THE IMPACT OF MODERN TRANSFORMER ARCHITECTURES ON FEDERATED LEARNING FOR REMOTE SENSING

Kenneth Weitzel [1], *Sebastian Völkers*[1], *Felix Zailskas*[1]

[1]Faculty of Electrical Engineering and Computer Science, Technische Universität Berlin, Germany

ABSTRACT

Remote sensing image data can be distributed across private decentralized data archives due to privacy concerns or legal regulations. To train a deep learning model on these archives, federated learning may be employed. The nature of these decentralized data archives, however, can greatly influence the convergence of the model training as the data might be not independent and identically distributed (non-IID). Recent work has shown that transformer architectures display some robustness against data heterogeneity. In this work, we investigated how three state-of-the-art transformer architectures, namely the MLP-Mixer, ConvMixer, and PoolFormer, fare against a ResNet-50 architecture, when used in a federated learning setup. The architectures were compared in terms of F1-score, local training complexity, and aggregation complexity in a multi-label classification task on the BigEarthNet-S2 benchmark archive. The dataset was artificially split to achieve different levels of non-IID distributions. Furthermore, the two federated learning aggregation techniques FedAvg and MOON were compared. In the concluding paragraph of this paper, we present some guidelines on architecture choice for federated learning in a multi-label classification task on remote sensing image data. The code used in this work is publicly available at https://git.tu-berlin.de/rsim/cv4rs-2023-winter/impact-of-modern-architectures-on-federated-learning

Index Terms— Federated learning, image classification, transformers, remote sensing.

1. INTRODUCTION

In remote sensing (RS) data is often distributed among different databases (i.e. clients) and they may not be shared. The public access at some clients might be limited due to commercial interests as the data providers consider their datasets a very valuable asset. Privacy concerns can also limit the availability, for example, farmers may be reluctant to share crop labels needed for crop mapping purposes [1]. Furthermore, legal regulations considering national security or privacy may prevent public accessibility of datasets [2]. The problem now lies in training a deep learning (DL) model without having access to all the training data, as full access is needed to learn optimal model parameters. In such a scenario the concept of federated learning (FL) can be employed to solve the aforementioned issue.

FL allows the training of a DL model without any direct access to the data of the clients and to find optimal model parameters of a global model. FL works by first sending the current global model, stored on the central server, to all the clients. Then the clients train the received model using their private data after which they send the local model parameter updates back to the central server. After the central server receives the parameter updates of every client they are aggregated according to a specific aggregation method and used to update the global model (Figure 1).

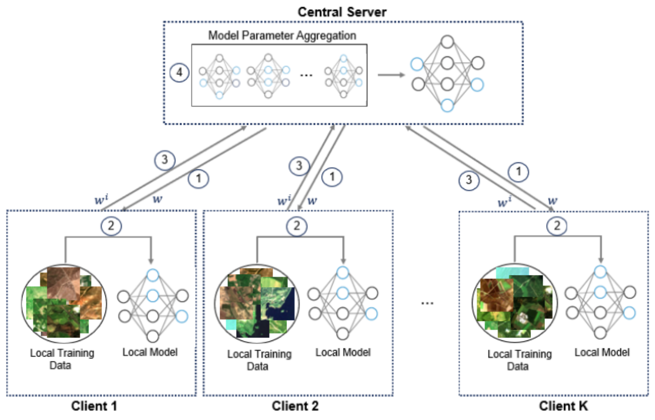

Fig. 1: An illustration of iterative model averaging from decentralized clients. From [3]

It should be highlighted that the training data across the clients might not be independent and identically distributed (non-IID). Reasons for that can be distribution skew (i.e. different distributions of labels), quantity skew (i.e. differing amounts of training data at each client), or concept drift (i.e. different data distributions of the same class at each client). All of these factors can reduce the performance of FL by leading to diverging local weights which in turn hinder the convergence of the global model [4].

While there exist other FL-related studies in the domain of RS, they mostly tackle different issues. In [5], an FL framework is presented to acquire deep learning model parameters from decentralized and unshared multi-modal RS training data, with a focus on RS image classification challenges. Another study includes [6], which presents a local differential privacy federated learning algorithm, aiming to train deep neural network models collaboratively while ensuring privacy through local differential privacy measures. The majority of proposed methods targeting training data heterogeneity aim to improve either standard empirical risk minimization of local training or standard parameter aggregation of model averaging [3].

As Transformer architectures showed promising results in tackling the problems of non-IID training data [7], we decided to explore the effectiveness of some state-of-the-art transformers, namely the MLP-Mixer [8], ConvMixer [9], and PoolFormer [10]. The models were trained using FL and tested on multi-label classification (MLC) for RS images, taken from the BigEarthNet-S2 benchmark archive [11]. Different levels of non-IID data distributions were simulated including distribution skew, quantity skew, and concept drift. Additionally, we compared two distinct FL algorithms (FedAvg [12] and MOON [4]) and compared how they affected the performance regarding data heterogeneity among clients. Aside from that, we explored an implementation of parallelized GPU training to decrease the training time of the local FL framework we built.

2. TRANSFORMERS FOR TACKLING DATA HETEROGENEITY IN FEDERATED LEARNING FOR RS IMAGE CLASSIFICATION

2.1. Problem Statement

Let K be the total number of clients and C^i be the ith client for $1 \leq i \leq K$. In detail, C^i holds the corresponding training set $D_i = \{(\boldsymbol{x}_z^i, \boldsymbol{y}_z^i)\}_{z=1}^{M^i}$, where M^i is the number of images, $\boldsymbol{x}_z^i$ is the zth RS image of the ith client and $\boldsymbol{y}_z^i$ is the corresponding class label. $\boldsymbol{y}_z^i = [y_{z,1}^i, ..., y_{z,P}^i] \in \{0,1\}^P$ is a multi-label vector of $\boldsymbol{x}_z^i$, stating the presence of P unique classes (i.e., $y_{z,p}^i = 1$ indicates that p-th class is present and $y_{z,p}^i = 0$ indicates that p-th class is not present in $\boldsymbol{x}_z^i$). There is at least one class label associated with each image. The DL model ϕ^i is trained using D_i for each C^i. To this end, we use binary cross-entropy (BCE) loss, which is given for a single sample as follows:

$$\begin{aligned}\mathcal{L}_{BCE}(\phi^i(\boldsymbol{x}_z^i), \boldsymbol{y}_z^i) &= \\ &-\sum_{p=1}^{P} y_{z,p}^i \log(r_{z,p}^i) + (1 - y_{z,p}^i \log(1 - r_{z,p}^i)\end{aligned} \tag{1}$$

where $\phi^i(\boldsymbol{x}_z^i) = [r_{z,1}^i, ..., r_{z,P}^i]$ is the class probability vector obtained by the DL model. In this paper, we assume that the data in clients is not shared and FL is used to learn global model parameters w over the whole training set $M = \bigcup_{i \in \{1,2,...,K\}} M^i$ while minimizing the following global objective function as follows:

$$\arg\min_{w} L(w) = \sum_{i=1}^{K} \frac{M^i}{|M|} L_i(w), \tag{2}$$

where L_i is the empirical loss of C^i. The parameter update of the algorithm is as follows:

$$w = \sum_{i=1}^{K} \alpha_i w^i, \tag{3}$$

where α_i is a hyperparameter to adjust the importance of the local parameters in each client for aggregation, w^i is the local model parameters of C^i.

2.2. Selected Federated Learning Algorithms

The first algorithm we utilized is the FedAvg algorithm [12], which was one of the first publications and is the most basic technique to aggregate the model parameters at the central server. As mentioned this algorithm just focuses on the aggregation part of FL and therefore will use the same steps 1-3 as shown in Figure 1. For the aggregation step (Figure 1 Step 4) FedAvg will simply collect the local model parameters of each client and average them with equal weights. The resulting averaged parameters are used to update the global model. The main problem with the FedAvg algorithm is that it does not directly address any of the issues caused by training data heterogeneity.

Due to this, we implemented a second algorithm called MOON [4] which aims to target the problems of non-IID training data. MOON focuses on local client training by adding a proximal term to the local objective function of each client. This proximal term is defined as:

$$l_{con} = -\frac{e^{S(z_{t+1},z)/\tau}}{e^{S(z_{t+1},z)/\tau} + e^{S(z_t,z)/\tau}} \tag{4}$$

where $S(.,.)$ measures the cosine similarity, τ is a temperature parameter and z_{t+1}, z_t, z are the representation vectors from the current local model w_{t+1}^i, the previous local model w_t^i, and the global model w, respectively. The proximal term will increase the similarity between the features of the global and local model while decreasing the similarity of features between the current and previous local model.
For the aggregation method, the central server will still just use an equally weighted average for all the model parameters. Despite that, the MOON algorithm promises a more robust performance with non-IID training data and was therefore chosen for our experiments.

2.3. Selected Transformer Architectures

2.3.1. MLP-Mixer

In [8], the MLP-Mixer is introduced as an alternative computer vision model to CNNs and the vision transformers (ViT) [13] with the primary goal of reducing computation times. Unlike many other transformer architectures, it does not use any self-attention or convolutions but rather only utilizes multi-layer perceptrons (MLPs). These MLPs are applied to the patch-embedded input image. Similar to the ViT, the MLP-Mixer employs the concept of channel mixing and token mixing to enable communication across different channels as well as spatial locations, both of which are implemented with separate MLPs. The authors of [8] observed improvements through the absence of any self-attention or convolution in their experiments. The MLP-Mixer performed on par with the ViT and other CNN based architectures, while having a much shorter computation time. The MLP-Mixer is therefore considered to have a good computation-accuracy trade-off.

2.3.2. ConvMixer

The ConvMixer architecture [9] builds upon the MLP-Mixer by including channel and token mixing mechanisms to process channel and spatial features. In contrast to the latter, the ConvMixer replaces all MLP operations from the MLP-Mixer with convolutional layers. The goal of using convolutional layers is to leverage high data efficiency in vision tasks. The image is initially embedded in a patch-wise manner via a convolutional layer. Then, the patch-embedded input is passed to a depth-wise convolutional layer, which performs grouped convolution to achieve token or spatial mixing. The following point-wise convolutional layer performs 1x1 convolution to achieve channel mixing. The ConvMixer outperformed standard computer vision models like the ResNet-152 and the vision transformer DeiT-B on ImageNet1K [9] while having fewer parameters, which makes them a considerable architecture for FL.

2.3.3. PoolFormer

In [10], it is proposed that the success of the transformers might not be due to a complex token mixer but rather the general form of a transformer block. The PoolFormer block is introduced by replacing the attention-based token mixer module with the simple average pooling operation to reduce the computational complexity and parameter amount. The full PoolFormer architecture consists of four stages each using an initial patch embedding and then a series of PoolFormer blocks. The paper suggests to use a distribution of $\frac{L}{6}$, $\frac{L}{6}$, $\frac{L}{2}$, and $\frac{L}{6}$ of these PoolFormer blocks across the four stages, where L is the total amount of PoolFormer blocks used. The authors of [10] showed that the PoolFormer architecture consistently outperformed multiple other transformers and CNNs for computer vision tasks on the ImageNet1K dataset.

3. EXPERIMENTAL SETUP

3.1. Dataset & Training

For the experiments in this project, we used the BigEarthNet-S2 benchmark archive [11]. Specifically, we used a subset that includes the Sentinel-2 images of the seven countries Austria, Belgium, Finland, Ireland, Lithuania, Serbia and Switzerland. Each of the images is labelled with multi-labels from a total of 19 labels. Of the available 13 bands for Sentinel-2 images, we used 10 bands, namely the bands with 10m and 20m spatial resolution. The 20m bands were up-sampled so that every image had a pixel resolution of 120x120. We used the train and test splits proposed in [11]. To simulate different non-IID levels we created three divisions of the training data set:

- *Decentralization Scenario 1 (DS1)*: The training images were randomly distributed to different clients.
- *Decentralization Scenario 2 (DS2)*: The training images were distributed in a way that each client holds the images associated with only one country.
- *Decentralization Scenario 3 (DS3)*: The training images were distributed in a way that each client holds the images associated with only one country in only one season.

The non-IID level is lowest for DS1 then next lowest for DS2 and highest for DS3. In our experiments we compared the performance of the three presented transformer architectures MLP-Mixer [8], ConvMixer [9] and PoolFormer [10] as well as the ResNet-50, which serves as a baseline of common CNN architectures. DS1 and DS2 were used for the main experiments, while DS3 was only used for the sensitivity analysis of the three transformer architectures. We did not include DS3 in our main results as we did not want to give an advantage to the models tuned to the third scenario. Furthermore, we tested the FedAvg [12] and MOON [4] algorithms and their impact on performance when used in DS1 and DS2 for each of the four architectures.

All training was done using the Adam optimizer with a learning rate of 0.001, no weight decay, and a mini-batch size of 128. In the sensitivity analysis, we performed training of 20 communication rounds and three epochs, while the full training runs used 30 communication rounds and three epochs. Performance was measured in terms of weighted F1-score, the time a single communication round took, and the amount of parameters each model uses. All experiments were performed on NVIDIA Tesla V100-SXM2 GPUs with 32GB of memory.

3.2. GPU Parallelization

As the training of each local client is inherently independent we tried to parallelize their training. The parallelization logic is triggered when the global model starts a communication round. First, the main process, in which the training is performed, creates a multiprocessing queue which is persistent across multiple processes. The queue is then filled with the data necessary for training each of the clients. This data includes the current model state as well as the data connected to the specific client. Additionally, the main process adds one stop signal per available GPU into the queue. Then the main process creates one worker process per GPU and starts them. These worker processes have access to the training queue. The worker processes now iteratively pull from the training queue. If they find data for training a client model, then the GPU will start training and store the results in a local list. Once the GPU has successfully finished the training of one client it will pull from the model queue again. Hence, the GPUs will be utilized as long as there are still clients to train. Once there are no more clients to train the worker processes will pull a stop signal from the training queue. This signal indicates that the training is done and the worker process will return the results list to the main process. Once the main process has received all result lists from the worker processes it will merge them and can continue the aggregation as if the results were computed sequentially. An illustration of how the parallel training is performed is shown in Figure 2 and Figure 3.

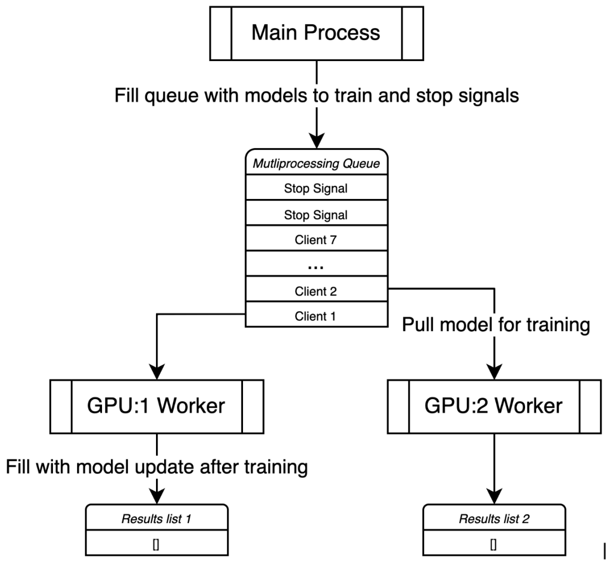

Fig. 2: GPU worker flow with full training queue.

Unfortunately, we did not observe a training time improvement when using this approach. The individual training times of the GPUs slowed down when training in parallel and using two GPUs at the same time resulted in roughly the same total training time for the ResNet-50 as when using only one GPU. We believe that this issue occurred due to the GPU processes accessing the same disk space where the dataset was stored during training. This likely resulted in a significant I/O slowdown during parallel training. Due to this issue, we did not apply the parallel training method for our experiments, as we did not have the resources to provide each GPU with a dedicated copy of the dataset on the hard disk.

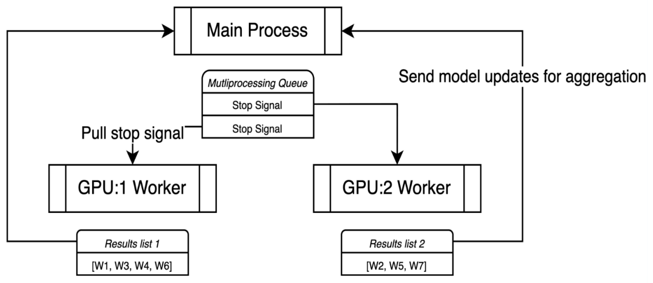

Fig. 3: GPU worker flow with empty training queue.

4. EXPERIMENTAL RESULTS

4.1. Sensitivity Analysis

All three transformer architectures are passed through a sensitivity analysis to find suitable model configurations for the federated learning setup. To increase comparability, all selected configurations should match in parameter size as much as possible. For each transformer architecture, 4-5 configurations are selected and trained according to the training setup presented in subsection 3.1.

Patch Size	#Blocks	Hidden Dimension	F1-Score
16	8	512	0.617
16	12	512	0.615
16	8	768	0.608
16	2	1024	0.621
8	8	512	**0.635**

Table 1: MLP-Mixer configurations in the sensitivity analysis

For the MLP-Mixer the three parameters patch size, number of MLP-Mixer blocks and the hidden dimension are iterated as seen in Table 1. The choice of parameter variations is based on the smallest model presented in the paper [8] with the configuration 16/8/512. All other configurations are selected intuitively by keeping one parameter fixed while changing others. Picking high values for #Blocks and hidden dimension was not considered, as this would increase the parameter count of the model substantially. Our analysis showed that the MLP-Mixer with patch size and #Blocks of 8 and hidden dimension of 512 performed best, with an F1-score of 0.635.

In Figure 4 we can see the training process of the five models during the analysis. We observed a smooth training curve with no unexpected deviations.

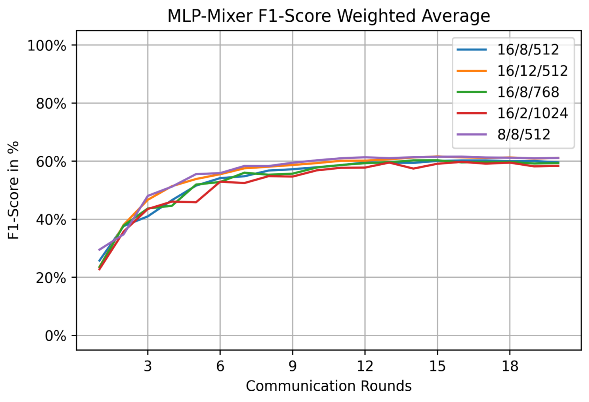

Fig. 4: Weighted average F-1 score of tested MLP-Mixer configurations along communication rounds.

During initial testing of the ConvMixer, a high discrepancy was observed between models that were pretrained on ImageNet1K and non-pretrained models. To explore this behaviour only ConvMixer configurations for which pretrained weights were available were tested. The comparison with their non-pretrained counterparts can be seen in Table 2 and Figure 5.

Hidden Dimension	Depth	Pretraining	F1-Score
768	32		0.353
768	32	✓	**0.544**
1024	20		0.452
1024	20	✓	0.500

Table 2: ConvMixer configurations in the sensitivity analysis

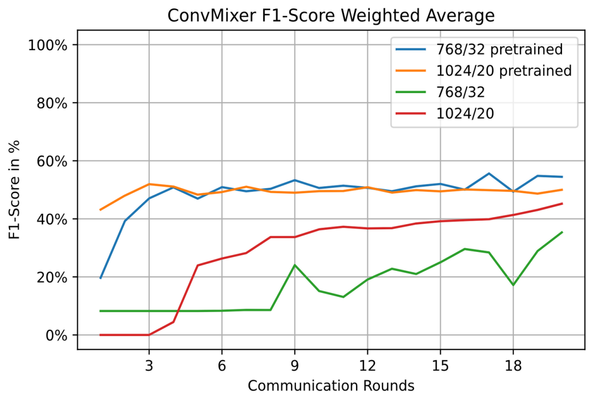

Fig. 5: Weighted average F-1 score of tested ConvMixer configurations along communication rounds.

The varied parameters are the hidden dimension of the patch embedding and the model depth, similar to the test setup in [9]. We can see this large difference in both the final scores of the models as well as the training curves. While the pretrained models show a fast convergence overall, the non-pretrained models seem to not improve at all for the first few communication rounds and display issues converging after that. The best-performing model was the pretrained model with hidden dimensions of 768 and a depth of 32, with an F1-score of 0.544.

For the PoolFormer different distributions of PoolFormer blocks across the four stages were tested. Increasing the number of total blocks would drastically increase the model size, thus a total amount of 12 blocks was considered. The selected configurations were chosen, so that the focus of the blocks lies on the earlier or later stages of the architecture. Configurations include one of the suggested configurations from the paper (i.e. (2,2,6,2)) [10], equally distributed blocks, a distribution that emphasizes the earlier stages, and one that emphasizes the later stages of the architecture.

#Poolformer Blocks				
Stage 1	Stage 2	Stage 3	Stage 4	F1-Score
2	2	6	2	**0.414**
3	3	3	3	0.406
2	3	5	3	0.404
3	5	3	2	0.412

Table 3: Poolformer configurations in the sensitivity analysis

The results can be seen in Table 3 and Figure 6, in which we noticed that after an initial drop in performance after the first communication round all models converged smoothly and performed very similarly. We chose the best-performing model with a block distribution of (2,2,6,2), with an F1-score of 0.414.

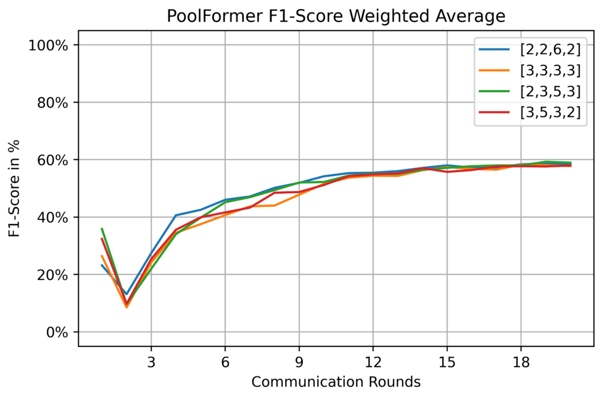

Fig. 6: Weighted average F-1 score of tested PoolFormer configurations along communication rounds.

4.2. Results

Table 4 shows the weighted average F1-scores (%) under both DS1 and DS2. We can see that the F1-scores under DS1 are higher than those under DS2. This indicates that the lower level of training data heterogeneity of DS1 is beneficial for model performance. Using the FedAvg algorithm and DS1, the highest accuracy was obtained by the MLP-Mixer (75.81%) closely followed by the ResNet-50 (75.24%) and then by the PoolFormer (74.09%). The worst-performing architecture in this setup was the ConvMixer (72.47%). In contrast to the results obtained under DS1, the worst-performing architecture under DS2 was the ResNet-50 (47.32%). The three transformer architectures significantly outperformed the ResNet-50 where the MLP-Mixer (62.77%) again performed best, followed by the PoolFormer (59.85%) and lastly the ConvMixer (53.66%). This indicates that the transformer architectures can handle the challenges of training data heterogeneity more effectively and produce a better-performing global model than the ResNet-50.

Using the MOON algorithm we can see similar F1-scores under DS1, while the MLP-Mixer (75.12%) performed best, followed by the PoolFormer (74.47%), then the ConvMixer (72.82%) and lastly the ResNet-50 (72.82%). Interestingly, the performance of the transformer architectures stayed relatively constant under DS1 when changing to MOON (fluctuations of $< 1\%$), while the ResNet-50's performance dropped by almost 2.5%. Under DS2 using the MOON algorithm we see that the PoolFormer (60.26%) performed best, closely followed by the MLP-Mixer (59.48%), then the ConvMixer (58.76%) and once more the ResNet-50 (54.83%) performed worst on the data with higher non-IID levels. However, it must be noted that the discrepancy between the transformers and the ResNet-50 is not as significant under DS2 when using the MOON algorithm, as this algorithm helps to alleviate the effects of training data heterogeneity. The transformers did not benefit from this property of the MOON algorithm as much as the ResNet-50 as they are inherently more robust against data heterogeneity.

We also assessed the effectiveness of the architectures in terms of local training complexity and aggregation complexity. Here local training complexity refers to the computational costs required to train a DL model per client [14], while aggregation complexity refers to the cost of aggregating model parameters at the central server [3]. Table 5 shows the computation times (in seconds) to complete a single communication round and the number of parameters of each model. We observe that the computation time for the MOON algorithm is higher than that of the FedAvg algorithm for all architectures. This is because the MOON algorithm requires extra forward passes to extract image feature vectors. Furthermore, we found that for both algorithms the computation time for the ResNet-50 was the lowest, followed by that of the PoolFormer, then the MLP-Mixer and finally the ConvMixer had the highest computation time. According to our results, the PoolFormer was the most efficient architecture among all the selected architectures in terms of aggregation complexity with 11.24M parameters. The MLP-Mixer and ConvMixer had similar aggregation complexity with 20.65M and 20.62M parameters respectively. Finally, the ResNet-50 had the highest aggregation complexity with a parameter count of 23.60M.

Algorithms	Architectures	DS1	DS2
FedAvg [12]	ResNet-50	75.24	47.32
	MLP-Mixer [8]	**75.81**	**62.77**
	ConvMixer [9]	72.47	53.66
	PoolFormer [10]	74.09	59.85
MOON [4]	ResNet-50	72.82	54.83
	MLP-Mixer [8]	**75.12**	59.48
	ConvMixer [9]	73.28	58.76
	PoolFormer [10]	74.47	**60.26**

Table 4: F1-scores (%) obtained by FedAvg and MOON algorithms with different architectures under two different decentralization scenarios.

Algorithms	Architectures	Computation Time	Number of Shared Parameters
FedAvg [12]	ResNet-50	**767**	23.60M
	MLP-Mixer [8]	1447	20.65M
	ConvMixer [9]	2432	20.62M
	PoolFormer [10]	1231	**11.24M**
MOON [4]	ResNet-50	**972**	23.60M
	MLP-Mixer [8]	1738	20.65M
	ConvMixer [9]	2683	20.62M
	PoolFormer [10]	1447	**11.24M**

Table 5: Computation times (in seconds) and number of parameters shared by each client with different architectures obtained by FedAvg and MOON algorithms.

5. CONCLUSION

This paper analyzes and compares different transformer-based architectures (MLP-Mixer, ConvMixer, PoolFormer) under different non-IID levels in the framework of MLC problems in RS. The selected architectures have been compared in terms of their: 1) accuracy; 2) local training complexity; and 3) aggregation complexity (see Table 6). Through an experimental comparison of these architectures, we have derived a guideline for selecting appropriate architectures in the context of FL for RS MLC problems as follows:

1. The architectures achieve similar accuracies when the

Architectures	Accuracy	Local Training Complexity	Aggregation Complexity
ResNet-50	L	L	H
MLP-Mixer [8]	H	M	H
ConvMixer [9]	M	H	H
PoolFormer [10]	H	M	L

Table 6: Comparison of the selected architectures. Three marks ”H” (High), ”M” (Medium), or “L” (Low) are given for the considered criteria.

level of training data heterogeneity is low. ResNet-50 has the lowest local training complexity. Therefore, it can be selected if the training data has low non-IID levels, although the aggregation complexity of ResNet-50 is the highest compared to the transformer architectures.

2. The accuracy of ResNet-50 drops significantly when the level of training data heterogeneity increases. MLP-Mixer and PoolFormer achieve the highest accuracies among the selected architectures when the level of training data heterogeneity is high. Since the aggregation complexity of PoolFormer is lower than that of MLP-Mixer, it can be selected if the training data is highly non-IID.

3. MOON outperforms FedAvg by a large margin when the level of training data heterogeneity is high with ResNet-50. However, the accuracy obtained with MOON is very close to that of FedAvg when the transformer architectures are used. Therefore, the FedAvg algorithm can be used with transformers instead of complex algorithms that alleviate the effects of training data heterogeneity, to achieve lower training times.

As a final remark, we would like to note that, although MLC has been selected as the classification task to assess the different transformer-based architectures for FL in this paper, our study can be extended to other RS tasks such as land-cover map generation or image retrieval systems. Future research can also focus on expanding on our results by testing more FL algorithms and other model architectures.

6. ACKNOWLEDGEMENTS

We would like to thank our project supervisor Barış Büyüktaş for his help and guidance throughout this project. We would also like to thank professor Begüm Demir for her valuable feedback and the opportunity to conduct research for the RSiM department at the TU Berlin.

7. REFERENCES

[1] D. Tuia, K. Schindler, B. Demir, G. Camps-Valls, X. X. Zhu, M. Kochupillai, S. Džeroski, J. N. van Rijn, H. H. Hoos, F. Del Frate *et al.*, “Artificial intelligence to advance earth observation: a perspective,” *arXiv preprint arXiv:2305.08413*, 2023.

[2] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar, “Semi-supervised knowledge transfer for deep learning from private training data,” *International Conference on Learning Representations*, 2017.

[3] B. Büyüktaş, G. Sumbul, and B. Demir, “Federated learning across decentralized and unshared archives for remote sensing image classification,” *arXiv preprint arXiv:2311.06141*, 2023.

[4] Q. Li, B. He, and D. Song, “Model-contrastive federated learning,” *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10 713–10 722, 2021.

[5] B. Büyüktaş, G. Sumbul, and B. Demir, “Learning across decentralized multi-modal remote sensing archives with federated learning,” *arXiv preprint arXiv:2306.00792*, 2023.

[6] Z. Zhang, X. Ma, and J. Ma, “Local differential privacy based membership-privacy-preserving federated learning for deep-learning-driven remote sensing,” *Remote Sensing*, vol. 15, no. 20, p. 5050, 2023.

[7] M. Mendieta, T. Yang, P. Wang, M. Lee, Z. Ding, and C. Chen, “Local learning matters: Rethinking data heterogeneity in federated learning,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8397–8406.

[8] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit *et al.*, “Mlp-mixer: An all-mlp architecture for vision,” *Advances in neural information processing systems*, vol. 34, pp. 24 261–24 272, 2021.

[9] A. Trockman and J. Z. Kolter, “Patches are all you need?” *arXiv preprint arXiv:2201.09792*, 2022.

[10] W. Yu, M. Luo, P. Zhou, C. Si, Y. Zhou, X. Wang, J. Feng, and S. Yan, “Metaformer is actually what you need for vision,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 819–10 829.

[11] G. Sumbul, A. D. Wall, T. Kreuziger, F. Marcelino, H. Costa, P. Benevides, M. Caetano, B. Demir, and V. Markl, “BigEarthNet-MM: A large-scale, multimodal, multilabel benchmark archive for remote sensing image classification and retrieval,” *IEEE Geoscience and Remote Sensing Magazine*, vol. 9, no. 3, pp. 174–180, 2021.

[12] 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.

[13] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *arXiv preprint arXiv:2010.11929*, 2020.

[14] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, “Cost-effective federated learning design,” in *IEEE Conference on Computer Communications*, 2021, pp. 1–10.