Quantum Vision Transformer: a study on the Quantum Orthogonal Transformer

Giovanni Maccioni
Università degli Studi di Firenze
Intelligenza Artificiale
giovanni.maccioni1@edu.unifi.it

Abstract

This work is a study on Quantum Vision Transformers from Cherrat et al. *[1]. The main idea is to implement the attention mechanism in a quantum fashion, to obtain advantages in computation compared to the classical counterpart.*

They introduced two novel quantum vision transformer architecture, one in which the attention is computed totally using quantum circuits and one in which the attention is partially computed in a classical manner and partially in a quantum way.

In [1] they sustain that, using such architectures, we have no difference in classification perfomance compared to the classical architectures, while the number of parameters drops. To prove that, they use the classification of the MedMNIST dataset as a benchmark.

We will study and implement the Quantum Orthogonal Transformer in particular, in which the attention is partially computed using quantum circuits, and use it to classify the MNIST dataset.

1. Introduction

The Transformer [5] architecture has become popular in recent years in Deep Learning. This architecture has been fundamental to achieve great results in the Natural Language Processing (NLP) field, making the development of Large Language Models (LLM) possible. The great achievments in NLP, pushed reasearchers to adopt this architecture for Computer Vision tasks as well; so in 2020 Vision Transformers [2] were introduced.

The attention mechanism is the fundamental module of these models. But in terms of computational complexity it is also its weakness: it is qudratic in the dimension of the input.

In Cherrat *et al*. [1] they focus on the attention layer implementing it with trainable quantum circuits. In particular to build the Quantum Orthogonal Transformer they use a Vector Loader circuit to encode classical vectorized data in a quantum state and the Orthogonal Quantum Layer to perform Vector-Matrix and Vector-Matrix-Vector product, needed to compute the attention.

In the following sections we will:

1. Revisit briefly how the Vision Transformer works with a focus on the attention mechanism and the simplification adopted in [1] to have simplier quantum circuits;

2. Introduce the quantum tools we need to build the circuits, the circuits themself and how they work;

3. Highlight the key elements of the implementation of the quantum circuits and the results obtained;

2. Vision Transformer

We can see an overview of the Vision Transformer used for classification tasks in *Figure* 1. As we can see, the first step is to divide the input image in patches. Then we need to embed those patches into vectors, done with a learnt embedding. We need also to add positional encoding.

The positional encoding is needed to add order information to the embedding of the patches because the input is considered as a set, not as a sequence. Thus, if we divide the image in n patches, we will have n vector as a result.

To perform classification tasks, an extra vector is added to the set of n embedded patches(in red in *Figure* 1) that is called class token. So in the end the input to the Transformer Layer is a set of $n + 1$ vectors.

These n + 1 embeddings will go in input to the sequence of L Transformer Layers, that we can see in details in *Figure* 2. In these layers, among other kind of processing, the patches will go through the attention layer.

In the end, the class token, thanks to the attention layer, will contain all the information we need to distinguish the class the image belongs to. So it will go in input to the final MLP, giving us in output the probabilities that the image belongs to the classes considered for the task.

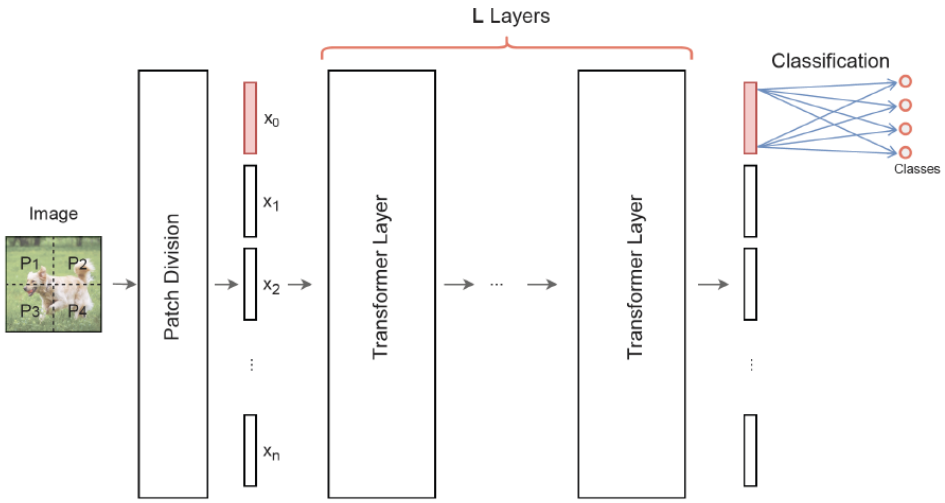

Figure 1. Vision Transformer overview

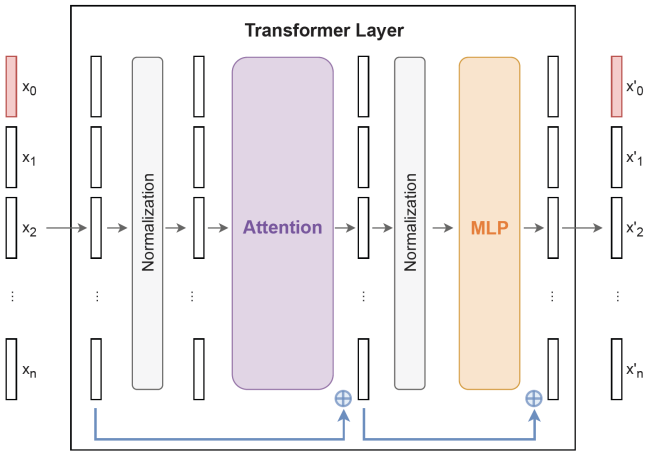

Figure 2. Details of the Transformer Layer

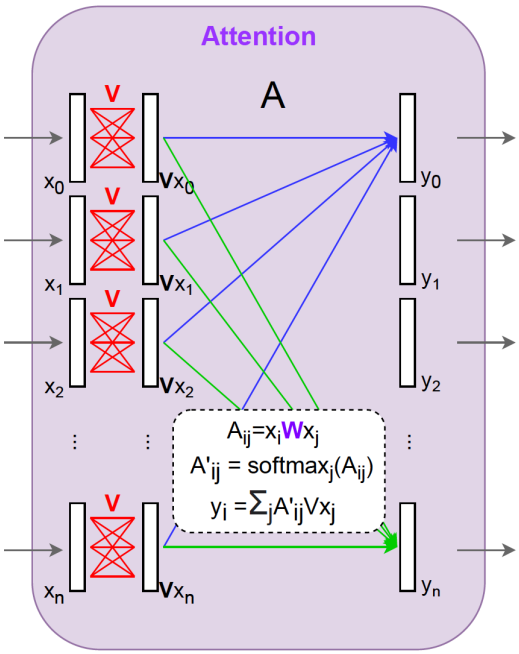

Figure 3. Details of the simplified Attention computation.

2.1. Attention Mechanism

Let's consider a set, where each of its elements have a **Value** and we can access them by consulting a corresponding **Key**. Let's say we have a request, a **Query**, to retrieve an element, under that request conditions, from this set. We can find the element that answers better to the request by confronting the Query with the Keys. Better the element answers to the request, higher the **similiarity** between the Query and the Key will be. Expanding this concept we can see those similarity scores as weights, and the answer to the Query as a weighted average of the values of each element in the set. From this the name attention as to how much each element in the set has to be considered given the request.

The attention applied inside the Transformer architecture is called self-attention. In self-attention, each sequence element provides a key, value, and query.

We can summarize the self attention in the equation below:

$$Attention(Q, K, U) = softmax(QK^T)U \qquad (1)$$

where $Q, K, U \in \mathbb{R}^{n+1 \times d}$ are respectively the Query, Key and Value matrices. The similarities are computed by the $Q \times K^T$ product; the softmax function is used to obtain the final weights, that will sum up to 1, that will multiply the

value matrix U.

In the Transformer architecture we use the Multihead self Attention in which we have multiple attention computed in parallel, that are then concatenated. In [1] they used a Singlehead Attention computation to ease up the quantum implementation and so we will use this simplification as well.

2.2. Simplified Attention Computation

The attention has to be trainable to be inserted in a neural network. So we can express the attention expliciting the trainable parameters:

$$\begin{aligned} &Attention(QW^Q, KW^K, UV) = \\ &softmax(QW^Q(KW^K)^T)UV \end{aligned} \tag{2}$$

where $W^Q, W^K, V \in \mathbb{R}^{dXd}$ are weight matrices.

The simplification adopted by Cherrat *et al*. [1] is to use two trainable matrix instead of three to have fewer quantum circuits in the final architecture. In particular W^Q, W^K are merged into one, W. So the attention will be computed as:

$$softmax(QWK^T)UV \tag{3}$$

To initialize Q, K, U in the first Transformer layer, the Q, K, V matrices are initialized with the input $X \in \mathbb{R}^{n+1Xd}$ (from this we have the name self attention). Thus we obtain:

$$softmax(XWX^T)XV \tag{4}$$

The attention computation in this setting is shown in *Figure* 3. To be precise in the subsequent Transformer layer the input sequence will be a transformation of the orginal one.

2.3. On the complexity of Attention

Let's define $A' = softmax(XWX^T)$. To compute this matrix, we have a computational complexity of $O(n^2d)$, so quadratic in the number of patches and to compute The multiplication UV we have a complexity of $O(nd^2)$. So the total complexity of the attention is given by $O(n^2d + nd^2)$.

In quantum circuits the computational complexity will be measured in the number of parametrised gates used; we will see that the complexity goes down to $O(dlog(d))$.

3. Quantum Tools

In order to compute the attention, in our case, using quantum circuits we will need two circuits one to compute the Matrix-Vector product VX and one to compute the Vector-Matrix-Vector product XWX^T. Both of these circuits can be build using a Vector loader and a Quantum Orthogonal Layer: the composition of these modules allow us to obtain the circuits mentioned before.

Being that the task at end requires to load classical data we will need also an encoding strategy. The encoding they choose is the Unary Amplitude Encoding.

In the sequent subsections we will introduce all these tools focusing on the ones actually used in this work. But first, in the following subsection, we will describe the gate that is the basic building block for all the circuits mentioned above.

But first we will introduce the RBS gate, which is the basic gate that is used to build the two circuits mentioned above

3.1. Reconfigurable Beam Splitter (RBS) gate

The Reconfigurable Beam Splitter (RBS) gate is a two qubit gate, that is parametrizable with one angle $\theta \in [0, 2\pi]$

$$RBS(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$RBS(\theta) : \begin{cases} |01\rangle \mapsto \cos\theta\,|01\rangle - \sin\theta\,|10\rangle \\ |10\rangle \mapsto \sin\theta\,|01\rangle + \cos\theta\,|10\rangle \end{cases} \tag{6}$$

We can think of this gate as a rotation in the two-dimensional subspace spanned by the basis $\{|01\rangle, |10\rangle\}$, while it acts as the identity in the remaining subspace $\{|00\rangle, |11\rangle\}$. A possible decomposition of this gate is shown in *Figure* 4. We can see that this gate will preserve the number of ones and zeros in any basis state, a property that will be of major importance to build the circuits we need.

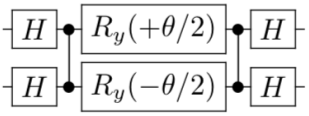

Figure 4. A possible decomposition of the $RBS(\theta)$ gate.

3.2. Unary Amplitude Encoding

To perform quantum computation and in particular machine learning tasks, we need to load classical data into the quantum circuit using an encoding strategy. They choosed to use the unary amplitude encoding where a vector $x = (x_0, \cdots, x_{d-1})$ is loaded in the quantum state:

$$|x\rangle = \frac{1}{\|x\|} \sum_{i=0}^{d-1} x_i\,|e_i\rangle \tag{7}$$

where $|e_i\rangle$ is the quantum state with all qubits in 0 except the i^{th} one in state 1. The number of qubits needed to

store the data in the circuit is the dimension of the vector x, d, due to the use of RBS gates; usually this encoding strategy would make the number of qubits needed to encode x smaller than its dimension.

3.3. Vector Loader

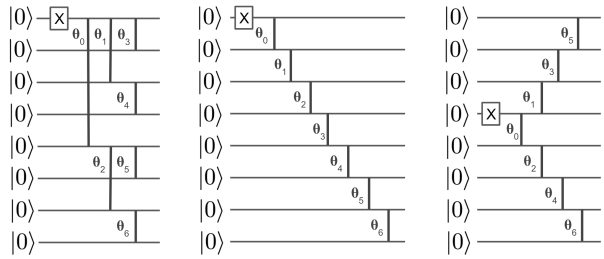

Figure 5. The three vector loaders. From lefto to right the Parallel, the Diagonal and the semi-diagonal loaders

As mentioned above, the Vector Loader circuits are the ones that will encode the vectors in a quantum state to perform the following computations.

There are three different implementation of this quantum circuit considered in [1] . These variations are characterized by the same number of RBS gates, but with different disposition and circuit depth. These circuits are built with different qubit connectivity in mind (*Table* 1). In *Figure* 5. we can see these three circuits.

To encode a vector of dimension d, we will need $d-1$ parameters in the quantum space. So given a vector $x = (x_1, \cdots, x_d)$ we have to compute in a classical way the $d-1$ parameters for the $d-1$ RBS gates that compose the vector loader. The algorithm to do that depends on the topology of the circuit.

We will consider two of the vector loader circuits, the Diagonal and the Parallel.

3.3.1 Diagonal Vector Loader

The diagonal vector loader (see *Figure* 5) is built considering a Nearest-Neighbour connection between qubits and has a circuit depth of $O(d)$.

The circuit starts in the all $|0\rangle$ state and flips the first qubit using an X gate, in order to obtain the unary state $|10\cdots0\rangle$. Then a cascade of d - 1 RBS gates allow to create the state $|x\rangle$ using a set of d - 1, angles $\theta_0, \cdots, \theta_{d-2}$.

To upload the classical data into the quantum circuit we will use one qubit per feature of the data.

Using Eq.(6), we will choose the angles such that, after the first RBS gate of the loader, the qubits would be in the state

$$x_0 |100\cdots\rangle + \sin(\theta_0) |010\cdots\rangle$$

and after the second one in the state

$$x_0 |100\cdots\rangle + x_1 |010\cdots\rangle + \sin(\theta_0)\sin(\theta_1) |001\cdots\rangle$$

and so on, until obtaining $|x\rangle$ as in Eq.(7). Note that all the computation are done considering the x vector to have unitary norm.

The whole algorithm to compute the parameters for the Diagonal Loader is described in *Algorithm 1*[1].

Algorithm 1: Compute parameters for the diagonal vector loader

Data: $x = (x_0, \cdots, x_{d-1})$
Result: $\theta_0, \cdots, \theta_{d-2}$
$\theta_0 \leftarrow \arccos(x_0)$;
$i \leftarrow 1$;
while $i \neq d-1$ **do**
  $\theta_i \leftarrow \arccos\left(\frac{x_i}{\prod_{j=0}^{i}\sin(\theta_j)}\right)$;
  $i \leftarrow i+1$;
end

3.3.2 Parallel Vector Loader

The Parallel Vector loader (*Figure* 5) is thought for quantum computer with an All-to-All qubit connectivity: as we can see the RBS gates are not connected only between neighbor qubits, but there are connections between far away ones.

Respect to the Parallel loader, we can see that the depth of the circuit is lower, $O(log(d))$.

To compute the parameters for this Loader in Johri *et al.* [3] they take advantage of tree-like structure of this circuit to optimize the computation; refer to [3] for the optimal algorithm to compute the parameters;

The algorithm to compute the parameters is detailed in *Algorithm 2*[1].

3.4. Quantum Orthogonal Layer

The Quantum Orthogonal layers are also circuits of d qubits, built with RBS gates. This guarantees that if the input to a quantum orthogonal layer is a vector in unary amplitude encoding, the output will be another vector in unary amplitude encoding (for the property of the RBS gates mentioned before). If we consider this circuit as a matrix W, each of its elements, W_{ij}, can be seen as the weight of all paths that map state $|e_j\rangle$ to state $|e_i\rangle$.

Like the Vector Loader circuits, there exist various possibilities for building a Quantum Orthogonal Layer. In their work they compare three of them: the Pyramid, the Butterfly and the X Orthogonal Layers. The differences in number of parameters and in depth are shown in *Table* 1

[1]The algorithms are written in pseudo code as close as possible to the implementation used

[1]The algorithms are written in pseudo code as close as possible to the implementation used

Algorithm 2: Compute parameters for the parallel vector loader

Data: $x = (x_0, \cdots, x_{d-1})$
Result: $\theta_0, \cdots, \theta_{d-2}$
$i \leftarrow 0$;
while $i \neq \frac{d}{2}$ **do**
  $r_{\frac{d}{2}+i-1} \leftarrow \sqrt{{x_{2i+1}}^2 + {x_{2i}}^2}$;
  $i \leftarrow i + 1$;
end
$i \leftarrow \frac{d}{2} - 2$;
while $i \gtrsim 0$ **do**
  $r_i \leftarrow \sqrt{{r_{2i+2}}^2 + {r_{2i+1}}^2}$;
  $i \leftarrow i - 1$;
end
$i \leftarrow 0$;
while $i \neq \frac{d}{2}$ **do**
  $\theta_i \leftarrow \arccos(\frac{r_{2i+1}}{r_i})$;
  $i \leftarrow i + 1$;
end
$i \leftarrow 0$;
while $i \neq \frac{d}{2}$ **do**
  if x_{2i+1} *is positive* **then**
    $\theta_{\frac{d}{2}+i-1} \leftarrow \arccos(\frac{x_{2i}}{r_{\frac{d}{2}+i-1}})$;
  else
    $\theta_{\frac{d}{2}+i-1} \leftarrow 2\pi - \arccos(\frac{x_{2i}}{r_{\frac{d}{2}+i-1}})$;
  end
end

There exist many possibilities for building a Quantum Orthogonal Layer, each with different properties. In [1] they compare three of them: the Pyramid, the Butterfly and the X Orthogonal Layers. The differences in number of parameters and in circuit depth are shown in *Table* 1. Also we can see the circuits themselves in *Figure* 6

Differently from the Vector Loaders, the parameters of these circuits will be trainable, so no algorithm to compute the parameters is needed, they will be updated by the training procedure.

In these work we used two of the Orthogonal Layers: The **Pyramid** and the **Butterfly** layer.

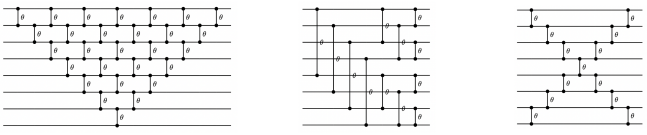

Figure 6. From left to right, the Pyramid, the Butterfly and the X layer.

Circuit	Hardware Connectivity	Depth	# Gates
Pyramid	NN	$2d-3$	$\frac{d(d-1)}{2}$
X	NN	$d-1$	$2d-3$
Butterfly	All-to-all	$log(d)$	$\frac{d}{2}log(d)$

Table 1. Comparison of different Quantum Orthogonal Layer circuits with d qubits. NN stands for Nearest Neighbor connectivity.

3.5. The circuits for the Quantum Attention Block

To build the Quantum Attention Block, as previously stated we need a circuit for the Vector-Matrix product and one for the Vector-Matrix-Vector product.

3.5.1 Matrix-Vector Circuit

As we can see in *Figure* 7, the circuit consist of a vector loader, that encodes a vector, x_i, and an Ortoghonal layer, V. Measuring this circuit will give us the result of the Vector-Matrix multiplication, Vx_i. In particular, for the property of the RBS gates, only the amplitudes of the states in which only one qubit is one will be potentially different from 0.

In the optic of the vision transformer, this operation will be repeated for each element of the set of patches in input. To perform this feat, a number of circuits equal to the number of patches is used to have as a result the product Vx_i for each patch.

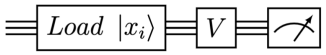

Figure 7. An high level scheme of the Matrix-Vector circuit

3.5.2 Vector-Matrix-Vector Circuit

The Vector-Matrix-Vector circuit is similiar to the previous one (*Figure* 7). Like the circuit described above, it consists of a Vector Loader and an Orthogonal Layer, with the addition of an adjoint Vector Loader after.

This configuration allow us to obtain the attention coefficients as the probability of measuring one in the first qubit. To be precise the probability, and so the attention coefficient obtained, would be ${A_{ij}}^2 = |x_i W x_j|^2$. A method to obtain also the sign of the coefficients exists, but in this study we worked with positive coefficient, in order not to add compllexity to the circuits. The suqare root is computed classically. Also here we will need as many circuits as all the possible combinations of two patches, the square number of patches.

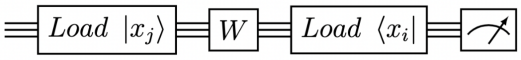

Figure 8. An high level scheme of the Vector-Matrix-Vector circuit.

3.5.3 Consideration on the complexity of the Quantum attention

As stated before, the complexity of the attention modukle in the classical case is $O(n^2d + nd^2)$, where n is the number of inputs(patches) and d is their dimension.

Let's take the best case scenario in terms of complexity for the Quantum Attention Block choosing the Parallel Vector Loader and the Butterfly layer. The first has a complexity of $O(log(d))$ and the second of $O(log(d))$. The total complexity will be $O(log(d))$, in terms of circuit depth. Also the number of trainable parameters is lower in the Quantum Attention $O(dlog(d))$ against $O(2d^2)$ in the Classical Attention(Single Headed, simplified with only two trainable matrices).

But to compute the Quantum Attention there is also an overhead to compute the parameters for the vector loader

3.6. Quantum Neural Networks

A Quantum Neural Network is a parametrised quantum circuit in which we can divide the parameters into two parts: the inputs and the weights. These circuits are usually composed by two sub-circuits:

- **Feature map**: circuit of which the parameter are the inputs;
- **Ansatz**: circuit of which the parameter are the weights; as in classical neural networks, the weights are the trainable parameters.

In our case the feature map will be the Vector Loader (and the Adjoint one), while the Ansatz will be the Orthogonal Layer. As in standard neural networks, we also need a way to estimate gradients.

3.6.1 Gradient Computation

The algorithm that will be used to compute the gradients for the quantum circuits parameters is the Parameter-shift Rule: the basic idea is that, if we consider the circuit as a function, the derivative can be computed using the same function, but with shifted parameters. So the same circuit is used to compute both the function and its derivatives, that finally are used to estimate the analytic gradients.

Those estimations will be used for the forward and backpropagation needed to train neural networks.

Although in Mathur *et al*. [4] an ad hoc algorithm for the forward and backpropagation computation is developed, it would have been a too heavy computation to perform.

4. Implementation

This study is implemented using PyTorch and the Qiskit library. In this section we will see some details of the implementation needed for this work. In particular we will see what two implementations of the Quantum Attention Blocks, but first we will introduce some of the tools used from the Qiskit library.

4.1. SamplerQNN and Torch Connector

SamplerQNN(`https://qiskit-community.github.io/qiskit-machine-learning/stubs/qiskit_machine_learning.neural_networks.SamplerQNN.html#qiskit_machine_learning.neural_networks.SamplerQNN`) is referred as a neural network in the Qiskit machine learning library. It is a wrapper for a parametrised circuit in order to support the forward and backward propagation. Given a parametrised circuit, we need to specify which of the parameters are the inputs and which the weights to be optimized. The output of this network will be the estimated quasi-probabilities for each quantum state.

To make the SamplerQnn compatible with Pytorch, we used the **Torch Connector** (`https://qiskit-community.github.io/qiskit-machine-learning/stubs/qiskit_machine_learning.connectors.TorchConnector.html`)wrapper, to which we pass the initialized SamplerQNN. In this way the qiskit neural networks becomes basically a PyTorch module, compatible with the automatic differentiation.

4.2. Quantum Attention Block

To build the quantum attention block, as mentioned in the previous sections we use one of the Vector Loader Circuits and one of the Orthogonal layers.

We decided to pair together the circuits that require the same qubit connectivity.

We can see the resulting two Quantum Attention Blocks in *Figure* 9 10 for the one built with the Diagonal vector loader and the Pyramid Layer, and in *Figure*11 12 for the one built with the ParallelVector Loader and the Butterfly layer. The first considers a Nearest-Neigbor connectivity between qubits, the other All-to-All.

In all those figures, the *sigmas* are the weights, (the trainable parameters) while the *thetas* and the *phi* are the parameters to be computed classically to load the vectors in input.

The number of qubits of those circuits is 4, as it is the dimension of the input vectors. We can also notice that the thetas and phis are of length 3 (d - 1 parameters, where d is 4).

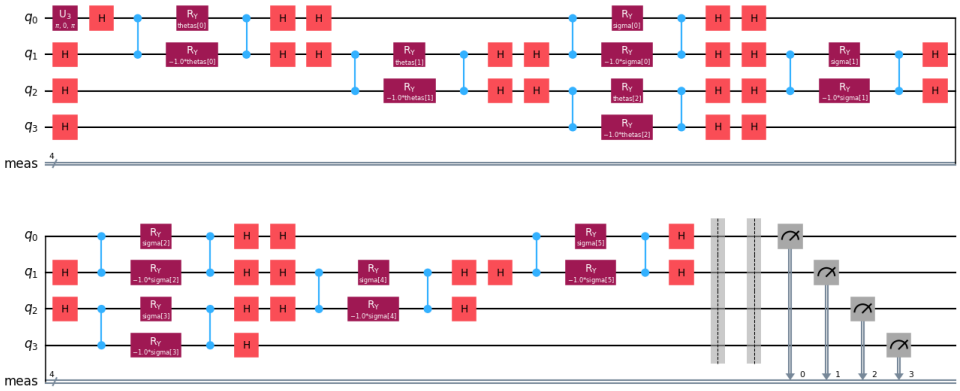

Figure 9. Diagonal-Pyramid Vector-Matrix circuit

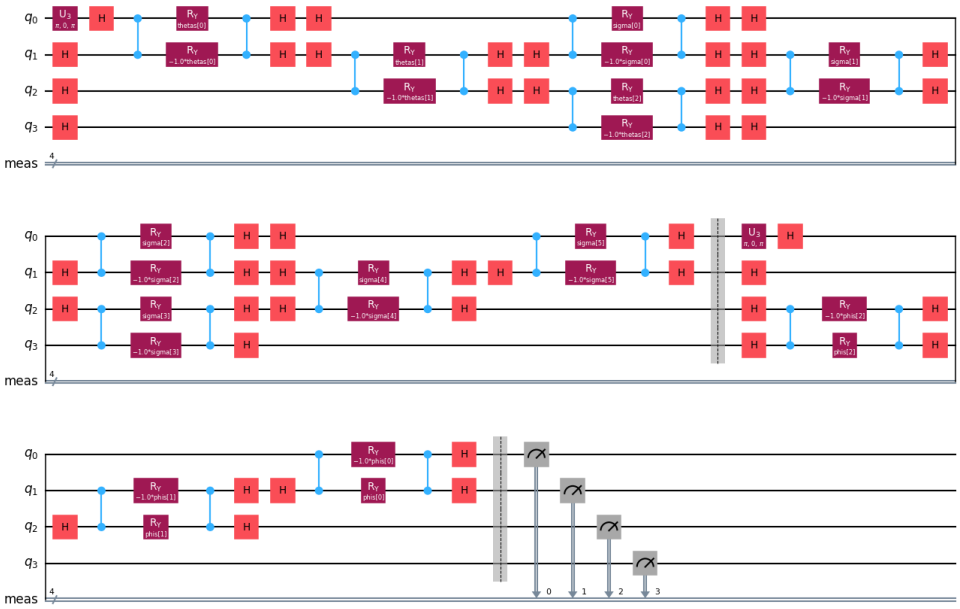

Figure 10. Diagonal-pyramid Vector-Matrix-Vector circuit

5. Experiments

Due to heavy computation requirments, the experiments are conducted under a more than restrictive setting:

- The dataset choosen is MNIST, limited to classes 0 and 1 (about 10.000 samples).
- The two Vision Transformers were trained for 5 epochs.
- The images are divided in 4 patches 14x14, giving us 4 vectors per-image as an input to the Transformer.
- The image patches are embedded in 4-dimensional vectors, to allow us using 4-qubit circuits.

Choosing only two classes of MNIST, therefore reducing also the number of samples, was necessary to have a time per epoch that wasn't to high. In fact using the whole dataset would have raised this time to more than 24 hours. With this restriction we reach a time per epoch of about 6 hours on average.

Following the first restriction, the two models were trained for 5 epochs, so 30 hours on average of total training time each.

The last two restrictions are connected to the heavy com-

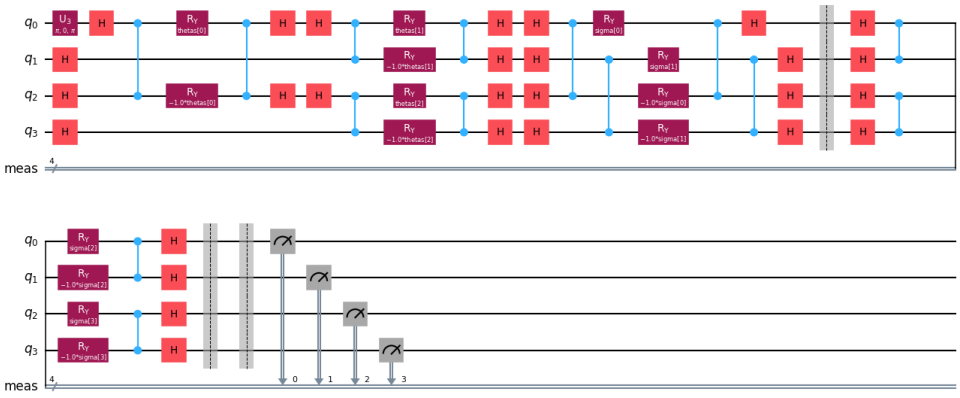

Figure 11. Parallel-Butterfly Vector-Matrix circuit

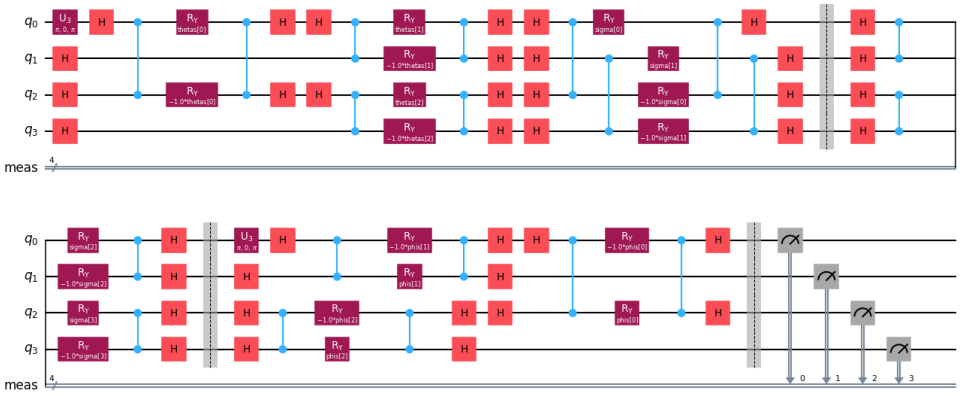

Figure 12. Parallel-Butterfly Vector-Matrix-Vector circuit

putational demands required to simulate quantum circuits on classical hardware. The number of image patches are limited because they determine how many circuits would run at training time, while the embedding size is limited because is directly linked with the number of qubit that would compose the circuits we require. The circuits were ran using a state vector simulator.

As for the optimizer Adam was used with learning rate $3e^{-3}$ for the Vision Transformer with the Diagonal-Pyramid Quantum Block, with $3e^{-4}$ for the other due to numeric instability during training.

6. Results

We report the training-accuracy plots for these two models in *Figure* 13. The accuracy is computed on unseen images with labels 0 and 1.

As we can see for both the two models, in 5 epochs we reach an high level of accuracy. To test the real capability of these models, more testing is required and these results are not valid for a direct comparison in performance with other models. The objective of these experiments was to show that this two models implemented do work and with further experiments we could assess the real models' capability.

Figure 13. Train Accuracy Plots

References

[1] E. A. Cherrat, I. Kerenidis, N. Mathur, J. Landman, M. Strahm, and Y. Y. Li. Quantum vision transformers. *arXiv preprint arXiv:2209.08167*, 2022.

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[3] S. Johri, S. Debnath, A. Mocherla, A. Singk, A. Prakash, J. Kim, and I. Kerenidis. Nearest centroid classification on a trapped ion quantum computer. *npj Quantum Information*, 7(1):122, 2021.

[4] N. Mathur, J. Landman, Y. Y. Li, M. Strahm, S. Kazdaghli, A. Prakash, and I. Kerenidis. Medical image classification via quantum neural networks. *arXiv preprint arXiv:2109.01831*, 2021.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.