MOON+GA: Federated Domain Generalization Using Both Global and Local Adjustments

AmirHossein Zamani (40168304)
Electrical and Computer Engineering
Concordia University, Montreal, Canada
amirhossein.zamani@concordia.ca

Anam Ayesha Shaykh (40205690)
Computer Science and Software Engineering
Concordia University, Montreal, Canada
anamayesha.shaikh@concordia.ca

***Abstract*—Domain generalization has attracted significant attention from various disciplines, including Federated Learning (FL). Methods that focus on this can be mainly divided into two directions: local and global model adjustment. Methods in the local direction conduct the adjustment on local model training on the client side, while methods in the global direction conduct the adjustment on the global model on the server side. Inspired by these two directions, we hypothesize that combining global and local adjustment methods could be beneficial to obtain more generalized results in the centralized federated setting. To this end, we first review methods in the literature and implement Generalization Adjustment [1] and MOON [2] as a global and local adjustment method, respectively. Then, we design a unified pipeline to combine the MOON and GA and train the model on a federated domain generalization dataset (PACS [3]). We quantitatively compare the results with baselines including FedAvg [4], FedProx [5], and SCAFFOLD [6] and show that when combined with GA, the MOON approach could give 1% improvement and better performance in terms of generalization. The code of all experiments is available at https://github.com/AHHHZ975/FedDG-Extension**

***Index Terms*—Centralized Federated Learning, Federated Domain Generalization, Local Adjustment, Global Adjustment**

I. Introduction

Conventional machine learning approaches often assume that the test data are well-represented by the training data, which is drawn from a target distribution. During the training phase, however, data from the target domains may be limited or altogether unknown in many practical cases. This leads to numerous degrees of heterogeneity between the test and training domains, resulting in test data that differs from training data in distribution. To address this challenge, the field of domain generalization has emerged. Domain generalization focuses on the development of models that can effectively generalize to new and even unseen populations that are not represented in the training domains [7].

Domain generalization has attracted significant attention from various disciplines, including Federated Learning (FL). More specifically, existing methods in the centralized setting of FL mainly focus on solving the heterogeneity issue from the optimization perspective [1]. Methods that focus on this can be mainly divided into two directions: local and global model adjustment. On the one hand, methods in the local direction conduct the adjustment on local model training at the client side, aiming at producing local models with smaller differences [8]. On the other hand, methods in the global direction conduct the adjustment on the global model on the server side, aiming at producing a global model with better performance [8].

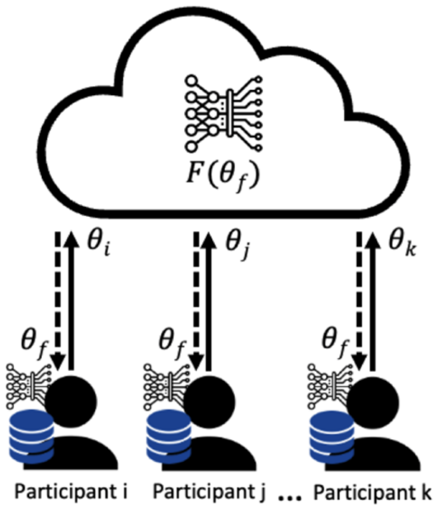


Fig. 1. An overview of how a centralized federated learning system works

Inspired by these two directions, we hypothesize that combining global and local adjustment methods could be beneficial to obtain more generalized results in the centralized federated setting. To this end, we suggest and implement the following steps as our contributions to this course project:

1) Reviewing global adjustment methods in the literature and re-implementing one of them (Generalization Adjustment [1])
2) Reviewing local adjustment methods in the literature and re-implementing one of them (MOON [2])
3) Combining the local and global adjustment techniques in a unified pipeline and training on a federated domain generalization dataset and comparing the results with baselines such as FedAvg [4], FedProx [5], and SCAFFOLD [6]

A. Existing methods

In a centralized setting of federated learning, shown in Fig. 1, there are two general steps: local updates (on the client side) and global aggregation (on the server side). On the one hand, in local updates, each worker (client) tries to perform several SGD steps to update its local model and then sends the parameters of the local model to the server and waits until the server sends back the updated parameters. On the other hand, in the global aggregation step, the server waits for all clients who participate in the federated algorithm to receive the parameters of their local model and then performs a simple weighted average on these parameters to compute the global model parameters. Note that, in this average operation, the weights are computed based on the data size of each client. In other words, each client contributes according to its data size in the global model. Having known this explanation of these two steps in the centralized FL setting, the local adjustment techniques focus on improving the performance of the global model by changing the local updates while the global adjustment techniques focus on improving the performance of the global model by changing the aggregation step. Hence, to understand better how these methods work, we review some of the local and global adjustment methods existing in the literature.

1) Global Adjustment Techniques: FedDG [1] is the main paper we focus on for the global adjustment technique in this project. The most important finding of FedDG [1] is that the dataset size is not the only factor for weighting the client updates in the aggregation step on the server side which is used by many well-known baselines such as FedAvg [4]. Hence, the authors in that work introduce the concept "Generalization Gap" which represents the difference between the global model's performance on a domain and the performance of the previous local model. They show that the generalization gap between local and global models could be a beneficial and complementary indicator for determining aggregation weights. Although existing methods try to address the issue by designing a training pipeline on the client side, authors in FedDG [1] believe that focusing only on an improved local training strategy cannot guarantee that the global model is generalizable enough to unseen domains. Therefore, they propose the FedDG that addresses the generalization issue on the server side by dynamically adjusting the aggregation weights of clients. More specifically, in FedDG [1], authors design a method called Generalization Adjustment that dynamically calibrates aggregation weights based on the observed generalization gaps during training to optimize the global objective. Authors claim that GA leads to a tighter generalization bound. Intuitively, General Adjustment incorporates the distribution of local data domains in the global objective to have more generalized results.

FedDisco [8] is another work focusing on the aggregation step. The primary objective of FedDisco is to learn representations that are both sensitive to domain-specific variations and invariant to domain shifts, thereby achieving better generalization. FedDisco addresses domain shifts in federated learning by introducing a disentangled representation learning approach. More specifically, the algorithm disentangles domain-specific and domain-invariant features during training, helping to mitigate the impact of domain shift and improve generalization across different domains.

2) Local Adjustment Techniques: Ditto [9] proposes a federated learning algorithm that creates personalized models for each client. Authors in that work introduce the concept of "Fairness" which measures how much the test performance of the clients varies across the network. Hence, one of the objectives of that research is to improve fairness similar to FedDG [1]. It means reducing the variance of the results between client models to achieve more generalized results. To this end, an L2 regularization term is added to the objective of client models, which encourages the personalized client models to be close to the global model as much as possible.

MOON (Model Contrastive Federated Learning) [2] is a state-of-the-art federated learning algorithm for visual classification and tries to propose a method to handle the heterogeneity of local data distribution across clients. The key idea of MOON is based on model contrastive learning in which the client models try to keep their parameters close to the parameters of the global model and far from the parameters of the client model from the previous communication round. In this way, MOON ensures there is always an improvement in the local models until the local models converge to the global model. Note that, Moon applies the contrastive idea on the model level which is different than the original contrastive learning idea [10] that focuses on the input data (e.g. an image).

In SCAFFOLD [6], the objective is to enhance the convergence and efficiency of federated learning by mitigating challenges such as client drift, non-IID data distribution, and communication costs. By introducing the notion of "control variates", the algorithm aims to stabilize the training process and reduce the variance in local model updates, contributing to faster and more robust convergence to a global model. More specifically, SCAFFOLD estimates the update direction for the server model with a global control variate (c) and the update direction for each client with local control variates (c_i). The difference ($c - c_i$) is then an estimate of the client drift which is used to correct the local updates. Here, c is the control variate at the server side and c_i are control variates at each client i.

B. Contributions of this work

In this section, we present an overview of our contributions to this project. First, we aim to re-implement and evaluate the FedDG on the PACS [3] dataset and compare the original results presented in the FedDG [1] paper. Second, as an extension of the FedDG and inspired by the idea of recently published the paper FedDisco [8], we plan to combine MOON [2] as a local adjustment method, with General Adjustment (GA) presented in FedDG [1] as a global adjustment method in a centralized federated learning framework and analyze the effect of different local adjustment methods on the result of the FedDG algorithm. We hypothesize that this approach could potentially be a way to incorporate both personalization and generalization abilities in a centralized federated learning framework (like FedAVG). Moreover, we believe it could give better results in terms of generalization because we incorporate both state-of-the-art methods in the local generalization and global generalization.

II. Preliminaries

In this section, we present some background on the methods we utilize as baselines in this project.

A. FedAvg [4]

In FederatedAveraging (FedAvg) [4] algorithm, there is a central server (parameter server) with the global model parameter θ and several clients (workers) with local model parameters θ_i in which i represents the index of a specific client in the set of clients. In each communication round, the server selects some clients for computation and sends them the global model parameter θ. Then, every selected client locally takes several steps of gradient descent on the current model using its local data and then sends its new parameters to the server (local update step). After all selected clients perform the local update step, the server aggregates the local model parameters into the latest global model parameters by taking a weighted average of local model parameters (aggregation step). The objective of FedAvg can be summarized as:

$$\min_{\theta} L_D(\theta) = \sum_{i=1}^{M} p_i L_{D_i}(\theta_i) \text{ s.t. } p_i = \frac{N_i}{\sum_{i'=1}^{M} N_{i'}} \quad (1)$$

where θ is the server model parameter, θ_i is the model parameter of client i, L is the loss function, D_i is the data domain of client i, D is the aggregation of all data domains, p_i is the aggregation weight corresponding to the client i, N_i is the size of local data of client i, and M is the number of clients.

B. FedProx [5]

FedProx [5] is similar to FedAvg except that FedProx has an additional term called proximal term that is added to local objectives (loss function of clients). Therefore, the objective of FedProx can be summarized as:

$$\begin{aligned} \min_{\theta} L_D(\theta) &= \sum_{i=1}^{M} p_i L_{D_i}(\theta_i) + \frac{\mu}{2} \|\theta - \theta_i\|^2 \\ \text{s.t. } p_i &= \frac{N_i}{\sum_{i'=1}^{M} N_{i'}} \end{aligned} \quad (2)$$

The proximal term is beneficial to address the issue of statistical heterogeneity (different distribution and size of data in clients) by restricting the local updates to be close to the global model without any need to manually set the number of local epochs. In this way, the algorithm ensures that the local models do not drift from the global model.

C. SCAFFOLD [6]

Similar to FedProx [5], SCAFFOLD [6] is an algorithm that is added to the top of FedAVg (which is a typical algorithm in FL) and tries to address the heterogeneity by estimating client drift and then correct it. More specifically, authors in [6] introduce two variables called global control variate c and local control variate c_i. On the one hand, c_i estimates the update direction for the client i by storing the gradient value at the first iteration of the previous round. On the other hand, c estimates the update direction for the server by taking the average of the local control variate c_i. Then, $c - c_i$ which is an estimation of the client drift from the server, will be added to the update rule to reduce the client drift as much as possible. Hence, the local updated rule for the client parameters becomes the following equation:

$$\theta_i \leftarrow \theta_i - \eta \left(\nabla L_{D_i}(\theta_i) + c - c_i\right) \quad (3)$$

Note that the clients in SCAFFOLD are stateful and retain the value of c_i across multiple rounds. Further, if c_i is always set to 0, then SCAFFOLD becomes equivalent to FedAvg [4].

III. Methodology

To elaborate on our idea, we will first adopt Algorithm 1 from the FedDG paper [1], which focuses on reducing variations in generalization performance across diverse domains. The goal is to create a global model that is more capable of generalizing well across different scenarios by taking both local and global adjustment methods. More specifically, we face an optimization problem in which the optimization process is divided into two phases: The local training phase and the aggregation (global training) phase.

Algorithm 1 Generalization Adjustment (GA)

```
Input: Global model $\theta = \theta_0$, $M$ clients, $\hat{D} = \{\hat{D}_1, \hat{D}_2, ..., \hat{D}_M\}$, the initial weights $a^0 = (\frac{1}{M}, \frac{1}{M}, ..., \frac{1}{M})$. (Hyperparameters: local epoch $E$, total communication round $R$, and step size $d$ for GA.)
Output: Global model $\theta^R$
Server: initialize the local models $\theta_i^0$ by the global model: $\theta_i^0 = \theta^0$.
for all $r$ in $0...R1$ do
    Client:   // Local adjustment method: FedAvg [4], FedProx [5], and SCAFFOLD [6]
    Compute $G_{\hat{D}_i}(\theta_r)$ for $\theta_r$ on each client.
    $\theta_i^r = LocalAdjustmentAlgorithm(\theta_i^r, \hat{D}_i, E)$
    return $\theta_i^r$ and $L_{\hat{D}_i}(\theta_i^r)$ to server
    Server:   // Global adjustment method: GA [1]
    $a^r = GA(a^{r-1}, \{G_{\hat{D}_i}(\theta^r)\}_{i=1}^{M}, d^r)$
    $\theta^{r+1} = \sum_{i=1}^{M} a_i^r.\theta_i^r$
    Broadcast $\theta^{r+1}$ to all clients $\theta_i^{r+1} = \theta^{r+1}$
```

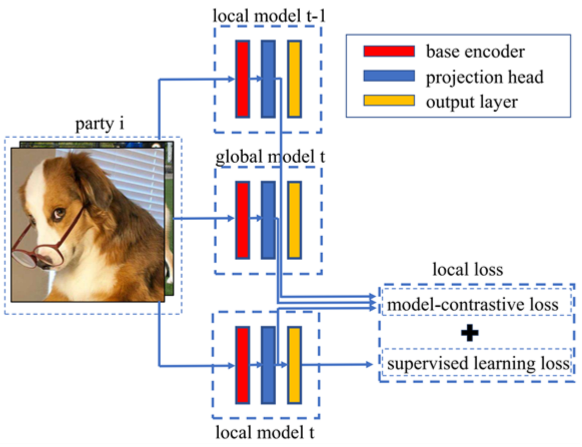


Fig. 2. The local loss in MOON [2]

A. *Local Training Phase*

In this phase, each client optimizes its local model parameters through gradient descent using its local dataset. However, this stage potentially differs from the one that is suggested in the FedDG [1] paper and Algorithm 1. We incorporate the local adjustment method proposed in MOON [2]. In this way, we would like to change the local objectives in a way that they tend to achieve more personalized and generalized results on their specific data. The key idea of MOON is based on model contrastive learning in which the client models try to keep their parameters close to the parameters of the global model and far from the parameters of the client model from the previous communication round. In this way, MOON ensures there is always an improvement in the local models until the local models converge to the global model. More specifically, we use MOON [2] in the local training phase which involves a two-part local loss for each participating client i. The first part is a conventional supervised learning loss L_{sup}, such as cross-entropy loss similar to FedAvg [4], FedProx [5], and SCAFFOLD [6], while the second part is a model-contrastive loss L_{con} proposed in MOON [2]. Specifically, in each communication round, every client i receives the global model θ^r from the server and updates its local model θ_i with θ. Then, each client computes the model-contrastive loss L_{con} based on the local representations of the input in the current communication round z, the global representation of the input computed by the server in the current communication round z_{glob}, and the local representation of the input from the previous communication round z_{prev}. The goal is to minimize the distance between z and z_{glob} while increasing the distance between z and z_{prev} (See Fig. 2). Then, the overall local objective for each client is a summation of the supervised loss and the weighted model-contrastive loss. This loss can be represented as:

Mathematically, the local loss in MOON is represented as:

$$L_{MOON} = L_{sup} + \mu L_{con} \tag{4}$$

where:

$$L_{sup} = CrossEntropyLoss(F_{\theta_i^r}(x), y),$$

$$L_{con} = -\log \frac{\exp(\text{sim}(z, z_{glob})/\tau)}{\exp(\text{sim}(z, z_{glob})/\tau) + \exp(\text{sim}(z, z_{prev})/\tau)}$$

The intuition behind this computation is based on the fact that, in each communication round, the global model is the best solution in the federated setting as it averages solutions of all the updated local models. On the other hand, the local models progressively obtain better solutions as training goes forward. Therefore, considering the convergence to the local minimum during the local training, each local model performs better in the current communication round rather than in the previous communication round. As a result, in each communication round, MOON tries to keep the parameters of the local models close to the parameters of the global model and keep the parameters of the local models far from their corresponding parameters in the previous communication round. In this way, MOON ensures a faster and better convergence.

B. *Aggregation (global training) Phase:*

After all the clients perform local training, the server performs a momentum update to compute the generalization weights by evaluating the generalization gaps on all domains. This step will be implemented in the same way the authors have done in [1]. These weights influence how much each client's model contributes to the global model. Mathematically, the overall objective of our MOON+GA framework becomes as:

$$\min_{\theta_1,\ldots,\theta_M,a} L_D(\theta) = \sum_{i=1}^{M} a_i L_{MOON}^{\hat{D}_i}(\theta) + \beta \,\text{Var}\left(\left\{G_{\hat{D}_i}(\theta)\right\}_{i=1}^{M}\right)$$
$$\text{s.t.} \sum_{i=1}^{M} a_i = 1, \theta = \sum_{i=1}^{M} a_i \cdot \theta_i, \text{ and } \forall i, a_i \geq 0. \tag{5}$$

where the generalization gap in the equation above is defined as is in [1]:

$$G_{D_i}(\theta) = G_{D_i}\left(\sum_j a_j \theta_j^*\right) = L_{D_i}\left(\sum_j a_j \theta_j^*\right) - L_{D_i}(\theta_i^*) \tag{6}$$

Algorithm 2 presents our main contribution in more detail in this project. In the local training, we try FedAvg [4], FedProx [5], SCAFFOLD [6], and MOON [2] and combine them with the General Adjustment (GA) method proposed in the FedDG [1] as a global adjustment method. The results of these different methods will be presented and compared in the next section.

Algorithm 2 The MOON+GA framework

```
Input: Global model $\theta = \theta_0$, $M$ clients, $\hat{D} = \{\hat{D}_1, \hat{D}_2, ..., \hat{D}_M\}$, the initial weights
       $a^0 = (\frac{1}{M}, \frac{1}{M}, ..., \frac{1}{M})$. (Hyperparameters: local epoch $E$, total communication round $R$, and step size
       $d$ for GA.)
Output: Global model $\theta^R$
Server: initialize the local models $\theta_i^0$ by the global model: $\theta_i^0$ = $\theta^0$.
for all $r$ in $0...R1$ do
    Client:                                   // Local adjustment method: MOON [2]
    for epoch $i$ = 1, 2, ..., $E$ do
        Compute $G_{\hat{D}_i}(\theta_r)$ for $\theta_r$ on each client.
        for each batch $b = \{x, y\}$ of $\hat{D}_i$ do
            $L_{sup} = CrossEntropyLoss(F_{\theta_i^r}(x), y)$
            $z = R_{\theta_i^r}(x)$
            $z_{glob} = R_{\theta^r(x)}$
            $L_{con} = -\log \frac{\exp(\text{sim}(z, z_{glob})/\tau)}{\exp(\text{sim}(z, z_{glob})/\tau) + \exp(\text{sim}(z, z_{prev})/\tau)}$
            $L = L_{sup} + \mu L_{con}$
            $\theta_i^r = \theta_i^r - \eta \nabla L$
    return $\theta_i^r$ and $L_{\hat{D}_i}(\theta_i^r)$ to server
    Server:                                   // Global adjustment method: GA [1]
    $a^r = GA(a^{r-1}, \{G_{\hat{D}_i}(\theta^r)\}_{i=1}^M, d^r)$
    $\theta^{r+1} = \sum_{i=1}^M a_i^r . \theta_i^r$
    Broadcast $\theta^{r+1}$ to all clients $\theta_i^{r+1} = \theta^{r+1}$
```

IV. Implementation, Simulation Results, and Analysis

A. Dataset and Experimental Setup

We train and evaluate all methods on the non-iid PACS [3] (9991 images, four domains) dataset. PACS dataset contains images of four domains: Photos (P), Art painting (A), Cartoon (C), and Sketch (S) [3]. There are in total 7 types of objects in this classification task, i.e., dog, elephant, giraffe, guitar, horse, house, and person. We also implemented a dataloader to properly load VLCS datasets. However, we could not train and evaluate our method on this dataset due to the lack of enough time. VLCS [11] contains images from four image classification datasets: PASCAL VOC2007 (V), LabelMe (L), Caltech-101 (C), and SUN09 (S), denoted as domains D_V , D_L, D_C, and D_S, respectively [12]. There are five common categories in the VLCS dataset: bird, car, chair, dog, and person.

Similar to FedDG [1], we carry out leave-one-domain-out evaluations for all experiments, which means, by turn, we select one domain as the unseen client and all the left domains are used as the source clients for training. The split of train and validation set within each source domain is kept the same as that in [1] for PACS and the whole target domain is used for testing.

We use PyTorch [13] to implement all methods. To have a fair comparison, we will explain the experiment setup used for all methods in each following subsection. The code of all experiments is available at https://github.com/AHHHZ975/FedDG-Extension

B. Results on Re-implementation of FedDG [1]

Similar to FedDG [1], we use the SGD optimizer with a learning rate of 0.001, weight decay of 0.00005, and momentum of 0.9. The batch size is set to 16. The number of local epochs is set to 5, the number of communication rounds is set to 40, and the number of clients is set to 3. We use the ImageNet pre-trained ResNet18 as both local and global models in all experiments. The step size (d) of the General Adjustment (GA) method is set to 0.05 by default.

We select several representative methods in the area of FL for comparison. The baseline method is FedAvg [4], which acts as a strong baseline under the FedDG paradigm. Also, we select FedProx [5] and SCAFFOLD [6] which are initially proposed to solve the heterogeneous issue across clients. Then, we implement, train, and evaluate these methods and their combinations with the General Adjustment method proposed in FedDG [1] on the PACS dataset [3]. We report the results in Table I. According to the comparison with the original results reported in FedDG [1], the General Adjustment achieves improvements when combined with different algorithms. These results demonstrate the superiority and generalizability of our GA method, compared to the conventional FedAvg that might overfit in the local training phase.

Note that there is a slight difference between the results of our implementation and the paper results. We believe that this is most probably because the authors of FedDG mention that "*All the reported results are averaged over three runs*" [1] while we have reported results over only one run due to the lack of enough time.

C. Results on Re-implementation of MOON [2] on the CIFAR-10 dataset

To ensure that we use the correct way of implementing MOON in our MOON+GA framework, we conduct an experiment on the CIFAR-10 dataset and try to reproduce the results of the MOON [2]. To have a fair comparison, we use the same setting as the one presented in the paper MOON [2]. Therefore, we set the temperature parameter (τ) to 0.5, the learning rate (η) to 0.01 the contrastive hyper-parameter (μ) to 5, local epochs (E) to 10, communication rounds (R) to 100. Also, the original paper tries a different neural architecture for CIFAR-10 than CIFAR-100 and Tiny-Imagenet. Therefore, as we implement the MOON for CIFAR-10, we also use a CNN network as the base encoder similar to MOON [2], which has two 5x5 convolution layers followed by 2x2 max pooling (the first with 6 channels and the second with 16 channels) and two fully connected layers with ReLU activation (the first with 120 units and the second with 84 units). We use also a two-layer MLP as the projection head similar to [2] proposed by [10]. The output dimension of the projection head is set to 256 by default. The code of this architecture is shown in Fig. 4. The comparison results are presented in Table II.

TABLE I
A QUANTITATIVE COMPARISON OF ACCURACY BETWEEN OUR RE-IMPLEMENTATION OF FEDDG [1] AND THE RESULTS PRESENTED IN FEDDG [1] ON THE PACS DATASET

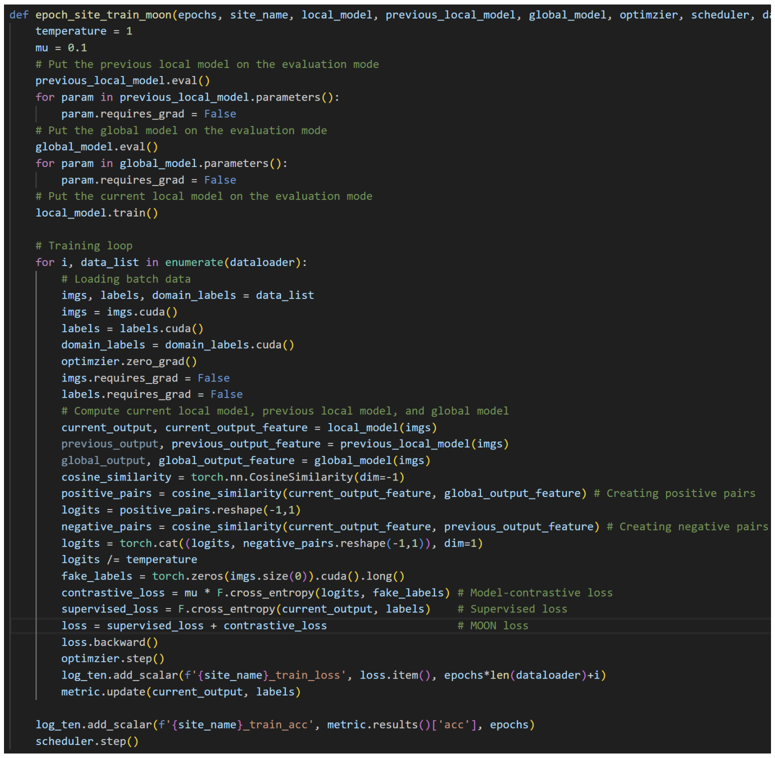
Method	Our Re-implementation					Results in the paper FedDG [1]				
	P	A	C	S	Avg.	P	A	C	S	Avg.
FedAvg	93.35	78.91	77.05	83.20	83.13	92.77	77.29	77.97	81.03	82.26
FedAvg+GA	94.43	80.96	78.37	81.45	**83.80**	93.97	81.28	76.73	82.57	**83.64**
FedProx	92.87	79.39	79.01	78.98	**82.56**	93.15	77.72	77.73	80.77	82.34
FedProx+GA	94.43	80.37	75.55	79.84	82.55	94.91	80.24	77.20	81.48	**83.46**
SCAFFOLD	91.80	77.29	78.80	81.09	82.25	92.50	78.09	77.23	80.67	82.12
SCAFFOLD+GA	93.71	81.01	75.26	80.02	**82.50**	94.79	80.14	76.91	82.12	**83.49**

TABLE II
A QUANTITATIVE COMPARISON OF ACCURACY BETWEEN OUR RE-IMPLEMENTATION OF MOON [2] AND THE RESULTS PRESENTED IN MOON [2] ON THE CIFAR-10 DATASET

Our Re-implementation	MOON Paper Results [2]
68.2%	69.1%

D. Combining local adjustment (MOON) and global adjustment (GA)

For this experimental setup, to have a fair comparison, we implement all methods including FedAvg [4], FedProx [5], SCAFFOLD [6], and MOON [2], with the same "SimpleCNN" network architecture proposed in MOON [2]. This choice also makes the training procedure much faster than when we train the model with the Resnet-18 model which was previously used for the experiment of re-implementing FedDG [1]. The network has three components: a base encoder, a projection head, and an output (classifier) layer. The base encoder is a simple CNN, illustrated in Fig. 4, used to extract representation vectors from inputs. The encoder has two 5x5 convolution layers followed by 2x2 max pooling (the first with 6 channels and the second with 16 channels) and two fully connected layers with ReLU activation (the first with 120 units and the second with 84 units). Like MOON [2], an additional projection head is introduced to map the representation to a space with a fixed dimension. When performing these experiments with Resnet-18 or SimpleCNN, we found out that removing this projection layer drastically reduces the performance which confirms the finding in the original contrastive learning paper [10]. Last, similar to the previous experimental setup, the output layer is used to produce predicted values for each class. The implementation code is presented in Fig. 3. Note that all baselines use the same network architecture as MOON (including the projection head) for a fair comparison. Also, note that, as SimpleCNN is designed to work on images in CIFAR-10 in the original paper [2], we had to perform preprocessing steps to make the images in the PACS dataset similar to the ones in the CIFAR-10. Among these steps, the most important one is that we resize the images from 224*244 to 32*32 to be able to feed the images to the network. The results are presented in Table III. According to the results, the proposed method MOON+GA and MOON itself outperforms other baselines on the PACS dataset. Here, we also want to compare the results of Table III and Table I. The main reason


```
def epoch_site_train_moon(epochs, site_name, local_model, previous_local_model, global_model, optimzier, scheduler, da
    temperature = 1
    mu = 0.1
    # Put the previous local model on the evaluation mode
    previous_local_model.eval()
    for param in previous_local_model.parameters():
        param.requires_grad = False
    # Put the global model on the evaluation mode
    global_model.eval()
    for param in global_model.parameters():
        param.requires_grad = False
    # Put the current local model on the evaluation mode
    local_model.train()

    # Training loop
    for i, data_list in enumerate(dataloader):
        # Loading batch data
        imgs, labels, domain_labels = data_list
        imgs = imgs.cuda()
        labels = labels.cuda()
        domain_labels = domain_labels.cuda()
        optimzier.zero_grad()
        imgs.requires_grad = False
        labels.requires_grad = False
        # Compute current local model, previous local model, and global model
        current_output, current_output_feature = local_model(imgs)
        previous_output, previous_output_feature = previous_local_model(imgs)
        global_output, global_output_feature = global_model(imgs)
        cosine_similarity = torch.nn.CosineSimilarity(dim=-1)
        positive_pairs = cosine_similarity(current_output_feature, global_output_feature) # Creating positive pairs
        logits = positive_pairs.reshape(-1,1)
        negative_pairs = cosine_similarity(current_output_feature, previous_output_feature) # Creating negative pairs
        logits = torch.cat((logits, negative_pairs.reshape(-1,1)), dim=1)
        logits /= temperature
        fake_labels = torch.zeros(imgs.size(0)).cuda().long()
        contrastive_loss = mu * F.cross_entropy(logits, fake_labels) # Model-contrastive loss
        supervised_loss = F.cross_entropy(current_output, labels)    # Supervised loss
        loss = supervised_loss + contrastive_loss                    # MOON loss
        loss.backward()
        optimzier.step()
        log_ten.add_scalar(f'{site_name}_train_loss', loss.item(), epochs*len(dataloader)+i)
        metric.update(current_output, labels)

    log_ten.add_scalar(f'{site_name}_train_acc', metric.results()['acc'], epochs)
    scheduler.step()
```

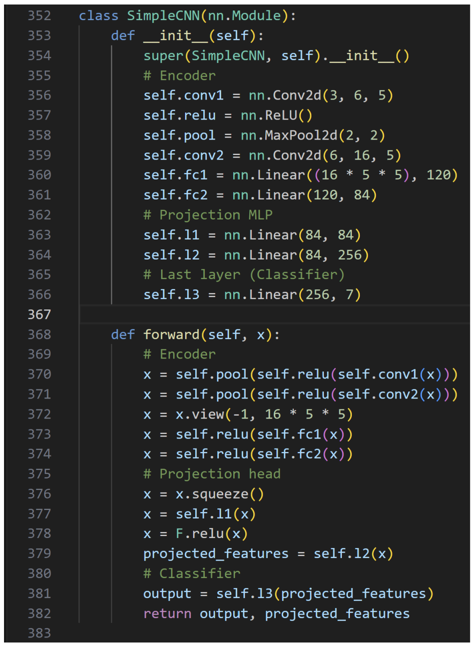
Fig. 3. The implementation code for training the MOON in our MOON+GA framework

behind this significant performance between the methods in these two tables is because the Resnet-18 is used as the backbone network for the experiment of Table I while the SimpleCNN is used as the backbone in other experiment which is a far lighter and simpler architecture than the Resnet-18. Furthermore, the 224*224 images of the PACS dataset are used during the training in the experiment of Table I while the images are scaled to the size of 32*32 for the experiment of Table III to be able to be properly fed to the SimpleCNN.

In all experiments, similar to MOON [2], we tune μ from $0.1, 5$ and τ from $0.1, 0.5, 1$ and report the best result. Table IV shows the results of MOON [2] on these different settings.

V. CONCLUSION

Inspired by the existing two directions, global adjustment methods, and local adjustment methods, in the literature for the problem of federated domain generalization, we hypothesize



```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        # Encoder
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear((16 * 5 * 5), 120)
        self.fc2 = nn.Linear(120, 84)
        # Projection MLP
        self.l1 = nn.Linear(84, 84)
        self.l2 = nn.Linear(84, 256)
        # Last layer (Classifier)
        self.l3 = nn.Linear(256, 7)

    def forward(self, x):
        # Encoder
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        # Projection head
        x = x.squeeze()
        x = self.l1(x)
        x = F.relu(x)
        projected_features = self.l2(x)
        # Classifier
        output = self.l3(projected_features)
        return output, projected_features
```

Fig. 4. The SimpleCNN architecture used in experiments with MOON [2]

TABLE III
A QUANTITATIVE COMPARISON OF ACCURACY BETWEEN THE PROPOSED METHOD (MOON+GA) AND OTHER BASELINES ON THE PACS DATASET

Method	**Our Re-implementation**				
	P	A	C	S	Avg.
FedAvg	43.63	28.60	36.68	38.82	36.93
FedAvg+GA	40.74	29.80	38.21	38.13	36.72
FedProx	38.52	28.09	36.96	36.84	35.10
FedProx+GA	39.48	30.50	36.52	38.04	36.14
SCAFFOLD	40.62	27.78	37.27	40.18	36.46
SCAFFOLD+GA	41.40	28.40	36.52	38.54	36.22
MOON	42.77	29.12	37.53	40.01	**37.36**
MOON+GA	43.21	27.85	36.81	40.12	**37.00**

that combining global and local adjustment methods could be beneficial to obtain more generalized results in the centralized federated setting. To this end, we systematically approached the problem and followed the following steps. First, we reviewed global adjustment methods in the literature and implemented one of them which is Generalization Adjustment [1]. Second, we reviewed local adjustment methods in the literature and implemented one of them which is MOON [2]. Last but not least, as our main contribution, we combined MOON [2] and GA [1] techniques in a unified pipeline and trained it on a federated domain generalization dataset (PACS [3]) and compared the results with baselines including FedAvg [4], FedProx [5], and SCAFFOLD [6]. We showed that when combined with GA, the MOON approach could give better performance in terms of generalization.

TABLE IV
A QUANTITATIVE COMPARISON OF TEST ACCURACY OF MOON [2] ON THE PACS DATASET USING DIFFERENT VALUE OF μ AND τ (SAME AS THE MOON [2] PAPER) (WITH PROJECTION HEAD IN THE ARCHITECTURE)

configuration	**Existing domains in the PACS dataset**				
	P	A	C	S	Avg.
$\tau = 0.5, \mu = 0.1$	42.77	29.12	37.53	40.01	**37.36**
$\tau = 0.1, \mu = 0.1$	39.62	29.15	36.14	38.30	35.80
$\tau = 1, \mu = 0.1$	41.28	28.78	36.50	39.04	36.4

VI. Group Members Contributions

Amir defined and formulated the problem. He maintained and implemented a base code for training, validation, and test processes of the MOON+GA framework based on the code of [1]. He re-implemented, trained, and evaluated FedAvg [4], FedProx [5], and SCAFFOLD [6] and their combinations with General Adjustment which are FedAvg+GA, FedProx+GA, and SCAFFOLD+GA all on the PACS dataset [3]. He also implemented the MOON [2] and its combination with the General Adjustment method which is MOON+GA and trained and evaluated on the PACS dataset [3].

Anam re-implemented and trained the MOON [2] method on the CIFAR-10 dataset and compared the results with the original results presented in [2]. She also wrote a data loader program to properly load the VLCS [11] dataset.

The writing of this report and the PowerPoint presentation are equally done by both of the group members.

References

[1] R. Zhang, Q. Xu, J. Yao, Y. Zhang, Q. Tian, and Y. Wang, "Federated domain generalization with generalization adjustment," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 3954–3963.

[2] M. . Li, Q. et.al (2021, "Model-contrastive federated learning (moon)." *arXiv.org*, 2021. [Online]. Available: https://arxiv.org/abs/2103.16257

[3] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, "Deeper, broader and artier domain generalization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5542–5550.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[5] S. A. K. Z. M. S. M. T. A. . S. V. . Tian, L., "Federated optimization in heterogeneous networks." *arXiv.org*, 2018. [Online]. Available: http://export.arxiv.org/pdf/1812.06127

[6] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International conference on machine learning*. PMLR, 2020, pp. 5132–5143.

[7] S. Li and L. Zhang, "Multi-dimensional domain generalization with low-rank structures," *arXiv preprint arXiv:2309.09555*, 2023.

[8] R. Ye, M. Xu, J. Wang, C. Xu, S. Chen, and Y. Wang, "Feddisco: Federated learning with discrepancy-aware collaboration," *arXiv preprint arXiv:2305.19229*, 2023.

[9] T. Li, S. Hu, A. Beirami, and V. Smith, "Ditto: Fair and robust federated learning through personalization," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6357–6368.

[10] M. N. Ting Chen, Simon Kornblith and G. Hinton., "A simple framework for contrastive learning of visual representations." *arXiv.org*, 2020. [Online]. Available: https://arxiv.org/abs/2002.05709

[11] C. Fang, Y. Xu, and D. N. Rockmore, "Unbiased metric learning: On the utilization of multiple datasets and web images for softening bias," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 1657–1664.

[12] A. Torralba and A. A. Efros, "Unbiased look at dataset bias," in *CVPR 2011*. IEEE, 2011, pp. 1521–1528.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.