BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

Secure Federated Learning of Conditional Generative Adversarial Networks

Supervisor

Asist. Drd. Mursa Bogdan-Eduard-Mădălin
Prof. Dr. Andreica Anca

Author
Pauliuc Eduard-Timotei

2023

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Antrenarea federată privată a rețelelor generative adversare condiționate

Conducător științific

Asist. Drd. Mursa Bogdan-Eduard-Mădălin
Prof. Dr. Andreica Anca

Absolvent
Pauliuc Eduard-Timotei

2023

ABSTRACT

Generative Adversarial Neural Networks (GANs) are part of state-of-the-art generative solutions across multiple disciplines such as medicine, finance, art and many others. In the current Artificial Intelligence Boom, as it is called, more and more industries seek ways to integrate these models into their workflows. The significant problem that became clear in the last decade is the increased amount of private user information exposed in data breaches, as many approaches to training better and better models require vast quantity of data. In order to make progress with respect to the models obtained while preserving the privacy of the data, new approaches have been published and used, of which an example is Federated Learning (FL). Cross-Silo FL enables the training of an AI model across multiple clients, without exposing their data to the other parties. This work explores the feasibility of employing FL in the training of a state-of-the-art generative model, Pix2Pix, which is widely used in complex image-to-image translation tasks. The NVIDIA FLARE training framework is extended in order to create an application that can train GAN networks, with included metrics computation and visualisation features. This work presents how the set up process should be conducted and the steps to train a model. The resulting application is further used to train a Pix2Pix model implementation in a collaborative scenario with multiple clients and analyze its performance, compared to other training methods that use the same model and data, but jeopardize the private data. The metrics presented are the Fréchet Inception Distance and visual comparison of the generated images, which are used to conclude whether large generative models can be trained in a real scenario using Federated Learning.

Contents

Chapter 1

Introduction

1.1 Motivation

Artificial Intelligence is a trending subject, as it is becoming increasingly useful to our society. More and more of humanity's problems are solved using a diverse range on AI models. However, most of these models are trained with large volumes of data, which seems to become an invaluable resource of the future. Data is collected from most of our interactions with technology, which rises interest in the question of data privacy. Data breaches have become recurrent news, regulations such as the General Data Protection Regulation (GDPR) have been imposed by governments and it seems that data privacy will be as important in the future as AI technologies.

Numerous AI applications, particularly those involving large models, rely on multi-party collaborations to enhance the accuracy of their predictive models. This is evident across various domains, whether it's a network of hospitals training a neural network using CT scans, banks developing financial prediction models, or educational institutions tracking student progress based on numerous parameters. However, a common challenge emerges in these instances: the critical data cannot be shared outside their respective institutions. This is where innovative approaches like federated learning come into play, providing effective solutions by addressing these constraints.

1.2 Objectives

The objective of this thesis is to answer whether Federated Learning is a practical solution to the scenario of multiple clients willing to collaborate in order to train a stat-of-the-art conditional generative adversarial network on sensitive data, without compromising its privacy. The focus points are finding the most adequate architecture for a distributed scenario, observing performance differences (in terms of

resulting model quality, communication costs and set-up overhead) and analysing the privacy of the data during and after the training process.

1.3 Structure

This thesis is organized into six chapters, each focusing on different aspects of the study. After the introduction, the next chapter provides an introduction into Artificial Neural Networks, exploring the history of this research field. Chapter three introduces Conditional Generative Adversarial Networks (cGANs) and the Pix2Pix model, setting the stage for understanding the experimental model used in this work. Chapter four offers an introduction to private federated learning, laying the groundwork for understanding the advantages and complexities of a multi-party training collaboration. In the fifth chapter, related work in the field is discussed, providing context and comparison for this study. Chapter six details the experiments conducted and their results. The original contribution of this chapter is the application created for these experiments. This application is customizable to train any GAN model using the NVFlare framework – a development not available in any public implementations at the time of this work. Finally, the last chapter draws conclusions from the study, synthesizing the findings from the experimental application of federated learning on Pix2Pix, and examining its implications for real-world multi-party collaborations. Multiple solutions and methods will be investigated and their results will be compared. In the conclusion, future work and open problems are mentioned.

1.4 Original contributions

While this research is not the first to explore Federated Learning outcomes, it stands out by focusing on practical implementations. There is a wealth of existing research on various algorithms and strategies, but many models trained with Federated Learning remain in the experimental phase and have not analyzed real-world usage.

The model used for evaluation in this work, Pix2Pix, a large Conditional Generative Adversarial Network, is known for its useful medical applications. However, it has not yet been trained in a federated scenario, in order to analyze the performance loss and to determine whether it is suitable for a multi-party training collaboration. This thesis aims to find whether current frameworks and cloud technologies provide the infrastructure required to train a large conditional GAN models in a federated method without major impact on model performance or training complexity.

An additional original contribution was the extension of the Federated Learning framework employed. At the time of this work, no public implementations for training Generative Adversarial Network (GAN) models within this framework were identified. The application created for the experiments in this work can be customised to train any GAN model using the NVFlare framework.

1.5 Scientific Presentations

Part of the results presented in this thesis are published in a paper, alongside my colleague Octavian Trifan, where we present a holistic view of Privacy-Preserving GANs. My contribution is the private training part of the model life cycle, with Federated Learning, while Trifan presents the possibility of using Fully Homomorphic Encryption in order to obtain private inference, which is useful when the data to be used as input or the generated output contain sensitive information which should not be exposed. These input and output objects can be encrypted, the user having solely access to the information. We presented this paper at the Computer Science Students' Scientific Communications Session 2023, organized by Babeș-Bolyai University and Technical University of Cluj-Napoca.

Chapter 2

Artificial Neural Networks

2.1 Beginnings

Artificial Neural Networks are based on collections of connected nodes called neurons, which take their inspiration from the human brain. A basic example of this type of model is a linear network, characterized by having only one layer of output nodes. The inputs are directly fed to the output nodes, multiplying their value by a series of weights. This method has the name of Linear Regression and it was used by Adrian-Marie Legendre (1805) and Johann Carl Friedrich Gauss (1795). In 1920, physicists Ernst Ising and Wilhelm Lenz introduced the first non-learning recurrent neural network, which had feedback connections, making the network more similar to the human brain, as it is an essential mechanism for implementing the behaviour of memory in processing [Sch22].

In the quest to mirror human cognition, it was understood that a complex computational representation of the human brain's intricate network of neurons had to be developed. Researchers and scientists hypothesized that modeling the billions of interconnections between neural cells, each with its capacity for processing and transmitting information, would provide a promising approach to crafting intelligent systems, as discussed at the first conferences on AI [con53][Bru17]. In 1958, psychologist Frank Rosenblatt published the first artificial neural network, the multi-layer perceptron [Ros58], but he did not have a complete deep learning algorithm yet.

Deep Neural Networks

A simple neural network consists of only three layers: an input layer, a hidden layer, and an output layer. Each layer contains a number of nodes or "neurons," and each neuron in one layer is connected to every neuron in the next layer. Deep Neural Networks (DNNs), on the other hand, as shown in Figure 2.2, are a significant extension

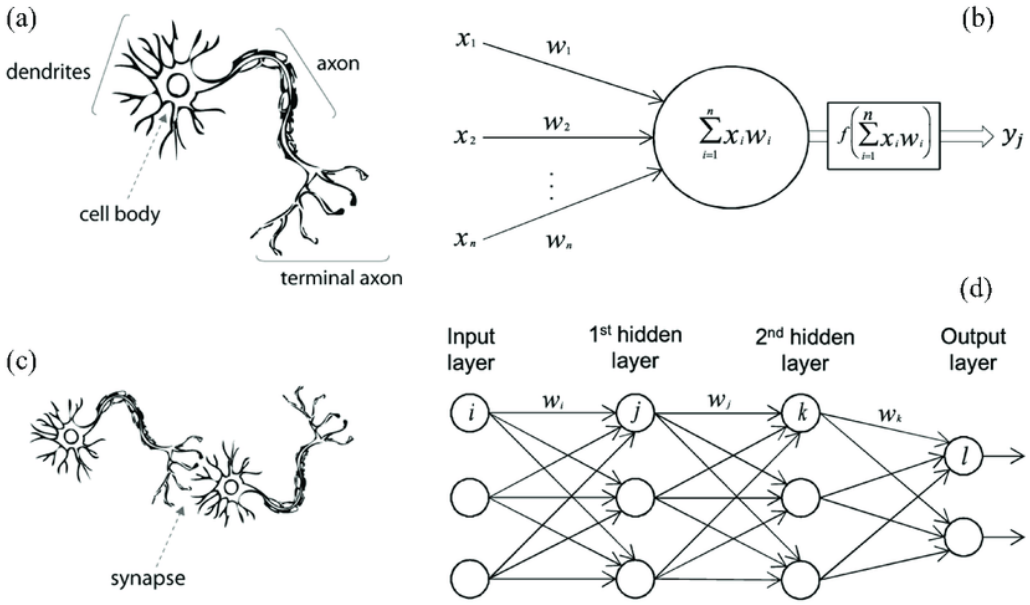


Figure 2.1: A comparison between a human neuron (a) and an artificial neuron (b). For an artificial neuron, its activation function is a linear combination of all its inputs followed by a nonlinear transformation. Deep neural networks (d) correspond to a network of biological synapses (c) [MHA20].

of these simple networks. The term "deep" refers to the presence of multiple hidden layers in the network, rather than just one. This allows the network to model more complex, higher-level features in the data. The advantage of deep neural networks lies in their ability to automatically learn and represent features from raw data, a process known as feature learning or representation learning [SML+21]. These Deep Networks represent human neurons synapses, as seen in Figure 2.1.

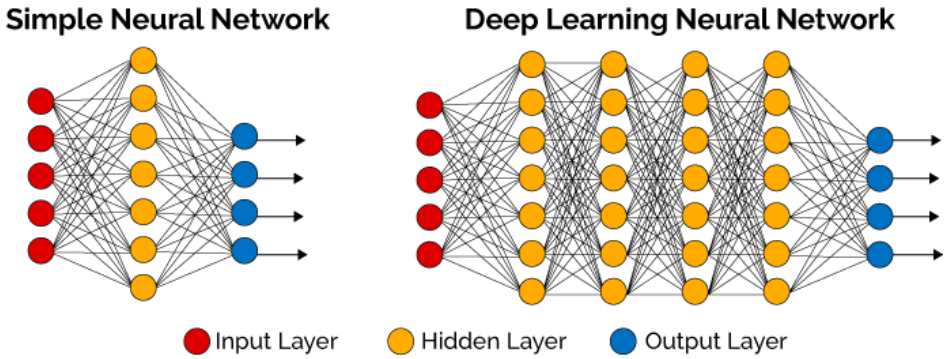


Figure 2.2: A visual comparison between Simple and Deep Neural Networks, which can have a variable number of hidden layers [dee18].

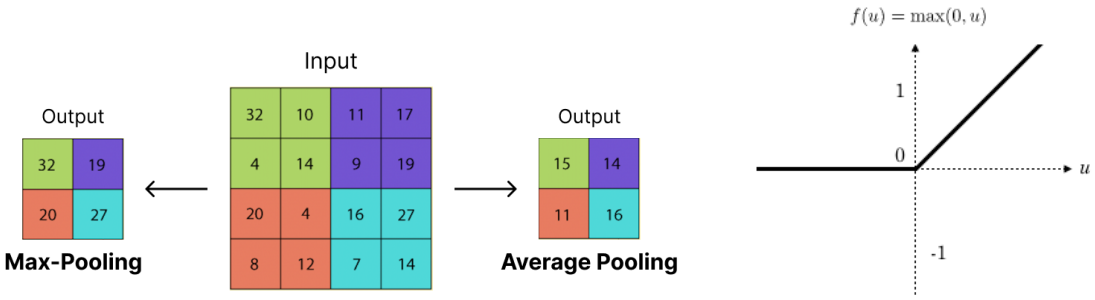
The first successful deep learning is attributed to Alexey Ivakhnenko and Valentin Lapa, which introduced the first algorithm to train deep multi-layer perceptrons in 1965 [ILE65]. However, the first to introduce the term of "deep learning" was Dechter, in 1986 [Dec86].

Backpropagation

Improvements to the previous models were published by Shun'ichi Amari in 1967, when he successfully used Stochastic Gradient Descent (SGD) to train deep networks. Neural networks increased in popularity with the advancements in computational power and the introduction of backpropagation by Seppo Linnainmaa, the algorithm that is widely used today to train neural networks [Sch22].

2.2 Convolutional Neural Networks

In 1979, Kunihiko Fukushima introduced the convolutional neural network (CNN), consisting of convolutional and down-sampling layers, naming it "neocognitron" [Fuk80]. He used the ReLU (rectified linear unit) activation function (Figure 2.3), also introduced by him in 1969 [Fuk69], which is the most popular and widely-used activation function in CNNs, with significant applications in computer vision.



(a) Pooling (b) ReLU

Figure 2.3: (a) Max-Pooling and Average Pooling, the two most used Down-sampling methods used in CNNs. (b) The ReLU activation function.

The architecture of a CNN can be seen in Figure 2.4. Pooling, exemplified in Figure 2.3, is usually done with Max-pooling or Average Pooling. These are important as they reduce the resolution for the feature map, required for classification, named Down-sampling [Fuk79].

Convolutional Neural Networks have been widely used in medical imaging, audio processing and pattern recognition. The technological breakthroughs in Graphical Processing Units (GPUs) development also provided the computational resources required to create more powerful models [Sch22]. The most cited and popular CNN is ResNet, short for Residual Neural Network, which addresses the challenge of training very deep neural networks by introducing residual connections. These connections allow the network to skip certain layers and pass the information directly to subsequent layers, thereby creating shortcuts. Using these connections, ResNet facilitates the training of extremely deep neural networks with hundreds or even

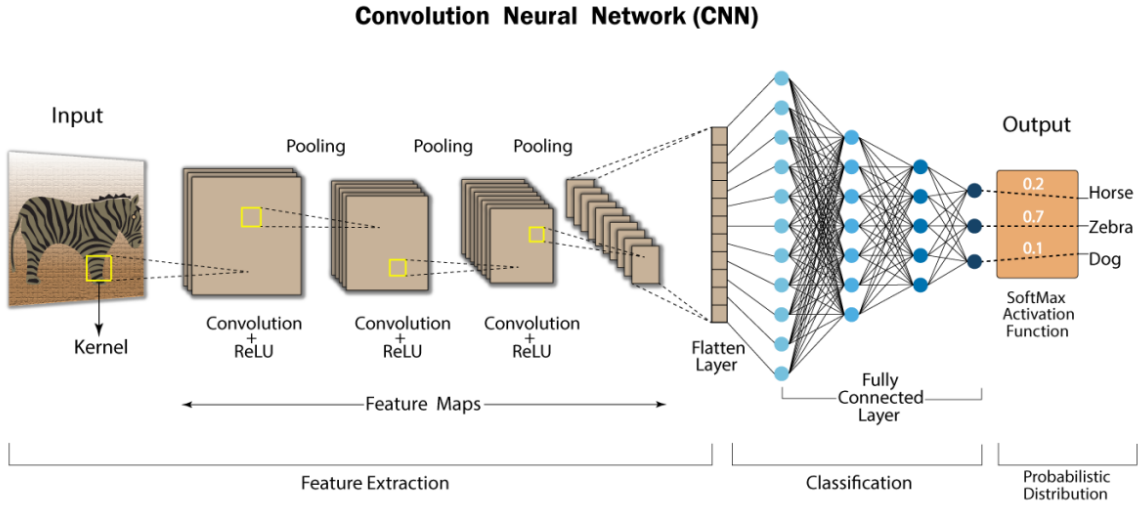


Figure 2.4: A Convolutional Neural Network model. In the feature extraction section, multiple Convolutional Layers extract the features. Each layer consists of a kernel applied to the input, followed by pooling and the ReLU activation function. This is followed by classification layers which output a probabilistic distribution into labels [cnn].

thousands of layers. This architecture has shown exceptional performance in image classification, object detection, and other computer vision tasks, becoming a cornerstone in the development of deep learning models [HZRS15].

Chapter 3

Conditional Generative Adversarial Networks

3.1 History of GANs

The first formal introduction of the GAN training loop is accredited to Ian Goodfellow [GPAM+14] using two Deep Neural Networks [Sch15], a Generator and a Discriminator that train in a min-max game. The Generator is trained to generate samples from the data distribution and the Discriminator predicts if a sample was generated by the Generator or belongs to the training data. The architecture of this network can be seen in Figure 3.1.

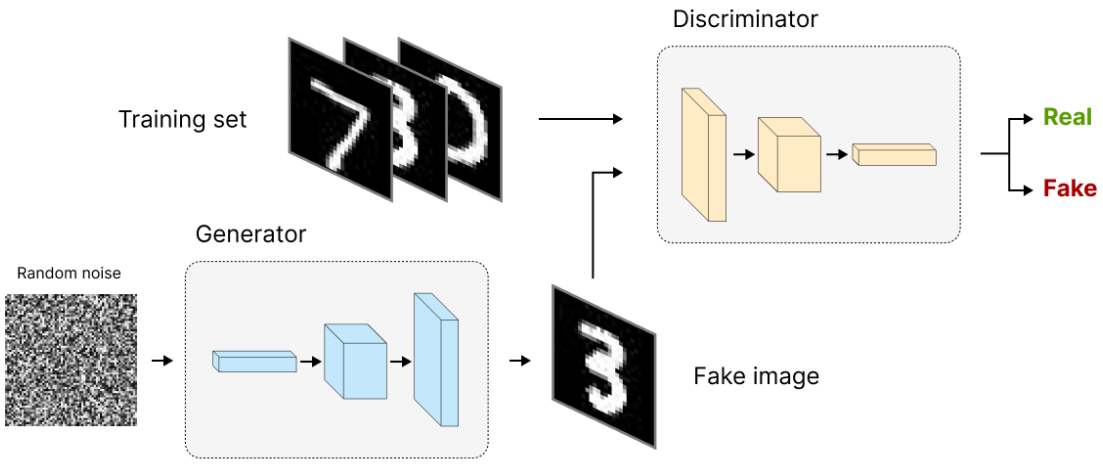


Figure 3.1: Architecture of a GAN

Goodfellow explains that the Generator can be thought of as counterfeiters trying to create fake samples, while the Discriminator acts as the police attempting to detect these fakes. The competition between the networks drives both models to

improve their methods until the generated samples become indistinguishable from the real data. The models are typically defined by multi-layer perceptrons and can be trained using backpropagation. The GAN framework does not require Markov chains, with sequences of possible events, or unrolled approximate inference networks during training or sample generation, which have been used in the other popular generative models.

The proposed GAN model has two functions, G, the generator, and D, the discriminator. The Generator take as input a random noise vector z and learns a mapping $G : z \rightarrow y$, where y is an output image. The discriminator takes as input an image and outputs a scalar value, where $D(x)$ signifies the probability of x being a real image, not generated by G.

D is trained to correctly classify both the training samples and instances generated by G. In parallel, G is trained to minimize the value of $\log(1 - D(G(z)))$, which encourages G to produce samples that D is more likely to classify as real.

3.2 Conditional GANs

In his paper, Goodfellow, when presenting future work capabilities, includes the possibility of creating a conditional generative, by providing an additional input c to both the Generator and Discriminator models, where $G : \{c, z\} \rightarrow y$ and $D : \{c, x\} \rightarrow p$, where p is a probability scalar. In an unregulated generative model, the modes of generated data aren't specifically controlled. Yet, when extra information is used to condition the model, as demonstrated by Mehdi Mirza in 2014 [MO14], the data generation process can be guided.

The objective of the classical conditional GAN, where G tries to minimise the objective and the adversarial D tries to maximize it is formulated as

$$\min_G \max_D \mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{c,x}[\log D(c, x)] + \mathbb{E}_{c,z}[\log(1 - D(c, G(c, z)))] \tag{3.1}$$

Goodfellow specifies that during training it is better, showing improved starting gradients, to train G to maximize $\log D(c, G(c, z))$. This optimization is used in most implementations of conditional GANs.

3.2.1 Pix2Pix

Mirza employed input text labels to condition the network, which can be beneficial for certain applications. However, in some situations, conditioning the GAN with an image input might be preferred. Addressing this, Phillip Isola and his team pro-

posed a solution for image-to-image translation challenges with the Pix2Pix model [IZZE18]. This model utilizes a U-Net style generator and a Markovian discriminator (the difference to a normal discriminator can be seen in Figure 3.2), which was termed PatchGAN in their paper.

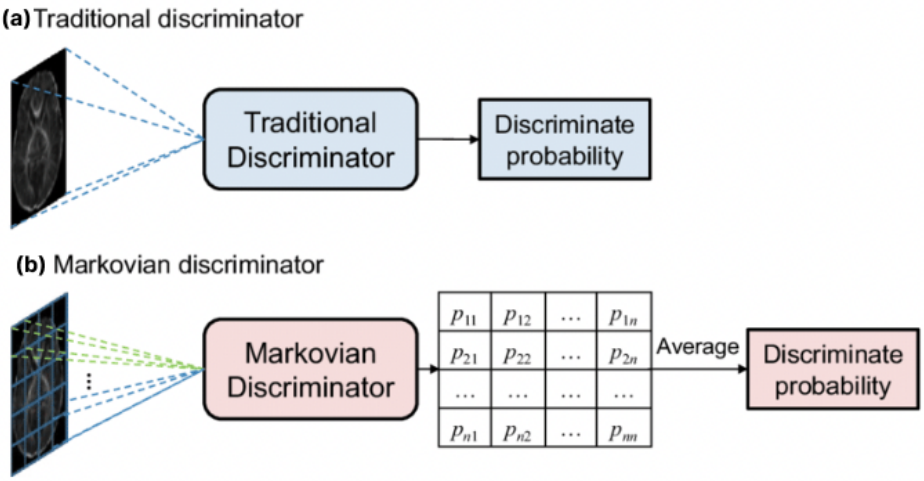


Figure 3.2: The difference between the traditional discriminator, which outputs a scalar value, as described by Goodfellow [GPAM+14] and a Markovian Discriminator, which outputs a matrix of probabilities, one for each local region [ZWL+20].

In addition to the classical cGAN objective (Equation 3.3), Isola also used a more classical loss function, L1, defined as:

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{c,x,y}[\|y - G(c,z))\|_1] \tag{3.2}$$

This traditional L1 loss has been proven [IZZE18] to produce less blurry results. The ultimate goal of the Pix2Pix model, with λ determining the influence of the L1 norm, can be expressed as follows:

$$\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G) \tag{3.3}$$

The authors have found that the best performing model architecture (as seen in Figure 3.3), is with discriminator patch size of 70x70.

Use cases

In the original Pix2Pix paper, multiple use cases are shown (Figure 3.5). The model is able to perform image segmentation, reverse image segmentation, coloring from grayscale photos and real image generation from sketches, without changes to the architecture or the objectives.

The practical utility of the Pix2Pix model has been demonstrated across various fields of research, particularly within the medical field, where it is one of the

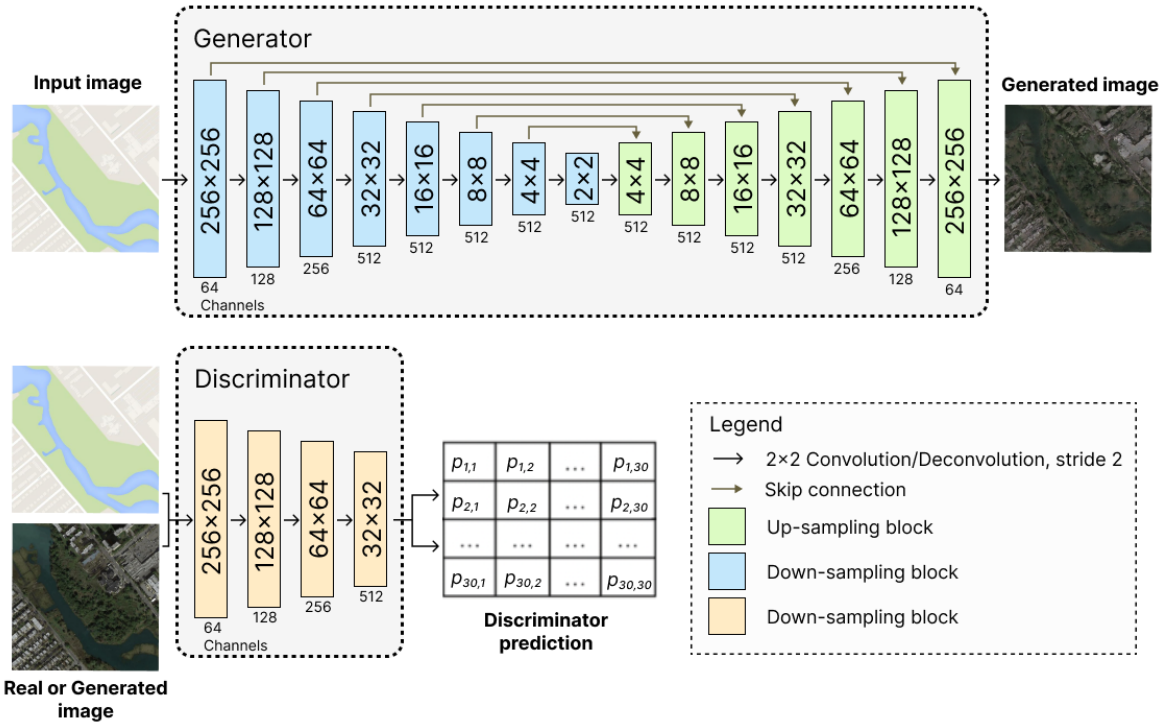


Figure 3.3: The architecture of the Pix2Pix model. The example sizes are for an input image of 256px by 256px. The generator takes as input an image and generates an output image conditioned by this input. The discriminator takes as input two images, the condition image and a generated or real image and outputs a prediction map, with a prediction for each patch.

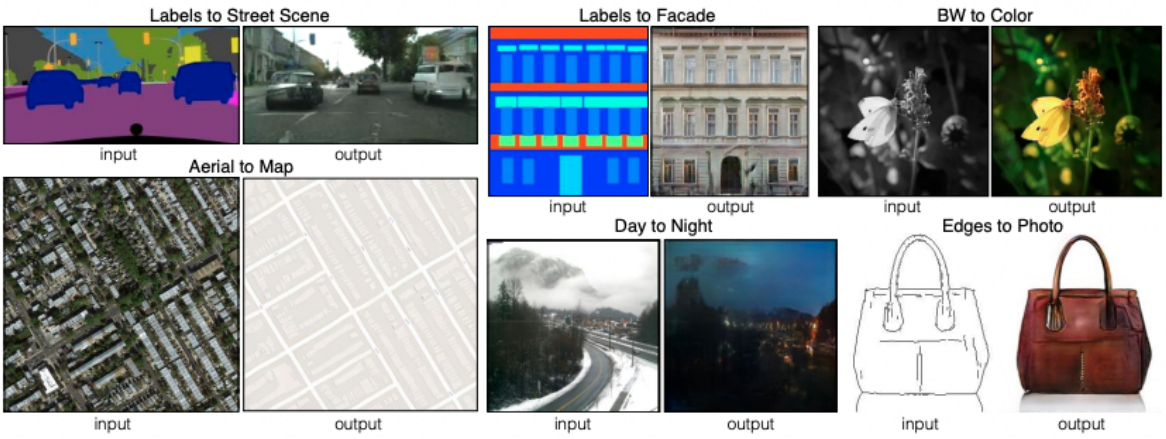


Figure 3.4: Pix2Pix use cases, using the same objectives and models architecture, trained with different data [IZZE18].

most used architectures for image-to-image translations. For instance, the application of Pix2Pix GAN in denoising low dose myocardial perfusion SPECT (Single Photon Emission Computed Tomography) images has proven to surpass conventional methods of post-reconstruction filtering and convolutional auto-encoder techniques, as evidenced in [SDL+21] (Figure 3.5). In another research, the Pix2pix model was successfully employed to produce Magnetic Resonance Imaging (MRI) images, as documented in [AA22].

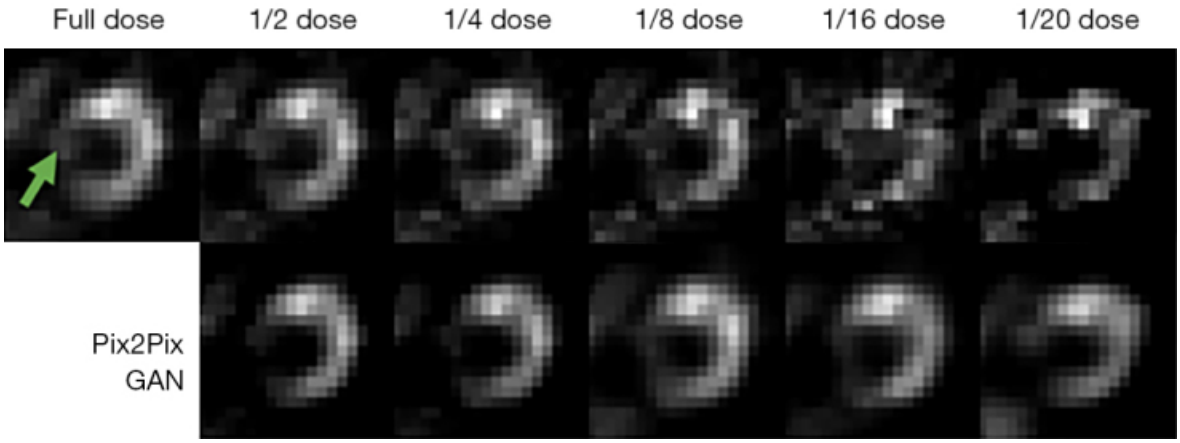


Figure 3.5: The generator receives as input low dose SPECT images, which contain noise, and is trained to transform these into images that resemble the corresponding full dose SPECT images [SDL+21].

3.2.2 Evaluation Metrics

Finding a good evaluation metric for generative networks is a difficult task, which has been addressed in literature many times. One of the metrics that is widely used is the Fréchet inception distance (FID) [HRU+18]. It uses the Fréchet distance, named after Maurice Fréchet [Fré57].

The first step is to extract the feature vectors of the images using the pretrained Inception-v3 model. These feature vectors effectively represent a compressed form of the images that still retains their main characteristics. After fitting the distributions with these feature vector, the following are obtained: $\mathcal{N}(\mu, \Sigma)$ for the real data and $\mathcal{N}'(\mu', \Sigma')$ for the generated data. The FID score is computed with:

$$FID(\mathcal{N}(\mu, \Sigma), \mathcal{N}'(\mu', \Sigma')) = \|\mu - \mu'\|_2^2 + Tr(\Sigma + \Sigma' - 2(\Sigma^{\frac{1}{2}} \cdot \Sigma' \cdot \Sigma^{\frac{1}{2}})^{\frac{1}{2}}) \tag{3.4}$$

The FID metric delivers promising results in terms of its ability to distinguish between different outcomes, resilience, and computational speed. It appears to be a robust measure, despite its reliance on only the first two moments of the distri-

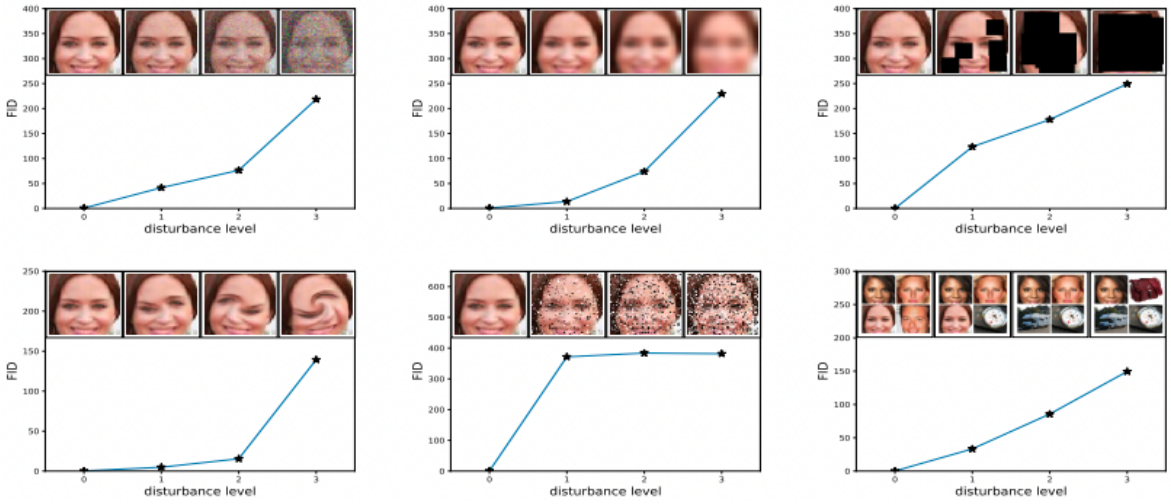


Figure 3.6: The FID metric can detect various types of disturbance when comparing a set of images generated by a generative model to a set of real images [HRU+18].

butions. FID has been demonstrated to align well with human assessments and is more resistant to noise compared to the Inception Score (IS). Notably, FID is capable of identifying when a model is generating only a single image per category, a phenomenon known as intra-class mode dropping. While such a model might achieve a high IS, its FID would likely be poor. Another advantage over the IS metric is that the FID score decreases when various types of distortions are introduced to images (Figure 3.6). IS and some other metrics evaluate the diversity and quality of generated samples, but FID is distinct in its ability to measure the difference between the distributions of generated and real images.

Chapter 4

Private Federated Learning

4.1 Distributed Learning

The need for a distributed approach

One of the problems when using generative networks is that they require large training data sets. For example, the work that generates MRI images [AA22] trained on 2000 patient scans. In a application that increases image resolutions [LTH+17], 350000 images from ImageNet database have been used.

In many practical cases, entities do not possess enough data to train high quality GANs, especially in complex use cases. Collaboration between multiple entities that collect similar data can result in a better model than any entity would be able to create with its own data. However, centralising all the data in a single location is not a secure solution, since the data has to leave the premises of the participants, making it susceptible to attacks, during transportation or directly to the centralised location. Such attacks could result in data leaks, which have risen in popularity in recent years. The number of healthcare data breaches (Figure 4.1) is even more alarming, as it can expose sensitive patient data.

Data anonymization prior to centralization could be considered as a potential solution. However, studies, such as the one referenced [RHM19], have demonstrated that an overwhelming 99.98% of Americans can be correctly re-identified in any data set by utilizing only 15 demographic attributes. This suggests that data anonymization alone is insufficient to address privacy worries.

4.1.1 Federated Learning and Federated Averaging

It is clear that there's a demand for a technique that allows multiple parties to cooperatively train a shared model without requiring data to leave contributor sites. This issue was tackled by H. Brendan McMahan, Eider Moore, and their colleagues

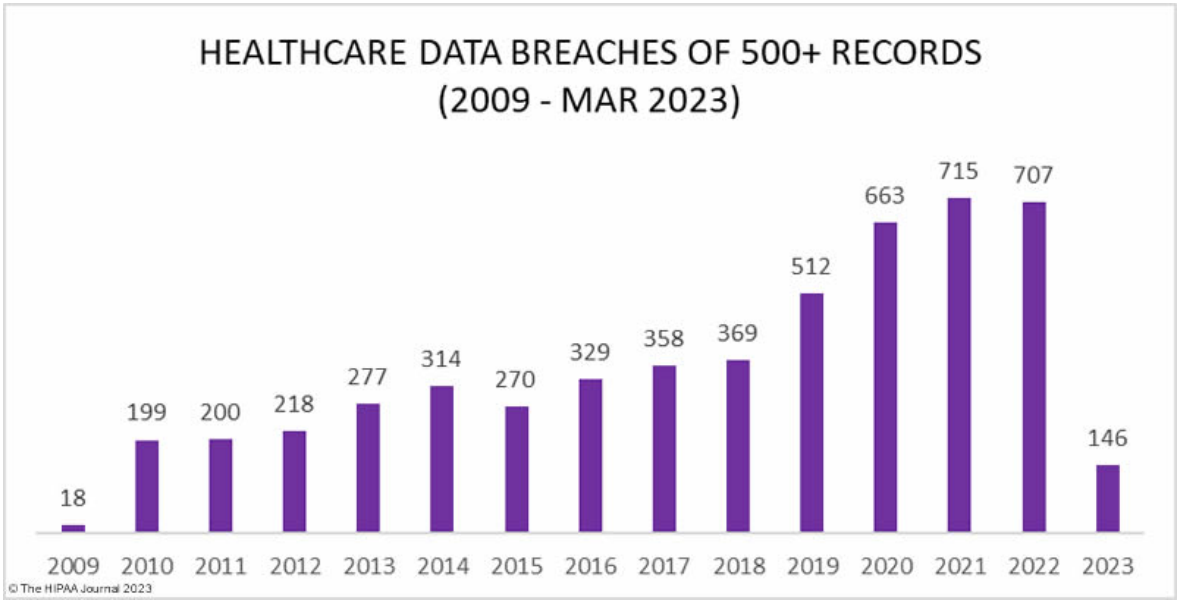


Figure 4.1: Number of medical data breaches by year [bre23].

at Google in 2016 [MMR+23]. In their research, they explored a method that leverages user mobile devices to train models for tasks like image classification and language modeling. This involved utilizing user photos and their text inputs, all while avoiding the centralization of this sensitive data.

The concept of distributing the training process to clients was not new. The previously used method were the Federated Stochastic Gradient Descent (FedSGD) algorithm or Asynchronous SGD algorithms [DCM+12]. In the FedSGD method, the server sends the current model parameters to all participating clients. Each client then computes the gradient of the model parameters using their local data and sends this gradient back to the server. The server then updates the model parameters by applying these gradients, generally by taking a weighted average, proportionally to the number of samples a client has.

The problem with this approach was the increased number of communication rounds, as for each learning step the server and clients communicate the model and gradients. The paper from Google introduced the FedAvg algorithm, which aims to overcome the downsides of FedSGD. In FedAvg, clients do not send the gradient after one step of local SGD. Instead, each client computes multiple steps of local SGD (i.e., performs local updates), and then sends these locally updated model parameters to the server. The pseudocode for this algorithm can be seen in Figure 4.2.

The server averages these models, again often using a weighted average where the weights are proportional to the number of local samples. Because each client performs multiple steps of SGD, the number of rounds of communication between server and clients is greatly reduced compared to FedSGD, leading to efficiency

gains.

Another topic in the context of federated learning is data distribution. Independent and Identically Distributed (IID) refers to a data scenario where all samples in a dataset are generated from the same distribution and are independent of each other. However, this assumption may not always hold true in a federated learning environment where data is distributed across numerous devices or servers, leading to non-IID data. Factors such as varied user behaviors and uneven data collection can result in different data distributions across devices. This introduces challenges as models trained on one client's data may not generalize well to others. Federated Averaging (FedAvg) addresses this issue by allowing each client to perform multiple rounds of local updates before sending the model parameters to the server for averaging. In FedAvg, because each client updates its model with multiple steps of SGD, the model can better fit the local data, which can be especially beneficial in the non-IID setting.

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $m_t \leftarrow \sum_{k \in S_t} n_k$
  $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

Figure 4.2: The pseudocode for the FedAvg algorithm. C is the fraction of clients to be used in a training round. n_k is the number of data points a client k holds [MMR+23].

4.1.2 NVIDIA FLARE

Federated Learning has been implemented in a number of frameworks such as NVIDIA FLARE, Flower, Fed ML, TensorFlow FL, Open FL and PySyft. NVIDIA FLARE (NVIDIA Federated Learning Application Runtime Environment) [RCW+23] stands out as an open-source, extensible SDK for Federated Learning that is not limited by any specific domain. It provides a platform for researchers and data scientists to transform their current Machine Learning workflows to conform to a federated model. Additionally, it empowers platform developers to construct a safe, privacy-focused solution for collaborative efforts across multiple distributed parties. This positions it as a favorable candidate for executing a federated learning

approach with Pix2Pix.

The framework provides implementations for the most used Federated Learning algorithms. The main features categories are: training workflows, evaluation workflows, privacy preserving algorithms and learning algorithms.

Training workflows define the training process, guiding the server and client communication. Two highlighted workflows are Scatter and Gather (SAG) and Cyclic. The SAG workflow is the most used FL training method, adopting a hub and spoke model. Here, a central Controller dispatches tasks to client Workers, who then send back shareable outcomes, such as model weights in Deep Learning. The Controller then aggregates these results, according to the used learning algorithm.

Evaluation workflows allow measuring metrics such as accuracy in a distributed approach. For example, cross-site model validation is a process that facilitates the evaluation of individual client models as well as the server's global model against each client's dataset. Instead of sharing data, a collection of models is shared to every client site for performing local validation. The server then accumulates the outcomes of these local validations, forming a matrix that compares model performance with each client's dataset.

Privacy-preserving algorithms are incorporated as filters (Figure 4.3), which are applied during data transmission between participants. For example, differential privacy measures include the ability to exclude specific variables (ExcludeVars), truncate weights by percentile (PercentilePrivacy), and implement sparse vector techniques (SVTPrivacy). Moreover, NVIDIA FLARE supports homomorphic encryption, offering encryption and decryption filters that clients can use to encrypt shareable data prior to transferring it to a peer. Although the server does not possess a decryption key, it can still perform operations on the encrypted data, aggregate it, and return the encrypted aggregated data to clients. The clients can subsequently decrypt this data using their local keys, enabling continued local training. These filters can be configured in the client and server configuration files.

Lastly, **learning algorithms** implement the common Federated Learning algorithms, such as FedAvg, FedProx [LSZ+20] and FedOpt [RCZ+21]. The FedAvg algorithm is integrated in the Scatter and Gather workflow, as shown in Figure 4.4. In this approach, the server sends to each client the task to train the global model. After training the model locally, the clients send the updated model or weights difference to the server, which aggregates the responses and then saves the model with a Model Persistor.

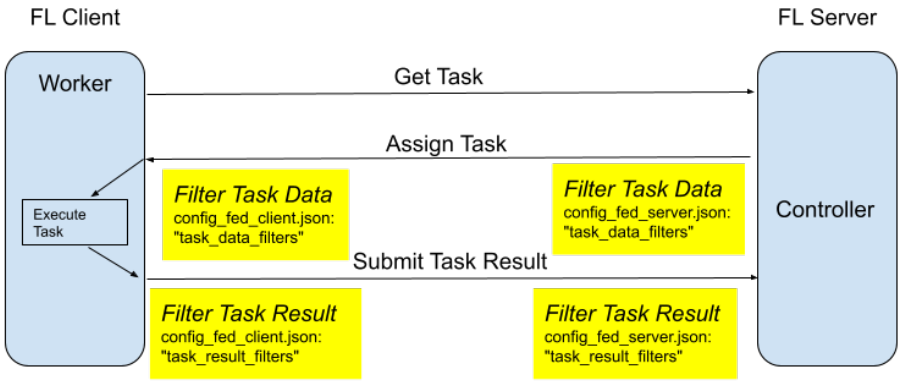


Figure 4.3: Filters can be used to transform the communication object before sending or after receiving it. Some privacy algorithms only modify the outgoing data (such as adding noise), while other require preprocessing (such as decryption in homomorphic encryption) [pro23].

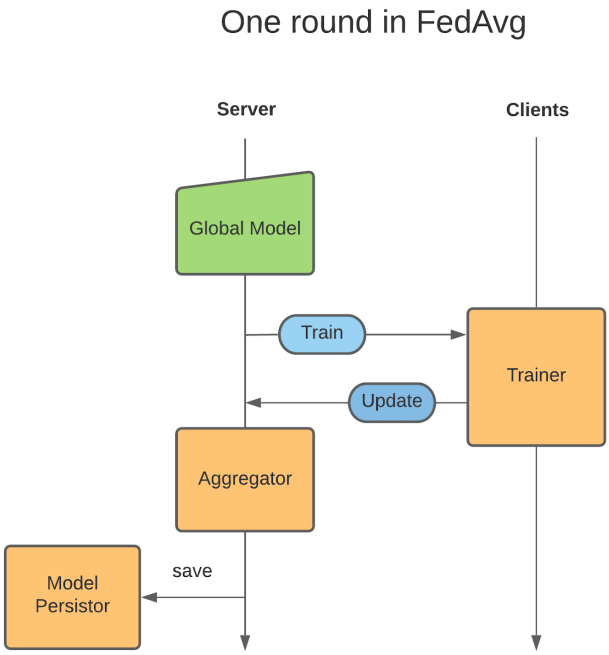


Figure 4.4: A simplified diagram of the FedAvg training round [pro23].

4.2 Privacy Attacks and Solutions

Attacks on Federated Learning systems have been extensively studied in literature [KMA+21]. These can be split into two main categories, Privacy Attacks, which aim to obtain information from the training data set, and Adversarial Attacks on Model Performance, which can lead to training failures or performance loss. Both these types of attacks have been widely studied and solutions have been proposed.

Attacks on Model Performance usually rely on adversaries poisoning the model

updates. In their research, Peva Blalnchard et al. [BEMGS17] have shown that distributed SGD is vulnerable to Byzantine attacks, where a participant failure can lead to the failure of the entire system. This has been acknowledged in many FL frameworks, which support resuming training after a client or server have failed. For example, the NVFLARE framework [RCW+23] allows having fail-over servers, which can take over the training process if the active server crashes.

One of the most acknowledged attacks on data privacy is the Model Inversion Attack [FJR15], which predicts the data the model was trained on. In a paper published by researchers at Apple and Stanford University [BDF+19], the authors have proposed some guidelines that protect users from attacks that infer information from the model updates sent after training. They propose two prevention approaches, which are exemplified in the case where the model trains on predicting next text message words:

(i) The participant submits the model update only if a specified amount of information has been collected (in their example, sufficiently many messages have been sent).

(ii) The data that the model has locally trained on should be sufficiently diverse. In the text messages example, this requires the sent messages to be distinct.

The authors have shown that following these practices highly improves the privacy of the data when submitting the model updates.

Another widely used solution is Differential Privacy [WLD+19], which has been mentioned above, as a part of the filters provided in the NVFLARE framework. In this approach, noise is added on the client side before submitting the model updates for aggregation. In their paper, Kang Wei et al. have shown that increasing the noise in the model updates increases privacy protection, but also lowers the performance of the model, concluding that a balance between these two factors is required.

Chapter 5

Related Work and State-Of-The-Art solutions

Chenyou Fan and Ping Liu have shown that GANs can be trained in federated scenarios and produce qualitative models [FL20]. They have found that synchronization of both Generator and Discriminator produces the best results (Figure 5.1). They have used a DCGAN architecture model, with the input being a class label (a digit for MNIST or category label for CIFAR-10). In their paper, they recommend synchronizing both G and D models when the communication costs are not an issue, and synchronizing only G otherwise.

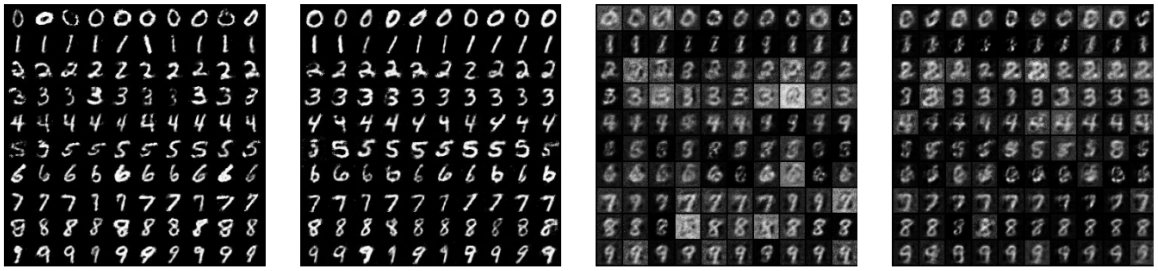


Figure 5.1: Comparison between different synchronization choices, with generated hand-written digits. The federated learning participants could choose to keep G or D models locally, without synchronizing them, but results show worse results for these options [FL20].

Recently, Yuezhou Wu et al. [WKL+22] have shown that conditional GANs trained federately are vulnerable to Deep Leakage from Gradients attacks [ZLH19], and proposed FedCG, a novel FL algorithm that aims to increase user privacy. Their proposed architecture uses, in addition to the generator and discriminator model, a feature extractor E and a classifier C. In the communication rounds, only G and C are uploaded to the server, while the D and E models are kept locally, for privacy concerns. Their proposed method provides a private FL algorithm, with negligi-

ble performance losses. The generator used to benchmark the algorithm is also a version of the DCGAN, which is a less complex model compared to the Pix2Pix generator.

The developers of the NVFlare framework have studied the possibilities of Gradient Inversion Attacks on FL systems [HYM+23a]. They provide ways in which the amount of information leaked in such attacks can be visualised and measured. They have found that increasing the amount of local iteration a client performs and a more diverse dataset decrease the amount of information that can be extracted in such an attack. This motivated the choice of increased number of local iterations per training round in our experiments.

Researchers at Apple have looked into ways to use FL concepts and apply them at scale. One of their publications [GSvD+20] shows how they managed to train a Speaker Identification model, using the FedAvg algorithm with different types of Differential Privacy implementations. Another application was to personalize their models for each client, in a distributed approach, inspired from FL [PSM+21]. They used a new method to personalize the models used in the News recommendation app and in their Automated Speech Recognition functionalities, namely Federated Evaluation and Tuning.

In the field of Federated Learning, other training algorithms have been proposed in the past years, which aim to achieve better results in some scenarios than the popular FedAvg. One of these is FedSM (Federated Super Model) [XLG+23], which addresses one weakness of FedAvg in particular in cross-silo scenario, the difficulty in generalizing the global model when the data is non-IID. Each client at the end of an iteration obtains a personalized model, but the averaged aggregation is not always the most suitable. FedSM proved to be more efficient in these scenarios.

Chapter 6

Experiments

6.1 Objective

The aim of the experimental part of this work is to determine whether federated learning is a practical solution for entities aiming to collaboratively develop high-quality generative models, all while safeguarding their data against potential privacy breaches.

Implementation ease and participant set-up complexity is one of the points analyzed. This assesses the simplicity and efficiency of incorporating federated learning into the existing infrastructure of the participant entities. It explores the process of participant configuration and the resources required to facilitate such a collaborative framework.

Another crucial factor in federated learning is the communication overhead, as potential communication burden can arise from the transmission of model updates across the nodes. The research will examine the volume of data transmitted and its impact on overall system performance.

Given the privacy-preserving premise of federated learning, the study will scrutinize the effectiveness of this approach in protecting the sensitive data of the participating entities from potential privacy attacks.

The final and arguably most critical point of analysis is the quality of the generative models produced via this federated learning approach. The research aims to determine whether this collaborative method can produce models that are on par with, or perhaps superior to, those generated via traditional, non-distributed learning techniques.

The use case chosen for this experiment is one of the use cases presented in the Pix2Pix paper [IZZE18], where the model is trained to generate aerial photo from a map image.

The data set was provided by the authors of the paper. It consists of 2196 pairs of

images (examples can be seen in Figure 6.1) from Google Maps, each having dimensions of 600 pixels by 600 pixels, as both aerial photo and map format. The images are of the region of New York City, split into train and test about the median latitude of the sampling region, with no training pixels appearing in the test set.

This particular use case does not use sensitive data, but it was chosen so that it can be compared with the original results when trained with the same hyperparameters.

The scenario considered in this work is of three entities with the same amount of images. The data is split randomly into three batches, the data therefore is independent and identically distributed (IID). Non-IID scenarios should also be analyzed in future work.

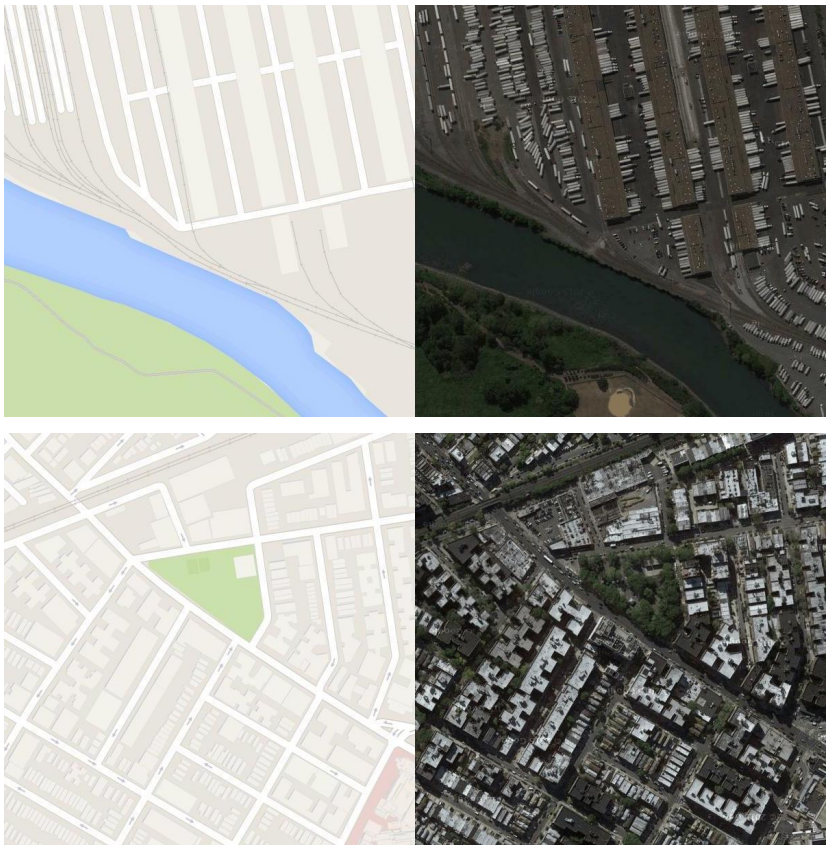


Figure 6.1: Examples of map (left) and aerial (right) images

6.2 Evaluation Metrics

As mentioned in the theoretical chapters, evaluating the performance of GAN architectures can be a difficult task [SGZ+16]. One of the more recent metrics proposed is the Fréchet inception distance, which was used the analyze the performance of the generator model. The used python implementation of this metric is developed and maintained by Maximilian Seitzer and it is available on GitHub [Sei20].

6.3 Methods

The initial phase of the experiments involves the preparation of the model and the corresponding data loaders. The generator and discriminator models are both implemented in PyTorch, following the architecture described in previous theoretical chapters. The specifics of the implementation follow parts of the code from both the original authors' GitHub repository [ZPIE17] and Aladdin Persson's implementation [Per].

The dataset class in this implementation is primarily responsible for the preprocessing of training images. Within its constructor, the class can be tailored to include or exclude the application of random color jitter to the images. This stochastic color perturbation alters the image characteristics such as brightness, contrast, and saturation. When loading the images, the model also applies random horizontal mirroring to the images. The color jitter and the random mirroring augment the data and potentially enhance model robustness.

Moreover, the constructor provides an option to specify the half of the image (left or right) to be used as the input. This is necessary as the input and ground truth images are conjoined within the same image file. Therefore, the class facilitates a customizable setup for preprocessing, which can be aligned according to the demands of the specific experiment or the nature of the images under study. For the experiments in this work, the color jitter is used only when training the model, and the input image is the right half of the files.

The implemented generator takes as input a colored (RGB) image as numpy array with size 256 px by 256 px and outputs generated image of the analogous size. In contrast, the discriminator intakes a pair of images, each of the same size as the generator output, one representing the input image and the other being either the ground truth of the input image or the generated image. The output of the discriminator is a tensor with 30x30 values between 0 and 1. This tensor represents a patch-wise classification matrix, each value indicates the discriminator's prediction of whether the corresponding image patch of the second input image is a real image (closer to 1) or a synthetic image generated by the generator model (closer to 0).

6.3.1 Centralized scenario

The first naive solution when multiple parties wish to collaborate on a shared model is to gather all the data in one central site and train the model on the whole set of images. Clearly, this approach compromises all the data, or at least some part of it, if the central entity is one of the participating parties. However, this method was chosen to be tested in order to analyze the performance and compare it with further,

more secure methods.

The model was trained for 200 epochs, with batch size of 1 and with random horizontal flipping and color jitter, as mentioned in the original paper. Each epoch trains the model over the whole data set of 1096 training images. The training was performed with 0.0002 learning rate, and Adam optimizer parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$. These hyper-parameters will also be used in the further scenarios. The expected results of this training was to obtain qualitative generated images, as the ones presented in the original paper.

The model was trained in Google Colab, running on a T4 GPU with 16 GB of VRAM. The training took 3.8 hours for the 200 epochs. The results generated on evaluation images at epochs 100 and 200 can be see in Figure 6.2. The output images at epoch 200 have more details than the 100 epochs output image. Sometimes, the generator produces chaotic output, such as the third example at epoch 100. These occur less with more training epochs, as the loss for such images is very high. The results obtained match the results in the original paper, so these will be used in the further comparisons.

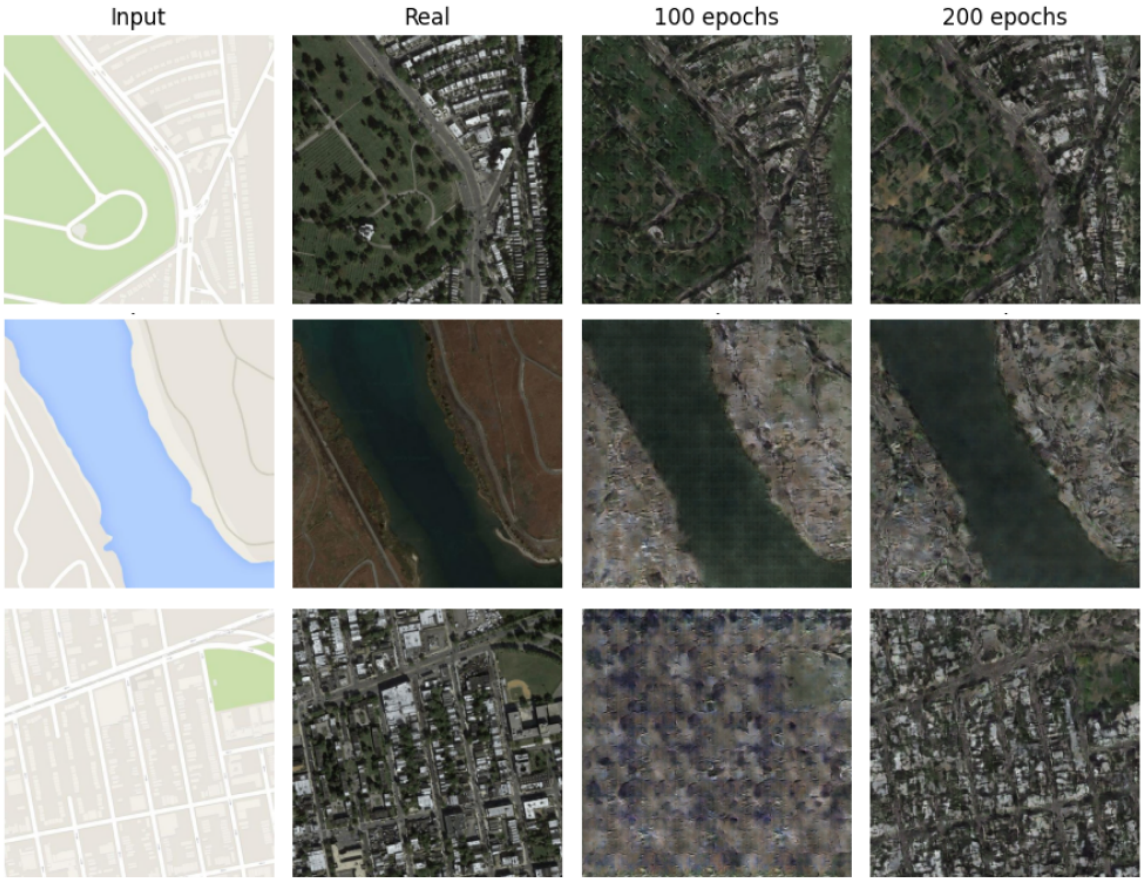


Figure 6.2: Central training evaluation images

The generator and discriminator metrics of this training can be seen in Figures

6.3 and 6.4. These are the average losses per epoch. D_{real} is Binary Cross Entropy between the discriminator result tensor for the real image as input and an all-ones matrix. D_{fake} is Binary Cross Entropy between the resulting tensor when the input is the generated image and a zero matrix. D_{total} is the average of these two losses.

For the generator losses, G_{disc} is the Binary Cross Entropy between the Discriminator result on generated image and all-ones matrix. G_{L1} is the L1 loss (mean absolute error) between the generated image and the real output. G_{total} is the sum of these other two losses.

In time, the total generator loss decreases. However, this is not a performance metric, as will be seen in future examples. As mentioned in the original paper, the G_{L1} loss is multiplied by a factor of $\lambda = 100$, making it a considerable bigger factor in the total generator loss. However, this decreases in time, as the G_{disc} increases, the discriminator getting better at detecting the fake images.

As can be seen in the figures, the discriminator loss has high variance throughout the training process, as the model learns to detect fake images.

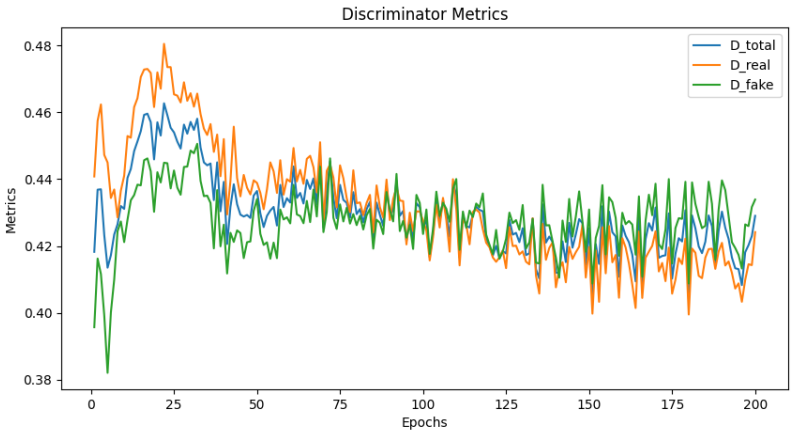


Figure 6.3: Discriminator Loss Values

Local site training

One of the objectives of this work is to determine whether federated learning is worth implementing in a cross-silo scenario. To answer this, a comparison has to be made between the results a participant obtains when training the model with its own data, and the model obtained from federated learning.

Therefore, a training run is required on the data of a single client. The experiment has the same settings as the previous centralized one, only with a third of the data, and it was run using the data of North Europe Site. The results of this run will be shown in the comparison chapter.

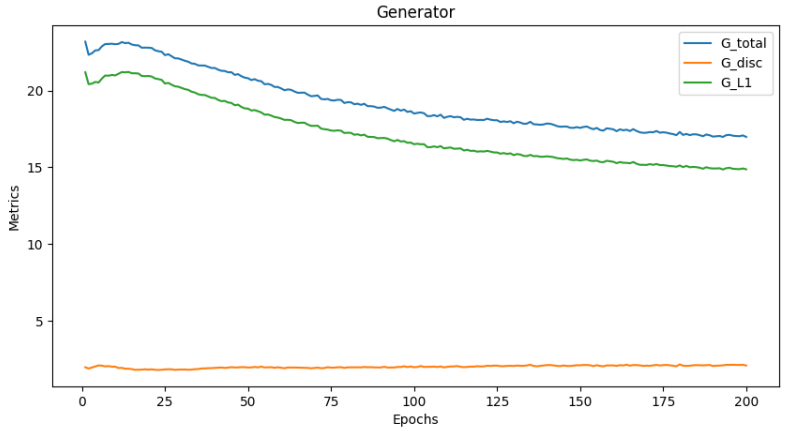


Figure 6.4: Generator Loss Values

6.3.2 Simulated federated training

The centralized scenario provided good results, but the privacy of the data was compromised, as the entities shared their data, making it vulnerable to attacks during data transfer or a potential central site data breach.

In order to collaborate in the model training while keeping the data private, a federated solution is preferred. In this scenario, each client trains the global model locally and then shares the new model to the central server, which gathers the new models from the clients and averages them, obtaining a new model. This is a training round.

The NVFlare framework is used to implement the federated learning scenarios. In the first experiment, the Proof of Concept (POC) mode of the framework is used. This performs federated learning locally, without TLS (Transport Layer Security) certificates. The server and clients run on separate processes. This mode is recommended for simulating a real-world scenario, making it a favorable choice for this experiment.

Four processes are launched simultaneously. A server and three participating clients, each using a third of the data. The training was performed for 20 rounds, each of 10 local epochs, for a total of 200 training epochs over the whole data set. In this implementation, after training locally, clients send the model weights to the server, which will prove to be a difficult case for adding differential privacy solutions in the following experiment.

The training was performed on a Google Colab instance as well, with the same GPU as the centralized scenario. The total training time was 3.3 hours, faster than the previous scenario. There is communication overhead of about 20 seconds between rounds, but the improvements due to parallelism outweigh the communication overhead, making the whole training process faster.

In Figure 6.5, the generator loss for one of the clients can be seen. At each new round, the model the model seems to get worse after the aggregation, but the round-average loss decreases over time.

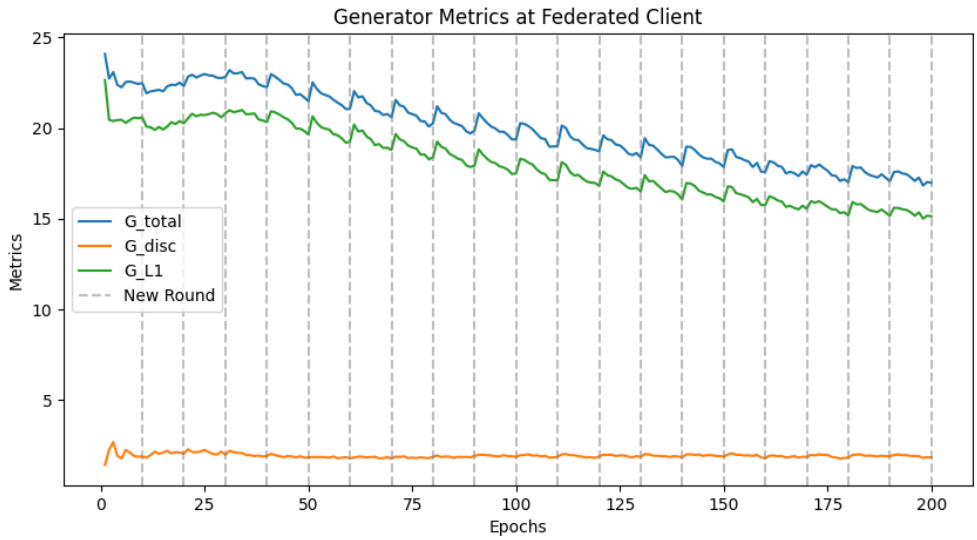


Figure 6.5: Federated Generator Loss Values

Some evaluation images can be seen in Figure 6.6. The model seems to learn to generate higher detail images, but in some cases the model does not learn how to generate some images, as can be seen in the second image.

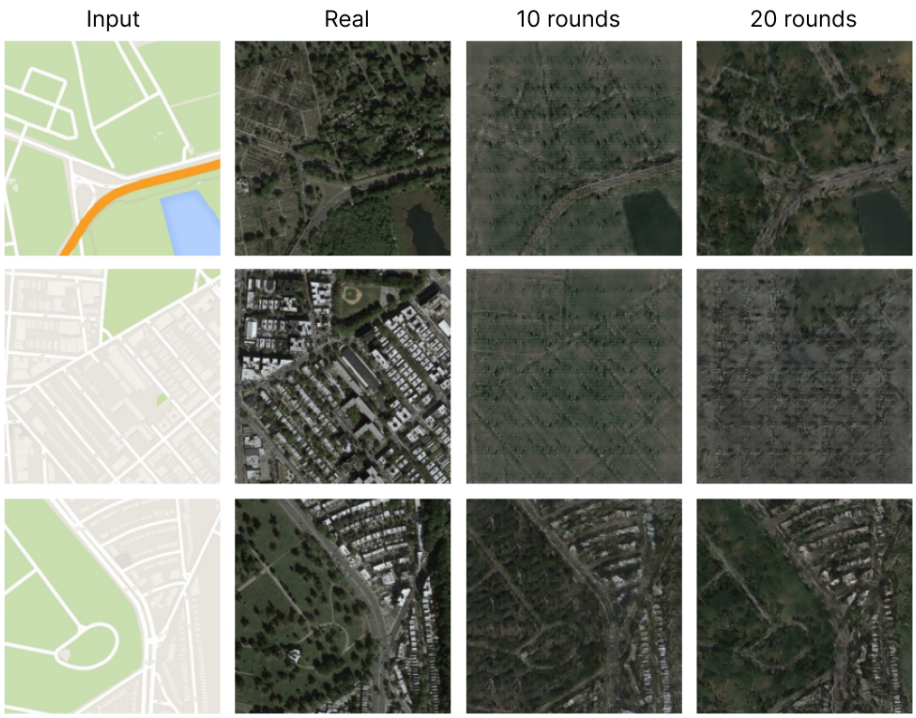


Figure 6.6: Federated training evaluation images

6.3.3 Distributed federated learning with Differential Privacy

In the simulated federated learning environment, the generator successfully learns to generate detailed images resembling the ground truth. Consequently, the following step is to implement an actual distributed scenario. This step is crucial as it permits the analysis of aspects such as communication security and the infrastructure architecture needed for federated learning. Moreover, as discussed in the theoretical chapters, Differential Privacy is needed for increased security. In this implementation, Gaussian noise will be added to the model updates.

The first two scenarios were run on Google Colab. However, running multiple GPU instances in Colab required a premium subscription at the time this work was done. Also, in order to create a scenario closer to reality, another approach was needed.

The solution was to use the Azure Machine Learning platform [Cha15], creating a separate Virtual Machine (VM) for each previous process, the server and the clients. The server was created on a VM without GPU, as the server does not use the GPU for the aggregation operation. The server was created in the North Europe region.

The clients were created in separate Azure ML workspaces, each in a different region. The first client was created in North Europe region, which we will refer to as North Europe Site. The second client was created in the West Europe region,

with the name West Europe Site. Lastly, the third client was created in East US and will be referred to as US Site. This system is displayed in Figure 6.7. Each compute instances is configred to use a NVIDIA Tesla M60 GPU.

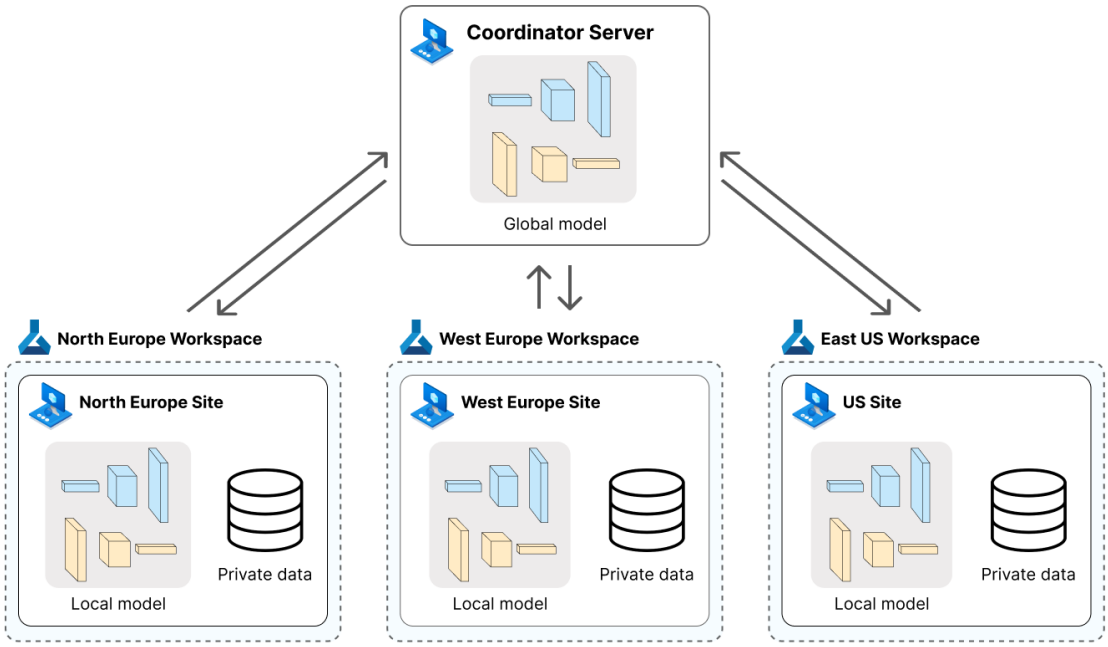


Figure 6.7: System architecture

Server setup

In order to communicate with the clients, the server must open port 8002. Moreover, the server actions are initiated through the admin console, which can be run from a separate machine and use port 8003 of the server, which also has to be opened. For this experiment, the admin console is used from the server VM. The sequence of actions of the training process can be seen in Figure 6.8.

The *project.yml* file contains the provisioning details which the NVFlare framework will use in order to create the startup kits for each users: server, admin and clients. It specifies the server address and the ports used, the role of each participant and other customizable features.

Executing the command `nvflare provision` will create a workspace with five startup kits. One for the server, one for the admin and one for each client. The directories contain TLS credentials, start scripts and other configuration. The `requirements.txt` file should also be added to these startup kits, in order for the clients to install the required packages.

Running the `start.sh` script from the server kit will start the NVFlare server. Running the `fl_admin.sh` script from the admin kit in another terminal window will provide access to the admin console, from which jobs will be triggered.

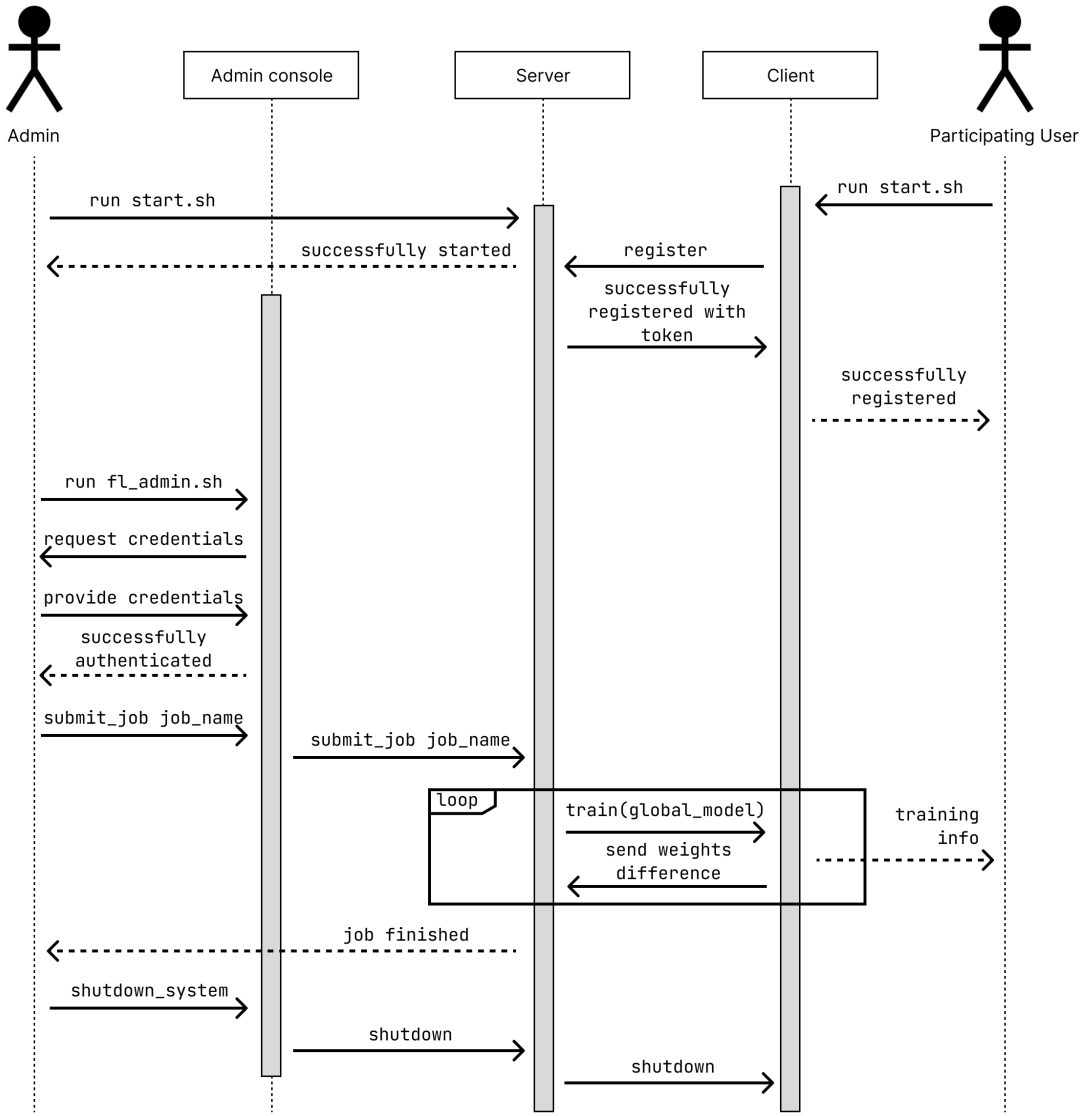


Figure 6.8: Overview sequence diagram for running a training job in the NVFlare framework. The admin starts the server process and the admin console, from which he submits jobs that are further coordinated by the server. Each participant starts a client process which automatically connects to the server and listens for instructions. Finally, the admin can shut down all the participating processes from its console.

Client setup

The startup kit for each client needs to be manually distributed via a secure method, such as email or Secure File Transfer Protocol. After receiving the kits, the clients need to install the required python packages, with

```
> pip install -r requirements.txt
```

executed in the startup kit directory. Each client has their data for training and validation located at path `~/localfiles/data`.

After preparing the client site, the federated client is started using the `start.sh` script from the startup kit. This will automatically connect to the server and will wait for a job to be submitted.

Running the experiment

The admin uploads the job directory to its transfer directory. The server will start the training process when running the command

```
> submit_job job_name
```

The job status is logged in the server window, and the local training status is logged into each client terminal. Each client also sends the generator and discriminator loss to the server as a TensorBoard event. These metrics can be observed during or after training with following command:

```
> tensorboard --bind_all --log_dir=path_to_tb_events
```

The `tb_events` directory can be found in the job directory in the server startup kit (`workspace\job_name\prod_00\server_name\job_id\tb_events`). A screenshot with the TensorBoard interface can be seen in Figure 6.9.

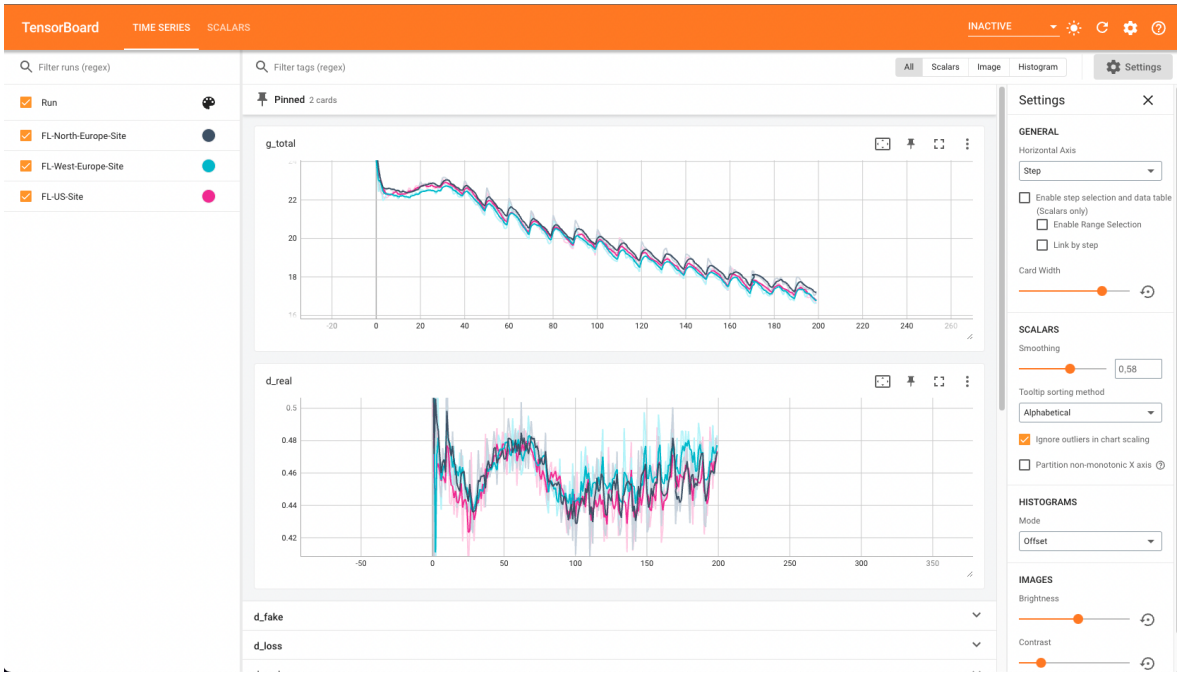


Figure 6.9: TensorBoard interface with model losses

Differential Privacy

In the NVFlare framework, differential privacy can be implemented through Filters, which can be used for both the data received by the participants as well as the data sent as response.

The used Gaussian Noise filter follows the implementation described in [HYM+23b]. It finds the 95th percentile of the data values to be filtered, multiplies this value by the sigma0 parameter, set to 0.1 in the client configuration file and defines the Gaussian distribution with zero mean for the sigma value of this computed value. Samples from this Gaussian distribution are used to add a specific amount of noise to each model update, in the following code line:

```
weights[var_name] += np.random.normal(0.0, noise_sigma, np.
    shape(weights[var_name]))
```

The problem with this approach was that it does not apply noise considering the weight updates, but rather the largest weight of the model, which proved to be too much noise. The result was that the generator loss, shown in Figure 6.10, was spiking too much after aggregation, without a decrease over time. The evaluation images were also not improving after a few rounds.

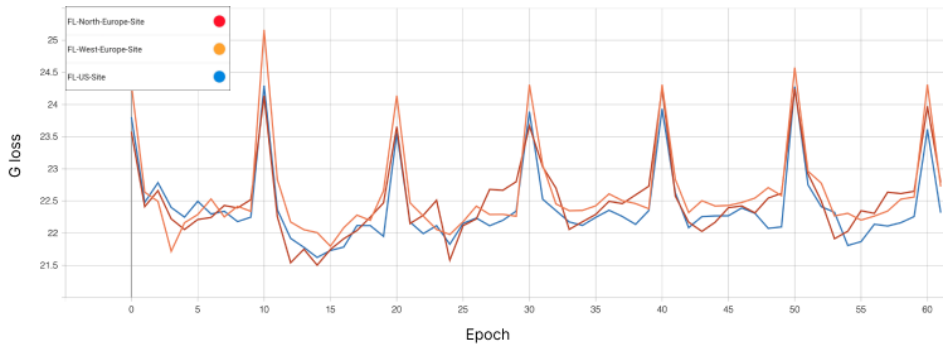


Figure 6.10: Generator loss when adding too much noise

At this point, it is evident that it would be a much better approach to modify the code to send back to the server only the weight differences, compared to the model at the beginning of the round. This way, if a client does an increased amount of changes to the model, more noise is added to the update message.

Results

One additional problem regarding the Differential Privacy filter was detected after a training round. Evaluating the generator at round 5 produced output tensors only with NaN (Not a Number) value. At rounds 10, 15 and the 20, the output image looks as expected. After some analysis, the conclusion was that the Gaussian filter added noise also to the Batch Normalisation variables. One of these is the `running_var` value, to which square root is applied when computing the normalisation. When adding noise, the filter added too much noise to this variable and it resulted in a negative value, producing the seen NaN results. The solution was to only add noise to the weight and bias variables of the models.

The successful training took 3.1 hours to finish. An example of evaluation image can be seen in Figure 6.11. The generated image at after round 10 is more blurry and less detailed in the model with Differential Privacy, but the final result seems to be of similar quality. The differences over the whole data set will be analyzed in the following subsection of this chapter.

One of the extra costs of this method is the communication between clients in different regions. For the US site, receiving the models of total size of 436 MB from the server took approximately 22 seconds and sending the weights difference after training took 17 seconds. This is only double the time that the site in the same region as the coordinator server took for communication, 11 seconds for download and 7 for upload. This proves that Federated Learning with FedAvg algorithm can be used in a real world scenario where the participants might be located in different parts of the globe, due to the reduced communication costs.

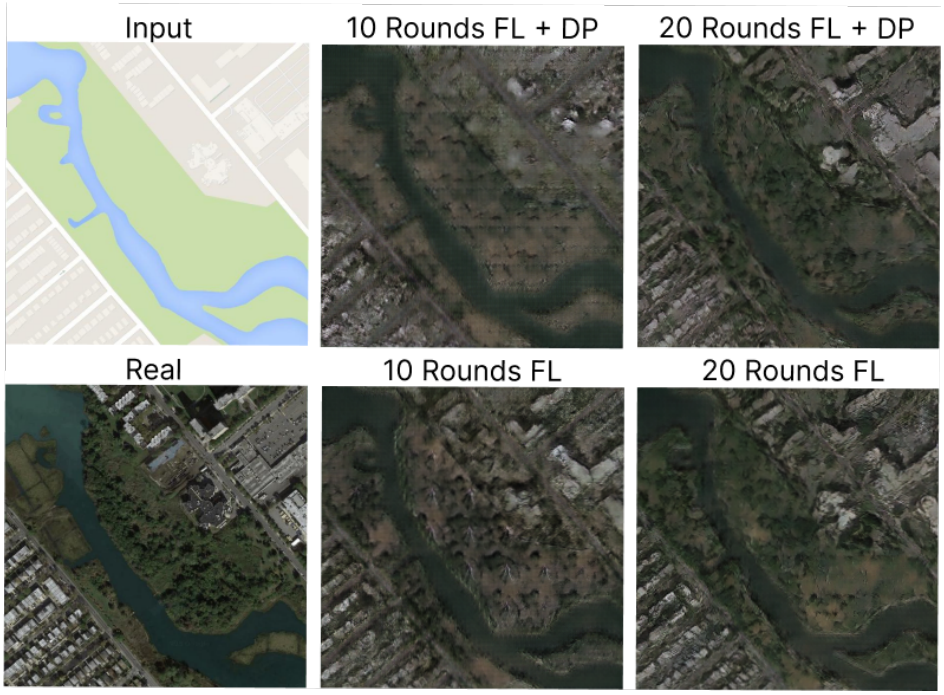


Figure 6.11: Evaluation image during training for both the DP and non-DP models

6.4 Comparing the experiments

Generator loss

As described in Chapter 3.2.2, the generator or discriminator loss are not objective metrics that can be used to analyze the performance of the model. However, it is worth comparing the model loss in the different settings we have described. Figure 6.12 shows the losses of these runs.

The local site training has the lowest loss, but, as will be seen further, it produces the worst results. This can be due to model overfitting, due to the limited data set.

The model loss when adding differential privacy to federated learning closely follows the federated learning loss. This shows that the added noise does not cumber the generator in reaching its subjective goal. However, the generator takes a longer number of epochs to reach the same loss as the centralized model.

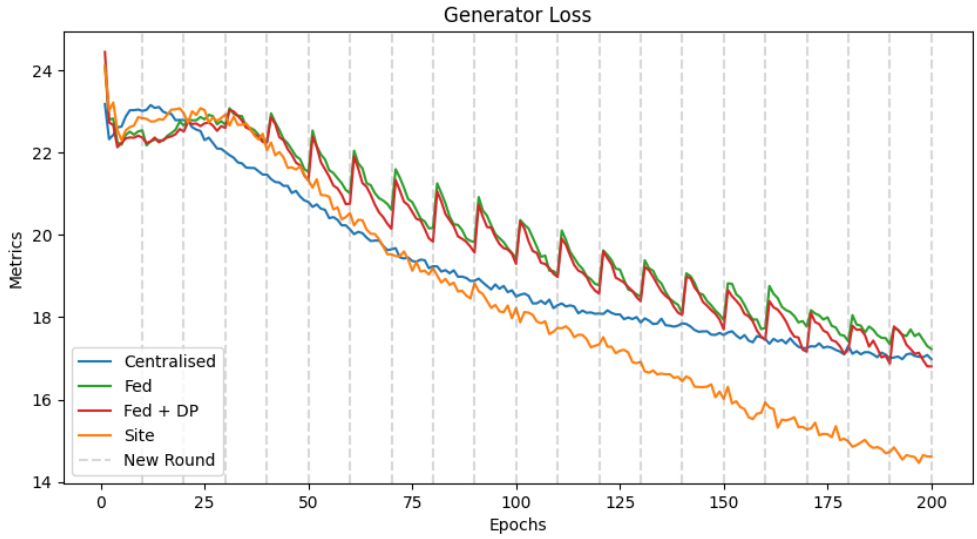


Figure 6.12: Generator Loss Values Comparison

FID scores

As of the chosen metrics for this work is the FID score, the generator state was saved at epochs 50, 100, 150 and 200. In the case of federated learning runs, these correspond to round 5, 10, 15 and 20, respectively. These models were used to generate outputs for each of the evaluation images. These generated collections of images were then used to compute the FID scores seen in 6.13, comparing to the ground truth of the evaluation data.

A problem of the computed FID scores is that models were evaluated over the whole data set, but such metric is not possible in a real world scenario, as the evaluation images are private and stored at the participating sites. It is worth comparing the FID scores only over the images the sites hold. In order to obtain these results, the final model of each training model was used to inference over the whole set of evaluation images of each site, storing the generated images locally at each site. The FID score was then computed for each of these generated sets.

The results are shown in Table 6.1. The FID scores are worse by a value of approximately 10 compared to the previous FID evaluation. However, the proportionality between the different training methods used is the same, compared to the

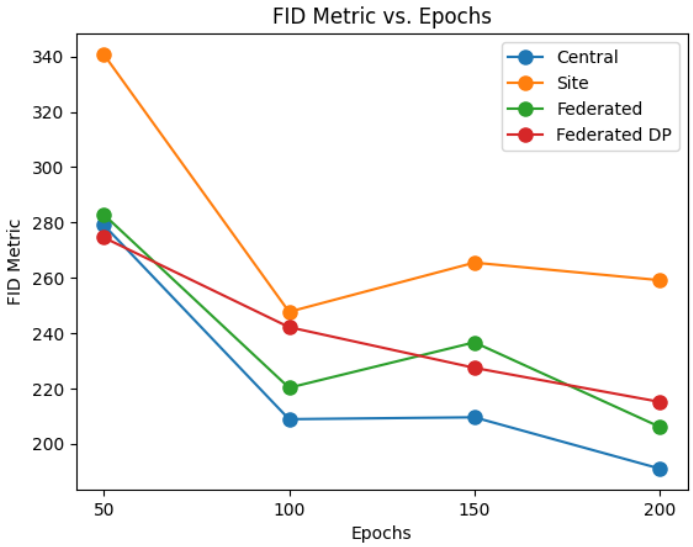


Figure 6.13: FID scores over the whole data set

centralised evaluation, which proves that analyzing the performance of the models locally and then communicating only the obtained FID score is a possible method to decide when the training should be stopped. These FID scores can be displayed in the TensorBoard instance, where the administrator can then conclude whether the training process can be finished.

Site	Centralized	FL + DP	Local
North EU	204.01	229.49	268.49
West EU	202.99	233.44	272.72
US	207.09	230.64	274.81

Table 6.1: FID scores over the local data after 200 epochs

Visual evaluation

The other metric used to analyze the models is visual evaluation, as the goal is to produce images indistinguishable from reality by humans. Some of the generated images at the last model state are seen in Figure 6.14. The locally trained models are the farthest from the ground truth, as the model produces detailed but fuzzy images. The images trained in the Federated Learning setting with Differential Privacy are close in quality to the ones generated by the centralized model, albeit less detailed.

These visual results support the results of the FID scores comparison. Federated training produces better results than training locally, but there is a loss in quality compared to the centralized model.

In Figure 6.15, some failures of the federated model can be seen. In the second example, the model struggles to produce details when little input information is

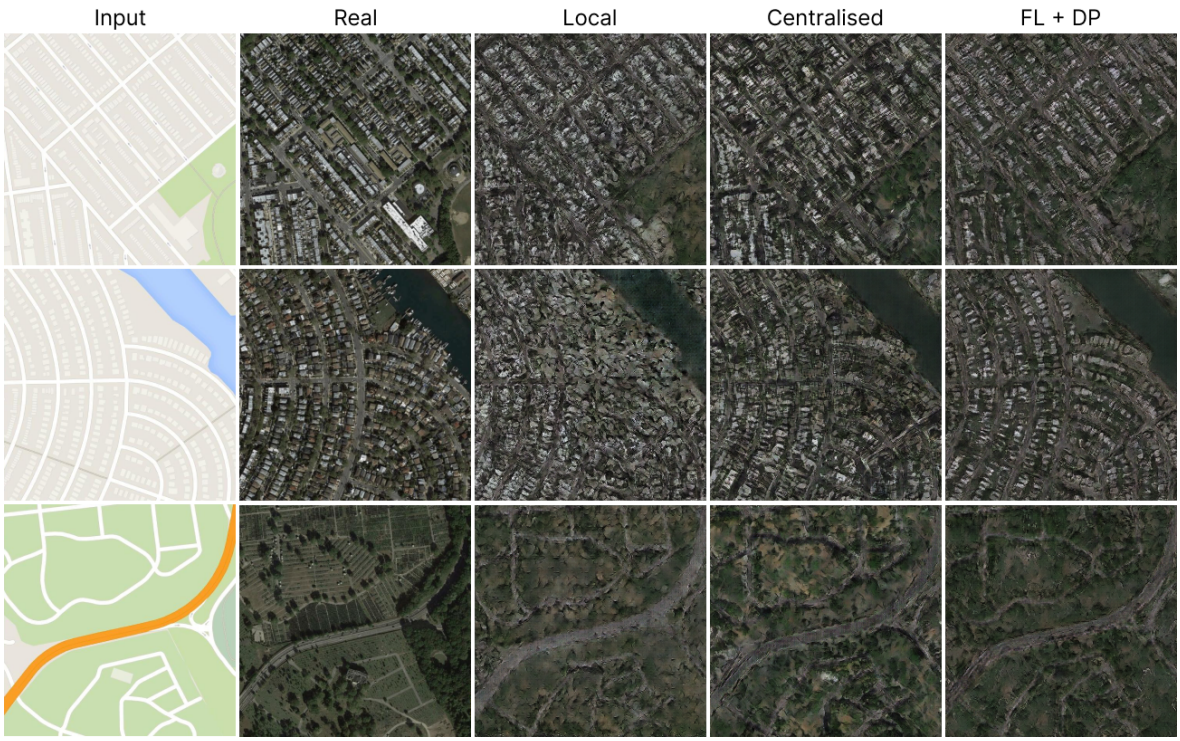


Figure 6.14: Visual comparison of the different training methods

given, even in the centralised model. However, in the first and the last examples, the generator produces images with artifacts. These outputs are sometimes also produced by both the federated model without differential privacy and the central model. In the central model, the generator stops outputting such images after the first half of the training process. In the Federated Learning model, the number of these images decrease as the generator trains more, but in these experiments, 20 rounds of 10 local epochs were not enough to prevent failures such as these.

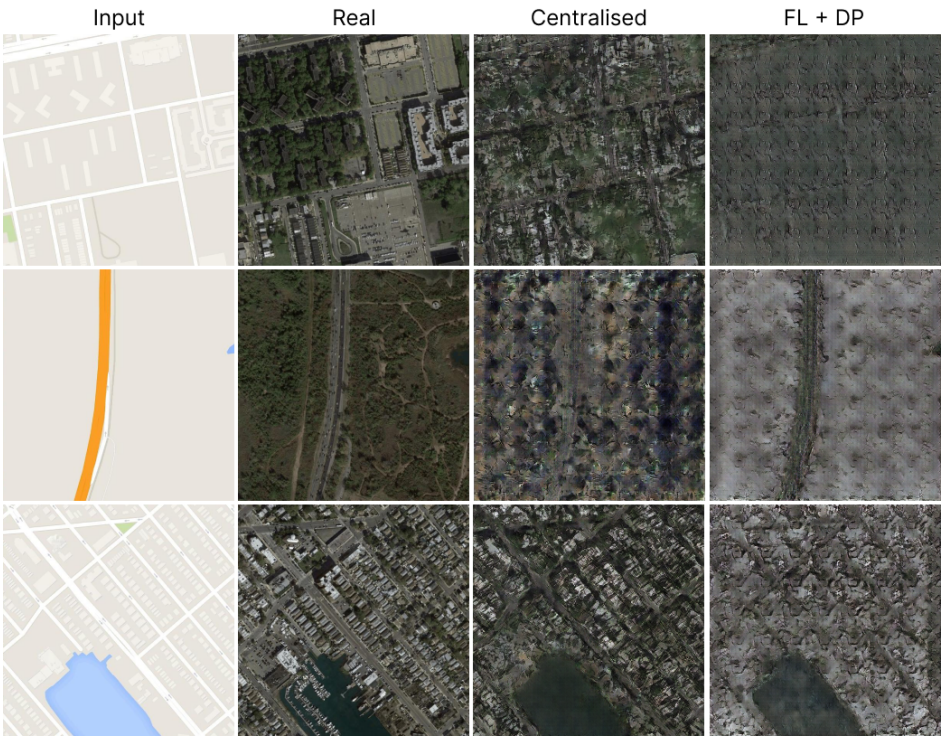


Figure 6.15: Generated images failure examples

Chapter 7

Conclusions

Motivated to integrate privacy-preserving methods into high-performing generative models, in an era characterized by increasing data breaches, the aim of this work is to provide insight into how these models can be efficiently trained while minimizing the exposure of private user information. Moreover, the performance loss of training a large generative model with Federated Learning is an open question in this research field, as most related work trained less complex architectures.

In order to answer these questions, this thesis explored the application of Federated Learning methods to the training of the Pix2Pix model, a state-of-the-art Conditional Generative Adversarial Network, which has been used in many useful applications, especially in medicine, a field in which data privacy is highly important. The chosen framework to train this model is NVIDIA FLARE, an open-source performant FL framework. As no public implementation offered the functionality of training a GAN model in this framework, this marks a significant contribution to the field. The application created for this thesis can be adapted to train any GAN architecture.

The proposed experiments start with training in a centralised scenario, in which the participants aggregate their data in a central site, where the model is trained. This was used as an upper bound performance benchmark and also a lower bound for data privacy, as the data leaves the participants machines. The second scenario analyzed was local training, in which a client trains the model locally, without collaboration, in order to find whether it is justified to propose collaboration for such a model. Lastly, two FL experiments were created. The first approach was to use simulated FL, training locally on multiple processes, in order to test the FL implementation. The next step was to create a distributed secure environment, with the clients and coordinator server as virtual machines in Azure. In this last experiment, Differential Privacy was also added to the FL training process, to enhance data privacy. This last experiment can be easily adjusted to be used in a real scenario.

Through the experiments, it was found that training Pix2Pix with Federated

Learning is indeed feasible and justified. The results show that training the model with FL provides better results than training only with the local data a client holds. The performance of the resulting model is still lower than the centralized approach, but the theoretical and practical results show that the client data is not exposed during this training process. Moreover, the communication overhead observed after the experiment was transferred to cloud proved to be negligible, due to the efficiency of the FedAvg algorithm.

Nonetheless, certain challenges surfaced during this study, such as the intricacies of coordinating multi-party collaborations, especially deciding at which moment the training should be finished, before the model diverges from the objective. Moreover, this study analyzed a scenario with IID data, but further research should also investigate the results of training a conditional GAN in FL with non-IID data distribution among participants, as is the case in many real world applications.

In conclusion, this research bridges the gap between high-performing generative models and privacy-preserving federated learning methods. It opens the door to further collaborative efforts among institutions or researchers, even when sensitive data is involved. This method empowers various entities to contribute to the training of state-of-the-art models while preserving data privacy. The results of this thesis shows that the technology in this area has advanced to the point where it is possibly to create such training scenarios, and provides guidelines on how to develop and utilise such a solution.

Bibliography

[AA22] Abeer Aljohani and Nawaf Alharbe. Generating synthetic images for healthcare with novel deep pix2pix gan. *Electronics*, 11(21), 2022.

[BDF+19] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection against reconstruction and its applications in private federated learning, 2019.

[BEMGS17] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[bre23] Healthcare data breaches by year. `https://www.hipaajournal.com/healthcare-data-breach-statistics`, 2023. Image source; accessed 15-April-2023.

[Bru17] H. Bruderer. The birthplace of artificial intelligence? *Communications of the ACM, BLOG@CACM*, Nov 2017.

[Cha15] David Chappell. Introducing azure machine learning. *A guide for technical professionals, sponsored by microsoft corporation*, 2015.

[cnn] Convolution neural network: Deep learning. `https://developersbreach.com/convolution-neural-network-deep-learning/`. Image source; accessed 10-June-2023.

[con53] Les machines a calculer et la pensee humaine. In *Colloques internationaux du Centre National de la Recherche Scientifique*, number 37, Paris, 1953.

[DCM+12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker,

Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[Dec86] Rina Dechter. Learning while searching in constraint-satisfaction-problems. pages 178–185, 01 1986.

[dee18] Deep learning in digital pathology. `https://www.global-engage.com/life-science/deep-learning-in-digital-pathology/`, 2018. Image source; accessed 10-June-2023.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1322–1333, New York, NY, USA, 2015. Association for Computing Machinery.

[FL20] Chenyou Fan and Ping Liu. Federated generative adversarial learning, 2020.

[Fré57] M. Fréchet. Sur la distance de deux lois de probabilité. *C. R. Acad. Sci. Paris*, 244:689–692, 1957.

[Fuk69] K. Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969.

[Fuk79] K. Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position—neocognitron. *Trans. IECE*, J62-A(10):658–665, 1979. The first deep convolutional neural network architecture, with alternating convolutional layers and downsampling layers.

[Fuk80] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[GSvD+20] Filip Granqvist, Matt Seigel, Rogier van Dalen, Áine Cahill, Stephen Shum, and Matthias Paulik. Improving on-device speaker verification using federated learning with privacy, 2020.

[HRU+18] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[HYM+23a] Ali Hatamizadeh, Hongxu Yin, Pavlo Molchanov, Andriy Myronenko, Wenqi Li, Prerna Dogra, Andrew Feng, Mona G. Flores, Jan Kautz, Daguang Xu, and Holger R. Roth. Do gradient inversion attacks make federated learning unsafe? *IEEE Transactions on Medical Imaging*, 2023.

[HYM+23b] Ali Hatamizadeh, Hongxu Yin, Pavlo Molchanov, Andriy Myronenko, Wenqi Li, Prerna Dogra, Andrew Feng, Mona G. Flores, Jan Kautz, Daguang Xu, and Holger R. Roth. Do gradient inversion attacks make federated learning unsafe? *IEEE Transactions on Medical Imaging*, pages 1–1, 2023.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[ILE65] A.G. Ivakhnenko, V.G. Lapa, and PURDUE UNIV LAFAYETTE IND SCHOOL OF ELECTRICAL ENGINEERING. *Cybernetic Predicting Devices*. JPRS 37, 803. Joint Publications Research Service [available from the Clearinghouse for Federal Scientific and Technical Information], 1965.

[IZZE18] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[KMA+21] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang

Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning, 2021.

[LSZ+20] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2020.

[LTH+17] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2017.

[MHA20] Zhenzhu Meng, Yating Hu, and Christophe Ancey. Using a data driven approach to predict waves generated by gravity driven mass flows. *Water*, 12, 02 2020.

[MMR+23] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023.

[MO14] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[Per] Aladdin Persson. Pix2Pix. `https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/GANs/Pix2Pix`. accessed 5-February-2023.

[pro23] One round in fedavg. `https://nvflare.readthedocs.io/en/2.3.0/programming_guide`, 2023. Image source; accessed 20-May-2023.

[PSM+21] Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, Sudeep Agarwal, Julien Freudiger, Andrew Byde, Abhishek Bhowmick, Gaurav Kapoor, Si Beaumont, Áine Cahill, Dominic Hughes, Omid Javidbakht, Fei Dong, Rehan Rishi, and Stanley Hung. Federated evaluation and tuning for on-device personalization: System design applications, 2021.

[RCW+23] Holger R. Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, Yuan-Ting Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu,

Zhihong Zhang, Wenqi Li, Andriy Myronenko, Dong Yang, Sean Yang, Nicola Rieke, Abood Quraini, Chester Chen, Daguang Xu, Nic Ma, Prerna Dogra, Mona Flores, and Andrew Feng. Nvidia flare: Federated learning from simulation to real-world, 2023.

[RCZ+21] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization, 2021.

[RHM19] Luc Rocher, Julien Hendrickx, and Yves-Alexandre Montjoye. Estimating the success of re-identifications in incomplete datasets using generative models. *Nature Communications*, 10, 07 2019.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[Sch22] Juergen Schmidhuber. Annotated history of modern ai and deep learning, 2022.

[SDL+21] Jingzhang Sun, Yu Du, Chien-Ying Li, wu Hsin, Bang-Hung Yang, and Greta Mok. Pix2pix generative adversarial network for low dose myocardial perfusion spect denoising. *Quantitative Imaging in Medicine and Surgery*, 12, 01 2021.

[Sei20] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. `https://github.com/mseitzer/pytorch-fid`, August 2020. Version 0.3.0.

[SGZ+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[SML+21] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J Anders, and Klaus-Robert Müller. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3):247–278, 2021.

[WKL+22] Yuezhou Wu, Yan Kang, Jiahuan Luo, Yuanqin He, Lixin Fan, Rong Pan, and Qiang Yang. FedCG: Leverage conditional GAN for protecting privacy and maintaining competitive performance in federated learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, jul 2022.

[WLD+19] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farokhi Farhad, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis, 2019.

[XLG+23] An Xu, Wenqi Li, Pengfei Guo, Dong Yang, Holger Roth, Ali Hatamizadeh, Can Zhao, Daguang Xu, Heng Huang, and Ziyue Xu. Closing the generalization gap of cross-silo federated medical image segmentation, 2023.

[ZLH19] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients, 2019.

[ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

[ZWL+20] Jie Zhong, Ying Wang, Jie Li, Xuetong Xue, Simin Liu, Miaomiao Wang, Xinbo Gao, Wang Quan, Jian Yang, and Xianjun Li. Inter-site harmonization based on dual generative adversarial networks for diffusion tensor imaging: Application to neonatal white matter development. *BioMedical Engineering OnLine*, 19, 01 2020.