A first Approach to Quantum Machine Learning for Computer Vision

Juan Carlos Martínez Moreno

Resum– Aquest projecte s'enfoca en l'estudi d'algoritmes de *Machine Learning*, per a Visió per Computador, en la seva versió quàntica. Per tant, es busca aconseguir crear i executar models de *Machine Learning* en simuladors quantics, evaluar els resultats obtinguts mitjançant metriques comuns, i poder estudiar la diferencia entre la computació clàssica i la computació quàntica, que son dos paradigmes de computació molt distants, i veure quins avantatges i quins inconvenients es poden trobar en cadascú. Aquest projecte te un alt contingut relacionat amb la fisica quàntica per donar context i tenir les bases per comprendre mes endavant els algoritmes implementats en circuits quàntics, a més de coneixements de *Machine Learning* en computació clàssica, ja que molts dels algoritmes quàntics tenen la seva part analoga clàssica, i l'objectiu principal es comparar aquestes versions.

Paraules clau– aprenentatge computacional quàntic, visió per computador, computació quàntica, circuits quàntics, portes quàntiques, algoritmes quàntics, qiskit, intel·ligència artificial, fisica quàntica

Abstract– This project focuses on the study of *Machine Learning* algorithms, for Computer Vision, in its quantum version. Therefore, it seeks to create and run models of *Machine Learning* in quantum simulators, evaluate the results obtained using common metametrics, and study the difference between classical and quantum computing, which are two paradigms of very distant computing forms, and see what advantages and disadvantages can be found in each. This project has a high content related to quantum physics to give context and have the basis for understanding algorithms implemented in quantum circuits later on, in addition to knowledge of *Machine Learning* in classical computing, as many quantum algorithms have their classical analog part, and the main objective is to compare these versions.

Keywords– quantum machine learning, computer vision, quantum computing, quantum circuits, quantum gates, quantum algorithms, qiskit, artificial intelligence, quantum mechanics

1 Introduction

NOWADAYS classical computer vision techniques are very powerful, since they allow solving complex problems, but in some cases, classical solutions aren't enough. Some problems are intractable for classical computers, and thus, quantum computers comes into play, even though this technology is not as developed as classical computers. Other problems are easily solved by classical computers, but quantum computers can improve/speed up results. For these reasons, in recent years the combination (underdeveloped) quantum technology with (powerful but still with limits) classical technology, appear as a very interesting field of research.

A number of intractable problems can be solved, in theory, with quantum technology. This is refered to as *quantum supremacy* [24], aiming at solving problems in polynomial time that, for classical computers, are intractable and cannot give a deterministic solution in polynomial time. Unfortunately, actual quantum computers are not developed enough to deal with very big problems, because of the engineering limitations, with no solution yet. Actual quantum computers can solve *toy problems*, but for real problems, actual technology is not developed enough to solve it, and we really need understand if quantum computers are able to solve those kind of problems.

In this context, this project aims to study and compare

• E-mail de contacte: juancarlos.martinez.moreno01@gmail.com
• Menció realitzada: Computació
• Treball tutoritzat per: Fernando Luis Vilariño Freire (Deparament de Ciencies de la Computació) / Gael Sentís Herrera
• Curs 2022/23

classical machine learning algorithms with its quantum version, investigating the results obtained, as first insight to the potential that quantum computing can have to solve problems that classical computers cannot do. We will expose the weaknesses and limitations of actual quantum technology, to be aware of where are we now, and about the road that remains to be walked to achieve big quantum functional technology. Although most classical algorithms have their analogous quantum counterpart, others have had to be adapted to their quantum version by means of newly invented algorithms, with their specific modifications and different coding.

In Section 2 we will see the actual progress of Quantum Computing and Quantum ML, in Section 3 there are the detail implementation of all quantum models, in Section 4 we see the results, and the last section is for the conclusions.

2 State of Art

We are now in technological era where notorious advances are occurring thanks to science, research and engineering, but... is this enough to say that actual technology is revolutionizing the contemporary world? We can strongly affirm that technology, and specially the quantum field, is growing and improving very fast [24]. Just 30 years ago, we couldn't think that we would reach the point on we are now, but we have a lot more way to go, it's just the beginning.

2.1 What is Quantum Computing

Quantum computing refers to the use of quantum computers to code quantum circuits, and then algorithms, using qubits and all its properties.

We know for sure that we are living now in NISQ (Noisy intermediate-scale quantum) era, where quantum devices and computers are developed compared to previous years, but there is not notorious speed up comparing to classical computers for large tasks [24]. We will focus now in describing the factors that are limiting this speed ups, and why it's so difficult to develop this technology.

2.2 NISQ: Noisy Intermediate-Scale Quantum era

NISQ refers to actual era, where we only have few quantum computers with approximately 50-100 qubits, which clearly limitates quantum computers performance regard classical computers. This actual technology, of course, is not fault-tolerant: real quantum gates have 0.1% error rate [19], what supposes a very big limitation to multiple operations. Also, these speed ups are not enough big to profit sustainably from quantum supremacy [24].

The most challenging problem quantum computers have to deal with is the unavoidable noise that is generated inside qubits, a native feature of quantum computers, due to the fact that they are very sensitive to the external environment, and they can lose their quantum state because of quantum decoherence [26]. There exists a solution to minimize or remove this noise, what is known as quantum error correction [21], but actual quantum computers are not so sticated enough to apply these error correction methods, and these methods would introduce so much overhead to see benefits instead of deterioration of the performance.

2.2.1 Quantum supremacy

Here are exposed three reasons why for thinking that quantum computers have capabilities surpassing what classical computers can do [24] (quantum supremacy):

- Quantum algorithms for classically intractable problems: we know some quantum algorithms, that have been discovered by researchers, that solve intractable classical problems or that can speed up classical problems. The best example is the problem of finding the prime factors of a large composite integer, solved with Shor Algorithm.
- Complexity theory arguments: theoretical computer scientists have provided arguments, based on complexity theory, showing that quantum states have superclassical properties that, for a quantum computer is very easy to modelate, but for a classical computer is impossible. One example of property is the correlated ability to create superposition of states (probability distribution of a quantum state).
- No known classical algorithm can simulate a quantum computer: we know that, although so many efforts from by physicists to find better ways to simulate quantum systems, it has been impossible for classic computers to simulate large quantum systems.

2.2.2 Why quantum computing is so hard?

The real problem of quantum systems relies on the external perturbations that affect the state of the quantum systems. As we cannot directly observe a quantum system without producing an incontrollable disturbance in the system, we need to keep the system perfectly isolated from the outside world, and also we need to make qubits interact with each other, so we need to control the system from outside, also, and it is very challenge to build a system with these contradictory desired requeriments.

Also, we will have to deal with the problem of protecting the system and scale it with the quantum error correction protocol [21], which means that we have to code a huge entangled state, and this will introduce a lot of overhead due to that additional physical qubits needed to code this entangled state (multiple qubit state that are not separable), for a 50-100 qubit quantum computer. It seems that computers with quantum error correction are not likely to be available very soon.

2.2.3 NISQ era unfolds

Nowadays, we are in the 50 qubit barrier. That means that actual quantum computers, with actual technology, can reach only 50 qubits to work with, and that's good because it gets over what a most actual big supercomputers can simulate with best bruteforce algorithms (factorizing large prime numbers), but it needs to be improved in the next years. However, number of qubits is not the only factor we have to worry about, there are many other factors that we have to work to make the best quantum computers.

In particular, quality of qubits is also very important, because it doesn't matter if we have millions of qubits: if we don't have good qubits, we will have decoherence behaviors, lose of information, and a large etc [26]. With the best hardware we have nowadays, the error rate per gate for two-qubits is above 0.1% [19], and that will limitate the number of operations we can execute in our algorithms, because with a large number of operations, it will appear a lot of noise that will result in bad results. With actual technology is not possible to have quantum error correction protocols, because of the big overhead this can apply to our system.

Other things we have to care about are, for example, the time it takes every gate to execute its operation, and the measurement accuracy of actual materials that implement physic qubits: now rate errors are arround 1% for superconducting qubits, for example, and worse for trapped ions. Another important thing is the amount of effective qubits in the device, because the interconnections of qubits can disable some of them, so we need to take care about the number of qubits we place in a device, in order to not overload that system.

2.3 Quantum Machine Learning (QML)

In the past decade, particularly in the past five years, the combination of powerful computers, special-purpose information processors capable of implementing deep networks with billions of weights, and innovated Quantum Machine Learning techniques has revealed that such deep learning methods are capable of learning complex patterns in data.

2.3.1 Linear-algebra based quantum machine learning

Some classical machine learning algorithms perform their operations with matrix operations on vectors in high dimensional vector space, but quantum mechanics are all about matrix operations on vectors in high dimensional vector spaces [12].

The key ingredient behind these methods is that a quantum state of n qubits is a vector in a 2^n-dimensional complex vector space, where quantum logic operations or measurements multiplies the corresponding state vector by 2^n by 2^n matrices [25]. By building up such matrix transformations, quantum computers have shown to perform common linear algebraic operations such as Fourier transforms, finding eigenvectors and eigvenvalues, and solving linear sets of equations over 2^n-dimensional vector spaces, in time polynomial n, exponentially faster than their best known classical counterparts.

- Principal Component Analysis (PCA) allows us to reduce the dimension of a set of data of dimension n into another dimension lower and also visible for humans, reducing also the complexity of the data by choosing only the most representative dimensions of the data. Because of the popularity of this algorithm, there have been many efforts to reduce the complexity of the algorithm, reaching a time complexity arround $O(n^2)$ with all the operations it has to do. On the other hand, his quantum counterpart uses another form of encoding the data and another operations to reach the objective, taking advantage also of the parallelism effect of quantum mechanics, reducing time complexity to $O((\log d)^2)$.
- SVM: Of of the most well-known example of supervised ML algorithms are linear support vector machines and perceptrons. These methods try to find an optimal separating hyperplane between two classes of data with high probability of one class are only found on one side of the hypterplane. Here, the "weights" learned in the training step are the parameters of the hyperplane. One of the greatest powers of the SVM lies in its generalization to nonlinear hyperplanes with kernel functions [27]. As its classical counterpart, quantum SVM is a paradigmatic example of QML algorithm. First algorithms of quantum SVM were proposed in early 2000s, using a variant of Grover's unstructured search for function minimization. Finding s support vectors out of N vectors consequently takes $\sqrt{N/s}$ iterations, over all the great amount of operations of classical SVM to find the best support vectors.

2.3.2 Quantum deep neural networks

Classical deep neural networks are highly effective tools for machine learning and are well suited to inspire the development of deep quantum learning methods. Special-purpose quantum information processors, such as quantum annealers [16] or programmable photonic circuits, are also well suited for constructing deep quantum learning networks [17]. The simplest deep neural network to quantize is the Boltzmann machine, which consist of bits with tunable interactions: is trained by adjusting those interactions so that the thermal statistics of the bits reproduces the statistics of the data.

2.3.3 Quantum data and quantum algorithms

There are several ways that quantum computers can give advantages in front of classical computers. First, quantum methods can make the system thermalize quadratically faster than its classical counterpart [28]. Second, quantum computers can accelerate Boltzmann training, because the neuron activation pattern in the Boltzmann machine is stochastic, many repetitions are needed to find success, and in Quantum computer, quantum coherence can quadratically reduce the number of samples needed to learn the performance.

In the same way as we can use either classical algorithms or their quantum counterparts, in quantum computing the nature of data can be either classical or quantum. Figure 10 shows the different combinations that can be obtained with classical or quantum data and algorithms. In our work we focus on classical data with quantum algorithms.

Thus, quantum algorithms can be applied to classical data after the needed data preparation step in order to define the quantum states, and with the aim of revealing the underlying features and patterns. For example, given multiple copies of a system described by an N N density matrix, QPCA [12] can be used to find its eigenvalues and to reveal the corresponding eigenvectors in time lower than its classical counterpart, with enough time to be executed with relatively small quantum computers -likely to be available over the next years-.

The data input can come from varios sources, for example from qRAM (quantum RAM) accessing classical data or a quantum subroutine preparing quantum states. Once data is available to the quantum computing device, it is processed with quantum phase estimation [13] and matrix inversion.

Particularly useful are the quantum simulators aimed at probing quantum dynamics. Quantum simulators are "quantum analog computers", quantum systems whose dynamics can be programmed to match the dynamics of some desired quantum system. A quantum simulator can be a special purpose device constructed to simulate a particular class of quantum systems, or a general purpose quantum computer. This exponentially reduces the number of measurements needed to perform the simulation.

2.4 Current software frameworks for quantum programming

Quantum programming is nowadays easy because of the great amount of options we have to do it. Specifically, we can find great options of packages for Python that are easy to use and very powerful. This project is focused on Python because of the simplicity and the speed up that offers this programming language, versus other types of languages like C or Java. Also, the main advantage of Python in front of others is the large amount of packages to use in our programs, and in quantum programming, it is essential. Another advantage is that Python is a interpreted language, and C uses a compiler, what makes executions a little bit slower than Python.

2.4.1 Qiskit

First of all, we will talk about the first package I used and, in my opinion, the easiest to use because of the simplicity of the functions and also of the tutorial. Qiskit [6] is the first option I tried because of the simplicity and at the same time complete tutorial it has got, that is ideal for programmers that are introducing themselves in quantum world with clear explanations of quantum mathematical bases. It also has got simple functions that allow the creation of quantum circuits with few and clear instructions, and the ease to create quantum gates and place it on quantum circuits. Then, simulators are easy to create and test the circuits, and Qiskit allows testing these quantum circuits on real quantum computers, owned by IBM, and see the real outputs of our circuits to compare the results and see the performance (which also includes elapsed time).

2.4.2 Cirq

Another alternative we have to do quantum programming is the package Cirq [2], made by Google and also available for Python. Cirq offers us the same tools that Qiskit does, but I think is a little bit more complex because of the form in that quantum circuits are made. While in Cirq we have to create sepparately qubits and the circuit, in Qiskit it is understood that they are together, and we only have to instantiate just one time the circuit. Also, in Cirq it is introduced the concept of Moment, that is the actual time in which we are at a certain time, and we have to deal with Moments and take control all the time of it, to not produce bad results. To finish, in Cirq we can not find as many quantum gates as in Qiskit, and we have to create custom gates derivated of other simple gates (for some cases, the possibility of creating custom gates is an advantage).

2.4.3 Pennylane

Now, a powerful package that is exclusively destined to Quantum Machine Learning, and offers a lot of powerful tools to create quantum models is Pennylane IA [5]. Pennylane IA is the strongest package for quantum programming, but is the most difficult to use also. As the others it has got the same tools to build quantum circuits, but it doesn't offer as much quantum machine learning algorithm templates as it offers other quantum libraries like Qiskit.

2.4.4 Other implementations

Other packages for quantum programming are, for example: Azure (Microsoft), Braket (Amazon) Open Source: ProjectQ

3. Building blocks of quantum computing

First of all, it is important to know some basic quantum concepts to understand the main topic of this article. Quantum mechanical phenomena are difficult to understand because most of our experiences are not directly applicable to it, but there are some experiments we can do (most popular is photon polarization [25]), that can show us the behavior of quantum mechanics [28] [29]. There is also a strong mathematical basis that we have to learn and understand: it is governed by a set of axioms, and it has also consequences that describe the behavior of quantum systems.

3.1 Quantum bits

A quantum bit, or a qubit, is a unit vector in a two dimensional complex vector space for a particular basis, denoted by $\{|0\rangle, |1\rangle\}$ [22] . The orthonormal basis $|0\rangle$ and $|1\rangle$ correspond to $|\leftrightarrow\rangle$ and $|\updownarrow\rangle$ polarizations of a photon, and basis $|0\rangle$ and $|1\rangle$ correspond to bit values 0 and 1. Unlike classical bits, qubits can be in a superposition of $|0\rangle$ and $|1\rangle$ such as $a|0\rangle + b|1\rangle$. Qubits are represented in Figure 1.

Even though a quantum bit can be put in in nitely many superposition states, it is only possible to extract a single classical bit of information from a qubit, because information can only be obtained by measurement. In measurement, we change the state to one of the basis states (collapsing the qubit to one determinate state).

For example: given a device for measuring the polarization of photons with associated basis $\{|\leftrightarrow\rangle, |\updownarrow\rangle\}$, the state $|\psi\rangle = a|\leftrightarrow\rangle + b|\updownarrow\rangle$ is measured as $|\leftrightarrow\rangle$ with probability $|a|^2$ and as $|\updownarrow\rangle$ with probability $|b|^2$.

3.2 Multiple qubits

To make useful qubits, we have to group them into a big multiple qubit, and each of its small pieces (a single qubit) can be described separately. So, in classical physics, we

Fig. 1: Multidimensional representation of a qubit (Bloch sphere). [8]

have that the possible states of a system of n particles, whose individual states can be described by a vector in a two dimensional vector space, form a vector space of n dimensions.

However, in a quantum system, the resulting state space is much larger: its space has 2^n dimensions. This exponential growth of the state space suggests a possible exponential speed-up of computation on quantum computers over classical computers.

3.3 Entangled qubits

There are some qubits that can give us information about each other with the measurement of only one of them. How can this be possible?

Imaginate the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, that is an entangled state that is part of the Bell's basis. Note that, if we measure the first qubit, we will know for sure which will be the outcome of the measurement of the second qubit, because if we measure first a 0, the only state with 0 as first qubit is the state $|00\rangle$, so we will know for sure that the second qubit will be also 0. Moreover, the state $\frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$ is not entangled, because measuring first qubit will give us the same uncertainty on the second measurement. This powerful property of quantum mechanics is essential for some algorithms to offer speedups versus classical algorithms.

3.4 Quantum gates

We have only seen systems without any change on qubits, only when measurement is realized, but the state of each qubit can suffer transformations that change its state [20]. For a complex vector space, linear transformations that preserve orthogonality are unitary transforms that can be described by a matrix. Let $M^\dagger$ denote the conjugate transpose of the matrix M, that is unitary (describes a unitary transformation) that $M^\dagger M = I$. You can think of unitary transformation as a rotation of a complex vector space. In addition, an important consequence of the fact that quantum transformations are unitary is that they are reversible, so quantum gates must be reversible.

To build quantum circuits, we have simple quantum gates that are the analogous form of classical gates like AND, NOT or OR. Then, there are others that are more complex that are composed of simple quantum gates.

3.5 Classic Quantum Algorithms

With all basic concepts of Quantum physics described, it's now time to start by exposing some classic quantum algorithms [23], that are the base of all Quantum Machine Learning algorithms that are developed and others that are being developed at this moment.

These algorithms are clasically being teached in courses of introduction to Quantum mechanics and computing, so it is understood that are simple algorithms, with associated simple quantum circuits that only use simple quantum gates, and sometimes other a little bit complex.

3.5.1 Grover's Algorithm

The main feature of Grover's Algorithm is the amplitude ampli cation trick, that consists on amplifying the amplitude of the element that we want to find. For that, we will pass to the algorithm a unstructured list, and the algorithm, with an oracle, will amplify the amplitude of the element that we want to find. That's why we will have to code the Oracle to do an specific task, on a specific element of the list.

We will call this element we want to find the winner, w. To find this element, in classical computing we would have to check, on average, N/2 of all the elements of the list (half of all), and in the worst case, all of them, and in the best case, we will find it at first position. On quantum computing, we can find it in $\sqrt{N}$ steps with Grover's amplitude ampli cation trick. Additionally, the algorithm doesn't use the list's internal structure, because it works with qubits, which makes it generic and reduces the complexity.

Now, we will talk about the oracle. We have to create a oracle that is able to change the phase of the element that we want to find. For example, we have 3 qubits (and that involves that our unstructed list is: [000,001,...,111]), and we want to find the element 101, so w = 101. The oracle will do:

$$U_! |x\rangle = \begin{cases} |x\rangle & \text{if } x \neq ! \\ -|x\rangle & \text{if } x = ! \end{cases}$$

Once we have the phase of the element we want to find, we will apply an additional reflection $U_s = 2|s\rangle\langle s| - I$) that reflects the unique negative state (the state we want to find) with large amplitude than the rest of the elements. Finally, when we measure all the qubits, we will find that the most probable state is the state we want to find, the winner state.

Fig. 2: Grover's Algorithm general components. [6]

3.5.2 Shor's Algorithm

Shor's Algorithm, described in Figure 3, is famous for being able to factor big integers in polynomial time, something that classical algorithms cannot, because is considered a NP

problem that cannot be resolved in polynomial time. The aim of this algorithm is to find the period of any function.

To solve the problem of factoring a large integer, we will first compute the period of that function, that automatically gives us the possibility to do that factoring: $a^x \bmod N$. This function has a period behavior, because we have that property we can take advantage of to find the period: $a^r \bmod N = 1$. Here, we can see that, every r times, this function result returns to 1. That's the periodic behavior we want to understand and we want to find.

3.5.3 Quantum Fourier Transform

Quantum Fourier Transform, described in Figure 4, or the analogue of the classical Fourier Transform, is widely used in physics and maths to transform signals from time or spatial domain to frequency domain. One example of application is, in Computer Vision, to correct images and delete low frequencies, in order to smooth an image that is full of noise.

Here, in the Quantum world, we are going to use the Quantum Fourier Transform in other algorithms, so QFT is an essential part of many algorithms such as Shor's Algorithm, or the Phase Estimation algorithm. However, Quantum Fourier Transformation has the same core, that is to transform from one domain to frequency domain, and we will reach that by applying some transformations over the amplitudes of the signal.

As a introduction, we are going to see the fundamentals of QFT and the transformations it does. The discrete Fourier transform acts on a vector (x0,...,xN) and maps it to another vector (y0, ... , yN) according to this formula:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk}$$

where $\omega_N^{jk} = e^{2\pi i \frac{jk}{N}}$, a simple rotation of the state over the z axis. So, the final mapping that QFT does to our initial qubit, is:

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} |k\rangle .$$

3.5.4 Quantum Phase Estimation

Phase Estimation Algorithm, described in Figure 5, that is part of many important algorithms in Quantum Computing, such as Shor's algorithm, and uses the Quantum Fourier Transform to take part into the process of what Phase Estimation does. As the name of the algorithm suggests, it is used to estimate the phase of an operator/function called U, so: $U|\psi\rangle = e^{2\pi i \theta}|\psi\rangle$, where θ is the phase we want to estimate.

Taking a look at a representation of the circuit that does the Phase Estimation:

Here we have the first register, that will act as a counter, and on which we will store 2^n. The last qubit is in the second register, and it is used to support the QFT operation that is being done at first part.

First of all, we will apply a Hadamart gate to all qubits of the first register, and then we will do Controlled-U operations for every qubit that is in the first register: qubits in first register will act as the control qubit, and the last qubit of the second register will act as the target qubit. Note that this process is the same as what we were doing when developing the QFT algorithm. (Controlled-U is a custom gate that has a control qubit and does the Unitary Operation we specify on the target qubit if control qubit is 1).
Then, we will have to undo the QFT operation by doing the inverse of QFT. This will give us the phase estimation of the Unitary Operator in a computational basis, because first we go to Fourier basis, and this gives us the phase, but we have to return to computational basis to measure the result.

3.6 Quantum models for Computer Vision

This section is to describes all the Quantum ML models implemented in this work. including the grade of "Quantum components" that have each algorithm/model, details of implementation and diagrams showing the structure of each one, and its components.

3.6.1 Classical data embedding

As mentioned above, we will process classical data because in Computer Vision problems the data of classical nature, so it doesn't make sense to treat with quantum data - we would skip the data embedding part if we have quantum data already available-. We will use quantum embedding techniques [7] to encode data from classic space to quantum states. There are a many techniques available, but the easiest one is to make arbitrary rotations over the data, in all its features, to prepare classical bits to quantum states that are over the Bloch sphere as shown in Figure 1. Depending on the type of the data we have (bits, number, float...) we have different forms of encoding the data. We can encode it only with the Angle of the rotation (Angle Embedding), or we can encode it with the amplitude of the state (Amplitude Embedding).

3.6.2 Quantum kernel methods for hybrid classification with SVM

Quantum kernel evaluation methods can be used withclassical classification models [27] in this case, Support Vector Machines, that is a very simple model that offers great results when using optimal parameters and kernel.
We have to first define what is a Kernel, and for what is used. A Kernel function is simply a function that can implement the dot product on a transformed space where data are linearly separable. In ML tasks, with the objective of finding clear patterns from data to predict new inputs, it is fundamental to find a method that does this transformation, but kernel methods do not perform this transform in reality, they do a "functional" transform that gives us the illusion that the transform was done, but in fact they only perform inner products of all pairs of data.

If we want to transfer this idea to the quantum world, Kernel functions can help us to do the encoding from classical data to quantum data, to have the data processed for being an input for qubits on quantum circuits. In other words, Kernel functions can also do a map between vector space of classical data to Hilbert space of quantum mechanics [27]. This new space has got the data linearly separable, while the old space had the data non-linearly separable.

Fig. 3: Quantum circuit associated with Shor's algorithm. [6]

Fig. 4: Quantum circuit associated with QFT algorithm. [6]

Fig. 5: Quantum circuit associated with PE algorithm. [6]

This method of quantum classi cation is the less "quantum effective", because the quantum fraction of all the algorithm is only a small part of it: the data embedding to get quantum state from data, and the kernel evaluation. Finally, the result state is measured to obtain the new data with the space transformation done.

The type of kernel functions we can develop are described below:

- Linear kernel: consists of simply doing the dot product of each pair of states (data encoded to quantum states). It is the simplest kernel possible, and the expected to work the worst.
- Polynomial kernel: consists of applying a polynomial function of degree x (chosen by the user) to the dot product of each pair of states (as linear kernel does, but applying polynomial function).
- RBF kernel (Radial Basis Function): consists of applying the Radial Basis function of every pair of states. This function calculates the distance between these pairs, multiplies it by the negative value of a gamma parameter, that is choosen by the user, and nally computes the exponential function. $K(x; x') = \exp(-\gamma \|x - x'\|^2)$
- Sigmoid function: consists of appyling the sigmoid function to every input pair of states. $K(x; x') = \frac{1}{1+\exp(-\gamma(x;x'))}$. This function maps every pair of states to a value between 0 and 1, representating the probability of being part of each class.

3.6.3 Quantum Neural Networks

Another type of classi cation model implemented in this project, that has direct tools for its implementation with Python packages, such as Qiskit, Pennylane, or Tensor ow, are the Quantum version of classical Neural Networks [17], described in Figure 16.

To do a brief introduction about Neural Networks, Neural Networks are models that follow the following structure: they are composed of layers (group of several neurons), and Neural Networks usually have an input layer, where data is encoded, some hidden layers, and the output layer, where the result of the classi cation is shown as the probability of being part of each class.

Another elements that are part of Neural Networks are the activation function, that are some kind of functions that are part of layers, and as the name suggests, they are activated when data is on them, and they can throw an output that passes through their function. There are a lot of activation functions: ReLU, softmax, sigmoid...

Also, the layers themselves are classified in groups depending on the operations they are doing, and their features. Finally, that type of models, like the others, have its own hyperparameters that the user has to tune to find the best values that give the best results and minimize the cost function. Some of the hyperparameters that have Neural Networks are the weights of each neuron, the bias, or the number of layers itself that has got the Neural Network. We can also test with some types of cost functions, and optimizers that find the optimal hyperparameters (ADAM optimizer is the most famous).

On the other hand, QNNC (Quantum Neural Network Classifiers) [17] are general instances of the general Neural Network architecture presented on, and adapted to the quantum mechanics combined with classical. Here we have the circuit prepared to receive the input data (corresponding to number of qubits = number of features/dimension of the data), and then the intermediate layers process the data with their weights and bias to take out the output.

Here, talking in quantum context, we can expect the output to be in various forms: we can receive the output in form of the estimation of the expected values (Estimator primitive), or we can receive the probabilities of quasi-probabilities of the bitstrings from the quantum circuits (Sampler primitive) [6].
The quantum parts of the classical Neural Networks are: the Feature Map, used to encode the data from classical form to quantum states, and the Ansatz, that is the core of the algorithm, what in classical Neural Networks are the layers with the neurons [6].

3.6.4 Variational Quantum Classifiers

VQC (Variational Quantum Classifiers) [14] are special and concrete instances of the general Neural Networks, but with the fact that they are a simulation of it. VQC are not made of pure layers like the classic Neural Networks, instead they are made of repetition of layers that are composed of quantum variational circuits, that are the base of the quantum computing.

Variational circuits are mainly circuits that ara parametrized, so the behavior and the output of the circuits changes with the value of the parameters, and usualy these parameters are used for rotations, to encode the data or output the probabilities of each class, in classification tasks, but Variational Circuits are used for other many types of tasks.

Qiskit gives us templates of VQC created to use easily, but Pennylane doesn't have templates to instantiate easily, so we have to create our custom VQC, needing basic concepts of quantum computing to build all the circuits needed. In this project, I have created custom circuits with Pennylane for classification and also I have used templates that Qiskit gives. One example of FeatureMap and Ansatz can be found in Figures 11 12

3.6.5 Quantum Convolutional Neural Networks

QCNN [15] are the quantum version of the classical Convolutional Neural Networks, described in Figure 15, that are able to extract features that basic Neural Networks cannot do, and that's why they are widely used in image classification problems, because of the ability of extract important features of images with computer vision techniques, that are important to get the essential information about an image and do a great classification. Here we have, what we call Convolutional Layers, that apply the technique of convolution that recognize patterns in image like corners or lines, and extracts useful information for the network. And then, we have pooling layers that reduce the information for the next layers.
As Qiskit nor Pennylane offers templates for QCNN, we have to made our custom Network by creating custom circuits for simulating a conv layer and pooling, also. One example of Convolutional Layer and Pooling Layer can be found in Figures 13 14

3.6.6 Quantum version of classical models: KNN

The last type of classification models that are implemented in this project is the general supervised models that are used for a lot of classification tasks: Logistic Regression, KNN, Decision Trees, Random forests... [18] But, because of the complexity of creating these models in quantum version, we will only see KNN model, that is ”relatively” easy to code with Qiskit/Pennylane.

It simply calculates the distance between each new input that enters to the system with all the data train set, and then the new data takes the majoritary class of the K nearest neighbors (like the name indicates). So, we have to find a routine in quantum computing that calculates the distance between every new data and the rest of the points. Fortunately, we can build a routine with the basic algorithms we already know that exist in quantum world, that can do this task very fast, taking advantage of all the properties that give quantum mechanics (superposition, entangling...). This routine is call SWAP-TEST and it is described in Figure 6.

Fig. 6: Representation of the SWAP-TEST routine. [9]

4 Results

4.1 Description of all developed models

In our experiments, we used 5 toy datasets: Iris, Wine, Random, circle [11], and images. The details of each dataset are described in Appendix and Figure 9.
The parameters/features used in all models (classical and quantum) are:

- Support Vector Machines: default parameters and degree = 2, gamma = 0.5
- K-Nearest Neighbors: number of neighbors = 3
- Neural Networks for classification: 3 hidden layers with ReLU activation function, an output layer with

softmax activation function (for binary classi cation), with ADAM optimizer and Binary Crossentropy loss function. 30 epochs in training phase.

- Convolutional Neural Network: 2 convolutional layers with ReLU activation function, 2 pooling layers, with ADAM optimizer and Categorical Crossentropy loss function.

4.2 Classical and Quantum results

The following basic classical models (toy models) [3] [10] to compare with quantum models (core of the project), for classi cation tasks, have the performance results in Table 1, picking accuracy score to compare the results between different models

And the results obtained on the execution of Quantum developed models [6] [5] are in Table 2, showing the score of each model tested with all datasets we have:

Focusing on neural networks, we can evaluate the performance tracking the training value of the objective function, that allows models to learn and improve performance:

(a) Quantum NN training track(b) Classical NN training track

Fig. 7: Classi cation Neural Networks (classical/quantum)

Note that Quantum version does a successfull reduction of the objective function, but is not as ef cient and faster as it is the classical version. The Quantum version cannot achieve the value 0 in 30 epochs, while classical version can achieve it in approx 15-20 epochs.

With Convolutional Neural Networks, happens something similar:

(a) QCNN training track (b) CNN training track

Fig. 8: Convolutional Neural Networks (classical/quantum)

With Quantum version, we can see that the reduction of the objective function is slowly and never reaches the value 0, while classical version has got fast reduction and reaches 0 in approx 70-75 epochs, what is a lot faster than quantum version.

5 Conclusions

With all the results exposed and brie y discussed, all the theory and the actual state of art also explained, it can be seen that nowadays the tools we have for quantum programming are many and varied, but this technology is not madure enough to develope quantum models that are useful por real problems, especially in Computer Vision that is a complex computer science area.

One aspect to take into account is that in this project it is only used quantum simulators, not real computers. So, quantum simulators simulate noise, but it doesn't work as a real computer. Consequently, a real study of the noise generated in the outputs of the models couldn't be done in this project.

From the results obtained, it can be extracted some important things to take into account in the development of quantum models of machine learning:

- In terms of current quantum technology, and as theory says, users don't have access to quantum computers with more than a few number of qubits (approx. 30 qubits), and other few number of premium access computers with max 130 qubits [4]. These computers are owned by IBM, and can be only accessed through internet (on cloud), where users can execute their quantum circuits with few number of qubits.
- In terms of current quantum algorithms, and especially in Machine Learning problems, they are now in an initial phase where these algorithms are not enough developed to process real data that is huge and more complex that some toy datasets, which these algorithms can process with some limitations. As theory suggests, the rst quantum algorithms that appeared were tested for proving that they bring some speedup comparing to classic algorithms that do the same tasks.
- Linking to the previous point, the results we obtain from classical and quantum algorithms (leaving aside the con guration of the best hyperparameters of the models), with the same hyperparameters usually classical models outscore quantum models, but there are not very big differences for the initial state we are nowadays.
- As the models are getting more complex, quantum models score decay. Although we are using quantum simulators to execute our models, theory suggests that, as bigger is the quantum circuit, the worse results we will obtain because of quantum decoherence and noise that is generated as it grows. It also happens with the data distribution, datasets with complex distribution of data do worse classi cation.
- Elapsed time to execute one model is also important. In the execution of all quantum models, it can be seen that elapsed time is bigger than classical models, even we are using quantum simulators. Some aspects are the time to prepare the classical data to quantum states, or the own simulators that are slower than real quantum computers.

To sum up, it is known that there is still a long way to go to achieve decent quantum technology, and algorithms

Model / Dataset	Iris	Circles	Random	Wine	Image
SVM (linear kernel)	100%	30%	50%	97%	-
SVM (polynomial kernel)	60%	95%	80%	88%	-
SVM (RBF kernel)	100%	95%	90%	94%	-
SVM (sigmoid kernel)	100%	60%	80%	88%	-
KNN	100%	100%	100%	80%	-
Neural Network	100%	91%	91%	100%	-
Convolutional Neural Network	-	-	-	-	100%

TABLE 1: CLASSICAL MODELS RESULTS

Model / Dataset	Iris	Circles	Random	Wine	Image
QSVM (linear kernel)	80%	44%	76%	95%	-
QSVM (polynomial kernel)	92%	84%	68%	97%	-
QSVM (RBF kernel)	84%	80%	63%	93%	-
QSVM (sigmoid kernel)	20%	44%	60%	40%	-
QKNN	100%	96%	88%	52%	-
Variational Quantum Classi ers (extensive QNN)	60%	80%	76%	75%	-
Quantum Convolutional Neural Network	-	-	-	-	60%

TABLE 2: QUANTUM MODELS RESULTS

capable of satisfying current needs, but more and more research is being done in this area, and it is currently a topic of great interest, since achieving quantum ML algorithms would mean a great advance in computer science, having algorithms capable of solving problems today too complex for a classical computer, or even for big supercomputers that are the most complex and powerful classical technology we have nowadays.

ACKNOWLEDGLEMENTS

I would like to thank all the support that my two tutors have given me, Fernando and Gael, because they have been fundamental in this project so that it could be carried out, helping me with every doubt and problem that arose, and giving me advice and tools to go deeper into the topic.

Also, I would like to thank all my family and friends for their strong support in this dif cult and sacri ced stage.

REFERENCES

[1] Arti cial neural network architecture. URL: https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051 .

[2] Cirq tutorial. URL: https://quantumai.google/cirq .

[3] Classical models from scikit-learn package of python. URL: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning .

[4] Ibm quantum. URL: https://quantum-computing.ibm.com/services/resources?tab=systems .

[5] Pennylane api for qml. URL: https://docs.pennylane.ai/en/stable/code/qml.html .

[6] Qiskit machine learning api. URL: https://qiskit.org/ecosystem/machine-learning/apidocs/qiskit_machine_learning.html .

[7] Quantum embedding — pennylane. URL: https://pennylane.ai/qml/glossary/quantum_embedding .

[8] Qubit - wikipedia. URL: https://en.wikipedia.org/wiki/Qubit .

[9] Swap test - wikipedia. URL: https://en.wikipedia.org/wiki/Swap_test .

[10] Tensor ow for neural networks. URL: https://www.tensorflow.org/api_docs/python/tf .

[11] Toy datasets from scikit-learn package of python. URL: https://scikit-learn.org/stable/datasets/toy_dataset.html .

[12] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. 2018.

[13] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. pages 53–74, 5 2000.

[14] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational quantum algorithms. Nature Reviews Physics, 3:625–644, 12 2020.

[15] Iris Cong, Soonwon Choi, and Mikhail D Lukin. Quantum convolutional neural networks.

[16] Diego de Falco and Dario Tamascelli. An introduction to quantum annealing. RAIRO - Theoretical Informatics and Applications, 45:99–116, 7 2011.

[17] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. 2018.

[18] D V Fastovets, Yu I Bogdanov, B I Bantysh, and V F Lukichev. Machine learning methods in quantum computing theory.

[19] Robin Harper and Steven T Flammia. Learning correlated noise in a 39-qubit quantum processor.

[20] Robert Hundt. Quantum computing for programmers. 2022.

[21] Akshaya Jayashankar and Prabha Mandayam. Quantum error correction: Noise-adapted techniques and applications. Journal of the Indian Institute of Science, 7 2022.

[22] Michael A. Nielsen and Isaac L. Chuang. Quantum Computation and Quantum Information.

[23] Renato Portugal. Basic quantum algorithms. 1 2022.

[24] John Preskill. Quantum computing in the nisq era and beyond. Quantum, 2, 1 2018.

[25] Eleanor Rieffel, F X Palo, Alto Labratory, and Wolfgang Polak. An introduction to quantum computing for non-physicists. ACM Computing Surveys, 32:300–335, 9 1998.

[26] Maximilian Schlosshauer. Quantum decoherence. Physics Reports, 831:1–57, 11 2019.

[27] Maria Schuld. Supervised quantum machine learning models are kernel methods. 1 2021.

[28] James D Whitfield, Jun Yang, Weishi Wang, Joshuah T Heath, and Brent Harrison. Quantum computing 2022. 2022.

[29] Hiu Yung Wong. Introduction to Quantum Computing. 2022.

APPENDIX

A.1 Description of all datasets used

Here we have a brief description of the different collection of datasets that have been used in the testing of the performance of all models created:

- Iris dataset: consists of data of 3 different types of irises (Setosa, Versicolour, and Virginica), describing 4 features about them (example: petal and sepal length). So, the original dataset is 150x4 and 3 different classes, but for our benchmark, we have reduced dataset length to 100 samples and 2 classes, for binary classification.
- Circles dataset: simple toy dataset that randomly creates synthetic data for classification or clustering problems. Makes a large circle containing a smaller circle in 2d, and we can specify as much samples as we want. For our benchmark, we specified the same amount of samples as Iris (100), with 2 features (2D), and 2 classes for binary classification.
- Random dataset: another toy dataset that also randomly creates synthetic data, with the difference that data is not structured along a circle, it is placed randomly over 2D space. As circles dataset, we can specify amount of samples to create (100 for our benchmark), with 2 features and classes.
- Wine dataset: consists of data about different samples of wine (178), and it has 13 different features measuring different things about wines, like for example % of alcohol, and it has 3 different classes. For our benchmark, we picked all samples with all 13 features, but we reduced the number of classes to 2, for binary classification, as the other datasets.
- Random generator of small images (4x2) containing horizontal or vertical lines, for image classification.

Some of them were preprocessed to reduce the dimension because Quantum models have limited number of qubits to use, because of the actual quantum technology, so it is compulsory to reduce the dimension, and for convenience to plot it, we have reduced it to 2 dimensions.

(a) Iris Dataset

(b) Circles Dataset

(c) Random Dataset

(d) Wine Dataset

Fig. 9: Datasets used in the testing process

Fig. 10: Types of ML Algorithms and data.

-