Rep-Net: Efficient On-Device Learning via Feature Reprogramming

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

Transfer learning, where the goal is to transfer the well-trained deep learning models from a primary source task to a new task, is a crucial learning scheme for on-device machine learning, due to the fact that IoT/edge devices collect and then process massive data in our daily life. However, due to the tiny memory constraint in IoT/edge devices, such on-device learning requires ultra-small training memory footprint, bringing new challenges for memory-efficient *learning. Many existing works solve this problem by reducing the number of trainable parameters. However, this doesn't directly translate to memory saving since the major bottleneck is the activations, not parameters. To develop memory-efficient on-device transfer learning, in this work, we are the first to approach the concept of transfer learning from a new perspective of* ***intermediate feature reprogramming*** *of a pre-trained model (i.e., backbone). To perform this lightweight and memory-efficient reprogramming, we propose to train a tiny* ***Reprogramming Network (Rep-Net)*** *directly from the new task input data, while freezing the backbone model. The proposed Rep-Net model interchanges the features with the backbone model using an* ***activation connector*** *at regular intervals to mutually benefit both the backbone model and Rep-Net model features. Through extensive experiments, we validate each design specs of the proposed Rep-Net model in achieving highly memory-efficient on-device reprogramming. Our experiments establish the superior performance (i.e., low training memory and high accuracy) of Rep-Net compared to SOTA on-device transfer learning schemes across multiple benchmarks. Code is available at https://github.com/ASU-ESIC-FAN-Lab/RepNet.*

1. Introduction

Nowadays, the utilization of IoT devices is significantly increased (e.g., 250 billion microcontrollers in the world today[1]), which collect and then process massive new data crossing various domains/tasks in our daily life. This fact arouses researchers great interest in on-device AI that expect not only inference but also the training or transferring of pre-trained models to new data on the device. Such learning scheme is strongly associated with the concept of transfer learning [5], where the goal is to transfer the well-trained deep learning models from a primary source task to a new task, leading to a recently rising research direction about *on-device transfer learning*. Compared with conventional scheme that learns the deep learning models on could and then performs inference on device, on-device transfer learning eliminates the communication issue between cloud and edge devices, as well as data-privacy concern. Although the benefits of learning on-device are clear, the memory-hungry training process is a extremely challenge for memory-constrained IoT/edge devices. Recently, the studies on memory-efficient learning [3, 6, 37] reveal that the bottleneck of training memory consumption is the storage of intermediate activations, not parameters. Thus, re-

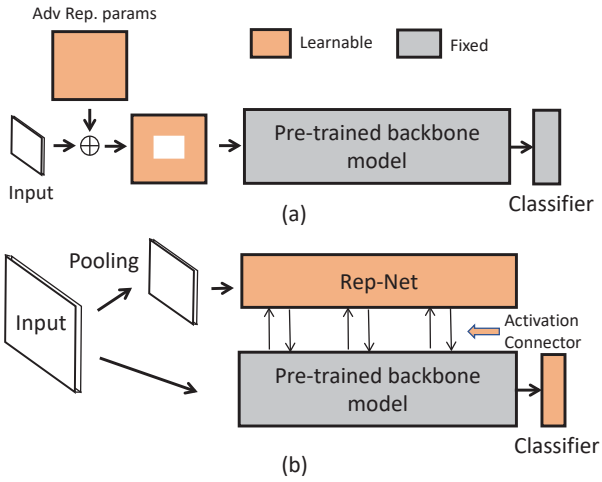

Figure 1. *The workflow of adversarial reprogramming (a) and the Rep-Net (b). Adversarial reprogramming reprograms the input by introducing an additive learnable parameter. Differently, Rep-Net takes the input data directly and learns to reprogram the intermediate activation features.*

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

ducing the training memory footprint, especially the activation storage of the pre-trained model, is the crucial issue to enable on-device transfer learning. However, the existing transfer learning methods either suffer from huge training memory consumption, or having limited transfer capacity.

Usually, it is a common practice to fine-tune the models that pre-trained on large scale dataset, like ImageNet [8], to perform transfer learning [5, 7, 12, 15, 20, 29, 38]. Compared with training from scratch, fine-tuning a pre-trained convolutional neural network on a target dataset can significantly improve performance. Generally, there are mainly two ways of fine-tuning: 1) Treating the pre-trained model as a fixed feature extractor, then only fine-tuning the last classification layer [5, 38]. This method is memory efficient for training, as it eliminates the intermediate activation storage of the pre-trained model. However, the low transfer capacity of this has method has been widely demonstrated on previous works [7, 20, 29]. 2) Fine-tuning the full or partial pre-trained model. This method can achieve better accuracy [7, 15, 20, 20, 29]. However, it updates the pre-trained model, causing a vast activation memory footprint. Thus it is not friendly for on-device learning. In addition, mask-based methods [28, 36] perform transfer learning by learning a binary mask w.r.t all the wights of the pre-trained model. But it only reduces the trainable parameter size, not the activation memory, still resulting in high memory footprint.

Alternatively, *adversarial input reprogramming* [10] performs transfer learning by reprogramming input. As shown in Fig. 1 (a), input reprogramming is basically an additive operation to change the existing input features using a trainable element-wise bias for each input pixels (refer to Eq. (3)). Unlike fine-tuning methods that need to update the model parameters, adversarial reprogramming keeps the model architecture unchanged. Instead, it introduces an additive learnable adversarial parameter for input data and designs an output label mapping on the target-domain data samples to perform transferring as shown in Fig. 1(a). Such a method is memory efficient during training as it only learns additive parameters to reprogram the input that does not require any updates to the pre-trained model, as well eliminating the memory-dominating activation storage. However, it comes with several shortcomings: 1) the transfer capacity of only adding a learnable parameter to input is limited; 2) such design only works when the input size of target data is much smaller than the source data (e.g., ImageNet (224×224) to MNIST (28×28)), essentially constraining the generality of the method on more complex dataset; 3) the additional parameter size and training memory cost are still high. For example, considering the ImageNet pre-trained model, it needs $224 \times 224 \times 3 \sim$ 150K additional trainable parameters overhead. Despite these shortcomings, the reprogramming concept is simple and has the potential to solve the existing on-device learning challenges.

To tackle the challenges and shortcomings of prior works, we propose *Reprogramming Network (Rep-Net)*, which, for the first time, treats the on-device transfer learning problem from a new perspective of *intermediate feature reprogramming*. As shown in Fig. 1(b), Rep-Net serves as a lightweight side-network that is executed with the pre-trained backbone model in parallel. The principle working mechanism of this design is to reprogram the fixed backbone model from the input data via proposed *activation connector* that enables feature exchange between the backbone and Rep-Net at regular intervals. Such feature exchange uses an additive operation that not only helps both the backbone model and Rep-Net model to update and improve their features, but also inherits the property of memory-efficiency from adversarial input reprogramming. To perform on-device transfer learning, we only train Rep-Net model and a task-specific last classification layer for the new task, while freezing the backbone model. In summary, our technical contributions include:

- In this work, for the first time, we approach the on-device transfer learning from a new perspective of *intermediate feature reprogramming* problem for a given pre-trained backbone model. To develop this lightweight reprogramming scheme, we propose a novel memory-efficient *Reprogramming Network (Rep-Net)* that is trained directly from the new task input data to reprogram the intermediate features of a backbone model. It is a small convolution network with little memory overhead. In fact, our proposed Rep-Net improves memory efficiency in comparison to adversarial input reprogramming, but providing much improved transfer capacity.

- We design dedicated *activation connectors* to enable feature exchange between the backbone and the Rep-Net models at regular intervals. Both models benefit mutually from the feature exchange operation. We perform an extensive analysis of the design specs of the connector, investigate and answer the following design space challenges: *i.e., How to exchange the features considering the limited memory budget during training? Why exchanging both (i.e., backbone and Rep-Net model) features? How many activation connectors are necessary? How many layers are necessary in Rep-Net?*.

- We conduct extensive experiments to corroborate the superiority of Rep-Net over current state-of-the-art (SOTA) transfer learning approaches across multiple benchmarks.

2. Related Works

2.1. Memory efficient learning

In recent layers, several works have been proposed to reduce the training memory consumption that mainly can be categorized as activation re-computation [6, 14] and compression [11, 23]. For the activation re-computation, the idea is to eliminate partial or full activation storage by re-computing when it is needed during backward pass. As a consequence, it involves additional computation cost. For the activation compression method, [23] proposes to reduce the activation by pruning. In addition, [11] compress the activation by adapting image compression algorithms and then de-compress when use it during training. Note that, our method is orthogonal to the activation compression method.

More recently, [2, 37] further analyze the memory utilization and highlight that the training memory is dominated by activations, not parameter size. 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

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

Where a is activation feature, $\mathbf{W}$ and b are learnable weights and bias respectively. Eq. (1) reveals that the memory-hungry activation needs to be stored for backward propagation during training if it has the multiplicative relationship with learned parameter (i.e.,weight), while the *additive* relationship (e.g., bias) is activation free [37]. In this work, the Rep-Net also has a completely additive relationship with the fixed pre-trained model, resulting in training memory efficiency.

2.2. Transfer Learning

Transfer learning via fine-tuning Currently, with the success of the deep neural network, fine-tuning a network that is pre-trained on large scale datasets such as ImageNet [8] to a new dataset is the standard method for knowledge transfer [5,7,12,13,15,20,29,38]. Generally, there are mainly two ways of fine-tuning: 1) Treating the pre-trained model as a fixed feature extractor, then only fine-tuning the last classification layer [5,38]. This method is memory efficient for training, as it eliminates the intermediate activation storage of the pre-trained model. However, such method suffer from low transfer capacity of this has method has been widely demonstrated on previous works [20]; 2) fine-tuning the full or partial pre-trained model This method can achieve better accuracy [7, 15, 20, 20, 29]. Specifically, [13] fine-tunes the whole model on domain-specific data to boost the accuracy on object detection. [7] proposes to only update batchnorm layer. Further, [15] automatically decides the optimal set of layers to fine-tune in a pretrained model on a new task. However, all of the methods updates the pre-trained model, causing a vast activation memory footprint. Thus it is not friendly for on device learning.

Transfer learning via input reprogramming The concept of reprogramming is first introduced by adversarial input reprogramming [10], where additive features to the input are trained to transfer a model's knowledge to a new domain of dataset. The idea is inspired by adversarial example attack [26] which is the first to demonstrate how small (less than 6 %) additive perturbation to the input can completely shift the performance of the target model into new behaviour. Let's consider an input batch to a neural network be $\boldsymbol{x} \in \mathbb{R}^{k \times k \times c}$ which is sampled from the dataset with an input dimension $k \times k \times c$. Now, we want to adopt a pre-trained network with a forward function h_θ to this new domain data $\boldsymbol{x}$ without changing the inference function h(.) or parameter θ. One possible solution would be to introduce a new set of learn-able additive parameters to the input $\boldsymbol{p}$:

$$\hat{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{p} \tag{2}$$

The objective of reprogramming is to minimize the loss function $\mathcal{L}_R$ for the correct label $\boldsymbol{y}$ by updating the input parameter $\boldsymbol{p}$:

$$\min_{\{\boldsymbol{p}\}} \mathbb{E}_{\mathbb{X}}\Big(\mathcal{L}_R(h_\theta(\hat{\boldsymbol{x}}), \boldsymbol{y})\Big) \tag{3}$$

The above process can be repeated for the whole dataset $(\mathbf{X}, \mathbf{Y})$ of the new domain by training a single bias $\boldsymbol{p}$. This concept of input reprogramming is simple which does not require any modification to the backbone model architecture or parameters. Based on the same idea, [19] proposes to improve the transfer capacity by resizing input reprogramming and re-train the last classification layer. In addition, [33] extends to black-box transfer learning which nothing but only the input-output model responses are observable. However, these input reprogramming methods come with several shortcomings. First, the learning capability of only adding input bias is minimal even after adding non-linearity (e.g., Hard Tanh) before addition [10]. Second, it will add additional $k \times k \times c$ learn-able parameters which is quite large considering large-scale dataset (e.g., on Imagenet models $224 \times 224 \times 3 \sim 150$k parameters). Third, the performance of input reprogramming is inferior and fails on simple tasks like MNIST [9]. To overcome these challenges, we propose to train a lightweight side network from new task input data and reprogram the intermediate activation features.

3. Proposed Method

3.1. Overview

In this section, we propose a lightweight *Reprogramming Network (Rep-Net)* which learns to reprogram the

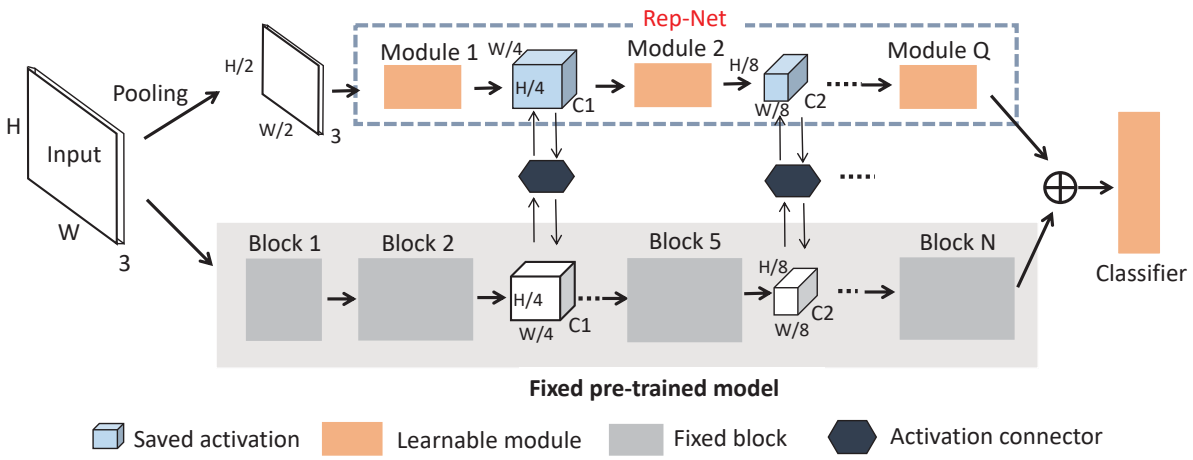

Figure 2. *The overview of Reprogramming Network (Rep-Net). The proposed Rep-Net model learns directly from the input. The input image is directly fed into both Rep-Net and backbone model parallelly. In addition, Rep-Net consists of a small number of layers positioned at specific locations where the backbone model observes a feature resolution reduction.*

backbone model activation features directly from the input data (Fig. 1). The principle working mechanism of this reprogramming network is to inter-exchange the features with the backbone model at regular intervals. The feature exchange operation helps both the backbone model and Rep-Net improve their respective features by updating them using an additive operation with the exchanged features. During reprogramming (i.e., learning new tasks), we only train Rep-Net and a task-specific last classification layer for the new task. Our proposed Rep-Net has the following characteristics, making it an ideal candidate for on-device learning. i) During training, since the backbone model is frozen, the features from the Rep-Net models helps adapt the backbone model features on the new task. ii) Similarly, since the backbone is already pre-trained on a primary domain, features transferred from this model to Rep-Net helps its learning process on the new task. iii) We optimize the specific positioning of the feature exchange operation using observations from prior pruning works [24, 35]. iv) Such an optimization ensures Rep-Net model requires only a few blocks/layers to learn the new task, which makes Rep-Net a lightweight network with almost negligible memory overhead. v) Finally, we ensure the feature exchange operations follow an additive rule to achieve memory-efficient on-device learning. As already discussed in Sec. 2.1, due to the additive nature of the learn-able parameters, the intermediate activations are not needed to be stored in memory during back-propagation. We present the Rep-Net design details in the following section and validate each design specs with a relevant ablation study.

3.2. Reprogramming Network (Rep-Net)

We design a novel *reprogramming network (Rep-Net)* which will be trained directly from the input data for learning new task. The overall framework of Rep-Net architecture is presented in Fig. 2. In this section, we introduce the important design specs of the proposed Rep-Net architecture and reprogramming procedure.

Architecture Overview. In our framework (Fig. 2), we feed the input data to both the backbone and Rep-Net models in parallel. The working principle of Rep-Net is to interchange its intermediate feature with the backbone model at regular intervals using an *activation connector*. In this way, both models (i.e., backbone & Rep-Net) can mutually benefit and improve their features. The objective is to reprogram the intermediate activations of the backbone model on a new domain of knowledge. As a result, the backbone model remains fixed (i.e., both weights and architecture) as highlighted in grey color in Fig. 2. We only train the Rep-Net & a shared last classification layer (highlighted in orange) on the new task to reprogram the backbone model. The proposed Rep-Net is a lightweight neural network with only a few convolution modules which has very minor memory overhead (i.e., activation storage, additional parameters) compared with backbone model. Following the lightweight convolution block design in [2, 16, 32], our convolution module consists of two stacked convolution layers.

Activation Connector. In our design, the Rep-Net model exchanges its intermediate activation features with the backbone model using an *activation connector*. Consider a deep neural network with N layers as the backbone model, and

our proposed Rep-Net model has Q layers ($Q \leq N$). In this case, Rep-Net will exchange its features with the backbone model Q times at activation connector. In our design, the number of activation connectors is equal to the number of layers in Rep-Net (i.e., Q) (*Reason: refer to design intuition* ***3***). If the backbone model sends features $\boldsymbol{x}_i$ to the activation connector at the i^{th} layer and Rep-Net model sends features $\boldsymbol{p}_i$ to the same connector where $(i = 1, 2, 3, ..., Q)$; then the forward path operation inside the activation connector performs a feature reprogramming as:

$$\hat{\boldsymbol{x}_i} = \hat{\boldsymbol{p}_i} = \boldsymbol{x}_i + \boldsymbol{p}_i \tag{4}$$

The activation connector then sends the reprogrammed features $\hat{\boldsymbol{x}_i}$ & $\hat{\boldsymbol{p}_i}$ to the backbone and Rep-Net model respectively. This operation enhances the features quality of both the backbone and Rep-Net models during training (*Evidence: refer to design intuition* ***2***).

Reprogramming Step. To reprogram the backbone model which has an inference function h_θ, parameterized by θ, we omit θ from our optimization step. Instead, we only train the Rep-Net model function g_β parameterized by β and a shared last classification layer parameters α. We represent the combined function of the backbone and Rep-Net model as $f_{\theta,\beta,\alpha}$. For a given input and target pair $(\boldsymbol{x}, \boldsymbol{y})$ the reprogramming optimization step can be summarized as follow:

$$\min_{\{\beta,\alpha\}} \mathbb{E}_{\mathbb{X}}\Big(\mathcal{L}(f_{\theta,\beta,\alpha}(\boldsymbol{x}), \boldsymbol{y})\Big) \tag{5}$$

where the $\mathcal{L}$ (.) is the cross-entropy loss function which is minimized by updating parameter set $\{\beta, \alpha\}$. Now that our design and training steps of Rep-Net is summarized. Next we introduce the design intuitions for optimizing Rep-Net model for an efficient on-device learning scheme while improving performance at the same time.

3.3. Design Intuition

Our proposed Rep-Net architecture has numerous design choices which could impact the eventual performance and efficiency of the reprogramming scheme. To validate our adopted design specs, we discuss three key questions:

1. *How to design the activation connector? (Refer to Design Intuition-1)*

2. *Is it necessary to reprogram both the features (i.e., $\hat{\boldsymbol{x}_i}$ & $\hat{\boldsymbol{p}_i}$)? What if we only reprogram either $\boldsymbol{x}$ or $\boldsymbol{p}$? (Refer to Design Intuition-2)*

3. *How many activation connector in Rep-Net is sufficient? or How many layers the Rep-Net model should have? (Refer to Design Intuition-3)*

Next, we present three design intuitions with relevant ablation study to validate our design choices by discussing those three questions.

Design Intuition 1. Typically, the feature exchange operation of the activation connector mainly has two choices: multiplication or addition. However, due to the property of the back-prorogation during training as mentioned in Sec. 2.1, the complete activation of the pre-trained model needs to be stored during training if it has the multiplicative relationship with the learned parameter (i.e., weight, mask), while the additive relationship (e.g., bias) is activation free. Thus, we adapt the addition as the reprogramming operation to exchange features using the activation connector. The additive relationship between the features provides significant gain in training memory efficiency.

Design Intuition 2. Following the *Design Intuition 1*, another important design parameter for feature exchange operation is highlighted in Fig. 3. Here, we present four alternative designs for the feature exchange operation. First, Fig. 3 (a) shows *no-connection* between Rep-Net and backbone model as they do not exchange any intermediate features. For this case, the Rep-Net model only interacts with the backbone model just before the last classification layer using an additive operation. For the *down connect* case in Fig. 3 (b), we only modify the backbone model features using Rep-Net features (i.e., $\boldsymbol{x} + \boldsymbol{p}$), but keeping the features from Rep-Net (i.e., p) unchanged. This case highlights the importance of proposed Rep-Net model features in reprogramming the backbone model. For the *up connect* case in Fig. 3 (c), we only modify the Rep-Net features, but keeping the intermediate backbone features unchanged. This case will evaluate the importance of backbone model features in training the Rep-Net model on this new task. Finally, we present the *dual connection* case in Fig. 3 (d). Our design intuition is that both models should mutually benefit by interchanging features between them. To validate this, we present the summary of our ablation study in Tab. 1. The study shows that across five dataset, our dual connection outperforms all the other cases. Another important observation in Tab. 1 is that *up connection outperforms down connection*. The reason is that Rep-Net is a tiny neural network that requires pre-trained features from the backbone model to improve its learning capability. Fig. 4 further validates that feature exchange operation mutually benefits both models to adapt to the new task as the model converges faster using a dual connection.

Design Intuition 3. We adopt the following strategy to determine the number of activation connectors and number of layers in Rep-Net:

Table 1. *The ablation study to validate the four design choices displayed in Fig. 3. 'Adv Rep + Last' is the adversarial reprogramming combining with re-training last classifier. The 6^{th} row shows the results for proposed Rep-Net. The last row (7^{th}) shows the result for using dual connector at all the convolution layer for the backbone model. We use the ImageNet pre-trained proxylessNAS-Mobile [4] as the backbone model.*

Method	Train. mem	Flowers	Cars	CUB	Pets	Aircraft
Adv Rep + Last	36MB	91.1	56.0	66.1	88.2	44.4
No connect (a)	34MB	93.3	51.1	65.5	88.0	36.9
Down connect (b)	34MB	91.8	73.4	67.8	89.6	56.3
Up connect (c)	34MB	95.2	76.8	71.4	90.7	59.3
Dual connect (d) (proposed Rep-Net)	*34MB (↑ 2-3MB)*	*96.1*	*85.8*	*77.1*	*91.8*	*77.4*
Dual connect - all layers	37MB	96.0	85.8	76.8	91.2	78.7

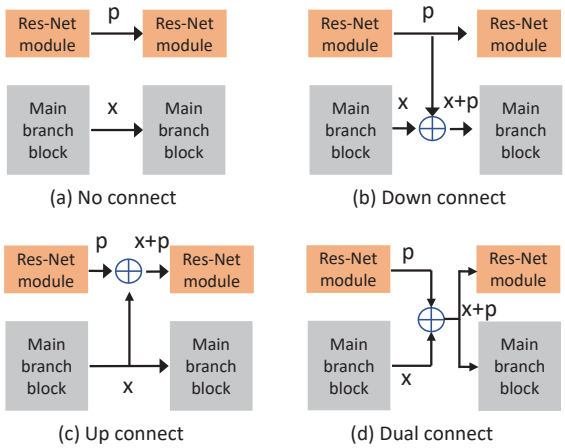

Figure 3. *Design choices for feature exchange operation in activation connector. We summarize the impact of using each choice in Tab. 1.*

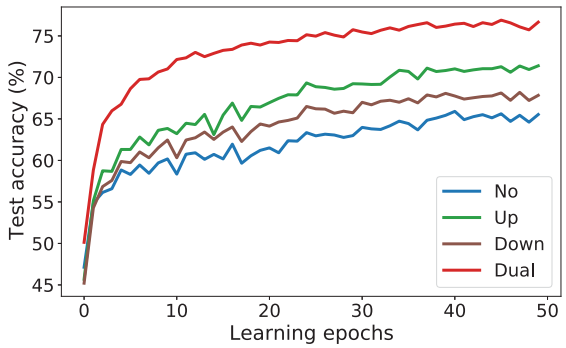

Figure 4. *The test accuracy vs learning epochs under four difference design choices highlighted in Fig. 3 on CUB dataset.*

***Strategy**: The number of activation connectors is determined by the number of down-sampling (e.g., pooling layer, convolutional blocks with stride 2) in the pre-trained model. For each down-sampling in activaiton feature map in the backbone model, we apply one Rep-Net module. Then, the Rep-Net module features could be connected with the output activation feature of the down-sampling layer/block of the backbone model via activation connector.*

Reason: Such design strategy is inspired by the recent observations from model pruning works [24, 35], where they conclude that down-sampling reduces the resolution in the activation feature. Hence, to compensate, more channels need to be added to carry the same amount of information. In our case, we compensate the activation features resolution degradation by exchanging and updating features with Rep-Net model. To validate the design strategies, we compare it with a complex version of Rep-Net model that has a connector modules for all convolutional blocks. As shown in Tab. 1, only having connector modules at the down-sampling layers (6^{th} row) can outperform a Rep-Net model with all layers connection (7^{th}/last row).

Second, since such down-sampling layers are much less than the total number of layers/blocks in modern model architectures, one Rep-Net layer for each downsampling layer results in improved inference memory and computing efficiency. For example, Rep-Net consists of only 6 modules on ResNet-50 architecture. Again, as highlighted in Tab. 1, by using only 6 modules in Rep-Net saves around *3MB* of memory with identical accuracy.

4. Experiments

4.1. Experimental Setup

Datasets and networks. Following the general setting in previous transfer learning methods [2, 7, 20, 29], we adapt ImageNet [8] as the dataset for all the pre-trained models, and then transfer the models to 8 downstream object classification tasks, including Cars [21], Flowers [30], Aircraft [27], CUB [34], Pets [31], Food [1], CIFAR10 [22], and CIFAR100 [22]. We adapt four different model architectures to demonstrate the effectiveness of the proposed Rep-Net, including ResNet50 [16], MobileNetv2 [32] and MobileNetv3 [17] and proxylessNAS-Mobile [4]. Note

Table 2. *Comparison with input reprogramming works. 'Adv. Rep' is the original adversarial reprogramming work; 'Adv Rep + Last' is the improved adversarial reprogramming work that further re-train the last classifier.*

Method	Net	Train. Mem.	MNIST	CIFAR10	Flower	CUB	Aircraft
Adv. Rep [10]	ResNet50	98MB	94.3	62.8	-	-	-
	MobileNetV2	53MB	93.1	58.3	-	-	-
	MobileNetV3	42MB	93.5	59.4	-	-	-
Adv. Rep + Last [19]	ResNet50	99MB	-	92.8	91.1	66.9	55.8
	MobileNetV2	54MB	-	85.1	91.8	66.8	52.2
	MobileNetV3	46MB	-	92.0	92.6	62.3	54.2
Ours	ResNet50	119MB	-	***96.4 (↑ 3.6 %)***	***96.7***	***80.3***	***85.9***
	MobileNetV2	***51MB***	-	***95.0 (↑ 9.9 %)***	***95.0***	***76.9***	***81.6***
	MobileNetV3	***43MB***	-	***95.3 (↑ 3.3 %)***	***95.1***	***77.5***	***78.9***

that, Rep-Net consists of 6 modules for all these four models. The detailed architecture descriptions are relegated to the Appendix.

Training details. Following the setting in [2], the models are fine-tuned for 50 epochs using the Adam optimizer [18] with batch size 8 on a single GPU. The initial learning rate is tuned for each dataset while cosine schedule [25] is adopted for learning rate decay.

Evaluation Metric. For all the experiments, we report the transfer accuracy for each dataset. In addition, to measure the training efficiency, we also report the training memory consumption, including parameter size and activation memory storage during training.

4.2. Main Results

Comparison with input reprogramming methods As shown in Tab. 2, we compare the proposed Rep-Net with previous adversarial reprogramming works [10, 19] using ResNet50 [16], MobileNetv2 [32] and MobileNetv3 [17] models as the backbone pre-trained model. First, because the adversarial reprogramming method can only work on the dataset with small image resolution, we compare it to the CIFAR10 dataset. The results show that Rep-Net could significantly jump the accuracy by more than $\sim$ *30 %*. Furthermore, compared to the improved adversarial reprogramming work that further re-train the final classifier, our Rep-Net could also achieve *3.6%, 9.9% and 3.3%* accuracy improvement on ResNet50, MobileNetv2 and MobileNetv3, respectively. In addition, proposed Rep-Net outperforms the accuracy of adversarial reprogramming [19] across CUB and Aircraft dataset by more than $\sim$ *10 %*. In terms of memory, our Rep-Net requires reduced training memory overhead for the MobileNet architecture; but a slightly higher training memory is needed for the ResNet-50 backbone model.

Comparison with state-of-the-art methods We further compare our method with the SOTA transfer learning techniques as shown in Tab. 3, specially with the most recent technique TinyTL [2]. However, we must highlight one crucial distinction with TinyTL. By default, TinyTL [2] uses the pre-trained weights on the pre-training dataset to initialize their additional residual modules. In contrast, we only apply Rep-Net in the transfer learning/reprogramming phase by training it from scratch with the random initialization, which gives TinyTL method an unfair advantage. Nevertheless, compared to TinyTL, our proposed Rep-Net can still achieve better/on-par accuracy on all the seven transfer dataset. On top of that, our proposed Rep-Net can save $\sim$ *3-4 MB* training memory compared to TinyTL. For a fair comparison, we also report TinyTL Random Initialization results that show significantly inferior performance compared to Rep-Net.

5. Discussions

5.1. Does Rep-Net transfer better by using better ImageNet Models?

In [20], one exciting study shows that there is a strong correlation between the pre-trained ImageNet accuracy and its corresponding transfer learning domain accuracy. Such study arouses our interest to see if the proposed Rep-Net also follows the same phenomenon, which is summarized in Fig. 5. We find that Rep-Net also has a similar trend. The order of pre-trained ImageNet model accuracy predicts fine-tuning performance order on Flowers, CUB, CIFAR10 dataset. In contrast, ImageNet pretraining does not necessarily improve accuracy on Aircraft dataset. The reason concluded by [20] is that its data distribution is far away from the ImageNet dataset. Interestingly, our proposed Rep-Net outperforms TinyTL [3] by the most significant 2 % accuracy increment margin on that Aircraft dataset (see Tab. 3).

Table 3. *Comparison with previous State-of-the-art (SOTA) transfer learning methods using different backbone neural networks, where'I-V3' is Inception-V3; 'N-A' is NASNet-A Mobile; 'M2-1.4' is MobileNetV2-1.4; 'R-50' is ResNet-50; 'PM' is ProxylessNAS-Mobile. In this table, we show our improvements in comparison to best existing transfer learning scheme TinyTL [3].*

Method	Net	Train. mem	Reduce Ratio	Flowers	Cars	CUB	Food	Pets	Aircraft	CIFAR10	CIFAR100
FT-Full	I-V3 [29]	850MB	1.0×	96.3	91.3	82.8	88.7	-	85.5	-	-
	R-50 [20]	802MB	1.1×	97.5	91.7	-	87.8	92.5	86.6	96.8	84.5
	M2-1.4 [20]	644MB	1.3×	97.5	91.8	-	87.7	91.0	86.8	96.1	82.5
	N-A [20]	566MB	1.5×	96.8	88.5	-	85.5	89.4	72.8	96.8	83.9
FT-Last	I-V3 [29]	94MB	9.0×	84.5	55.0	-	-	-	45.9	-	-
TinyTL-Random [3]	PM	37MB	22.9×	88.0	82.4	72.9	79.3	84.3	73.6	95.7	81.4
TinyTL [3]	PM	37MB	22.9×	95.5	85.0	77.1	79.7	91.8	75.4	95.9	81.4
Ours	PM	***34MB (↓ 3)***	***25×***	***96.1***	***85.8***	***77.8***	***80.5***	***91.8***	***77.4(↑2%)***	***95.9***	***81.9***
TinyTL [3]	PM@320	65MB	13.1×	96.8	88.8	81.0	82.9	92.9	82.3	96.1	81.5
Ours	PM@320	***61MB (↓ 4)***	***13.9×***	***97.1***	***89.0***	***82.3(↑1.3%)***	***83.3***	***92.5***	***82.4***	***96.6***	***82.3***

Table 4. *Combining the Rep-Net with learnable binary mask based method for efficient inference. 'TinyTL-Last' means only re-training the last classifier on the TinyTL ImageNet pre-trained model. The inference Flops is reported on Flowers dataset.*

Method	Inference Flops	Flowers		Cars		CUB		Food		Pets		Aircraft		CIFAR10		CIFAR100	
		Acc	Sparsity	Acc	Sparsity	Acc	Sparsity	Acc	Sparsity	Acc	Sparsity	Acc	Sparsity	Acc	Sparsity	Acc	Sparsity
TinyTL-Last [2]	394.6	90.1	-	50.9	-	73.3	-	68.7	-	91.3	-	44.9	-	85.9	-	68.8	-
Ours	346.8	96.1	-	85.8	-	77.8	-	80.5	-	91.8	-	77.4	-	95.9	-	81.9	-
Ours+Bin. Mask	280.9	96.3	19.3	85	30.3	79.2	19.4	83.7	33.8	92.2	5.9	82.5	21.4	96.2	30.3	82.7	31.8

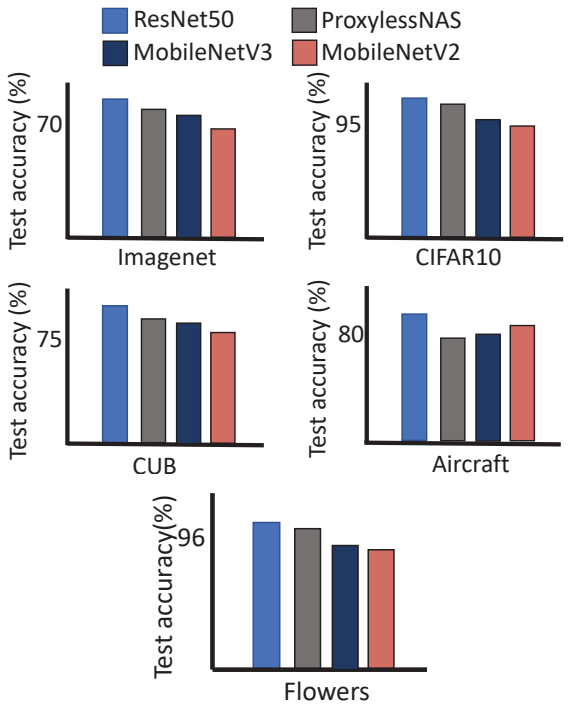

Figure 5. *The accuracy comparison between pre-trained imagenet and transfer to CIFAR10, CUB, Aircraft and Flowers on four different models.*

5.2. Can Rep-Net combine with other transfer learning methods for efficient inference?

The primary objective of Rep-Net is to enable on-device learning by reducing the training memory. In this subsection, we further explore if Rep-Net could combine with inference-efficient methods. To achieve this, we apply the learnable binary mask [28, 36] on the fixed main branch model combining with Rep-Net to reduce the computing cost for inference. As shown in Tab. 4, 'Ours+Bin. Mask" could further boost the accuracy and achieve $\sim 25\%$ sparsity on average across all the datasets. In summary, our designed Rep-Net can be incorporated in existing transfer learning schemes that look to change/train (e.g., mask) the backbone model as well.

6. Conclusion

In this work, we are the first to consider transfer learning from a new perspective of feature reprogramming of pre-trained model. To achieve on-device transfer learning, we propose Rep-Net, a novel architecture design, which takes the input data directly and learns to reprogram the intermediate feature of the pre-trained model. Moreover, we design dedicated activation connectors to perform feature exchange between the backbone and the Rep-Net models. Extensive experiments on transfer learning show the effectiveness and memory-efficiency of Rep-Net.

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

References

[1] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *European conference on computer vision*, pages 446–461. Springer, 2014. 6

[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. 3, 4, 6, 7, 8

[3] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. *arXiv preprint arXiv:2007.11622*, 2020. 1, 7, 8

[4] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 6

[5] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014. 1, 2, 3

[6] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. 1, 3

[7] 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, 6

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 2, 3, 6

[9] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 3

[10] Gamaleldin F Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial reprogramming of neural networks. *arXiv preprint arXiv:1806.11146*, 2018. 2, 3, 7

[11] R David Evans, Lufei Liu, and Tor M Aamodt. Jpeg-act: accelerating deep learning via transform-based lossy compression. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 860–873. IEEE, 2020. 3

[12] Jonathan Frankle, David J Schwab, and Ari S Morcos. Training batchnorm and only batchnorm: On the expressive power of random features in cnns. *arXiv preprint arXiv:2003.00152*, 2020. 2, 3

[13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 3

[14] 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

[15] 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. 2, 3

[16] 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. 4, 6, 7

[17] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019. 6, 7

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7

[19] Eliska Kloberdanz, Jin Tian, and Wei Le. An improved (adversarial) reprogramming technique for neural networks. In *International Conference on Artificial Neural Networks*, pages 3–15. Springer, 2021. 3, 7

[20] 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, 3, 6, 7, 8

[21] 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

[22] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[23] Liu Liu, Lei Deng, Xing Hu, Maohua Zhu, Guoqi Li, Yufei Ding, and Yuan Xie. Dynamic sparse graph for efficient deep learning. *arXiv preprint arXiv:1810.00859*, 2018. 3

[24] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3296–3305, 2019. 4, 6

[25] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 7

[26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. 3

[27] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 6

[28] 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, 8

[29] 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, 3, 6, 8

[30] 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. 6

[31] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE, 2012. 6

[32] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 4, 6, 7

[33] Yun-Yun Tsai, Pin-Yu Chen, and Tsung-Yi Ho. Transfer learning without knowing: Reprogramming black-box machine learning models with scarce data and limited resources. In *International Conference on Machine Learning*, pages 9614–9624. PMLR, 2020. 3

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

[35] Li Yang, Zhezhi He, Yu Cao, and Deliang Fan. Non-uniform dnn structured subnets sampling for dynamic inference. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020. 4, 6

[36] 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. 2, 8

[37] Li Yang, Adnan Siraj Rakin, and Deliang Fan. *da*$\tilde{3}$: Deep additive attention adaption for memory-efficient on-device multi-domain learning. *arXiv preprint arXiv:2012.01362*, 2020. 1, 3

[38] Guoqiang Zhong, Shoujun Yan, Kaizhu Huang, Yajuan Cai, and Junyu Dong. Reducing and stretching deep convolutional activation features for accurate image classification. *Cognitive Computation*, 10(1):179–186, 2018. 2, 3