FEDMORPH: COMMUNICATION EFFICIENT FEDERATED LEARNING VIA MORPHING NEURAL NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

The two fundamental bottlenecks in Federated Learning (FL) are communication and computation on heterogeneous edge networks, restricting both model capacity and user participation. To address these issues, we present FedMorph, an approach to automatically morph the global neural network to a sub-network to reduce both the communication and local computation overloads. FedMorph distills a fresh sub-network from the original one at the beginning of each communication round while keeps its 'knowledge' as similar as the aggregated model from local clients in a federated average (FedAvg) like way. The network morphing process considers the constraints, e.g., model size or computation flops, as an extra regularizer to the objective function. To make the objective function solvable, we relax the model with the concept of soft-mask. We empirically show that FedMorph, without any other tricks, reduces communication and computation overloads and increases the generalization accuracy. E.g., it provides an $85\times$ reduction in server-to-client communication and $18\times$ reduction in local device computation on the MNIST dataset with ResNet8 as the training network. With benchmark compression approaches, e.g., TopK sparsification, FedMorph collectively provides an $847\times$ reduction in upload communication.

1 INTRODUCTION

Federated Learning (FL) (Li et al., 2021; Bonawitz et al., 2019; Kairouz et al., 2019; Li et al., 2019) ensures data privacy by decoupling the training on the dataset in local clients and model aggregation in the global server. The iterative process of the local updates optimized on each client and aggregating the updates by the global server assures the convergence of the global model and the excellent generalization on test datasets.

(A). Despite the benefits gained by the paradigm of local-train and global-aggregation of FL, the server-clients communication burden and limited local computing resources suspend the deployment of the State-of-The-Art (SOTA) large neural networks on a wide scale. Recently the lossy model compression techniques, such as gradient sparsification (Wangni et al., 2017; Zhou et al., 2021) and quantization (Courbariaux et al., 2016; Hubara et al., 2017), have been studied to relieve the communication issues. Practical as they announced, but the limitations are also apparent. The gradient sparsification technique reduces the communication burden by selectively updating local gradients. Hence, they only work for client-to-server communication and have no compression benefits in the downstream direction. Meanwhile, the quantization technique compresses the communication and computation by changing the floating operations into low precision operations, and it works in both directions. However, its compression ability is at most $32\times$ by reducing 32-bits floating operations to the 1-bit operations, not to mention the performance degradation.

Absorbing the benefits from the former lossy compression techniques, FedDropout (Caldas et al., 2018) is another trial to reduce the downstream direction communication. It builds upon the basic idea of dropout (Srivastava et al., 2014) by randomly dropping some neurons of the global model at each round, resulting in a smaller sub-network with smaller weight matrices before broadcasting it to local clients. The updated sub-networks by local clients will map back to the global network.

Distillation of the 'knowledge' from an extensive neural network (Hinton et al., 2015) is another branch to solve local devices' communication and computation burdens. The main idea is to train a small model to keep the output similar to the large one on a sizeable prepared dataset. It has significant performance on the recent developments (Sanh et al., 2019) (Jiao et al., 2019). However, it requires a well pre-trained large model and a vast dataset for training the smaller network, which is not always satisfactory in the context of FL.

(B). Besides considering communication and computation budget, broadcasting a *raw* neural network may also cause a severe generalization problem. While it is almost common sense in the deep learning community that training a well-designed deep neural network performs better than a shallow one, the statement only establishes when the training dataset is sufficient to avoid overfitting (Pitt & Myung, 2002; Hinton et al., 2012). In FL settings, each local client holds a dataset in a small ratio of the whole, which is universal when the number of clients is enormous. Hence, human labor's well-designed deep neural network works well on the whole dataset, nevertheless easily overfit the local datasets. The overfitting will impede the number of local iterations, which is a crucial hyper-parameter in FL settings (McMahan et al., 2017), resulting in a deteriorated model performance. Kairouz et al. (2019) has a similar statement in their Network Architecture Search (NAS) section that a well-designed network architecture in centralized settings may not work well in federated learning settings.

This paper tries to solve the earlier issues by decoupling the network architectures on local clients and the global server. In detail, the global server maintains a large neural network suitable in a centralized setting, and keeps the architecture untouched throughout the entire process. During each communication round, the global server morphs a new sub-network from the maintained architecture and broadcasts it to selected clients for local optimization. The maintained network optimizes its weights at the morphing process by learning from the average aggregated network updated by local clients. Meanwhile, a newly morphed sub-network is constrained by keeping the knowledge similar to the average aggregated one while minimizing the number of its parameters. By morphing the shared neural network into a smaller size among all communication rounds, we (1) decrease the communication in both upload and download directions; (2) relieve the computation overload of local clients; (3) reduce the generalization error caused by overfitting on local datasets with a large network.

2 Related Works

Neural Architecture Search Our work is closely related to the NAS (Zoph & Le, 2016), whose purpose is automatically designing neural network architecture such as the number of layers and the number of neurons or filters in each layer. The search strategies include random search, evolutionary methods, Bayesian optimization, and gradient-based methods (Elsken et al., 2019). Since NAS involves a vast search space, recent works focus on improving the speed and efficiency by attentive sampling (Wang et al., 2021), untrained scheme (Mellor et al., 2021), and block-wisely search with knowledge distillation (Li et al., 2020). Besides, the network architecture search under the scenario of federated learning scenario was also explored in (Zhu et al., 2021), aiming to reduce both computation and communication.

Model Compression To reduce the complexity of deep neural networks, model compression was first proposed in (Buciluǎ et al., 2006), followed by tremendous attention in both academia and industry. One of the most straightforward method to reduce model size is parameter pruning and sharing by removing redundant parameters that are not critical to model performance (Han et al., 2015; Blakeney et al., 2020). Similarly, the informative parameters can also be measured and selected by low-rank factorization (Sainath et al., 2013; Denton et al., 2014). To simultaneously reduce the computation and storage of a deep model, approaches based on transferred or compact convolutional filters were further proposed by designing special structural kernels (Cohen & Welling, 2016; Wu et al., 2016) and achieved benefits in domains with human prior. Besides, some other works are focusing on transferring the learned knowledge of a large teacher network to a small and lightweight student network, which yields the concept of knowledge distillation (Hinton et al., 2015; Mirzadeh et al., 2020; Chen et al., 2021; Gao et al., 2021), which is suitable for small- or medium-size data sets (Cheng et al., 2018).

3 PROBLEM DEFINITION

3.1 PRELIMINARY

In this work, we consider the following federated learning optimization problem:

$$\min_{\mathbf{w}} \left\{ F(\mathbf{w}) \triangleq \sum_{k=1}^{K} p_k F_k(\mathbf{w}) \right\}, \tag{1}$$

where K is the number of clients, and p_k is the weight of the k-th client such that $p_k \geq 0$ and $\sum_{k=1}^{K} p_k = 1$. Suppose the k-th client holds the n_k training data: $x_{k,1}, x_{k,2}, \cdots, x_{k,n_k}$. The local objective $F_k(\cdot)$ is defined by

$$F_k(\mathbf{w}) \triangleq \frac{1}{n_k} \sum_{j=1}^{n_k} \ell\left(\mathbf{w}; x_{k,j}\right), \tag{2}$$

where $\ell(\cdot;\cdot)$ is a user-specified loss function. Specifically we consider a C class classification problem defined over a compact space $\mathcal{X}$ and a label space $\mathcal{Y} = [C]$, where $[C] = \{1, \ldots, C\}$. The data point $\{x, y\}$ is a random sample over $\mathcal{X} \times \mathcal{Y}$. A function $f_\theta : \mathcal{X} \to \mathcal{S}$ maps x to the probability simplex $\mathcal{S}$, where $\mathcal{S} = \left\{ \mathbf{z} \mid \sum_{c=1}^{C} z_c = 1, z_c \geq 0, \forall c \in [C] \right\}$, and $\mathbf{z}$ is a C-dimensional column vector. f_θ is parameterized over the hypothesis class $\mathbf{w}$, i.e., the weight of the neural network θ. We define the loss function $\ell(\mathbf{w}, x)$ with the widely used cross-entropy loss as $\ell(\mathbf{w}, x) = \mathbb{E}_y \left[\log f_\theta(\mathbf{w}, x)\right]$.

3.2 ALGORITHM DESCRIPTION

Definition 1. *Morphing Set: Given a neural network θ with its parameter weight $\mathbf{w}$, its Morphing Set ($\mathbf{\Theta}$, $\mathbf{W}$) is defined as a set that contains every pair of sub-network and the corresponding weight of the original network θ.*

We define the concept of *Morphing Set* for the convenience of explanation. Before the description of the proposed algorithm, we define θ^o and $\mathbf{w}^o$ as the *server maintained neural network* and its parameter weights, and denote ($\mathbf{\Theta}^o$, $\mathbf{W}^o$) as the *Morphing Set* of θ^o.

Here, we describe one round (say the t-th) of our proposed algorithm. First, the central server broadcasts the latest model, $(\theta_t, \mathbf{w}_t)$, to the selected clients (say the $\mathcal{K}$).

Second, every client (say the k-th) begins with $\mathbf{w}_{t,e=0}^k = \mathbf{w}_t$, and then performs $E \geq 1$ local updates with a randomly selected batch samples $\xi_{t,e}^k$ as follows,

$$\mathbf{w}_{t,e+1}^k \longleftarrow \mathbf{w}_{t,e}^k - \eta_{t,e} \nabla F_k \left(\mathbf{w}_{t,e}^k, \xi_{t,e}^k\right) \tag{3}$$

Third, unlike Federated Average (FedAvg) (McMahan et al., 2017) or other conventional algorithms letting $\tilde{\mathbf{w}}_{t+1} = \sum_{k \in \mathcal{K}} \frac{\mathbf{w}_{t,E}^k}{|\mathcal{K}|}$ be the model weights for next round, our method, in order to compress the communication, morphs a small neural network from θ^o by minimizing the performance divergence of the morphed model $\mathbf{w}_{t+1}$ with $\tilde{\mathbf{w}}_{t+1}$ as follows,

$$\begin{aligned}(\theta_{t+1}, \mathbf{w}_{t+1}) &= \underset{(\theta, \mathbf{w}) \in (\mathbf{\Theta}^o, \mathbf{W}^o)}{\arg\min} \mathcal{L}^t(\mathbf{w}; \tilde{\mathbf{w}}_{t+1}) \\ &= \underset{(\theta, \mathbf{w}) \in (\mathbf{\Theta}^o, \mathbf{W}^o)}{\arg\min} \sum_{x \in \mathcal{X}_v} \mathrm{JS}\left(f_\theta(\mathbf{w}, x) || f_{\theta_t}(\tilde{\mathbf{w}}_{t+1}, x)\right),\end{aligned} \tag{4}$$

where $\tilde{\mathbf{w}}_{t+1} = \sum_{k \in \mathcal{K}} \frac{\mathbf{w}_{t,E}^k}{|\mathcal{K}|}$ follows the FedAvg algorithm to be the average aggregated value of local clients updated parameter weights. $(\theta, \mathbf{w}) \in (\mathbf{\Theta}^o, \mathbf{W}^o)$ denotes the newly morphed network and its weights, which is a sub-network of the server maintained one. f_θ is the corresponding mapping function defined on the morphed neural network θ. The Jensen-Shannon (JS) divergence is a well-known metric for symmetrically measuring the distance of two probabilities. We apply the JS divergence as the loss function to minimize the output performance of the morphed sub-network

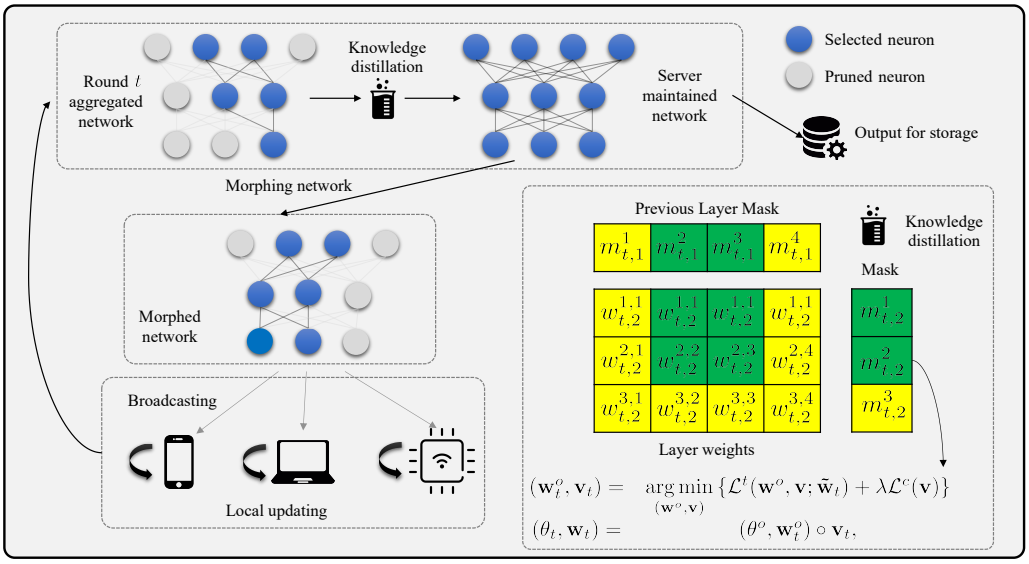

Figure 1: FedMorph procedures applied on three fully connected layers. The server maintained network weights and masks are backward updated according to the knowledge distilled from t round aggregated network. The network for the $(t+1)$ round is morphed from the server maintained network by the soft masks. The intra-layers are constrained by the masks to keep the layer coherent with the activation dimensions. The color of yellow and green means the corresponding values are masked and unmasked.

and average aggregated network on the validation compact space. Here, we use the finite-sum on the validation dataset $\mathcal{X}_v$ (without label information) to evaluate the performance divergence.

Finally, the newly morphed sub-network with the updated weights $(\theta_{t+1}, \mathbf{w}_{t+1})$ are broadcasted to selected clients for next round local optimization.

The optimization objective is to maintain as much knowledge as possible from the aggregated model in previous round, while keeping the morphed network as small as possible. Therefore, we consider the network architecture as a regularizer and reformat equation (4) as follows,

$$(\theta_{t+1}, \mathbf{w}_{t+1}) = \underset{(\theta, \mathbf{w}) \in (\mathbf{\Theta}^o, \mathbf{W}^o)}{\arg\min} \left\{ \mathcal{L}^t(\mathbf{w}; \tilde{\mathbf{w}}_{t+1}) + \lambda \mathcal{L}^c(\theta) \right\}, \tag{5}$$

where $\mathcal{L}^c(\theta)$ is a regularizer to measure the constraints on the number of neural network parameters (PARAMs) or the computation floating-point operations per second (FLOPs) or any other constraints as required. λ is an adjustable non-trained variable to balance the two losses.

4 Optimization Solution and Relaxation

4.1 Soft Masks

The optimization process of the equation (5) is incompatible with the sophisticated deep learning frameworks such as Pytorch (Paszke et al., 2019) or TensorFlow (Abadi et al., 2016) for the reasons that 1) the network architecture is not a well-defined parameter for optimization; 2) the categorical loss of the regularizer is incompatible with the neural network optimizer on differentiable functions.

In order to solve the issues mentioned above, we apply the soft-mask trick (Jang et al., 2016; Chaudhuri et al., 2020), a popular method to relax the categorical variables into continuous. Specifically, we append a soft-mask layer after each layer in the original neural network, which requires to be constrained. The soft-mask layer is a parameter vector whose dimension equals the number of constrained values of its previous layer. For instance, the dimension of the soft-mask layer would be the

Algorithm 1 Communication efficient federated morphing neural network.

1: **for** t in $\{0, \cdots, T-1\}$ **do**
2: **Broadcast** the masked results to selected clients;
3: **Local Update** $\mathbf{w}_t^k$ on the selected clients according to (3);
4: **Distill** knowledge from the aggregated model $\tilde{\mathbf{w}}_{t+1}$ to ($\mathbf{w}_{t+1}^o$, $\mathbf{v}_{t+1}$) according to (7);
5: **Morph** (θ^o, $\mathbf{w}_{t+1}^o$) with $\mathbf{v}_{t+1}$ to (θ_{t+1}, $\mathbf{w}_{t+1}$) for next step iteration;
6: **end for**
7: **return** (θ^o, $\mathbf{w}_T^o$);

number of the output channels if the previous layer is a conv2d layer; Or it would be the number of neurons if the previous layer is a linear layer. The activation outputs of the previous layer passing through the soft-mask layer have the same effect of masking directly on the previous layer.

For each variable in the soft-mask layer (parameter vector), we apply the differentiable S-shape-like functions $f : \mathbb{R} \to (0, 1)$ to relax the categorical representations into differentiable variables. The relaxation function is as follows,

$$m = Sigmoid\left(\left(\log(\frac{\pi}{1-\pi}) + l\right)/\tau\right). \tag{6}$$

As the temperature $\tau \to 0$, m approaches $\{0, 1\}$ with probability $\{1 - \pi, \pi\}$, respectively. $l \sim logistic$ is randomized to increases the expressiveness of the masked channel or neuron. Specifically, it leaks information with a possibility even when the channel or neuron is decided to be masked by π. The optimizer operates on $v = \log(\frac{\pi}{1-\pi})$ instead of π, as m's learnable parameters for numerical stability. We naively choose the standard logistic function, aka sigmoid function, for our differentiable relaxation. The comparisons among different kinds of S-shape-like functions is not the main object, and we leave it for the future research.

With the help of the soft-mask layer, the calculation of the architecture constraints is relaxed to the L1 norm of the soft-mask layer's parameters with a non-parametric multiplication factor. Taking the conv2d as an example, its number of output channels can be approximated as $\sum_{i=1}^{CH_o} m_i$, with CH_o being the number of output channels. Meanwhile, the output channel i would be pruned if $v_i < 0$ ($\pi < 0.5$), resulting in a regularized neural network.

Hence, with $\mathbf{v}_{t+1}$ being the neural architecture soft mask parameters at round $t + 1$, we re-organize the objective function (5) as follows:

$$\begin{aligned} (\mathbf{w}_{t+1}^o, \mathbf{v}_{t+1}) = &\quad \arg\min_{(\mathbf{w}^o, \mathbf{v})} \{\mathcal{L}^t(\mathbf{w}^o, \mathbf{v}; \tilde{\mathbf{w}}_{t+1}) + \lambda \mathcal{L}^c(\mathbf{v})\} \\ (\theta_{t+1}, \mathbf{w}_{t+1}) = &\quad (\theta^o, \mathbf{w}_{t+1}^o) \circ \mathbf{v}_{t+1}, \end{aligned} \tag{7}$$

where the operation $\circ$ means the pruning operation on the server maintained network architecture θ^o and the corresponding model weights $\mathbf{w}_{t+1}^o$ at the round $t + 1$ according to the mask parameters $\mathbf{v}_{t+1}$. The whole process of the proposed algorithm can be found from Algorithm 1, and the process of FedMorph is presented in Figure 1.

4.2 Intra-Layer Constraints

We relaxed the problem by soft-masks previously, and the constraints of two connected layers are solved in this subsection. The foot-stone of recent SOTA neural architectures, e.g., ResNet, VGG for Computing Vision (CV) tasks or Transformer, BERT for Natural Language Processing (NLP) tasks, is the stacking of shallow layers or blocks in depth to increase the express-ability. Pruning the previous layer will also influence the next layer. Consider the example of a two-layers MLP $f(x) = \sigma(x * W_1) * W_2$, with W_1 and W_2 denotes the weight matrix of linear mappings, and σ denotes the non-linear activation functions. Assume $\mathbf{m}_1$ and $\mathbf{m}_2$ be the learned masking layers of W_1 and W_2, taking effect on the columns, whose sizes equal the number of columns of W_1 and W_2, respectively. In order to be coherent with matrix multiplication, the pruned columns of W_1 also affect W_2, which will prune the corresponding rows of W_2. Moreover, we keep the final output layer unmasked to keep the coherence of the input and output of the morphed model with the server maintained one.

4.3 UNDERSTANDING FEDMORPH

Theorem 1. *The learning of server maintained network weights $\mathbf{w}^o$ with the knowledge distillation has the same effect of learning with random dropout layer and a regularizer loss.*

Proof. The learning process of randomly dropout (Srivastava et al., 2014) is as follows, 1) training the network weights with dropout layers; 2) evaluate on the test dataset without activating the dropout layers. By setting $\tau \to 0$, the weights of soft-mask layers would generate either 1 or 0 (approximated) and mimic the random dropouts characters on the neuron level. The knowledge transfer process of the first equation in (7) equals to the optimization of $\mathbf{w}^o$ with randomly dropout layers stacked after every learnable layer. The expectation on $(v + l)$ determines the dropout ratio. The convergence of $\mathbf{w}^o_{t+1}$ at each knowledge distillation process is guaranteed with sufficient researches on deep learning with dropouts. □

With Theorem 1, we know that the server maintained network will have a good performance on the validation dataset as long as the aggregated network performs well on it. Now, we want to study the generalization performance of the morphed network. In other words, how the *knowledge* on the training datasets of local clients is perceived by the morphed network via learning from the aggregated network on the validation dataset.

Theorem 2. *Let a sample $\mathcal{U}_S$ of size m drawn from $\mathcal{D}_S$ for knowledge distillation at round t, and $\delta < 1$ being its upper bound of the empirical error, such that $\frac{1}{m}\mathcal{L}^t_{x\in\mathcal{U}_S}(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1}) \leq \delta$. Then the expectation evaluation error of $\mathbf{w}_{t+1}$ on $\mathcal{D}_T$ is bounded, with the probability $1 - \delta$:*

$$\ell_{\mathcal{D}_T}(\mathbf{w}_{t+1}) \leq \ell_{\mathcal{D}_T}(\tilde{\mathbf{w}}_{t+1}) + \delta\left(1 + \sqrt{\frac{-m\log\delta}{2}}\right) + \frac{1}{2}d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right),$$

where $d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right)$ measures the distance of the knowledge distillation dataset distribution $\mathcal{D}_S$ and local training dataset distribution $\mathcal{D}_T$.

With Theorem 2, we know that the error upper bound of the morphed neural network on the local training dataset distribution mainly consists of 1) the expectation error of the averaged aggregated model on the local dataset; 2) the empirical error δ of knowledge distillation on the validation dataset; 3) terms dependent on the divergence between local training dataset distribution and evaluation dataset distribution. If the local training dataset and validation dataset follows the same distribution, we have $d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right) = 0$, then the knowledge perceived by the newly morphed network is only determined by the empirical risk of knowledge distillation.

5 EXPERIMENTS

5.1 SETUP

Datasets and Networks We evaluate the performance of our method on multiple tasks on different networks architectures, specifically on the dataset of CIFAR-10 (Krizhevsky & Hinton, 2010) with the network of ResNet18 (He et al., 2016) and VGG16 (Simonyan & Zisserman, 2014). All networks with the corresponding dataset are initialized randomly. We refer the readers to find the experimental results and details of other datasets and network architectures in the Appendix.

Baselines. We focus on testing our methods against already well-established FL benchmarks. In particular, we restrict our baseline algorithm to the use of FedAvg, the same algorithm we use to calculate $\tilde{w}_{t+1}$ in equation 4. To fairly evaluate the compression ratio of our proposed method, we combine the tricks of sparsification for client-to-server communication, precisely the Top-K sparsification trick proposed in (Lin et al., 2017), with the baseline algorithm. We also compare our method with FedDropout (Caldas et al., 2018), a strategy to compress communications in the server-to-client direction. Besides, we consider a new benchmark method, 'FedPruned,' which applies the FedAvg algorithm with a sub-network randomly pruned from the global network at initialization. The FedAvg, FedDropout, and FedPruned are also evaluated under the Top-K sparsification trick without losing fairness.

Table 1: The test accuracy, PARAMs and FLOPs compression ratio of VGG16 and ResNet18 on CIFAR-10 in IID partition. The compressed model PARAMs and FLOPs are normalized according to FedAvg. The maximum communication rounds are 4000 for both networks. For FedMorph, we output the averaged PARAMs and FLOPs before the test accuracy reaching the maximum value.

Method	VGG16 on CIFAR-10			ResNet18 on CIFAR-10		
	Accuracy	Params	Flops	Accuracy	Params	Flops
FedAvg	0.9167			0.9294		
FedAvg 25%	0.9140	1.000	1.000	0.9239	1.000	1.000
FedAvg 10%	0.9008			0.9149		
FedDropout	0.8987			0.9048		
FedDropout 25%	0.8885	0.3610	0.3622	0.8896	0.3607	0.3611
FedDropout 10%	0.8595			0.8589		
FedPruned	0.9063			0.9182		
FedPruned 25%	0.9054	0.3600	0.3705	0.9134	0.3534	0.3734
FedPruned 10%	0.8996			0.9086		
FedMorph	**0.9206**	**0.1066**	**0.2893**	**0.9311**	**0.1172**	**0.2289**
FedMorph 25%	**0.9175**	**0.0946**	**0.2357**	**0.9300**	**0.1165**	**0.2501**
FedMorph 10%	**0.9065**	**0.0884**	**0.2300**	**0.9200**	**0.0975**	**0.2027**

Hyper-Parameters. The dataset is partitioned within 100 clients at the initialization procedure in an IID fashion, and we randomly sample ten candidates for local optimization during each communication round. The optimized model gradients would be sparsified by the TopK method at a ratio of $\{10\%, 25\%\}$ before updating to the global server. For local training at each client, we use the SGD optimizer with momentum to be 0.9 and weight decay to be $5e-4$. We set the static learning rates of 0.05 for CIFAR-10. Each selected client trains for one epoch per round using a batch size of 100. We set the knowledge distillation learning rates as 0.005 and the batch size as 50 while keeping other hyper-parameters following the local training procedure. We use a static temperature τ of $1e-3$ for all experiments during the whole process. We set the regularizer as the morphed model parameters and evaluate our method on model PARAMs and FLOPs, and set the regularizer ratios λ of CIFAR-10 for ResNet18 and VGG16 as $1.2e-6$, and $1e-6$, respectively. The soft-mask layer parameter v is initialized randomly following the logistic distribution with its mean value being the preset model compression ratio. In detail, we set the mean value of $\mathbf{v}$ to be 0.3 for both networks. For FedDropout and FedPrune, we set the same initialized compression ratios of 0.6. Moreover, we set the random seed to be '12345' and built all experiments on a single GPU V100 to make the results reproducible.

Training Tricks. The knowledge distillation process that happens on the server-side at each communication round is computation-heavy. Therefore, to overcome the computation overloads, we 1) morphs the network when the aggregated model weights have changed significantly and 2) initialize the mapped part of the server maintained network with the average aggregated weights before the knowledge distillation procedure. In practice, we distill knowledge from the aggregated model every ten communication rounds and set the optimization epochs as 2 for every dataset.

5.2 Test Accuracy Comparison

We present the accuracy evaluation results as in Figure 2. In order to make a clear tendency, we apply the moving average on the result data points, saying to replace the point with the unweighted mean of its nearest 10 points. Besides, we report the maximal accuracy and the average compressed PARAMs and FLOPs in Table 1 for further clarification. The three main takeaways from these experiments are: 1) our proposed method produce competitive results (accuracy) comparing to the benchmarks; 2) FedMorph can further compress PARAMs and FLOPs without degrading the test accuracy; 3)FedMorph is compatible with the gradient sparsification methods, hence can further reduce the communication burden in FL settings. With the compressed PARAMs and FLOPs from Table 1, in other words, we achieves $\frac{1}{0.0884} \approx 11\times$ and $\frac{1}{0.0884} * 10 \approx 113\times$ reduction of communication for CIFAR-10 with VGG16 in both server-to-clients and clients-to-server sides respectively. Meanwhile, the computation FLOPs achieves $\frac{1}{0.2300} \approx 4\times$ reduction for it. Similar analysis on CIFAR-10 dataset with ResNet18 network, without degrading the test accuracy comparing with Fe-

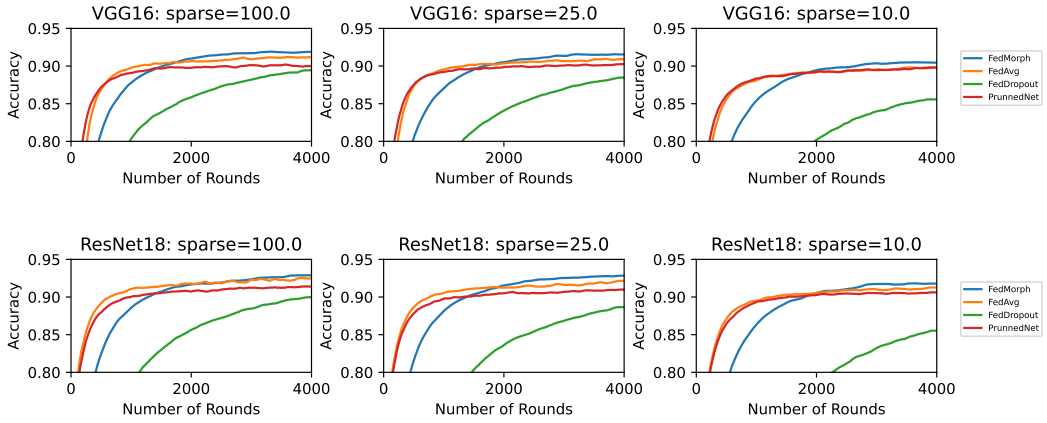

Figure 2: Test accuracy evaluation on CIFAR-10 with varied sparsification percentages and methods.

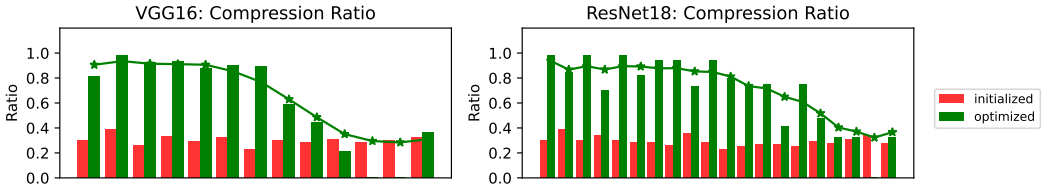

Figure 3: FedMorph compression ratio for each layer on CIFAR-10.

dAvg, we achieves around $10\times$, $102\times$, and $5\times$ reduction of server-to-clients side communication, clients-to-server side communication, and local clients computation FLOPs, respectively.

5.3 Layer Compression Ratio

Another observation from Table 1 is that FedMorph has unbalanced PARAMs and FLOPs compression ratios. Therefore, we plot the compressed ratio of each masked layer as presented in Figure 3. The labels on the x-axis denote the masked layers from shallow to deep. The red bar, blue bar, and blue line denote the initialized compression ratio (soft-mask layer parameters), optimized compression ratio (soft-mask layer parameters when test accuracy reaches the maximum), and the trendline of the blue bar with moving average, respectively. It is clear to observe that FedMorph tends to compress the deep layers while leaving the shallow layers uncompressed for both network architecture, even though the mask layer parameters are initialized uniform-randomly. The reason is that the architectures of ResNets and VGGs are like pyramids, which have heavy parameters at deep layers. The unbalanced compression ratio of each layer proves the effectiveness of the PARAMs regularizer in our objective function. Besides, by comparing the test accuracy with FedPruned, a method with randomly pruned neurons at initialization, we know that the PARAMs regularizer can not only reduce the model size but also aim at not decreasing its accuracy.

5.4 Generalization Gap

It is also necessary to know why FedMorph generalizes better than other baselines. We use the gap between the training accuracy and test accuracy to measure the generalization performance of the experimented algorithms, as presented in Figure 4. We calculate the training accuracy by the mean of local clients' accuracy, who has been selected for local updates between two consecutive times of the evaluation on the test dataset. For both datasets, Fedmorph has a significantly smaller generalization gap than that of FedAvg and FedDropout. It is worthwhile noticing that comparing to FedMorph, the training accuracy of FedAvg quickly reaches near to 1, and the training accuracy of

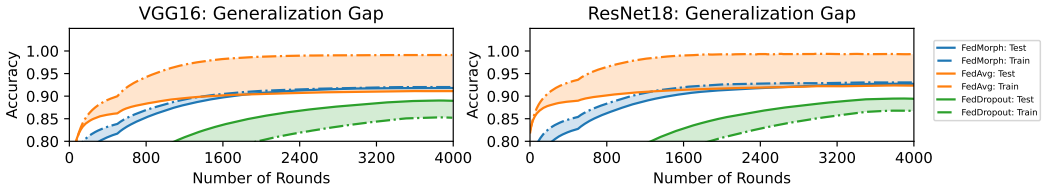

Figure 4: The generalization gaps for different methods on CIFAR-10.

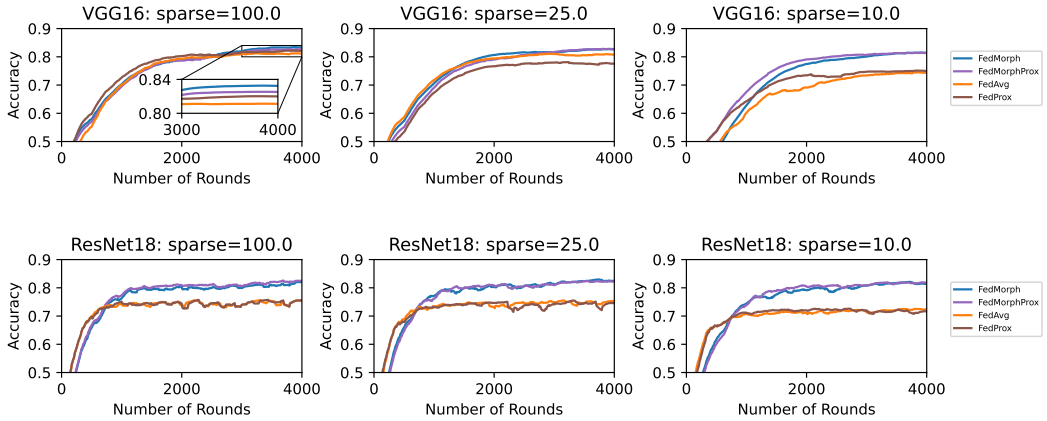

Figure 5: Test accuracy evaluation on CIFAR-10 with FedMorphProx in NonIID partition settings.

FedDropout is less than its test accuracy. FedAvg and FedDropout perform worse than FedMorph because FedAvg quickly gets over-fitted on the local dataset, which impedes its generalization well on the test dataset, while FedDropout somehow under-fits on the local training dataset. FedMorph balances well on the training and the evaluation parts, away from either over-fitting or under-fitting, making it generalizes the best among its competitors.

5.5 FedMorph Variant

This section evaluates FedMorphProx, a variant of FedMorph, by replacing the FedAvg algorithm for aggregation to FedProx (Li et al., 2018), on the NonIID partitions. FedProx limits the over-fitting of FedAvg on local datasets by constraining the local updated parameters not far away from the received model parameters at the beginning. In experimental details, we set the number of samples of each class in each client follows a Dirichlet distribution with $\alpha = 0.4$ and set the proximal ratio μ in FedProx as $1e-2$ for CIFAR-10 while keeping other hyper-parameter untouched. We present the test accuracy results in Figure 5. The FedMorphProx and FedMorph algorithms perform significantly better than either FedAvg or FedProx. Besides, FedMorphProx performs as well as FedMorph, proving our method works for FedAvg and other FedAvg-like FL algorithms.

6 Conclusion

In this paper, we proposed that sharing a sub-network among the clients and the server can reduce edge devices' communication and computation overloads and increase the generalization accuracy by avoiding over-fitting of FL on local datasets. The empirical experiments on varied datasets and network architectures with different FL settings, e.g., IID partition and Dirichlet partition, proved the superior performance of FedMorph. By analyzing the optimized morphed network, our work might also guide the design of network architectures in FL settings in the future.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

Cody Blakeney, Yan Yan, and Ziliang Zong. Is pruning compression?: Investigating pruning via network layer similarity. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 914–922, 2020.

K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*, 2018.

Shraman Ray Chaudhuri, Elad Eban, Hanhan Li, Max Moroz, and Yair Movshovitz-Attias. Fine-grained stochastic architecture search. *arXiv preprint arXiv:2006.09581*, 2020.

Defang Chen, Jian-Ping Mei, Yuan Zhang, Can Wang, Zhe Wang, Yan Feng, and Chun Chen. Cross-layer distillation with semantic calibration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7028–7036, 2021.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35 (1):126–136, 2018.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pp. 2990–2999, 2016.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

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

Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, pp. 1269–1277, 2014.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Mengya Gao, Yujun Wang, and Liang Wan. Residual error based knowledge distillation. *Neurocomputing*, 433:154–161, 2021.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

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*, pp. 770–778, 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7):1–9, 2010.

Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1989–1998, 2020.

T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873*, 2019.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pp. 6357–6368. PMLR, 2021.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In *International Conference on Machine Learning*, pp. 7588–7598. PMLR, 2021.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5191–5198, April 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32: 8026–8037, 2019.

Mark A Pitt and In Jae Myung. When a good fit can be bad. *Trends in cognitive sciences*, 6(10): 421–425, 2002.

Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6655–6659, 2013.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. Attentivenas: Improving neural architecture search via attentive sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6418–6427, June 2021.

Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. *arXiv preprint arXiv:1710.09854*, 2017.

Bichen Wu, Alvin Wan, Forrest Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. *arXiv preprint arXiv:1612.01051*, 2016.

Xiao Zhou, Weizhong Zhang, Hang Xu, and Tong Zhang. Effective sparsification of neural networks with global sparsity constraint. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3599–3608, 2021.

Hangyu Zhu, Haoyu Zhang, and Yaochu Jin. From federated learning to federated neural architecture search: a survey. *Complex & Intelligent Systems*, 7(2):639–657, 2021.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A DETAILS OF INTRAL-LAYER CONSTRAINTS

We give details of the intra-layer constraints on these modules, which cannot be sequentially stacked, e.g., the skip connection. Skip connections, successfully applied in ResNet, a sub-module connecting two unconnected layers of the backbone network, have increased the depth of neural networks and improved the benchmark results of deep learning. In mathematics, skip connection can be presented as $out = out + skip(x)$, where x is the input of its previous connected layer, and out is the output of its last connected layer. The skip connection sub-module can have several stacked simple layers inside it or output the input directly. For the first condition, we attach the mask of its final layer with its last connected layer's mask. For the second condition, we constrain the mask of its last connected layer to be its previous connected layer's mask.

B EXPERIMENTAL SETUP

B.1 SUMMARY OF DATASETS AND NETWORKS

We conduct the experiments on the MNIST (Deng, 2012), EMNIST (Cohen et al., 2017), and CIFAR-10 (Krizhevsky & Hinton, 2010) to consider the tasks on multiple visual computation tasks. For MNIST and EMNIST, we apply ResNet8 (He et al., 2016) as global network architecture. For CIFAR-10, we apply ResNet18 (He et al., 2016) and VGG16 (Simonyan & Zisserman, 2014) as the global networks. The network architectures used in our experiments are simply presented in Table 7, Table 8, and Table 9 at the end of this appendix. We abbreviate the details of datasets as follows,

- **MNIST:** is a handwritten digits (0-9) classification task with 60,000 data points in the training dataset and 10,000 data points in the test dataset. The input of the dataset is a flattened 784-dimensional (28 × 28) image, and the output is a class label between 0 and 9.
- **EMNIST:** is a set of handwritten character digits derived from the NIST Special Database 19 and converted to a 28x28 pixel image format. One unbalanced split, named ByClass, has 62 unbalanced class labels with 697,932 training samples and 116,323 test samples.
- **CIFAR-10:** consists of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. There are 50,000 data points in the training dataset and 10,000 data points in the test dataset.

In abbreviation, the corresponding datasets and the networks are summarized in Table 2. The experimental settings can roughly evaluate the performance of FedMorph within different conditions:

- multiple tasks with the corresponding network architecture and dataset;
- the same task with different network architectures, e.g., CIFAR-10 with ResNet18 and VGG16;
- the same network architecture on different tasks, e.g., MNIST and EMNIST with ResNet8;
- the huge number of available clients, e.g., EMNIST with 5000 available clients.

Table 2: The summary of datasets and network architectures.

Model	sample$_{train}$	sample$_{distill}$	#classes	PARAMs	FLOPs
MNIST	60,000	10,000	10	4,904,670	15,302,164
EMNIST	697,932	116,323	62	4,931,326	15,328,768
CIFAR10$_{ResNet18}$	50,000	10,000	10	11,173,962	556,651,520
CIFAR10$_{VGG16}$	50,000	10,000	10	14,728,266	314,031,616

B.2 HYPER-PARAMETERS

It is always artistic to find the optimal hyper-parameters applied in deep learning. Instead of optimizing the hyper-parameters, we assign them casually but reasonably and present the detailed hyper-parameters applied in our experiments. The dataset is partitioned within 100 clients in an artificially IID fashion at the initialization procedure for MNIST and CIFAR-10, and 5000 clients

Table 3: The summary of neural networks and hyper-parameters used in our experiments.

Model	lr_{train}	$lr_{distill}$	λ	$\mathbf{v}$	#epochs	#rounds	#clients
ResNet8$_{MNIST}$	0.1	0.01	1.6e-6	0.1	1	1500	100
ResNet8$_{EMNIST}$	0.1	0.01	2e-5	0.2	5	3000	5000
ResNet18$_{CIFAR10}$	0.05	0.005	1.2e-6	0.3	1	4000	100
VGG16$_{CIFAR10}$	0.05	0.005	1e-6	0.3	1	4000	100

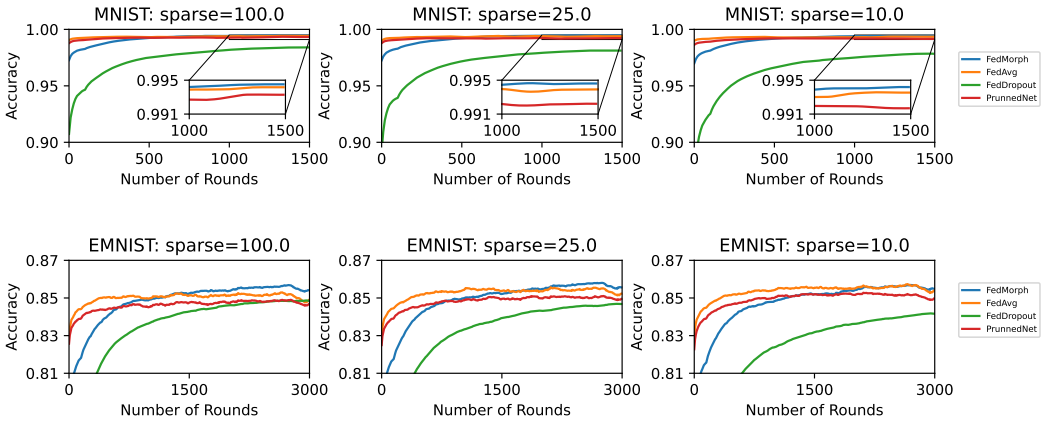

Figure 6: Test accuracy evaluation on MNIST and EMNIST with various sparsification percentages and methods.

for EMNIST. The updated model would be sparsified by the TopK method at a ratio of $\{10\%, 25\%\}$ before aggregation. For local training at each client, we use the SGD optimizer with momentum to be 0.9 and weight decay to be $5e-4$. We set the static learning rates of 0.1 for MNIST and EMNIST. For CIFAR-10, we select 0.05 as the learning rate. Ten clients are randomly selected to train for one epoch per communication round using a batch size of 100 for MNIST and CIFAR-10. For EMNIST, local clients train for five epochs per round. We set the knowledge distillation dataset as the test dataset without label information, and its training procedure follows the same SGD settings as the local training procedure. We set the distillation learning rates as one-tenth of that for local training and the batch sizes as 50, 500, and 50 for MNIST, EMNIST, and CIFAR-10, respectively. We use a static temperature τ of $1e-3$ for all experiments during each round. We set the regularizer as the morphed model parameters and evaluate our method on model PARAMs and FLOPs. We set the regularizer ratios λ to be $1.6e-6$ and $2e-5$ for MNIST and EMNIST. For CIFAR-10 with ResNet18 and VGG16, λ is $1.2e-6$, and $1e-6$, respectively. The soft-mask layer parameter v is initialized randomly following the logistic distribution, with its mean value being the preset model compression ratio. Specifically, we set the mean value of $\mathbf{v}$ to be 0.1, 0.2, 0.3, and 0.3, respectively. For FedDropout and FedPrune, we set the same initialized compression ratios of 0.25, 0.35, 0.6, and 0.6, respectively, for the four tasks. Moreover, the maximum communication rounds are 1500, 3000, 4000, and 4000. To make the results reproducible, we set the random seed to be '12345' for all experiments and build the experiments on GPU V100 within Pytorch. In conclusion, the corresponding hyper-parameters are listed in Table 3. If not specified, the hyper-parameters are kept unchanged.

C Additional Experimental Evaluations

This section presents the additional experimental results missed in the main text due to the page limitation. In the following, we evaluate the performance of FedMorph on the datasets of MNIST and EMNIST in terms of test accuracy, compression ratio, and generalization gap. At last, we present the comparison results of FedMorph and its transient networks to help understand the algorithm.

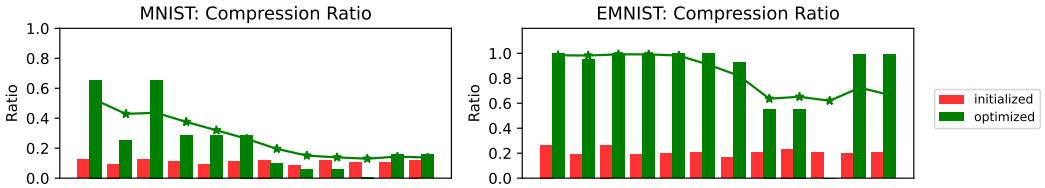

Figure 7: FedMorph compression ratio for each layer on MNIST and EMNIST.

Table 4: The test accuracy, PARAMs and FLOPs compression ratio of different methods on datasets of MNIST and EMNIST in IID partition. The compressed model PARAMs and FLOPs are normalized according to FedAvg. The maximum communication rounds are 1500 and 3000, respectively. For FedMorph, we output the averaged PARAMs and FLOPs before the test accuracy reaches the maximum value.

Method	MNIST			EMNIST		
	Accuracy	Params	Flops	Accuracy	Params	Flops
FedAvg	0.9954			0.8605		
FedAvg + 25% TopK	0.9950	1.000	1.000	0.8619	1.000	1.000
FedAvg + 10% TopK	0.9943			0.8627		
FedDropout	0.9870			0.8507		
FedDropout + 25% TopK	0.9844	0.0630	0.0705	0.8485	0.1240	0.1328
FedDropout + 10% TopK	0.9819			0.8436		
FedPruned	0.9941			0.8554		
FedPruned + 25% TopK	0.9934	0.0616	0.0765	0.8570	0.1209	0.1327
FedPruned + 10% TopK	0.9930			0.8590		
FedMorph	**0.9955**	**0.0120**	**0.0550**	**0.8622**	**0.1083**	**0.3643**
FedMorph + 25% TopK	**0.9951**	**0.0117**	**0.0492**	**0.8628**	**0.0838**	**0.2881**
FedMorph + 10% TopK	**0.9952**	**0.0118**	**0.0570**	**0.8625**	**0.0901**	**0.3463**

C.1 Test Accuracy Comparison

The results of test accuracy versus the number of communication rounds are presented in Figure 6. FedMorph converges on the dataset of MNIST and EMNIST with ResNet8, and generalizes slightly better than the benchmarks. The results of MNIST and EMNIST on the same network architecture of ResNet8 prove that FedMroph generates good results irrelevant to datasets. The average compression ratio when reaches the maximal test accuracy is presented in Table 4. For MNIST dataset with ResNet8, we achieves $\frac{1}{0.0118} \approx 85\times$ and $\frac{1}{0.0118} * 10 \approx 847\times$ reduction of server-to-client side communication and client-to-server side communication, and $\frac{1}{0.0570} \approx 18\times$ reduction of local clients computation. For EMNIST dataset with ResNet8, we achieves $\frac{1}{0.0901} \approx 1\times$ and $\frac{1}{0.0901} * 10 \approx 111\times$ reduction of server-to-client side communication and client-to-server side communication, and $\frac{1}{0.3463} \approx 3\times$ reduction of local clients computation.

C.2 Layer Compression Ratio

We present the layer compression ratio of FedMorph on MNIST and EMNIST in Figure 7. As expected in the analysis of CIFAR-10 with ResNet18 and VGG16, FedMorph compresses the deep layers of ResNet8 while keeping the shallow layers untouched to maintain its performance and the model size.

C.3 Generalization Gap

In order to prove that a small morphed network generates better generalization performance, we present the generalization gap of MNIST and EMNIST in Figure 8. FedMorph has a significantly smallest generalization gap for MNIST among the competitors. Comparing with FedMorph, FedAvg

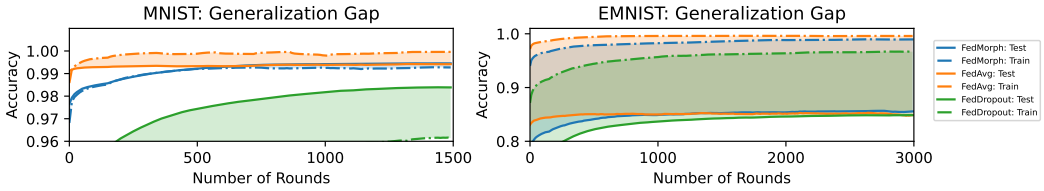

Figure 8: The generalization gaps for different methods on MNIST and EMNIST.

Table 5: The test accuracy, PARAMs and FLOPs compression ratio of different methods on datasets of MNIST and EMNIST in NonIID partition settings.

Method	MNIST			EMNIST		
	Accuracy	Params	Flops	Accuracy	Params	Flops
FedAvg	0.9928			0.8599		
FedAvg + 25% TopK	0.9919	1.000	1.000	0.8558	1.000	1.000
FedAvg + 10% TopK	0.9912			0.8444		
FedProx	0.9929			0.8587		
FedProx + 25% TopK	0.9924	1.000	1.000	0.8544	1.000	1.000
FedProx + 10% TopK	0.9899			0.8463		
FedMorph	0.9939	0.0117	0.0576	0.8606	0.0986	0.3240
FedMorph + 25% TopK	0.9934	0.0105	0.0495	0.8595	0.0904	0.3105
FedMorph + 10% TopK	0.9920	0.0103	0.0401	0.8588	0.0850	0.2968
FedMorphProx	0.9938	0.0105	0.0506	0.8599	0.1003	0.3252
FedMorphProx + 25%	0.9930	0.0102	0.0480	0.8597	0.0824	0.2761
FedMorphProx + 10% TopK	0.9934	0.0107	0.0448	0.8589	0.0932	0.3190

Table 6: The test accuracy, PARAMs and FLOPs compression ratio of different methods on datasets of CIFAR-10 in NonIID partition.

Method	VGG16 on CIFAR-10			ResNet18 on CIFAR-10		
	Accuracy	Params	Flops	Accuracy	Params	Flops
FedAvg	0.8323			0.7679		
FedAvg + 25% TopK	0.8262	1.000	1.000	0.7651	1.000	1.000
FedAvg + 10% TopK	0.7758			0.7357		
FedProx	0.8377			0.7667		
FedProx + 25% TopK	0.8015	1.000	1.000	0.7636	1.000	1.000
FedProx + 10% TopK	0.7774			0.7385		
FedMorph	0.8492	0.0903	0.2467	0.8295	0.0948	0.2161
FedMorph + 25% TopK	0.8444	0.0925	0.2497	0.8348	0.1012	0.3125
FedMorph + 10% TopK	0.8360	0.0934	0.2616	0.8278	0.0861	0.2991
FedMorphProx	0.8510	0.0953	0.2737	0.8355	0.0955	0.2210
FedMorphProx + 25% TopK	0.8446	0.0900	0.2611	0.8326	0.0882	0.2181
FedMorphProx + 10% TopK	0.8311	0.0935	0.2415	0.7357	0.9998	0.0078

is over-fitted as its training accuracy quickly reaches 1. While FedDropout is under-fitted as its training accuracy converges below that of FedMorph. For EMNIST, the dataset of size 697,932 is partitioned to 5000 clients, resulting in that each client has lesser than 140 samples. Furthermore, we set the number of local optimization epochs to be 5. Hence, all methods are easily over-fitted. However, comparing with FedAvg and FedDropout, FedMorph still performs better than them.

C.4 FedMorphProx with NonIID settings

We set the number of samples of each class in each client follows a Dirichlet distribution with $\alpha = 0.4$, and set the proximal ratio μ in FedProx to be $1e-2$ for all tasks. For MNIST and EMNIST, we decrease the local learning rates to be 0.01 for a smooth convergence while keeping other hyper-

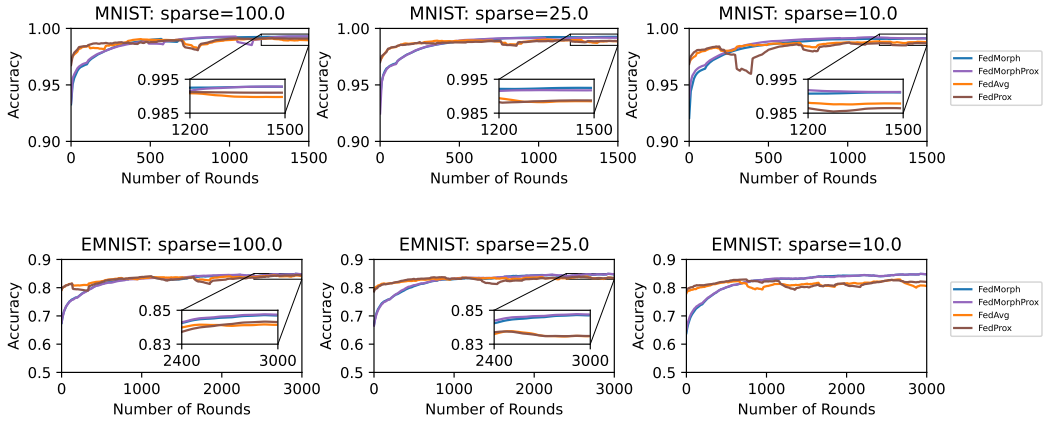

Figure 9: Test accuracy evaluation on MNIST and EMNIST with FedMorphProx in NonIID partition settings.

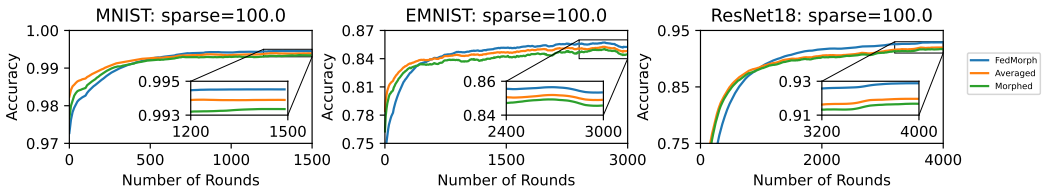

Figure 10: The test accuracy of FedMorph and the transient networks in IID partition settings.

parameter untouched. We present the result of test accuracy versus the number of communication rounds for MNIST and EMNIST in Figure 9. In our FL settings, FedMroph and FedMorphProx have slightly better accuracy results than that of FedAvg and FedProx. Besides, FedMorphProx performs as well as FedMorph, proving our method works for FedAvg and other FedAvg-like FL algorithms. It is also worthwhile pointing that FedMorph and FedMorphProx converge smoother than FedAvg and FedProx. We present the maximal test accuracy and the compressed PARAMs and FLOPs of all datasets in Table 5 and Table 6 for clarification.

C.5 Test Accuracy of the Transient Network

We study the performance of the transient networks, specifically, the morphed network $\mathbf{w}_t$ of each communication round t and the average aggregated network $\tilde{\mathbf{w}}_t$ of the morphed networks from local updates, and denote them as 'Morphed' and 'Averaged.' The results of the test accuracy versus the number of rounds for different datasets are presented in Figure 10. Interestingly, FedMorph has lower test accuracy than the other two transient networks but reaches the highest accuracy when converging. The explanation is that FedMorph learns knowledge from the smaller network 'Averaged' at an idle state, which causes its accuracy to be lower than that of 'Averaged' at the begging phase. By morphing a new network before each communication round, FedMorph can learn knowledge from offbeat teachers and store the knowledge inside the different parts of its larger network. When the smaller teacher network becomes converged, FedMorph combines all teachers' knowledge, hence generating better performance than the 'Averaged' when converging.

D Ablation Study

This section studies the convergence issues with different hyper-parameters, e.g., the initial values of v and the regularizer ratio λ. The following experiments are built on CIFAR-10 with ResNet18.

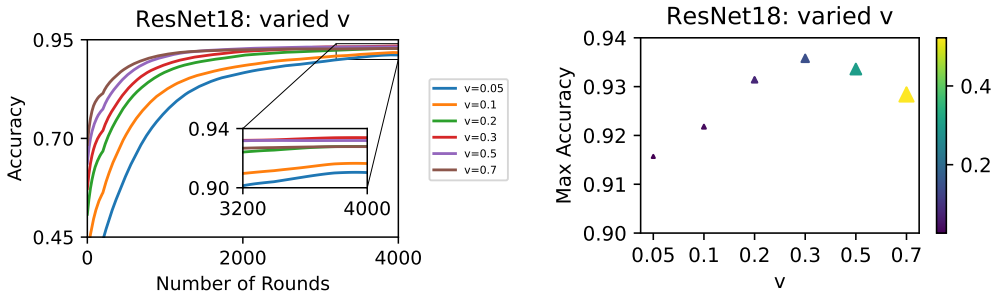

Figure 11: FedMorph test accuracy versus number of rounds for different initial compression ratios.

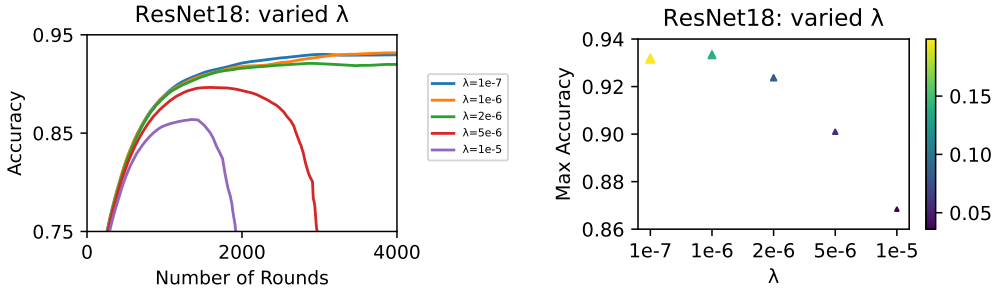

Figure 12: FedMorph test accuracy versus number of rounds for different model parameter regularizer ratios.

D.1 The Convergence Issues with Different v

For different initial v, we present the test accuracy versus the number of communication rounds and the maximal test accuracy versus varied v in Figure 11. We keep the hyper-parameter unchanged and set $\lambda = 1.2e-6$ for different experiments. The left-hand-side (LHS) figure presents that test accuracy converges faster when the initial v is higher. The right-hand-side (RHS) figure presents the maximal test accuracy and the average PARAMs compression ratio for different v when it reaches the maximal test accuracy. We use the size of the triangle and the color bar to denote the value of PARAMs. It appears that a more significant initial v leads to a higher compression ratio. Moreover, the maximal test accuracy increases with the initial v and decreases after reaching optimal. In this condition, the optimal initial v is around 0.3.

D.2 The Convergence Issues with Different λ

We present the results of test accuracy versus communication rounds of different regularizer ratios λ in Figure 12. We set the initial v to be 0.3 for all experiments and keep the hyper-parameters unchanged. The LHS figure presents that increasing the regularizer ratio to a large value will significantly deteriorate the test performance. The right-hand-side (RHS) figure presents the maximal test accuracy and the average PARAMs compression ratio when it reaches the maximal test accuracy for different regularizer ratios. As can be found from the color bar of triangles, a larger λ generates a smaller compression ratio but leads to terrible test accuracy. Moreover, an over-small λ can generate a consistent performance but increases the compression ratio. Therefore, an optimal λ exists to balance the test accuracy and compression ratio. In this condition, the optimal λ is around $1e-6$.

E UNDERSTANDING FEDMORPH

In this section, we want to understand

- why learning from a small average aggregated sub-network guarantees the performance of the global model;
- how much 'knowledge' is transmitted from the average aggregated model in the previous round to the next round.

In the following, we first analyze the convergence of server maintained network weights by considering the morphing process similar to adding dropout layers in Theorem 1. Then we analyze the bound of the expected error of the morphed model on both the training and test dataset in Theorem 2 and Theorem 3 with theorems from Domain Adaptation (Ben-David et al., 2010).

E.1 ANALYSIS ON SERVER MAINTAINED NETWORK

Theorem 1. *The learning of server maintained network weights $\mathbf{w}^o$ with the knowledge distillation has the same effect of learning with random dropout layer and a regularizer loss.*

Proof. The learning process of randomly dropout (Srivastava et al., 2014) is as follows, 1) training the network weights with dropout layers; 2) evaluate on the test dataset without activating the dropout layers. By setting $\tau \to 0$, the weights of soft-mask layers would generate either 1 or 0 (approximated) and mimic the random dropouts characters on the neuron level. The knowledge transfer process of the first equation in (7) equals to the optimization of $\mathbf{w}^o$ with randomly dropout layers stacked after every learnable layer. The expectation on $(v + l)$ determines the dropout ratio. The convergence of $\mathbf{w}^o_{t+1}$ at each knowledge distillation process is guaranteed with sufficient researches on deep learning with dropouts. □

E.2 ANALYSIS ON MORPHED NETWORK

We want to study how the morphed neural network perceives the knowledge of local training datasets from the distillation on the average aggregated model with the validation dataset. Our theories and proofs referred a lot from Ben-David et al. (2010).

Before giving any theories, we make some important definitions to clarify the problem. Define a *domain* as a pair consisting of a distribution $\mathcal{D}$ on inputs $\mathcal{X}$ and the labeling function $f : \mathcal{X} \to [0, 1]$, which can have a fractional (expected) value when labeling occurs non-deterministically. Initially, we construct two domains with the validation dataset for knowledge distillation and the training datasets for local optimization, denoted by $\langle \mathcal{D}_S, f_S \rangle$, and $\langle \mathcal{D}_T, f_T \rangle$, naming as source domain and target domain, respectively. The labeling function $f_{\mathcal{D}_S}(x)$ and $f_{\mathcal{D}_T}(x)$ on the sample x generate the *True* label of x in corresponding datasets.

Define a hypothesis as a function $h : \mathcal{X} \to \{0, 1\}$. The probability according to the distribution $\mathcal{D}_S$ that a hypothesis h disagrees with a labeling function f (which can also be a hypothesis) is defined as

$$\ell_S(h, f) = \mathrm{E}_{\mathbf{x} \sim \mathcal{D}_S}[|h(\mathbf{x}) - f(\mathbf{x})|]$$

When we want to refer to the source error (sometimes called risk) of a hypothesis, we use the shorthand $\ell_S(h) = \ell_S(h, f_S)$. We write the empirical source error as $\tilde{\ell}_S(h)$. The similar definitions can be built on $\ell_T(h)$, $\ell_T(h, f_T)$ and $\tilde{\ell}_T(h)$.

Proposition 1. *(Hoeffding's inequality) Let $x_1, \ldots, x_m$ be independent bounded random variables with $x_i \in [a, b]$ for all i, where $-\infty < a \leq b < \infty$. Then, for all $\zeta \geq 0$ we have,*

$$\Pr\left(\left|\frac{1}{m}\sum_{i=1}^{m}(x_i - \mathbb{E}[x_i])\right| \geq \zeta\right) \leq \exp\left(-\frac{2m\zeta^2}{(b-a)^2}\right)$$

Remark 1. *Hoeffding's inequality provides an upper bound on the probability that the sum of bounded independent random variables deviates from its expected value by more than a certain*

amount. Furthermore, if $x_1, \ldots, x_m$ are sampled from the same distribution such that $x \sim \mathcal{D}$, then Hoeffding's inequality can be simplized as follows,

$$\Pr\left(\left|\frac{1}{m}\sum_{i=1}^{m} x_i - \mathbb{E}_{x\sim\mathcal{D}}\left[x\right]\right| \geq \zeta\right) \leq \exp\left(-\frac{2m\zeta^2}{(b-a)^2}\right)$$

Before giving the theories, we refer Ben-David et al. (2010) to define the symmetric difference hypothesis space $\mathcal{H}\Delta\mathcal{H}$and the bounding of any two hypothesises in different domains as follows.

Definition 2. *For a hypothesis space $\mathcal{H}$, the symmetric difference hypothesis space $\mathcal{H}\Delta\mathcal{H}$ is the set of hypotheses*

$$g \in \mathcal{H}\Delta\mathcal{H} \Longleftrightarrow g(\mathbf{x}) = h(\mathbf{x}) \oplus h'(\mathbf{x}) \quad \textit{for some } h, h' \in \mathcal{H}$$

where $\oplus$ is the XOR function. In words, every hypothesis $g \in \mathcal{H}\Delta\mathcal{H}$ is the set of disagreements between two hypotheses in $\mathcal{H}$.

The following simple lemma shows how we can make use of the $\mathcal{H}\Delta\mathcal{H}$-divergence in bounding the error of our hypothesis.

Lemma 1. *For any hypotheses $h, h' \in \mathcal{H}$, the error difference in any two domains S and T is bounded as follows,*

$$\left|\ell_S\left(h, h'\right) - \ell_T\left(h, h'\right)\right| \leq \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right)$$

Proof. By the definition of $\mathcal{H}\Delta\mathcal{H}$-distance, we have

$$\begin{aligned} d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right) &= 2 \sup_{h,h'\in\mathcal{H}} \left|\Pr_{x\sim\mathcal{D}_S}\left[h(x) \neq h'(x)\right] - \Pr_{x\sim\mathcal{D}_T}\left[h(x) \neq h'(x)\right]\right| \\ &= 2 \sup_{h,h'\in\mathcal{H}} \left|\ell_S\left(h, h'\right) - \ell_T\left(h, h'\right)\right| \geq 2\left|\ell_S\left(h, h'\right) - \ell_T\left(h, h'\right)\right| \end{aligned} \tag{1}$$

□

With the bounding of Lemma 1, we give the following two lemmas on domain adaptation on different hypothesises.

Lemma 2. *For any two hypothesises $h, h' \in \mathcal{H}$,*

$$\ell_T(h) \leq \ell_T(h') + \ell_S(h, h') + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right)$$

Proof. This proof relies on the triangle inequality for classification error, which implies that for any labeling functions f_1, f_2, and f_3, we have $\ell(f_1, f_2) \leq \ell(f_1, f_3) + \ell(f_3, f_2)$. Therefore, we have,

$$\begin{aligned} \ell_T(h) &\leq \ell_T(h') + \ell_T(h, h') \\ &\leq \ell_T(h') + \ell_S(h, h') + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right) \end{aligned} \tag{2}$$

□

Lemma 3. *Define $\lambda \triangleq \ell_S(h^*) + \ell_T(h^*)$ with h^* be the optimized hypothesis of the combined datasets, then for any hypothesis $h, \tilde{h} \in \mathcal{H}$,*

$$\ell_S(h) \leq \ell_T(h') + \ell_S(h, h') + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right) + \lambda$$

Proof. This proof relies on the triangle inequality for classification error, which implies that for any labeling functions f_1, f_2, and f_3, we have $\ell(f_1, f_2) \leq \ell(f_1, f_3) + \ell(f_3, f_2)$. Therefore, we have,

$$\begin{aligned} \ell_S(h') &\leq \ell_S\left(h^*\right) + \ell_S\left(h', h^*\right) \\ &\leq \ell_S\left(h^*\right) + \ell_T\left(h', h^*\right) + \left|\ell_S\left(h', h^*\right) - \ell_T\left(h', h^*\right)\right| \\ &\leq \ell_S\left(h^*\right) + \ell_T\left(h', h^*\right) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_T, \mathcal{D}_S\right) \\ &\leq \ell_S\left(h^*\right) + \ell_T(h') + \ell_T\left(h^*\right) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_T, \mathcal{D}_S\right) \\ &= \ell_T(h') + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right) + \lambda \end{aligned} \tag{3}$$

Combining the above inequality with the truth $\ell_S(h) \leq \ell_S(h') + \ell_S(h, h')$, we obtain the lemma. □

Remark 2. *The Lemma 2 and Lemma 3 give the bounding of hypothesis h on both the source and target domain with another hypothesis h' on the target domain. With two lemmas mentioned above, we can study the error of the morphed network on the local training datasets and validation dataset bounded by the average aggregated network on local datasets.*

The following lemma studies the expected error bound of the knowledge distillation process given the empirical error on m data samples with Hoeffding's inequality.

Lemma 4. *Let a sample $\mathcal{U}_S$ of size m drawn from $\mathcal{D}_S$ for knowledge distillation at round t, and $\delta < 1$ being its upper bound of the empirical error, such that $\frac{1}{m}\mathcal{L}^t_{x\in\mathcal{U}_S}(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1}) \leq \delta$. Then, with the probability $1 - \delta$ its expectation error is bounded:*

$$\mathbb{E}\left[\mathcal{L}^t(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1})\right] \leq \delta\left(1 + \sqrt{\frac{-m\log\delta}{2}}\right).$$

Proof. Because of the truth that $\mathcal{L}^t_x(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1}) \geq 0$ for $\forall x \in \mathcal{U}_S$ and the empirical loss of $\mathcal{L}^t_{x\in\mathcal{U}_S}(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1})$ is bounded by δ, we obtain

$$\mathcal{L}^t_x(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1}) \leq m\delta, \qquad \forall x \in \mathcal{U}_S \tag{4}$$

It means $\mathcal{L}^t_x(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1})$ is bounded by $[0, m\delta]$ for any random variable $x \in \mathcal{U}_S$. Apply the *Hoeffding's Inequality* here, we obtain as follows,

$$\Pr\left(\left|\mathcal{L}^t(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1}) - \mathbb{E}\left[\mathcal{L}^t(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1})\right]\right| \geq \zeta\right) \leq 2\exp\left(\frac{-2\zeta^2 m}{(m\delta)^2}\right) \tag{5}$$

Let $\delta = \exp(\frac{-2\zeta^2}{m\delta^2})$, we easily obtain that $\zeta = \delta\sqrt{\frac{-m\log\delta}{2}}$. Hence, we know that at least at probability $1 - \delta$ that

$$\mathbb{E}\left[\mathcal{L}^t(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1})\right] \leq \mathcal{L}^t(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1}) + \delta\sqrt{\frac{-m\log\delta}{2}} \tag{6}$$

It is also worthy noticing that, $\delta\sqrt{\frac{-m\log\delta}{2}} \to 0$ as long as $\delta \to 0$.

□

Theorem 2. *Let a sample $\mathcal{U}_S$ of size m drawn from $\mathcal{D}_S$ for knowledge distillation at round t, and $\delta < 1$ being its upper bound of the empirical error, such that $\frac{1}{m}\mathcal{L}^t_{x\in\mathcal{U}_S}(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1}) \leq \delta$. Then the expectation evaluation error of $\mathbf{w}_{t+1}$ on $\mathcal{D}_T$ is bounded, with the probability $1 - \delta$:*

$$\ell_{\mathcal{D}_T}(\mathbf{w}_{t+1}) \leq \ell_{\mathcal{D}_T}(\tilde{\mathbf{w}}_{t+1}) + \delta\left(1 + \sqrt{\frac{-m\log\delta}{2}}\right) + \frac{1}{2}d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right),$$

where $d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right)$ measures the distance of the knowledge distillation dataset distribution $\mathcal{D}_S$ and local training dataset distribution $\mathcal{D}_T$.

Proof. Let $\ell_{\mathcal{D}_T}(\mathbf{w}_{t+1})$, $\ell_{\mathcal{D}_S}(\tilde{\mathbf{w}}_{t+1})$ be the error of the morphed model and the average aggregated model on local training datasets. Combine with Lemma 2, we have

$$\ell_{\mathcal{D}_T}(\mathbf{w}_{t+1}) \leq \ell_{\mathcal{D}_T}(\tilde{\mathbf{w}}_{t+1}) + \ell_{\mathcal{D}_S}(\mathbf{w}_{t+1}, \tilde{\mathbf{w}}_{t+1}) + \frac{1}{2}d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right), \tag{7}$$

where $\ell_{\mathcal{D}_S}(\mathbf{w}_{t+1}, \tilde{\mathbf{w}}_{t+1}) = \mathbb{E}_{\mathcal{D}_S}\left[\mathcal{L}^t(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1})\right]$. Combine with Lemma 4, we obtain the theorem. □

Theorem 3. *Let a sample $\mathcal{U}_S$ of size m drawn from $\mathcal{D}_S$ for knowledge distillation at round t, and $\delta < 1$ being its upper bound of the empirical error, such that $\frac{1}{m}\mathcal{L}^t_{x\in\mathcal{U}_S}(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1}) \leq \delta$. Then the expectation evaluation error of $\mathbf{w}_{t+1}$ on $\mathcal{D}_S$ is bounded, with the probability $1 - \delta$:*

$$\ell_{\mathcal{D}_S}(\mathbf{w}_{t+1}) \leq \ell_{\mathcal{D}_T}(\tilde{\mathbf{w}}_{t+1}) + \delta\left(1 + \sqrt{\frac{-m\log\delta}{2}}\right) + \frac{1}{2}d_{\mathcal{H}\Delta\mathcal{H}}\left(\mathcal{D}_S, \mathcal{D}_T\right) + \lambda \quad ,$$

where $d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$ measures the distance of the validation dataset $\mathcal{D}_S$ and local training dataset $\mathcal{D}_T$, and $\lambda = \inf_{h\in\mathcal{H}}(\ell_{\mathcal{D}_T}(h) + \ell_{\mathcal{D}_S}(h))$ is the optimal loss on the combined dataset distribution;

Proof. Let $\ell_{\mathcal{D}_T}(\mathbf{w}_{t+1})$, $\ell_{\mathcal{D}_T}(\tilde{\mathbf{w}}_{t+1})$ be the error of the morphed model and the average aggregated model on local training datasets. Combine with Lemma 3, we have

$$\ell_{\mathcal{D}_S}(\mathbf{w}_{t+1}) \leq \ell_{\mathcal{D}_T}(\tilde{\mathbf{w}}_{t+1}) + \ell_{\mathcal{D}_S}(\mathbf{w}_{t+1}, \tilde{\mathbf{w}}_{t+1}) + \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + \lambda, \tag{8}$$

where $\ell_{\mathcal{D}_S}(\mathbf{w}_{t+1}, \tilde{\mathbf{w}}_{t+1}) = \mathbb{E}_{\mathcal{D}_S}[\mathcal{L}^t(\mathbf{w}_{t+1}; \tilde{\mathbf{w}}_{t+1})]$. Combine with Lemma 4, we obtain the theorem. □

Table 7: ResNet8 Architecture

Layer	Kernel	Stride	Channels
Input	-	-	3
Conv+BN+ReLU	3	1	64
Conv+BN+ReLU	3	1	64
Conv+BN	3	1	64
Skip (Conv+BN+ReLU)	1	1	64
Conv+BN+ReLU	3	2	128
Conv+BN	3	1	128
Skip (Conv+BN+ReLU)	1	2	128
Conv+BN+ReLU	3	2	256
Conv+BN	3	1	256
Skip (Conv+BN+ReLU)	1	2	256
Conv+BN+ReLU	3	2	512
Conv+BN	3	1	512
Skip (Conv+BN+ReLU)	1	2	512
AvgPooling	4	4	512
Linear	-	-	10

Table 8: VGG16 Architecture

Layer	Kernel	Stride	Channels
Input	-	-	3
Conv+BN+ReLU	3	1	64
Conv+BN+ReLU	3	1	64
MaxPooling	2	2	64
Conv+BN+ReLU	3	1	128
Conv+BN+ReLU	3	1	128
MaxPooling	2	2	128
Conv+BN+ReLU	3	1	256
Conv+BN+ReLU	3	1	256
Conv+BN+ReLU	3	1	256
MaxPooling	2	2	256
Conv+BN+ReLU	3	1	512
Conv+BN+ReLU	3	1	512
Conv+BN+ReLU	3	1	512
MaxPooling	2	2	512
Conv+BN+ReLU	3	1	512
Conv+BN+ReLU	3	1	512
Conv+BN+ReLU	3	1	512
MaxPooling	2	2	512
AvgPooling	1	1	512
Linear	-	-	10

Table 9: ResNet18 Architecture

Layer	Kernel	Stride	Channels
Input	-	-	3
Conv+BN+ReLU	3	1	64
Conv+BN+ReLU	3	1	64
Conv+BN	3	1	64
Skip (Conv+BN+ReLU)	1	1	64
Conv+BN+ReLU	3	1	64
Conv+BN	3	1	64
Skip (Conv+BN+ReLU)	1	1	64
Conv+BN+ReLU	3	2	128
Conv+BN	3	1	128
Skip (Conv+BN+ReLU)	1	2	128
Conv+BN+ReLU	3	1	128
Conv+BN	3	1	128
Skip (Conv+BN+ReLU)	1	2	128
Conv+BN+ReLU	3	2	256
Conv+BN	3	1	256
Skip (Conv+BN+ReLU)	1	2	256
Conv+BN+ReLU	3	1	256
Conv+BN	3	1	256
Skip (Conv+BN+ReLU)	1	2	256
Conv+BN+ReLU	3	2	512
Conv+BN	3	1	512
Skip (Conv+BN+ReLU)	1	2	512
Conv+BN+ReLU	3	1	512
Conv+BN	3	1	512
Skip (Conv+BN+ReLU)	1	2	512
AvgPooling	4	4	512
Linear	-	-	10