POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Elettronica

Master's Degree Thesis

Homomorphic Encryption for Spiking Neural Networks

Supervisors

Prof. Maurizio Martina

Ing. Alberto Marchisio

Ing. Farzad Nikfam

Candidate

Raffaele Casaburi

December 2022

Abstract

Nowadays, Machine learning is employed in many different everyday problems, especially in the form of Neural Networks. However, the raising computational load and the heavy resource requirements of these structures have led to solutions based on cloud-based services. Moreover, to improve the efficiency and the power consumption of the models' calculations, it has been introduced a new generation of neural networks called Spiking Neural Network which aims to overcome the previous limitations by imitating the human brain behavior. Its sparse, dynamic, and event-driven analytic capabilities improve meaningfully the performances reducing overall costs.

In the last few years, however, what has been considered concerning about this type of service is related to the privacy preservation of the confidential data evaluated by the networks held by servers. To solve the issue several strategies have been proposed, but one of the most promising is Homomorphic Encryption, a particular scheme based on a special method to encrypt information that allows some kinds of computations directly on the encrypted data without decrypting it. Performing this type of Encryption can overcome privacy concerns while using Machine Learning in cloud platforms letting the client employ a network model owned by another party by sending the encrypted data in complete safety.

The project is about the application of this security strategy on 2 different generations of neural networks, the artificial and the spiking ones. Hence, the main features of the neural networks have been collected and the homomorphic encryption strategy chosen, the Brakerski/Fan-Vercauteren, has been discussed as background research. Then it has been selected a simple convolutional architecture (LeNet5) as a model to build a spiking version of the same structure, to perform the classification of images from MNIST, FashionMNIST, and CIFAR10 datasets. After the training phases, the performances of the 2 obtained networks have been compared when a hypothetical client requires encrypted computations.

The simulations have been fulfilled using PyTorch, useful to deal with tensors and networks, and the libraries NORSE and Pyfhel. The first provides the spiking functions while the latter, based especially on SEAL, has been used to deal with the encryption.

More specifically, the Homomorphic Encryption scheme has been employed to encrypt the input images and to encode all the layers of the 2 architectures that are characterized by linear operations, making the networks able to manage encrypted inputs. In addition, a multi-party computation pattern has been used to execute the non-linear operations, needed by the models to learn, without approximating the activation functions which characterize the 2 networks.

The experimental results portrayed a better performance, in terms of accuracy and of the errors made, of the SNN when the homomorphic encryption scheme is used. On the other hand, SNN requires a huge amount of time with respect to CNN since the necessary initial time encoding makes the execution as slower as the observation sequence grows.

In conclusion, this project has underlined the necessity of better strategies to merge the non-linearity of neural networks with the constraints of Homomorphic Encryption, better SNN training methods, and better configuration of the homomorphic scheme.

Acknowledgements

I would like to express my gratitude to my primary supervisor, Prof. Maurizio Martina, for giving me the opportunity to work on this interesting project. I would also like to thank Ing. Alberto Marchisio and Ing. Farzad Nikfam, who assisted me, and helped me finalize the project.

The assistance, patience, and guidance provided were greatly appreciated.

I would like to thank my family, my mother and father, who have found a new way to support me every day, and my sister and my cousin Marco, who have sustained me in different ways.

I would like to extend my sincere thanks to my grandmother, my aunt Fiorina, the rest of the family, and those who are no longer there, for which I am grateful.

Finally, my heartfelt thanks to all my friends for every moment they all have blessed me with.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last few decades research in Machine Learning has spread to the point that it has found a large range of applications in different contests: Finance, Diagnosys, security, robotics, Autonomous systems, etc. Its diffusion has made it so pervasive that nowadays a lot of information is collected constantly and automatically while providing a certain service.

Machine Learning concepts can be exploited by neural network models, sets of layers made of nodes that perform easy jobs that together solve difficult problems aiming at the human brain functionalities emulation. These architectures, during the years, have been through an evolution process that has led to the formation of several generations of them with different aspects. The most common type is the Convolutional Neural Network (CNN) which works fine though its behavior is far away from the realistic biological one of the neuron, the human brain node. Since the latter is way more efficient, a new generation has been built to reproduce more closely the functionality of the human neuron, the Spiking Neural Network(SNN) which relies on spikes to compute its operations and to deal with communication between nodes, saving power, making the elaborations more efficient and simplifying the Hardware requirements.

Essentially, technological improvements, new platforms and strategies for storage, and new increasing computational possibilities algorithms have led to efficient predictive models able to elaborate huge amounts of data which in most cases, it is desired to preserve. Privacy has become one of the most concerning subjects since clearly, machine learning has shown a lot of previously unknown vulnerabilities in everyday used software. Not only the user data could be stolen, but even the model can be stolen.

This project has as its focus the application of a security measure to the machine learning model in order to protect the data of a hypothetical client when employing a neural network owned by a server to solve a classification problem. This security measure is Homomorphic Encryption. It is a particular form of encryption that

uses polynomial features to enable the elaboration of data while still encrypted. Basically, it allows the client to employ the server network for a task sending its sensitive information obscured by this encryption scheme with some fundamental parameters useful to make the model in the cloud able to apply the operation required on the encrypted inputs. At the end of the computation, the server can send back the outcomes still encrypted, and once received, only the client is able to decrypt them gaining the plain results.

Executing the evaluations in this secure way is possible due to the main characteristics of the encryption scheme which needs to be homomorphic which means that it is able to preserve the algebraic structures of a system if certain conditions are valid making possible the addition and multiplication operations on encrypted variables. Even though in theory it existed before, Homomorphic encryption caught massive interest only after the publication of Gentry resources.

In conclusion, this project aims at comparing 2 different generations of Networks, the convolutional and the spiking ones, when dealing with a computation characterized by a Homomorphic Encryption scheme, the Brakerski/Fan-Vercauteren. Thus, the following chapter discusses the main concepts of Neural Networks, in particular, describes the basic features of CNN and its layers, and introduces the fundamental peculiarities of SNN. The third chapter exposes the definitions and the functionalities of Homomorphic Encryption, its limitations, and classifications, before portraying the encryption scheme used in the implementation. The fourth chapter is about the framework and the relative choices made: the datasets used for the comparison, the 2 architectures employed based on the LENET5, the training phase required by both, the inference and scheme details, and the protocol used to overcome the encryption limitations. The fifth and last chapter regards the obtained results, the conclusions that can be deduced from the comparison between the 2 neural structures analyzed, and a discussion on the areas of improvement and possible future work.

Chapter 2

Neural Networks

2.1 Convolutional Neural Networks

Seen as a part of artificial intelligence, Machine learning (ML) is a field that aims to understand and create methods that 'learn', more specifically methods that can influence data to improve performance on a particular task. It collects methods developed in the last decades of the twentieth century and its algorithms are based on acquiring sample data, the training data, in order to update itself and become able to make predictions or decisions without being explicitly programmed to do so[1].

Machine learning is used in many applications, such as medicine, email filtering, speech recognition, and computer vision. ML methods can be unsupervised and supervised and in particular, Artificial Neural Networks are mathematical patterns, a subset of supervised ones.

During the training step, input from the trainset is given to the model to be analyzed. The output obtained is compared with the desired one to adjust the model by updating some tunable parameters called hyperparameters. During this phase, a loss function is used to minimize the error between the two outcomes. This learning behavior is obtained without explicit instructions[2].

Essentially, the aim of Artificial neural networks is to emulate the analytic attitude of biological brains, to predict and solve tasks reaching more or less the same performances and results. To achieve this purpose, it is fundamental to understand how the animal brain works.

It is based on particular units (neurons) that react to electrical and chemical stimuli. These units are interconnected employing synapses to create a neural pathway able to send information from one section of the brain to another. In this way, a neural circuit able to fulfill some simple specific function is obtained. Obviously, with more interconnected neural circuits, it can be derived a large-scale

brain network, able to perform more complex functions [3].

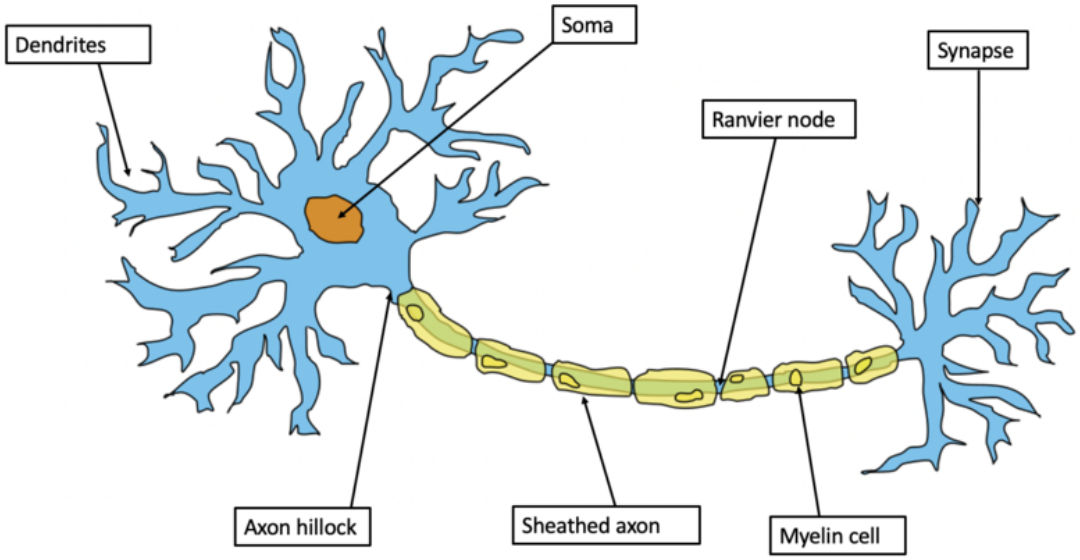


Figure 2.1: Neuron from [4]

Since the basic operations of the complex functions performed by the network are computed by the neurons, it is fundamental to determine their model. The neuron model characterizes the single unit utilizing an activation function that defines the output depending on the input received.

Neurons are usually organized in groups called layers and several layers form a neural network. There are different kinds of layers since each one needs to execute a specific step of the computation. Layers inside the network can be divided in:

- Input layer that receives an input from the user;
- Hidden layer that elaborates the information received from the previous one;
- Output layer: that returns the results of the evaluation.

Obviously, the network can be enlarged by including more layers and using more complex layers. As the neural networks became popular, the model used to characterize the neurons went through a process of evolution that made them more complicated so that three generations of networks could be defined [5].

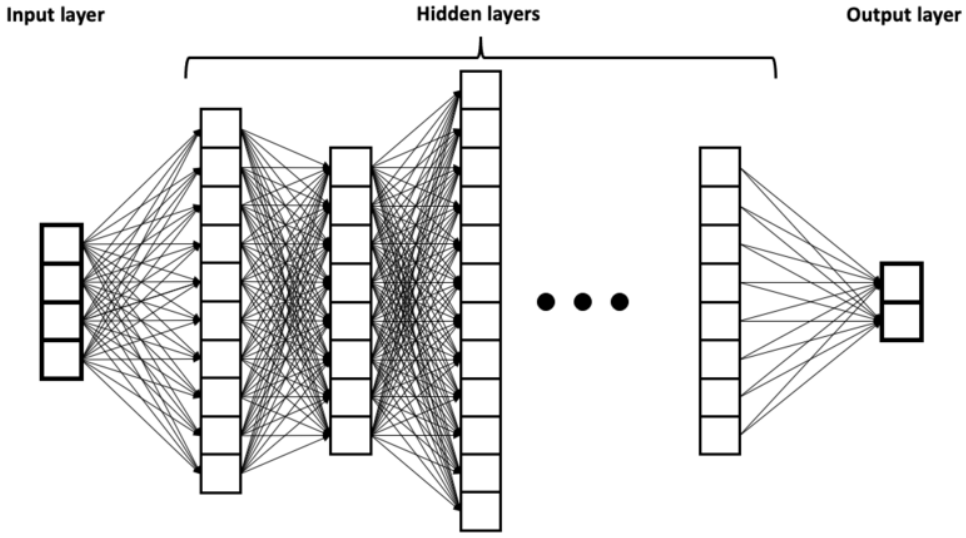


Figure 2.2: DNN from [4]

The first generation of ANNs, the perceptron, is called also threshold-gates and they are based on the McCulloch–Pitts model:

$$f(x) = \begin{cases} 1 & \text{if } h = \sum_{\#neurons} x_i w_i > u \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $f(x)$ is the outcome evaluated by the neuron that can result in activation, if it is equal to 1, or not if it is 0; h is the state of the neuron; w_i is the synaptic weight; x_i is the input of the neuron which is the output from the previous layer and u is the threshold. The latter can be set to 0 or to a specific value by means of bias. Using bias makes the network more flexible.

To perform the training step, it is used the Hebb1 theory due to the binary nature of this kind of network. Essentially, the theory claims that if a pre-synaptic neuron (whose output acts as input) causes the activation of the post-synaptic neuron (that receives the input stimulus), the synaptic weight is enhanced [6]. However, the first generation of neural networks does not behave properly with continuous data.

An evolution of the first generation is represented by the second one which is based on a model that provides a neuron with a continuous activation function g according to the following equation:

$$f(x) = g\left(\sum_{\#neurons} x_i w_i - b\right) \tag{2.2}$$

where g is usually a sigmoid function and b is the bias.

In order to perform the training step for a network of the second generation the rule of Hebb is not used anymore. Indeed, there have been employed new types of algorithms such as gradient descent. This technique aims to modify the weights in the last layer exploiting the error between the predicted result and the expected one. In this case, it is fundamental the back-propagation step because it is essential to go through the network from the final layers back to the first ones to propagate the error to the hidden layers. In this way, even a network with many hidden layers can be trained (deep learning)[6].

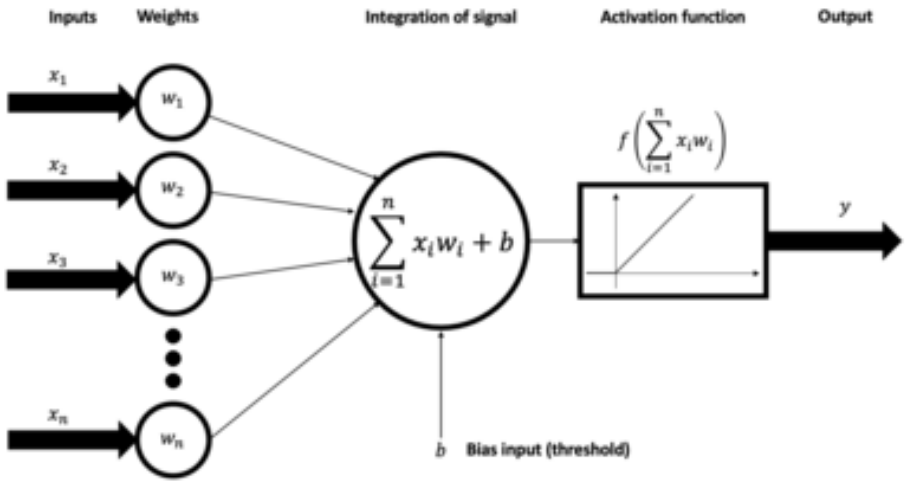


Figure 2.3: Neural unit from [4]

Even though the second generation of ANNs can simulate all the boolean functions with fewer neurons with respect to the first generation and can approximate any continuous functions with just one hidden layer the third generation of ANNs is even more efficient.

Indeed, this last generation is able to reproduce the spiking dynamics of the biological neurons. They are called Spiking Neural Networks due to their behavior which is modeled by means of an Ordinary Differential Equation (ODE). To reach the spiking behavior the information needs to be encoded over time. This step can be obtained using the firing rate or the period of time between two consequent spikes [7].

2.1.1 CNNs and Layers

Overall, Machine Learning is a method of computation based on automatized analysis of large datasets and an efficient way of building analytical models. There

are three different types of Machine Learning tasks:

- Reinforcement Learning, where the algorithm receives rewards or penalties during its interaction with an environment [1]. Learning is obtained by the maximization of the reward.
- Unsupervised Learning, where the model simply looks for patterns in the given data, but it does not have any information about the inputs [8].
- Supervised Learning, where samples of the data with their corresponding output labels are used to perform a mapping between the input and the output.

As previously clarified, Neural Networks are models able to fulfill complex functions by means of layers of simple processing units that communicate due to weighted interconnections [9].

Since each layer has its own goal and its own characteristics and since they essentially build the model, it is fundamental to define the problem to solve and to understand how the layers work and how they are useful to accomplish the specific task the network is built for.

The project discussed in this thesis is about supervised learning of classification in the case of spiking neural networks. Several datasets have been used but all of them have an output that must be classified into ten classes, from 0 to 9.

For what concerns the layers of the analyzed network, in the case of the MNIST dataset, the input image has a dimension of 28*28 pixels with 1 channel so the input layer would be made of 28*28 input nodes. Since the aim is to classify the incoming image, the final layer consists of 10 nodes, and their values determine the winner digit. To solve the problem of image recognition with high accuracy, Convolutional Neural Networks (CNN)s represent a solid solution.

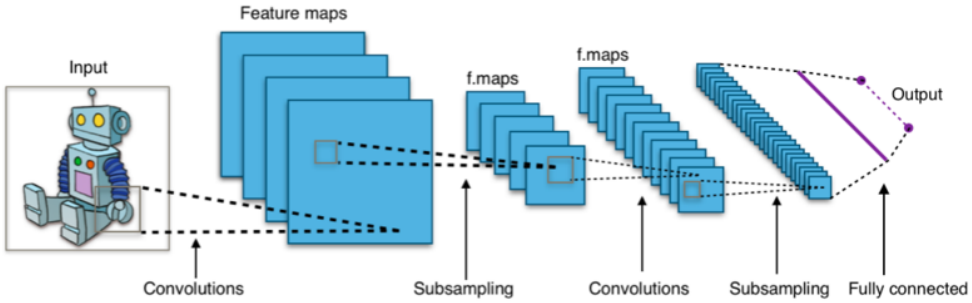


Figure 2.4: Architecture of CNN from [10]

Inspired by the biological processing capabilities of the animal visual cortex, CNNs have proven their effectiveness in image classification problems due to their mastery in capturing the spatial topology with respect to standard networks.

Convolution Layer

The Convolution Layer is the one that determines the extraction of the input features evaluating a convolution between the input and a filter, named kernel. In this way, a feature map is produced. The kernel is a matrix composed of weights that slides over the input with a step determined by the stride.

So at each step, matrix multiplication is computed and the outcomes are summed onto the feature map. During this step is fundamental the padding process that ensures that the output has the correct dimension. Usually, it is used the zero-padding which contemplates the addiction of pixels, each with zero as a value, around the input. This layer is commonly the most computationally heavy since numerous matrix multiplications are expected. Overall, the characteristics, Hyperparameters, of the layer are:

- Kernel size
- Filter count
- Stride
- Padding

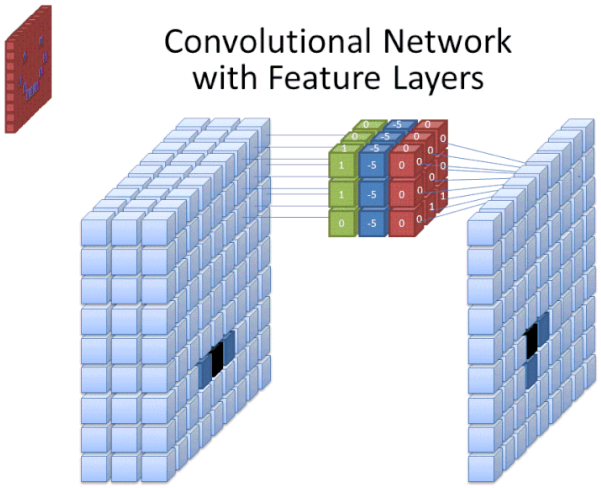


Figure 2.5: Convolutional layer from [11]

Fully connected Layer

The convolutional layer is the first one of the network and it is what the input encounter, while the last layer of the model is the fully connected one. Exactly like

the first one, even this layer provides only linear calculation performing a weighted sum of the inputs and the bias addiction. As specified by the layer name, all the nodes from the previous one are connected with the current ones. Moreover, since it is the final step, it maps the visual features obtained during the evaluation to the outcomes [9].

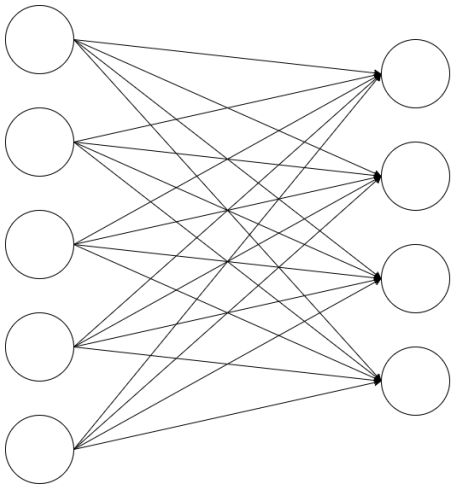


Figure 2.6: Fully connected layer

Activation Layer

Between the analyzed steps, which are characterized by completely linear calculations, there is a non-linear layer that simply contemplates the application of an activation function on the received input. The most common ones are the sigmoid, the hyperbolic tangent (tanh), and the Rectified linear operation(ReLu). This type of layer is essential to build effective models.

Pooling Layer

It is useful to include in the model some pooling layers that decrease the size of the feature maps. Thus, the number of parameters used in the matrix calculations in the linear layers is reduced, simplifying the learning step, and, since further operations are executed on summarized features, making the network more robust to the input characteristics positions changes. There are two widespread types of pooling:

- Max-Pooling, which takes into account only the higher value of the window, collecting the most prominent features of the previous feature map.

- Average-Pooling, which evaluates the mean value of the window, granting the average of features present in a patch [12].

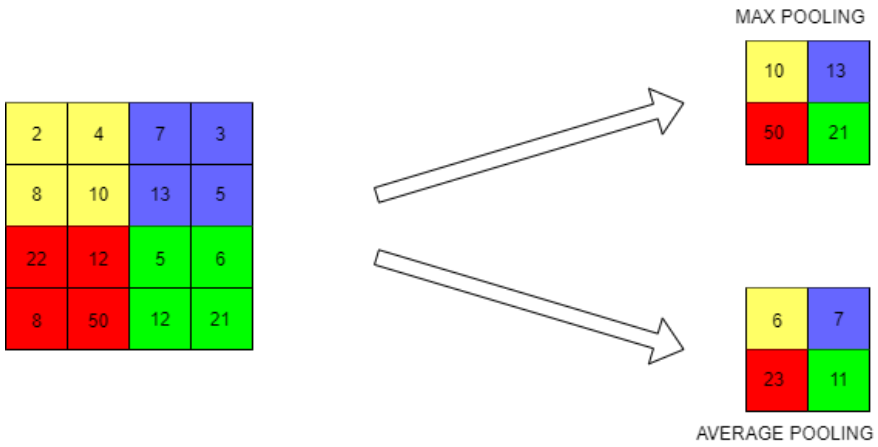


Figure 2.7: Pooling layer

Softmax

At the end of the fully connected layer, it is used the softmax step in the case of classification problems with discrete class labels to obtain as output a probability per each class. More specifically, the sum of these probabilities is 1 and the association with each class is performed with a standard exponential function. Thus, the output with the higher value has a higher probability too and it is considered the most likely class for the input. A variation of this layer is represented by the Log Softmax which has advantages over Softmax for what concerns numerical stability, optimization, and heavy penalization for highly incorrect classes.

Loss Function

Once the outcome is obtained, a loss function is used to verify the effectiveness of the network. It is essential to compare the results of the computation with the required outcome and to measure their divergences. If the loss function is high, weights and biases need to be adjusted in order to minimize it. There are a lot of loss functions like Mean Squared Error, Mean Absolute Error, Cross Entropy, and Negative Log Likelihood and they are used with the backpropagation to understand how to update the parameters of the model [13].

Backpropagation

Backpropagation is the most important step during the training phase. It is used the analyze the network in the backward direction and to change the weights and the biases of the model using the loss function as the main information. The most used algorithm, named Stochastic gradient descent (SGD), is based on the evaluation of the gradient of the last layer with the partial derivatives of the loss function with respect to the weights and on the calculation of the local gradients of the inner layers that depend on the outputs of the next layers. Thus, there is a backward flow of propagation in which the gradient is used to adjust the weights of the previous layer [14].

Optimization

The training step can be a lot of time-consuming depending on the model parameters, the model dimensions, the dataset complexity, and the specific task the system is aimed at. Moreover, the learning rate parameter can affect the efficiency of the training step since the size of the adjustment the weights receive depends on it. Essentially, a too big value of the learning rate can lead to a high oscillation of the variation making it difficult to reach the minimum of the cost function. A too much low learning rate, instead, can slow down a lot the operations needed to achieve the optimum situation [15]. Hence, it is fundamental to update the learning rate during the process, reducing the parameter as the weights get closer to the desired values. There are several types of optimization: SGD with momentum, Adagrad, Adam, etc.

2.2 Spiking Neural Networks

According to recent research, ANN ML requires so many resources in terms of energy, time, and memory consumption during training and inference phases that questioning about the technology accessibility has become fundamental. Indeed, the human brain is able to solve more general and sophisticated problems with a fraction of the power and time needed by ANNs. This is the reason why neuromorphic computing has the goal of reproducing the operations of the brain obtaining a huge improvement (orders of magnitude) in terms of energy efficiency and requirements.

Biological neurons are essentially complex systems with analog dynamics that communicate through the timing of digital spikes. These connections are characterized by large fan-out, feedback, and recurrent signaling paths in a different way with respect to feedforward or recurrent structures of ANNs. Moreover, the sparse, dynamic, and event-driven operations of biological neurons are perfectly capable of realizing complex online adaptation and learning mechanisms with few resources.

Overall, it appears clear how a huge improvement a network with these qualities, the SNN, can represent.

More specifically, SNNs are trainable dynamic systems that use the temporal dimension to encode data, in this way the information is evaluated as asynchronous and sparse spike trains that go through interneuron communications and intraneuronal computing.

Obviously, all these specifications have led to the development of prototype neuromorphic hardware platforms able to deal with time-encoded data like IBM's TrueNorth and SpiNNaker. Hence, to process the encoded inputs, systems based on hybrid digital-analog circuitry and in-memory computing are usually implemented, since it has been demonstrated to perform a remarkable energy-saving execution of complex behaviors [16].

As previously described, biological neural systems consume orders of magnitude less energy than ANNs and are much more flexible and robust, adaptive, and efficient due to their way of operating in a massively parallel way rather than time-multiplexed computing units. Furthermore, neurons of the brain follow continuous-time dynamics in real, physical time instead of operating on a discrete dimension that is an abstraction of real-time.

In addition, conventional computers use Boolean logic, bit-precise digital representations, and time-multiplexed and clocked operations, that burn extra power, while nervous systems use inherently noisy analog units in the continuous time domain in an event-driven and massively parallel way [17].

2.2.1 Neuron models

There are several ways to represent the biological behavior of the neuron which is based on the firing characteristic. Essentially, the unit processes the data received and sends the spike trains ahead. The chances to verify a spike depend on the membrane potential that needs to be higher than a threshold to make the neuron fire [18].

After the firing phenomenon, the neuron goes through a refractory period during which it would not be possible to fire again. More specifically, when the threshold is reached, depolarization occurs and the action potential rises. Subsequently, there is the repolarization phase which leads the neuron to a voltage value below the resting state causing the absolute refractory period. The resting state is obtained only after the hyperpolarization during which the neuron can fire though it is highly unlikely(relative refractory period).

This biological behavior is represented with high accuracy in the Hodgkin-Huxley model though there are other fundamental models which are less precise but less complex.

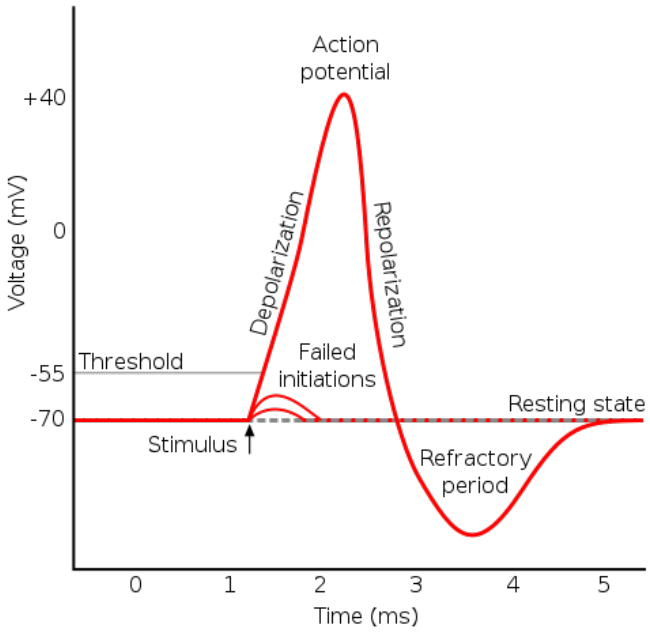


Figure 2.8: Action potential from [19]

Overall, the most important models are:

- Hodgkin-Huxley model
- Spike response model
- Izhikevich neuron model
- Leaky integrate-and-fire model

The Hodgkin-Huxley is the model that shows higher affinity with the biological neuron but its complexity makes it impossible to build huge spiking neural networks using it.

The Spike response model is an approximation of the original functionality and its main characteristic is that it does not depend on the voltage but on the difference between the current instant and the time of the last spike produced. Furthermore, its describing equation is of integral type:

$$u_m(t) = \eta(t - \hat{t}_k) + \sum_j w_{jk} \sum_f \epsilon_{jk}(t - \hat{t}_k, s) + \int_0^\infty \kappa(t - \hat{t}_k, s) i_{ext}(t - s) ds (2.3) \quad (2.3)$$

where $u_m(t)$ is the membrane potential and $\epsilon_{jk}(s)$ is the trend of post-synaptic potential since j is the pre-synaptic neuron and k is the actual neuron. The number

of spikes is described by f, $s = t - t_j^f$, $\kappa(t - \hat{t}_k, s)$ represents the response to the current i_{ext} and $\eta(t - \hat{t}_k)$ is the behavior the potential when the threshold is reached. In conclusion, the threshold is defined by the time of the last spike: $\theta = \theta(t - \hat{t}_k)$ [20].

Izhikevich is probably the best model due to its affinity with brain functionality obtained at an affordable computational cost. It is essentially a simplification of the Hodgkin-Huxley model and it is based on two differential equations:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u \tag{2.4}$$

$$\frac{du}{dt} = a(bv - u) \tag{2.5}$$

$$if v \geq \theta then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{2.6}$$

where v is the membrane potential, u is a recovery variable, θ is the threshold and a, b and c are dimensionless parameters [21].

The last approximation is the Leaky integrate-and-fire model based on an RC circuit to represent the membrane potential. Thus, after this potential exceeds the threshold, neurons fire, and the membrane is set to u_{rest}. It is possible to define an absolute refractory period and a relative membrane value [22].

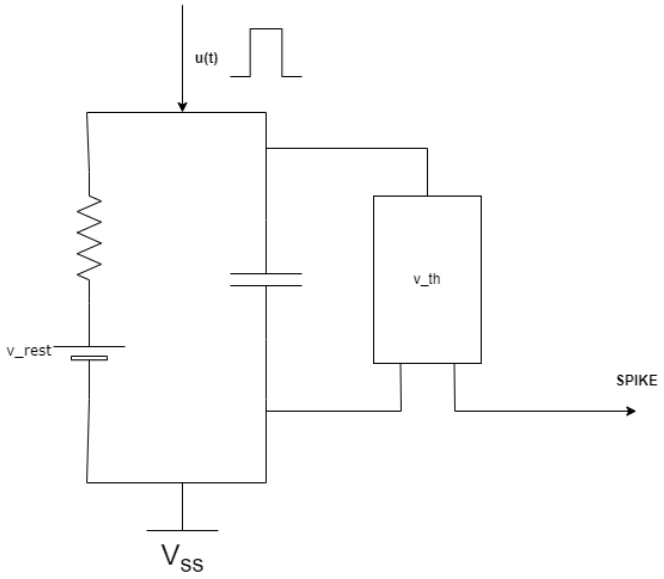


Figure 2.9: LIF circuit

Clearly, the LIF model is the easiest one to implement, but it is not able to reproduce properly all the biological neuron features. Depending on the requirements the most effective solution can be chosen though, usually, LIF and Izhikevich are preferred due to their moderate computational cost [22].

Chapter 3

Homomorphic Encryption

Nowadays deep learning is used in a lot of applications such as research, content filtering, labeling recommendation, and e-commerce systems due to its capability to reproduce the way humans fulfill their tasks. Based on complex models, needed to capture the main input data features, Deep Learning is a complicated technology that requires proficiency in several disciplines like neurology and computer science. Since the production of the required models can be inaccessible, it has become fundamental to use clouds since they represent a perfect platform to host pre-learned models at cheap costs and high computational capability following the philosophy of Deep Learning as a Service [17].

Thus, functionality is provided affordably and reasonably even though there can be observed some other kind of disadvantages. The most concerning ones are related to data privacy since cloud platforms are used. In fact, the cloud receives the data from users and, after the computations, it sends back the results executed by the model, but, as the process goes on, the information can be compromised. Therefore, it is essential to guarantee a secure and non-interactive method to perform the computation required respecting the privacy of the data.

A possible solution is represented by Homomorphic Encryption, a special kind of cryptography based on the idea of performing all the computations over encrypted inputs without decryption [23].

The main feature of this technique is to protect both the data and the model of the server. The first is guarded with the encryption and the latter is obscured to the client that receives the encrypted outcomes. Usually, this approach allows a non-interactive procedure between the client and the server. In conclusion, the main drawback of this strategy is the computational overhead that the encryption causes.

Homomorphic Encryption has been first studied by Rivest, Adleman, and Dertouzos as a method useful for third parties to work with encrypted data [24]. In the following decades, there have been many attempts to design similar systems

and several solutions have been considered, thus leading to different types of Homomorphic Encryption schemes:

- Partially Homomorphic Encryption (PHE)
- Somewhat Homomorphic Encryption (SWHE)
- Fully Homomorphic Encryption (FHE)

Partial Homomorphic Encryption Schemes

The earliest HE schemes were of Partial Homomorphic Encryption type. Paillier, Goldwasser and Micali, and El Gamal are some of the most famous methods of this kind. The main feature of PHE strategies is that it is possible to perform either addition and multiplication but not both making them useful for a limited number of applications. Despite the restricted qualities of these schemes, Paillier one is still used nowadays where the main operation required is addiction. A solution that could guarantee the use of both operations has been looked for since the birth of this technique, for example, Boneh et al's scheme allows unlimited additions but only a single multiplication [25].

Somewhat Homomorphic Encryption Schemes

Many Homomorphic Encryption strategies have been proposed in the last decades, Polly Cracker, for example, proposed the first scheme able to perform additions and multiplications simultaneously though it has been considered not exploitable due to the growing size of cipher texts. Essentially, Somewhat Homomorphic Encryption techniques allow the computation of polynomials but only the ones of lower degrees. Thus, SWHE has been considered useless in terms of providing a solution to practical and real problems. Something changed when Machine Learning and Cloud Computing spread and the relative privacy issues became concerning [26].

Fully Homomorphic Encryption Schemes

A turning point for the Homomorphic Encryption evolution has been Gentry's seminal work which led to a solution capable of both additions and multiplications without the limitations of previous strategies. Gentry proposed a Fully Homomorphic Encryption scheme and a framework to build it and since privacy needed to be ensured in the new applications, many other schemes have been designed following his work. The main features of Gentry's FHE scheme are based on ideal lattices and gave birth to the first generation of FHE schemes called over integers (Van Dijk). The second generation of FHE techniques is based on (Ring) Learning With Errors

problem (Brareski-Vaikuntanathan, and Fan-Vercauteren are the most relevant works), while the last of these families is defined by the NTRU-like schemes [27].

3.1 Homomorphic Encryption: basic idea

Homomorphic Encryption is essentially a transformation that preserves the initial structure. It is defined considering the function $\varphi : \mathrm{Z} \to Z_q$ where $\varphi(z) = zmodq$. The map φ is called a ring homomorphism between the rings Z and Z_q if it preserves the additive and multiplicative structure of the integers so that $\varphi(z1 + z2) = \varphi(z1) \oplus \varphi(z2)$ and $\varphi(z1 \times z2) = \varphi(z1) \otimes \varphi(z2)\ \forall z1, z2 \in Z$ where $\oplus$ and $\otimes$ are addition and multiplication modulo q (in Z_q). So the idea behind homomorphic encryption is to preserve the structures between the rings of the plain texts and cipher texts while applying encryption and decryption:

$$Decsk(f(Encpk(a), Encpk(b))) = f(a, b) \tag{3.1}$$

$$Decsk(f(Encpk(a), b)) = f(a, b) \tag{3.2}$$

Obviously, though the HE allows the performing of some computations on the encrypted data, it is necessary to guarantee that the effectiveness of the security level of this kind of cryptography is equal to the non Homomorphic ones.

Overall, in order to perform all the needed computations in the encrypted domain, it is essential to support the operations of Key generation, Encryption, Decryption, and Evaluate. It is considered homomorphic for a function $\in$ FM applied on messages, a scheme where it is possible to define another function $\in$ FC like:

$$Decsk(Encpk(m_1) \diamond Encpk(m_2)) = m_1 \circ m_2 \forall m_1, m_2 \in M \tag{3.3}$$

So FM is the set of functions that can be evaluated according to the HE scheme through functions in FM and FC that do not necessarily correspond. In fact, the Paillier algorithm is characterized by the transformation: $FM = \{+\}$, $FC = \{\times\}$.

To homomorphically evaluate every function, it is needed to be able to compute polynomials requiring unlimited homomorphic additions and multiplications and making possible the approximation of any suitably smooth function. Since multiplication corresponds to an AND operation and additions to an XOR one for a single bit, it is possible to obtain a Fully homomorphic encryption "if it can encrypt 0 and 1, and ADD and MULTIPLY encrypted data".

However, it is difficult to build a scheme like the one described due to the growing noise, added with the aim of turning the computation non-deterministic. In fact, this noise term grows exponentially with the protracted homomorphic calculations (especially multiplications). Thus, at the end of the calculations, the

decrypted result will not match the expected one because of the corruption of the message due to this growing noise.

A strategy with this limitation is of the SWHE type since it can still work if the degree of polynomials is kept low enough. Clearly, a Fully homomorphic encryption scheme can be obtained by a somewhat homomorphic encryption one if it is possible to reduce the noise level during the process.

Essentially Gentry proposed the first feasible FHE algorithm based on lattice cryptography, starting with a somewhat homomorphic one where the noise in cipher texts has been taken under control with two additive operations: squashing and bootstrapping [28]. Squashing is the step in which the decryption function is manipulated and simplified in order to make the scheme bootstrappable. Bootstrapping is the process of refreshing a ciphertext in order to produce a new ciphertext of the same message, but with a lower level of noise so that more homomorphic operations can be performed on it. Basically, it is like decrypting the ciphertext with an encryption of the secret key (the bootstrapping key) and then re-encrypting the message to proceed with the encrypted calculations.

In conclusion, this has been the turning point for the spread of a practical form of homomorphic encryption and for the beginning of new research that led to new variants of the original algorithm.

3.1.1 The Brakerski/Fan-Vercauteren HE Scheme

The encryption scheme chosen for this project is the Brakerski/FanVercauteren (BFV) version that is based on the Ring learning with errors problem. Learning with errors was introduced by Regev (2009), and has become one of the most important and hard problems in lattice-based cryptography since it has been used to construct several cryptosystems. Based on the LWE problem, the LWE encryption is characterized by two positive integers, n, and q, and an error distribution χ over Z, normally a discrete Gaussian of width αq, with $0 < \alpha < 1$.

Definition 3.1.1 (LWE distribution) *Given a secret vector $s \in Z_q^n$, the LWE distribution LW $E_{s,\chi}$ over $Z_q^n \times Z_q$ is sampled by picking $\overrightarrow{a} \leftarrow Z_q^n$, an error $e \leftarrow \chi$, and returning $(\overrightarrow{a}, b = \langle \overrightarrow{s}, C \rangle + e)$ [29].*

So, for n, q, t $\in N^*$ with t|q and a message m $\in Z_t \subset Z_q$, the encryption with the key $\overrightarrow{s} \leftarrow \chi_{key}(Z_q^n)$ of m is defined as:

$$LWE_{q,\overrightarrow{s}}(m) := (\overrightarrow{a}, b) = (\overrightarrow{a}, \langle \overrightarrow{a}, \overrightarrow{s} \rangle + \tilde{m} + e) \in Z_q^{n+1} \tag{3.4}$$

where $\overrightarrow{a} \leftarrow Z_q^n$, error $e \leftarrow \chi_{error}(Z_q)$ and $\tilde{m} = mq/t$. While the decryption of a ciphertext $(\overrightarrow{a}, b)$ of m is

$$LWE_C^{-1}(\overrightarrow{a}, b) := \lceil t/q(b - \langle \overrightarrow{a}, \overrightarrow{s} \rangle) \rfloor \in Z_q \tag{3.5}$$

If $Err_{LWE}((\overrightarrow{a}, b), m) = t/q(b - \langle a, s\rangle) - m = et/q$ and if |e t/q| $\in [0, 1/2]$ then $LWE^{-1}(\overrightarrow{a}, b) = m$, making the decryption correct. It is clearly fundamental to keep the function Err_{LWE_s} small in order to obtain a perfectly working decryption. In conclusion, every time a homomorphic addiction is computed, noise is accumulated and it is essential to be able to reduce it (for example using bootstrap) to obtain the correct decrypted result.

RLWE encryption was introduced [30] and is the encryption based on the LWE problem on a suitably defined polynomial quotient ring [31].

Definition 3.1.2 (Decision-RLWE) *For λ, the security parameter, let $f(x)$ be a cyclotomic polynomial $\Phi m(x)$ with $deg(f) = \varphi(m)$ depending on λ and set $R = Z[x]/(f(x))$ and the integer $q = q(\lambda) \geqslant 2$. For a random element $s \in R_q$ and a distribution $\chi = \chi(\lambda)$ over R, denote with $A^{(q)}_{s,\chi}$ the distribution obtained by choosing a uniformly random element $a \leftarrow R_q$ and a noise term $e \leftarrow \chi$ and outputting $(a, [a \cdot s + e]_q)$. The Decision $- RLWE_{d,q,\chi}$ problem is to distinguish between the distribution $A^{(q)}_{s,\chi}$ and the uniform distribution $U(R^2_q)$ [29].*

It is possible to restrict s to be sampled from χ instead of taken uniformly in R_q without modifying any security aspects. Moreover, the hardness of the problem does not depend on the precise shape of q which does not need to be prime and can be chosen as a power of 2 [32].

Based on this decision problem, it is possible to define the following encryption scheme.

The plaintext space is taken as R_t for some integer t > 1. If $\triangle = \lfloor q/t \rfloor$ and $r_t(q) = qmodt$ then $q = \triangle \cdot t + r_t(q)$. It is fundamental to clarify that q nor t have to prime and that t and q do not need to be coprime. The main operations are:

- $SecretKeyGen(1^\lambda)$: sample s $\leftarrow \chi$ and output $sk = s$
- $PublicKeyGen(sk)$: set $s = sk$, sample a $\leftarrow R_q$, e $\leftarrow \chi$ and output $pk = ([-(a \cdot s + e)]_q, a)$.
- $Encrypt(pk, m)$: to encrypt a message m $\in R_t$, let $p_0 = pk[0], p_1 = pk[1]$, sample $u, e_1, e_2 \leftarrow \chi$ and return $ct = \left([p_0 \cdot u + e_1 + \triangle \cdot m]_q, [p - 1 \cdot u + e_2]_q\right)$
- $Decrypt(sk, ct)$: set $s = sk$, $c_0 = ct[0], c_1 = ct[1]$ and compute $\left[\left\lfloor \frac{t \cdot [c_0 + c_1 \cdot s]_q}{q} \right\rceil\right]_t$
- $Add(ct_1, ct_2) := ([ct_1[0] + ct_2[0]]_q, [ct_1[1] + ct_2[1]]_q)$. Where the noise grows additively by a maximum of t since $||r|| \leq 1$.
- Homomorphic multiplication is a quite complex operation since it consists of two steps: the first is basically the multiplication of polynomials ct1(x) and

ct2(x) scaled by t/q, and the second is "relinearisation" needed because at the end of the first computation it is obtained a ciphertext consisting of 3 ring elements instead of 2.

3.2 Multi-party Computation

Homomorphic encryption (HE) is a fundamental strategy to build a privacy-preserving neural network but there is also another algorithm used to achieve it called secure multi-party computation (SMC) and, usually, a combination of both is performed. The first to propose a privacy-preserving neural network classification algorithm based on both methods has been Barni et al in 2006 [33]. Neural networks are described as a succession of scalar products secured by means of homomorphic encryption and activation functions protected with protocols based on secure multi-party computation. Essentially, the client encrypts each component of its data with Paillier's homomorphic encryption and sends them to the server which evaluates homomorphically the encryption of the scalar product [34]. Then the server sends back the encrypted result to the client who finally decrypts it in the plaintext domain.

Since homomorphic encryption has been used, it is impossible for the server to learn features about the client's data. For what concerns the complexity of communication, it is equal to $O(Nn^2)$ with N number of components per vector, and n the RSA-modulus for the Paillier cryptosystem. The evaluation of each threshold activation function processing is obtained with Yao's solution to the millionaire's problem [35]. It is fundamental to observe that the values of all intermediate neurons which are the outputs of scalar products and activation functions are revealed to the client making the protocol inefficient and imperfect since it does not fulfill the privacy requirement.

An enhancement of the protocol has been evaluated in Orlandi et al [36] where it is proposed a solution that does not reveal intermediate results to the client. The most relevant improvements are the use of the cryptosystem version by Damgard and Jurik [37] in place of the original Paillier's and a security measure to guarantee that the client does not learn anything about the intermediate neurons. Essentially the server information is protected by the use of a random value inserted in the computation before sending the encrypted partial results to the client. Though this solution does represent an improvement it has been still considered inefficient.

The methods that followed have the aim of reducing the communication complexity by avoiding the interaction between client and server. The idea is to have a client that sends the data encrypted with an FHE scheme and a server that processes a version of the classification algorithm adapted to operate on encrypted data without decrypting them and only at the end of all the calculations the

results are sent back to the client for the decryption. Though privacy is ensured and communication complexity is highly reduced, the procedure has an important increase in processing complexity due to the algebraic nature of the classification algorithm and its multiplicative depth. The most concerning step to execute this procedure is to adapt the classification algorithm to make it compatible with homomorphic encryption without modifying its behavior. One of the most used solutions is to replace the activation function with the squared function which has multiplicative depth 1. Due to the low degree polynomial activation function, the multiplicative depth of the neural network is maintained acceptable during the calculations making the YASHE encryption scheme efficient enough [38].

However, the squared function has an unbounded derivate so that a strange behavior during the training phase can be verified making this protocol suitable only for small neural networks. Moreover, the accuracy for neural networks with more than 2 non-linear layers is very low. Even though there are the described limitations, it has to be taken into account that in nowadays applications small neural networks are still the most common [39].

3.3 Related works

Using a Homomorphic Encryption scheme to preserve the privacy of data during the computation is a strategy introduced in [24]. While the first methods thought allowed only additions and in a few cases, only multiplications, in [23] has proposed a solution able to manage both operations. In the following years, many other important schemes have been created such as Brakerski-Gentry-Vaikuntanathan (BGV) [31], Cheon-Kim-Kim-Song (CKKS), and the Brakerski/Fan-Vercauteren (BFV) [29] used in this project.

For what concerns the HE in machine learning, [40] is based on Cryptonets, neural networks that can be applied to encrypted data and, since it is impossible to apply HE for non-linear calculations, it introduces several tactics to approximate these types of fundamental layers. The framework proposed in the next chapter is similar to Cryptonets in terms of intention though it uses a different HE scheme, the Brakerski/Fan-Vercauteren (BFV) instead of the YASHE [38], and a different strategy to deal with the non-linearity.

Works related to this thesis are [41] and [42] that, exactly as in the [40], make use of HE to guarantee the privacy of an eventual client data while going through the layers of the neural network owned by a server. Obviously, neither of these works deals with Spiking Neural Networks but focuses only on classical CNNs and on how to merge them with these security measures. In addition, another main difference here is the insertion of a Multi-party computation approach to avoid the approximation of activation functions that characterize the models used, especially

the spiking one. This solution drastically reduces the security measure effectiveness to leave unaffected the behavior of the spiking networks.

For what concerns SNNs, due to the energy advantages that they imply, they have been analyzed in many works as [43], [44], [45], especially in image recognition tasks ([46], [47]). The main issue explored in the majority of the papers and articles regarding SNNs is the training phase and the troubles it causes due to the non-differentiability of the spiking activation functions, which makes the gradient descent techniques, typically used for traditional ANNs, unusable.

One of the most successful training algorithms is the ANN-SNN conversion due to its high performance. Essentially, ANNs are pre-trained and then converted to SNNs replacing their activation function (ReLU) with the Leaky- Integrate-and-Fire (LIF) activation by means of weight or threshold balancing ([48]).

The algorithm proposed in [48] has been employed in [49] to study another side of the privacy-preserving problem while using SNNs. In fact, conversion is usually obtained considering the chance to access the whole training dataset which however may contain sensitive information that enterprises would not be disposable to share. A second issue regards the leakage of the classes features since it is possible to obtain a representative class image from network parameters by utilizing gradient backpropagation allowing an attacker to find blind spots in neural structures, generate strong adversarial perturbations, and disguise a system in the real-world. The first problem is defined as data leakage while the second is called class leakage and in [49] both are addressed to build a secure neural system.

The conversion method is based on copying the weight parameters of the ANN model previously trained to an SNN and computing the maximum activation across all time steps for every layer to finally set the firing threshold to that value. Thus, data leakage is solved with a strategy of Data-Free Conversion which is the generation of synthetic images without accessing the actual data but simply from a pretrained ANN. Due to the conversion algorithm used, false images are obtained considering the class relationships from the weights of the last fully-connected layer. Once the synthetic dataset is fulfilled the system is data leakage free but still vulnerable to class leakage. It is possible, indeed, to recover the original ANN by accessing the weights of the SNN and turning the LIF activation function back to the ReLU. As a security measure, [49] proposed the Encryption of the SNN by means of temporal spike-based learning rule, with a small number of time steps to avoid resource consumption and guarantee a certain level of energy efficiency. What has been gained in this way is some data that is really hard to be interpreted in the spatial domain, encoded using a Poisson spike generator without huge information loss that can be used to perform the training phase with a spike-based learning method.

Lastly, it is used a distillation technique to avoid the overfitting of the network and consequentially the performance degradation caused by the reduced number

of training samples. In fact, in [50] it has been proved that distillation of the knowledge from ANNs to SNNs improves the generalization ability of the network solving the troubles of using a small number of data samples.

Furthermore, [49] presented 2 types of attacks to verify the effectiveness of the privacy-preserving structure built against class leakage. The first scenario is based on the chance of recovering the original ANN taking the weights of the SNN and so reconstructing class representation by backpropagation. The second scenario is a direct attack on the SNN, specifically, the backpropagation of gradients to reconstruct class representation. In this case, it is necessary to follow a proper procedure based on approximated gradient functions for LIF neurons due to the fact that it and the Poisson spike generator are non-differentiable. Overall, the first attack is managed by the post-conversion encryption training since it makes the weight of SNN encrypted in the spatial domain while the second one showed that SNNs are robust to gradient-based security attacks due to the discrepancy between real gradients and approximated gradients.

To sum up, the privacy-preserving method in [49] is built by the 3 strategies described: Data-free Conversion, Encryption Training, and Distillation, and no significant performance drop across all datasets is verified.

Moreover, the results show that PrivateSNN is more efficient from the energy consumption point of view than both ANN and standard converted SNN.

This work has not dealt with SNN training problems or its security vulnerabilities because the Norse library used manages it by means of the Superspike algorithm. It is a derivation of the surrogate gradient approach, based on the approximation of the partial derivative of the hidden unit results as the product of the filtered presynaptic spike train and a nonlinear function of the postsynaptic voltage instead of the postsynaptic spike train [51].

Chapter 4

Encryption Framework Design

This section introduces a possible solution to the problem of maintaining the privacy of data while computed by a convolutional neural network. It has been thought a protocol that relies on a predictive model held by a server employed by a client without revealing any information about the client's data. Essentially, this project aims to compare a convolutional neural network and a spiking neural network in terms of reliability and efficiency while working on encrypted data. The models in the cloud need to be trained first on unencrypted datasets to obtain the optimized parameters, weights, and biases, and then adapted to make possible the encrypted calculations. To handle the task, it has been used Python 3, more precisely Pytorch [52], to deal with the neural networks, NORSE [53], for the spiking models, and Pyfhel [54], which relies on SEAL [55], for the encryption functions [41]. The comparison between the two generations of networks is performed through three datasets: MNIST, FashionMNIST, and CIFAR10.

4.1 Network model

Due to the complexity of the encrypted calculations, it has been chosen as a network model one of the simplest and shortest architectures, the LeNet5, in order to maintain the simulation's time and computational cost reasonable.

4.1.1 Datasets

To understand how the structure of the model has been built, it is fundamental to notice the main features of the used datasets.

The first one, MNIST, is a database of handwritten digits, so 10 classes, and has a training set of 60,000 examples, and a test set of 10,000 examples. Known as a subset of a larger set from NIST, it is one of the easiest to deal with since it does not need a lot of preprocessing and formatting. In fact, MNIST is often the first dataset researchers try. Its images contain grey levels and are centered in a 28x28 image [56].

Fashion-MNIST is a dataset of Zalando's article images. More precisely, it is made of a training set of 60,000 examples and a test set of 10,000 examples and, exactly like in MNIST, each example is a 28x28 grayscale image for a total of 10 classes. Each of the 784 pixels has a value associated with it, an integer between 0 and 255, useful to evaluate the brightness of the pixel. Each training and test example is assigned to one of the following labels: T-shirt/top, trousers, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot [57].

CIFAR-10 is the hardest dataset used in this project since it is composed of 60000 32x32 color images in 10 classes, with 6000 images per class. Since it is not based on grayscale images, it needs three channels to describe the RGB color, instead of the one used for the previous datasets. The training set consists of 50000 images while there are 10000 images for the test set. Overall, the 60000 images are divided into 6 batches of 10000 samples, 5 for the training phase and 1 containing 1000 randomly-selected images from each class for the test phase. Only the training batches may contain more images from one class than another but in total, they contain exactly 5000 images from each class. The 10 classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck; and are completely mutually exclusive with no chances of overlap [58].

4.1.2 LENET5

The features of the dataset to analyze are fundamental to determining the characteristics of some layers of the network. Indeed, there are a few differences between the LENET5 architecture for MNIST and FashionMNIST datasets, which are similar, and the one suitable for CIFAR-10.

Considering the first two datasets, the input of the network is a tensor of one channel, 28×28 matrix of pixels, where each pixel has its own gray level (0-255), and the label which describes the class of the image with an array of 10 values in one hot format.

Thus, the first layer of the network, the Convolution layer needs to have 1 input channel for the 28×28 matrix, and perform the convolution using a kernel of size (5, 5) and a stride of (1, 1) obtaining 6 output channels with the shape 24x24 as

described by the following equations:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[0] - dilation[0] \times (kernel\_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor \tag{4.1}$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor \tag{4.2}$$

where H and W are the height and weight of the matrix, padding has a default value of 0 and the dilatation, which controls the spacing between the kernel points, has a default value of 1.

What comes after is the Activation layer to apply non-linearity to the network. It has been chosen the ReLU activation function which applies the rectified linear unit function element-wise:

$$ReLU(x) = (x)^+ = max(0, x) \tag{4.3}$$

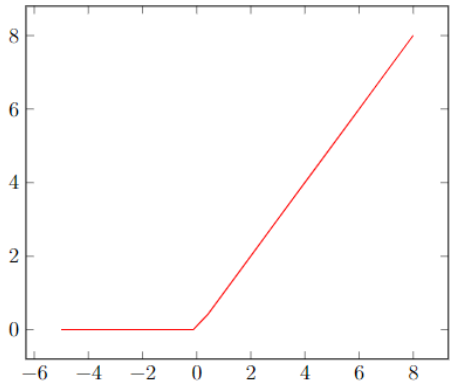


Figure 4.1: ReLU activation function

The output has the same shape as the input.

The Average pooling layer performs a 2D average pooling over an input signal with a window characterized by a kernel size of (2,2) a stride of (2,2) and a 0 padding. The shape after this layer is determined by:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[0] - (kernel\_size[0])}{stride[0]} + 1 \right\rfloor \tag{4.4}$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[1] - (kernel\_size[1])}{stride[1]} + 1 \right\rfloor \tag{4.5}$$

Hence the output has dimensions of 12×12. It has been selected this type of pooling because the Max pooling layer is incompatible with homomorphic encryption computation.

Here the network has a second convolution layer with the same parameters as the first one, a kernel of size (5, 5) and a stride of (1, 1). Thus, the 6 input channels become 16 output channels of shape 8x8.

A new ReLU activation function followed by an average pooling layer is performed exactly as before, giving the output the shape of 4x4.

After the flatten layer, the matrix is flattened into an array of length 256 (4x4x16).

At the end of a fully connected layer, the array has 120 nodes.

In conclusion, there is a sequence of ReLU, a Linear to turn the vector a dimension of 84, another ReLU, and a final linear layer that reduces the tensor to 10 output nodes, each one corresponding to one of the 10 classes.

Once the input has passed through all the model layers, it is applied on the 10 output nodes a Log Softmax function that simply represents the logarithm of the softmax function. Essentially, the evaluation of log probabilities means representing probabilities on a logarithmic scale, instead of the standard [0,1] interval. Thus, each node has an associated value, the highest of which represents the selected prediction.

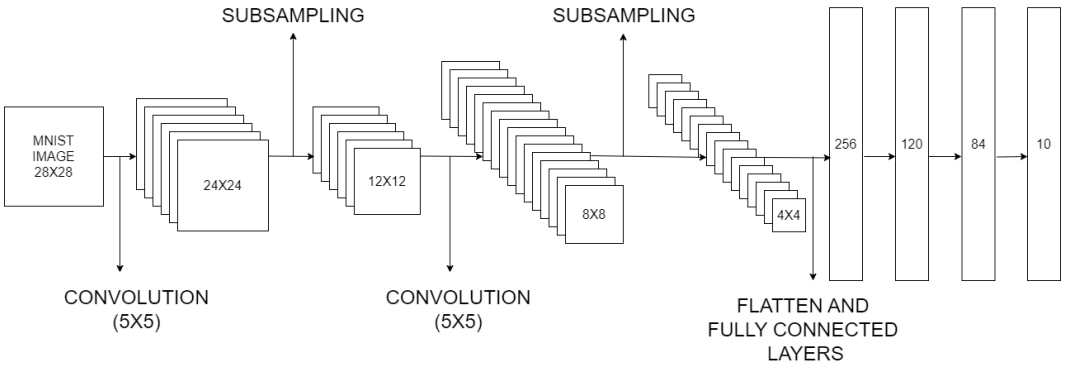


Figure 4.2: LeNet5 Architecture for MNIST and FashionMNIST dataset

The described network needs some modifications if the input images are from the CIFAR-10 dataset. In fact, as previously described, these data are in the form of a tensor with 3 channels and a matrix of 32x32 pixels. So the first layer needs to deal with the initial 3 channels as input and it shows 6 outputs of 28x28 nodes. The first pooling layer has as output a tensor with a matrix of 14x14 while at the end of the second convolution the tensor has the dimensions of 16 outputs of 10x10. Following the rest of the architecture, the tensor has a matrix of 5x5 after the second mean pooling and a flattened dimension of 400 (5x5x16) after the flatten

layer. The rest of the execution is exactly as already discussed for the MNIST and FashionMNIST model.

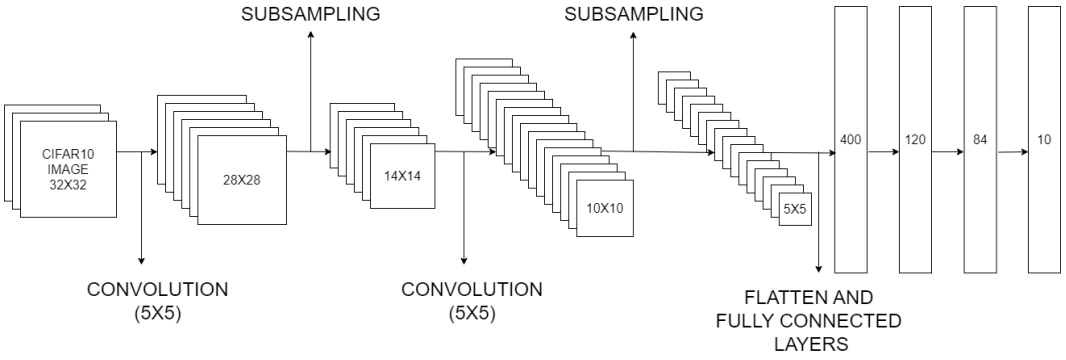


Figure 4.3: LeNet5 Architecture for CIFAR10 dataset

4.1.3 SNN version of LENET5

In the previous section, it has been described a LENET5 structure, one of the smallest, characterized by the ReLU activation function. In the following, it is built the same architecture, with the same dimensions and complexity, but with the spiking activation function turning the classical and convolutional LENET5 into a Neural Network of the third generation: SNN. Since the aim is to obtain a good comparison between the convolutional model and the spiking model, it is fundamental to maintain the same structure dimensions and the same characteristics of each layer (pooling, linear, and the others with the same kernel size and stride) so that the only changes are represented by the activation function used, represented by the module LIFCell, and the last linear layer which is substituted by LILinaerCell, both from Norse library.

Neuron Model

To define the spiking activation function it is necessary to clarify which of the possible neuron model to use. As briefly summarized in the first chapter, the model that best emulates the human brain is the Hodgkin-Huxley but it has been ignored due to its complexity. It is necessary to consider that the homomorphic encryption evaluations are managed at a heavy computational cost and consume a huge amount of time, making the simpler neuron model the best solution. Among the remaining models, Izhikevich and Leaky integrate-and-fire are the most feasible, and the latter, probably the most popular, has been chosen to fulfill the simulations.

LIF works by combining the Norse leaky-Integrator model with spike thresholds to produce events (spikes). The Leaky integrator unit evaluates a leaky neuron

membrane that integrates incoming currents over time without dealing with spiking phenomena. Thus, the neuron adds up incoming input current, losing a bit of it at every timestep. Its behavior is modeled by means of the 2 following ODE equations:

$$\dot{v} = 1/\tau_{\text{mem}}(v_{\text{leak}} - v + i) \tag{4.6}$$

which illustrates how the membrane voltage varies over time with the addition of the current (i) and the leakage of the constant amount v_{leak};

$$\dot{i} = -1/\tau_{\text{syn}} i \tag{4.7}$$

which describes the way the current flows into the neuron in every timestep.

The time constant present in both equations controls how fast the voltage and the current change, in fact, when it is large there are small variations of these quantities. However, in Norse, the inverse of this time constant is portrayed for convenience which means that a large inverse time constant implies rapid variations while a small inverse time constant implies slow variations. LIF model adds to the LI one the firing event. In fact, if the neuron voltage increases over a defined threshold(v_{th}), the firing phenomenon occurs.

$$z = \Theta(v - v_{th}) \tag{4.8}$$

To sum up, the used module, the LIFCell, computes a single euler-integration step of a LIF model without recurrence implementing one integration step of the described ODE (equations 4.6 and 4.7) with the jump condition portrayed by the equation 4.8 and the transition:

$$v = (1 - z)v + zv_{reset} \tag{4.9}$$

The behavior of this activation function can be defined by means of a method, usually the Superspike which synthesizes a gradient approach through the Heaviside step function:

$$H[n] = \begin{cases} 0, n <= 0 \\ 1, n > 0 \end{cases} \tag{4.10}$$

Overall, this type of network operates on spikes combined by linear transformations and integrated by neuron circuit models obtaining an architecture that needs to work on temporal data. More specifically, it can be seen as a recurrent neural network (RNN) where time is identified with the sequence dimension explicitly and only binary values are exchanged between layers at each timestep.

It has been described an activation function able to reproduce approximately the behavior of a biological brain but neurons need another variable to work perfectly. The membrane state is indeed fundamental to lead the neuron to spike if the voltage is above a threshold because it keeps track of membrane voltage as long as

timesteps succeed. Without this state, the membrane potential can not be updated and the neuron would never produce any spikes.

In conclusion, since it is necessary to notice the evolution of timesteps, which is portrayed explicitly by a sequence of a defined length (how many milliseconds the analysis is desired to last), it is clear that the spiking neural networks need encoded input tensors with an additional dimension that reports this observation length.

LIF Parameters

Analyzing the LIFCell module, It has been illustrated how the Leaky integrate-and-fire neuron behaves through some features called LIF-Parameters. These values are indispensable requirements to describe the neuron unit since they give an indication of its biological qualities and of the way it responds to stimuli. The Norse default parameters are:

- $\tau_{syn\_inv} : torch.Tensor = torch.as\_tensor(1.0/5e-3)$
- $\tau_{mem\_inv} : torch.Tensor = torch.as\_tensor(1.0/1e-2)$
- $v_{leak} : torch.Tensor = torch.as\_tensor(0.0)$
- $v_{th} : torch.Tensor = torch.as\_tensor(1.0)$
- $v_{reset} : torch.Tensor = torch.as\_tensor(0.0)$
- $method : str = \text{"}super\text{"}$
- $alpha : float = torch.as\_tensor(100.0)$

where alpha and method are relevant hyperparameters for surrogate gradient computations. These parameters are clearly a simplification of the real biological neuron parameters that are evaluated as follows:

- $\tau_{syn\_inv} : torch.Tensor = torch.as\_tensor((1/0.5)$
- $\tau_{mem\_inv} : torch.Tensor = torch.as\_tensor(1/20.0)$
- $v_{leak} : torch.Tensor = torch.as\_tensor(-65.0)$
- $v_{th} : torch.Tensor = torch.as\_tensor(-50.0)$
- $v_{reset} : torch.Tensor = torch.as\_tensor(-65.0)$

To optimize the calculations for what concerns the observation length, it has been chosen to employ the default parameters reducing only the threshold voltage to make the neuron fire more easily:

- $\tau_{syn\_inv} : torch.Tensor = torch.as\_tensor(1.0/5e-3)$
- $\tau_{mem\_inv} : torch.Tensor = torch.as\_tensor(1.0/1e-2)$
- $v_{leak} : torch.Tensor = torch.as\_tensor(0.0)$
- $v_{th} : torch.Tensor = torch.as\_tensor(0.5)$
- $v_{reset} : torch.Tensor = torch.as\_tensor(0.0)$

Spiking encoding

As previously illustrated, the most evident new feature of spiking neural networks is the way they work on temporal data encoded as spikes. Since, in machine learning, the most common datasets do not contemplate any encoding, to provide the required additional temporal dimension it is indispensable to include an encoding step. In fact, SNN expects to operate on input spikes seen simply as a sequence of tensors containing binary values. Norse deals with encoding providing several encoding possibilities.

ConstantCurrentLIFEncoder - Input currents are turned into constant voltage currents and a simulation of the spikes that occur during a selected window of timesteps (seq_length) is performed.

PoissonEncoder - Inputs are assumed in the range [0,1] and are turned onto a tensor of one dimension higher of binary values (spikes).

PoissonEncoderStep - Inputs are considered built as values in the range [0,1] and encoded in a tensor of binary values (spikes).

PopulationEncoder - Inputs are seen as population codes where each value is depicted with a list of numbers, evaluated by a radial basis kernel. Lists are characterized by a length equal to out_features and an activity that increases if a number gets close to its "receptive field".

SignedPoissonEncoder - Inputs are assumed in the range [-1,1] and encoded into a tensor of one dimension higher of values in -1,0,1, representing signed spikes.

SpikeLatencyEncoder - Inputs are encoded measuring the time needed for a neuron to perform the first spike

SpikeLatencyLIFEncoder - Inputs are encoded by the time the first spike occurs. Its behavior looks like the ConstantCurrentLIFEncoder where the LIF is considered with an infinite refractory period.

Most of these encoding strategies do not produce spike patterns that are necessarily biologically realistic. This result can be obtained by employing cells with varying thresholds and a finer integration time step.

Decoding

After the encoding and the evaluation fulfilled by the network, what is achieved is a tensor of 10 membrane voltage traces. Usually, those values are decoded into a probability distribution by determining the maximum along the time dimension before computing the softmax. Other ways of decoding the final results are to consider the average membrane voltage in a selected window of timesteps or to use a LIF neuron output layer and consider the time to the first spike or to consider only the membrane trace at the last measured time step.

4.2 Training phase

Every network needs to be trained to update weights and biases and to be able to perform the classification or other kinds of tasks. Though training is an indispensable step, it can be even an interesting first way to compare the classical convolutional LENET5 and the spiking version of that structure. Moreover, it is required to pick an encoding method between all the strategies previously presented to perform the simulations. Thus, it has been chosen to execute the training step for the SNN LENET5 with 3 different encoding algorithms provided by Norse: ConstantCurrentLIFEncoder, PoissonEncoder, and SpikeLatencyLIFEncoder. The training has been obviously performed for all the 3 datasets discussed before, while the comparison between encoding functions has been fulfilled using MNIST.

4.2.1 Training hyperparameters

The parameters used for this phase are the Optimization algorithm, the loss function, and the number of epochs the training needs to last.

As optimization algorithm, it has been selected the Adam optimizer because it is usually considered the best solution. It essentially determines one single learning rate for each parameter and it adapts each rate independently. In this case, it has been applied a learning rate of 0.001.

For what concerns the loss function, it has been used the negative log-likelihood loss (NLLLoss) that is useful to train a model in case of a classification problem. Furthermore, there can be used some optional arguments to deal more efficiently with unbalanced training sets.

The input of this function is expected to be in the form of log probabilities for each of the C classes. In fact, input has to be a Tensor of size either $(minibatch, C)$

Algorithm 1 Adam optimizer algorithm. All operations are element-wise, even powers. Good values for the constants are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. ϵ is needed to guarantee numerical stability.

procedure ADAM($\alpha, \beta_1, \beta_2, f, \theta_0$)
  ▷ α is the stepsize
  ▷ $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates for the moment estimates
  ▷ $f(\theta)$ is the objective function to optimize
  ▷ θ_0 is the initial vector of parameters which will be optimized
  ▷ Initialization
  $m_0 \leftarrow 0$ ▷ First moment estimate vector set to 0
  $v_0 \leftarrow 0$ ▷ Second moment estimate vector set to 0
  $t \leftarrow 0$ ▷ Timestep set to 0
  ▷ Execution
  while θ_t not converged **do**
    $t \leftarrow t + 1$ ▷ Update timestep
    ▷ Gradients are computed w.r.t the parameters to optimize
    ▷ using the value of the objective function
    ▷ at the previous timestep
    $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
    ▷ Update of first-moment and second-moment estimates using
    ▷ previous value and new gradients, biased
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
    ▷ Bias-correction of estimates
    $\hat{m}_t \leftarrow \dfrac{m_t}{1 - \beta_1^t}$
    $\hat{v}_t \leftarrow \dfrac{v_t}{1 - \beta_2^t}$
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \dfrac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ ▷ Update parameters
  end while
  return θ_t ▷ Optimized parameters are returned
end procedure

or $(minibatch, C, d_1, d_2, ..., d_K)$ with $K \geq 1$ for the K-dimensional case. Log probabilities are a result of the application of the LogSoftmax layer at the end of the network computation so to use this loss function is mandatory to add this processing step. However, another solution might be to avoid the LogSoftmax using directly the CrossEntropyLoss function which instead works on raw, unnormalized scores for each class. The expected target is a class index in the range $[0, C - 1]$, while the loss depends on the argument called reduction.

4.2.2 Confrontations between encoders

The training environment described has been used for both LENET5 and SNN LENET5 models, but for the latter, as previously portrayed, it is necessary to define the length of the timesteps sequence to be analyzed. This sequence length has been set to 30 milliseconds.

Thus, with the aim of comparing the effects of the encoders on the accuracy of the spiking model, it has been firstly executed the training phase for the SNN with ConstantCurrentLIFEncoder for the MNIST dataset. The following table portrays the training features when a constant current is used to produce input spikes:

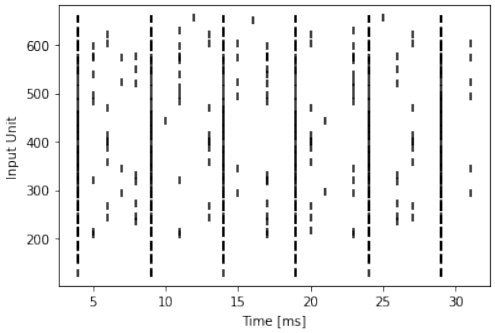


Figure 4.4: Spike pattern ConstantCurrentLIFEncoder

MNIST - ConstantCurrentLIFEncoder	
Optimizer	Adam
LOSS Function	NLLLoss
Learning rate	0.001
Epochs	10
Results:	
Accuracy	98.38%
Required time	255,7 sec

The results in the case it is used the PoissonEncoder, which encodes the inputs into Poisson spike trains building a more biologically plausible condition, is illustrated in the table below. The conditions are exactly the same as before.

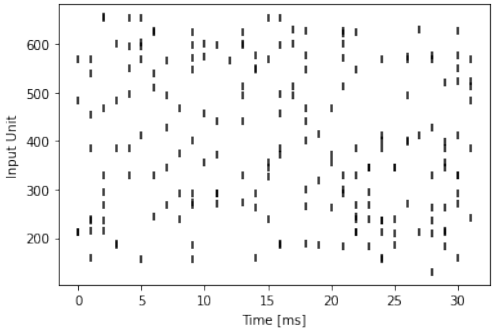
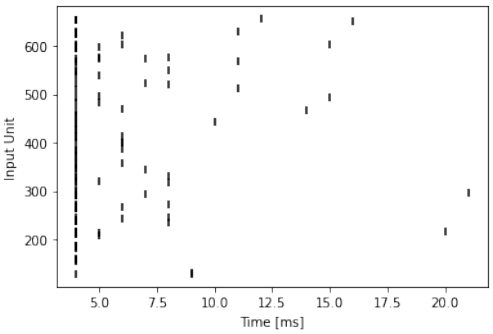


Figure 4.5: Spike pattern PoissonEncoder

MNIST - PoissonEncoder	
Optimizer	Adam
LOSS Function	NLLLoss
Learning rate	0.001
Epochs	10
Results:	
Accuracy	96.3%
Required time	701,68 sec

Lastly, the SpikeLatencyLIFEncoder makes each input neuron spike only the first time the threshold is exceeded. The results obtained again with an observation of $30ms$ are illustrated below.



MNIST - SpikeLatencyLIFEncoder	
Optimizer	Adam
LOSS Function	NLLLoss
Learning rate	0.001
Epochs	10
Results:	
Accuracy	98,18%
Required time	272,39 sec

Figure 4.6: Spike pattern SpikeLatencyLIFEncoder

The values of the final tensor achieved by the network have been decoded in each of the cases into a probability distribution by determining the maximum along the time dimension before computing the softmax.

It appears evident from the tables that the encoding strategy that gives the best accuracy ad that takes less time is the ConstantCurrentLIFEncoder and for this reason, it has been used for all the following simulations.

However, as with any machine-learning approach, it can be difficult to find the combination of network features, hyperparameters, decoding, and encoding scheme to achieve the highest performances. Though most of the time it is not clear at first what might be the best choices, it is predictable that the PoissonEncoder does give the worst results since the encoded inputs converge with $1/\sqrt{Timesteps}$. So, due to the low number of timesteps, only $30ms$, the performance of this encoder is the poorest.

To sum up, the differences in the efficiency of the design and the training of artificial neural network classifiers are related to hyperparameters, the dimensionality of the classification layer, and weight initialization. There are many uncertainties about their interplay for spiking neural network architectures.

Furthermore, in the case of SNN, it has to be considered even the addition of the decoding and encoding schemes and the number of integration timesteps that are proper characteristics of this new generation of networks. Overall, it is still unclear how to efficiently choose these features.

4.2.3 Trained Models

Once the hyperparameters and the model features have been set the training step can be performed. The following tables report the accuracy obtained for the LENET5 architecture for each dataset. Since MNIST is the easiest dataset to learn, it is achieved an accuracy of over 98% with only 10 Epochs of training.

MNIST - LENET5	
Optimizer	Adam
LOSS Function	Negative log-likelihood loss
Learning rate	0.001
Epochs	10
Results:	
Accuracy	98,58%
Required time	123,5 sec

Table 4.1: Training CNN - MNIST

FashionMNIST has similar features with respect to MNIST but it is a little bit harder to work on. Thus, the process requires more epochs and reaches a lower accuracy.

FashionMNIST - LENET5	
Optimizer	Adam
LOSS Function	Negative log-likelihood loss
Learning rate	0.001
Epochs	30
Results:	
Accuracy	89,63%
Required time	366.71 sec

Table 4.2: Training CNN - FashionMNIST

CIFAR10 is clearly the most troublesome of the 3 and though its processing has lasted for 100 epochs, it still gives the worst results in terms of accuracy.

Below there are the tables (4.4, 4.5, 4.6) describing the training effects for the spiking version of the LENET5 in the same conditions as the convolutional model. As illustrated in the previous section, it has been chosen the ConstantCurrentLIFEncoder scheme for the encoding task with an observation length of $30ms$. The decoding strategy used is the Log-softmax function applied over the maximum along the time dimension.

CIFAR10 - LENET5	
Optimizer	Adam
LOSS Function	Negative log-likelihood loss
Learning rate	0.001
Epochs	100
Results:	
Accuracy	57.14%
Required time	1241.09 sec

Table 4.3: Training CNN - CIFAR10

The MNIST is the only dataset that makes the SNN behave almost like the original LENET5 with an accuracy slightly lower.

MNIST - SNN LENET5 - ConstantCurrentLIFEncoder	
Optimizer	Adam
LOSS Function	Negative log-likelihood loss
Learning rate	0.001
Epochs	10
Results:	
Accuracy	98.38%
Required time	255,7 sec

Table 4.4: Training SNN - MNIST

For FashionMNIST, the computation presents a lower accuracy but is still similar to the LENET5 model.

For what concerns CIFAR10, the proposed SNN has shown some difficulties, and the accuracy after an extended training phase has reached a particularly low value.

It is important to notice that CIFAR10 is the slowest to be computed and the most difficult to learn since its inputs have 3 channels and a 32x32 pixels matrix instead of 1 channel and a 28x28 pixel matrix typical of the first 2 datasets.

Though the epochs and the structure of the models are the same for the 2 networks, it appears evident that the CNN achieves the best outcomes in terms of accuracy and time required. The reason why SNN is slower is due to its need to analyze the sequence of 30 timesteps.

Overall, on a small supervised learning task, it is relatively easy to define a spiking neural network that behaves similarly to a non-spiking artificial network. Essentially, the SNN architecture used corresponds to the one of an artificial neural network that might be involved with the non-linearities replaced by spiking units.

FashionMNIST - SNN LENET5 - ConstantCurrentLIFEncoder	
Optimizer	Adam
LOSS Function	Negative log-likelihood loss
Learning rate	0.001
Epochs	30
Results:	
Accuracy	83,92%
Required time	767,41 sec

Table 4.5: Training SNN - FashionMNIST

CIFAR10 - SNN LENET5 - ConstantCurrentLIFEncoder	
Optimizer	Adam
LOSS Function	Negative log-likelihood loss
Learning rate	0.001
Epochs	100
Results:	
Accuracy	42,66%
Required time	2388,52 sec

Table 4.6: Training SNN - CIFAR10

4.3 Inference and HE parameters

Once the training step is completed, it is possible to implement the inference phase which has the aim of keeping the privacy of a client's information untouched. It operates in an encrypted environment and is based on computations over encrypted data. There are some limitations in the types of calculations that can be fulfilled and there are some constraints, due to the noise added as a security measure, on the number of encrypted operations that can be executed without corrupting the cipher text.

Thus, the Brakerski/Fan-Vercauteren (BFV) scheme is used to perform additions and multiplications on cipher texts and polynomials. Due to the mathematical limitation of HE, it is not possible to use the exactly same network as the one in the training phase. In fact, Softmax operation can not be performed on encrypted data, due to its non-linearity so it can not be used with this type of encryption. However, since it is a monotonically increasing function, it can be avoided as the cipher texts of the outcomes of the last network layer can be evaluated to give the prediction.

Moreover, it is fundamental to notice that the complexity of the calculations

in the encrypted domain is a lot higher than before making it essential the focus on keeping limited the number of multiplications. In this case, a multi-party computation strategy has been used, mostly because of the non-linearity of the activation functions, solving the issue of the raising noise and obtaining similar results with respect to the plain text version.

4.3.1 HE setting

As portrayed in chapter 2, the homomorphic encryption scheme used is the Brakerski/Fan-Vercauteren (BFV) one. Like the other methods, it relies on an encryption function E, and on its decryption function, D, that if applied on a function f it has to be possible to find a function g such that $f(x) = D(g(E(x)))$. Moreover, this encryption strategy depends on the context built by means of 3 important parameters:

- m, the Polynomial modulus degree, which requires to be set as a positive integer power of 2 since it is the degree of the cyclotomic polynomial;
- t, the Plaintext modulus, set as a positive integer, it is the module of the coefficients of the polynomial ring;
- q, the Ciphertext coefficient modulus, set as a large positive integer, is the product of distinct prime numbers representing the modulo of the coefficients of the polynomial ring.

These 3 fundamental parameters characterize the encryption scheme by making it possible to generate the keys that allow encryption and decryption. The are several keys that can be used in an encryption-based computation, but the most important ones are the public key used especially for the encryption step, and the secret key responsible for the decryption.

Furthermore, the scheme is based on a quantity, called Noise Budget, measured in bits, which gives an indication, in each stage, of how many operations in the ciphertext domain can be performed without compromising the truthfulness of the outcomes once decrypted. In fact, when encryption is applied, some noise is added to the ciphertext in order to create from the same value, 2 different corresponding ciphertexts using always the same public key. Thus, Noise Budget (NB) is the amount of noise affordable by the system and it is partially consumed every time an operation is fulfilled on a ciphertext. Not all the types of calculations are equal for what concerns NB, in fact, additions and multiplications between ciphertext and plaintext dissipate a small dose of NB, while multiplications between ciphertexts, instead, require a huge amount of NB. Overall, it is clear that if NB reaches 0, decrypting the obtained result of encrypted evaluations generates an incorrect outcome.

To sum up, the strategy that relies on the application of Homomorphic encryption on the client data when computed by a neural network model owned by a server depends on encryption, decryption, and public and secret key generation.

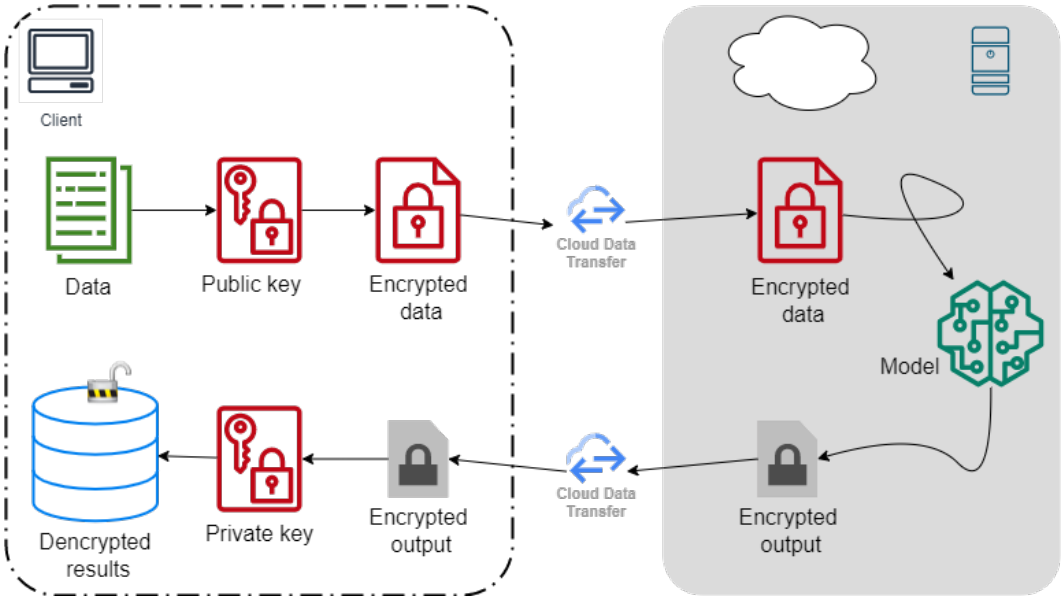


Figure 4.7: Secure Cloud Computing

It is fundamental to notice that the neural network evaluations are executed in the cloud due to its typically huge demand in terms of computational cost and memory occupation. Moreover, due to the privacy-preserving algorithm requirements, the model needs to be approximated by using only addition and multiplication to process the ciphertext. Once approximated, the model requires an additional step of encoding according to the encryption scheme selected and set through the parameters described. The encoding step is essential to make the model able to compute the ciphertext by means of the conversion of the plain parameters of the model.

Framework

The interest of this project is to analyze the results of a classification problem of input images, solved with the two types of described networks, the convolutional and the spiking one. The client's data, in this case in the form of images from the discussed datasets, are encrypted following the HE scheme exposed in chapter 2 set by means of the 3 selected encryption parameters. Thus, the plain image is encrypted through the function E and the generated public key. The client is even responsible for and the only entity able to perform the decryption of the encrypted

output received from the server, obtaining the plain classification label by means of the generated secret key.

For what concerns the server, the neural network models are mostly composed of additions and multiplications such that their structures are naturally fitting to HE methods. However, they can not compute operations on encrypted data without a suitable approximation. In fact, since only additions and multiplications are allowed in the HE scheme, only polynomial functions can be performed straightforwardly, making it necessary either to adapt every type of non-polynomials operations, turning them into a polynomial function, or to substitute them with other kinds of calculations.

The neural architectures described above show, for instance, the use of the mean pooling layer and not the max pooling one since the first can be converted to an addition and a division, which is essentially a multiplication by $1/pooling\_size$ which is a fixed value, while the latter is a non-polynomial evaluation.

Unfortunately, activation functions are fundamental for both the networks presented. In the LENET5 there is the Rectified linear operation (ReLU) while the SNN version is based on the spiking activation function, the Leaky integrator and fire one (LIF).

The most common strategy, when it is necessary to deal with ReLU, that is non-polynomial, is to approximate the model by means of another function used in its place, like the square function. The latter is a polynomial function and has a non-linear behavior as required to maintain the learning capabilities of the networks. However, it has a not negligible drawback, in fact, its derivative is unbounded, which can cause some strange behavior during the training phase while running gradient descent. Using this alternative layer in place of the ReLU activation function requires, clearly, to execute the training step of the new approximated network due to the fact that the weights of the original model can not be considered consistent after the replacement of activation functions or other layers. So, the training algorithm has to be performed again in the same conditions, hyperparameters, settings and for all the datasets as in the original case. The new accuracy obtained can obviously be different from the one originally achieved.

Changing the activation function is possible for the LENET5 model but it is impossible to replace the spiking activation of the SNN since it is the model's main characteristic. It has been chosen to use a Multi-party computation (MPC) approach to solve this issue. Once the layers of the models are determined, they can be encoded according to the HE scheme.

Figure 4.8 shows how a multiparty-like approach has been chosen to deal with the activation functions. More specifically, the client builds the encryption context by choosing the relative parameters of the HE scheme and the security level. Then the image is encrypted by means of the public key generated through the HE method and its parameters. The encrypted image is then sent to the server for

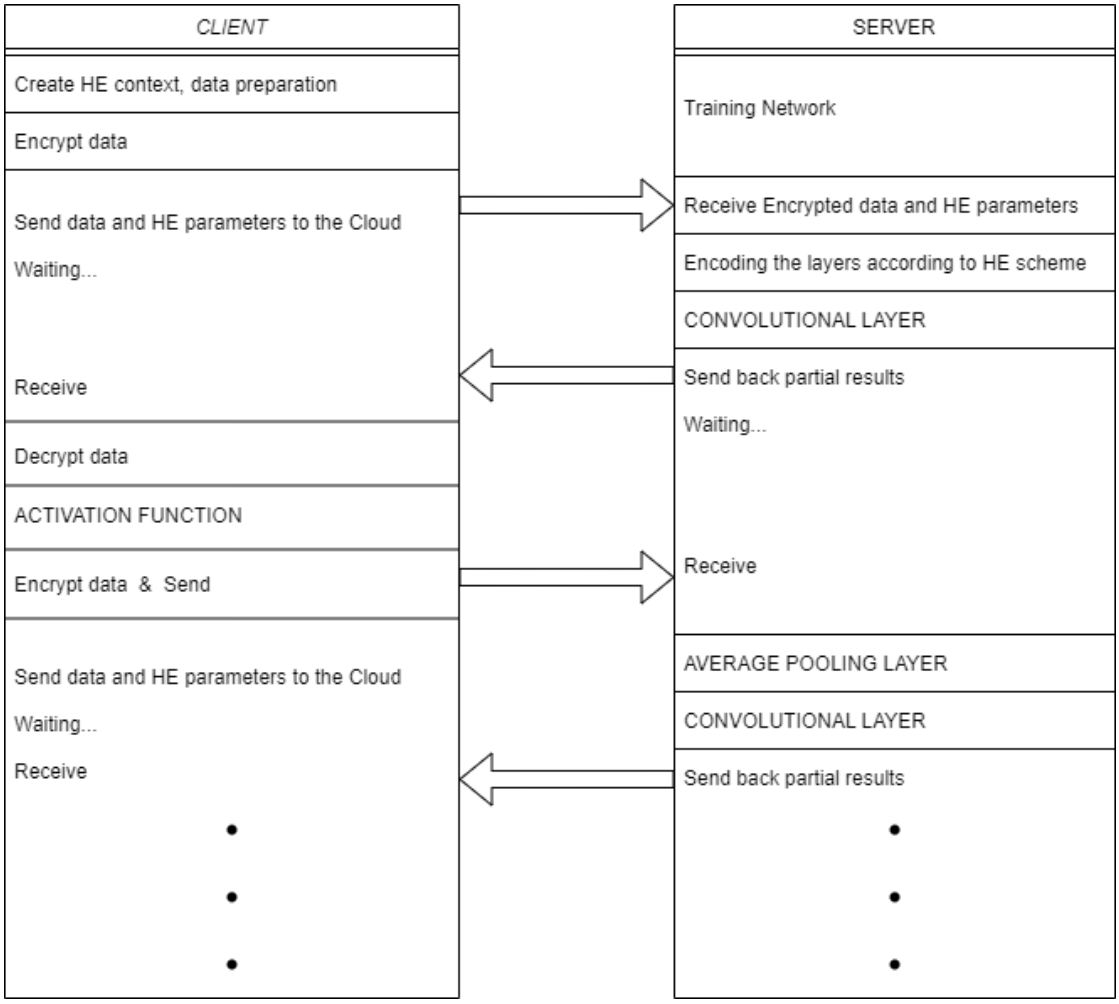


Figure 4.8: Client and Server dynamic approach

classification together with the HE scheme features in order to perform the encoding of the polynomial layer of the network and to obtain a model able to evaluate the encrypted calculations.

Once it is necessary to apply the activation function, ReLU or LIF, the server sends the results of the internal neurons back to the client which is responsible for the decryption and the execution of the non-linear operations. At the end of the process, the client encrypts the outcomes of the activation functions again and sends the data to the cloud in order to go through the rest of the network.

In this way the server can perform the estimation of the convolution, the flatten, the mean pooling, and the linear layers once their parameters have been properly encoded, while the client has to perform the activation functions and the decryption, using the secret key, of the partial and the final results to gain the transparent

classification. As described in the second chapter this approach has its limitations and its security vulnerabilities, especially for what concerns the server side.

4.3.2 Encryption parameters Selection

As anticipated above, choosing the encryption parameters means building a context for ciphertext and plaintext operations and determining the correctness of results. There are two main factors in deciding the parameters, the noise growth evaluation, and the efficiency.

In fact, a particular set of these values determines the initial amount of Noise Budget, the way it gets consumed by each calculation and so the number of operations executable without issues, the security level of the encrypted evaluation, the computational and memory cost of the overall process, and the accuracy of the decrypted outcomes.

Polynomial modulus

The parameter m, the Polynomial modulus, is directly responsible for the quantity of NB disposable at the beginning of the calculations, for the memory occupation, and for the computational load. More specifically, higher values of m lead to a high initial NB, but also higher consumption for what concerns memory and costs because a higher polynomial modulus also means bigger cipher text sizes making the operations slower, and inefficient.

Furthermore, combined with the Ciphertext coefficient modulus, it affects the security level of the scheme, which increases with higher values of m. Usually, the range of possible values for this parameter, which needs to be a power of 2, includes 1024, 2048, 4096, 8192, 16384, and 32768. Choosing it lower than 1024 makes the scheme insecure while exceeding 32768 is avoided due to the large number of computations in the network. In this project, only the first 2 values of the range are taken into account due to the limitation described.

- m: $1024, 2048$

Plaintext modulus

The plaintext modulus t is probably the most interesting since it determines the size of the plaintexts and the truthfulness of the outcome. In fact, with higher values of t, it can be reached a more precise encrypted computation such that the decrypted results are much similar to the expected one achieved without the use of encryption.

However, increasing this value implies a reduction of the NB. Moreover, the plain modulus also affects the consumption of noise budget in homomorphic operations.

One of the aims of this project is to observe how the accuracy of the encryption changes in the 2 neural network models proposed as this modulus increases, so the following array of t has been set.

- t: $10, 25, 35, 50, 75, 100, 150, 200, 300, 400, 650, 1500, 65537$

Ciphertext coefficient modulus

The parameter q, the Ciphertext coefficient modulus, is fundamental because it is involved in the definition of the initial NB and of the security level of the encryption. In fact, it can be evaluated as any integer, as long as it is not too large to cause vulnerability problems. Overall, a high value of q increases the NB but decreases the security level, once the other 2 parameters are fixed. For this reason, when it is required to raise q it is necessary to increase m too, in order to make the computation secure again but at the cost of the reduction of efficiency.

For what concerns the choice of q, the SEAL library provides the generation of the suitable Ciphertext coefficient modulus when the polynomial modulus and the required AES-equivalent security level are fixed by means of predetermined functions. Hence, the choice of this parameter does not depend on the user but it is automatically determined directly by the polynomial modulus according to the following table [59].

m	Bit-length of default q		
	128-bit security	192-bit security	256-bit security
1024	27	19	14
2048	54	37	29
4096	109	75	58
8192	218	152	118
16384	438	300	237
32768	881	600	476

Table 4.7: SEAL library: q parameter selection

In this case, the security level has been set to 128 bits.

In conclusion, the main focus in the selection of the parameters is the multiplicative depth of the arithmetic circuit the ciphertexts have to go through along the layers since if the NB reaches 0 the noise overruns into the bits containing the message, corrupting the useful information and making it impossible to recover the correct data after decryption. Clearly, picking these 3 parameters is a trade-off between accuracy, performance, and security, making it essential to analyze the complexity of the computations to perform and the disposable resources.

Chapter 5

Results

This chapter reports the results of the simulations performed in the condition described before. Every computation takes into account a dataset, 1 of the network model, and a couple of encryption parameters (m, t). The aim is to compare the outcomes of these 2 architectures and the effects obtained as the parameters change.

5.1 Effects of Polynomial modulus m variations

The consequences of changing the polynomial modulus and its relation with the ciphertext coefficient modulus have been exposed in the previous section. The following tables illustrate the time needed to encrypt an image and the time taken by each model to fulfill the computation in each condition. The first 2 tables,5.1 and 5.2, report the classification of 1 image from the MNIST dataset when performed by the LENET5 and the spiking model. Exactly as observed in the training phase,

LENET5	m	Time required for encryption	HE Classification
MNIST	1024	0,91 seconds	35/37 seconds
	2048	1.82 seconds	77/80 seconds

Table 5.1: Encryption time required - MNIST - CNN

SNN	m	Time required for encryption	HE Classification
MNIST	1024	24/25 seconds	945/1150 seconds
	2048	55 seconds	2400 seconds

Table 5.2: Encryption time required - MNIST - SNN

it is clear how the SNN is slower than the Convolutional model due to the sequence length of its encoding scheme ($30ms$).

For what concerns the FashionMNIST dataset, the time required for each step is similar to the MNIST case since the model structure and the input tensors are the same.

LENET5	m	Time required for encryption	HE Classification
FashionMNIST	1024	0,91 seconds	35/37 seconds
	2048	1.82 seconds	77/80 seconds

Table 5.3: Encryption time required - FashionMNIST - CNN

SNN	m	Time required for encryption	HE Classification
FashionMNIST	1024	24/25 seconds	945/1150 seconds
	2048	55 seconds	2400 seconds

Table 5.4: Encryption time required - FashionMNIST - SNN

The CIFAR10 dataset is more complicated and requires a slightly different structure to be computed and for this reason, the relative tables show the worst outcomes.

LENET5	m	Time required for encryption	HE Classification
CIFAR10	1024	0,91 seconds	35/37 seconds
	2048	1.82 seconds	77/80 seconds

Table 5.5: Encryption time required - CIFAR10 - CNN

SNN	m	Time required for encryption	HE Classification
CIFAR10	1024	103/110 seconds	2380 seconds
	2048	210 seconds	4900 seconds

Table 5.6: Encryption time required - CIFAR10 - SNN

It is important to notice that the execution time for 1 image without any encryption measure, in any case, and for every dataset, never exceeds 0.09 seconds. Thus, it has been illustrated how computationally heavy can the encryption be and the amount of time that requires with respect to the classical and insecure classification. As expected, the time required by the network increases as the polynomial modulus gets higher.

Furthermore, it is interesting to verify how the Polynomial modulus does not interfere with the calculation results. In fact, repeating the classification for 1 image from the MNIST dataset which has a target label of 7, with a fixed value of $t = 150$, and changing m does not give different results.

Dataset	m	Resulting tensor
MNIST target=7	m=1024	-1.5756e+1, -9.5055, -9.2050, -8.2560, -1.0068e+1, -1.4172e+1, -1.9151e+1, -8.5591e 4, -1.2715e+1, -7.8896
	m=2048	-1.5756e+1, -9.5055, -9.2050, -8.2560, -1.0068e+1, -1.4172e+1, -1.9151e+1, -8.5591e 4, -1.2715e+1, -7.8896

Table 5.7: CNN - MNIST - Resulting tensors

The same behavior is obtained for the other 2 datasets:

Dataset	m	Resulting tensor
FashionMNIST target=4	m=1024	-5.8703, -6.4715, -2.9943, -4.5461, -0.5762, -9.2192, -0.9901, -9.8227, -6.8643, -8.5474
	m=2048	-5.8703, -6.4715, -2.9943, -4.5461, -0.5762, -9.2192, -0.9901, -9.8227, -6.8643, -8.5474

Table 5.8: CNN - FashionMNIST - Resulting tensors

Dataset	m	Resulting tensor
CIFAR10 target=5	m=1024	-5.0038, -4.4079, -2.9772, -1.6931, -1.2493, -1.8964, -2.5618, -1.8934, -4.6872, -2.6248
	m=2048	-5.0038, -4.4079, -2.9772, -1.6931, -1.2493, -1.8964, -2.5618, -1.8934, -4.6872, -2.6248

Table 5.9: CNN - CIFAR10 - Resulting tensors

The same outcome is achieved in the case of the spiking network.

Paying attention to the CIFAR10 case, it can be noticed that the resulting tensor gives as a classification result the label 4 which is wrong. In fact, the error rate for this dataset is not low due to the not-so-high accuracy reached during the training step. However, the final tensor gained is equal and the same error is verified with an increasing m proving that this parameter is not involved in the accuracy of the encryption computation.

5.2 Effects of Plaintext modulus t variations

In this paragraph instead, it is reported the evolution of the simulation when changing the t parameter. Again, the aim is to compare the 2 network model behavior using 650 images from each of the 3 datasets. The following plots describe essentially how the accuracy of the encryption computation increases as the value of t gets higher.

5.2.1 MNIST

For what concerns the MNIST dataset, the first figure shows the match rate of the 2 architectures. For match rate, it is intended how many times the normal execution and the encrypted computation give the same final classification label during the simulation of 650 images.

Figure 5.1: MNIST MATCH LeNet5 vs SNN

As explained in the previous chapter it has been set the array of possible values for the parameter t: $10, 25, 35, 50, 75, 100, 150, 200, 300, 400, 650, 1500, 65537$. From figure 5.1, it is evident that the SNN gets better results in terms of truthfulness for lower values of t with respect to the classic LENET5.

Figure 5.2 illustrates the average error of the 2 networks obtained by comparing the final tensors of the normal, insecure execution with the one at the end of the encrypted process.

Figure 5.2: MNIST Average errors LeNet5 vs SNN

According to the 5.1, 5.2 shows that SNN gets better outcomes than LENET5 for what concers the errors made.

The following 3 figures 5.3, 5.4, and 5.5, instead, have as focus the mean mismatch observed after each layer for 3 different values of t: 50, 150, 650. Essentially the internal tensor computed after every single layer in the encrypted computation has been compared with its respective one when the encryption is not applied.

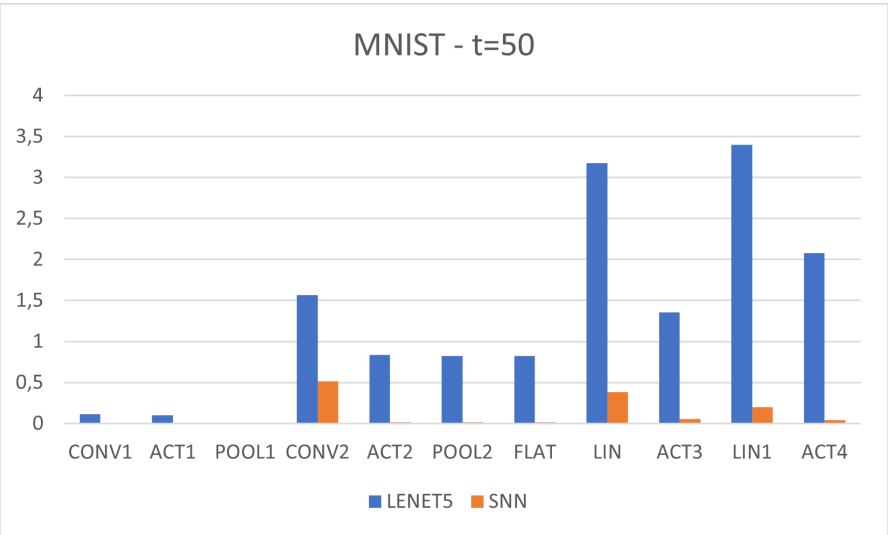


Figure 5.3: MNIST (Mean error/layer) t = 50 LeNet5 vs SNN

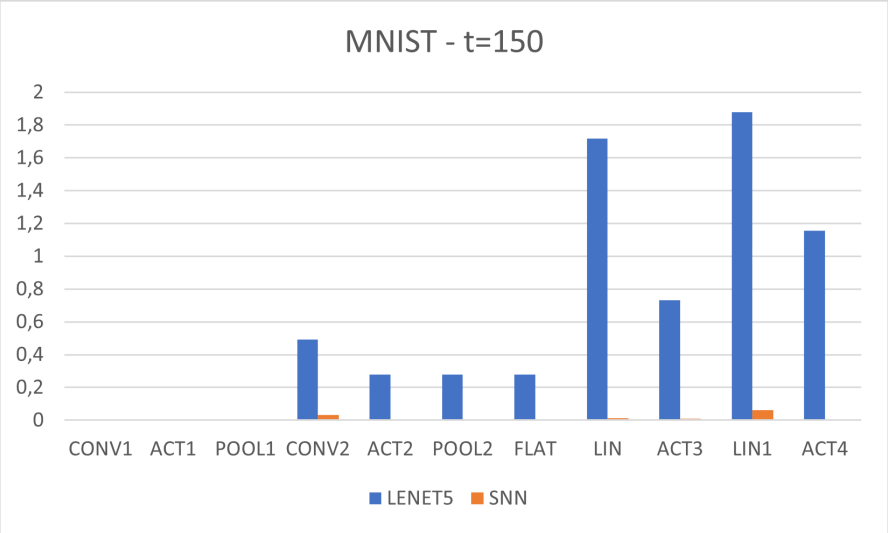


Figure 5.4: MNIST (Mean error/layer) t = 150 LeNet5 vs SNN

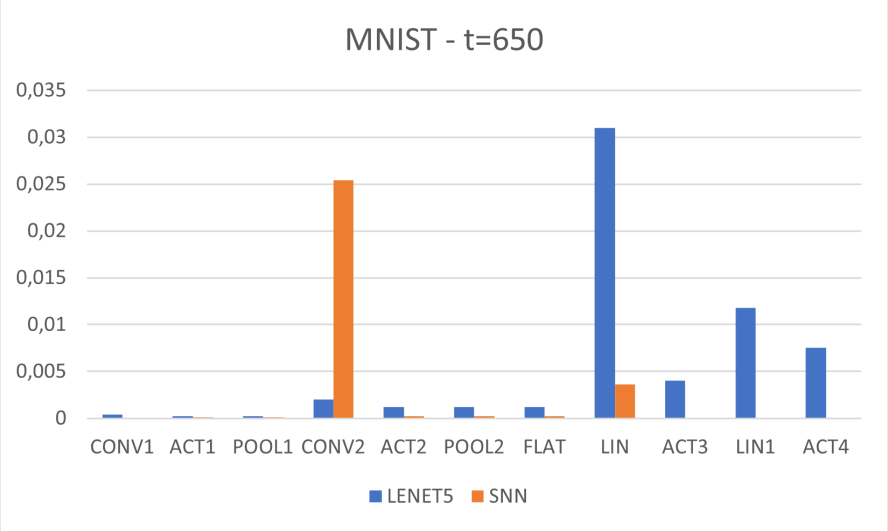


Figure 5.5: MNIST (Mean error/layer) t = 650 LeNet5 vs SNN

Clearly, as t grows the mean errors always decrease as expected, and again the spiking version shows better results with respect to LENET5. What is interesting to notice is that the most troublesome layers are the linear and the convolutional ones since they are the most complicated ones.

5.2.2 FashionMNIST

The figures below regard the simulations computed for the FashionMNIST dataset computed by the 2 models. The exact same kind of information collected for MNIST is reported.

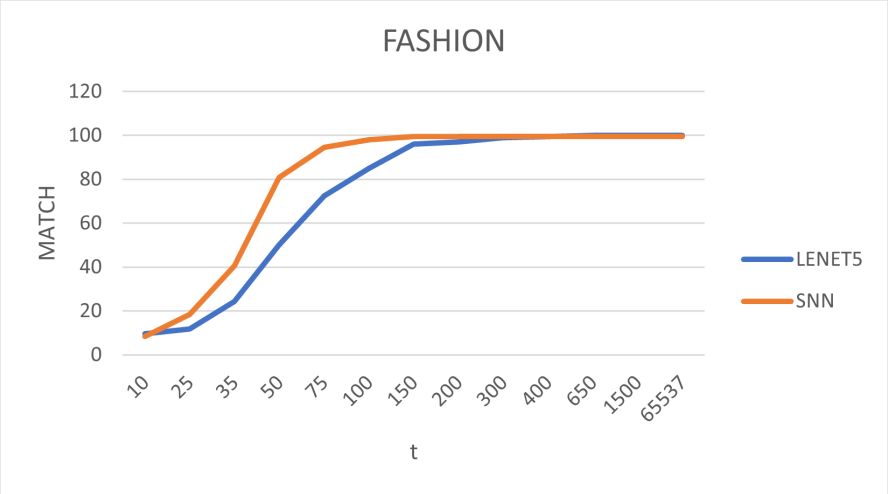


Figure 5.6: FashionMNIST MATCH LeNet5 vs SNN

Again the spiking version shows a better match with the normal execution of the classification along the 650 images processed.

Though it has to be noticed that figure 5.6 illustrates how this dataset is slightly more complicated than MNIST and the difference between SNN and LENET5 is less evident. More precisely, at the initial point of the graph, for t = 10, it seems that CNN acts better than SNN but it is due to the extremely low value of the t parameter which leads to huge errors and almost random outcomes. Probably, by analyzing a lot more images, SNN would have reached a better result with respect to CNN even for that low value of t.

Figure 5.7 is pretty similar to 5.2 in fact, the average error has the same behavior when using these 2 datasets.

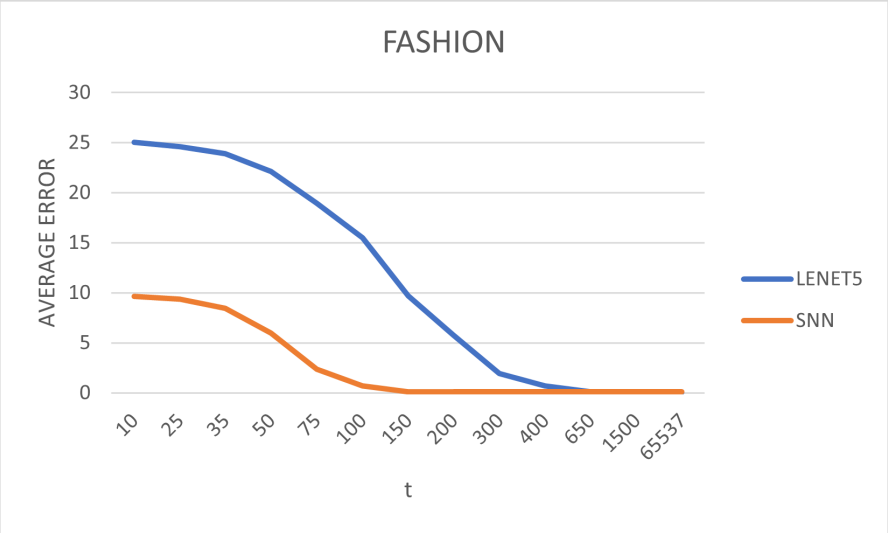


Figure 5.7: FashionMNIST Average errors LeNet5 vs SNN

The last 3 figures 5.8, 5.9, and 5.10 report the mean error made after each layer when t is fixed respectively to 50, 150, 650.

According to what has been observed before, the layers which cause the error to rise the most are the linear and the convolutional ones and as the value of t grows the benefit of using the spiking functions becomes less conspicuous.

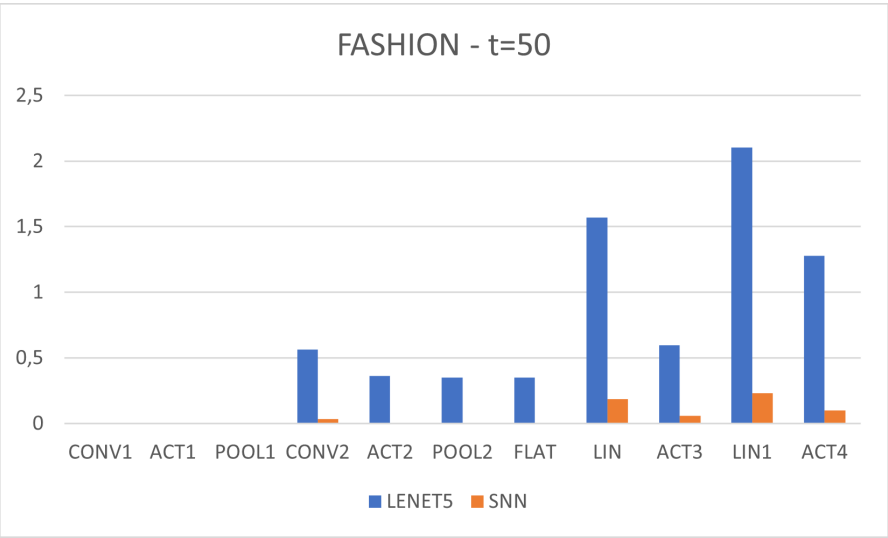


Figure 5.8: FashionMNIST (Mean error/layer) t = 50 LeNet5 vs SNN

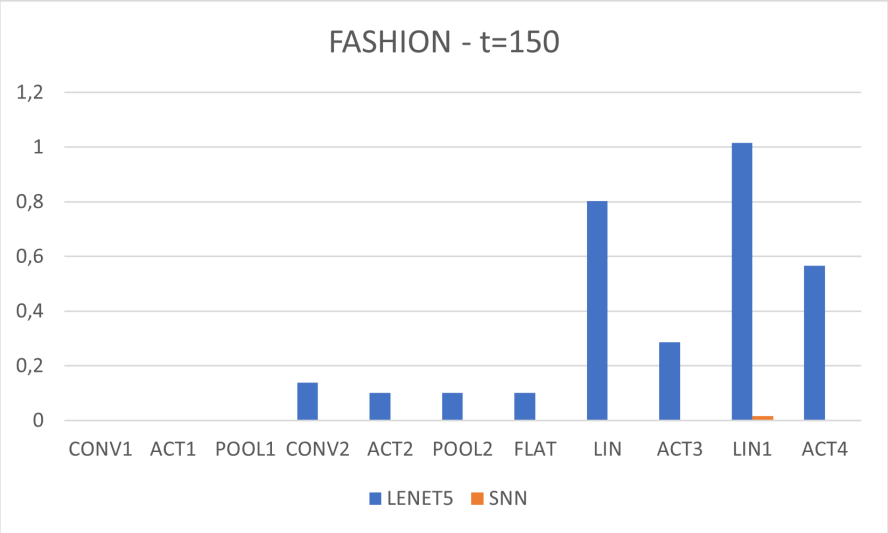


Figure 5.9: FashionMNIST (Mean error/layer) t = 150 LeNet5 vs SNN

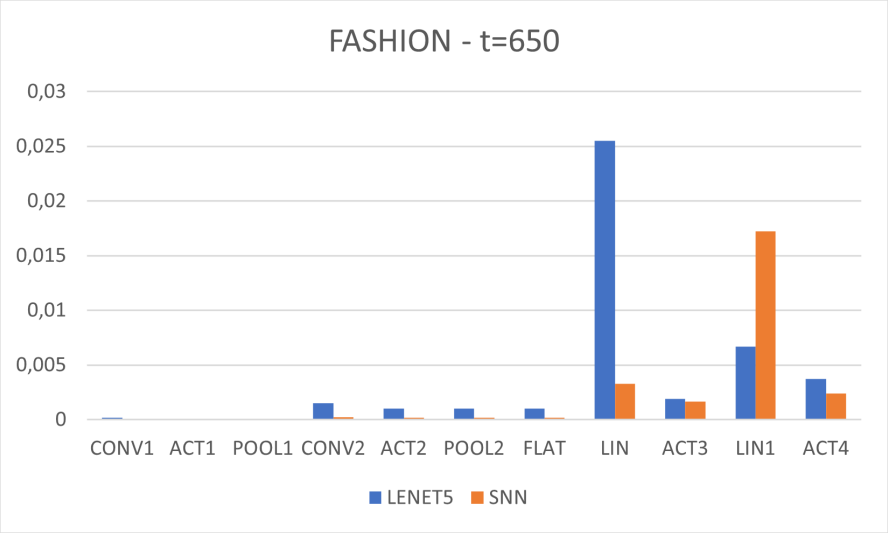


Figure 5.10: FashionMNIST (Mean error/layer) t = 650 LeNet5 vs SNN

5.2.3 CIFAR10

The following figures describe what happens when the images processed are from CIFAR10.

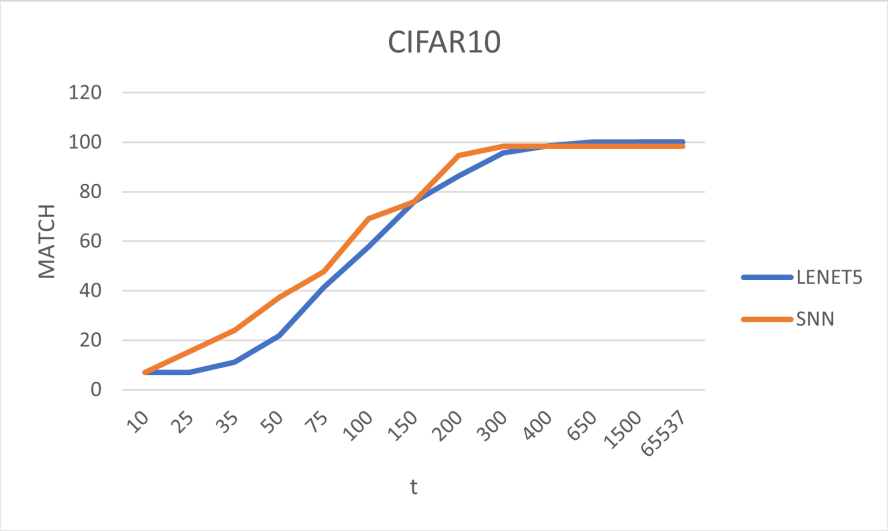


Figure 5.11: CIFAR10 MATCH LeNet5 vs SNN

Figure 5.11 illustrates a smoother trend with respect to previous cases which means that the accuracy of the encryption computation is lower than before for low values of t and acceptable accuracies are reached with more strict requirements with respect to previous datasets evaluations. In this case, the behavior of the 2 structures is almost the same.

Figure 5.12 illustrates the average error.

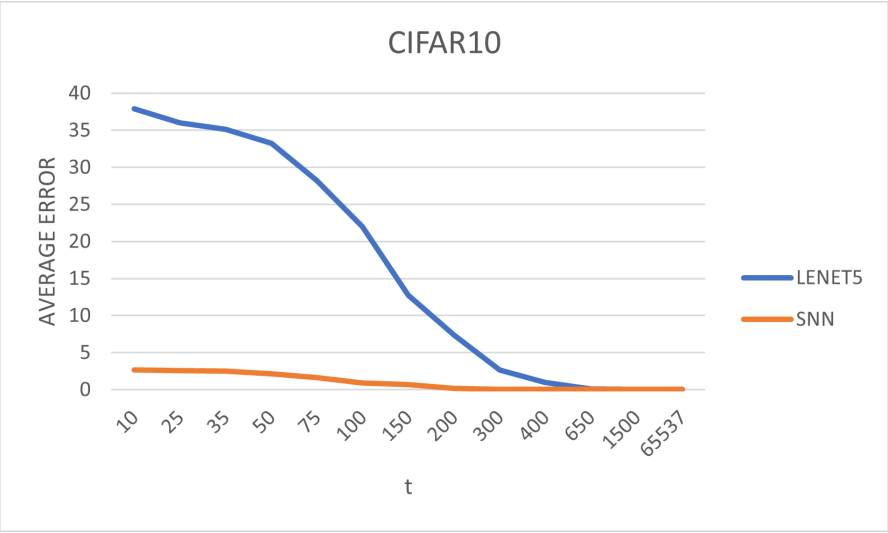


Figure 5.12: CIFAR10 Average errors LeNet5 vs SNN

It has the same course as the other 2 datasets but with higher values for the

CNN execution, as expected due to the higher complexity of the input data. For the SNN instead, lower errors have been found.

Using CIFAR10, it is necessary to use a higher t to get the maximum match between the normal and the encrypted execution. Hence, it is clear that in this case more mistakes than before are made, as a consequence of reduced accuracy achieved during the training phase.

The last 3 figures 5.13, 5.14, 5.15 regard, again, the mean error at each layer.

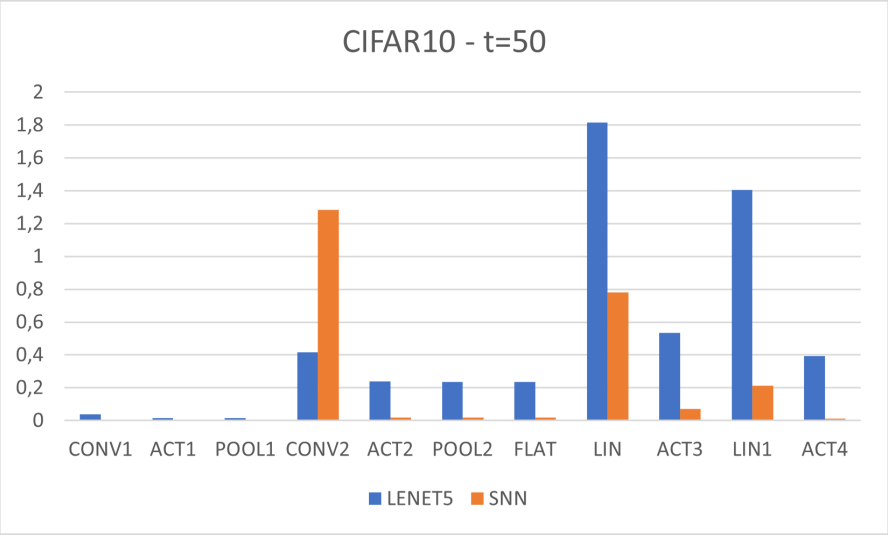


Figure 5.13: CIFAR10 (Mean error/layer) t = 50 LeNet5 vs SNN

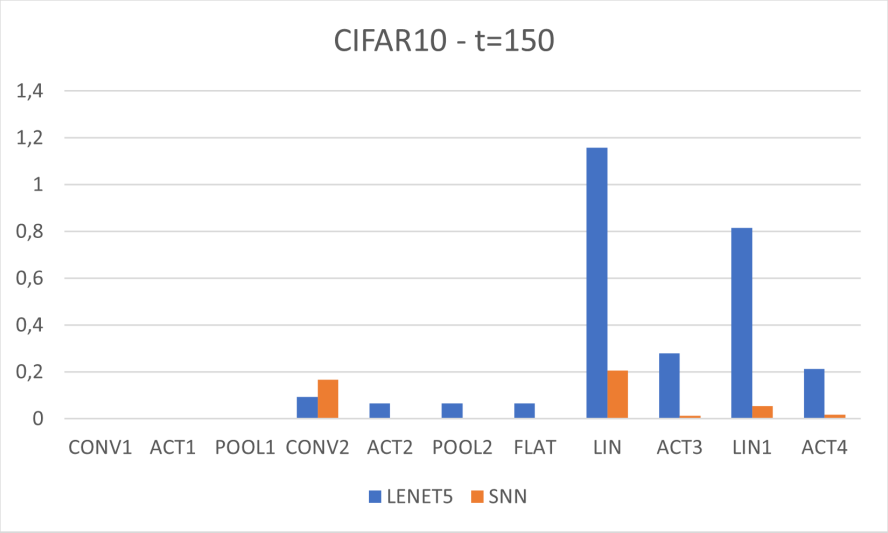


Figure 5.14: CIFAR10 (Mean error/layer) t = 150 LeNet5 vs SNN

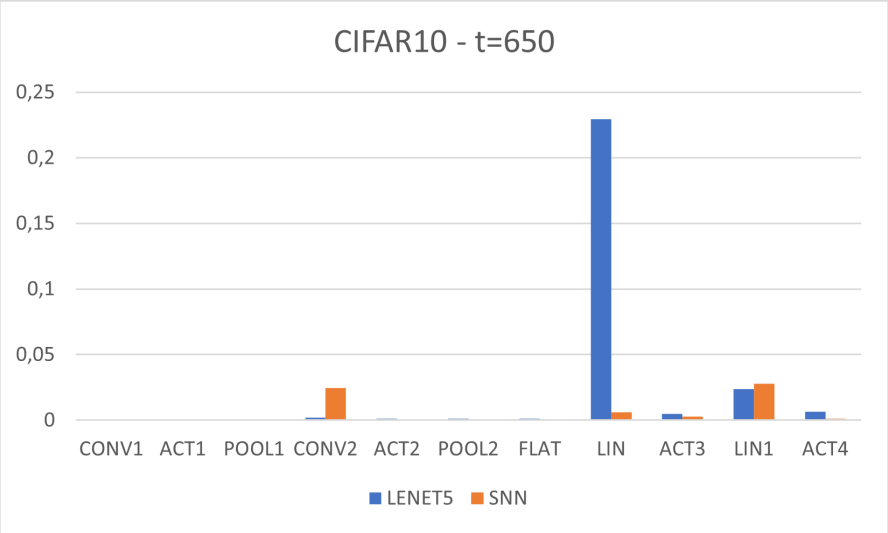


Figure 5.15: CIFAR10 (Mean error/layer) t = 650 LeNet5 vs SNN

Even in this case, SNN performs usually better than LENET5 though the mean of the errors extracted after the second convolutional layer is higher for the first architecture.

To sum up, observing all the figures regarding the mean error after each layer, it appears clear how the flatten layer is not involved in the error since it is only a geometrical manipulation of the tensor. Moreover, the error is slightly reduced after the pooling layer because it performs an average making the error more smooth.

The reason why the SNN is better is that it is based on the encoding scheme and LIF activation function and both turn their input tensors into a set of 0 and 1 while in the LENET5 computations the numbers to deal with are usually more difficult to process and easily create mistakes. For higher values of t, the averages of errors committed are so low that the 2 models become comparable.

5.3 Noise Budget

Noise budget is a fundamental quantity when it is necessary to use an encryption scheme because as long as operations are evaluated on the ciphertexts, the noise budget gets continuously consumed. Its importance and the way it depends on the 3 parameters have been already discussed in the previous chapters. Considering the analyzed values of the Polynomial modulus m and the Plaintext modulus t, the initial noise budgets for the simulations performed are reported in the following table.

m	t	Noise Budget
1024	10	14
	25	12
	35	12
	50	12
	75	11
	100	10
	150	10
	200	9
	300	9
	400	8
	650	8
2048	1500	33
	65537	28

Table 5.10: Initial Noise Budget

Since m is not involved in the effectiveness of the encrypted computations but it influences the RAM consumed and the time required for the execution it has been used the lowest value, 1024, as much as possible. Fortunately, using the multi-party and dynamic strategy proposed in the third chapter, the simulations do not require a relevant amount of initial budget because the client decrypts the internal outputs every time there is an activation function to perform, restoring the original quantity of Noise budget. However, when t is too high, the initial Noise Budget becomes too low to fulfill the calculation without mistakes. For this reason, when t is set to 1500 or 65537, it has been necessary to increase m to 2048 to obtain a higher noise budget slowing down a lot the computations and increasing the required resources.

Thus, Noise Budget shows once again how SNNs can be useful since they achieve better results for lower t which means that they can perform encrypted calculations for lower values of m.

5.4 Conclusions

This project has as its focus a comparison between two different generations of neural networks, the convolutional and the spiking ones, while dealing with a classification problem. Convolutional models already provide optimum results in solving this type of issue though the spiking architectures aim mainly to reduce the energy consumption of the computation without making the performance worse. The main feature of the SNNs is the realistic behavior of their nodes which are pretty similar to the biological neuron. Indeed, the human brain relies on spikes

to fulfill the computation and the transmission of information. Reproducing this conduct makes the SNNs more hardware friendly and more efficient than ANNs in general.

More precisely, the 2 structures have been compared while operating with encrypted data. In fact, it is possible, lately always more often, that predictions and other machine learning services need to be executed on sensitive data. Thus, it has become fundamental to achieve systems able to provide those services while keeping intact the privacy of the client information. In the last few decades, many research works have been conducted to build such kinds of systems based on privacy-preserving algorithms and architectures. For what concerns neural networks, the most relevant steps are the training and the inference phases which need to be evaluated in different ways when dealing with security. In particular, training is mostly related to differential privacy. Essentially, it is possible to train the model on encrypted data by performing the procedure using only additions and multiplications, though this kind of strategy still needs to be implemented and leads to an efficiency drop in the computations.

The aim of this thesis has been the comparison of the 2 models, and their requirements, in the usual training phase and in the encrypted execution of the inference phase using as security measures a homomorphic encryption scheme and a multi-party computation strategy. The first has been employed to encrypt the images of 3 datasets and to fulfill the classification task by encoding all the layers of the 2 architectures that are characterized by linear operations, making the networks able to manage encrypted inputs. The Homomorphic Encryption scheme used is the Brakerski/Fan-Vercauteren one and like all these types of schemes, it is computationally heavy and requires a lot of time to complete the evaluations. For these reasons, LeNet5 has been chosen as the neural architecture since it is one of the simplest and shortest, in order to keep control of the execution cost. In addition, a simple multi-party computation pattern has been used in order to execute the non-linear operations, needed by the models to learn, without approximating the activation functions which characterize the 2 networks.

However, even though this strategy leads to a more accurate execution of the calculations, it makes the framework less effective from the server privacy-preserving goal point of view. In fact, the client does need to dynamically interact with the server as the classification is performed. As a consequence of this intermediate exchange of outcomes, the server is more vulnerable to leakage of its model information and the client has to carry out, in addition to the encryption and decryption operations, the activation function and, in the case of the spiking neural network, the initial time encoding of the input. The security of the client data, instead, is still guaranteed.

Overall, the experimental results obtained by means of simulations in Python on the Google Colab platform portrayed a better performance, in terms of accuracy

and of the errors made, of the SNN when the homomorphic encryption scheme is used.

On the other hand, SNN requires a huge amount of time with respect to CNN since the necessary initial time encoding makes the execution as slower as the observation sequence grows. The higher precision of the SNNs is an interesting result since it allows to perform the encrypted calculations with lower values of the Plaintext modulus t which essentially leads to lower values for Polynomial modulus m, reducing the computation time and the resources consumption, though in the cases examined, the advantages are lost due to the necessary time encoding.

To sum up there are a lot of limitations while using HE and a lot of improvements are required for simpler management of the SNNs since their implementation is still troublesome, especially for what concerns the training phase, due to the lack of frameworks, and studies. Furthermore, SNNs still represents an important innovation in the machine learning field, thus it has not been studied yet a solution that merges a HE scheme with them without drastically changing the behavior of the spiking activation functions.

In conclusion, future works can lead to better encoding schemes that may use better polynomials or noise budgets, reducing their computational load, or to a better configuration and selection of the encryption parameters. In particular, this project has underlined the necessity of better strategies to merge the non-linearity of the neural networks with the constraints of Homomorphic Encryption. Usually, this issue is avoided by means of the approximation of the activation functions, but, for what concerns the SNNs, these methods have not been examined yet.

Bibliography

[1] Alex M. Andrew. «REINFORCEMENT LEARNING: AN INTRODUCTION by Richard S. Sutton and Andrew G. Barto, Adaptive Computation and Machine Learning series, MIT Press (Bradford Book), Cambridge, Mass., 1998, xviii 322 pp, ISBN 0-262-19398-1, (hardback, £31.95).» In: *Robotica* 17.2 (1999), pp. 229–235. DOI: 10.1017/S0263574799211174 (cit. on pp. 3, 7).

[2] *Machine learning*. URL: https://en.wikipedia.org/wiki/Machine_learning (cit. on p. 3).

[3] *Neuron*. URL: https://en.wikipedia.org/wiki/Neuron (cit. on p. 4).

[4] Wikibooks. *Sensory Systems/old/Biological Machines/Print version — Wikibooks, The Free Textbook Project*. 2016. URL: https://en.wikibooks.org/w/index.php?title=Sensory_Systems/old/Biological_Machines/Print_version&oldid=3160119 (cit. on pp. 4–6).

[5] A. Diamond, T. Nowotny, and M. Schmuker. *[PDF] comparing neuromorphic solutions in action: Implementing a bio-inspired solution to a benchmark classification task on three parallel-computing platforms: Semantic scholar*. 2016 (cit. on p. 4).

[6] Julius von Kügelgen. *On Artificial Spiking Neural Networks: Principles, Limitations and Potential*. June 2017 (cit. on pp. 5, 6).

[7] Boudjelal Meftah, Olivier Lézoray, Soni Chaturvedi, Aleefia A. Khurshid, and Abdelkader Benyettou. «Image Processing with Spiking Neuron Networks». In: *Artificial Intelligence, Evolutionary Computing and Metaheuristics*. 2013 (cit. on p. 6).

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. «Unsupervised Learning». In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer New York, 2009, pp. 485–585. ISBN: 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7_14. URL: https://doi.org/10.1007/978-0-387-84858-7_14 (cit. on p. 7).

[9] Jürgen Schmidhuber. «Deep learning in neural networks: An overview». In: *Neural Networks* 61 (2015), pp. 85–117. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2014.09.003. URL: https://www.sciencedirect.com/science/article/pii/S0893608014002135 (cit. on pp. 7, 9).

[10] Wikipedia contributors. *Convolutional neural network — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=1123767034 (cit. on p. 7).

[11] Wikimedia Commons. *File:Convolutional Neural Network NeuralNetworkFeatureLayers.gif — Wikimedia Commons, the free media repository*. [Online; accessed 25-November-2022]. 2022. URL: https://commons.wikimedia.org/w/index.php?title=File:Convolutional_Neural_Network_NeuralNetworkFeatureLayers.gif&oldid=644687307 (cit. on p. 8).

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. «ImageNet classification with deep convolutional neural networks». In: *Communications of the ACM* 60 (2012), pp. 84–90 (cit. on p. 10).

[13] Katarzyna Janocha and Wojciech Marian Czarnecki. «On loss functions for deep neural networks in classification». In: *arXiv preprint arXiv:1702.05659* (2017) (cit. on p. 10).

[14] Laurene V. Fausett. «Fundamentals of neural networks: architectures, algorithms, and applications». In: 1994 (cit. on p. 11).

[15] Diederik P. Kingma and Jimmy Ba. «Adam: A Method for Stochastic Optimization». In: *CoRR* abs/1412.6980 (2015) (cit. on p. 11).

[16] Osvaldo Simeone, Bipin Rajendran, Andre Gruning, Evangelos S. Eleftheriou, Mike Davies, Sophie Deneve, and Guang-Bin Huang. «Learning Algorithms and Signal Processing for Brain-Inspired Computing [From the Guest Editors]». In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 12–15. DOI: 10.1109/MSP.2019.2935557 (cit. on p. 12).

[17] Giacomo Indiveri and Yulia Sandamirskaya. «The Importance of Space and Time for Signal Processing in Neuromorphic Agents: The Challenge of Developing Low-Power, Autonomous Agents That Interact With the Environment». In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 16–28. DOI: 10.1109/MSP.2019.2928376 (cit. on pp. 12, 16).

[18] Filip Ponulak and Andrzej J. Kasinski. «Introduction to spiking neural networks: Information processing, learning and applications.» In: *Acta neurobiologiae experimentalis* 71 4 (2011), pp. 409–33 (cit. on p. 12).

[19] Wikipedia contributors. *Action potential — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-November-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Action_potential&oldid=1123326124 (cit. on p. 13).

[20] Filip Ponulak. «Supervised learning in spiking neural networks with ReSuMe method». In: *Phd, Poznan University of Technology* 46 (2006), p. 47 (cit. on p. 14).

[21] E.M. Izhikevich. «Simple model of spiking neurons». In: *IEEE Transactions on Neural Networks* 14.6 (2003), pp. 1569–1572. DOI: 10.1109/TNN.2003.820440 (cit. on p. 14).

[22] Hélène Paugam-Moisy and Sander M. Bohté. «Computing with Spiking Neuron Networks». In: *Handbook of Natural Computing*. 2012 (cit. on pp. 14, 15).

[23] Craig Gentry. «Fully homomorphic encryption using ideal lattices». In: *Symposium on the Theory of Computing*. 2009 (cit. on pp. 16, 22).

[24] Ronald L. Rivest and Michael L. Dertouzos. «ON DATA BANKS AND PRIVACY HOMOMORPHISMS». In: 1978 (cit. on pp. 16, 22).

[25] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. «Evaluating 2-DNF Formulas on Ciphertexts». In: *Theory of Cryptography Conference*. 2005 (cit. on p. 17).

[26] Nicolas Papernot, Patrick Mcdaniel, Arunesh Sinha, and Michael P. Wellman. «Towards the Science of Security and Privacy in Machine Learning». In: *ArXiv* abs/1611.03814 (2016) (cit. on p. 17).

[27] Zvika Brakerski and Vinod Vaikuntanathan. «Efficient Fully Homomorphic Encryption from (Standard)». In: *SIAM Journal on Computing* 43.2 (2014), pp. 831–871. DOI: 10.1137/120868669 (cit. on p. 18).

[28] Craig Gentry. *A fully homomorphic encryption scheme* (cit. on p. 19).

[29] Junfeng Fan and Frederik Vercauteren. «Somewhat practical fully homomorphic encryption». In: 1970. URL: https://eprint.iacr.org/2012/144 (cit. on pp. 19, 20, 22).

[30] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. «Efficient public key encryption based on ideal lattices». English. In: *Advances in Cryptology - ASIACRYPT 2009 - 15th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*. Ed. by Mitsuru Matsui. Vol. 5912 LNCS. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 15th International Conference on the Theory and Application of Cryptology and Information Security, ASIACRYPT 2009 ; Conference

date: 06-12-2009 Through 10-12-2009. 2009, pp. 617–635. ISBN: 3642103650. DOI: 10.1007/978-3-642-10366-7_36 (cit. on p. 20).

[31] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. *Fully Homomorphic Encryption without Bootstrapping*. Cryptology ePrint Archive, Paper 2011/277. https://eprint.iacr.org/2011/277. 2011. URL: https://eprint.iacr.org/2011/277 (cit. on pp. 20, 22).

[32] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2012/144. https://eprint.iacr.org/2012/144. 2012. URL: https://eprint.iacr.org/2012/144 (cit. on p. 20).

[33] Mauro Barni, Claudio Orlandi, and Alessandro Piva. «A privacy-preserving protocol for neural-network-based computation». In: *Workshop on Multimedia & Security*. 2006 (cit. on p. 21).

[34] Pascal Paillier. «Public-Key Cryptosystems Based on Composite Degree Residuosity Classes». In: *Advances in Cryptology — EUROCRYPT '99*. Ed. by Jacques Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238 (cit. on p. 21).

[35] Andrew Chi-Chih Yao. «Protocols for secure computations». In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)* (1982), pp. 160–164 (cit. on p. 21).

[36] Claudio Orlandi, Alessandro Piva, and Mauro Barni. «Oblivious Neural Network Computing via Homomorphic Encryption». In: *EURASIP Journal on Information Security* 2007 (2007), pp. 1–11 (cit. on p. 21).

[37] Ivan Damgård and Mads Jurik. «A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System». In: *International Conference on Theory and Practice of Public Key Cryptography*. 2001 (cit. on p. 21).

[38] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. *Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme*. Cryptology ePrint Archive, Paper 2013/075. 2013 (cit. on p. 22).

[39] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. «Privacy-Preserving Classification on Deep Neural Network». In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 35 (cit. on p. 22).

[40] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Robert Wernsing. «CryptoNets: applying neural networks to encrypted data with high throughput and accuracy». In: *International Conference on Machine Learning*. 2016 (cit. on p. 22).

[41] Simone Disabato, Alessandro Falcetta, Alessio Mongelluzzo, and Manuel Roveri. «A Privacy-Preserving Distributed Architecture for Deep-Learning-as-a-Service». In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8 (cit. on pp. 22, 25).

[42] Shreya Garge. *Neural Networks for Encrypted Data using Homomorphic Encryption*. 2018 (cit. on p. 22).

[43] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. «Towards spike-based machine intelligence with neuromorphic computing». In: *Nature* 575 (2019), pp. 607–617 (cit. on p. 23).

[44] Bing Han and Kaushik Roy. «Deep Spiking Neural Network: Energy Efficiency Through Time Based Coding». In: *European Conference on Computer Vision*. 2020 (cit. on p. 23).

[45] Peter Udo Diehl and Matthew Cook. «Unsupervised learning of digit recognition using spike-timing-dependent plasticity». In: *Frontiers in Computational Neuroscience* 9 (2015) (cit. on p. 23).

[46] Junhaeng Lee, Tobi Delbrück, and Michael Pfeiffer. «Training Deep Spiking Neural Networks Using Backpropagation». In: *Frontiers in Neuroscience* 10 (2016) (cit. on p. 23).

[47] Chankyu Lee, Syed Shakib Sarwar, and Kaushik Roy. «Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures». In: *Frontiers in Neuroscience* 14 (2020) (cit. on p. 23).

[48] Abhronil Sengupta, Yuting Ye, Robert Y. Wang, Chiao Liu, and Kaushik Roy. «Going Deeper in Spiking Neural Networks: VGG and Residual Architectures». In: *Frontiers in Neuroscience* 13 (2019) (cit. on p. 23).

[49] Youngeun Kim, Yeshwanth Venkatesha, and Priyadarshini Panda. «PrivateSNN: Privacy-Preserving Spiking Neural Networks». In: *AAAI*. 2022 (cit. on pp. 23, 24).

[50] Li Yuan, Francis E. H. Tay, Guilin Li, Tao Wang, and Jiashi Feng. «Revisit Knowledge Distillation: a Teacher-free Framework». In: *ArXiv* abs/1909.11723 (2019) (cit. on p. 24).

[51] Friedemann Zenke and Surya Ganguli. «SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks». In: *Neural Computation* 30 (2018), pp. 1514–1541 (cit. on p. 24).

[52] Adam Paszke et al. «PyTorch: An Imperative Style, High-Performance Deep Learning Library». In: *ArXiv* abs/1912.01703 (2019) (cit. on p. 25).

[53] Christian Pehle and Jens Egholm Pedersen. *Norse - A deep learning library for spiking neural networks*. Version 0.0.6. Documentation: https://norse.ai/docs/. Jan. 2021. DOI: 10.5281/zenodo.4422025. URL: https://doi.org/10.5281/zenodo.4422025 (cit. on p. 25).

[54] Alberto Ibarrondo and Alexander Viand. «Pyfhel: PYthon For Homomorphic Encryption Libraries». In: *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography* (2021) (cit. on p. 25).

[55] *Microsoft SEAL (release 4.0)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Mar. 2022 (cit. on p. 25).

[56] corinna cortes yann lecun corinna cortes and chris burges. *The mnist database*. URL: http://yann.lecun.com/exdb/mnist/ (cit. on p. 26).

[57] Zalando Research. *Fashion mnist*. Dec. 2017. URL: https://www.kaggle.com/datasets/zalando-research/fashionmnist (cit. on p. 26).

[58] Alex Krizhevsky. «Learning Multiple Layers of Features from Tiny Images». In: 2009 (cit. on p. 26).

[59] Kim Laine. «Simple Encrypted Arithmetic Library v2.3.1». In: (cit. on p. 45).