The Impact of Federated Learning on Distributed Remote Sensing Archives

1st Anand Umashankar
Technische Universität Berlin
Berlin, Germany
anand.c.umashankar@campus.tu-berlin.de

2nd Karam Dawoud
Institute of Computer Science
Technische Universität Berlin
Berlin, Germany
karam.dawoud@campus.tu-berlin.de

3rd Nicolai Schneider
Institute of Software Engineering
Technische Universität Berlin
Berlin, Germany
nicolaischneider@mail.tu-berlin.de

***Abstract*—When it comes to Machine Learning in remote sensing, one of the main obstacles researchers are facing is the large scale of the datasets. Just the size of freely available earth observation proposes a challenge for personal computers. For instance, a variety of missions such as Sentinel-1, -2, and -3 gathered a collective of 5 PetaBytes [1]. Given the size of the datasets, they are stored and processed across multiple platforms (often referred to as clients) which implies that decentralized Machine Learning has to be applied. *Federated Learning* is one decentralized learning approach that was first introduced recently[2] by Google and is adopted in their Android ecosystem. Since its release, the original *Federated Learning* technique has been fine-tuned and further developed. The scope of this project is to apply multiple *Federated Learning* models on remote sensing datasets and understand their implications considering different data splits across clients.**

***Index Terms*—Machine Learning, Federated Learning**

I. Introduction

Remote sensing (RS) datasets are oftentimes too large to be trained on a centralized Machine Learning model. For this matter, the data is split into various partitions and trained separately. One exciting new approach was first introduced by Google researchers in 2017 is *Federated Learning* (FL) [2].
The idea behind FL is to send the Deep Learning model to the data instead of sending the data to the model. In the case of Google, this method is used to apply Machine Learning on Android devices. The data from each phone are not being sent to a central server. In instead, each device, oftentimes referred to as a *client*, trains a model received from a host/central server based on the client's own data. The trained models from each device are sent back to a central host and averaged.

Accessing the data from different devices is not the root of the issue in our case, however, we consider a bigger dataset and split it into a variety of partitions to apply FL. The approach might solve the issue of training big datasets, nevertheless, it also comes along with two main challenges:

- The first obstacle being the extensive communication between clients and host for model averaging which can highly drain the training process.
- The second hurdle arises through the client data distribution. Considering a remote sensing dataset with images from all over the world, there are certain classes like 'desert' which can only be found in a few regions of the world. In case that the data is distributed by country, most clients wouldn't have access to such classes (as 'desert'). This characteristic is also called non-IID (non independent and identically distributed) data partition. [3]

Over the past years, a variety of FL approaches have been developed to tackle these issues. For instance, *FedAvg* [4] decreases the client-server communication by only training a randomly chosen fraction of clients during each epoch. Another approach is *FedProx* [5] which addresses the hurdle of non-IIDness by adding a *proximal term* to consider the degree of IIDness of each client during training. The goal of this project is to apply these FL approaches using different data partitions to both understand the impact of Federated Learning on non-IIDness and how different data distributions can affect the results.

A. Goals and Challenges

Federated Learning is still a new topic both in the world of academia and the industry. When applied correctly it can solve many issues but it also proposes new challenges. We intend to implement three different Federated Learning models (*Bulk Synchronous Parallel (BSP)* [6], *Federated Averaging (FedAvg)*, and *Federated Proximal(FedProx*) on a RS dataset to understand their impact in comparison to an ordinarily used centralized approach. All implementations will be tested with the Deep Learning models *ResNet34* [7], *AlexNet* [8] and *LeNet* [9].

We evaluate if these federated learning algorithms are effective on remote sensing datasets. We intend to make comparisons among different deep learning models when using federated learning. Lastly, we would like to modify different hyperparameters and other experiment settings to evaluate the extent of the effects that these have on the outcome. The main criteria for these comparisons are the accuracy of the output models and also communication costs and runtime. **Based on these comparisons we empirically conclude the optimal federated algorithm, deep learning model, and**

hyperparameter choices that can be used for future RS applications.

Classical RS datasets tend to be very large, making the computational process much more difficult. Nevertheless, this issue goes beyond the scope of our project, therefore we chose UC Merced Landuse [10], a multilabel RS dataset containing 2100 images and 18 classes.

We expect to gain similar results to current literature and to find the optimal parameters for each FL model.

II. Background and Related Work

The first section provides information about the basic implementation of Federated Learning, the chosen FL algorithms, and the applied Deep Learning models for our experimental evaluation. We then discuss current findings and approaches in Federated Learning.

A. Federated Learning

The main idea of *Federated Learning* is to reverse the common procedure of Machine Learning: instead of sending the data to the model, the model is sent to the data. In a FL scenario we have two parties: the *host* and the *clients*. The host contains the Deep Learning model which will later be trained, while each client holds a fraction of a dataset. The main steps are depicted in Fig. 1. In step 1, the *host* initializes a Machine Learning model and sends it to each *client* (step 2). Next, each *client* trains the received model based on its data (step 3) and sends the trained model back to the *host* (step 4). The *host* then collects all models and averages them (step 5). It should be noted that the training of each client takes place in parallel.

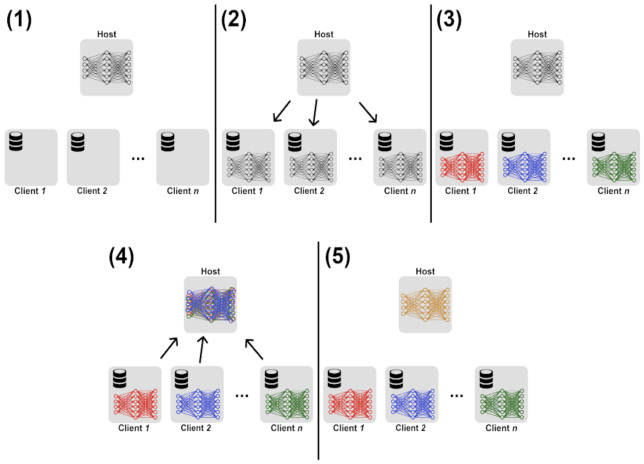


Fig. 1. Basic steps of Federated Learning

1) Federated Averaging: The basic *Federated Learning* model presents one major issue, which is the enormous communication between the *clients* and the *host* and the high computation. One of the most common Federated Learning algorithms, that tries to tackle these issues, introduced in [4], is *FedAvg*.

Let K be the set of clients, for each training round *FedAvg* only sends the model to a random fraction with a fixed size $C \subseteq K$ of clients. For instance, for the experimental evaluation of [4] only 10% of clients were trained each round. Furthermore, the communication is reduced by running multiple *local epochs* E as depicted in Fig. 2. The authors used up to 5 local epochs for their experiments. Finally, each client's local dataset can be split into batches by applying the parameter B, where $B = \infty$ specifies that the whole local dataset is used as a batch. Once all clients $k \in C$, with their respective data partition n_k, sent their trained weights ${w_k}^t$ back to the host, the new averaged model w_{avg}^t is computed with

$$w_{avg}^t = \sum_{k=1}^{|K|} \frac{n_k}{n} w_k^t \tag{1}$$

where t indicates the training round and n the length of the whole dataset.

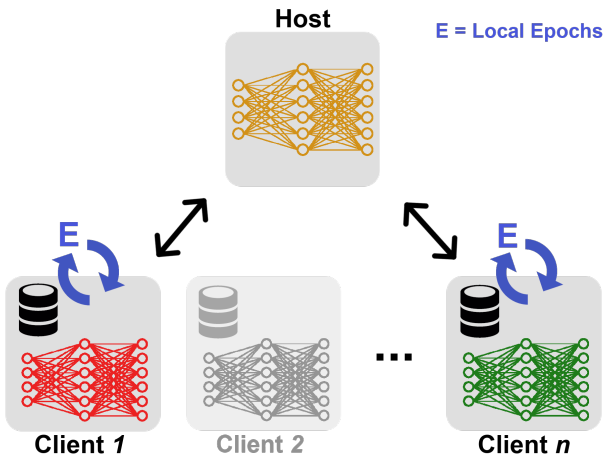


Fig. 2. Basic steps of Federated Averaging

2) FedProx: FedProx [5] is an extension to *FedAvg*, that has modifications to tackle non-identical distributions in data and also accounts for systems heterogeneity. *FedProx* provides more reliable convergence when compared to *FedAvg*. On average, a 22% accuracy improvement is shown across highly heterogeneous settings. Their work is mainly based on adding a "proximal" term to a standard local loss function. The objective is the usual loss function summed with a penalty when the local model deviates too much from the global model. This addresses the issues of data heterogeneity and allows for safely incorporating variable amounts of local work resulting from systems heterogeneity.

3) Bulk Synchronous Parallel: Bulk Synchronous Parallel (*BSP*) [6] is an older approach that misses the key FL element of averaging the models. In terms of *FedAvg* the parameters are set the following way:

- $C = 1$, therefore all clients are used in each round
- $E = 1$, such that each client runs 1 local epoch

Instead of passing the model to each client and averaging the trained models, *BSP* passes the model from one client to the

other. Once a client is done with training it sends the model to the next client. A round is complete once the model has been passed to each client. A more communication-heavy version of BSP will pass the model between clients after training on a single training batch, this communication costly approach is more robust to the non-identical distributions in data since it takes more small update steps towards convergence instead of large updates that might skew the model in one direction or the other.

B. Deep Learning Models

LeNet is one of the earlier Machine Learning approaches and was first proposed in 1990. The original architecture of *LeNet-5* consisted of two convolutional layers, two sub-sampling layers, two fully connected layers, and an output layer with Gaussian connection [9]. To adapt to the image size of 256x256 we adjusted the kernel size for all convolutional layers to 5x5.

AlexNet was first introduced by Alex Krizhevesky in 2012 and was considered a State-of-the-Art Deep Learning model for visual recognition and classification at the time. The architecture consists of a total of 8 layers: five convolutional layers, two fully connected layers with dropout and a soft-max layer.

ResNet is one of the most popular approaches in Image classification and has been published in 2015 by Kaiming He. The main architecture consists of convolutional layers with 3x3 filter and concludes with an average pooling layer and a 1000-way fully-connected layer with softmax. Additionally, ResNet stacks building block (shown in Fig 3), using the so-called shortcuts to skip the input over the next two layer which makes the CNN *residual* [7]. The shortcuts can only be used when the input and the output have the same dimensions, and they help solve vanishing gradients problem that is one of the main problem in training deeper and deeper Neural Networks.

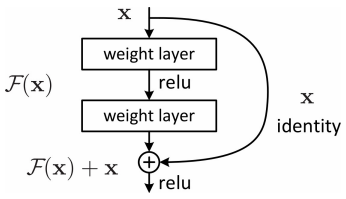


Fig. 3. Residual block used by ResNet architecture [7]

C. Related Work

[3] show that training over skewed label partitions is challenging problem to solve especially for decentralized learning, as all the algorithms in their study suffer major accuracy loss. Secondly, DNNs with batch normalization were found to be vulnerable in the Non-IID setting. They also prove that the difficulty level of this problem varies greatly with the degree of skew. They use three decentralised training algorithms which are Gaia [11], Federated Averaging and Deep Gradient Compression [12].

D. Other FL Algorithms

Gaia [11] accumulates updates to model weights and updates it to other data partitions when its relative magnitude exceeds a defined threshold which means that the insignificant communication between data centers is reduced while still retaining correctness of machine learning approaches. They observed speedup of almost 1.8x to 53.5x over leading distributed ML frameworks and is 0.94x - 1.4x when using the same ML approaches on nodes connected on a local area network.

Deep Gradient Compression [12] communicates only a pre-specified amount of gradients each training step to reduce communication costs. This is also called as gradient clipping and is done on the local nodes. They also use other approaches like momentum correction, momentum factor masking and warm up training. In their experiments they achieve a compression ratio of 270x to 600x without losing accuracy.

SCAFFOLD [13] uses variance reduction technique to correct the drift off in local clients in its local updates. SCAFFOLD requires significantly lower communication rounds when compared to *FedAvg* and also performs well irrespective of data heterogeneity or client sampling. SCAFFOLD also can take advantage of similarity in different clients' data thus resulting in even faster convergence in those cases. Their experiments prove that they are always at least as fast as normal SGD and can be much faster depending on the data similarity between clients.

FedBoost [14] provide ensemble algorithms which are made optimised to have low communication for federated learning. In their work the per-round communication cost is independent of the size of the ensemble. Unlike other previously discussed works [12] [4], their approach reduces the communication between both server-to-client and client-to-server communication.

FetchSGD [15] compresses model updates using Count Sketch. This enables the solution to take advantage of the combinability of the sketches to combine model updates from many nodes to one update. The Count Sketch is linear in nature and hence momentum and error accumulation can be performed inside the sketch. This helps to move the momentum and error accumulation from clients to the central aggregator, thus solving the problems associated with client participation and also achieving high compression rates and good convergence.

III. Methodology

A. Dataset & data augmentation

Our Dataset of choice for this experiment is UC Merced Land Use Dataset [10], but instead of using the provided single label, we opt for using the multilabel [16], because multilabel are usually more realistic and challenging for a Remote Sensing classification case study (examples shown in Fig 4).

The dataset contains 2100 images, which is a small number for training, especially when using a large number of clients.

Therefore before training, we used data augmentation to double the dataset in size to 4200. We apply one of four common corruption methods on each image once; "**Impulse noise** is a color analogue of salt-and-pepper noise and can be caused by bit errors...**Motion blur** appears when a camera is moving quickly...**Snow** is a visually obstructive form of precipitation...**Pixelation** occurs when up-sampling a low-resolution image." [17]. Furthermore during training a random horizontal flipping were also applied.

Fig. 4. Some UC Merced Land Use Dataset examples, showing both the original single label (s.l) as well as the multilabel(m.l).

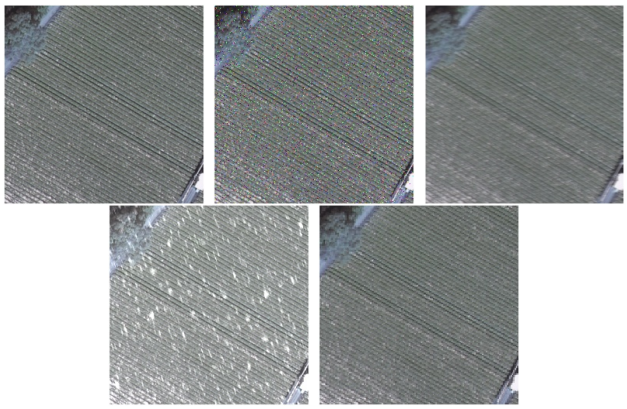


Fig. 5. Example of the different augmentation methods on the same image, in the first row from left to right: original image, impulse noise, motion blur. In the second row: snow, and pixelation.

B. Main aspects of our experiment

Similar to [3], our study focuses on the following criteria:

(i) The ML models: for this we compare the influence of FL on the validation accuracy for different neural networks. AlexNet [8], LeNet [9], and ResNet34 [7]. Training parameters were set to learning rate = 0.001 and Momentum = 0.9 for all the models.

(ii) Federated Learning algorithms: as described in II, we compare FedAvg and FedProx against each other as well as against BSP. For FedAvg, we used the following hyper-parameters: $C_{fraction} \in \{0.5, 0.75, 1\}$ (meaning in each round the model is sent to half, three-quarter, and all clients for training which effectively reduces the amount of data used for each round of training), with local epoch number on each client $E_{local} = 5$.

(iii) Degree of label skew-ness of the datasets partitions: the idea here is that each client has a monopoly of some percentage over a certain label in the dataset, where as the rest of the dataset is uniformly distributed over all the clients.

But there is a inherent problem in artificial label skewing multilabel datasets over a certain number of clients. As seen in the label distribution in Fig. 6, the dataset has 2 clear types of labels dominance, so there is two cases for skewing:

- common labels: there is only 7 labels that are present in more than 10% (6 of them are in more that 25%) of the images, which mean if we distributed the dataset over 4 clients for example there is only a certain degree of skew-ness possible before the label overlaps and the skewness loses its meaning because of the high correlation[1] between these labels. For our tests when splitting over those dominant labels we use 4 clients and skewness $\in \{0, 20, 40\%\}$. .
- less common labels: 9 labels are presence in roughly 5% and one label around 10% of the dataset, and they are highly uncorrelated which mean we can freely skew the monopoly of the clients to a higher percentage, and we can use more clients in this case. In our tests we used mainly 8 clients with skewness $\in \{40, 60, 80\%\}$. We tested also increasing the number of clients to $\{10, 25, 50\}$, with skewness of 40%, this means the first 9 clients will have the 40% monopoly over the small 9 labels and the rest of the dataset is uniformly distributed over all the clients. We can see here that for this dataset, as we increase the number of clients being used the data distribution becomes more IID in nature.

Furthermore we don't consider using a mix of common and less common labels for splitting over the clients, since it will cause an imbalanced distribution of data among client, that is a different kind of FL problem that we are not tackling in this work.

(iv) Furthermore we test the influence of the training batch size on such set up, with batch sizes $\in \{1, 4, 8, 16, 32, 64, 128, 256\}$

C. Experiments

To evaluate all aspects mentioned in III-B we divide our experiments into 8 sections. The parameters for all our experiments are noted in Table I[2]. Each training runs for 100 Rounds and $E_{local} = 5$ for *FedAvg* and *FedProx*.

[1]As shown in Fig 7, using the Cosinus similarity measurement clearly shows that the common labels co-occur in the same image much more than less common labels.

[2]In the table I, the flag called *Small Skew* refers to skewing over the less common label classes.

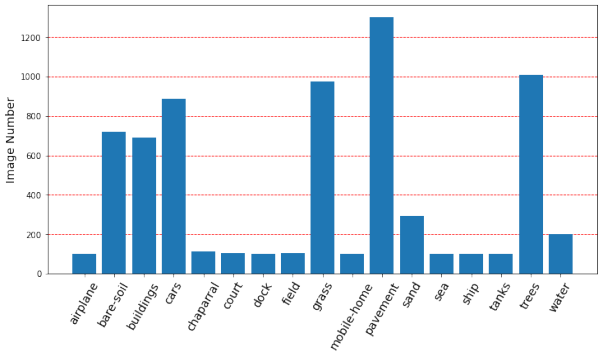


Fig. 6. UC Merced Land Use Dataset multilabel distribution: the total number of label occurrences in the 2100 images of the Dataset. We define "common labels" are labels that are present in more than 10 % (210 data point), where as "less common labels" are in less than 10 %.

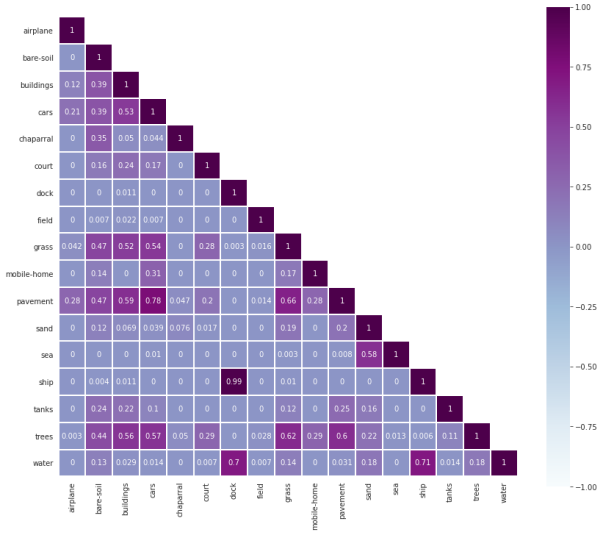


Fig. 7. UC Merced Land Use Dataset multilabel cosinus similarity matrix; shows that "less common" labels are for the most part decorrelated, where as "common label" are much more correlated. The darker purple a matrix field gets (closer to 1) the more correlation (co-occurrences) between two labels there are, where as the bluer (at zero) the 2 labels never exist in the same image.

(i) The first experimental section analyzes a centralized Machine Learning training and BSP using *LeNet*, *ResNet* and *AlexNet* to get a picture of their impact without using Federated Learning.

(ii) In the following section we compare the impact of different $C_{fraction}$. We use *FedAvg* and run each Deep Learning model with $C_{fraction} \in \{0.5, 0.75, 1\}$ and 8 clients.

(iii) The next part of the experiment considers each Deep Learning model on *FedAvg* using 8 clients and $C_{fraction} = 0.75$. This examination increases the *Skewness* in comparison to other experiment sections to 60% and 80%.

(iv) This section focuses on a smaller skew percentage with *Skewness* set to 40%. 20% and 0%. We use each of the three Deep Learning models and apply them on *FedAvg* and *BSP* with 4 clients.

(v) We compare the impact of different client numbers in this experimental section. For each model we run a training with client numbers $n \in 10, 25, 50$ on *FedAvg* with $C_{fraction} = 0.5$.

(vi) We repeat the experiment from (v) using *FedProx*.

(vii) Finally, we measure the weight of different batch sizes bs with $bs \in 1, 4, 8, 16, 32, 64, 128, 256$ on *FedAvg* with 4 clients using *LeNet*.

D. Evaluation metrics

Simple Accuracy is defined as:

$$\text{Accuracy} = \frac{1}{m}\sum_{i=1}^{m}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|} \tag{2}$$

where Z_i denotes the model prediction for the data point x_i, Y_i denotes the true label of x_i, and $i \in \{1, ..., m\}$. This measure however can be misleading in measuring the quality of the learned model for multilabel applications (also depends on the nature of the dataset). For example, in UC Merced LandUse multilabel dataset using this race metric one can achieve 80% by predicting the single label *pavement* for all the images. Hence this was eventually dropped from our final evaluation metric.

Other metrics such as Classification Accuracy [18], defined as:

$$\text{ClassificationAccuracy} = \frac{1}{m}\sum_{i=1}^{m}\delta\left(Z_i, Y_i\right) \tag{3}$$

where $\delta = 1$ only if the prediction match the true label for all the labels otherwise $\delta = 0$, can be too rigid of a metric.
In general the evaluation of methods that learn from multi-label data requires different measures than those used in the case of single-label data [19], those evaluation measurements can be divided in example based, label based, and ranking based.[19]
For our experiment, we use one of the label based measurements, that is the harmonic mean between precision and recall also known as F1 score 4, which also can be used for evaluating single-label classifier.

$$F_1 = \frac{1}{m}\sum_{i=1}^{m}\frac{2\,|Y_i \cap Z_i|}{|Z_i| + |Y_i|} \tag{4}$$

E. Implementation Details

The implementation is done completely in python. It can be accessed in our github repository. For ease of use, the anaconda distribution of python was used. We use PyTorch [20] as the preferred choice of Deep Learning Framework. To simulate the different clients for federated learning, we initially considered using PySyft [21], but ran into many issues because of the nascent nature of the library. It was incompatible with multilabel dataloader on pytorch and also did not support custom data splitting for the data on different clients. These

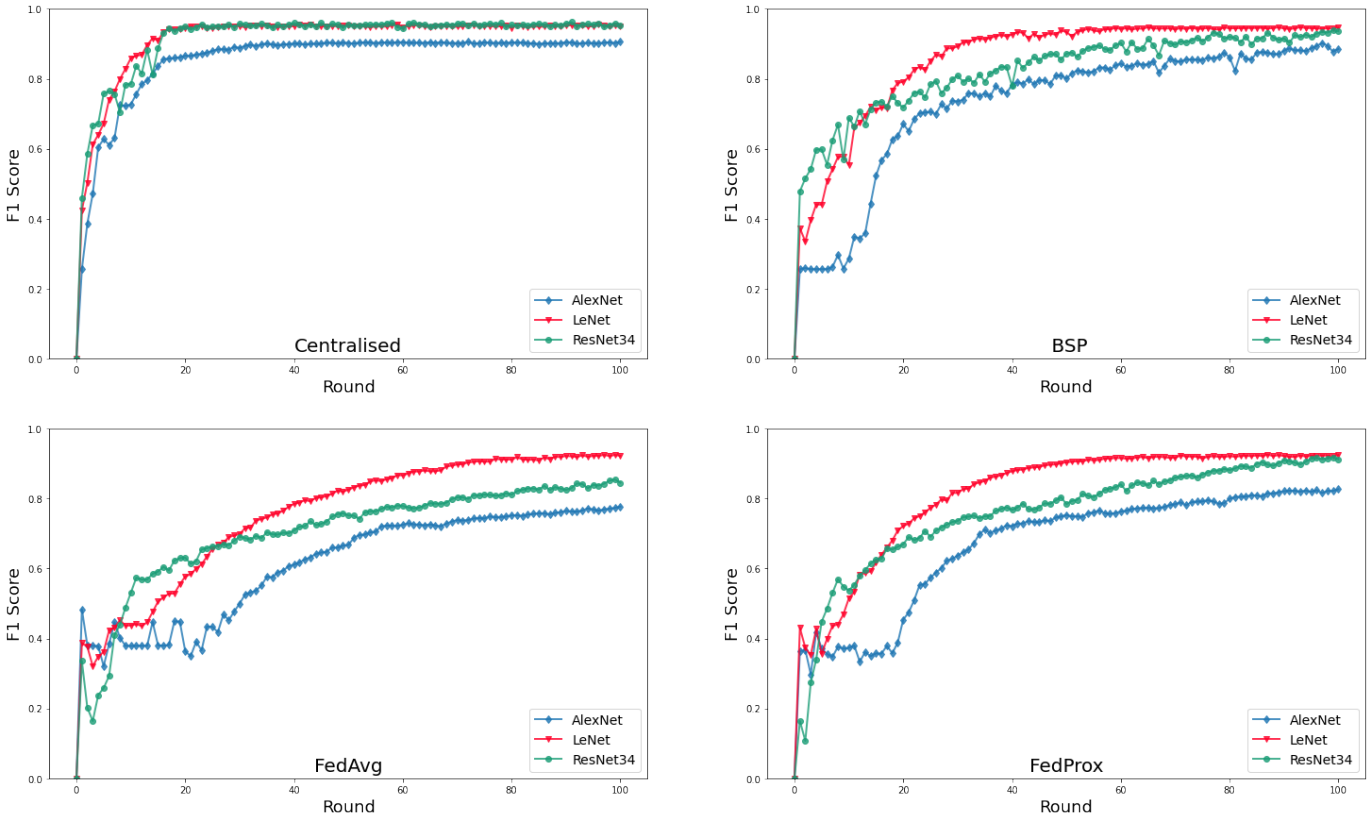


Fig. 8. Comparing centralized vs BSP and federated Learning. For BSP, FedAvg and FedProx 8 clients 40% skewness on the less common labels were used. The $c_{fraction}$ for FedAvg and FedProx is set to 0.75 here.

challenges proved to be too big and we then decided to use PyTorch directly to simulate the clients and concentrate more on the implementation of the federated algorithms. We use Pandas, matplotlib, numpy packages for the visualisation of the data and generate line plot, bar graphs, etc.

F. Experimental Setup

The experiments were conducted on a workstation with Intel(R) Xeon(R) W-2133 12 core CPU @ 3.60GHz with 64GB RAM. It was equipped with a NVIDIA GeForce RTX 2080 12GB GPU. It used Ubuntu 18.04 with CUDA, and cuDNN for GPU acceleration.

IV. Experimental Results and Discussion

This paper aims to analyze the effect of decentralized learning on different deep learning models for multi-label remote sensing data. In this section we present our results, showing federated model quality differences compared to centralized learning for the three Deep Learning models (*AlexNet*, *ResNet* and *LeNet*). Then we look at the influence of hyperparameters and other settings such as batch size, $c_{fraction}$, number of clients and the degree of skewness for Federated Averaging.

A. Overall Training Results

We present the overall training results with two main criteria in mind. The F1-score and convergence time.

1) Centralized v. Federated: As shown in Fig 8, centralized learning converges better and quicker than all the federated learning algorithms in terms of F1 quality score. But it is important to keep in mind that for centralized learning, skewness is not considered at all and a direct comparison to federated learning is unfair. Centralized training results however must therefore be seen as a benchmark results and not used for direct comparison.

2) Comparison of Federated Algorithms: BSP started to converge fast in the first 20 training rounds, it slowed down thereafter but steadily approached the upper bound of the centralized learning. This pattern is the same for all the three Deep Learning models.

Both FedAvg and FedProx are much slower in converging (than BSP) for most deep learning models. They also fail to reach the upper bound of centralized learning results, sometimes even after training 100 rounds. In direct comparison between FedAvg and FedProx, seen better in Fig 9, it is clear that FedProx (in pink plots) has the upper hand in both quality and convergence speed. That is because of FedProx's loss function being able to keep the clients in check and prevents diverging from the central average model thus being capable of handling different data distribution and skewness better than FedAvg.

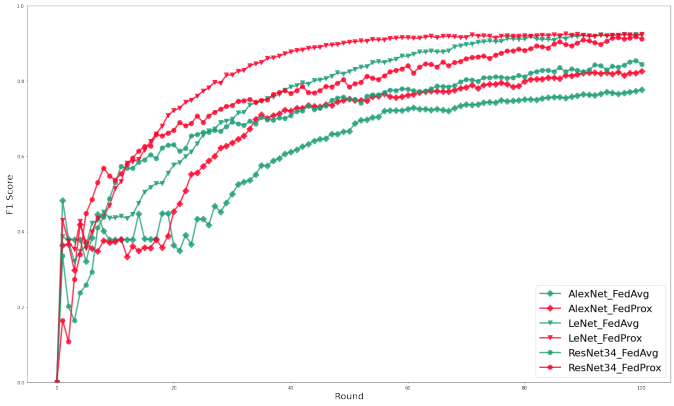


Fig. 9. FedAvg vs FedProx for different Deep Learning Models. 8 clients with a $c_{fraction}$ of 0.75, as well as 40% skewness on the less common labels were used here.

B. Comparison of Deep Learning Models

We compare the three deep learning models, i.e. LeNet, ResNet and AlexNet based on the results presented in Fig 8. For all the federated and centralized training experiments, AlexNet performs the worst in terms of F1 score. This could be because AlexNet is a very large model and requires large datasets to be trained correctly. Given that our dataset size is quite small even with augmentation, AlexNet needs the dataset to be much larger. Out of the other two models, LeNet generally converges very quickly, compared to ResNet. This could again be due to the fact that LeNet is a small model and hence is quite suitable for our application. Finally, ResNet manages to converge to the same level as LeNet for BSP. In FedAvg, however, ResNet lags behind LeNet. This is again fixed using FedProx which has better convergence for ResNet.

C. Drill-down Experiments for Federated Averaging

In this section, we take a deeper look into results using *Federated Averaging* and varying different hyperparameters and other settings to analyze the effect that it has on the convergence of the deep learning models. For each of these sets of experiments, we vary one of the parameter, while keeping all the other values and settings the same.

1) Client Fraction: We vary the Client Fraction ($c_{fraction}$) between 0.5, 0.75 and 1. We use 8 clients for these experiments and consider the less common labels to maintain equal data distribution among the clients. Looking at Fig. 10, the training using $c_{fraction} = 1$ converges the best which was expected since it uses all the clients and effectively all the training data during each round. Predictably, the convergence drops for $c_{fraction} = 0.75$ for ResNet and AlexNet. Setting $c_{fraction} = 0.5$ for these models further deteriorates the F1-score. It is remarkable that LeNet is still able to achieve optimum results with $c_{fraction} = 0.5$, even though it takes longer to converge. Even though the accuracy drops occur with lowering the client fraction, it should be taken into consideration that this also reduces communication costs which can help to manage bandwidth costs. This will be further discussed in Section IV-D.

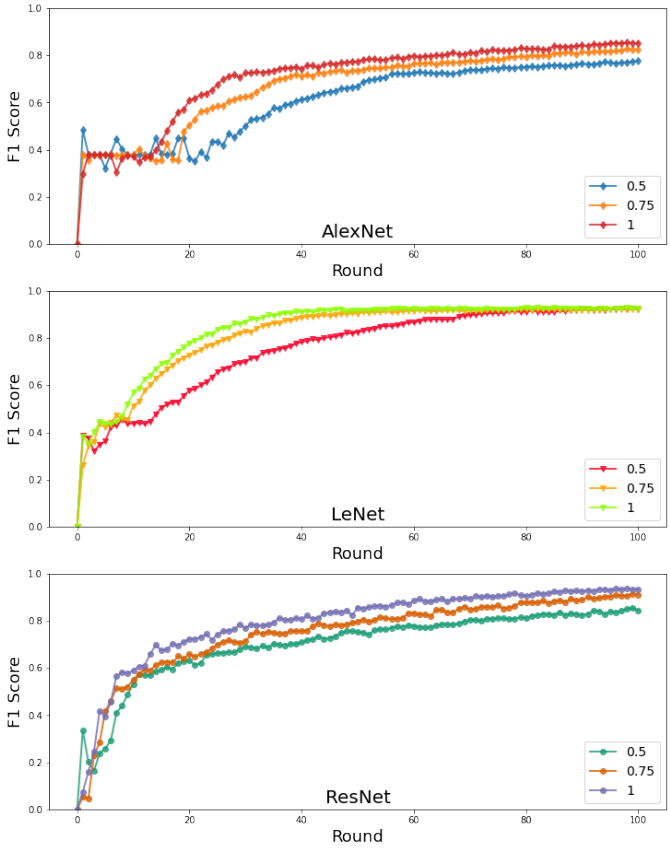


Fig. 10. Effect of varying $C_{fraction} \in \{0.5, 0.75, 1\}$. 8 clients, and 40% skewness on the less common labels were used.

2) Number of clients: We vary the number of clients between 10, 25 and 50 with client fraction set to 0.5 for all the runs. This effectively means approximately half of the data is used for training on each round. Since the client numbers are large, we have to use the less common labels to have equal number of data in all the clients, thus effectively making the data IID in nature. From Fig 11, it is quite clear that increasing the number of clients impacts the convergence quite drastically. All the three models converge faster and better for $n = 10$. Next, there is a drop in F1-score for $n = 25$, and a further drop for $n = 50$. In the case of AlexNet, given that it is a very big model, our hardware restrictions did not allow us to scale beyond 30 clients and hence the experiment for $n = 50$ was not completed.

3) Batch Size: Batch size is varied between 1, 4, 8, 16, 32, 64, 128, and 256. We use 4 clients with $c_{fraction} = 0.75$ for these experiments. Increasing the batch size affects the convergence of the deep learning model conversely as seen from Fig. 12(a). For batch size of 1 the model converges the quickest and to highest score. With increase in the batch size, the model consistently takes longer to converge and converges to lower F1-scores. The biggest drop in F1-score is between 16 and 32 where the f1 score drops by around 22%, and for

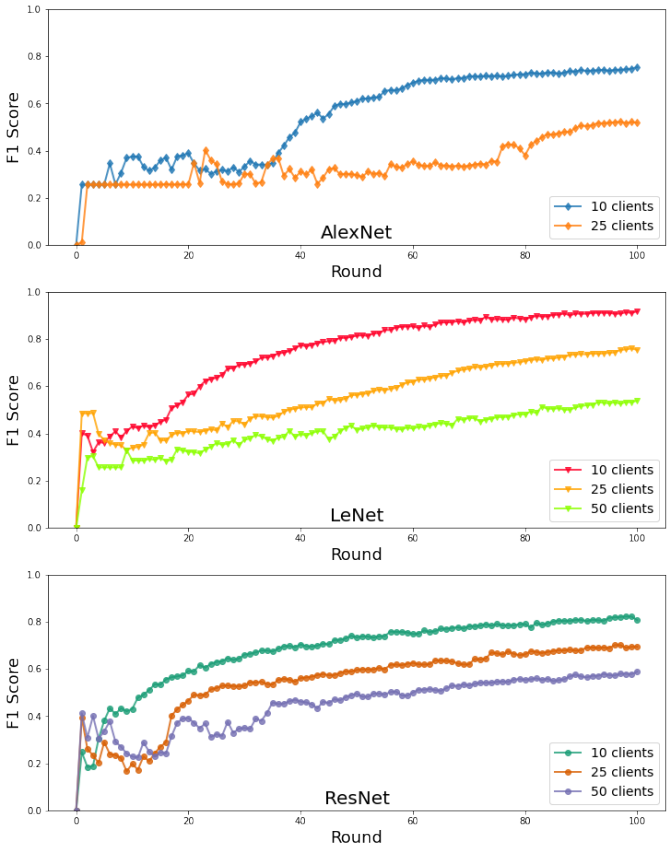


Fig. 11. Effect of varying number of $clients \in \{10, 25, 50\}$. $c_{fraction}$ of 0.5, and 40% skewness on the less common labels were used here, however since the number of clients are more than the number of unique labels the distribution over the clients end up more IID.

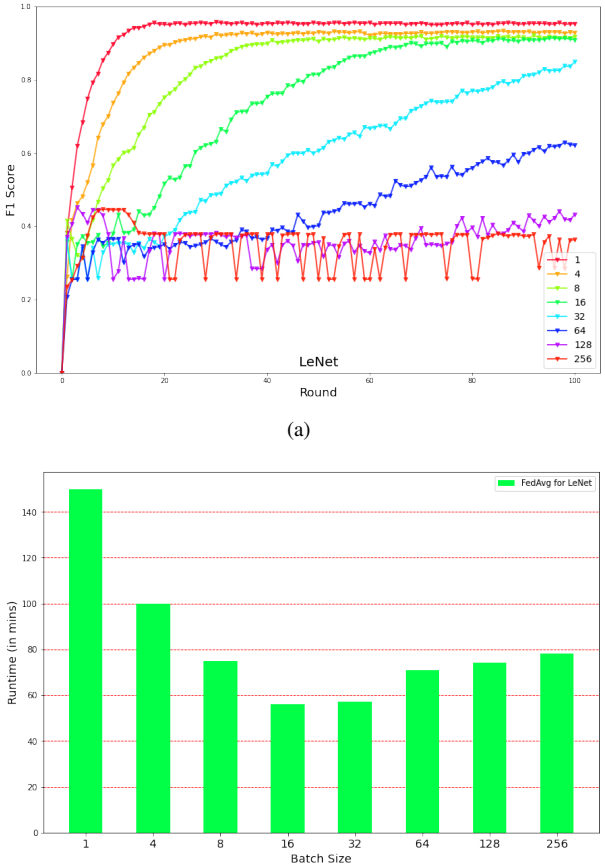


Fig. 12. (a) shows the effect of convergence for different batch sizes, with 4 clients and 40% skewness on common labels. (b) shows the runtime for 100 rounds for different batch sizes. The training runs are for LeNet.

larger batch sizes (64, 128, 256) the model fails to converge to any meaningful results.

While it is quite evident that batch size 1 performs the best, when looking into the bar graph presented in Fig. 12(b) we see that the run times are very different for different batch sizes. The runtimes presented are for 100 rounds. A batch size of 1 takes around 2.5 times longer than for batch size 16, where the drop in F1-score between them is 3.5%. So depending on the application and the efficiency and hardware required it could be more suitable to use larger batch sizes for drop of a few accuracy points. Further increase in batch size leads to slight increase in run times and this could be due to the overhead costs due to memory restrictions. While the empirical results show that batch size 16 might be an ideal balance between run time and f1 score, but this is only for LeNet model and the optimum batch size number heavily depends on the deep learning model used. These results could vary for AleXNet and ResNet.

4) Data Skewness between Clients: Initially, we use common labels for splitting the data to generate non-IID distribution. The intial baseline for these experiments are 0% skewness which represents the IID data distribution. Next we increase the skewness to 20% and 40%. The number of clients used is 4 and for our dataset there was no apparent difference in the convergence for these degrees of skewness. As seen in Fig. 13, on all the 3 Deep Learning models the model convergence for different skewness are very similar and converge also to similar final F1-score values. This indicates that FedAvg is able to handle low levels of non-IIDness in the data quite well. Next, we further increase the skewness to higher values of 60% and 80%. For this we will have to use higher number of clients and the less common labels as explained in section III-B(iii).

With 8 clients we use skewness of 40%, 60% and 80%. These learning curves are shown in Fig 14(a). Again we see very similar learning curves but there is a slight drop in the learning curves convergence time for skewness of 80% in all the 3 models. This is especially seen clearly in the LeNet learning curves where convergence takes longer than the lower skewness case. To showcase the difference better, we present the maximal F1-scores of the three different skewness degrees in Fig. 14(b) for the three Deep Learning models compared to BSP for same skewness settings. We can see that there is a drop of F1-score with increase in skewness albeit slightly. Overall, for the dataset we have used, we can summarise that

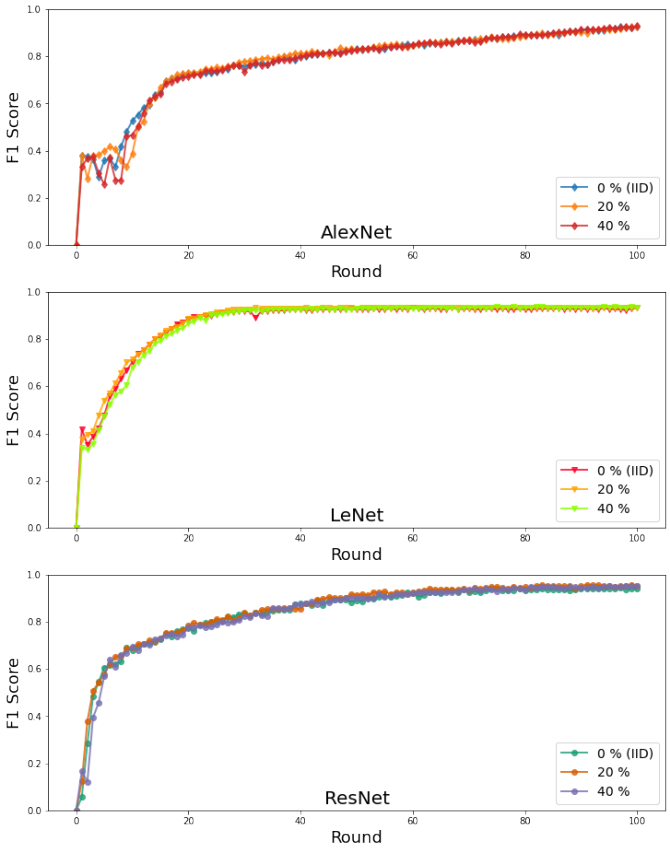


Fig. 13. Effect of varying data $skewness \in \{0, 20, 40\}\%$ on common labels on LeNet. 4 clients with a $c_{fraction}$ of 0.75 were used.

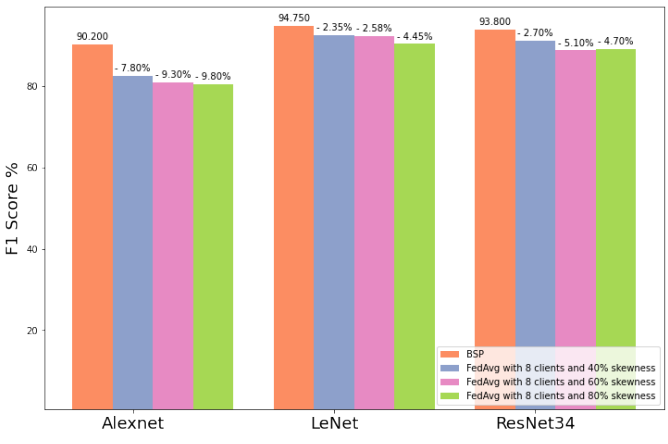
skewness of the data does not impact the learning of the Deep Learning models using FedAvg. These results might vary when using a larger dataset.

D. Communication Cost Comparison

In this section we present the communication costs associated with the different federated algorithms and try to evaluate the most optimal trade off between accuracy and communication cost among the different setups.

In Fig 15(a) we see a comparison of BSP at maximum communication mode (i.e when a model is moved from one client to another after each batch of data) with other methods of federated learning. The communication costs are so high that the other methods are almost unnoticeable on the bar graph. Next we compare BSP with better efficiency technique, where the model is moved after training on the entire partition of a client to the next client. *Federated Averaging* and *Federated Proximal* methods have the same communication cost and hence they are shown using the same bars. The client fraction for these two federated algorithms is varied. Lower client fraction means that the model is moved to lesser number of clients and this hence translated to a linear reduction in communication cost with lowering the client fraction. BSP still has the highest communication cost even after the optimization, but the difference now is much smaller.

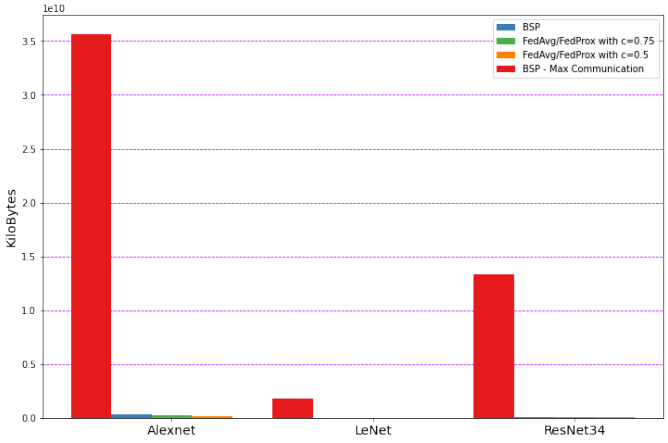
(a)



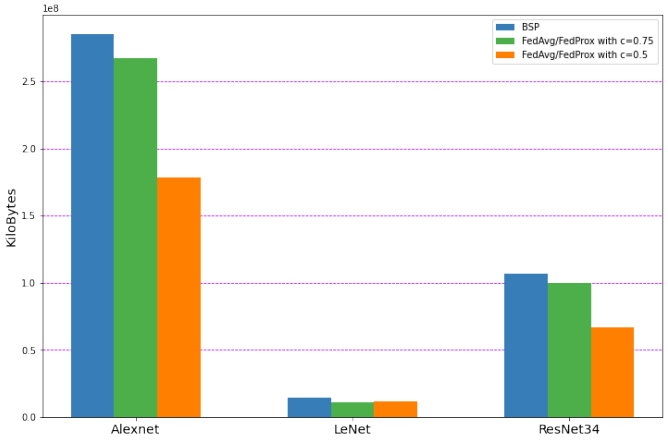
(b)

Fig. 14. (a) shows the effect of varying data $skewness \in \{40, 60, 80\}\%$ on less common labels on LeNet. 8 clients with a $c_{fraction}$ of 0.75 were used. (b) shows the difference between the max F1 scores achieved by BSP and FedAvg.

The problem with BSP however is that the model has to be trained sequentially on each client and this means the runtime on BSP is again high compared to FedAvg where the training can happen on n different clients at once. The communication costs also depend heavily on the deep learning model used. In case of LeNet the model communication cost is quite low



(a)



(b)

Fig. 15. Both graphs show the training communication costs in KiloBytes on our dataset: (a) shows the cost in compare to a BSP-max (in red) where the communication between the server and the client is happening after each training batch, that is very costly in compare to our BSP (in Blue) or FedAvg/FedProx. (b) shows the communication difference between our BSP (where communication happens after training on all the batches of a client) and FedAvg/FedProx with $C_{fraction} \in \{0.5, 0.75\}$

since it's a smaller network, but ResNet and AlexNet require more communication to be trained well. Depending on the size and nature of the dataset, we can opt to choose different Deep Learning networks. For our use case, LeNet was able to classify the data well and is generally also the most optimal with communication costs in mind.

V. Conclusion

We present the findings and results of applying federated learning for multi-label image classification on a remote sensing dataset. Federated learning has known advantages which become more relevant for remote sensing cases and our experiments certainly show that federated learning can be a useful training solution for remote sensing even when the data on different clients is non-IID in nature.

We evaluate three different federated learning algorithms: Bulk Synchronous Parallel (BSP), Federated Averaging (FedAvg) and Federated Proximal (FedProx) using three Deep Convolutional networks *LeNet*, *AlexNet* and *ResNet34*. BSP performs the best among the three federated algorithms, but given its high communication costs and/or runtimes for a practical use case, FedAvg and FedProx might be more suitable. Albeit a slight drop in F1-score, these algorithms achieve results quite efficiently and also provide a parameter called client fraction which can be used to control the trade off between communication cost and accuracy. For the UC Merced Land Use dataset LeNet performed the best in our experiments. We also discussed the effect of varying different hyperparameters on the overall model convergence and presented the best practices for the same.

A. Future Work

In the future, we would like to test out the experiments on a bigger dataset as this would help to validate these results for a larger scale. We speculate that using larger datasets might also give different results when it comes to a high data skewness use case.

We would also like to experiment on a more complex dataset, where the remote sensing images have more channels than the rgb imagee in UC Merced Land Use dataset. One dataset that could suffice both size and complexity requirement could be BigEarthNet that contains more than 500 thousand 13-channels images[22].

We also plan to implement another federated learning approach which manipulate the gradients rather than weights to handle client divergence. Deep Gradient Compression [12] is an ideal candidate for such a method. This will give us more insight to which federated algorithm works for which application.

B. Acknowledgements

This work was supported by our mentor Arne de Wall. We thank him for all his feedback provided to us on a weekly basis and for his technical support and tips to achieve the required results quickly.

References

[1] Vitor C. F. Gomes, Gilberto R. Queiroz, and Karine R. Ferreira. An overview of platforms for big earth observation data management and analysis. *Remote Sensing*, 12(8), 2020.

[2] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html, 2017.

[3] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B. Gibbons. The non-iid data quagmire of decentralized machine learning. *CoRR*, abs/1910.00189, 2019.

[4] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016.

[5] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *CoRR*, abs/1812.06127, 2018.

[6] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[7] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. *CoRR*, abs/1707.02921, 2017.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[10] Yi Yang and Shawn D. Newsam. Bag-of-visual-words and spatial extensions for land-use classification. In Divyakant Agrawal, Pusheng Zhang, Amr El Abbadi, and Mohamed F. Mokbel, editors, *GIS*, pages 270–279. ACM, 2010.

[11] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. Gaia: Geo-distributed machine learning approaching lan speeds. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 629–647, USA, 2017. USENIX Association.

[12] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training, 2020.

[13] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning, 2020.

[14] Jenny Hamer, Mehryar Mohri, and Ananda Theertha Suresh. FedBoost: A communication-efficient algorithm for federated learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3973–3983. PMLR, 13–18 Jul 2020.

[15] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. Fetchsgd: Communication-efficient federated learning with sketching, 2020.

[16] B. Chaudhuri, B. Demir, S. Chaudhuri, and L. Bruzzone. Multilabel remote sensing image retrieval using a semisupervised graph-theoretic method. *IEEE Transactions on Geoscience and Remote Sensing*, 56(2):1144–1158, 2018.

[17] Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and surface variations, 2019.

[18] Shenghuo Zhu, Xiang Ji, Wei Xu, and Yihong Gong. Multi-labelled classification using maximum entropy method. pages 274–281, 08 2005.

[19] Oded Z Maimon. Data mining and knowledge discovery handbook, 2005.

[20] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A Matlab-like Environment for Machine Learning. Technical report.

[21] OpenMined. Openmined/pysyft.

[22] Gencer Sumbul, Marcela Charfuelan, Begüm Demir, and Volker Markl. Bigearthnet: A large-scale benchmark archive for remote sensing image understanding. *CoRR*, abs/1902.06148, 2019.

TABLE I
EXPERIMENTAL SETUP: PARAMETERS

DL Model	FL Algorithm	Epochs	Clients	Batch Size	C-Fraction	Skewness	Client Epochs	Small Skew
LeNet	Centralized	100	NA	4	NA	NA	NA	NA
ResNet	Centralized	100	NA	4	NA	NA	NA	NA
AlexNet	Centralized	100	NA	4	NA	NA	NA	NA
LeNet	BSP	100	8	4	0.5	40	5	TRUE
ResNet	BSP	100	8	4	0.5	40	5	TRUE
AlexNet	BSP	100	8	4	0.5	40	5	TRUE
LeNet	FedAvg	100	8	4	0.5	40	5	TRUE
LeNet	FedAvg	100	8	4	0.75	40	5	TRUE
LeNet	FedAvg	100	8	4	1	40	5	TRUE
ResNet	FedAvg	100	8	4	0.5	40	5	TRUE
ResNet	FedAvg	100	8	4	0.75	40	5	TRUE
ResNet	FedAvg	100	8	4	1	40	5	TRUE
AlexNet	FedAvg	100	8	4	0.5	40	5	TRUE
AlexNet	FedAvg	100	8	4	0.75	40	5	TRUE
AlexNet	FedAvg	100	8	4	1	40	5	TRUE
LeNet	FedAvg	100	8	4	0.75	60	5	TRUE
LeNet	FedAvg	100	8	4	0.75	80	5	TRUE
ResNet	FedAvg	100	8	4	0.75	60	5	TRUE
ResNet	FedAvg	100	8	4	0.75	80	5	TRUE
AlexNet	FedAvg	100	8	4	0.75	60	5	TRUE
AlexNet	FedAvg	100	8	4	0.75	80	5	TRUE
LeNet	BSP	100	4	4	0.75	40	5	FALSE
AlexNet	BSP	100	4	4	0.75	40	5	FALSE
ResNet	BSP	100	4	4	0.75	40	5	FALSE
LeNet	FedAvg	100	4	4	0.75	40	5	FALSE
AlexNet	FedAvg	100	4	4	0.75	40	5	FALSE
ResNet	FedAvg	100	4	4	0.75	40	5	FALSE
LeNet	FedAvg	100	4	4	0.75	20	5	FALSE
AlexNet	FedAvg	100	4	4	0.75	20	5	FALSE
ResNet	FedAvg	100	4	4	0.75	20	5	FALSE
LeNet	FedAvg	100	4	4	0.75	0	5	FALSE
AlexNet	FedAvg	100	4	4	0.75	0	5	FALSE
ResNet	FedAvg	100	4	4	0.75	0	5	FALSE
LeNet	FedAvg	100	10	4	0.5	40	5	TRUE
AlexNet	FedAvg	100	10	4	0.5	40	5	TRUE
ResNet	FedAvg	100	10	4	0.5	40	5	TRUE
LeNet	FedAvg	100	25	4	0.5	40	5	TRUE
AlexNet	FedAvg	100	25	4	0.5	40	5	TRUE
ResNet	FedAvg	100	25	4	0.5	40	5	TRUE
LeNet	FedAvg	100	50	4	0.5	40	5	TRUE
AlexNet	FedAvg	100	50	4	0.5	40	5	TRUE
ResNet	FedAvg	100	50	4	0.5	40	5	TRUE
LeNet	FedProx	100	8	4	0.75	40	5	TRUE
AlexNet	FedProx	100	8	4	0.75	40	5	TRUE
ResNet	FedProx	100	8	4	0.75	40	5	TRUE
LeNet	FedProx	100	25	4	0.5	40	5	TRUE
AlexNet	FedProx	100	25	4	0.5	40	5	TRUE
ResNet	FedProx	100	25	4	0.5	40	5	TRUE
LeNet	FedProx	100	10	4	0.5	40	5	TRUE
AlexNet	FedProx	100	10	4	0.5	40	5	TRUE
ResNet	FedProx	100	10	4	0.5	40	5	TRUE
LeNet	FedAvg	100	4	1	0.75	40	5	FALSE
LeNet	FedAvg	100	4	4	0.75	40	5	FALSE
LeNet	FedAvg	100	4	8	0.75	40	5	FALSE
LeNet	FedAvg	100	4	16	0.75	40	5	FALSE
LeNet	FedAvg	100	4	32	0.75	40	5	FALSE
LeNet	FedAvg	100	4	64	0.75	40	5	FALSE
LeNet	FedAvg	100	4	128	0.75	40	5	FALSE
LeNet	FedAvg	100	4	256	0.75	40	5	FALSE