UNIVERSITÀ DI PISA

DIPARTIMENTO DI FISICA

Corso di Laurea Magistrale in Fisica

Hybrid quantum-classical approach to quantum machine learning for computer vision

Relatori:

Prof. Oliver Morsch

Dr. Roberto Cappuccio

Candidato:

Niccolò Francesco Tiezzi

ANNO ACCADEMICO 2022/2023

Contents

Chapter 1

Introduction

As medium scale quantum computers are becoming a reality, more and more applications of quantum algorithms are taking places in a wide range of fields such as simulation of physical systems, chemistry, numerical optimization or cryptography. Quantum algorithms have the aim to exploit quantum features such as entanglement and superposition to overcome the limits of classical computation and reach a quantum advantage. Due to the fact that current quantum computers are small to medium scale, and still affected by noise (called NISQ, noisy intermediate scale quantum devices), it is of interest to find applications that don't necessarily need error correction routines (that would increase the number of qubits needed) but still present quantum advantage.

In parallel to this, another field that has seen a rapid growth in recent years is machine learning: an approach to problem solving that consists in supplying the machine with enough data or general information related to the problem in order for it to figure out 'it's own algorithm' without being explicitly told what to do. Examples of this are data classification and clustering, data generation or autonomous driving. A drawback to this is the need of an enormous amount of data and computational resources to train the machine and whats more the closed nature of this kind of algorithms brings difficulty in assessing if these resources are optimally used.

Having underlined this two points, it becomes natural to ask if quantum computers can aid machine learning algorithms, in particular if quantum features can help to optimize the performance of automatic learning algorithms without having to rely on faster and faster hardware which represents a serious bottleneck in machine learning research.

In recent years a lot of studies tried to tackle this questions, pointing to a possible quantum advantage of quantum machine learning consisting of hybridizing the best of both world: the expressibility of quantum variational algorithms and the large scale optimization routines used in classical computation frameworks.

In this thesis it has been chosen to test the performance of hybrid quantum classical algorithms in one of the most important ML sub-field: computer vision, with particular focus given to image classification and generation. Fol-

lowing the most promising results in recent literature two algorithms have been implemented and tested: the data re-uploading scheme, that has been adapted to image classification, and the patch GAN model, ideal for small scale quantum devices. The two algorithms reported performances comparable or better than classical algorithms with the same complexity, proving the importance of small scale quantum algorithms and hinting for bigger quantum advantages in future years.

The thesis starts with an introduction to the basic concepts of quantum computation: the idea of quantum circuit and quantum gates and how these are subject to errors. Quantum circuits and gates are the building blocks of every quantum algorithms, in particular for quantum neural networks. Studying the effects of noise is also crucial to evaluate if an algorithm can be actually implemented.

An introduction to machine learning will follow, in particular the sub-field called deep learning which consists in using deep neural networks as the core computational unit of the learning algorithm. The focus will be both on convolutional neural networks, the most used architecture for image processing, and generative adversarial network, models used to reproduce data (in this case images) starting from random noise exploiting a minimax type of process in which one of the networks produce the data and the other asses if the data are real or fake.

The following chapter will merge these two concepts introducing quantum machine learning: variational quantum circuits (quantum circuits with trainable parameters) will be defined and how to use them as quantum neural network will be explained. The aforementioned quantum machine learning algorithms (namely data re-uploading and patch GAN) will be explained and their usage justified.

In the last chapter these algorithms will be implemented and tested on simulators (perfect and noisy) and on a real quantum device for the data re-uploading algorithm. In the end, conclusions will be drawn and possible future paths outlined.

Chapter 2

Quantum Computation

In this chapter a brief introduction of quantum computation will be given, pointing out the key concepts that will be explored in the thesis such as the idea of quantum circuit and how to model errors in quantum algorithms.

Quantum computations is the representation and manipulation of data relying on the principles of quantum mechanics such as superposition and entanglement. Unlike in classical computation, which represents information as binary scalar bits, in quantum computation the information is encoded in quantum bits (qubits) which are represented by states in a Hilbert space.

A very popular way of representing qubits is with discrete-variables states such as

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.1}$$

$|0\rangle, |1\rangle \in \mathbb{C}^2$. Unlike classical bits qubits can be represented in a superposition by a change of basis

$$|+\rangle = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad |-\rangle = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{2.2}$$

where $|+\rangle = \frac{1}{\sqrt{2}}\big(\,|0\rangle + |1\rangle\,\big)$ and $|-\rangle = \frac{1}{\sqrt{2}}\big(\,|0\rangle - |1\rangle\,\big)$. In order to preserve unitarity the qubits are manipulated by unitary operators called 'quantum gates', in a fashion that resembles the logic gates used in classical computation

$$\begin{aligned} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \equiv \hat{H}\,|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = |+\rangle \\ \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \equiv \hat{H}\,|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = |-\rangle \end{aligned} \tag{2.3}$$

where $\hat{H}$ is called *Hadamard gate*.

With multiple qubits one has to just tensorize the Hilbert space, enabling also the entanglement between qubits

$$
\begin{aligned}
\left|\Phi^{+}\right\rangle &= \frac{1}{\sqrt{2}}\left|00\right\rangle + \frac{1}{\sqrt{2}}\left|11\right\rangle, \\
\left|\Phi^{-}\right\rangle &= \frac{1}{\sqrt{2}}\left|00\right\rangle - \frac{1}{\sqrt{2}}\left|11\right\rangle, \\
\left|\Psi^{+}\right\rangle &= \frac{1}{\sqrt{2}}\left|01\right\rangle + \frac{1}{\sqrt{2}}\left|10\right\rangle, \\
\left|\Psi^{-}\right\rangle &= \frac{1}{\sqrt{2}}\left|01\right\rangle - \frac{1}{\sqrt{2}}\left|10\right\rangle.
\end{aligned}
\tag{2.4}
$$

The states in (2.4) forms what is called *Bell basis* and is an example of maximally entangled states.

2.1 Quantum circuits

In general a n-qubits system is represented by a grid called *Quantum Circuit* depicting the operators that act on the qubits belonging to the Hilbert space $\mathcal{H} = \bigotimes_i^n \mathbb{C}_i^2$

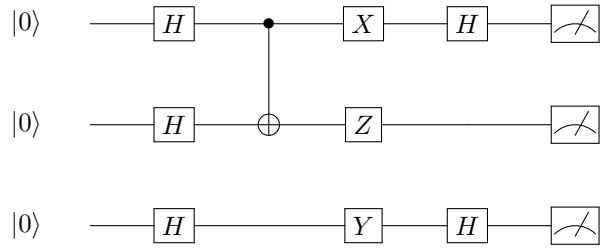


Figure 2.1: 3-qubit quantum circuit

In figure (2.1) are depicted all the fundamental quantum gates:

- The Hadamard gate $\hat{H}$
- The Pauli gates $\hat{X}, \hat{Y}, \hat{Z}$
- The entangling controlled 'not' (CNOT) gate

The CNOT gate acts applying the $\hat{X}$ operator on the target qubit based on the value of the control qubit

$$
\mathrm{CNOT}\left|c\right\rangle\left|t\right\rangle = \left|c\right\rangle \hat{X}^{c}\left|t\right\rangle \qquad c = 0, 1 \tag{2.5}
$$

and if paired with an Hadamard gate acts as an entangling gate producing the first of the bell states in eq. (2.4)

$$
\mathrm{CNOT}\big(\hat{H}\otimes\mathbb{I}\big)\left|0\right\rangle\left|0\right\rangle = \frac{1}{\sqrt{2}}\big(\left|00\right\rangle + \left|11\right\rangle\big) \tag{2.6}
$$

Other gates can be derived by the aforementioned ones by decomposition [41] or by exponentiation like the rotation gates which are represented in general as a parameterized gate:

$$\hat{R}(\boldsymbol{\theta}) = \exp\Big\{ -\frac{i}{2}\boldsymbol{\theta}\cdot\hat{\boldsymbol{\sigma}}\Big\} \tag{2.7}$$

In general the execution of a quantum algorithm is divided in 3 steps:

1. Initialization of the qubits to $|0\rangle$ or any other desired state applying additional gates
2. The quantum gates act on the qubits
3. The qubits are measured in a desired basis, usually $\hat{\sigma}_z$ or others via a basis-change gate at the end of the circuit

2.1.1 Errors and noise

Contemporary quantum computers are far from ideal and thus subject to errors in the computations due to noise with the external environment or among the qubits themselves. Errors in quantum circuits are easily representable in the formalism of *Kraus operators* [41].

The system-enviroment interaction is described as a global closed quantum system in order to have a globally unitary operation that results in different kind of errors on the quantum circuit once the environment has been traced out.

A general interaction with the environment is thus representable as

$$\hat{\rho}' \equiv \mathcal{E}(\hat{\rho}) = \mathrm{Tr}_{env}\Big[\hat{U}\big(\hat{\rho}\otimes\hat{\rho}_{env}\big)\hat{U}^\dagger\Big] \tag{2.8}$$

One can always assume that the environment starts in a pure state (otherwise one can consider a purification of the environment, either way it will be traced out at the end [41]).

$$\hat{\rho}_{env} = |e_0\rangle\langle e_0| \tag{2.9}$$

writing the trace explicitly over the environment base $\{|k\rangle\}$

$$\mathcal{E}(\hat{\rho}) = \sum_k \langle k|\hat{U}\big(\hat{\rho}\otimes|e_0\rangle\langle e_0|\big)\hat{U}^\dagger|k\rangle \equiv \sum_k \hat{E}_k\hat{\rho}\hat{E}_k^\dagger \tag{2.10}$$

$\hat{E}_k = \langle k|\hat{U}|e_0\rangle$ is the Kraus operator associated with the (global, acting on both the system and the environment) unitary operator $\hat{U}$ and this gives the possibility of modelling various type of noise through $\hat{U}$ by each $\hat{E}_k$.

Bit-flip channel

Given a probability p that the qubit undergoes a $\hat{X}$ gate (considering the canonical base this is the equivalent of a bit flip) we have the quantum operation

$$\mathcal{E}(\hat{\rho}) = (1-p)\hat{\rho} + p\hat{X}\hat{\rho}\hat{X} \tag{2.11}$$

results in the Kraus operators

$$\hat{E}_0 = \sqrt{1-p}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \hat{E}_1 = \sqrt{p}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.12}$$

Phase flip channel

The simplest noise channel that doesn't have a classical counterpart is assuming a random application of a $\hat{Z}$ gate (which is the same as a bit-flip in Hadamard base)

$$\mathcal{E}(\hat{\rho}) = (1-p)\hat{\rho} + p\hat{Z}\hat{\rho}\hat{Z} \tag{2.13}$$

with Kraus operators

$$\hat{E}_0 = \sqrt{1-p}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \hat{E}_1 = \sqrt{p}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.14}$$

Amplitude damping channel

A more severe error is when the qubit transitions from an exited state to a less energetic one (e.g. for stimulated or spontaneous emission) because the actual physical system that is represented by the qubits dissipates energy because of the environment (e.g. a photon in a cavity that is subject to scattering and attenuation)

$$\mathcal{E}(\hat{\rho}) = \hat{E}_0\hat{\rho}\hat{E}_0 + \hat{E}_1\hat{\rho}\hat{E}_1 \tag{2.15}$$

with

$$\hat{E}_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\lambda} \end{pmatrix} \quad \hat{E}_1 = \sqrt{1-\lambda}\begin{pmatrix} 0 & \sqrt{\lambda} \\ 0 & 0 \end{pmatrix} \tag{2.16}$$

with λ the rate of the process. The $\hat{E}_1$ operation changes a $|1\rangle$ state into a $|0\rangle$ state, corresponding to the physical process of losing a quantum of energy to the environment. $\hat{E}_0$ leaves $|0\rangle$ unchanged, but reduces the amplitude of a $|1\rangle$ state; physically, this happens because a quantum of energy was lost to the environment, and thus the environment now perceives it to be more likely that the system is in the $|0\rangle$ state, rather than the $|1\rangle$ state.

Depolarizing channel

The depolarizing channel is a type of noise that with probability p replaces the qubit with the completely mixed state $\hat{I}/2$ i.e.

$$\mathcal{E}(\hat{\rho}) = \frac{\hat{I}p}{2} + (1-p)\hat{\rho} \tag{2.17}$$

Observing that for an arbitrary state ρ

$$\frac{\hat{I}}{2} = \frac{\hat{\rho} + \hat{X}\hat{\rho}\hat{X} + \hat{Y}\hat{\rho}\hat{Y} + \hat{Z}\hat{\rho}\hat{Z}}{4} \tag{2.18}$$

we get

$$\mathcal{E}(\hat{\rho}) = \Big(1 - \frac{3p}{4}\Big)\hat{\rho} + \frac{p}{4}(\hat{X}\hat{\rho}\hat{X} + \hat{Y}\hat{\rho}\hat{Y} + \hat{Z}\hat{\rho}\hat{Z}) \tag{2.19}$$

i.e. a combination of various random gate errors.

Studying the behaviour of quantum circuits subjects to errors is crucial for assessing how a quantum algorithm is practically usable. In the following the performance of quantum circuits used as neural networks will be studied and quantum noise will be added to the circuits in the form of quantum operations before the measurement (or it will be naturally present in the real quantum device when used). Kraus operators hence give a quantitative way to model and study schematizations of quantum noise.

Chapter 3

Deep Learning

Learning algorithms are a vast group of computation tools with basically one goal: to extrapolate information from existing data in a robust way in order to be able to generalize them and make predictions. The discipline that encompasses learning algorithms and implement them on computers is machine learning, which comprehend a wide range of different algorithms such as regression, support vector machines (or more general kernel methods) or clustering.

The general steps of a machine learning algorithms can be summarized as:

1. The algorithm is fed with the input data
2. A series of parameterized operations (both linear and/or nonlinear) is acted upon the input data which is now a function of the parameters $\boldsymbol{\theta}$
3. The end result is outputted and fed to a so called *loss* or *cost* function $\mathcal{L}(\theta)$ to be minimized in the parameters space
4. The loss function is updated with the optimal parameters with the rule $\theta_i' = \theta_i - \varepsilon \partial_{\theta_i} \mathcal{L}(\theta)$ for each parameter θ_i, with ε the *learning rate* which is the magnitude of the step
5. Steps 2-4 are to be repeated until the algorithm converges to the desired level

The loss function $\mathcal{L}(\boldsymbol{\theta})$ is a lower bounded real function with the explicit expression depending on the algorithm used and the problem at hand; e.g. in supervised learning problems (given a set of paired $(\boldsymbol{x}, y)$ data find the best fitting function) is usually a metric which measures the distance between the algorithm output and the actual value corresponding to the data input (also called *ground truth*). At last, the optimization process in which the parameters of the algorithm are updated at each step is called *training phase* and is done with different techniques depending on the algorithm itself.

Among them there is a set of algorithm that stand apart, especially with the rapid improvement of hardware capabilities that the last decade has seen: neural networks (NN) architectures, which are named as *deep learning* algorithms [17].

Being very broad in terms of variety of the components of the different architectures, the basic ingredients of NNs can be summarized as:

- An input module
- A set of versatile computation units that can be staked together in series or in a parallel way
- Non linear operations between each computation unit
- An output module

The peculiarity of having a set of modular units that can be composed one after another, thus creating a large (deep) series of operations, is what has given this kind of algorithms the name of deep learning. The nonlinearities are what gives this architectures the ability of fitting even very nonlinear data without resorting to high dimensional mappings as with kernel methods, which could lead to a well known problem in learning algorithms: the curse of dimensionality. That means the difficulty in separating high dimensional data due to the sparsity they inherently gain as the dimensionality increases.

The most simple and paradigmatic but still very effective kind of NN is the *multilayer perceptron* [50].

3.1 Multilayer perceptron

A multilayer perceptron (MLP), or *feed-forward* neural network as it is called nowadays due to the absence of recurrent computation, is the paradigmatic example of a deep learning architecture.

It consists of layers of computational nodes (called units) each of which operates a weighted sum of its inputs, with the output value first fed to a non linear function, called *activation function* in resemblance to the schematization of a biological neuron.

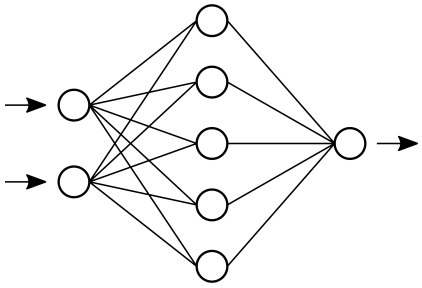


Figure 3.1: Schematization of a MLP with 2 input nodes, 1 hidden layer with 5 nodes and 1 output node

As portrayed in figure (3.1), the MLP consists in a few basic elements:

- Some input nodes (as many as the dimension of the input data)
- An arbitrary number of so called *hidden layers* that apply linear operations on their inputs followed by a nonlinear function
- An output layer with as many nodes as the dimension of the output of the problem considered (e.g. in a classification problem it is equal to the number of classes our data fall into)

In a more concrete way, as depicted in figure (3.1), an MLP acts as follows Let $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ be the input vector. A linear map is then applied

$$\boldsymbol{x} \to W\boldsymbol{x} + \boldsymbol{b} \equiv \boldsymbol{y} \tag{3.1}$$

with $W \in \mathcal{M}_{5x2}(\mathbb{R})$ and $\boldsymbol{b} \in \mathbb{R}^5$ respectively the weights matrix and biases vector, which values are the parameters optimized during the training phase. A nonlinear activation function is then applied and the result is again mapped to the output layer, which in this example consists in just one node:

$$\boldsymbol{y} \to f(\boldsymbol{y}) = \begin{pmatrix} f(y_1) \\ f(y_2) \\ f(y_3) \\ f(y_4) \\ f(y_5) \end{pmatrix} \tag{3.2}$$

$$f(\boldsymbol{y}) \to W'\boldsymbol{y} + b' \equiv y' \tag{3.3}$$

with another set of trainable parameters W' and b (where in this case the matrix-vector product is just a scalar product and b is a real number).

Due to the extreme flexibility of implementation and ability to generalized well especially unstructured data [17], NNs are becoming more and more popular in the majority of machine learning tasks such as computer vision, natural language processing and reinforcement learning.

Loss functions

The actual functional expression of the loss function depend heavily on the problem at hand:

- For regression problems (i.e. when the output is a real number (or vector), e.g. estimating the price of a house on the base of various data like city, conditions etc.) is a simple norm between the outputted value and the actual value

$$\mathcal{L}_{MSE} \equiv \frac{1}{N}\sum_{i}^{N} ||y_{true}^{(i)} - y_{pred}^{(i)}||_2 \tag{3.4}$$

like the *mean square error* (or L_2 norm) loss in eq. (3.4) in which N datapoints are considered. Another valid loss function could be L_1 norm, which choice penalizes equally big and small mistakes, while the mean square error loss, being quadratic, penalizes less small errors than big ones.

- For classification tasks, i.e. predicting the class label of an input data point based on its features, a loss function that measures how well the model's predicted classes match the true classes labels is needed. One of the most common choices is *cross-entropy* [17] loss, reported in eq. (3.5) for the binary 2-classes case

$$\mathcal{L}_{BCE} \equiv -\frac{1}{N}\sum_{i}^{N}\Big\{y_i \log(y_i') - (1-y_i)\log(1-y_i')\Big\} \tag{3.5}$$

  defining $0\log(0) \equiv 0$ the $\mathcal{L}_{BCE}$ function is minimized when $y_i = y_i' \quad \forall i$. For the general multiclass case the loss is a generalization of eq. (3.5)

$$\mathcal{L}_{CE} \equiv \frac{1}{N}\sum_{i}^{N}\mathcal{L}_{CE}^{(i)} = -\frac{1}{N}\sum_{i}^{N}\frac{1}{C}\sum_{c}^{C} y_i^{(c)}\log(y_i^c)' \tag{3.6}$$

  for a C classes problem.

The choice of the loss function for a given problem goes also by personal preferences and experience. These are just examples of what the functional expression of a loss function could be but in literature a lot more different functions exists and it is up to the practitioner to choose the best performing one.

Activation functions

Similarly to the loss function, there is a plethora of nonlinear activation functions to choose from.

- **Sigmoid**: the sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.7}$$

  is one of the first activation function to have been employed, especially because it resembles (figure (3.2)) the schematization of a biological neuron in which it fires/remains 'turned off' depending on a threshold value.

- **ReLU**: the rectified-linear unit (ReLU) is one of the (if not the) most used activation function. Contrary to the sigmoid it's not affected by the vanishing of the derivative for inputs outside a neighborhood of 0, as reported in figure (3.2), which stops the optimization routine.

$$\text{ReLU}(x) = \text{Max}\{x, 0\} \tag{3.8}$$

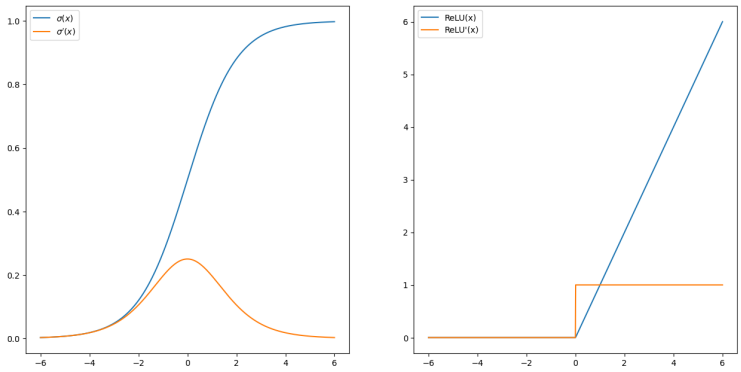


Figure 3.2: $\sigma(x)$ and ReLU(x) activation function. Note how $\sigma'(x)$ is non zero only in the neighborhood of 0

As with the loss function, the choice of the activation function for each layer is up to the user which has to choose the one that guarantees the best results. In order to make non zero the derivative also for negative inputs there exists different versions of the ReLU, which carry extra hyperparameters to hand-pick or higher computational complexity.

In figure (3.3) are reported two variants of the ReLU activation functions: notice how they both have non zero derivative around zero.

$$\text{Leaky-ReLU}(x) = \text{Max}(\alpha x, x) \tag{3.9}$$

$$\text{GeLU}(x) = x\Phi(x) \tag{3.10}$$

namely the leaky-ReLU in eq. (3.9) and the gaussian rectified liner unit (GeLU) in eq. (3.10). The former set a small but non zero slope even for negative inputs while the latter multiplies the input by $\Phi(x)$, the cumulative function of the normal distribution.

3.1.1 Optimization: the backpropagation algorithm

In general, after an input $\boldsymbol{x}$ has gone though a neural network with L hidden layer and θ the parameters, resulting in the output [17]

$$\tilde{y}(\boldsymbol{x}; \boldsymbol{\theta}) = f^L\big(W^L f^{L-1}\big(W^{L-1} \dots f^1\big(W^1 \boldsymbol{x}\big)\big)\big) \tag{3.11}$$

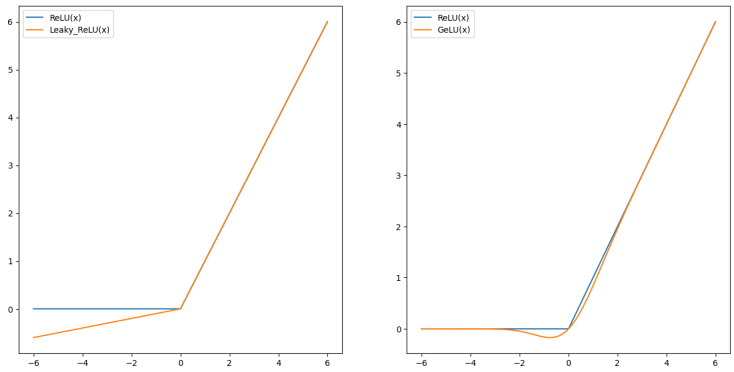


Figure 3.3: Comparison between ReLU and two variants: Leaky ReLu and GeLU activation functions

with W^i the linear operation of the i-th layer (comprising the bias addition, not illustrated for clarity of the equation) and f^i the activation function after the i-th layer, the training procedure starts.

The output is plugged in the loss function $\mathcal{L}\big(y, \tilde{y}(\boldsymbol{x}; \theta)\big)$ which is to be minimized. The minimization process is done with the so called *gradient descent* technique, i.e. finding the minimum of the loss (in the parameters space) calculating its gradient and choosing the new parameters going in the opposite direction, as illustrated in figure (3.4)

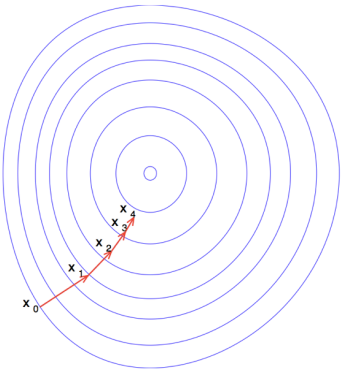


Figure 3.4: Gradient descent in the case of a 2D valued function

Each individual component of the gradient $\frac{\partial \mathcal{L}}{\partial w^l_{jk}}$ (w^l_{jk} the jk element of the l-th layer's weight matrix) can be computed by the chain rule; however. Backpropagation efficiently computes the gradient by avoiding duplicate calculations

without computing unnecessary intermediate values. It does this by computing the gradient of each layer, specifically the gradient of the weighted input, denoted as δ^l, from back to start.

The key point is that each weight matrix W^l affects the loss only through its effect on the next layer and in a linear way. Because of this, δ^l is the only data you need to compute the gradients of the weights at layer l, then one can compute δ^{l-1} of the previous layer and proceed recursively.

Doing this avoids inefficiency in two ways:

1. When computing the gradient at layer l it is not necessary to recalculate all the derivatives of following layers $l+1, l+2, \dots L$

2. Avoids unnecessary intermediate calculations since at each stage it directly calculate the gradient of the weights with respect to the last output (the loss function (3.3)), rather than unnecessarily computing the derivatives of the values of hidden layers with respect to changes in weights

Let be z^l, a^l respectively the weighted input and the output of each hidden layer l, $(f^l)'$ the derivatives of the activation function (evaluated in z^l) and x the input. The derivative of the loss function in terms of the input is given by the chain rule

$$\frac{d\mathcal{L}}{a^L} \circ \frac{da^L}{dz^L}\frac{dz^L}{da^{L-1}} \circ \frac{da^{L-1}}{dz^{L-1}}\frac{dz^{L-1}}{da^{L-2}} \circ \cdots \circ \frac{da^1}{dz^1}\frac{\partial z^1}{\partial x} \tag{3.12}$$

with $\circ$ the Hadamard element-wise product. These terms are respectively: the derivative of the loss, the derivatives of the activation functions and the matrices of weights:

$$\frac{d\mathcal{L}}{a^L} \circ (f^L)' \cdot W^L \circ (f^{L-1})' \cdot W^{L-1} \circ \cdots \circ (f^1)' \cdot W^1$$

Since the gradient ∇_x is the transpose of the derivative of the output with respect to the input, the matrices are transposed and the order reversed, but with the same entries

$$\nabla_x \mathcal{L} = (W^1)^T \cdot (f^1)' \circ \cdots \circ (W^{L-1})^T \cdot (f^{L-1}) \circ (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L}\mathcal{L} \tag{3.13}$$

Backpropagation basically consists of evaluating this gradient from left to right, calculating the gradient at each layer on the way. Let δ^l be defined as

$$\delta^l \equiv (f^l)' \circ (W^{l+1})^T \circ \cdots \circ (W^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L}\mathcal{L} \tag{3.14}$$

a vector of length equal to the number of nodes in layer l, each component interpreted as the error attributable to that node.

The gradient of the loss w.r.t. the weights in layer l is then

$$\nabla_{W^l}\mathcal{L} = \delta^l (a^{l-1})^T \tag{3.15}$$

and δ^l can easily be calculated recursively from right to left (so the name backpropagation)

$$\delta^{l-1} = (f^{l-1})' \circ (W^l)^T \cdot \delta^l \tag{3.16}$$

and so the gradients of the loss w.r.t. the weights can be computed using a few matrix multiplications for each layer.

Compared with the forward computation (eq. (3.17)) of δ^l, backpropagation (3.16) is clearly simpler.

$$\begin{aligned}
&\delta^1 = (f^1)' \circ (W^2)^T \cdot (f^2)' \circ \cdots \circ (W^{L-1})^T \cdot (f^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L} \mathcal{L} \\
&\delta^2 = (f^2)' \circ \cdots \circ (W^{L-1})^T \cdot (f^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L} \mathcal{L} \\
&\vdots \\
&\delta^{L-1} = (f^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L} \mathcal{L} \\
&\delta^L = (f^L)' \circ \nabla_{a^L} \mathcal{L}
\end{aligned} \tag{3.17}$$

These are the two main advantages of backpropagation with respect to forward propagation:

- Calculating the error δ^{l-1} in terms of δ^l avoids duplicate multiplications of layers from l and beyond
- Starting multiplying from $\nabla_{a^L} \mathcal{L}$ means that in each step only a vector-matrix product (between δ^l and $(W^l)^T$) and derivatives of the activations are done. On the other hand in the forward method each multiplication is a matrix-matrix multiplication, more expensive and corresponding to tracking every possible change in one layer l propagating forward in the layer $l + 2$, meaning unnecessary computations of the intermediate quantities of how a change in the weight affects the values of hidden nodes.

3.1.2 Training and evaluation: generalizing the data

As with every machine learning algorithm, one has to be sure that the model not only has learned the data, but has done it in an 'intelligent way' i.e. it is able to provide good results even with unseen data, when the model is deployed for usage.

In order to evaluate that, one has to take a look at the so called *loss curve*: after every iteration of the optimization process (called *epoch* in literature) the model is fed with a subset of the data available that has not been used for the updating of the parameters, the *validation* set. In doing so, one tests the performance of the model on data not used for the training and can plot the value of the loss function after each epoch with both the train and validation set, as in figure (3.5)

As the model learns (i.e. the training loss decreases), it must become better at generalizing (i.e. the validation loss must decrease), meaning that a good

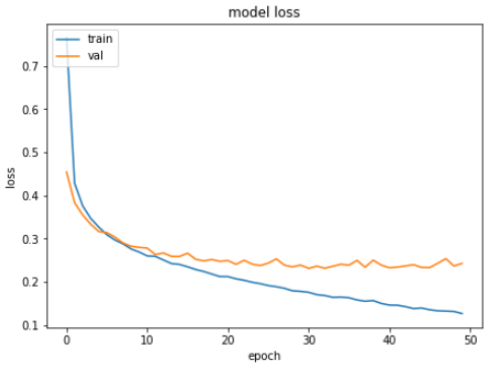


Figure 3.5: Training vs validation loss curve. Note how after epoch $\sim$ 10 the validation loss curve starts diverging from the train curve signalling overfitting

model must generate overlapping or at least parallel (with a small difference) train and validation losses. The worst case scenario is when the two start to diverge as in figure (3.5). This behaviour is commonly known as and happens when the model learns irrelevant but recurrent features in the training data, without finding a true correspondence in the unseen data (just like a student can learn mechanically how to do some exercises memorizing frequent calculations but without grasping the true meaningful concepts).

Avoiding overfitting

When overfitting is present (which is very frequent in neural networks operations) some tools can be used to overcome such a problem:

- Enlarging the dataset: as overfitting happens when a model cannot generalize data well, often the best way to avoid this problem is to supply the model with more data to learn (in fact NNs notoriously need a large amount of data to work properly).
- Augmenting the dataset: when getting more data is impossible one can add variety to the dataset in an artificial way: data augmentation techniques consist of modifying each datapoint applying (random or not) transformations. The reason of that is that the performance of neural networks can greatly improve even with a small change in input. For example in computer vision the most popular data augmentation techniques consists in small translations or rotations of the image, flips along the axes and slight changes of exposure. Each of these transformations help the neural network in the generalization of the data.
- Reducing the model complexity: often overfitting is simply a result of an overparameterized model (much like trying to fit n data points with and

n-1 degree polynomial, the functions will pass through all the data but is unlikely to be the real best-fit function).

- Dropout: with the same philosophy of reducing the model complexity but without reducing the number of parameters, a dropout layer 'turns off' a certain percentage of connections between nodes (meaning it puts the activation function to 0). This results in connections that are all equally important, without having a small fraction of parameters do the heavy lifting with the others hardly ever updated (which leads also to numerical difficulties during gradient descent).

- Regularizations: by forcing the weight not to acquire values too large one ensures a better fitting model. This is done by adding a term to the loss which minimizes also the weight norm, as in eqs. (3.18, 3.19) where both the L_2 and L_1 norms are reported.

$$\mathcal{L}(\boldsymbol{X};\boldsymbol{\theta}) \rightarrow \tilde{\mathcal{L}} = \mathcal{L}(\boldsymbol{X};\boldsymbol{\theta}) + \alpha||\boldsymbol{\theta}||_2 \tag{3.18}$$

$$\mathcal{L}(\boldsymbol{X};\boldsymbol{\theta}) \rightarrow \tilde{\mathcal{L}} = \mathcal{L}(\boldsymbol{X};\boldsymbol{\theta}) + \alpha|\boldsymbol{\theta}| \tag{3.19}$$

Obviously when this methods are used (outside from enlarging the dataset and reducing model complexity) one has to introduce additional hyperparameters like the dropout percentage p, the (in general much smaller than 1) constant α multiplying the additional loss term and the type of norm itself.

3.2 Convolutional neural networks

As MLP models present all nodes of each layer connected with all the nodes in the previous and subsequent layers (and thus also called *fully connected* or *dense* models nowadays), they present a huge number of parameters. In fact each connection between layers is characterized by a weight matrix W, the connections between two layers of n and m nodes result in m x n parameters and any correlations in the input data are lost.

The main difference between CNNs and fully connected models relies in how inputs are processed: in a CNN the connections between the input and the subsequent layers are limited to a portion of the input itself (thus resulting in way less parameters) and realized with a special type of layer called a *convolutional layer*, which applies a set of learnable filters (also called kernels) to the input.

The name 'convolutional' derives by the operation this kind of NN acts upon the input; it must be noticed that by applying the convolution kernels only locally on the input the correlations inside the input data are preserved.

Because of this improvement in comparison to dense models, in the last decade CNNs have been used in a ever increasing set of machine learning problems, from computer vision to speech recognition or natural language processing.

In general every time the input presents an intrinsic dimensionality (e.g. the spatial dimension of images or the length of a spectrogram).

Regarding computer vision tasks, in particular image recognition, the workflow of a CNN is illustrated in figure (3.6)

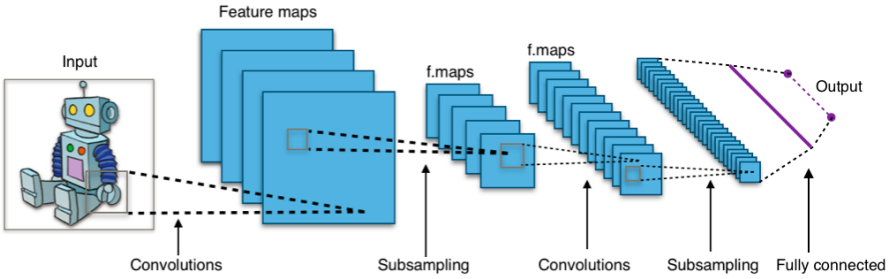


Figure 3.6: Convolutional neural network applying kernels on the input image and extracting feature maps

Convolutional filters are applied on the input image as a normal 2D convolution operation, outputting a still 2D object called *feature map* [17]

$$F^l(i,j) = (I * K^l)(i,j) = \sum_m^k \sum_n^k I(m,n) K^l(i-m, j-n) \tag{3.20}$$

with $I(m,n)$ the matrix of N x M pixels corresponding to the input image, K^l the k x k convolutional kernel of the l-th layer and F^l the feature map extracted.

As in every NN, after the weights the output is fed to a nonlinear activation function and the resulting feature maps are downsampled in order to limit the number of parameters of the network. The downsampling is usually a *MaxPool* operation in which from an area of p x p pixels the largest value is extracted (there also exists different possible downsampling e.g. extracting the mean value of the pixels).

After this the feature maps go through the same convolution-downsampling process for an arbitrary number of times, thus the key features of the initial image (e.g. lines, shadows, colored parts and so on) are mapped to more abstract feature maps which are smaller in spatial dimension but larger in number.

At last all the feature maps are reshaped as a column vector and concatenated into one dense layer which outputs the label belonging to the image.

The key advantage of CNNs over fully connected models is their ability to learn spatially invariant features. Because the same filter is applied across the entire input image, a CNN can detect the same feature regardless of its location in the image. This makes CNNs particularly well-suited for image classification tasks, where the location of the object in the image may vary. Additionally, the use of convolutional and pooling layers helps to reduce the number of parameters in the model, which makes it more efficient and less prone to overfitting.

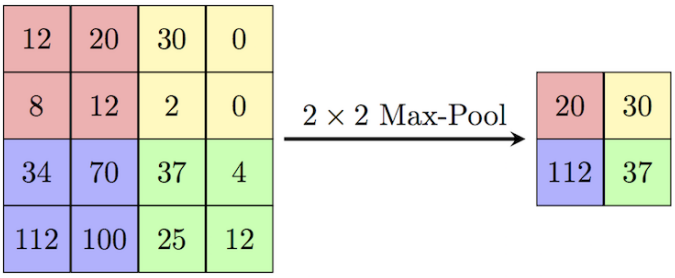


Figure 3.7: Action of a MaxPooling 2x2 layer on a 4x4 feature map

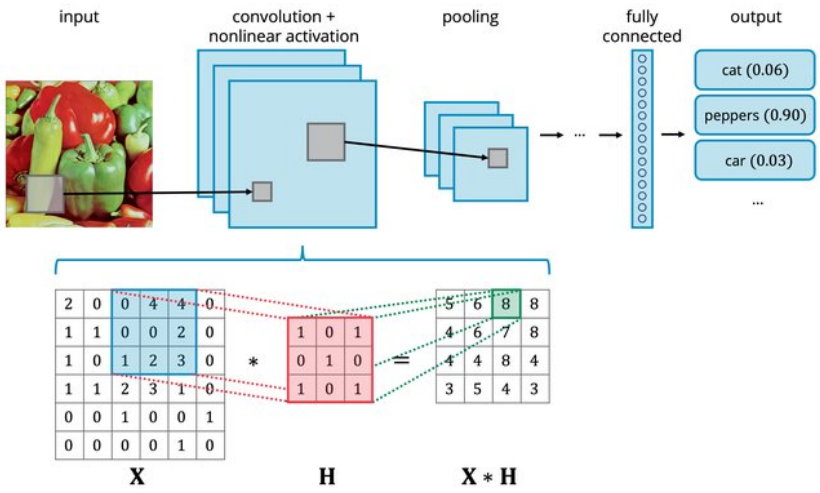


Figure 3.8: Working of a CNN with a focus on the convolution operation: the kernel acts like a dot product on a portion of the image ($X * H$) outputting a number

To summarize how a CNN works for image classification task:

1. The image is fed to the CNN
2. The image goes through the convolution operation: for each kernel a feature map is extracted from the image
3. The feature maps are downsampled
4. Repeat step 2,3 a given number of times
5. The feature map is flattened resulting in a 1d vector
6. The vector is fed to a fully connected layer which outputs the class which the image belongs to

In particular the label which the image belongs to is computed by a particular activation function: the *softmax* activation (3.21) $S : \mathbb{R}^n \to (0,1)^n$ with $(0,1)^n$ the space of normalized n dimensional vector for an n classes problem [17].

$$S(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_j^n e^{x_j}} \in (0,1) \tag{3.21}$$

After the flattened vector has passed through the fully connected layer with n output nodes is fed to the softmax function: as reported in eq. (3.21) each component is exponentiated (in order for it to be non negative) and normalized resulting in a vector which contains the probability for each class (i-th entry is the probability that the initial image belongs to the i-th class).

3.3 Generative models

Generative models are a class of machine learning algorithms with the aim of learning patterns in data in order to reproduce them. Unlike classification/regression models, generative architectures are trained in a unsupervised (or self-supervised) way: during the training the model is fed with the unlabelled training data and the goal is to reproduce them as accurately as possible, usually starting from random noise.

3.3.1 Generative adversarial networks

Generative adversarial networks (GANs) are a recent and popular class of generative models with the capability of reproducing accurately data in various machine learning tasks such as computer vision, natural language processing and audio recognition [18].

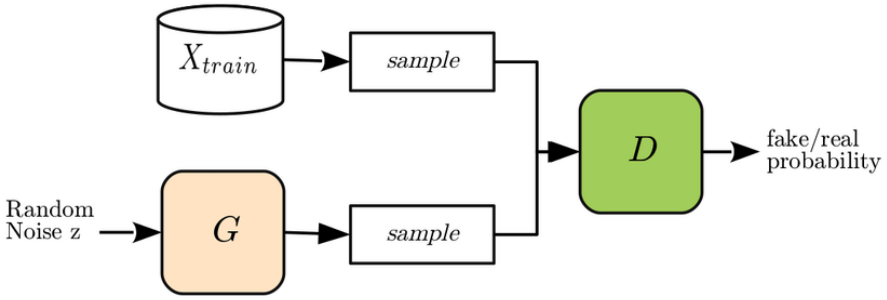


Figure 3.9: Scheme of the training process in a GAN

As reported in figure (3.9) a GAN model is composed of 2 neural networks: the generator G and the discriminator D. The aim of G is to produce synthetic

data as realistic as possible starting from random noise sampled from a low dimensional latent space while D serves as a discriminator between real and fake data: as G learns to produce more and more realistic data by trying to fool the discriminator, D tries to be as accurate as possible in the binary classification of real/fake data.

More formally, the training of a GAN model is a minimax game: given

- $p_z(\boldsymbol{z})$ the distribution over the latent space of the initial random noise $\boldsymbol{z}$
- $D(\boldsymbol{x};\theta_d)$ the parameterized discriminator function given by a MLP that defines the probability $D(\boldsymbol{x})$ that $\boldsymbol{x}$ comes from the data distribution $p_{data}(\boldsymbol{x})$ instead of the generator's distribution $p_g(\boldsymbol{z})$
- $G(\boldsymbol{z};\theta_g)$ the function (again given by a MLP) that maps the latent space to the data space representing the generator

We train D to maximize the probability of assigning the correct label to both training examples and samples from G while simultaneously train G to minimize $\log(1 - D(G(z)))$ resulting in the minimax problem

$$\min_G \max_D V(G, D) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D(\boldsymbol{x})] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(\boldsymbol{z})))] \tag{3.22}$$

The expectation values $\mathbb{E}$ is to be intended over the probability distributions given by the data, $P(\boldsymbol{x})$, and from which the noise is extracted, $P(\boldsymbol{z})$. In the training algorithm [18] one lets D guide the process: a weak discriminator results in a generator that outputs fake samples rather different from the real ones while a too strict discriminator prevents G from learning how to reproduce data in the first place.

The training is successful when the discriminator is no longer able to tell if data produced by the generator is real or fake, i.e. when $D(G(\boldsymbol{z})) \sim 0.5$.

Although successful architectures, GANs models are difficult to train and use: the learning rates for the discriminator and generator have to be carefully handpicked in order to avoid the collapsing of the model. What's more choosing the initial noise distribution and the generator's architecture is a far from trivial task.

A good generator has to be able to capture general patterns in the original data and exploit the dimensionality of the latent space as much possible in order to avoid overfitting. Usually the actual model consists of *upsampling* (sometimes called transposed convolutions) layers, i.e. operations that starting from a low dimensional vector (reshaped to be 2D) output an higher dimensional tensor (e.g. an image if the GAN model is used to generate images). Similarly to convolutions, upsampling layers works with a kernel that slides across the image (or in general across the dimensionality of the data), multiplies each pixel value with the values in the kernels and combine the results as illustrated in figure (3.10).

Algorithm 1 GAN Training algorithm with SGD. The hyperparameter k has to be chosen by the user

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$
- Sample minibatch of m real samples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \Big[\log D(\boldsymbol{x}^{(i)}) + \log(1 - D(G(\boldsymbol{z}^{(i)}))) \Big]$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(\boldsymbol{z}^{(i)})))$$

end for

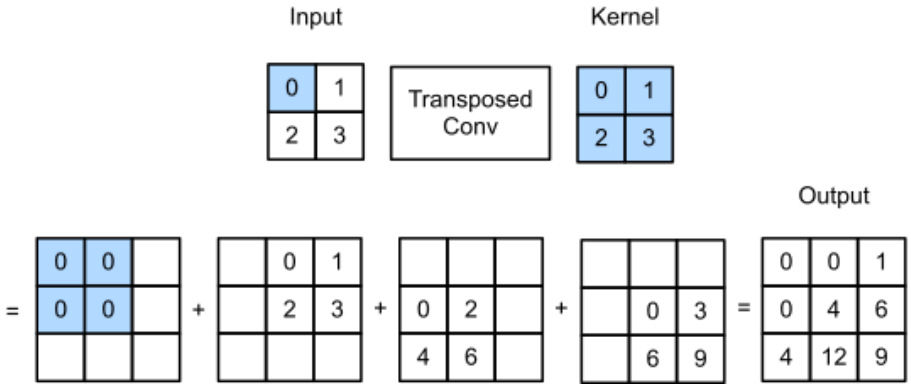


Figure 3.10: 2x2 transpose convolution kernel acting upon a 2x2 image producing a 3x3 image

The kernel acts upon each pixel value individually: each value is multiplied by the kernel's value producing a larger image. Each image is then combined (summed as in this case) and form the actual output.

On the other hand the choice of the discriminator is fairly simple: a binary classification network for the problem at hand (e.g. a CNN for image recognition or a recurrent network for speech recognition) will suffice to complete the task.

In figure (3.11) is illustrated how, from noise, a GAN outputs images more and more similar to the original ones as the training goes on.

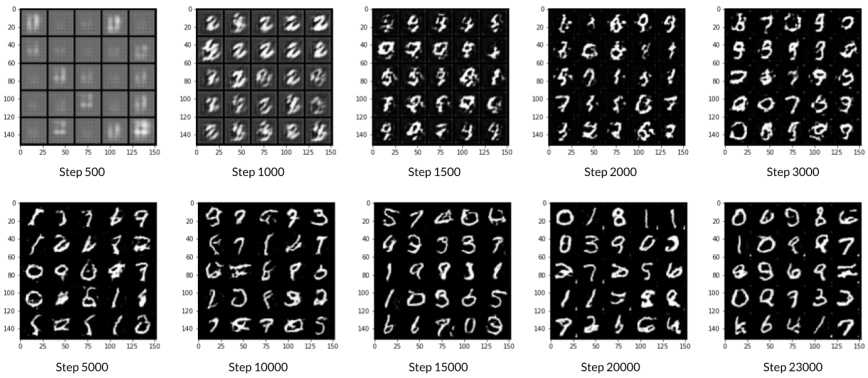


Figure 3.11: As the training steps increase the images are more and more similar to real ones, in this case images of written digits

Conditional generative adversarial networks

Defined as it is, the generative model is capable of reproducing the data in the train set but nothing else: in particular the user cannot decide what kind of data (i.e. which class) to reproduce.

In order for the generator to be able to forge a particular kind of data, additional information have to be fed to the model: by providing to the generative model also the class labels one conditions it not only to reproduce data similar to the original ones, but also paired with the right label as now the job of the discriminator is to tell whether the data itself is fake or real but also if it is paired with the correct label (a perfectly generated image but of the wrong class would still be rejected by the discriminator).

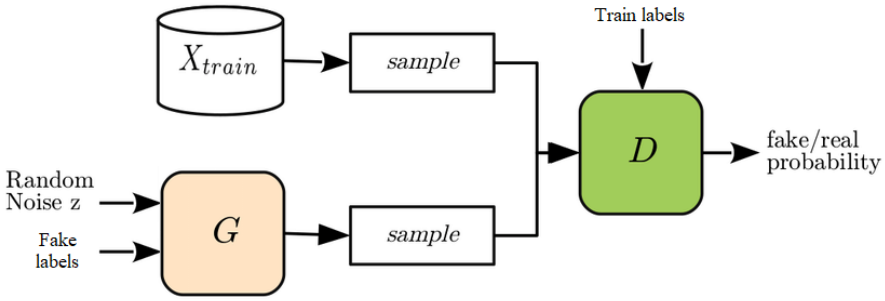


Figure 3.12: Conditional GAN

Chapter 4

Quantum Machine Learning

Exploiting the capability of quantum computers of solving linear algebra problems [19], in recent years many attempts to prove a quantum advantage also with machine learning algorithms have been made [12, 34, 47, 54, 35] but they all suffer from the same drawbacks: the limitations of the HHL [1] algorithm once implemented and the large resources required (both in terms of qubits and circuit-depth).

In order to understand the capabilities of NISQ devices pertaining machine learning one necessitates of quantum algorithms that are backed-up by some classical algorithms where the formers still lack feasible implementation, such as in solving the optimization problem of the fitting function.

Being small scale and less susceptible to noise [37, 39], variational quantum circuits are the best candidates for enhancing machine learning with quantum computation, at least for the current state of quantum devices.

4.1 Variational quantum circuits

Variational quantum circuits (VQC) are hybrid quantum-classical architectures composed of a parameterized quantum circuit followed by a classical optimization routine, such as stochastic gradient descent (SGD). The architecture of a VQC is relatively simple, consisting of two main parts: the quantum circuit and the classical optimizer.

The quantum circuit is typically composed of a set of quantum gates that can be parameterized, usually single or controlled rotation gates. The rotation angles are the parameter which the classical optimizer will adjust during the optimization process.

The classical optimizer is responsible for finding the optimal values of these parameters that minimize a given cost function. A wide range of cost functions are used, both problem dependent (such in the studies of energy level

[1] Harrow-Hassidim-Lloyd

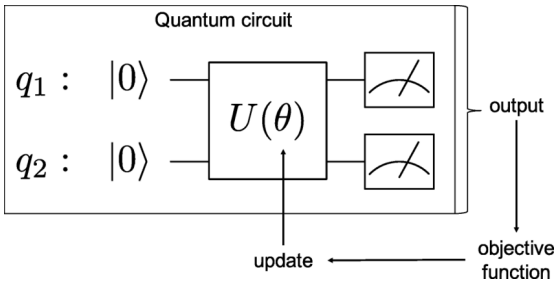


Figure 4.1: Schematization of a VQC, the parametrized circuit $U(\theta)$ is applied to the qubits which are then measured. The output of the measure is plugged in the cost function which is minimized by the classical routine, resulting in an update of the aforementioned circuit's parameters

of molecules [28]) or problem independent, usually connected with a classical supervised setting [14].

The basic steps of a VQC are as follows:

1. Initialize the quantum circuit with some initial parameter values
2. Initialize the qubits in a certain base (usually the $|0\rangle$ state)
3. Apply the quantum circuit $\hat{U}(\theta)$ to an input state
4. Measure the output of the quantum circuit using a set of measurement operators
5. Calculate the cost function $C(\theta)$ based on the measurement results and which will depend on the circuit's parameters
6. Use a classical optimizer to adjust the parameters of the quantum circuit
7. Iterate steps 2-5

The key advantage of VQCs is that they can potentially solve problems faster than classical algorithms and this because quantum circuits can generate exponentially large spaces of quantum states[20], which can be used to search for optimal solutions more efficiently than classical algorithms.

Variational quantum circuits have a wide range of potential use cases, some of which include:

- Quantum chemistry: VQCs can be used to simulate the electronic structure of molecules and calculate their energy levels, which is crucial for understanding chemical reactions and designing new compounds.
- Optimization: VQCs can be used to solve optimization problems, such as the traveling salesman problem, more efficiently than classical algorithms. This could have applications in logistics, finance, and other industries.

- Machine learning: VQCs can be used to perform machine learning tasks, such as classification and regression, by optimizing the quantum circuit parameters to minimize a cost function. This could lead to more efficient and accurate machine learning models.
- Quantum error correction: VQCs can be used to study and develop quantum error correction codes, which are essential for building large quantum computers.

The process of having a parameterized architecture for which the parameters are tuned accordingly to the value of a certain cost function resembles closely the functioning of a neural network. The next natural step is then study how one can have the best of both computational paradigms: the advantage of quantum computation and the versatility of neural networks.

4.1.1 Gradients with quantum circuits

As the output of a VQC is given by a quantum measurement, one will have in general a result expressed by

$$f(\theta) = \langle\psi|\hat{U}^\dagger(\theta)\hat{A}\hat{U}(\theta)|\psi\rangle \tag{4.1}$$

with $\hat{A}$ some observable (usually the Pauli matrix $\hat{Z}$) and $U(\theta)$ the VQC.

The result $f(\theta)$ will be plugged in the chosen cost function during the computations and so a way of differentiating a function with respect to the circuits parameters θ is needed.

The usual way in which this is done comes by the name of *parameter shift rule* [10, 6, 52] and doesn't require additional ancilla qubits.

For the single qubit gate, due to the fact that it can be expressed by the smooth function

$$\hat{U}(\theta) = e^{-\frac{i}{2}\theta\hat{H}}$$

with $\hat{H}^\dagger = \hat{H}$, calculating the derivative with respect to θ gives [10]

$$\frac{\partial}{\partial\theta}f(\theta) = r\left[f\Big(\theta + \frac{\pi}{4r}\Big) - f\Big(\theta - \frac{\pi}{4r}\Big)\right] \tag{4.2}$$

independently of $\hat{A}$ and $\hat{H}$ [10] assuming that $\hat{H}$ has 2 distinct eigenvalues λ_1, λ_2 and with $r = \frac{1}{4}|\lambda_1 - \lambda_2|$, which is the case with all the operators used in quantum circuits.

For a general VQC composed of various gates (and remembering that each multi qubit gate can be decomposed in single qubit rotations and non parametrized controlled gates [46, 40]) one simply use the fact that each parameter is independent of each other and gets

$$\nabla_\theta f(\theta) = \sum_i \langle\psi_{i-1}|\frac{\partial}{\partial\theta_i}\left\{\hat{U}_i^\dagger(\theta_i)\hat{A}_{i+1}\hat{U}_i(\theta_i)\right\}|\psi_{i-1}\rangle \tag{4.3}$$

with

$$f(\theta) = \langle\psi|\hat{U}_1^\dagger(\theta_1)\dots\hat{U}_n^\dagger(\theta_n)\hat{A}\hat{U}_n(\theta_n)\dots\hat{U}_1(\theta_1)|\psi\rangle$$

and

$$|\psi_i\rangle = \hat{U}_i(\theta_i)\dots\hat{U}_1(\theta_1)|0\rangle$$

$$\hat{A}_{i+1} = \hat{U}_N^\dagger(\theta_N)\dots\hat{U}_{i+1}^\dagger(\theta_{i+1})\hat{A}\hat{U}_{i+1}(\theta_{i+1})\dots\hat{U}_N(\theta_N)$$

$|0\rangle = |0\rangle^{\bigotimes N}$ the multi qubit initialized state, so the gradient is analytically computable for every parameter and can be plugged in a bigger backpropagation-like chain of derivaties.

Derivative of a rotation gate

As an example let's see how parameter shift rule works with the standard rotation gate

$$\hat{U}_i(\theta_i) = \exp\Big(-\frac{i}{2}\hat{\sigma}_i\theta_i\Big) \tag{4.4}$$

being a smooth function it can be easly derived

$$\partial_{\theta_i}\hat{U}_i(\theta_i) = -\frac{i}{2}\hat{U}_i(\theta_i)\hat{\sigma}_i \tag{4.5}$$

and plugging this in equation (4.3) we obtain

$$\partial_{\theta_i} f(\theta_i) = \frac{i}{2}\langle\psi_{i-1}|\hat{U}_i^\dagger(\theta_i)\Big[\hat{\sigma}_i, \hat{A}_{i+1}\Big]\hat{U}_i(\theta_i)|\psi_{i-1}\rangle \tag{4.6}$$

as proven by [39] the following identity holds for Pauli matrices and observables operators:

$$\Big[\hat{\sigma}_i, \hat{A}_i\Big] = -i\left(\hat{U}_i^\dagger\Big(\frac{\pi}{2}\Big)\hat{A}\hat{U}_i\Big(\frac{\pi}{2}\Big) - \hat{U}_i^\dagger\Big(-\frac{\pi}{2}\Big)\hat{A}\hat{U}_i\Big(-\frac{\pi}{2}\Big)\right) \tag{4.7}$$

substituting in equation (4.6) we finally obtain

$$\begin{aligned}\partial_{\theta_i} f(\theta_i) =& \frac{1}{2}\langle\psi_{i-1}|\Bigg\{\hat{U}_i^\dagger\Big(\theta_i + \frac{\pi}{2}\Big)\hat{A}_{i+1}\hat{U}_i\Big(\theta_i + \frac{\pi}{2}\Big) - \\ &- \hat{U}_i^\dagger\Big(\theta_i - \frac{\pi}{2}\Big)\hat{A}_{i+1}\hat{U}_i\Big(\theta_i - \frac{\pi}{2}\Big)\Bigg\}|\psi_{i-1}\rangle\end{aligned} \tag{4.8}$$

which in terms of the single state quantum function, like equation (4.1), it is rewritten like

$$\partial_{\theta_i} f(\theta_i) = \frac{1}{2}\Big[f\Big(\theta_i + \frac{\pi}{2}\Big) - f\Big(\theta_i - \frac{\pi}{2}\Big)\Big] \tag{4.9}$$

just like in equation (4.2) with $r = \frac{1}{2}$ (as with Pauli matrices) and independently of the observable $\hat{A}$.

4.1.2 Variational quantum circuits and noise

As with all quantum circuits, VQCs are affected by different kinds of noise and errors aswell but due to the fact that a variational quantum circuit is 'learned' (in contrast of other well-known algorithms which have fixed parameters or so called 'black-box' gates), it is worth to analyze how different kind of errors change the performances of VQCs.

Stochastic noise

During the optimization process, the update rule for each parameter is given by the equation

$$\theta_i' = \theta_i - \varepsilon \partial_{\theta_i} \mathcal{L}(\theta) \tag{4.10}$$

with $\mathcal{L}(\theta)$ the cost function, $\varepsilon > 0$ the learning rate and the with the derivatives computed with the back-propagation algorithm or parameter-shift rule for classical and quantum operations respectively.

It goes without saying that where the derivative is zero the parameters are not updated, i.e. the learning process is stopped, which is desirable when the loss has reached a minimum but unfortunately it may happen in saddle points too, where the first derivative is zero but the hessian matrix $\partial_{\theta_i}\partial_{\theta_j}\mathcal{L}(\theta)$ has eigenvalues of different signs.

In order to avoid being stuck in saddle points various methods can be applied, among which the recent *perturbed gradient descent* algorithm [27], which modifies the update rule of the optimization routine:

$$\theta_i' = \theta_i - \Big(\varepsilon \partial_{\theta_i} \mathcal{L}(\theta) + \zeta\Big) \tag{4.11}$$

with ζ a stochastic variable normally distributed as $\mathcal{N}(0, \sigma)$. In [27] the authors show that using this update rule the probability of getting stuck in a saddle point is significantly reduced, pointing out that this result is generalizable to more general distribution of the ζ stochastic variable.

As the authors in [33] pointed out, VQCs are naturally subjected to this kind of noise during the estimation of the derivatives via parameter-shift rule:

$$\theta_i' = \theta_i - \varepsilon g_i(\theta) \tag{4.12}$$

where $g_i(\theta)$ is the estimation of the derivative ∂_{θ_i} after N measurements of the quantum device such in equation (4.3). Being the quantum measurement a stochastic process one can define the quantity

$$\zeta_N \equiv \partial_{\theta_i} \mathcal{L}(\theta) - g_i(\theta) \tag{4.13}$$

which has zero mean (as shown in [27]). Putting all together, the stochastic nature of the quantum measure gives a natural implementation of the stochastic gradient descent algorithm without introducing some handpicked random variable as in the classic case:

$$\theta_i' = \theta_i - \Big(\varepsilon g_i(\theta) + \zeta_N\Big) \tag{4.14}$$

As shown by [33] both on simulators and on quantum devices, the presence of measurement noise, even the readout noise (which affect all current quantum devices and not only the noise intrinsically present in quantum measurements), gives a drastic improvement in the minimization of the cost function avoiding saddle points.

Flip noise

As illustrated in [37], VQCs are robust to like bitflip-like and phase flip-like noise errors. While ideally a VQC can be represented as a gate $\hat{U}(\boldsymbol{\theta})$, due to errors usually one has a different operator $\hat{U}'(\boldsymbol{\theta})$. An error like this is said to be suppressable if there exists a vector $\boldsymbol{\beta}$ such that

$$||\hat{U}(\boldsymbol{\theta}) - \hat{U}'(\boldsymbol{\theta} + \boldsymbol{\beta})|| \ll 1 \tag{4.15}$$

with the norm intended as operatorial. The value of such vector $\boldsymbol{\beta}$ is irrelevant due to the fact that can be learned during the optimization routine and it doesn't affect the performance if it just translates the optimum of the cost function to an equivalent optimum, which is true when the errors preserve the circuit symmetry [37].

This is the case of bitflip and phaseflip errors, encoded respectively as the Pauli gates $\hat{X}$ and $\hat{Y}$. Usually VQCs encodes information as parameterized rotations on the states and so for these kind of errors the symmetry is preserved having that

$$\left[\hat{X}, e^{-\frac{i}{2}\theta_x \hat{X}}\right] = \left[\hat{Y}, e^{-\frac{i}{2}\theta_y \hat{Y}}\right] = 0$$

Decoherent noise

While certain kind of noise like bit error gates or stochastic measurement noise are, up to a certain amount, tolerated by VQCs, decoherent noise like amplitude damping and depolarizing channels are still a problem to overcome in order to realize fault-tolerant quantum computation.

As shown in [15], especially depolarizing noise is detrimental for the correct execution of a quantum algorithm: the more the depolarizing channel is strong, the more the states (with $d = dim(\mathcal{H})$ being the dimension of the Hilbert space) are turned in the maximally mixed state

$$\hat{\rho} = \frac{1}{d}\hat{\mathbb{I}}_d \tag{4.16}$$

yielding to the impossibility of distinguishing between different states during the measurement.

Amplitude damping also has a severe effect on the functioning of the VQC: the relaxation from the excited states to the ground states results again in the impossibility of classifying the data from the output of the circuit.

On the other hand recently remarkable progress have been made in order to mitigate decoherent noise in variational quantum circuits, although still remaining very problem-dependent solutions: the most interesting approaches comes from the studies of energy levels and dynamics of chemistry-related hamiltonians [38, 32] where additional operators are added to the circuits in order to protect the information contained in the states in a redundant way, resulting in a mitigation of the decoherence errors.

4.2 Quantum neural networks

A quantum neural network (QNN) can be defined as a traditional neural network in which the computation is carried out by a quantum circuit.

Usually the computation of a feed-forward neural network consist of a linear operation performed on the input data followed by a non-linear activation function whereas in a QNN the building blocks of classical neural networks, such as neurons and layers, are replaced with quantum mechanical systems, such as qubits and quantum gates.

Being an hybrid quantum-classical paradigm of computation, QNN are usually cast in two categories depending on the data they eleborate:

- Quantum data, where one has the interest of studying the evolution of the quantum states itself
- Classical data, where the goal is to map a problem which is usually dealt by classical neural network in a Hilbert space

4.2.1 Quantum data: simulating quantum systems with quantum computers

Both cases are being currently explored. The former, it is usually addressed as 'quantum simulation'. In every problem concerning the study of a quantum system, from the spectrum of a certain hamiltonian operator associated to a molecule or material to the simulation of the quantum dynamics of a model, being able to codify the states directly on a quantum computer can result in being able to solve problems of prohibitive complexity for classical computation.

A lot of focus is being given to condensed matter [49, 3, 4] and quantum chemistry [28, 51] due to the natural mathematical description of these systems with a hamiltonian operators.

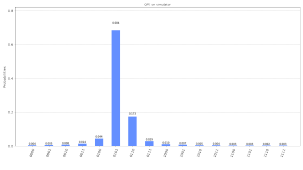
There are different use cases of quantum neural networks applied to quantum simulation; some of the more prominent are:

- Mapping the interactions of the hamiltonian directly in the topology of the quantum device [4, 3]
- Compressing the ground state in order to reduce the computational resources needed
- Studying quantum materials and phase transition directly on quantum computers [9, 26]

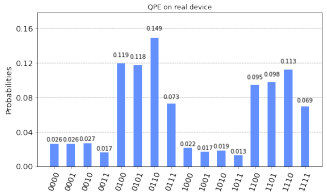
Quantum simulation through VQCs seems the best road to follow even when compared to the other most prominent method: the quantum phase estimation (QPE) algorithm.

QPE have the aim to compute eigenvalues of unitary gates encoding them in the phase of quantum states on which the gate is applied. Being one of the so called 'black-box' algorithms where one as to assume the possibility of implementing the aforementioned gate in the actual quantum circuit, the possibility of simulating it directly (analogical simulations) or via trotterization (digital simulation) makes it suitable for this kind of tasks.

Even if one of most promising quantum algorithm, QPE suffers a lot from noise and errors: since it is a gatewise-deep and complicated algorithm it is very susceptible to spurious interaction between qubits and the enviroment and also between qubits themselves. In figure (4.2) are reported some experiments underlining the problems of QPE:



(a) QPE on perfect simulator



(b) QPE on real device

Figure 4.2: Comparison between QPE on ideal device vs on noisy device. It can be noted the almost uniform distribution of the states on the real device, making the true phase 0101 indistinguishable

In the experiment the QPE algorithm is being studied with the following gate:

$$U = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2}{3}\pi i} \end{pmatrix}$$

which has a phase of $\varphi = \frac{1}{3} = 0.\bar{3}$, expressible as 0101 with 4 bits precision while the quantum computing framework used is IBM's `qiskit`, with the 5-qubits `ibmq-bogota` as real device. It is evident how QPE still suffers a lot from noise, a problem to which VQCs are instead a lot less subject to.

4.2.2 Classical data: enhancing neural networks with quantum computation

Regarding quantum data the benefits are obvious: being able to study the properties and evolution of quantum states directly (in a digitally or analogically simulation) on a quantum device avoids computationally-expensive numerical simulations.

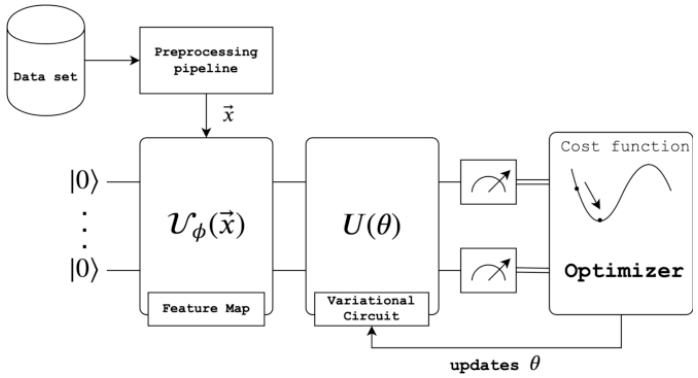


Figure 4.3: Working blocks of a QNN: the classical data $\boldsymbol{x}$ is encoded in the variational quantum circuit by the ansats $\mathcal{U}_\phi(\boldsymbol{x})$ and then processed by the gates $U(\theta)$

On the other hand the question whether and how quantum computing could aid classical machine learning algorithms remains to be explored. One not yet so well understood drawback of neural networks is the ability to generalize data although being over parameterized: because of this, recent research has been focused on the question whether a quantum circuit is better at sampling distribution and taking advantage of the high dimensionality of the Hilbert space where the classical data must be embedded into [20].

As illustrated in figure (4.3), when classical data is to be processed one has first to encode it in the quantum circuit itself: this can be done in different ways according to a certain arbitrary *ansatz*, that is a map from the data space to the Hilbert space of the circuit $\phi : \mathbf{R}^n \to \mathcal{H}$. Some of the most popular ansatzes are [30]

- Amplitude embedding: the value of each datapoints x_i is encoded in the amplitude of each qubits $x_i \to x_i|0\rangle_i$
- Basis embedding: for binary data only, each string of bits is encoded in the correspondent qubit in σ_z base $0101 \to |0101\rangle$
- Angle embedding: each datapoint is encoded in a phase applied to each qubit with a certain rotation matrix $x_i \to \hat{R}_j(x_i)|0\rangle_i = \exp\{-\frac{i}{2}x_i\hat{\sigma}_j\}|0\rangle_i$ with $j = x, y, z$ one of the 3 Pauli matrices

The search for the best ansatzes is currently an active area of research, and each of the aforementioned ones present pros and cons: amplitude embedding is simple but requires an handpicked very small constant in order to give a non zero amplitude to the qubits associated with zero valued datapoints; basis embedding requires a logarithmically increasing number of qubits (despite the linearity of the other 2 methods) but works only with binary data; angle embedding requires an additional gate for the encording, increasing the circuit depth.

Recently it has also been studied how certain ansatzes are inefficient [20]: methods that results in product states like

$$\boldsymbol{x} = (x_1, x_2, x_3) \rightarrow |x_1\rangle|x_2\rangle|x_3\rangle$$

are very easily simulated by a classical computer thus not providing any quantum advantage, suggesting the need of more carefully thought embeddings and that entanglement could play an important role in the choice of the ansatz in a fashion like the one reported in figure (4.4)

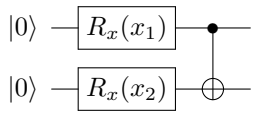


Figure 4.4: Entangling anstatz

Although a lot of research is still based on empirical results and conjectures, recent years have seen an increase of quantitative results, usually leveraging on Statistical Learning Theory. This is the discipline that analyzes statistical models from a rigorous and fundamental point of view, trying to define quantitatively concepts such as expressibility (i.e. being able to fit complicated functions with a finite amount of data) and generalizability (i.e. predicting accurately unseen data) of a model.

A particular focus has been given to information-theory properties of quantum neural networks [2], studying the properties of the Fisher information matrix

$$F_{ij}^n \equiv \frac{1}{n}\sum_k^n \frac{\partial}{\partial\theta_i}\log p(x_k, y_k; \boldsymbol{\theta})\frac{\partial}{\partial\theta_j}\log p(x_k, y_k; \boldsymbol{\theta}) \tag{4.17}$$

In eq. (4.17) is reported the so called *empirical Fisher information matrix*. The neural network is seen under the statistical learning theory point of view: a statistical model describing the joint relationship between data pairs (x,y) as

$$p(x, y; \boldsymbol{\theta}) = p(y|x; \boldsymbol{\theta})p(x) \tag{4.18}$$

$\forall x \in \mathcal{X} \subset \mathbf{R}^{d_{in}}, y \in \mathcal{Y} \subset \mathbf{R}^{d_{out}}, \boldsymbol{\theta} \in \boldsymbol{\Theta}$ where $\mathcal{X}$ is the set of input datapoints (the features), $\mathcal{Y}$ the set of output datapoints (the labels for classification tasks

or scalar values for regression tasks) and $\boldsymbol{\Theta}$ the empirical parameters space. In fact it is explicitly constructed considering a finite amount of datapoints available (n in eq. (4.17)).

The reasons because the Fisher information matrix is a useful object to study when assessing the properties of a model are manifold:

1. Trainability of the model: for the majority of the common loss functions the fisher matrix is proven to be equivalent to the hessian matrix of the loss itself [29].

   Studying the eigenvalues of the hessian matrix [31] sheds light to the profile of the loss functions: quantifying the curvature of the loss landscape helps predicts whether a model will present barren plateaus [8], i.e. wide and flat areas that inhibit the gradient descent optimization, resulting in a worse and sub-optimized model.

   In particular one is interested in whether quantum neural networks present less vanishing eigenvalues respect to classical models that could results in QNNs having better and faster (epoch-wise) training, which has been verified in [2].

2. Model capacity: the class of functions a model can fit with a certain number of parameters and datapoints (so a concept connected to the aforementioned expressibility). As stated by [2] the model capacity is represented by the *effective dimension* (eq. (4.19)), a quantity that depends on the fisher information matrix and describe the model's parameter space, in particular how much the parameters are significant when fitting a function. An higher effective dimension is thus associated with the need of less parameter and a simpler model in order to achieve similar performances.

As reported in [2, 5], the effective dimension of a model $\mathcal{M}_\Theta$ for a model with a d-dimensional parameter space Θ, n datapoints and $\tilde{F}_{ij}^n$ normalized fisher matrix is expressed by

$$d(\mathcal{M}_\Theta) \equiv 2\frac{\log\left[\int_\Theta \mathrm{d}\theta\sqrt{\det\left(\mathbf{I}_d + \frac{n}{2\pi\log n}\tilde{F}(\boldsymbol{\theta})\right)}\right]}{\log\left(\frac{n}{2\pi\log n}\right)} \tag{4.19}$$

with

$$\tilde{F}(\boldsymbol{\theta})_{ij} \equiv d\frac{V_\Theta}{\int_\Theta \mathrm{d}\theta \mathrm{tr}\big[F(\boldsymbol{\theta})\big]}F_{ij}(\boldsymbol{\theta}) \tag{4.20}$$

$$V_\Theta \equiv \int_\Theta \mathrm{d}\theta \tag{4.21}$$

the volume of the d-dimensional parameter space.

We have again that large eigenvalues of the fisher matrix are associated with a better model: as the eigenvalues grow so does the determinant in eq. (4.19) and so does the effective dimension by monotonicity of the square root function. As studied both numerically and on real quantum computers by [2], this is actually the case: the fisher matrix of quantum neural network models have proven to exhibit a more evenly distributed spectrum, with a greater part of eigenvalues taking values greater than 0.

Quantum convolutional neural networks

Based on the result of the previous section a lot of work has been directed towards the question whether a QNN can be used to process images in a fashion similar to convolutional neural networks.

In one of the first studies of this field [21] it has been proved that a simple quantum feature map (i.e. extracting feature maps of images using a quantum circuit before a CNN) is not sufficient to guarantee superior performances to classical models.

From that point it has been clear that one necessitates a stronger bond between the convolutional neural network and the quantum operations: resembling the versatility and stacked-blocks design of classical deep learning architectures one way could be building modules of quantum computation that can be serialized with the standard classical layers such as pooling and flattening, as illustrated in figure (4.5).

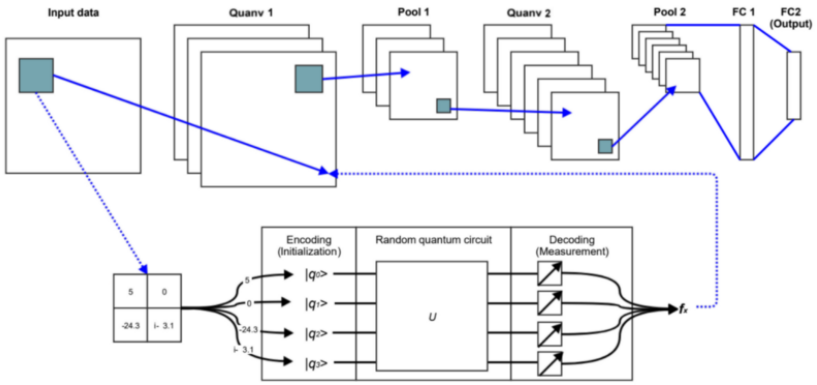


Figure 4.5: Schematizing of a quantum convolutional neural network (QCNN)

As in a CNN the kernel processes local portions of the image, in a quantum convolutional neural network one processes the input data feeding it to a variational quantum circuit (that according to [2], with a good encoding scheme, could result in better performances than a classical kernel) which is then measured in order to extract the pixel values for the subsequent feature map, with

the quantum measurement itself acting as the nonlinear activation function (although because of the hybrid nature of QCNN it is possible to use other activations as well).

4.2.3 Data Re-Uploading scheme

Currently a lot of different approaches to QCNN are being studied with promising results [9, 53, 24] but all of them have to fight with some important obstacles: the need for shallow and small architectures in order to employ current small-scale and noisy quantum devices [45] but at the same time complicated enough to ensure a quantum advantage [20, 2] and the capability to process datasets belonging to real-world problems.

A promising compromise between circuit simplicity and effectiveness is the *Data Re-Uploading* scheme [43], capable of dealing with non trivial datasets even with a single qubit.

The data re-uploading scheme solves for the practitioner the task of choosing the embedding gates: the embedding and variational parts of the circuits, usually separated as in figure (4.3), are encoded in the same gates which reduce the number of gates stacked in the circuit. As an example, if $\boldsymbol{x} \in \mathbf{R}^3$

$$\boldsymbol{x} = (x_1, x_2, x_3) \to \hat{\mathcal{U}}(\boldsymbol{x}; \boldsymbol{\phi}) \left|0\right\rangle = \hat{U}(\boldsymbol{x} \circ \boldsymbol{w} + \boldsymbol{\theta}) \left|0\right\rangle \tag{4.22}$$

The input is embedded in a *learned* parameterized rotation matrix where $\boldsymbol{\phi} = \{\boldsymbol{w}, \boldsymbol{\theta}\}$ are the parameters and $\circ$ is the component-wise product

$$\boldsymbol{x} \circ \boldsymbol{y} = (x_1, x_2, x_3) \circ (y_1, y_2, y_3) = (x_1 y_1, x_2 y_2, x_3 y_3)$$

The operator $\hat{\mathcal{U}}(\boldsymbol{x}; \boldsymbol{\phi})$ act both as and embedding layer and quantum kernel

$$\hat{\mathcal{U}}(\boldsymbol{x}; \boldsymbol{\phi}) = \exp\Big\{ -\frac{i}{2} \hat{\boldsymbol{\sigma}} \cdot (\boldsymbol{x} \circ \boldsymbol{w} + \boldsymbol{\theta}) \Big\} \tag{4.23}$$

The choice of the rotation axis $\hat{R}_x$, $\hat{R}_y$ or $\hat{R}_z$ performed both in the embedding and in the computation part is operated by the values of the parameters: small values of w_i and θ_i implies a small rotation angle around the $\boldsymbol{i}$ axes.

For higher dimensional case is easily generalized: the input

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$$

is subdivided in groups of 3 datapoints each

$$\{(x_1, x_2, x_3), \ldots, (x_{n-2}, x_{n-1}, x_n)\} \equiv \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$$

$k = \left\lceil \frac{n}{3} \right\rceil$, 0 padded if the dimension is not divisible by 3. The data is then embedded in the circuit applying eq. (4.22) k times:

$$\boldsymbol{x} \to \hat{\mathcal{U}}_k(\boldsymbol{x}_k; \boldsymbol{\phi}_k) \ldots \hat{\mathcal{U}}_1(\boldsymbol{x}_1; \boldsymbol{\phi}_1) \left|0\right\rangle \tag{4.24}$$

resulting in the quantum circuit in figure (4.6)

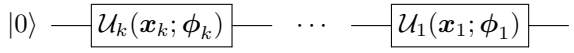


Figure 4.6: Data Re-Uploading circuit for higher dimensional data

The embedding scheme $\varphi : \mathbf{R}^n \to \mathcal{H}$ is a non-linear map since:

$$\varphi(\boldsymbol{x}+\boldsymbol{y}) \neq \varphi(\boldsymbol{x}) + \varphi(\boldsymbol{y}) \tag{4.25}$$

with

$$\varphi(\boldsymbol{x}+\boldsymbol{y}) = \exp\left\{ -\frac{i}{2}\sum_j \hat{\sigma}_j\Big((x_j+y_j)w_j+\theta_j\Big)\right\}|0\rangle \tag{4.26}$$

and

$$\begin{aligned}\varphi(\boldsymbol{x})+\varphi(\boldsymbol{y}) = \exp\left\{ -\frac{i}{2}\sum_j \hat{\sigma}_j\big(x_j w_j+\theta_j\big)\right\}|0\rangle + \\ + \exp\left\{ -\frac{i}{2}\sum_j \hat{\sigma}_j\big(y_j w'_j+\theta'_j\big)\right\}|0\rangle\end{aligned} \tag{4.27}$$

This a crucial feature in neural networks computations [17] and results in a circuit complex enough to satisfy the requirements suggested in [20] for quantum advantages. It can be generalized also to multiple qubits introducing also entanglement, as reported in figure (4.8).

Multi-qubit case

For an arbitrary number of datapoints we denote the gates in figure (4.6) as

$$\hat{\mathcal{U}}_k(\boldsymbol{x}_k;\phi_k)\ldots\hat{\mathcal{U}}_1(\boldsymbol{x}_1;\phi_1)\,|0\rangle \equiv \hat{L}(1) \tag{4.28}$$

and each datapoint can be re introduced in the circuit (so the name "data re-uploading") in order to increase the nonlinearity and complexity of the model (figure (4.7)):

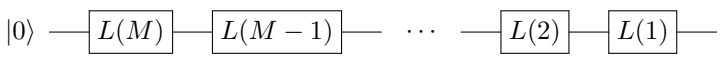


Figure 4.7: Data Re-Uploading circuit with M uploading layers

Introducing multiple qubits one can also exploit entanglement:

The datapoints are embedded in the circuit multiple times on each register, every time with a different gate $\hat{L}_j(i)$ for the i-th layer of the j-th register

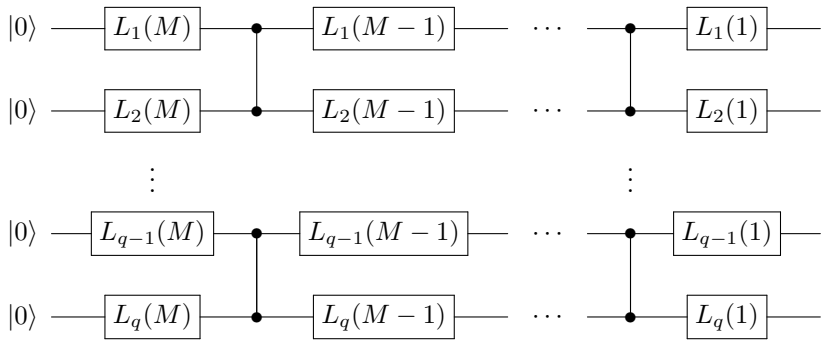


Figure 4.8: Data Re-Uploading circuit with M uploading layers, q qubits and entangling layers between each gate. The way in which the entanglement is implemented is up to the practitioner

introducing more parameters and nonlinearities between the data. After each uploading layer one can also use entangling layers to increase the model complexity.

With a multi-qubit quantum circuit one has the flexibility of choosing the number of outputs: with a n qubits quantum circuit one can implement a fully-connected layer with n-output nodes or a convolutional layer. Since a convolutional layer outputs a single number (as illustrated in section 3.2) that can be achieved measuring only one of the qubits (e.g. the last one as in [43]) or measuring all of them and calculating the mean of the outputs, in resemblance of what happens in a classical convolutional layers. In this work the latter approach has been chosen as it has been observed it gives better results when compared to the former.

A single qubit as a universal classifier

The intuition and justification for this particular scheme comes from the classical NNs theorem *Universal approximation theorem* [22]:

Let $I_m = [0,1]^m$, $C(I_m)$ the space of continuous functions on I_m. $\varphi : \mathbf{R} \to \mathbf{R}$ a non constant bounded and continuous function and $f : I_m \to \mathbf{R}$ a function. Then $\forall \varepsilon > 0$ $\exists N \in \mathbf{N}$, $h : I_m \to \mathbf{R}$ defined as

$$h(\boldsymbol{x}) = \sum_i^N \alpha_i \varphi(\boldsymbol{w}_i \cdot \boldsymbol{x} + b_i) \tag{4.29}$$

with $\alpha_i, b_i \in \mathbf{R}$, $\boldsymbol{w}_i \in \mathbf{R}^m$ such that h is approximation of f, i.e.

$$|h(x) - f(x)| < \varepsilon \tag{4.30}$$

$\forall \boldsymbol{x} \in I_m$.

In a neural network, with φ the activation function, this theorem states that is possible to approximate a function with a single layer of N nodes. In [22] a proof of the theorem for a general activation function and a corollary are given: φ could be a nonconstant finite linear combination of periodic functions, e.g. a trigonometric polynomial.

The layers of a single qubit classifier are composed of general $SU(2)$ operators, each of which can be decomposed as follows:

$$\hat{U}(\boldsymbol{\phi}) = \hat{U}(\phi_1, \phi_2, \phi_3) = e^{i\phi_2\hat{\sigma}_z} e^{i\phi_1\hat{\sigma}_y} e^{i\phi_3\hat{\sigma}_z} \tag{4.31}$$

that can be expressed as a single exponential [43]:

$$\hat{U}(\boldsymbol{\phi}) = e^{i\boldsymbol{\omega}(\boldsymbol{\phi})\cdot\hat{\boldsymbol{\sigma}}} \tag{4.32}$$

with $\boldsymbol{\omega} \in \mathbf{R}^3$

$$\begin{aligned}
\omega_1(\boldsymbol{\phi}) &= d\mathcal{N}\sin\Big((\phi_2 - \phi_3)/2\Big)\sin\Big(\phi_1/2\Big) \\
\omega_2(\boldsymbol{\phi}) &= d\mathcal{N}\sin\Big((\phi_2 - \phi_3)/2\Big)\sin\Big(\phi_1/2\Big) \\
\omega_1(\boldsymbol{\phi}) &= d\mathcal{N}\sin\Big((\phi_2 + \phi_3)/2\Big)\cos\Big(\phi_1/2\Big)
\end{aligned} \tag{4.33}$$

where $\mathcal{N} \equiv \big(\sqrt{1-\cos^2 d}\big)^{-1}$ and $\cos d \equiv \cos\Big((\phi_2+\phi_3)\Big)\cos\Big(\phi_1/2\Big)$. The data re-uploading scheme codifies the datapoints in the parameters $\boldsymbol{\phi}$ of the $SU(2)$ gate, in particular one has

$$\boldsymbol{\phi}(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \phi_3(\boldsymbol{x})) = \boldsymbol{w}\cdot\boldsymbol{x} + \boldsymbol{\theta} \tag{4.34}$$

and when datapoints are uploaded N times like in figure (4.7) we obtain

$$\hat{\mathcal{U}}(\boldsymbol{x}) = \hat{U}_N(\boldsymbol{x})\ldots\hat{U}_1(\boldsymbol{x}) = \prod_i^N e^{i\boldsymbol{\omega}(\boldsymbol{\phi})\cdot\hat{\boldsymbol{\sigma}}} \tag{4.35}$$

Applying the Baker-Campbell-Hausdorff (BCH) formula one obtains [43]

$$\hat{\mathcal{U}}(\boldsymbol{x}) = \exp\Big\{i\sum_i^N \boldsymbol{\omega}\big(\boldsymbol{\phi}(\boldsymbol{x})\big)\cdot\hat{\boldsymbol{\sigma}} + \hat{\mathcal{O}}_{corr}\Big\} \tag{4.36}$$

with the remaining corrections $\mathcal{O}_{corr}$ proportional to the Pauli matrices due to the commutation relations $[\sigma_i, \sigma_j] = 2i\varepsilon_{ijk}\sigma_k$.

The $\boldsymbol{\omega}$ are trigonometric functions hence nonconstant, continuous and bounded and using (4.34) we obtain

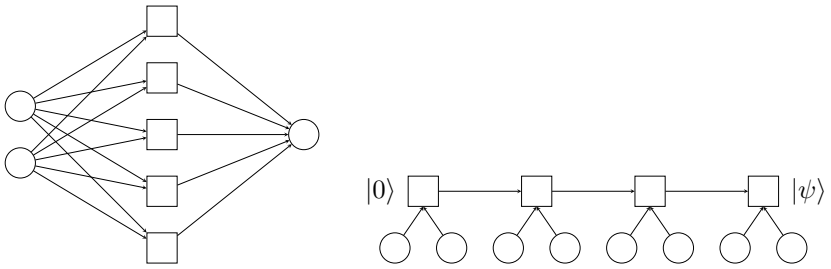
$$\begin{aligned}
\sum_i^N \boldsymbol{\omega}\big(\boldsymbol{\phi}(\boldsymbol{x})\big) &= \sum_i^N \Big(\omega_1\big(\boldsymbol{\phi}(\boldsymbol{x})\big), \omega_2\big(\boldsymbol{\phi}(\boldsymbol{x})\big), \omega_3\big(\boldsymbol{\phi}(\boldsymbol{x})\big)\Big) = \\
&= \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), f_3(\boldsymbol{x})\big)
\end{aligned} \tag{4.37}$$

What remains to do is deal with the $\hat{\mathcal{O}}_{corr}$ terms due to the BCH formula. Because of the group composition properties of $SU(2)$ one can write $\hat{\mathcal{U}}(\boldsymbol{x}) = e^{i\boldsymbol{\xi}(\boldsymbol{x})\cdot\hat{\boldsymbol{\sigma}}}$ where $\boldsymbol{\xi}(\boldsymbol{x})$ is a trigonometric function resulting from the composition of the N gates. Being the $\hat{\mathcal{O}}_{corr}$ terms proportional to the pauli matrices we have $\hat{\mathcal{O}}_{corr} = \boldsymbol{\rho}(\boldsymbol{x}) \cdot \hat{\boldsymbol{\sigma}}$ for some function $\boldsymbol{\rho}(\boldsymbol{x})$. Putting all together one has

$$\hat{\mathcal{U}}(\boldsymbol{x}) = e^{i\boldsymbol{\xi}(\boldsymbol{x})\cdot\hat{\boldsymbol{\sigma}}} = e^{i\boldsymbol{f}(\boldsymbol{x})\cdot\hat{\boldsymbol{\sigma}}+i\boldsymbol{\rho}(\boldsymbol{x})\cdot\hat{\boldsymbol{\sigma}}} \tag{4.38}$$

and thus the $\hat{\mathcal{O}}_{corr}$ terms can be absorbed into $\boldsymbol{f}(\boldsymbol{x})$. We than have a correspondence between the universal approximation theorem for classical neural networks in [22] and the working of tha data re-uploading scheme.

A more intuitive way of seeing the similarity between the data re-uploading scheme and an MLP is in the idea of feeding the model with the data multiple times itself. In a classical neural network the vector of input datapoints is fed to each computation unit (i.e. each node) due to the fact that a matrix vector product consists in multiple dot products between the input vector and each row-vector of the weight matrix. Similarly in the data re-uploading scheme the datapoints are fed to the model multiple times, one for each layer as in eq. (4.35).



(a) MLP (b) Data re-uploading scheme

Figure 4.9: As in the MLP the data are introduced in each node, in the data re-uploading architecture the data is uploaded in each layer (squares)

In figure (4.9) is illustrated how the multiple introduction of the data is done in the quantum circuit. In fact, because of the no cloning theorem one cannot simply upload the data, process them, save the state and redo this routine again with the resulting state as the next initial state.

The data re-uploading scheme solves this problem merging the embedding and computation gates of the circuit: the data are embedded with general $SU(2)$ rotation which are also parameterized. Doing that allows the upload of the data an arbitrary number of time.

4.2.4 Quantum circuits as generative models: the patch GAN

Recently a lot of effort has been put into proving both theoretically and empirically [48, 36, 16, 11] the capabilities of variational quantum circuit as generative models.

As quantum circuits presents an high expressibility in representing continuous distributions of unstructured data [45], the most fitting role for a variational quantum circuit is the one of the generator in the discriminator-generator scheme of GANs. Still following the line of choosing the best compromise between performance and computational resources required (i.e. when $N \lesssim \log\lceil M \rceil$ with N the available qubits and M the dimensionality of the data), one solution could be as suggested in [23]: to distribute the whole generative task to multiple subcircuits and still benefit of quantum advantage, the so called *patch GAN*.

In this scheme the generator G is composed of a subset of generators $\{G_k\}_{t=1}^{K}$ where each G_k is a parameterized quantum circuit $\hat{U}_k(\boldsymbol{\theta}_k)$.

Starting from an embedded low-dimensional vector of noise $\boldsymbol{z} \to |\boldsymbol{z}\rangle$, the aim of each G_k is to output a state $|G_k(\boldsymbol{z})\rangle = \hat{U}_k(\boldsymbol{\theta}_k)\,|\boldsymbol{z}\rangle$ that represent a portion of the high-dimensional data.

Once the outputs of all the generator's quantum circuits are measured and the generated data reconstructed, the training still follows the minimax game as in the classical case

$$\begin{aligned} &\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \mathcal{L}(D_{\boldsymbol{\theta}_d}(G_{\boldsymbol{\theta}_g}(\boldsymbol{z})), D_{\boldsymbol{\theta}_d}(\boldsymbol{x})) = \\ &\mathbb{E}_{\boldsymbol{x}\sim P_x}\big[\log D_{\boldsymbol{\theta}_d}(\boldsymbol{x})\big] + \mathbb{E}_{\boldsymbol{z}\sim P_z}\big[\log(1 - D_{\boldsymbol{\theta}_d}(G_{\boldsymbol{\theta}_g}(\boldsymbol{z})))\big] \end{aligned} \tag{4.39}$$

with P_x, P_z the distribution of real data and latent noise data respectfully and $\mathcal{L}$ a suitable loss function, analogous to the classical case.

The patch GAN model is reported in figure (4.10):

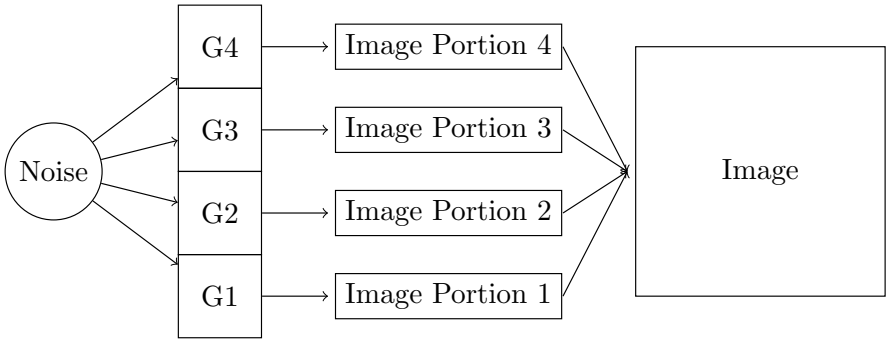


Figure 4.10: Quantum generator

Assuming that an N-qubit quantum device is available, the functioning of each generator G_k is the same: a noise vector $\boldsymbol{z}$ is fed to G_k (figure (4.10))

preparing the state $|\boldsymbol{z}\rangle = \bigotimes_i^N \hat{R}_Y(\alpha_{z_i}) |0\rangle^{\otimes N}$ with α_{z_i} the refactored noise element to be $\in (0, \pi)$, e.g. if $\boldsymbol{z} \sim \text{Unif}(0, \pi)^N$, $\alpha_{z_i} = z_i$, $i = 1 \dots N$.

$|\boldsymbol{z}\rangle$ is then fed to the generator $\hat{U}^{(k)}(\boldsymbol{\theta}^{(k)})$, whose architecture is reported in figure (4.11): it consists of L layers of trainable unitary single qubits rotation gates $\hat{U}^{(k)}(\boldsymbol{\theta^{(k)}})$ and controlled Z gates for entanglement.

The final part of the circuit consists of a partial measure in order to introduce additional non-linearities to the output on top of the usual final total measure in $\boldsymbol{z}$ base. The N total qubits are divided in N_A ancillary qubits for the partial measure and $N_G = N - N_A$ qubits for the subgenerator itself. Denoting $|\Psi(\boldsymbol{z})\rangle$ the quantum state at the end of the circuit in figure (4.11) a partial measure on the ancillary qubits is taken resulting in the state

$$\hat{\rho}^{(k)}(\boldsymbol{z}) = \frac{\text{Tr}_{\mathcal{A}}\big(\Pi_{\mathcal{A}} \otimes \mathbb{I}_{2^{N_G}} \left|\Psi^{(k)}(\boldsymbol{z})\right\rangle \left\langle\Psi^{(k)}(\boldsymbol{z})\right| \big)}{\text{Tr}\big(\Pi_{\mathcal{A}} \otimes \mathbb{I}_{2^{N_G}} \left|\Psi^{(k)}(\boldsymbol{z})\right\rangle \left\langle\Psi^{(k)}(\boldsymbol{z})\right| \big)} \tag{4.40}$$

with $\mathcal{A}$ the ancillary subsystem, $\Pi_{\mathcal{A}} = \bigotimes^{N_A} |0\rangle \langle 0|$. The equation (4.40) denotes a nonlinear mapping in $\boldsymbol{z}$ that helps further the expressibility of the model [17, 23]. The output of subgenerator $G^{(k)}$ (e.g. one of the image portions in figure (4.10)) is obtained by measuring $\hat{\rho}^{(k)}(\boldsymbol{z})$ with a complete set of computational basis vectors $\{|j\rangle\}_{j=0}^{2^{N_G}-1}$. For image generation the result $P(j)$ represents the j-th pixel value of the k-th subgenerator:

$$P(j) = \text{Tr}\big(|j\rangle \langle j| \, \hat{\rho}^{(k)}(\boldsymbol{z}) \big) \tag{4.41}$$

The output of the k-th subgenerator is thus obtained concatenating the results

$$G_k(\boldsymbol{z}) = [P(0) \dots P(2^{N_G} - 1)] \tag{4.42}$$

and globally the image is reconstructed by further concatenating all the previous outputs

$$G(\boldsymbol{z}) = [G_1(\boldsymbol{z}) \dots G_K(\boldsymbol{z})] \tag{4.43}$$

Due to the fact that the K quantum circuits implementing each subgenerator can be executed serially reconstructing the image once the process is finished: the patch GAN is then capable of elaborating fairly high dimensional data (e.g. image generation benchmark datasets such as MNIST, CIFAR10 etc.) still using a limited number of qubits.

As reported in [23], the algorithm has been implemented using R_Y rotations and CZ entangling gates.

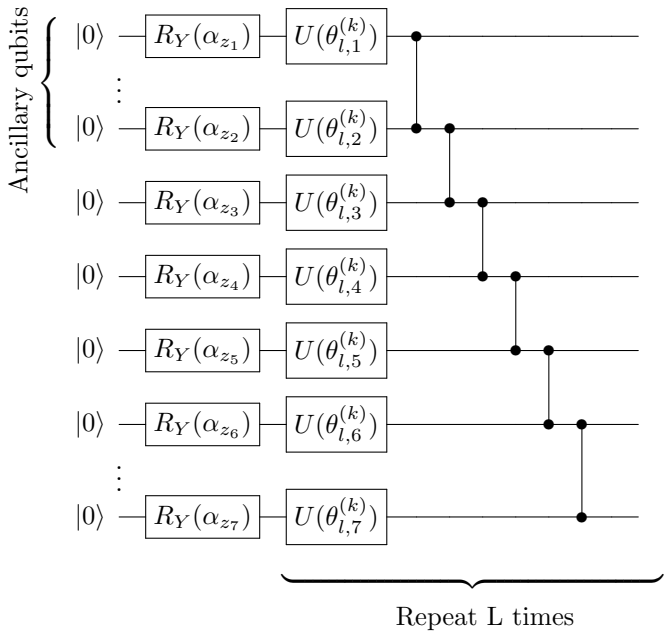


Figure 4.11: Quantum circuit for patch quantum GAN

Chapter 5

Quantum machine learning for computer vision

As a demonstration of the effectiveness of quantum computation applied to machine learning tasks, some application on real-world problems is needed. Computer vision (CV) is one of the most machine learning based area of computer science, especially in recent years with the rapid improvement of computation hardware and optimization of deep learning architectures.

Due to the high dimensionality of the data (a NxN pixels RGB image is represented as a tensor with dimension $3N^2$), the area of computer vision is always looking for improvements on both computational capabilities and algorithm efficiency and thus a quantum computation framework for CV could be a game changer for the years to come.

In order to adapt the quantum machine learning algorithms presented in chapter 4 for computer vision tasks some changing have been made, in particular for the data reuploading scheme.

As classical CNN elaborates data in a local way preserving the spatial features of the data such as relative positions and shapes applying a convolution on the image, the data reuploading ansatz has been adapted to process only local portions of the input data.

The use of quantum machine learning in computer vision problems will be divided in two main cases: image generation and image classification.

For image generation the performance of the PATCH GAN architecture will be proven with the standard CV benchmark datasets: MNIST (a collection of handwritten digits in greyscale), Fashion MNIST (a collection of clothing items images in grey scale) and CIFAR10 (everyday objects such as vehicles and animals in RGB format).

With regard to the data reuploading scheme applied to CV, beside the aforementioned standard datasets, a real-world problem will be addressed: the identification of parkinson desease by analysis of PET scan brain images [44].

The algorithm will be first analysed studying performance and scalability

also addressing the effect of noise on the results. In the end the results of the aforementioned architectures will be given and compared to classical neural networks with a similar number of parameters.

5.1 Image classification

In order to have a correspondence with CNNs, the input scheme of the data reuploading ansatz has been modified: instead of loading the full image in the quantum circuit (operation both computationally inefficient and extremely time consuming), the data are fed to the quantum circuit in a fashion similar to the classical case: a sliding window is applied to the image and instead of doing a convolution between the pixels under the window and the neural network's kernel, the pixel values under the window are fed to the quantum circuit that acts as a quantum kernel, in a similar way as described in [20] and illustrated in image (4.5).

The code for the quantum neural networks has been developed in order to be fully integrable within the `tensorflow`[1] framework as callable Python classes with the aid of the `tensorflow-quantum` [7] and `cirq` [25] libraries. All of the quantum layers are usable within the classical pipeline of an usual deep learning model i.e. they are fully compatible with all the others layers present in the library.

5.1.1 Performance analysis

First it has been studied how the data reuploading scheme scales by modifying the number of layers and qubits: using the MNIST dataset, a small quantum neural network (figure (5.1)) composed of 2 convolutional layers with a maxpool operation in the middle followed by a fully connected classical softmax output layer has been benchmarked in order to asses the rough usability.

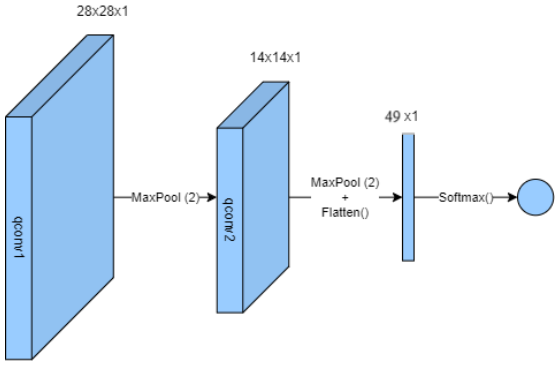
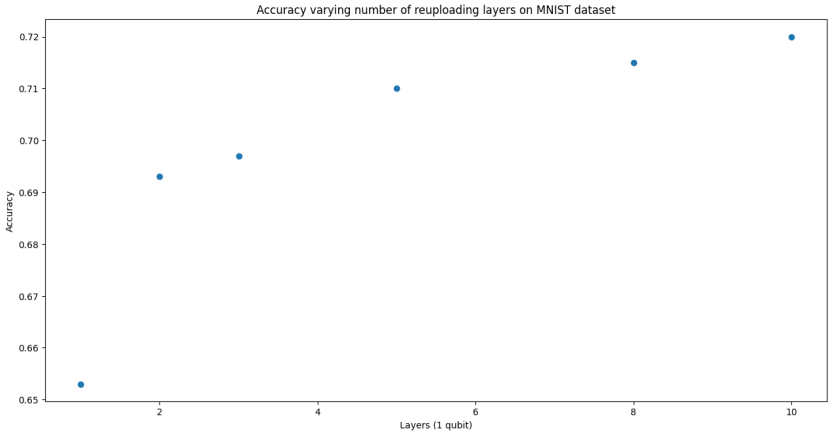


Figure 5.1: Quantum neural network used for the performance analysis

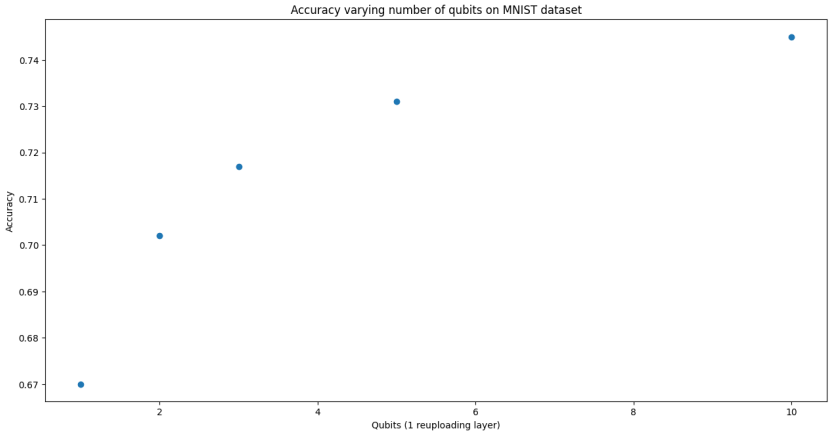
The parameters of interest are how the classification accuracy varies with the

number of reuploading layers in each convolutional layer and with the number of qubits in the quantum circuit utilized.

The second type of test concerned the effects on noise in the quantum circuit, in particular the classification accuracy with different amplitude damping parameters.



(a) Accuracy vs number of reuploading layers



(b) Accuracy vs number of qubits

Figure 5.2: Accuracy of the data reuploading image classifier on MNIST dataset varying the number of reuploading layers/qubits with fixed number of qubits/layers

Amplitude damping noise has been seen to have a serious detrimental effect

on quantum circuit performance [13]. Proving that a certain algorithm is robust for at least small amplitude damping parameters is crucial in the search for quantum advantage.

As reported in subsection 4.1.2 and proved by the results in the next section, bit and phase flip type errors do not constitute a particularly serious issue with error rate comparable to contemporary quantum devices.

The tests are done with a numerically simulated quantum circuit in order to have the possibility of studying the ideal system and isolating only amplitude damping noise, which is implemented as a quantum operation given by the kraus operators 2.15 at the end of each quantum circuit composing the neural network (figure (5.3)).

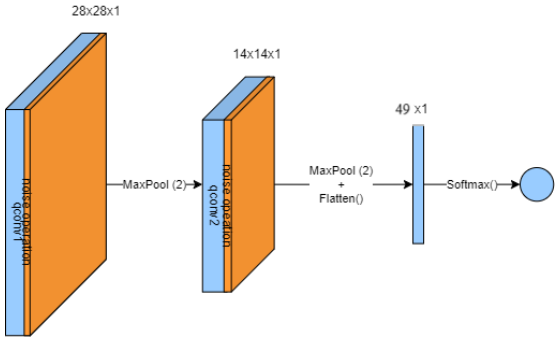


Figure 5.3: Quantum neural network used for the performance analysis in the presence of amplitude damping noise. The noise operation is applied at the end of the quantum circuit corresponding to the quantum convolutional layer

As show in figure (5.2a) the performance are indeed improved by adding more reuploading layers, contributing to proving the validity of the reuploading scheme. In figure (5.2b) is shown how even the number of qubits in the quantum circuit plays a central role in the performance of the model as a multiqubit and entangled quantum circuit has a richer expressibility in terms of functions it can fit [20].

	Acc VS reuploading layers	Acc VS qubits	Acc VS noise
Learning rate	10^{-3}	10^{-3}	10^{-3}
Reuploading layers	1-10	1	10
Qubits	1	1-10	10
λ	0	0	0.0-0.4

Table 5.1: Hyperparameters used for the tests, see chapter 3 for definition of learning rate

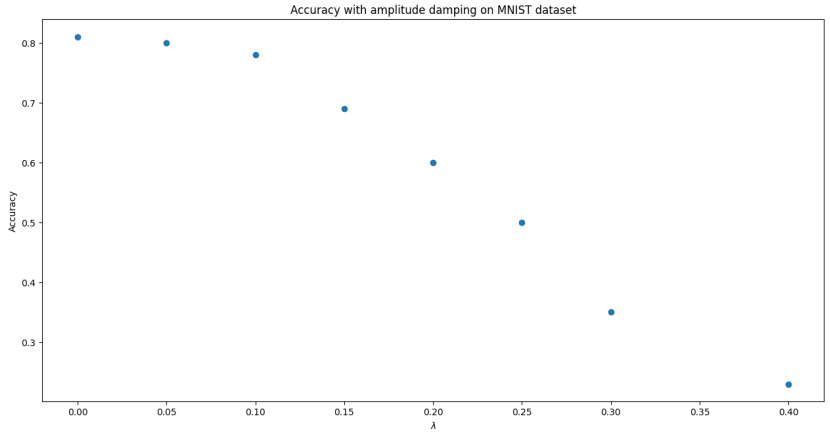


Figure 5.4: Accuracy of the data reuploading image classifier on MNIST datatest with the presence of various degree of amplitude damping noise λ

Figure (5.4) shows how amplitude damping noise actually presents a serious threat to the reach of quantum advantage: starting from a value of $\lambda = 0.1$ the performance are drastically decreased due to the impossibility of discriminating quantum states, hence almost random accuracy for value beyond 0.2-0.3. At the current state of quantum devices this value represent an obstacle for extremely deep and entangled quantum circuits (i.e. the ones that could bring a distinct quantum advantage in the future) but not so much for hybrid shallow circuits such as the one used for this benchmark, as can be seen in the next section.

The details of the architecture used for the tests are reported in table (5.1)

5.1.2 Results

After verifying the capabilities of the algorithm, a bigger and fine tuned quantum neural network has been trained to classify different benchmark datasets and compared to a classical CNN with a similar number of learnable parameters.

Globally the quantum CNN with the data reuploading ansatz presents an improvement upon the classical one: in every datasets the learning process is

faster and the asymptotic accuracy is higher. Moreover at test time the quantum neural network has achieved higher accuracy than the classical one.

As said above, the tests are done with four different datasets in order to asses the performances in different settings: MNIST, Fashion MNIST, CIFAR10 and brain pet images (CDOPA). For each datasets the quantum neural network is trained first with an ideal simulator, than with a noisy simulator with bit and phase flips errors with rate 0.1, typically higher than the usual single gate error rate in real devices and finally on a real quantum computer, in particular the 11 qubits Google's IonQ Harmony, with a single gate error rate of ≈ 0.05.

From each dataset a random subset of datapoints ($\approx$15-20% depending on the dataset) has been removed and used as a test set in order to validate the models. Each test accuracy is reported in the corresponding table and for each dataset the training-validation loss and accuracy curves are reported as figures.

MNIST dataset

The MNIST dataset is composed of 60000 28x28 greyscale images of digits from 0 to 9 (10 classes), as illustrated in figure (5.5). The size of the test set used is 10000 images.

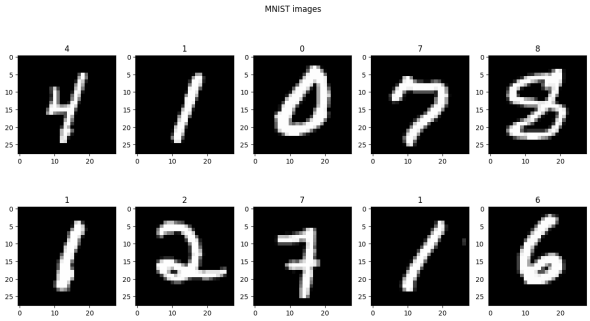


Figure 5.5: Examples of images in MNIST dataset

On this dataset the quantum neural network has surpassed the classical one in terms of both accuracy and convergence time: as reported in figures (5.6, 5.7) the ideal simulator reports a stable training dynamic and the one with the bit and phase flip added also, although a bit more rugged due to the random errors.

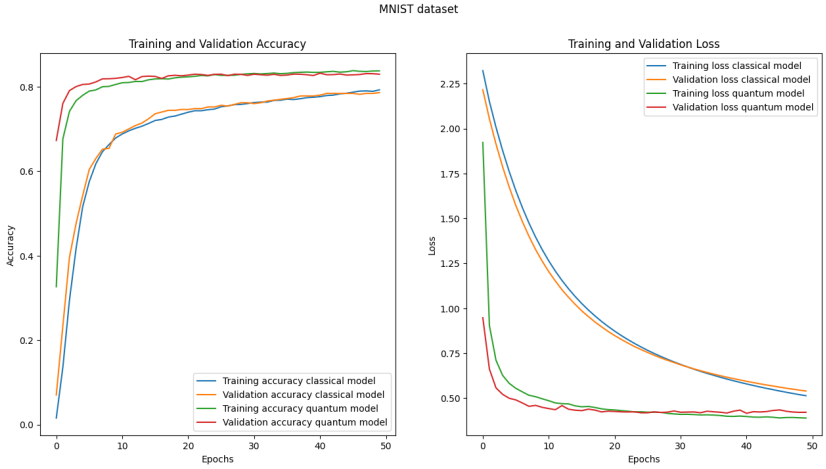


Figure 5.6: Quantum classifier on MNIST dataset with ideal simulator confronted with classical classifier

Probably due to the (relatively small) size of the images, the quantum neural

network trained on the real device (figure (5.8)) is very similar to the one trained on the ideal simulator. This points out the actual usability of NISQ devices for proof of concept in practical applications.

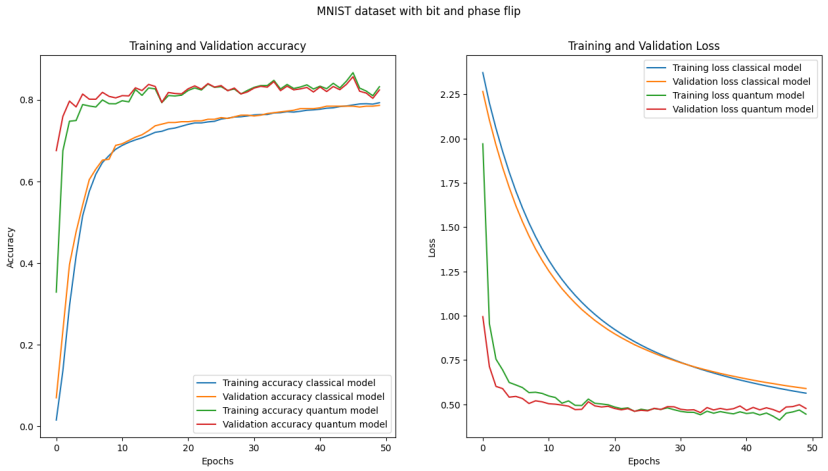


Figure 5.7: Quantum classifier on MNIST dataset with bit and phase flip errors confronted with classical classifier

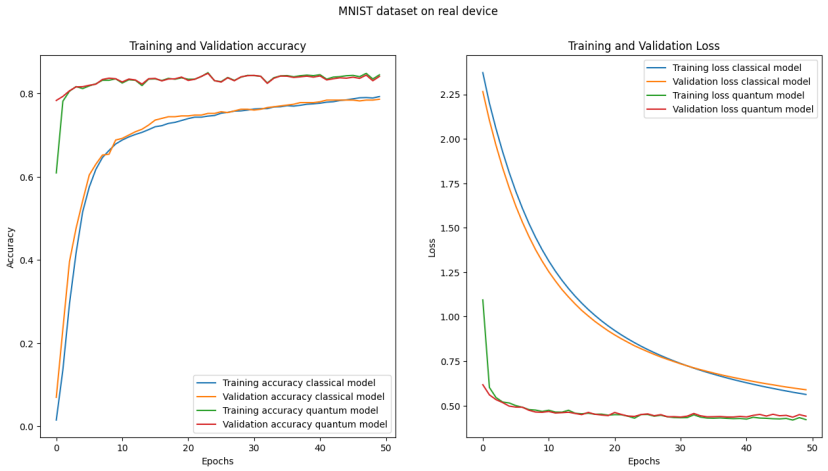


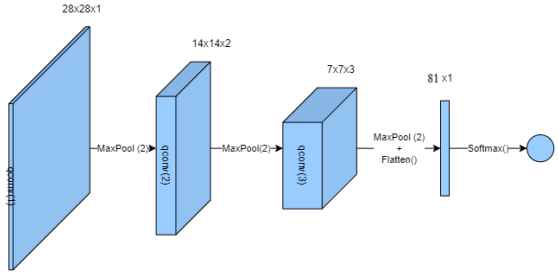
Figure 5.8: Quantum classifier on MNIST datase trained on a real quantum device confronted with classical classifier

The details of the neural networks used such as number of trainable parameters, hyperparameters for the QCNN, error parameter for the noisy simulator,

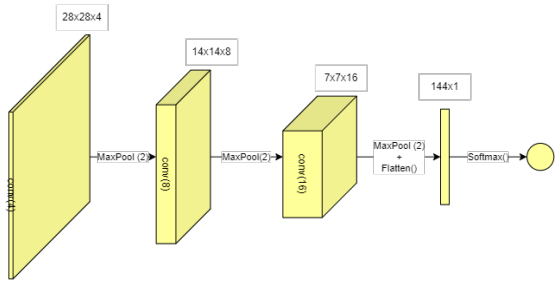
learning rate and the test results are reported in table (5.2) while an illustration of the models is reported in figure (5.9).

	Parameters	lr	Qubits	Reup. layers	Flip param.	Test acc.
CNN	≈2900	10^{-3}	-	-	-	0.76
QCNN (ideal)	≈2800	10^{-3}	5	5	-	0.82
QCNN (noisy)	≈2800	10^{-3}	5	5	0.1	0.80
QCNN (real)	≈2800	10^{-3}	5	5	-	0.81

Table 5.2: Details of the comparison between the classical CNN and the quantum QCNN for MNIST dataset



(a) QCNN model used for MNIST dataset



(b) CNN model used for MNIST dataset

Figure 5.9: Comparison between the quantum and classical models used for MNIST dataset

Both models presents 3 convolutional layers but while the classical CNN presents 4,8 and then 16 filters, the quantum model used a series of only 1,2 and 3 filters. The rest of the models are equal: 2x2 maxpool and final softmax activation function with 10 (number of classes) outputs.

Fashion MNIST dataset

The Fashion MNIST dataset is composed of 60000 28x28 greyscale images of clothing items (t-shirts, boots, bags, pants for a total of 10 classes) as illustrated in figure (5.10). The size of the test set used is 10000 images.

Due to the similarity between this dataset and the MNIST, the results are analogous to the preceding section as one can see from the graphs in figures (5.11, 5.12, 5.13).

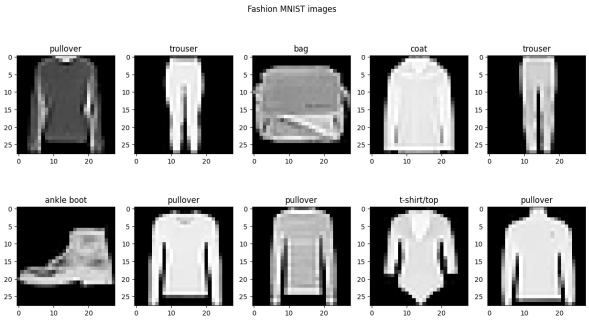


Figure 5.10: Examples of images in Fashion MNIST dataset

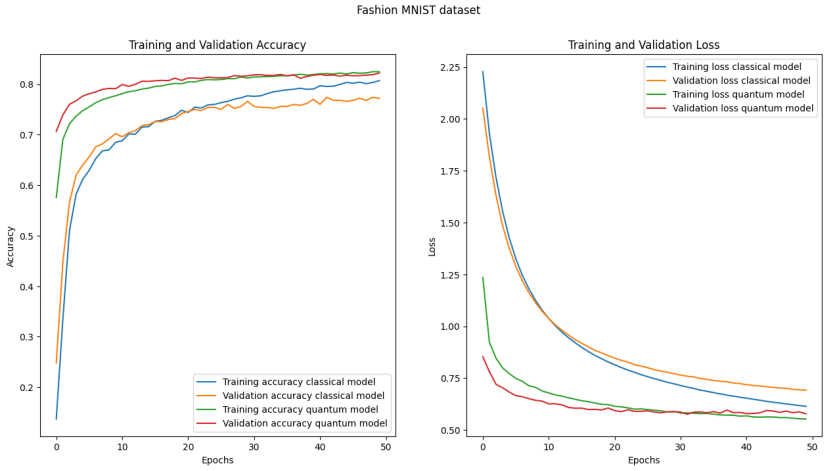


Figure 5.11: Quantum classifier on Fashion MNIST dataset with ideal simulator confronted with classical classifier

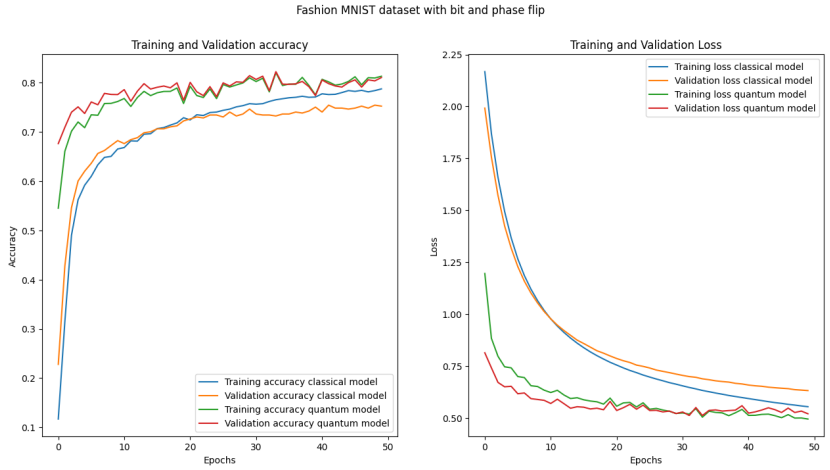


Figure 5.12: Quantum classifier on Fashion MNIST dataset with bit and phase flip errors confronted with classical classifier

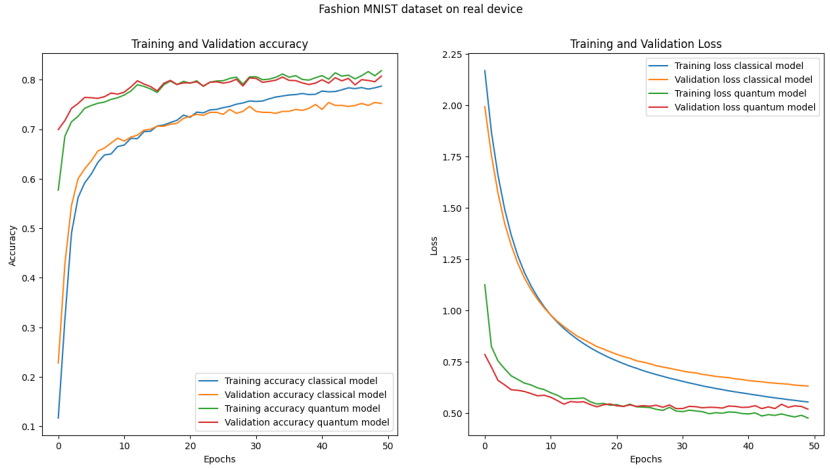
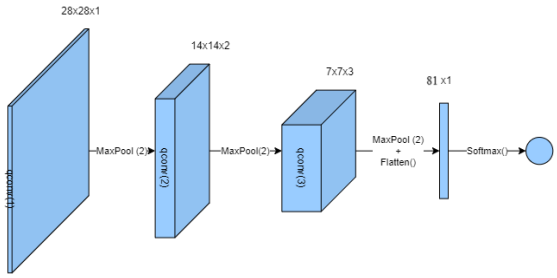


Figure 5.13: Quantum classifier on Fashion MNIST dataset trained on a real quantum device confronted with classical classifier

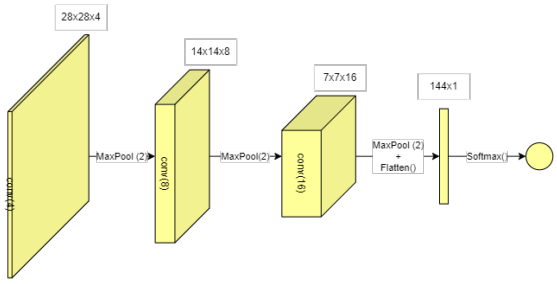
For this dataset the model used is the same as the one used for the MNIST as reported in table (5.3) and figure (5.14).

	Parameters	lr	Qubits	Reup. layers	Flip param.	Test acc.
CNN	≈2900	10^{-3}	-	-	-	0.74
QCNN (ideal)	≈2800	10^{-3}	5	5	-	0.80
QCNN (noisy)	≈2800	10^{-3}	5	5	0.1	0.79
QCNN (real)	≈2800	10^{-3}	5	5	-	0.80

Table 5.3: Details of the comparison between the classical CNN and the quantum QCNN for the Fashion MNIST dataset



(a) QCNN model used for Fashion MNIST dataset



(b) CNN model used for Fashion MNIST dataset

Figure 5.14: Comparison between the quantum and classical models used for Fashion MNIST dataset

Both models presents 3 convolutional layers but while the classical CNN presents 4,8 and then 16 filters, the quantum model used a series of only 1,2 and 3 filters. The rest of the models are equal: 2x2 maxpool and final softmax activation function with 10 (number of classes) outputs, as with the models used for the MNIST dataset.

CIFAR10

CIFAR10 dataset is composed of 60000 32x32 RGB images depicting different kinds of objects and animals (e.g. birds, trucks ,cars, aiplanes, horses for a total of 10 classes), as shown in figure (5.15). The size of the test set used is 10000 images

Due to the highly dimensionality of the data (the images are not only bigger than MNIST or Fshion MNIST but presents 3 times the pixels due to the 3 RGB channels) in this case the training is harder and slower (figure (5.16)) but nonetheless the fully quantum neural networks still presents advantages over the CNN: the loss descent is steeper and the overall accuracy remains higher.

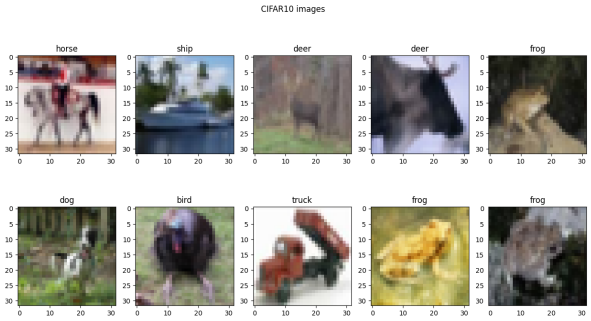


Figure 5.15: Examples of images in CIFAR10 dataset

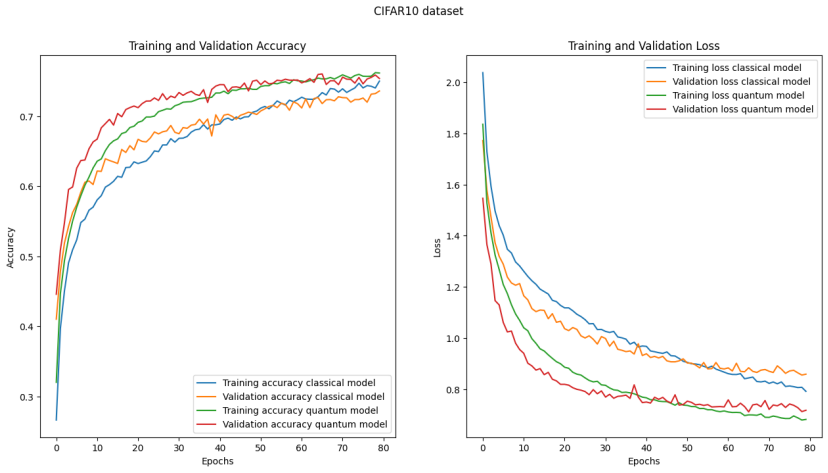


Figure 5.16: Quantum classifier on CIFAR10 dataset with ideal simulator confronted with classical classifier

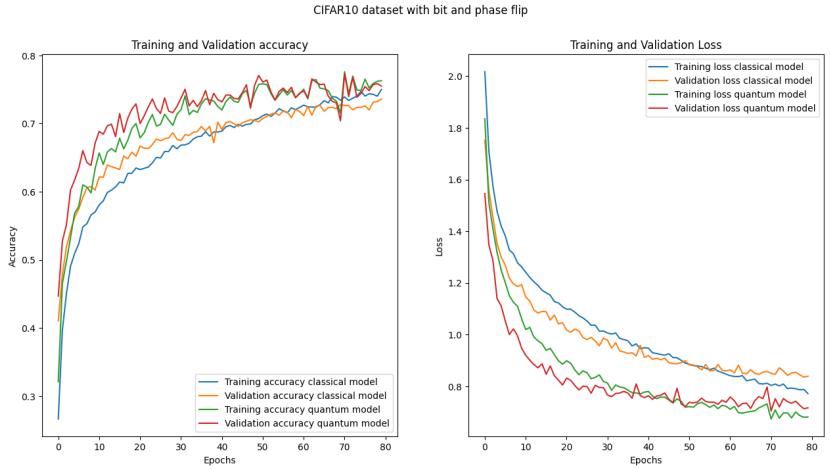


Figure 5.17: Quantum classifier on CIFAR10 dataset with bit and phase flip errors confronted with classical classifier

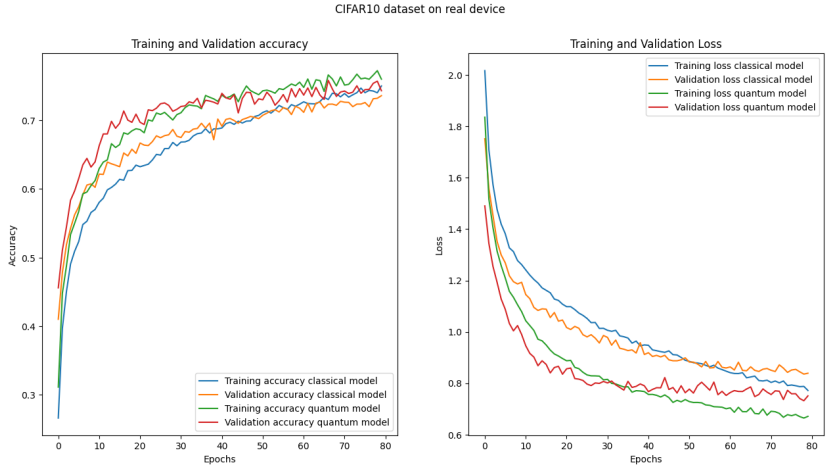


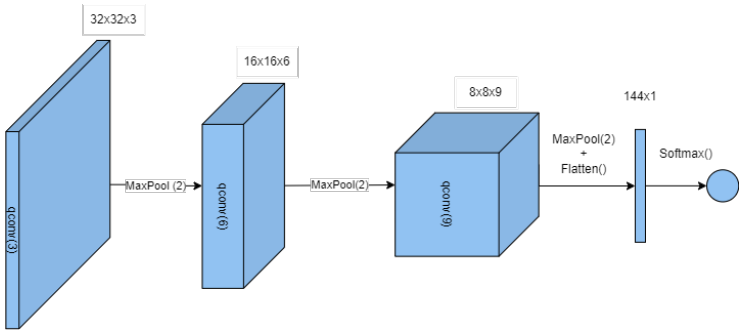
Figure 5.18: Quantum classifier on CIFAR10 dataset trained on a real quantum device confronted with classical classifier

Also in this case the architecture has been validated in three ways: using an ideal simulator (5.16), a noisy simulator with bit and phase flips (5.17) and on a real device (5.18), with the results reported in table (5.4).

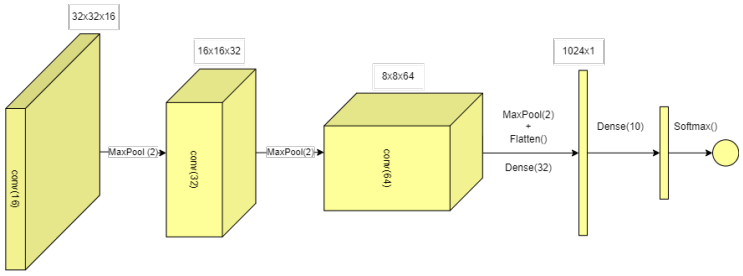
	Parameters	lr	Qubits	Reup. layers	Flip param.	Test acc.
CNN	≈55000	$5x10^{-4}$	-	-	-	0.71
QCNN (ideal)	≈13000	$5x10^{-4}$	8	5	-	0.78
QCNN (noisy)	≈13000	$5x10^{-4}$	8	5	0.1	0.73
QCNN (real)	≈13000	$5x10^{-4}$	8	5	-	0.77

Table 5.4: Details of the comparison between the classical CNN and the quantum QCNN for the CIFAR10

Although more susceptible to flip noise (figure (5.17)) overall the QCNN used still presents advantages on the classical one, especially comparing the number of trainable parameters. The two neural networks are illustrated in figure (5.19).



(a) QCNN model used for the CIFAR10 dataset



(b) CNN model used for the CIFAR10 dataset

Figure 5.19: Comparison between the quantum and classical models used for the CIFAR10 dataset

The CNN presents 3 convolutional layers with respectively 16, 32 and 64 convolutional filters. After the flattening of the images the data go through an additional fully connected layer with 32 nodes and then is classified by the softmax activation function after passing through the usual linear layer with 10 (as 10 are the classes) output nodes.

On the other hand, the QCNN presents only 3,6 and 9 convolutional filters and only one final fully connected layer before the softmax fucntion.

CDOPA

As a real-world application, the diagnosis of parkinson disease through binary classification of PET brain images has been chosen. The dataset is composed of 98 110x110 greyscale 3D PET images, each of which formed by 10 slices (figure (5.20)) and belongs to a single patient. The size of the test set used is 20 images.

The images represents negative and positive patients (binary classification) distinguished by the reaction of the marker with the base ganglia (the green C-shaped part in figure (5.20)): a well define base ganglia imply a negative ganglia while a deformed one indicates the presence of parkinson disease.

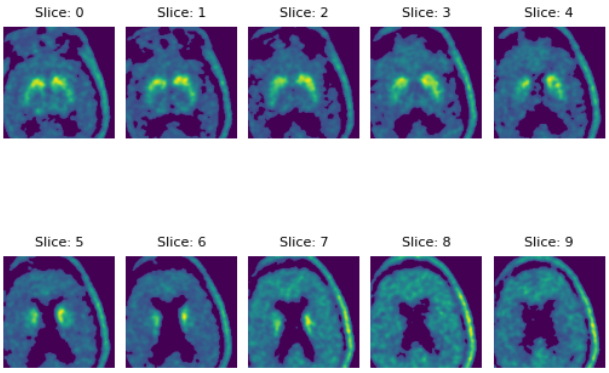


Figure 5.20: Examples of images in CDOPA dataset. The images belongs to a single positive patient and represent the 3D slices of the PET scan

Due to the small number of datapoints the dataset has been augmented operating small translations along the x and y axes in order to have a dataset four times larger (plus the original one, so five times the data) for the training. In order to reduce the slices, only the 5 most central one has been selected: as visible in figure (5.20) not every slice contains a significant portion of marker.

In this case, due to the very high dimensionality of the data (110x110x5 = 60500 numbers), an hybrid approach has been used: in the QCNN the first layer are classical tensorflow convolutional layers and maxpooling. Once the dimension of the images reach a value tractable for current simulators and quantum devices a quantum convolutional layer is inserted instead of a normal one as in the classical CNN.

Due to the peculiar nature of the dataset, the so called *recall* has been chosen as a metric. The recall is defined as

$$\text{Recall} \equiv \frac{\text{tp}}{\text{tp} + \text{fn}} \tag{5.1}$$

with tp and fn the rate of *true positive* (a positive image has correctly been classified as positive) and *false negative* (a positive image has wrongly been classified as negative).

In cases where the classes are unbalanced (like this where the number of negative images is greater than the number of posite ones) the accuracy fails to be an expressive metric (e.g. if only 5% of the data belongs to class 1, the accuracy of the model could be 95% even if it misclassify all the class 1 datapoints) and in particular when a false positive output has not the same importance of a false negative.

In medical imaging a false positive result leads to a second round of checks while a false negative diagnosis would mean not being aware of an actual illness. The recall metric has the advantage to give more weight to false negatives, that indeed lower the value.

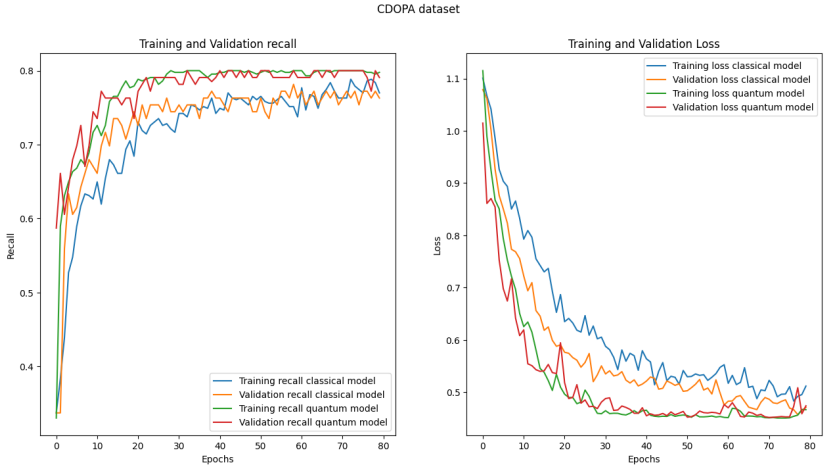


Figure 5.21: Quantum classifier on CDOPA dataset with ideal simulator confronted with classical classifier

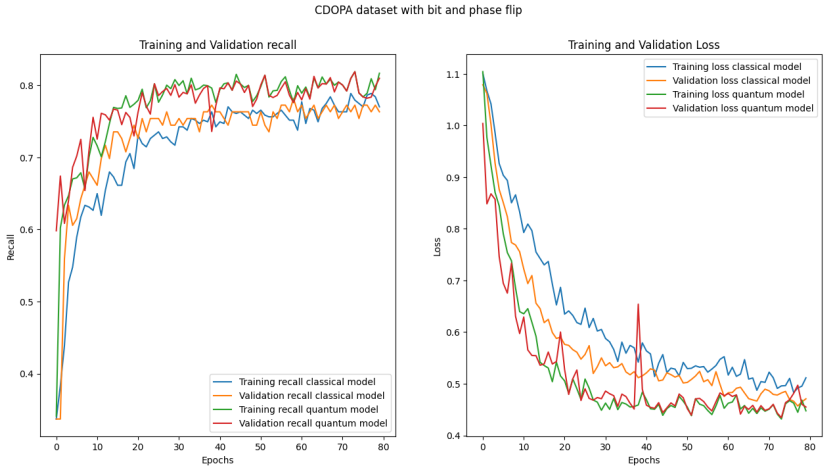


Figure 5.22: Quantum classifier on CDOPA dataset with bit and phase flip errors confronted with classical classifier

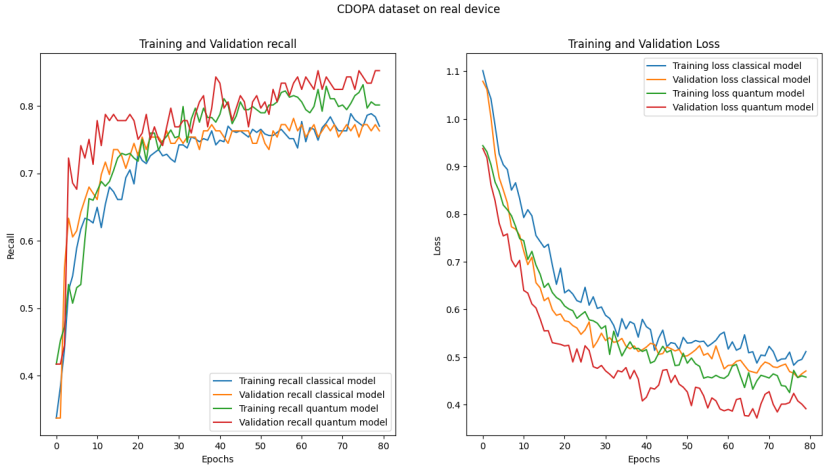


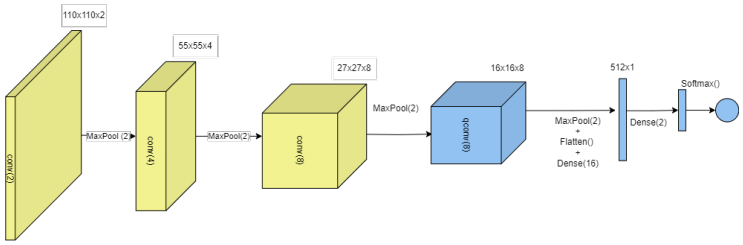
Figure 5.23: Quantum classifier on CDOPA dataset trained on a real quantum device confronted with classical classifier

As before the details of the training with ideal simulator (figure (5.21)), noisy simulator (figure (5.22)) and real device (figure (5.23)) are reported in table (5.5) and an illustration of the models in figure (5.24).

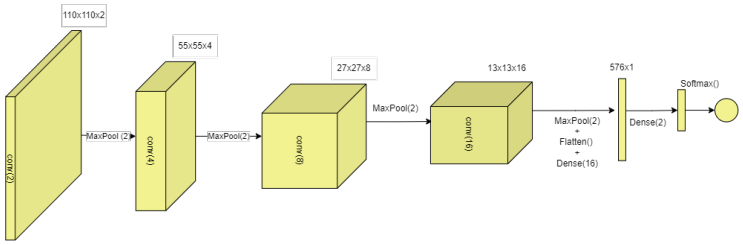
	Parameters	lr	Qubits	Reup. layers	Flip param.	Test recall
CNN	≈18000	510^{-4}	-	-	-	0.74
QCNN (ideal)	≈15000	10^{-4}	8	5	-	0.79
QCNN (noisy)	≈15000	10^{-4}	8	5	0.1	0.78
QCNN (real)	≈15000	10^{-4}	8	5	-	0.75

Table 5.5: Details of the comparison between the classical CNN and the quantum QCNN for the CDOPA dataset

Due to the fact that in this model we have only the last parts as quantum layers (the blue ones in figure (5.24)) the results are more similar between the two models but nonetheless the hybrid quantum-classical model has achieved an higher test recall and overall better loss decay, even for the real device case although with a more irregular dynamic.



(a) Hybrid QCNN model used for the CDOPA dataset



(b) CNN model used for the CDOPA dataset

Figure 5.24: Comparison between the quantum and classical models used for the CDOPA dataset

In the first part of both model there are 3 convolutional layers with respectively 2, 4 and 8 convolutional filters. After that we have a final convolutional layer of still 8 filters for the quantum model and 16 filters for the classical one. After that the usual dense layers with final softmax activation function are present.

5.2 Image generation

As a second computer vision problem to tackle with quantum machine learning the task of image generation has been chosen. In the following parts the results obtained through the implementation of the PATCH GAN algorithm are reported: also in this part the three most famous computer vision benchmark dataset are used, namely MNIST, Fashion MNIST and CIFAR10. In detail, the full discriminator-generator framework is set as follow: a fully-connected type classical discriminator is paired with both the classical and quantum generator models. The classical generator model is composed of a usual conditional convolutional generative network while the quantum patch architecture has been modified from the original article to enable the conditioning on the classes labels in order to be able to generate images belonging to a certain particular class at will.

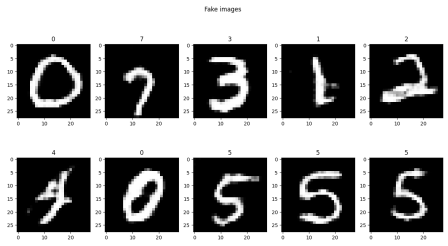
Due to being a more qualitatively task, (on of the best metric in order to judge how well an image has been forged is indeed the human eye) as a demonstration of the quantum model's effectiveness a comparison between the original images, images generated through a classical GAN and images generated through the quantum GAN will be reported. In addition also the training dynamic will be plotted (as from the discriminator's loss value the goodness of the process can be inferred) and the characteristics of the models reported in tables.

Similarly with the quantum classifier, the quantum generative model has been developed as fully integrable within `pytorch` [42] deep learning framework as Python classes written with the aid of the `pennylane` library [6].

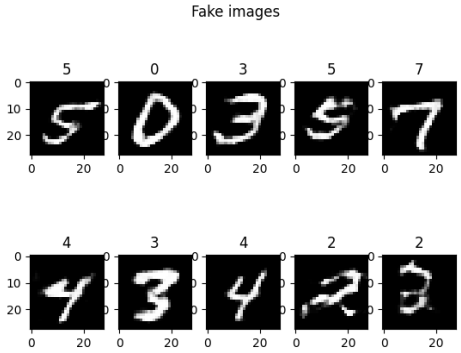
Results

For the three datasets the patch gan has successfully reproduced the images as good as a normal classical generative models (figures 5.25, 5.27, 5.29).

As one can see the quantum generative model is capable of reproducing both greyscale and rgb images as good as a comparable classical model and without anomalies in the training (figures 5.26, 5.28, 5.30): the discriminator loss approach 0.5 (blue curves) i.e. it just guesses randomly if an image is generated or not while the quantum generator loss decreases with the training steps.



(a) MNIST images generated by quantum GAN



(b) MNIST images generated by classic GAN

Figure 5.25: Comparison between MNIST images generated by quantum (above) and classical (below) GANs

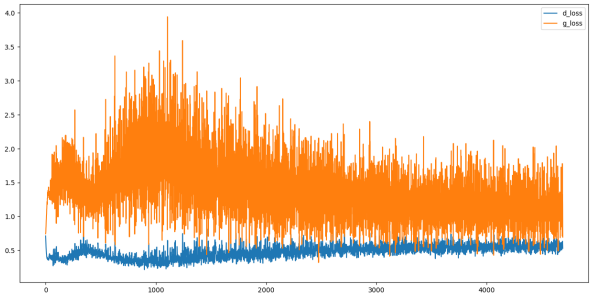
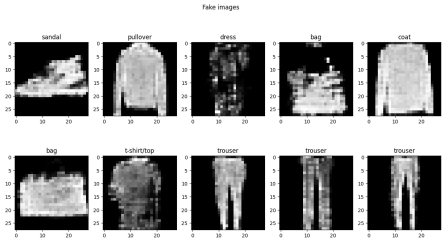
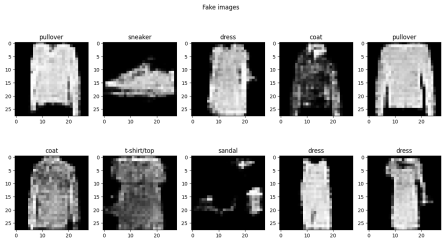


Figure 5.26: Loss vs iteration steps of discriminator (blue) vs quantum generative (orange) model for MNIST dataset



(a) Fashion MNIST images generated by quantum GAN



(b) Fashion MNIST images generated by classic GAN

Figure 5.27: Comparison between Fashion MNIST images generated by quantum (above) and classical (below) GANs

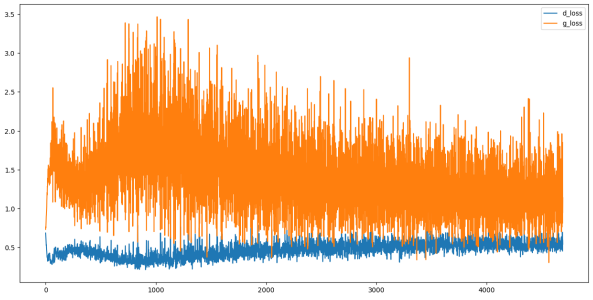
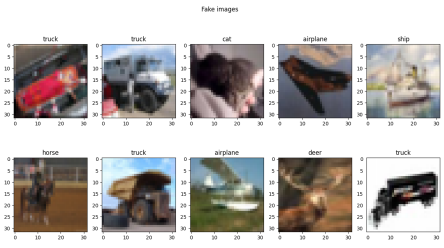
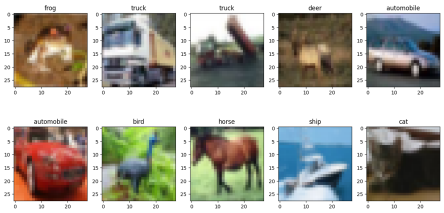


Figure 5.28: Loss vs iteration steps of discriminator vs quantum generative model for Fashion MNIST dataset



(a) CIFAR10 images generated by quantum GAN



(b) CIFAR10 images generated by classic GAN

Figure 5.29: Comparison between CIFAR10 images generated by quantum (above) and classical (below) GANs

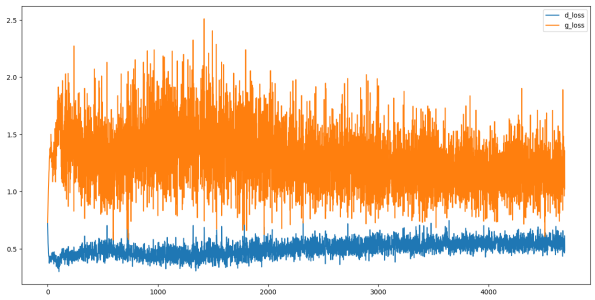


Figure 5.30: Loss vs iteration steps of discriminator vs quantum generative model for CIFAR10 dataset

All the trainings have been done on the pennylane's ideal simulators using the following parameters:

- Total qubits: 12
- Ancillary qubits: 1
- Number of sub-generators: 14
- Depth of quantum circuit: 24

The classical discriminator is implemented as a MLP with 2 hidden layers of 64 and 32 nodes and the learning rate for both the generator's and discriminator's optimizers have been set to 2x10−4.

The classical generator used for a comparison is a standard convolutional generator with 2 transposed convolution layers that starting from a latent space of dimension 50 outputs the image, with a total of ≈ 10000 parameters while for the quantum model, due to the smal number of qubits needed the total trainable parameters are ≈ 4500 counting the 14 different sub-generator's circuits (each of which is a different circuit), hence showing again comparable performance with a smaller amount of training parameters.

5.3 Conclusions

The results lead to the conclusion that quantum computation could be used to enhance machine learning, even in tasks outside physical research: the image classifier has reached performance comparable if not better to other similar classical architectures and the image generator has accurately reproduced the most used benchmark datasets for image generation. Due to the simple and hybrid nature of the variational quantum circuits used, bit and phase flip-like noise did not pose a serious threat (at least when present in small amounts). Amplitude damping errors are instead detrimental to quantum algorithms.

Due to the small scale and error-prone nature of current quantum device, hybrid algorithms that gets the best of both quantum and classical computation, seem to be an interesting path to follow in order to reach quantum advantage. The main areas in which one could pursue this research could be identifying better embedding ansatzes as it is clear from literature how this influences the possibility of quantum advantage.

Secondly, making quantum computers more and more available is crucial to asses the functioning of algorithms: as the circuit grows, simulators become exponentially slower making impossible to run even the simplest algorithms. The possibility of running quantum algorithms on real quantum computers can highlight their actual performances, avoiding the usage simulated noise. A big help could also come from the computer science side: as training a neural networks requires running the same architecture several times, a quantum circuit has to be executed a lot of times in order to be trained. Libraries that automate the multiple executions of the quantum neural network on the same quantum device are direly needed.

In the end the search for a real quantum advantage could have to go through implementing the quantum neural network completely as a quantum algorithm: NISQ devices require a classical optimization routine and lack the possibility of parallelizing the computations of the data in a big quantum circuit and of exploiting quantum superposition and entanglement to the fullest.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. 2016. arXiv: 1603.04467 [cs.DC].

[2] Amira Abbas et al. "The power of quantum neural networks". In: *Nature Computational Science* 1.6 (June 2021), pp. 403–409. DOI: 10.1038/s43588-021-00084-1. URL: https://doi.org/10.1038%2Fs43588-021-00084-1.

[3] Marcello Benedetti, John Realpe-Gó mez, and Alejandro Perdomo-Ortiz. "Quantum-assisted Helmholtz machines: A quantum–classical deep learning framework for industrial datasets in near-term devices". In: *Quantum Science and Technology* 3.3 (May 2018), p. 034007. DOI: 10.1088/2058-9565/aabd98. URL: https://doi.org/10.1088%2F2058-9565%2Faabd98.

[4] Marcello Benedetti et al. "Quantum-Assisted Learning of Hardware-Embedded Probabilistic Graphical Models". In: *Physical Review X* 7.4 (Nov. 2017). DOI: 10.1103/physrevx.7.041052. URL: https://doi.org/10.1103%2Fphysrevx.7.041052.

[5] Oksana Berezniuk et al. *A scale-dependent notion of effective dimension*. 2020. arXiv: 2001.10872 [stat.ML].

[6] Ville Bergholm et al. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2022. arXiv: 1811.04968 [quant-ph].

[7] Michael Broughton et al. *TensorFlow Quantum: A Software Framework for Quantum Machine Learning*. 2021. arXiv: 2003.02989 [quant-ph].

[8] M Cerezo and Patrick J Coles. "Higher order derivatives of quantum neural networks with barren plateaus". In: *Quantum Science and Technology* 6.3 (June 2021), p. 035006. DOI: 10.1088/2058-9565/abf51a. URL: https://doi.org/10.1088%2F2058-9565%2Fabf51a.

[9] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. "Quantum convolutional neural networks". In: *Nature Physics* 15.12 (Aug. 2019), pp. 1273–1278. DOI: 10.1038/s41567-019-0648-8. URL: https://doi.org/10.1038%2Fs41567-019-0648-8.

[10] Gavin E. Crooks. *Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition*. 2019. arXiv: 1905.13311 [quant-ph].

[11] Pierre-Luc Dallaire-Demers and Nathan Killoran. "Quantum generative adversarial networks". In: *Phys. Rev. A* 98 (1 July 2018), p. 012324. DOI: 10.1103/PhysRevA.98.012324. URL: https://link.aps.org/doi/10.1103/PhysRevA.98.012324.

[12] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. "Quantum-Enhanced Machine Learning". In: *Physical Review Letters* 117.13 (Sept. 2016). DOI: 10.1103/physrevlett.117.130501. URL: https://doi.org/10.1103%2Fphysrevlett.117.130501.

[13] Philip Easom-Mccaldin et al. "On Depth, Robustness and Performance Using the Data Re-Uploading Single-Qubit Classifier". In: *IEEE Access* 9 (2021), pp. 65127–65139. DOI: 10.1109/ACCESS.2021.3075492.

[14] Edward Farhi and Hartmut Neven. *Classification with Quantum Neural Networks on Near Term Processors*. 2018. arXiv: 1802.06002 [quant-ph].

[15] Enrico Fontana et al. "Evaluating the noise resilience of variational quantum algorithms". In: *Physical Review A* 104.2 (Aug. 2021). DOI: 10.1103/physreva.104.022403. URL: https://doi.org/10.1103%2Fphysreva.104.022403.

[16] Xun Gao, Zhengyu Zhang, and Luming Duan. *An efficient quantum algorithm for generative machine learning*. 2017. arXiv: 1711.02038 [quant-ph].

[17] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[18] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[19] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum Algorithm for Linear Systems of Equations". In: *Physical Review Letters* 103.15 (Oct. 2009). DOI: 10.1103/physrevlett.103.150502. URL: https://doi.org/10.1103%2Fphysrevlett.103.150502.

[20] Vojtěch Havlíček et al. "Supervised learning with quantum-enhanced feature spaces". In: *Nature* 567.7747 (Mar. 2019), pp. 209–212. DOI: 10.1038/s41586-019-0980-2. URL: https://doi.org/10.1038%2Fs41586-019-0980-2.

[21] Maxwell Henderson et al. *Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits*. 2019. arXiv: 1904.04767 [quant-ph].

[22] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T. URL: https://www.sciencedirect.com/science/article/pii/089360809190009T.

[23] He-Liang Huang et al. "Experimental Quantum Generative Adversarial Networks for Image Generation". In: *Physical Review Applied* 16.2 (Aug. 2021). DOI: 10.1103/physrevapplied.16.024051. URL: https://doi.org/10.1103%2Fphysrevapplied.16.024051.

[24] Tak Hur, Leeseok Kim, and Daniel K. Park. "Quantum convolutional neural network for classical data classification". In: *Quantum Machine Intelligence* 4.1 (Feb. 2022). DOI: 10.1007/s42484-021-00061-x. URL: https://doi.org/10.1007%2Fs42484-021-00061-x.

[25] Sergei V. Isakov et al. *Simulations of Quantum Circuits with Approximate Noise using qsim and Cirq*. 2021. arXiv: 2111.02396 [quant-ph].

[26] Younes Javanmard et al. *Quantum simulation of dynamical phase transitions in noisy quantum devices*. 2022. arXiv: 2211.08318 [quant-ph].

[27] Chi Jin et al. *On Nonconvex Optimization for Machine Learning: Gradients, Stochasticity, and Saddle Points*. 2019. arXiv: 1902.04811 [cs.LG].

[28] Ivan Kassal et al. "Simulating Chemistry Using Quantum Computers". In: *Annual Review of Physical Chemistry* 62.1 (May 2011), pp. 185–207. DOI: 10.1146/annurev-physchem-032210-103512. URL: https://doi.org/10.1146%2Fannurev-physchem-032210-103512.

[29] Frederik Kunstner, Lukas Balles, and Philipp Hennig. *Limitations of the Empirical Fisher Approximation for Natural Gradient Descent*. 2020. arXiv: 1905.12558 [cs.LG].

[30] Ryan LaRose and Brian Coyle. "Robust data encodings for quantum classifiers". In: *Physical Review A* 102.3 (Sept. 2020). DOI: 10.1103/physreva.102.032420. URL: https://doi.org/10.1103%2Fphysreva.102.032420.

[31] Yann A. LeCun et al. "Efficient BackProp". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_3. URL: https://doi.org/10.1007/978-3-642-35289-8_3.

[32] Ying Li and Simon C. Benjamin. "Efficient Variational Quantum Simulator Incorporating Active Error Minimization". In: *Physical Review X* 7.2 (June 2017). DOI: 10.1103/physrevx.7.021050. URL: https://doi.org/10.1103%2Fphysrevx.7.021050.

[33] Junyu Liu et al. *Noise can be helpful for variational quantum algorithms*. 2022. arXiv: 2210.06723 [quant-ph].

[34] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. *Quantum algorithms for supervised and unsupervised machine learning*. 2013. arXiv: 1307.0411 [quant-ph].

[35] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. "Quantum principal component analysis". In: *Nature Physics* 10.9 (July 2014), pp. 631–633. DOI: 10.1038/nphys3029. URL: https://doi.org/10.1038%2Fnphys3029.

[36] Seth Lloyd and Christian Weedbrook. "Quantum Generative Adversarial Learning". In: *Physical Review Letters* 121.4 (July 2018). DOI: 10.1103/physrevlett.121.040502. URL: https://doi.org/10.1103%2Fphysrevlett.121.040502.

[37] Jarrod R McClean et al. “The theory of variational hybrid quantum-classical algorithms”. In: *New Journal of Physics* 18.2 (Feb. 2016), p. 023023. DOI: 10.1088/1367-2630/18/2/023023. URL: https://doi.org/10.1088%2F1367-2630%2F18%2F2%2F023023.

[38] Jarrod R. McClean et al. “Hybrid quantum-classical hierarchy for mitigation of decoherence and determination of excited states”. In: *Physical Review A* 95.4 (Apr. 2017). DOI: 10.1103/physreva.95.042308. URL: https://doi.org/10.1103%2Fphysreva.95.042308.

[39] K. Mitarai et al. “Quantum circuit learning”. In: *Physical Review A* 98.3 (Sept. 2018). DOI: 10.1103/physreva.98.032309. URL: https://doi.org/10.1103%2Fphysreva.98.032309.

[40] Ken M. Nakanishi, Takahiko Satoh, and Synge Todo. *Quantum-gate decomposer*. 2021. arXiv: 2109.13223 [quant-ph].

[41] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[42] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[43] Adriá n Pérez-Salinas et al. “Data re-uploading for a universal quantum classifier”. In: *Quantum* 4 (Feb. 2020), p. 226. DOI: 10.22331/q-2020-02-06-226. URL: https://doi.org/10.22331%2Fq-2020-02-06-226.

[44] Arnoldo Piccardo and Roberto Cappuccio. *The role of the deep convolutional neural network as an aid to interpreting brain [18F]DOPA PET/CT in the diagnosis of Parkinson's disease*. 2021.

[45] John Preskill. “Quantum Computing in the NISQ era and beyond”. In: *Quantum* 2 (Aug. 2018), p. 79. DOI: 10.22331/q-2018-08-06-79. URL: https://doi.org/10.22331%2Fq-2018-08-06-79.

[46] Pé ter Rakyta and Zoltán Zimborás. “Approaching the theoretical limit in quantum gate decomposition”. In: *Quantum* 6 (May 2022), p. 710. DOI: 10.22331/q-2022-05-11-710. URL: https://doi.org/10.22331%2Fq-2022-05-11-710.

[47] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. “Quantum Support Vector Machine for Big Data Classification”. In: *Physical Review Letters* 113.13 (Sept. 2014). DOI: 10.1103/physrevlett.113.130503. URL: https://doi.org/10.1103%2Fphysrevlett.113.130503.

[48] Jonathan Romero and Alan Aspuru-Guzik. *Variational quantum generators: Generative adversarial quantum machine learning for continuous distributions*. 2019. arXiv: 1901.00848 [quant-ph].

[49] Jonathan Romero, Jonathan P Olson, and Alan Aspuru-Guzik. “Quantum autoencoders for efficient compression of quantum data”. In: *Quantum Science and Technology* 2.4 (Aug. 2017), p. 045001. DOI: 10.1088/2058-9565/aa8072. URL: https://doi.org/10.1088%2F2058-9565%2Faa8072.

[50] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519. URL: http://dx.doi.org/10.1037/h0042519.

[51] Philipp Schleich et al. *Partitioning Quantum Chemistry Simulations with Clifford Circuits*. 2023. arXiv: 2303.01221 [quant-ph].

[52] Maria Schuld et al. "Evaluating analytic gradients on quantum hardware". In: *Physical Review A* 99.3 (Mar. 2019). DOI: 10.1103/physreva.99.032331. URL: https://doi.org/10.1103%2Fphysreva.99.032331.

[53] Samuel A. Stein et al. *QuCNN : A Quantum Convolutional Neural Network with Entanglement Based Backpropagation*. 2022. arXiv: 2210.05443 [quant-ph].

[54] Guoming Wang. "Quantum algorithm for linear regression". In: *Physical Review A* 96.1 (July 2017). DOI: 10.1103/physreva.96.012335. URL: https://doi.org/10.1103%2Fphysreva.96.012335.