Bioinformatics Program

Technical University of Munich

Ludwig-Maximilians-Universität München

Bachelor's Thesis in Bioinformatics

Communication-Efficient Approaches to Federated Deep Neural Networks

Adrian Edward Thomas Henkel

Bioinformatics Program

Technical University of Munich

Ludwig-Maximilians-Universität München

Bachelor's Thesis in Bioinformatics

Communication-Efficient Approaches to Federated Deep Neural Networks (English)

Kommunikationseffiziente Ansätze für föderate künstliche neuronale Netzwerke (Deutsch)

Author:	Adrian Edward Thomas Henkel
Supervisor (Mentor):	Reza Nasirigerdeh, Technical University Munich
Advisors:	Prof. Dr. Jan Baumbach, University of Hamburg
	Dr. Josch Pauling, Technical University Munich
Submitted:	15.03.2021

Statement of Originality

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Date	Adrian Henkel

Acknowledgments

First of all, I want to thank Prof. Dr. Jan Baumbach and Dr. Josch Pauling for giving me the opportunity to work on this thesis and for the general supervision of the project.

Next, I would like to thank my mentor Reza Nasirigerdeh for always supporting me throughout the whole project, helping me with debugging the simulation-framework, checking the results, and proofreading the thesis report.

My gratitude goes to Vanessa Schöne, who helped me with creating the figures for the background where my designing skills were limited.

I also want to thank my wonderful girlfriend, who has always supported me and distracted me from work at the right times.

Lastly, I want to express my thankfulness to my always supporting parents and grand mothers who have sustained me thorough this whole journey.

Abstract

Due to new data mining technologies and the resulting large amounts of data, automated analysis methods have to be used in order to gain valuable information about biological pathways, protein structure or tissue phenotype. Since conventional algorithms often lack the ability to generalize, machine learning, a methodology that aims to create iterative self learning computational models on centralized data, has recently provided significant success. Artificial neural networks are an advanced machine learning algorithm inspired by the biological brain, that consist of multiple interconnected layers. The training process aims to find the optimum model that achieves the best classification of the training data. Although this methodology has already been applied successfully in many fields, it is associated with privacy concerns if training is performed in a centralized manner. To tackle this challenge, *federated learning* was introduced, where multiple and possibly geographically independent clients collaboratively train a global model on their respective local dataset without sharing their data but only sending the model parameters to the server and vice-versa, enabling privacy-preserving model training. With increased model size, the network bandwidth that is mandatory, to exchange the model parameters between the server and the clients, becomes a bottleneck. In order to cope with this challenge, different communication-efficient approaches have been developed including *gradient quantification*, *gradient sparsification* and more local updates using the *FederatedAveraging* algorithm. In this thesis, we first present theoretical formulas to model the network bandwidth usage of each approach. Next, we develop a software framework to simulate a federated deep neural network training using the different communication-efficient approaches. Finally, we evaluate the approaches from the accuracy and network efficiency perspectives using multiple models and datasets under independent and identically distributed (IID) data across the clients.

Aufgrund neuer Data-Mining-Technologien und den daraus entstehenden immer größer werdenden Datenmengen müssen automatisierte Analysemethoden eingesetzt werden, um wertvolle Informationen über beispielsweise biologische Reaktionswege, Proteinstrukturen oder Gewebephänotypen zu gewinnen. Da konventionellen Algorithmen oft die Fähigkeit zur Abstraktion fehlt, hat das maschinelle Lernen, eine Methodik, die darauf abzielt, iterativ selbstlernende Computermodelle auf zentralisierten Daten zu erstellen, in letzter Zeit bedeutende Erfolge erzielt. Ein künstliches neuronales Netzwerk ist ein, von der Struktur des biologischen Gehirns inspirierter, maschineller Lernalgorithmus, welcher aus mehreren miteinander verbundenen Schichten besteht. Der Trainingsprozess zielt darauf ab, eine optimale Belegung der einzelnen Modellparameter zu finden, die die beste Klassifizierung der Trainingsdaten erreicht. Obwohl diese Methodik bereits in vielen Bereichen erfolgreich eingesetzt wurde, ist sie beispielsweise mit Datenschutzbedenken verbunden, wenn das Training zentralisiert durchgeführt wird. Um diese Herausforderung zu bewältigen, wurde das *Föderate Lernen* eingeführt, bei dem mehrere potenziell geografisch unabhängige Klienten kollaborativ ein globales Modell auf ihrem jeweiligen lokalen Datensatz trainieren, ohne ihre Daten zu teilen. Anstelle der einzelnen Datenpunkte werden nur die lokal berechneten Gradienten an den Server beziehungsweise an die Klienten gesendet. Dies ermöglicht ein datenschutzgerechtes Training. Mit zunehmender Modellgröße entwickelt sich jedoch die Bandbreitennutzung, welche bei dem Austausch der Modellparameter zwischen dem Server und den Klienten entsteht, zu einem Engpass. Um diese

Herausforderung zu bewältigen, wurden verschiedene kommunikationseffiziente Ansätze entwickelt, darunter "*gradient quantification*", "*gradient sparsification*" und "*more local updates*". In dieser Arbeit stellen wir zunächst theoretische Formeln zur Modellierung der Bandbreitennutzung der einzelnen Ansätze vor. Anschließend stellen wir ein Software-Framework vor, um das Training eines föderierten künstlichen neuronalen Netzwerks mit den verschiedenen kommunikationseffizienten Ansätzen zu simulieren. Schließlich evaluieren wir die Ansätze in Bezug auf ihre Genauigkeit und der effektiven Nutzung der Bandbreite, unter Verwendung von mehreren Modellen und Datensätzen mit unabhängigen und identisch verteilten (IID) Daten.

List of Abbreviations

ANN Artificial Neural Networks.

CASP Critical Assessment of Structure Prediction.

CCH Colorectal Cancer Histology (dataset).

CNN Convolutional Neural Network.

CT Computer Tomography.

DL Deep Learning.

DNN Deep Neural Networks.

FedGD Federated Gradient Decent.

FMNIST Fashion-MNIST (dataset).

GD Gradient Descent.

GQ Gradient Quantification (approach).

GS Gradient Sparsification (approach).

ML Machine Learning.

MNIST Modified National Institute of Standards and Technology.

MRI Magnetic Resonance Imaging.

MU Multiple Local Updates (approach).

NN Neural Network.

SGD Stochastic Gradient Descent.

UML Unified Modeling Language.

Contents

1. Introduction

Machine learning is one of the biggest fields of research in modern computer science and thus important for all domains, especially for those dealing with large amounts of data. It addresses the challenge of creating computational models, that improve themselves automatically through iterative training processes and has already been applied to multiple areas as biology, bio-medicine and, bioinformatics [23, 32, 19, 13]. Multiple machine learning algorithms have been developed over the years as linear/logistic regression, support vector machines, decision trees or K-Means but deep neural networks (DNN) have achieve higher performance on high-dimensional data [36].

Deep learning or deep neural networks are an advanced form of artificial neural networks (ANNs) which is an interconnected layer-wise model in which the layers are determined by the output of its predecessors. They differ from conventional ANNs by having multiple, so called, hidden layers which increases the size but also the ability of a more complex analysis of the data. Although DNNs have shown significant success, they pose various challenges too. First of all, after a DNN is trained, it is almost impossible to determine the correlations of the different dimensions the model has recognized and thus to provide meaningful interpretations that can lead to a progress of understanding the subjects matter. To tackle this problem, many methods have been proposed to make deep neural networks interpretable [37, 18]. Another challenge is that the *computational complexity* of the training process is very high, and therefore, time-consuming, even with state of the art hardware components if the model size and data is very large [29].

The centralized training poses *data privacy*, issue, which is very relevant to the General Data Protection Regulation (GDPR), coming into effect in 2018 [1]. In centralized training, the data requires to be moved to a centralized site, which might not be possible due to the privacy policies of the institutions [35]. To tackle this challenge, *federated learning* was introduced by Google in 2016 [15].

Federated learning is a machine learning approach that uses multiple, possibly geographically independent clients to train a (global) model collaboratively without sharing their data but only sending the model parameters. The fact that this methodology is preserving the privacy of the data hold by the clients, makes it interesting for learning tasks in which highly sensitive data (e.g. medical data) is used in order to optimize a model [20]. With the clients storing only a subset of the total data and updating their models simultaneously (data parallelism), the computation time for training is reduced and thus the computational costs are decreased without having a significant loss on the model performance [21]. Although this approach is an efficient and privacy-preserving machine learning methodology, the bottleneck shifts from computational costs to communication costs that emerge when sending the model parameters between the clients and the server, especially for federated deep neural networks that contain millions of model parameters.

In order to cope with the network communication challenge in federated deep neural networks, different approaches know as communication-efficient approaches have been developed [21, 4, 29]. They can be categorized into three general categories: (1) Gradient quantification (GQ) which uses lower-bit representation of the gradient values; (2) gradient sparsification (GS) which exchanges a fraction of gradients instead of the whole; (3) More local updates in the clients (MU) which performs higher number of local updates to

converge to the optimum in a fewer number of iterations.

In this thesis, we first introduce a theoretical formula for the network bandwidth usage of each approach. Next, we present the simulation framework we developed to simulate the aforementioned communication-efficient approaches to federated deep neural networks. Finally, we discuss the simulation results (accuracy and network bandwidth usage) for each approach under different scenarios.

The structure of the thesis is as follows. In chapter 2, we first provide an overview about machine learning and its application in biology and bio-medicine. We then explain the principles behind artificial neural networks and federated learning, why it is important and what are the associated problems. Together with their theoretical formulas that calculate the bandwidth usage, we introduce the three evaluated approaches are in chapter 3. Chapter 4 describes the implemented software structure, the architectures of the evaluated models and the training and testing data. In chapter 5 we describe the results of the simulation i.r.t. to the accuracy and the bandwidth usage. The final chapter 5 aims to summarize the work, to discuss the results and to give an insight in the future work.

2. Background

This chapter aims to provide the required background for further understanding of Machine Learning (ML), its application in modern biology and the need for communication-efficient approaches to federated ML. Section 2.1 explains the need for automated analysis methods for the increasing amount of generated biological data and shows why ML helps to gain a deeper insight into biochemical processes. In section 2.2, the methodology of Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNNs) are presented along with the Gradient Descent (GD) optimization algorithm. The last section introduces the functionality of Federated Learning and Federated Gradient Descent (FedGD) and demonstrates why communication efficient approaches are necessary for large federated ML models.

2.1 Machine Learning and Biology

With the sequencing of the whole human genome in 2001, the following development of the first high-throughput DNA sequencer in 2005 and the drastically lowered costs (from 0.5-1 billion to less than 1000 USD per genome) for sequencing, a new generation of data mining and thus data analysis was born [24]. The new field that emerged from this success is named Genomics. Later other revolutionary "omics" approaches followed such as Proteomics, Transcriptomics or Metabolomics which can be summarized as the so called "OMICS-revolution" [28] that aims to gain new information from the new acquired data and to create predictive models (Fig. 2.1). Accompanying with this, new tasks such as function and structure prediction of proteins, finding of transcription factor binding sites, revealing of unknown metabolic pathways or re-purposing of drugs [26] have emerged. Figure 2.2 shows the challenges of biology and bioinformatics and how the different subfields intersect with each other. The wealth of data that can be obtained from new technologies increased drastically in the following years, that is why it became inevitable that, with a parallel increment of computational power, new automated analysis methods had and have to be developed. This was the beginning of bioinformatics. But not only sequence based algorithms for analyzing DNA, RNA or amino acid sequences are topic of todays research, analysis of images generated by Computer Tomography (CT), Magnetic Resonance Imaging (MRI), Ultrasound, X-Ray or Histology protocols demand for automated methods. Conventional algorithms mostly define a set of already known patterns or scenarios and help to categorize the applied data. However, they are not suitable for tasks with large amount of presumably unknown data because they lack the ability to generalize and the power to obtain new information. To tackle this problem, Machine Learning (ML), a sub-field of Artificial Intelligence comes into play which aims to create a computational model that learns and predicts patterns based on data [3]. One can distinguish

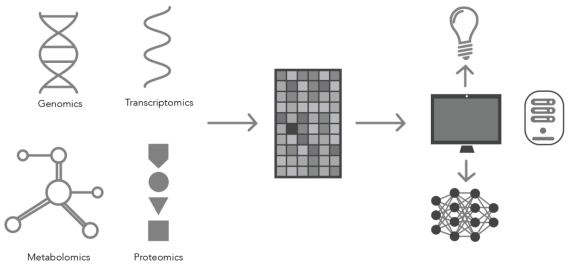


Figure 2.1: The idea behind the "OMICS"-revolution: Generate information and predictive computational models from biological data.

between two main types of machine learning namely supervised learning, which learns a relationship between input variables and dependent output variables (labels) which then is used to predict unseen variables. Unsupervised learning however tries to recognize patterns and to cluster data into unknown categories. Deep Learning (DL) however is a special type of Machine Learning algorithm which is inspired by the structure of the brain an can be both, supervised or unsupervised. The functionalities and architectures of DL is explained in detail in chapter 2.2. ML has shown success already in many biological studies and helped for example to discover RNA-binding regulatory proteins and to predict non-coding variant effects from a DNA sequence. [34] Another recent groundbreaking

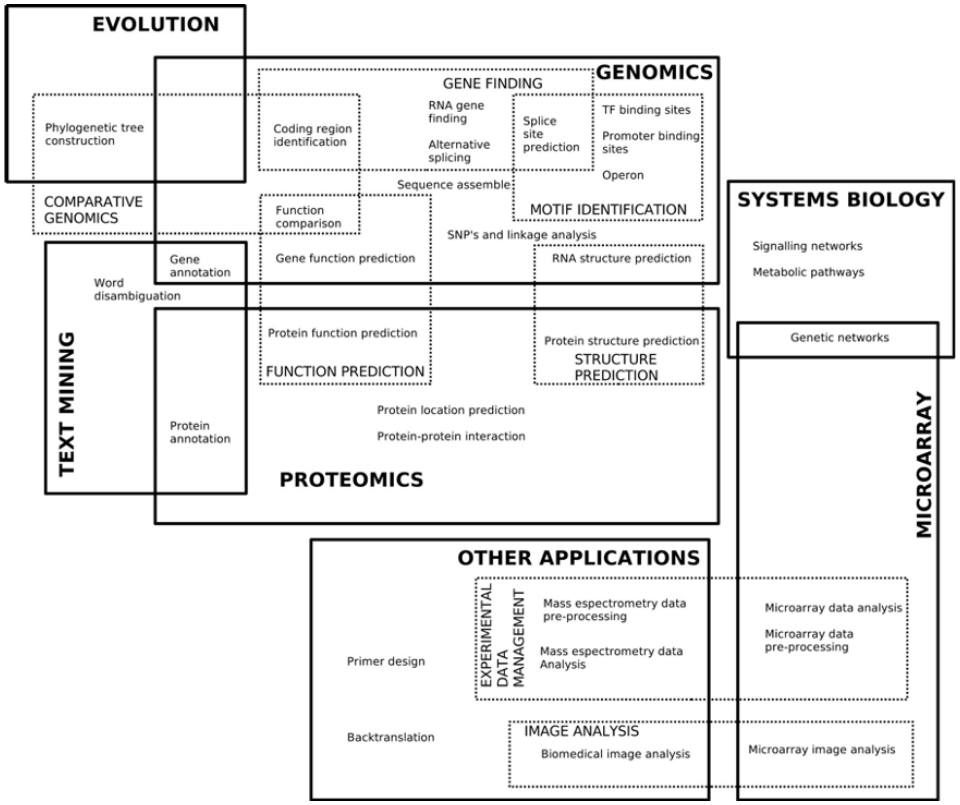


Figure 2.2: The research fields of biology and their to be mastered tasks. [Figure taken from 19]

application of Deep Neural Networks was performed by the research group of Deepmind (a subsidiary company of Google). This group not only invented the first NN that beat a human in the game of Go [9], in the end of 2020 they made a big step into the direction of solving one of biology's hardest questions, namely the prediction the 3D-Structure of proteins which has the chance to reveal unknown protein interactions and can lead to a deep understanding of protein function. In the scope of the Critical Assessment of Structure Prediction (CASP) challenge, the team developed a DNN (AlphaFold \AlphaFold2) that could predict the 3D-Structure of proteins with a similarity around 90% to the structures that are generated by the golden standard experimental methods [5]. The next section will provide an insight of the theory behind DNNs.

2.2 Artificial Neural Networks and Deep Learning

An Artificial Neural Network (ANN) describes a computational model inspired by the biological brain [11]. The brain contains neurons that can either be active or inactive depending on the activation threshold. These neurons communicate with each other in order to learn patterns so that they can make decisions. An ANN is characterized by its parameters (neurons) x and consists of one input layer, one output layer and zero, one or multiple hidden layers. The terms Deep Learning (DL) and Deep Neural Network (DNN) are used when an ANN uses multiple, complex hidden layers. Figure 2.3 is a graphical representation of the described structure with 3 neurons in the input, 5 neurons in the hidden and 2 neurons in the output layer. Each neuron n of a layer is connected with

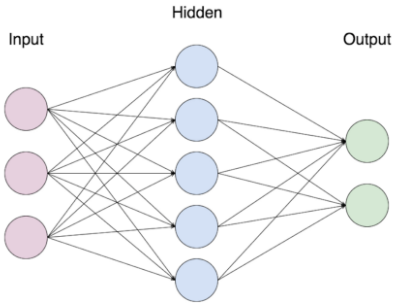


Figure 2.3: A graphical representation of a simple artificial neural network.

weights ($\vec{W} = (w_0, w_1, ..., w_n)$) to all neurons of the previous layer ($\vec{X} = (x_0, x_1, ..., x_n)$). Each neuron also contains a bias which can be seen as the threshold potential of that neuron that is only active when this value is exceeded. A neuron can be represented as following linear function [16]:

$$n(\vec{X}, \vec{W}) = x_0 w_0 + x_1 w_1 + ... + x_n w_n + b = \vec{X} \cdot \vec{W} + b = \sum_{i=0}^{n} x_i w_i + b \tag{2.1}$$

In order to transform the value of a neuron into an high dimensional parameter, equation 2.1 is being transformed by an non linear activation function σ. The activation functions that was used in the hidden layers of the model architectures analyzed in this work are the Rectifier function (ReLu) (eqn. 2.2).

$$\sigma(x) = max(0, x) \tag{2.2}$$

The Softmax function (eqn. 2.3) transforms the parameters of the output layer into a probability distribution with $0 < \mathrm{n} < 1$, which simplifies the calculation of the loss function ℓ.

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{n \in \vec{X}} e^n} \tag{2.3}$$

with $x_i \in \vec{X}$.

ℓ is indicating how precise the prediction of the data is performed by an ANN and is high when the classification is bad and low when the classification is good. In this work, classification problems were investigated where one data-point unambiguously belongs to only one class. This leads us to the categorical-cross entropy loss function (equation 2.4)

that indicates the similarity between two discrete probabilities (N is the sample size). The function delivers a high number when the distributions are different and 0 when they are the exact same. In the use case of an ANN, the function compares the last layer y, that was transformed by the Softmax function with a binary representation of the actual category $\hat{y}$ of the input.

$$\ell(y, \hat{y}) = -\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) \tag{2.4}$$

The goal of the training of an ANN x is, to find the global minimum of the loss function ℓ in respect to the training data ξ. This is being achieved by backpropagation, a algorithm that calculates the gradient of each parameter w.r.t. the loss function. Backpropagation aims to adjust and correct the weights and biases in that manner, that the model performs correct classification on the data.

2.2.1 Gradient Descent

In order to achieve the previously mentioned minimization of the loss function, many adaptive gradient-based optimization algorithms such as Adam [14] or Adagrad [7] have been developed, however they follow the same baseline algorithm, namely Gradient Descent (GD). GD achieves the minimization of ℓ by calculating the gradients R of the model parameters x that lead to a decrease of ℓ and is achieved by partial derivation according to ℓ using the chain rule. The calculation of the gradients in iteration t in general has following form:

$$R_t(x_t) = \nabla(\ell(x_t; \xi_t)) \tag{2.5}$$

After the determination of the gradients, the model x is updated in the opposite direction since the negative of the steepest ascent is the strongest drop. Equation 2.6 determines how the model is updated where η is the step-size or learning-rate. The learning rate defines how fast the gradients are influencing the parameters in one iteration and thus tries to prevent the model from missing the minimum.

$$x_{t+1} = x_t - \eta \cdot R_t(x_t) \tag{2.6}$$

There are different flavors of Gradient Decent. In *Full Gradient Descent*, R is calculated for the entire dataset ξ. *Stochastic Gradient Decent (SGD)* in contrast calculates the gradients and updates the parameters x based on each data-point in ξ which leads to the following gradient determination equation:

$$R_t(x_t) = \nabla(\ell(x_t; \xi_t^{(i)})) \tag{2.7}$$

where $\xi_t^{(i)} \in \xi$ and $|\xi_t^{(i)}| = 1$.

One can see quickly that SGD performs superflous computations and updates for increasing datasets, that is why mini-batch gradient decent combines GD and SGD in the manner that the model is being updated subsequently on random batches of size k taken from the dataset. In equation 2.8 it is demonstrated how the gradients are calculated for one batch:

$$R_t(x_t) = \nabla(\ell(x_t; \xi_t^{(i:k)})) \tag{2.8}$$

where $\xi_t^{(i:k)} \subseteq \xi$ and $1 < k \leq |\xi_t|$ [25].

2.2.2 Convolutional ANNs, Padding, Stride and Maximum Pooling

One have to note that we only will explain the principles of CNNs, since a more detailed explanation would go beyond the scope of this chapter. Fully connected ANNs are a good way to learn on higher dimensional data, but when the dimension of the input gets too high (e.g. images), their size will increase, making the model training too hard, even with large graphical computation power. To tackle this challenge, convolutional layers were introduced as hidden layers that aim the detection of meaningful and recurring structures within the presented data. Each convolutional layer consists of multiple $k \times k$ channels with k being smaller than the previous layer. The convolution layers are being updated in the backpropagation step. A convolution happens in the manner that the dot product of the channel and the previous layer is taken and then stored in the output layer at the respective position. Then the channel is moved by the stride in x-direction and the steps are repeated until the border of the matrix is reached. Then the channel is moved by the stride in y-direction until the entire input was convoluted. One can see quickly that a convolution reduces the dimension of a layer what often is not wanted. To avoid this padding is introduced. This technique extends the matrix by k-1 entries in each direction. Zero-padding is a padding that is often used, in which, as the name proposes, the matrix is extended by the value zero.

Often, when the data is too high dimensional, it is advantageous to gradually reduce the dimensions of the hidden representation of the data within the CNN using pooling layers. Maximum pooling (max-pool) is again a $k \times k$ matrix which is applied to the input matrix where the maximum of the matrix is passed to the next layer. Then it is shifted by the stride length just like a convolution channel.

2.2.3 Evaluating a Predictive Model

There are different metrics to evaluate the performance of a deep neural network model such as accuracy, precision, recall or the F-measure. In this work, we train the models on balanced data classes that is why the accuracy (eqn. 2.9) of the model is the suitable metric for our evaluations:

$$Accuracy = \frac{T}{T + F} \tag{2.9}$$

where T (True) and F (False) are the correct and false classified samples, respectively. The accuracy is the percentage of correctly classified samples.

2.3 Federated Learning

Although the application of ANNs was often successful in the past, the optimizing of a single model based on centrally stored data often yields some issue including data privacy (described in Chapter 1). To cope with this challenge, the idea of federated learning was introduced by Konevčný et al., in which the data is distributed across multiple clients and the clients collaboratively train one global model [15] on their local data under the coordination of a central server. The crucial point of this approach is that the data never

leaves the clients but only the gradients of the training are exchanged and averaged on server side. Then the updated model is being sent to the clients to train the global model again. Figure 2.4 is showing a exemplary graphical representation workflow of federated learning on data generated in hospitals. For the experiments performed in this thesis, we assume that all clients have different training samples but the same features, train the same model and use the same learning rate η for calculating the gradients.

In federated training of a deep neural network model, a client $i \in K$ receives the global model x_t^g from the server and optimizes it on its local data using mini-batch GD in each communication round t. The resulting gradients ∇x_{ti}^l are shared with the server, which in turn, when all clients have sent the gradients, aggregates them by taking the weighted average over the gradients. Assuming that every client updated their gradients with the same learning rate η, the resulting weighted average is multiplied by it. The update step is performed by subtracting the latter result from x_t^g. Based on equation 2.10 the global model at communication round $t+1$ is calculated as following with n_i being the number of training samples of client i.

$$x_{t+1}^g = x_t^g - \eta \cdot \frac{\sum_{i=1}^{K} n_i \cdot \nabla x_{ti}^l}{\sum_{i=1}^{K} n_i} \tag{2.10}$$

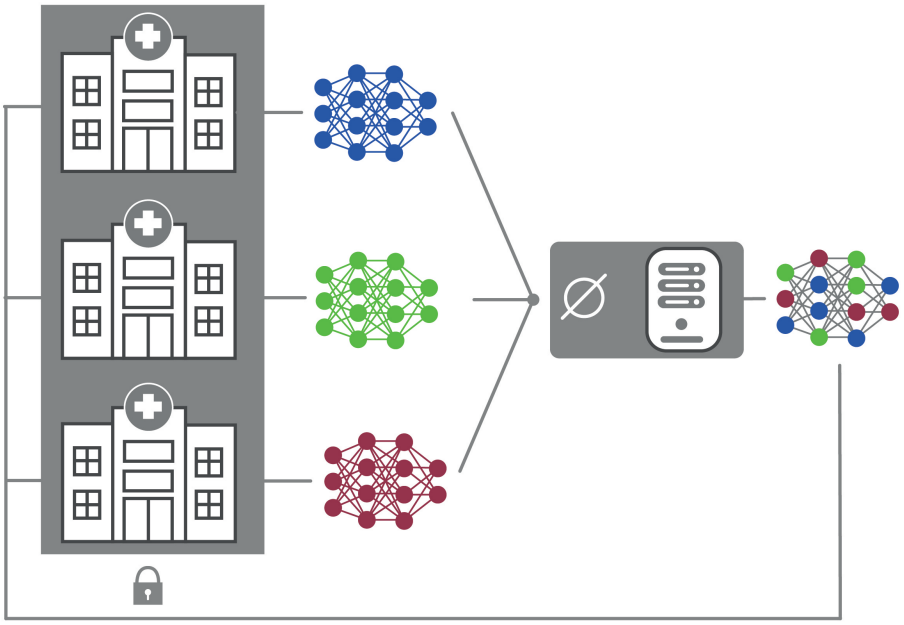


Figure 2.4: Exemplary representation of federated learning in hospitals. Each client generates its local model (colored network) which is then sent to a central server. The global model is updated by averaging the model parameters of the clients on the server side . Then, the global model is sent back to the clients. This approach allows for privacy-preserving training of the model.

While federated machine learning can be a suitable paradigm in terms of privacy and computation time, the bottleneck shifts to the bandwidth usage for the communication between the server in the clients, especially for large models such as deep neural networks. The focus of this thesis is the network communication challenge of the federated learning for deep neural network models.

3. Communication-Efficient Approaches

In this chapter, we introduce the communication-efficient approaches to federated deep neural networks (DNN), which aim to reduce the network traffic exchanged between the clients and the server (bandwidth usage). In *conventional* federated learning, each client trains the *global* model on multiple batches from its local dataset in each communication round. Then, the clients send the updated model parameters (*local* gradients) to the server, where the global model is updated by taking the average of the local model parameters. The training process is repeated until the global model converges to an optimum.

In equation 3.1 we calculate the bandwidth usage (in bits) of the conventional federated DNN:

$$\pi_C = 2 \cdot K \cdot G \cdot L \cdot N \tag{3.1}$$

where K is the number of clients, G is the number of model parameters, L is the size of each model parameter (in bits), and N is the number of iterations. Notice that we assume all clients employ the same DNN model (i.e. the same G and L for the clients). In this thesis, we implemented and evaluated three different communication-efficient approaches: *Gradient Quantification (GQ)*, *Gradient Sparsification (GS)*, and *Multiple local Updates (MU)*, described in the following sections.

3.1 Gradient Quantification (*GQ*)

The main idea behind GQ is to convert the gradients to a lower-bit representation before exchanging them between the server and clients. In practice, the default 32-bit floating-point numbers is used to represent the gradients in popular machine learning libraries such as *TensorFlow*. In the GQ approach, the gradients are transformed into 16-bit floating-point numbers before being transmitted in the network. The gradients are re-transformed to 32-bit floating-point representation before training in the clients and aggregation at the server. Therefore, the GQ approach halves L and reduces the bandwidth usage by 50% (equation 3.2).

$$\pi_{GQ} = 2 \cdot K \cdot G \cdot \frac{L}{2} \cdot N = \frac{\pi_C}{2} \tag{3.2}$$

3.2 Gradient Sparsification (*GS*)

A federated DNN model might be over-parameterized, where many of the model parameters do not change significantly during training and have not much effect on the model performance. The GS approach takes advantage of this observation and decides which gradients to ignore and which one to consider based on their values before sending them on the network. To this end, GS computes the P^{th} percentile of the gradients in each layer and sends only F percent of the gradients whose values are above the percentile value. It also employs an additional binary value of size G (number of gradients) to indicate which gradients are ignored/considered. The receiving end point uses the old value of the gradient if it is ignored by sparsification.

Equation 3.3 indicates the bandwidth usage of the GS approach, where F_i^{server} and F_i^j indicate the percentage of the gradient sent by the server and client j in iteration i, respectively. The GS approach reduces the bandwidth consumption by decreasing the number of gradients exchanged (G).

$$\pi_{GS}(P) = \left(\frac{\sum_{i=1}^{N} \frac{F_i^{server}}{100}}{2N} + \frac{\sum_{j=1}^{K}\sum_{i=1}^{N} \frac{F_i^j}{100}}{2K \cdot N} + \frac{1}{L}\right) \cdot \pi_C \tag{3.3}$$

3.3 Multiple Local Updates (MU)

In the GQ and GS approaches, the clients update the model once on their dataset using mini-batch gradient descent in each communication round. That is, the number of local updates is 1 for these approaches ($E = 1$). The idea behind MU (and $FedAvg$) approach is to increase the number of local updates ($E > 1$) in each communication round, so that the global model converges faster (in iteration $N' < N$.). This way, the model needs a lower number of iterations and hence reduces the network traffic. Equation 3.4 shows the bandwidth usage of the MU approach.

$$\pi_{MU} = 2 \cdot K \cdot G \cdot L \cdot N' = \frac{N'}{N} \cdot \pi_C \tag{3.4}$$

3.4 Combination of the Approaches

While applying a single approach to federated learning can already lead to a significant decrease of bandwidth usage, they can also be applied in sequence to further improve the network efficiency. To this end, first the model is trained in each client using the MU approach. Then, the gradients are sparsed using the GS approach, and finally, they are transformed to *16-bit* floating point values using the GQ approach before being sent in the network. Equation 3.5 indicates the overall network bandwidth usage after applying all the approaches.

$$\pi_{MU\_GS\_GQ}(P) = \frac{N'}{N} \cdot \left(\frac{\sum_{i=1}^{N} \frac{F_i^{server}}{100}}{2N} + \frac{\sum_{j=1}^{K}\sum_{i=1}^{N} \frac{F_i^j}{100}}{2K \cdot N} + \frac{1}{L}\right) \cdot \frac{1}{2} \cdot \pi_C \tag{3.5}$$

3.5 Trade-off between the Communication Efficiency and Performance

While the communication-efficient approaches can save a great deal of network bandwidth, they might adversely affect the performance of the model. One of the goals of this thesis is to evaluate the impact of different communication-efficient approaches on the accuracy of the global model trained in the federated environments. Another goal is to compare the different approaches from the communication efficiency perspective assuming a given target accuracy. To this end, we consider a target accuracy and determine the communication rounds each approach requires to reach that accuracy. Given that, we calculate the network bandwidth usage for each approach and compare them with each other.

4. Implementation

In this chapter, we describe the overall setting for the simulations. First, we overview the packages and the software structure employed to simulate the federated environment. Then, we introduce the neural network models and datasets used in the simulations. All produced data and software can be found in the GitLab repository [10]. Also all utilized Python packages with their respective versions are provided in A.1 and can be found in th *requirements.txt* file in the repository.

4.1 Software and Hardware

Python is a widely distributed open-source, object-oriented, high level programming language with numerous scientific and machine learning libraries [31] . This makes it an ideal choice for implementing the federated learning simulations and the communication-efficient approaches. In this project, Python version 3.7.3 was used.

TensorFlow [2] is an open-source machine learning framework initially created by the Google Brain Team in 2015, and it allows for efficient computations on data-flow graphs with C++ back-end. In the graphs, each node represents a mathematical operation on edges, indicating the n-dimensional arrays (tensors). With its tensor processing unit (TPU) introduced in 2016, its became one of the most suitable frameworks for computations on Graphic processing units (GPU). All neural networks used in this project are defined using Keras [6] (high-level application programming interface for TensorFlow). In this project, TensorFlow version 2.4.0 and Keras version 2.4.3 are leveraged in the simulations.

NumPy [22] is an open-source package for scientific computing. It is available in programming languages like Java, FORTRAN and Python, and it has already been employed successfully in multiple packages such as Pandas and SciPy. It allows for fast operations on array-like objects. We used NumPy version 1.19 in the project.

We carried out the simulations in a server equipped with 4 Intel(R) Xeon(R) Silver 4114 CPUs (2.2 GHz, 10 cores, 14MB cache), 100 GB main memory, and 3 Nvidia TITAN V GPUs (12 GB memory).

4.2 Simulation Package Structure

We implemented the simulation-framework in object-oriented fashion in Python to represent the physical abstraction of the clients and the server. The main class - Simulator - is composed of one Server object and multiple Client objects which contain a DnnModel object. Furthermore the Simulator class has one instance of the Dataset, the ConfigReader and the Logger class. Figure 4.1 describes the simulation code structure using a truncated UML diagram that only focuses on the main functionalities. To perform a simulation the user should define a (yaml-)configuration file (an example is provided in A.1) in which all adjustable settings like model, dataset, communication efficient approach options, datatype, data-size, result output path etc. are stated. Afterwards, the Simulator instance handles the setup of the DnnModel, the Server, the Clients and the Dataset class.

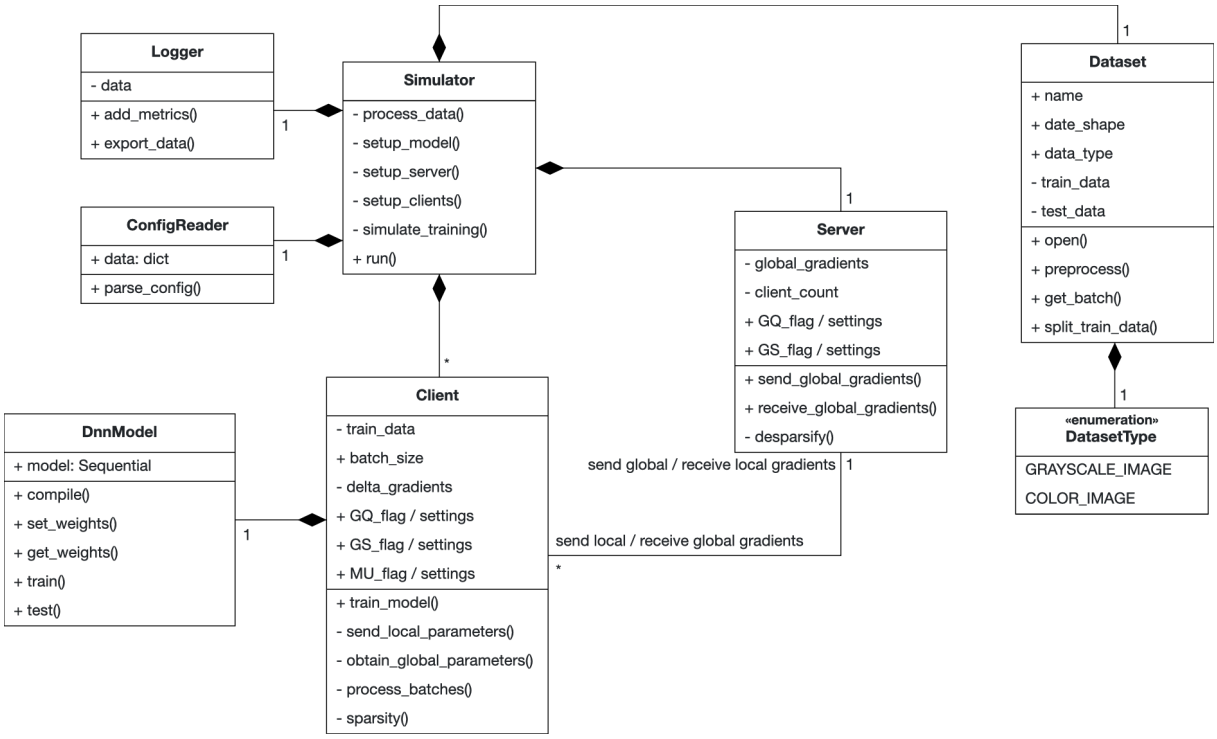


Figure 4.1: Truncated UML diagram for the simulation package

To that end, first of all the Logger instance is created which prepares the result file. Then the ConfigReader instance is parsing the configuration file based on which the Dataset instance of the Simulator is opening and processing the data. In this step the entire data is split into the train and test sets. Also each data-point is bundled with its specific one-hot encoded label and split into the data subsets that are later distributed to the clients. After the data splits are generated, the DnnModel class creates its model with plugin helper functions, sets the learning rate for SGD and compiles the model. Then the Server and Client instances are initialized. Each client gets the previously determined dataset portion along with a DnnModel instance.

After the setup is done, the federated training starts. Each client obtains the global model (gradients) from the server, trains the model (one or multiple times) on its local dataset and sends the updated gradients back to the server. Dependent on the configurations the data is transformed/sparsified before sending the gradients. When the server has received all gradients from the clients, it updates the global model using equation 2.10. After the model is updated, the model is evaluated on the test dataset provided by the Dataset instance and the resulting accuracy is stored together with the iteration in the result file. This training process is repeated for the maximum number of communication rounds stated in the configurations.

4.3 Neural Network Models

We used three different convolutional neural networks in the simulations: *2CFNN* [21, 4], *3CFNN* [30, 4], and *VGG16* [27]. In the following, we describe each model in more details.

The **2CFNN** model is the smallest model used and contains two 5x5 convolution

layers. The first one has 32 filters and the second one has 64 filters. Each of these layers is followed by a 2x2 max pooling layer. The last max-pooling layer is followed by a fully connected layer with 512 neurons (ReLu activation) and the output layer (softmax activation). The model has 1,663,370 total trainable parameters. Table 4.1 shows a summary of the $2CFNN$ model. The **3CFNN** model consists of three 3x3 convolution layers, each utilizing the ReLu activation function. The first two layers are connected to a 2x2 max pool layer and the third to a fully connected dense layer with 1024 neurons.

Model: Sequential		
Layer (type)	Output Shape	Param #
input_layer	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 28, 28, 32)	832
activation (Activation)	(None, 28, 28, 32)	0
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	51264
activation_1 (Activation)	(None, 14, 14, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 512)	1606144
activation_2 (Activation)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130
activation_3 (Activation)	(None, 10)	0
Total params: 1,663,370		
Trainable params: 1,663,370		
Non-trainable params: 0		

Table 4.1: The *2CFNN* model: The input shape is 28x28x1 and the number of classes is 10.

The first convolution layer has 128 channels, the second 256 and the last 512 channels. The size of the fully connected output layer again depends on the number of classes of the training data-set and uses the softmax activation function. The model has 9.878.794 total trainable parameters. Table 4.2 provides a summary of the *3CFNN* model. The **VGG16** model differs from the other two models in terms of size and complexity. It contains its convolution layers are of the size 3x3 whereas channels of other models are often of the size 5x5 or even larger. [27] The original model that was proposed was trained on the Image Net data-set which contained over 14 million 224x224px RGB images of 1000 classes. We modified the model in that way, that it fits the input and output dimensions. The model consits of five blocks of two or three convolution layers and a final max-pooling layer. The first block has 64 3x3 filters, the second 128, the third 256, and the fourth and fifth 512. The last max-pooling layer of the fifth block and the output layer are connected by two fully connected dense layers with 4096 neurons each. It has a total of 65.087.304 trainable parameters. List 4.3 shows a detailed representation of the used ***VGG16*** model.

Model: Sequential		
Layer (type)	Output Shape	Param #
input_layer	(None, 32, 32, 3)	0
conv2d (Conv2D)	(None, 30, 30, 128)	3.584
activation (Activation)	(None, 30, 30, 128)	0
max_pooling2d (MaxPooling2D)	(None, 15, 15, 128)	0
conv2d_1 (Conv2D)	(None, 13, 13, 256)	295.168
activation_1 (Activation)	(None, 13, 13, 256)	0
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 256)	0
conv2d_2 (Conv2D)	(None, 4, 4, 512)	1.180.160
activation_2 (Activation)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 1024)	8.389.632
activation_3 (Activation)	(None, 1024)	0
dense_1 (Dense)	(None, 10)	10.250
activation_4 (Activation)	(None, 10)	0
Total params: 9,878,794		
Trainable params: 9,878,794		
Non-trainable params: 0		

Table 4.2: The *3CFNN* model: The input shape is 32x32x3 and the number of classes is 10.

4.4 Datasets

To train and evaluate the models, we used three different image datasets that fit the complexity of each model: Fashion-MNIST (*FMNIST*) [33], CIFAR-10 [17], and colorectal cancer histology [12]. In the following, we provide the description of each dataset.

The ***FMNIST*** dataset was introduced by the Zalando assortment in 2017. It was created as a "drop-in replacement" for the original MNIST dataset. It consists of 60,000 training and 10,000 testing samples. Each sample is a 28 x 28 greyscale image from one of the classes T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. Each label has 6,000 samples in the train set and 1,000 in the test set. All classes are mutually exclusive which allows an unambiguous classification of each image. Figure 4.2 displays a selection of each class.

The **CIFAR-10** dataset consists of a train set with 50,000 samples and a test set with 10,000 samples. Each sample is a 32 x 32 color image (three channels from the red-green-blue (RGB) spectrum) and belongs to one of the classes Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck classes. Each label has 5,000 samples in the training set and 1,000 samples in the test set. Similar to Fashion-MNIST, CIFAR-10 is also mutually exclusive which again allows an unambiguous classification of each image. Figure 4.3 shows an exemplary selection of samples from the CIFAR-10 dataset.

The **colorectal cancer histology** (*CCH*) dataset contains 4,000 color images of shape 150 x 150. Each pixel represents $0.495\mu m$ of tissue leading to a total tissue surface of $74.25\mu m^2$ per tile. The images were generated with Aperio ScanScope using a magnification of 20x. All images are fully anonymized and approved by an ethical committee

Model: Sequential		
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
fc1 (Dense)	(None, 4096)	33558528
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 8)	32776
Total params: 65,087,304		
Trainable params: 65,087,304		
Non-trainable params: 0		

Table 4.3: The *VGG16* model: The input shape is 150x150x3 and the number of classes is 8.

(medical ethics board II, University Medical Center Mannheim, Heidelberg University, Germany; approval 2015-868R-MA) [12]. The dataset images are from 8 different classes, again, mutually exclusive specific histology types. Each label is represented by around 500 samples. Table 4.4 lists a short description of each class and Figure 4.4 provides an exemplary presentation of sample images from each class .

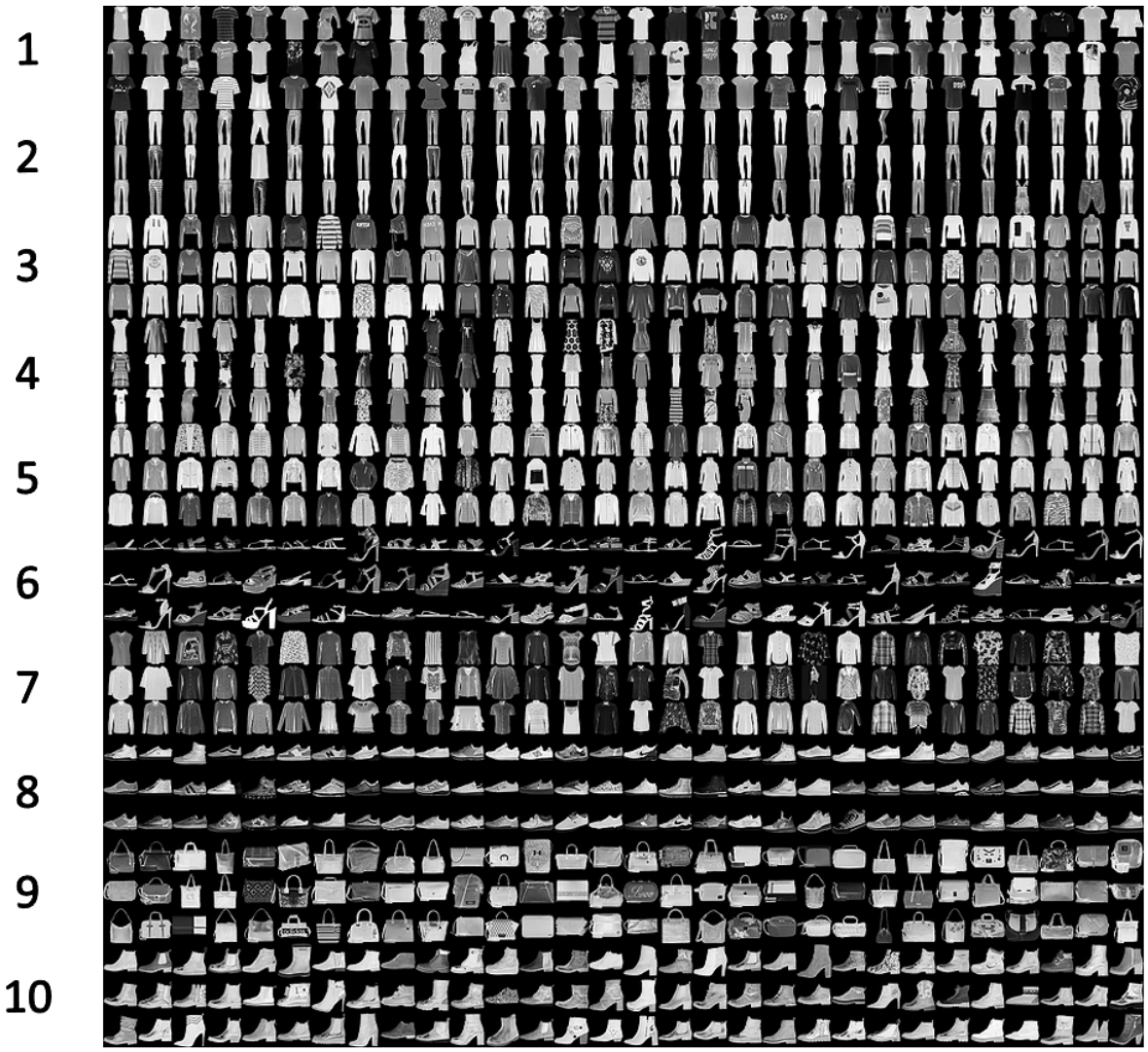


Figure 4.2: This figure shows a random sample image selection of the classes from the Fashion-MNIST dataset with their corresponding label. (**1**) T-shirt/top (**2**) Trouser (**3**) Pullover (**4**) Dress (**5**) Coat (**6**) Sandal (**7**) Shirt (**8**) Sneaker (**9**) Bag (**10**) Ankle boot. Image taken from [8]

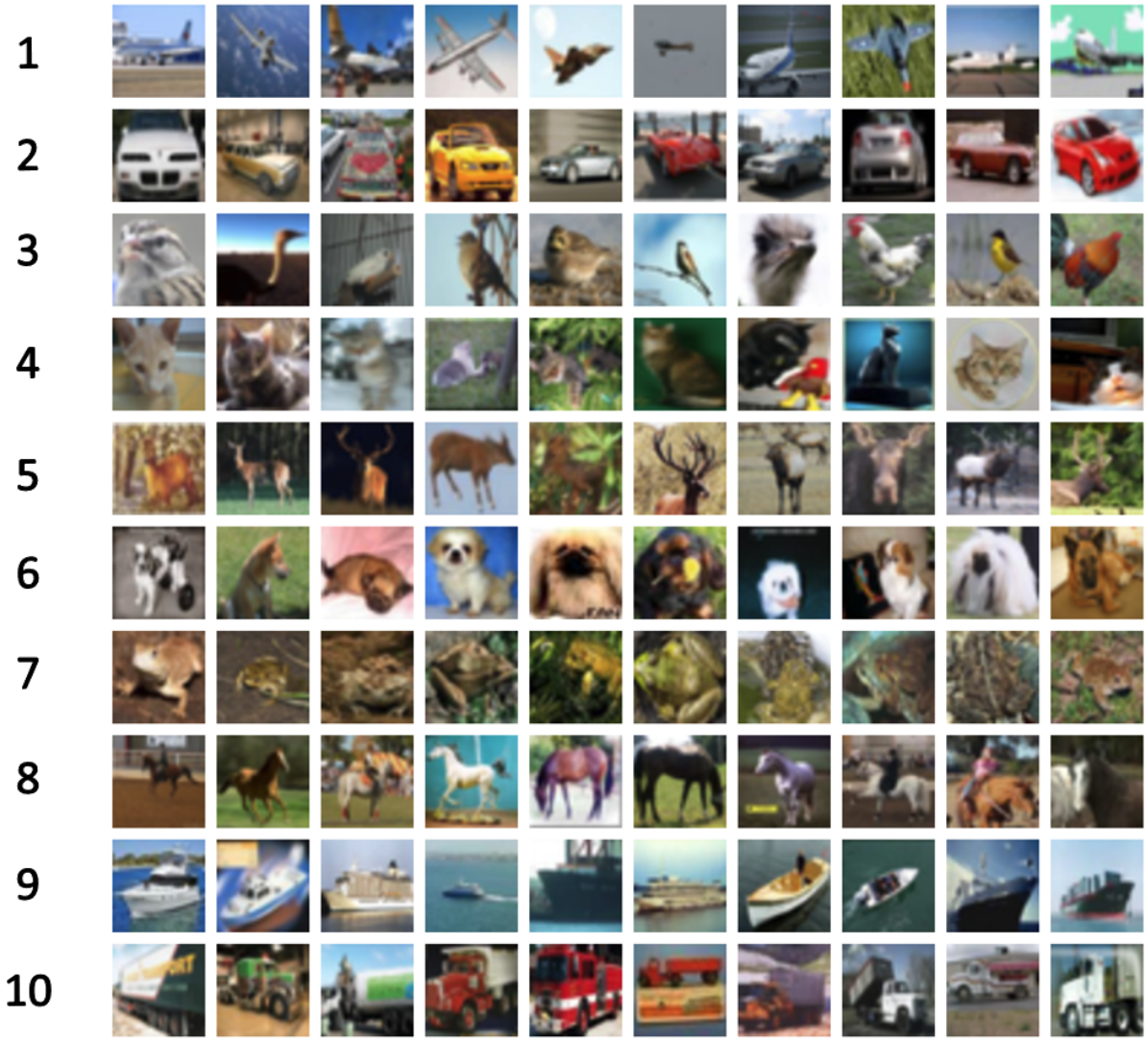


Figure 4.3: This figure shows a random sample image selection of the classes from the CIFAR-10 dataset with their corresponding label. (**1**) Airplane (**2**) Automobile (**3**) Bird (**4**) Cat (**5**) Deer (**6**) Dog (**7**) Frog (**8**) Horse (**9**) Ship (**10**) Truck. Image taken from [17].

label	name	description
1 | a	tumour epithelium	Tumorous tissue
2 | b	simple stroma	"Supporting frame" of an organ
3 | c	complex stroma	Stroma with scattered tumour / immune cells
4 | d	immune cell conglomerates	Dense accumulation of immune cells
5 | e	debris and mucus	Dead tissue (necrosis) / coating of inner organs
6 | f	mucosal glands	Normal mucosal glands
7 | g	adipose tissue	Fatty tissue
8 | h	background	Image tiles containing no tissue

Table 4.4: Description of the classes in the colorectal histology dataset.

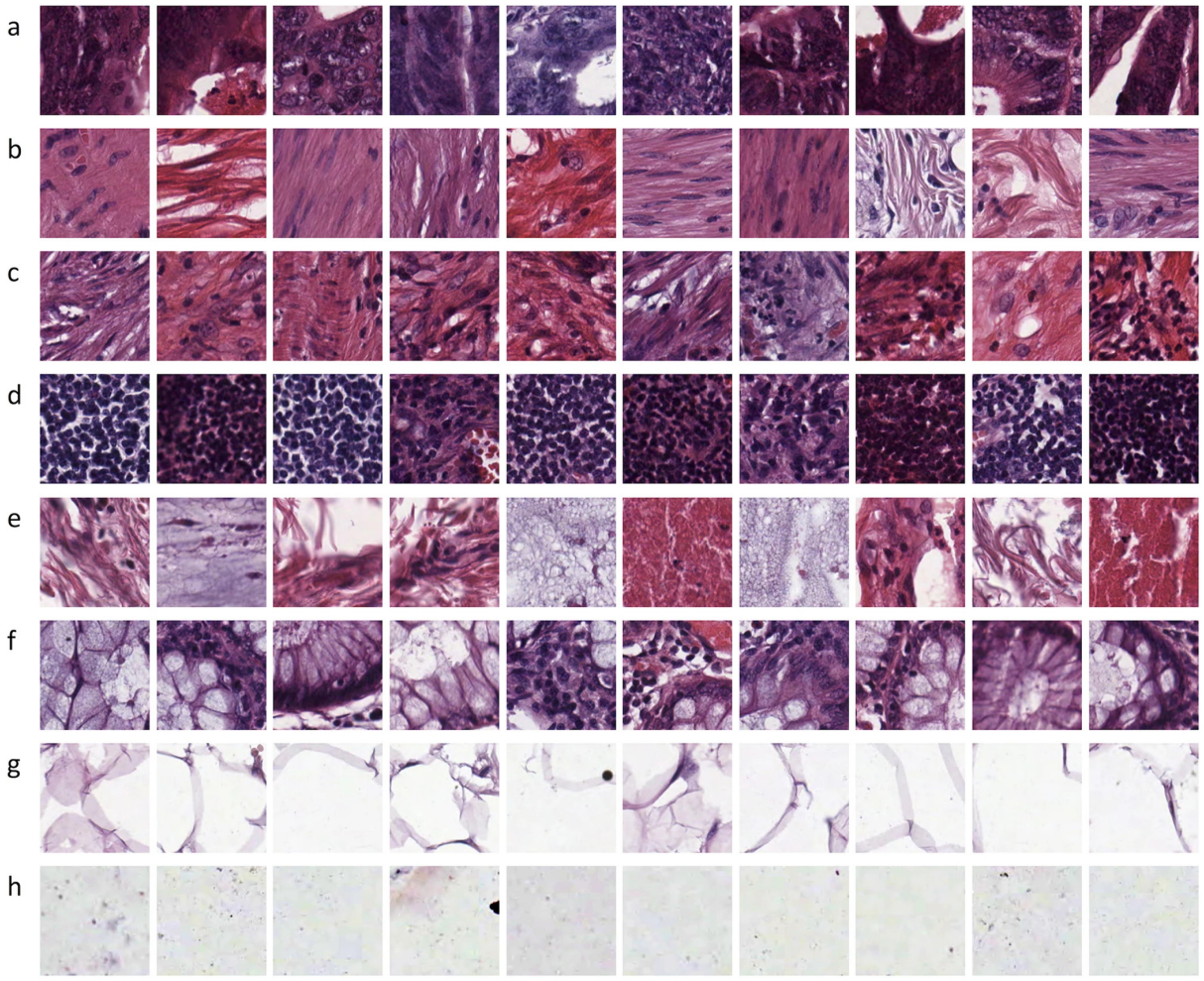


Figure 4.4: This figure shows a random sample image selection of the classes from the colorectal cancer histology dataset with their corresponding label. (**a**) tumour epithelium (**b**) simple stroma (**c**) complex stroma (**d**) immune cell conglomerates (**e**) debris and mucus (**f**) mucosal glands (**g**) adipose tissue (**h**) background. Image taken from [12].

5. Results

In this chapter, we present the results for the different communication-efficient approaches introduced in chapter 3. We evaluate each dataset with a model adjusted to its complexity: The *2CFNN*, *3CFNN*, and *VGG16* models are trained on the *FMNIST*, *CIFAR-10*, and *CCH* datasets, respectively. We evaluate the three approaches separately and as a combination of all from the performance and network efficiency perspectives. We considered accuracy (eqn. 2.9) as the performance metric and network bandwidth usage (the amount of traffic exchanged in the network) to reach a target accuracy as the network efficiency metric.

The simulations for *2CFNN-FMNIST* and *3CFNN-CIFAR-10* are performed on a federated environment with 10 clients while the number of clients is 5 for *VGG16-CCH* since this dataset includes less samples. The batch size is 64, optimizer is *stochastic gradient descent* (*SGD*), and loss function is categorical cross-entropy for all simulations. The learning rate is different for each scenario and will be specified in the corresponding evaluation section. The baseline for each model-dataset pair is the conventional federated learning, where the local epochs E is 1, the gradients are represented by 32-bit floating point numbers and no gradient sparsification before sending the gradients. The maximum accuracy of the baseline simulation is further indicated in the plots by a horizontal dashed line. *2CFNN-FMNIST*, *3CFNN-CIFAR-10*, and *VGG16-CCH* employ learning rates of $\eta = 0.1$, $\eta = 0.1$, and $\eta = 0.01$ in the conventional federated learning, respectively.

Approach \ Target accuracy	0.84	0.86	0.88	0.90	0.92
P=10	665.35 | 642.48 | 3.44	1064.56 | 1156.46 | -8.63	1996.04 | 1927.43 | 3.44	3725.95 | 4240.35 | -13.81	11177.85 | 11179.10 | -0.01
P=50	665.35 | 509.41 | 23.44	1064.56 | 815.05 | 23.44	1996.04 | 1528.22 | 23.44	3725.95 | 2852.68 | 23.44	11177.85 | 7437.35 | 33.46
P=90	665.35 | 376.34 | 43.44	1064.56 | 677.41 | 36.37	1996.04 | 1053.75 | 47.21	3725.95 | 1655.89 | 55.56	11177.85 | 3989.18 | 64.31
E=2	665.35 | 399.21 | 40.00	1064.56 | 532.28 | 50.00	1996.04 | 1064.56 | 46.67	3725.95 | 1996.04 | 46.43	11177.85 | 7318.83 | 34.52
E=10	665.35 | 133.07 | 80.00	1064.56 | 133.07 | 87.50	1996.04 | 266.14 | 86.67	3725.95 | 665.35 | 82.14	-
E=20	665.35 | 133.07 | 80.00	1064.56 | 133.07 | 87.50	1996.04 | 266.14 | 86.67	3725.95 | 399.21 | 89.29	11177.85 | 9181.80 | 17.86
GQ	665.35 | 399.21 | 40.00	1064.56 | 1663.37 | -56.25	-	-	-
GQ + GS(P=50) + MU(E=10)	665.35 | 50.01 | 92.48	1064.56 | 100.01 | 90.61	1996.04 | 150.02 | 92.48	3725.95 | 200.02 | 94.63	-

Table 5.1: *2CFNN-FMNIST*: Bandwidth usage by the conventional federated training (in MB) | Bandwidth usage by the approach (in MB) | Bandwidth saving of the approach (in percent) to reach the target accuracy

Approach \ Target accuracy	0.5	0.6	0.68	0.7	0.72	0.733
P=10	7112.73 | 6868.26 | 3.44	13435.16 | 12973.36 | 3.44	23709.11 | 20604.73 | 13.09	26870.32 | 24420.42 | 9.12	36353.96 | 30525.52 | 16.03	60063.07 | 54182.78 | 9.79
P=50	7112.73 | 5445.69 | 23.44	13435.16 | 9076.15 | 32.44	23709.11 | 15126.91 | 36.20	26870.32 | 17547.22 | 34.70	36353.96 | 19967.52 | 45.07	60063.07 | 25413.21 | 57.69
P=90	7112.73 | 6655.84 | 6.42	13435.16 | 10891.38 | 18.93	23709.11 | 16337.06 | 31.09	26870.32 | 18757.37 | 30.19	-	-
E=2	7112.73 | 4741.82 | 33.33	13435.16 | 9483.64 | 29.41	23709.11 | 15806.07 | 33.33	26870.32 | 18967.28 | 29.41	36353.96 | 26080.02 | 28.26	-
E=10	7112.73 | 1580.61 | 77.78	13435.16 | 2370.91 | 82.35	23709.11 | 6322.43 | 73.33	26870.32 | 14225.46 | 47.06	-	-
E=20	7112.73 | 790.30 | 88.89	13435.16 | 1580.61 | 88.24	23709.11 | 11854.55 | 50.00	-	-	-
GQ	7112.73 | 3556.37 | 50.00	13435.16 | 6717.58 | 50.00	23709.11 | 11064.25 | 53.33	26870.32 | 13040.01 | 51.47	36353.96 | 16201.22 | 55.43	60063.07 | 19757.59 | 67.11
GQ + GS(P=50) + MU(E=2)	7112.73 | 1781.89 | 74.95	13435.16 | 3563.78 | 73.47	23709.11 | 5939.63 | 74.95	26870.32 | 6830.57 | 74.58	36353.96 | 10097.37 | 72.22	-

Table 5.2: *3CFNN-CIFAR-10*: Bandwidth usage by the conventional federated training (in MB) | Bandwidth usage by the approach (in MB) | Bandwidth saving of the approach (in percent) to reach the target accuracy

Approach \ Target accuracy	0.55	0.65	0.7	0.72	0.76	0.771
P=10	205.68 | 202.92 | 1.34	236.92 | 262.94 | -10.98	-	-	-	-
P=90	205.68 | 164.93 | 19.81	-	-	-	-	-
E=2	205.68 | 145.80 | 29.11	236.92 | 221.30 | 6.59	343.66 | 27597 | 19.70	367.09 | 328.04 | 10.64	466.03 | 377506.36 | 18.99	643.06 | 387.92 | 39.68
E=5	205.68 | 166.62 | 18.99	236.92 | 221.30 | 6.59	343.66 | 294.20 | 14.39	367.09 | 333.25 | 9.22	466.03 | 557.15 | -19.55	643.06 | 577.98 | 10.12
E=10	205.68 | 46.86 | 77.22	236.92 | 78.10 | 67.03	343.66 | 91.12 | 73.48	367.09 | 124.97 | 65.96	-	-
E=20	205.68 | 41.66 | 79.75	236.92 | 67.70 | 71.43	343.66 | 119.76 | 65.15	367.09 | 130.17 | 64.54	-	-
GQ	205.68 | 91.12 | 55.70	236.92 | 100.23 | 57.69	343.66 | 135.38 | 60.61	367.09 | 184.85 | 49.65	-	-
GQ + GS(P=50) + MU(E=5)	205.68 | 76.06 | 63.02	236.92 | 102.53 | 56.72	-	-	-	-

Table 5.3: *VGG16-CCH*: Bandwidth usage by the conventional federated training (in GB) | Bandwidth usage by the approach (in GB) | Bandwidth saving of the approach (in percent) to reach the target accuracy

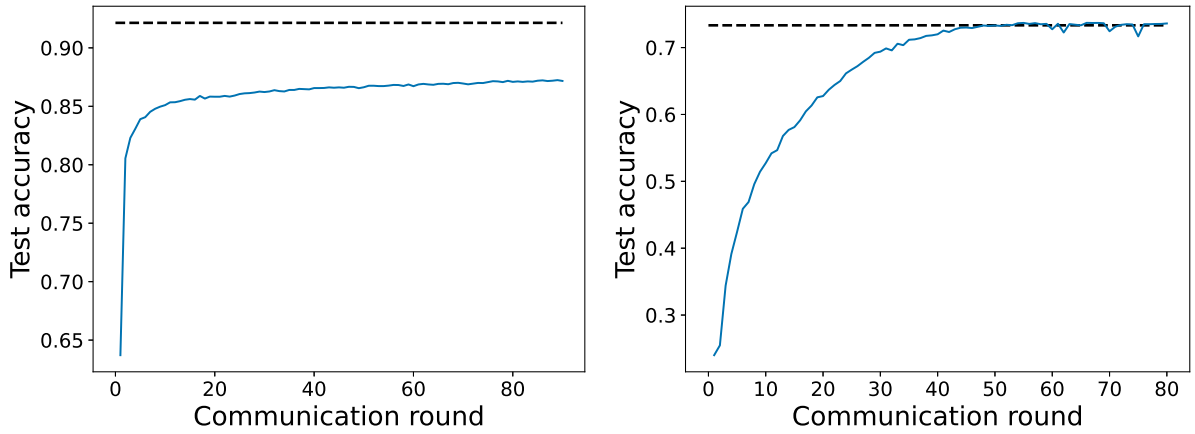
5.1 Gradient Quantification

We use the same learning rates as the conventional federated learning for the *GQ* approach. Figure 5.1 shows the accuracy curves for each model-dataset pair. We observe that the *GQ* approach achieves the accuracy of the baseline in the *3CFNN-CIFAR-10* scenario. However, the accuracy of the model decreases by $\approx 6.5\%$ in the *2CFNN-FMNIST* and *VGG16-CCH* scenarios. Regarding communication efficiency (Tables 5.1, 5.2, and 5.3), the *GQ* saves the network bandwidth usage by 67.11% and 60.61% in the *3CFNN-CIFAR-10* and *VGG16-CCH* scenarios while it incurs 56.26% more bandwidth usage in the *2CFNN-FMNIST* scenario to achieve the corresponding maximum accuracy. In sum, we observe that the efficiency of the *GQ* approach depends on the dataset and the model.

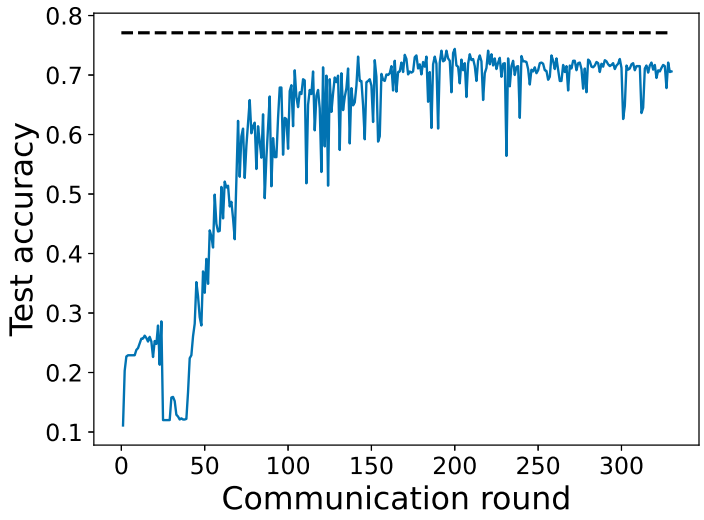
5.2 Gradient Sparsification

For the simulations of the *GS* approach, we again leverage the same learning rates as the conventional federated learning. The sparsification is performed only on the client-side and the complete global model (without sparsification) is shared by the server after aggregation. Figure 5.2 shows the accuracy curves for each model-dataset pair. According to the figure, the *GS* approach provides an accuracy comparable to the baseline for *2CFNN-FMNIST* and *3CFNN-CIFAR-10* scenarios. However, employing gradient sparsification has a slightly negative impact on the accuracy of the model in *VGG16-CCH* scenario.

In terms of bandwidth usage (Tables 5.1, 5.2, and 5.3), the intuitive assumption of a higher percentile leading to a decreased data traffic can only be confirmed for the *2CFNN-FMNIST* scenario with maximum percentage savings for $P = 10, 50, 90$ being 3.44%, 33.46% and 64.31%. The *3CFNN-CIFAR-10* scenario however demonstrates that a higher percentile will not necessarily lead to a decreased bandwidth usage because $P = 50$ leads to the highest savings for every single target accuracy with maximum savings of 57.69%. *VGG16-CCH* with $P = 10$ increases the bandwidth usage by 10.98% for the target accuracy of 0.65. *GS* with $P = 90$, on the other hand, achieves the accuracy of 0.55 with the saving of 19.81% and accuracy above this value have not been observed for *VGG16-CCH* after 120 communication rounds.



(a) *2CFNN-FMNIST* **(b)** *3CFNN-CIFAR-10*



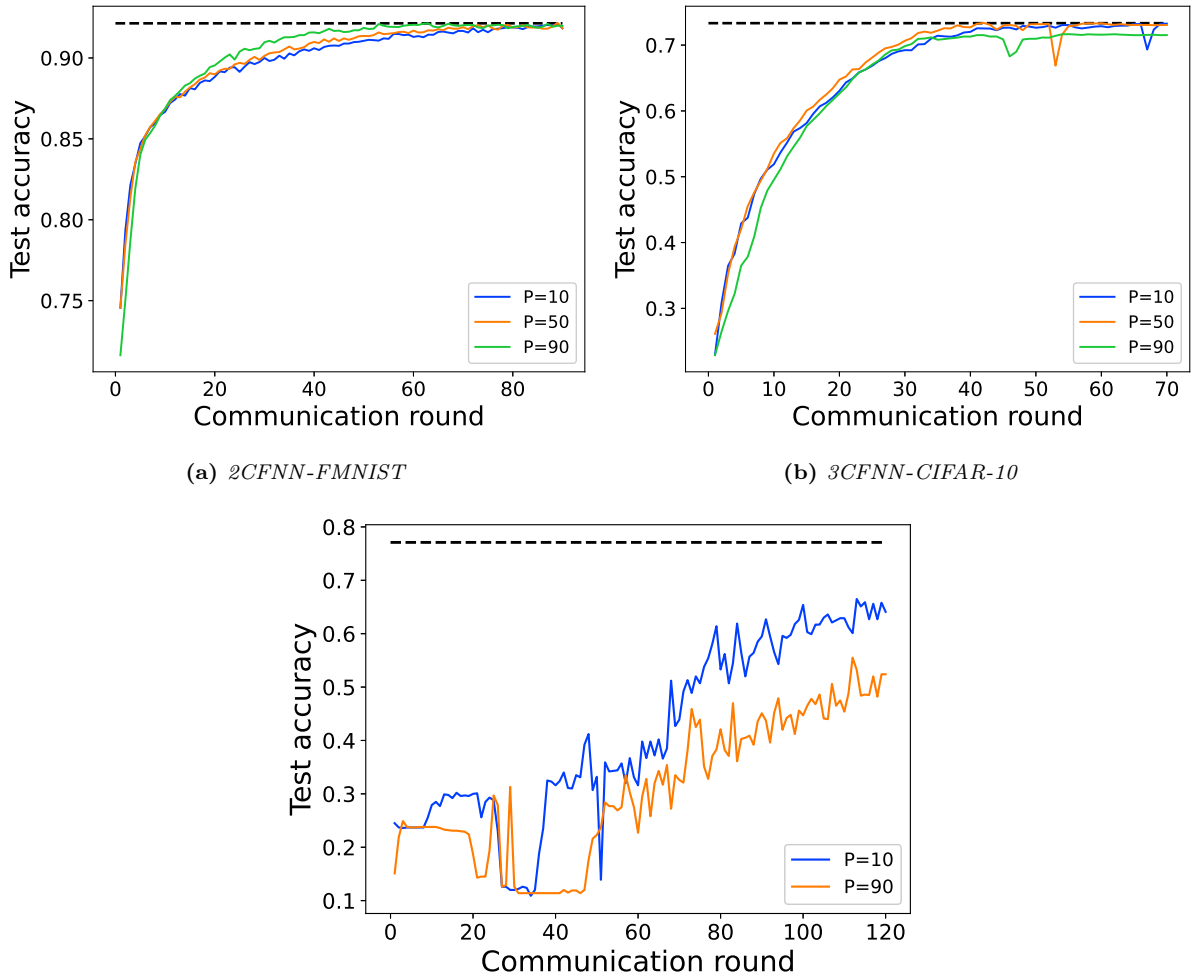
(c) *VGG16-CCH*

Figure 5.1: The learning process of the ***GQ*** approach

5.3 Multiple Local Updates

We evaluate the *MU* approach with different numbers of local updates: $E = 2$, $E = 10$, $E = 5$ (only *VGG16-CCH*) and $E = 20$. To this end, we use different learning rates since the models diverges if a suitable learning rate is not used. Following learning rates have been employed: $\eta = 0.1$ in *2CFNN-FMNIST* and $\eta = 0.001$ in *3CFNN-CIFAR-10* for all three E values. Regarding *VGG16-CCH*, we choose a lower learning rates for higher E values: $\eta = 0.005$ for $MU(E = 2)$, $\eta = 0.002$ for $MU(E = 10)$, and $\eta = 0.001$ for $MU(E = 20)$. The accuracy curves for all model-dataset pairs are shown in figure 5.3.

We observe that for every setting the slope of the accuracy curves significantly increases which is the determining factor for *MU* being communication efficient. Further, a higher number of local updates leads to a reduction of the maximum accuracy. $E = 10$ and $E = 20$ reduce the accuracy of the model in *3CFNN-CIFAR-10* by $\approx 4\%$ and $\approx 8\%$, respectively. This was also the case for *2CFNN-FMNIST* but only with a small difference of $\approx 1 - 2\%$. The *MU* approach with $E = 2$ and $E = 5$ provides a competitive accuracy compared to the baseline in *VGG16-CCH* but with $E = 10$ and $E = 20$, it reduces the



(a) *2CFNN-FMNIST*

(b) *3CFNN-CIFAR-10*

(c) *VGG16-CCH*

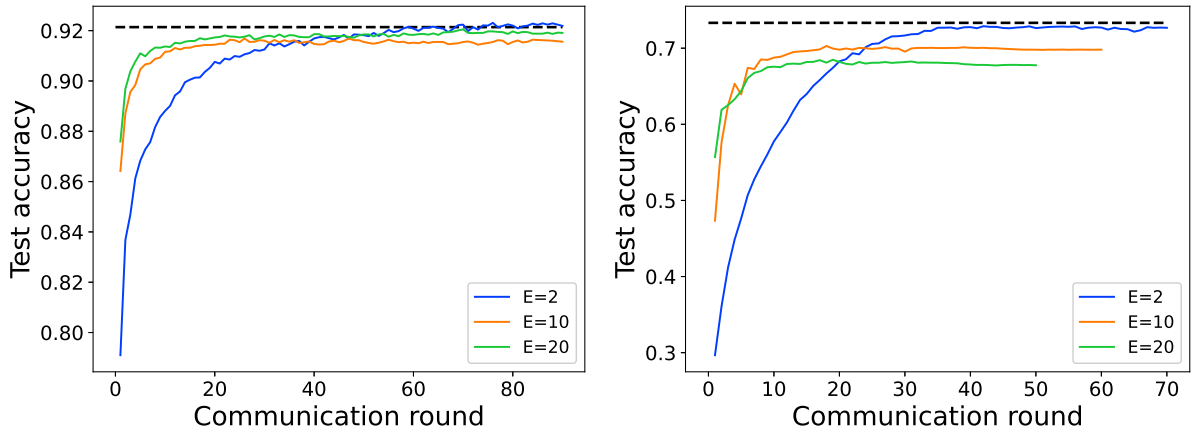
Figure 5.2: The learning process of the ***GS*** approach

accuracy by $\approx 3\%$.

When it comes to bandwidth savings (Tables 5.1, 5.2 and 5.3) *MU* is the approach that performs the best on its own compared to *GS* and *GQ*. The maximum percentage saving of 50% for $E = 2$ can be observed for *2CFNN-FMNIST* when exceeding the target accuracy of 0.86. When applied to *VGG16-CCH*, this setting is the only one reaching the maximum accuracy of 0.733 with a saving of 39.68%. For $E = 10$, the bandwidth savings up to 87.50% are achieved (*2CFNN*, target accuracy of 0.86) where $E = 20$ saves up to 86.21% (*2CFNN*, target accuracy of 0.90). With larger model sizes, *MU* provides higher absolute bandwidth savings. For example, the maximum bandwidth saving for *2CFNN-FMNIST* is $\approx 7GB$ ($E = 2$, target accuracy of 0.92) and $\approx 17GB$ for *3CFNN-CIFAR-10*; *MU* with $E = 2$ on *VGG16-CCH* saves $\approx 256GB$ (15 times more compared to *3CFNN-CIFAR-10*) to reach the maximum accuracy of 0.771.

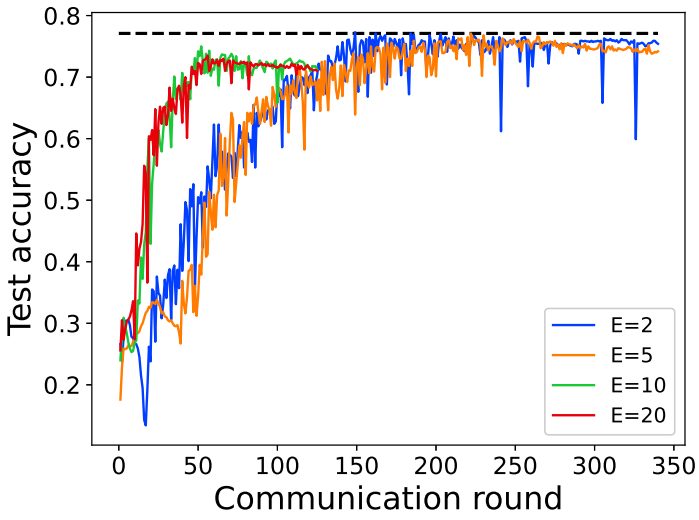
5.4 *GQ + GS + MU*

This section aims to analyze performance and communication efficiency if all three approaches applied (Figure 5.4). We use $\eta = 0.1$, $\eta = 0.001$, and $\eta = 0.001$ for *2CFNN-*



(a) *2CFNN-FMNIST*

(b) *3CFNN-CIFAR-10*

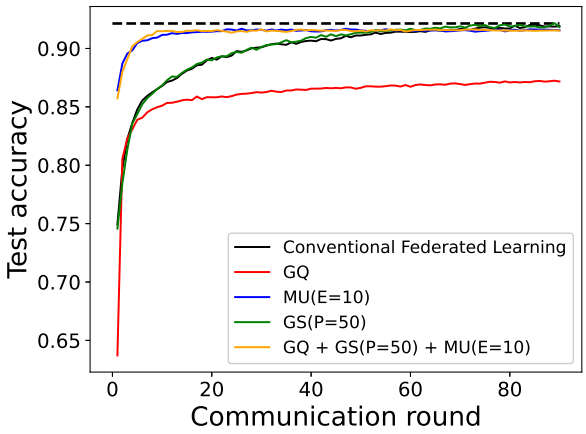


(c) *VGG16-CCH*

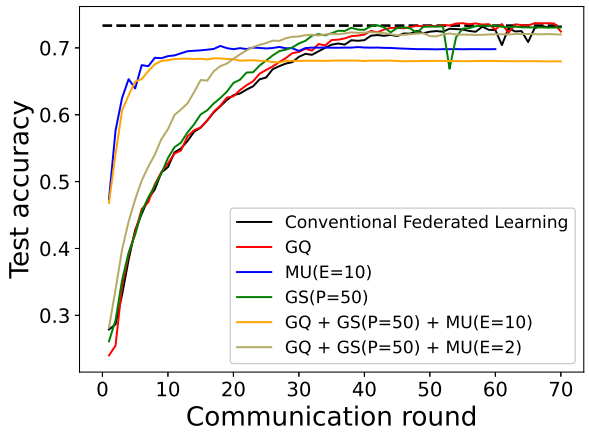
Figure 5.3: The learning process of the ***MU*** approach

FMNIST, *3CFNN-CIFAR-10*, and *VGG16-CCH*, respectively. We observe that the accuracy curve is positively affected by the *MU* approach. In the *2CFNN-FMNIST* scenario, the combination of the approaches leads to accuracy curve very similar to *MU* and reaches the baseline accuracy. In the *3CFNN-CIFAR-10* scenario, the combination with $E = 10$ cannot achieve the accuracy of the baseline while with $E = 2$ it does. This indicates the higher contribution of the *MU* approach in the combination case too. For the *VGG16-CCH* scenario, we also see a similar behavior from the *MU* approach and the combination.

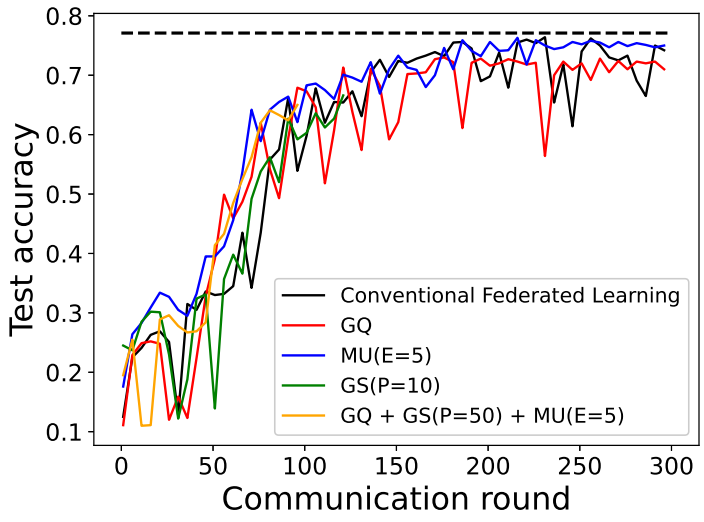
In terms of communication efficiency, the combination of the approaches performs better compared to each individual approach. Bandwidth saving up to 93.52% is observed in the *2CFNN-FMNIST* scenario to reach the target accuracy of 0.9. In the *3CFNN-CIFAR-10* scenario, $GQ + GS(P = 50) + MU(E = 2)$ achieves bandwidth saving of 72.02% for the target accuracy of 0.72. In the *VGG16-CCH* scenario, the combination can save 56.72% to reach the target accuracy of 0.65.



(a) *2CFNN-FMNIST*



(b) *3CFNN-CIFAR-10*



(c) *VGG16-CCH*

Figure 5.4: The learning process of the combination $GQ + GS + MU$

6. Conclusion

This last chapter aims to summarize our contributions in the thesis and discuss the results, the limitations, and future works.

6.1 Summary

In this thesis, we addressed the network communication challenge in federated deep neural networks. We first provided theoretical formulas for three different communication-efficient approaches to federated deep neural networks (Chapter 3): Gradient Quantification (GQ) which minimizes the size of each gradient before sending the model to the server and vice-versa. Gradient Sparsification (GS) which ignores gradients which have not changed beyond a certain level after the local updates. Multiple local Updates (MU) executes the mini-batch SGD multiple times in one communication round.

Afterwards, we presented a simulation framework to simulate the approaches and evaluated them from the accuracy and network bandwidth usage perspectives. The simulations were performed on three different CNN-based neural networks (Chapter 4.3) and three different datasets (Chapter 4.4) in IID settings. Our results show all three approaches can significantly save the network bandwidth but they also affect the accuracy of the models. Considering each approach separately, the MU approach is the most suitable choice taking into account both communication efficiency and performance. Considering the combination of the approaches, we see that the MU approach has much more contribution compared to the other two approaches and using the combined approach provides further network bandwidth saving in comparison with each individual approach.

Computation parallelism was one limitation of the implemented simulation framework that set time boundaries to our research. The simulation framework does not use multi-processing or multi-threading, and consequently, the client side training cannot be done in parallel. This made our simulations significantly more time consuming (almost a day for one simulation on the VGG16 model). Another limitation was the lack of efficiency of the gradient sparsification implementation and thus, the corresponding simulations required a couple of days for the VGG16 model to be completed.

6.2 Future Work

We are going to add computation parallelism to our simulation framework so that multiple clients can train the model concurrently and improve simulation runtime. Moreover, we will optimize the implementation of the gradient sparsification in the framework to further investigate the behavior of large models using the GS approach. Finally, we will evaluate the approaches in more federated scenarios especially Non-IID label distribution environments and imbalanced sample distributions across the clients. In the former, the clients have samples from a subset of the labels instead of all. In the latter, the clients have datasets with different number of samples. Evaluating the communication-efficient approaches in the combination of different data and sample distributions among clients is an interesting direction for the future work.

Bibliography

[1] *2018 reform of EU data protection rules*. European Commission. May 25, 2018. URL: https://ec.europa.eu/info/sites/info/files/data-protection-factsheet-changes_en.pdf (visited on 02/10/2021).

[2] Martin Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[3] Christof Angermueller et al. "Deep learning for computational biology". In: *Molecular systems biology* 12.7 (2016), p. 878.

[4] Mohammad Bakhtiari et al. "Federated Multi-Mini-Batch: An Efficient Training Approach to Federated Learning in Non-IID Environments". In: *arXiv preprint arXiv:2011.07006* (2020).

[5] Ewen Callaway. *It will change everything: DeepMind's AI makes gigantic leap in solving protein structures*. URL: https://www.nature.com/articles/d41586-020-03348-4. (accessed: 01.02.2021).

[6] François Chollet et al. *Keras*. https://keras.io. 2015.

[7] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[8] *Fashion MNIST GitHub repository*. https://github.com/zalandoresearch/fashion-mnist. Accessed: 2021-02-24.

[9] Elizabeth Gibney. "Google AI algorithm masters ancient game of Go". In: *Nature News* 529.7587 (2016), p. 445.

[10] A.E.T. Henkel. *com-eff*. https://gitlab.lrz.de/000000000149C8EB/com-eff. 2021.

[11] John J Hopfield. "Artificial neural networks". In: *IEEE Circuits and Devices Magazine* 4.5 (1988), pp. 3–10.

[12] Jakob Nikolas Kather et al. "Multi-class texture analysis in colorectal cancer histology". In: *Scientific reports* 6.1 (2016), pp. 1–11.

[13] El-Sayed M El-kenawy et al. "Advanced Meta-heuristics, Convolutional Neural Networks, and Feature Selectors for Efficient COVID-19 X-ray Chest Image Classification". In: *IEEE Access* (2021).

[14] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[15] Jakub Konevcny et al. "Federated learning: Strategies for improving communication efficiency". In: *arXiv preprint arXiv:1610.05492* (2016).

[16] Andrej Krenker, Janez Bešter, and Andrej Kos. "Introduction to the artificial neural networks". In: *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech* (2011), pp. 1–18.

[17] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[18] Sebastian Lapuschkin et al. "Unmasking clever hans predictors and assessing what machines really learn". In: *Nature communications* 10.1 (2019), pp. 1–8.

[19] Pedro Larranaga et al. "Machine learning in bioinformatics". In: *Briefings in bioinformatics* 7.1 (2006), pp. 86–112.

[20] Zengpeng Li, Vishal Sharma, and Saraju P Mohanty. "Preserving data privacy via federated learning: Challenges and solutions". In: *IEEE Consumer Electronics Magazine* 9.3 (2020), pp. 8–16.

[21] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.

[22] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[23] Kaiyang Qu et al. "Application of machine learning in microbiology". In: *Frontiers in microbiology* 10 (2019), p. 827.

[24] Jason A Reuter, Damek V Spacek, and Michael P Snyder. "High-throughput sequencing technologies". In: *Molecular cell* 58.4 (2015), pp. 586–597.

[25] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. Mar. 2020. URL: `https://ruder.io/optimizing-gradient-descent/`.

[26] Sepideh Sadegh et al. "Exploring the SARS-CoV-2 virus-host-drug interactome for drug repurposing". In: *Nature Communications* 11.1 (July 2020), p. 3518. ISSN: 2041-1723. DOI: `10.1038/s41467-020-17189-2`. URL: `https://doi.org/10.1038/s41467-020-17189-2`.

[27] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[28] Claudio D Stern. "The omics revolution: how an obsession with compiling lists is threatening the ancient art of experimental design". In: *Bioessays* 41.12 (2019), p. 1900168.

[29] Zhenheng Tang et al. "Communication-efficient distributed deep learning: A comprehensive survey". In: *arXiv preprint arXiv:2003.06307* (2020).

[30] *TensorFlow CNN architecture*. `https://www.tensorflow.org/tutorials/images/cnn`. Accessed: 2021-02-24.

[31] Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Vol. 620. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[32] Keith Worden and Graeme Manson. "The application of machine learning to structural health monitoring". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 365.1851 (2007), pp. 515–537.

[33] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

[34] Chunming Xu and Scott A Jackson. *Machine learning and complex biological data*. 2019.

[35] Qiang Yang et al. "Federated machine learning: Concept and applications". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2 (2019), pp. 1–19.

[36] Marijana Zekić-Sušac, Sanja Pfeifer, and Nataša Šarlija. "A comparison of machine learning methods in a high-dimensional classification problem". In: *Business Systems Research Journal* 5.3 (2014), pp. 82–96.

[37] Zhongheng Zhang et al. "Opening the black box of neural networks: methods for interpreting neural network models in clinical applications". In: *Annals of translational medicine* 6.11 (2018).

List of Figures

List of Tables

A. Appendix

A.1 Supplementary Figures

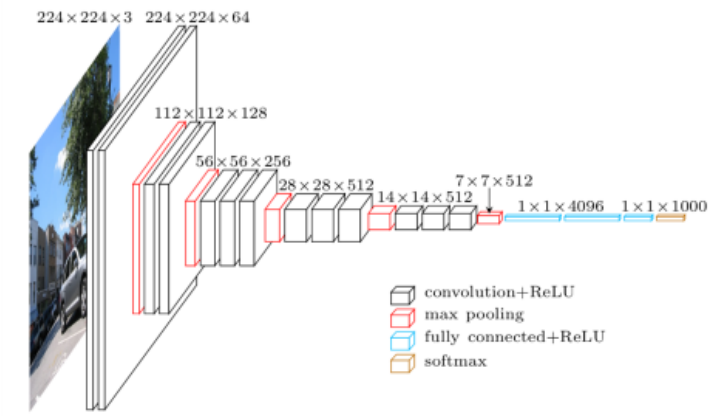
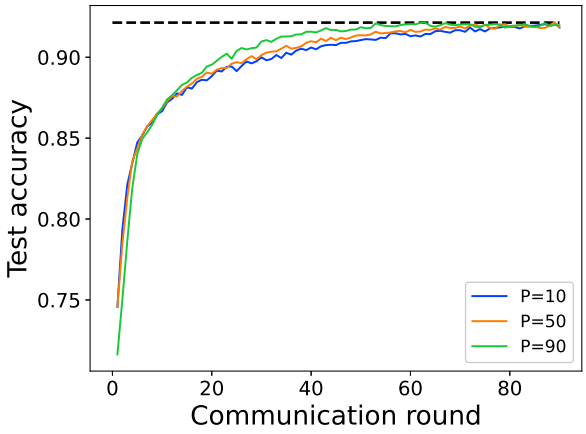
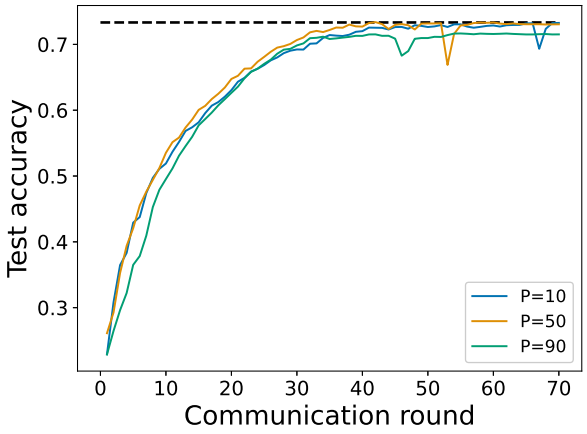


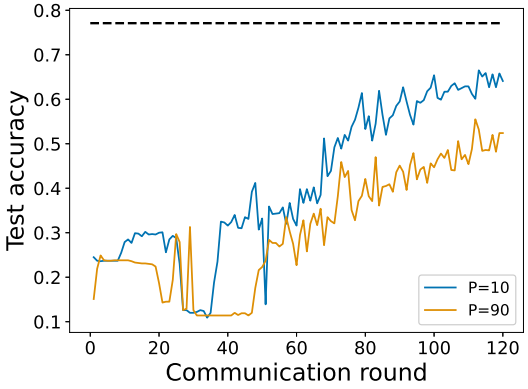
Figure A.1: Structure of VGG16 model 224x224x3 input layer and with 1000 output neurons



(a) *2CFNN-FMNIST*

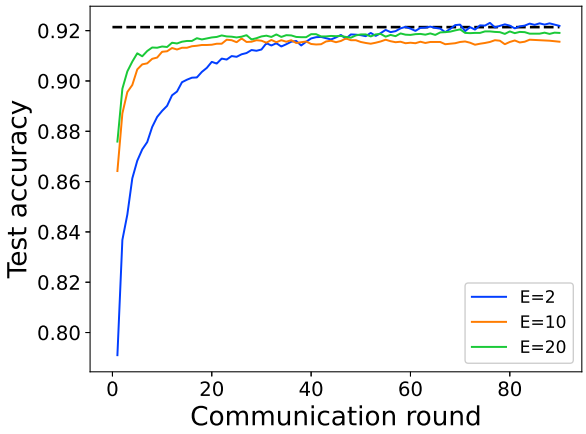


(b) *3CFNN-CIFAR-10*

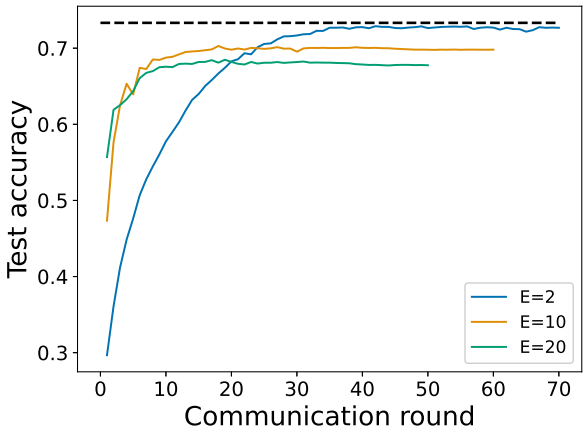


(c) *VGG16-CCH*

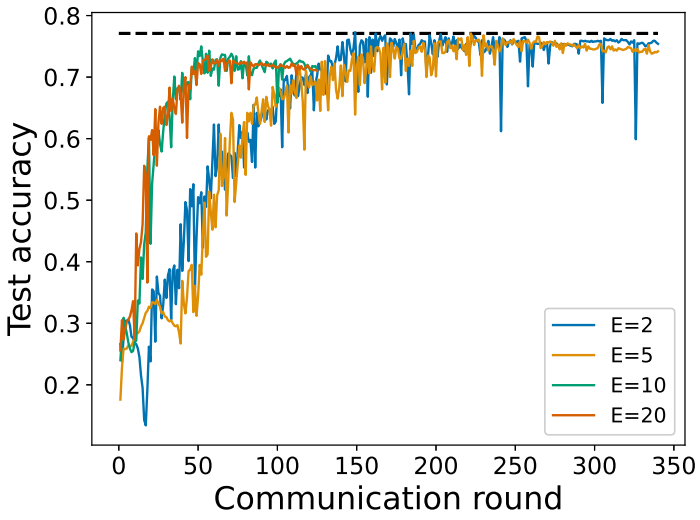
Figure A.2: Red-green visual impairments: The *GS* approach



(a) *2CFNN-FMNIST*

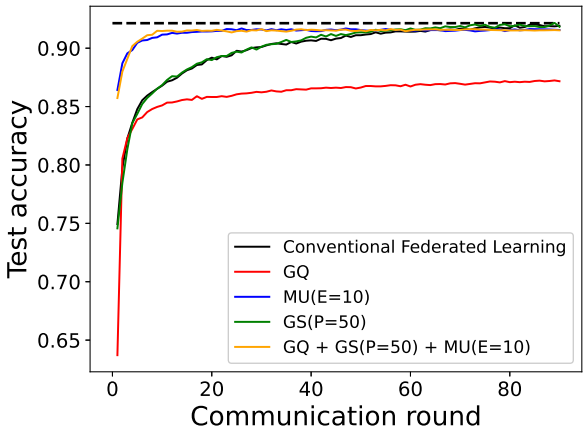


(b) *3CFNN-CIFAR-10*

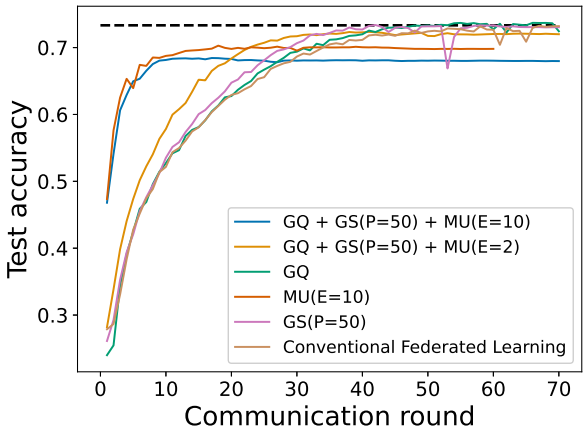


(c) *VGG16-CCH*

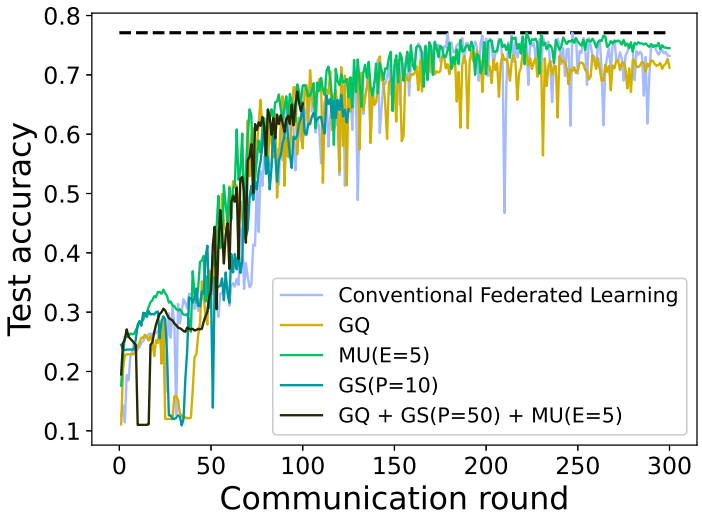
Figure A.3: Red-green visual impairments: The *MU* approach



(a) *2CFNN-FMNIST*



(b) *3CFNN-CIFAR-10*



(c) *VGG16-CCH*

Figure A.4: Red-green visual impairments: Approach combination *GQ + GS + MU*

A.2 Supplementary Tables

```yaml
dataset_name: "cifar10"
model_type: "tlcnn"
output_path: "cifar10/tlcnn/"
image_shape: !!python/tuple [32, 32]
data_type: "float32"
max_value: 255
label_count: 10
batch_size: 64
learning_rate: 0.05
loss: "categorical_crossentropy"
metrics: ["accuracy"]
max_iterations: 50
client_count: 10
quantification_flag: True
quantification_options: {"dtype": np.float16}
more_local_updates_flag: True
more_local_updates_options: {"local_iterations": 10}
sparsification_flag: True
sparsification_options: {"percentile": 50}
dataset_type: c
non_iid: False
```

Listing A.1: Example configuration yaml-file for an federated IID simulation with the approach settings GQ, QS(P=50) and MU(E=10)

absl-py==0.10.0	protobuf==3.13.0
aiohttp==3.6.2	pyasn1==0.4.8
alabaster==0.7.12	pyasn1-modules==0.2.8
arrow==0.17.0	Pygments==2.7.3
astunparse==1.6.3	pyparsing==2.4.7
async-timeout==3.0.1	PySocks==1.7.1
attrs==20.2.0	python-dateutil==2.8.1
Babel==2.9.0	python-dotenv==0.15.0
beautifulsoup4==4.9.3	pytest==6.2.2
bibtexparser==1.2.0	pytz==2020.1
cachetools==4.1.1	PyYAML==5.3.1
certifi==2020.6.20	requests==2.24.0
chardet==3.0.4	requests-oauthlib==1.3.0
cycler==0.10.0	rsa==4.6
docutils==0.16	scholarly==1.0
fake-useragent==0.1.11	scipy==1.5.2
free-proxy==1.0.2	seaborn==0.11.0
future==0.18.2	selenium==3.141.0
gast==0.3.3	six==1.15.0
google-auth==1.22.0	snowballstemmer==2.0.0
google-auth-oauthlib==0.4.1	soupsieve==2.0.1
google-pasta==0.2.0	Sphinx==3.3.1
grpcio==1.32.0	sphinx-rtd-theme==0.5.0
h5py==2.10.0	sphinxcontrib-applehelp==1.0.2
idna==2.10	sphinxcontrib-devhelp==1.0.2
imagesize==1.2.0	sphinxcontrib-htmlhelp==1.0.3
importlib-metadata==2.0.0	sphinxcontrib-jsmath==1.0.1
Jinja2==2.11.2	sphinxcontrib-qthelp==1.0.3
Keras==2.4.3	sphinxcontrib-serializinghtml==1.1.4
Keras-Preprocessing==1.1.2	stem==1.8.0
kiwisolver==1.2.0	tensorboard==2.3.0
lxml==4.6.2	tensorboard-plugin-wit==1.7.0
Markdown==3.2.2	tensorflow==2.4.0
MarkupSafe==1.1.1	tensorflow-datasets==4.2.0
matplotlib==3.3.1	tensorflow-estimator==2.3.0
multidict==4.7.6	termcolor==1.1.0
numpy==1.19.2	typing-extensions==3.7.4.3
oauthlib==3.1.0	urllib3==1.25.10
opt-einsum==3.3.0	Werkzeug==1.0.1
packaging==20.7	wrapt==1.12.1
pandas==1.0.5	yarl==1.6.0
Pillow==7.2.0	zipp==3.4.0

Table A.1: All Python packages that were utilized throughout the thesis with their respective version