DA^3: Dynamic Additive Attention Adaption for Memory-Efficient On-Device Multi-Domain Learning

Li Yang, Adnan Siraj Rakin and Deliang Fan
School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287
lyang166@asu.edu, asrakin@asu.edu, dfan@asu.edu

Abstract

Nowadays, one practical limitation of deep neural network (DNN) is its high degree of specialization to a single task or domain (e.g., one visual domain). It motivates researchers to develop algorithms that can adapt DNN model to multiple domains sequentially, while still performing well on the past domains, which is known as multi-domain learning. Almost all conventional methods only focus on improving accuracy with minimal parameter update, while ignoring high computing and memory cost during training, which makes it difficult to deploy multi-domain learning into more and more widely used resource-limited edge devices, like mobile phone, IoT, embedded system, etc. During our study in multi-domain training process, we observe that large memory used for activation storage is the bottleneck that largely limits the training time and cost on edge devices. To reduce training memory usage, while keeping the domain adaption accuracy performance, we propose ***Dynamic Additive Attention Adaption*** *(DA^3), a novel memory-efficient on-device multi-domain learning method. DA^3 learns a novel additive attention adaptor module, while freezing the weights of the pre-trained backbone model for each domain. Differentiating from prior works, our proposed DA^3 module not only mitigates activation memory buffering for reducing memory usage during training, but also serves as dynamic gating mechanism to reduce the computation cost for fast inference. We validate DA^3 on multiple dataset against state-of-the-art methods, which shows great improvement in both accuracy and training time. Moreover, we deploy DA^3 into the popular NIVDIA Jetson Nano edge GPU, where the measured experimental results show our proposed DA^3 reduces the on-device training memory consumption by* ***5-37***$\times$*, and training time by* ***2***$\times$*, in comparison to the baseline methods (e.g., standard fine-tuning, Parallel and Series Res. adaptor, Piggyback and TinyTL).*

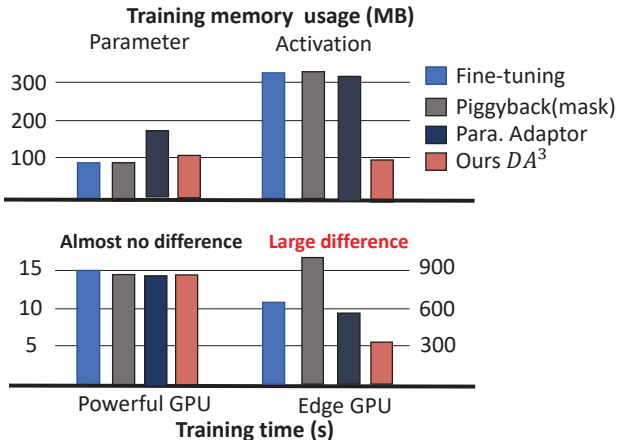

Figure 1. An example of adapting ResNet50 (pre-trained on ImageNet dataset) to Flower dataset [16]. *Top:* model parameters and activation memory of three different methods. *Bottom:* training time of one epoch on two different platforms: one powerful GPU (Quadro RTX 5000) and one edge GPU (Jetson Nano)

1. Introduction

Nowadays, one practical limitation of deep neural network (DNN) is its high degree of specialization to a single task or domain (e.g., one visual domain). It motivates researchers to develop algorithms that can adapt DNN model to multiple domains sequentially, while still performing well on the past domains. This process of gradually adapting DNN model to learn from different domain inputs over time is known as *multi-domain learning*. Nowadays, the utilization of IoT devices is greatly increased (e.g., 250 billion microcontrollers in the world today[1]), that collect massive new data crossing various domains/tasks in our daily life. To process the new data, a general way is to perform learning/training on cloud servers, and then transfer the learned DNN model back to IoT/edge devices for inference only. However, such method (i.e., *learning-on-*

[1]https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/

cloud and inference-on-device) is inefficient or unacceptable due to the huge communication cost between cloud and IoT/edge devices, as well as data-privacy concern (e.g., sensitive health care application). These challenges lead to a recently rising research direction about 'on-device multi-domain learning'.

Conventional multi-domain learning methods can be mainly divided into three directions: fine-tuning based method, adaptor-based method, and mask-based method. As the first approach, inspired by the success of transfer learning, fine-tuning [4, 11] is a natural approach to optimize the whole pre-trained model from old domains to new target domains. However, the training cost is huge since all the parameters need to be updated, and the overall size of parameters will increase linearly w.r.t the number of domains. One alternative method is to only fine-tune the batchnorm and last classifier, but suffering from limited domain adaption capacity [15]. In the second approach, [2, 17, 18] propose an adapter-based method, which learns a domain-specific residual adaptor while freezing the pre-trained model. In addition, such method also needs to fine-tune the batchnorm layer of the pre-trained model to avoid domain shift. Different from that, Piggyback [13], as a representative work of the third mask-based learning approach, proposes to only learn a binary element-wise mask ($\{0, 1\}$) w.r.t all weights, while keeping the pre-trained model fixed.

To investigate the training cost of on-device multi-domain learning, we tested three representative methods on these three directions respectively, in both powerful GPU (Nvidia RTX5000 used in desktop or cloud server training) and edge GPU (Nvidia Jetson Nano GPU used in edge device training). The measured training memory usage and training time are shown in Fig.1.

Observation 1. *training process is memory-intensive, where the* ***intermediate activation buffering in memory during back-propagation*** *is the bottleneck (at least 3X more than model itself as shown in Fig.1), to limit the speed of on-edge-device learning.*

During training, the memory usage for activation storage (defined as *activation memory* in this work) is almost 3X larger than the model itself. Such large training memory is not an issue (assuming with the same training time) in a powerful GPU with large enough memory capacity. However, for memory-limited edge GPU typically used in edge device training, such large memory usage becomes the bottleneck to limit training speed, and correspondingly leading to significantly different training speeds across different training methods for the same network and dataset as shown in Fig.1. Almost all prior domain adaption schemes only emphasize improving accuracy with minimal parameter update, while ignoring the computing- and memory-intensive nature of their methods, which makes it in-efficient to deploy into resource-limited edge-based training devices, like mobile phones, embedded system, IoT, etc.

In this work, we propose *Dynamic Additive Attention Adaption (DA^3)*, a new training scheme for memory-efficient on-device multi-domain learning (simplified as on-device learning in this work). Differentiating from prior works, our DA^3 is designed to eliminate the storage of intermediate activation feature map (i.e., dominating memory usage during on-device learning) to greatly reduce overall memory usage. Furthermore, to improve the adaption accuracy performance, DA^3 is embedded with a novel *dynamic additive attention adaptor* module, which is not only designed to avoid activation buffering for memory saving during training, but also reduces the computation cost through a dynamic gating mechanism. In summary, our technical contributions include:

- First, we present a complete analysis of memory consumption during training to prove that activation memory buffering is the key memory bottleneck during on-device multi-domain learning. More importantly, based on this analysis, we further discover an important observation to guide our design: the complete activation map (i.e., dominating memory usage) needs to be stored for backward propagation during training if it has **multiplicative relationship** with learned parameters (i.e., weight, mask), while the **additive relationship** (i.e., bias) is activation free.

- Motivated by our memory usage analysis, we propose a novel training method, called *Dynamic Additive Attention Adaption (DA^3)*, for memory-efficient on-device multi-domain learning. The main idea of DA^3 is that it freezes the parameters which have a multiplicative relationship with input activation, and only updates the learnable parameters that have an additive relationship. By doing so, there is no need to store the memory-dominating activation feature map during backward propagation. Moreover, to further enrich the adaption capacity, we propose a novel *additive attention adaptor* module that not only follows the additive principle to eliminate dominating activation memory buffering, but also implements a dynamic gating mechanism to reduce inference computation complexity. Such adaptor can plug in and play on any popular backbone model architectures for memory-efficient multi-domain learning.

- We conduct extensive experiments of the proposed DA^3 method comparing with prior competitive baselines. DA^3 could achieve state-of-the-art accuracy on popular multi-domain adaption dataset. More importantly, unlike previous methods, we, for the first time, test the training cost (in terms of time and memory usage) on an edge GPU (i.e., NVIDIA Jetson Nano) to

prove DA^3 could greatly reduce both the training time and memory in real device. The experimental results show that DA^3 reduces the on-device training memory consumption by *5-37*$\times$ and actual training time by *2*$\times$ in comparison to the baseline methods (e.g., standard fine-tuning [7], Parallel and Series Res. adaptor [17, 18], Piggyback [13] and TinyTL [2])

2. Related Work

2.1. Multi-Domain Learning

Multi-domain learning [4, 14, 17–19, 24] aims to build a model, which can adapt a task into multiple domains without forgetting previous knowledge, meanwhile learning as few parameters as possible. Series Res. Adaptor [17] addresses this challenge by learning additional residual adaptor for each layer while freezing the original pre-trained model except batch norm layer. Based on the same idea, the authors further update the topology of the residual adapter to be parallel rather than serial [18], resulting in better performance. Furthermore, [1] proposes Budget-ware Adaptor which aims to reduce the model parameters to enable efficient inference, but has no benefit for training. [24] achieves training-efficiency by reducing the training run-times for different tasks. However, the training procedure is memory-intensive, which is impractical for resource-limited on-device learning. [19] proposes to recombine the weights of the backbone model via controller modules in channel-wise. In short, all these methods above tackle the multi-domain challenge by learning additional domain-specific modules. Different from that, Piggyback [13] learns task-specific binary masks for each task. It achieves this by first generating the real-value masks which own the same size with weights, then passing through a binarization function to obtain binary masks that are then applied to existing weights. Furthermore, [14, 26] combine the binary mask with additional reparametrization methods to increase adaption capacity, but suffering from even more computation and memory cost during the training procedure.

2.2. Memory-Efficient Training

There are two conventional techniques to reduce the activation memory: gradient checkpointing [3] and reversible network [6]. Gradient checking only stores a subset of the network activations instead of all. As a consequence, the activations that are not stored must be recomputed during the backward pass. Furthermore, reversible network achieves backpropagation without storing activations, however, that means each layer's activations have to be recomputed exactly from that of the next layer. These two methods reduce the activation memory usage by involving additional computation. As mentioned in [21], for a 30% increase in computing overhead, checkpointing can reduce the memory required for the activations by 5.8x. [6] also shows that the reversible network has roughly 50% computational overhead than ordinary backpropagation in practice. More recently, TinyTL [2] proposes a lite residual learning module to lower the activation memory in transfer learning. However, this method involves additional training and searching procedure to find the suitable sub-network architecture for each target task.

2.3. Attention and Dynamic Mechanism

Attention mechanism has proven to be a promising method to enhance DNN accuracy. SE-Net [9] is the first that presents an effective mechanism to learn channel attention and achieves good performance. Later, CBAM [25] combines channel and spatial attention to enhance feature aggregation. Furthermore, inspired by the attention mechanism, dynamic pruning aims to reduce the computation cost for fast inference. [22] proposes a spatial dynamic convolution method, which adapts a residual block where a small gating branch learns which spatial positions are evaluated. All of the above methods focus on developing sophisticated attention modules for better performance of a single task. They optimize these attention modules jointly with the original model. Different from them, we aim to utilize the attention mechanism to enhance the adaption capacity of the pre-trained model on multi-domain learning.

3. Memory Analysis in Multi-Domain Training

In this section, we first explore the training memory usage under different multi-domain learning methods. Then, we will conduct a quantitative analysis of memory usage for each layer of DNN model. Moreover, such analysis will guide us to investigate a possible solution to achieve on-device memory-efficient learning method.

Fine-tuning and adaptor-based methods Both Fine-tuning and adaptor-based training schemes are popular in this research area, which requires fine-tuning all or part of parameters in the pre-trained model. Fine-tuning based training method on the target dataset domain is intuitive to understand. But, to explain the adaptor-based method, we illustrate the architecture of two popular adaptor-based methods. Such method needs to fine-tune the additional convolution layer and the original batchnorm (BN) layers. To understand the training memory consumption, let's assume a linear layer whose forward process be modeled as: $a_{i+1} = a_i\mathbf{W} + b$, then its back-propagation process is

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial a_i} &= \frac{\partial \mathcal{L}}{\partial a_{i+1}}\frac{\partial a_{i+1}}{\partial a_i} = \frac{\partial \mathcal{L}}{\partial a_{i+1}}\mathbf{W}^T, \\ \frac{\partial \mathcal{L}}{\partial \mathbf{W}} &= a_i\frac{\partial \mathcal{L}}{\partial a_{i+1}}, \frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a_{i+1}} \end{aligned} \tag{1}$$

According to Eq. 1, to conduct conventional back propagation based training for entire model, model weights-$\mathbf{W}$, gradients and activation-a_i all need to be stored for computing, leading to large memory usage. However, it is interesting to see that, if only updating bias, which has an additive relationship with activation-a_i, no activation storage is needed since previous activation a_i is not involved in the backward computation. The same phenomena can also be found in both Conv and BN layers.

Mask-based learning method For the mask-based learning method, assuming a linear layer whose forward process is given as: $a_{i+1} = a_i(\mathbf{W} \cdot \mathbf{M}) + b$, where $\mathbf{M}$ is the mask to be learned with the same size as $\mathbf{W}$. The weights-$\mathbf{W}$ is fixed, while only training the mask-$\mathbf{M}$. Then the backward process can be shown as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{M}} = a_i \frac{\partial \mathcal{L}}{\partial a_{i+1}} \cdot \mathbf{W}, \frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a_{i+1}} \tag{2}$$

Eq.2 shows that learning mask needs to store not only activation-a_i, but also the mask-$\mathbf{W}$ and weights-$\mathbf{W}$ during training. In terms of computation, comparing Eq.2 with Eq.1, such method also needs additional multiplication computation in both forward and backward pass. These observations explain why Piggyback has the largest training time in edge GPU as shown in Fig.1. Other mask-based methods [14, 26] even need more computation cost than Piggyback, since they involve additional reparameterization techniques. In addition, similar to fine-tuning and adaptor-based methods, training bias does not involve activation storage.

Table 1. Summary of the parameters and activation memory consumption of different layers. We denote the weights $\mathbf{W}^{(l)} \in \mathbb{R}^{c_{\text{in}} \times c_{\text{out}} \times kh \times kw}$, where $c_{\text{in}}, c_{\text{out}}, kh, kw$ refers the weight dimension of l-th layer, including #output channel, #input channel, kernel height and width, respectively. We also denote the input activation $\mathbf{A}^{(l)} \in \mathbb{R}^{n \times c_{\text{in}} \times h \times w}$, where n, h, k refers the batch size, activation height and width, respectively.

Layer Type	Trainable Param. (p)	Activation (a)
Conv	$c_{\text{in}} \times c_{\text{out}} \times kh \times kw$	$n \times c_{\text{in}} \times h \times w$
FC	$c_{\text{in}} \times c_{\text{out}} + c_{\text{out}}$	$n \times c_{\text{in}} \times h \times w$
BN	$2 \times c_{\text{out}}$	$n \times c_{\text{in}} \times h \times w$
ReLU	0	$n \times c_{\text{in}} \times h \times w$
Sigmoid	0	$n \times c_{\text{in}} \times h \times$ w

Training memory usage analysis. Here, we first define training memory usage that will be used in the rest of this paper. As displayed in Table 1, memory usage is proportional to the number of parameters during training, which can be treated as two main groups: i) # of trainable parameters - p (i.e. weights, bias) and gradient of each parameter; ii) activation memory consisting of the feature maps stored to update the parameters of previous layers using the chain rule. Note, trainable parameter memory has the same size as gradient memory. We only list # of trainable parameters - p in Table 1.

For most convolution layers, kernel height/width is much smaller than activation channel width/height (i.e., kh $\ll$ h; kw $\ll$ w). Thus, for a moderate batch size (e.g., $n = 64/128/256$), activation memory size is much larger than that of trainable parameters (i.e., $a \gg p$). More interestingly, even though BN and sigmoid function have a negligible amount of trainable parameters (p), both functions produce an activation output (a) of the same size as a CONV/FC layer.

From the above analysis, it can be easily seen that **DNN training memory usage is dominated by the activation feature map storage rather than the model parameter itself**. It is important to optimize the activation feature map memory usage if targeting memory-efficient learning. As for existing multi-domain learning methods, both mask-based and fine-tuning methods require heavy memory consumption during the backward propagation, requiring all weights, gradients, activation storage. Moreover, extra mask memory is required for mask-based method. It is also interesting to observe that, if it is possible to only update bias in multi-domain learning, the dominating memory usage component - activation is not required anymore. It is because bias has an additive-only relationship with input activation, enabling backward propagation independently. Based on above analysis, we summarize the underlying reason as the observation-2 below, which motivates and justifies our proposed DA^3 method in the next section.

Observation 2. *The complete activation map needs to be stored for backward propagation during training if it has the* ***multiplicative relationship*** *with learned parameter (i.e., weight, mask), while the* ***additive relationship*** *(e.g., bias) is activation free.*

4. Proposed Method

Motivated by the above memory usage analysis, we propose a new training method, named *Dynamic Additive Attention Adaption* (DA^3) as illustrated in Fig.2. DA^3 introduces a novel *additive attention adaptor* module in each block for a given DNN model, that follows the additive relationship with the weight of the main branch (i.e., pre-trained model) as mentioned in **observation 2**. To learn each new domain, DA^3 only updates the additive attention adaptor and the bias of the pre-trained model, while freezing the corresponding weight to preserve the knowledge of the previous domains. As the detailed structure of the additive attention adaptor illustrated in Fig.2, it aims to refine the ac-

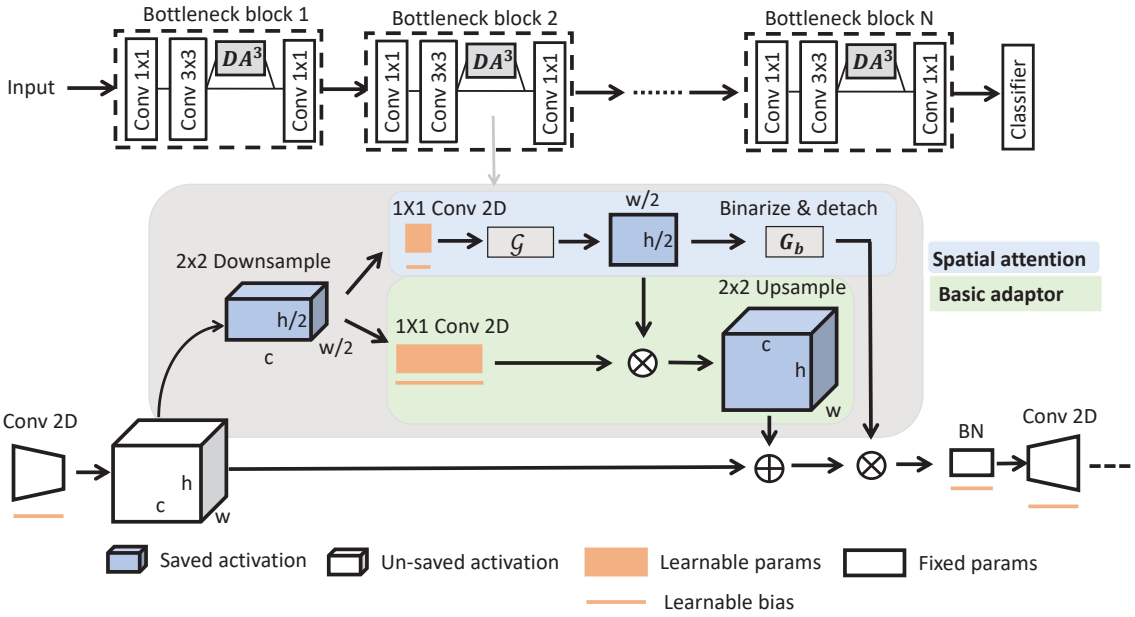

Figure 2. Overview of the proposed DA^3 , consisting of spatial attention (top branch) and basic adaptor (bottom branch).

tivation of the pre-trained model, which is computed as:

$$\mathbf{A}_i^* = (\mathbf{A}+\mathbf{H}(\mathbf{A})) \tag{3}$$

Where $\mathbf{H}$ denotes the output activation of the additive attention adaptor module. To design an efficient yet powerful module, we first compute the spatial attention $\mathbf{H}_s$ and the basic adaptor $\mathbf{H}_a$ at two parallel branches, then combining them as:

$$\mathbf{H}(\mathbf{A}) = \mathcal{G}(\mathbf{H}_s(\mathbf{A})) \otimes \mathbf{H}_a(\mathbf{A}) \tag{4}$$

Where $\otimes$ denotes the element-wise multiplication and $\mathcal{G}(\cdot)$ is a Gumbel-Softmax [10] function to obtain the spatial-wise soft attention of the basic adaptor activation $\mathbf{H}_a$. Benefiting from the spatial attention design, the soft attention output can indicate the importance of the main branch activation. Inspired by this, instead of fully utilizing the pre-trained model as prior works [2,17,18], we further select the important activation for current domain by turning the soft attention $\mathcal{G}(\mathbf{H}_s(\mathbf{A}))$ into binary hard gating $\mathbf{G}_b \in \{0,1\}$. Then, the Eq.3 can be further modified as:

$$\begin{aligned} \mathbf{G}_b &= Bin(\mathcal{G}(\mathbf{H}_s(\mathbf{A}))) \\ \mathbf{A}_i^* &= (\mathbf{A}+\mathbf{H}(\mathbf{A})) \otimes \mathbf{G}_b^{detach} \end{aligned} \tag{5}$$

Importantly, as the gating $\mathbf{G}_b$ has the multiplicative relationship with the activation of the pre-trained model $\mathbf{A}$, we detach the gating $\mathbf{G}_b$ from backward computation graph. By doing so, the changing of the $\mathbf{G}_b$ relies on the updating of the soft attention $\mathcal{G}(\mathbf{H}_s(\mathbf{A}))$ that has additive relationship with the pre-trained model. Thus, the detached $\mathbf{G}_b^{detach}$ is only used for forward pass that has no gradient to do backward propagation during training, having no additional activation memory storage from the main branch pre-trained model.

Basic adapter branch. As shown in Eq.1, the activation size grows quadratically with the resolution (i.e., height and width). Thus, to reduce the activation size, in the basic adapter branch, a 2×2 average pooling is used to downsample the input feature map, followed by a 1x1 convolution layer.

Spatial attention branch. To sample the activation in spatial-wise (i.e.,$n \times 1 \times h/2 \times w/2$) after downsampling, we adopt a 1x1 convolution layer with the output channel as 1. Then, following the Gumbel-softmax function $\mathcal{G}(\cdot)$, we obtain the soft attention. Such soft attention plays two roles: 1) it will be multiplied with the basic adaptor output to strengthen the domain-refined activation; 2) it turns to binary hard gating $\mathbf{G}_b \in \{0,1\}$ by applying a binarization trick and then multiplying with the output of main branch activation. By doing so, it could dynamically select the input-relevant spatial position for current domain. To avoid the activation storage of the main branch during training, the binary gating is detached from the computation graph that has no gradient to do backward propagation. The detailed Gumbel-softmax and binarization trick are relegated to the appendix-A.

Following the spatial attention branch and basic adaptor branch, the up-sampled and domain-refined activation will be added to the main branch (pre-trained backbone model) output activation. Note that, different from the conventional

attention scheme, where the output directly multiplies the main branch output activation, we design our additive attention adaptor in a way to *add* it to the main branch. The main benefit of doing so is the proposed additive attention adaptor module can be processed during backward independently, without creating a new backward pass as in the traditional multiplication based mechanism. Therefore, the increased memory usage for the proposed additive attention adaptor is very limited, which will be discussed in the later experimental section.

Dynamic Additive attention adaptor integration. Fig.2 illustrates an example to integrate the proposed additive attention adaptor in bottleneck block on ResNets [8]. For the basic block which has two connected convolution layers, we plug in the additive attention adaptor after the last convolution layer. For the bottleneck block, the last convolution layer will enlarge the output channels (i.e. $4\times$), which increases the output activation linearly. To avoid involving large activation increases, we add the additive attention adaptor after the second convolution layer.

5. Experiments

5.1. Experimental Setup

Datasets and Evaluation Metrics. To evaluate the efficacy of the proposed DA^3 method, we use standard and popular multi-domain learning dataset similar as many prior works [7, 13]. This setting includes five datasets (e.g., WikiArt [20], Sketch [5], Standford Cars [12], CUBS [23] and Flowers [16]). For each of the dataset, we report the test accuracy (%) on the publicly available test set.

Additionally, we also evaluate our proposed method on the Visual Decathlon Challenge [18]. The challenge is designed to evaluate the performance of learning algorithms on images from ten visual domains. The score (S) is evaluated as: $S = \sum_{i=1}^{10} \alpha_i \{0, E_{imax} - E_i\}^2$; where E_i is the best error on domain D_i, E_{imax} is the error of a reasonable baseline method, and the co-efficient α_i is the $1000(E_i^{max})^{-2}$.

Finally, to evaluate the training efficiency of DA^3 , we run our algorithm in the NVIDIA Jetson Nano GPU, which has 4GB DRAM with 20W power supply. We evaluate the training time on this edge GPU (i.e., constrained memory) to demonstrate the memory-efficient training through DA^3 .

Training Configuration. To demonstrate the efficiency of on-device training, we use Nvidia Jetson Nano GPU with 4-GB memory as our training platform. We evaluate the model performance using PyTorch as the simulation platform [2].Note that, the reported activation memory usage is calculated by our definition in Table.1, since PyTorch does not support explicit fine-grained memory management. In the training, for ResNet-50, we use Adam as the optimizer with cosine learning rate decay, an initial rate of 1e-3, and the number of iteration was set to 30. For ResNet-26 training on the challenge dataset, we use an SGD optimizer with an initial learning rate of 0.1. We schedule the learning rate decay at 40,80 and 100 epoch with a rate of 0.1. Again, as shown in Fig.1, we use right configuration of DA^3 for ResNet-50 and left configuration to train ResNet-26 model.

[2]https://forums.developer.nvidia.com/t/pytorch-for-jetson-version-1-7-0-now-available/72048

Baseline Methods. In this work, we primarily compare our method with three different baseline methods:

- **Fine-tuning-based method:** There are mainly two general fine-tuning strategies. The first baseline fine-tunes all the parameters of the pre-trained model on each new dataset [27]. Alternatively, the second one only fine-tunes the batchnorm and last classifier layers [15].
- **Adaptor-based method:** This baseline learns a residual adaptor for each convolution layer, while freezing the pre-trained weights except batchnorm layer. We compare with three different residual adaptor designs: series adaptor [17], parallel adaptor [18] and TinyTL [2]. Note that, TinyTL is reproduced by applying the lite residual adaptor without network architecture search.
- **Mask-based method:** We choose piggyback [13], a popular binary mask learning scheme that keeps the under-lying pre-trained weights fixed. It only trains the binary mask to learn a large number of filters on top of a fixed set of pre-trained weights.

5.2. Results and Analysis

We first compare our algorithm's efficacy with baseline methods by evaluating the performance on the test dataset listed in Table 2. Next, we evaluate the efficiency in reducing the training cost after deploying the models in NVIDIA Jetson Nano GPU in Table 3. The detailed experiments configuration is relegated in appendix-A.

Accuracy Comparison. In this evaluation section, each baseline method and DA^3 train a ResNet-50 model with pre-trained weights on ImageNet dataset. As shown in Table 2, our proposed method DA^3 achieves the *best* test accuracy in CUBS, Stanford Cars and Flowers dataset. As for WikiArt, standard fine-tuning outperforms all the other techniques. Since WikiArt has a smallest number of samples between training and testing dataset in comparison to the other datasets, it helps to mitigate the over-fitting issue of fine-tuning the entire model. Finally, most notably,

Table 2. Summary of the results (i.e., accuracy %) of the proposed method and comparison with the baseline techniques on five datasets (e.g., CUBS, Stanford Cars, Flowers, WikiArt and Sketches).

Model	CUBS	Stanford Cars	Flowers	WikiArt	Sketches	Average
Standard Fine-tuning [7]	81.86	89.74	93.67	75.60	79.58	84.09
BN Fine-tuning [15]	80.12	87.54	91.32	70.31	78.45	81.54
Parallel Res. adapt [18]	82.54	91.21	96.03	73.68	82.22	85.14
Series Res. adapt [17]	81.45	89.65	95.77	72.12	80.48	83.89
Piggyback [13]	81.59	89.62	94.77	71.33	79.91	83.45
TinyTL* [2]	82.34	90.23	94.63	71.39	80.44	83.80
Ours (DA^3)	*83.33*	*91.50*	*96.65*	*72.79*	*82.20*	***85.29***

Table 3. Summary of the results (i.e., activation memory(MB), training time (s)% and inference computation (GFlops)) of the proposed method and comparison with the baseline techniques on four datasets (e.g., CUBS, Stanford Cars, Flowers and Sketches) on NVIDIA Jetson Nano GPU. Note that, the reported training time is the time of training one epoch with batchsize 4 on average.

	Dataset			Flowers	CUBS	Cars	Sketches
Methods	Model param (MB)	Active. mem (MB)	Inference GFlops	< ——— Training Time (s) ——— >			
Standard Fine-tuning	91.27	343.76	4.15	686	1977	2676	5843
BN Fine-tuning	91.27	174.17	4.15	173	507	683	1300
Parallel Res. adapt	177.8	308.8	4.68	558	1741	2310	4669
Series Res. adapt	178	309.55	4.68	570	1832	2490	4783
Piggyback	94.12	343.76	3.44	1061	3015	4327	9783
TinyTL	117.3	50.9	4.42	493	1570	2103	4372
Ours (DA^3)	*98.64*	***10.49***	***3.17***	*308*	*834*	*1073*	*2274*

DA^3 achieves comparable accuracy in comparison to the best baseline technique Parallel Res. Adapter [18], achieving fractionally improved test accuracy in CUBS, Stanford Cars, and Flowers dataset, but much smaller training time shown in later Table 3. In summary, the proposed DA^3 method achieves improved or comparable test accuracy in comparison to all the baseline techniques on five evaluation datasets.

Training and Inference Cost Comparison In Table 3, we summarize our key contribution in reducing the training and inference cost of multi-domain learning. Note, those are evaluated in a real memory-limited NVIDIA Jetson Nano GPU. As shown in the Table 3, the proposed DA^3 method increases the model size by only a small fraction in comparison to Standard/BN Fine-tuning [15] and Piggyback [13] method. But DA^3 reduces the activation memory size by *5-37* $\times$ in comparison to the baseline techniques. As stated before, Parallel Res. Adapter [18] has shown superior performance (i.e., higher test accuracy) across four dataset. But we demonstrate that DA^3 reduces the activation memory size by *34* $\times$ when compared with Parallel Res. Adapter's activation memory size while maintaining a similar test accuracy. Furthermore, compared with TinyTL [2], DA^3 still can *5* $\times$ memory reduction with better accuracy.

Apart from the reduction in memory cost, our proposed DA^3 speeds-up the actual training time for on-device learning as well. As shown in Table 3, the training time reduces nearly by *2* $\times$ in comparison to all the baseline techniques except for BN Fine-tuning [15]. The faster training of BN Fine-tuning can be attributed to the presence of significantly less learnable parameters (less than 1MB), resulting in the worst accuracy performance in Table 2. Nevertheless, our method still outperforms BN based Fine-tuning in terms of both reduced activation memory size (i.e., 19 $\times$) and improved test accuracy across four datasets (e.g., CUBS, Stanford Cars, Flowers and Sketches). To summarize, in Fig. 3, we show that DA^3 reduces training cost (i.e., time) in comparison to all the baseline methods (except BN fine-tuning); while maintaining on-par or improved test accuracy compared with the best (i.e., highest test accuracy %) baseline method (i.e., Parallel Residual [18]).

Moreover, we also summarize the averaged inference computation cost on the five dataset in Fig. 2. Benefit from the spatial adaptor design, DA^3 further achieves $1.30\times$, $1.47\times$ and $1.08\times$ inference computation cost reduction compared with fine-tuning-based, adaptor-based and mask-based method respectively.

5.2.1 Visual Decathlon Challenge

In Table 4, we show the effectiveness of our learning scheme on all the ten datasets of Visual Decathlon Chal-

Table 4. Summary of the results (i.e., test accuracy %) on the Visual Decathlon Challenge dataset. Here # par denotes the number of model parameters with respect to a ResNet-26 baseline model in [17].

Methods	Model mem (MB)	Activ. mem (MB).	ImNet	Airc.	C100	DPed	DTD	GTSR	Flwr	OGlt	SVHN	UCF	Score
Scratch	22.29	1315	59.87	57.1	75.73	91.2	37.77	96.55	56.30	88.74	96.63	43.27	1625
Fine-tuning	22.29	1315	60.32	61.87	82.12	92.82	55.53	99.42	81.41	89.12	96.55	51.2	3096
Series Res. adapt	24.94	1963	60.32	61.87	81.22	93.88	57.13	99.27	81.67	89.62	96.57	50.12	3159
Parallel Res. adapt	23.62	1405	60.32	64.21	81.92	94.73	58.83	99.38	84.68	89.21	96.54	50.94	3412
Piggyback	22.29	1315	57.69	65.29	79.87	96.99	57.45	97.27	79.09	87.63	97.24	47.48	2838
Ours (DA^3)	*25.36*	*202*	*62.74*	*64.58*	*82.82*	*96.85*	*59.43*	*99.44*	*88.62*	*89.73*	*97.47*	*51.29*	*3498*

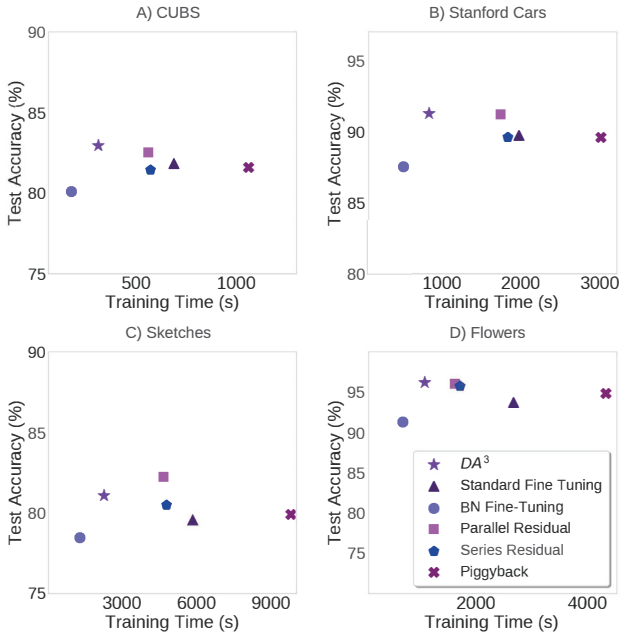

Figure 3. Trade-off between the Test Accuracy (%) and Training Time (s) for four datasets A) CUBS, B) Stanford Cars , C) Sketches and D) Flowers.

lenge on ResNet-26. Note that, for this experiment, we plug an additive attention adaptor to each convolution layer. As reported in Table 4, DA^3 achieves $\sim$ 3 % accuracy gain on ImageNet and $\sim$ 4 % accuracy gain on Flower dataset in comparison to the baseline methods. Moreover, it achieves the best S score (3498) out of all the previous techniques demonstrating the effectiveness of our method in adapting to multi-domain tasks. Finally, it can also reduce the activation memory storage overhead during training by *7-11* $\times$ in comparison to other methods; thus emerging as an ideal candidate for on-device learning purposes.

5.2.2 Ablation Study of Additive Attention Adaptor

We study the effectiveness of each component in the proposed additive attention adaptor on ImageNet-to-Sketch dataset setting. As shown in Table.5, we consider four different combinations to perform this ablation study: 1) Only updating bias (Only bias); 2) Only updating the basic adaptor module (Only Basic adap.); 3) Jointly updating the bias and spatial adaptor (Bias + Basic adap); 4) Jointly updating the proposed additive attention adaptor with bias. First, only bias has the worst accuracy, demonstrating the limited learning capacity using only a few bias parameters, supporting our initial hypothesis of adding the attention adapter to improve learning capacity. As a result, after adding the spatial adaptor, we observe a clear accuracy gain. Furthermore, jointly updating bias and spatial adaptor could improve accuracy even further. In the end, we introduce our proposed DA^3 utilizing the channel attention module which connects the spatial adaptor in parallel to achieve the best performance. As DA^3 succeeds in maintaining a reasonable test accuracy while drastically reducing the training overhead (as shown in Table 3 & Fig. 3).

Table 5. The ablation study on the proposed method

Method	CUBS	Cars	Flowers	WikiArt	Sketch
Only bias	74.53	83.85	87.30	68.73	71.93
Only Basic adap.	82.01	89.03	95.03	71.33	80.42
Bias + Basic adap.	82.15	89.73	95.56	71.88	80.70
Proposed DA^3	*83.33*	*91.50*	*96.65*	*72.79*	*81.20*

6. Conclusion

We propose DA^3 for memory-efficient on-device multi-domain learning, which is designed to eliminate the storage of intermediate activation feature maps. We design a method equipped with a novel additive attention adaptor to adapt original model to a new domain accurately. Such method not only reduces the training cost (e.g., time and memory) significantly, but also enables fast inference. Extensive experiments on domain adaption datasets consistently show the effectiveness and memory-efficiency of DA^3 , paving a new way for memory-efficient on-device multi-domain learning.

Acknowledgements This work is supported in part by the National Science Foundation under Grant No.1931871 and No. 2144751

References

[1] Rodrigo Berriel, Stephane Lathuillere, Moin Nabi, Tassilo Klein, Thiago Oliveira-Santos, Nicu Sebe, and Elisa Ricci. Budget-aware adapters for multi-domain learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 382–391, 2019. 3

[2] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tiny transfer learning: Towards memory-efficient on-device learning. *arXiv preprint arXiv:2007.11622*, 2020. 2, 3, 5, 6, 7

[3] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. 3

[4] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4109–4118, 2018. 2, 3

[5] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Transactions on graphics (TOG)*, 31(4):1–10, 2012. 6

[6] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in neural information processing systems*, pages 2214–2224, 2017. 3

[7] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4805–4814, 2019. 3, 6, 7

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6

[9] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 3

[10] Eric Jang et al. Categorical reparameterization with gumbel-softmax, 2017. 5

[11] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2661–2671, 2019. 2

[12] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 554–561, 2013. 6

[13] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018. 2, 3, 6, 7

[14] Massimiliano Mancini, Elisa Ricci, Barbara Caputo, and Samuel Rota Bulo. Adding new tasks to a single network with weight transformations using binary masks. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018. 3, 4

[15] Pramod Kaushik Mudrakarta, Mark Sandler, Andrey Zhmoginov, and Andrew Howard. K for the price of 1: Parameter-efficient multi-task and transfer learning. *arXiv preprint arXiv:1810.10703*, 2018. 2, 6, 7

[16] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008. 1, 6

[17] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516, 2017. 2, 3, 5, 6, 7, 8

[18] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8119–8127, 2018. 2, 3, 5, 6, 7

[19] Amir Rosenfeld and John K Tsotsos. Incremental learning through deep adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 2018. 3

[20] Babak Saleh and Ahmed Elgammal. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *arXiv preprint arXiv:1505.00855*, 2015. 6

[21] Nimit Sharad Sohoni, Christopher Richard Aberger, Megan Leszczynski, Jian Zhang, and Christopher Ré. Low-memory neural network training: A technical report. *arXiv preprint arXiv:1904.10631*, 2019. 3

[22] Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2320–2329, 2020. 3

[23] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011. 6

[24] Xudong Wang, Zhaowei Cai, Dashan Gao, and Nuno Vasconcelos. Towards universal object detection by domain attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7289–7298, 2019. 3

[25] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 3

[26] Li Yang, Zhezhi He, Junshan Zhang, and Deliang Fan. Ksm: Fast multiple task adaption via kernel-wise soft mask learning. *arXiv preprint arXiv:2009.05668*, 2020. 3, 4

[27] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014. 6