	—
1	73.27	—	—	3.68	2.97	0.10	0.60
1	—	96.88	—	3.78	2.97	0.21	0.60
1	—	—	71.80	4.39	2.97	0.82	0.60
2	74.58	95.90	—	4.48	2.97	0.31	1.19
2	74.25	—	70.21	5.10	2.97	0.92	1.19
2	—	96.34	70.77	5.19	2.97	1.03	1.19
3	74.77	96.25	69.84	5.89	2.97	1.13	1.79

(a)

				params (M)			
# dom	HMDB	UCF	K400	total	base	head	adap.
1	74.18	—	—	3.09	2.97	0.10	0.01
1	—	96.67	—	3.19	2.97	0.21	0.01
1	—	—	72.00	3.80	2.97	0.82	0.01
2	73.73	96.27	—	3.30	2.97	0.31	0.01
2	74.44	—	70.99	3.91	2.97	0.92	0.01
2	—	96.56	71.48	4.02	2.97	1.03	0.01
3	74.77	96.38	71.00	4.13	2.97	1.13	0.02

(b)

Table 5: Effect of the number of domains on the top-1 performance with the 3D ResNet backbone of (a) the all and (b) early-1 configurations. Note that "—" in the left-most column shows the cases using no adapters.

				params (M)			
# dom	HMDB	UCF	K400	total	base	head	adap.
—	73.07	—	—	31.74	31.63	0.10	—
—	—	96.11	—	31.84	31.63	0.21	—
—	—	—	70.95	32.45	31.63	0.82	—
1	66.34	—	—	81.93	31.63	0.10	50.19
1	—	94.18	—	82.03	31.63	0.21	50.19
1	—	—	69.12	82.65	31.63	0.82	50.19
2	65.56	92.89	—	132.33	31.63	0.31	100.38
2	64.71	—	67.99	132.94	31.63	0.92	100.38
2	—	93.52	68.27	133.04	31.63	1.03	100.38
3	63.33	93.29	67.80	183.34	31.63	1.13	150.58

(a)

				params (M)			
# dom	HMDB	UCF	K400	total	base	head	adap.
1	73.59	—	—	31.78	31.63	0.10	0.04
1	—	95.66	—	31.88	31.63	0.21	0.04
1	—	—	70.77	32.49	31.63	0.82	0.04
2	71.57	94.82	—	32.02	31.63	0.31	0.08
2	71.18	—	69.98	32.63	31.63	0.92	0.08
2	—	96.14	70.22	32.74	31.63	1.03	0.08
3	71.90	96.30	69.39	32.88	31.63	1.13	0.11

(b)

ration considerably increases the number of parameters because 3D ResNet have larger channels (1024) than X3D-M (192), which leads to 150 times more parameters of adapters for 3 domains, and the deterioration of the performance.

In contrast, the early-1 configuration shown in Tab.5(b) have fewer parameters and better performance. Again, this observation supports the discussion in Sec.4.2.3 that the early layers play an important role for transforming low-level temporal information of different domains.

5. Conclusion

In this paper, we propose a multi-domain learning model for action recognition that inserts domain-specific adapters between layers. The proposed method enables an end-to-end learning with multiple domains simultaneously, and experimental results showed that the proposed methods is more effective than a multi-head architecture, and more efficient than training a model for each domain separately. Our future work includes the further investigation on the inserting locations and structures of adapters to facilitate extracting common features across different domains, as well as domain-specific features suitable for each domain. In addition, other datasets [28], [29] which are largely different from datasets used in the experiments of this paper, are planned to be used for further experiments.

Acknowledgement

This work was supported in part by JSPS KAKENHI Grant Number JP22K12090.

References

[1] M.S. Hutchinson and V.N. Gadepally, "Video action understanding," IEEE Access, vol.9, pp.134611–134637, 2021.

[2] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman, and A. Zisserman, "The kinetics human action video dataset," CoRR, vol.abs/1705.06950, 2017.

[3] K. Soomro, A.R. Zamir, and M. Shah, "UCF101: A dataset of 101 human actions classes from videos in the wild," CoRR, vol.abs/1212.0402, 2012.

[4] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "Hmdb: a large video database for human motion recognition," 2011 International conference on computer vision, pp.2556–2563, IEEE, 2011.

[5] G. Wilson and D.J. Cook, "A survey of unsupervised deep domain adaptation," ACM Transactions on Intelligent Systems and Technology (TIST), vol.11, no.5, pp.1–46, 2020.

[6] I. Redko, E. Morvant, A. Habrard, M. Sebban, and Y. Bennani, "A survey on domain adaptation theory," CoRR, vol.abs/2004.11829, 2020.

[7] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," Neurocomputing, vol.312, pp.135–153, 2018.

[8] S.A. Rebuffi, H. Bilen, and A. Vedaldi, "Efficient parametrization of multi-domain deep neural networks," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

[9] Y. Li and N. Vasconcelos, "Efficient multi-domain learning by covariance normalization," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[10] S.A. Rebuffi, H. Bilen, and A. Vedaldi, "Learning multiple visual domains with residual adapters," Advances in Neural Information Processing Systems, ed. I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Curran Associates,

Inc., 2017.

[11] M. Antonelli, A. Reinke, S. Bakas, K. Farahani, A. Kopp-Schneider, B.A. Landman, G. Litjens, B.H. Menze, O. Ronneberger, R.M. Summers, B. van Ginneken, M. Bilello, P. Bilic, P.F. Christ, R.K.G. Do, M. Gollub, S. Heckers, H.J. Huisman, W.R. Jarnagin, M. McHugo, S. Napel, J. Goli-Pernicka, K.S. Rhode, C. Tobon-Gomez, E. Vorontsov, J.A. Meakin, S. Ourselin, M. Wiesenfarth, P. Arbelaez, B. Bae, S. Chen, L.A. Daza, J. Feng, B. He, F. Isensee, Y. Ji, F. Jia, N. Kim, I. Kim, D. Merhof, A. Pai, B. Park, M. Perslev, R. Rezaiifar, O. Rippel, I. Sarasua, W. Shen, J. Son, C. Wachinger, L. Wang, Y. Wang, Y. Xia, D. Xu, Z. Xu, Y. Zheng, A.L. Simpson, L. Maier-Hein, and M.J. Cardoso, “The medical segmentation decathlon,” CoRR, vol.abs/2106.05735, 2021.

[12] S. Albanie, Y. Liu, A. Nagrani, A. Miech, E. Coto, I. Laptev, R. Sukthankar, B. Ghanem, A. Zisserman, V. Gabeur, C. Sun, K. Alahari, C. Schmid, S. Chen, Y. Zhao, Q. Jin, K. Cui, H. Liu, C. Wang, Y. Jiang, and X. Hao, “The end-of-end-to-end: A video understanding pentathlon challenge (2020),” CoRR, vol.abs/2008.00744, 2020.

[13] S. Masaki, T. Hirakawa, T. Yamashita, and H. Fujiyoshi, “Multi-domain semantic-segmentation using multi-head model,” IEEE International Conference on Intelligent Transportation Systems, 2021.

[14] M.H. Chen, Z. Kira, G. AlRegib, J. Yoo, R. Chen, and J. Zheng, “Temporal attentive alignment for large-scale video domain adaptation,” Proceedings of the IEEE/CVF International Conference on Computer Vision, pp.6321–6330, 2019.

[15] B. Pan, Z. Cao, E. Adeli, and J.C. Niebles, “Adversarial cross-domain action recognition with co-attention,” Proceedings of the AAAI Conference on Artificial Intelligence, pp.11815–11822, 2020.

[16] J. Choi, G. Sharma, S. Schulter, and J.B. Huang, “Shuffle and attend: Video domain adaptation,” European Conference on Computer Vision, pp.678–695, Springer, 2020.

[17] J. Munro and D. Damen, “Multi-modal domain adaptation for fine-grained action recognition,” IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2020.

[18] C. Feichtenhofer, “X3d: Expanding architectures for efficient video recognition,” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2020.

[19] S.H. Khan, M. Naseer, M. Hayat, S.W. Zamir, F.S. Khan, and M. Shah, “Transformers in vision: A survey,” CoRR, vol.abs/2101.01169, 2021.

[20] R. Girdhar, J. Carreira, C. Doersch, and A. Zisserman, “Video action transformer network,” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[21] C. Feichtenhofer, H. Fan, J. Malik, and K. He, “Slowfast networks for video recognition,” Proceedings of the IEEE/CVF international conference on computer vision, pp.6202–6211, 2019.

[22] K. Hara, H. Kataoka, and Y. Satoh, “Learning spatio-temporal features with 3d residual networks for action recognition,” Proceedings of the IEEE International Conference on Computer Vision Workshops, pp.3154–3160, 2017.

[23] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, “A closer look at spatiotemporal convolutions for action recognition,” Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp.6450–6459, 2018.

[24] Z. Qiu, T. Yao, and T. Mei, “Learning spatio-temporal representation with pseudo-3d residual networks,” ICCV, 2017.

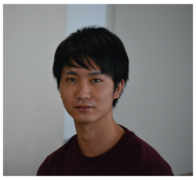
[25] S. Xie, C. Sun, J. Huang, Z. Tu, and K. Murphy, “Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification,” Proceedings of the European Conference on Computer Vision (ECCV), September 2018.

[26] H. Fan, T. Murrell, H. Wang, K.V. Alwala, Y. Li, Y. Li, B. Xiong, N. Ravi, M. Li, H. Yang, J. Malik, R. Girshick, M. Feiszli, A. Adcock, W.Y. Lo, and C. Feichtenhofer, “PyTorchVideo: A deep learning library for video understanding,” Proceedings of the 29th ACM International Conference on Multimedia, 2021. `https://pytorchvideo.org/`.

[27] X. Wang, R. Girshick, A. Gupta, and K. He, “Non-local neural networks,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

[28] R. Goyal, S. Ebrahimi Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, *et al.*, “The" something something" video database for learning and evaluating visual common sense,” Proceedings of the IEEE international conference on computer vision, pp.5842–5850, 2017.

[29] D. Damen, H. Doughty, G.M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, *et al.*, “Scaling egocentric vision: The epic-kitchens dataset,” Proceedings of the European Conference on Computer Vision (ECCV), pp.720–736, 2018.



Kazuki Omi received B.E. from Nagoya Institute of Technology in 2022. His research interests include computer vision and action recognition.

Jun Kimata received B.E. from Nagoya Institute of Technology in 2022. His research interests include computer vision and action recognition.

Toru Tamaki received his B.E., M.S., and Ph.D. degrees in information engineering from Nagoya University, Japan, in 1996, 1998 and 2001, respectively. After being an assistant professor at Niigata University, Japan, and an associate professor at Hiroshima University, Japan, he is currently a professor at the Department of Computer Science, Nagoya Institute of Technology, Japan. He was an associate researcher at ESIEE Paris, France, in 2015. His research interests include computer vision, image recognition, machine learning, and medical image analysis.