

ALMA MATER STUDIORUM - UNIVERSITÀ DI
BOLOGNA
CAMPUS DI CESENA

Dipartimento di Informatica - Scienza e Ingegneria
Corso di Laurea in Ingegneria e Scienze Informatiche

Federated Learning for Morphing Attack Detection

Elaborato in Basi di Dati

Relatore

Prof.ssa Annalisa Franco

Co-relatore

Dott. Guido Borghi

Presentata da

Fabio Notaro

Anno Accademico 2022/2023

Abstract

The following thesis has the classic purpose of exposing a problem and evaluate and analyze a possible solution to it.

In this case, the problem is Face Morphing Detection, that is the recognition of alterations and counterfeits of facial images in order to enhance today's biometric security systems.

The solution explored is the Federated Learning, which is a branch of Deep Learning more concerned with privacy and data protection, which could represent one of the few approaches able to respect the stringent regulations in force today regarding the protection of sensitive data.

In particular the thesis, after a detailed explanation of the morphing problem and the potential of Federated Learning, explores the use of the NVFlare framework, a product developed by NVIDIA that enables the training of machine learning models using Federated Learning.

At the end of the thesis, conclusions are drawn and considerations are made on the work done and the results obtained.

"Temet nosce"

Latin locution

Dedicated to my family, for their constant and selfless support in every choice I make and for the love shown to me on a daily basis.

Dedicated to my friends, far and near in time, inside and outside the university, for the moments of sincere joy and indelible leisure they have given me, even and especially in dark and complicated times.

Dedicated to my thesis advisors, Annalisa Franco and Guido Borghi, for having accompanied me with passion, patience and extreme availability in the writing of the thesis, as well as for the invaluable advice and suggestions they donated to me.

Dedicated to myself, for having learned over time to act in the conviction and awareness that moments of difficulty are an integral part of life itself and that it is easier to face them if we are accompanied by the people closest to us, as if we are not alone these are a little less scary.

Contents

1	FACE MORPHING	5
1.1	Definition	5
1.2	A case study	8
1.3	Face morping generation	14
1.3.1	Landmark-based morphing	15
1.3.2	Deep Learning based morphing	17
2	FACE MORPHING DETECTION	20
2.1	Introduction	20
2.2	MAD techniques	22
2.2.1	S-MAD	23
2.2.2	D-MAD	25
2.2.3	Performance study	27
2.2.4	The new approaches	34
2.3	Databases for MAD	37
3	FEDERATED LEARNING	39
3.1	Definition	39
3.2	Scope of application	39
3.3	The workflow	40

3.4	The limitations still present	42
3.5	Split Learning Network	44
3.6	Conclusions	45
4	PRELIMINARY CONCEPTS	48
4.1	Training and testing	48
4.2	Epochs	49
4.3	Cross-site validation	51
4.4	Learning curves	52
4.5	Tensors	55
4.6	Neural networks	56
4.7	Convolutional neural networks	62
4.8	Image classification	66
4.9	Scatter and gather	78
5	NVFLARE	80
5.1	Overview	81
5.2	System architecture	82
5.3	Design principles	83
5.4	Key features	84
6	PROGRAMMING GUIDE	87
6.1	Requirements	89
6.2	Installation	89
6.2.1	The importance of a virtual environment	89
6.3	The FL simulator	93
6.4	POC FL	99
6.4.1	CIFAR10	100
6.4.2	Secure workspace	105

6.5	Real world FL on single host	108
6.6	Distributed real world FL	117
6.7	FL for morphing	121
6.7.1	Client-side FL for morphing	122
6.7.2	Server-side FL for morphing	124
7	EXPERIMENTAL RESULTS ON MORPHING ATTACK	
	DETECTION	131
7.1	Trainings conducted and results obtained	131
7.2	Overall evaluation of the framework	139
8	CONCLUSIONS AND FINAL CONSIDERATIONS	143

Chapter 1

FACE MORPHING

1.1 Definition

Face morphing is a computer graphics technique that consists in combining two or more images of real existing human faces to create an intermediate image (containing facial features of both subjects) that shows an additional, realistic but nonexistent face, which can be seen as a specific frame belonging to a gradual transition from one face to the other of the input faces[1].

This technique is used in both artistic and scientific fields, for example to create special effects in movies or to study human perception of facial expressions, but it can also pose a problem as it can be exploited by malicious people to carry out illegal actions.

However, the most common process of face morphing, which will be explored in more detail in this chapter, consists in identifying key points in the source images, such as eyes, mouth and nose, and create a kind of

map of these.

Then a geometric distortion is applied in combination with a blending operation (averaging of light intensity values) to create an intermediate image showing a smooth transition between the two source images as reported in figure 1.1.

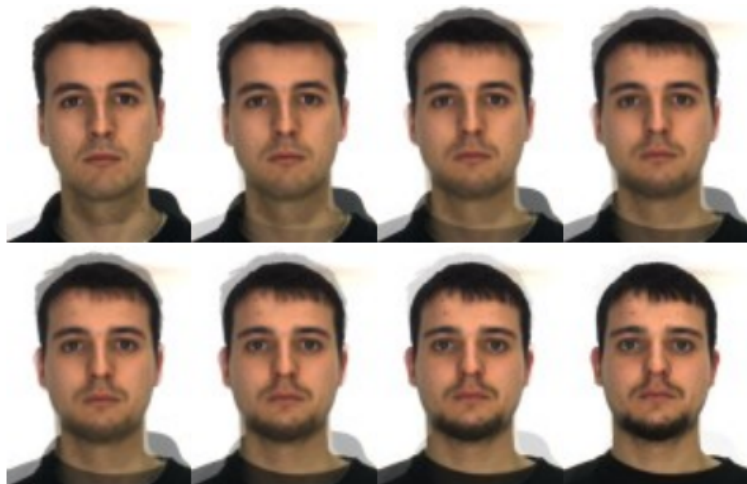


Figure 1.1: transition of frames, gradually fading from the first subject to the second. Source: [1].

In addition to this method, there are also other techniques that allows face morphing and it's important to cite GAN-based methods that don't use landmark points and don't need the blending and warping process. Face morphing can be done manually or by using specialized software, which helps automate the process of identifying key points of the face and generating the intermediate image quickly and accurately[2].

In particular, this report addresses face morphing from its most prob-

lematic point of view, highlighting its potential risks, consequences and possible solutions.

In fact face morphing, by creating hybrid images of two or more people, can be used as an attack and deception tool for biometric security systems that are increasingly beginning to be adopted by entities such as banks or airports[3].

Consider in particular:

- bypassing facial recognition systems through the creation of hybrid images that allow malicious attackers to gain undeserved access to private and sensitive information or places where they could not gain access
- violation of the principle that the link between a document and its rightful owner must be unique
- falsification and identity theft
- privacy threats → given by the fact that face morphing is capable of creating images that represent an individual in a false or misleading way, also causing damage to his reputation
- difficulty in identification and thus loss of effectiveness of biometric security measures.

In order to prevent these security problems, it is important to implement a strong awareness and sensitization campaign about the limitations of state-of-the-art biometric technologies, and thus stimulate and fund research not only related to the creation of more advanced and robust facial recognition systems, but also research related to the study and implemen-

tation of sophisticated algorithms capable of doing face morphing detection (a topic covered extensively in the following chapter).

1.2 A case study

An exemplary case study to further explore the concepts expressed above may be that of a criminal who intends to trick an airport's facial recognition systems to catch a flight despite being prohibited from doing so.

Well, since 2002, the face has been chosen as the anatomical biometric trait for automatic identity confirmation in machine readable electronic travel documents (eMRTDs[4]).

However, to be considered valid, the facial image portrayed in the electronic document must meet stringent requirements set by ISO[5].

Nevertheless, nowadays it is possible to submit the image that will be attached to the electronic document in two ways, depending on the country:

- or by capturing the applicant's face using a high-resolution camera carried out directly in the office issuing the document → rather safe method, which effectively prevents the possibility of any alteration to the image
- or, and it is much more problematic in that it allows face morphing, through a paper printout of a photo of one's face that the citizen delivers to the office that issues the electronic document.

The second case in fact makes it possible to alter the photo by morphing before its delivery[6].

In any case, we point out that this type of attack does not affect the validity of the document: the attack is not about altering its content, but

about deceiving the controller (human or electronic).

It is also worth noting that unintentional or otherwise independent of face morphing but equally problematic alterations may be present, such as embellishment filters or distortions and resizing, which still alter the morphology and geometry of the face.

Indeed, any alteration, whether intentional or not, is problematic because it risks compromising the proper process of identification and automatic identity verification[7].

Several studies[8] in the literature have gone into detail of the effects of image alteration on facial recognition systems, demonstrating that currently popular recognition algorithms work properly only on slight and simple alterations, while proving to be unrobust and dangerously vulnerable to more significant alterations, causing a significant increase in the false rejection rate, that is the number of times in which the system wrongly denies access to those in good faith and therefore in possession of authentic images with respect of the total number of attempts.

In practice, what is analyzed by the studies cited above is the feasibility of an attack on an ABC system[9] carried out using morphed facial images obtained by combining the faces of different subjects.

At the time of verification at ABC, a live captured facial image of the person presenting (for example at the airport) is compared with the face image saved in the eMRTD.

Well, if the digital image contained in the e-passport is the result of a face morphing process, then the same document can be brought back by the people to whom the input faces belong, potentially allowing criminals

and malicious people to take advantage of the passport of an accomplice or unsuspecting victim to get through security checks[10].

This process is shown in figure 1.2.

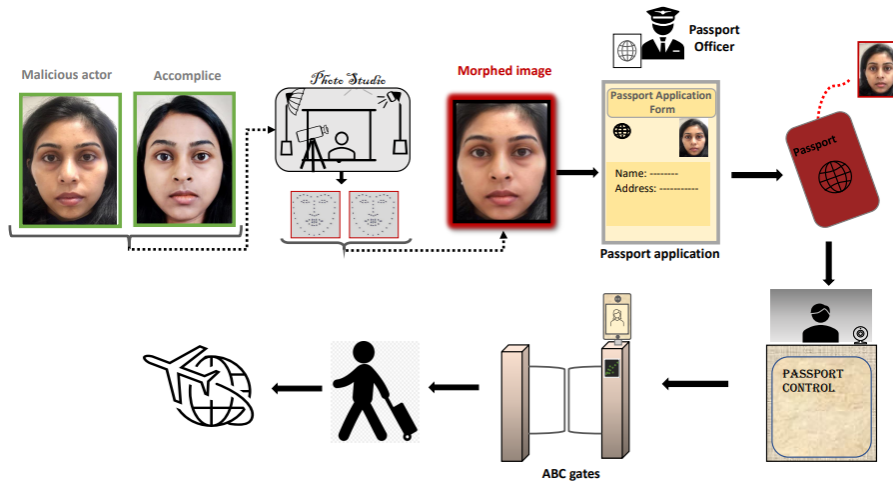


Figure 1.2: illustration of the classic scenario of a possible attack carried out by submitting a morphed photo during electronic document creation. Source: [10].

As anticipated, several studies in the literature have tried to evaluate these scenarios, focusing not only on the feasibility of such deception, but also on the robustness of currently popular control systems in the presence of morphed images[11].

Regarding the attack on ABC systems, an emblematic study is that carried out by the University of Bologna[1], conducted through two common facial recognition software (Neurotechnology VeryLookSDK and Luxand FaceSDK) but confirmed by subsequent studies carried out through other software commonly used in airports.

This study is exceptionally valuable because it places a great deal of emphasis on the realistic reliability of its process, suffice it to say that a preliminary phase of setting the operational threshold of the facial recognition software was conducted in a manner that complies with what is specified by the guidelines provided by FRONTEX[12], the European agency for the management of cooperation at the borders of the member states of the European Union.

In particular the reference maximum threshold value is 0.1% FAR with maximum 5% FRR.

Once this parameter was adjusted, the attack was structured by selecting two images of different subjects having some facial similarity but whose individual images would not cause a false match using the threshold described above[13].

Finally that pair of images was combined into a third image by a morphing process described below.

At this point the third image generated was compared with the input images and the results obtained were studied which, as anticipated, underscore a clear vulnerability of the currently used systems.

The morphing process of the two images is discussed in detail below before commenting on the results obtained.

Keep in mind that morphing can be seen as an animation, a special transition effect that shows one face morphing into another.

In the reported study in particular, morphing was performed on the images using GIMP and GAP (GIMP Animation Package) with the aim of producing an artificial image that is very similar to one of the two subjects but has particular facial features taken from the other subject, so that it is possible to confuse and swap the two faces.

It is good to specify that the goal of creating a realistic third face is easily achievable even if the two input faces are quite different from each other[14].

Taking two images in particular, the morphing process used in the first study mentioned above involves the following steps using the two software programs above:

- the two images are overlaid manually according to the position of the eyes but on two different layers
- a set of relevant facial points such as eyes, cheekbones, eyebrows, nose, chin and forehead are marked in the two images
- a long sequence of frames showing the transition from one face to another is automatically generated through a special function of the GAP software
- one of the resulting frames is carefully selected, scanning the frames starting from the image of the accomplice and advancing until the examined frame also matches with a test photo of the criminal's face
- finally, it is possible to manually retouch the resulting frame to make it more realistic, such as removing visible artifacts introduced by the morphing process.

In figure 1.3 is shown the final step of morphing process.

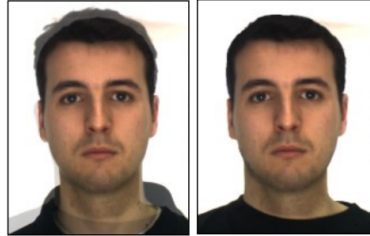


Figure 1.3: representation of the selected frame before and after manual retouching. Source: [1].

The above experiment was performed on 5 pairs of men, 5 pairs of women and two extra cases: a man-woman mix and a mix between three men. The test showed that all of these trials resulted in a successfully executed attack, that is a deception of the facial recognition software, which was found to be unable to detect morphing and thus recognize the counterfeit image as belonging to both individuals involved.

The study's conclusions focus on a few main aspects:

- first of all these critical issues are traced to the possibility given to the citizen to submit, when applying for the eMRTD, a printed photo produced prior to the application and therefore potentially subject to alteration → the study therefore suggests eliminating this possibility by maintaining as the only method of face presentation its acquisition in the office by an attendant alone;
- moreover, the study underscores that the quality of morphing can be so high as to fool even an experienced human examiner;

- finally, it could be shown that the blending operation[15] is responsible for producing, especially in some areas near landmark points, noticeable artifacts that can lower the quality of morphing.

1.3 Face morphing generation

It is necessary to point out that actually the one given in the previous section is only one among many possible ways to generate a morphed image, called Landmark-based morph generation[16].

In addition to this there are also other more advanced techniques that take advantage of Deep Learning[17], specifically neural networks called Generative Adversarial Networks[18].

All the face morphing generation methods are depicted in figure 1.4.

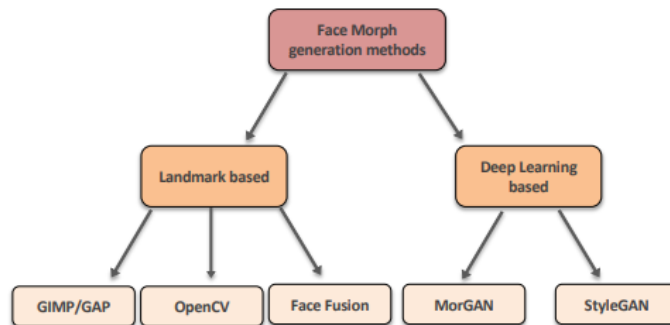


Figure 1.4: taxonomy and subdivision of major face morphing generation techniques [10].

1.3.1 Landmark-based morphing

Landmark-based morphing algorithms, as mentioned earlier, rely on the detection of facial points of interest, such as the nose, eyes and mouth, which are used to align and combine the faces of the two contributing subjects.

There are now several software programs that can perform automatic detection of the specific somatic features required for morphing the two images (such as Dlib[69]), and once this is done, these points are overlaid and aligned in an intermediate position.

There are various techniques for doing this procedure, among which it is good to mention Free Form Deformation[19] and the Deformation by moving least squares[20].

It should be specified that the accuracy of the previously described procedure of identifying and overlaying facial points of interest has a direct impact on the quality and credibility of the generated morphed images.

It is also specified that manual detection currently provides more accurate points than automatic localization, which in fact focuses only on the central region of the face, ignoring the forehead and cheekbones.

Once the relevant points of the two faces have been identified, the morphing process can be described by the following mathematical formulas[21]. Defined I_0 and I_1 as the two images that are intended to be combined and P_0 and P_1 as the corresponding two sets of points, each frame is a weighted linear combination of I_0 and I_1 , formally:

$$I_\alpha(p) = (1 - \alpha) \cdot I_0(\omega_{P_\alpha \rightarrow P_0}(p)) + \alpha \cdot I_1(\omega_{P_\alpha \rightarrow P_1}(p))$$

where

- p is a generic pixel position

- α is the frame weight factor, that is the contribution of the image I_1 in morphing (for example $\alpha = 0.4$ means that the morphed image will be obtained for 40% from I_1 and for 60% from I_0)
- P_α is the set of corresponding points aligned according to the parameter α
- $\omega_{P_\alpha \rightarrow P_\beta}(p)$ is a warp function[22].

As mentioned, as a result of this process some blocks of pixels are replaced, leading to the misalignment of other pixels that contributes to the generation of some inconsistent and unrealistic features, which risk compromising the quality of the morphing process and thus its credibility[23]. In particular, the problems that arise are mainly two:

- macroscopic ghost effect in the area surrounding the face \rightarrow caused by the fact that landmarks are placed in the central region of the face, while landmarks are not considered for hair or ears[24]
- minor counterfeit marks (for example double edges or unnatural iris reflections) present in facial features for which insufficient or inaccurate landmark points were used.

These two defects need to be addressed by adjustment operations such as image smoothing, image sharpening, edge correction, manual retouching, background substitution and skin color equalization.

As mentioned, many open source libraries and software including GIMP and OpenCV take advantage of landmark-based morphing.

1.3.2 Deep Learning based morphing

The second type of technology that can generate morphed images is called Deep Learning Based.

These techniques use special neural networks called Generative Adversarial Networks (GANs), which can synthesize morphed images generated from sampling the two input facial images in the latent space of the network itself (that is in a compressed and summarized representation of it). Generally this type of generation produces morphing with some problems, especially low similarity with the two starting subjects, as it is difficult to preserve their identity.

They are still used because they do not exhibit the artifacts of landmark-based methods, although artifacts typical of GANs remain[25].

In essence, although morphing via GAN does not require manual corrective interventions subsequent to image generation or even preliminary steps of landmark-points search and alignment, landmark-based methods remain the best in terms of the quality of the face produced.

GANs are based on the action of two different agents: a generator and a discriminator.

The former is concerned with producing distribution samples that are as indistinguishable from the training distribution as possible, while the latter trains to determine whether the samples submitted to it come from the real set of training images or the artificially generated samples from the generator.

The training process of these neural networks gradually improves the quality of the samples produced by the generator, which thus perfects its ability to deceive the discriminator.

It should be noted, however, that this basic idea has been progressively refined and improved over time, introducing other actors (for example encoders) as well, in order to achieve better and better performance.

Note in the table 1.1 below a summary of the advantages and disadvantages pertinent to the two morphed image generation techniques seen above:

TECHNIQUE	ADVANTAGES	DISADVANTAGES
LANDMARK BASED	Availability of free tools, high quality of the generated image, ability to easily deceive the FRSs today widespread, ease of usage from automatic procedures[26] and similarity between subjects increases the chances of success[27]	Needs of manual interventions for artefact removal and needs of posthumous actions
DEEP LEARNING BASED	Manual interventions not required, greater simplicity and similarity between subjects increases the chances of success	Needs of complex training, possibility of geometric undesirable distorsions, it needs attention when choosing subject for what concerns age, gender and ethnicity and low similarity with contributing subjects

Table 1.1: advantages and disadvantages pertinent to the morphed image generation techniques cited.

Chapter 2

FACE MORPHING DETECTION

2.1 Introduction

Face morphing detection[28] (or MAD, Morphing Attack Detection) is the process that detects whether a face image has been generated using morphing techniques.

These techniques are important for verifying the authenticity of images and fight the use of false images for illicit purposes.

As mentioned, the goal of a morphing attack is to circumvent the facial recognition system (FRS) in an ABC (Automatic Border Control) gate by presenting an eMRTD or electronic document whose image was obtained through a face morphing process.

In this way a criminal could, with the help of an accomplice, unduly pass such controls and gain access to areas prohibited to him.

Not only border controls, this kind of deception can increase the vulnerability of so many environments: health care, banking and even law.

In fact, it should be noted that today the presence of free software that makes it easy and accessible to anyone (even non-experts) to create these types of counterfeit images exponentially increases exposure to facial recognition system deception.

A further factor contributing to the increase in the prevalence of this type of scam concerns the possibility of autonomous renewal of documents: more and more nations, including the United States, New Zealand and Ireland, are in fact offering the possibility of renewing a passport without physically going to any office, but simply uploading to a web portal an updated facial photo, which therefore may have been previously counterfeited through face morphing.

Given all these vulnerabilities, several studies have been published in recent years[29][30][31], both academic and industrial, some of which are also funded by government institutions[32] (such as the European Union), which aim to research, describe and illustrate ever-better techniques and algorithms for performing the so-called MAD (Morphing Attack Detection).

Even the University of Bologna has contributed actively in this area of research by submitting the SOTAMD project, which with its platform is now a benchmark for the evaluation of MAD techniques through a dataset containing nearly 6,000 high-quality images to perform comparative analyses under realistic conditions.

2.2 MAD techniques

As anticipated in the previous sections, various tests[33] have shown that a human examiner, whether expert or not, dramatically fails to detect whether morphing has been applied to an image.

Automatic approaches are therefore needed to solve the problem of MAD[34].

The MAD techniques available today can be classified into two sets based on the number of images provided as input:

- **Single-image based MAD**[35] (S-MAD) → the presence of morphing alterations is detected using only the single suspected image
- **Differential image based MAD** (D-MAD) → receives as input a pair of images, one of which can be viewed as the image on the passport and the other as a trusted photo captured live.

All the MAD techniques are shown in figure 2.1.

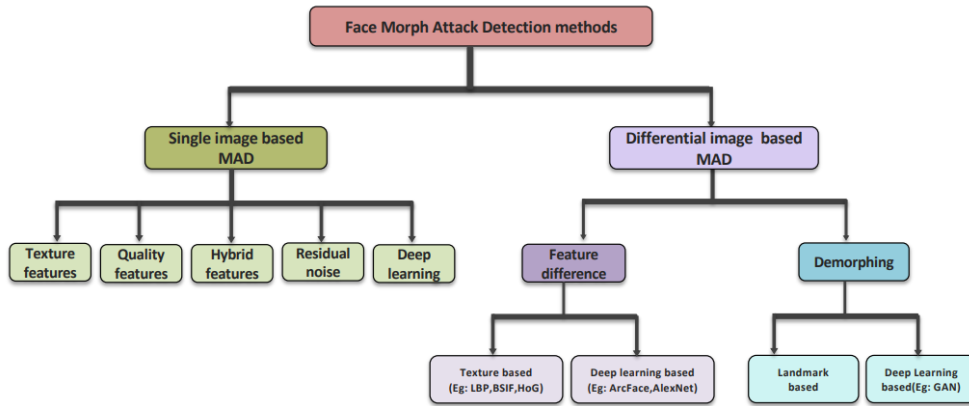


Figure 2.1: taxonomy and subdivision of MAD techniques. Source: [10].

Whatever MAD technique is used, it is worth noting that facial images may come from different sources and therefore may also have very different characteristics, such as size and resolution.

Therefore, it is important to preliminarily normalize the photos before subjecting them to the detection process.

2.2.1 S-MAD

S-MAD techniques are the one applied when one needs to perform detection based solely on the image submitted to the algorithm, not on other hypothetical images with which to perform analysis and comparisons.

This is for example the practical case of verifying the image during enrollment.

Figure 2.2 briefs the S-MAD process.

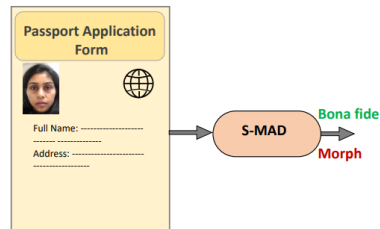


Figure 2.2: note that in this case the detection of the attack must be done using a single image. Source: [10].

In turn, it is possible to distinguish these techniques into 5 additional subtypes based on the functional characteristics employed:

- **Texture Features Based S-MAD**[36] → based on extracting and analyzing surface layer texture characteristics of the image, that is using image processing algorithms to identify unique properties of textures such as gradient distribution, variance and correlation between neighboring pixels to identify any defects or inconsistencies → this method is all in all accurate and fast and is based on the fact that some studies[37] have shown that morphed images are characterized by a different texture than unaltered images and that JPG compression emphasizes this aspect
- **Quality Based S-MAD**[38] → uses multiple aesthetic quality parameters of the acquired images to identify any anomalies or inconsistencies, for example measurement of brightness and sharpness or angle distortion → these evaluations are then aggregated and processed to determine and estimate the degradation of the image and

if it is greater than a certain threshold the algorithm assumes it has been manipulated → this method is less precise than others but is often faster and easier to implement

- **Residual Noise Based S-MAD**[39] → analyzes pixel discontinuity through a process of subtraction between the given image and the version cleaned of noise (for example imperfections and distortions present in a digital image such as chromatic aberrations or ghosting) to identify any anomalies
- **Deep Learning Based S-MAD**[40] → as the name implies, this method uses neural networks previously trained through a large dataset to autonomously identify any anomalies in the image
- **Hybrid S-MAD**[41] → as the name suggests, combines two or more morphing detection methods to improve the accuracy and overall robustness of the system → unfortunately, this approach also involves a significant increase in cost and computational complexity.

S-MAD techniques are generally considered to be the most complex compared to D-MAD techniques, since having a second comparison image is a great help in correctly solving the problem.

Since there is no second image available, S-MAD techniques base their decision on the principle that morphing procedures leave marks and traces on the counterfeit image that, although slight and imperceptible, help spotting the attack[42].

2.2.2 D-MAD

The goal of Differential Image Based MAD is to verify whether the analyzed image has been morphed by comparing it with another photo of the

same subject acquired in a safe environment.

These techniques are therefore particularly well suited to the classic border control scenario described above, in which precisely the image on the passport potentially subject to morphing is compared with an image acquired live by the ABC system.

Figure 2.3 briefs the D-MAD process.

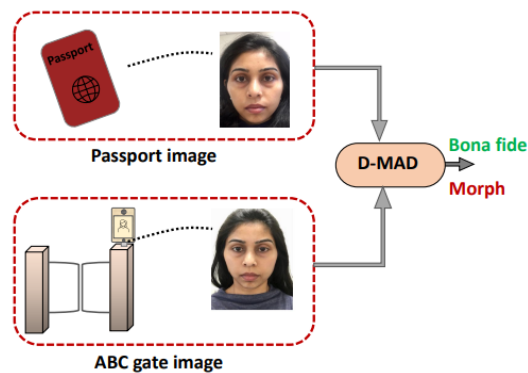


Figure 2.3: in this other case, however, you have two photos of the subject, which are therefore compared to detect the possible attack. Source: [10].

This approach can also be divided basically into two techniques:

- **Feature Difference Based D-MAD**[43] → uses the difference between the characteristics extracted from the two representations of the subject (the one presented and the one acquired) to identify any anomalies and inconsistencies → examples of algorithms used to extract features may be the study of landmark points, spectral analysis or the study of textures → image points with a high difference are identified as defects and thus possible morphing signals

- **Demorphing**[44] → involves reversing the morphing process to reveal and study the basic images initially provided as input → this inversion can be done either by landmark points or by deep learning.

2.2.3 Performance study

Several studies analyzing the behavior and performance of the most popular MAD techniques can be found in the literature.

Well, such studies focus on identifying and evaluating the following parameters:

- vulnerability of the facial recognition system
- MAD performance.

Regarding the first point, that is the vulnerability assessment of the FRS, this is intended to measure whether a set of morphed images is capable of deceiving a face recognition system (FRS).

There are mainly two mathematical metrics in the literature that are usually considered and that accurately describe this parameter:

- **Mated Morph Presentation Match Rate** → defines the proportion between the morphed images identified and the morphed images contributed by the formula

$$MMPMR = \frac{1}{M} \cdot \sum_{m=1}^M [\min(1..N_m) \cdot S_m^n] > \tau$$

Where M is the number of morphed images, N_m is the total number of subjects that contributed to morphing, S_m^n is the degree of compatibility of the n-th subject in morph number m and finally τ

is the threshold of the FSR \rightarrow in practice, this mathematical formula is based on the fact that a morphing attack is successful if all of the contributors are recognized in the morphed image, and thus this metric assesses the degree of vulnerability of an FRS by making an authentication attempt for each contributor to produce the morphed image and verifying that this attempt exceeds the predetermined minimum authentication threshold

- instead the **Fully Mated Morph Presentation Match Rate** can be seen as an evolution of the previous metric \rightarrow always concerns the vulnerability of an FRS but also takes into account the weight and number of successful attempts made using test images of the subjects that contributed to the morphed image \rightarrow

$$FMMPMR = \frac{1}{P} \cdot \sum_{M,P} (S1_M^P > \tau) AND \dots AND (Sk_M^P > \tau)$$

where P is the number of attempts made comparing all test images of subjects with the M-th morphed image, while $K=1..k$ is the number of subjects who contributed to the creation of the morphed image, Sk_M^P is the degree of compatibility of the k-th subject in the p-th attempt corresponding to the M-th morphed image and finally also here τ is the threshold of the FSR \rightarrow the fact that the number of attempts evaluated for each morphed image is also taken into account results in greater accuracy and reliability.

As for the evaluation of the second significant aspect, MAD performance, however, these focus on measuring the robustness of MAD algorithms through metrics established by ISO[5].

Since MAD can be viewed as a binary classification problem, the two main metrics for evaluating performance are:

- **Attack Presentation Classification Match Rate** → calculates the percentage of examples of morphing attacks misclassified as authentic facial images (false negative, a forged image is mistakenly considered valid)
- **Bona Fide Presentation Classification Match Rate** → calculates the percentage of authentic facial images misclassified as morphing attacks (false positive, an image that is instead authentic is mistakenly believed to be forged).

From an algebraic point of view:

$$BP CER(\tau) = \frac{1}{N} \cdot \sum_{i=1}^N H(b_i - \tau)$$

$$AP CER(\tau) = 1 - \left[\frac{1}{M} \cdot \sum_{i=1}^M H(m_i - \tau) \right]$$

where:

- N is the number of training datasets
- M is the number of models trained
- b_i and m_i are the detection scores
- τ is the score threshold at which are compared b_i and m_i
- $H(x)$ is a generic step function.

There is also an additional metric that encapsulates the two parameters described above to assess the quality of morphing recognition systems, namely the Detection Equal-Error-Rate (EER), which corresponds to the error rate where BPCER and APCER coincide.

To detect and visualize EER, the Detection Error Trade-off (DET) curve is usually used, which shows the change in BPCER values as a function of APCER values (that is the change in their ratio).

The EER is identified by intersecting the DET curve with the bisector of the plane (also called the identity line).

The EER representation is shown in figure 2.4.

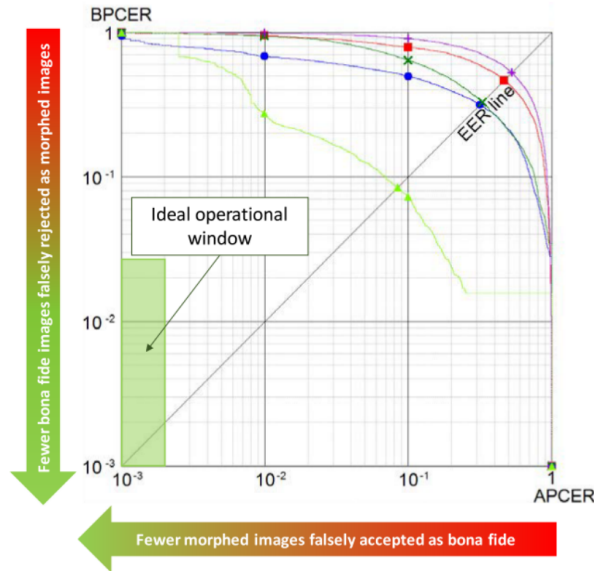


Figure 2.4: EER representation. As easy to guess, a low EER indicates better morphing attack recognition capability. Note also the distance between the points and the admissible region, that is the one that meets the operational thresholds required by FRONTEX.

Finally, note that in addition to EER, other points of interest are often highlighted on the DET curve, such as:

- $BPCER_{0.1}$ → BPCER value when the APCER is equal to 10%, that is is the percentage of genuine images that are misclassified as manipulated when APCER is 10%
- $BPCER_{0.05}$ → BPCER value when the APCER is equal to 5%, that is is the percentage of genuine images that are misclassified as

manipulated when APCER is 5%

- $BPCER_{0.01}$ → BPCER value when the APCER is equal to 1%, that is the percentage of genuine images that are misclassified as manipulated when APCER is 1%.

It is worth noting that these particular metrics are useful to evaluate because FRONTEX has established minimum acceptable thresholds concerning them that must be met for a system performing morphing attack detection to be deemed sufficiently reliable.

Also for this topic, the technical and scientific literature has conducted and published several studies and experiments[2][3][4], evaluating the above metrics also through the databases and datasets produced.

The most interesting results can be summarized in the list below:

- gender weighs heavily on the chance of deception → the chance of cheating recognition software appears to be 10% higher whether the subject portrayed is a woman
- the accuracy and quality of manual corrective retouching is a determining parameter → mild or poor interventions can lead to attack failure, while correct and sophisticated interventions increase the chance of a successful attack
- the behavior of the recognition software changes depending on which morphing algorithm was used to produce the image
- as expected, the quality of morphing directly and still too heavily impacts S-MAD and D-MAD techniques

- is generally much more problematic with S-MAD than with D-MAD, because of the less information in preliminary possession that is useful for correctly estimating the authenticity of the image
- none of the algorithms today have performance compatible with the operational requirements demanded by FRONTEX → this confirms that the problem of face morphing attack detection remains challenging to solve and absolutely urgent[45].

The studies present today also agree about the identification of outstanding challenges and environmental and operational conditions that hinder the search for optimal algorithms for MAD resolution, such as:

- the absence of large-scale public datasets, due to GDPR and regulations on privacy protection of portrayed subjects
- confusing, contradictory and nonstandardized criteria about the selection of subjects for morphing
- problems related to the recognition of twins, look-alikes and people with similar facial features
- one of the most complex challenges involves the fact that photos, which are natively digital, are first printed by the photographer and then scanned to be attached to the electronic document → MAD is therefore done on the image after undergoing this process known as printing/scanning (P&S), which increases the level of difficulty in evaluation because it removes most of the small digital details that would be useful in detecting morphing.

It is good to elaborate on this last point since, in order to achieve high accuracy, neural networks used in Deep Learning based MAD, as mentioned

extensively in this thesis, need a large training dataset.

But unfortunately, it is difficult to collect adequate databases of samples due to the fact that manual production of morphed images at high quality is a long and tedious process.

An interesting proposed solution to this problem is a study[29] that investigates the possibility of artificially and automatically generating large sets of morphed images even simulating their submission to the process P&S, which would therefore be very useful in the training process of neural networks.

The results of that study show that the use of imaging P&S artificially simulated significantly improves MAD performance.

2.2.4 The new approaches

Recent studies[46] are focusing on overcoming the obstacles and restrictions in current approaches to MAD, such as training data that are limited in quantity and variety or privacy issues that prevent the exchange and sharing of data.

Some studies, in particular, are exploring the possibility of using incremental training for MAD on data also held by different research groups, a learning strategy that would no longer involve sharing data but rather transferring the entire model.

The specific problem that this method attempts to overcome is that privacy regulations in place today impose stringent practical limitations about the collection, dissemination and sharing of biometric data (particularly facial images).

In such a scenario, the search for new and more effective algorithms for

MAD is greatly slowed by the fact that each laboratory is forced to develop, train, evaluate and test its own MAD model on its own data.

In particular, these limitations have a strong impact on the reproducibility and generalization ability of the model, which in fact sees its performance drop dramatically when subjected to previously unseen data.

There are now two specific types of Deep Learning that could overcome these limitations:

- **Continual Learning**[47] → a learning paradigm described in this section
- **Federated Learning**[48] → the approach explored and studied in this thesis.

Continual Learning focuses on the ability of models to learn new knowledge continuously without forgetting previously acquired knowledge[49]. This is a major problem with regard to artificial intelligence because many models tend to forget previous information when trained with new data. This technique, which is still being studied in terms of its application to the MAD problem, involves showing, during model training, each dataset once, so that incremental learning is simpler and can be extended to the case of new data encountered for the first time, without the need for time-consuming review procedures or storing all training data in the same place.

This peculiarity helps to counteract the so-called catastrophic forgetting[50], that is the phenomenon that occurs in artificial intelligence models when they forget previously acquired knowledge during training on new data, caused by the fact that most models are optimized to fit new training data by overlapping or deleting previously acquired information.

Luckily, the Continual Learning paradigm assumes that not all training data are always available at all times, effectively leading to the creation of agents adept at continuously learning from newly acquired data rather than accessing past data.

The classic workflow of this approach is summarized in figure 2.5 below:

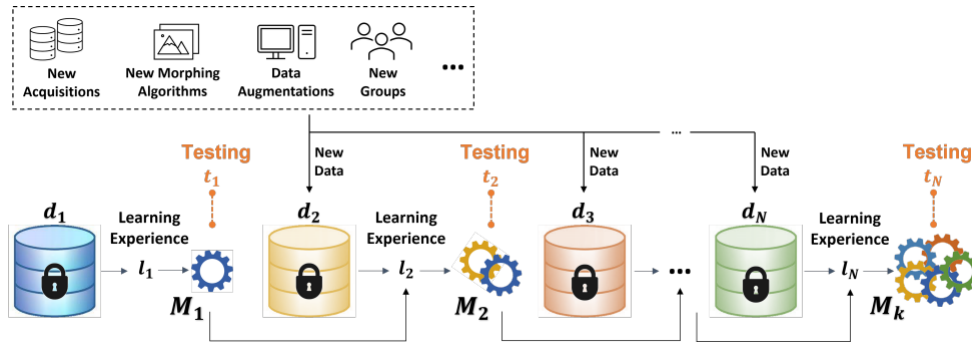


Figure 2.5: note that the model is trained incrementally on different datasets ($d_1..d_N$) and evaluated at many validation stages ($t_1..t_N$). The training procedure can be continued at will and without the need to re-train the model from the beginning or transfer past model data. Source: [46].

This particular development allows a research group to train its model internally on its own data and then share it with other researchers so that they can continue to train it on their own datasets, thus without any data transfer.

This way of training could foster the emergence of new MAD algorithms or the updating and improvement of existing ones.

Emblematic turns out to be the study[46], conducted by carrying out several experiments with an increasing degree of Continual Learning in order to observe changes in training behavior and performance:

- **Naive** → classical classifier training, in which data are available at all times and catastrophic forgetting is not counteracted
- **Fine tuning** → the model is initially trained as in the previous point, but after some time the intermediate part of the neural network is suspended from its work, while the terminating part of the network (that is the part that is charged with classification) continues the learning process
- **Continual Learning** → implemented through the methods of Learning Without Forgetting (in which in addition to the current model being trained, the previous one is also available) and Elastic Weight Consolidation (where past data classification weights are considered less reliable).

Well, the results obtained from comparing the metrics calculations made in these different situations suggest that Continual Learning may be a suitable and effective solution in MAD, although further in-depth studies are needed.

2.3 Databases for MAD

MAD techniques benefit from the existence of so many datasets and databases, both public and private, that contain useful information to evaluate the reliability of FRS systems and the performance of MAD algorithms.

Incidentally, it is worth pointing out that one of the first databases for face morphing, based on morphed image generation through landmark-based techniques, was introduced by researchers and professors at the University of Bologna.

Despite its initial small size, this dataset is continuously expanding with public databases.

However, despite the existence of many different morphing-themed datasets, almost all of them are private due to licensing and regulations regarding the privacy of the portrayed subjects.

Chapter 3

FEDERATED LEARNING

3.1 Definition

Federated Learning is a distributed machine learning technology that allows devices to train common patterns without having to share their sensitive data.

This type of learning takes place at a decentralized level, as data is processed on the device itself, without being transferred to a central server[51].

3.2 Scope of application

Federated Learning was originally developed to solve privacy and security problems in the processing of personal data.

Sensitive data can be difficult to protect and its dissemination or processing on a third-party server, in addition to being a point of huge vulnerability, violates data protection laws.

We recall in fact that, as is also the case with MAD, collecting and annotating datasets is a costly, laborious and problematic process from the standpoint of compliance with regulations that protect user privacy (such as GDPR, HIPAA, SHIELD, and PCI).

It is worth noting that today, however, there are technologies that already enhance the protection of personal data, including:

- **cancelable biometrics**[52] → provides for changing data through revocable but nonreversible transformations
- **BioHashing**[53] → uses random projections to generate templates
- **Differential privacy**[54] → the behavior of the algorithm changes little if a single individual joins or leaves the dataset
- **Homomorphic encryption**[55] → implements processing and calculations on encrypted data.

Nevertheless, the Federated Learning-oriented approach introduces a more robust and wider protection mechanism: efficient neural network learning from decentralized data.

3.3 The workflow

Through Federated Learning, machine learning models are trained on data on individual devices, such as smartphones, laptops or tablets, and information is shared only in the form of statistical aggregates, without individual raw data being transferred directly to other devices.

This makes the model training process more private and secure.

The Federated Learning process can be summarized in the following steps:

- **diffusion** → the server distributes the basic model to the various devices (clients) that form the machine learning network
- **local training** → each device trains the model locally on its local data and sends the aggregated results to the central server
- **model aggregating** → the central server uses these results to perform a secure aggregation[56] of the received parameters, improving the global model without storing any local information of the individual client
- **parameters broadcasting** → the server distributes the aggregated parameters to the devices
- **model updating** → all clients update their local models with the received aggregate parameters and check their performance.

This process is repeated several times until the overall model reaches a satisfactory level of accuracy.

The devices network required for Federated Learning is shown in figure 3.1.

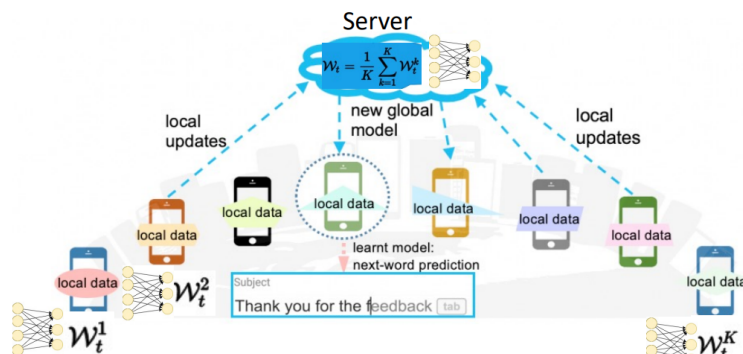


Figure 3.1: network structure for Federated Learning. Source: [57].

Federated Learning is already used today in many fields, including health care, artificial intelligence for mobile devices, advertising and cybersecurity.

In fact, today's applications of greatest interest are distributed learning on smartphones (for example for predicting the next word in Google's keyboard[58] or developing Apple's voice assistant without sharing user data with a central server), learning between organizations (fostering multi-institutional cooperation and collaboration) and the Internet of Things (think about wearable devices, autonomous vehicles and home automation).

3.4 The limitations still present

Despite the many advantages that have emerged so far, the Federated Learning paradigm still introduces challenges and obstacles[59] related

to:

- **communication** → internal communications in federated networks are much slower than local computations → methods are therefore needed to improve and speed up communications by iteratively sending model updates as part of the training process
- **heterogeneity of systems** → the different storage, performance, power and connectivity capabilities of each device lead to a system that is less stable and more prone to failures and slowdowns
- **non-IID data** → devices often generate and collect data in a non-IID manner (that is not independently and identically distributed), thus leading to unbalanced data
- **privacy-related pathological problems** remain.

An example of a problem related to the last point is collaborative deep learning, described in figure 3.2 below:

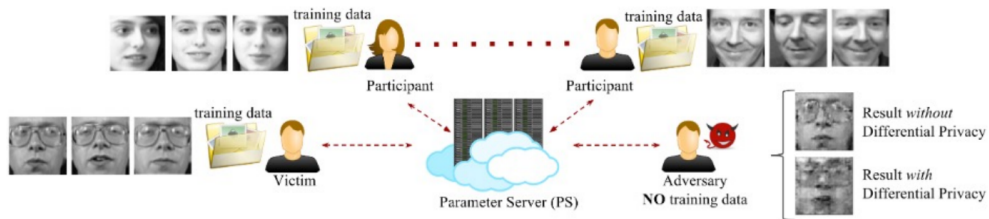


Figure 3.2: an example of collaborative[60] deep learning, in which the purpose of the "adversary" user is to reconstruct face images stored in the "victim" device via GAN. Source: [57].

The image above also allows us to expose three key properties of the

combined use of Federated Learning and Differential Privacy, which used together can bring benefits in some specific area of research:

- performance and privacy level are inversely proportional
- fixed the level of data protection, increasing the number of clients participating in Federated Learning can improve performance and convergence
- it is possible to calculate the optimal number of aggregations (or communication rounds) from the desired convergence and privacy level.

3.5 Split Learning Network

Federated Learning is made possible through the use of particular neural networks known as Split Learning Networks[61].

These involve each client training a partial neural network only up to a certain layer (called the cut layer).

The outputs of the cut layer are then sent to the server, which completes the training on the remaining part of the neural network and does back-propagation up to the cut layer.

The gradients of the cut layer are then sent back to the individual clients and the process continues until the network has completed training.

Figure 3.3 shows a more detailed view about Split Learning Networks.

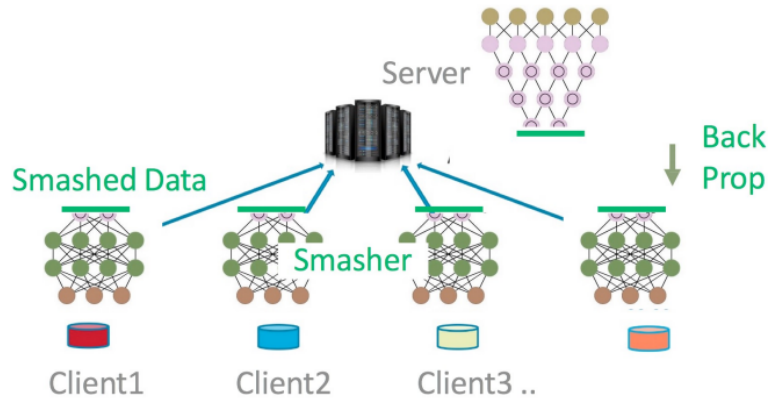


Figure 3.3: structure of Split Learning Networks [57].

Using a large number of clients and partitioning the learning process shows positive results and an increase in computational, communication and memorization efficiency.

3.6 Conclusions

Let us finally focus on the possible benefits that the Federated Learning approach brings in relation to the two main problems highlighted in the previous chapters: face recognition and face morphing attack detection. Note in particular the following image:

The Federated Face Recognition process is depicted and summarized in figure 3.4.

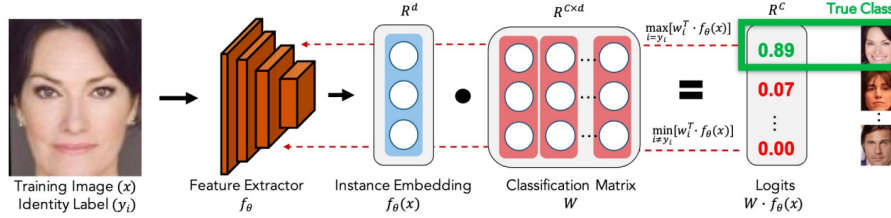


Figure 3.4: illustration of the so-called Federated Face Recognition[62]. Source: [57].

As noted in figure, the Federated Face Recognition process is initiated by passing a labeled input image through a feature extractor to obtain the feature vector, which is then multiplied by the classification matrix to obtain the probability that the input image belongs to each of the identities.

To summarize, the Federated Learning paradigm requires that training data and procedures are distributed across multiple devices and that the model is shared and learned collaboratively.

In this particular architecture, the server does not disappear, but rather plays the key role as the center for coordinating interactions.

By doing so, this approach turns out to be compliant with current data protection regulations and overcomes the problems highlighted in previous chapters regarding MAD algorithm research, namely the obstacles associated with sharing and transferring data between different research groups.

Figure 3.5 represents the interactions between server, user and data centers in a network based on Federated Learning.

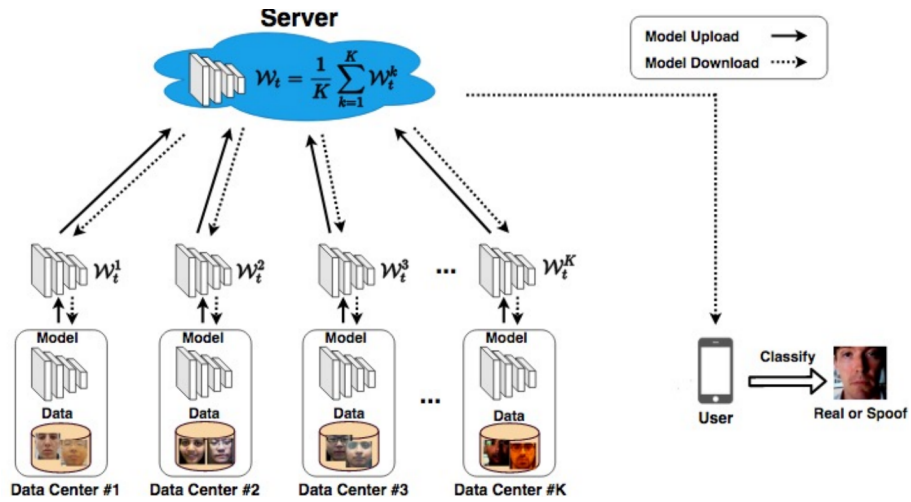


Figure 3.5: example of training and application of Federated Learning for the facial recognition problem. Source: [57].

However, despite the presence of some technical limitations such as connection latency or low throughput, Federated Learning is undoubtedly an interesting alternative to traditional approaches.

The in-depth evaluation of its applicability to the MAD problem, specifically, is studied in this report and presented in the following chapters.

Chapter 4

PRELIMINARY CONCEPTS

Before embarking on the study and testing of the practical operation of the Federated Learning framework NVFlare, some basic preliminary theoretical concepts essential for understanding the operations that will be carried out in the programming guide are given in the following sections.

4.1 Training and testing

The machine learning process that characterizes machine learning abstractly consists of finding a model that can describe and/or predict a phenomenon from a set of data.

The model is built using a dataset called training set, while its validity and correctness is verified using a different dataset called test set.

The training set is used to train the model, that is to make it learn from the data.

Specifically, the model is trained on the data in the training set in order

to study it, understand it and thus be able to describe and/or predict the phenomenon in question.

The testing set, on the other hand, is used to evaluate how the model behaves on data it has never seen before.

In this way, it is possible to check its correctness and robustness, but also its generalization, that is its ability to make good predictions on new data not present in the training set.

To give a concrete example, we can mention one of the most popular areas of machine learning use: image classification, a topic that is very extensive but which we will nonetheless go into in more detail in a following section.

Let us imagine that we want to build a model that can recognize images of cats.

The training set might consist of a thousand images of cats, while the testing set might consist of a hundred images of cats that the model has never seen before.

As stated earlier, during training the model learns to recognize cats, while during testing we assess how well the model has learned and thus how well it can recognize cats in the novel images in the testing set.

4.2 Epochs

Epochs are a key concept in deep learning: an epoch represents a complete cycle through the entire training dataset.

Thus, an epoch can also be defined as the time it takes the model to see and learn from all the samples in the training dataset.

Then an epoch, which is an important training hyperparameter, is completed whenever the model sees the entire training dataset.

For example, if a training dataset consisting of 100 images is used and 10 epochs are set, the model will evolve its ability to learn and generalize by studying all 100 images 10 times.

In practice, this means that as many iterations will be performed as the number of epochs set and each iteration covers all the images that make up the dataset.

The number of epochs is a very important parameter in training a machine learning model and is inherently dependent on the complexity of the model itself and the amount of training data available.

In general, it is important to find the right balance between the number of epochs needed to achieve good accuracy and the time and cost that can be invested in model training activities.

In theory, in fact, a model might learn better if it sees the data several times.

In reality, however, if too many epochs are used, the model may overtrain and then begin to memorize the data from the training set and wrongly repeat it to the new instances instead of generalizing the notions learned to the new data.

This problematic behavior is known as overfitting.

To avoid overfitting, it is important to use well-established strategies to identify the appropriate number of epochs, such as cross-validation evaluation (which is the method used by the NVFlare framework) or early stopping, which allow model training to be stopped when significant im-

provement in accuracy is no longer achieved.

4.3 Cross-site validation

As anticipated in the previous section, cross-validation is a technique used to evaluate the generalization ability of a machine learning model.

There are several variants of cross-validation, the best known of which are:

- **k-fold cross-validation** → the training data are divided into k sets and the model is trained on $k-1$ sets and evaluated on the last set
- **holdout cross-validation** → the training data are divided into a training set and an evaluation set and the model is trained on the training set and then evaluated on the evaluation set.

As mentioned in the previous section, it is of particular interest that cross-validation helps to identify the appropriate number of epochs for a machine learning model.

This can be done in several ways:

- by monitoring the evaluation error → during training the evaluation error should decrease while the number of epochs increases until a point is reached where no significant improvement is observed → the number of epochs in which such a halt in the decrease in evaluation error is observed can be considered as the optimal number of epochs
- analyzing the learning curve, that is how the occurrence of the rating error changes in relation to the current epoch → the number of

epochs at which the inflection point of the learning curve is observed, following which the rating error no longer decreases significantly despite the progression of epochs can be considered as the optimal number of epochs.

4.4 Learning curves

In deep learning, loss function and accuracy are two fundamental metrics commonly used to evaluate the performance of a model.

Specifically, loss is a measure of the model's error with respect to the training data.

There are different loss functions depending on the type of machine learning problem being addressed, such as the log-loss function for classification problems or the mean-squared-error loss function for regression problems.

Instead, accuracy is a measure of model precision that is calculated as the percentage of correct predictions to the total number of predictions made.

It is a common metric for classification problems, but can also be used for regression problems.

Both of these metrics allow the model to be evaluated throughout its operation, both during and after training.

In particular, loss is used to monitor the performance of the model during training, while accuracy is essential to evaluate the performance of the model on the test data, thus once the training is over.

A more refined tool for evaluating model performance than the individual indices of loss and accuracy are the graphs of these two parameters as a function of advancing time or epochs (called learning curves) which, in a classical correctness situation, should show a general trend toward model improvement.

The loss graph should in fact show a descending trend towards a lower value as the model learns from the training data and the error decreases. Initially, the loss might be high and fluctuate significantly, but it should stabilize and decrease over time.

The accuracy graph, on the other hand, should show a trend toward a higher value as accuracy is expected to increase.

Also in this case, large fluctuations are possible in the initial phase of training, but the value should stabilize and increase over time.

Examples of learning curves are depicted in figure 4.1.

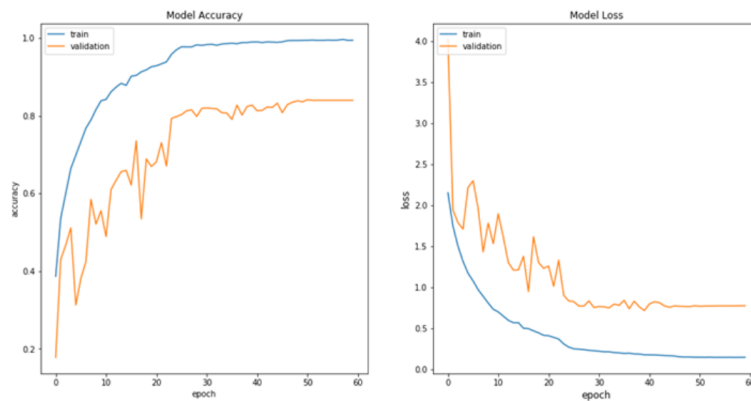


Figure 4.1: ideal curves for accuracy and loss. Source: Microsoft, machine-learning.paperspace.com/wiki/accuracy-and-loss.

In both graphs, one should then observe a point where the model performance does not improve significantly.

In this case, as extensively expressed in the previous sections, an early stopping technique can be used to stop training the model.

It is also important to note that the graphs may look different depending on the characteristics of the model and the training data.

For example, there may be spikes in the graphs due to overfitting or underfitting.

Loss and accuracy graphs are a powerful tool for studying and evaluating the performance of the training process for another reason as well: through these, it is possible to inviduate unexpected errors or behaviors, which could be symptoms of weakly robust training.

For example, if a high fluctuation of the loss function or a trend toward a high value is observed during training, these could be signs of overfitting or underfitting.

This could indicate that the model is too complex or too simple compared to the training data used, and therefore it is advisable to take countermeasures such as regularizing the model or providing more training data. Similarly, if a trend toward a low value of accuracy or its excessive fluctuation is observed, these could be signals of underfitting or a problem with the training data (for example poor quality or unbalanced distribution). In this case, it may be necessary to collect more data or improve the quality of existing data in order to increase their completeness and representativeness.

Usually each graph is actually composed of two curves, one describing the behavior during the training process and one describing the validation process.

4.5 Tensors

A tensor is a mathematical object that can be used to represent and manipulate multidimensional data efficiently.

There are several types of tensors (scalars, vectors, matrices...) but they all share the property of using indexes to refer to a specific element they contain.

In general, a tensor can in fact be represented by an array of numbers, each of which is associated with a set of indices that locates it.

For example, a third-dimensional tensor can be represented by a three-dimensional array of numbers in which each element is associated with three indices i , j and k that uniquely locate it.

In the context of neural networks, that is the context of interest for this report, tensors are used to represent input data, neuron weights and the intermediate and final results of the neural network's computational operations.

For example, the input data of a neural network can be represented by a tensor of size $N \times M \times C$, where:

- N and M are the dimensions of the image
- C is the number of channels (for example 3 for RGB images).

Instead, the weights of neurons can be represented by a tensor of dimension $K \times L \times M \times N$, where:

- K and L are the sizes of the filters used by the neural network
- M and N are the sizes of the input data.

Tensors are also used to represent intermediate results of the neural network's computational operations, such as neuron activation values and the results of convolution (a function that through mathematical vector product operations allows the identification of specific features in the input data such as lines and shapes) and pooling operation (an operation used after convolution to reduce the resolution of the data and make a kind of summary of the information that emerged, also to avoid overfitting problems and make the model more robust).

Summarizing, using tensors to represent the data and operations of the neural network allows complex calculations and operations to be performed efficiently, quickly and accurately.

In addition, the use of tensors underlies the development of numerical computing libraries such as TensorFlow and PyTorch, which have become an essential tool for creating and training neural networks.

Both of these libraries are discussed in detail in the chapters and sections that follow.

4.6 Neural networks

As is well known, the technology behind deep learning is the so-called neural networks.

They are nothing more than an artificial imitation of neurons in the human brain.

In fact, such innovative idea has rather old origins, so much so that the first artificial neuron was introduced by McCulloch and Pitts in 1943.

In an artificial neuron (AN), simulating what occurs in nature:

- there are inputs \rightarrow not analog signals but digital numbers
- these inputs are weighted, that is each is given a priority specifying its importance
- the inputs are also merged via a sum function
- finally an activation function is used to generate the final output.

As easy to imagine, a single artificial neuron can only solve linear problems, but one can solve this problem using multiple NAs organized in layers.

This technique is called Multi Layer Perceptor (MLP) and, although it greatly improves computational capacity, it introduces several complex mathematical problems.

In summary, neural networks can thus be said to consist of groups of artificial neurons divided into layers, typically structured as follows:

- surely one input layer and one output layer are present.
- there are also one or more hidden, that is intermediate, layers present.

The structure of neural layers is shown in figure 4.2.

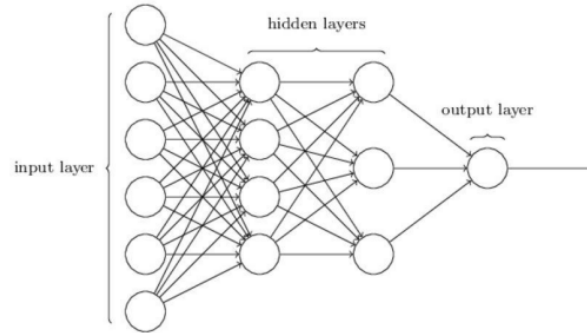


Figure 4.2: structure of neural layers.

The peculiarity of this structure is that each neuron is completely connected with those of the next level: this feature is also inherited from the hierarchical nature of human neurons.

In specific, it is possible to distinguish between two types of neural networks:

- **feedforward** → is the most widely used type and involves neurons in one level being connected to those in the next level, that is no connections backward or to neurons in the same level
- **recurrent** → in this type, recommended when the need is to simulate a short-term memory effect, feedback or even recursive connections are possible instead.

On the other hand, as far as training layers of neural networks is concerned, the basic consideration to emphasize is that the greater is the

number of hidden layers , the greater is the performance .

However, it must also be considered that with a high number of layers comes the need for more training data and reflexively greater computational complexity.

Although training tasks are extremely complicated, there are specific frameworks that greatly simplify the work, some of which are also used in the NVFlare package.

From a purely statistical point of view, training a neural network means minimizing the loss function, that is the mathematical function that measures the error between the prediction of the label and its actual value and returns a real number as a presentation of the performance.

Regarding classification tasks, the best-known loss functions are:

- **Cross Entropy** → indicates the difference between what the model predicts as the output of the distribution and the actual value of the distribution
- **Binary Cross Entropy (BCE)**
- **Categorical Cross Entropy (CCE)**.

Mathematically, the last two functions can be calculated using the formulas

$$BCE = -\gamma_i \cdot \log\beta_i - (1 - \gamma_i) \cdot \log(1 - \beta_i)$$

$$CCE = -\sum \gamma_i \cdot \log\beta_i$$

where:

- β_i represents the i-th scalar value of the model output (that is its prediction)
- instead γ_i is the value of the corresponding label.

It must be specified that to use the Cross Entropy loss function it is necessary for the output layer to return probabilities.

To do this it is possible to use a layer called softmax which, given an n-dimensional vector of real numbers, is able to transform it into another vector of real numbers in the interval $[0, 1]$ whose sum is exactly 1.

Any continuously differentiable mathematical function has these properties.

Defined the loss function, to minimize it it is sufficient to apply a process of adjusting the weights and biases of each neuron by using gradients.

The use of these mathematical tools is justified by the fact that, looking at the mathematical definition of the derivative of a function, it can be said that it measures the sensitivity to the change in the value of the function itself (that is its output) in relation to the change in its argument (that is its input), and that is its gradient.

In other words, the gradient is the mathematical tool that can describe the direction in which a function is pointing.

Introduced this mathematical tool, it is possible to minimize the loss function with descending gradient approaches that adjust the weights in the following manner:

$$w' = w - \eta \frac{\partial L}{\partial w}$$

In fact, the formula states that the new weights are calculated from the old ones and a certain amount of gradient having as coefficient the learn-

ing rate η .

In the literature, a wide variety of descending gradient approaches (called optimizers) are described, including:

- **Vanilla Gradient Descent** → computes the gradient relative to parameters of the entire training dataset → is not recommended for large training datasets given the slowness and computational cost of the algorithm
- **Stochastic Gradient Descent** → is based on an iterative approach, in which for each iteration a simple batch of data is selected and gradients are computed on it → disadvantages are that this algorithm is rather tortuous and requires strong parameterization
- **Adam**[63] → is an improved and more efficient optimizer than the previous type, in fact it requires fewer parameters and memory requirements.

Related to what was stated earlier in this section, calculating the gradient in neural networks can be complex since there are many layers present that make the loss dependent on hidden parameters despite having access only to the output values.

Luckily, it is possible to use an algorithm called backpropagation for calculating the gradient of loss functions considering all the parameters present, the procedure of which is based on calculating the derivative of a compound function via chain rule.

The backpropagation is a local process, in which neurons do not need to know the topology of the global network containing them.

By jointly using backpropagation and chain rule, it is therefore possible

to propagate the derivative of the loss function to the prior inputs of the network.

For all these processes to be possible, it is shown that the loss function must be derivable.

4.7 Convolutional neural networks

A convolutional neural network (CNN[64]) is a type of feed-forward neural network, that is one in which data flows unidirectionally from input to output, without any loop or return.

As is also the case with classical neural networks, data in this technology also flows through a series of layers of neurons, each of which processes data independently of the others, that is without interacting in any way with other layers before or after it.

Convolutional neural networks are special neural networks specifically designed to process images.

In fact, the traditional MLP architecture has difficulty adapting to images, this is because it attempts to flatten the input data, which physiologically would be three-dimensional.

In contrast, CNNs exploit the reverse approach: they do not twist the structure of the input images but convert that of the neurons into a 3D format.

In fact, unlike those in MLP networks, CNN layers are composed of neurons methodically arranged in 3 dimensions: width (W), height (H) and depth (C, often greater than 1):

Figure 4.3 shows how to convert traditional MLP into a structure suitable for CNN.



Figure 4.3: transition from traditional MLP structure to three-dimensional structure for CNN.

At this point, a mathematical operation called convolution, applied through a sliding-windows process, is exploited to go from the three-dimensional input to the kernel of neurons that is also three-dimensional.

In image convolution, each kernel is convolved with the input data volume, producing a 2D mapping.

Since the depth is often greater than 1, multiple kernels exist and therefore a mapping is produced for each kernel.

All these produced maps are then stacked on top of each other to create the output volume.

An example of convolution operation is reported in figure 4.4.

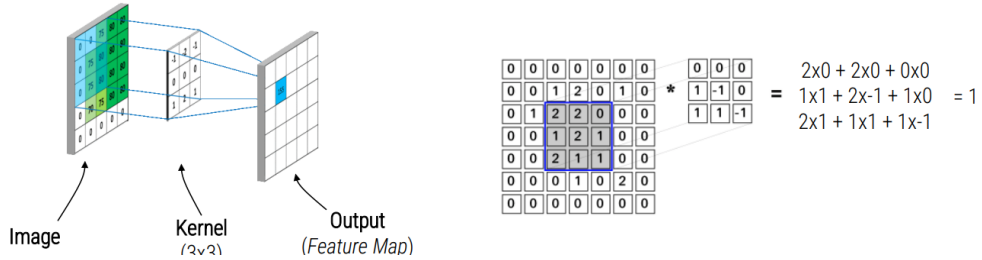


Figure 4.4: convolution operation.

At this point, the learnable weights of the MLP network are grouped in the kernel, where each cell can be viewed as a single learnable weight.

Taking these tasks, the convolutional layers learn to extract various types of visual information from the image in a hierarchical manner such as lines, shapes and textures:

- in the layers closer to the input the CNNs include filters to extract simple visual information
- in the layers placed deeper, in contrast, filters are able to extract much more complex visual information.

As repeated many times, this feature of differentiated layered processing of information also simulates the actual working mechanism of the cerebral cortex of the human brain.

In addition, it has been shown in the literature that the filters in question can be equated in both behavior and characteristics with the neurological

stimuli of the human brain.

In addition to the convolution layers mentioned above, there are also other types of layers, which perform fundamentally important tasks, among which it is worth mentioning:

- **pooling layers** → partition the input volume into spatial subsamples in order to more accurately provide the location of detected features, reduce the required storage space and filter features by importance
- **activation layers** → deal with the calculation of activation features
- **flatten layers** → used for their ability to connect a 3D feature extractor with a one-dimensional classifier
- **fully-connected layers** → are used last to perform the final image classification based on the extracted features.

Finally, training a CNN often requires having a large number of examples of images labeled during the training phase, that is images that have been previously labeled with the correct categories.

During the training process, the CNN in fact learns to recognize the relevant features for each category and use them to generalize and classify new images.

More specifically, training of CNNs can be done in different ways:

- **from scratch** → when starting only from the initialization of weights and using a large amount of training data
- **pre-trained** → when a previously trained network is used

- **fine-tuned** → when a previously trained network is used, in which, however, the feature extractor is kept unchanged but changes are made to the classifier, such as changing the number of classes.

Once the model has been trained, the testing process allows its accuracy to be evaluated on a set of new test data, which consists of images not used during the training process.

At this stage, the model uses the features extracted from each test image to try to make its exact classification and the predicted label is compared with the correct label to calculate the accuracy of the model.

4.8 Image classification

In the programming guide that follows, the establishment of a Federated Learning system in order to solve the problem of image classification (an area to which the MAD problem can also be traced) will be frequent.

Image classification is an active research area in machine learning and artificial intelligence, with many practical applications in various fields such as medicine, security, robotics and entertainment.

The main task of an image classification system is to assign a label to an image based on the recognized content.

For example, an image classification system could be trained to recognize pets in a picture and label it as "cat" or "dog" based on its content.

There are several methods for classifying images, but one of the most popular and successful is the use of convolutional neural networks (CNNs).

Image classification is a rapidly evolving area, with new techniques and methods being continuously developed to improve the accuracy and speed of the classification process.

In order to better understand the problem of image classification, it is useful to study the Python script proposed below (excerpted from the guide found at the link <https://learn.microsoft.com/en-us/windows/ai/windows-ml/tutorials/pytorch-data>), which also provides an opportunity to introduce the PyTorch library, a topic addressed in more detail later.

For the moment, suffice it for us to know that PyTorch is an open-source machine learning library developed by Facebook that provides a wide range of features for data processing, machine learning and its optimization, including support for neural networks, machine learning algorithms, pre-trained models and many other utility functions.

The Torchvision package, in particular, includes the most popular machine vision datasets, including CIFAR10, which is also widely used in NVFlare.

In the next chapter, a section has been devoted specifically to delve into the features of this dataset: for now, it is sufficient to know that it is widely used to train an image classification model and consists of 50000 training images and 10000 test images, all of size 32×32 with RGB color coding.

The images are divided into 10 classes: airplane, car, bird, cat, deer, dog, frog, horse, ship and truck.

We now begin to study an example of a Python script dedicated to image classification, which is a significant example of what will also happen at

the algorithm level during the execution of jobs involving this application in NVFlare.

As we see in the first code snippet below, one must initially follow three steps to import and load the CIFAR10 dataset into PyTorch:

- define the transformations that are to be applied to the images → to train the model it is indeed necessary to transform the images into normalized tensors in the interval $[-1, 1]$
- create an instance of the dataset and load it → this is possible using the Dataset class
- access the data using the DataLoader class, whose methods allow you to fetch data from the dataset and load it into memory, effectively handling all interfacing and access operations to the dataset.

```
1 from __future__ import print_function
2 from torchvision.datasets import CIFAR10
3 from torchvision.transforms import transforms
4 from torch.utils.data import DataLoader
5 import torch
6 import torch.nn as nn
7 import torchvision
8 import torch.nn.functional as F
9 from torch.optim import Adam
10 from torch.autograd import Variable
11 import matplotlib.pyplot as plt
12 import numpy as np
13
14 # Loading and normalizing the data.
15 transformations = transforms.Compose([
16     transforms.ToTensor(),
```

```

17     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
18 ])
19
20 #We define the batch size of 10 to load 5,000 batches of
    images.
21 batch_size = 10
22 number_of_labels = 10
23
24 # Create an instance for training: the CIFAR10 train dataset
    will be downloaded locally.
25 train_set =CIFAR10(root="./data",train=True,transform=
    transformations,download=True)
26
27 # Create a loader for the training set.
28 train_loader = DataLoader(train_set, batch_size=batch_size,
    shuffle=True, num_workers=0)
29 print("The number of images in a training set is: ", len(
    train_loader)*batch_size)
30
31 # Create an instance for testing, note that train is set to
    False.
32 test_set = CIFAR10(root="./data", train=False, transform=
    transformations, download=True)
33
34 # Create a loader for the test set, note that each shuffle is
    set to false for the test loader.
35 test_loader = DataLoader(test_set, batch_size=batch_size,
    shuffle=False, num_workers=0)
36 print("The number of images in a test set is: ", len(
    test_loader)*batch_size)
37
38 print("The number of batches per epoch is: ", len(train_loader
    ))

```

```
39 classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog',  
            ', 'horse', 'ship', 'truck')
```

Once the dataset has been acquired, the next step is to define the convolutional neural network (CNN) to be used.

The theory regarding this topic is given in the previous section, but it should be specified that the CNNs used in the script being analyzed are structured in 14 layers, each of which is capable of detecting specific features in an image.

From a more practical point of view, during the training tasks the network will process and study the input across its many layers, compute the loss function to understand how far the predicted label deviates from the actual label and finally propagate the gradients backward to update the layer weights.

Iterating this process for all the input data that make up the dataset, the network is able to learn how to set its weights to get the best results.

As anticipated earlier, remember that CNNs are called feed-forward networks, so they require the definition of a forward function, whose job is simply to compute the value of the loss function.

The backward function in PyTorch is in fact implicitly and automatically defined.

Based on the above, the following script fragment defines the CNN:

```
1 class Network(nn.Module):  
2     def __init__(self):  
3         super(Network, self).__init__()
```



```

4
5     self.conv1 = nn.Conv2d(in_channels=3, out_channels=12,
6     kernel_size=5, stride=1, padding=1)
7     self.bn1 = nn.BatchNorm2d(12)
8     self.conv2 = nn.Conv2d(in_channels=12, out_channels
9     =12, kernel_size=5, stride=1, padding=1)
10    self.bn2 = nn.BatchNorm2d(12)
11    self.pool = nn.MaxPool2d(2,2)
12    self.conv4 = nn.Conv2d(in_channels=12, out_channels
13    =24, kernel_size=5, stride=1, padding=1)
14    self.bn4 = nn.BatchNorm2d(24)
15    self.conv5 = nn.Conv2d(in_channels=24, out_channels
16    =24, kernel_size=5, stride=1, padding=1)
17    self.bn5 = nn.BatchNorm2d(24)
18    self.fc1 = nn.Linear(24*10*10, 10)
19
20    def forward(self, input):
21        output = F.relu(self.bn1(self.conv1(input)))
22        output = F.relu(self.bn2(self.conv2(output)))
23        output = self.pool(output)
24        output = F.relu(self.bn4(self.conv4(output)))
25        output = F.relu(self.bn5(self.conv5(output)))
26        output = output.view(-1, 24*10*10)
27        output = self.fc1(output)
28
29        return output
30
31    # Instantiate a neural network model
32    model = Network()

```

The next step is to define the loss function.

In the fragment we will see is used a loss function that is characterized by two complex statistical tools introduced in the previous sections, the

Cross-Entropy loss and the Adam optimization algorithm.

Then follows the code that trains the model, which actually cycles over the data iterator, feeds inputs to the network and takes care of optimization.

Note that the device defined in the code will be an NVIDIA GPU if present, otherwise the CPU:

```
1 loss_fn = nn.CrossEntropyLoss()
2 optimizer = Adam(model.parameters(), lr=0.001, weight_decay
   =0.0001)
3
4 def saveModel():
5     path = "./myFirstModel.pth"
6     torch.save(model.state_dict(), path)
7
8 def testAccuracy():
9     model.eval()
10    accuracy = 0.0
11    total = 0.0
12
13    with torch.no_grad():
14        for data in test_loader:
15            images, labels = data
16            outputs = model(images) # run the model on the
test set to predict labels
17            _, predicted = torch.max(outputs.data, 1) # the
label with the highest energy will be our prediction
18            total += labels.size(0)
19            accuracy += (predicted == labels).sum().item()
20
21    accuracy = (100 * accuracy / total) # compute the accuracy
over all test images
```

```

22     return(accuracy)
23
24 def train(num_epochs):
25     best_accuracy = 0.0
26
27     device = torch.device("cuda:0" if torch.cuda.is_available
28 () else "cpu") # Define your execution device
29     print("The model will be running on", device, "device")
30     model.to(device) # Convert model parameters and buffers to
31     CPU or Cuda
32
33     for epoch in range(num_epochs): # loop over the dataset
34     multiple times
35         running_loss = 0.0
36         running_acc = 0.0
37
38         for i, (images, labels) in enumerate(train_loader, 0):
39
40             # get the inputs
41             images = Variable(images.to(device))
42             labels = Variable(labels.to(device))
43
44             optimizer.zero_grad() # zero the parameter
45             gradients
46             outputs = model(images) # predict classes using
47             images from the training set
48             loss = loss_fn(outputs, labels) # compute the loss
49             based on model output and real labels
50             loss.backward() # backpropagate the loss
51             optimizer.step() # adjust parameters based on the
52             calculated gradients
53
54             # Let's print statistics for every 1,000 images

```

```

48         running_loss += loss.item()      # extract the loss
value
49         if i % 1000 == 999:
50             print('[%d, %5d] loss: %.3f' %
51                   (epoch + 1, i + 1, running_loss / 1000))
52             running_loss = 0.0
53
54         # Compute and print the average accuracy fo this epoch
when tested over all 10000 test images
55         accuracy = testAccuracy()
56         print('For epoch', epoch+1, 'the test accuracy over the
whole test set is %d %%' % (accuracy))
57
58         # we want to save the model if the accuracy is the
best
59         if accuracy > best_accuracy:
60             saveModel()
61             best_accuracy = accuracy

```

Finally, as reported in the last snippet below, we can also proceed to define the testing function and the main of the script.

Also note in particular the `testClasses()` function, which reports the accuracy performed for each category:

```

1 def imageshow(img):
2     img = img / 2 + 0.5      # unnormalize
3     npimg = img.numpy()
4     plt.imshow(np.transpose(npimg, (1, 2, 0)))
5     plt.show()
6
7 def testBatch():
8     images, labels = next(iter(test_loader)) # get batch of
images from the test DataLoader
9

```

```

10     imshow(torchvision.utils.make_grid(images)) # show all
images as one image grid
11
12     # Show the real labels on the screen
13     print('Real labels: ', ' '.join('%5s' % classes[labels[j]]
14                                     for j in range(batch_size)))
15
16     # Let's see what if the model identifies the labels of
those example
17     outputs = model(images)
18
19     _, predicted = torch.max(outputs, 1) # We got the
probability for every 10 labels. The highest (max)
probability should be correct label
20
21     # Let's show the predicted labels on the screen to compare
with the real ones
22     print('Predicted: ', ' '.join('%5s' % classes[predicted[j]
23                                     ])
24                                     for j in range(batch_size)))
25 def testClasses():
26     class_correct = list(0. for i in range(number_of_labels))
27     class_total = list(0. for i in range(number_of_labels))
28     with torch.no_grad():
29         for data in test_loader:
30             images, labels = data
31             outputs = model(images)
32             _, predicted = torch.max(outputs, 1)
33             c = (predicted == labels).squeeze()
34             for i in range(batch_size):
35                 label = labels[i]
36                 class_correct[label] += c[i].item()

```

```

37         class_total[label] += 1
38
39     for i in range(number_of_labels):
40         print('Accuracy of %5s : %2d %%' % (
41             classes[i], 100 * class_correct[i] / class_total[i
42         ]))
43 if __name__ == "__main__":
44
45     # Let's build our model
46     train(5)
47     print('Finished Training')
48
49     # Test which classes performed well
50     testAccuracy()
51
52     # Let's load the model we just created and test the
53     accuracy per label
54     model = Network()
55     path = "myFirstModel.pth"
56     model.load_state_dict(torch.load(path))
57
58     # Test with batch of images
59     testBatch()
60     testClassess()

```

Proceeding with the execution of the script just created, it can be seen that the loss value will be printed every 1000 batches of images and that this value should decrease as time progresses.

The accuracy, that is the number of images correctly identified by the model, will also be reported.

After the training phase is completed, in fact, an output similar to the following shown in figure 4.5 is obtained:

```
Verifying https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100.0%
Extracting ./data/cifar-10-python.tar.gz to ./data
The number of images in a training set is: 50000
Files already downloaded and verified
The number of images in a test set is: 10000
The number of batches per epoch is: 5000
The model will be running on cpu device
[1, 1000] loss: 1.774
[1, 2000] loss: 1.490
[1, 3000] loss: 1.348
[1, 4000] loss: 1.258
[1, 5000] loss: 1.173
For epoch 1 the test accuracy over the whole test set is 61 %
[2, 1000] loss: 1.095
[2, 2000] loss: 1.033
[2, 3000] loss: 1.026
[2, 4000] loss: 0.995
[2, 5000] loss: 0.999
For epoch 2 the test accuracy over the whole test set is 65 %
[3, 1000] loss: 0.906
[3, 2000] loss: 0.891
[3, 3000] loss: 0.902
[3, 4000] loss: 0.905
[3, 5000] loss: 0.895
For epoch 3 the test accuracy over the whole test set is 67 %
[4, 1000] loss: 0.804
[4, 2000] loss: 0.824
[4, 3000] loss: 0.827
[4, 4000] loss: 0.832
[4, 5000] loss: 0.816
For epoch 4 the test accuracy over the whole test set is 68 %
[5, 1000] loss: 0.733
[5, 2000] loss: 0.755
[5, 3000] loss: 0.758
[5, 4000] loss: 0.772
[5, 5000] loss: 0.790
For epoch 5 the test accuracy over the whole test set is 69 %
Finished Training
```

Figure 4.5: output following the training phase of the script shown above. Note, as expected, the increase in accuracy and decrease in loss.

After printing the training-related outputs, the script also reports the results obtained during the testing phase.

The expected output would be similar to the one depicted in figure 4.6.

```
Finished Training
Real labels:   cat  ship  ship plane  frog  frog  car  frog  cat  car
Predicted:    ship  ship plane plane  frog horse  dog  deer  cat  car
Accuracy of plane : 76 %
Accuracy of car : 78 %
Accuracy of bird : 46 %
Accuracy of cat : 54 %
Accuracy of deer : 37 %
Accuracy of dog : 49 %
Accuracy of frog : 32 %
Accuracy of horse : 67 %
Accuracy of ship : 82 %
Accuracy of truck : 83 %
```

Figure 4.6: output following the testing phase of the script reported above. Note the behavior of the model in a test of 10 images and the accuracy reported relative to the category.

It is important to understand the reported script well since it summarizes very clearly and faithfully what the jobs we will study below present in the framework (specifically **cifar10** and **hello-pt-tb**) consist of.

4.9 Scatter and gather

The concept of scatter and gather is central to parallel programming, as it allows data to be distributed and gathered among different processors or compute nodes.

It makes it possible to take full advantage of the computing power of multiprocessor systems and clusters, significantly speeding up data processing.

The scatter function distributes input data across multiple processors so that each can process a portion of the data in parallel.

For example, if you have a large array of data that you want to process, scatter can divide this data into smaller parts and assign each part to a different processor.

In this way, each processor can independently process its portion of the data in parallel, greatly speeding up the overall processing.

After the data has been processed by the various processors, the gather function collects and merges it into a single output.

For example, if each processor has processed a portion of data, gather can gather them into a single output array.

In this way, data processed in parallel are gathered and returned as a single data set.

In general scatter and gather are used for distributing and collecting data in parallel environments such as computer clusters or single multi-core computers.

These operations are used in a variety of applications, such as big data processing, parallel computing and distributed programming: in fact, they form the basis of the overall NVFlare workflow.

There are several high-level libraries that provide scatter and gather functions to facilitate parallel programming.

Chapter 5

NVFLARE

After an introduction and explanation of the main deep learning concepts, it is now possible to go deeper into the study of NVFlare, based on the detailed documentation at the link <https://nvflare.readthedocs.io/en/2.3.0/>.

The structure of the framework can be seen in figure 5.1.

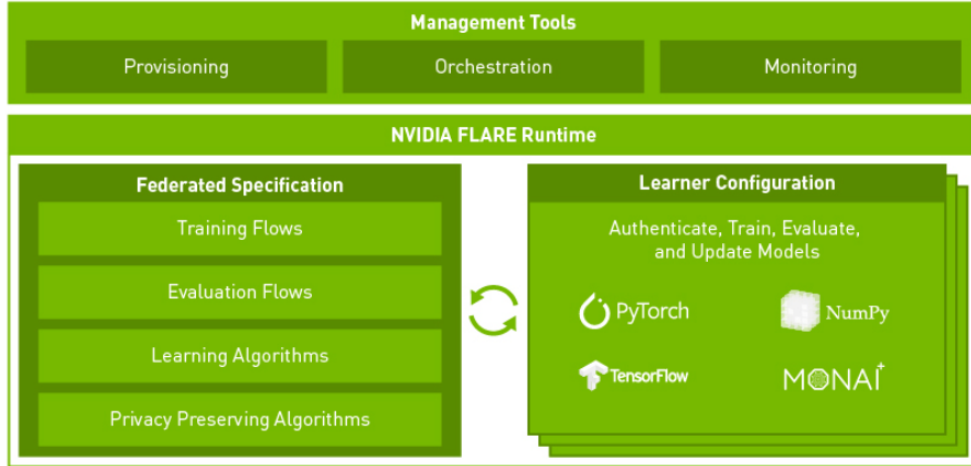


Figure 5.1: NVFlare general overview. Source: NVIDIA.

5.1 Overview

As presented in the documentation, NVFlare (NVIDIA Federated Learning Application Runtime Environment) is a Python-based, open-source, extensible and flexible SDK that allows researchers and users to adapt the existing classical Deep Learning workflow to the Federated Learning paradigm.

Through the NVFlare framework, developers can collaborate on the creation of distributed applications in a secure and privacy-friendly manner. The framework is also designed to ensure scalability and robustness to realistic deployments of Federated Learning applications.

The kit offers:

- a runtime development environment that easily enables Federated Learning experiments in realistic scenarios, even supporting parallel execution of multiple tasks to optimize productivity
- a system capable of implementing Federated Learning with a highly flexible and malleable infrastructure
- built-in implementation of various tools, including federated training workflows (for example scatter-and-gather), federated evaluation workflows (global model evaluation and cross site model validation), learning algorithms, privacy-preserving algorithms (homomorphic encryption and Differential Privacy), extensible management tools for secure provisioning (TLS certificates) and for easier management and orchestration (admin console and admin APIs), as well as tools for monitoring experiment results (such as TensorBoard)
- a rich set of programmable APIs that allows researchers to create new and original workflows and privacy-friendly learning algorithms.

5.2 System architecture

As outlined very superficially above, NVFlare includes a wide range of components that enable researchers and developers to build and deploy end-to-end Federated Learning applications.

The high-level architecture is depicted in figure 5.2 below:

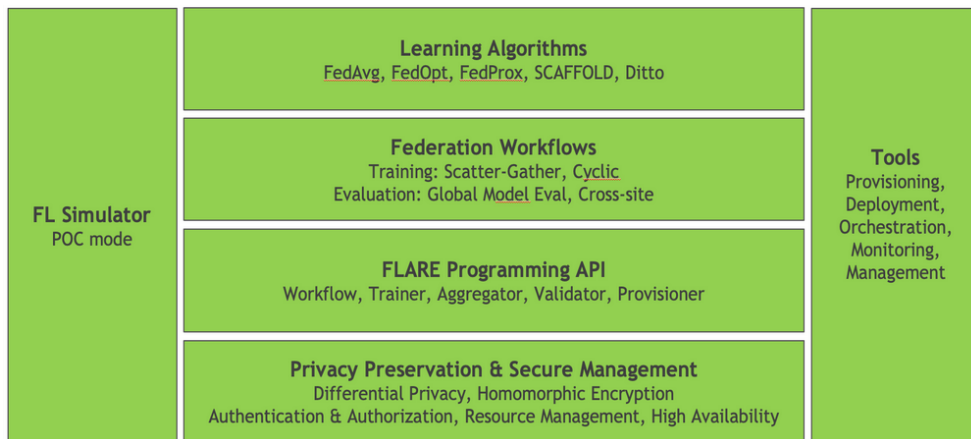


Figure 5.2: NVFlare component stack overview. Source: NVIDIA.

The center column shows the fundamental components and tools for proper and secure management of the platform.

On either side of the center column are two sections devoted respectively to experimentation via simulator on a single local workstation on the one hand and to tools dedicated to workflow management and effective and secure deployment on the other.

5.3 Design principles

To further explore the philosophy behind this framework, it may be useful to list the design principles that guided its development:

- keep it simple, less is more → only the necessary components have been built and described
- design to specifications → the design is based on well-defined and

approved specifications, which are in fact a guide for using the system and help ensure that it meets the needs of users

- build for real-world scenarios → the framework has been developed considering the situations, circumstances and real-world use cases in which it will be used
- keep the system general-purpose → the framework has been designed to be flexible and adaptable to a wide range of applications and uses, that is it has a general architecture that allows it to be easily expanded or modified to meet future needs without the need for a complete rewrite → this principle emphasizes the importance of avoiding overly specific solutions that may make the system more rigid and less suitable for situations other than those for which it was originally designed
- client system friendly → the framework is designed to be easy to use and accessible for clients or end users, that is it is intuitive and compatible with the equipment and systems most commonly used by clients.

5.4 Key features

The main functionalities offered by the NVFlare platform can be distinguished into 5 areas:

- **training workflows** → it is possible to choose between a scatter and gather type workflow (that is the default one, in which the central server acts as controller and sends the tasks to be executed to the clients, who respond with their shareable results, that is the

training model weights, which are finally aggregated by the server) or a cyclic workflow (in which the server submits a series of tasks scheduled to be performed in a cyclic execution by a group of clients and in this case the shareable results are passed from one client to the next to be refined until the last client, which returns the final results to the server)

- **evaluation workflows** → it is possible to adopt either cross site model validation (in which the results of a local validation are collected by the server) or global model evaluation (the server's global model is distributed to each client for local validation)
- **privacy protection algorithms** → including differential privacy and homomorphic encryption
- **different learning algorithms** → including FedAverage (in which a set of the initial weights is distributed among the clients performing training locally and after that they return the local weights in aggregate and in turn these are redistributed to the clients), Fed-Prox (implements a loss function to penalize a client's weights based on its deviation from the global model) or FedOpt (implements a generator that can use a specific optimizer when updating the global model)
- rich and extensive **set of example applications** to practice about the whole process of Federated Learning.

Among the new features introduced by the latest version, 2.2, it is worth mentioning the efforts made toward:

- workflow acceleration

- simplification of deployment in real-world scenarios
- enabling and enhancing integration with other systems and platforms.

Chapter 6

PROGRAMMING GUIDE

Having addressed the general overview, a more practical analysis of the operation that characterizes the NVFlare framework is given in this chapter, so that key concepts can be better understood through simulations and meaningful practical examples.

Note that the examples that follow will have an incremental difficulty, in fact starting from the simulation of small systems and analyzing simple situations will lead to the construction of an increasingly complex and complete architecture that can potentially be used to establish a realistic Federated Learning infrastructure.

This guide therefore also aims to summarize and simplify the documentation present today for the use of NVFlare.

In fact, this chapter seeks to provide a reference point not only for new and inexperienced users, but also for developers with experience in Machine Learning, who may struggle with the excessive verbiage of the present doc-

umentation and may instead find in the following document an extraction of the main concepts, at least to understand the features and enormous potential of the framework.

In particular, the guide will delve more concretely into the following aspects of the framework:

- installation
- application simulation
- evaluation of its use applied to the problem of face morphing detection.

It should be noted that this programming guide does not aim to study in depth the individual files and algorithms used by the framework, but intends to focus more on the infrastructure that this innovative tool enables and the interrelationships that emerge between the various components. Consequently, only rarely we will look at the code of any of the scripts that make up the framework and make an assessment of them, as this is beyond the scope of this report.

Finally, it should be noted that the examples addressed will often require interacting with external tools that are famous in the Deep Learning field, such as Numpy, PyTorch and Tensorflow.

Therefore, these tools will be introduced and explained before they are used, in order to make this paper accessible to all and self-contained, that is searchable without the need to query external resources for its understanding.

6.1 Requirements

The only requirement for the proposed practice tests is to have a Unix-Like operating system and basic knowledge of bash scripting.

In particular, we recommend the use of Ubuntu 22.04.1 LTS (downloadable from the link <https://releases.ubuntu.com/22.04/>) since it corresponds exactly to the version used for the experiments that follow.

Note that the choice to deploy the NVFlare framework exclusively on Unix-Like operating systems is not accidental: not only does NVIDIA's development team prefer these systems, but one must also consider that the Linux environment greatly simplifies many aspects related to Deep Learning and that some libraries do not work on other operating systems, for example the library related to homomorphic encryption.

6.2 Installation

The easiest way to install NVFlare is to follow the guide in the documentation attached at the beginning of this chapter, on which the concepts that will be explained in the following sections are also based.

6.2.1 The importance of a virtual environment

A virtual environment is an isolated system environment, created within an existing operating system, that allows software to run independently of the configurations of the host operating system.

This means that a virtual environment can have its own system settings, libraries and software versions, which may also be different from those of

the host operating system.

Virtual environments are used primarily for the enormous advantages they provide:

- isolation → virtual environments allow software to run in an environment isolated from the host operating system → so any changes made to the virtual environment will not affect the host operating system and vice versa and this is critical when working with software that might conflict with other applications or operating system configurations
- reproducibility → virtual environments allow you to create reproducible development and test environments, that is ones that can be recreated multiple times with the same configurations and libraries
- simplicity → virtual environments are generally easy to configure and use and allow for easy management of software dependencies, packages and library versions → this means that developers no longer have to deal with any conflicts with other versions of libraries or software on the host operating system
- collaboration and sharing → virtual environments make it easy to share a development environment with other users
- portability → virtual environments can be easily exported to other systems, ensuring that the software runs completely identically on different platforms
- resource saving → virtual environments allow users to use only the system resources they actually need, saving host operating system

resources.

It is possible to create a virtual environment at the level of the entire operating system, using virtualization software such as VirtualBox or VMware, but there are also some programming language-specific virtual environments such as virtualenv or conda for Python (which we will in fact set up and use extensively throughout the guide).

As also specified by the long list of advantages, this guide recommends the use of a virtual environment as it allows you to work in a more protected environment, where you can install different versions of Python and its libraries without creating conflicts and indeed greatly simplifying the learning process also thanks to the possibility of specifying from the beginning the necessary dependencies, that is which libraries to download.

In particular, for the creation of the virtual environment, we suggest the use of Anaconda[70], an open-source distribution of Python that includes a wide range of libraries and tools for data processing, statistical analysis and data visualization, as well as a package manager called conda that allows you to easily install, update and manage Python libraries and packages.

It makes it possible and easy to create custom virtual environments.

Among other things, Anaconda also offers a user-friendly GUI called Anaconda Navigator that allows users to easily manage packages and virtual environments or launch other applications more easily.

We then proceed to install Anaconda, substituting from the commands under the tag <VERSION>with the version you want to install (for ex-

ample 2022.10):

```
1 root $> sudo apt update
2 root $> wget https://repo.anaconda.com/archive/
    Anaconda3-<VERSION>-Linux-x86_64.sh
3 root $> bash Anaconda3-<VERSION>-Linux-x86_64.sh
4 root $> cd ~
5 root $> source .bashrc
```

We proceed with the creation and activation of the virtual environment, specifying 3.8 as the version of Python.

Please note that the latest available version is not used but an earlier one for reasons of security, compatibility, stability and robustness.

```
1 root $> conda create -n "venv" python=3.8 ipython
2 root $> conda activate venv
```

You then need to update the pip and setuptools versions:

```
1 (venv) $> python3 -m pip install -U pip
2 (venv) $> python3 -m pip install -U setuptools
```

Finally, the latest stable version of the NVFlare framework needs to be installed:

```
1 (venv) $> python3 -m pip install nvflare
```

6.3 The FL simulator

As mentioned earlier, one of the main features of the NVFlare framework is the ability to run simulations, that is to emulate the behavior of a Federated Learning server and several clients that are connected but actually reside in the local workstation and are nothing more than processes and threads.

This possibility represents a great tool for doing a quick deployment on which perform testing and debugging.

In particular, once the installation is complete, it is possible to access and control the FL simulator via CLI.

It is possible to obtain the list of parameters accepted by the simulator by running the command

```
1 (venv) $> nvflare simulator -h
```

Among the most interesting parameters that appear, it is worth noting those that will be needed shortly, which are essential for the proper execution of the simulation:

- `w` → `WORKSPACE` → to specify the folder dedicated to execution
- `n` → `N_CLIENTS` → to specify the number of clients to be simulated
- `t` → `THREADS` → to specify the number of clients executing the task in parallel.

Finally, to verify the correct installation of the framework, you can run one of the many test applications in the repository, for example **hello-pt-tb**, a simple example of image classification (we will explain in more

detail what this example includes, for now let's just use it as an example to verify the correctness of the previous steps).

Configured the working environment, we need to clone the repository containing the framework and run the job.

Note in the following commands that the **hello-pt-tb** application also needs to install some dependencies for its execution:

- **PyTorch**[65] → open-source machine learning library in Python that provides a wide range of machine learning algorithms and data processing tools and is very useful for working with different neural network architectures and creating machine learning models → a peculiar feature of this library is that it supports parallelization on GPUs, which makes it easier and faster to process large amounts of data
- **Torchvision** → PyTorch-based machine vision library that provides the most popular pre-trained datasets and models for machine vision and therefore greatly simplifies classification, segmentation, object detection and image generation problems
- **TensorBoard**[66] → web-based interface to visualize and analyze via intuitive and interactive graphs the data generated during the training of a machine learning model → more precisely offers graphs of the loss and accuracy and values of model weights in a spatiotemporal manner, which makes it easy to detect any problems or anomalies during the training process of the model → TensorBoard also supports visualization of performance profiling data such as memory consumption and CPU utilization.


```

1 (venv) $> cd Desktop
2 (venv) $> sudo apt install git
3 (venv) $> git clone https://github.com/NVIDIA
  /NVFlare.git
4 (venv) $> mkdir simulator-example
5 (venv) $> cp -rf NVFlare/examples/hello-pt-tb
  simulator-example/
6 (venv) $> python3 -m pip install torch torchvision
  tensorboard
7 (venv) $> mkdir simulator-example/workspace
8 (venv) $> nvflare simulator -w simulator-example/
  workspace -n 2 -t 2 simulator-example/hello-pt-tb

```

The key execution command is the last one, which in fact starts the simulator using two dummy clients using two parallel threads.

Once the command is executed, the directory designated as the workspace will fill with the necessary codes and configurations, as well as the outputs and evolutions of the local and global models.

Taking a more detailed look at the important elements of this directory structure, it can be seen that:

- the **hello-pt-tb** folder contains the configuration files needed to run the job
 - in the file **config_fed_client.json**, in which the structure of the client is described, we note that this is a `LearnerExecutor` (that is a special type of `Executor` that delegates the training task to

- the Learner class for reasons of decoupling and encapsulation, that is to have this class handle overly specific communication constructs or, for example, error handling, thus allowing the end user to concentrate on the training and validation process) and that it has among its tasks `train`, `submit_model`, and `validate`, that is exactly the list of actions that a client can perform
- in the **config_fed_server.json** file instead you can see the decidedly more complex structure of the server → this in fact is the union of several components, among which it is good to mention Persistor (object that helps to save and keep some specific information, so as to increase abstraction), Shareable Generator (component that converts a shareable object into model objects), Aggregator (Federated Learning server component used to accept client contributions and thus aggregate them with previous contributions), Model Locator (deals with finding models to include for cross-site evaluation), JSON Generator (whose purpose is to generate, from the validation results received as a result of the cross-site validation process, a results file.json containing for each client the accuracy of its local model validated by him) and TB Analytics Receiver (component that receives the log data from the clients and writes it to TensorBoard) → also note that the server workflow includes two actions extensively described in the previous sections, namely scatter and gather and cross-site validation
 - in the **custom** folder we find the customizable Python codes → of particular interest turn out to be the files **pt_learner.py** and **simple_network.py**, which we will elaborate later

- also the **meta.json** file contains important information, mainly related to the deployment of the application (minimum number of clients, minimum requirements, etc.)
- instead the **workspace** folder contains all the results obtained from the job execution
 - here are stored all the log files (both of the server and of each client)
 - also provides an additional subdivision into folders to better separate the server (**app_server** folder), client (**app_site-1** and **app_site-2** folders), validator (**cross_site_val** folder) and TensorBoard events (**tb_events**) components.

Understanding well the structure above is crucial because it is repeated for all the application examples in the repository, changing only a few parameters.

We therefore end the explanation of the simulator by going to look at the performance obtained by the two clients during the training process having the **hello-pt-tb** application as its object:

```
1 (venv) $> tensorboard --logdir=simulator-example/  
workspace/simulate_job/tb_events
```

It should be noted, also for subsequent experiments, that the command given above must specify as a parameter the path to reach the **tb_events** folder.

Following the instructions that appeared on the terminal and opening the search browser, it is possible to evaluate the correctness of the two graphs of train loss and accuracy, the characteristics of which have been described in a separate section.

An example of the results obtained is given in figure 6.1 below:

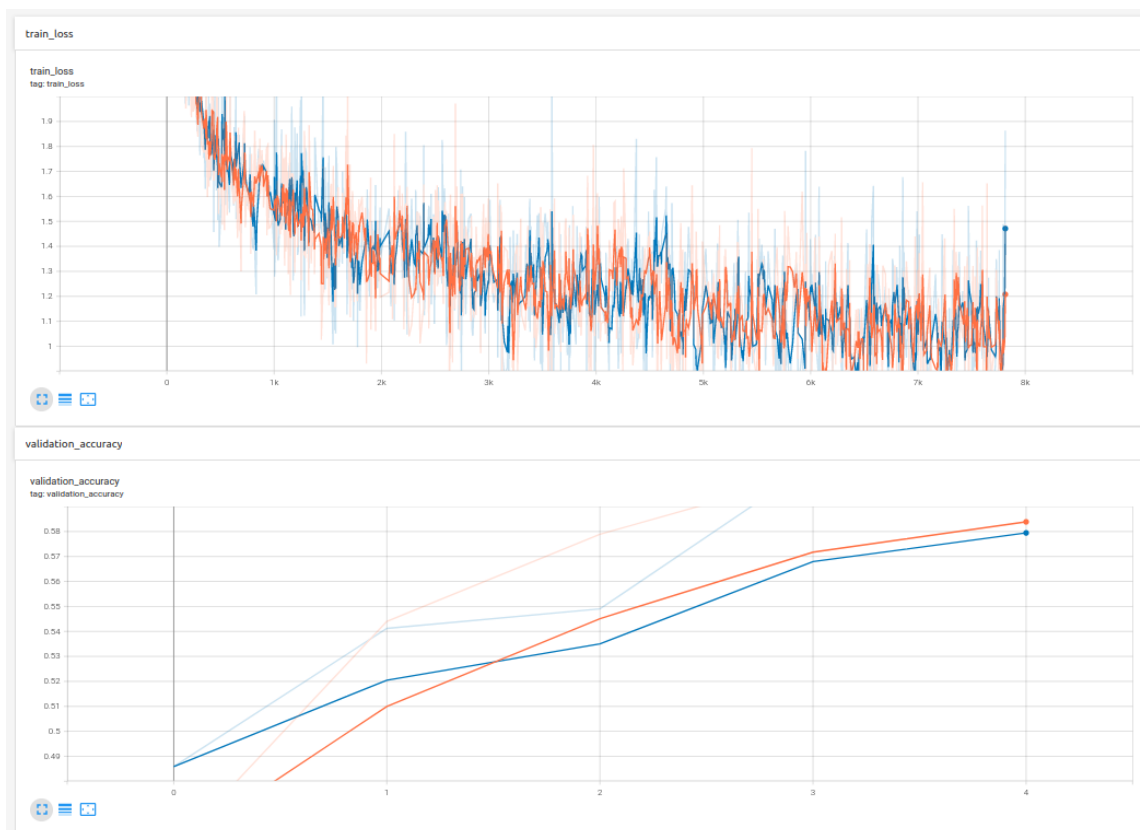


Figure 6.1: loss and accuracy graphs related to the training performance of two models on two clients with training on 5 epochs.

The next section will take a next step toward a more realistic (though still very limited) application, while also taking the opportunity to delve into some additional features of the framework.

6.4 POC FL

The next step is to run an example of Federated Learning in the so-called Proof Of Concept mode, which is a slightly more advanced mode than the simulator seen in the previous section for simulating a deployment.

Also absent in this mode is the Overseer, as well as HA and security mode (all concepts that we will explain later since they will be added as we continue with the experiments).

We will also see examples run on a single machine and not distributed. We will still have processes for the FL server, multiple clients and the admin console, again to invoke a realistic deployment.

From a more technical point of view, NVFlare commands in POC mode enable configuration and execution of training processes without the provisioning steps.

Provision refers to the process of generating and deploying component software packages (called startup kits).

In fact, the various clients and servers may be in different locations and there is therefore a clear need for mutual trust and reliability to communicate properly.

The provisioning process is precisely to generate the SSL certificates used for client authentication, which will be part of the startup kits, which eventually need to be distributed to the different organizations before

starting the Federated Learning process.

In POC mode, on the other hand, the server will communicate with clients and the admin without HTTPS encryption and without the need for SSL certificates.

As mentioned, POC still allows all components to run on the same machine.

To better understand how this architecture works, which is still simulated locally but comes closest to a realistic deployment, let us consider the **CIFAR10** application, which is always present in the application examples that can be found in the framework repository.

6.4.1 CIFAR10

In the present section, the topic of the CIFAR10 dataset is discussed further, as the **CIFAR10** application of the NVFlare package is perhaps one of the most interesting from the educational point of view, in fact it allows experimentation with both realistic and POC deployment, deals with image classification, introduces a more in-depth study of the CIFAR10 dataset and finally anticipates that it is possible to use different algorithms during the same training process, also with the purpose of making comparisons and evaluations.

As mentioned earlier, CIFAR10 is a small 32×32 image dataset, consisting of 60,000 RGB images divided into 10 mutually exclusive object classes, that is with no possibility of overlapping.

This dataset is often used to test the performance of machine learning models on classification and detection problems.

It was developed by the Canadian Institute For Advanced Research to attempt to achieve the best possible performance in image recognition using convolutional neural networks.

In the end, it should be pointed out that there are numerous interesting papers in the literature today claiming to have achieved state-of-the-art results by working with this dataset.

This application uses and therefore also allows for further investigation of other interesting concepts:

- **matched averaging** → optimization technique used in neural networks that consists of maintaining a weighted average of the states of all layers in the optimization of the model, so as to correct for the variance of the parameters that might occur during training → the result is a more stable and predictable model with therefore better performance[67]
- **Dirichlet distribution**[68] → a multivariate probability distribution describing the distribution of probabilities relating to multiple events → is commonly used to model the probability distribution of a category, where the probabilities relating to subgroups are uncertain → it is defined by a set of parameters (alpha) that determine the shape of the distribution
- **homomorphic encryption** → cryptographic technology that makes it possible to operate on encrypted data while maintaining its confidentiality → this means that data can be securely transmitted or processed without the need to decrypt it first, unlike in traditional cryptography, where data must be decrypted before any operation can be performed on it.

Homomorphic encryption explanation is summarized in figure 6.2.

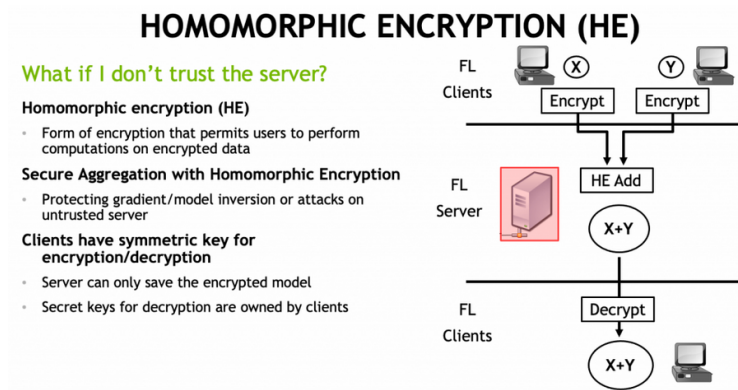


Figure 6.2: summary diagram on how homomorphic encryption works.

In fact, this application within the framework uses an algorithm to implement heterogeneous generation of data splits from CIFAR10 based on the Dirichlet distribution, where the alpha parameter expresses the heterogeneity of the distribution of data splits.

To better understand how POC FL works, we execute the job named just CIFAR10 in POC mode.

Let us remember that we are still at a rather simulative level which is, however, very useful for understanding the structure and operation of the system.

First, it is necessary to configure the virtual environment so that it meets some requirements:


```
1 (venv) $> cd Desktop
2 (venv) $> cp -rf NVFlare/examples/cifar10 .
3 (venv) $> cd cifar10/
4 (venv) $> pip install -r ./cifar10-real-world/
    virtualenv/min-requirements.txt
5 (venv) $> pip install -r ./cifar10-real-world/
    virtualenv/plot-requirements.txt
```

Specifically, these two commands have installed additional libraries and dependencies that are secondary but necessary for the experiments that will follow, including seaborn (to visualize data in Python via a simple but complete graphical interface that also allows the creation of advanced and appealing statistical graphs) and pandas (to have specific data structures and functions for the manipulation and analysis of tabular and time series data, as well as methods for data cleaning, filtering, aggregation and manipulation of missing data).

We continue by downloading the dataset:

```
1 (venv) $> python3 ./pt/utils/
    cifar10_download_data.py
```

Having reached this point, it is necessary to open the `/cifar10/cifar10-real-world/job_configs/cifar10_fedavg_stream_tb/meta.json` file and, if necessary, modify the minimum requirements for each client to suit your possibilities.

By default the file assumes you have 8 clients each with GPUs of at least 1 GB memory.

If you do not have any GPUs, you cannot perform the following experiment, but to get an idea of how Federated Learning works in POC mode anyway you can skip to the next section, which shows how to run the CIFAR10 application in a secure workspace even without GPUs, performing the training and testing operations on CPU (resulting in a considerable lengthening of time).

From the terminal we continue with the present example by running the following commands that, in a very simulator-like manner, will create a FL structure with 8 different clients but in POC mode:

```
1 (venv) $> nvflare poc --prepare -n 8
2 (venv) $> export PYTHONPATH=\${PWD}/..
3 (venv) $> nvflare poc --start
```

In particular, the first command will also create a temporary folder used as a workspace and provide its path.

This folder has identical structure to the one seen in the simulator case but it will contain a larger number of clients.

Executed the last command, we will see the 8 clients that will gradually begin to connect to the server and finished this procedure the terminal will change to be the CLI dedicated to the admin, from which we can run the main commands to manage the progress of the job (job submission, information request, etc.).

At this point, we need to open an additional terminal that will be used to run the script that sends the application into execution:

```
1 (venv) $> conda activate venv
2 (venv) $> cd cifar10-real-world/
3 (venv) $> ./submit_job.sh cifar10_fedavg_stream_tb
    1.0 --poc
```

In doing so, we ran the `cifar10_fedavg_stream_tb` application in POC mode.

Note that the parameter specified with value 1.0 corresponds to the alpha explained earlier concerning the heterogeneity of the Dirichlet distribution regarding the data splits of the dataset.

Having executed the command, the terminal will provide us with the id of the job (which is a unique code suitable to identify it, for example 4524151e-8be7-4440-826f-641a9d421551) and thanks to this, once the job is finished, it is possible to download the results from the previous terminal (the one showing the admin CLI) just by specifying the id there:

```
1 > download_job <ID>
```

To better understand how to use the results obtained and possibly graph them, refer to the final part of the following subsection.

6.4.2 Secure workspace

In order to solve the problem seen above of the technical requirements of nodes (that is the need for them to possess a GPU), it is possible to resort to the so-called secure workspace, which is an execution environment that is more flexible with regard to these requirements.

Well, it should be specified that actually the main feature of the secure workspace is that it is an execution environment with more security and therefore considered more realistic.

In fact, using this mode increases the reliability and confidentiality of communications since SSL/TLS encryption certificates will be generated during the provisioning phase and will actually be used to encrypt transactions (which is exactly what is needed in a real-world deployment).

Finally, the secure workspace also allows you to generate the keys for the homomorphic encryption explained in the previous subsection.

To start working in the secure workspace, you can preliminarily download the dataset in order to speed up the following experiments.

Once you are placed in the directory of the application you want to start (in the following case **cifar10/cifar10-real-world**), simply run:

```
1 (venv) $> python3 ../pt/utils/  
cifar10_download_data.py
```

Having reached this point, you can continue with secure provisioning, which is done by submitting to the system a `.yaml` file containing the outline structure of the FL network you intend to create (by default, you can specify the **secure_project.yaml** file, which is already present in the framework):

```
1 (venv) $> cd cifar10-real-world/
2 (venv) $> cd ./workspaces
3 (venv) $> nvflare provision -p ./secure_project.yml
4 (venv) $> cp -r ./workspace/secure_project/prod_00
   ./secure_workspace
5 (venv) $> cd ..
```

At this point, taking advantage of the potential offered by the secure workspace and its improved flexibility, it is necessary to act on the specifications required of the nodes, so as to eliminate the requirement on the mandatory presence of GPUs.

To do this, it is sufficient to modify the file

cifar10/cifar10-real-world/job_configs/cifar10_fedavg_stream_tb/meta.json as shown in figure 6.3 by going to remove the "resource_spec" section, obtaining a file with contents similar to the one shown below:

```
1 {
2   "name": "cifar10_fedavg_stream_tb_alpha1.0",|
3   "deploy_map": {
4     "cifar10_fedavg_stream_tb": [
5       "@ALL"
6     ]
7   },
8   "min_clients": 8
9 }
```

Figure 6.3: meta.json file modified for use on even non-GPU nodes.

Having completed this minor modification to the initial configurations,

you can start the Federated Learning system in safe mode and submit the job via CPU:

```
1 (venv) $> ./start_fl_secure.sh 8
2 (venv) $> ./submit_job.sh cifar10_fedavg_stream_tb
    1.0
```

Note again that 8 indicates the number of clients, while 1.0 is the alpha parameter explained earlier.

Having executed the first command we will see the creation of the infrastructure and thus the initialization of the server and the connection of the clients.

The second command activates the execution of training on the clients and a long exchange of validation information with the server.

Once the process is finished, it is possible to observe the loss and accuracy graphs in a similar way as before, that is by accessing the admin terminal via the appropriate folder, downloading the job via the **download_job <JOB_ID>** command, and graphing the TensorBoard events via the appropriate **tensorboard - -logdir** command specifying the path to reach the **tb_events** folder.

6.5 Real world FL on single host

Once the potential of the secure workspace has been explained, we can proceed to study the most advanced possible case of FL deployment but still on a single node.

In fact, as is also the case in a realistic deployment, a chosen node con-

ducts the infrastructure configuration process (that is provisioning) and thus is tasked with generating the startup kits for every other node that is to be used for the FL project.

We therefore proceed to create a new working directory (let's call it `rw` for example), which in fact can be seen as the file system of this special node.

Within the newly created directory we insert the NVFlare folder, obtained as a result of cloning the framework's GitHub repository, and within the same folder (that is `rw`, not NVFlare) we execute the provisioning command, so as to define the structure of the FL network we intend to establish:

```
1 (venv) $> nvflare provision
```

The terminal will warn us that it has not found any suitable provisioning file in the current folder and will therefore generate a default one, waiting for our choice based on whether we want a provisioning file for HA mode or not.

In short HA stands for High Availability and it is a more reliable mode since it also provides for a node to play the role of Overseer.

While the classic roles of Client, Server and Admin perform exactly the tasks specified by their names and therefore it is possible to omit a description of them, the role of the Overseer may not be so clear, so it is good to make it explicit.

An Overseer is a node that is responsible for monitoring the behavior of the FL server.

In HA mode, at least two servers are indeed required (the second one as

a backup).

Well, the Overseer is the node that specifies to the admin which server to connect to in case of problems: in fact, if a server becomes unreachable in the middle of the Federated Learning process, the Overseer will be the one to have clients automatically point to the secondary server.

Therefore, the Overseer is a role attributable to a node only if HA mode is active.

For the purposes intended by this practical proof, we can also choose the non-HA mode and continue.

Pressed then the button corresponding to our choice, the terminal alerts us to the creation of the provisioning file **project.yml**.

It is possible to edit that file to adapt it to the configuration we need.

Trying to open the file that has just been created, you can spot an interesting section: that of the participants, shown in figure 6.4.


```

1 api_version: 3
2 name: example_project
3 description: NVIDIA FLARE sample project yaml file
4
5 participants:
6   # change example.com to the FQDN of the server
7   - name: server1
8     type: server
9     org: nvidia
10    fed_learn_port: 8002
11    admin_port: 8003
12   - name: site-1
13     type: client
14     org: nvidia
15   - name: site-2
16     type: client
17     org: nvidia
18   - name: admin@nvidia.com
19     type: admin
20     org: nvidia
21     role: project_admin
22
23 # The same methods in all builds are called in their order defi:

```

Figure 6.4: participants section of the **project.yml** file, which is useful for understanding and modifying the structure you are going to create.

Indeed, from the image it can be seen that the default generation has created an infrastructure consisting of a server, two clients and an admin. For each node, additional information is then specified, such as the organization to which it belongs or the communication port (which we do not suggest changing).

Then note that the admin name represents a generic username that will be useful in the setup phase of the node having the admin role (in fact, for reachability reasons, an e-mail address must be specified).

Once you have finished studying and editing the provisioning file, you can proceed by running the command to actually create the structure

and startup kits:

```
1 (venv) $> provision -p project.yml
```

Following this command, a workspace folder is created, containing among others the **prod_00** folder, which in turn contains a folder for each node specified in the **project.yml** file.

These folders are the basic constituents that each node must possess in order to perform their designated role within the network structure designed to perform FL properly.

So we will have one folder for the admin and as many folders for server and client as specified when editing the **project.yml** file (plus possibly a folder for the Overseer, if any).

A distinction should be made here in particular:

- in a realistic deployment, all the folders described above would have to be properly distributed to the various nodes (the folder **site-1** to the first client, the folder **server1** to the server, etc.) → until the last version of the framework such distribution provided for folder compression and folder extraction only via password, so as to increase the confidentiality of this deployment, however this feature has recently been removed and now this process is easier but also less secure → in a real deployment one must therefore adopt proper security and encryption techniques to solve the problem, as this undoubtedly represents a very vulnerable point to attacks and intrusions
- on the other hand, as far as the case under consideration with this

experiment is concerned, since it is still the use of a single node, the security issues expressed above do not exist, however a preliminary step must be taken.

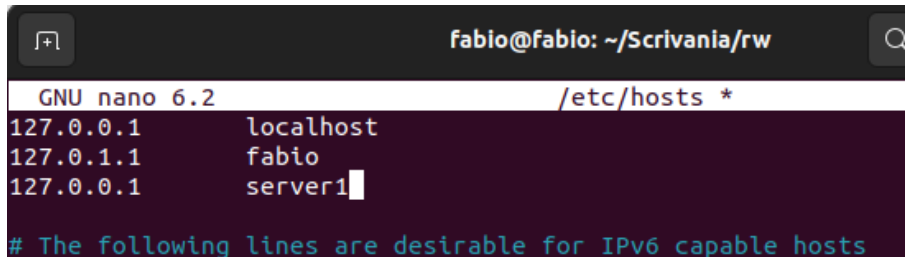
Indeed, it is necessary, since this is a centralized architecture, to associate the name of the server with the machine being used (thus localhost) so that the machine itself, when performing client operations, does not attempt to resolve the name conventionally via DNS.

To do this, it is necessary to modify the system file `/etc/hosts`, which is used as the primary source for mapping IP addresses to host names in order to provide an alternative name resolution mechanism to name resolution services using DNS.

To do this, the write permissions of that file must first be changed:

```
1 (venv) $> sudo chmod 777 /etc/hosts
2 (venv) $> sudo chmod 777 /etc
3 (venv) $> nano /etc/hosts
```

We therefore also enter the association between server1 and localhost in the list that appears, as shown in figure 6.5:

A screenshot of a terminal window showing the /etc/hosts file being edited with nano. The terminal title is 'fabio@fabio: ~/Scrivania/rw'. The nano header shows 'GNU nano 6.2' and the file path '/etc/hosts *'. The file content is as follows:

```
127.0.0.1 localhost
127.0.1.1 fabio
127.0.0.1 server1
```

The cursor is positioned at the end of the 'server1' line. A comment at the bottom of the file reads: '# The following lines are desirable for IPv6 capable hosts'.

Figure 6.5: `/etc/hosts` file as a result of adding the association between the name `server1` and the localhost ip address.

Note that if the `project.yml` file has been modified, for correct resolution of server names it is necessary not only to add an association line for each server present, but also that the name specified is correct (in fact, if the name has been modified when editing the `project.yml` file it may no longer be "server1" and in that case it is therefore necessary to enter the name chosen).

Note also that when editing the `project.yml` file it is not possible to assign the same name to two different nodes, otherwise one would not know how to resolve that name, that is which association to refer to.

Finally, note that only the association between server and IP should be specified in the `/etc/hosts` file and not also the association between client and IP, this is for logical reasons: since the client's IP could also vary, it would be useless to specify a static one in this file, in fact unlike the server it is not important that the client node has a certain IP address, it is enough only that it has the correct cryptographic certificates to establish communication with the server, also because it is the client itself that requests to communicate to the server, which in this way is aware of the client's IP address anyway.

At this point, simulating that you have spread the various folders to the right nodes, you can place the folder of the jobs you intend to submit inside the transfer folder of the folder reserved for the admin (in the following example we will submit **hello-pt-tb** and therefore copy the appropriate folder by taking it from **rw/NVFlare/examples**).

At this point you can open a terminal for each component (the recommended order is first the overseer if present, then all servers, then all clients and only at the end the admin) and on each terminal run the **startup/start.sh** script (pay attention to the fact that actually the script contained in the admin's folder is called **fl_admin.sh**, so run this).

In this way it is possible to understand what happens even in the case of truly distributed nodes when connecting and creating the network:

- started the server, it is waiting to receive new connections
- for each client started this will register correctly to the server → in the server terminal it is in fact possible to see that a new client has been added, reporting also its IP address and a unique token assigned to it
- finished the registration of all clients it is possible to register also the admin by entering the username, which corresponds to the email entered when editing the provisioning file → the terminal reserved for the admin is a fundamental constituent, since it is able to command the FL processes, that is to submit jobs or obtain information about the status of other nodes or previously submitted jobs.

For example, by running the commands `check_status server` and `check_status client` it is possible to check the status of the nodes and thus verify the correctness of the steps so far.

The expected output is depicted in figure 6.6.

```
> check_status server
Engine status: stopped
-----
| JOB_ID | APP NAME |
-----
Registered clients: 2
-----
| CLIENT | TOKEN | LAST CONNECT TIME |
-----
| site-1 | 4a359480-7003-41b2-890f-f769c37d48c0 | Wed Jan 25 11:47:41 2023 |
| site-2 | dc954ec3-d457-403a-b712-9f874c965e75 | Wed Jan 25 11:47:39 2023 |
-----
Done [77361 usecs] 2023-01-25 11:48:09.718377
> check_status client
-----
| CLIENT | APP_NAME | JOB_ID | STATUS |
-----
| site-1 | ? | ? | No Jobs |
| site-2 | ? | ? | No Jobs |
-----
Done [485409 usecs] 2023-01-25 11:48:28.334527
>
```

Figure 6.6: execution of commands to check the status of servers and clients.

It is then finally possible, again in the admin’s terminal, also to submit the job through the `submit_job <FOLDER_NAME>` command, where the folder name refers to the name of the folder inserted inside the admin’s own transfer folder.

Submit the command, in the various terminals we can observe the following events:

- in the server terminal it is possible to see that he receives from

the scheduler the job to execute and specifies a scatter and gather workflow to start the training process

- on the other hand each client pulls the job folder from the server and proceeds with the training process, of which it is possible to see the progression in epochs and the evolution of the loss parameter which, as extensively mentioned in the previous sections, should decrease as time progresses.

Again, loss and accuracy graphs can be studied using TensorBoard by having it pointed via **logdir**:

- to the **tb_events** folder contained in the server folder if the job is still running
- otherwise to the **workspace/tb_events** folder contained in the folder produced as a result of downloading the final results of the job from the admin terminal with the **download_job <JOB_ID>** command.

Having finished this experiment, we can finally move to a distributed infrastructure, in which the various client and server nodes are truly remote, networked nodes.

6.6 Distributed real world FL

The purpose of this section is to move from the examples seen above centralized to truly distributed training over the network.

First we need to restore the **/etc/hosts** file to its original content, so

if it has been modified for the previous experiments we need to remove those modifications.

As in the previous section, we then provision the application in a working directory by running the command

```
1 (venv) $> nvflare provision
```

This time, however, we choose from the possible options number 1, which is the one related to the high reliability provisioning mode. In our case, let us assume that we want to create a network infrastructure having:

- one FL server having IP 137.204.72.9
- two clients, understood as two different processes on the same machine having IP 137.204.72.37
- one admin and one overseer, understood as additional processes on the server machine.

Well, we need to modify the **project.yml** file to match the specifications we are interested in.

In particular, we need to change the section "participants" and "overseer_agent" as shown in the following figures (6.7 and 6.8):


```

5 participants:
6 - name: overseer
7   type: overseer
8   org: biolab
9   protocol: https
10  api_root: /api/v1
11  port: 8443
12 - name: 137.204.72.9
13   type: server
14   org: biolab
15   fed_learn_port: 8002
16   admin_port: 8003
17 - name: site-1
18   type: client
19   org: unibo
20 - name: site-2
21   type: client
22   org: unibo
23 - name: admin@nvidia.com
24   type: admin
25   org: nvidia
26   role: project admin

```

Figure 6.7: note the changes to the server name parameter and the org parameter of all participants.

```

57 overseer_agent:
58   path: nvflare.ha.dummy_overseer_agent.DummyOverseerAgent
59   overseer_exists: false
60   args:
61     sp_end_point: 137.204.72.9:8002:8003

```

Figure 6.8: the overseer section, on the other hand, needs to be modified more.

Once these changes to the file are finished, you can proceed to provisioning and thus creating the startup kits of all participants by repeating the command

```

1 (venv) $> nvflare provision

```

A few things need to be made sure before going any further:

- the server and clients must have the same version of NVFlare → you can verify the version number by running the command `conda list`
- the ports 8002 and 8003 of the server must be free, that is not listening → you can verify this by running the command `sudo lsof -i -P -n | grep LISTEN` and making sure that the ports of our interest do not appear in the list
- any active firewall rules must allow communication between clients and server.

Once all these checks have been made, it is possible to distribute the startup kits (that is the subfolders in `./workspace/example_project/prod_00`) to the respective nodes, that is the **site-1** and **site-2** folders must be sent to the client (in our case this was done using the `scp` command).

The next step is to copy the folder of the job you intend to run (that is **hello-numpy-sag**) within the subfolder `admin@nvidia.com/transfer`.

It is necessary at this point to perform an additional verification that within this folder:

- in the file `hello-numpy-sag/meta.json` the parameter "min_clients" is compatible with our architecture, otherwise change it
- same thing in the file `hello-numpy-sag/app/config/config_fed_server.json`.

After finishing this final check, exactly as in the previous section you can proceed to run the `start.sh` script on overseer, server and client and the

fl_admin.sh script on the admin node and in the CLI that will open submit the job (**hello-numpy-sag**) through the **submit_job hello-numpy-sag** command.

Note that after the training is finished, it may be necessary to run, via the admin's CLI, the client **shutdown** command.

This allows clients to be disconnected from the server, which is useful to prevent client processes from remaining active and in communication with the server even when not needed.

6.7 FL for morphing

This section represents the closure of the chapter on programming guide. In fact, this encapsulates the notions and knowledge learned above and explains how they have been modified and adapted in order to conduct training of a model doing face morphing attack detection via Federated Learning implemented with NVFlare.

In particular, compared to the previous section, some aspects remain to be clarified and certain problems solved in order to achieve the goal of training a model for MAD using Federated Learning:

- assign each client a different dataset among those available for morphing
- set some hyperparameters such as learning rate or number of epochs
- in the presence of clients belonging to the same physical machine, assign each one a different GPU

- implement a truly distributed training process, that is in which client, admin and server do not reside on the same machine but must communicate over the network.

So let us see how to proceed to solve the problems raised above, distinguishing the steps to be taken between client and server to properly configure and start the training with the desired settings.

6.7.1 Client-side FL for morphing

Remember to perform the following steps in a virtual environment (for example using Anaconda):

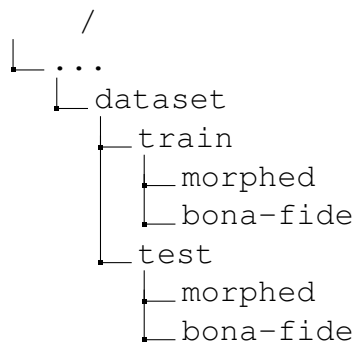
1. install the NVFLARE package →

```
1 (venv) $> python3 -m pip install nvflare
```

2. install utility libraries →

```
1 (venv) $> python3 -m pip install torch  
torchvision tensorboard facenet-pytorch tqdm
```

3. organize the available dataset in a manner similar to that reported →



4. notify the server administrator of the absolute path to the dataset folder
5. notify the server administrator on which GPU of the client machine you intend to run the training → it is suggested to use the GPU labeled as the result of the command

```
1 (venv) $> nvidia-smi
```

6. request from the server administrator the startup kit (folder) dedicated to the client
7. once obtained the folder wait until the server is properly started and listening
8. move to the **startup** folder in the startup kit received and execute the script →

```
1 (venv) $> ./start.sh
```

6.7.2 Server-side FL for morphing

The following steps must also be performed in a virtual environment (for example Anaconda):

1. install the NVFLARE package →

```
1 (venv) $> python3 -m pip install nvflare
```

2. install utility libraries →

```
1 (venv) $> python3 -m pip install torch  
torchvision tensorboard
```

3. clone the github repository and create a folder as workspace →

```
1 (venv) $> git clone  
https://github.com/NVIDIA/NVFlare.git  
2 (venv) $> cd NVFlare  
3 (venv) $> git switch 2.2  
4 (venv) $> mkdir ws
```

4. within the workspace folder created in the previous step run the provisioning command and select option number 1 (the one referring to provisioning in HA mode, high reliability) →

```
1 (venv) $> nvflare provision
```

5. edit the **project.yml** file created as a result of the previous command by specifying the architecture you intend to implement → specifically, in this file you need to add, remove and change the parameters referring to the nodes you plan to include in the Federated Learning network you want to implement → certainly you need to change the parameters shown in figure 6.9 and 6.10 →

```
5 participants:
6 - name: overseer
7   type: overseer
8   org: biolab
9   protocol: https
10  api_root: /api/v1
11  port: 8443
12 - name: <IP>
13   type: server
14   org: biolab
15   fed_learn_port: 8002
16   admin_port: 8003
17 - name: site-1
18   type: client
19   org: unibo
20 - name: site-2
21   type: client
22   org: unibo
23 - name: admin@nvidia.com
24   type: admin
25   org: nvidia
26   role: project_admin
```

Figure 6.9: in the participants section replace <IP> with the IP address of the node designated as the server and add/remove clients according to the network configuration you want to create.

```
57     overseer_agent:
58         path: nvflare.ha.dummy_overseer_agent.DummyOverseerAgent
59         overseer_exists: false
60         args:
61             sp_end_point: <IP>:8002:8003
```

Figure 6.10: in the section for `overseer_agent` carefully change the parameters exactly as shown in the figure, also taking care to replace `<IP>` with the IP address of the server.

- finished the above file changes, run the provisioning command again to generate the startup kits →

```
1 (venv) $> nvflare provision
```

- move to the `./workspace/prod_00` folder and distribute the various startup kits to the respective components (that is the first client should receive the **site-1** folder and so on)
- copy in the **transfer** folder of the startup kit dedicated to the admin the folder of the training job that you intend to submit → if you do not have your own job, it is suggested to follow the steps below, copying and editing one of the examples in the **NVFlare/examples** folder, generated as a result of cloning the repository (step 3)
- request from the various clients the absolute path to their datasets
- within the folder transferred in step 8 modify the file **app/custom/pt_learner.py** in a manner similar to figure 6.11 below →


```

90     client_name = fl_ctx.get_identity_name()
91     if client_name == "site-1":
92         self.train_dataset = datasets.ImageFolder(root="PATH/TO/TRAIN/FOLDER/CLIENT1", transform=transforms)
93         self.train_loader = DataLoader(self.train_dataset, batch_size=32, shuffle=True)
94         self.n_iterations = len(self.train_loader)
95
96         # Create CIFAR10 dataset for validation.
97         self.test_data = datasets.ImageFolder(root="PATH/TO/TEST/FOLDER/CLIENT1", transform=transforms)
98         self.test_loader = DataLoader(self.test_data, batch_size=32, shuffle=False)
99         print("Loaded dataset A")
100     elif client_name == "site-2":
101         self.train_dataset = datasets.ImageFolder(root="PATH/TO/TRAIN/FOLDER/CLIENT2", transform=transforms)
102         self.train_loader = DataLoader(self.train_dataset, batch_size=32, shuffle=True)
103         self.n_iterations = len(self.train_loader)
104
105         # Create CIFAR10 dataset for validation.
106         self.test_data = datasets.ImageFolder(root="PATH/TO/TEST/FOLDER/CLIENT2", transform=transforms)
107         self.test_loader = DataLoader(self.test_data, batch_size=32, shuffle=False)
108         print("Loaded dataset B")

```

Figure 6.11: image shows how to delete the default DataLoader and replace it with one that can accept the different paths referred to the various clients.

11. request clients on which GPU they prefer to start training → once the response is received, edit the `pt_learner.py` file similarly to figure 6.12 below:

```

def initialize(self, parts: dict, fl_ctx: FLContext):
    client_name = fl_ctx.get_identity_name()

    if client_name == "site-1":
        self.device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    if client_name == "site-2":
        self.device = torch.device("cuda:1" if torch.cuda.is_available() else "cpu")
    print('Running on {}'.format(self.device))

```

Figure 6.12: image shows how and where to specify for each client the GPU you prefer to devote to the training process.

12. it is also possible to change other important parameters in the file `pt_learner.py` → their use is beyond the scope of this report, so

the exact code for making the changes is not given, but it is still considered useful to mention the possible changes, in particular:

- it is possible to specify a different model than the default one simply by reassigning the variable `self.model` → in case of a change to the model it is also necessary to change the path parameter in the `config_fed_server.json` file as shown in figure 6.13 below

```
9  "components": [
10     {
11         "id": "persistor",
12         "path": "nvflare.app_common.pt.pt_file_model_persistor.PTFileModelPersistor",
13         "args": {
14             "model": {
15                 "path": "<NEW_MODEL>"
16             }
17     }
```

Figure 6.13: replace the previous template with the new one, having previously placed the Python script containing it in the `app/custom` folder.

- it is also possible to specify a different dataset loader by defining and using your own custom class
 - is also possible to specify a different learning rate through the variable `self.lr`
 - finally, it is also possible to specify a different loss function by reassigning the variable `self.loss`.
13. at this point, having finished editing the `pt_learner.py` file, before proceeding you need to make sure that:
 - all components (clients and servers) are using the same version of NVFlare

- ports 8002 and 8003 on the server are available, not already listening
 - any firewalls present allow communication between all components
 - in the file **meta.json** present in the folder specified in step 8 the parameter "min_clients" is compatible with the architecture of the network you want to implement (for example if you have two clients available this parameter must be set to 2) → perform the same check also in the file **./app/config/config_fed_server.json**
 - in the file **./app/config/config_fed_client.json** the desired number of epochs is specified.
14. if the network architecture includes an overseer, move to its startup kit and run the script **./startup/start.sh**
 15. open a new terminal, move to the server's startup kit, run the **./startup/start.sh** script and alert clients that the server is finally listening → executing this step may result in the terminal failing to start the server due to a process already listening on the specified port (this is often due to an incorrect server shutdown that occurred earlier), so to resolve it you need to run the following commands and unfortunately also restart the server machine, so that the process is safely and permanently eliminated

```
1 (venv) $> kill <PID>
2 (venv) $> rm ../daemon_pid.fl
```

16. wait for the various clients to connect to the server
17. open a new terminal, move to the admin startup kit, run the script `./startup/fl_admin.sh`, enter the credentials and inside the terminal that will open run the command `check_status server` to verify the correct connection of the clients to the server and finally run the command `submit_job <FOLDER>` to start the training process → obviously replace <FOLDER> with the name of the folder copied in step 8
18. once the training process is finished, you can download the folder containing the results via the `download_job <JOB_ID>` command, always run in the admin terminal → obviously replace <JOB_ID> with the ID of the job whose results you intend to download, which can be found by running the `list_jobs` command
19. you can use TensorBoard to show learning curves by running the command `tensorboard - -logdir=<PATH>`, where <PATH> is the path pointing to the `tb_events` folder contained in the folder downloaded in the previous step.

Chapter 7

EXPERIMENTAL RESULTS ON MORPHING ATTACK DETECTION

7.1 Trainings conducted and results obtained

With the skills and knowledge learned from studying the NVFlare framework, it was possible to perform the training of 11 different models capable of doing Face Morphing Detection.

It was decided to set up the training of the various models based on various combinations of the available datasets, so that subsequently we could evaluate how the different distributions of datasets affected the accuracy of the models.

It is possible to briefly summarize the composition of the datasets used:

- **UBO** → 2216 bona fide images and 1108 morphed images, appropriately divided between training (2326), testing (665) and validation

(333)

- **OPENCV, FACEMORPHER, STYLEGAN** → 5405 bona fide images and 2715 counterfeit images, appropriately divided between training (5685), testing (1623) and validation (812).

Table 8.1 summarizes the composition of the different experiments:

ID	CLIENT N°	EPOCHS	DATASET
ID1	1	48	UBO
ID2	1	14	OPENCV
ID3	1	25	FACEMORPHER
ID4	1	30	STYLEGAN
ID5	2	31	UBO/OPENCV
ID6	2	37	UBO/FACEMORPHER
ID7	2	39	UBO/STYLEGAN
ID8	2	20	OPENCV/FACEMORPHER
ID9	2	22	OPENCV/STYLEGAN
ID10	2	28	FACEMORPHER/STYLEGAN
ID11	4	30	UBO/OPENCV/ FACEMORPHER/STYLEGAN

Table 7.1: Composition of the different experiments.

Note that the number of epochs for each training was chosen unconventionally.

Remember that the number of epochs is a fundamental hyperparameter, which heavily affects the accuracy of the model, so normally one would have to identify the most suitable one by mathematical estimation or by several practical attempts.

Since the purpose of this thesis was never the best possible training of a model capable of doing MAD but rather to study and evaluate the potential of the Federated Learning framework, it was deemed sufficient to adopt as the number of epochs optimal values used in previous successful trainings with the same datasets (without the use of Federated Learning). Obviously, these values were then manipulated for experiments with mul-

multiple clients and therefore datasets, using the arithmetic mean of the number of epochs used for individual instances of the datasets involved.

For example, the number of epochs in experiment ID5 (UBO/OPENCV) is 31 because this corresponds to the average of the epochs used in trainings ID1 (UBO only) and ID2 (OPENCV only).

It is considered important to reiterate that this is by no means an optimal nor recommended choice, however it was deemed acceptable given the purposes of this report.

Once the 11 models had been trained, the next step of validating each was done through a framework known as Revelio.

More precisely, Revelio extracts a set of datasets for MAD and begins a so-called warmup phase, that is for each image in each dataset it crops the face so that they all have approximately the same size.

Then the framework takes the trained model submitted to it and validates it on the previously prepared images by returning in a .json file the value of the metrics, already divided by dataset, which can be specified in a special configuration file.

In this way, the results and metrics of the model can be conveniently obtained on the various testing datasets.

With regard to the results of the conducted trainings, we report a few tables containing not the classical accuracy and loss results, but the usual metrics dedicated to face morphing and described in the previous chapters, namely EER and BPCER@APCER (computed at different APCER values).

It is also necessary to point out that in addition to the classic MAD metrics, an additional metric called WAED has also been included: this is

not a conventional nor even unified measure, however it is very useful because with a single value calculated on the basis of all those in the table but weighted by importance it is possible to summarize and describe the behavior of the model and this greatly simplifies the comparison between the results obtained from the different models.

Finally, it should be noted that the tables of each training are not shown but only a few representative cases that show the variation of metrics in the presence of multiple clients and therefore multiple training datasets. In particular, the tables for the trainings are shown:

- ID1 → single client trained on UBO
- ID2 → single client trained on OPENCV
- ID5 → two different clients trained separately on UBO and OPENCV
- ID11 → four different clients trained separately on UBO, OPENCV, FACEMORPHER and STYLEGAN.

MORPHING ALG.	EER	BPCER_{0.1}	BPCER_{0.05}	BPCER_{0.01}
UBO	.000	.000	.000	.000
OpenCV	.145	.188	.248	.367
FaceMorpher	.123	.145	.207	.307
StyleGAN	.210	.317	.394	.688
AMSL	.006	.000	.000	.100
Webmorph	.300	.450	.550	.750
FaceFusion	.199	.349	.508	.800
NTNU	.136	.221	.396	.757
UTW	.134	.193	.329	.748
WAED ↓			.3886	

Table 7.2: ID1 training results (UBO).

MORPHING ALG.	EER	BPCER_{0.1}	BPCER_{0.05}	BPCER_{0.01}
UBO	.044	.023	.029	.072
OpenCV	.004	.000	.000	.000
FaceMorpher	.006	.000	.000	.006
StyleGAN	.070	.036	.130	.537
AMSL	.001	.000	.000	.000
Webmorph	.222	.500	.600	.950
FaceFusion	.158	.284	.460	.773
NTNU	.147	.245	.485	.806
UTW	.375	.663	.763	.899
WAED ↓			.4614	

Table 7.3: ID2 training results (OPENCV).

MORPHING ALG.	EER	BPCER_{0.1}	BPCER_{0.05}	BPCER_{0.01}
UBO	.000	.000	.000	.000
OpenCV	.140	.168	.244	.376
FaceMorpher	.120	.137	.210	.367
StyleGAN	.186	.287	.418	.627
AMSL	.050	.050	.050	.050
Webmorph	.321	.600	.650	.850
FaceFusion	.187	.337	.461	.730
NTNU	.101	.107	.266	.622
UTW	.249	.452	.593	.844
WAED ↓			.4241	

Table 7.4: ID5 training results (UBO/OPENCV).

MORPHING ALG.	EER	BPCER_{0.1}	BPCER_{0.05}	BPCER_{0.01}
UBO	.000	.000	.000	.000
OpenCV	.138	.172	.219	.444
FaceMorpher	.103	.109	.193	.337
StyleGAN	.206	.301	.406	.737
AMSL	.012	.000	.000	.050
Webmorph	.250	.650	.800	.900
FaceFusion	.208	.432	.593	.822
NTNU	.172	.362	.603	.868
UTW	.144	.233	.478	.823
WAED ↓			.4106	

Table 7.5: ID11 training results (UBO/OPENCV/FACEMORPHER/STYLEGAN).

The results obtained show that unfortunately, contrary to expectation, performance does not improve by training the model on different datasets but rather even worsens, so that paradoxically the best model obtained (ID1) falls among those trained on a single dataset.

These unexpected results may actually also be caused by the fact that some important hyperparameters such as learning rate and number of epochs were reused from previous experiments carried out through classical Deep Learning instead of Federated Learning, whereas instead one should identify what the optimal values are and use them.

However, although the results obtained sometimes deviate from those achieved by classical models for MAD, surely it would be reductive to evaluate only these numbers and not to continue further with the experiments, also because we repeat that many important corrective actions could be taken in the training phase that instead were not taken as they were too far from the objectives of this thesis.

Certainly such actions would lead to improved performance: however, it is difficult to estimate how much impact they might have on performance.

In any case, it has already been ascertained in several areas that the potential offered by Federated Learning can truly be a factor that can solve many problems and revolutionize classical machine learning approaches, so this avenue deserves to be explored more thoroughly through new, more accurate and extensive tests, so as to have a more realistic estimate of how Federated Learning affects the morphing detection problem.

On the other hand, as far as the purposes of this thesis are concerned, it is reiterated that among them there has never been to identify a competitive

model for MAD, so the training of the 11 models mentioned above must be evaluated as positive as it represents the most complex and complete experiment that it has been possible to carry out using the framework under study, and this certainly represents an important achievement.

7.2 Overall evaluation of the framework

A table follows detailing all the advantages and disadvantages, strengths but also weaknesses that have emerged from using the NVFlare framework.

Table 8.6 is in fact the answer to the main question of this thesis, which is whether it is possible to explore and exploit the potential offered by a Federated Learning framework to apply it to the Morphing Attack Detection problem.

While the answer may seem obvious, since in fact this has been successfully achieved and already reported in the previous section, NVFlare is a very broad framework with numerous peculiarities, both positive and negative, that deserve attention as they greatly affect its effectiveness and intuitiveness.

ADVANTAGES	DISADVANTAGES
Very detailed user guide	Significant difficulty in use due to lack of intuitiveness of certain passages
Support team available for assistance	High number of shares to be accomplish even to carry out simple tasks
Examples of code base to study and modify as needed	Lack of an integrated part dedicated to the validation of the trained model
Ease of installation	Difficulties in finding resources examples of use on the web
Open source	Frequent and consistent updates often cause disorientation and confusion if they touch upon mechanics that instead were thought to be learned

Table 7.6: advantages and disadvantages on NVFlare framework.

It is worth noting that, despite it is natural and conventional, it is actually not entirely correct to simply label a given feature as advantageous or disadvantageous.

Think, for example, of the user guide: it has been included among the advantages because its presence certainly makes it easier to learn the framework, but it has also major flaws, such as excessive verbosity and prolixity or inconsistency due to the fact that it is not updated in parallel with the framework.

Such as in any context, one must therefore be able to look at each characteristic in an informed way, not merely evaluating it in a binary way (advantage or disadvantage), but having the maturity to recognize the gradualness and depth behind the evaluation assigned to it.

This clarification is necessary not only because it is emblematic of the logic with which the various evaluations were assigned, but also and above all because the framework under study represents a very complex product, so it deserves a careful analysis that takes into account all the various peculiarities offered, contextualizing them in their own and precise way.

In this sense, an attempt has been made to exploit as much as possible the objectivity and critical spirit developed also thanks to the skills and knowledge received from the course of study undertaken, skills that are essential not only in the engineering field but powerful everyday tools that allow one to analyze reality in a more in-depth, detailed and meticulous way than it may seem.

Turning to mere conclusions, surely one must count among the strengths of the framework the extreme helpfulness of the support team NVIDIA and the presence of the user guide.

However, it is also necessary to mention the great and sometimes even serious obstacles present to date: not only the lack of intuitiveness of even simple tasks such as the setup of nodes in the network, but the extreme verbosity and repetitiveness of the steps to be performed even for simple tasks and the lack of a part dedicated to the validation of the post-training model represent unfortunately critical flaws that compromise the effectiveness, completeness and simplicity of the framework.

Surely these aspects can be improved by the development team as time progresses, but as of today they remain major drawbacks that result in excessive effort and time consuming not only to learn how to use the framework, but also to continue to know how to use it given the high volume of frequent updates that often undermine established mechanics going to disorient the user, who not infrequently finds himself overnight implementing changed or removed actions or steps.

This does not detract from the fact that once the steep learning curve has been overcome with a lot of patience and technical skills, the framework proves to be extremely adaptable to the different needs and situations required, in fact turning into a valuable helper tool, effective and precise in carrying out its tasks of training and graphical visualization of performance, unfortunately less useful for the later stages of testing and validation, unfortunately making the use of other external systems necessary.

In any case, it should be stressed that this is a software dedicated more to the world of research, not a real product finished and released on the market, so it is also normal that there are some dynamics that can lead to some disorientation such as continuous updates.

Chapter 8

CONCLUSIONS AND FINAL CONSIDERATIONS

Although, as anticipated several times, the purpose of the present thesis was to study and evaluate the application of Federated Learning via NVFlare to the problem of Morphing Attack Detection, driven by curiosity and aided by the invaluable help of the professors following me, I was able to go further and deepen the main objective: not only I summarized the knowledge and skills learned in this thesis and in a brief guide to using the framework, but I was also able to train models via Federated Learning capable of performing morphing detection, which is a first in this field of research since to date there are no other studies or experiments in this regard.

Despite of the fact that this thesis work represents the longest, most impressive and complicated challenge faced during my university career, I feel satisfied and proud of the results obtained.

While on the one hand I am happy with the enrichment not only cultural and technical but above all human that the writing of the thesis has left me, I am perhaps even happier with the falls and mistakes made during this work, since it is mainly thanks to them that I have learned to know my limits and deal with them, so I believe that, as is also often the case in many different fields, the moments of difficulty have had the greatest impact on my maturity and the development of my skills.

With regard to the enrichment that I mentioned earlier, it is good to emphasize that I am not referring to the simple acquisition of technical and theoretical knowledge, but rather to the discovery of some mechanisms that I ignored and that instead the context of the thesis highlighted to me: I am referring to the immeasurable richness obtained thanks to the confrontation and dialogue with everyone who helped me to finish the present work, therefore professors and students who, with commitment, dedication and passion, work in the university laboratory of biometrics that hosted me to carry out my research.

From the very first days I could feel an inner growth, composed of practical and methodological notions regarding the subject studied, but also and above all of behavior, mutual help, sharing, collaboration and exchange of ideas and knowledge.

I believe that the main legacy of my thesis is precisely this, that is the awareness that confrontation with others and the exchange of ideas are always positive acts, capable of improving and growing all those involved: looking at things from a different point of view or in any case with someone's help, one often receives confirmation or discovers valid alternatives to his way of thinking, but in any case the confrontation ends with an

increase and a direct acquisition of knowledge, a real sharing.

In this regard, I believe that the course of study undertaken, often structured through abstract technical and mathematical notions, could improve through more direct and focused explanations at times, accepting the loss of some detail or technicality if this is necessary to ensure a better understanding among interlocutors.

I therefore consider myself extremely lucky to have been able to enjoy the opportunity of the thesis also to note this aspect, which is often taken for granted but is in fact a skill that certainly needs to be developed and improved.

In a virtual and connected world in which it is increasingly difficult to describe reality, the figure of the engineer needs to adapt and to improve his communication skills so that his wealth of experience and knowledge continues to help him in understanding and analyzing as objectively as possible the world around him, but above all to share with others what he has learned, so as to give even noble meaning to his research, moving from the effort and commitment of the individual to collective growth and knowledge.

Bibliography

- [1] Matteo Ferrara, Annalisa Franco, Davide Maltoni, "*The Magic Passport*"
- [2] Matteo Ferrara, Annalisa Franco, "*Morph Creation and Vulnerability of Face Recognition Systems to Morphing*", 2022
- [3] Gomez-Barrero M., "*Is your biometric system robust to morphing attacks?*", 2017
- [4] ICAO, "*Biometric Deployment of Machine Readable Travel Documents*", 2004
- [5] M. Ferrara, A. Franco, D. Maio, D. Maltoni, "*Face Image Conformance to ISO/ICAO standards in Machine Readable Travel Documents*", IEEE, 2012
- [6] M. Ferrara, A. Franco, D. Maltoni, Y. Sun, "*On the Impact of Alterations on Face Photo Recognition Accuracy*", 2013
- [7] Ferrara M., Franco A., Maltoni D., "*On the effects of image alterations on face recognition accuracy*", 2016
- [8] Scherhag U., Rathgeb C., Merkle J., Breithaupt R., Busch C., "*Face recognition systems under morphing attacks: a survey*", IEEE, 2019

- [9] IATA, "*Airport with Automated Border Control Systems*", 2014
- [10] Sushma Venkatesh, Raghavendra Ramachandra, Kiran Raja, Christoph Busch, "*Face Morphing Attack Generation & Detection: A Comprehensive Survey*", 2020
- [11] Gomez-Barrero M., "*Predicting the vulnerability of biometric systems to attacks based on morphed biometric information*", 2018
- [12] FRONTEX-Research and Development Unit, "*Best Practice Technical Guidelines for Automated Border Control (ABC) Systems*", 2012
- [13] N. Damer, A. M. Saladie, S. Zienert, Y. Wainakh, P. Terh, F. Kirchbuchner, A. Kuijper, "*To detect or not to detect: The right faces to morph*", 2019
- [14] A. Rottcher, U. Scherhag, C. Busch, "*Finding the suitable doppelganger for a face morphing attack*", 2020
- [15] Valetine J., "*Optimizing blending operations*", Hydrocarbon Engineering, 2006
- [16] O. Celiktutan, S. Ulukaya, B. Sankur, "*A comparative study of face landmarking techniques*", 2013
- [17] Seibold C., "*Detection of face morphing attacks by deep learning*", 2017
- [18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, "*Generative adversarial nets*", 2014

- [19] Yu H., Liao X., "*Free form deformation image registration using edge information measure*", 2008
- [20] Yu C., Chen X., Xie Q., Li G., Yin L., Han H., "*Image deformation using modified moving least squares with outlines*", IEEE, 2017
- [21] Rogers D.F., Adams J.A., "*Mathematical elements for computer graphics*", 1989
- [22] Wolberg G., "*Digital image warping*", IEEE, 1994
- [23] Zinelabidine Boulkenafet, Jukka Komulainen, Abdenour Hadid, "*Face spoofing detection using colour texture analysis*", IEEE 2016
- [24] Y. Weng, L. Wang, X. Li, M. Chai, K. Zhou, "*Hair interpolation for portrait morphing*"
- [25] Venkatesh S., "*Can gan generated morphs threaten face recognition systems equally as landmark based morphs?*", 2020
- [26] Makrushin A., Neubert T., Dittmann J., "*Automatic Generation and Detection of Visually Faultless Facial Morphs*", 2017
- [27] Young A.W., Burton A.M., "*Recognizing faces*", 2017
- [28] K. Raja, M. Ferrara, A. Franco, L. Spreeuwers, I. Batskos, F. De Wit, M. Gomez-Barrero, U. Scherhag, D. Fischer, S. Venkatesh, "*Morphing attack detection-database, evaluation platform and benchmarking*", IEEE, 2020
- [29] Matteo Ferrara, Annalisa Franco, Davide Maltoni, "*Face morphing detection in the presence of printing/scanning and heterogeneous image sources*", 2021

- [30] Raghavendra R., Raja K. B., Busch C., "*Detecting morphed face images*", 2016
- [31] Spreeuwers L., Schils M., Veldhuis R., "*Towards robust evaluation of face morphing detection*", 2018
- [32] Ministry of the Interior National Office for Identity Data and Kingdom Relations, "*State of the art of morphing detection*", 2020
- [33] Robertson D.J., "*Morphed passport photo detection by human observers*", 2020
- [34] Robertson D.J., Kramer R.S.S., Burton A.M., "*Fraudulent ID using face morphs: experiments on human and automatic recognition*", 2017
- [35] Scherhag U., Rathgeb C., Busch C., "*Morph detection from single face image: a multi-algorithm fusion approach*", 2018
- [36] Du X., Zhang R., "*Fusing color and texture features for blurred face recognition*", 2014
- [37] M. Rabbani, "*Jpeg2000: Image compression fundamentals, standards and practice*", Journal of Electronic Imaging, 2002
- [38] Raghavendra R., Li G., "*Multimodality for reliable single image based face morphing attack detection*", IEEE, 2022
- [39] S. Venkatesh, R. Ramachandra, K. Raja, L. Spreeuwers, R. Veldhuis, C. Busch, "*Morphed face detection based on deep color residual noise*", IEEE, 2019

- [40] K. Raja, S. Venkatesh, R. Christoph Busch, “*Transferable deep-cnn features for detecting digital and print-scanned morphed face images*”, IEEE, 2017
- [41] Venkatesh S., “*Multilevel fusion of deep features for face morphing attack detection*”, 2022
- [42] Neubert T., “*Face morphing detection: an approach based on image degradation analysis*”, 2017
- [43] Singh P., Chandler D. M., “*F-MAD: A feature-based extension of the most apparent distortion algorithm for image quality assessment*”, 2013
- [44] Ferrara M., Franco A., Maltoni D., “*Face demorphing*”, IEEE, 2018
- [45] Scherhag U., “*On the vulnerability of face recognition systems towards morphed face attacks*”, 2017
- [46] Guido Borghi, Gabriele Graffieti, Annalisa Franco, Davide Maltoni, “*Incremental Training of Face Morphing Detectors*”
- [47] G. Graffieti, G. Borghi, D. Maltoni, “*Continual learning in real-life applications*” IEEE, 2022
- [48] Tian Li, Anit Kumar Sahu, Ameet Talwalkar Virginia Smith, “*Federated learning: Challenges, methods, and future directions*”, 2019
- [49] Z. Li, D. Hoiem, “*Learning without forgetting*”, IEEE, 2017
- [50] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, “*Overcoming catastrophic forgetting in neural networks*”, 2017

- [51] M.J. Sheller, B. Edwards, G.A. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, R.R. Colen, "*Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data*", Nature, 2020
- [52] Manisha D., Kumar N., "*Cancelable biometrics: A comprehensive survey*", 2020
- [53] Zhou X., Kalker T., "*On the security of biohashing*", 2010
- [54] K. Wei, J. Li, M. Ding, C. Ma, H.H. Yang, F. Farokhi, S. Jin, T.Q., Quek, H.V. Poor, "*Federated learning with differential privacy: Algorithms and performance analysis*", IEEE, 2020
- [55] Chaudhary P., Gupta R., Singh A., Majumder P., "*Analysis and comparison of various fully homomorphic encryption techniques*", 2019
- [56] Bonawitz K., Ivanov V., Kreuter B., Marcedone A., McMahan H.B., Patel S., Ramage D., Segal A., Seth K. "*Practical secure aggregation for federated learning on user-held data*", 2016
- [57] Vishal M.Patel, "*Federated Learning for Biometrics Applications*", IEEE, 2022
- [58] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, F. Beaufays, "*Applied federated learning: Improving google keyboard query suggestions*", 2018
- [59] P. Kairouz, H.B. McMahan, B. Avent, A. Bellet, M. Bennis, A.N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, "*Advances and open problems in federated learning*", 2019

- [60] Hitaj, "Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning", 2017
- [61] Moldoveanu M., Zaidi A., "On in-network learning. A comparative study with federated and split learning", IEEE, 2021
- [62] Rui Shao, Pramuditha Perera, Pong C. Yuen, Vishal M. Patel, "Federated Face Presentation Attack Detection", 2020
- [63] D.P. Kingma, "Adam: a method for stochastic optimization", 2014.
- [64] Simonyan K., Zisserman A., "Very Deep Convolutional Networks for Large-Scale Image Recognition", 2015
- [65] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, "Pytorch: An imperative style, high-performance deep learning library", 2019
- [66] Jha A.K., Ruwali A., Prakash K.B., Kanagachidambaresan G.R., "Tensorflow basics", 2021
- [67] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, Yasaman Khazaeni "Federated Learning with mathced averaging", 2020
- [68] Bouguila N., Ziou D., Vaillancourt J., "Novel mixtures based on the dirichlet distribution: Application to data and image classification", 2003
- [69] Dlib, "Dlib C++ library, dlib.net/python/index.html", 2013

[70] Anaconda, *Anaconda Documentation*, docs.anaconda.com, 2023.