Brain Tumor Detection Using Quantum Neural Network

A thesis
submitted in partial fulfillment of the requirement for the Degree of Master of Technology in Computer Technology of
Jadavpur University

By
Ekata Kumari

Registration No: 149842 of 2019-2020
Exam Roll No: M6TCT22009

Under the Guidance of
Debotosh Bhattacharjee
Professor
Department of Computer Science and Engineering
Jadavpur University Kolkata-700032
India
2022

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Certificate of Recommendation

This is to certify that the dissertation entitled "**Brain Tumor Detection Using Quantum Neural Network**" has been carried out by Ekata Kumari (University Registration No.: 149842 of 2019-22, Examination Roll No.: M6TCT22009) under my guidance and supervision and be accepted in partial fulfillment of the requirement for the Degree of Master of Technology in Computer Technology. The research results presented in the thesis have not been included in any other paper submitted for the award of any degree in any other University or Institute.

…….

Debotosh Bhattacharjee
Professor
Department of Computer Science and Engineering
Jadavpur University, Kolkata-700032

Countersigned

…….

Anupam Sinha
Professor and Head, Department of Computer Science and Engineering,
Jadavpur University, Kolkata-700032

…….

Chandan Mazumder
Dean, Faculty of Engineering and Technology
Jadavpur University, Kolkata-700032

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Certificate of Approval*

This is to certify that the thesis entitled — "**Brain Tumor Detection Using Quantum Neural Network"** is a bonafide record of work carried out by **Ekata Kumari** in partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Technology in the Department of Computer Science and Engineering, Jadavpur University during the period of July 2019 to June 2022. It is understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed, or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

……

……..…Signature of Examiner 1

Date:

……

…..……Signature of Examiner 2

Date:

*Only in case the thesis is approved

FACULTY OF ENGINEERING AND TECHNOLOGY

JADAVPUR UNIVERSITY

Declaration of Originality and Compliance of Academic Ethics

I hereby declare that this thesis entitled —"**Brain Tumor Detection Using Quantum Neural Network"** contains a literature survey and original research work by the undersigned candidate, as part of her Degree of Master of Technology in Computer Technology.

All information has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name: Ekata Kumari
Registration No: 149842 of 2019-2020
Exam Roll No: M6TCT22009
Thesis Title: **Brain Tumor Using Quantum Neural Network**

……..
Signature with Date

Acknowledgment

I am pleased to express my gratitude and regard to my Project Guide Professor Debotosh Bhattacharjee, Department of Computer Science and Engineering, Jadavpur University, for his invaluable guidance, constant patience, and encouragement during the period of my project.

I would also like to thank the Head of the Computer Science and Engineering Department, Jadavpur University, and my family members for their constant support and motivation during the tenure of this course.

Any omission of acknowledgment does not reflect my lack of regard or appreciation.

…….

Ekata Kumari

Registration No: 149842 of 2019-2020

Examination Roll No: M6TCT22009

Department of Computer Science and Engineering

Jadavpur University

Abstract

This engineering thesis includes the approaches to implementing the Quantum Neural Networks to classify the Brain tumor on the MRI Brain Tumor dataset and the classification of digits from the MNIST database. The different architectures of Quantum Neural Network are trained on this dataset and compared in terms of prediction performance.

Contents

Chapter1

Introduction

This thesis aims to create quantum neural networks that perform tasks similar to those of classical neural networks and simulate their operation on a classical computer. Since the problems related to quantum computing are not included in the scope of studies, the thesis attempts to deepen the knowledge about quantum algorithms and their possible applications in machine learning.

The technical part of the thesis is related to attempts to solve real (non-synthetic) problems through quantum neural networks (QNN) and their comparison with classical equivalents, i.e., neural networks (NN). The whole implementation of the system simulating the operation and comparison of the QNN with classic NN and comparison of different QNN models, on the example of the data image recognition(classification) tasks, is prepared using the Python programming language. The code utilizes machine learning libraries dedicated to neural and quantum neural computing, Pytorch and Qiskit, respectively.

The work is organized as follows. A description of neural networks as a part of classical machine learning is provided in Chapter 2. Next, the adopted approach to extending classical algorithms into their quantum counterparts is introduced. This topic makes Chapter 3. Chapters 4 and 5 describe experimental frameworks on two different datasets for image recognition. These chapters also include the results of the conducted experiments, providing a quantitative comparison of different quantum neural networks. Chapter 6 concludes the thesis.

Chapter 2

Classical neural networks

2.1 Concept of neural network

The human brain's cognitive part comprises 86.06 ± 8.12 billion cells called *neurons* [4]. The total number of synapses is estimated to be 10^{15} [16]. The division of both numbers gives us the average number of single-neuron connections with other neurons scattered around the brain, roughly 10 thousand. The structure of a single neuron includes the body and parts called dendrites and an axon. The many aforementioned connections are possible because the dendrites and the axon are highly branched. The task of a neuron is to receive electric signals from other neurons and, after processing, send one signal to the latter neurons. The neural cell receives signals through dendrites. Dendrites are connected with other neurons, and these connections called synapses are dynamically formed or reduced during one's lifetime. The strength and maintenance of such a connection depend on its utilization. The more it is used, the more critical it becomes. Signals coming from the dendrites merge in the cell body. If the total number of signals, weighted by their connection strengths, received by the body exceeds the required activation threshold, the neuron propagates a new signal along its axon to other neurons' dendrites [7].

The first mathematical description of a biological neuron appeared in 1943 [17], and since then, many considerations related to the possible solution of machine learning problems came out. For this thesis, the mathematical model of a neuron shown in the

Figure will be considered. The artificial neuron takes inputs x_1, x_2, ..., x_n, where n is the number of input neurons. Inputs x_i, i = 1, 2, ..., n coming from other neurons are multiplied by respective weights w_i. Weighted inputs are aggregated in the neuron according to the following equation

$$z = \sum_{i=1}^{n} w_i x_i + b, \tag{2.1}$$

where b is a bias term introduced inside the neuron, its value is equivalent to the mentioned activation threshold in biological neurons. The output of the neuron y is created by passing 2.1 to activation function F, as in

$$y = \mathcal{F}(z)$$

and can be transmitted to other neurons.

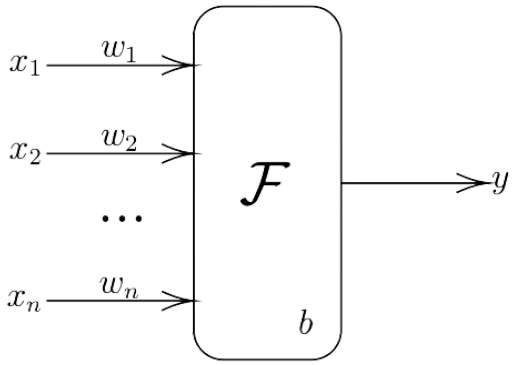


Fig:2.1 Schematic model of an artificial neuron

Let us denote the input vector as $\boldsymbol{X}$ defined in the following way. If we combine k neurons into a layer with n inputs, as in Figure 2.2, we can see that the number of

weights equal $k \times n$. Hence, let us denote the weight matrix as $\boldsymbol{W}$, which has the following form

$$\boldsymbol{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,i} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,i} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{j,1} & w_{j,2} & \cdots & w_{j,i} & \cdots & w_{i,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \cdots & w_{k,i} & \cdots & w_{k,n} \end{bmatrix}. \qquad (\quad)$$

The bias term has now the form of matrix $\boldsymbol{B}$ of size $k \times 1$:

$$\boldsymbol{B} = \begin{bmatrix} b_1 & b_2 & \cdots & b_j & \cdots & b_k \end{bmatrix}^T. \qquad (2.5)$$

The output of such layer has the form of the matrix $\boldsymbol{Y}$ of size $k \times 1$:

$$\boldsymbol{Y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_j & \cdots & y_k \end{bmatrix}^T, \qquad (2.6)$$

given by the equation

$$\boldsymbol{Y} = \mathcal{F}(\boldsymbol{W}\boldsymbol{X} + \boldsymbol{B}), \qquad (2.7)$$

Where activation function F is applied element-wise to the resulting matrix.

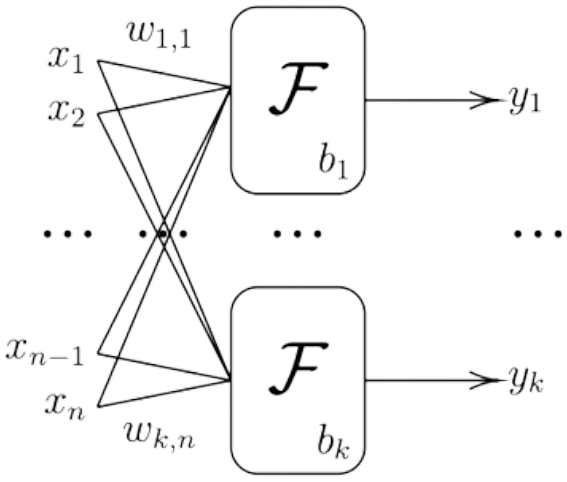


Figure 2.2: Schematic model of artificial neural layer

The layers can be combined by feeding $\boldsymbol{Y}_l$ of the layer l ($l = 1, \cdot \cdot \cdot, L$) to the following layer $l + 1$, as in

$$\boldsymbol{Y}_{l+1} = \mathsf{F}_{l+1}(\boldsymbol{W}_{l+1}\boldsymbol{Y}_l + \boldsymbol{B}_{l+1}), \tag{2.8}$$

with activation functions F_l possibly different for each layer, forming a neural network.

2.2 Concept of Convolutional Neural Network

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous topic 2.1; they are made up of neurons with learnable weights and biases. Each neuron receives inputs, performs a dot product, and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function from the raw image pixels on one end to class scores at the other. And they still have a loss function like SVM and Softmax on the last fully-connected layer, and all the methods we developed for learning regular Neural Networks still apply.

Convolutional Neural Network architectures assume that the inputs are images, allowing us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the number of parameters in the network compared to a Classical Neural Network.

2.1.1 Architecture Overview

As we saw in the previous chapter, Neural Networks receive an input of a single vector and transform it through a series of hidden layers. Each hidden layer comprises a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function independently and do not share any connections. The last fully-connected layer is called the “output layer” like softmax, and in classification settings, it represents the class scores.

Regular Neural Nets don’t scale well to complete images. Let's take the example of

CIFAR-10. The image sizes are 32x32x3 (width, height, color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly, this fully-connected structure does not scale to larger images. For example, an image of a more respectable size, e.g., 200x200x3, would lead to neurons with 200*200*3 = 120,000 weights. Moreover, we would almost certainly want to have several such neurons so that the parameters would add up quickly. So, this full connectivity is wasteful, and the huge number of parameters would quickly lead to overfitting.

Convolutional Neural Networks take advantage of the fact that the input consists of images and constrain the architecture more sensibly. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, and depth. For example, the input images in CIFAR-10 are an input volume of activations, and the volume has dimensions 32x32x3 (width, height, and depth, respectively). As we will soon see, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer for CIFAR-10 has dimensions 1x1x10, because, by the end of the ConvNet architecture, we will reduce the full image into a single vector of class scores, arranged along the depth dimension.

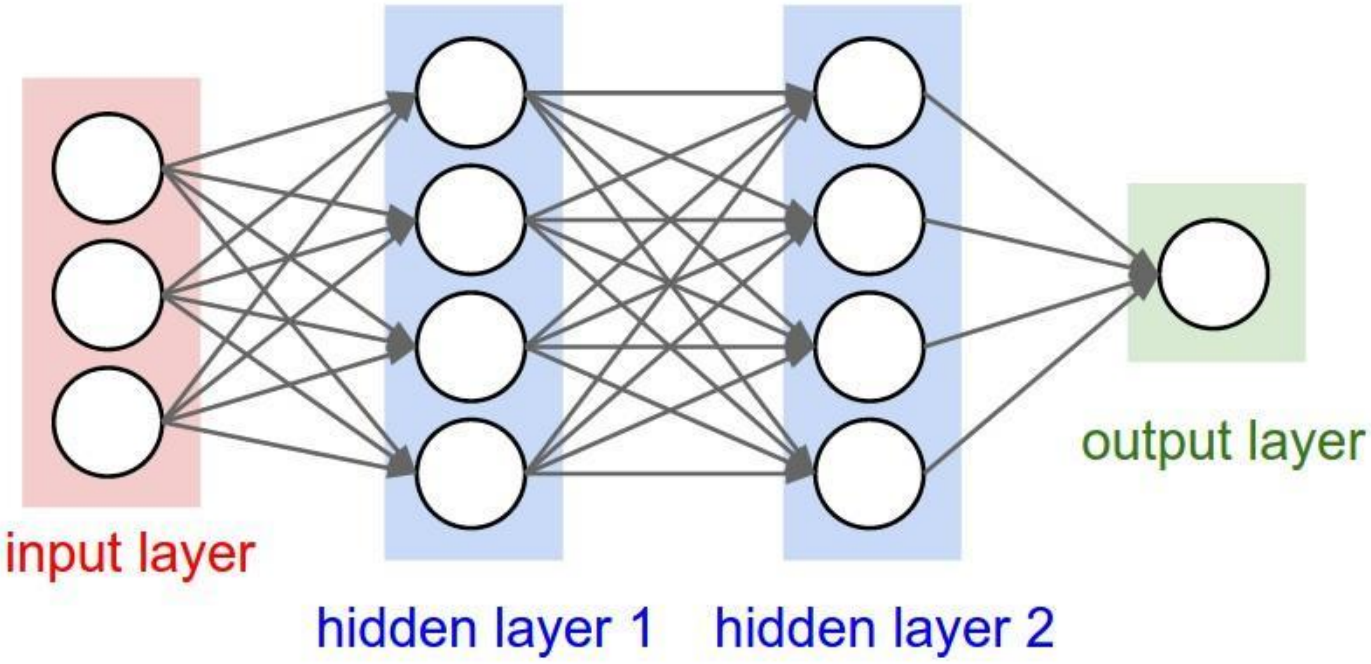


Fig:2.3 Fully Connected Neural Network

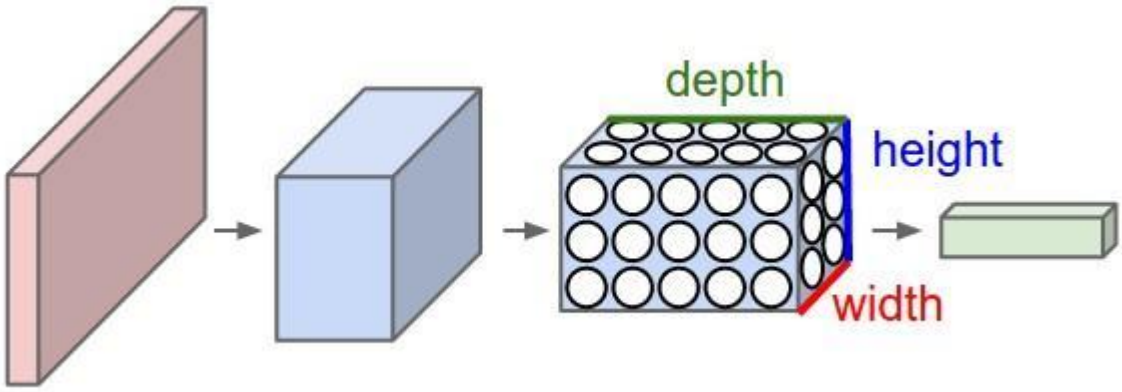


Fig:2.4 Convolutional Neural Network

A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its

width and height would be the image's dimensions, and the depth would be 3 (Red, Green, and Blue channels).

2.1.2 Layers used to build ConvNets

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. By stacking these layers to form a full ConvNet architecture.

A simple ConvNet for CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC].

- INPUT [32x32x3] will hold the raw pixel values of the image, in this case, an image of width 32, height 32, and with three color channels R, G, and B.
- The CONV layer will compute the output of neurons connected to local regions in the input, each computing a dot product between their weights and a small region connected to the input volume. This might result in volume such as [32x32x12] if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the
- max(0,x)
- max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].

- FC (i.e., fully-connected) layer will compute the class scores, resulting in a volume of size [1x1x10], where each of the 10 numbers corresponds to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks, and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters, and others don't. In particular, the CONV/FC layers perform transformations that are a function of not only the input volume's activations but also the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in training set for each image.

2.2 Activation functions

The main advantage of neural networks over linear estimators is the automatic introduction of nonlinearities into the estimated formula. If a linear activation function or no activation function is used, the output signal is just a linear combination of inputs [18], i.e., a hyperplane, and such a deep neural network can be reduced to a linear estimator [7].

To capture more complex relationships between features and create their crosses, nonlinearities must be introduced. Some of the most commonly used activation functions, and those used in the thesis, are:

1. linear function

$$F(z) = az,\ a \in R, \qquad (2.9)$$

2. hyperbolic tangent function

$$F(z) = \tanh(z), \qquad (2.10)$$

3. Rectified Linear Unit (ReLU)

$$F(z) = \max(0, az) = a \max(0, z),\ a \geq 0,$$

4.. softmax function $$\sigma(z)_j = \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_i}}, \qquad (2.12)$$

which yields probabilities σ_j for each class, where $j = 1, 2, ..., k$ represents consecutive output neurons, where z is defined in (2.1).

The combination of activation functions among deep and output layers allows for tackling regression and classification problems, e.g., (2.9) and (2.12) as output layer activations, respectively.

2.3 Training of Neural Networks

Training the neural network requires, fundamentally, the specification of the objective function f . The solution to training the neural network is, generally, to find such decision variables, i.e., weights $\boldsymbol{W}_l$ and bias terms $\boldsymbol{B}_l$, l = 1, 2, ..., L, where L is the number of layers in the network, that minimize f [19]:

$$\text{argmin}\, f(\boldsymbol{W}_l, \boldsymbol{B}_l). \tag{2.13}$$

However, the formulation and evaluation of complete objective function f can be computationally inefficient and sometimes impossible to solve analytically. Thus the optimization is performed based on a definition of the error (loss) function E:

$$E(\boldsymbol{Y}_{true}, \boldsymbol{Y}_{pred}) \rightarrow \mathsf{R}, \tag{2.14}$$

which allows comparison of $\boldsymbol{Y}_{true}$ and $\boldsymbol{Y}_{pred}$ (true outputs and those predicted by the model), yielding an error value to be minimized, e.g., root mean square error or categorical cross-entropy [7].

The update of network weights is performed iteratively for each layer and can be performed in three ways, with respect to the number of input-output pairs m after which the update takes place and the total number of input-output pairs n. We can distinguish batching (when $m = n$), mini-batching (when $m < n$) and no batching. The latter means that parameter update is calculated after considering each input-output pair.

2.3.1 Backpropagation

One of the approaches to the iterative minimization and parameters update is so-called backpropagation. In this method, the error calculated with (2.14) is propagated backward through the consecutive layers of the network.

The update of network parameters can be performed with any type of batching and has the value of

$$\Delta w_{l,j,k} = -\eta \frac{\partial E}{\partial w_{l,j,k}}, \tag{2.15}$$

where $w_{l,j,k}$ is the updated weight for the i-th input value for a k-th neuron in the l-th layer, and

η is a scaling factor called the learning rate. Alternatively, to train the neural network, a more complex but faster converging acceleration method may be used, in which the current gradient is used to modify the velocity of the point in weight space instead of changing its position:

where t represents the current epoch, and α is a decay factor [Rumelhart et al., 1986]. The difference in interpretations between (2.15) and (2.16) is such that (2.15) updates weights by jumping over specific values. In contrast, (2.16) updates them by introducing inertia in the weight space, hence, the possibility of overcoming saddle points and avoiding getting stuck in local minima during the optimization process.

2.3.1 The Adam algorithm

The learning algorithm used in this thesis is the Adam algorithm, the stochastic gradient descent algorithm that adaptively estimates the first and second moments of gradients to determine the direction of update [13]. One should point out the similarity to the backpropagation with acceleration by (2.16), in that both methods change the step dynamically, depending on a local gradient. Equation (2.16) uses inertia to keep the point moving by changing its movement trajectory, while (2.17) uses local gradient statistics to choose the direction of the next jump.

The following formula gives the update of parameters Δw:

$$\Delta w_{l,j,k} = -\eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.17}$$

where $t \in \mathbb{N}$ is the number of current learning epochs ($t = 0$ for initial epoch), η is the learning rate (step size), E is a small numerical stabilization constant, $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected first and second raw moment estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{2.18}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{2.19}$$

where $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates for moment estimates, m_t and v_t

are biased first and second raw moment estimates, respectively:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t, \tag{2.20}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g^2, \tag{2.21}$$

and g_t is gradient of stochastic objective function f_t in epoch t, given by

$$g_t = \nabla_w f_t(w_{t-1}). \tag{2.22}$$

Chapter 3

Quantum Neural Networks

3.1 Quantum computing

The basic idea of quantum is fundamentally different from the operating principles of what will be called classical computers. Quantum algorithms consist of a series of operations performed on a quantum analogy of bits based on quantum mechanics principles. Quantum computers are not standalone devices since they must be controlled by a classical computer [20].

3.1.1 The qubit

The general called *classical* computing is based on the concept of a *bit*. The bit can be in one of two states denoted as 0 and 1, and the technical implementation is nowadays realized in the form of transistors. A *quantum bit*, or shortly *qubit*, is a quantum analogy of a classical bit, with such difference that qubit can be $|0\rangle$, $|1\rangle$ or in both of $|0\rangle$ and $|1\rangle$ at the same time, with non-zero probabilities for measuring either of the states. The mathematical representation of a qubit in the state $|\psi\rangle$, is a linear combination of states $|0\rangle$ and $|1\rangle$, namely, the *superposition* of those states:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{3.1}$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. Values $|\alpha|^2$ and $|\beta|^2$ represent the probabilities of

measuring states $|0\rangle$ and $|1\rangle$, respectively. Numbers $|\alpha|$ and $|\beta|$ are complex numbers,and (3.1) can be rewritten in a polar form:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\,, \tag{3.2}$$

where $\theta, \phi \in \mathbb{R}$. Thus, the state can be interpreted as a point on a complex sphere with radius 1, called the *Bloch sphere*, shown in Figure 3.1.[20].

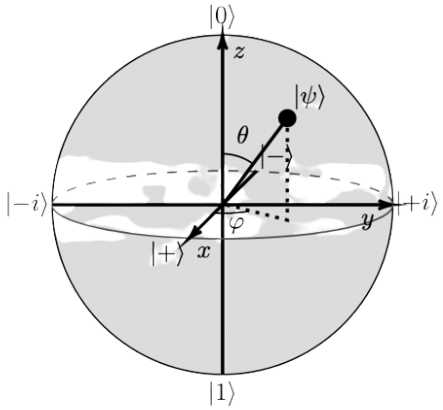


Fig:3.1 Bloch Sphere Representation Of Qubit

The technical aspects of physical implementations of qubits are very sophisticated, while new realizations are still being developed. The common aspect of all the methods is that single particles carry the quantum representation of information. The qubit can be physically realized by, e.g., electron spin [21], electron charge [10], [22], photon polarization [23], [24], nuclear spin [25], using Joseph- son junction effects [12], or using van der Waals heterostructure effects [26].

Unfortunately, all current physical implementations are subjected to noise and measurement errors. It is worth mentioning, however, that the *Noisy Intermediate-Scale*

Quantum technology, as defined in [27], has already emerged with the Sycamore processor that uses 54 interconnected qubits [2]. The 'noisy' in the term means the lack of qubits available to be dedicated to error correction in algorithms, and 'intermediate-scale' for that the number of used qubits (specified as 50) will be high enough to be impossible to simulate on classical computers, but not high enough to provide the universal operation of a quantum computer.

3.1.2 Quantum Circuits

The quantum computations are based on the evolution of a quantum circuit, where the quantum gate operations, modifying the qubits' physical states with, e.g., electric voltage [28], are applied sequentially, as programmed by the scientist. After completing the program, the chosen qubits are observed (measured). At this moment, nature chooses one of all existing evolutions, and thus a single, determined state can be observed. The experiment is repeated many times to gather statistically significant data.

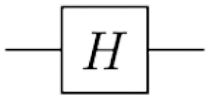


Figure 3.2: The Hadamard gate

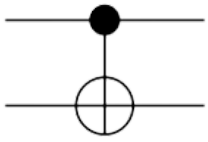


Figure 3.3: The Controlled NOT gate

Since any quantum state can be represented as a unitary vector, quantum gates

applied to the qubit to change its state can be interpreted as rotations on the Bloch sphere and represented using their corresponding rotation matrix. A notable example of a single qubit quantum gate is the Hadamard gate, represented by symbol *H* and described by the Hadamard matrix

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \tag{3.3}$$

and its quantum circuit representation is shown in Figure 3.2. It can be interpreted as rotation by π radians about the axis defined by the vector [1, 0, 1]. Another important example is the 2-qubit Controlled NOT (*CNOT*) gate, shown in Figure 3.3 and represented by a matrix

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{3.4}$$

It allows control of the *z*-axis flip of a qubit, depending on the state of the control qubit. If the control qubit is in the superposition, introduced by, e.g., (3.3), as in Figure 3.4, the

two qubits become entangled since the measurement of one of them immediately determines the state of the other. Hence, both qubits are in one of the so-called *Bell states*, or *EPR states*:

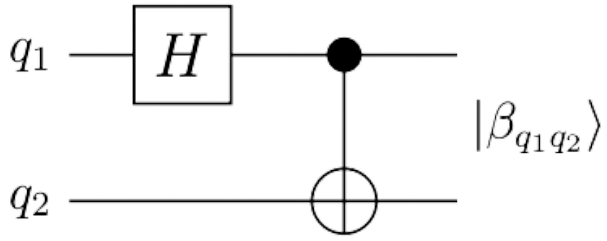


Figure 3.4: Quantum circuit creating the Bell states

The generalization of the CNOT gate is the parametrized Power CNOT gate, defined as

$$CNOT(z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & gc & -igs \\ 0 & 0 & -igs & gc \end{bmatrix}, \tag{3.9}$$

where $c = cos(\ _2 z)$, $s = sin(\ _2 z)$, $g = e$ [redacted] [29].

The other gate used in the thesis is the R_x rotation gate, allowing rotation of qubit state about the x axis, by the value θ, as in:

$$R_x(\theta) = e^{-i\theta X/2} = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, \tag{3.10}$$

where X is the X Pauli matrix that represents the quantum NOT (Pauli X) gate [20]:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{3.11}$$

3.2 Quantum Deep Learning

The author in [27] postulates that the advantage of quantum deep learning over classical deep learning may be coming from quantum implementations of linear algebra-related tasks. The quantum state of n qubits is a vector in 2^n dimensional complex vector space, i.e., it can store 2^n bits of classical information. The quantum algorithms related to linear algebra give exponential speed up, as stated in [5].

Another possible advantage of quantum computing over classical computing in deep learning or other machine learning tasks may be the concept of Quantum Random Access Memory (QRAM), where data of size n may be stored using log n qubits. Unfortunately, there is a non-negligible drawback of quantum computing, which is the fact that the classical data needs to be at first encoded in a quantum way, which is a time-consuming process and may nullify the advantage. [27]

This thesis, however, does not focus on quantum algorithms that can speed up the data handling but rather on quantum circuits themselves, as possible trainable deep neural network layers and their applicability in solving the real classical data problems.

3.3 Hybrid quantum-classical neural networks

The recent release of TensorFlow Quantum (TFQ) [6], Pytorch, and Qiskit bring new opportunities to experiment with quantum machine learning algorithms by allowing easy prototyping of quantum neural networks. Well set position of classical TensorFlow (TF) encourages the use of the new library and allows the use of all the capabilities of its classical version to easily combine quantum neural network layers with classical ones with the help of high-level Keras API [8]. For simulation on a classical computer, the TFQ uses *qsim* simulator [30] and *Cirq* library to build quantum layers [29], but the library is prepared to change the target device to a Quantum Processing Unit easily.

This thesis is based on the approach proposed by Tensorflow, Pytorch and Qiskit, which combines classical layers and *parameterized quantum circuits* to create hybrid quantum-classical models. A general mathematical description of a quantum neural network proposed in this article is given in the form

$$\hat{U}(\boldsymbol{\theta}) = \prod_{l=1}^{L} \hat{V}_l \hat{U}_l(\boldsymbol{\theta}_l), \tag{3.12}$$

where l is the layer index, L is the number of layers, $\hat{V}l$ is a non-parametric unitary (circuit), $\hat{U}l(\boldsymbol{\theta}_l)$ is unitary with variational parameters (parametrized circuit), and $\boldsymbol{\theta}$ is a vector of parameters.

The loss function of the quantum neural network is, contrary to (2.14), often defined as an expectation value of a cost Hamiltonian

$$f(\boldsymbol{\theta}) = \langle \hat{H} \rangle_{\boldsymbol{\theta}} \equiv \langle \boldsymbol{\psi}_0 | \hat{U}^{\dagger}(\boldsymbol{\theta}) \hat{H} \hat{U}(\boldsymbol{\theta}) | \boldsymbol{\psi}_0 \rangle \,, \tag{3.13}$$

where $|\psi)_0$ is the input state of the parametrized circuit, since the quantum circuit requires sampling, getting an estimate of the expectation value hθ within the accuracy E requires ~ O(⊥) measurements of all observables [6].

To train the network defined in (3.12), using cost function (3.13), the parameter shift has a general form of

$$\frac{\partial}{\partial \eta_{l,j,k}} f(\boldsymbol{\eta}) = f(\boldsymbol{\eta} + \frac{\pi}{4}\boldsymbol{\Delta}_{l,j,k}) - f(\boldsymbol{\eta} - \frac{\pi}{4}\boldsymbol{\Delta}_{l,j,k})), \tag{3.14}$$

where $\boldsymbol{\Delta}_{l,j,k}$ is a vector representing unit-norm perturbation of the variable $\eta_{l,j,k}$ in the positive direction. The procedure is, however, computationally costly; hence, the gradient descent is performed stochastic [6].

Chapter 4

Brain Tumor MRI Dataset

One of the problems solved in this thesis is the classification of Brain Tumor MRI Dataset, and the goal of this chapter is to show that we can use a quantum neural network to classify Brain Tumor and also tried different architectures of Quantum Neural networks on Brain MRI Tumor dataset

4.1 About the dataset

A Brain tumor is considered one of the most aggressive diseases among children and adults. Brain tumors account for 85 to 90 percent of all primary Central Nervous System(CNS) tumors. Every year, around 11,700 people are diagnosed with a brain tumor. The 5-year survival rate for people with a cancerous brain or CNS tumor is approximately 34 percent for men and 36 percent for women. Brain Tumors are classified as benign, Malignant Tumors, pituitary tumors, etc. Proper treatment, planning, and accurate diagnostics should be implemented to improve the life expectancy of the patients. The best technique to detect brain tumors is Magnetic Resonance Imaging (MRI). A huge amount of image data is generated through the scans. The radiologist examines these images. A manual examination can be error-prone due to the complexities involved in brain tumors and their properties.

Brain tumors are abnormal cell aggregations that grow inside the brain tissues. Brain tumors can be divided into benign tumors and malignant tumors. Brain benign tumors can be cured by surgery, while malignant brain tumors are one of the most deadly types of cancer and can

lead directly to death [31]. Brain tumors can also be divided into primary tumors formed in the brain or derived from the brain nerves and metastatic brain tumors metastasized from other body parts to the brain. The most common primary brain tumors in adults are primary central nervous system lymphoma and gliomas, of which gliomas originate from the periglacial tissue and account for more than 80% of malignant brain tumors. Different symptoms appear with different lesion areas, such as headache, vomiting, visual decline, epilepsy, and confusion. A more detailed classification divides brain tumors into four grades. The higher the grade, the more malignant the brain tumor is. According to the Global cancer statistics 2020 [32], there are about 308,000 new cases of brain cancers in 2020, accounting for about 1.6% of all new cases of cancers, and about 251,000 deaths from brain cancers, accounting for about 2.5% of all cancer deaths.

Early detection is essential for effectively treating brain tumors [33]. With the development of medical imaging, imaging techniques play an important role in brain tumor diagnosis and treatment evaluation and can provide doctors with a clear human brain structure. These imaging techniques can provide information on brain tumors' shape, size, and location, assisting doctors in making an accurate diagnosis and developing a treatment plan. Magnetic resonance (MR) imaging is one of neurology's most commonly used scanning methods. MR imaging uses radiofrequency signals to excite the target tissue under a powerful magnetic field to produce an image of its interior. It has the advantages of high soft-tissue contrast and zeroes ionizing radiation exposure. Therefore, MR imaging is more suitable for detecting brain lesions [34].

Brain Tumors are complex. There are many abnormalities in the size and location of the brain tumor(s). This makes it difficult to understand the nature of the tumor thoroughly. Also, a professional Neurosurgeon is required for MRI analysis. In developing countries,

the lack of skillful doctors and knowledge about tumors often makes it challenging and time-consuming to generate MRI reports. So an automated system in Cloud can solve this problem.

There are three different architectures we will construct in this chapter

4.2 Quantum Convolutional Neural Network

4.2.1 Classical convolution

As we have seen in the previous chapter, a convolutional neural network (CNN) is a standard model in classical machine learning which is particularly suitable for processing images. The model is based on the idea of a convolution layer where a local convolution is applied instead of processing the full input data with a global function.

Small local regions are sequentially processed with the same kernel if the input is an image. The results for each region are usually associated with different channels of a single output pixel. The union of all the output pixels produces a new image-like object, which additional layers can further process.

4.2.2 Quantum convolution

We can extend the same idea also to the context of quantum variational circuits. Below is the possible methodology, which is very similar to the one used in [35]. The scheme is also represented in figure 4.1 given below.

1. A small region of the input images(in our example, a 2×2 square) is embedded into a

quantum circuit. The architecture below achieves this with parameterized rotations to the qubits initialized in the ground state.

2. A quantum computation associated with a unitary U is performed on the system. The unitary could be generated by a variational quantum circuit, as discussed in [35].
3. The quantum system is finally measured, and a list of classical expectation values is obtained. The measurement results could also be classically post-processed, as discussed in [35]. In our implementation, we have used raw expectation value.
4. Like a classical convolution layer, each expectation value is mapped to a different channel of a single output pixel.
5. Iterating the same procedure over different regions and to different channels, it scans the full input image and produces an output object which will be structured as a multichannel image.
6. The quantum convolution can be followed by further quantum layers or by classical layers. In our case, we have taken the classical layer to classify into different classes
7. Since the input Brain MRI tumor dataset has three channels, the output after convolution will have 12 channels because every pixel outputs three pixels after convolution.

8. The main difference concerning a classical convolution is that a quantum circuit can generate highly complex kernels whose computation could be, at least in principle, classically intractable.

4.3 Quantum Convolutional Neural Network Architecture

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten_6 (Flatten)          (None, 150528)            0

 dense_6 (Dense)              (None, 4)                 602116

=================================================================
Total params: 602,116
Trainable params: 602,116
Non-trainable params: 0
_________________________________________________________________
```

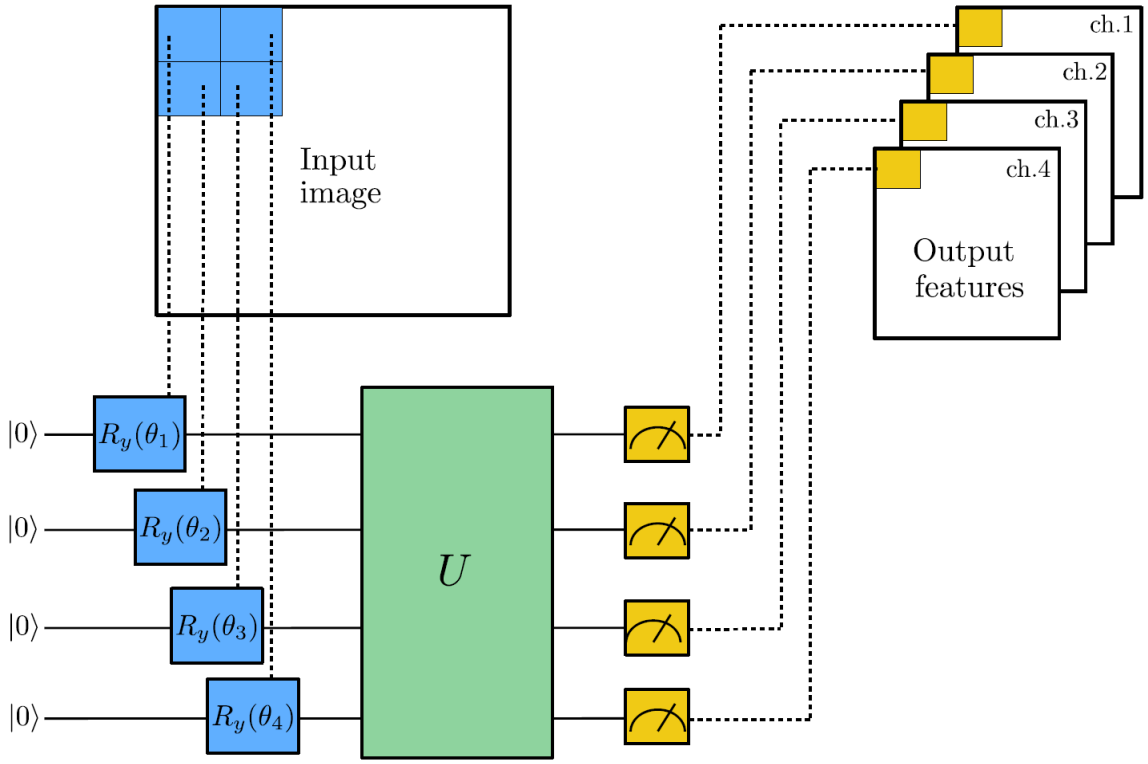


Fig-4.1 Quantum Convolution Neural Network Architecture

4.4 Methodology

4.4.1 Quantum Circuit as a Convolution Kernel

We have followed the method described in the introduction and represented in the figure above. We simulated a system of 4 qubits. The associated qnode represents the quantum circuit consisting of:

1. an embedding layer of local Ry rotations (with angles scaled by a factor of π);
2. A random circuit of n_layers;
3. A final measurement in the computational basis and estimating 4 expectation values.

4.4.2 Convolution Method:

1. The input image is divided into squares of 2×2 pixels.
2. The quantum circuit processes each square.
3. The four expectation values are mapped into 4 different channels of a single output pixel.
4. The above process halves the resolution of the input image. In terms of a Convolutional Neural Network, it corresponds to a convolution with a 2×2 kernel and a stride equal to 2
5. Since we are not going to train the quantum convolutional layer, So that's why it is efficient to make it a pre-processing layer for all the images of our dataset. After that entirely classical model will be directly trained and tested on the pre-processed dataset. It will help to avoid repetitions of quantum computations.
6. After applying the quantum convolution layer, we feed the resulting features into a

classical neural network that will be trained to classify the Brain MRI Tumor input image into four classes, glioma_tumour, meningoma_tumour, no_tumour, and pituitary_tumour.

7. In the final Layer (**Fig-4.1**), we used a fully connected layer with 4 output nodes with a final *softmax* activation function.
8. The model is compiled with a stochastic-gradient-descent optimizer and a cross-entropy loss function.

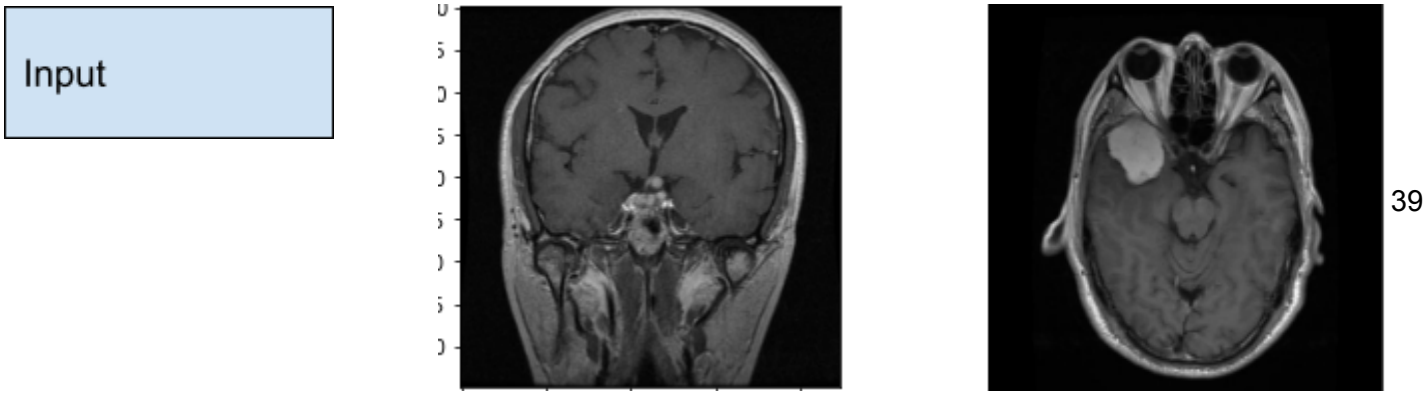
Dense Layer

It is nothing but a classical, fully connected layer with 4 neurons because we are classifying the Brain MRI Tumor dataset into four classes, namely **glioma_tumour, meningoma_tumour, no_tumour, and pituitary_tumour**

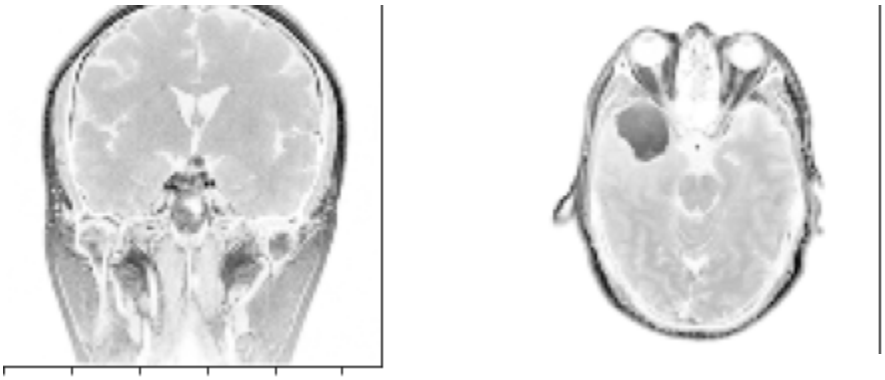
4.5 Output of Quantum Convolutional Neural Network

Below each input image **fig-4.2**, the 12 output channels generated by the quantum convolution are visualized in grayscale. One can notice the downsampling of the resolution and some local distortion introduced by the quantum kernel. On the other hand, the global shape of the image is preserved, as expected, for a convolution layer.

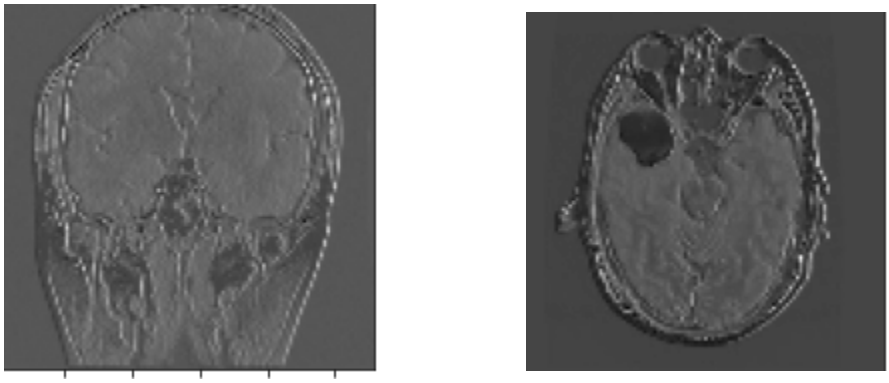
This is the 12-channel output after Quantum Convolution Operation



Channel 1 output

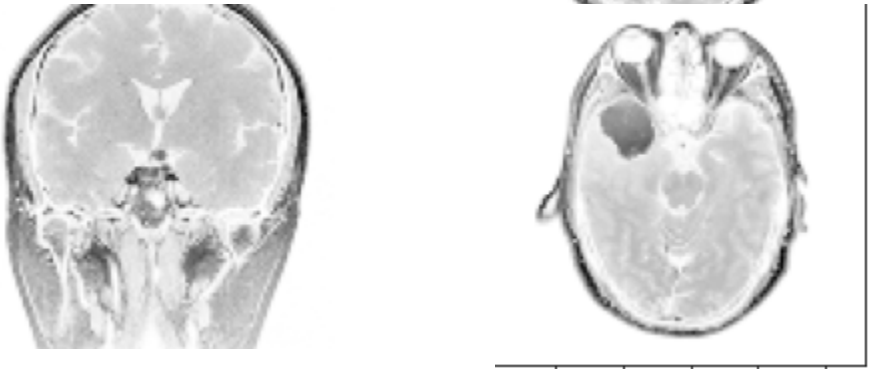


Channel 2 output

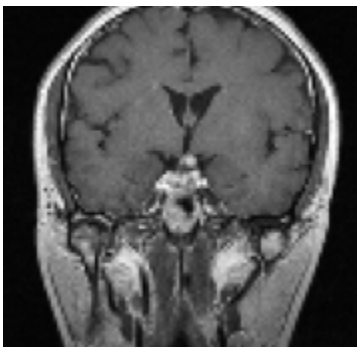


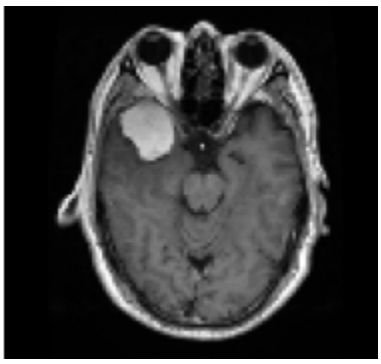
Channel 3 output

Channel 4 output

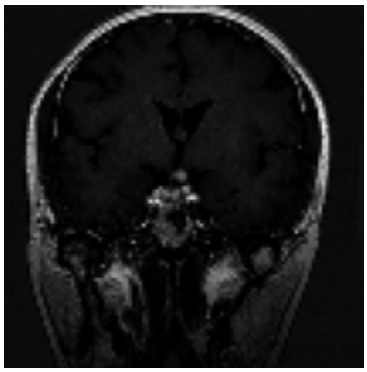


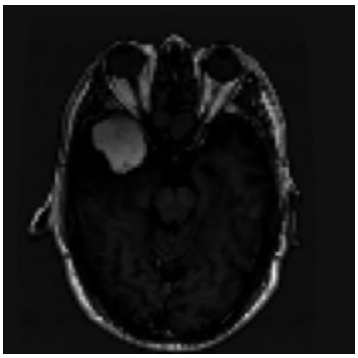
Channel 5 output



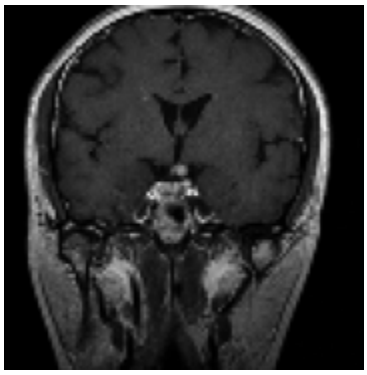


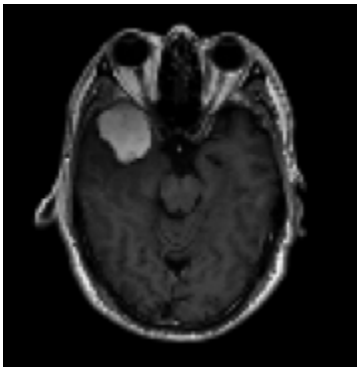
Channel 6 output



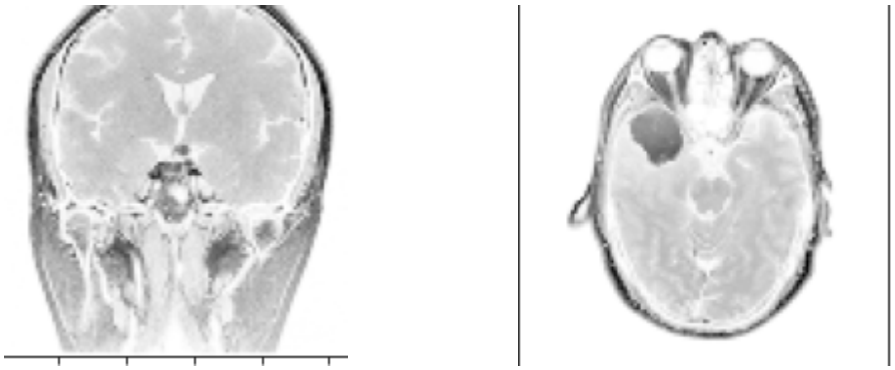


Channel 7 output

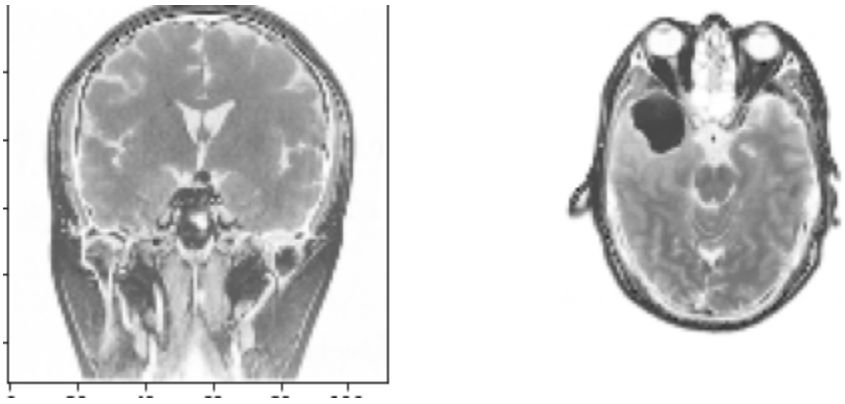




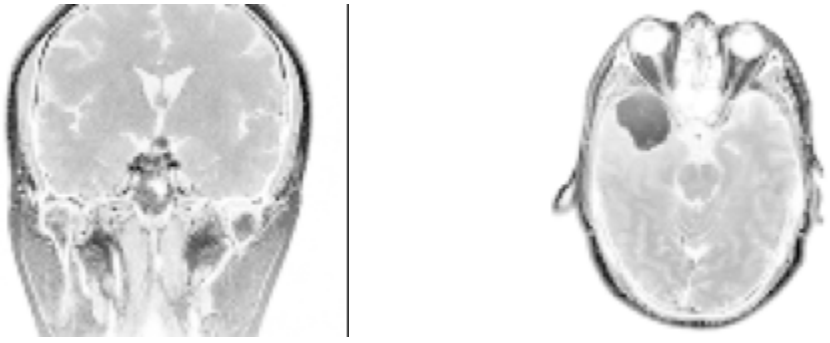
Channel 8 output



Channel 9 output



Channel 10 output



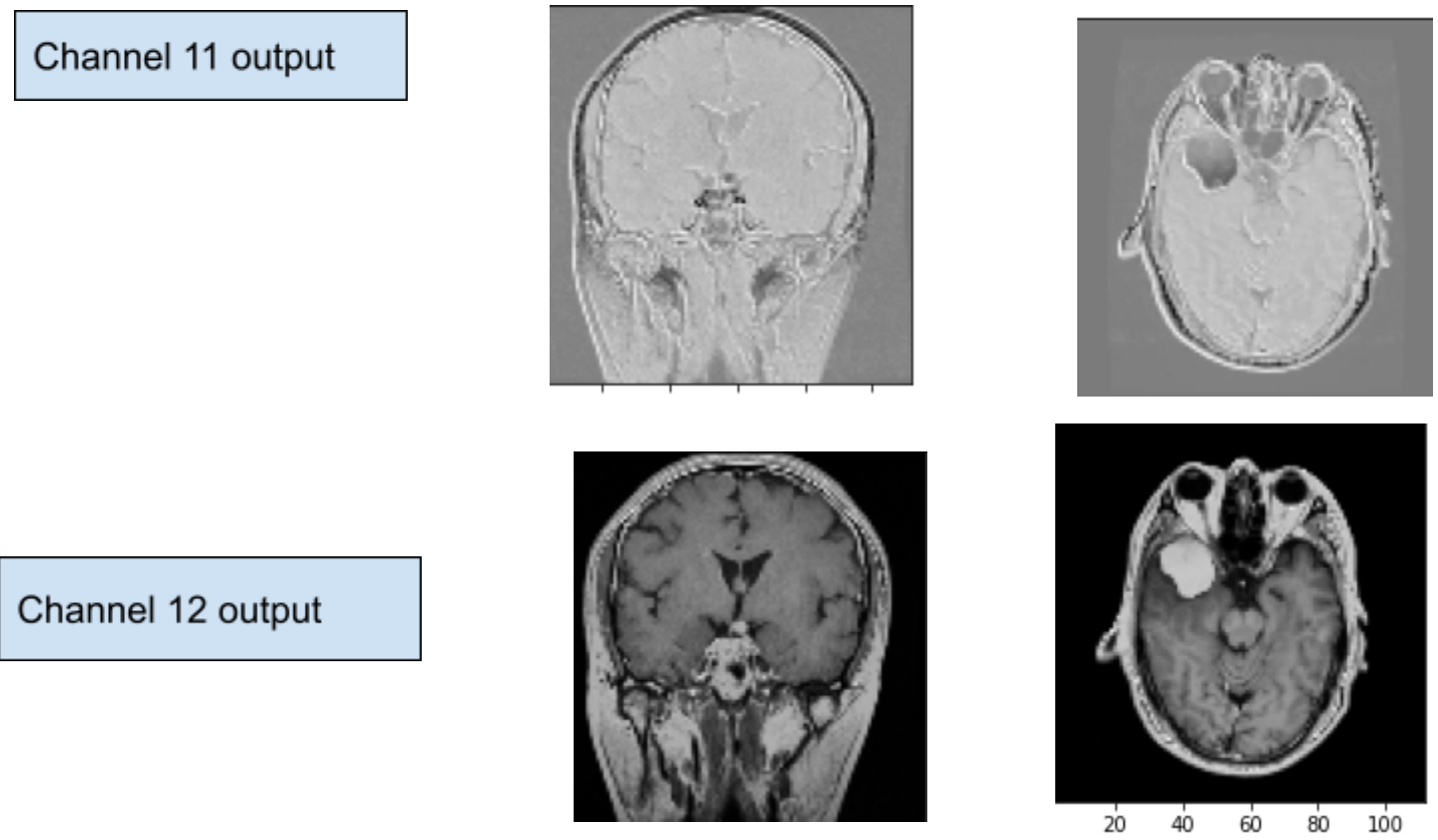


Fig:4.2: 12 channels output after Quantum Convolution Operation

Training of Quantum Convolution Neural Network Architecture

We have trained the classical neural network for 15 epochs and cross-entropy as a loss function, and after training, we have achieved a training accuracy of 96% and testing accuracy of 63.3%, as shown in **figure 4.3**

```
Epoch 1/15
13/13 - 1s - loss: 65.0645 - accuracy: 0.1600 - val_loss: 61.9764 - val_accuracy: 0.2667 - 556ms/epoch - 43ms/step
Epoch 2/15
13/13 - 0s - loss: 26.2617 - accuracy: 0.4800 - val_loss: 15.5157 - val_accuracy: 0.3333 - 142ms/epoch - 11ms/step
Epoch 3/15
13/13 - 0s - loss: 13.9558 - accuracy: 0.6000 - val_loss: 13.5931 - val_accuracy: 0.4000 - 147ms/epoch - 11ms/step
Epoch 4/15
13/13 - 0s - loss: 4.5821 - accuracy: 0.7600 - val_loss: 11.3038 - val_accuracy: 0.4667 - 133ms/epoch - 10ms/step
Epoch 5/15
13/13 - 0s - loss: 2.2233 - accuracy: 0.8800 - val_loss: 6.8469 - val_accuracy: 0.6000 - 175ms/epoch - 13ms/step
Epoch 6/15
13/13 - 0s - loss: 4.2169 - accuracy: 0.7800 - val_loss: 5.3257 - val_accuracy: 0.6333 - 133ms/epoch - 10ms/step
Epoch 7/15
13/13 - 0s - loss: 3.9355 - accuracy: 0.8200 - val_loss: 8.6820 - val_accuracy: 0.6000 - 131ms/epoch - 10ms/step
Epoch 8/15
13/13 - 0s - loss: 9.8512 - accuracy: 0.7400 - val_loss: 47.4195 - val_accuracy: 0.2667 - 188ms/epoch - 14ms/step
Epoch 9/15
13/13 - 0s - loss: 5.9041 - accuracy: 0.6800 - val_loss: 8.3452 - val_accuracy: 0.6000 - 141ms/epoch - 11ms/step
Epoch 10/15
13/13 - 0s - loss: 2.6108 - accuracy: 0.9200 - val_loss: 5.8693 - val_accuracy: 0.7333 - 144ms/epoch - 11ms/step
Epoch 11/15
13/13 - 0s - loss: 1.0606 - accuracy: 0.9400 - val_loss: 24.8125 - val_accuracy: 0.4333 - 142ms/epoch - 11ms/step
Epoch 12/15
13/13 - 0s - loss: 2.7018 - accuracy: 0.9200 - val_loss: 26.6381 - val_accuracy: 0.5333 - 140ms/epoch - 11ms/step
Epoch 13/15
13/13 - 0s - loss: 4.9861 - accuracy: 0.8200 - val_loss: 13.8559 - val_accuracy: 0.6667 - 179ms/epoch - 14ms/step
Epoch 14/15
13/13 - 0s - loss: 0.7353 - accuracy: 0.9600 - val_loss: 22.6719 - val_accuracy: 0.4667 - 180ms/epoch - 14ms/step
Epoch 15/15
13/13 - 0s - loss: 2.1443 - accuracy: 0.8800 - val_loss: 6.1749 - val_accuracy: 0.6333 - 143ms/epoch - 11ms/step
```

Fig 4.3 Training of Quantum Convolution Neural Network Architecture

To compare the results achievable with and without the quantum convolution layer, we also initialize a “classical” instance of the model that will be directly trained and validated with the raw Brain MRI Tumor Dataset images (i.e., without quantum pre-processing).

Classical Neural Network Model Architecture

We have built a simple classification model with 4 neurons as output and Softmax as an activation function, as shown in **fig-4.4**

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_7 (Flatten)         (None, 150528)            0

 dense_7 (Dense)             (None, 4)                 602116

=================================================================
Total params: 602,116
Trainable params: 602,116
Non-trainable params: 0
_________________________________________________________________
```

Fig:4.4

Classical Neural Network Model Training

We have trained the classical neural network for 15 epochs and cross-entropy as a loss function, and after training, we have achieved a training accuracy of 96% and testing accuracy of 56%, as shown in figure 4.5

```
Epoch 1/15
13/13 - 1s - loss: 18.6996 - accuracy: 0.2800 - val_loss: 11.9263 - val_accuracy: 0.2667 - 586ms/epoch - 45ms/step
Epoch 2/15
13/13 - 0s - loss: 8.0351 - accuracy: 0.5800 - val_loss: 11.0381 - val_accuracy: 0.3000 - 130ms/epoch - 10ms/step
Epoch 3/15
13/13 - 0s - loss: 7.2531 - accuracy: 0.5600 - val_loss: 7.8073 - val_accuracy: 0.3667 - 129ms/epoch - 10ms/step
Epoch 4/15
13/13 - 0s - loss: 2.2402 - accuracy: 0.7600 - val_loss: 8.6712 - val_accuracy: 0.5667 - 149ms/epoch - 11ms/step
Epoch 5/15
13/13 - 0s - loss: 2.0267 - accuracy: 0.8000 - val_loss: 13.4618 - val_accuracy: 0.2333 - 133ms/epoch - 10ms/step
Epoch 6/15
13/13 - 0s - loss: 1.2580 - accuracy: 0.8000 - val_loss: 6.0682 - val_accuracy: 0.6333 - 127ms/epoch - 10ms/step
Epoch 7/15
13/13 - 0s - loss: 0.8073 - accuracy: 0.9000 - val_loss: 6.3492 - val_accuracy: 0.4333 - 125ms/epoch - 10ms/step
Epoch 8/15
13/13 - 0s - loss: 0.3128 - accuracy: 0.9600 - val_loss: 8.2024 - val_accuracy: 0.4333 - 135ms/epoch - 10ms/step
Epoch 9/15
13/13 - 0s - loss: 0.2515 - accuracy: 0.9400 - val_loss: 5.2641 - val_accuracy: 0.5333 - 127ms/epoch - 10ms/step
Epoch 10/15
13/13 - 0s - loss: 0.1540 - accuracy: 0.9600 - val_loss: 7.7288 - val_accuracy: 0.4667 - 139ms/epoch - 11ms/step
Epoch 11/15
13/13 - 0s - loss: 0.2675 - accuracy: 0.9600 - val_loss: 11.4246 - val_accuracy: 0.4667 - 151ms/epoch - 12ms/step
Epoch 12/15
13/13 - 0s - loss: 0.4383 - accuracy: 0.9200 - val_loss: 6.9473 - val_accuracy: 0.5000 - 139ms/epoch - 11ms/step
Epoch 13/15
13/13 - 0s - loss: 0.0384 - accuracy: 0.9800 - val_loss: 5.7850 - val_accuracy: 0.5667 - 176ms/epoch - 14ms/step
Epoch 14/15
13/13 - 0s - loss: 7.4469e-05 - accuracy: 1.0000 - val_loss: 6.5650 - val_accuracy: 0.5667 - 136ms/epoch - 10ms/step

13/13 - 0s - loss: 7.4143e-04 - accuracy: 1.0000 - val_loss: 6.8242 - val_accuracy: 0.5667 - 126ms/epoch - 10ms/step
```

Figure 4.5 Classical Neural Network Model Training

We have finally plotted the test accuracy and the test loss with respect to the number of training epochs for Architecture with Quantum Layer and Without Quantum Layer. With Quantum Layer, we have achieved the testing accuracy of 66%, and without a quantum layer, we have achieved the accuracy of 56%, as shown in figure 4.6

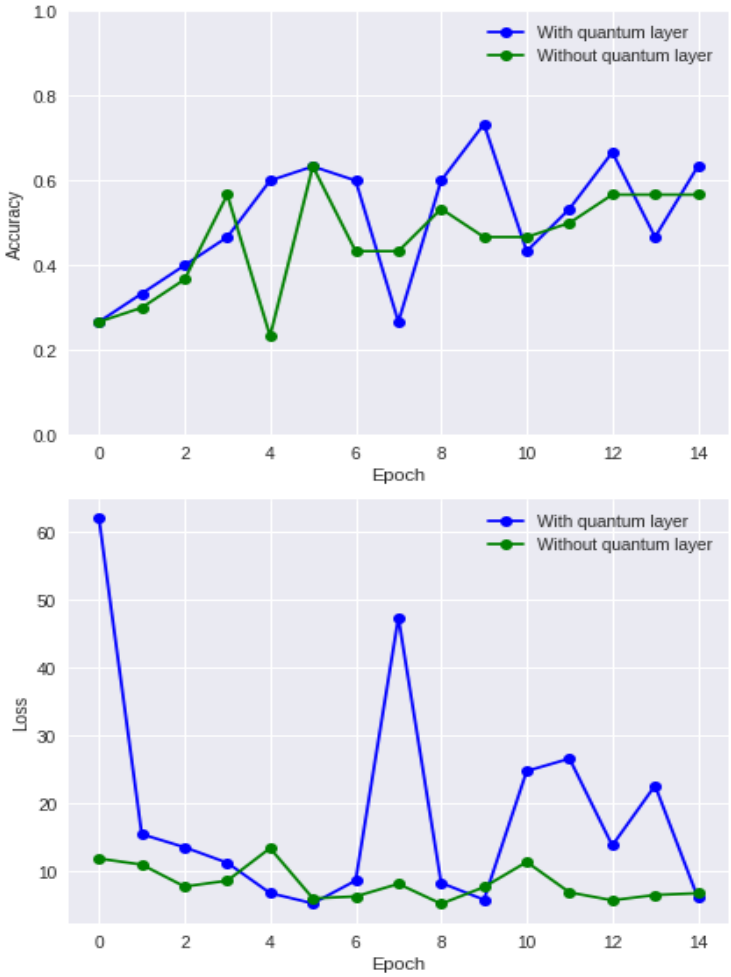


Fig:4.6: Plot of Accuracy with respect to epoch and loss with respect to the epochWith quantum and without a quantum layer

4.6 Hybrid Quantum Convolutional Neural Network

Neurons and Weights

As we have seen in the previous chapter, a neural network is ultimately just an elaborate function built by composing smaller building blocks called neurons. A neuron is typically an easy-to-compute nonlinear function that maps one or more inputs to an output which is nothing but a single real number. The single output of a neuron becomes an input into other neurons. Graphically, neurons are represented as nodes in a graph, and directed edges are used between nodes to indicate how the output of one neuron will become input to other neurons. Each edge in the Neural Network graph is often associated with a scalar value called a weight. Each input to a neuron will be multiplied by a scalar before being collected and processed into a single value. The objective when training a neural network consists primarily of choosing our weights such that the network behaves in a particular way.

Feed Forward Neural Networks(FFNN)

Feed Forward Neural Networks means that as data flows through our neural network, it will never return to a neuron it has already visited. Equivalently, we can say that the graph described by a neural network is a directed acyclic graph.

IO Structure of Layers

The input to a neural network is a real-valued vector. Each input vector component is multiplied by a different weight and fed into a layer of neurons. After each neuron in the layer has been evaluated, the results are collected into a new vector where the i'th

component records the output of the i'th neuron. This new vector can then be treated as an input for a new layer, and so on. We will use the term as a hidden layer to describe all except our network's first and last layers.

Backpropagation in Hybrid Quantum Neural Network

As we have seen Backpropagation in classical ML similarly, we can view a quantum circuit as a black box, and the gradient of this black box concerning its parameters can be calculated as follows:

$$\nabla_\theta \boxed{\text{Quantum Circuit}\,(\theta)} = \boxed{\text{Quantum Circuit}\,(\theta + s)} - \boxed{\text{Quantum Circuit}\,(\theta - s)}$$

where θ represents the parameters of the quantum circuit and s is a macroscopic shift. The gradient is then simply the difference between our quantum circuit evaluated at θ+sand θ−s

Thus, we can systematically differentiate our quantum circuit as part of a more extensive backpropagation routine. This closed-form rule for calculating the gradient of quantum circuit parameters is known as the parameter shift rule.

There are two different architectures we will construct in this chapter. Figures 4.8 illustrate the first architecture we will construct in this chapter. Ultimately, we will create a hybrid quantum-classical neural network that seeks to classify the Brain Tumor MRI Dataset. Note that the edges in this image are all directed downward; however, the directionality is not

visually indicated.

In this architecture, we focused on the binary classification of Brain MRI tumor Datasets to **no_tumor** and **glioma_tumour**.

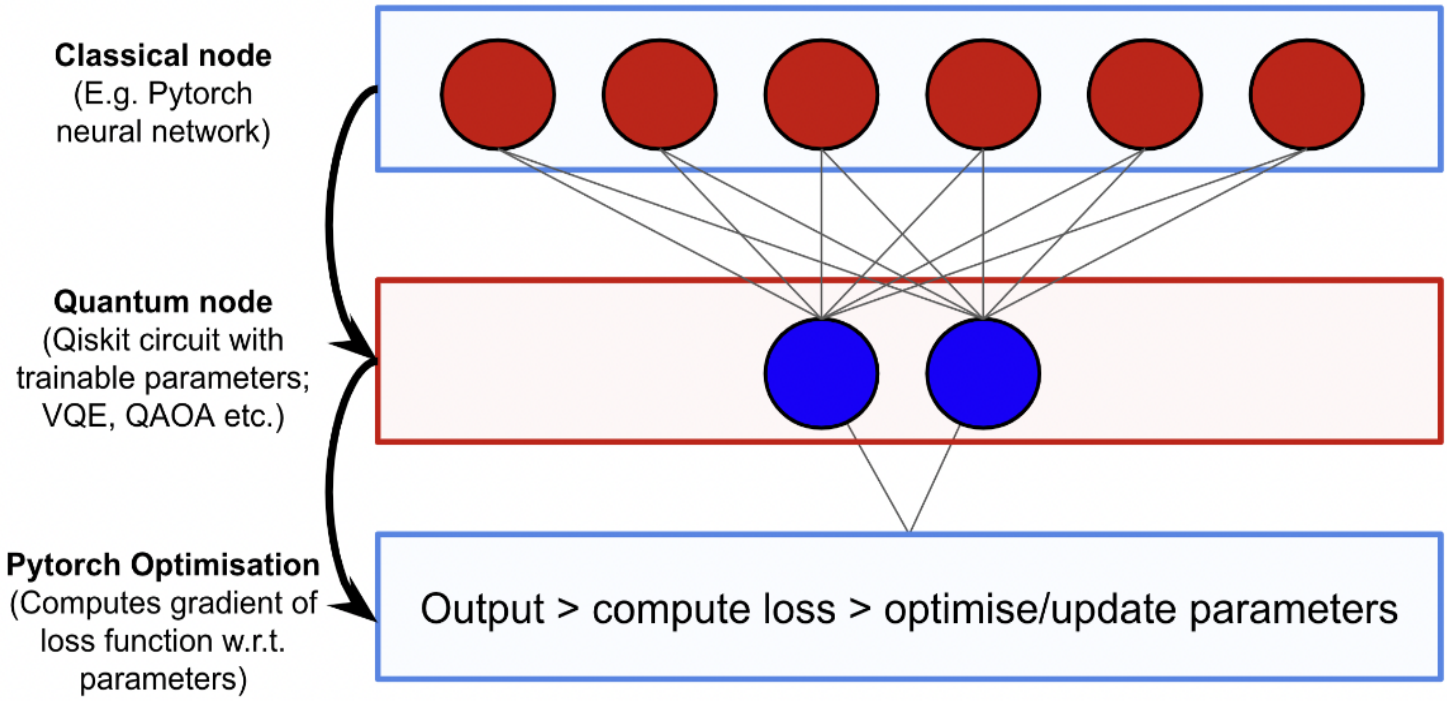


Fig 4.7 Hybrid Quantum Convolution Neural Network

4.7 First Architecture

```
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=1, bias=True)
  (qcsim): Linear(in_features=1, out_features=1, bias=True)
)
```

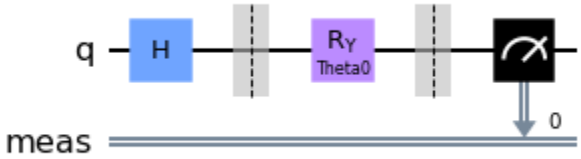


Figure:4.8 First Architecture

In the first experimentation, in place of a Classical neural network, we used Convolutional Neural Network and in the final layer, we used a quantum layer where Figure 4.8 is the architecture we used for training. We have used two convolution layers with filter size 5 *5 and a stride of one; after that, one dropout layer is used with a p-value of 0.5. And finally, two fully connected layers we have used, and after that quantum, the layer is used where we have taken 1 qubit with a single Ry architecture.

Training of Hybrid Quantum Neural Network(First Architecture)

We have trained this above model for 10 epochs with binary cross-entropy loss function and reached the loss of 1.2056 after 10 epochs of iteration, as shown in **Figure 4.9**

```
Training [10%]  Loss: 1.2357
Training [20%]  Loss: 1.2224
Training [30%]  Loss: 1.2170
Training [40%]  Loss: 1.2137
Training [50%]  Loss: 1.2105
Training [60%]  Loss: 1.2072
Training [70%]  Loss: 1.2062
Training [80%]  Loss: 1.2070
Training [90%]  Loss: 1.2057
Training [100%] Loss: 1.2056
```

Figure:4.9- Training of Hybrid Neural Network First Architecture

We have plotted the cross-entropy loss concerning the number of training iterations, as shown in **Fig 4.10**

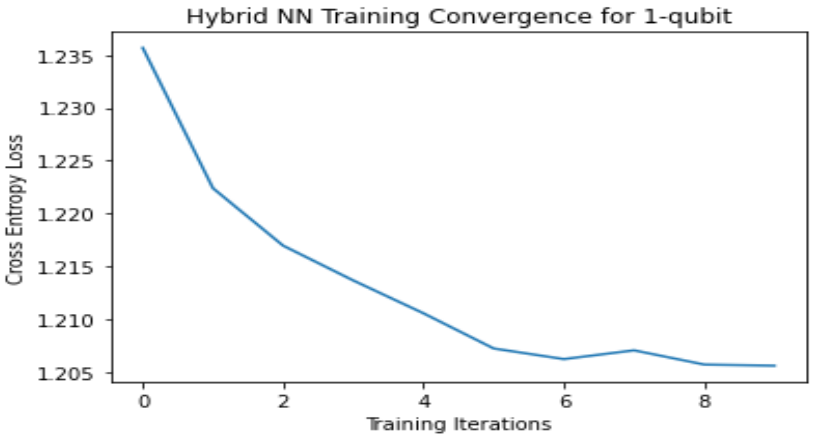


Fig 4.10: Plot of Cross entropy loss function concerning the number of training iterations

Result

Using this architecture to classify the Brain MRI Tumor dataset into no_tumour and glioma_tumour, we have achieved the testing accuracy of 54.63% as shown in the figure below.

Accuracy is a metric that generally describes how the model performs across all classes. It is useful when all classes are of equal importance. It is calculated as the ratio between the number of correct predictions to the total number of predictions we have implemented in code, which is shown in Figure 4.11.

$$Accuracy = \frac{True_{positive} + True_{negative}}{True_{positive} + True_{negative} + False_{positive} + False_{negative}}$$

```
] print("Performance on test data is : {}/{} = {}%".format(accuracy,number,100*accuracy/number))

Performance on test data is : 112/205 = 54.634146341463415%
```

Fig:4.11 Performance on the test data

Output

We have tested the model with some test data and predicted its corresponding classes, and also put its actual classes on top of every image, and represented in **figure 4.12**

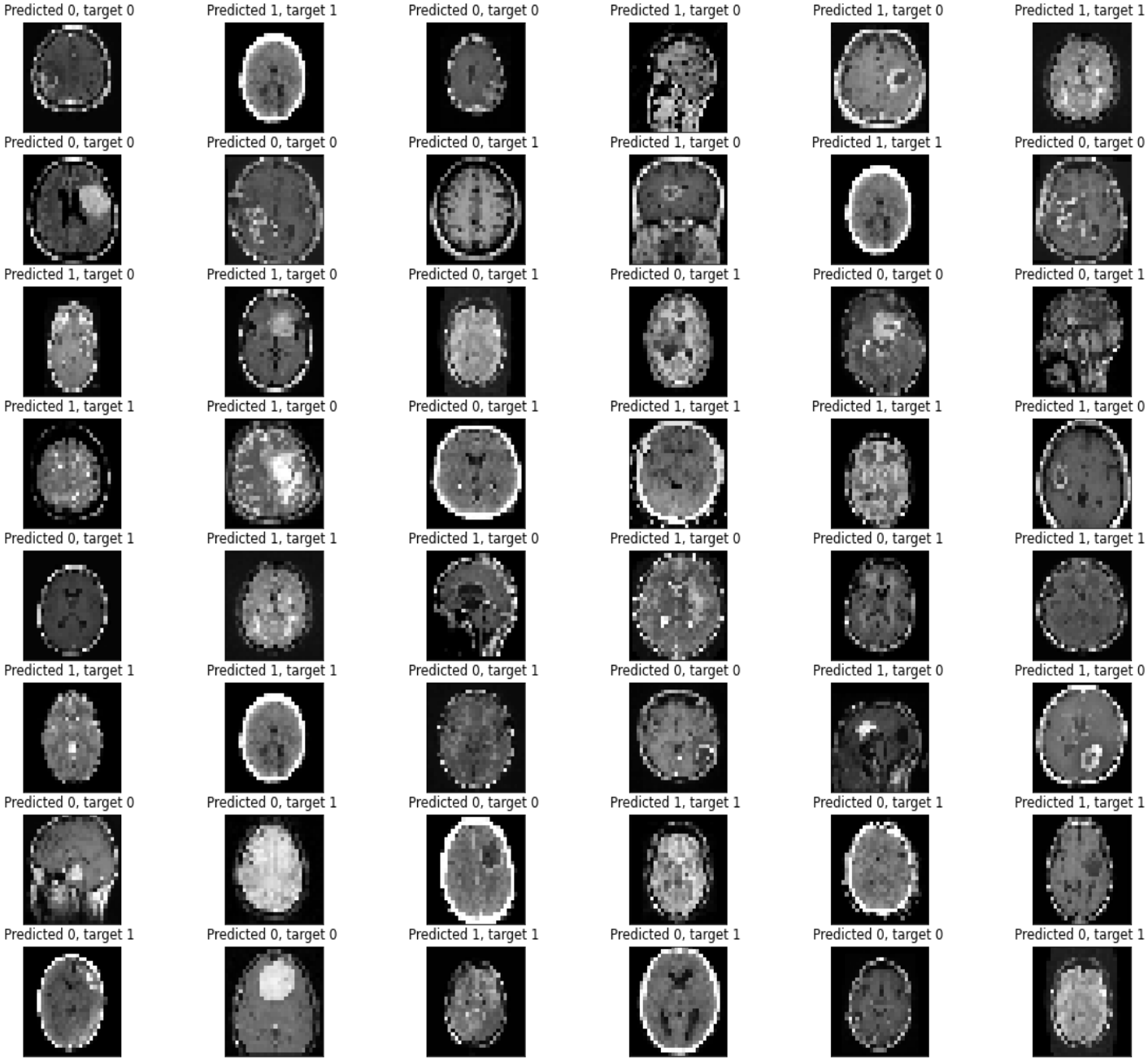


Fig 4.12: Visualization of prediction on test data

4.7.1 First Architecture(Resnet as a Convolution Neural Network)

In place of a normal convolution neural network, Resnet 18 is used, and the final layer is a quantum layer, as shown in **Fig 4.13**.

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

```
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )

  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Quantumnet(
    (pre_net): Linear(in_features=512, out_features=4, bias=True)
    (post_net): Linear(in_features=4, out_features=2, bias=True)
  )
)
```

Fig:4.13: Model Summary of First Architecture

4.7.1 ResNet18

ResNet-18[37] is a convolutional neural network. The network has 18 layers(as shown in **Fig 4.13**), that's why it is called resnet18. As a result, the network has learned rich feature representations for a wide range of images.

Two main types of blocks are used in a ResNet, depending mainly on whether the input/output dimensions are the same or different

1. Identity Block

The identity block is the standard block used in ResNets and corresponds to the case where the input dimension has the same as the output dimension.

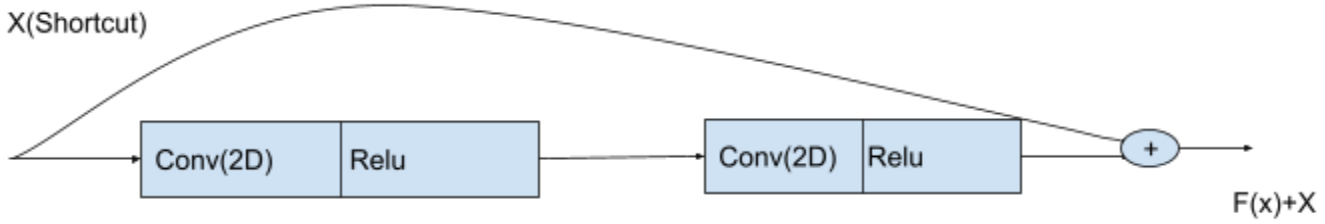


Fig4.14: Identity Block

The upper path is the shortcut path or skip connection, and the lower path is the main path. The first component of the main path: The first Convolution layer(Conv2D) has $F1$ filters of shape (1,1) and a stride of (s,s). Its padding is "valid."Then apply the ReLU activation function, which adds nonlinearity to the output. The second component of the main path: The second Convolution layer(CONV2D) has $F2$ filters of shape(f,f) and a stride of (1,1). Its padding is "same" .Then apply the ReLU activation function. In the final step, the shortcut and the input are added together.

2. The convolution Block

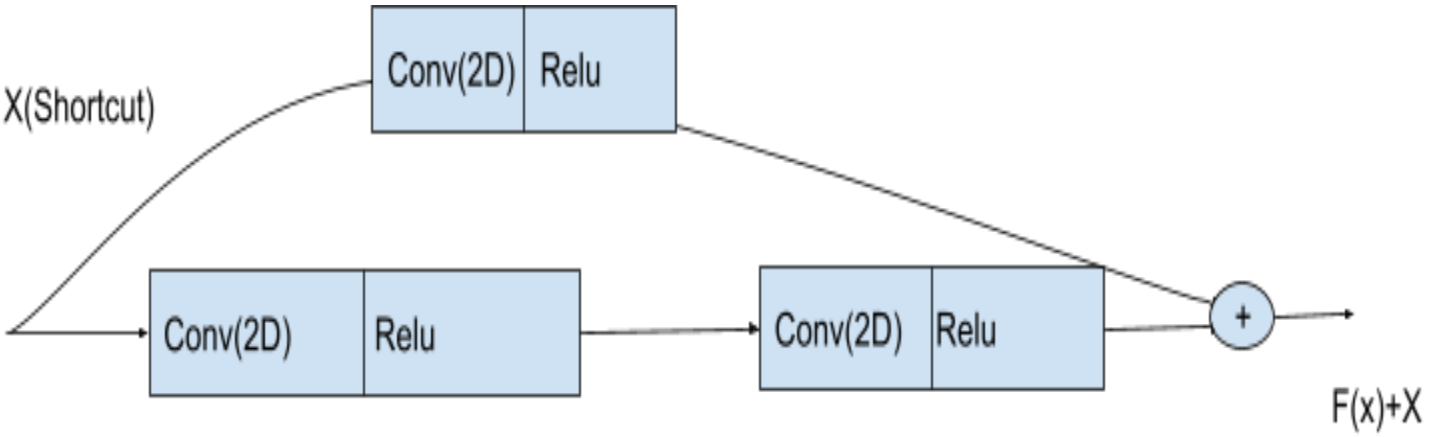


Fig4.15: Convolutional Block in Resnet

The ResNet "convolutional block" is the other type of block. This block type is used when the input and output dimensions don't match up. The difference with the identity block is that there is a CONV2D layer in the shortcut path. The CONV2D layer in the shortcut path is used to resize the input *x*x to a different dimension so that the dimensions match up in the final addition needed to add the shortcut value back to the main path**.**

We now have the necessary blocks to build a very deep ResNet. Figure 4.14 describes in detail the architecture of this neural network.

Training of Hybrid Quantum Neural Network(Resnet as a Classical Neural Network)

We have trained this above model for 30 epochs with binary cross-entropy loss function and reached the loss of 1.1045 after 30 epochs of iteration and training accuracy of 93.53% and testing accuracy of 66.83%, and test loss of 0.7763 as shown in Figure 4.16

```
Phase: val   Epoch: 26/30 Loss: 0.1795 Acc: 0.9427
Phase: val   Epoch: 26/30 Loss: 0.9804 Acc: 0.6341
Phase: val   Epoch: 27/30 Loss: 0.1857 Acc: 0.9337
Phase: val   Epoch: 27/30 Loss: 1.0985 Acc: 0.5951
Phase: val   Epoch: 28/30 Loss: 0.2028 Acc: 0.9304
Phase: val   Epoch: 28/30 Loss: 1.1716 Acc: 0.6000
Phase: val   Epoch: 29/30 Loss: 0.1928 Acc: 0.9296
Phase: val   Epoch: 29/30 Loss: 1.1122 Acc: 0.6000
Phase: val   Epoch: 30/30 Loss: 0.1967 Acc: 0.9353
Phase: val   Epoch: 30/30 Loss: 1.1045 Acc: 0.5854
Training completed in 95m 29s
Best test loss: 0.7763 | Best test accuracy: 0.6683
```

Fig:4.16: Training of Hybrid Quantum Neural Network

Output

We have tested the model with some test data and predicted its corresponding classes, and also put its actual classes on top of every image, and represented in **figure 4.17**

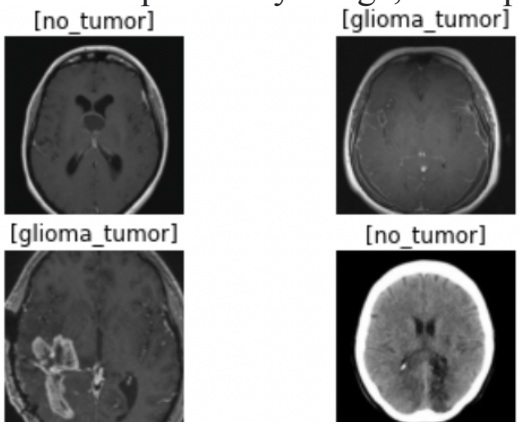


Fig:4.17: Some output of model on test data

4.8 Second Architecture

There are 2 parts to a Hybrid Quantum Classical Neural Network: The classical NN and the quantum circuit. The classical NN is your standard NN with its inputs, weights, and biases. The quantum part is a Parameterized Quantum Circuit with its qubits, rotations, and CNOT gates.

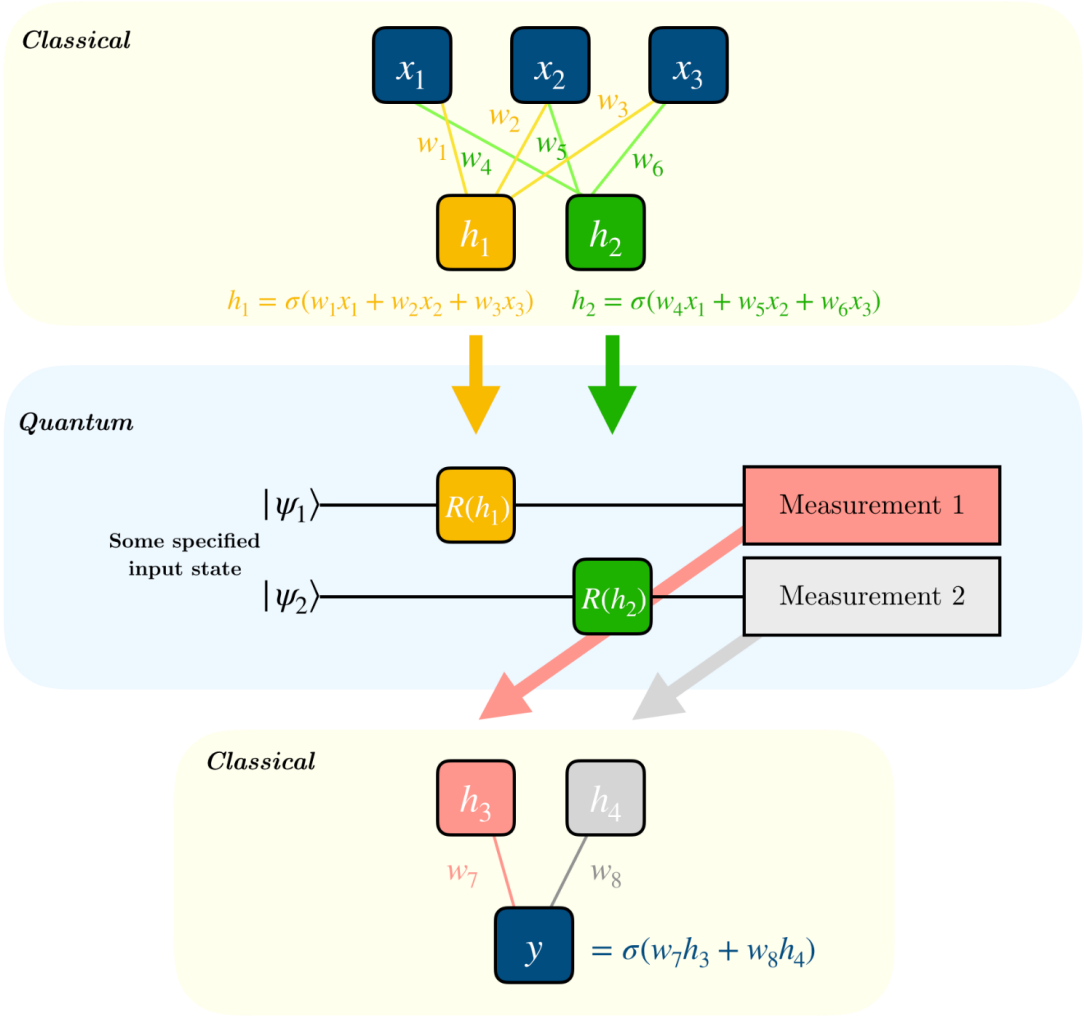


Fig 4.18: Second Architecture

Fig 4.18 shows the second type of Hybrid Quantum Classical Neural Network.

We have a normal neural network at the top that takes in all the data, computes it, and compresses it down to a smaller size. Then we take the outputs of that classical network and use them as the rotations of our qubits. Finally, we take the outputs of those qubits and put them inside another classical neural network, which then outputs the result

We did it a bit differently. For the first part, I didn't just use classical neural networks. We used a Resnet18 as in the previous architecture with some linear layers attached to the end (with dropout and Relus), and after that, there is a quantum layer and, again, some classical linear layer. As the Resnet18 layer is the same as the previous, we have only shown a few last layers.

Quantum Architecture with 3 qubit

We have created the Quantum Architecture with 3 qubits and have 9 trainable parameters.

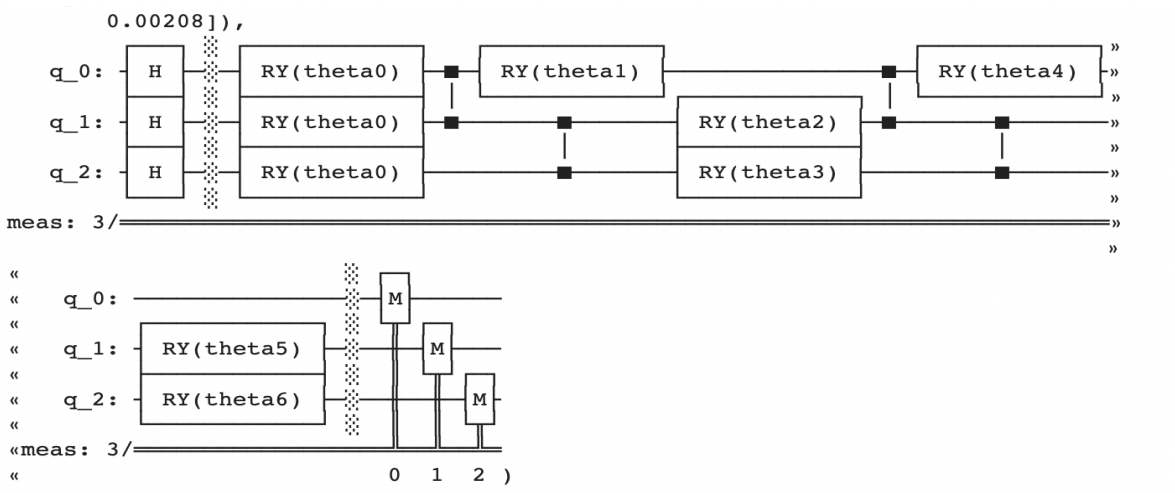


Fig 4.19: Quantum Architecture

Training of the model

We have trained this above model for 30 epochs with binary cross-entropy loss function and reached the training loss as 0.3686 after 30 epochs of iteration and training accuracy of 94.10% and testing accuracy of 65.85%, and test loss of 0.6479 as shown in Figure 4.20

```
Phase: Testing   Epoch: 24/30 Loss: 0.6999 Acc: 0.6000
Phase: train Epoch: 25/30 Loss: 0.3728 Acc: 0.9386
Phase: Testing   Epoch: 25/30 Loss: 0.7038 Acc: 0.6049
Phase: train Epoch: 26/30 Loss: 0.3692 Acc: 0.9410
Phase: Testing   Epoch: 26/30 Loss: 0.7031 Acc: 0.6098
Phase: train Epoch: 27/30 Loss: 0.3708 Acc: 0.9386
Phase: Testing   Epoch: 27/30 Loss: 0.6982 Acc: 0.6098
Phase: train Epoch: 28/30 Loss: 0.3790 Acc: 0.9353
Phase: Testing   Epoch: 28/30 Loss: 0.6947 Acc: 0.6098
Phase: train Epoch: 29/30 Loss: 0.3730 Acc: 0.9402
Phase: Testing   Epoch: 29/30 Loss: 0.7076 Acc: 0.5951
Phase: train Epoch: 30/30 Loss: 0.3686 Acc: 0.9410
Phase: Testing   Epoch: 30/30 Loss: 0.6948 Acc: 0.6049
Training completed in 108m 42s
Best test loss: 0.6479 | Best test accuracy: 0.6585
```

Fig 4.20: Training of Hybrid Quantum Neural Network(HQNN)

Result

This model is trained for 30 epochs and achieved a training accuracy of 94.10% and a testing accuracy of 65.85%. Some of the output of the model is shown below.

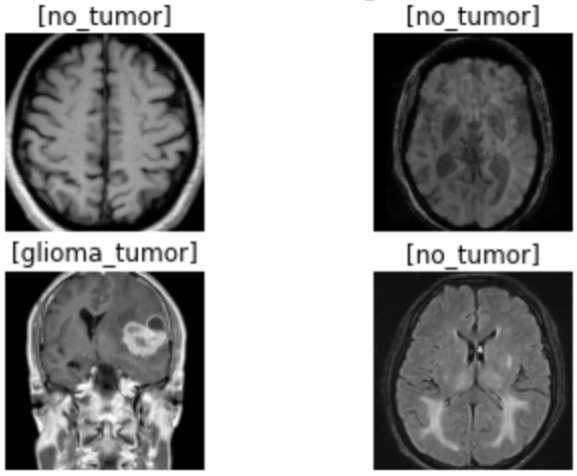


Fig:4.21 Visualization of prediction on test data

Chapter 5

MNIST classification

The second problem solved in this thesis is the well-known MNIST classification problem. This chapter aims to experiment with the different architecture described above to classify the MNIST Handwritten dataset.

5.1 About the dataset

The Modified National Institute of Standards and Technology (MNIST) database of handwritten digits [15] contains 70 000 grayscale, centered images normalized into the size of 28×28 with a value from range 0 to 255, where 0 represents the white color (background), and 255 represents the black color (foreground). The dataset is labeled by digits corresponding to the images and is initially divided into training and test set (60 000 and 10 000 examples, respectively). The dataset was constructed based on NIST's Special Databases 1 and 3 (*SD-1*, *SD-3*) by taking 30 000 patterns for the training set and 5 000 patterns for the test set from both databases and ensuring the writer's groups are disjoint, normalizing the binary images from bi- nary, black-white, to grayscale and centering them. The set contains digits written by approximately 250 people from among the US Census Bureau employees (SD-3) and US high school students (SD-1). The data was accessed using the Keras API [8], being a part of the TensorFlow library [1]

5.2 Quantum Convolutional Neural Network

Firstly we have experimented with the QCNN architecture described in the previous chapter in Section 4.4. This model is trained on the MNIST dataset for 10 classes. We have trained the model for 30 epochs and reached the training accuracy of 100% and testing accuracy of 66%

Output after Preprocessing using Quantum Convolutional module

First, the dataset's images are convoluted using the Quantum Convolution module, which is considered a preprocessing stage. Fig 4.22 shows input and its corresponding output. Since MNIST images are single images, every input image has 4 channel output.

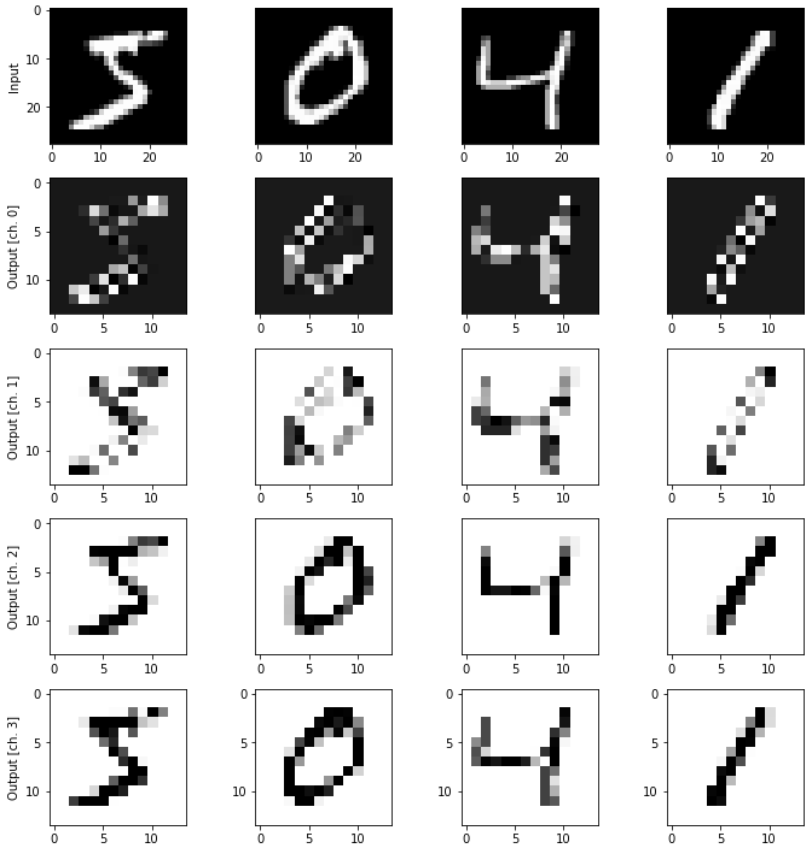


Fig4.22: Output after preprocessing of MNIST Data

5.2.1 Training of Quantum Convolution Neural Network

We have trained the QCNN model for 15 epochs and cross-entropy as a loss function, and after training, we have achieved the training accuracy of 98% and testing accuracy of 56%, as shown in figure 4.6

```
Epoch 1/15
13/13 - 1s - loss: 18.6996 - accuracy: 0.2800 - val_loss: 11.9263 - val_accuracy: 0.2667 - 586ms/epoch - 45ms/step
Epoch 2/15
13/13 - 0s - loss: 8.0351 - accuracy: 0.5800 - val_loss: 11.0381 - val_accuracy: 0.3000 - 130ms/epoch - 10ms/step
Epoch 3/15
13/13 - 0s - loss: 7.2531 - accuracy: 0.5600 - val_loss: 7.8073 - val_accuracy: 0.3667 - 129ms/epoch - 10ms/step
Epoch 4/15
13/13 - 0s - loss: 2.2402 - accuracy: 0.7600 - val_loss: 8.6712 - val_accuracy: 0.5667 - 149ms/epoch - 11ms/step
Epoch 5/15
13/13 - 0s - loss: 2.0267 - accuracy: 0.8000 - val_loss: 13.4618 - val_accuracy: 0.2333 - 133ms/epoch - 10ms/step
Epoch 6/15
13/13 - 0s - loss: 1.2580 - accuracy: 0.8000 - val_loss: 6.0682 - val_accuracy: 0.6333 - 127ms/epoch - 10ms/step
Epoch 7/15
13/13 - 0s - loss: 0.8073 - accuracy: 0.9000 - val_loss: 6.3492 - val_accuracy: 0.4333 - 125ms/epoch - 10ms/step
Epoch 8/15
13/13 - 0s - loss: 0.3128 - accuracy: 0.9600 - val_loss: 8.2024 - val_accuracy: 0.4333 - 135ms/epoch - 10ms/step
Epoch 9/15
13/13 - 0s - loss: 0.2515 - accuracy: 0.9400 - val_loss: 5.2641 - val_accuracy: 0.5333 - 127ms/epoch - 10ms/step
Epoch 10/15
13/13 - 0s - loss: 0.1540 - accuracy: 0.9600 - val_loss: 7.7288 - val_accuracy: 0.4667 - 139ms/epoch - 11ms/step
Epoch 11/15
13/13 - 0s - loss: 0.2675 - accuracy: 0.9600 - val_loss: 11.4246 - val_accuracy: 0.4667 - 151ms/epoch - 12ms/step
Epoch 12/15
13/13 - 0s - loss: 0.4383 - accuracy: 0.9200 - val_loss: 6.9473 - val_accuracy: 0.5000 - 139ms/epoch - 11ms/step
Epoch 13/15
13/13 - 0s - loss: 0.0384 - accuracy: 0.9800 - val_loss: 5.7850 - val_accuracy: 0.5667 - 176ms/epoch - 14ms/step
Epoch 14/15
13/13 - 0s - loss: 7.4469e-05 - accuracy: 1.0000 - val_loss: 6.5650 - val_accuracy: 0.5667 - 136ms/epoch - 10ms/step
Epoch 15/15
13/13 - 0s - loss: 7.4143e-04 - accuracy: 1.0000 - val_loss: 6.8242 - val_accuracy: 0.5667 - 126ms/epoch - 10ms/step
```

Fig4.22: Training of Model for 15 epoch

We have finally plotted the test accuracy and the test loss with respect to the number of training epochs for Architecture with Quantum Layer and Without Quantum Layer. With Quantum Layer, we have achieved the testing accuracy of 65%, and without a quantum layer, we have achieved the accuracy of 63%, as shown in figure 4.23

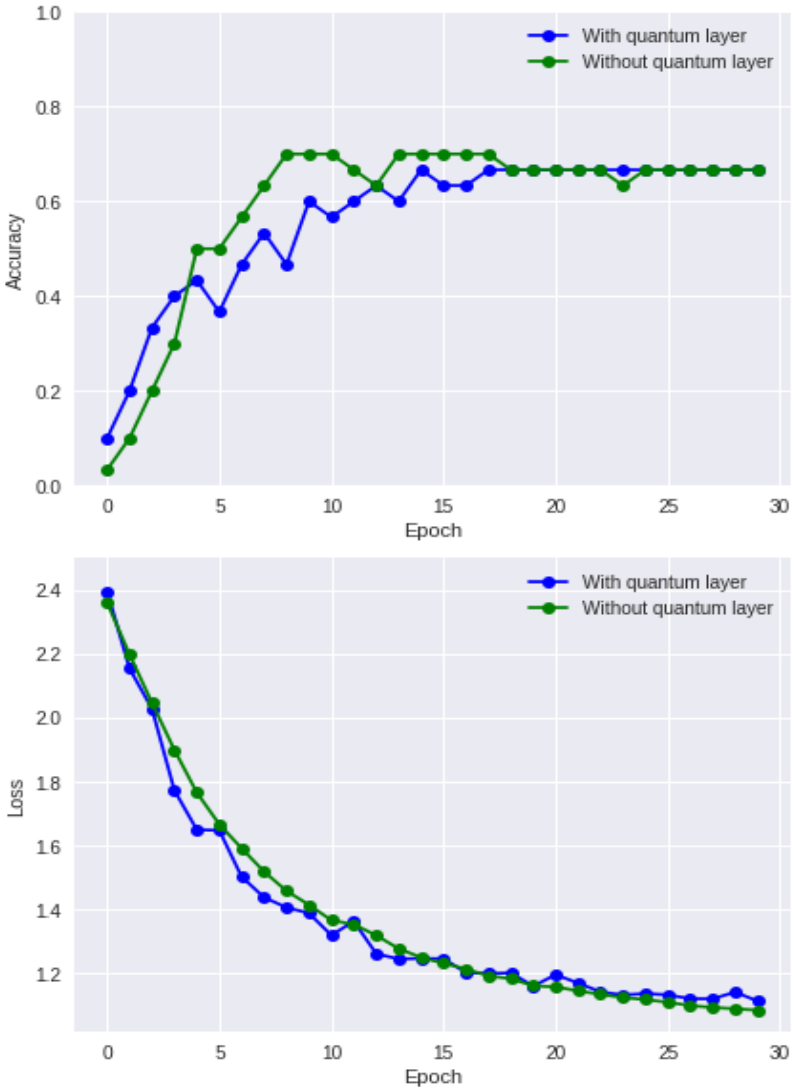


Fig:4.23: Plot of Accuracy with respect to epoch and loss with respect to epoch With quantum and without a quantum layer

5.3 Hybrid Quantum Neural Network

In the second experiment, we trained the Hybrid Quantum Neural Network(First Architecture) described in the previous chapter in section 4.7 with the MNIST dataset for 10 epochs and achieved a testing accuracy of 85%

First Architecture Training on MNIST Dataset

We have trained this above model for 10 epochs with binary cross-entropy loss function and reached the loss of 3.3046 after 10 epochs of iteration, as shown in Figure 4.24

```
Training [10%]  Loss: 3.4093
Training [20%]  Loss: 3.3534
Training [30%]  Loss: 3.3319
Training [40%]  Loss: 3.3261
Training [50%]  Loss: 3.3179
Training [60%]  Loss: 3.3113
Training [70%]  Loss: 3.3075
Training [80%]  Loss: 3.3099
Training [90%]  Loss: 3.3050
Training [100%] Loss: 3.3046
```

Fig4.24: Training of HQNN for 10 epoch

We have plotted the cross-entropy loss concerning Training iterations. Fig 2.5 shows the plot.

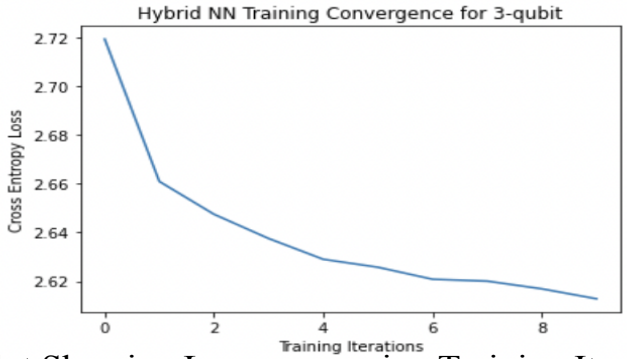


Fig4.25:Plot Showing Loss concerning Training Iteration

Output

We have tested the model with some test data and predicted its corresponding classes, and also put its actual classes on top of every image and represented in figure 4.26

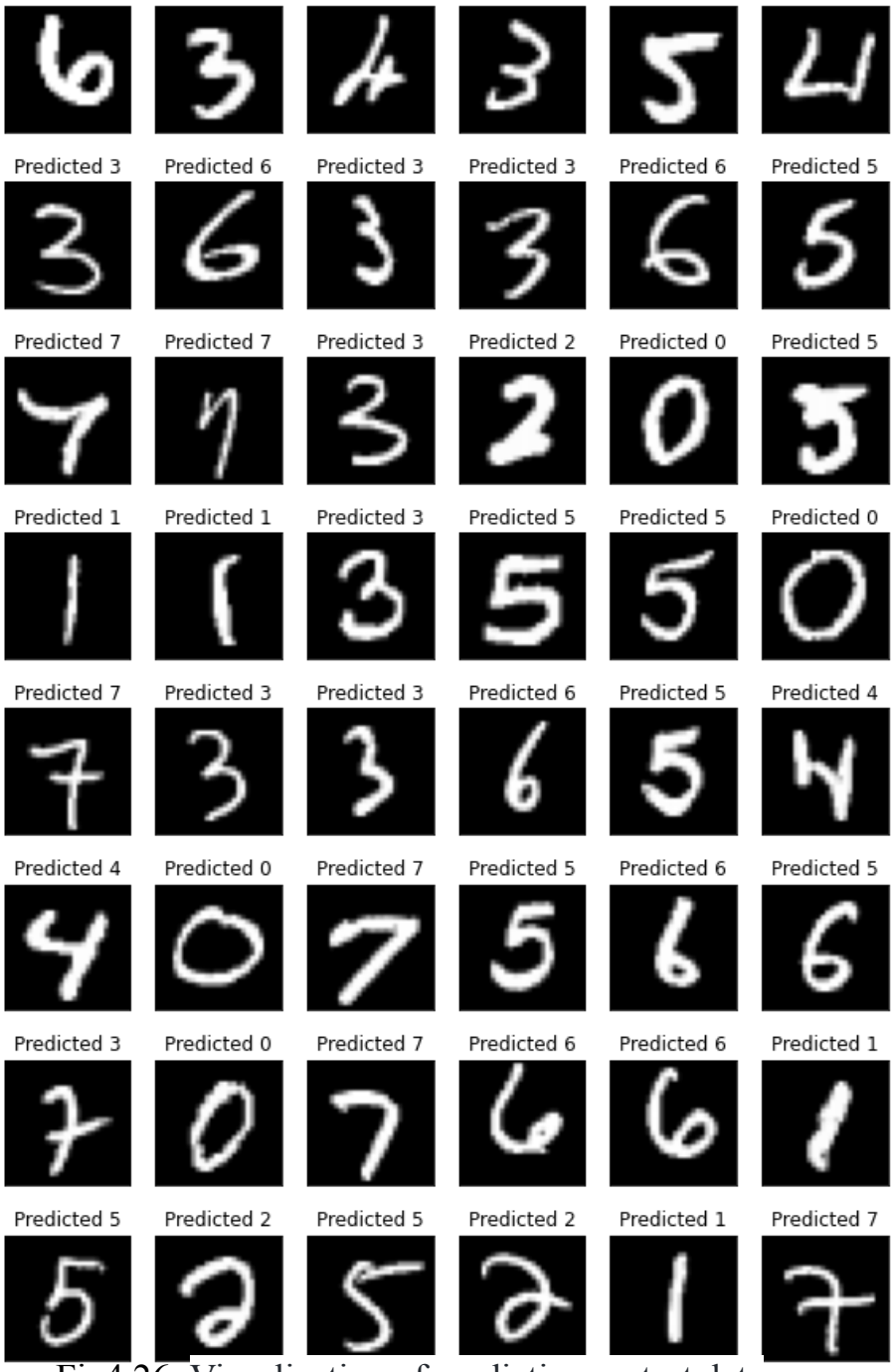


Fig4.26: Visualization of prediction on test data

5.4 Second Architecture

In the third experiment, we trained the Hybrid Quantum Neural Network(Second architecture) with the MNIST dataset and achieved a testing accuracy of 82%.

In this architecture, there are four parts of the model. The first one is the classical part, where two convolution and max-pooling layers are there with dropout at the end. The second part is the middle part, where two dense layers are, and the third part is the Quantum layer described in **Fig 4.28**. And then the last part is the dense layer and softmax layer, as shown in **fig:4.27**.

```
Net(
  (classical): Sequential(
    (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Dropout(p=0.5, inplace=False)
  )
  (middel): Sequential(
    (0): Linear(in_features=256, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=7, bias=True)
  )
  (quantum): Hybrid()
  (end): Sequential(
    (0): Linear(in_features=8, out_features=5, bias=True)
    (1): Sigmoid()
  )
)
```

Fig 4.27: Model Summary of Second Architecture

5.5 Quantum Architecture with 3 qubit

We have created the Quantum Architecture with 3 qubits and have 9 trainable parameters.

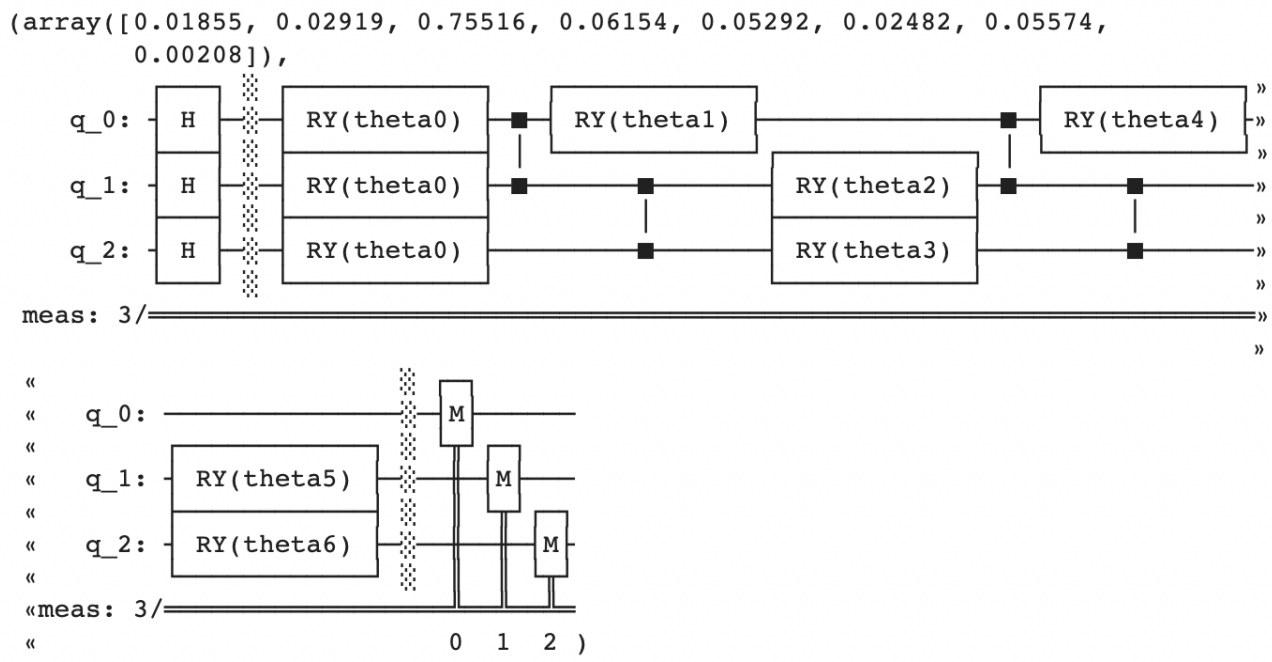


Fig 4.27: Quantum Architecture

We have plotted the cross-entropy loss concerning Training iterations. Fig 2.5 shows the plot.

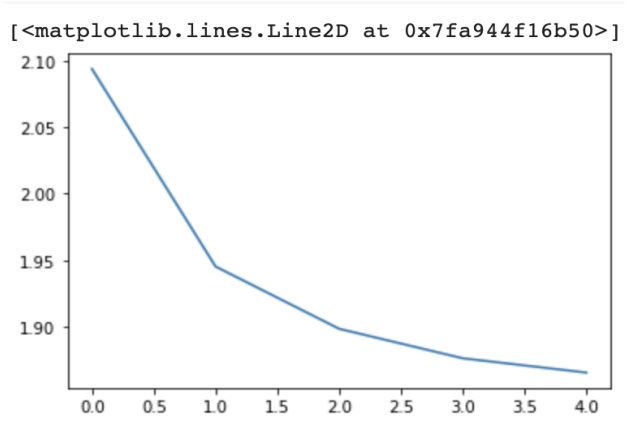


Fig 4.28: Visualization of minimization of loss

5.6 Output

We have tested the model with some test data and predicted its corresponding classes, and also put its actual classes on top of every image and represented in figure 4.29

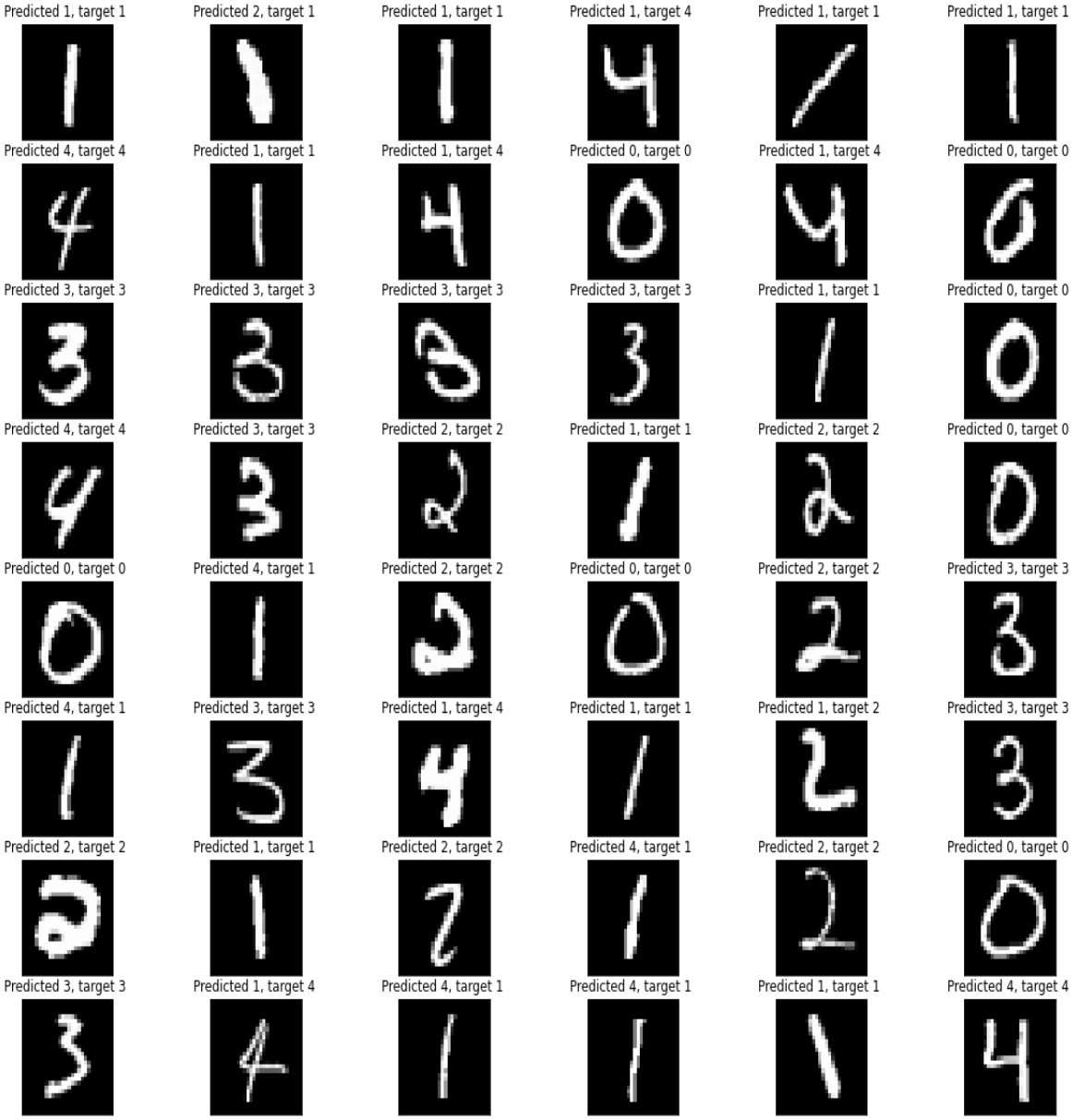


Fig4.29: Visualization of prediction on test data

5.7 Conclusions

We have trained different architectures of Quantum Neural Network with two datasets, namely Brain Tumor MRI Dataset and MNIST Dataset. We have compared different architectures, namely Quantum Convolution Neural Network, Hybrid Quantum Neural Network(Fist Architecture), and Second Architecture.

Table 1:Accuracy Table

Models	**BRAIN TUMOR MRI DATASET (Training accuracy)**	**BRAIN TUMOR MRI DATASET (Testing accuracy)**	**MNIST (Training accuracy)**	**MNIST (Testing Accuracy)**
QUANTUM CONVOLUTION NEURAL NETWORK	95%	66%	96%	66.67%
HYBRID QUANTUM NEURAL NETWORK(FIRST ARCHITECTURE)	85%	54%	94%	85%
HYBRID QUANTUM NEURAL NETWORK(FIRST ARCHITECTURE **RESNET as a classical Neural Network**)	93.53%	65.85%	NA	NA

HYBRID QUANTUM NEURAL NETWORK(SECOND ARCHITECTURE)	94.10%	66.83%	93.5%	82%

After doing all experimentation, we have found out that Hybrid Quantum Neural Network(Second Architecture) performs well in the Brain MRI Tumor dataset with a training accuracy of 94.105 and testing accuracy of 66.83%, and Hybrid Neural Network(First Architecture) performs well on MNIST dataset with training accuracy of 94% and testing accuracy of 85%.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasude- van, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G., Buell, D. A., et al. (2019). Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510.

[3] Assessor's Office at King County (2017). Residential glossary of terms. https://info.kingcounty.gov/assessor/esales/ Glossary.aspx?type=r. Date of access: 2021-01-23.

[4] Azevedo, F. A., Carvalho, L. R., Grinberg, L. T., Farfel, J. M., Ferretti, R. E., Leite, R. E., Filho, W. J., Lent, R., and Herculano-Houzel, S. (2009). Equal numbers of

neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541.

[5] Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., and Lloyd, S. (2017). Quantum machine learning. *Nature*, 549(7671):195–202.

[6] Broughton, M., Verdon, G., McCourt, T., Martinez, A. J., Yoo, J. H., Isakov, S. V., Massey, P., Niu, M. Y., Halavati, R., Peters, E., et

(2020). Tensorflow quantum: A software framework for quantum machine learning. *arXiv preprint arXiv:2003.02989*.

[7] Buduma, N. and Locascio, N. (2017). *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms*. " O'Reilly Media, Inc.".

[8] Chollet, F. et al. (2015). Keras. https://keras.io.

[9] Farhi, E. and Neven, H. (2018). Classification with quantum neural networks on near term processors.

[10] Fujisawa, T., Hayashi, T., Jung, S. W., Jeong, Y.-H., and Hi- rayama, Y. (2006). Single-electron charge qubit in a double quantum dot. In *Quantum Computing in Solid State Systems*, pages 279–287. Springer.

[Gawron et al., 2016] Gawron, P., Cholewa, M., and Kara, K. (2016). *Rewolucja stanu– fantastyczne wprowadzenie do informatyki kwantowej*. Instytut Informatyki Teore- tycznej i Stosowanej Polskiej Akademii Nauk.

[11] Google LLC (2021). Google colaboratory. https://colab. research.google.com/.

[12] Josephson, B. (1974). The discovery of tunneling supercurrents.

Reviews of Modern Physics, 46:251–254.

[13] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[14] Le, P. Q., Dong, F., and Hirota, K. (2011). A flexible representation of quantum images for polynomial preparation, image compression, and processing operations. *Quantum Information Processing*, 10(1):63–84.

[15] LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http: // yann. lecun. com/*

[16] Deweerdt, S. (2019). Deep connections. *Nature*, 571(7766):S6–S8.

[17] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133

[18] Sharma, S. (2017). Activation functions in neural networks. Towards Data Science, 6.

[19] Rao, S. S. (2019). Engineering optimization: theory and practice. John Wiley & Sons.

[20] Nielsen, M. A. and Chuang, I. (2002). Quantum computation and quantum information.

[21] Pla, J. J., Tan, K. Y., Dehollain, J. P., Lim, W. H., Morton, J. J., Jamieson, D. N., Dzurak, A. S., and Morello, A. (2012). A single-atom electron spin qubit in silicon. Nature, 489(7417):541–545.

[22] Mosakowski, J., Owen, E., Ferrus, T., Williams, D., Dean, M., and Barnes, C. (2016). An optimal single-electron charge qubit for solid-state double quantum dots. arXiv preprint arXiv:1603.05112.

[23] Zhou, Z.-Q., Lin, W.-B., Yang, M., Li, C.-F., and Guo, G.-C. (2012). Realization of reliable solid-state quantum memory for photonic polarization qubit. Physical review letters, 108(19):190505.

[24] Riedl, S., Lettner, M., Vo, C., Baur, S., Rempe, G., and Dürr, S. (2012). Bose-einstein condensate as a quantum memory for a photonic polarization qubit. Physical Review A, 85(2):022318.

[25] Morton, J. J., Tyryshkin, A. M., Brown, R. M., Shankar, S., Lovett, B. W., Ardavan, A., Schenkel, T., Haller, E. E., Ager, J. W., and Lyon, S. (2008). Solid-state quantum memory using the 31 p nuclear spin. Nature, 455(7216):1085–1088.

[26] Lucatto, B., Koda, D. S., Bechstedt, F., Marques, M., and Teles, L. K. (2019). Charge qubit in van der waals heterostructures. Physical Review B, 100(12):121406. [McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133.

[27] Preskill, J. (2018). Quantum computing in the nisq era and beyond. Quantum, 2:79.

[29] Quantum AI team and collaborators (2020a). Cirq.

[30] Quantum AI team and collaborators (2020b). qsim. 39 BIBLIOGRAPHY

[Rao, 2019] Rao, S. S. (2019). Engineering optimization: theory and practice. John Wiley & Sons.

[31] Yang, Y., Yan, L. F., Zhang, X., Han, Y., Nan, H. Y., Hu, Y. C., et al. (2018). Glioma grading on conventional MR images: a deep learning study with transfer learning. *Front. Neurosci.* 12:804. DOI: 10.3389/fnins.2018.00804

[32] Sung, H., Ferlay, J., Siegel, R. L., Laversanne, M., Soerjomataram, I., Jemal, A., et al. (2021). Global cancer statistics 2020: gLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA Cancer J. Clin.* 71, 1–41. DOI: 10.3322/caac.21660

[33] Gumaei, A., Hassan, M. M., Hassan, M. R., Alelaiwi, A., and Fortino, G. (2019). A hybrid feature extraction method with regularized extreme learning machine for brain tumor classification. *IEEE Access* 7, 36266–36273. DOI: 10.1109/ACCESS.2019.2904145

[34] Zeng, K., Zheng, H., Cai, C., Yang, Y., Zhang, K., and Chen, Z. (2018). Simultaneous single-and multi-contrast super-resolution for brain MRI images based on a convolutional neural network. *Comput. Biol. Med.* 99, 133–141. DOI: 10.1016/j.compbiomed.2018.06.010

[35] Maxwell Henderson, Samriddhi Shakya, Shashindra Pradhan, Tristan Cook. "Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits." arXiv:1904.04767, 2019.