Politecnico di Torino

POLITECNICO DI TORINO

MSc's Degree in Data Science and Engineering
A.y. 2020/2021
November/December 2021

Optimization of Convolutional Neural Networks Training for Federated Learning on Embedded Systems

Supervisors

Prof. Andrea CALIMERA

Ph.D. Valentino PELUSO

Candidate Erich MALAN

Abstract

The advancement of low power technologies and the improvement of wireless communication systems and infrastructures, fueled the Internet of Things (IoT), enabling the proliferation of connected sensors able to collect and transmit data over the internet. Meanwhile, thanks to the recent breakthroughs in Artificial Intelligence (AI), Convolutional Neural Networks in particular, computers can learn trends from the collected data and extract meaningful insights to make decisions autonomously. IoT and AI are twin pillars of a new revolution: the Artificial Intelligence of Things (AIoT).

This revolution poses several opportunities and challenges. The availability of large-scale datasets generated by pervasive networks of sensors enabled the development of AI models achieving unprecedented accuracy in many domains, e.g., computer vision and natural language understanding. At the same time, the creation and management of centralized datasets are raising several privacy and security concerns. For example, security cameras, smart speakers, and smartphones collect sensitive information that users might be unwilling to share with service providers. Therefore, the major challenge today is to develop a new class of learning strategies that enable to process data locally, i.e., on embedded systems at the edge of the IoT.

This is the goal of Federated Learning (FL), the target of this work, an emerging learning paradigm where the model training is distributed over a connected fleet of devices. Each device uses its own data to train a local version of the model; periodically, the devices send their model versions (instead of the data) to a centralized server; here, the local versions are aggregated in a global model, that is sent back to the edge devices. Although previous research demonstrated that training from decentralized data is feasible, at least for what concerns the model accuracy, how to manage the limited energy resources of the IoT end-nodes remains an open problem. Indeed, FL involves frequent upload and download of models to achieve competitive accuracy, introducing a massive communication overhead, which shortens the battery life of the edge devices.

This work introduces an optimized training strategy that gradually reduces the number of model parameters that are trained and synchronized with the global model. Experimental results collected on standard datasets demonstrated that the benefits of the proposed strategy are twofold: (i) substantially reduce the communication overhead at the same accuracy, with savings ranging from 14% to 59% (depending on the accuracy levels and on the dataset) with respect to a standard FL; (ii) increase the number of model updates at the same communication cost, thus improving the model accuracy, up to +2.5%.

Acknowledgements

I would like to thank Prof. Calimera, who gave me the golden opportunity of researching the ultimate technological frontier, along with his help and directive.

I would also like to express my immense gratitude to V. Peluso Ph.D., for his guidance and support, as expression of countless suggestions and encouragements.

In the end, thanks to my parents, my family, and my friends for their suggestions and support.

I acknowledge that this work has no conflict of interest.

" Es ist nicht unsere Aufgabe, einander näher zu kommen, so wenig wie Sonne und Mond zueinander kommen, oder Meer und Land. Unser Ziel ist, einander zu erkennen und einer im anderen das zu sehen und ehren zu lernen, was er ist: des anderen Gegenstück und Ergänzung."

H. Hesse

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 General overview

Technological shift The data production increases with such rapidity that the technology evolution of servers' hardware components will not be able to match in a few decades. The main issues arising in a big data environment are due to computation and communication bottlenecks, while the burdened system must account for and deal with its heterogeneity. The rapidly growing performances of end-user devices such as smartphones, IoT-driven market directions, and the more and more shaped and restricting privacy laws require a system design different from the classical centralization of data and computational cores. The computation performed by the system is more and more shifted away towards the edge devices. The edge devices have been initially conceived just to be interfaced with reality, by sensing and transmitting signals, receiving and actuating third parties actions. The latency provided required them to perform at least partially the inference on the spot, which then evolved in local training. A shift that in the last few years headed towards federated learning with its many variants.

Deep learning on Mobile Deep learning is one of the hottest topics since the beginning of the third millennium, however, its mobile branch comes later, relocating the research efforts on efficiency rather than fully-accuracy driven optimization. Many recent works consider the exploitation of mobile devices for many different aspects. The creation of shallow models and mobile efficient models [1] allowed initially the inference directly over mobile devices. More optimization techniques have been introduced to allow more and more devices, such as Micro Controller Units (MCUs) to be usable, varying from model quantization (a technique that reduces the precision of parameters) to model compression (reducing the number of parameters).

Distributed vs Federated Learning Besides the definition of edge devices, this work constitutes a schematic definition of Federated learning by comparing it to the Distributed Learning paradigm, stating the common grounds and the differences. It is simple to think of distributed learning as the middle point that divides the basic classical deep learning (Single dataset over a single machine with a single GPU) in opposition to Federated learning. The main distinction points between distributed and federated learning are:

- **privacy by design** Privacy by design is typical and critical of FL. The pivotal theme of FL different types is that raw data is never transmitted across the internet, it is so processed in the same generating and storing device. Despite the data decentralization privacy is not always ensured, techniques as homomorphic encryption and differential privacy are proposed in particular for vertical federated learning where gradients and features are sent over the network. FL enables data processing locally, so it enables tasks that could be not allowed by the law for the classical paradigm [2].

- **performances: accuracy vs time** Decentralization provides benefits of energy and time consumption compared to classical training. However, the drawback lies of course in the accuracy performance, decreased by decentralized training and more importantly by federated learning. The time performance is instead shown to be better for both distributed and federated, however, distributed is still faster than federated. [2] .

- **System complexity** Even if in [2] the other different milestone is considered to be the aggregation of models instead of gradients, it can be seen in 3 that this distinction is more blurred. The distinction seems to be more clear for the system complexity, FL is usually identified by the geographical scattering of nodes, the data is generated and maintained locally producing the side effect of heterogeneity, an issue that still presents a challenge for agnostic FL.

1.2 Contribution

1.2.1 Descriptive contribution

A summarizing effort has been made to provide clarity through a notions scheme for FL, pointing out the principal types and definitions which are much broader than the one addressed by the proposed work. Along with the definition of the structural heterogeneity are then proposed the main challenges of FL in particular for the macro area addressed.

Federated Learning The FL definition provided is a merging attempt of the most referenced contextual literature, which unfortunately is not always coherent. The most extensive publication is "Advances and Open Problems in Federated Learning" [3] which with the strength of most of the most important FL publishers presents the challenges and future directions also considering the specific topics concerned. Many other surveys or topic related papers such as [4], [5], [6], [7],[8],[9], [10],[11] show differences towards FL and heterogeneity definitions. The work proposed merges together and balances the related works by proposing a distinction based on:

- **Infrastructure** The infrastructure difference focuses on the physical attributes of the participating components, such as servers, devices, and communication links. It is related to their cardinality and capability denoted by the hardware characteristics. It is more common to consider a single cloud server (Distributed learning), many servers (Cross Silo), or lots of devices (Cross-Device).

- **Architecture** The architecture is to be intended about the data split. Different data partitioning needs different processing and systems, with relative issues and challenges. In particular, the different partitioning could be horizontal based on examples, vertical based on features, or transfer which is a mix of both.

- **Topology** Following the proposal of [12], a new definition is presented: the underlying topology, more specific and more focused on the topological aspect, considering the graphs types related to the most interesting scenario: the star graph of the centralized topology.

Heterogeneity The heterogeneity is an unavoidable characteristic of FL, it severely impacts the design and the performance. It is such an important topic that some works such as [8][9] focus their efforts on estimating and defining the sole heterogeneity. The heterogeneity and the FL paradigm are not independent, in fact, each definition of FL (collapsed over the previously schematic definitions) has its type and degree of heterogeneity. The heterogeneity more considered is towards the data, its local generation and storage could present skew over example amounts, labels, and attributes. The data distribution hardly is homogeneous in decentralized settings, instead, it presents most likely highly skewed trends (heterogeneous). Moreover, data could be non-independent, because of the system design (e.g. pictures sampled from a video). Other types of heterogeneity are agglomerated into the system one, varying from hardware, state, and performances of the participating clients to the law enforcement that they might be liable to.

1.2.2 Empirical contribution

The novel methodology refers to the traditional FL, which is mostly referred considering for communication issues, optimizations, and convergence under heterogeneous settings. Accordingly to the definitions provided it is characterized by the centralized topology, cross-device infrastructure with horizontal architecture, based on multiple synchronous updates. The challenges of traditional FL are today focused on communication, the primary sink of energy. However, the communication must be optimized considering the other challenges: heterogeneity, privacy, and accuracy. The empirical work proposed is rooted in the traditional FL algorithm [13], and outperforms it in terms of communication efficiency, while maintaining the same accuracy levels without loss of privacy. The communication efficiency is introduced for layer-architectured deep models by a new methodology: FedNILO (Narrowing Iteratively the Layer Optimization in Federated Learning).

FedNILO The key concept of the proposed method is to limit the model updates and computation to specific layers, which are reduced in terms of cardinality over time. Classical transfer learning techniques and network similarity studies depict that the shallow layers converge faster and tend to be similar even between different datasets that share the same task (e.g. computer vision classification). Instead, the deeper the layers are, the more computation is required to obtain their convergence. FedNILO takes advantage of the concept and limits the number of iteration of each layer depending on its architectural position. The parameters of these layers are then considered to be frozen, they are not locally trained anymore and nor are synchronized with the server. Two hyperparameters K, F are introduced to regulate the frozen or unfrozen state of the network layers. FedNILO has empirically shown to outperform traditional FL in terms of communication, narrowing harshly the burden of single devices over single rounds, yet achieving competitive accuracy with a substantial reduction of cumulative communication.

Search Space FedNILO provides communication savings by introducing a system complexity, which is brought by the addition of two variables K and F. The two hyperparameters determine when to start, and how frequently toggle the next step of optimization. They could be intended like two knobs: the first one considers more the target accuracy to be reached (in a real-world case the convergence needed is estimated through the algorithm convergence settings), the second one is more fine-grained and allows better savings for small rounds. Experimental settings consider a grid search of $K \in \{350,400,450,500\}$ and $F \in \{25,50,75\}$. Different hyperparameters choices could allow more savings under certain assumptions, low values of both K and F for lower target accuracies tend to be particularly efficient in terms of communication, while to reach higher final accuracy levels, they must

be increased.

Metrics The performances of the employed methodology are determined by the accuracy and the communication cost. The average accuracy measured over 30 rounds (ACC.) determines the capability of the trained model to correctly classify the test target labels, it is inspired by a common ground baseline [14]. The communication cost (CC) measures the communication effort of the system to reach a target ACC., it is measured both in communication rounds (CR) and in Gigabytes (GB).

1.3 Work Organization

The ensuing work organization follows a simple outline, from a general description gradually focused on the target topic (communication efficiency for cross-device horizontal federated learning with a centralized topology), the challenges, related works, and state of art and then introducing the innovative solution FedNILO, its methodology and relative results with the experimental setup to reproduce them.

Given the many recent technological innovations related to the work, the very first chapter 2 provides a general overview of the technological context, rather than directly addressing federated learning (which is extensively treated in its chapter) deals with the technological scenarios and systems that surround it.

In chapter 3 will be presented a structural definition of Federated Learning, by reporting the gold standard definition and then schematically defining the main characteristics that could define a particular context. The abstract partitioning is based on infrastructure (devices involved), architecture (data partitioning requires different system functions), and topology (how nodes are connected).

A specific section 3.5 deals with the heterogeneity types that could occur (and typically occur) in an agnostic federated learning scenario, where data, hardware, and systems, in general, could be highly skewed. 4 is an overview of the most relevant related works, the practical concept, and the relative savings, starting from the baseline [13] and its relative optimization techniques.

The novel approach is depicted in chapter 5, focusing on the algorithm and its functioning procedures over the server or client sides. The chapter discloses the details of the implementation, and it provides purposeful insights into the most important steps along with the policy and their motivations. It follows chapter 6, which is composed of two parts (i) the experimental setup needed to perform the experiments, comprehending the research scenario, models, and hyperparameters, and (ii) the experimental results obtained with the definition of relative metrics used. In the end, 7 is a quick recap, which briefly resumes the presented work

the theoretical notions provided, and the experiment performed with the results accomplished.

Chapter 2

Background

2.1 General context

This chapter presents the surrounding context motivating FL, referring to the broader scenario and presenting the standards of the common terms that are operated throughout the work. It provides the standard definitions of IoT and its relative actors. Besides, it refers to the technological breakthroughs provided by AI to introduce their energy-driven optimizations that enabled the AIoT and inherent notions.

2.1.1 IoT and AI

The International Telecommunication Union (ITU) provided the standard definition of IoT as "A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" [15] [1]. An extensive network of devices [2] that merges or at least mutually interfaces physical and cyber worlds, deeply revolutionizing both the scope of the internet and the role played by actors. Fridges, fitness trackers, traffic lights, and more are pushed toward their interconnection, constituting the network depicted in figure 2.1 which

[1] "NOTE 1 – Through the exploitation of identification, data capture, processing and communication capabilities, the IoT makes full use of things to offer services to all kinds of applications, whilst ensuring that security and privacy requirements are fulfilled.
NOTE 2 – From a broader perspective, the IoT can be perceived as a vision with technological and societal implications. " [15]

[2] Device : "With regard to the Internet of things, this is a piece of equipment with the mandatory capabilities of communication and the optional capabilities of sensing, actuation, data capture, data storage and data processing" [15].

is divided by hierarchical levels: The centralized cloud server, the edge nodes, and the edge devices [3] [4]. The monitoring part is performed by sensors [5], while the action decided by the AI is performed by actuators [6]

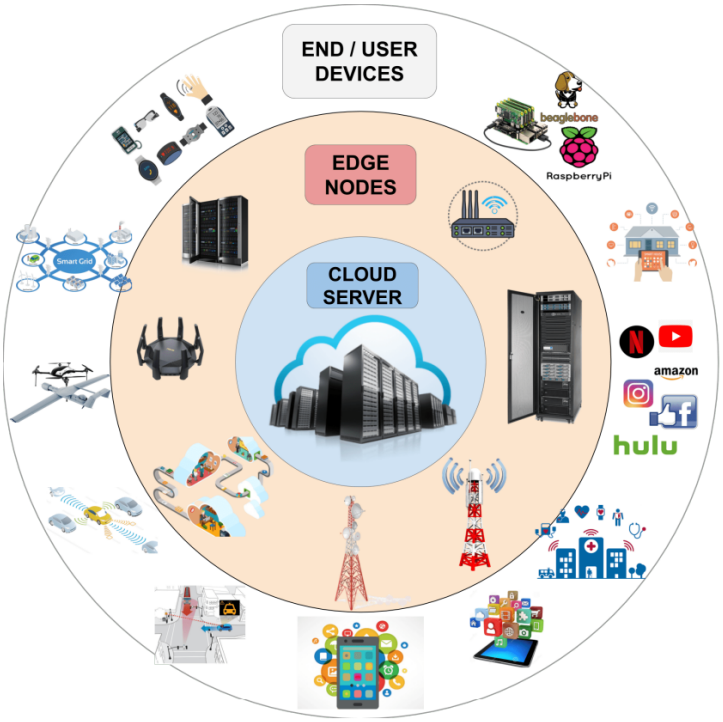

Figure 2.1: Levels of network hierarchy in IoT adapted from [17]

Initially conceived as a centralized system, IoT stumbles into the Big Data issue, where immense volumes of highly heterogeneous data need instant or rapid processing to extract information and/or make decisions. The description of big data is tight by the 3 Vs: Volume, Velocity, and Variety [5] [4]. A boundless number of devices produces endless data that must be sent over the internet to be

[3]Edge in this context refers to any computing and network resource along the path between data sources and cloud data centers

[4]"Device: A technical, physical component (hardware) with communication capabilities linking it to other IT systems. A device can be either attached to or embedded inside a physical entity or monitor a physical entity in its vicinity." [16]

[5]"Sensor: A device identifying or recording features of a given physical entity." [16]. E.g., camera or microphone.

[6]"Actuator: Mechanical device for moving or controlling a mechanism or system. It takes energy, usually transported by air, electric current or liquid, and converts it into a state change, thus affecting one or more physical entities." [16] E.g. Electrical Switch.

processed, overcrowding communication links, computational nodes and introducing dramatic complexity when dealing with its heterogeneity (Variety). Distributed learning paradigms have been developed to substantially reduce the time required by training, the centralized data is distributed across different nodes, that pertain to the cloud server, to achieve the same goal in a synchronized way exploiting multithreading and parallel computation [5] [18].

AI needs huge tons of data to improve implemented models, in particular, its subcategory defined as Deep Learning is particularly greedy of data examples [5]. The IoT data generation pointed out before comes to be double ending: while drastically undermining the execution, it could significantly improve AI which empowers the performances of IoT. Deep learning models introduced unprecedented performances especially in computer vision (CV) and natural language understanding (NLU), with brain-like inspired mathematical models. The most famous architectures depicted in 2.2 comes from the Multi-Layer Perceptron (MLP) [figure a, left], which then inspired Convolutional Neural Networks (CNNs) [figure b, middle] for the CV tasks and Recurrent Neural Networks (RNNs) [figure c, right] or Long-Short Term Memory cells (LSTMs) particularly suited for NLU due to their memory oriented design. The CNNs shown to outperform every other state-of-the-art technique, in particular in their capacity to extract and spot complicated and hierarchical patterns compared to the finest handmade features [19]. CNN's evolved along with the capacity and computational capabilities of GPUs and dedicated hardware, starting with a rush focused on the accuracy performances which produced ever more complex models, increasing the number of parameters and/or the number of operations, from AlexNet to ResNet152.

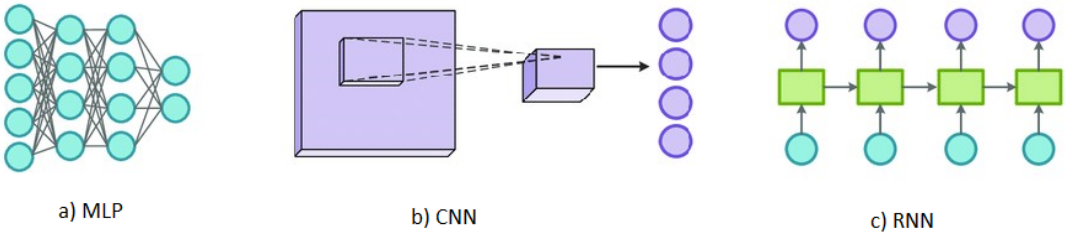

Figure 2.2: Neural networks, MLP, CNN and RNN image readapted from [20]

2.1.2 Towards AIoT

The mutual empowering of IoT and AI, however, is hampered by many challenges. The improvements of data quality and amounts weigh ever more on the network links and low-end devices. Their technological trend lead to unfeasible communication costs, but also processing and memory unavailable resources at the central server

[21]. The resources unavailability could produce single tasks failures, introduce system latency, or in extreme circumstances system breakdowns. The chain of action that links a sensor to an actuator laying in the same place is composed of the meddled components such as edge nodes and cloud-server. Their connection is bidirectional: (i) forward from device to cloud-server, through edge nodes, and (ii) back to the device again. The latency introduced by this paradigm gives infeasible operational times. E.g., a camera in a train station that captures real-time videos for critical security reasons, sends the frames to the local device, which forwards them to the cloud server. The centralized server queues their computation and provides the result (e.g., risk/not risk) that must be sent back to the device that must order the actuator to cut the power down. Real-time predictions are essential, while the centralized computation paradigm presents too much latency and failure points.

Edge intelligence: Inference Fortunately, also the evolution curves of low-end devices have seen a positive boost during the last decade. It enabled deep models on cutting-edge capabilities, meanwhile clearing the way of research efforts towards efficiency. The aim shifts so to preserve as much as possible accuracy while drastically diminishing the required resources in terms of memory, energy, and power [22]. These optimization techniques allow inference tasks to be directly carried out by the end devices involved, without the strict necessity to employ the server. New distributed systems emerged relying on the inference capabilities of low-end devices such as the Big/Small model predictions and the Ensemble Learning. The first (Big/Small) provides a server that is asked to substitute the edge devices only when the accuracy confidence is under a certain threshold. Ensemble learning, instead, employs a fleet of devices to perform their predictions and then aggregates them by mean of some decision function (e.g., majority voting) the prediction probabilities to obtain the consensus target.

Edge intelligence: Learning Inference over the edge is yet not enough. The communication burden is not completely resolved, and most of all privacy legacies undermine the feasibility of the task. The decentralization of data and its processing is thus following the inferential computation distribution. However, training requires more resources than inference, because an additional step (back-propagation) must be performed to optimize the model. The memory need is more than doubled because of the computation of the gradients, while the data batch size provides different results and is less arbitrary. But most importantly, is the data present or generated onto the device sufficient to train a Deep Model? As mentioned before, differently from other machine learning algorithms, deep models need substantial training examples. Besides, their performances suffer a lot because of class unbalancing being neural networks prone to overfitting. The training

performed purely locally is then prone to divergence, and it does not appeal to fairness, melting down the interconnection and prospects of IoT. The path of AIoT goes down towards the decentralization of data and computation, to reach efficiency and privacy goals not by segregating nodes but by building mutual empowering connections between them.

Chapter 3

Federated Learning

This chapter provides an extensive overview of FL and its characteristics. It starts with the gold standard definition and then proposes a schematic taxonomy defining the core types. Then it specifies the infrastructure, the architecture, and the possible topologies that have been deployed as particular cases or enhancements of traditional centralized FL. It furtherly exposes the meaning of heterogeneity, focusing on the data heterogeneity that severely impacts the training performance. It concludes with the challenges to motivate the solution provided.

3.1 General view

3.1.1 Gold standard definition

The broader scenario addresses the actors involved with terms such as Trainers and Coordinator so that it complies with the broader definition of FL [12]. The prevalent paradigm is indeed the client-server one, but it is not the only one available. However, to smooth the introduction of the theme, this work starts from the trivial paradigm and then explores the possible variances.

The gold standard definition provided by [3] asserts: "Federated learning (FL) is a machine learning setting where many clients (e.g., mobile devices or whole organizations) collaboratively train a model under the orchestration of a central server (e.g., service provider) while keeping the training data decentralized."

Accordingly to the leading definition, the coordinator (central server) orchestrates the updates and aggregations (parameters synchronization) to achieve a global learning objective.
The figure 3.1 points the crucial steps of a traditional FL round (R) out, which are pointedly defined below.

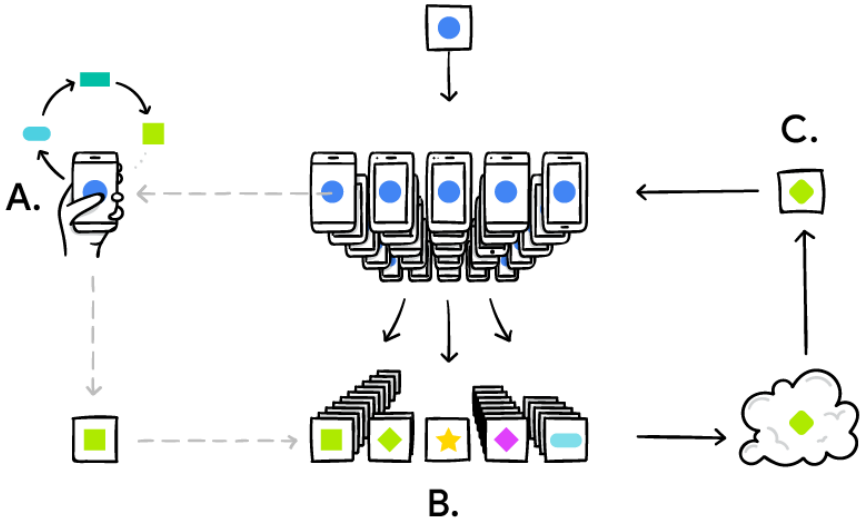

Figure 3.1: Federated learning graphical representation provided by Google [23]

- **client identification & selection** Even though the pool of clients could be composed by every device, the clients' availability is not always granted. The server could apply a further selection of the available clients based on some decision function which typically addresses the performances of the client given the connection stability, the battery life, the hardware capability as well as the quality of the model previously submitted (toxic nodes and adversarial attackers)

- **Broadcast** (C. - A.) The coordinator broadcasts the model parameters to the end nodes. The clients could overwrite the new parameters over the local model or modify it accordingly.

- **Client training / computation** (A.) The end nodes perform the training. For example, they train the model over the local data through multiple stochastic gradient descent (SGD) steps.

- **Upload** (A. - B.) The end nodes upload/push the trained parameters of the model back to the coordinator, along with optional statistics or pieces of information (for example, the number of data examples)

- **Sampling and aggregation** (B.) Once the coordinator has collected a sufficient number of updates, it exploits an agglomeration algorithm to merge them. Traditional FL performs the weighted average of the parameters. The

weighting regards the data examples that clients used during the training. Stragglers and toxic updates could be discarded depending on the policy.

- **Coordinator model update** (B. - C.) The coordinator updates the global model with the aggregation of the new parameters received. There are different aggregating techniques, the most common one is overwriting it with the weighted average (FedAVG).

3.2 Infrastructure

Distributed learning should take into account three main scenarios where first of all denote the main difference between distributed (Datacenter) vs federated (cross silo / cross device) learning. In each case the computational part is distributed over more machines, but generally FL implies the local generation of heterogeneous decentralized data displaced over a geographical distribution, while the data is never sent across the internet or shared between parties.

Datacenter - (Distributed Learning) Cisco defines the public cloud (server cloud) as a collection of data centers that supply data resources generally fully accessible (depicted in figure 3.2). The distribution of data must not be intended as decentralized. It is more like a virtual distribution over different machines, allowing aggregation, redistribution, and sharing depending on the need. The devices involved are computational nodes usually inside the same computation center, e.g., a cluster of servers or machines supposed to be always available.

The data is centralized, so the distributed learning is performed on homogeneous partitions, while the system architecture provides reliable links and fast connections. The main bottleneck comes to be computation. The datacenter paradigm shows to be strengthened by the model performances, nodes' reliability, and communication cost at least during the training phase. However, it shows to be a weak approach considering the privacy, the feasibility, or the communication cost for what concerns the massive data gathering [3].

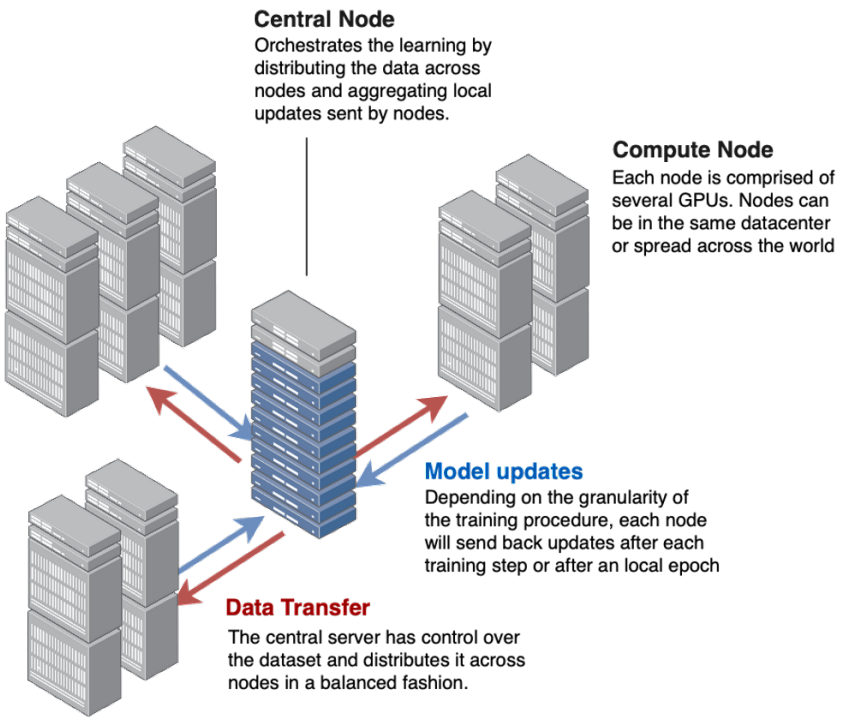

Figure 3.2: Datacenter Distributed Learning - picture adapted from flower documentation [24]

Cross-Silo (FL) Particularly interesting for organizations such as governments or private companies is the Cross-Silo FL. The 'Silo' term comes from siloed data, distributed over different geographically distributed data centers. The nodes involved are generally reliable and perform similarly to the Distributed Learning paradigm [3]. Nevertheless, the data is generated, stored, and processed locally. One key example is the Cross-Silo federated learning for Health-Care organizations of different hospitals like depicted in figure 3.3.

The number of actors is limited, from two up to hundreds, generally speaking, depending on the width of the organization involved. The communication links are assumed to be reliable, but the geographical distribution introduces higher network complexity levels. The computation and/or the communication could provide a challenge depending on data volumes and types. Here is the figure of the orchestrator (server), the node keeping the target labels. Instead, every other trainer party (also servers) participates at every round, providing none or somewhat

very few failures[3]. [3]

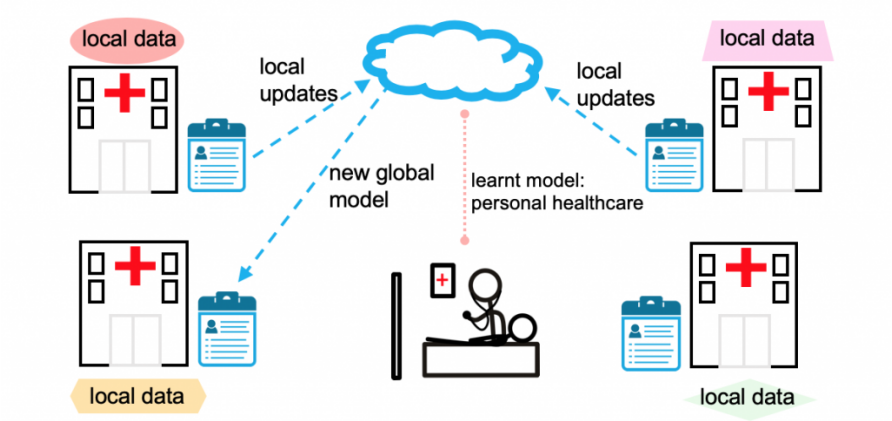

Figure 3.3: Cross silo federated learning for healthcare, image readapted from [25]

Cross Device (FL) Cross-device infrastructure is the emerging paradigm proposed by the Google research team [13], which is addressed by the proposed work with the optimization provided. Figure 3.4 depicts an example of cross-device FL. The devices, in this case, are in the range of medium to low computational power (from the end-user computers to mobile phones). The leading reference targets mobile phones due to their impressive distribution and increasing computational power, furthermore, they have many incorporated peripherals that generate tons of data.

The geographical distribution is supposed to be hugely ample and sparse, while the number of nodes is immense. The data is generated locally and remains local, while the communication is about the gradients or generally about the model parameters. System heterogeneity is one challenge still in its definition. It could occur in terms of data (usually horizontally partitioned), system, hardware, software, or even law. The most challenging aspect reflects the communication cost [3]. Transferring models instead of gradients is less data-dependent, however, deep models are well-known for the massive parametrization.

The convergence of a globally shared model requires many and many steps. The main scenario considers one orchestrator server which synchronizes the clients' models. The clients' pool is huge, considering up to 10^{10} different devices, but it must be assumed the partial participation at least over a single round. Cross-device FL is affected by highly unreliable networks and client states, while the devices involved have limited computational power and battery resources constrained. Different configurations have been developed, such as fully decentralized or hierarchical FL [3].

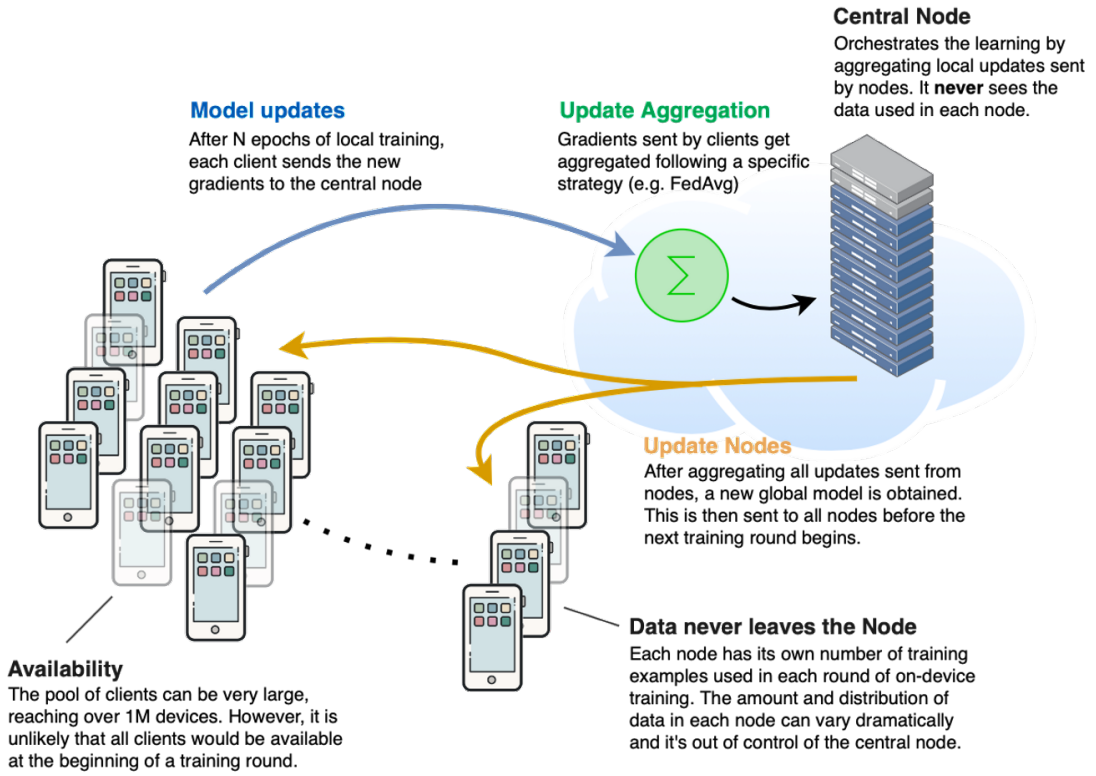

Figure 3.4: Cross Device Federated Learning - picture adapted from flower documentation [24] [26]

3.3 Architecture

The FL architecture takes into account the data splits performed across the various clients. It is helpful to consider the aggregated data as a big table where the rows represent the examples while the columns represent the features. It allows a visual partitioning of data by row (example) or column (feature). The main three types are Horizontal, Vertical, and Transfer which is a mix of the two. Of course, these definitions have many facets depending on the case, as the others definitions shall be considered as the centroids of clustered works [27].

The common diversity is the horizontal/vertical splits. They are addressed as partition by example and partition by feature, respectively. Even if the focus is on the data partition types, the differences are on the architectural level. The parties act and interact discordantly from an architectural point of view, requiring different coordination methodologies.

Horizontal or homogeneous graphically depicted in figure 3.5 is the directly addressed architecture of this work. The data partition over the actors is performed by example [8] [3] [27]. Every actor is supposed to have different data points in terms of amount and or targets. It is typical of cross devices settings, the traditional FL example holds a key example. Traditional FL exploits end-user smartphones where users could have their pictures of family, pets, etcetera. The exchanges between parties are supposed to respect privacy by design, by sharing models instead of their raw data. Privacy corresponds to the pivotal advantage. The challenges consider mostly the communication cost and the capability to deal with the heterogeneity, concerning both system and data. The heterogeneity could be extremely treacherous.

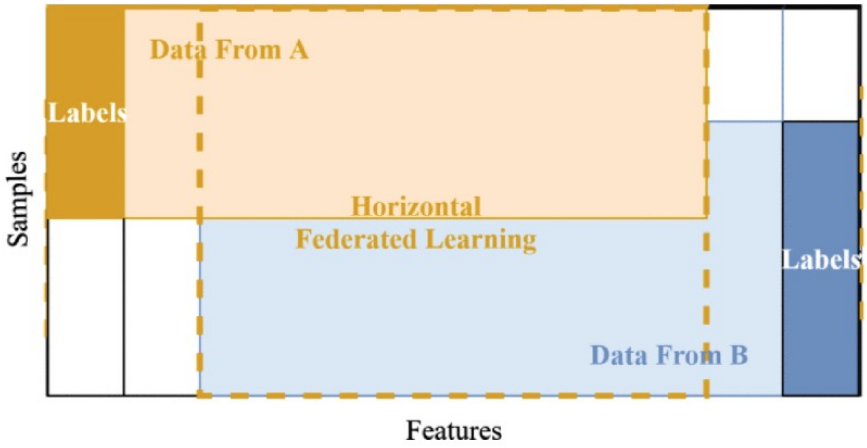

Figure 3.5: Horizontal Federated Learning adapted from [27]

The communication cost is due to the model size and convergence mostly. The table 3.1 is an example inspired from [8], on the left client 1 has the data records of only two different patients, both alive and non-smokers. Every client has its own labels that are locally stored. Client 2 (on the right) has many more patient records (different from client 1) with different distributions of both features and labels, which instead are conceptually shared (both clients have the names of the same columns).

Vertical or heterogeneous graphically depicted in figure 3.6, is not addressed by the proposed optimization but its definition gives a better understanding of the generalized concept of FL. The data partition over the actors is performed by feature level [8] [3] [27]. The server and client definitions are a bit misleading because vertical FL highly correlates to cross-silo structure, where the parties are composed of equal servers. It is better to change their definition to active/host

Table 3.1: Horizontal FL

Table 3.2: Client 1

Patient	Age	Smoker	Alive
1	68	N	1
2	23	N	1

Table 3.3: Client 2

Patient	Age	Smoker	Alive
3	68	Y	0
4	23	N	1
5	67	N	1
	...		
832	76	Y	1
833	42	-	0

party for the nodes owning the target labels, while passive/guest otherwise. The nodes communicate partial gradients throughout the links. A single node has the labels' availability and can compute the loss function to update the other, or also the other, local models, while data remains local.

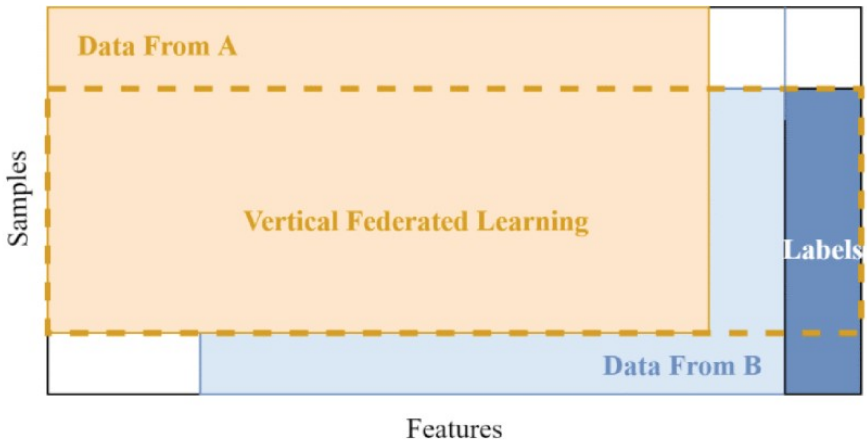

Figure 3.6: Vertical Federated Learning adapted from [27]

The model is then globally distributed instead of shared or personalized, accurately it is parallelly distributed, which distinguishes itself from serial model distribution more typical of Split Federated Learning [25]. In the cross-silo scenario, computation and privacy take the critical issues ranking over the communication bottleneck. However, their precise ordering depends more on the data and batch size, compared with the gradients. In table 3.4, the partition provided is more clear. On the left, client 1 acts namely as an active (or host) party. It has the target labels (Alive) along with its own features and data points. Client 2 instead has no labels (guest - passive party), shares the same data examples but with different features (donation and income compared to Age and Smoker).

Table 3.4: Vertical FL

Table 3.5: Client 1

Patient	Age	Smoker	Alive
1	68	Y	1
2	23	N	1
3	67	N	1
4	68	Y	0
5	42	-	0

Table 3.6: Client 2

Patient	Donations	Income
1	10.00	1M
2	200.00	300k
3	5k	300k
4	NaN	400k
5	200M	-

Federated Transfer learning One additional architectural classification is the federated transfer learning which, rather than a whole new architectural concept, is a mix of vertical and horizontal architectures 3.7. Federated transfer learning comes to be a compelling approach when data is highly heterogeneous. It exploits the transfer learning of traditional machine learning concept to benefit the target domain from a general source domain [11] [7] [27]. In addition to guests and hosts of vertical FL, it is inserted the role of the arbiter, defined as the node that occupies of sharing the public keys, collecting gradients, and checking the loss convergence.

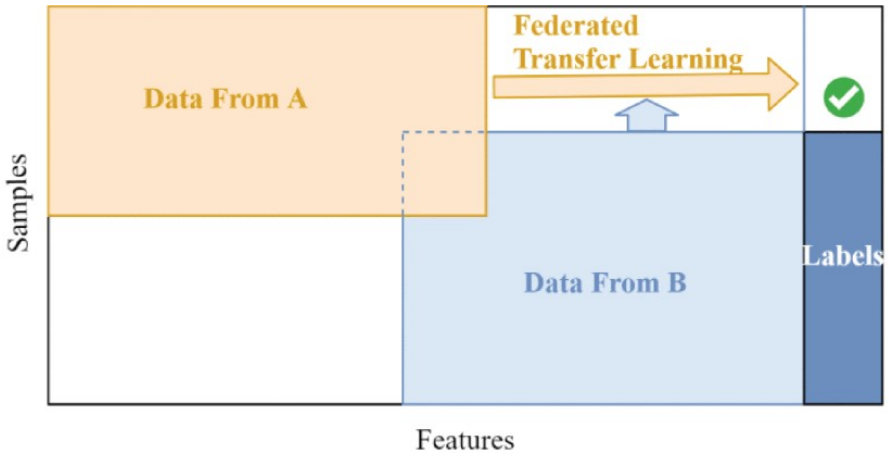

Figure 3.7: Transfer Federated Learning adapted from [27]

3.4 Topology

The network's topology is represented as an indirectly connected graph. The graph is a conceivable illustration that depicts the involved parties (nodes) and their relationships (edges). There are correlations between the previous definitions, although there is an abstraction in terms of exchanges. Even if two more additional

topologies are cited in [12] (Split and Vertical, 3.8 the two graph representations on the right), this work focuses on the topologies that are typical of standard FL (centralized) and two of its enhancements, still applicable to cross-device settings while mostly sharing the horizontal partitioning: Hierarchical and Decentralized FL.

The centralized topology is represented by a star graph (figure 3.8 on the left), where the central node is the server and leaves are the trainer devices. The server has a bidirectional edge with every client (the server sends the model to the clients, and then gathers their versions whenever possible). Typically exploited by cross-device horizontal FL, it is the topology targeted as the principal FL scenario provided by Google in 2017, with FedAVG [13]. Under cross-device settings, it evolves in a dynamic star graph because of partial participation. It is one of the first stage implementations, afterward reshaped into hierarchical and decentralized topologies, far more complex. Despite the recent evolutions, it serves as a yardstick for optimization. The naive concept is yet still valid and required, allowing a simpler reproducibility of the results. The server presents the very single point of failure, mainly dreaded because of malicious attacks.

Hierarchical [28] proposed hierarchical FL, an optimization of the centralized topology. The star graph of centralized topology advance to a tree graph (figure 3.8, middle). The root node is still the cloud server while the leaves are the end-user devices. However, instead of a direct connection, the graph provides a mid-step layer, typically composed of edge nodes or servers. The idea is related again to cross-device learning. In particular, it has been engineered for mobile networks, considered highly unreliable and heterogeneous. The centralized concept recurs in the sub-graphs, which replicate at least one time over the leaves. The cloud server is still the root node and coordinator, head of the MacroBase central Station (MBS), and samples and aggregates the models of its child nodes. The children nodes are both coordinators and producers. They do not directly train models, but act as Small cell bases (SBS), reproducing the centralized topology with end-user devices. SBS broadcast, collect and aggregate trained models to update their local versions similar to the center node, but then they push it to the MBS. The stability and performances provided come with a cost. It increases latency because of topological relationships: one round consists of many rounds of SBS and one for the MBS.

Decentralized Decentralized topology is also referred to as peer-to-peer FL or fully decentralized FL in works such as [29] [30] [31]. It is graphically representable through a partially connected graph, usually random or K-regular. One of the most challenging aspects relies on the consensus of the model. The coordinator becomes no more centralized or even not present at all (every node is an end-device or an end-node). Hence the solution considers alternating the subgraphs composing

nodes to provide coordinated services. Another solution exploits the social network learning theory, provided by DeGroot, to find consensus. However, the consensus is not always achievable depending on the graph. The graph dynamically evolves its topology: the relationships (links) are based on the availability of the nodes. The security and privacy prove to be more guaranteed without a third-party coordinator requiring. However, it requires more computation for encryption and decryption, whenever present. [27].

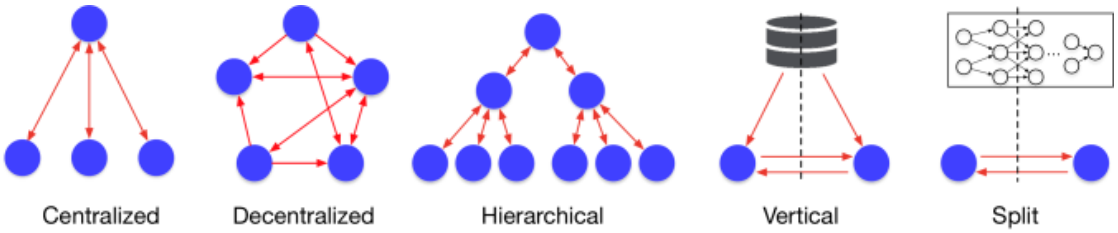

Figure 3.8: Federated learning topologies adapted from [12]

3.5 Heterogeneity in Federated Learning

In federated learning applications, heterogeneity is inherent to the common ground of problem definition. It relates to many aspects but addresses most commonly the data skew. Other relations are hardware, network, geographical, and model represent critical challenges in a real-world scenario. This section presents the most meaningful data heterogeneity definition with its many facets, followed by other heterogeneity definitions that affect the system or the application, focusing on the system heterogeneity and suggesting broader sense notions such as law enforcement.

3.5.1 Data heterogeneity or skew

Data heterogeneity is the most taken into account in FL research. It severely affects the deep learning models and their global convergence. It is intrinsic of FL notably for the horizontal partitioning of dataset brought by Cross-Device settings. The identically and independently distributed (IID) assumption is almost self-explanatory. This section reports and defines it to give a better understanding of the concept. By so, it is possible to enlighten its comparison to the nonIID case. NonIID instead, presents various mixes of unbalanced data amounts with different distributions of features and target labels. For research purposes, the need for datasets embedding the heterogeneity assessment is targeted conspicuously. However, the generation of FL datasets is still far from realistic deployment. Different FL tools and frameworks provide three generating methods: (i) partitions

of classical deep learning datasets (MNIST, CIFAR10, ImageNet, Shakespear...), (ii) some ad hoc premade datasets (Federated EMNIST), or (iii) the generated in an FL fashion synthetic datasets (Federated Synthetic datasets) [12].

Amount Balance

The effortless concept when dealing with homogeneity is the amount of data available in each device. The training of a neural network is performed over mini-batches of data to compute the gradients, which are then averaged to perform the back-propagation. If a device must divide its 1000 training examples into 50-sized data batches, it performs 5 epochs based on 100 local steps, otherwise, if it has only 10 examples it performs 5 local steps for the same amount of epochs. The different batch sizes could lead to better or worse results [13] affecting the convergence of the standard FL algorithm.

Balanced Every client has the same amount of data with the corresponding target labels. A more precise mathematical notion would be: $\mathbf{x}$ features, $\mathbf{y}$ labels, $\mathcal{D}(\mathbf{x}, \mathbf{y})$ dataset, i, j clients (for any arbitrarily chosen clients i, j without loss of generality.):

$$|\mathcal{D}_i(\mathbf{x}, \mathbf{y})| \approx |\mathcal{D}_j(\mathbf{x}, \mathbf{y})|$$

The absolute value represents the cardinality of the dataset. Each client is supposed to have the same amount of data points or at least the difference between them must be close to the definition of pointless. Even if it is simpler for testing purposes, it poorly represents reality conditions. The number of data examples created and stored by different devices could depend on the storage capacity, the state statistics of the device, or the use it makes the owner. A smartphone with limited storage capacity is unlikely to have massive data points, nonetheless, the state for an autonomous MCU equipped with a camera could be affected by the battery life, sampling fewer examples than one another. In the end, one user could be more interested in capturing more pictures (a fashion blogger), yet the independence of the storing capacity and the battery life is not respected.

Unbalanced Much more close to the facts on the ground is the unbalanced assumption. It considers the differing amounts of data points across devices. From a statistical point of view, the amount of data examples over devices represents a distribution. However, there are some inconsistencies in the literature to produce FL synthetic datasets or to provide unbalanced partitioning over different clients. The first straightforward solution distributed labels singularly between the clients. Two distributions of use split the dataset across clients in a more reality-trustworthy way: the lognormal distribution and the Dirichlet distribution (usually paired with

the label partition with the Latent Dirichlet Allocation, LDA). The mathematical definition is pointed out below: [12]

$$|\mathcal{D}_i(\mathbf{x}, \mathbf{y})| > |\mathcal{D}_j(\mathbf{x}, \mathbf{y})|$$

which represents more a lognormal distribution with slight variance or

$$|\mathcal{D}_i(\mathbf{x}, \mathbf{y})| \ggg |\mathcal{D}_j(\mathbf{x}, \mathbf{y})|$$

that instead is more representative for the LDA with low values of α. The side effects of unbalancing are partially mitigated by standard FL [13] on the aggregation phase by a weighted average of the models accordingly to the dataset size. Clients' selection could be exploited to disqualify the too under-representative clients that might be toxic for the model convergence (for example if $|\mathcal{D}_i(\mathbf{x}, \mathbf{y})| < B$ where B is the batch size).

Feature and label distributions

Deep models more than requiring a sufficiently high number of examples are not robust against the class unbalancing suffering of overfitting. Despite that, it is exactly what the real-world scenario introduces in FL. The clients' local datasets are not only unbalanced in terms of cardinality, but they also present features and labels distributions' skew. The label distribution is the most addressed nonIID challenge because it deeply impacts standard FL performances. It stretches the number of rounds needed to reach convergence and deteriorates the accuracy's steadiness, generality, and top levels. The nonIID assumption leads the devices to diverge from the minimal risk of the global model, but even more from the other devices, distancing from a smooth convex optimization function.

Notions and notation In this case, this work refers to the classification task, a supervised method that considers the features of a data sample to predict the label previously defined for the training. A little notion introduces the mathematical concepts needed by the subsection. Given $\mathbf{x}$ features, $\mathbf{y}$ labels, i, j clients, their local data distribution denoted as $\pi(\mathbf{x}, \mathbf{y})$ and a relative drawn example $(\mathbf{x}, y) \sim \mathcal{P}_i(\mathbf{x}, y)$ As suggested in [3], it is convenient for definition purposes rewrite it using the conditional probability formula:

$$\begin{aligned} \mathcal{P}_i(\mathbf{x}, y) &= \mathcal{P}_i(\mathbf{x}|y)\mathcal{P}_i(y) \\ &= \mathcal{P}_i(y|\mathbf{x})\mathcal{P}_i(\mathbf{x}) \end{aligned} \tag{3.1}$$

Identical and Independent Distribution (IID) IID fashion implies that each client has approximately the same distribution of both labels and features,

moreover assumed as independents. That means that every client has at least one representative data point ($\mathbf{x}$) per each class (y), while the ratio of examples is the same for each class of each client dataset. For example, if the i-th client's dataset accounts of I dogs, I cats and I frogs pictures and the client k-th has K dogs, K cats and K for any values of $I, K \in \mathbb{N}^+$, the distribution is assumed to be identical. The probability distribution $\mathcal{P}(\mathbf{x}, y)$ is shared or close between any arbitrary clients couple i, k:

$$\mathcal{P}_i(\mathbf{x}, y) \approx \mathcal{P}_j(\mathbf{x}, y)$$

The independence violation usually occurs because of eligibility requirements, when the devices are not under users' use and are fully charged, by so are introduced significant daylight patterns [3]

Non Identical and Independent Distribution (NonIID) A completely comprehensive definition and assessment of NonIID is still an opened challenge in FL, however some guidelines must be assumed whenever cross device with horizontal partitioning settings are applied. Generally the nonIID assumption considers the difference across clients in the distribution of paired data examples with their labels

$$\mathcal{P}_i(\mathbf{x}, y) \neq \mathcal{P}_j(\mathbf{x}, y) \tag{3.2}$$

However, the clients' distributions may change, likewise could change the defined distribution for a generic client i. [3]. The inequality 3.2 is due to the difference in one of the four terms composing the conditional probability transcription of the formula, producing four different types of nonIID-ness and skew.

Label distribution The label distribution skew could happen in terms of the marginal distribution $P(y)$ (different labels distribution across clients) or the conditional distribution $P(x|y)$ (distinct labels distribution between two clients that share the features' distributions). Common strategies provide the whole labels partitioning across clients (e.g., ten labels and two clients produce 5-fold mutually exclusive label datasets), or the Dirichlet distribution regularized by α. [12]

Prior probability shift (Labels distribution skew) From a mathematical point of view, the equation 3.2 is not satisfied because of different marginal probabilities of the labels between at least two clients, and so $\mathcal{P}_i(y) \neq \mathcal{P}_j(y)$. It does not mean that the conditional probability of x based on y must be different as well [3]. It could be that:

$$\mathcal{P}_i(\mathbf{x}|y) = \mathcal{P}_j(\mathbf{x}|y), \quad \mathcal{P}_i(y) \neq \mathcal{P}_j(y)$$

The most inducing cause grounds in the intrinsic geographical distribution of FL, distant regions provide different distributions. E.g., white polar horses do not live in tropical forests, wombats' pictures are unlikely to be sampled from Nord-Europe

users (except for voyages and or Zoos). Different facial treats distributions locate in different continents (example readapted from [3]). The bias introduction has been addressed by some work to respect fairness [32]. Figure 3.9 depicts an example of the prior probability shift where a wombat (typical of Australia's) may be present only on the dataset of client 1 (a), while the Blue-footed bird can only be found in the Galapagos (b). It is an extreme example but efficient: the samples are dependent on the geographical location, or by the preferences of the user, who may dislike wombats and never take pictures of them.

(a) client 1 **(b)** client 2

Figure 3.9: Example of concept drift, typical animals of localised regions (a. wombat:Australia and b. Blue-footed bird:Galapagos)

Concept shift It could also happen that the same features link different labels. From a mathematical point of view by exploiting the second equation of 3.1 it could happen that even if the marginal distribution of the feature is respected, the conditional one is not:

$$\mathcal{P}_i(\mathbf{x}) = \mathcal{P}_j(\mathbf{x}), \quad \mathcal{P}_i(y|\mathbf{x}) \neq \mathcal{P}_j(y|\mathbf{x})$$

It often happens for tasks such as sentiment analysis or next word prediction, e.g., if some clients prefer cats over dogs, they could like a comment or a video where a cat scares away the dog and vice versa [3]. Fig 3.10 represents an example: a sticker is pasted to the picture of a cat, the glasses can be approved by one client while annoying one another which could be a strict radical animal lover.

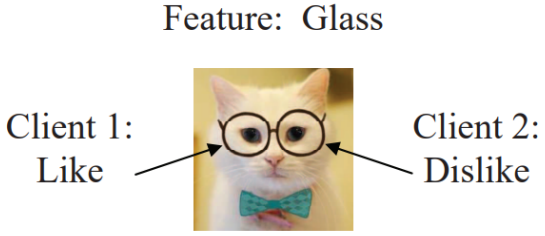

Figure 3.10: Concept shift labels distribution skew example, adapted from [8]

Feature distribution / Attribute skew The feature distributions consider the differences between datasets' attributes. The pointing differences scheme is still the same, it underlies the divergences of the marginal distribution of the attributes (Covariate shift), or the distribution of the conditional features of the same labels (Concept drift). The feature distribution heterogeneity, differently from labels, is less characteristic of horizontal FL, shifting the importance towards the vertical FL (because of example/feature partitioning).

Covariate Shift The clients feature distribution variance is due to the marginal probability distribution of the features per se $\mathcal{P}(\mathbf{x})$, independently from the conditional probability of the labels, given the features:

$$\mathcal{P}_i(y|\mathbf{x}) = \mathcal{P}_j(y|\mathbf{x}), \quad \mathcal{P}_i(\mathbf{x}) \neq \mathcal{P}_j(\mathbf{x})$$

I.e., two users in a hand-writing letter recognition task have different writing styles, i-th user is used to press down harder whenever it writes the "**i**" letter but faintly draws the "l", the j-th user instead always draws both of them emphasised ("**i**","**l**"). If the first user has a dataset composed by ("**i**":I,"**i**":I,"l":L), and the second users instead has ("**i**":I,"**i**":I,"**l**":L), they share the labels, however the width distribution of one dataset differs from the other [3].

The work provided by [8] introduces an inner distinction of covariate shift. The attributes may also have different overlapping levels: Non-overlapping (vertical FL), partially overlapping, and fully overlapping (horizontal FL) represented in figure 3.11 which proposes for each type an example of covariate shift between two clients.

Fully overlapping refers more to the horizontal FL, two cameras of two different devices could have different lenses or sensors quality, introducing various levels of noise into the same subject, figure (c). **Non-Overlapping** refers more to the vertical FL, which almost represents its definition (feature partitioning). In figure (a), it is possible to see that even if the two pictures represent the same panda, they are two mutually exclusive partitions of a larger picture. The **Partially Overlapping** features are depicted in figure (b). The same icon of a PC can show a partially shared image, with different grades and directions of the rotation based on the horizontal axis. Such a difference could be due to the sampled pictures of a multi-view camera system.

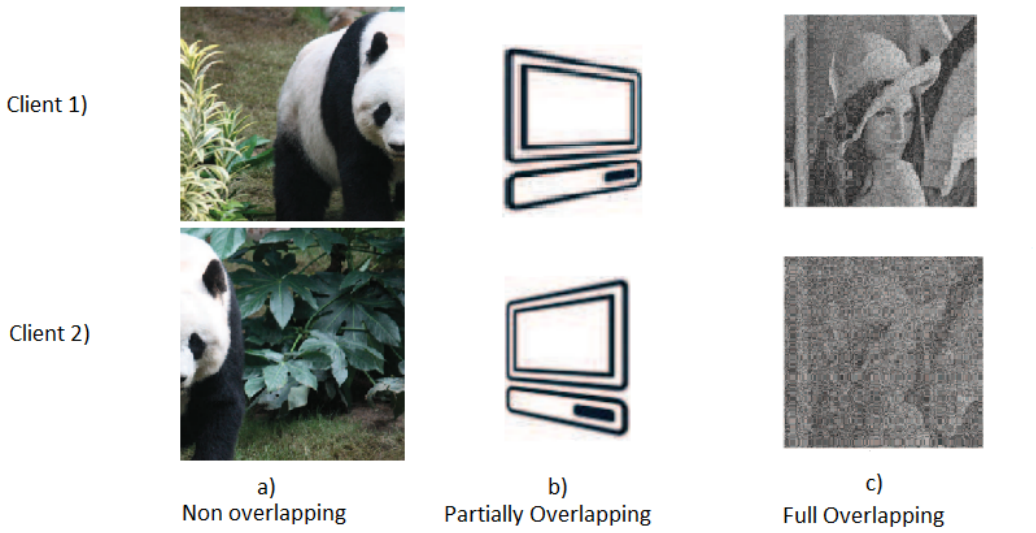

Figure 3.11: Covariate shift based on overlapping levels: adapted from [8]

Concept drift The last member that could not satisfy the equation introducing feature heterogeneity is the concept drift, which refers to the variation of the conditional distribution $\mathcal{P}(\mathbf{x}, y)$. It could also occur when the marginal distribution of labels ($\mathcal{P}(y)$) is shared.

$$\mathcal{P}_i(\mathbf{x}|y) \neq \mathcal{P}_j(\mathbf{x}, y), \quad \mathcal{P}_i(y) = \mathcal{P}_i(y)$$

Similar to the prior probability shift, the heterogeneity introduced is mainly due to the geographical distribution of the data collected. Two clients native to remote regions (such as Quebec and Saudi Arabia) share the number of cars pictures. Nevertheless, the first ones are covered by snow, presenting low luminance (the light hours in cold countries are limited), contrast, and temperature values. On the other hand, the other client's pictures may be about cars laying on the sand, having the opposite values of the attributes mentioned [3]. In figure 3.12 there are two images of car drifting (as the concept) on different kinds of powders: (a) sand and (b) snow. It must be considered as a subsample of larger datasets of both clients mainly represented by similar topics.

(a) client 1 **(b)** client 2

Figure 3.12: Example of concept drift, drifting cars' pictures with different conditional attributes' probabilities, (a) desert drifting car and (b) winter drifting car

3.5.2 System heterogeneity

The system heterogeneity heavily impacts the performances and the design of FL algorithms, especially under Cross-Device settings. Besides the dissimilarities in the local data, each device has its own characteristics based on hardware, software, and state. This heterogeneity could introduce stragglers or failures that the paradigm must account for.

Hardware and State The **hardware** architecture of the devices can impact the computation and the communication of different devices. It relates to the CPU capability, the possible presence of a GPU, the network connectivity (3G, 4G, 5G, or wifi), and the memory (in particular the RAM to perform the training and the ROM to store the models and data). Different performances introduce failures and latency in the training phase. The coordinator must then wait for stragglers and account failures. Many techniques have been deployed to provide robustness from fault tolerance, asynchronous FL to model personalization, quantization, and compression [3] [25].

The **client state** also has a deep impact, even if the client's equipment shows relatively good hardware and connection. The battery life or the processes already running on the device could play a key role. While AIoT devices devote themselves entirely to the task, smartphones handle many tasks requested by the owner. Their employment could ruin so both the user experience and the training performance. In the end, the hardware heterogeneity may also impact the data heterogeneity, providing different datasets in terms of amount, colors, etcetera.

Software differences can impact the feasibility and reliability of the devices, even if it is more distinctive of cross-silo settings [3]. Different operating systems, frameworks, versioning, protocols, and standards could impact the performances, the expected results, or even the feasibility. FL market-driven algorithms should

be multi-platform. So they should be agnostic to the software versions and tools, in particular for smartphone devices training. Dealing directly with ten servers is remarkable, while it is unimaginable to do it with billions of devices).

Model One overcoming technique that addresses the hardware heterogeneity introduces the model heterogeneity, different hardware computation capabilities could preclude tons of devices from participation. The idea of FL is the opposite: it shall distribute the load across a fleet of devices. The more load distribution, the better. Many techniques of model compression already deployed for inference over highly constrained devices [22] and every energy-driven optimization of traditional deep learning could be transposed for the need.

The global reduction of the model size possibly reduces its accuracy, while precluding portions of the devices pool withholds access to their data. The application of specific techniques over participating devices still provides enhancements to the global model. The architectural heterogeneity implies different model architectures, while the cardinality reduction could be provided by pruning singular weights (namely sparse pruning) [33] or clustered weights, grouped by channel (channel skipping) or layer (layer skipping). Also, the width of a layer could be tuned to the need, providing the same number of layers but reducing the number of feature maps. Moreover, the quantization could shrink the precision of the weights and biases, introducing the precision variety.

Different model aggregation techniques directly address the model heterogeneity. E.g., FedMA considers the similarity of neural layers output [32], while the model/-knowledge distillation [34] requires public and private datasets applying transfer learning techniques, and personalization (clients share only some layers with the global model).

Political and others The geo-delocalization could be affected by different governments and laws enforcement or regulations. Some data could not be available, illegal to gather, or process depending on the countries and regions. The figure 3.13 represents the map of worldwide data protection and privacy legislation extrapolated on the site of the united nations conference on trade and development (UNCTAD), which partially addresses the legislation presence and levels over the countries. However, the legislation differs even further. The European region is regulated by the GDPR, while Convention 108 establishes similarities with Russia. Different regulations concern the United States and China. To conclude assessing the real-world heterogeneity is still a challenge: "Nothing has more degrees of freedom than reality" cit. Elon Musk.

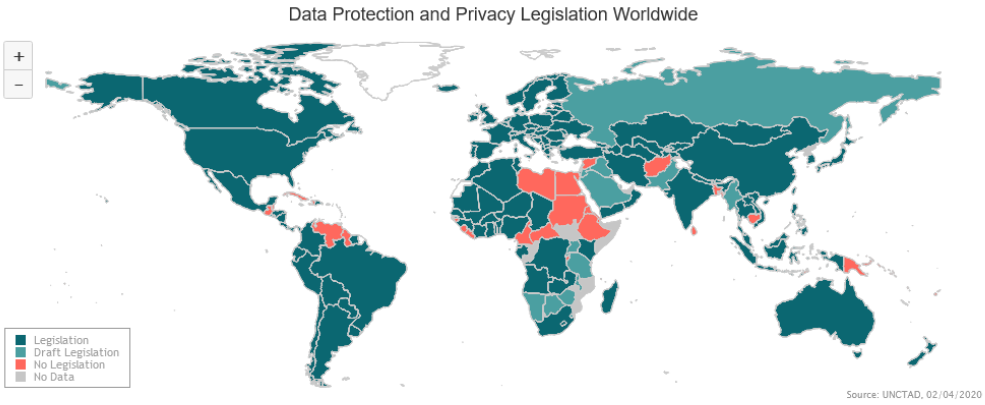

Figure 3.13: Worldmap of data protection law, picture adapted from [35]

3.6 Challenges

This section points out the main challenges of FL ranked by importance. They represent the ones that mainly focus on mobile edge networks and low computational devices typical of AIoT.

Communication The most important challenge lay over the core bottleneck of FL laying in the communication. The remarkable number of rounds and the model sizes result in massive communication loads. The higher the number of rounds, the higher is the cumulative communication cost, which is the most substantial energy sink. As previously mentioned, the devices involved to train the models have constrained resources. They are connected through wireless networks and are supplied with batteries. The model size leverages the cumulative communication overhead, but furthermore, it increases the energy consumption for the single round of the single device. By so, it impacts the affected devices in terms of load and time. The most common solutions provide faster convergence techniques based on regulation or different aggregation, while few other techniques involve the model compression techniques such as quantization and sparsification. [27] [3]

Heterogeneity The definition of heterogeneity is provided in section 3.5. It may vary on the system heterogeneity, which considers more the hardware and relative state of the devices involved, or on the data heterogeneity, which considers the partitioning of data and the relative skew that shows up in real-world scenarios). The main problem caused by statistical heterogeneity or data heterogeneity is the complexity generated. Deep models training performs well for massive data points,

identically and independently distributed. Despite that, FL produces the exact opposite situation between different client datasets. The client drift, introduced by the divergent optimization of the model, could reveal itself to be toxic for the global model convergence. Contrariwise, the global model's update could not fit at all the clients' needs. Many regularization and personalization techniques deal with data heterogeneity. However, still the statistical metric definition a priori is an open challenge.

The system heterogeneity is also typical of cross-device FL, where the involved devices have different capabilities in terms of communication, computation, storage, and availability. The client state could vary because of low battery levels and bad communication bandwidth. The highly varying performances could severely affect the methodology, inducing stragglers or failures. Client selection, asynchronous FL, and timings are some techniques that deal with them. [27] [3].

Privacy & Security Respecting privacy becomes more important, FL address privacy by design avoiding data transfer across the internet. Despite that, malicious clients or even servers still could infer data using gradient reversal techniques. Few gradients can be sufficient to reverse the data entirely. FL, like any other internet service, is not devoid of cyber threats. They could concern the many stages and devices, from training to inference, from client to server. For such techniques as homomorphic encryption (cryptographic technique) and differential privacy (noise addition over the transferred message such as model or gradient) have been deployed. [3] [10]

Chapter 4

Related works

FL is subject to an increasing evolution interest, which leads to more and more techniques towards many directions as could be seen in the hint of chapter 3 for the open challenges. This section presents a few related publications, ordered by clusters of common ground techniques. The papers presenter are the ones that have shown to respect some coherence with the FL type referenced by this thesis and inspired the novel method. A context where the server exploits the edge devices to perform the computational task required by deep learning. The clients train locally a previous globally shared model with their local data. Once the local training ended up, the clients share their models' parameters instead of their data or gradients, (exception made for [36]). Based on the previously stated definition 3, the related works are in an FL context which is supposed to be: horizontal, cross-device, and centralized, yet typically synchronous (Except [37]). More specifically, the works considered are the baseline [13], and its relative optimization techniques, exception made for [38] which is based on gradients aggregation (FedSGD), the technique then passed by FedAVG.

4.1 FedAVG - Pioneer

FedAVG [13] is considered to be the most representative work for the referred FL scenario. It is used as the baseline comparison by almost every other subsequent related paper. For this reason, the comparison proposed is yet also referred to it, and it is the only one of which the algorithm will be presented. It provides the backbone shared by the proposed FedNILO (similarly to cited works). The key concept proposed is to perform multiple steps of Stochastic Gradients Descent (SGD) over the computation clients, then the model parameters are transmitted instead of the gradients. The communication paradigm is similar to its ancestor. It is based on multiple steps of decentralized training, performed synchronously.

The model parameters transmission enables a more secure privacy paradigm, but most importantly, optimizes the communication from a range of 10x up to 100x communication rounds reduction compared to FedSGD.

The algorithm 1 represents the task that runs on the server for the training of a global model. The table 4.1 is a brief resuming of the notation operated.

Algorithmic notation

$\boldsymbol{\Theta}$ Model Parameters	η Learning Rate
$\mathcal{S}_k$ Reachable devices	$\nabla l(\boldsymbol{\Theta}^k; \mathbf{b})$ Gradient of the loss computed on data batch $\mathbf{b}$ to update model parameters $\boldsymbol{\Theta}^k$

Table 4.1: Notation

FedAVG (Server)

Parameters:
Rounds T

Algorithm 1 FedAVG - Server side

1: Initialize $\boldsymbol{\Theta}_0$; ▷ initialize model parameters on server
2: **for** r (round) in 1...T **do**
3: Sample $\mathcal{S}_k$ subset of reachable clients
4: $n \leftarrow 0$ ▷ training examples counter
5: **for** k (client) in $\mathcal{S}_k$ in parallel **do**
6: $\boldsymbol{\Theta}_r^k, n_k \leftarrow ClientUpdate(\boldsymbol{\Theta}_{r-1})$
7: $n \leftarrow n + n_k$
8: **end for**
9: $\boldsymbol{\Theta}_r \leftarrow \frac{1}{n} \sum_k^{|\mathcal{S}_k|} n^k \boldsymbol{\Theta}_r^k$ ▷ update server model through weighted average
10: **end for**
11: return $\boldsymbol{\Theta}$

The algorithm requires as input the number of rounds along with the number of epochs, they determine the number of steps to be performed on the server-side and the client sides, respectively. At line 1 the server must initialize the model and its parameters, at line 2 up to line 10 there is the iterative procedure with the key steps of standard FL performing T steps which are referred to as rounds. In line 3, the server begins to check the availability of the devices and it performs the client selection, which is simulated through a random uniform extraction with the replacement of the available clients. Then it initializes n, a counter for the number of data examples employed in the cycle. Lines 5 to 8 propose a loop to be

parallelly accomplished (to spare execution time). Every selected client is requested to perform the training of the server model, returning the new model parameters and the number of data points that contributed to its training (line 6). The server stores momentarily the models and updates the counter. At the end of the cycle, the server updates its model with the weighted average of the received models for each k-th client accordingly to the number of data processed by them (line 9). At the end (line 11), the server returns the last global model version.

Client side 2: The clients receive from the server the last version of the global model, and then, they (line 3) overwrite their local models' parameters. For every epoch in the range previously defined (by the design of the application as a given parameter or by the server), it performs a cycle (lines 4-8), which perform an inner loop (lines 5-7) to update the model with the local data for E epochs. In line 6, for every batch of local data, the client performs an SGD step, to improve the model. Once they conclude the training (line 9), they return the model to the server along with the number of examples that have been exploited during the local training (line 9).

FedAVG (Client)

Parameters:
Epochs E

Algorithm 2 FedAVG - Client side

1: **Client:**
2: **procedure** $ClientUpdate(\mathbf{\Theta}_r)$
3: $\quad \mathbf{\Theta}^k \leftarrow \mathbf{\Theta}$
4: $\quad$ **for** e (epoch) in E **do**
5: $\quad\quad$ **for b** in batches **do**
6: $\quad\quad\quad \mathbf{\Theta}^k \leftarrow \mathbf{\Theta}^k - \eta \nabla l(\mathbf{\Theta}^k; \mathbf{b})$ ▷ local training step
7: $\quad\quad$ **end for**
8: $\quad$ **end for**
9: $\quad$ return $\mathbf{\Theta}^k$, $num\_examples$
10: **end procedure**

FedAVG has been tested over MNIST and CIFAR10, which the last unfortunately is presented only under homogeneous assumptions. Accordingly to the hyperparameters search, the work enounces that epochs E and batch size B are highly correlated with the algorithm convergence. Especially under heterogeneous data settings, a higher number of local epochs resulted in deteriorated performances, as subsequently confirmed by [39] and [32].

4.2 Optimizations of FedAVG

The following related works are based on FedAVG, they especially improve the target accuracy under heterogeneous assumptions, or the convergence speed to reduce the communication overhead to reach it. They are complementary with FedNILO, but probably need different hyperparameters due to the different convergence rates. In the end, this section exposes two paragraphs, the first one has similar works for the method proposed, which are the less orthogonal methods because they already remove or freeze already some parameters. They follow different policies more fine-grained, so the result could be also satisfactory. Instead, the second considers other techniques that reduce the model only over the communication link through lossy compression techniques, which again are completely orthogonal.

4.2.1 General - convergence under nonIID assumption

Data and Server learning rate This subsection considers papers that are more generic or not too strictly related. They share the same objective of optimizing the communication, but usually, they engage regularization techniques or different problem setups towards the scope. They focus more on the reduction of rounds to reach the target accuracy instead of the communication cost.

The most straightforward concept of FedAVG optimization based on heterogeneous settings is [9] **5% Data Sharing**, which suggests sharing only 5% of data. The final accuracy increasing claim is in the range of 30%, but yet the privacy is still violated even under impressive data sharing reduction. The tradeoff benefit versus cost is satisfactory, but it could not hold under law restrictions.

Another work that comes to be considered as a baseline addresses instead the server possibilities, **FedOpt** [14]. The algorithm proposition considers a new optimization methodology, performed over the aggregation of models on the server. The algorithm takes advantage of adaptive optimizers, such as Adam, Adagrad, and Yogi to update the global model. It proposes a more sophisticated way to assemble the trained parameters of the clients. Instead of computing the weighted average, it updates the global model as the same principle of the clients' training. It introduces a new hyperparameter that defines the server learning rate. The learning rate is used in conjunction with an adaptive optimizer to determine how much the new upcoming parameters should affect the global model. The final accuracy considered as the average accuracy of 100 rounds shows a boost of 8% on CIFAR10 and CIFAR100.

Regularization Data heterogeneity assumptions lead clients to overfit their models over a few labels. The local model parameters poorly generalize the other clients' needs, introducing bias and divergence for the global optimization.

Regularization techniques are widely employed to reduce the variance of the models across clients, to prevent or bound the weights divergence. Usually, these works add a term to the optimization loss that considers the distance between the locally updated model, and the one received by the server.

The clients' optimization divergence by the global optimal solution has initially been addressed by Stochastic Controlled Averaging **SCAFFOLD** [36], which comprehend a dynamically updated client state for each participating device. The method introduces the client state to measure and counteract the client drift due to its heterogeneity. The communication savings are in the range of 1x up to 4x, for the same accuracy levels (in terms of convergence speed up). However, as reported by [40] SCAFFOLD requires the gradients' communication, augmenting the communication overhead x2 along with the privacy issues provided by the gradient reversal attacks.

FedProx [6] adds a dynamic proximal term into the client loss function which works as a regulator. Basing the euclidean distance between the updated model and the received server's model penalizes the new optimization steps for better-shared consensus. The regularization is similar to the L2 penalty of traditional machine learning but is adapted to the model divergence considering the Euclidean distance between the optimized and the global parameters. A static parameter that weights the importance of the update reduction governs the strength of the regularization.

Similarly does **FedDyn** [40] which is based on a dynamical regularization over the client. The regularization shares some similarities between FedProx and SCAFFOLD. Nonetheless, it is a combination of two different terms. The first is an L1 penalty based on the inner product of the client's model and server gradient, while the second consists of an L2 penalty. Moreover, differently from SCAFFOLD, only the model parameters are sent through the network. During the aggregation phase on the server, it adopts a similar concept of FedOpt to update the shared model. It does not anymore restore the global model with the naive weighted average, though it performs the update through the last averaged aggregation regularized by an additional parameter conceptually similar to the learning rate, obtaining 4x of convergence speedup. The many results reported for FedAVG lead the choice of this paper results as the baseline comparison to match the state-of-the-art of FedAVG, given some due differences.

Beyond the average aggregation In the end, the last work considered as a baseline is **FedMA**[32]. It allows many more local steps by changing the aggregation on the server of the clients' updates. The shift provides a gap between the naive average of the gathered models and the one performed, which considers the network similarity of the layers. Moreover, the matched aggregation allows different widths of the network's layers to participate and contribute to the model update because the parameters are smartly clustered and then averaged whenever similar. The

proposal benefits up to hundreds of local epochs differently from FedAVG especially under heterogeneous assumptions, the testing scenario, however, is much more a Cross-Silo rather than Cross-device. The final accuracy increases up to 16% in parity condition of communication cost. Though, considering the very first step this solution could be considered similar to 1-round learning because it obtains a boost of 65% of accuracy with respect to FedAVG.

4.2.2 Communication & computation

Other works that aim to reduce energy consumption are mainly focused on the number of parameters that need to be optimized and/or communicated. In such cases, they are entitled to model compression techniques. On the other hand, communication could be reduced by compressing the message sent between the parties, usually through lossy compression techniques.

Model compression

Some works such as **PruneFL**[33] take advantage of the previously mentioned techniques already dispensed over low-end devices in particular for inference tasks, to exploit memory, power, and energy bounded devices, or in general with massively constrained resources. The objective prefixed is to minimize the energy consumption, besides they address the system heterogeneity, reducing the communication and computation time through a pruning technique based on the hardware resources and the client states. The pruning technique allows having different sized models per each client, dealing with the system heterogeneity. It compresses the models through a sparsification technique: only the larger weights (the ones with a sufficient magnitude) are optimized and synchronized. Even if the final accuracy endures, the results are close enough, while the loss is supported by a time reduction to $\frac{1}{3}$ and a density size to $\approx 15\%$. The tradeoff takes into account accuracy, time, and model size. It is noteworthy that this allows more devices to complete the training dealing also the stragglers and failures issues.

TWAFL [37] propose a less fine-grained reduction technique, by taking advantage of asynchronous federated learning and so partially resolves the stragglers' issue. This type of asynchronous federated learning still provides a methodology based on rounds for the procedure. However, it does not require every selected client anymore to synchronize its parameters at each session. Latecomers' results could be aggregated with new versions weighted by a staleness parameter: the more recent is the update the more it weighs on the round aggregation. Nevertheless, the main role of the contribution considers splitting the deep models into two subsets of layer groups: shallow layers versus deep layers. The first ones lay at the beginning of the network, for a Convolutional Neural Network (CNN) for example

could be intended as the convolutional layers, in opposite the deep layers are placed at the end, like the Fully Connected (FC) layers of a CNN. The work agrees on the previous convergence of shallow layers and similarly updates more frequently the deep ones, reducing the communication cost in a range from 40% up to 90% on MNIST, reporting even accuracy improvements.

In the end, **FedAPF**[41] freezes sparsely the model parameters affected by stability or staleness measuring the distances between updates. The concept is similar to PruneFL where the training involves sparse parameters structures. However, the clients share the same model architecture while the model is not pruned but just sparsely frozen. The communication benefits in the same way but the model size has no improvements. To avoid the transmission of masks every client must keep a record of every update of the global model, which could reveal itself as unfeasible under cross-device settings. Moreover, each client not only has to process the whole model but must also have sufficient memory to store the fine-grained mask, producing more faulty nodes.

Message compression and quantization

In the end, a little consideration is given to the completely orthogonal techniques of message quantization and compression. The compression and quantization techniques are less treated because are beyond the scope of this work but are just cited for the context exploration of the reader.

FedPAQ [42] exploits a low-precision quantizer on the difference between the trained model versus the one received by the server, because of the lossy compression technique the convergence is slightly slower, and also the accuracy drops little. But the time reduction reported is impressive with a difference of 10^2 under the same loss reporting.

While **FedDGC** [38] which is rooted on FedSGD considers the gradients updates. The algorithm takes advantage of the Deep Gradient Compression technique to compress the gradients but only if the gradient magnitude is above a given threshold. Whenever the gradients meet the threshold are disposable to be sent over the network. The algorithm introduced a compression ratio of 300x up to 600x.

4.3 Heterogeneity

In the end, a little consideration should be given to [43] that firstly defined a state-of-the-art technique for the nonIID data generation. It exploits the Dirichlet Distribution ($Dir(\alpha)$) to perform a random horizontal partition of the data. Much more close to the real-world scenario than other approaches such as label-based partitions. α scales the heterogeneity of the partitioning, for $\alpha \to \text{inf}$ it provides a homogeneous scenario, while for $\alpha \to 0$ every client has a single label dataset.

The publication inspired the authors of [12], which created a standard for client data heterogeneity relying on the Latent Dirichlet Distribution (LDA). LDA is widely used to represent the distribution of words across documents and topics. By abstraction, it could apply to define the similar distribution of the images between clients and labels. The process still divides the data labels by a $Dir(\alpha)$ but allocates the data examples by label across the clients, creating a heterogeneous federated dataset that provides nonIID and unbalanced fashion. Moreover, [12] accurately reports at the end of the paper the missing reference baseline sharing: dataset, hyperparameters, clients, clients selection, data partitioning, and so on.

Chapter 5

Methodology

This chapter describes the novel communication-oriented optimization FedNILO. It starts from the classical deep learning optimization that inspired it and then shortly defines the environment of its application along with the backbone of the algorithm. It gradually delves into the methodology. It brings forward the FedNILO implementation with the support of a graphical scheme. Then, it points out and details the algorithms that take place on server and client sides. The algorithms are again about the methodology, but they provide the notions and the fine-grained steps. In the end, this chapter provides a deepening for the relevant procedures to explain their workflow and show their logic. Indeed, FedNILO exploits the layer freezing, however, its distributed environemnt requires the distinction between their training and their synchronization.

5.1 Origins, backbone and objective

Classical deep learning origins Classical Deep learning techniques and optimizations such as transfer learning [44], and network similarity [45] declare the proportionality between the relative layers position and their convergence. Layers close to the input converge faster than the ones close to the output. Moreover, the first layers dedicated to feature extraction appear similar even for different datasets sharing the same task. The feature extractors of CIFAR10 and CIFAR100 as reported in [45] have high similarity trends.

LayerOut [46] applies a similar concept for the training time reduction in the classical context. In Fig. 5.1 it is represented the application of transfer learning from ImageNet dataset to benefit the colorectal cancer dataset accuracy performances. Only the last three convolutional layers and the fully connected layers are subject to the training. The previous ones are maintained frozen because they are similar across different datasets [45] They are exploited to process the

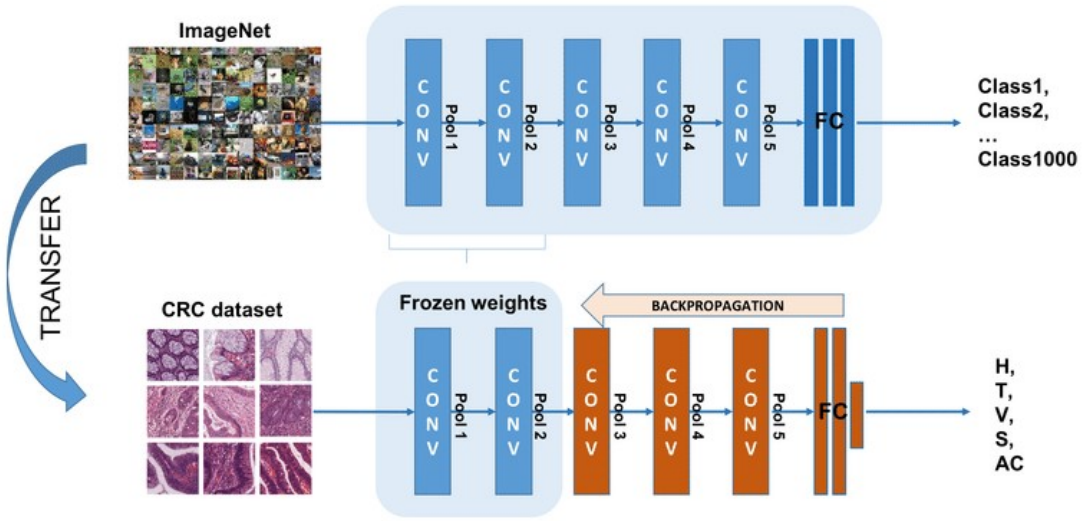

Figure 5.1: Transfer learning from ImageNet to histological images adapted from [47]

images and extract the features but do not evolve during training.

Federated Learning Backbone The previous chapters detailed the background of FL along with its taxonomy of application. The focus of this work is to emulate the traditional FL scenario brought by [13]. Namely, it provides synchronous updates for cross-device settings. The server acts as coordinator, while the clients, partially participate in training and have a horizontal partitioning of the data. As usual, the proposed algorithm shares the FedAVG backbone for the optimization.

The technique provided by FedAVG does not strictly follow classical convergence. The aggregation of model parameters after a few or many epochs is structurally different even from the average of the gradients, in particular under heterogeneity assumptions. Figure 5.2 graphically shows the weights divergence introduced by FedAVG compared to the classical optimization with centralized data and SGD depending on the heterogeneity settings. [41] demystifies the static layer freezing because of accuracy performances starvation. Hence, this work proposes it as a final stage optimization. FedNILO exploits the previously enounced discoveries and applies them to benefit FL through two fine-tunable knobs.

FedNILO objective and application The core objective of the algorithm proposed is to minimize the global empirical risk, through the distributed training of the server's model, reducing the communication cost of FedAVG. Even if the target of this work is to optimize the communication cost under heterogeneous

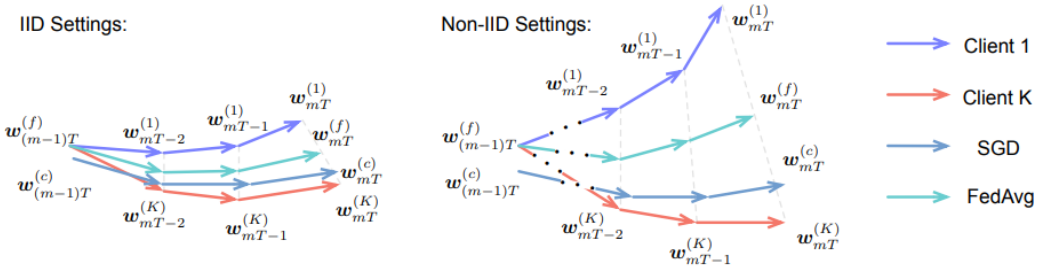

Figure 5.2: Weights divergence provided by different heterogeneity levels between FedAVG and SGD, readapted from [9]

settings, it reports the results also for IID to consider the stability relevance of the algorithm under agnostic heterogeneity conditions.

The algorithm proposal is based on a computer vision task, mainly referred to in the related literature. It exploits datasets such as CIFAR10 and CIFAR100 and particularly refers to the CNNs models. However, it could adapt to different datasets and other neural networks, characterized by a layered architecture such as MLPs. The FL infrastructure is centralized but there should not be any barrier in hierarchical and decentralized implementation with the due modifications. The variety of the scenarios provided consider the heterogeneous and homogeneous cases. Respectively, their partitioning makes use of the random uniform allocation and the LDA.

5.2 FedNILO

Wrapping up the FedAVG backbone, FedNILO is delivered with the scope to reduce the communication cost. Rather than reducing the number of rounds providing faster convergence like many optimizations proposed, it reduces the cardinality of the parameters to be trained and synchronized in a layer-wise way. The optimization performed is regularized by two additional hyperparameters, K, and F. They represent respectively the kick-start between the first freezing step, and the period regulating the layer-freezing scaling frequency. These two knobs allow different empirical results and should be fine-tuned accordingly to the underlying optimization algorithm.

The layered structure of deep models such as MLPs and CNNs is suitable for progressive layer freezing. The layer freezing technique is applied reversely, distinct from the common practice of transfer learning. The algorithm iteratively reduces the layers' active state in the direction of the network's depth. The previous convergence of the first layers allows their earlier globally shared freezing state. This purpose is complied with the progressive layer state freezing, according to the

direction of the layered structure composing the proposed networks.

5.2.1 Workflow

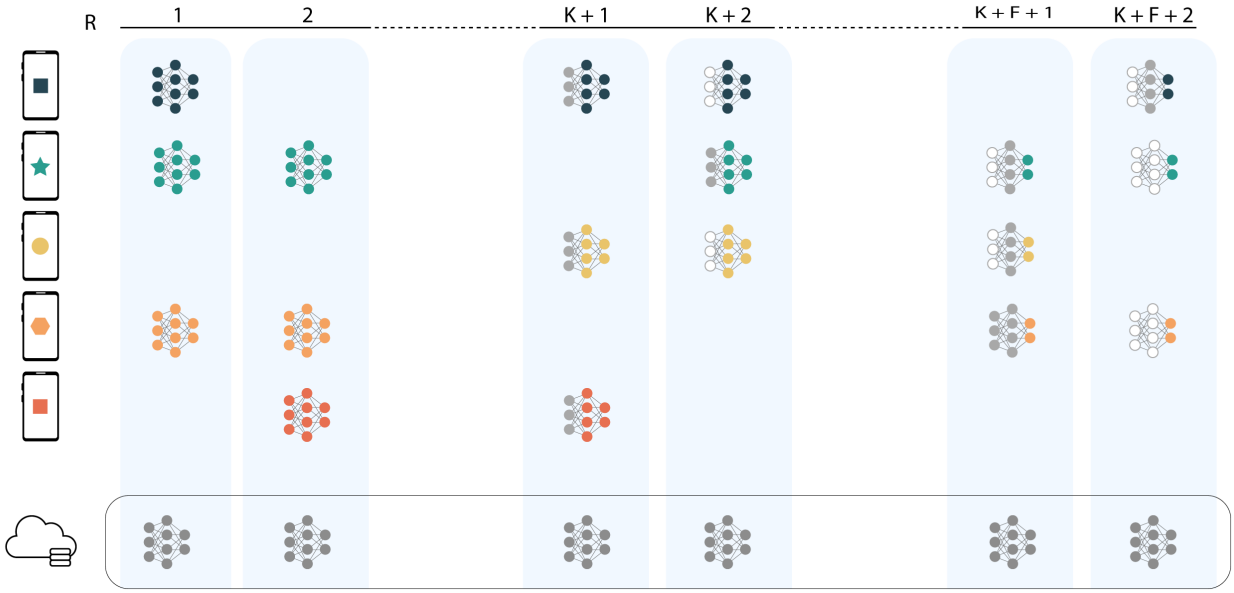

Figure 5.3: FedNILO methodology

In figure 5.3 is shown a graphical representation for the algorithm, deeply treated in the next subsection 5.2.2. The horizontal axis at the top of the image represents the rounds, the steps of standard synchronous FL upon which FedNILO plays out. It starts from one and theoretically does not end, while the dotted lines represent the round shift that elapses between two optimization phases. Each represented round is delimited by a blue box, comprehending the steps occurring within. The left vertical axis represented by devices shows the complete pool of clients (underrepresented as a toy example). The smartphone icons represent hypothetical end-user devices, which could be any end-user devices such as laptops.

Their representation shows different shapes and colors within, to represent the heterogeneous local data, the balancing, and the non-identical and independent distributions. The clients do not always participate in each round because cross-device settings entail partial participation. Their models are present only when they are participating in the represented round. The layers of their models share the same color of their data when they train them. The grey layers represent the parameters received by the server. These parameters are frozen so the clients do not train them anymore. The white dots also represent the frozen parameters, but

they set forth their already updated state. At the bottom lays the cloud server, represented by a cloud icon with the grey global model sketch. The transparent and grey bounded box represents the evolution of the server procedure. The cloud server is always operative and participates at each elapsed round.

At the beginning of each round, the cloud server sends the model (grey) to the reachable clients. The clients then perform the training over local data and send back the optimized model to the server. The server aggregates them and proceeds toward the next round. After K rounds, the server sets the first layer in a frozen state, yet it still needs to communicate it to the clients. The clients receive the model and the index of the first frozen layer. They completely overwrite it ($r = K + 1$), but they only train and send back the layers needed (the colored ones, the grey ones are left equal with the server). In round $K + 2$, the server sends only the complete model to the green client because the blue and yellow have already received the first layer parameters. Thus, they train only the unfrozen layers. At the right of the image, for ($R = K + F + 1$), the orange client receives the complete model because it was never available since before the first layer froze. Accordingly to the defined policy, it optimizes and then transmits back only the last layer.

5.2.2 Implementation

The FedNILO algorithm is computed in a distributed fashion by different devices. The server acts as the global coordinator. It broadcasts and updates the global parameters, while the end-user devices train the model whenever available. The algorithm presentation is separated into two pieces of pseudocode, taking place on the server-side and a generic client-side (every client follows the same procedure).

The algorithm 3 describes the server's procedures. It takes as inputs the number of rounds to be performed, the values of K and F previously introduced. At the beginning (line 1), the server initializes the model parameters. In lines 2 - 13, it starts the iterative process that provides the final model. Each iteration represents a round of FL. In line 3, the server selects the available clients. The algorithm exploits a uniform distribution to emulate the clients' availability. The strategy extracts 10% of the clients and replaces them at every round iteration. In line 4, the server initializes the samples counter. In line 5, $frozenIndex(r, F, K)$ defines I_o in function of the current round and the K/F hyperparameters. I_o is the index of the first layer that remains unfrozen. Every previous layer becomes frozen, at least for training. The subsection 5.2.3 details I_o, the formula to compute it, and its temporal role.

FedNILO (Server)

Parameters:
Rounds T
Start Freezing Round K
Freezing frequency F

Algorithm 3 Server algorithm - FedNILO

1: Initialize $\mathbf{\Theta}_0$; ▷ initialize model parameters on server
2: **for** r (round) in 1...T **do**
3: Sample $\mathcal{S}_k$ subset of reachable clients
4: $n \leftarrow 0$ ▷ training examples counter
5: $I_o \leftarrow frozenIndex(r, F, K)$
6: **for** k (client) in $\mathcal{S}_k$ in parallel **do**
7: $\mathbf{\Theta}_{u,k} \leftarrow reduce([I_{u,k}, end])$
8: $\mathbf{\Theta}_r^k, n_k \leftarrow ClientUpdate(\mathbf{\Theta}_{r-1}^k, I_o)$
9: $n \leftarrow n + n_k$
10: $map(k, I_{u,k})$ ▷ Maps the client id that already updated the frozen layer
11: **end for**
12: $\mathbf{\Theta}_r \leftarrow \frac{1}{n} \sum_k^{|\mathcal{S}_k^r|} n_k \mathbf{\Theta}_r^k$ ▷ update server model through weighted average
13: **end for**
14: return $\mathbf{\Theta}$

Then, between lines 6-10, the algorithm presents a parallel loop. In line 7, for each k-th selected client, the server reduces the global model to the one needed to reduce the communication overhead. In section 5.2.4 takes place the detailed explanation of the communication of the parameters, along with their update and overwrite. In line 8, the server sends the model to the client. The server waits for the clients to train and send their models back. Whenever a k client fulfills its assignment, the server receives the k-th model. In association with the model, it seizes the number of data examples that contributed to the training. In line 9, it updates the data examples counter and maps the client id with the indexes of the updated layers. So that it traces the indexes $I_{u,k}$ required to send the model reduced to the clients' needs. Once it has sampled the trained models of the selected clients, in line 12, it aggregates the model parameters through their average. The server computes the weighted average by summing the pointwise products of the n_k number of data examples and the models' parameters, for every k-th client belonging to $\mathcal{S}_k^r$. Then it normalizes it with the counter n, which is the sum of the

total data points exploited. At the end (line 14), the server returns the globally optimized model $\mathbf{\Theta}$, concluding the training process.

FedNILO (Client)

Parameters:
Epochs E

Algorithm 4 Client algorithm - FedNILO

1: **procedure** $ClientUpdate(\mathbf{\Theta}_r, I_o)$
2: $\quad\mathbf{\Theta}^k \leftarrow \mathbf{\Theta}$
3: $\quad\mathbf{\Theta}^k \leftarrow freeze(I_o)$
4: $\quad$**for** e (epoch) in E **do**
5: $\quad\quad$**for b** in batches **do**
6: $\quad\quad\quad\mathbf{\Theta}^k_{[I_o,end]} \leftarrow \mathbf{\Theta}^k - \eta\nabla l(\mathbf{\Theta}^k; \mathbf{b})$ ▷ local training step
7: $\quad\quad$**end for**
8: $\quad$**end for**
9: $\quad$return $\mathbf{\Theta}^k_{[I_o,end]}$, $num\_examples$
10: **end procedure**

The second algorithm 4 refers to the procedure that takes place over a generic client k, which is selected during an arbitrary round r. Line 2, the selected clients receive the global model parameters $\mathbf{\Theta}$ and the indication of the first layer subject to training I_o.They overwrite their local parameters with the globally incoming ones. The subsection 5.2.4 presents the detailed procedure. Line 3, the client sets every l-layer previous or equal to I_o in the frozen state. Instead, the parameters of any layers l, such that $I_o < l$ are subject to the back-propagation during the training. Once defined, the client starts the training procedure. In lines 4-8, it executes a loop over the prefixed number of epochs. In lines 5 - 7, takes place another loop that cycles over the data batches. The data batches are the result of the local data partitioning, divided in $\mathbf{b}$ batches of a given batch size B shared between clients. In line 6, the clients iteratively update the unfrozen parameters $\mathbf{\Theta}^k_{[I_o,end]}$ performing an SGD step. Line 9, at the end of the cycle, they return the optimized parameters $\mathbf{\Theta}^k_{[I_o,end]}$ along with the number of data examples exploited to carry out the training to the server.

5.2.3 Freezing index

This subsection details the index I_o that defines the frozen state of the layers. The index of the first layer that needs to be optimized is defined accordingly to the following formula, Given K kick-start round, F rounds period regulating frequency, and L the number of layers composing the network architecture:

$$I_o \in \{0, ..., L-1\} \subseteq \mathbb{N},$$

$$I_o(r, K, F) = \begin{cases} 0 & r \leq K \\ \lceil \frac{r-K}{F} \rceil & K \leq r \leq K + (L-1)F \\ L-1 & otherwise \end{cases}$$

Where $\lceil x \rceil$ is the next superior part of the fraction.

It means that for $r \leq K \rightarrow I_o = 0$ each layer $l \in L$ is subject to optimization, therefore to synchronization. For $r = K + 1 \rightarrow I_o = 1$, the server prescribes the frozen state of the first layer. For $r = K + F + 1 \rightarrow I_o = 2$, the server imposes the first and the second layers in the frozen state. Starting from K-th round, and every after F rounds, the server charges the ensuing layer $l = I_o + 1$ the frozen state. Indeed it updates I_o with the last value of l. The server increases I_o until it reaches the last layer $l = L$, which never gets frozen.

Even if the dynamic exchange of the index layer introduces a slight communication overhead, it is unremarkable. A short unsigned integer is sufficient to enumerate each layer of every network, requiring only 16 bits per client per round. It is far smaller than the size of the network parameters. The training developed by clients optimizes only the unfrozen state layers pointed out as $[I_o, end]$. However, the whole model actively participates during the forward propagation needed to compute the loss and its gradient, as previously pictured in Fig. 5.1.

5.2.4 Parameters synchronization

This subsection details the way the server and the clients exchange the model parameters. It also describes how they update and overwrite them while showing how it is supposed to work.

Server mapping for clients' download savings The server maps the clients to keep track of which client has already updated a given layer $k : I_{u,k}$. This mapping operation does not require the devices' state nor gradients, only their ids, and the layers indexes. Indeed, it helps to avoid sending already updated and no

more optimized layers. After K rounds, the server sets the first layer in a frozen state. It still needs to be broadcast to clients because it was involved in the last aggregation. Depending on the ratio of client selection will be completely updated in a few rounds. E.g., the client k-th is available to train the model for the round $K+1$. The client receives the model with the first layer in a frozen state, $I_o = 1$. If at round $K+2$ it is again available, it needs to be updated only over the last $L-1$ layers. Indeed, the first layer parameters would be the same as the one received before.

Download versus upload curves Hypothetically, given the 10% of client selection ratio with uniform independent distribution, one generical device has a probability of 99.5% to train the model at least once in 50 rounds. However, this is a simulation of the real-world scenario, where the independence assumption could not be satisfied. If a device has a full-loaded battery and perfect state, it is likely to be resampled between close rounds. On the other hand, if it consumed its battery to perform the previous round's training, it will not be available for many consecutive rounds. The higher is the client's participation ratio, the faster the server-to-client (s-c) communication curve collapse to the client-to-server (c-s) one. As pictured in Fig. 5.4 which shows for a generic example of FedNILO with $F = 450, K = 50$ the round costs plots of download (blue line) and uploads (orange line).

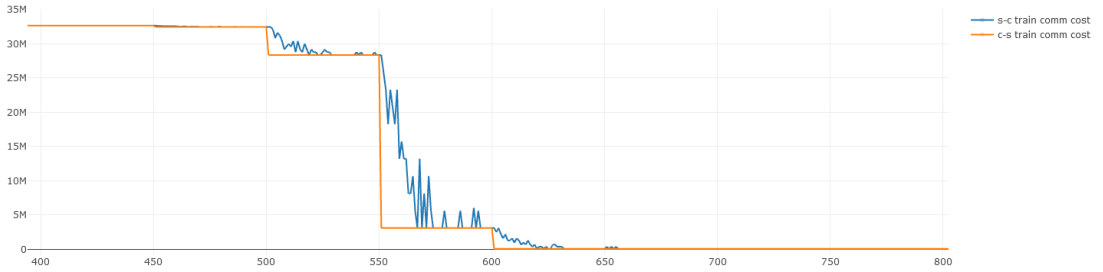

Figure 5.4: FedNILO 450/50 upload(c-s) vs download(s-c) cost in bytes per round

Transmission procedure Both server and clients, flatten the model's parameters before transmitting them to the other party. The procedure suggests that it is avoidable to inform the client about which it must restore and which instead overwrite. Whenever the clients obtain the new flattened parameters can overwrite their local ones. They start to rewrite the local parameters array, starting from the last index, until they completely overwrote the ones received. Then, they can restore the model with the shape of the proper tensors. The server, instead, has tracks of the parameters that must receive and does not require any index. At

every round, it receives only the layers' parameters that have been trained by the clients, globally starting from I_o.

Chapter 6

Experimental Results

This chapter consists of two sections, the experimental setup and experimental results. The first one shows an overview of the foundation employed, a sort of input of the methodology. The second instead shows the results obtained given the setup depicted.

6.1 Experimental setup

This section presents the setup employed to carry out the experiments. It starts with platform that carried out the experiments and continues with the model selected, giving its details in a resuming table. Then it introduces the datasets and the preprocessing occurring in train and test phases. It continues with the partitioning methods to simulate heterogeneous and homogeneous settings. It presents the several hyperparameters required by traditional FL, and in a dedicated paragraph treats the ones introduced by FedNILO. In the end, it shows the metrics applied in the following section.

6.1.1 Hardware and Software

The experiments have been carried out using the standalone simulation paradigm, over a workstation equipped with Nvidia Titan Xp (12 GB). The experiments took advantage of the multi-thread processing. It enabled parallel training that drastically reduces the training time. Nevertheless, it requires approximately 16 hours on average to complete under the proposed stopping conditions. The software configuration is composed of Python3.9.5 for the programming language and PyTorch 1.8.1 for the deep learning environment. Flower [24], a framework for federated learning which is framework agnostic (e.g., TensorFlow or PyTorch for the clients' training), has been modified to avoid the localhost communication,

which presented a bottleneck. In the end, the data processing and partitioning are an adaptation of the library FedML [12] another tool for federated learning.

6.1.2 Model - CIFARCNN

The model employed is a shallow convolutional neural network, which origins come from a tutorial of TensorFlow addressing the classification task of CIFAR10. It did not achieve state-of-the-art results on CIFAR10 but is sufficient for the purpose. Moreover, its architecture is not too deep nor over parametrized. So it is suitable for cross-device settings, which leads it to be commonly mentioned in the literature. [13] initially used the mentioned network. However, the reference link is not available anymore, while a modified version is proposed by [40]. Its description allowed its reconstruction, which claims to differ from [13] proposal because it avoids the use of batch normalization layers. The total parameters of the model are less than 1 million, distributed over two convolutional and three fully connected layers.

Network Structure The network employed for the test is summarized in Table 6.1 and is a shallow model for visual computational tasks, in particular for CIFAR10 and CIFAR100 datasets classification. The first line of the table reports the input image shape that will be processed orderly by the subsequent layers.

Operator (activation)	**Number of filters**	**Kernel size (stride)**	**Weights (bias)**	**Output shape**
Input Image				(32,32,3)
Conv2d (ReLU)	64	5x5 (1)	4800 (64)	(64,28,28)
MaxPool		2x2 (2)		(64,14,14)
Conv2d (ReLU)	64	5x5 (1)	102400 (64)	(64,10,10)
MaxPool		2x2 (2)		(64,5,5)
Linear (ReLU)			630400 (394)	394
Linear (ReLU)			75648 (192)	192
Linear (LogSoftmax)			1920 (10 or 100)	LABELS (10,100)
Total Parameters			815892	

Table 6.1: Cifar CNN architecture

The first layer is composed of a bidimensional convolution (Conv2d), performed by 64 filters with size 5x5, stridden across the images with 1 stride pixel. The output is processed throughout the ReLU activation function and then sampled with the bidimensional max-pooling (MaxPool) technique. MaxPool filters out every textile of 4-pixel neighbors neurons, leaving only the one with the higher magnitude. A subsequent convolutional layer with the same characteristics (Conv2d + ReLU + MaxPool) processes the output of the first layer.

The features extracted are then flattened to be processed by the first fully connected dense filter (Linear). The second convolutional layer and the first dense one compose most of the network parameters. Two more hidden linear layers are then affixed at the end. The last layer is the only one that, instead of ReLU activations, uses the LogSoftmax for the Cross entropy loss computation. The output shape of the ultimate layer depends on the dataset. CIFAR10 and CIFAR100 have 10 and 100 output neurons, respectively. The final row reports, instead, the total number of parameters.

Weights initialization The model weights initialization depends on their pertaining layer type (Convolutional versus Fully Connected). The convolutional layers weights are initialized accordingly to a normal distribution defined as $\mathcal{N}(0, \frac{2}{\sqrt{5x5x64}})$, where 5x5 is the kernel size while 64 represent the number of the channel, shared between both. The bias is instead initialized to zero. Nevertheless the weight distribution of fully connected layers is uniform and it is defined as $\mathcal{U}_{[-s,s]}$ where $s = \frac{1}{\sqrt{\#weights}}$, which depends on the number layer weights. Referring to table 6.1 they are respectively 630400, 75684 and 1920. The bias of fully connected layers is initialized with the same distribution, sharing the same parameter s.

6.1.3 Datasets and partitioning

CIFAR10 (C10) and CIFAR100 (C100) are two standard datasets widely used for FL comparisons. They are medium-sized, composed of 60000 images with three channels (RGB) of 32×32 pixels with int8 precision, with 10 and 100 target labels as suggested by their name. The datasets are available through the torchvision library with a train/test partition of 50000 and 10000 examples, respectively. The training subdivision presented is equally sized with 5000 and 500 examples per class for CIFAR10 and CIFAR100, correspondingly. The train set is partitioned over the clients accordingly to the homogeneous and heterogeneous settings. The following paragraph focuses on the partitioning of clients' data, while the test set exploited to measure the performances is kept centralized on the server.

Preprocessing A different pre-processing pipeline unfolds for the train and test sets. However, the test set shares part of the training processing, the one that does

not provide data augmentation. It is composed of the image transformation into a tensor and the application of the normalization of the RGB channels (CIFAR10 μ = [0.49139968, 0.48215827, 0.44653124] and σ = [0.24703233, 0.24348505, 0.26158768], CIFAR100 μ = [0.5071, 0.4865, 0.4409] and σ = [0.2673, 0.2564, 0.2762]). The training pipeline instead provides data augmentation. It converts the input data to PIL images, first padded by 4 pixels and then randomly cropped back to the original shape of (32,32,3). Then, it randomly flips them along the horizontal axes. It follows the tensor transformation with the normalization steps, which coincide with the ones applied for the test set. Though, it concludes with the cutout transformation. The cutout transformation randomly defines a mask, which has obscures a subset of elements both over the width and height of images, providing a sort of black rectangle applied on the image, with random base, high, and position.

Heterogeneity The dataset exploited is an adaptation coming from the classical deep learning tasks. By so, it has not been purposely designed to represent FL partitions. New federated datasets are emerging on the specialized frameworks. Despite this, CIFAR10 and CIFAR100 are still the baselines of the recent publications. So that, it requires their partitioning to simulate the FL environment. The proposed work replicates it throughout the adaptation of the data loaders provided by [12] framework, both for homogeneous and heterogeneous partitioning.

Homogeneous partitioning - IID and Balanced The homogeneous scenario is used more for the convergence and as a measure towards the baseline shown by [13]. The examples are partitioned across clients in equal balance, such that each client has 500 training images to train its model. The IID assumption considers the label distribution. The target examples follow a balanced separation over the clients. The process randomly shuffles the data points and then partitions them across the devices using the array-split function of NumPy. The label's distribution is not precisely shared and with the same cardinality. However, the result is very close. Each client owns approximately 50 or 5 different targeted images for each label. In fig 6.1 there is a graphical representation of homogeneous partitioning for CIFAR10 dataset. The x-axis represents the clients involved, while the y axis denotes the number of data examples. It is a stacked bar chart, divided by client, where the number of examples is stacked label by label orderly. The graph highlights the balance and the similar distributions, showing almost parallel and equally sized color bars. Every client has approximately the same total number of data examples which is close to 500.

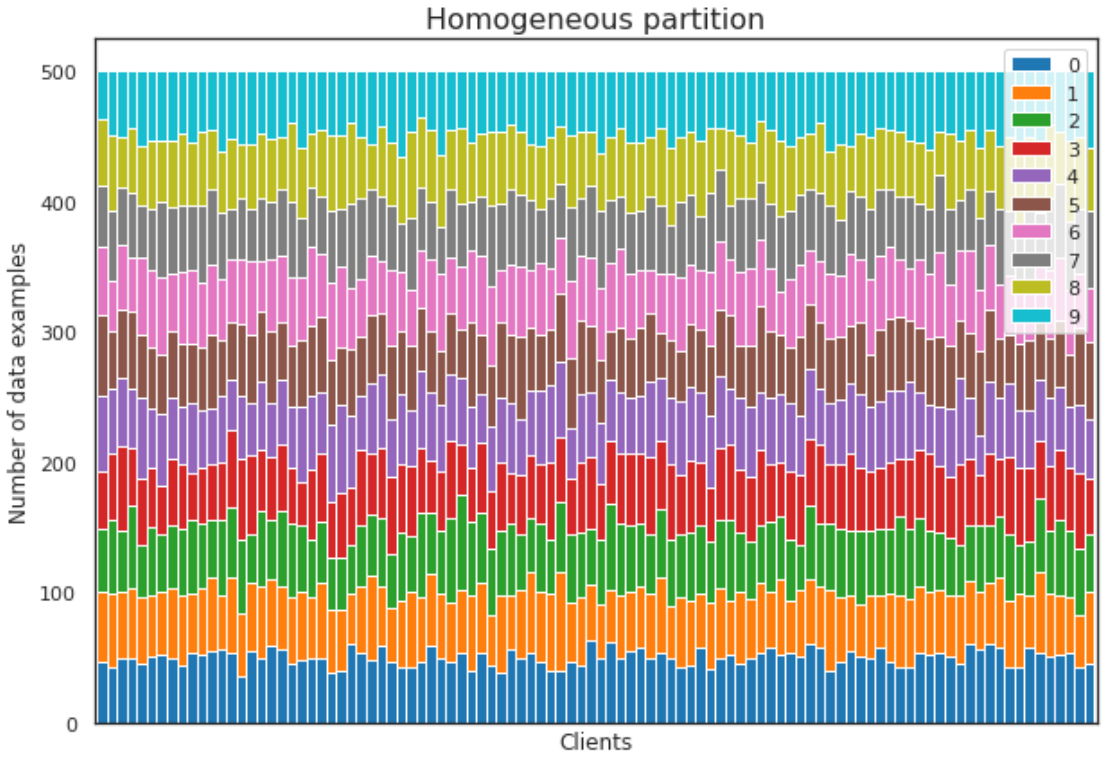

Figure 6.1: Homogeneous data distribution for IID and balanced assumptions across 100 clients

Heterogeneous partitioning - extremely nonIID and unbalanced The heterogeneous partitioning is more sophisticated. Label splits and lognormal distribution demonstrated to be quite far from the real heterogeneity levels. The LDA technique proposed by [12] uses a Dirichlet distribution to allocate the amount of data examples across the devices. The examples are distributed label by label across the C clients with the probabilities defined by a Dirichlet distribution with vector parameter $\alpha \in [0,1]^C$ that concerns the label partitioning over the C clients. The α parameter scales the heterogeneity levels in a range of $(0, \inf)$. Low values of α denote highly heterogeneous settings (lower than 1, $\alpha \to 0$), while higher values (more than 100, $\alpha \to \infty$) converge to the IID balanced assumption. The experiments carried are based on $\alpha = 0.3$, causing a highly heterogeneous scenario. The data points may vary in a range from 50 to more than 1300, while the distribution of the labels could provide one-fold, twofold, or many-folds, independently from the amount, being possible to have 1300 examples of two classes on one client, and 50 examples of 7 classes on one other. Similar to the previous image without lacking comparison there is proposed in fig 6.2 the partitioning performed for heterogeneity assumptions, in particular the one performed with Dir(0.3). The balance is no

more respected because the examples between clients vary a lot, likewise, it does the data distribution.

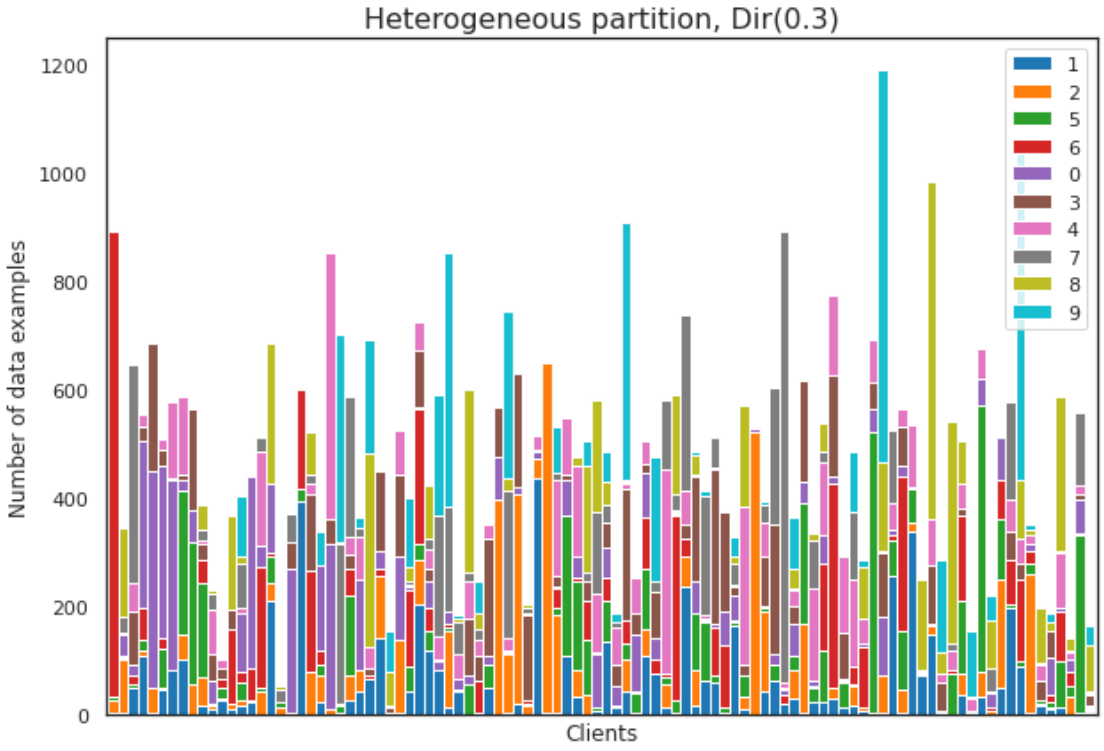

Figure 6.2: Heterogeneous data distribution for IID and balanced assumptions across 100 clients

It is almost impossible to spot geometrical patterns even if the stacking of the labels provides the same order. By taking a look at the first client (on the left), it can be spotted that its dataset's composition relies on only two labels, approximately composed of 900 data examples. Meanwhile, the 20th one has nearly fifty examples and four labels. Finally, approaching the right of the image, one client provides almost 1200 data examples. At the center-left of the image, one client instead provides a single label dataset (orange).

6.1.4 Hyperparameters

Core hyperparameters

To face and respect the cross-device settings while maintaining reasonable training times, the total number of clients amounts to 100. In addition, at each round ten clients are uniformly sampled to compute the training process. It follows the

prevailing technique to simulate massive targets with partial participation. Each selected client performs respectively 5 epochs over B batches of his data, where the batch size is fixed to 50, to propose a comparison with FedAVG.

The referring optimizer is the stochastic gradient descent (SGD), which starts with an initial learning rate of 0.01 and a weight decay of 1e-3. The loss used to perform the backpropagation is the Cross-Entropy Loss. FedAVG requires the learning rate decay to achieve convergence, especially under nonIID data assumptions where the accuracy fluctuations are bold. The learning rate decay fixed to 0.998 is applied, and it modifies the learning rate according to the formula $\eta_t = \eta_0 0.998^t$ depending on the round currently elapsing.

The process ends when at least one of the following constraints is reached:
1) the total number of rounds elapsed reaches 2000;
2) the total number of communication rounds reaches 1000;
In the case of FedAVG, the first constraint is the one that blocks the training, while for FedNILO for the provided hyperparameters is the second one that toggles the training state. Indeed, it relates to the communication dispensed by the two algorithms. The hyperparameters selection grid testing FedNILO leads it not to reach the same communication amount; nor it would under dozens of thousands of rounds, because at the final stage of optimization, the communication of one round of FedAVG implies almost 500 rounds of FedNILO.

Additional Hyperparameters

FedNILO provides two additional knobs to finetune the optimization algorithm, K, and F, which regulate the starting round and the toggle frequency of layer freezing. The hyperparameters' choice is arbitrarily but should depend on the task. The study of the convergence rates of the backbone algorithm employed, or a fixed communication budget benefits the following rule of thumb. The suggestion considers K as $\frac{1}{2}, \frac{1}{3}$ of the number of rounds needed for the target accuracy or the total amount of communication.

K $K \in \mathbb{N}^+$ determines whenever start the optimization, higher values of K allow better final accuracy levels, per contra, it also determines lower communication savings. On the opposite, low values of K lead to highly beneficial communication savings. The peak accuracy reached tends to be lower while the rounds required are highly stretched. For $K = 1$, the optimization starts at the very beginning harshly reducing the performances of the model.
For such reasons, K has been tested for values of $K \in \{350,400,450,500\}$

F $F \in \mathbb{N}^+$ determines instead the period enumerated in rounds which are maintaining the same optimization before toggling the next step, providing a scaling

frequency function (the frequency is the reciprocal of F, lower values of F denote higher frequencies). Here again, the results reported show that low values of F provide better communication savings and are usually associated with low values of K, while experimentally higher values of both are shown for higher final accuracy targets. For $F = 1$ the last optimization phase occurs in L rounds where L corresponds to the number of layers. F has been tested for every K value, for the following values $F \in \{25,50,75\}$

6.1.5 Evaluation metrics

Round (R) One round of synchronous FL considers one cycle of the iterated steps elapsing after every global update. To quickly rehearse them, they are composed of the client selection, the model broadcasting performed by the server, the training of the model parameters accomplished by the selected devices, the update of the devices' local models, and its aggregation on the server. It is a measure that provides a step-based definition timeline, exploited for convergence rates and as a relative measure for other metrics such as accuracy and communication cost.

Communication cost (CC) or Communication Round (CR) The communication cost takes place in two different phases during one round of training. The first one is the download of the model by the clients (server-client interaction, s-c), the second is the upload of the clients' models (client-server interaction, c-s). The two parts of communication that occur compose a communication round, which can be measured in GBs by dumping the message or estimating the size of the model. So if a model size M_s is 3.5 MB the communication round corresponds to $CC = (3.5_{s-c} + 3.5_{c-s}) n_c c_f = 70MB$, where n_c is the total number of clients and c_f is the ratio of selected clients. The message size is not estimated but calculated as the size system function (python) of the dumped message with the pickle library. In these terms, it is possible to use a notation similar to [40] where the communication cost is commuted in rounds, using the definition of communication rounds, denoted as: $2M_s n_c c_f = 1CR$ that is the model size for download and upload tasks, while a halved sized model fills one communication round in two rounds, or oppositely if the gradient is needed to track the client's state, the message size comes to be the double of the model size, so 1 round corresponds to 2 CRs. The two measures have a similar concept, but CC is an empirical measure that shows the effective communication required. While CR is an estimated metric of the communication referred to the steps performed.

Accuracy Accuracy could be measured in many ways in FL. It inherits from classical training the percentage of correctly classified examples over the test set. For research purposes of convergence or optimization, it is useful to consider the

centralized accuracy over a local dataset on the server. In this case, the two datasets are the test cifar10 and test cifar100 already partitioned by the torchvision library. Federated accuracy could be more practical in an effectively cross-device platform, while the results for IID test-sets are very similar, but the testing is particularly slower.

Round accuracy The round accuracy is the accuracy obtained over the test set in a particular round. It provides low-quality information in particular under nonIID settings because the accuracy plots present noise. The accuracy could vary up to 10/15% over a single round. The learning rate decay is needed to reach consensus/convergence in nonIID settings [48], but the number of required rounds could be very high (up to 5-20k rounds).

Top accuracy over a steady space One overcoming solution is to consider the accuracy whenever reaching a steady space. The steady space is represented by a sufficiently large interval presenting low variance between rounds. But again, the steadiness could be hard to reach in a context of limited resources and limited communication budget, moreover the round accuracy jitters a lot under heterogeneous settings.

Mean accuracy of the last L-rounds The other overcoming solution is to consider the average accuracy of the previous L-rounds, better if considered over a steady space. This solution is proposed also in [14] and [33]. It is favorable because knowing the accuracy performances of a model is not trivial in the real-world scenario. The federated evaluation requires additional communication costs. The proposed results consider the mean accuracy of 30 rounds, which is the most representative for the task. Indeed, [14] used 100 rounds of average, bu they tested the algorithms for much many rounds. The number of rounds chosen is the proportion between the elapsed rounds of the two tests.

6.2 Results

This section represents the output comparison between traditional FL and the proposed FedNILO. It starts with the per device-round savings that verify in different steps of the procedure. Then it shows the cumulative communication cost savings that the new methodology could spare. It reports the results for both CIFAR10 and CIFAR100, separately for heterogeneous and homogeneous settings. The heterogeneity assumption is the most interesting because it is close to the real-world application. Nevertheless, it reports the homogeneous assumption for completeness of the results, though interesting to show the generalization capability of the algorithm under agnostic presumptions. Reducing the communication burden represents the scope of the proposed work. Concluded the needed period of massive computation defined by K, FedNILO reaches the prefixed scope, both

considering the single round reduction, but most importantly achieves huge CC savings considering the complete training, ranging from 18% up to 59% of the comprehensive CC depending on the dataset, heterogeneity and the target accuracy. The achieved results perform with different values of K and F. Despite introducing system complexity they can be estimated for the need. The next subsection will present the results under heterogeneous and homogeneous contexts, both for CIFAR10 and CIFAR100, giving insights into the similarities and differences across the tables.

6.2.1 Per round communication savings

The single round-device reduction is helpful to reduce the failure and straggling probabilities of the involved devices. The less a device is required to download, train, and upload, the higher are its odds to successfully submit its contribution. FedNILO harshly reduces the low-end devices' stress time, depending on the optimization phase. The last optimization phase, which considers the refinement of the model, leaving the device almost unaffected. It reduces the required time, preventing adverse events such as battery consumption, network jittering, device failure, or loading concomitances.

Rounds interval	layers	parameters	MB	CR
$[1, K]$	5	815892	3.112	1.000
$[K+1, K+F]$	4	811028	3.093	0.994
$[K+F+1, K+2F]$	3	708564	2.702	0.868
$[K+2F+1, K+3F]$	2	77770	0.296	0.095
$[K+3F+1, S^*]$	1	1930	0.007	0.002

Table 6.2: Cifar CNN size and parameters changing over K/F rounds interval phases (CIFAR10)

Table 6.2 summarizes the parameters to be locally optimized and the communication burden that occurs along to the optimization phases of the algorithm for CIFAR10. Given arbitrary $K, F \in \mathbb{N}^+$ the round statistic reduction is reported below, depending on the optimization phase. Rounds interval column relates to the range of rounds during which the number of layers. It provides for such intervals, the parameters along their statistics. The layers column is the number of layers optimized at that point, the parameters and MB columns instead refer to the number of parameters and the size of the model that need to be optimized and synchronized with the server. The last column instead is the relative comparison for the communication round. At the bottom line, the interval of pertaining round

is $[K + 3F + 1, S^*]$, where S^* denotes the round such that $S^* + 1$ violates the constraints. Unfolding line by line the table highlights the methodology. The number of layers reduces by one, starting from $K + 1$ rounds (line 1), scaling down to 1 with a step of 1 every F round.

Rounds interval	layers	parameters	MB	CR
$[1, K]$	5	833262	3.179	1.000
$[K + 1, K + F]$	4	828398	3.160	0.994
$[K + F + 1, K + 2F]$	3	725934	2.769	0.871
$[K + 2F + 1, K + 3F]$	2	95140	0.363	0.114
$[K + 3F + 1, S^*]$	1	19300	0.074	0.023

Table 6.3: Cifar CNN size and parameters changing over K/F rounds interval phases (CIFAR100)

Similarly, Table 6.3 reports the same logic applied to CIFAR100. The output labels cardinality changes and requires more parameters for the last layer, that updates the statistics. It has ten times the weights, and so the last phase of the algorithm requires ten times the communication cost because of the labels. One final round of CIFAR100 corresponds then to ten final rounds of CIFAR10.

6.2.2 Cumulative communication cost over accuracy

The total communication cost, however, is the most important target of this work. The communication reduction burden for the single devices is not enough without being accompanied by the accuracy-communication tradeoff. FedNILO outperforms standard FL considering the overall CC same levels of average accuracy, up to 59%. In particular, the harshly reduced communication cost encumbering at the final stage of optimization allows more rounds. By so, FedNILO obtained up to 2.5% of accuracy increase, which would not be possible otherwise under the same constraints, still maintaining the communication cost lower of more than 30% respect to the best accuracy of FedAVG. This subsection starts with the trustworthy simulation, but then it studies the results under homogeneous settings. The behavior of the novel method on the opposite extreme distribution assumption is helpful to infer its behavior under agnostic assumptions.

Heterogeneous context results - Dir(0.3)

The most important context roots in highly heterogeneous settings, which are representative of typical real-world scenarios. Yet, FedNILO outperforms standard

FL by strictly reducing the cumulative CC (up to 59%) and obtaining higher levels of accuracy. Yet, still saving more than 30% of the reported CC.

Cifar10 nonIID				
Acc.	CC (GBs)		savings percentage	K/F
	FedAVG	FedNILO		
74.5	45.35	24.58	45.8%	350/25
75.0	51.19	24.59	52.0%	350/25
75.5	60.13	24.65	**59.0%**	350/25
76.0	+	24.7	+	350/25
76.5	+	27.7	+	350/25
77.0	+	30.65	+	350/50
77.5	+	33.76	+	450/50
78.0	+	**39.82**	+	500/75
Avg			52,57%	

Table 6.4: CIFAR10 heterogeneous settings providing Communication Cost over same accuracy levels. Comparison between FedNILO versus FedAVG, reporting the values of K and F achieving the best results. FedNILO achieved higher accuracy scores for higher values of K and F outperforming FedAVG in terms of CC and ACC.

Table 6.4 reports the comparison of the results between FedAVG and FedNILO for CIFAR10 Dir(0.3) in terms of communication costs, providing for both the total GBs sent over the internet, the relative saving percentage, and the settings of K/F that achieved the results. As can be noted the accuracy levels are proportional to the values of K and of F by order, indeed in the last column, K/F equals 350/25 is present for the first five examples, then followed by 350/50, 450/50 and 500/75. The FedNILO CC trend reproduces the more abstract concept of table 6.2, for only one accuracy point increasing standard FL needs 15 GBs of communication. While FedNILO required only 0.07 GBs, obtaining in comparison to the top-accuracy score of FedAVG a savings percentage of 59% of CC savings. The value is highlighted for better exposition. The line, that rules the table below the first three values denotes the accuracy that has been possible to achieve without violating the constraints. More importantly, still sparing a portion of the standard FL CC. Even if the comparison cannot be provided, it must be noted that the last highlighted value has more than 30% of communication savings concerning the standard FL with 2.5% of increased accuracy. In the end, the bottom row reports the average saving percentage for the three comparable values, under the range of

accuracy of 75% FedNILO(350/25) spares more than 50% of the communication overhead.

Cifar100 nonIID				
Acc.	CC (GBs)		savings percentage	K/F
	FedAVG	FedNILO		
40.5	42.9	25.68	27.4%	350/25
41.0	47.69	26.19	45.1%	350/25
41.5	59.3	26.81	54.8%	350/25
42.0	+	29.81	+	350/50
42.5	+	38.73	+	500/50
Avg			42.43%	

Table 6.5: CIFAR100 heterogeneous settings providing Communication Cost over same accuracy levels between FedNILO and FedAVG. It reports the saving percentage along with the values of K and F that obtained the best score reported. FedNILO outperforms FedAVG in terms of final accuracy and CC. In the end, it reports the average savings percentage.

The Table 6.5 reports instead the results obtained for CIFAR100 also under heterogeneous settings. Again the proportionality trend that bounds the increasing values of F and K follow the increasing top accuracy, similarly to the CIFAR10, with the major difference that the accuracy boost comes only with one additional percentage point with similar costs and savings for the top accuracy reached by standard FL (lines 3 and 5). Also, the savings reported are quite lower, especially for the first line. Even if the datasets' characteristics are similar, become quite different under FL training. The training images for each target label are reduced from 5000 to 500, still, need to be partitioned across clients. However, FedNILO still outperforms FedAVG with the same grid search showing a stability trend of results under different datasets, achieving an average in this case of 42.43% of CC savings (last line).

Homogeneous context results - IID

The homogeneous context is less interesting from an empirical point of view. The literature comes to be interested in it for convergence rates studies more than empirical results because of its convex and bounded empirical error theoretical demonstration. However, it has been tested and studied to check the FedNILO performances. This test denotes, or at least suggests, its capability under the agnostic assumptions, whenever the alpha parameter is not known a priori. By so

it is useful to consider the extremal opposite to the heterogeneity setup. FedNILO performances are demonstrated to be lower but still valid, indicating a stabile capacity supported by 30% of CC savings on average for IID and balanced settings. Unfortunately, it could not obtain higher accuracy scores, which probably happened because of a generalization capability of FedNILO under the Dirichlet distribution. While the optimization for the smooth and convex optimization typical of homogeneous settings could suffer a bit more of layer removal optimization.

Cifar10 IID				
Acc.	CC (GBs)		savings percentage	K/F
	FedAVG	FedNILO		
80.5	33.26	27.3	17.9%	350/50
81.0	41.16	30.58	25.7%	400/50
81.5	55.14	39.64	28.1%	500/75
Avg			23.9%	

Table 6.6: CIFAR10 homogeneous settings Communication Cost for same accuracy levels: comparison between FedAVG and FedNILO reporting the values obtaining the best performance. The last line reports the average CC savings obtained.

Table 6.6 reports the results achieved for CIFAR10 under homogeneous settings. While the proportionality pattern is still respected. FedNILO did not overcome the traditional FL accuracy scores under the imposed constraints. The main difference comes with the absence of 350/25 instead present in each reported table, starting directly from 350/50 (line 1). This is the case where FedNILO performed worst, presenting the lowest CC saving percentage of 17.9% (line 1 - 80.5% of accuracy reference) by an average of 24% (the last line reporting the average percentage savings). Nevertheless, the results reported are still quite impressive, exhibiting savings that relieve the communication burden of more than 6 and 15 GBs of network traffic cost.

Cifar100 IID				
Acc.	CC (GBs) FedAVG	CC (GBs) FedNILO	savings percentage	K/F
40.5	39.74	26.17	34.1%	350/25
41.0	47.19	32.54	31.0%	400/25
41.5	57.68	34.92	39.5%	450/50
Avg			34.87%	

Table 6.7: CIFAR10 homogeneous settings Communication Cost for same accuracy levels: comparison between FedAVG and FedNILO reporting the values obtaining the best performance. The last line reports the average CC savings obtained.

In the end, Table 6.7 presents the results of CIFAR100 under IID and balanced settings fashion. Here also the values K/F represent the same increasing pattern, but while 350/25 still occupies the first line (40.5% of accuracy), $K = 500$ does not figure at all. While the previous results tables had an increasing trend of savings, comparing the percentages of the second line with the first, stands out a drop of CC saving percentage, from 34.1% to 31%, finally fixed for the last values reported of 39%. This is due to the scaling of hyperparameters. The provided grid search probably does not fit at best the accuracy space. In the end, the average CC savings of CIFAR100 IID evidenced at the bottom line of the table hovers to 35%. It proves again the partial stability of FedNILO and its hyperparameters, over different heterogeneity levels and different datasets.

Chapter 7

Conclusions

This thesis introduced FedNILO, an innovative methodology to reduce the communication cost in Federated Learning. It explores the context, the importance, the taxonomy of FL, and its characteristical concepts. It focuses on the cross-device FL and states the communication cost as the principal challenge. Communication cost under cross-device settings becomes crucial because it puts strain on the devices. The devices are not dedicated servers equipped with wired networks and high computational capabilities. Indeed, the higher the load, the more they are prone to failures and delays. The communication drains the batteries that supply them and commits the smartphones' resource, ruining the owner's user experience. FedNILO, as the related works, comes from the traditional algorithm FedAVG, the pioneer of the topic. It ensures the independence between communication and data, by communicating the model parameters.

FedNILO adapts and applies the concept of layer freezing exploiting the previous convergence of the layers close to the input. It gradually reduces the model parameters that are trained and synchronized with the server, becoming more suitable for low-end devices. The freeze occurs in a layer-wise fashion, targeting layered-architecture networks commonly used for visual classification. Standing out from pruning or skipping, the model only freezes the layers instead of removing them, in the following the topological direction. However, whenever the server freezes a new layer, it ensures its communication to be avoidable once updated on the clients. The clients receiving the freezing instruction only train and synchronize the profitable layers.

FedNILO generates a slight complexity into the system, introducing two knobs K and F. However, their estimation can take place in light of the communication budget or the convergence rate. Moreover, K and F choices present a unidirectional pattern: their values' rise is proportional to the target accuracy score and the communication cost. The benefit of K and F is to regulate the kickstart and the scaling frequency of the FedNILO optimization, allowing its general and fine-grained

adaptability to different setups. K and F allow to trade-off the balance between accuracy and communication cost, allowing fine-tunable results depending on the principal scope.

This work provides the testing on traditional visual classification datasets, CIFAR10 - CIFAR100, suitably partitioned for realistic industrial implementations. For such conditions and respectively, FedNILO outperforms FedAVG; (i) gradually reducing the round-device's payload of interest to negligible levels (0.2% / 2%), (ii) Optimizing the cumulative communication cost for equal accuracy scores (from 45.8% / 27.4% up to 59% / 54.8%), (iii) reaching higher accuracy scores still using less communication (35% CC saving, ACC: +2.5% / +1%).

In the end, to test the agnostic capability of FedNILO, this work proposes the results for homogeneous partitioning. As the opposite extreme of heterogeneous, the partitioning is IID, balanced, and not representative of industrial representations. Nevertheless, it hints at FedNILO's capability of generalization for agnostic settings. FedNILO, by definition, still gradually reduced the round-device's burden. However, it showed incapability to reach higher accuracy and the same levels of communication savings. Yet, it demonstrated its stability, outperforming FedAVG in terms of cumulative CC (from 17.9% / 34.1% to 28.1% / 39.5%).

Overall the results show the benefits of the proposal. The proposed solution gradually diminishes the number of layers along the topological order. Therefore, it allows more rounds, distributes the load over more devices, increases the accuracy, and halves the communication cost required.

Bibliography

[1] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. «Mobilenets: Efficient convolutional neural networks for mobile vision applications». In: *arXiv preprint arXiv:1704.04861* (2017) (cit. on p. 1).

[2] Kunal Chandiramani, Dhruv Garg, and N Maheswari. «Performance analysis of distributed and federated learning models on private data». In: *Procedia Computer Science* 165 (2019), pp. 349–355 (cit. on p. 2).

[3] Peter Kairouz et al. «Advances and open problems in federated learning». In: *arXiv preprint arXiv:1912.04977* (2019) (cit. on pp. 3, 12, 14–16, 18, 24–29, 31, 32).

[4] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. «Big Data technologies: A survey». In: *Journal of King Saud University-Computer and Information Sciences* 30.4 (2018), pp. 431–448 (cit. on pp. 3, 8).

[5] Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. «A survey of machine learning for big data processing». In: *EURASIP Journal on Advances in Signal Processing* 2016.1 (2016), pp. 1–16 (cit. on pp. 3, 8, 9).

[6] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. «Federated optimization in heterogeneous networks». In: *arXiv preprint arXiv:1812.06127* (2018) (cit. on pp. 3, 37).

[7] Mohammed Aledhari, Rehma Razzak, Reza M Parizi, and Fahad Saeed. «Federated learning: A survey on enabling technologies, protocols, and applications». In: *IEEE Access* 8 (2020), pp. 140699–140725 (cit. on pp. 3, 20).

[8] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. «Federated Learning on Non-IID Data: A Survey». In: *arXiv preprint arXiv:2106.06843* (2021) (cit. on pp. 3, 18, 26–28).

[9] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. «Federated learning with non-iid data». In: *arXiv preprint arXiv:1806.00582* (2018) (cit. on pp. 3, 36, 43).

[10] Lingjuan Lyu, Han Yu, and Qiang Yang. «Threats to federated learning: A survey». In: *arXiv preprint arXiv:2003.02133* (2020) (cit. on pp. 3, 32).

[11] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. *Federated Machine Learning: Concept and Applications*. Feb. 2019 (cit. on pp. 3, 20).

[12] Chaoyang He et al. «Fedml: A research library and benchmark for federated machine learning». In: *arXiv preprint arXiv:2007.13518* (2020) (cit. on pp. 3, 12, 21–25, 40, 52, 54, 55).

[13] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. «Communication-efficient learning of deep networks from decentralized data». In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282 (cit. on pp. 4, 5, 16, 21, 23, 24, 33, 42, 52, 54).

[14] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečny, Sanjiv Kumar, and H Brendan McMahan. «Adaptive federated optimization». In: *arXiv preprint arXiv:2003.00295* (2020) (cit. on pp. 5, 36, 59).

[15] *Recommendation Y.4000/Y.2060 (06/12) - Overview of the internet of things*. 2012/2016. URL: https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=y.2060 (cit. on p. 7).

[16] Roberto Minerva, Abyi Biru, and Domenico Rotondi. «Towards a definition of the Internet of Things (IoT)». In: *IEEE Internet Initiative* 1.1 (2015), pp. 1–86 (cit. on p. 8).

[17] Ahnaf Hannan Lodhi, Barış Akgün, and Öznur Özkasap. «State-of-the-art techniques in deep edge intelligence». In: *arXiv preprint arXiv:2008.00824* (2020) (cit. on p. 8).

[18] Vishakh Hegde and Sheema Usmani. «Parallel and distributed deep learning». In: *Tech. report, Stanford University*. 2016 (cit. on p. 9).

[19] Itamar Arel, Derek C Rose, and Thomas P Karnowski. «Deep machine learning-a new frontier in artificial intelligence research [research frontier]». In: *IEEE computational intelligence magazine* 5.4 (2010), pp. 13–18 (cit. on p. 9).

[20] Fangxin Wang, Miao Zhang, Xiangxiang Wang, Xiaoqiang Ma, and Jiangchuan Liu. «Deep Learning for Edge Computing Applications: A State-of-the-Art Survey». In: *IEEE Access* PP (Mar. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.2982411 (cit. on p. 9).

[21] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. «Edge computing: Vision and challenges». In: *IEEE internet of things journal* 3.5 (2016), pp. 637–646 (cit. on p. 10).

[22] Valentino Peluso. «Optimization Tools for ConvNets on the Edge» (cit. on pp. 10, 30).

[23] Google. *Tensorflow federated learning*. `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`. 2017 (cit. on p. 13).

[24] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D Lane. «Flower: A Friendly Federated Learning Research Framework». In: *arXiv preprint arXiv:2007.14390* (2020) (cit. on pp. 15, 17, 51).

[25] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. «Federated Learning: Challenges, Methods, and Future Directions». In: *IEEE Signal Processing Magazine* 37.3 (May 2020), pp. 50–60. ISSN: 1558-0792. DOI: `10.1109/msp.2020.2975749`. URL: `http://dx.doi.org/10.1109/MSP.2020.2975749` (cit. on pp. 16, 19, 29).

[26] Javier Fernandez-Marques. *FL flower*. `https://flower.dev/blog/2020-12-16-running_federated_learning_applications_on_embedded_devices_with_flower/`. 2020 (cit. on p. 17).

[27] KM Jawadur Rahman, Faisal Ahmed, Nazma Akhter, Mohammad Hasan, Ruhul Amin, Kazi Ehsan Aziz, AKM Muzahidul Islam, Md Saddam Hossain Mukta, and AKM Najmul Islam. «Challenges, applications and design aspects of federated learning: A survey». In: *IEEE Access* 9 (2021), pp. 124682–124700 (cit. on pp. 17–20, 22, 31, 32).

[28] Mehdi Salehi Heydar Abad, Emre Ozfatura, Deniz Gunduz, and Ozgur Ercetin. «Hierarchical federated learning across heterogeneous cellular networks». In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 8866–8870 (cit. on p. 21).

[29] Anusha Lalitha, Osman Cihan Kilinc, Tara Javidi, and Farinaz Koushanfar. «Peer-to-peer federated learning on graphs». In: *arXiv preprint arXiv:1901.11173* (2019) (cit. on p. 21).

[30] Anusha Lalitha, Shubhanshu Shekhar, Tara Javidi, and Farinaz Koushanfar. «Fully decentralized federated learning». In: *Third workshop on Bayesian Deep Learning (NeurIPS)*. 2018 (cit. on p. 21).

[31] Yuzheng Li, Chuan Chen, Nan Liu, Huawei Huang, Zibin Zheng, and Qiang Yan. «A blockchain-based decentralized federated learning framework with committee consensus». In: *IEEE Network* 35.1 (2020), pp. 234–241 (cit. on p. 21).

[32] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. «Federated learning with matched averaging». In: *arXiv preprint arXiv:2002.06440* (2020) (cit. on pp. 26, 30, 35, 37).

[33] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. «Model pruning enables efficient federated learning on edge devices». In: *arXiv preprint arXiv:1909.12326* (2019) (cit. on pp. 30, 38, 59).

[34] Daliang Li and Junpu Wang. *FedMD: Heterogenous Federated Learning via Model Distillation*. 2019. arXiv: `1910.03581 [cs.LG]` (cit. on p. 30).

[35] URL: `https://unctad.org/page/data-protection-and-privacy-legislation-worldwide` (cit. on p. 31).

[36] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. «Scaffold: Stochastic controlled averaging for federated learning». In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5132–5143 (cit. on pp. 33, 37).

[37] Yang Chen, Xiaoyan Sun, and Yaochu Jin. «Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation». In: *IEEE transactions on neural networks and learning systems* 31.10 (2019), pp. 4229–4238 (cit. on pp. 33, 38).

[38] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. «Deep gradient compression: Reducing the communication bandwidth for distributed training». In: *arXiv preprint arXiv:1712.01887* (2017) (cit. on pp. 33, 39).

[39] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. *On the Convergence of FedAvg on Non-IID Data*. 2020. arXiv: `1907.02189 [stat.ML]` (cit. on p. 35).

[40] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas, Matthew Mattina, Paul Whatmough, and Venkatesh Saligrama. «Federated learning based on dynamic regularization». In: *International Conference on Learning Representations*. 2020 (cit. on pp. 37, 52, 58).

[41] Chen Chen, Hong Xu, Wei Wang, Baochun Li, Bo Li, Li Chen, and Gong Zhang. «Communication-Efficient Federated Learning with Adaptive Parameter Freezing». In: *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2021, pp. 1–11 (cit. on pp. 39, 42).

[42] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. «FedPAQ: A Communication-Efficient Federated Learning Method with Periodic Averaging and Quantization». In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 2021–2031. URL: https://proceedings.mlr.press/v108/reisizadeh20a.html (cit. on p. 39).

[43] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. *Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification*. 2019. arXiv: 1909.06335 [cs.LG] (cit. on p. 39).

[44] Yuqing Gao and Khalid M Mosalam. «Deep transfer learning for image-based structural damage recognition». In: *Computer-Aided Civil and Infrastructure Engineering* 33.9 (2018), pp. 748–768 (cit. on p. 41).

[45] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. «Similarity of neural network representations revisited». In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3519–3529 (cit. on p. 41).

[46] Kelam Goutam, S Balasubramanian, Darshan Gera, and R Raghunatha Sarma. «LayerOut: Freezing Layers in Deep Neural Networks». In: *SN Computer Science* 1.5 (2020), pp. 1–9 (cit. on p. 41).

[47] Francesco Ponzio, Gianvito Urgese, Elisa Ficarra, and Santa Di Cataldo. «Dealing with Lack of Training Data for Convolutional Neural Networks: The Case of Digital Pathology». In: *Electronics* 8 (Feb. 2019), p. 256. DOI: 10.3390/electronics8030256 (cit. on p. 42).

[48] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. «On the convergence of fedavg on non-iid data». In: *arXiv preprint arXiv:1907.02189* (2019) (cit. on p. 59).