CHARLES UNIVERSITY

Faculty of Arts

Department of Logic

BACHELOR THESIS

Martin Georgiu

Federated learning

Thesis title in Czech:	Kolaborativní učení
Supervisor of the bachelor thesis:	Mgr. Ing. Petr Svarny PhD
Study programme:	Logic

Prague 2023

I would like to thank Petr Svarny for all the help and effort he put into this thesis, particularly for his always insightful comments and also his patience with my English. Moreover, I'm thankful to Ema Viackova for helping out with the English grammar even as my additional questions kept coming.

Title: Federated learning

Author: Martin Georgiu

Department: Logic, Faculty of Arts, Charles University

Supervisor: Mgr. Ing. Petr Svarny PhD

Keywords in English: federated learning | machine learning | collaborative learning

Keywords in Czech: federativní učení | strojové učení | kolaborativní učení

Abstract in English:

The remarkable advancements in machine learning in recent years have been unprecedented, yet the constant need for more and more data to train artificial neural networks (ANNs) remains. In sectors such as healthcare, it is unrealistic to aim to create one single dataset that would consolidate all the information about patients from various hospitals. When training ANNs, the data cannot leave the hospital, and therefore any ANN training can be executed solely locally on the given hospital's data. Federated learning (FL) is a novel approach that can be used in such settings, maintaining the system's privacy without compromises. In this thesis, we are comparing FL against other approaches striving for the same objective, diving into the security of FL and investigating concrete strategies for FL. Lastly, we've created a fully working open-source example of skin spot analysis trained using FL, which can also be easily extended and used with other datasets.

Abstract in Czech:

Pokrok v oblasti strojového učení v posledních letech byl bezprecedentní, přesto je stále potřeba stále více dat pro trénování umělých neuronových sítí (artificial neural network, ANN). V sektorech jako je zdravotnictví je velice těžké, většinou nereálné, vytvořit jeden soubor dat, který by konsolidoval všechny pacientské informace z různých nemocnic. Proto lze jakékoli trénování ANN provádět výhradně lokálně na datech jedné dané nemocnice. Federativní učení (FL) je nový přístup, který lze v takovém prostředí použít a uchovat tak uživatelská data v soukromí. V této práci porovnáváme FL s jinými přístupy usilujícími o stejný cíl, zaměřujeme se na bezpečnost FL a prozkoumáváme konkrétní strategie pro FL. Nakonec jsme také vytvořili plně funkční open-source ukázku analýzy pih natrénovanou pomocí FL. Tu lze snadno rozšířit a použít i s jinými soubory dat a cíly.

Contents

Chapter 1

Introduction

The progress in recent years and development in AI has been revolutionary. The steps in AI research taken in the past decade have been incredible, and everything is speeding up even more. In 2020, an art-generating contest (Artathon) was held. Only mere two years later, an AI art generator, DALL-E 2, sweeps all the contestants by a long shot. While art generators are not the focus of this thesis, it's an excellent and ever-so-present example of the untamed growth of AI.

Even during the writing of this thesis (from October 2022 to May 2023), there were far-reaching achievements occurring right before our eyes. On the 30th of November, ChatGPT [45] was released, and the sudden growth of popularity and speed of adoption were rapid. Just two months after the initial release, it reached 100 million active users. The previous record holder for the fastest reach of 100 million active users was TikTok and it took it long nine months to reach [28].

However, the main requirement needed for making such inventions and discoveries is data. In a few areas, relevant data is heavily and readily available to the public. That is certainly not the case with medical data. When dealing with datasets such as medical records and images, each piece of data needs to be stored securely and safely, ideally without exposing the contents to any third party. In this thesis, we plan to come up with some possible solutions and ideas on how we could create an artificial neural network for image classification from a bunch of different sources without the need to send the data to one central location. The aim is to provide a comprehensible summary of federated learning in a popular and understandable way by using a familiar tone. It is expected that the reader has at least an elementary level of understanding AI and of artificial neural networks training.

Firstly we'll go over different ideas and define some terms in the chapter Federated learning approaches. Then we will highlight the key strategies for federated learning and try to explain them easily and in the context of the others in Strategies for federated learning. Next, we dive into the overall security of federated learning, if there are any novel risks when the training is spread between many devices and we question if even just model weight updates can be compromised in any way, in the chapter Security. Lastly, in Ready-to-use example, we will create a working example (`https://github.com/martingeorgiu/spots_federated`) in Pytorch [48] and Flower [7] for which we provide the thinking preceding the training and especially how to tackle the dataset imbalance. After that, we want to show and compare the training results, to see if and how much federated learning can decrease the overall accuracy.

Chapter 2

Federated learning approaches

In this section, we would want to investigate what are the possible solutions for training a model without the need for one central location. This thesis focuses on a few example approaches of federated learning and does not aspire to list all of them.

2.1 Universal example

Firstly, we will propose a universal example to which we will refer throughout the thesis. Imagine that we are building a mobile application that can analyze a photo of skin and decide whether it contains any spots that might be signs of skin cancer and show its analysis confidence in percentages. Eventually, it could evolve into a more complex analysis like other skin defect diagnoses like Psoriasis Vulgaris and even recommend the proper treatment. From a “first-world” point of view, this might sound strange or outright illegal as diagnosing an illness and recommending treatment is typically something that only a qualified doctor can legally do. But also consider other, usually poorer, countries where essential healthcare is absent or usually unaffordable, like Nigeria or USA.

The issue now is how to create such an app. The old-school approach would be to understand what these starting skin cancers look like and what are their shapes, colours, sizes, etc. On top of this knowledge, we would try to create some algorithms. However, every solution which is based on a manually configured algorithm carries with itself the same problem of future changes and corrections. What would one do when a patient would bring a photo classified by the algorithm as carcinogenic, but after an examination later by a doctor, it was classified as harmless?

The universal answer to such a problem is ML and a trainable artificial neural network (or ANN). But of course, it has some shortcomings. The major one is usually the amount of data required to train it. This always depends on the complexity of the network and the goal in question. Then there is the issue of attempting to comprehend what is the model doing under the hood to be able to explain its decisions and reasons for a given classification. For exactly that, there is a sub-field of AI called “explainable artificial intelligence” (XAI) and currently there is a lot of ongoing research regarding this exact topic [53, 57]. There are some general techniques like probing classifiers [5], knowledge distillation [47] and visualizing the model or the dataset using tools like UMAP [38], but here aren’t any standard tools for such endeavours as every inspection is very different from network to network and from goal to goal [57].

But the most pressing issue, especially when dealing with medical records, is to preserve the privacy of the data in question. This can be resolved by signing an agreement with the 3rd party. But is it the proper solution? Would a patient want to sign some data-sharing agreement so that some company, of which he never heard before, can have access to his medical records? Usually, we wouldn’t say so. But because there wasn’t the technology that would enable a better approach, that is the state we are currently at. Yet in the past years, there has been quite a progress on this exact question, and this thesis focuses mainly on the major solution, which is Federated learning.

2.2 Main issues

Before diving deep into the possible solutions, we would like to step back a bit and discuss possible systematic issues with Federated learning. The main reason is that usually, creating an artificial neural network is only a small portion of what an ML engineer is actually doing. The actual work starts much earlier than that.

Ideally, there should be initial research on the topic to get at least more-than-amateur domain knowledge of the topic in question. Let us adhere to the universal example we've described above. In that case, and if we assume the best-case scenario, the researcher would meet a couple of dermatologists, general practitioners, nurses, and overall everyone in charge of analyzing the spots currently. Then he can start creating a plan for creating an app for the general public based on the newly acquired knowledge. Let's say he decided that the best solution for the app would be to create a dataset containing photos of the skin lesions and use the latest ML techniques on top of that. It will be annotated with a diagnosis (whether it is melanocytic nevi, melanoma, or others), sex, age, and possibly many other parameters. He will work with the field experts to get these photos and these descriptions.

But most importantly, during this collection period, he would oversee the incoming data and their quality as he'll be collecting the data from practitioners. Every client has different lighting, they might even be from different parts of the world, which means some doctors will have 95% black people, and some others will have a majority of white people. Some of the doctors might be well-known experts in diagnosing and treating specific lesions, so their opinion might matter more than the opinion of others. Doctors in poorer regions might have a worse capturing device, making the image quality of their collected images noticeably inferior. There are many other things one could do. Therefore we definitely won't list everything. But the point we want to make is that this inspection before you train an artificial neural network is almost always needed. With a bit of luck, you might train a successful network even without that, but you won't have many hints on how to improve the networks for the next generations and updates.

Apart from the issue portrayed in the paragraph above, the researcher can still do all the other stuff without seeing the final data, like meeting domain experts and trying to decide the most important parameters. However, he still cannot do the data inspection. So now the primary goal of us researchers/developers/engineers is not to build the biggest, most extraordinary dataset on which we can do some analysis, data mining, and build our ML models. Instead, we need to focus more on the process in advance and know precisely what the goal is and how we want to achieve that, as once we release it, it is a black box over which we do not have any control, and we hope for the best.

2.3 IID vs Non-IID

IID, or "independent and identically distributed" (sometimes abbreviated i.i.d. or iid), is a term very often used in the context of machine learning and is even more critical in federated learning. When a dataset is IID, it means that all elements in the dataset have the same probability distribution as the others, and they are independent of each other [8].

A typical IID dataset is the output of a dice roll where the probability is the same for all throws, and the die faces aren't dependent on each other. On the other hand, a typical example of a non-IID dataset could be stock prices on New York Stock Exchange. The prices are clearly not identically distributed, as the prices of each individual company are influenced by a multitude of factors like the company's economic indicators or latest achievements. Also, the prices aren't even independent. For example, a sudden drop in the stock price of company A can boost the stock price of competing company B.

When setting up federated learning, having highly non-IID data can substantially slow the learning process or spoil it altogether [35]. The reason is that each individual client can have vastly different data (as mentioned in Main issues), and the clients would aim to converge to entirely different locations. Then combining them would make sense as the divergence would be immense (later discussed in Strategies for federated learning).

2.4 Cross-silo vs Cross-device approach

In general, there are two approaches you can take when doing Federated learning. You can store the data in some intermediary. In our example from the introduction, it would be the hospital, retirement home, some clinic, etc. This way, the provider of the data, in our case the patient, doesn't need to extend their trust to any other party and trusts only the institution which already collects their data, where already exists some explicit or implicit data sharing agreement. This method is called cross-silo [32].

However, this introduces a small number of clients with all the data, implying a security breach at a single client could expose a lot of data to the adversary. Considering the recent large-scale cyber attack on the Czech Republic's second-largest hospital and similar smaller incidents [30], it raises the question of how secure the data truly are in such a setup. It might be argued that rather than storing all the data in the hospital or any intermediary, it would be better to store them at the final client, in our case, the patient's mobile device. That is describing the second approach, cross-device. As the name suggests, the raw data will never escape the device on which it was taken, and that device also serves as the client for the federation learning. This way, the patient is in the centre of the healthcare and not the institutions.

2.4.1 Cross-device drawbacks and difficulties

Although the idea of cross-device is intriguing as it rethinks where the data should be stored and who owns them, the cross-silo approach is usually easier to set up and maintain since you have only a few clients you need to communicate with and provide them with the training executable.

On the other hand, the cross-device approach typically involves creating a convenient mobile app for the clients that can take care of everything that the training company needs. The app can record, store and analyze the data and do the ANN training. This then eliminates the need for distributing training executables and tutorials on how to run such training. Rather, the patient benefits from high-quality image classification and the provider is in return receiving valuable model updates from the local ANN training.

Still, there are some requirements that the device should meet, like being charged, connected to stable Wi-Fi, etc. [32], which further complicates the training. For example, one could decide to execute one training every day overnight in the Europe timezone (when the devices are most likely to be charging, on Wi-Fi and idle [32]) and thus have a new updated model every day. Yet this decision could let to systematically ignoring whole continents like Asia and the Americas as the likelihood of devices from these areas meeting the same requirements at that moment is lower because of the time shift.

Furthermore, this dependence on the patient's device leads to possible disadvantaging of poorer regions where there won't be such widespread adoption of fast and modern smartphones as in richer and developed nations. The issue is that the lower tier or older phones could run such local ANN training really slowly, if at all, and therefore their model contributions would be rather insignificant. We will discuss this more in Systems heterogeneity, but even with the introduced techniques to tackle this like Proximal term, this can never be fully resolved and needs to be thoroughly thought out when doing cross-device federation.

However, this thesis mainly focuses on the cross-silo approach that better follows our Universal example, so we won't go into further details on cross-device. For such details, there are publications like "Advances and Open Problems in Federated Learning" and "Towards Federated Learning at Scale: System Design", which mainly focus on this configuration and even investigate various possibilities of local fine-tuning.

2.5 Standard federated learning

With regard to the declared problem Universal example, we assume that in standard federated learning, there are:

- one company which acts as a coordinator and also runs and maintains the server
- multiple individual clients, each with a unique secret training dataset.

We've also highlighted this Server to Clients relationship in Figure 2.1. The five main steps, which are present in every federated learning, are represented in Figure 2.2. A detailed look at each step will be in the next chapter Strategies for federated learning, where we will go over even concrete functioning strategies like FedAvg, which behave exactly as portrayed in 2.1 and 2.2 and we will even introduce some other strategies that are in some ways extending these five steps.

Nonetheless, before we do that, we will now diverge a bit and introduce some other completely different takes on these privacy-preserving techniques.

2.6 Other approaches to federated learning

2.6.1 Fully decentralized

In contrast to Figure 2.1 in fully decentralized learning, there is no server for aggregating model weights updates or dictating the settings of the protocol. There are only the clients who send

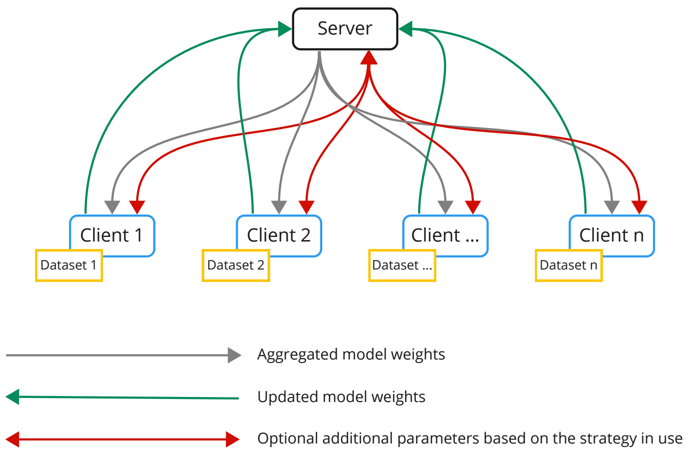

Figure 2.1: Diagram of Server-Client relation in standard training of ANN using Federated learning

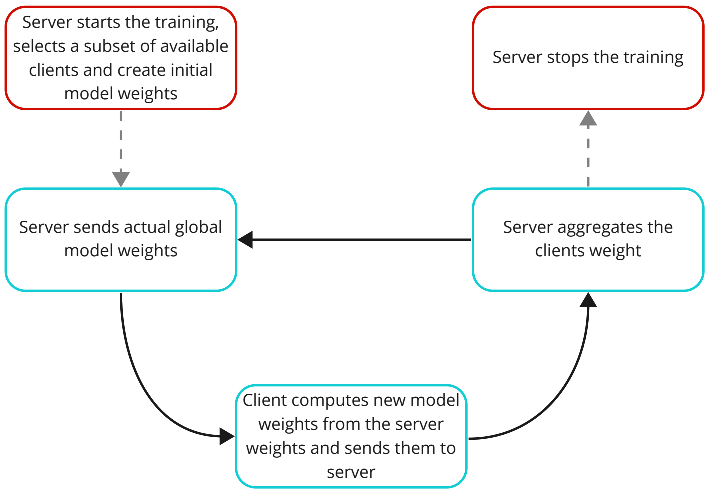
Server starts the training, selects a subset of available clients and create initial model weights
Server stops the training
Server sends actual global model weights
Server aggregates the clients weight
Client computes new model weights from the server weights and sends them to server

Figure 2.2: Five main steps in Federated learning (inspired by [32, Section 1.1.2])

the updates to their neighbouring clients and aggregate themselves the updates received from the neighbours [32].

Even though there are already created and researched algorithms for such decentralized training, like D^2, which is a modification of standard stochastic gradient descent [59] or coordinate descent [6], there are many systematic issues still unresolved.

First and foremost, how could one even set up such a network when there is no coordinator or authority? This depends on the particular situation, but there always needs to be someone who sets up the algorithm. Interestingly in contrast to the standard federated learning with a central server, the algorithm then shouldn't need any centralized maintenance or updating. For things like hyperparameters, there could even be some consensus schemes so that the clients themselves could elect a new set of hyperparameters [32].

There are also other challenges and open questions, some of which we summarize briefly here:

- Efficiency and Accuracy - while there are some publications like [71] claiming the efficiency of the network shouldn't be hurt by unreliable networks, it is still an open question how much differently would a fully distributed network compare against cross-silo one assuming we work with same data.
- Security - the fact that clients are sending model updates to each other opens up a possibility for some malicious adversary to send intentionally bad model weights to ruin the training.

This technique is called model update poisoning and it is a common problem for strategies that don't have any intermediaries but rather communicate directly with the end devices [4] (like the aforementioned cross-device).

- Longevity - without any centralized authority, it is hard to imagine how the network could update itself in the future. AI training and setups are constantly evolving, and it is unclear how that would be possible even with some election system built-in.

2.6.2 Split learning

The idea of split learning is that the ANN itself can be split between the client and the server so that the client runs a forward pass with their private data on the first part of the split ANN and then sends the activation values to the server where the forward pass is finished [50, 66]. The backward pass is then executed equivalently, just the other way around.

The main issue here is the fact that you are still sending a potentially private model's layer activation values, which has the risk of revealing some private information about the dataset or even concrete dataset examples. There are techniques to tackle this and make the cut layer (the layer where the original network was split) more privacy-preserving [65], but there aren't any comprehensive independent benchmarks that would confirm it or deny it.

2.6.3 Multi-model and local fine tuning

Multi-model approaches (as defined in [32]) are strategies where the clients can use distinct models from one another. This can be used even in Standard federated learning as one might not want to distribute a newly trained model all at once but rather do the release in phases and, if possible, monitor the usage, whether it is performing better than the previous model. Both Apple and Google support such distribution for mobile apps in their stores for the same reasons, so that the developer can limit the risks of releasing new versions [43].

However, the main new concept which the multi-model allows is local fine-tuning [20, 27, 58]. In general, fine-tuning refers to taking a trained model, which was usually trained on more broad and general data and then later it is again trained using more specific data, which can result in a better performance rather than training the model from scratch on the specific data. That way, you can train a model on a bunch of annotated medical images (spanning from x-rays of broken bones to MRI of the brain) and then with a focused dataset (e.g. focused on skin cancer) only fine-tune the pre-trained model [13, 20, 22]. This is also very helpful in applications like word completion for smartphone keyboard. Even though there are some general language rules, everyone uses a language in a different way, so if the keyboard would be able to fine-tune itself to fit exactly the user, it could achieve the perfect personalization [27].

2.6.4 Secure multi-party computation

Secure multi-party computation (Secure MPC, SMPC) is a subfield of cryptography that enables multiple parties to jointly compute a given function over their private inputs while keeping these inputs confidential[14, 32]. Nowadays, SMPCs aren't just theoretical concepts proven mathematically like it was with pioneers like [68]. There are fully working algorithms like [3, 41, 42] some of which are using new discoveries in cryptography like homomorphic encryption [69].

These SMPCs can be even used for training ANN using techniques like Secure aggregation [9]. There the SMPC protocol is used for a joint calculation of gradient descent during the training of the network without the need of revealing any private dataset. Each party does their part of the calculation on their data and no privacy is compromised [9, 42].

Chapter 3

Strategies for federated learning

Now focus more on the concrete strategies one can use when preparing for standard federated learning. In chapter Ready-to-use example, concrete examples with results will be shown using some of the said strategies.

Please note that all these strategies are simplified or, in some cases, a bit changed version from their original paper. The paper is always cited, so feel free to dive into the strategies more. But our goal was to explain only the core which you might encounter when using libraries like Flower [7]. For example, right in the following section FedAvg, we are deliberately neglecting details about **B** and η, as they are more about classical training of ANNs than some Federated learning specialty.

3.1 FedAvg

Federated Averaging or FedAvg [39] is the oldest strategy studied in this thesis and most commonly used[34, p. 5]. The original paper does a great job of explaining the strategy thoroughly, therefore let us only summarize it and point out the key thoughts:

- **R** - number of rounds
- **K** - number of clients
- **C** - the fraction of clients that perform computation on each round
- **E** - number of training passes each client makes over its local dataset on each round

Algorithm 1 `FedAvg`. Simplified version from the original paper [39]. TrainModel here refers to an arbitrary training procedure that returns model weights and can use arbitrary loss function, optimizer, or some other techniques like mini-batching

```
Server executes:
  Initialization: $w_0$
  for each round $r = 1, 2, \dots \mathbf{R}$ do
    $m \leftarrow \max(\mathbf{C} \cdot \mathbf{K}, 1)$
    $S_r \leftarrow$ (random set of $m$ clients)
    $m_r \leftarrow \sum_{k \in S_r} n_k$ // Get number of all records in this round
    for each client $k \in S_r$ in parallel do
      $w_{r+1}^k \leftarrow \text{ClientUpdate}(k, w_r)$
    $w_{r+1} \leftarrow \sum_{k \in S_r} \frac{n_k}{m_r} w_{r+1}^k$ // Calculate weighted average for the final round weight

ClientUpdate($k, w$): // Run on client $k$
  for each local epoch $e$ from 1 to $\mathbf{E}$ do
    $w \leftarrow \text{TrainModel}(w)$
  return $w$ to server
```

Setting **C** to anything other than 1 is usually only worth it when dealing with cross-device federations with thousands and tens of thousands of possible devices. The reason is that, even though the computation of local weight update is done in parallel, there is still more than just fitting the network on each client. Additional clients will still be a bigger load for the server in terms of connectivity (most likely using the internet), processing the results, and aggregating that.

The saved processing power (and subsequently time) when using fewer clients can be used to have more rounds, which is even on the intuitive level better as you are giving the fresh newly aggregated weights to the clients more often, so you are learning and converging quicker. But of course, as already written, this is true when dealing with large amounts of clients, and because of that, it can be mostly used for cross-device federation. When dealing with cross-silo federation, where you have only a few clients, you usually want to use them all.

Choosing the correct **E**, i.e. the number of epochs on each client per one round, depends heavily on the dataset which will be used and especially how it is distributed. If the spread is close to being IID (IID vs Non-IID), setting **E** to 5 or even more epochs can substantially speedup your learning process [39, Figure 2] as also shown in Figure 5.5. However, if the data is not IID, choosing a large **E** can not only slow down the learning speed but the final test accuracy can be decreased as well [39, Figure 2]. The reason is that the weights calculated by each client can start converging in different directions, so the weighted average of these weights can produce weights that will be performing poorly (having large loss value) for all the clients [39, Figure 3].

3.2 FedProx

FedProx can be considered as an extension of FedAvg, which is trying to solve two main problems in federated learning: systems heterogeneity and statistical heterogeneity [35]. Systems heterogeneity is significant variability in terms of the system's characteristics on each device in the network and statistical heterogeneity is non-identically distributed data across the network [35].

3.2.1 Systems heterogeneity

Systems heterogeneity is a much greater problem generally when dealing with the cross-device setting, as in the cross-silo, you can usually dictate or at least strongly recommend some exact hardware/connectivity. But it still could be a factor when for example, dealing with a mix of small and big public hospitals. In the small ones, you would be more than happy for some ten years server to run the client, and in the bigger hospitals, better and newer hardware is usually the standard.

But if we would stick to the premise of cross-device. If you are running on tens of thousands of devices and using FedAvg, you need to set some specific **E** and some timeout for the clients to finish up. If some of the clients won't meet the timeout, their results will be dropped. This means that you either focus on the devices which have the higher computing power and cut off the slower ones, which obviously can lead to a huge bias. Or you will wait for almost all devices setting the timeout quite big, but that will lead to the speedy devices wait and hang, till the slow ones are ready.

3.2.2 Statistical heterogeneity

The second problem lies in the statistical heterogeneity, i.e. what to do when the clients have data not distributed equally or aren't independent (see IID vs Non-IID). Our data distribution when trying out all different setups and techniques later here in Training results was intentionally the same across the clients as the goal was more about seeing the best-case scenario, and we wanted to limit the scope, and the imbalanced setting would do it even more complicated. However, statistical heterogeneity is definitely a significant issue that cannot be overlooked as it is present in almost every real-world data.

3.2.3 Proximal term

So what is the "do it all" solution to these real problems? The proposed change is to only add μ a hyper-parameter (alias "proximal mu"), which will be sent to the clients from the server and the clients will change the logic around their loss function.

First, we define the required components. We need to store the original global weights before the training, say in g, and during the training, the new updated local weights in l. That makes both g and l list of tensors of the same length n. We will also use function *L2_norm*() as a standard Frobenius matrix norm.

Then before each loss function call, we will compute the proximal term:

$$\text{proximal\_term} = \sum_{i=0}^{n} L2\text{\_norm}(l_i - g_i) \tag{3.1}$$

After that, we will call the standard loss function as previously and multiply the result by $(\mu/2) * proximal\_term$, which will be our final loss value.

This way, you can effortlessly ensure that you aren't diverging in each round from the global weights, as you penalize the model when the weights diverge too far. Nevertheless, the concrete approaches can vary. Even the original paper doesn't specify the concrete implementation and leaves the exact use up for the reader. We will more or less stick to the Flower [7] interpretation (version 1.3.0) which uses the μ as a parameter sent to the clients. Then when implementing the Flower client, you add the already-mentioned code to your loss function and that is pretty much it.

However, this can be extended much more. One simple extension would be to include the incomplete clients (clients who didn't finish all required rounds in a given time) in the server aggregation. When aggregating the weights, these non-complete results could have lower weights in the weighted average, e.g. the weight can be:

$$client\_weight = \frac{finished\ rounds}{specified\ rounds\ E} \tag{3.2}$$

or similar. As we've already pointed out in Systems heterogeneity, this could be really helpful as we aren't ignoring the slow devices and with the help of μ, we are making sure that the more performant devices aren't skewing the training in any direction.

One approach could be to use E more as a ceiling, which only a few most powerful clients will achieve, and all the others will finish whatever rounds they can. This way, we are utilizing almost all devices, no matter their powerfulness, while achieving some homogeneity using the μ.

3.3 FedBN

FedBN [36] is an easy extension of FedAvg where with no added logic on the server or in the communication itself, you can improve the convergence and speed of the training.

The standard preliminary of each ANN training is to normalize data. The idea is to have the data nicely spread out (usually from 0 to 1) instead of having custom scales with different distributions. A simple algorithm for that can be:

Algorithm 2 Mean and standard normalization example

```
images[]
for channel $c = 1, 2, 3$ do
    $m, s = mean\_and\_std(images[c])$
    for picture $p$ and it's id $i_{pic}$ in images[c] do
        for pixel $px$ and it's id $i_{px}$ in $p$ do
            $imagec[c][i_{pic}][i_{px}] = \frac{px-m}{s}$
```

However, this isn't the only stage when normalization happens. Almost every modern ANN has batch norm layers [31], and the one we've used in Ready-to-use example, MobileNetV3 [26], is no exception. In short, the normalization happens even in between the layers, as it normalizes the output from the previous layer and sends it to the next layer. It is also trainable, so the model is polished and improved even more with each batch round (that is why "batch norm").

Now we have the terms explained. The idea behind FedBN is to update the batch norm layer in the network locally, which means that we don't send them to the server at all for aggregation. The idea is that batch norm layers should be normalized and trained on the same data instead of trying to normalize them using some aggregated average across all the clients. The batch norm layers, by design, should only be there for the data normalization and not the results themselves, so it is quite intuitive that it shouldn't hurt the performance. Even more, it should be helpful and improve the performance as the data can be quite different across all the clients.

Imagine, for example, that we have two clients, one is a Nordic hospital, and the second is an Italian. The images from the Italian hospital will be much more yellowy on average as the sunshine there is much more present than in the cold north. In this case, trying to use some aggregated average wouldn't make sense as it is much better for the networks to train their batch norm layers themselves to be best for normalizing their data.

3.4 FedOpt

In the section about FedAvg, we were oversimplifying a bit. More precisely, the TrainModel() function in the Algorithm 1 was doing a bit more under the hood. It was retrieving gradient descent from a loss function, multiplying the gradient descent by a hyperparameter called learning rate η, and this gradient was then subtracted from the original weights. These final weights were then returned from TrainModel().

However, this notion of using an optimizer can be relatively extended. FedOpt [52] authors are proposing this new view on FedAvg:

Algorithm 3 `FedOpt` Edited version of algorithm copied from [52]

```
Server executes:
  Input: ClientOpt, ServerOpt
  Initialization: $w_0$
  for round $r = 0, \cdots, R-1$ do
    Sample a subset $\mathcal{K}$ of clients
    for each client $k \in \mathcal{K}$ in parallel do
      $\Delta_k^r \leftarrow \text{ClientUpdate}(k, w_r, r)$
    $\Delta_r = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \Delta_k^r$
    $w_{r+1} = \texttt{ServerOpt}(x_r, -\Delta_r, r)$
ClientUpdate($k$, $w$, $r$):   // Run on client $k$
  $w_{k,0}^r = w$
  for $b = 0, \cdots, B-1$ do
    Compute an unbiased estimate $g_{k,b}^r$ of $\nabla F_k(w_{k,b}^r)$
    $w_{k,b+1}^r = \texttt{ClientOpt}(w_{k,b}^r, g_{k,b}^r, r)$
  $\Delta_k^r = w_{k,B}^r - w_r$
  return $\Delta_k^r$ to server
```

Instead of returning the weights, we are returning the gradient in this setting, and the server is also aggregating the gradient. The idea is that we can have some optimizer which is running during the training itself called `ClientOpt` and simultaneously some optimizer, which is run on top of the aggregated gradients called `ServerOpt`. The authors of FedOpt are proposing strategies FedAdam, FedAdagrad, and FedYogi, where the idea is to use Adam, Adagrad and Yogi optimizers respectively for the `ServerOpt` and the `ClientOpt` will use standard SGD. The advantage is that SGD itself doesn't require any state, so there is no added work or communication requirement for the clients as the "stateful" optimizer is only on the server.

Algorithm 4 `FedAdam` Edited version of algorithm copied from [52]. We've highlighted the addition of Adam optimizer on the server (with standard notation β_1 for momentum and β_2 and τ for the RMSprop) and SGD on the client (with standard notation η for learning rate).

```
Server executes:
  Initialization: $w_0, v_{-1} \geq \tau^2$, decay parameters $\beta_1, \beta_2 \in [0, 1)$
  for round $r = 0, \cdots, R-1$ do
    Sample a subset $\mathcal{K}$ of clients
    for each client $k \in \mathcal{K}$ in parallel do
      $\Delta_k^r \leftarrow \text{ClientUpdate}(k, w_r, r)$
    $\Delta_r = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \Delta_k^r$
    $m_r = \beta_1 m_{r-1} + (1 - \beta_1)\Delta_r$
    $v_r = \beta_2 v_{r-1} + (1 - \beta_2)\Delta_r^2$
    $w_{r+1} = w_r + \eta \frac{m_r}{\sqrt{v_r} + \tau}$
ClientUpdate($k$, $w$, $r$):   // Run on client $k$
  $w_{k,0}^r = w$
  for $b = 0, \cdots, B-1$ do
    Compute an unbiased estimate $g_{k,b}^r$ of $\nabla F_k(w_{k,b}^r)$
    $w_{k,b+1}^r = w_{k,b}^r - \eta_k^r g_{k,b}^r$   // Standard SGD
  $\Delta_k^r = w_{k,B}^r - w_r$
  return $\Delta_k^r$ to server
```

This idea of using `ServerOpt` or `ClientOpt` can be definitely played with, and the list of possible strategies is almost endless. You can try to have a stateful Adam optimizer and, on the server, a simple stateless SGD, or put Adam both on the server and on the client, or you can experiment even with some novel optimizers which can take advantage of the server position. But this depends on your exact specific federation, dataset, client availability and many other factors. A nice extension of this would be to mix it with FedBN as both of these extensions of FedAvg are changing the original strategy in entirely independent stages.

3.5 Other possible candidates

There are many other approaches FedDyn [2], FedCurv [54] and it is left up for the reader to dive deep into some if they will find them interesting. As always with machine learning, every business case is different, every dataset is different, and all the goals are different, so it is up to the machine learning engineer to try out the ones they see the biggest potential and do their own experiments on their specific business case, dataset and goals.

Chapter 4

Security

One of the main reasons why we concern ourselves with federated learning is, of course, privacy. We want our framework to preserve the privacy of the users' private data and not compromise it in any way.

Up to this point, we've assumed that the updates the users send to the server are just weights from which the server cannot extract any information. However, this assumption is not entirely accurate. While the data sent are indeed only model weights, the server also knows the previous model weights, so it can subtract them and get the calculated gradient. From that adversary can execute Inference attacks or, in the context of GAN [23], even try to generate statistically similar images to the original images in the training set [25]. All this would go against our goal of creating a privacy-preserving model.

In this chapter, we'll first specify the two types of adversaries and what is their tactic to compromise users' data privacy. Then we will go over different types of attacks on federated learning training, and lastly, we will show how one can protect his training against these attacks.

The security of training can be compromised in two different ways based on the motivation of the adversary. These are:

Honest-but-Curious

Honest-but-Curious is a type of adversary who is not tampering the protocol or the data (in our case, model weights for an ANN) to get some otherwise unobtainable data but thoroughly inspects all the data which it receives, whether there is something valuable in them. Thus he isn't receiving anything more than we're receiving, so it could seem that privacy is not compromised. But he'll try to do his best to get as much information as possible from all the model updates, which is not something the client would want or agree to. In our case, it could mean that the adversary would hack our server but wouldn't do anything suspicious such as tinker with the model weights sent to the clients, but he would only observe and note all weights coming from and going to the clients.

Malicious adversary

A malicious adversary is a type of adversary who is actively trying to manipulate the federated learning protocol and retrieve some knowledge from this manipulation. There are many exploits the adversary can use. For example, if the server controls the amount of data each client should use for the learning, then if the server got hacked, the adversary can send them intentionally low values. The malicious server could extract from the model's update a lot of knowledge about one piece of data (e.g. one image) and, in some particular settings, even the original training data themselves.

It is important to think about both types of adversaries (honest-but-curious and malicious) when trying to protect yourself against possible attackers. This is especially in federated learning, where there is one server which collects model weights without seeing training data (run by, say, company A) and many clients who each have their own data. This type of protocol needs to be very well protected against Honest-but-Curious as if not, company A could still thoroughly observe the incoming data without the clients' knowledge. The protocol also needs to be protected against Malicious adversary, so if the company goes bad and tinkers with the protocol and data, it should not ever be able to compromise the clients' data privacy. Hence, there should be rules to ensure the server can't alter the protocol like that.

4.1 Inference attacks

Inference attacks refer to privacy attacks in machine learning where the adversary is trying to extract some knowledge from the model's updates or the trained models [40]. In classical machine learning, usually, the latter is the case as the model updates are not shared between other devices, as is the case in federated learning. However, in a federated setting, the risk of accessing the model updates is much more present as the updates are sent back and forth between the server aggregator and the clients, usually over the internet, instead of running the entire training on a single machine. Typically, these inference attacks have two types: membership or properties inference attacks [40].

4.1.1 Membership inference attacks

Membership inference attacks are a type of attack where the adversary tries to confirm whether some specific data point (e.g. an image) was used in training. It is an actively researched topic with multiple publications, surprisingly even in federated settings, with mixed results. Although some authors claim that the results are staggering, usually the results are quite close to being random and definitely not something you could use as evidence for stealing data [44, 55].

It is up for debate whether this should even be considered an attack, as it could be argued that you should always have the option to check if the data point itself has been used for the training or not. The reasoning is that it would be simple to verify if some data points were used for the training even though the company in charge of the training didn't have the required approvals, which could be even useful in settling some lawsuits [11].

4.1.2 Properties inference attacks

Properties inference attacks are an attack where the adversary tries to deduce some properties about the dataset (or its subset) on which the model was trained [40]. In a federated setting, this could again be extended, focusing on the properties of individual clients' datasets [40].

4.2 Protecting the model

A typical scenario is that you've trained the model for classifying moles we mentioned in Universal example, and you want to commercialize. You are thinking of a simple application server with a simple Rest API where you can send the image and receive what the model thinks. In the real world, you would also need a UI for that, like a mobile application or a website, but we won't need that for our thought experiment.

An important objective now is to secure the model in a way that no adversary can steal it from you. Server security when hosting some application server on the internet is a topic in and of itself [51] (like having all the latest hotfixes and patches, all data at rest encrypted using a key in HSM, etc.), and we won't go into any details in this thesis. However, there are still some risks that the adversary could try to steal the model using model extraction attacks [63].

In the "Stealing Machine Learning Models via Prediction APIs" paper, they discuss all the options the adversary can use to extract the model when running on it on some cloud provider with ready-made Prediction API. The issue here is in finding the perfect balance in having the Prediction API as wide and informative as possible while not exposing too many details about the inner workings of the hosted model. Then there is a risk of not only Inference attacks but also compromising the knowledge of the model itself.

Nonetheless, the issue is not only concerning the cloud-hosted models with rich Prediction API but the compromise of the model could be achieved with a more undemanding black box approach. In that case, you train your "knockoff network" solely on the processed outputs [46]. Imagine there is an API to which you can send a photo, and it will return the most prevalent thing in the photo (e.g. car/tree/dog/human). From this simple output, when one has a good strategy, time and computing power, one could, over time, imitate the net response with almost the original network's accuracy [46]. For example, this was done publicly in breakthrough paper *Stanford Alpaca: An Instruction-Following LLaMA Model*, where Stanford's researchers took pre-trained LLaMA model [62] and fine-tuned it using self-instruct [67] method from initially much more advanced and capable OpenAI's text-davinci-003 [12]. In simple terms, Stanford's researchers were using text-davinci-003 to generate questions for the LLaMa model and then for the evaluation of LLaMa's answers. Because of that, the model could learn really fast from the generated high-quality data even on relatively small 7B parameters LLM. However, it could be argued that it isn't creating any new knowledge, and it only leeches on big LLMs like text-davinci-003.

4.3 Differential Privacy

The term differential privacy was coined by Cynthia Dwork in 2006 [17], describing a technique for creating and maintaining privacy-preserving datasets and systems. The core principle is to add random noise to your dataset so it doesn't contain any private information. That means if for example, each row in the dataset refers to one human, after this procedure, one shouldn't be able to connect any row to the corresponding human.

This work was heavily inspired by work from Dalenius "Towards a Methodology for Statistical Disclosure Control" where he laid down foundations for privacy-preserving data science. The two key goals, which were later simultaneously solved by Dwork, were to preserve the confidentiality of the data in the dataset even if the adversary could have direct access to the dataset and remove an option to deduce any private or sensitive information from statistical outputs of the dataset (sums, averages, maximums, etc.).

In 2016, Abadi et al. published the paper "Deep Learning with Differential Privacy", which was a continuation of [16, 17, 18, 56] and introduced differentially private stochastic gradient descent (DP-SGD) with three new parameters in comparison to standard SGD:

- Noise scale σ - A real number which can be changed as part of hyperparameter tuning.
- Noise function $\mathcal{N}$ - Function which takes the noise scale σ and outputs the noise which is added to the gradient.
- Gradient norm bound C - Computed gradient for each example is clipped before adding the noise.

The actual algorithm only extends SGD optimization:

1. Go over every example x from dataset X like you would normally do in SGD
2. Calculate a gradient g_x using the loss function for a given example x
3. Clip the gradient: $g_x \,/\, max(1, \frac{L2\_norm(g_x)}{C})$
4. Add the noise to the gradient calculated by noise function $\mathcal{N}$ and configured by noise scale σ
5. The final gradient is then processed in standard SGD fashion by aggregating the gradients and applying the learning rate parameter

Clipping the gradient is there primarily to prevent outliers from substantially standing out and therefore increase the chance that the model will try to memorize the outliers [1, 70]. Nevertheless, gradient clipping is a technique also used as a standalone SGD extension, used for non-privacy reasons, as it prevents the outliers from skewing the aim of the averaged gradient [1] (it is used, for example, in the aforementioned *LLaMA: Open and Efficient Foundation Language Models*).

There are already libraries like Opacus [70], which are extending standard Python ML frameworks like PyTorch [48], with which you can easily implement differential privacy to your training using easy-to-use API like *PrivacyEngine* and only initializing it with your network and optimizer.

Chapter 5

Ready-to-use example

The goal of this work, apart from being an informative guide throughout the new field of Federated learning, is to provide a ready-to-use example on which you can easily start doing your own networks.

Because of that, we've created an example project (`https://github.com/martingeorgiu/spots_federated`) built on top of industry-standard libraries like Pytorch [48]. In addition to that, we've decided to use Pytorch Lighting [19], which serves as an optional extension/wrap of Pytorch and gives you code nice structure and removes loads of boilerplate.

For the federation, we've used Flower [7], which along with OpenFL [21], are both quite advanced libraries for federated learning with a wide range of strategies already implemented out of the box.

Both Flower and Pytorch Lightning were quite easy to extend at any point, which came in really handy when we wanted to improve and expand logging during training/testing/evaluation phases. We wanted to use TensorBoard for which there was already some implementation but we wanted to expand it. Our TensorBoard logging supports not only the final aggregated server metric for loss and accuracy, but we were able to log the individual client losses and accuracies as well.

With regard to the Universal example was the HAM10000 dataset [64] quite an easy choice as it is a fully labelled dataset with a bit over 10000 photos. There are many already working examples using typical machine learning libraries like this Jupyter notebook [72], so they can even be used as a nice starting point.

Disclaimer

The goal wasn't to make the best network possible on provided data. We've trained an average-performing model as a starting point to which we can compare the models trained using federation. After that, we created the federation using a standard strategy FedAvg[39] and manually played with some hyperparameters. The goal wasn't to make any deep comparison of strategies as that is usually included in the papers introducing the strategy in question [35, 36, 52] or there are some unbiased benchmarks from others like FLamby [61]. Rather the goal was to introduce a real example to an uninformed reader on which he could build a network of his own. The developed example is provided for free under the BSD-3 licence, and any individual is more than encouraged to try it out and work with it.

5.1 HAM10000 issues

However, it's not all sunshine and roses. There are two main issues with this dataset:

1. Skin lesions example can have multiple photos from different angles.
2. Some classes are much more over-represented than others (e.g. melanocytic nevi itself is 67% of the dataset, and there are six other classes).

The first can be resolved when creating train-validation-test data split with caution. Basically, the main possible problem that can come up is to have two separate images of the same skin lesion, one in the train and the second one in the validation dataset. Then, the objectivity of the results from the validation dataset can be skewed as we cannot be sure whether the network is learning by identifying patterns and not just memorizing the training dataset (i.e. overfitting), which is usually the main purpose of the validation dataset to spot. Zhuang in *Skin Lesion Classification(Acc>90%)(Pytorch)* did a great job of resolving that issue, so we will use his methods for splitting the datasets.

The second one, however, was a harder nut to crack.

5.2 Imbalanced dataset

One of the most serious difficulties upfront was an imbalanced dataset. There are many techniques which can tackle this issue. The easiest ones can be oversampling the underrepresented classes or, similarly, the other way around, so undersampling the overrepresented classes. In reality, it could mean using only half of the Melanocytic nevi photos and duplicate pictures of Dermatofibroma. Of course, when duplicating the pictures, there should definitely be some transformations before (like rotation, crop, colour shift, etc.), but that is needed anyway not to pass the same data in the same form every epoch.

Initially, our loss function of choice was a popular cross-entropy loss. The results from that were good, but they could be better. The main issue with the dataset was that the network could see promising results from returning a low loss value for a class with the biggest representation, as it is quite likely that it will be correct because of the dataset imbalance.

We've played with adding some weights to each class, and the results were better but still not great. The issue with adding weights to your loss function is that it emphasises some classes over another, not because they are harder to distinguish but because there are many of them. In the example we wrote, there are two predeclared ways to use these weights.

5.2.1 Proportional weights

This was calculated by picking the one class with the lowest amount of samples, and then for all classes, we divided that number by the number of samples in a given class. That way, only one class (the one with the lowest amount of representations) has 1, and all of the others have weight in the $(0, 1)$ range. However, that promoted the underrepresented values for no reason, like whether they are hard to distinguish, but only based on the amount.

5.2.2 Reduced weights

We still wanted to use the weights to some degree, as the imbalanced nature of the dataset was so substantial. Therefore we've used the Proportional weights and took every weight to the power of $\frac{1}{4}$. That way, the differences weren't as enormous as before. Originally the most represented class had a weight of 0.019 and after this reduction 0.37.

5.2.3 Alternatives for selecting weights

The main goal of this thesis isn't to create the greatest model overall for a mole analysis but rather to focus on federated learning and how it can be used for training such a model. Nonetheless, we still wanted to create a well-performing and useful model. Therefore it could be argued that the weights shouldn't be used for tackling imbalance at all but rather to give importance to specific critical classes over others.

Then the natural idea would be to prioritize (to give higher weights) classes related to serious diagnoses like cancer or pre-cancer and to give lower weights to the benign ones. This way, you are more likely to receive false positives (in this case, positive suppose to mean a serious diagnosis, and negative is a non-serious diagnosis) rather than false negatives. That means more people will go to the doctor to double-check the model prediction of some serious diagnosis rather than remaining in a false conviction that they are fine even though they might be having potentially deadly lesions.

5.2.4 Focal loss

Because of that, we've decided to use an extension of cross-entropy. It is called Focal loss [37]. The great feature of this particular loss function is its ability to focus on the hard problems, which is crucial, especially for imbalanced datasets, where you don't know in advance (because of lack of deep domain knowledge) whether some examples are harder than others [37].

In figure 5.1, you can see that by increasing the γ value, the loss value will be decreased overall as the curve bends down in the entire interval $(0, 1)$ of the probability of ground truth class. Yet the cases with a low probability of ground truth class (i.e. the hard examples to classify) will have

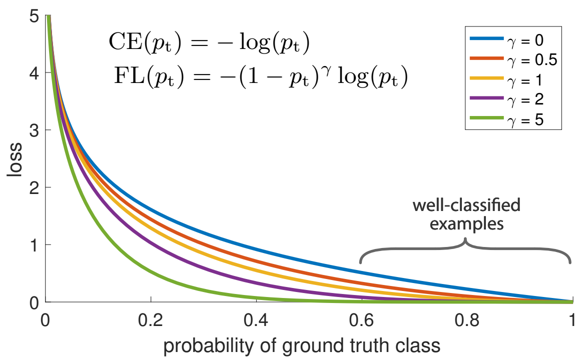

Figure 5.1: Instructive graph from the original paper about Focal loss [37]

the loss value decreased much less than the easy ones. Also note that when $\gamma = 0$, focal loss is the same function as cross-entropy.

5.2.5 Other tactics

We didn't use the following tactics in the example, but this section should give you some additional ideas for future improvements that might be used for tackling imbalanced datasets.

Undersampling

The idea is that you do not use all the examples of the over-represented class. In our case, the most over-represented class would be Dermatofibroma, so we could even use only half of all the images in each epoch. The great thing about this approach is that you don't need to use the same subset for each epoch, so you aren't losing all the data entirely. You are just using a different subset every time. This method is especially useful when working with large datasets when skipping some training data wouldn't ruin the entire training.

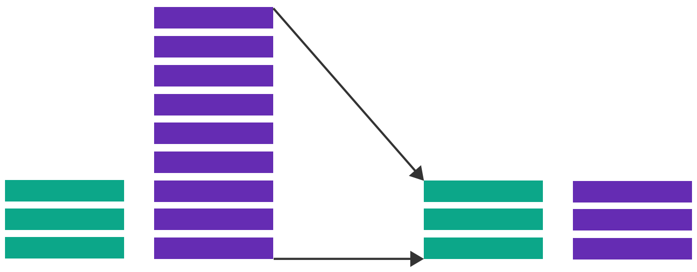

Figure 5.2: Undersampling

Oversampling

As the name suggests, this approach is a bit similar to the Undersampling, but only the other way around. Instead of reducing the over-represented, you are multiplying the underrepresented. Remember, of course, to use some random augmentation, like colour shift, random rotation or random zoom, because otherwise, the results wouldn't be that great as you are showing the model a picture that it already saw and it would be more likely to overfit the model [49]. Also, it would be a great addition if the same examples which you decide to put into the dataset more times would be the examples which are hard to classify. That will result in an increase in the overall difficulty of the problem, but it will try to guide the model in a better and more precise direction.

Generate pseudo new data

The idea here is to generate new images from an existing underrepresented dataset. One could argue that this is the same as Oversampling, but the main difference is that in oversampling,

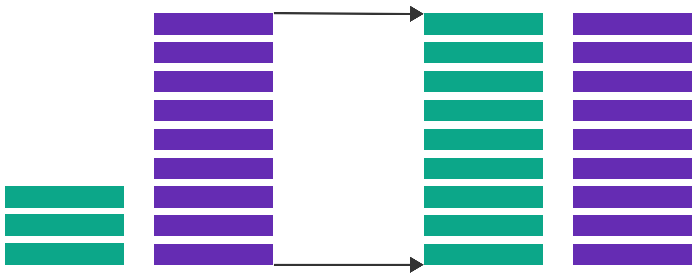

Figure 5.3: Oversampling

you are duplicating the image and only augmenting it somehow. Except here, you are using an algorithm like SMOTE [33] to generate the images using the more advanced method. For example, the already mentioned SMOTE works in the following manner: Picks an image, finds k nearest neighbour [15] of that image and moves it a small bit in the direction of the neighbour. In this way, you don't get substantially different results from what you already have in the dataset, but you'll still get some never-seen-before pictures [33].

5.3 Models

Pytorch [48] library already contains a lot of models for image classification to choose from. Because of limited resources, we couldn't use some really advanced and deep models like ResNet101 [24] or DenseNet121 [29]. Hence we've decided to use MobileNet V3 [26], which is much faster to train the other mentioned with still acceptable accuracy.

5.4 Training results

To train ANNs, you need to have powerful hardware, but if you are doing it the federated way, it is twice as true. Although you can host all the theoretical clients for the federation on one machine, which was the way we did it, it was still much more demanding than the usual training. Because of that, we've decided to test it using only three clients who were still possible to run without any significant computation slowdown on our training machine (MacBook Pro 14 2021 M1 Pro).

We ran multiple pieces of training and found an interesting comparison of the five runs selected in table 5.4. All the training was run with focal loss, and the parameter gamma was set to 2. Also, we would like to point out that when testing all the different federations locally, with more and more clients, can become tremendously demanding on computing power. Luckily Flower library provides ways for training the model federatively but only simulating the clients under the hood [7]. That means you don't have to run all the clients separately and individually, which saves a ton of computing power. The "type" in the table 5.4 specifies whether the run was using simulated or real federation, and it is clear that the time difference is enormous.

color	**federated**	**type**	**rounds**	**clients**	**weights**	**epochs per round**	**accuracy [%]**	**duration**
🟪	n	-	-	-	reduced	50	94.9	18h 47m
🟧	y	sim	100	3	reduced	1	86.1	5h 3m
🟥	y	sim	25	3	reduced	5	86.6	5h 22m
🟦	y	sim	40	3	proportional	1	85.0	1h 46m
🟩	y	real	20	3	reduced	1	82.5	5h 8m

Figure 5.4: Test runs

The first run is there to showcase the theoretical limit of the federation, as it was accomplished using standard Pytorch Lightning training. That means there were no rounds or clients or any other parameter for federated learning, just 50 epochs of standard training. The fact that it took the longest might have been caused by not properly utilizing the entire power of the training

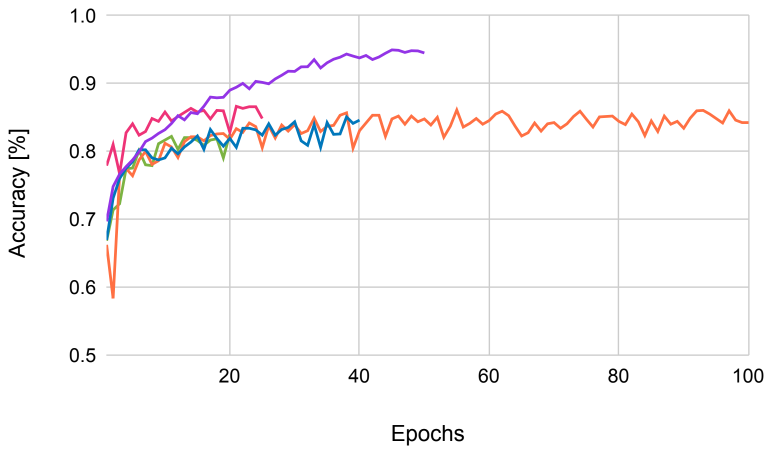

Figure 5.5: Accuracy comparison

machine and not parallelising the workload, as the CPU was only 50% busy which was significantly lower than in federated training when it was always 90% - 95%.

Subsequently, we experimented with various available parameter settings. We found out that choosing more than one epoch per round can, for our dataset and strategy, be a working tactic (as mentioned in 3.1), as both 🟧 and 🟥 took approximately 5 hours, but the accuracy of 🟥 was noticeably better with upwards trend. The 🟩 training is there to show that there isn't really a difference between using and not using the Flower simulation. Lastly, the accuracy difference between 🟧 and 🟦 is basically non-existent, which showcases that even though we choose to use substantially different weights for the loss function, the result remains almost the same.

Chapter 6

Conclusion

This thesis aimed first to introduce various concepts and techniques for federated learning. Initially, we discussed different types of privacy-preserving training methods and introduced an example to better explain these approaches. We emphasized key concepts such as IID data and cross-silo architecture, which are important for understanding the subsequent chapter, Strategies for federated learning. The chapter tried to cover all well-established strategies for federated learning like FedAvg, FedOpt and many others.

In the following chapter, we've focused on the role of security in federated learning, exploring potential attack vectors and their prevention methods. We also introduced the concept of differential privacy, a valuable technique for creating privacy-preserving datasets and machine-learning models.

Finally, we developed a fully working example with which one can easily train an ANN in a federated manner. Although the dataset was highly unbalanced, we resolved this issue using a loss function which was less affected by the imbalance. Instead of creating a novel federated learning framework, we built upon an existing popular solution Flower [7], which we've extended and implemented some features, like proper TensorBoard support or a summary log of the training. After that, we conducted multiple training sessions with varying parameters like epochs per round or class weights. From that, we've concluded that in our specific case, having five epochs per round was really beneficial, while the class weights ultimately had a minimal impact.

To develop a production-ready model, it would be better to first improve the non-federated model training by further addressing the class imbalance and spending more time comparing ANN models. We picked MobileNetV3 only based on its fast learning speed. Next, we could focus again on the federation, potentially use more advanced strategies and try to combine the best of them. For example, we could merge all strategies presented (FedProx, FedBN and FedOpt), as they each enhance the underlying FedAvg in different ways. Since all tests were concluded on a laptop, we didn't experiment with varying numbers of clients for performance reasons. However, future work could explore this aspect further using more suitable hardware for AI training. Another intriguing topic would be to investigate how the model accuracy would be affected if the class distribution across the clients would be different.

Bibliography

[1] Martin Abadi et al. "Deep Learning with Differential Privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. New York, NY, USA: Association for Computing Machinery, Oct. 24, 2016, pp. 308–318. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978318. URL: https://doi.org/10.1145/2976749.2978318 (visited on 05/07/2023).

[2] Durmus Alp Emre Acar et al. "Federated Learning Based on Dynamic Regularization". In: *International Conference on Learning Representations*. International Conference on Learning Representations. Feb. 10, 2022. URL: https://openreview.net/forum?id=B7v4QMR6Z9w (visited on 01/15/2023).

[3] Toshinori Araki et al. "Generalizing the SPDZ Compiler For Other Protocols". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. New York, NY, USA: Association for Computing Machinery, Oct. 15, 2018, pp. 880–895. ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243854. URL: https://doi.org/10.1145/3243734.3243854 (visited on 05/09/2023).

[4] Eugene Bagdasaryan et al. "How To Backdoor Federated Learning". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. International Conference on Artificial Intelligence and Statistics. PMLR, June 3, 2020, pp. 2938–2948. URL: https://proceedings.mlr.press/v108/bagdasaryan20a.html (visited on 05/09/2023).

[5] Yonatan Belinkov. "Probing Classifiers: Promises, Shortcomings, and Advances". In: *Computational Linguistics* 48.1 (Apr. 4, 2022), pp. 207–219. ISSN: 0891-2017. DOI: 10.1162/coli_a_00422. URL: https://doi.org/10.1162/coli_a_00422 (visited on 05/09/2023).

[6] Aurélien Bellet et al. "Personalized and Private Peer-to-Peer Machine Learning". In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. International Conference on Artificial Intelligence and Statistics. PMLR, Mar. 31, 2018, pp. 473–481. URL: https://proceedings.mlr.press/v84/bellet18a.html (visited on 05/09/2023).

[7] Daniel J. Beutel et al. *Flower: A Friendly Federated Learning Research Framework*. Mar. 5, 2022. DOI: 10.48550/arXiv.2007.14390. arXiv: 2007.14390 [cs, stat]. URL: http://arxiv.org/abs/2007.14390 (visited on 01/15/2023). preprint.

[8] Joseph K. Blitzstein and Jessica Hwang. *Introduction to Probability, Second Edition*. CRC Press, Feb. 8, 2019. 636 pp. ISBN: 978-0-429-76674-9. Google Books: LciHDwAAQBAJ.

[9] Keith Bonawitz et al. *Practical Secure Aggregation for Federated Learning on User-Held Data*. Nov. 14, 2016. DOI: 10.48550/arXiv.1611.04482. arXiv: 1611.04482 [cs, stat]. URL: http://arxiv.org/abs/1611.04482 (visited on 03/12/2023). preprint.

[10] Keith Bonawitz et al. "Towards Federated Learning at Scale: System Design". In: *Proceedings of Machine Learning and Systems* 1 (Apr. 15, 2019), pp. 374–388. URL: https://proceedings.mlsys.org/paper/2019/hash/bd686fd640be98efaae0091fa301e613-Abstract.html (visited on 05/09/2023).

[11] Blake Brittain and Blake Brittain. "Getty Images Lawsuit Says Stability AI Misused Photos to Train AI". In: *Reuters. Legal* (Feb. 6, 2023). URL: https://www.reuters.com/legal/getty-images-lawsuit-says-stability-ai-misused-photos-train-ai-2023-02-06/ (visited on 04/18/2023).

[12] Tom Brown et al. "Language Models Are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html (visited on 05/09/2023).

[13] Victor Ion Butoi et al. *UniverSeg: Universal Medical Image Segmentation.* Apr. 12, 2023. DOI: 10.48550/arXiv.2304.06131. arXiv: 2304.06131 [cs]. URL: http://arxiv.org/abs/2304.06131 (visited on 04/30/2023). preprint.

[14] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen (aut). *Secure Multiparty Computation.* Cambridge University Press, July 15, 2015. 385 pp. ISBN: 978-1-107-04305-3. Google Books: HpsZCgAAQBAJ.

[15] Padraig Cunningham and Sarah Jane Delany. "K-Nearest Neighbour Classifiers: 2nd Edition (with Python Examples)". In: *ACM Computing Surveys* 54.6 (July 31, 2022), pp. 1–25. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3459665. arXiv: 2004.04523 [cs, stat]. URL: http://arxiv.org/abs/2004.04523 (visited on 04/21/2023).

[16] Tore Dalenius. "Towards a Methodology for Statistical Disclosure Control". In: (1977). URL: https://ecommons.cornell.edu/handle/1813/111303 (visited on 04/17/2023).

[17] Cynthia Dwork. "Differential Privacy". In: *Automata, Languages and Programming.* Ed. by Michele Bugliesi et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 1–12. ISBN: 978-3-540-35908-1. DOI: 10.1007/11787006_1.

[18] Cynthia Dwork et al. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Theory of Cryptography.* Ed. by Shai Halevi and Tal Rabin. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 265–284. ISBN: 978-3-540-32732-5. DOI: 10.1007/11681878_14.

[19] William Falcon and The PyTorch Lightning team. *PyTorch Lightning.* Version 1.4. Mar. 2019. DOI: 10.5281/zenodo.3828935. URL: https://github.com/Lightning-AI/lightning.

[20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning.* International Conference on Machine Learning. PMLR, July 17, 2017, pp. 1126–1135. URL: https://proceedings.mlr.press/v70/finn17a.html (visited on 05/09/2023).

[21] Patrick Foley et al. "OpenFL: The Open Federated Learning Library". In: *Physics in Medicine & Biology* 67.21 (Oct. 2022), p. 214001. ISSN: 0031-9155. DOI: 10.1088/1361-6560/ac97d9. URL: https://dx.doi.org/10.1088/1361-6560/ac97d9 (visited on 01/15/2023).

[22] Mohsen Ghafoorian et al. "Transfer Learning for Domain Adaptation in MRI: Application in Brain Lesion Segmentation". In: vol. 10435. 2017, pp. 516–524. DOI: 10.1007/978-3-319-66179-7_59. arXiv: 1702.07841 [cs]. URL: http://arxiv.org/abs/1702.07841 (visited on 04/30/2023).

[23] Ian Goodfellow et al. "Generative Adversarial Networks". In: *Communications of the ACM* 63.11 (Oct. 22, 2020), pp. 139–144. ISSN: 0001-0782. DOI: 10.1145/3422622. URL: https://dl.acm.org/doi/10.1145/3422622 (visited on 05/09/2023).

[24] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[25] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. "Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* CCS '17. New York, NY, USA: Association for Computing Machinery, Oct. 30, 2017, pp. 603–618. ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3134012. URL: https://doi.org/10.1145/3133956.3134012 (visited on 04/23/2023).

[26] Andrew Howard et al. "Searching for MobileNetV3". In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV). IEEE Computer Society, Oct. 1, 2019, pp. 1314–1324. ISBN: 978-1-72814-803-8. DOI: 10.1109/ICCV.2019.00140. URL: https://www.computer.org/csdl/proceedings-article/iccv/2019/480300b314/1hVlGG4j720 (visited on 05/09/2023).

[27] Jeremy Howard and Sebastian Ruder. "Universal Language Model Fine-tuning for Text Classification". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* ACL 2018. Melbourne, Australia: Association for Computational Linguistics, June 2018, pp. 328–339. DOI: 10.18653/v1/P18-1031. URL: https://aclanthology.org/P18-1031 (visited on 05/09/2023).

[28] Krystal Hu. “ChatGPT Sets Record for Fastest-Growing User Base - Analyst Note”. In: *Reuters. Technology* (Feb. 2, 2023). URL: https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/ (visited on 04/23/2023).

[29] Gao Huang et al. “Densely Connected Convolutional Networks”. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, July 1, 2017, pp. 2261–2269. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.243. URL: https://www.computer.org/csdl/proceedings-article/cvpr/2017/0457c261/12OmNBDQbld (visited on 05/07/2023).

[30] International cyber law: interactive toolkit contributors. *Brno University Hospital Ransomware Attack (2020)*. International cyber law: interactive toolkit. June 3, 2021. URL: https://cyberlaw.ccdcoe.org/w/index.php?title=Brno_University_Hospital_ransomware_attack_(2020)&oldid=2400 (visited on 04/25/2023).

[31] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, June 6, 2015, pp. 448–456.

[32] Peter Kairouz et al. “Advances and Open Problems in Federated Learning”. In: *Foundations and Trends® in Machine Learning* 14.1–2 (June 22, 2021), pp. 1–210. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000083. URL: https://www.nowpublishers.com/article/Details/MAL-083 (visited on 05/07/2023).

[33] Joos Korstanje. *SMOTE*. Medium. Aug. 30, 2021. URL: https://towardsdatascience.com/smote-fdce2f605729 (visited on 03/01/2023).

[34] Tian Li et al. “Federated Learning: Challenges, Methods, and Future Directions”. In: *IEEE Signal Processing Magazine* 37.3 (May 2020), pp. 50–60. ISSN: 1053-5888, 1558-0792. DOI: 10.1109/MSP.2020.2975749. arXiv: 1908.07873 [cs, stat]. URL: http://arxiv.org/abs/1908.07873 (visited on 01/15/2023).

[35] Tian Li et al. “Federated Optimization in Heterogeneous Networks”. In: *Proceedings of Machine Learning and Systems* 2 (Mar. 15, 2020), pp. 429–450. URL: https://proceedings.mlsys.org/paper/2020/hash/38af86134b65d0f10fe33d30dd76442e-Abstract.html (visited on 05/07/2023).

[36] Xiaoxiao Li et al. “FedBN: Federated Learning on Non-IID Features via Local Batch Normalization”. In: International Conference on Learning Representations. Jan. 12, 2021. URL: https://openreview.net/forum?id=6YEQUn0QICG (visited on 05/07/2023).

[37] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *Focal Loss for Dense Object Detection*. Proceedings of the IEEE International Conference on Computer Vision. 2017, pp. 2980–2988. URL: https://openaccess.thecvf.com/content_iccv_2017/html/Lin_Focal_Loss_for_ICCV_2017_paper.html (visited on 02/20/2023).

[38] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. Sept. 17, 2020. DOI: 10.48550/arXiv.1802.03426. arXiv: 1802.03426 [cs, stat]. URL: http://arxiv.org/abs/1802.03426 (visited on 04/23/2023). preprint.

[39] Brendan McMahan et al. “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Artificial Intelligence and Statistics. PMLR, Apr. 10, 2017, pp. 1273–1282. URL: https://proceedings.mlr.press/v54/mcmahan17a.html (visited on 05/07/2023).

[40] Luca Melis et al. “Exploiting Unintended Feature Leakage in Collaborative Learning”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019 IEEE Symposium on Security and Privacy (SP). May 2019, pp. 691–706. DOI: 10.1109/SP.2019.00029.

[41] Andrew Miller et al. “The Honey Badger of BFT Protocols”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. New York, NY, USA: Association for Computing Machinery, Oct. 24, 2016, pp. 31–42. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978399. URL: https://doi.org/10.1145/2976749.2978399 (visited on 04/30/2023).

[42] Payman Mohassel and Yupeng Zhang. “SecureML: A System for Scalable Privacy-Preserving Machine Learning”. In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017 IEEE Symposium on Security and Privacy (SP). May 2017, pp. 19–38. DOI: 10.1109/SP.2017.12.

[43] Leo N. *Google Staged Rollouts & App Store Phased Release: With Less Stress.* Geek Culture. Nov. 13, 2022. URL: https://medium.com/geekculture/google-staged-rollouts-app-store-phased-release-with-less-stress-1d51dffde7a7 (visited on 04/30/2023).

[44] Milad Nasr, Reza Shokri, and Amir Houmansadr. "Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning". In: 2019 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, May 1, 2019, pp. 739–753. ISBN: 978-1-5386-6660-9. DOI: 10.1109/SP.2019.00065. URL: https://www.computer.org/csdl/proceedings-article/sp/2019/666000a739/1dlwhtj4r7O (visited on 05/07/2023).

[45] OpenAI. *Introducing ChatGPT.* URL: https://openai.com/blog/chatgpt (visited on 04/23/2023).

[46] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. "Knockoff Nets: Stealing Functionality of Black-Box Models". In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019, pp. 4954–4963. URL: https://openaccess.thecvf.com/content_CVPR_2019/html/Orekondy_Knockoff_Nets_Stealing_Functionality_of_Black-Box_Models_CVPR_2019_paper.html (visited on 03/25/2023).

[47] Wonpyo Park et al. "Relational Knowledge Distillation". In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019, pp. 3967–3976. URL: https://openaccess.thecvf.com/content_CVPR_2019/html/Park_Relational_Knowledge_Distillation_CVPR_2019_paper.html (visited on 05/09/2023).

[48] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems.* 721. Red Hook, NY, USA: Curran Associates Inc., Dec. 8, 2019, pp. 8026–8037.

[49] Luis Perez and Jason Wang. *The Effectiveness of Data Augmentation in Image Classification Using Deep Learning.* Dec. 13, 2017. DOI: 10.48550/arXiv.1712.04621. arXiv: 1712.04621 [cs]. URL: http://arxiv.org/abs/1712.04621 (visited on 04/21/2023). preprint.

[50] Ramesh Raskar. *Split Learning Project: MIT Media Lab.* URL: https://splitlearning.mit.edu/ (visited on 04/29/2023).

[51] Vamsi Ravula. *10 Essentials to Mitigating API Security Risks.* Red Hat Developer. Oct. 20, 2022. URL: https://developers.redhat.com/articles/2022/10/20/10-essentials-mitigating-api-security-risks (visited on 04/16/2023).

[52] Sashank J. Reddi et al. "Adaptive Federated Optimization". In: International Conference on Learning Representations. Jan. 12, 2021. URL: https://openreview.net/forum?id=LkFG3lB13U5 (visited on 05/07/2023).

[53] Michael Ridley. "Explainable Artificial Intelligence (XAI):" in: *Information Technology and Libraries* 41.2 (2 June 15, 2022). ISSN: 2163-5226. DOI: 10.6017/ital.v41i2.14683. URL: https://ejournals.bc.edu/index.php/ital/article/view/14683 (visited on 04/23/2023).

[54] Neta Shoham et al. *Overcoming Forgetting in Federated Learning on Non-IID Data.* Oct. 17, 2019. arXiv: 1910.07796 [cs, stat]. URL: http://arxiv.org/abs/1910.07796 (visited on 03/06/2023). preprint.

[55] Reza Shokri et al. "Membership Inference Attacks Against Machine Learning Models". In: *2017 IEEE Symposium on Security and Privacy (SP).* 2017 IEEE Symposium on Security and Privacy (SP). May 2017, pp. 3–18. DOI: 10.1109/SP.2017.41.

[56] Shuang Song, Kamalika Chaudhuri, and Anand D. Sarwate. "Stochastic Gradient Descent with Differentially Private Updates". In: *2013 IEEE Global Conference on Signal and Information Processing.* 2013 IEEE Global Conference on Signal and Information Processing (GlobalSIP). Austin, TX, USA: IEEE, Dec. 2013, pp. 245–248. ISBN: 978-1-4799-0248-4. DOI: 10.1109/GlobalSIP.2013.6736861. URL: http://ieeexplore.ieee.org/document/6736861/ (visited on 04/18/2023).

[57] Gautam Srivastava et al. *XAI for Cybersecurity: State of the Art, Challenges, Open Issues and Future Directions.* June 2, 2022. DOI: 10.48550/arXiv.2206.03585. arXiv: 2206.03585 [cs]. URL: http://arxiv.org/abs/2206.03585 (visited on 04/23/2023). preprint.

[58] Chen Sun et al. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017 IEEE International Conference on Computer Vision (ICCV). Oct. 2017, pp. 843–852. DOI: 10.1109/ICCV.2017.97.

[59] Hanlin Tang et al. "D^2: Decentralized Training over Decentralized Data". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 10–15, 2018, pp. 4848–4856. URL: https://proceedings.mlr.press/v80/tang18a.html.

[60] Rohan Taori et al. *Stanford Alpaca: An Instruction-Following LLaMA Model*. 2023. URL: https://github.com/tatsu-lab/stanford_alpaca.

[61] Jean Ogier du Terrail et al. "FLamby: Datasets and Benchmarks for Cross-Silo Federated Learning in Realistic Healthcare Settings". Version 2. In: (2022). DOI: 10.48550/ARXIV.2210.04620. URL: https://arxiv.org/abs/2210.04620 (visited on 04/19/2023).

[62] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. Feb. 27, 2023. DOI: 10.48550/arXiv.2302.13971. arXiv: 2302.13971 [cs]. URL: http://arxiv.org/abs/2302.13971 (visited on 04/18/2023). preprint.

[63] Florian Tramèr et al. "Stealing Machine Learning Models via Prediction APIs". In: *Proceedings of the 25th USENIX Conference on Security Symposium*. SEC'16. USA: USENIX Association, Aug. 10, 2016, pp. 601–618. ISBN: 978-1-931971-32-4.

[64] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. "The HAM10000 Dataset, a Large Collection of Multi-Source Dermatoscopic Images of Common Pigmented Skin Lesions". In: *Scientific Data* 5.1 (1 Aug. 14, 2018), p. 180161. ISSN: 2052-4463. DOI: 10.1038/sdata.2018.161. URL: https://www.nature.com/articles/sdata2018161 (visited on 05/09/2023).

[65] Praneeth Vepakomma et al. "NoPeek: Information Leakage Reduction to Share Activations in Distributed Deep Learning". In: 2020 International Conference on Data Mining Workshops (ICDMW). IEEE Computer Society, Nov. 1, 2020, pp. 933–942. ISBN: 978-1-72819-012-9. DOI: 10.1109/ICDMW51313.2020.00134. URL: https://www.computer.org/csdl/proceedings-article/icdmw/2020/901200a933/1rgGjECfp3W (visited on 05/09/2023).

[66] Praneeth Vepakomma et al. *Split Learning for Health: Distributed Deep Learning without Sharing Raw Patient Data*. Dec. 3, 2018. DOI: 10.48550/arXiv.1812.00564. arXiv: 1812.00564 [cs, stat]. URL: http://arxiv.org/abs/1812.00564 (visited on 04/29/2023). preprint.

[67] Yizhong Wang et al. *Self-Instruct: Aligning Language Model with Self Generated Instructions*. Dec. 20, 2022. DOI: 10.48550/arXiv.2212.10560. arXiv: 2212.10560 [cs]. URL: http://arxiv.org/abs/2212.10560 (visited on 04/18/2023). preprint.

[68] Andrew Chi-Chih Yao. "How to Generate and Exchange Secrets". In: *27th Annual Symposium on Foundations of Computer Science (Sfcs 1986)*. 27th Annual Symposium on Foundations of Computer Science (Sfcs 1986). Oct. 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25.

[69] Xun Yi, Russell Paulet, and Elisa Bertino. "Homomorphic Encryption". In: *Homomorphic Encryption and Applications*. Ed. by Xun Yi, Russell Paulet, and Elisa Bertino. SpringerBriefs in Computer Science. Cham: Springer International Publishing, 2014, pp. 27–46. ISBN: 978-3-319-12229-8. DOI: 10.1007/978-3-319-12229-8_2. URL: https://doi.org/10.1007/978-3-319-12229-8_2 (visited on 04/30/2023).

[70] Ashkan Yousefpour et al. *Opacus: User-Friendly Differential Privacy Library in PyTorch*. Aug. 22, 2022. DOI: 10.48550/arXiv.2109.12298. arXiv: 2109.12298 [cs]. URL: http://arxiv.org/abs/2109.12298 (visited on 04/18/2023). preprint.

[71] Chen Yu et al. "Distributed Learning over Unreliable Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, May 24, 2019, pp. 7202–7212. URL: https://proceedings.mlr.press/v97/yu19f.html (visited on 05/07/2023).

[72] XINRUI ZHUANG. *Skin Lesion Classification(Acc>90%)(Pytorch)*. May 2019. URL: https://kaggle.com/code/xinruizhuang/skin-lesion-classification-acc-90-pytorch (visited on 02/19/2023).