Universidade Estadual de Campinas

CXQ - A Real World Study on Quantum Machine Learning - F894

Tadeu Pereira da Silva

July 16, 2022

Contents

1 Introduction

Machine Learning is a topic that has been getting increased attention from all over the world in the last decade.
The hability to make computers learn specific tasks is now possible by applying automatic differentiation in large collected datasets. This has enabled different problems that used to be inefficient to be solved in a couple lines of code. At the same time, quantum computers became more feasible, with some companies achieving more than 50 qubits [2]. Thus a new field was created: **QML - Quantum Machine Learning**, the idea of applying quantum computers to solve Machine Learning tasks. Progress has been made in classification tasks using a quantum version of SVM [7], and even a quantum neural network method called QVC - Quantum Variational Circuits [3], using the same ideas as the "Classical" Neural Networks (bakpropagation and optimization by gradient descent). Those techniques use quantum principles like superposition and interference to learn parameters that can be trained and solve specific tasks (e.g. determine which digit is being shown in a image). One of the challenges of this new field of research is to determine two things: if the quantum approach has advantages in comparison with the classical one (which in some cases has been shown to be true for some quantum-related problems [8])and if the computation is time efficient due to the limitation of qubits number of today's quantum devices.

2 Motivation

Among the various classical machine learning techniques probably the most important is the **CNN - Convolutional Neural Network** for image data. This method has nice properties like spacial independence (which makes it possible to generate more data from the same image using linear transformations e.g. rotation) and high-dimensional featurization (the output can be higher dimensional, the CNN is able to abstract information in the image even if it's blurred / noisy). It was this particular method that proved to successfully train neural networks to learn to classify 14 Million images in the Imagenet Dataset [6]. A quantum version of the Convolutional Layer would be of immense value to the field, because it would bring the power of the image information extraction to the quantum world.

Following that reasoning a paper with this attempt was published [5], introducing a **Quanvolutional Layer**, which would make it possible to apply the same method for quantum algorithms. The idea of this project is to apply the concepts presented in that paper for real world scenarios, using an open-source library called **Pennylane** provided publicly by the Xanadu company, and extend the concepts for

different approachs.

With this new Quanvolutional layer available, the goal is to compare the Classical Convolutional with the Quantum Convolutional (Quanvolutional) in different datasets (including real world scenarios)

The code and results are also published in the public available Github repository `https://github.com/dedsylva/CXQ-ML`

3 Study & Results

The computations used in this project were applied for the following datasets:

- MNIST (60k Images of classifying 0-9 digits)
- Imagenet Subset (240 Images of Classifying Ants/Bees)
- COVID-19 (317 Images of X-rays for COVID detection)
- Malaria (27.5k Images of Blood Cells for Malaria detection)

For each dataset this CNN was trained using TensorFlow Library with the following parameters:

Dataset	Epochs	Optimizer	Batch Size
MNIST	5	Adam	32
IMAGENET	20	Adam	32
COVID	5	Adam	32
MALARIA	20	Adam	32

Table 1: Specification for each Dataset

The project was divided in three parts: the **Classical Approach**, the **Quantum Approach** and the **Information Extraction**.

3.1 Classical Approach

For the classical approach the images of each dataset was downloaded publicly on the internet, divided in different folders, separated the data from training and testing,

and a typical CNN Neural Network (see Appendix A) was created with the following structure:

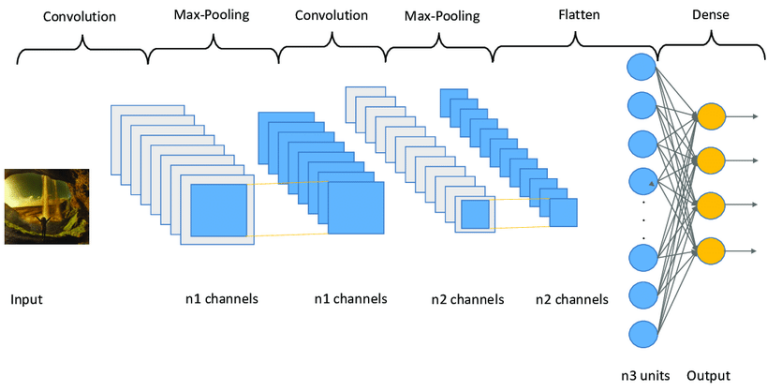


Figure 1: Classical Neural Network Architecture [4]

After training, each CNN was tested on their respective test data. The accuracy was measured accordingly to each dataset:

Dataset	Accuracy (%)	AUC (%)
MNIST	98,4	99,9
IMAGENET	61,4	68,4
COVID	84,5	95,4
MALARIA	51,2	52,4

Table 2: Results for Classical Neural Networks

3.2 Quantum Approach

For the Quantum Approach the same images were used, and following the guidance of the quanvolutional paper, the quanvolutional operation was applied in the first layer of the neural network, which can be though as an quantum pre-processing of the classical image data, as shown below:

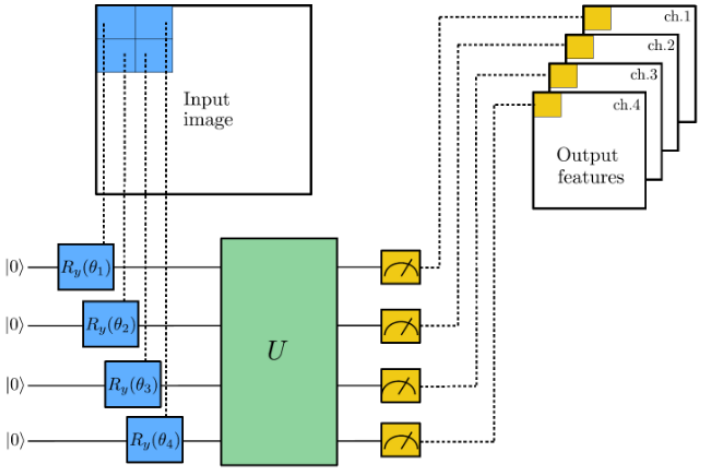


Figure 2: Quanvolutional Neural Network Architecture (Quantum Convolutional Layer) [1]

The quantum pre-processing was performed for each image of each dataset, using two different methods: **Random Quantum Gates** and **Fixed Hadamard Gates**. The idea is to test if the choice of the Quantum Gates Operators (see Appendix D) matters, or if a random quantum gate operation is suficient for extracting information from the image.

After passing the quanvolutional layer, the process was the same as the classical approach: using a Classical Dense Neural Network (see Appendix B) and generating the output accordingly to each dataset.

Dataset	Prediction Output
MNIST	10 Dimensional
IMAGENET	1 Dimensional
COVID	3 Dimensional
MALARIA	1 Dimensional

Table 3: Probability Vector Output

3.2.1 Random Quantum Gates

For the Random Quantum Gates (where each image was going through a random quantum gate after being quantized by Y-Rotation) the results were the following:

Dataset	Accuracy (%)	AUC (%)
MNIST	98,3	99,9
IMAGENET	58,9	61,7
COVID	86,4	96,5
MALARIA	55,0	61,5

Table 4: Results for Random Quantum Neural Networks

3.2.2 Hadamard Quantum Gates

For the Fixed Hadamard Gates (where each image was going through a Hadamard Gate after being quantized by Y-Rotation) the results were the following:

Dataset	Accuracy (%)	AUC (%)
MNIST	96,7	99,5
IMAGENET	55,5	63,2
COVID	81,8	93,6
MALARIA	55,0	55,0

Table 5: Results for Hadamard Quantum Neural Networks

3.3 Information Extraction

For the third and last part of the project, the idea was to create a visualization tool that enabled a clear understanding of what each algorithm is producing.

So for each type of Neural Network (the Classical Convolutional, the Random Quanvolutional and the Hadamard Quanvolutional) a test was performed: an image from each dataset was taken, passed through each method, and the output was plotted, showing what each produced.

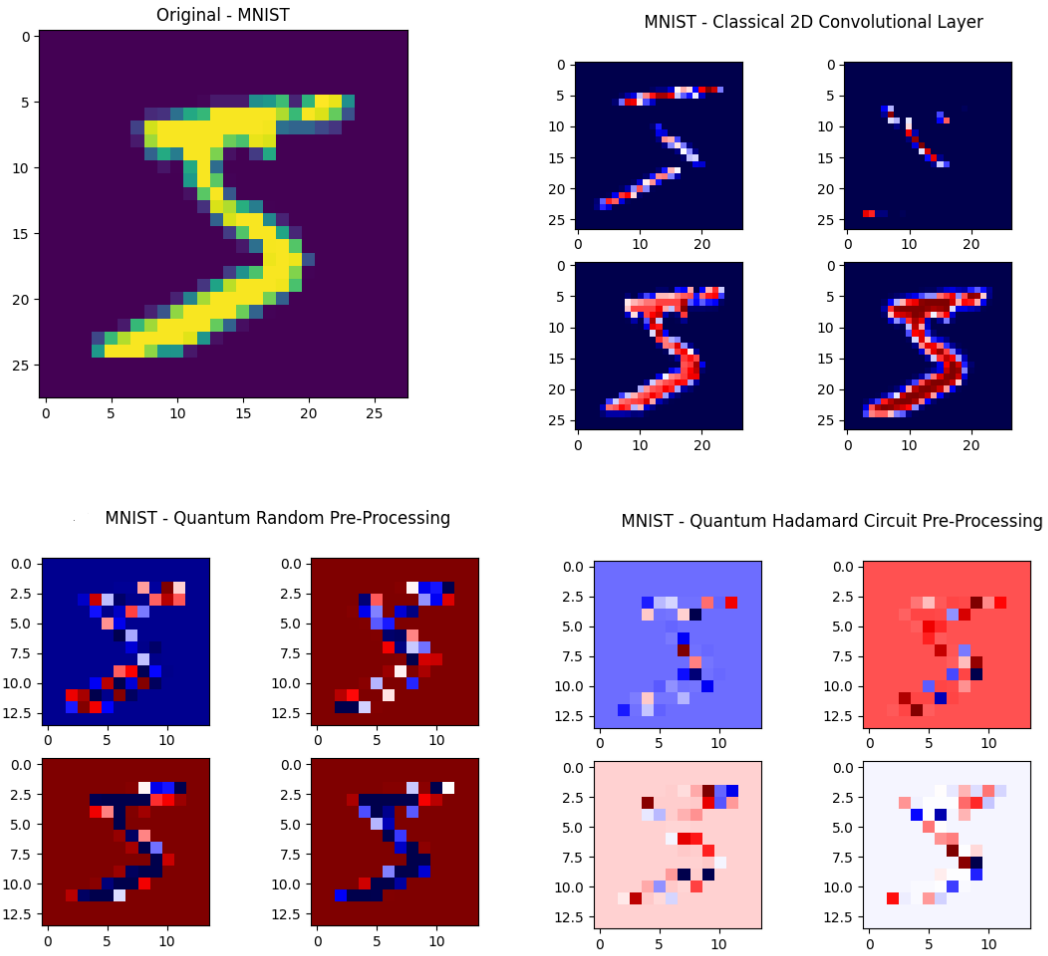


Figure 3: MNIST Dataset: Numeric Digit Before / After Transformation

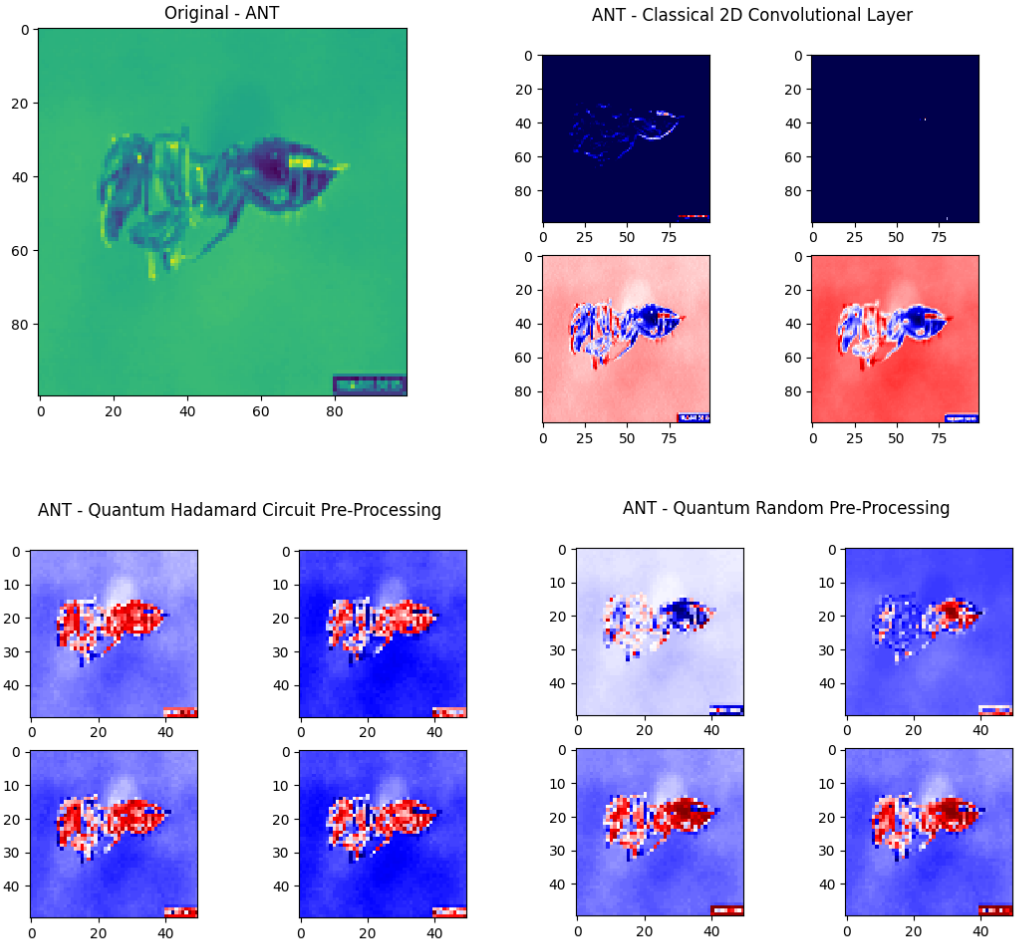


Figure 4: IMAGENET Dataset: Ant Before / After Transformation

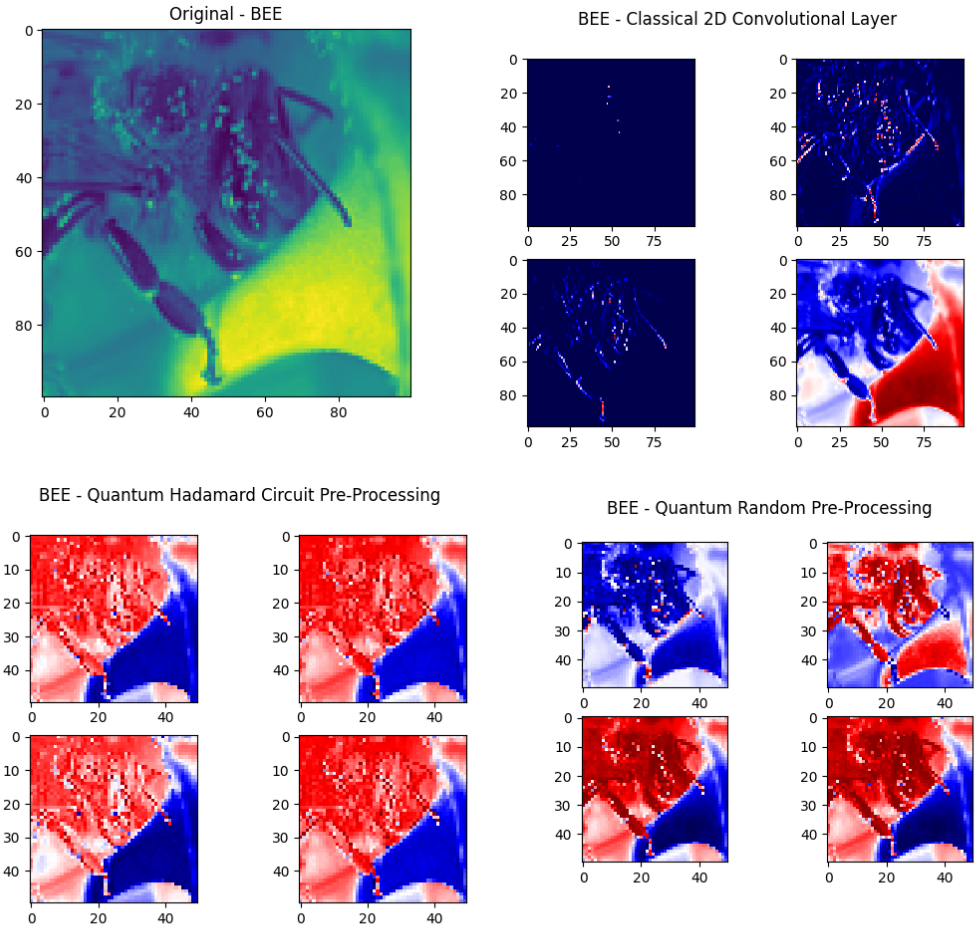


Figure 5: IMAGENET Dataset: Bee Before / After Transformation

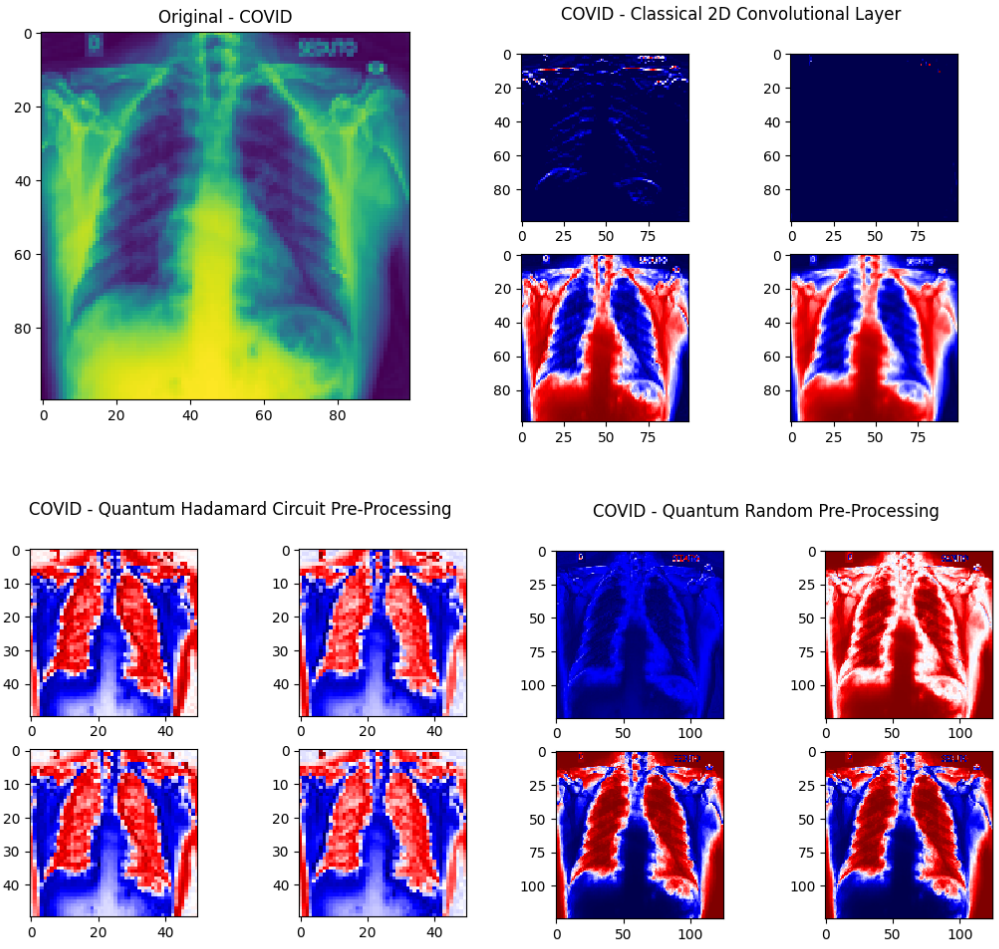


Figure 6: COVID Dataset: X-Ray Before / After Transformation

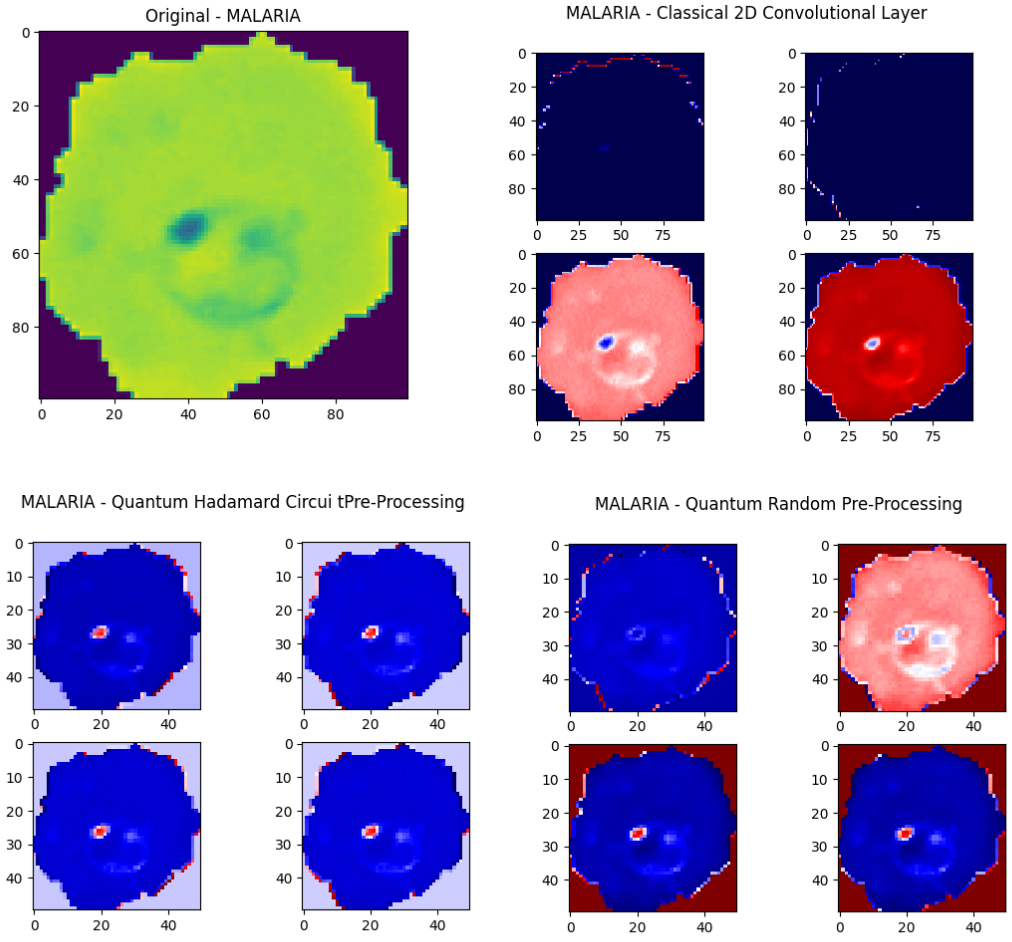


Figure 7: MALARIA Dataset: Blood Cell Before / After Transformation

As the images above show, all of the algorithms are capable of extracting information of the picture for each dataset. This shows two things: the quanvolutional method works, and the quantum gate doesn't matter (since both Random and Hadamard's Neural Networks give the same accuracy and AUC). This shows the resiliency of the neural networks, since the Random Gate will eventually produce some noise in the data, the neural network is still capable of learning the desired output).

4 Conclusion

Neural Networks is today the most important algorithm of Artificial Intelligence, and the rise of Quantum Computers had enable a new type of field called Quantum Machine Learning. Taking inspiration from the classical CNN - convolutional neural networks - a new type of algorithm was introduced: quanvolutional neural networks. [5]. Comparing the new and the old approach, the results in the section before shows that the quantum version of the convolution gives the same level of accuracy for different datasets, in the approach taken in this study: put each image to the convolutional algorithm (quantum or classical) and then using classical tools like TensorFlow for training the output.

Dataset	Classical (%)	Q-Random (%)	Q-Hadamard (%)
MNIST	98,4	98,3	96,7
IMAGENET	61,4	58,9	55,5
COVID	84,5	86,4	81,8
MALARIA	51,2	55,0	55,0

Table 6: Accuracy Results for each type of Neural Network

Note that even though a 90% accuracy wasn't reached in the IMAGENET and MALARIA datasets, the range of the accuracy are the same, which was the goal of the study. There are various types of improvements that can be made to increase accuracy, like data augmentation (make a transformation in the input image e.g. rotating it), increasing dropouts and searching for the optimal architecture for the prediction neural network.

Dataset	Time (h)
MNIST	2
IMAGENET	1
COVID	1
MALARIA	36

Table 7: Time spent in quantum pre-processing

The most important disadvantage is the time that takes to pass the image in the quantum pre-processing (a few hours for MNIST dataset, more than a day for MALARIA dataset). This happens because currently there isn't a Q-RAM or Q-GPU available, which makes computing the quantum gates through each image computationally expensive.

The conclusion is that this Quantum Algorithm have the same ability to train Neural Networks for real world application(even taking classical tools for producing the outupt). Another key thing to keep in mind is the fact that the quantum gates doesn't seem to have significance in training, because using Random Gates or Hadamard Gates produce the same accuracy.

5 References

References

[1] Quanvolutional neural networks tutorial - pennylane. `https://pennylane.ai/qml/demos/tutorial_quanvolution.html`. Accessed: 2022-06-01.

[2] Frank Arute. Quantum supremacy using a programmable superconducting processor. *Nature*, October 22 - 2019.

[3] M. Cerezo and Andrew Arrasmith. Variational quantum algorithms. *arXiv: 2012.09265*, 4 October - 2021.

[4] María Teresa García-Ordá. Detecting respiratory pathologies usingconvolutional neural networks and variationalautoencoders for unbalancing dat. *Sensors - MDPI*, February - 2020.

[5] Maxwell Henderson, Samriddhi Shakya, Shashindra Pradhan1, and Tristan Cook. Quanvolutional neural networks: Powering image recognition with quantum circuits. *arXiv: 1904.04767*, 9 April - 2019.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural network. *NIPS*, 2012.

[7] Patrick Rebentrost. Quantum support vector machine for big data classification. *arXiv: 1307.0471*, 10 July - 2014.

[8] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. *arXiv: 1807.04271*, 9 May - 2019.

A Convolutional Neural Networks

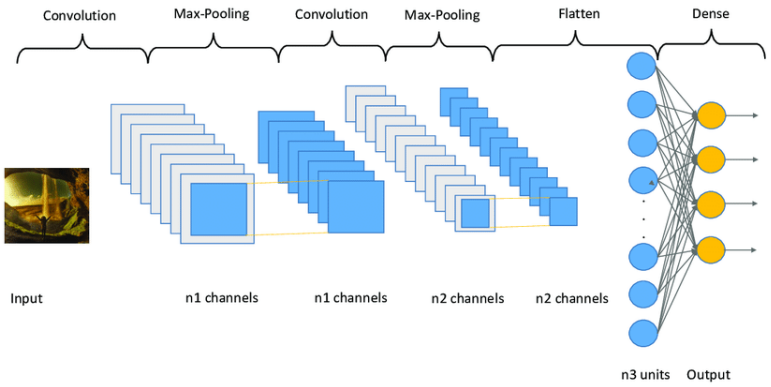


Figure 8: Classical Neural Network Architecture [4]

A Convolutional Neural Network is constituded by two stages: The **Feature Extraction** and the **Classification**. The first is created by stacking layers (repeating the computation) of Convolutional Layers, and the second is stacking layers of Dense Layers.

The Convolutional Layer is (simplifying) defined as: create a N x N grid which goes through each pixel of the input image, compute the dot product of this N x N grid with a matrix called **filters** and store in the output.

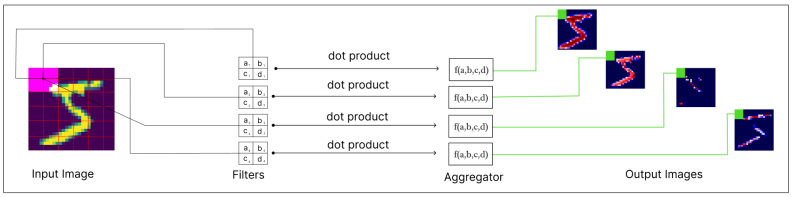


Figure 9: Classical Neural Network Computation

The idea is to have C number of filters, wich will produce C number of channels, and the **Filter** Matrices are constituded by parameters that are learned in the training phase.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} \theta_{00} & \theta_{01} \\ \theta_{10} & \theta_{11} \end{bmatrix} = \begin{bmatrix} a\theta_{00} + b\theta_{10} & a\theta_{01} + b\theta_{11} \\ c\theta_{00} + d\theta_{10} & c\theta_{01} + d\theta_{11} \end{bmatrix} = F_1(\theta_{00}, \theta_{01}, \theta_{10}, \theta_{11})$$

This step will create outputs with the following dimensions (if we use a grid of 2x2 and c filter matrices):

$$(n, n, 1) \Rightarrow (n/2, n/2, C) \tag{1}$$

After this, there are a few moew algorithms. The most common is the **Max/Average Pool**, which simply consist of creating a NxN grid, swipe through each "feature" (the convolution layer outputs features, not exactly images) and taking the maximum/average value of the grid.

$$MaxPool(\begin{bmatrix} a & b \\ c & d \end{bmatrix}) = max(a, b, c, d)$$

$$AvgPool(\begin{bmatrix} a & b \\ c & d \end{bmatrix}) = \frac{a + b + c + d}{4}$$

This downsizes the dimension, returning (using a 2x2 grid):

$$(n/2, n/2, c) \Rightarrow (n/4, n/4, C) \tag{2}$$

This concludes the convolutional algorithm: the Max/Average Pool is where the general information is stored. The idea is that the first convolutional step creates the features on the image (if the image is a cat then a filter could produce a paw, another could produce the cat ear, etc) and the Max/Average Pool would store the information in a concise way. The Max/Average Pool is where the locality independence comes from (doesn't matter if the image is rotated, shift, or if you produced some noise, if the image is of a cat, there will be a paw or a cat ear). Hence you can use data augmentation and produce more samples with the same input image.

Lastly for using the Dense Neural Network, there is a **Flatten** layer, which simply flattens the high-dimensional tensor in a 1D vector, bringing the dimension of the output to be:

$$(n/4, n/4, c) \Rightarrow \frac{c \times n^2}{16} \tag{3}$$

In Summary, the dimensions are increased and decreased in the following way (assuming we are with a grey n x n image, using a 2x2 grid for the convolutional and Max/Avg Pool steps):

Layer	Input Dimension	Output Dimension
Convolutional	(n,n,1)	(n/2,n/2,c)
Max/Avg Pool	(n/2,n/2,c)	(n/4,n/4,c)
Flatten	(n/4,n/4,c)	$\frac{(c \times n^2)}{16}$

Table 8: Dimensionality change with each step of a CNN

B Dense Neural Networks

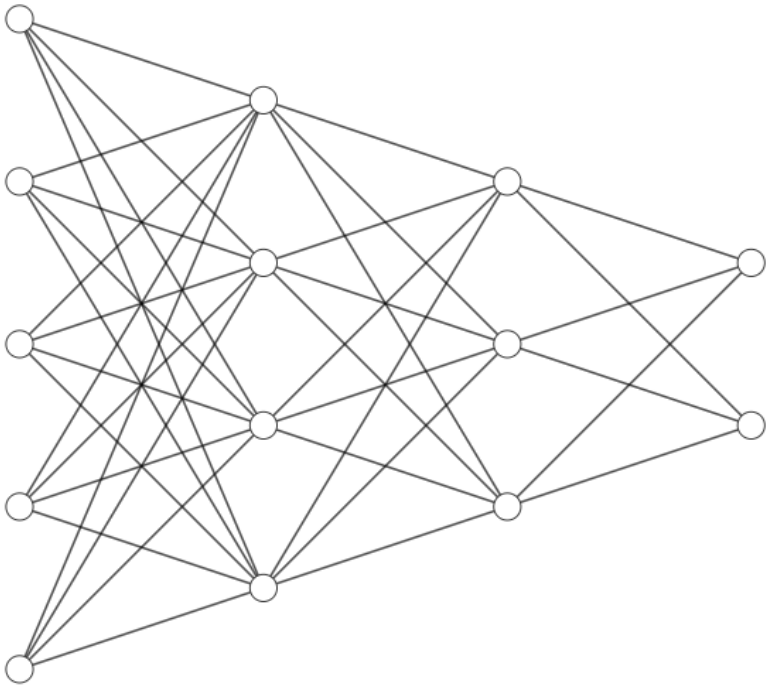


Figure 10: Architecture of Neural Network

A Neural Network is made by two steps: **Forward** pass and **Backward** pass. In the Forward pass, the next layer is a linear transformationa between the previous one, with an activation function in the end of each result. In the Backward pass, we asses the **loss** (how our prediction is wrong from the actual data) and update the parameters (weights and bias) according to an **Optimizer** (in this case Gradient Descent).

B.1 Forward Pass

Given an input $\overrightarrow{x}$ in the Input Layer, the Hidden Layer #1 has output:

$$z^1 = W^1 \cdot \overrightarrow{x} + b^1 \tag{4}$$

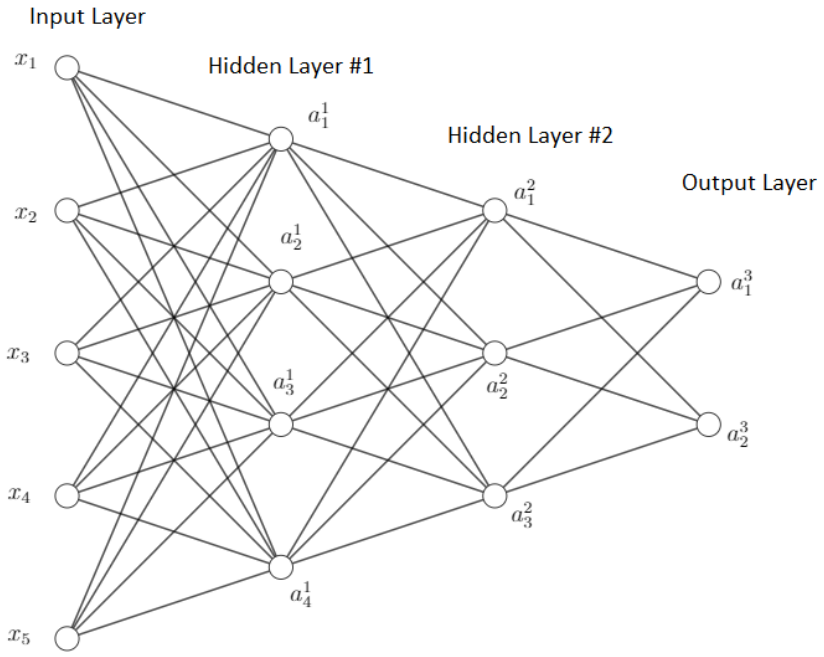


Figure 11: Dense Neural Network: Activation

where W is the **Weights** matrix, b is the **bias** vector and $\overrightarrow{x}$ is the **output** vector. Obs:

- $\overrightarrow{x}$ is a $(m, 1)$ vector
- W is a (n, m) matrix
- b is a $(n, 1)$ matrix

Then we **activate** the next layer by an Activation Funtion, to simulate nonlinear behavior (otherwise we are only doing linear regression). The final result for the next layer is:

$$a^1 = \sigma(z^1) = \sigma(W^1 \cdot \overrightarrow{x} + b^1) \tag{5}$$

Obs: The input is called $\overrightarrow{x}$, but the other layers are called **a**

So, in general, the activation of layer l is given by:

$$a^l = \sigma(z^l) = \sigma(W^l \cdot a^{l-1} + b^l) \tag{6}$$

B.1.1 Example

In our example at figure 11 we have:
Hidden Layer # 1

$$\begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ a_4^1 \end{bmatrix} = \sigma\left(\begin{bmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \\ z_4^1 \end{bmatrix}\right) = \sigma\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \cdot \begin{bmatrix} W_{11}^1 & W_{12}^1 & W_{13}^1 & W_{14}^1 & W_{15}^1 \\ W_{21}^1 & W_{22}^1 & W_{23}^1 & W_{24}^1 & W_{25}^1 \\ W_{31}^1 & W_{32}^1 & W_{33}^1 & W_{34}^1 & W_{35}^1 \\ W_{41}^1 & W_{42}^1 & W_{43}^1 & W_{44}^1 & W_{45}^1 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \\ b_4^1 \end{bmatrix}\right) \tag{7}$$

Hidden Layer # 2

$$\begin{bmatrix} a_1^2 \\ a_2^2 \\ a_3^2 \end{bmatrix} = \sigma\left(\begin{bmatrix} z_1^2 \\ z_2^2 \\ z_3^2 \end{bmatrix}\right) = \sigma\left(\begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ a_4^1 \end{bmatrix} \cdot \begin{bmatrix} W_{11}^2 & W_{12}^2 & W_{13}^2 & W_{14}^2 \\ W_{21}^2 & W_{22}^2 & W_{23}^2 & W_{24}^2 \\ W_{31}^2 & W_{32}^2 & W_{33}^2 & W_{34}^2 \end{bmatrix} + \begin{bmatrix} b_1^2 \\ b_2^2 \\ b_3^2 \end{bmatrix}\right) \tag{8}$$

Output Layer

$$\begin{bmatrix} a_1^3 \\ a_2^3 \end{bmatrix} = \sigma\left(\begin{bmatrix} z_1^3 \\ z_2^3 \end{bmatrix}\right) = \sigma\left(\begin{bmatrix} a_1^2 \\ a_2^2 \\ a_3^2 \end{bmatrix} \cdot \begin{bmatrix} W_{11}^3 & W_{12}^3 & W_{13}^3 \\ W_{21}^3 & W_{22}^3 & W_{23}^3 \end{bmatrix} + \begin{bmatrix} b_1^3 \\ b_2^3 \end{bmatrix}\right) \tag{9}$$

B.2 Backward Pass

B.2.1 Loss

In the Backward pass, we first compute our **loss** (aka the error of our output) and average along the data, to get a measure of our total uncertainty. So:

$$loss = (\frac{1}{n}) \sum_{i=0}^{n} loss(a_i^2, y_i) \tag{10}$$

where n is the dimension of the output layer. One option is to use the **Mean Squared Error** as the loss. So in our example we have:

$$loss = (\frac{1}{n}) \sum_{i=0}^{n} (a_i^2 - y_i)^2 \tag{11}$$

B.2.2 Backpropagation

The second part is to calculate how each weight and bias had influence in the loss. So we want to calculate:

$$\nabla C = \begin{bmatrix} \dfrac{\partial C}{\partial W} \\ \\ \dfrac{\partial C}{\partial b} \end{bmatrix} \tag{12}$$

We do that using the Backpropagation (chain rule). Remembering from calculus, the derivative of a composite function $f(g(x))$ can be written as:

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx} \tag{13}$$

Another thing that is going to be useful is remember how to calculate the derivative of matrices. So, given a scalar function C (even do our loss depends of $\mathbf{y}$ and $\mathbf{a}$, we can treat it as scalar), the derivative with respect to the matrix W, is given by:

$$\frac{\partial C}{\partial W} = \begin{pmatrix} \dfrac{\partial C}{\partial W_{11}} & \dfrac{\partial C}{\partial W_{12}} & \cdots & \dfrac{\partial C}{\partial W_{1m}} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial C}{\partial W_{n1}} & \dfrac{\partial C}{\partial W_{n2}} & \cdots & \dfrac{\partial C}{\partial W_{nm}} \end{pmatrix} \tag{14}$$

Similarly to the bias b:

$$\frac{\partial C}{\partial b} = \begin{pmatrix} \dfrac{\partial C}{\partial b_1} \\ \vdots \\ \dfrac{\partial C}{\partial b_n} \end{pmatrix} \tag{15}$$

Following that same logic, we have $a(z(W))$, then we start from the Output layer and go backwards to the Input layer, as follows:

$$\frac{\partial C}{\partial W} = \left(\frac{\partial C}{\partial a}\right)\left(\frac{\partial a}{\partial z}\right)\left(\frac{\partial z}{\partial W}\right) \tag{16}$$

$$\frac{\partial C}{\partial b} = \left(\frac{\partial C}{\partial a}\right)\left(\frac{\partial a}{\partial z}\right)\left(\frac{\partial z}{\partial b}\right) \tag{17}$$

For the third term, note that:

$$z^l = W^l \cdot a^{l-1} + b^l \Rightarrow \frac{\partial z^l}{\partial W^l} = a^{l-1}$$
$$z^l = W^l \cdot a^{l-1} + b^l \Rightarrow \frac{\partial z^l}{\partial b^l} = 1$$

For the second term:

$$\frac{\partial a}{\partial z} = \frac{d\sigma}{\partial z}$$

The activation function σ depends on the problem at hand. But let's suppose we are at a **classification** problem, so we'll go with the **Softmax** function, defined as:

$$\sigma(z) = \frac{e^z}{\sum_{j=0}^{n} e^{z_j}} \tag{18}$$

So the equation is:

$$\frac{\partial a^l}{\partial z^l} = \frac{d\sigma}{\partial z^l} = z^l(1 - z^l)$$

For the first term, we have two options:

1. Output Layer
For the Output Layer, we just have the loss as our main metric. So the equation is:

$$\frac{\partial C}{\partial a} = \frac{dloss}{da}$$

In our example loss is **MSE**, so:

$$\frac{\partial C}{\partial a^2} = 2(a_2^2 - y_i)$$

2. Other Layers
In other layers, we have $a^l = a^{l+1}(z^{l+1}(a^l))$
This means that the error in the layer l influences the error of layer l+1, and that's why the chain rule is so powerfull in keeping track of those links. So the equations goes as follows:

$$\frac{\partial C}{\partial a^l} = \left(\frac{\partial C}{\partial a^{l+1}}\right)\left(\frac{\partial a^{l+1}}{\partial z^{l+1}}\right)\left(\frac{\partial z^{l+1}}{\partial a^l}\right)$$

B.2.3 Example

Let's denote the following two operations:

- $\times$: element-wise multiplication, i.e. same dimensions only
- $\cdot$: dot product, i.e (m,k) $\cdot$ (k,n) = (m,n)

This means that if:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \Rightarrow AXB = \begin{bmatrix} a_{11} \cdot b_{11} & a_{12} \cdot b_{12} \\ a_{21} \cdot b_{21} & a_{22} \cdot b_{22} \end{bmatrix}$$

Analogously, if:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix} \Rightarrow A \cdot B = \begin{bmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} \end{bmatrix}$$

In our example at figure 10 we have:
Output Layer - Weights

$$\frac{\partial C}{\partial W^3} = \left(\frac{\partial C}{\partial a^3}\right)\left(\frac{\partial a^3}{\partial z^3}\right)\left(\frac{\partial z^3}{\partial W^3}\right) \tag{19}$$

$$\frac{\partial C}{\partial W^3} = \left[2\left(\begin{bmatrix} a_1^3 - y_1 \\ a_2^3 - y_2 \end{bmatrix}\right) \times \begin{bmatrix} z_1^3(1 - z_1^3) \\ z_2^3(1 - z_2^3) \end{bmatrix}\right] \cdot \left(\begin{bmatrix} a_1^2 \\ a_2^2 \\ a_3^2 \end{bmatrix}\right)^T \tag{20}$$

Output Layer - Bias

$$\frac{\partial C}{\partial b^3} = \left(\frac{\partial C}{\partial a^3}\right)\left(\frac{\partial a^3}{\partial z^3}\right) \tag{21}$$

$$\frac{\partial C}{\partial b^3} = 2\left(\begin{bmatrix} a_1^3 - y_1 \\ a_2^3 - y_2 \end{bmatrix}\right) \times \begin{bmatrix} z_1^3(1-z_1^3) \\ z_2^3(1-z_2^3) \end{bmatrix} \tag{22}$$

Hidden Layer # 2 - Weights

$$\frac{\partial C}{\partial W^2} = \left(\frac{\partial C}{\partial a^2}\right)\left(\frac{\partial a^2}{\partial z^2}\right)\left(\frac{\partial z^2}{\partial w^2}\right) \tag{23}$$

where:

$$\frac{\partial C}{\partial a^2} = \left(\frac{\partial C}{\partial a^3}\right)\left(\frac{\partial a^3}{\partial z^3}\right)\left(\frac{\partial z^3}{\partial a^2}\right) \tag{24}$$

Note that:

$$z^3 = W^3 \cdot a^2 + b^3 \Rightarrow \frac{\partial z^3}{\partial a^2} = W^3$$

So:

$$\frac{\partial C}{\partial a^2} = \left(\left[2\left(\begin{bmatrix} a_1^3 - y_1 \\ a_2^3 - y_2 \end{bmatrix}\right) \times \begin{bmatrix} z_1^3(1-z_1^3) \\ z_2^3(1-z_2^3) \end{bmatrix}\right]^T \cdot \begin{bmatrix} W_{11}^3 & W_{12}^3 & W_{13}^3 \\ W_{21}^3 & W_{22}^3 & W_{23}^3 \end{bmatrix}\right)^T \tag{25}$$

And finally:

$$\frac{\partial C}{\partial W^2} = \left(\frac{\partial C}{\partial a^2}\right) \times \left[\begin{bmatrix} z_1^2(1-z_1^2) \\ z_2^2(1-z_2^2) \\ z_3^2(1-z_3^2) \end{bmatrix}\right] \cdot \left(\begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ a_4^1 \end{bmatrix}\right)^T \tag{26}$$

Hidden Layer # 2 - Bias

$$\frac{\partial C}{\partial W^2} = \left(\frac{\partial C}{\partial a^2}\right) \times \begin{bmatrix} z_1^2(1-z_1^2) \\ z_2^2(1-z_2^2) \\ z_3^2(1-z_3^2) \end{bmatrix} \tag{27}$$

Hidden Layer # 1 - Weights

$$\frac{\partial C}{\partial W^1} = \left(\frac{\partial C}{\partial a^1}\right)\left(\frac{\partial a^1}{\partial z^1}\right)\left(\frac{\partial z^1}{\partial w^1}\right) \tag{28}$$

where:

$$\frac{\partial C}{\partial a^1} = \left(\frac{\partial C}{\partial a^2}\right)\left(\frac{\partial a^2}{\partial z^2}\right)\left(\frac{\partial z^2}{\partial a^1}\right) \tag{29}$$

Analogously as the Hidden Layer # 2 we get the influence from every layer, starting from the output layer, remembering now that:

$$z^2 = W^2 \cdot a^1 + b^2 \Rightarrow \frac{\partial z^2}{\partial a^1} = W^2$$

So the final equation is:

$$\frac{\partial C}{\partial W^1} = \left(\frac{\partial C}{\partial a^1}\right) \times \begin{bmatrix} z_1^1(1-z_1^1) \\ z_2^1(1-z_2^1) \\ z_3^1(1-z_3^1) \\ z_4^1(1-z_4^1) \end{bmatrix} \cdot \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \right)^T \tag{30}$$

Hidden Layer # 1 - Bias

$$\frac{\partial C}{\partial W^1} = \left(\frac{\partial C}{\partial a^1}\right) \times \begin{bmatrix} z_1^1(1-z_1^1) \\ z_2^1(1-z_2^1) \\ z_3^1(1-z_3^1) \\ z_4^1(1-z_4^1) \end{bmatrix} \tag{31}$$

B.3 Optimizing

For the last step, we optimize the weights accordingly to the changes calculated on the previous section. There are a few optimizers, but the two most used are: **SGD** (Stochastic Gradient Descent) and **Adam**.

SGD - Stochastic Gradient Descent
The SGD is based in the simple idea that: the gradient shows the growth direction. Because we want to **decrease** our error/loss, we simple get the opposite direction adding the minus sign. So the equation is:

$$W^l = W^l - lr \times \frac{\partial C}{\partial W^l} \tag{32}$$

$$b^l = b^l - lr \times \frac{\partial C}{\partial b^l} \tag{33}$$

where lr is the **learning rate**, a parameter ($lr < 1$) just to make sure that we don't make great jumps and 'miss' the local minima.

C Quanvolutional Neural Networks

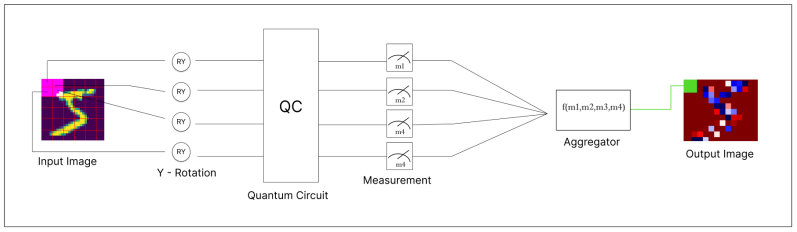


Figure 12: Quanvolutional Neural Network Computation

A quanvolutional neural network is the quantum version of the classical Convoltional Neural Network. The idea is to take the same principles such as creating NxN grids that pass through the entire input image, make some computation in this grid through quantum gates, and extract general information about it. As with any quantum neural network approach dealing with classical data, there are three stages: **Quantization**, **Computation**, **Measurement**.

C.0.1 Quantization

Quantization is the name given for transforming classical data into quantum data. The idea is to map in this case an image to a state vector

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \Longrightarrow |\psi\rangle = f(a,b,c,d)$$

The most common way to do this is using a Y-Rotation Gate, which can be viewed as a 2D rotation, which can be visualized as maping the input in to what's called Bloch Sphere

$$RY_\theta \left(\begin{bmatrix} a & b \\ c & d \end{bmatrix} \right) = \begin{bmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{bmatrix} \Longrightarrow |\psi\rangle = \begin{bmatrix} cos(\theta/2) \\ sin(\theta/2) \end{bmatrix} |0\rangle + \begin{bmatrix} -sin(\theta/2) \\ cos(\theta/2) \end{bmatrix} |1\rangle$$

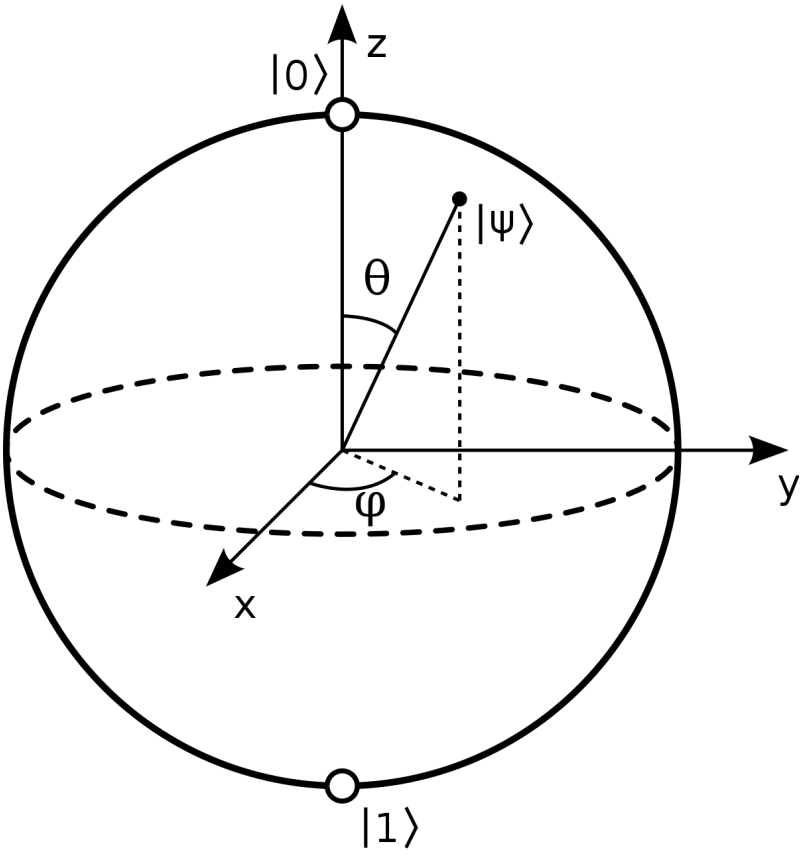


Figure 13: Bloch Sphere

where: $|0\rangle$ and $|1\rangle$ are the qubits (base of the Vector Space) defined as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad , \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

C.0.2 Computation

The Computation stage is deciding which Quantum Gate to extract the information from the input image. In the study there were used two types: a random, and the Hadamard Gate, which would be the Quantum Circuit (QC) step on the figure 12.

C.0.3 Measurement

The Measurement is the last stage, which maps the data produced by the quantuum circuit back to classical form, in a way that classical computers can use this new

information. In the original Quanvolutional Paper [5] each pixel of the image was turned into a new image, so the output would always be producing 4 channels, and the dimensions would be:

$$(n, n, 1) \Rightarrow (n/2, n/2, 4) \tag{34}$$

A small change was introduced in this study, to be more similar to the classical Convolutional Algorithm, and the Aggregator step was introduced on figure 12. Then the aggregator would simply **sum** the output produced by the Quantum Circuit. Doing this alowwed the user to choose the desired number of channels wanted, and the dimensions now are:

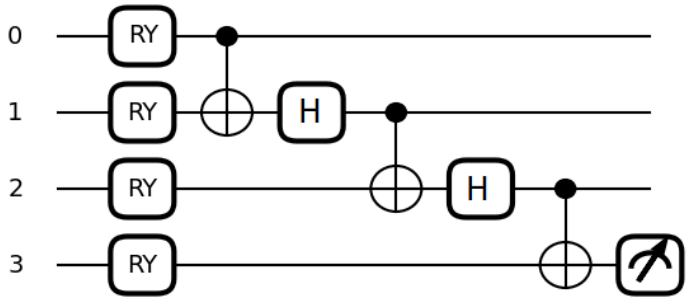
$$(n, n, 1) \Rightarrow (n/2, n/2, c) \tag{35}$$

where c (number of channels) is defined by the user

Finally, the Measurement is commonly choosed to be the expectation value of the PauliZ Gate, which is just the average of measuring the qubit (e.g. the average value of the spin of a particle).

D Quantum Circuits

Quantum Circuits is a named given by the set of Quantum Logical Gates applied in an algorithm. Each gate acts on a qubit (or a wire) an produces a different output.



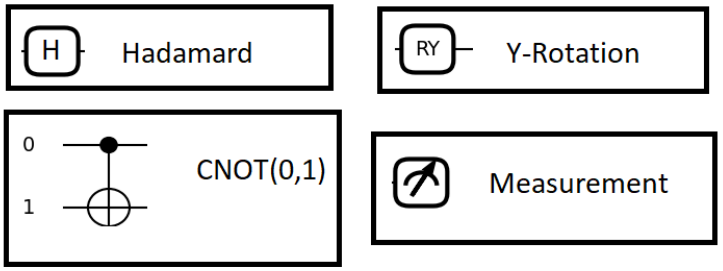


Figure 14: Quantum Circuits

Here are some of the most common used:

D.1 Hadamard Gate

The Hadamard Gate is a way of expanding a qubit in a superposition of two states, it is represented as follows:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The two possible outputs are:

$$H(|0\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$$

$$H(|1\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle$$

So we can see that the result is splitting the qubit in a superposition of its two possible states (which can be seen as an method of high dimensionality)

D.2 Rotations

Rotations is different types of rotating the qubit a long a given axis of the Bloch Sphere, so there are three of them: **X-Rotation**, **Y-Rotation**, **Z-Rotation**, defined as:

$$RX_\theta = \begin{bmatrix} cos(\theta/2) & -isin(\theta/2) \\ -isin(\theta/2) & cos(\theta/2) \end{bmatrix}$$

$$RY_\theta = \begin{bmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{bmatrix}$$

$$RZ_\theta = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$

The two possible results for each Rotation is given by:

$$RX_\theta(|0\rangle) = \begin{bmatrix} cos(\theta/2) & -isin(\theta/2) \\ -isin(\theta/2) & cos(\theta/2) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = cos(\theta/2)\,|0\rangle - isin(\theta/2)\,|1\rangle$$

$$RX_\theta(|1\rangle) = \begin{bmatrix} cos(\theta/2) & -isin(\theta/2) \\ -isin(\theta/2) & cos(\theta/2) \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = -isin(\theta/2)\,|0\rangle + cos(\theta/2)\,|1\rangle$$

$$RY_\theta(|0\rangle) = \begin{bmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = cos(\theta/2)\,|0\rangle + sin(\theta/2)\,|1\rangle$$

$$RY_\theta(|1\rangle) = \begin{bmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = -sin(\theta/2)\,|0\rangle + cos(\theta/2)\,|1\rangle$$

$$RZ_\theta(|0\rangle) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = e^{-i\theta/2} |0\rangle$$

$$RZ_\theta(|1\rangle) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = e^{i\theta/2} |1\rangle$$

The Y-Rotation is the most used because it is simply a 2D Matrix Rotation, and the other two axis has either complex terms (X-Rotation), or zeros (Z-Rotation).

D.3 Pauli Gates

Pauli Gates are a collection of gates that comes from the Pauli Matrices. They are π rotations on a given axis. They come in three types: **PauliX**, **PauliY** and **PauliZ**. The PauliX is also the quantum equivalent of the NOT classical gate.

$$PauliX = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$PauliY = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

$$PauliZ = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

D.4 CNOT Gate

The CNOT gate is a gate used to entangle or disentangle Bell States (used for entanglement, which provide the hability of multiple computation on fewer steps). It acts on two qubits, and is usually used with a Hadamard Gate (for creating a

Bell State). It is also a quantum version of the classical CNOT logical gate (flips the second bit if the first one is 1). It is defined as:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Since it acts on two qubits, it has 4 possible outputs. The inputs are now the basis of 4D vectors:

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad , \quad |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad , \quad |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The possible outputs are the following:

$$CNOT(|00\rangle) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = |00\rangle$$

$$CNOT(|01\rangle) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |01\rangle$$

$$CNOT(|10\rangle) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |11\rangle$$

$$CNOT(|11\rangle) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle$$