MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION
FEDERAL STATE AUTONOMOUS INSTITUTION OF HIGHER EDUCATION
«NOVOSIBIRSK NATIONAL RESEARCH STATE UNIVERSITY»
(NOVOSIBIRSK STATE UNIVERSITY, NSU)

Department of Mechanics and Mathematics

Chair of Computational System

Field of Study: 01.04.02 Applied Mathematics and Informatics. Profile: Big Data Analytics

MASTER THESIS

Ravi Kumar

Thesis title: **Discovering Applicability of Quantum Neuron Models in Classification of Medical Images**

«Admitted to Defense»

Head of Chair,

.....

.... /…….

«……» …….. 2020

Scientific Supervisor

Head of the SDAML Laboratory NSU,

Pavlovsky E.N. / …….

«……» …….. 2020

Date of Defense: «……» …….. 2020

Novosibirsk, 2020

Contents

1 Chapter

1.1 Abstract

This research will introduce you with possibility of Qubit Neural Network(Qubit NN) in the filed of computer vision and be specifically in medical image analysis. We are going to discuss Qubit NN in more details in further section. We introduced Qubit NN layer alone in Neural Network(NN) and also explored the possibility of combination of real-valued Linear Neuron Model, real-valued Convolutions and with complex-valued convolutions repectively. There is no other research exist yet which explored the combination of these networks, but we hope to find good and some outstanding results. Development of neural network is one of the task, other is validation of developed neural network. So, for validating our results, we used same layer combination real-valued network with same optimizer and same error function, it's also helped us in comparison base study of Qubit NN. Initial validation of constructed NN, we used MNIST dataset(because of it's popularity and it's very well managed for computer vision tasks) then for medical dataset, we explore x-ray dataset and MRI dataset that found from various completions and available open source for public research uses.

1.2 Introduction

Machine Learning(ML) and Artificial Intelligence(AI) application is one of the fastest growing field in complex medical data analysis. From the new research has revealed that AI growth is faster than the growth predicted as per Moore's Law [1]. AI is showing very promising results in making complex decisions very easy in Medical Science and other areas. In recent years many of the researchers came forward and contributed in medical data analysis and got significant results. This progress is showing we can do complex and costly medical data analysis at very low cost and can reduce analysis time significantly. AI has different research areas in Medical Science. One of the most popular medical science application of AI is Medical Image analysis(MIA). MIA are used various areas of medical science, like Surgical Planning, Surgical Guidance, Neuroscience, different kind of cancer detection etc. This research is more focused on computer vision(CV) tasks in medical data analysis using Qubit NN [2]. If we talk about current trends in MIA, then you will profound knowledge that most of the researcher focused there research on MIA using well formulated AI and ML tools which are based on real-valued Neural Network(NN) and real-valued Convolution Neural Networks(CNN). Currently, most common architecture in MIA is UNet. [2]The paper is showing very promising results and future in AI, so it's worth of exploring. Qubit NN is complex in nature and works on the principle of Quantum bits. Because Qubit NN working principal, we have to describe dataset in the form of Quantum bits, and for this purpose authors used complex representation and complex computation. As Qubit NN works on complex-valued principle then it also becomes very obvious to explore the complex-valued convolution(because in-depth knowledge of current researches showing that Convolutional Neural Network(CNN) plays a great role in computer vision tasks and can increase performance and accuracy of AI model).

1.3 Literature Review

In 1985 Deutsch [3] developed a theoretical machine and named Universal Quantum Turing Machine (UQTM), that is, a machine based on quantum theory capable of performing computations, and showed that such machine was a generalization of a universal Turing machine. It is also shown in [3] that a quantum computer (QC), that is, a physical system capable of performing computations according to the rules of quantum mechanics, can perform certain tasks faster than its classical counterpart. Shor [4] showed concrete problems where such speed-up is possible.

Among QC properties, we find:

1. Superposition of states. The basic component of a QC is a qubit, that is, a physical entity (such as an electron or a photon) that can be mathematically represented as a vector in a 2D Hilbert space H^2 .The general form of a qubit is,

   $|\Psi\rangle=\alpha|x\rangle+\beta|y\rangle$

   where α and β are complex numbers constrained by $\alpha^2+\beta^2=1$ and $\{|\alpha\rangle,|\beta\rangle\}$ is an arbitrary basis of H^2 [5]. Thus, $|\Psi\rangle$ is a superposition of states $|\alpha\rangle$ and $|\beta\rangle$ and therefore $|\Psi\rangle$ can be prepared in an infinite number of ways by varying the values of $|\alpha\rangle$ and $|\beta\rangle$. In contrast, classical computers measure bit values using only one basis, {0,1} and the only two possible states are those that correspond to the measurement outcomes, 0 or 1.
2. Entanglement is a special correlation among quantum systems that has no paragon in classical systems. Entanglement is seen to be at the heart of Quantum Information Processing(QIP) unique properties, and an example of it is its role in Quantum Teleportation [5].
3. Superposition of states. The basic component of a QC is a qubit, that is, a physical entity (such as an electron or a photon) that can be mathematically represented as a vector in a 2D Hilbert space H^2 .The general form of a qubit is,

   $|\Psi\rangle=\alpha|x\rangle+\beta|y\rangle$

   where α and β are complex numbers constrained by $\alpha^2+\beta^2=1$ and $\{|\alpha\rangle,|\beta\rangle\}$ is an arbitrary basis of H^2 [5]. Thus, $|\Psi\rangle$ is a superposition of states $|\alpha\rangle$ and $|\beta\rangle$ and therefore $|\Psi\rangle$ can be prepared in an infinite number of ways by varying the values of $|\alpha\rangle$ and $|\beta\rangle$. In contrast, classical computers measure bit values using only one basis, {0,1} and the only two possible states are those that correspond to the measurement outcomes, 0 or 1.

4. Entanglement is a special correlation among quantum systems that has no paragon in classical systems. Entanglement is seen to be at the heart of Quantum Information Processing(QIP) unique properties, and an example of it is its role in Quantum Teleportation [5].

QIP, which exploits quantum-mechanical phenomena such as quantum superpositions and quantum entanglement [[3] [6]], allows one to overcome the limitations of classical computation and reaches higher computational speed for certain problems like factoring large numbers [[4] [7]], searching an unsorted database [8], boson sampling [9], quantum simulation [10], solving linear systems of equations [11], [6]and machine learning [12]. These unique quantum properties, such as quantum superposition and quantum parallelism, may also be used to speed up signal and data processing [13]. For quantum image processing, quantum image representation (QImR) plays a key role, which substantively determines the kinds of processing tasks and how well they can be performed. A number of QImRs [14] have been discussed. Yao firstly introduce the basic framework of quantum image processing, then present the experimental demonstration for several basic image transforms on a nuclear magnetic resonance (NMR) quantum information processor[15]. Yao also propose a highly efficient quantum edge detection algorithm, along with the proof-of-principle numerical and experimental demonstrations[15]. In 2005 Kuoda proposed Qubit NN[2] and Qubit NN showed quite good result in experiment on linear analysis and as well as Image Processing. As per Kuoda, his other future work will focus on the use of Qubit NN in practical applications with more neural parameters to establish this method concretely, and want compare Qubit NN with Hypercomplex-valued neural networks including Complex-valued NN in detail[2]. We selected Qubit NN for our experiment, based on previous results and used of Kuoda research in other explorations.

CNN also plays a very significant role in computer vision tasks. So for building a network with convolution we explored the complex-valued convolution because. Nita[16] is attracted the attention of researchers on complex numbers based NN due their potential to enable easier optimization, better generalization characteristics [17] and [18] show that using complex numbers in recurrent neural networks(RNN) allows the network to have a richer representational capacity. Guberman[19] researched on the complex convolution found similar test accuracy 97.3% compare to real-valued network 97.5% with better test loss on SIMCEP dataset. [20] Also researched complex networks for computer vision tasks and outperformed real-valued network with less number of parameters 8.8M to 10M and also in accuracy 72.9% compared to real-valued NN 69.6%. Zhang and their colleagues [21] developed complex-valued convolution and verified classification tasks on SAR image dataset and got good result compare with real-valued NN outperforming classification accuracy 97.7% to 97.2%.

These researches are showing promising results and development of AI and Neural Network in complex-valued and these motivated me to work in this area.

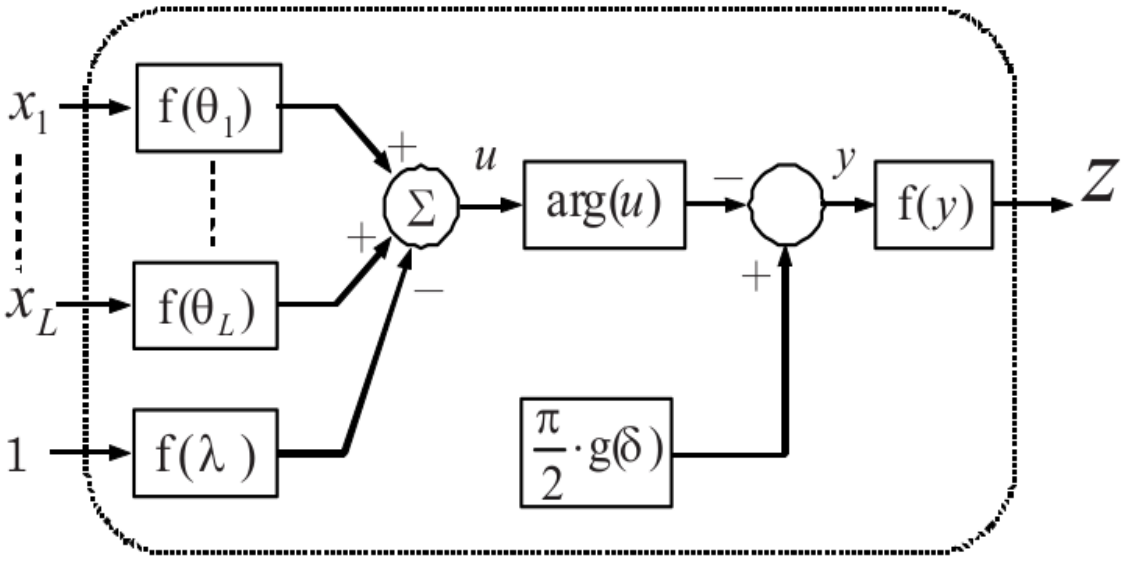
2 Chapter : Neural Network and Dataset

2.1 Qubit NN

The Qubit NN [2] is a neuron model inspired by quantum physics and quantum computing: its neuron states are connected to quantum states, and transitions between neuron states are based on operations derived from quantum logic gates. To make the connection between the neuron states and the quantum states and assume that the state of a firing neuron is defined as qubit state $|1\rangle$, the state of a non-firing neuron is defined as qubit state $|0\rangle$, and the state of an arbitrary neuron is the coherent superposition of the two. In quantum computers, ‘‘qubits’’ are the counterparts of ‘‘bits’’ in classical computers, and they are used to store the states of circuits used for quantum computations. Labeled $|0\rangle$ and $|1\rangle$, the two quantum states express one bit of information: $|0\rangle$ corresponds to the bit 0 of classical computers, and $|1\rangle$ to the bit 1. Qubit state $|\Psi\rangle$ maintains a coherent superposition of states. $|0\rangle$ and $|1\rangle$ according to the expression

$|\Psi\rangle=\alpha|0\rangle+\beta|1\rangle$,

where $|\alpha\rangle$ and $|\beta\rangle$ are complex numbers called probability amplitudes. That is, when qubit state $|\Psi\rangle$ collapses into either the $|0\rangle$ state or $|1\rangle$ state, it does so with probabilities α^2 and β^2, respectively, where $\alpha^2+\beta^2=1$.



Qubit neuron model : Qubit NN model is defined as follows (see in fig). The neuron state z that receives inputs from K other neurons is given by

$$u=\sum_{k}^{K} f(\theta_k).x_k - f(\lambda)=\sum_{k}^{K} f(\theta_k).f(y_k) - f(\lambda) \qquad (1)$$

$$y=\frac{\pi}{2}.g(\delta)-arg(u) \tag{2}$$

$$z=f(u) \tag{3}$$

Here, f is the same function and g is the sigmoid function which has the range [0, 1]. y_k is the quantum phase of the k-th neuron state x_k .

Two kinds of parameters exist in this neuron model: phase parameters in the form of weight connection θ_k and threshold λ , and the reversal parameter δ . The phase parameters correspond to the phase of the rotation gate, and the reversal parameter corresponds to the controlled NOT gate. By substituting $y=g(\delta)$ in equation, we obtain a generalized reversal representation operating as the controlled NOT gate, the basic logic gate in quantum computing. Equation (1) expresses the state u of a neuron in the usual way, i.e., as the weighted sum of the states of the inputs minus a threshold. Equation (2) adjusts output qubit phase y more roughly than (1). In (2), arg(u) means the argument of the complex number u, and is implemented by arctan(Im(u)/Re(u)). Unlike in classical neural networks, the multiplication of weights $f(\theta_k)$ to the inputs $x_k=f(y_k)$ results in the rotation of the neuron state based on the rotation gate.

Qubit Neural Network : The neural network employing qubit neurons (called Qubit NN). The sets $\{I_l\}(l=1,2,..,L)$, $\{H_m\}(m=1,2,...,M)$,and $\{O_n\}(n=1,2,...,N)$ represent neuron elements, whereby the variables I, H, and O indicate the Input, Hidden, and Output layers, respectively. L, M, and N are the numbers of neurons in the input, hidden, and output layers, respectively. When input data (denoted by input l) is fed into the network, the input layer consisting of the neurons in $\{I_l\}$ converts input values in the range [0, 1] into quantum states with phase values in the range $[0,\pi/2]$. The output Z_l^I of input neuron I_l :

$Z_l^I=f(\frac{\pi}{2}.input_l)$

becomes input to the hidden layer. The hidden and output layers with neurons from the sets $\{H_m\}$ and $\{O_n\}$, respectively. We obtain the output to the network, denoted by output n , with the probability of the state in which $|1\rangle$ is observed from the nth neuron in the output layer:

$output_n=|I(z_n^0)|^2$, where I represent Im.

2.2 Complex Convolution Neural Network(CV-CNN)

2.2.1 Real-valued Convolution Neural Network(RV-CNN) or CNN

Before going into CV-CNN, we will discuss RV-CNN. RV-CNN or CNN is very common and very popular in computer vision tasks. CNNs are very similar to other ordinary neural networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some input, performs a dot product and optionally follows it with a non-linearity. It also some form of differentiable score function: from the raw image pixels on one end to the class scores at the other. In particular, unlike a regular NN, the layers of CNN have neurons arranged in 3 dimensions: width, height and depth(Note the word depth here refers to the third dimension of an activation volume, not the depth of full neural network, which can refer to the total number of layers in the network).

Convolution layer output:

$$O_i^{(l+1)} = f\left(V_i^{(l+1)}\right) = \frac{1}{1+e^{-V_i}(l+1)}$$

$v_i^{(l+1)} = \sum_{k=1}^{K} w_i^{(l+1)} . O_k^l + b_i(l+1)$,where l is layer, O_i^l and $O_i^{(l+1)}$ is output of l and l+1 layers, w_i^l weight and b_i^l bias of corresponding layers. The hyper-parameters of convolution layer include number of feature maps *I,* filter size $F \times F \times K$, stride *S* and zero padding size *P*. The stride means the intervals of each slide of filter.

Average Pooling layer output: Apart from reducing the dimension of feature using convolution, pooling also helps to make the representation invariant to small shifts and distortions of the input. Mathematical formulation of average pooling layer is defined as follows,

$$O_i^{(l+1)}(x) = \underset{u=0,\ldots,g-1}{avg} O_i^l(x \cdot s + u)$$

ReLU: The most common activation function used by the CNN is the rectified linear unit.

Definition: $ReLU(x) = \begin{cases} x & x \geqslant 0 \\ 0 & 0.w \end{cases}$

2.2.2 CV-CNN

CV-CNN used in our research is very similar to RV-CNN. We found some similarity and used some CV-CNN proposed in various research papers[22]. IN the CV-CNN layer, the complex output feature maps $O_i^{(l+1)} \in \mathbb{C}^{W_2 \times H_2 \times I}$ are computed by the convolution between all previous layer's input feature maps $O_i^{(l)} \in \mathbb{C}^{W_1 \times H_1 \times I}$ and filters $w_{ik}^{(l+1)} \in \mathbb{C}^{F \times F \times K \times I}$, and add a bias $b_i^{(l+1)} \in \mathbb{C}^I$,where C denotes the complex domain and the superscript in it's domain. The convolution is calculated by,

$$O_i^{(l+1)} = f\left(\Re\left(V_i^{(l+1)}\right)\right) + j f\left(\Im\left(V_i^{(l+1)}\right)\right)$$
$$= \frac{1}{1+e^{-\Re\left(V_i^{(l+1)}\right)}} + \frac{1}{1+e^{-\Im\left(V_i^{(l+1)}\right)}}$$
$$V_k^{(l+1)} = \sum_{k=1}^{K} w_{ik}^{(l+1)} * O_k^l + b_i(l+1)$$
$$= \sum_{k=1}^{K} \left(\Re\left(w_{ik}^{(l+1)}\right)\cdot\left(\Re\left(O_k^{(l)}\right)\right) - \Im\left(w_{ik}^{(l+1)}\right)\cdot\left(\Im\left(O_k^{(l)}\right)\right)\right)$$
$$+ j\sum_{k=1}^{K} \left(\Re\left(w_{ik}^{(l+1)}\right)\cdot\left(\Im\left(O_k^{(l)}\right)\right) + \Im\left(w_{ik}^{(l+1)}\right)\cdot\left(\Re\left(O_k^{(l)}\right)\right)\right) + b_i^{(l+1)}$$

where $j=\sqrt{-1}$ is the imaginary unit. Character * is the convolution operation. $\Re$ And $\Im$ denotes the real and imaginary part of the complex number. O_k^l Is the unit of the *k*th input feature map in layer *l,* and $V_k^{(l+1)}$ denotes the weighted sum of inputs to the *i*th output feature map in layer *l*+1. The hyper-parameters of convolution layer include number of feature maps *I,* filter size $F\times F\times K$, stride *S* and zero padding size *P*. The strides means the intervals of each moving when the filter contact to the input feature maps. Due to the convolution operation, the valid output feature map is smaller than that of the input. By the zero padding process, the size shrinking with depth can be well compensated with proper *P,* which helps to make an arbitrary deep convolutional network. If the input is composed of *K* feature maps with size $W_1\times H_1$ and output will be with *I* feature maps of size $W_2\times H_2$,where $W_2=(W_1-F+2P)/S+1$ and $H_2=(H_1-F+2P)/S+1$.

Complex-valued Pooling: A straight forward extension of average pooling from real-valued to complex-valued can be defined as,

$$O_i^{(l+1)}(x,y) = \underset{u,v=0,\ldots,g-1}{avg} O_i^l(x\cdot s+u, y\cdot s+v)$$

,where *g* is pooling size and *s* is the stride. $O_i^{(l+1)}(x,y)$ Is the unit of the *i*th input feature map at position (x,y) .However max pooling, extension from real to complex is not readily available, because there nothing exist max of a complex number, so a natural way is to simply take the amplitude maximum. We will use average pooling based on this principal defined by above formula.

Complex-valued ReLU: To stay as close as possible to real model, we construct the complex ReLU in th same manner as it's real value counterpart. Definition,

$$ReLU(z) = \begin{cases} z & \Re,\Im \geqslant 0 \\ 0 & 0.w \end{cases} = \begin{cases} z & arg(z) \in [0, \frac{\Pi}{2}] \\ 0 & 0.w \end{cases}$$

2.3 Experimental Datasets

2.3.1 MNIST

I think MNIST is most popular basic dataset in Computer Vision tasks. For verifying CNN and NN for classification tasks, it's very common dataset. MNIST quite very large database of handwritten digits. The black and white images from National Institute of Standards and Technology(NIST) were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduce gray scale levels. The MNIST dataset contains 60,000 training images and 10,000 testing images. Sample:

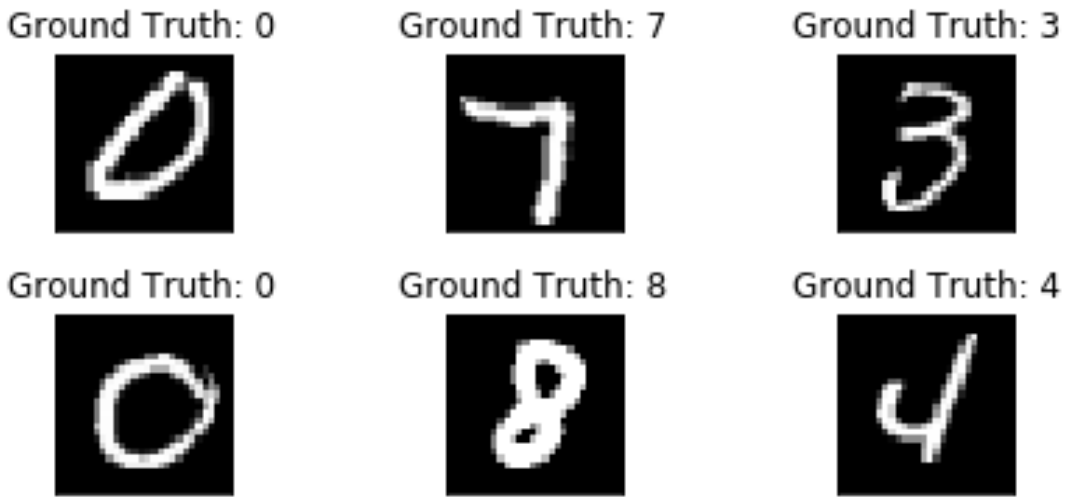


Figure 1: MNIST handwritten text image

2.3.2 Chest X-ray Dataset

This dataset is open source dataset available on Kaggle[23]. This dataset contains chest x-ray of 3 types normal people, bacterial pneumonia infection and virus pneumonia infection patients. There are 5,863 X-Ray images and 3 categories normal, bacteria and viral pneumonia infection patients. The Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care. For the analysis of chest x-ray images, all chest radiography were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert. The normal chest X-ray (left panel) depicts clear lungs without any areas of abnormal opacification in the image. Bacterial pneumonia (middle) typically exhibits a focal lobar consolidation, in this case in the right upper

lobe (white arrows), whereas viral pneumonia (right) manifests with a more diffuse ‘‘interstitial’’ pattern in both lungs.

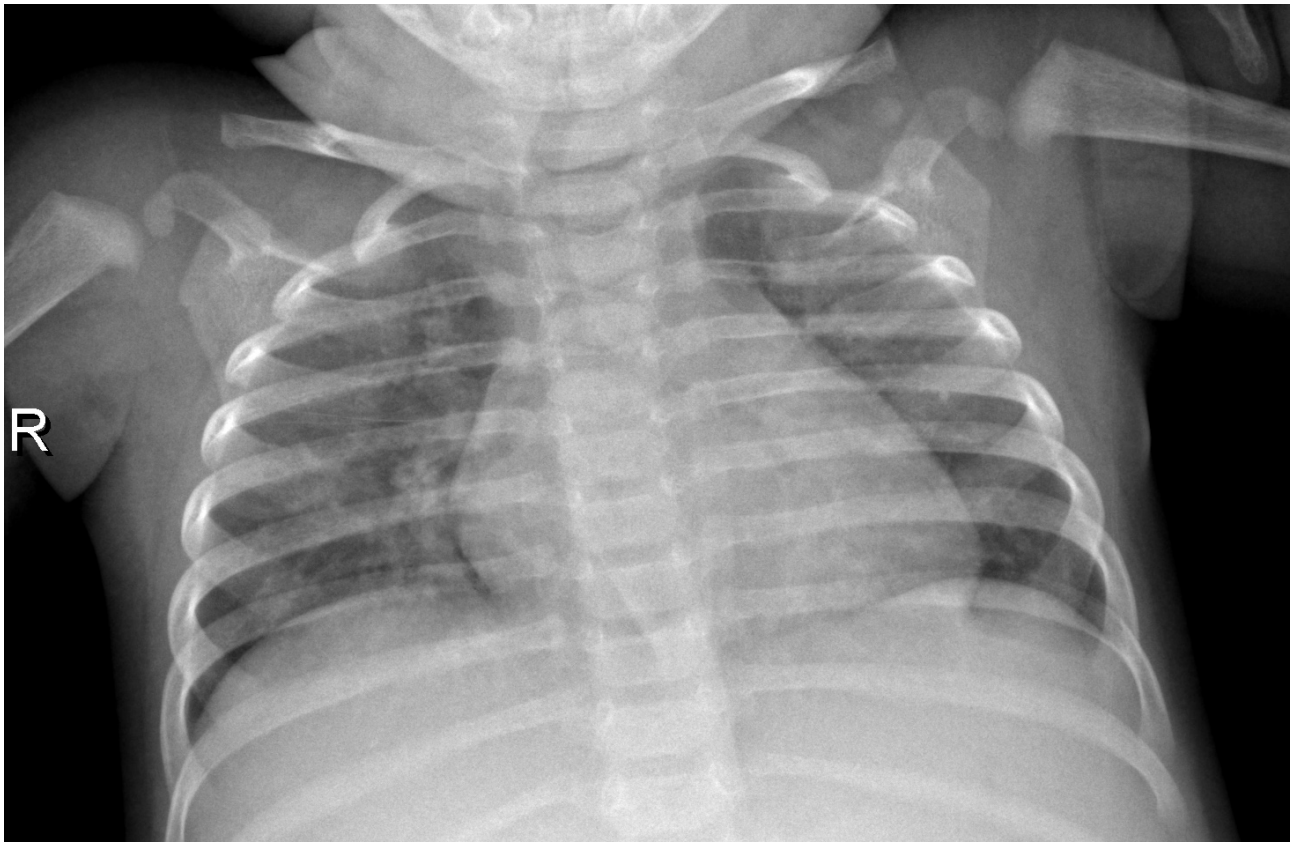


Figure 2: Chest X-Ray of normal patient

Figure 3: Chest X-Ray of patient suffering from bacterial pneumonia infection

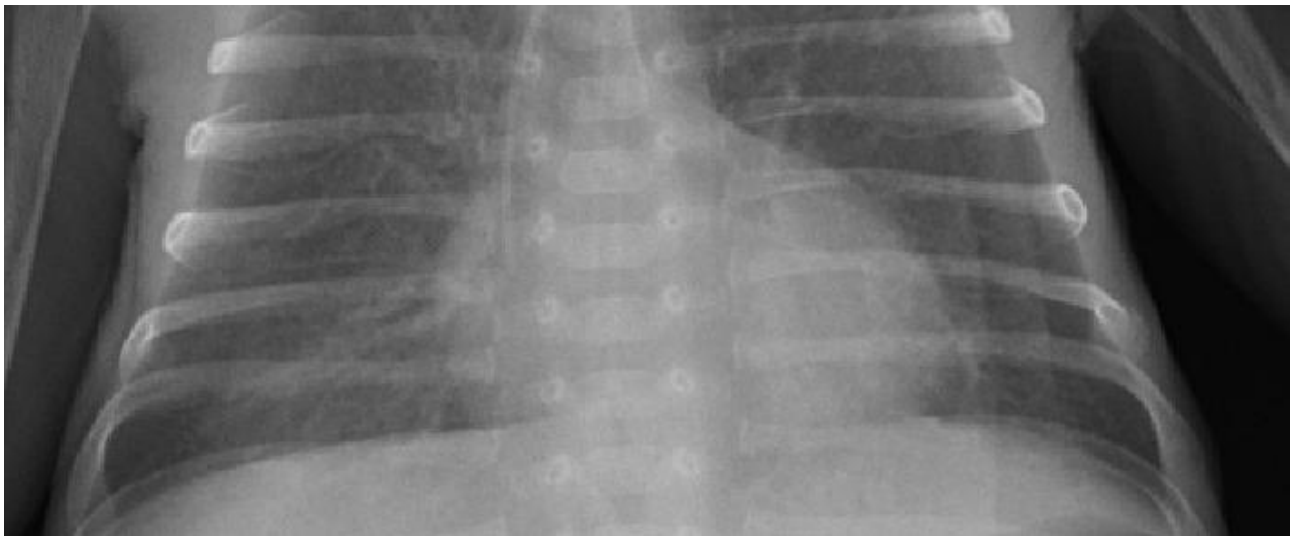


Figure 4: Chest X-Ray of patient suffering from viral pneumonia infection

2.3.3 Brain Tumor Segmentation(BraTS) 2018 Dataset

BraTS has always been focusing on the evaluation of state-of-the-art methods for the segmentation of brain tumors in multi-modal magnetic resonance imaging (MRI) scans and organized by Medical Image Computing and Computer Assisted Intervention(MICCAI). BraTS dataset encompasses MRI-DCE data of 285 patients with diagnosed gliomas 210 high-grade glioblastomas (HGG), and 75 low-grade gliomas (LGG). Each study was manually annotated by one to four experienced readers. The data comes in four co-registered modalities: native pre-contrast (T1), post-contrast T1-weighted (T1CE), T2-weighted (T2), and T2 Fluid Attenuated Inversion Recovery (FLAIR). All the pixels have one of four labels attached: healthy tissue, Gd-enhancing tumor (ET), peritumoral edema (ED), the necrotic and non-enhancing tumor core (NCR/NET). T1 and T2 weighted are two very basic types of images.

T1 weighted: The timing of radio frequency pulse sequences used to make T1 images results in images which highlight fat tissue within the body. Below is the T1 MRI image visualization of random patient in dataset.

T2 weighted: The timing of radio frequency pulse sequences used to make T2 images results in images which highlight fat and water within the body.

T1CE: Axial 3D contrast enhanced T1-weighted images (T1CE) were obtained after administration of a standard dose (0.1 mmol/kg) of gadodiamide with a power injector.

FLAIR: The Flair sequence is similar to a T2-weighted image except that the time of echo and repetition times are very long. By doing so, abnormalities remain bright but normal CSF fluid is

attenuated and made dark. This sequence is very sensitive to pathology and makes the differentiation between CSF and an abnormality much easier.

We visualize each in below figures:

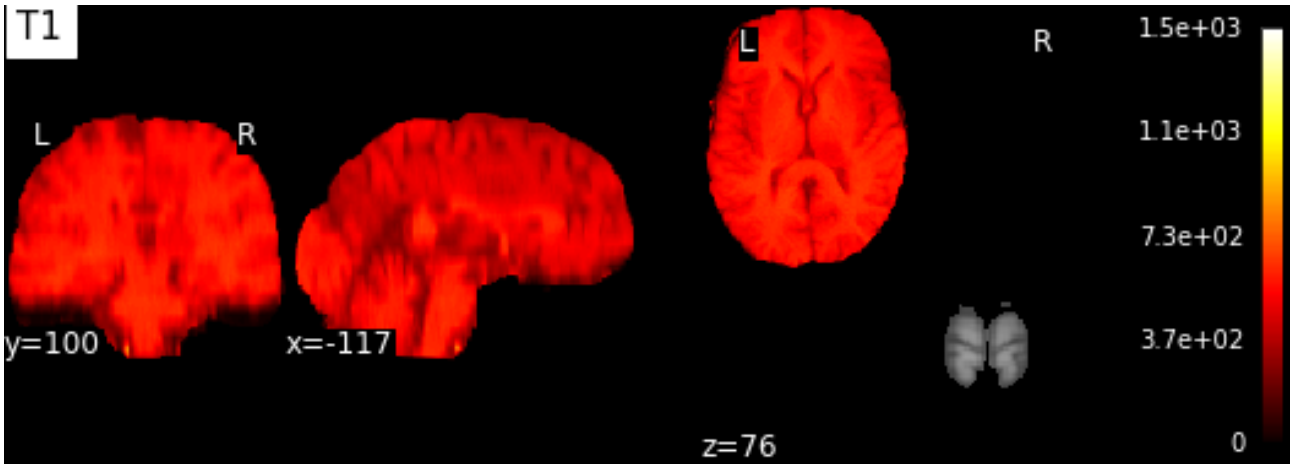


Figure 5: T1-Weighted

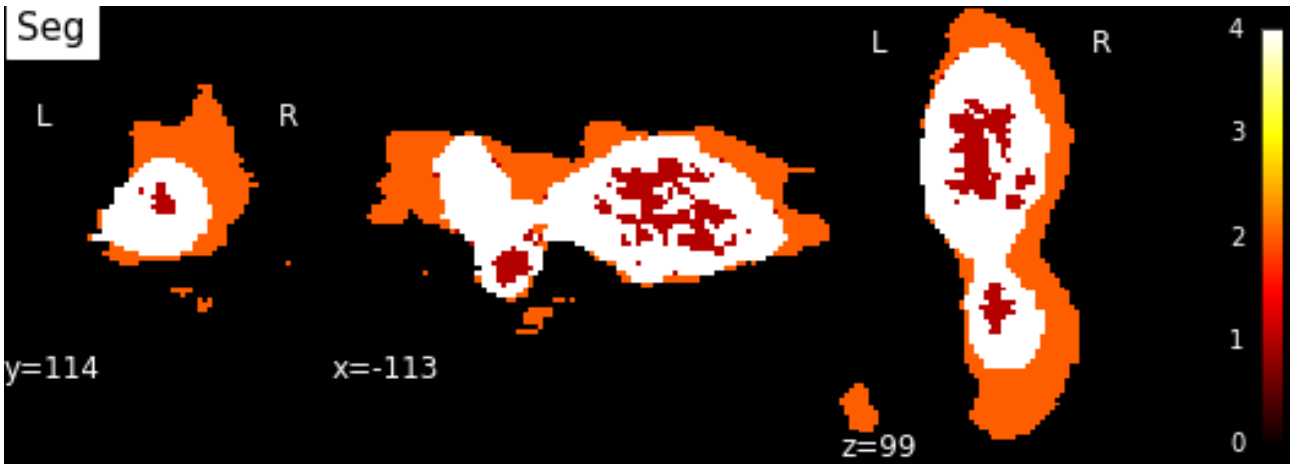


Figure 9: Segmented Brain Tumor Label

2.3.4 Brain Tumor dataset

This dataset is provided by School of Biomedical Engineering, Southern Medical University, China. We are using this dataset for brain tumor classification task. Generally tumors are categories into 3 types.

- Meningioma : A meningioma is a tumor that forms on membranes that cover the brain and spinal cord just inside the skull. Specifically, the tumor forms on the three layers of membranes that are called meninges.
- Glioma: Glioma is a common type of tumor originating in the brain. About 33 percent of all brain tumors are gliomas, which originate in the glial cells that surround and support neurons in the brain, including astrocytes, oligodendrocytes and ependymal cells.

- Pituitary : A pituitary tumor is an abnormal growth in the pituitary gland. The pituitary is a small gland at the base of the brain. It regulates the body's balance of many hormones.

This brain tumor dataset contains 3064 T1-weighted contrast-inhanced images from 233 patients with all three kinds of brain tumors: meningioma (708 slices), glioma (1426 slices), and pituitary tumor (930 slices).

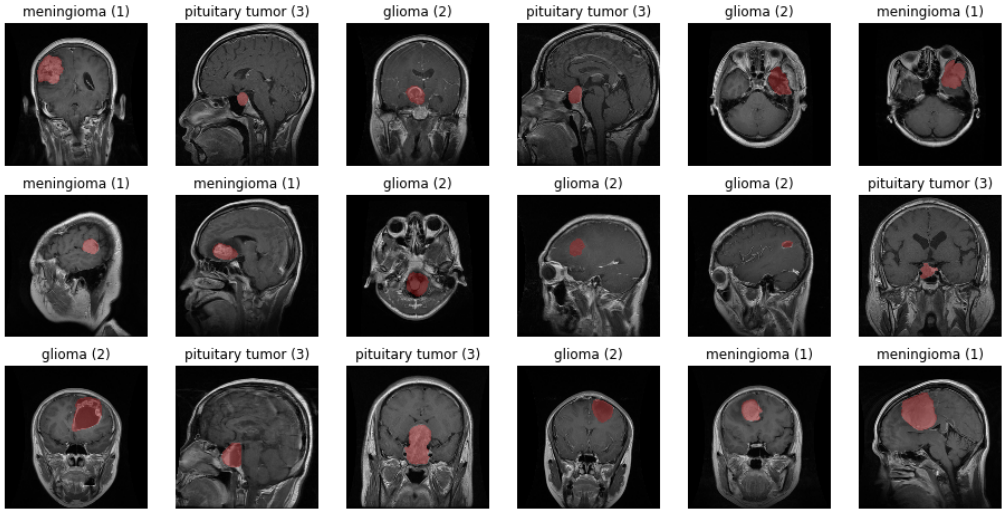


Figure 7: 2-D slices of MRI brain images

3 Experiments, Results and Analysis

3.1 Experiments and Result

For constructing Qubit NN and experiments we selected Pytorch as base deep NN library and used Python as language. We went through different stages and carefully selected loss function and optimizer. For MNIST and Chest X-Ray classification tasks, we finalized with Cross Entropy loss and Stochastic Gradient Descent(SGD) optimizer. In experiments we tried to provide comparison study of Qubit NN and CV-NN in comparison with real-valued NN. More or less in experiments number of layers in real-valued NN and Qubit NN based are same and we also tried to compare with same loss function and optimizer.

3.1.1 NN with two Qubit NN layers for MNIST dataset classification task

For first experiment we constructed two neural networks and for training and validation we used MNIST dataset with batch size 128 and input size 28x28x1. Qubit NN layer based architecture we used 2-Qubit NN layers, first Qubit NN with (28,28) input size and 128 output size followed by ReLU layer and second Qubit NN with (128,1) input size and 10 output size followed by ReLU and sigmoid followed by output layer. As per comparing Qubit NN based architecture, we constructed a similar network by using two Linear NN layer with inputs sizes 28 and 28x14 and output sizes 14 and 10 respectively, followed by ReLU layer and output layer followed sigmoid. For both architecture we used Cross Entropy loss with SGD optimizer with learning rate 0.01. Below is the train and test loss and accuracy of networks during 10 epochs. In this experiment Qubit NN architecture is outperformed by Simple NN architecture. Simple NN architecture has 1.5558 best batch loss and 1.6086 average loss during training where Qubit NN architecture has 2.1320 best batch loss and 2.1495 average loss during training. Simple NN and Qubit NN architectures have 88.57 and 36.77 accuracy respectively.

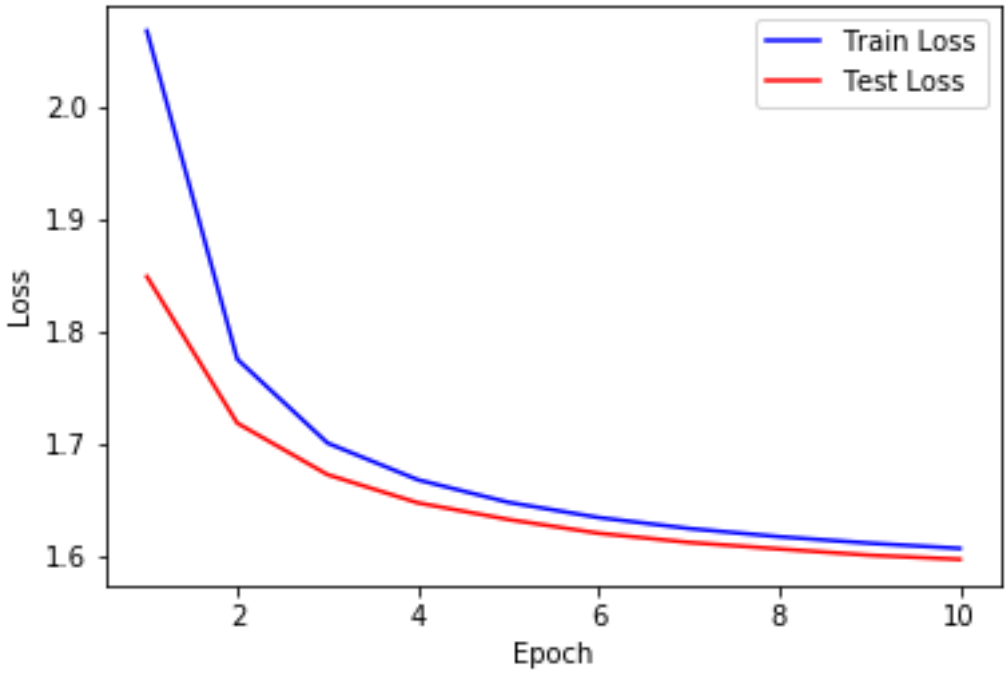


Figure 8: NN with Lienar Layer Loss MNIST dataset

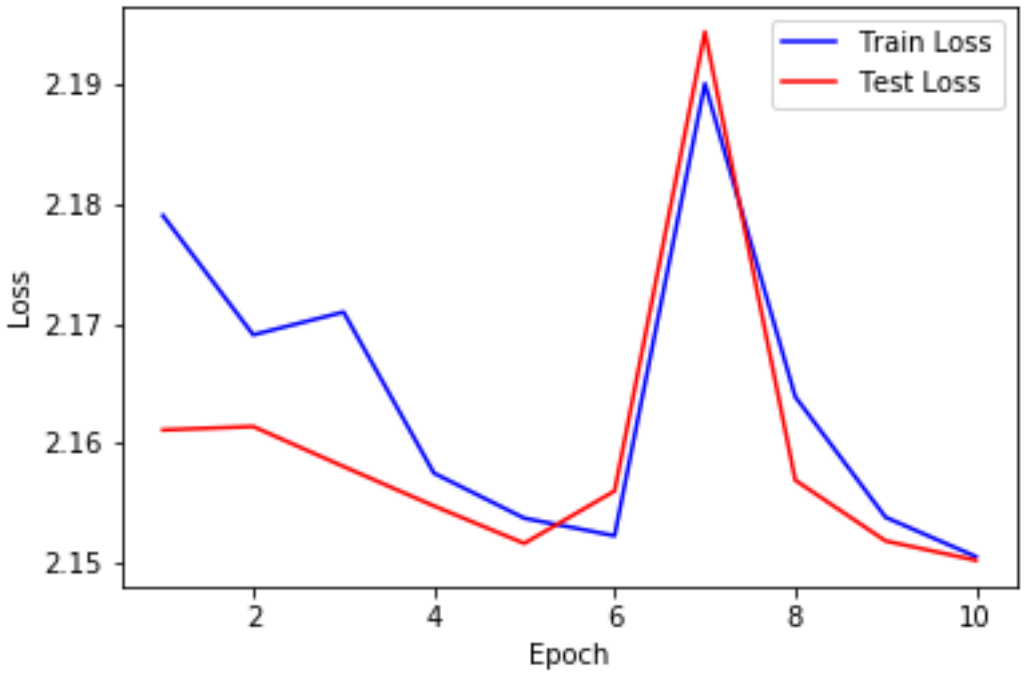


Figure 9: NN with Qubit NN Layer Loss MNIST dataset

3.1.2 Qubit NN with 2D Convolution layers for MNIST dataset classification task

Before this experiment, we find out the Qubit NN is not independently not fit for computer vision classification tasks. So in this experiment we explored, how Qubit NN is performing, if we mix with CNN. We constructed two new neural network architecture one composed 2 CNN layers, 2 Max Pooling layers and 3 Linear NN layers and other composed of 2 CNN layers, 2 Max pooling layers, 2 Qubit NN layers and 1 linear output layer. Next figure illustrate the overall input, output channels with filter, where upper brackets represent input size followed by output size of particular layer. We use zero-padding and stride 1 in CNNs. Experimental results we used same MNIST dataset with batch size 128 and input size 28x28x1.

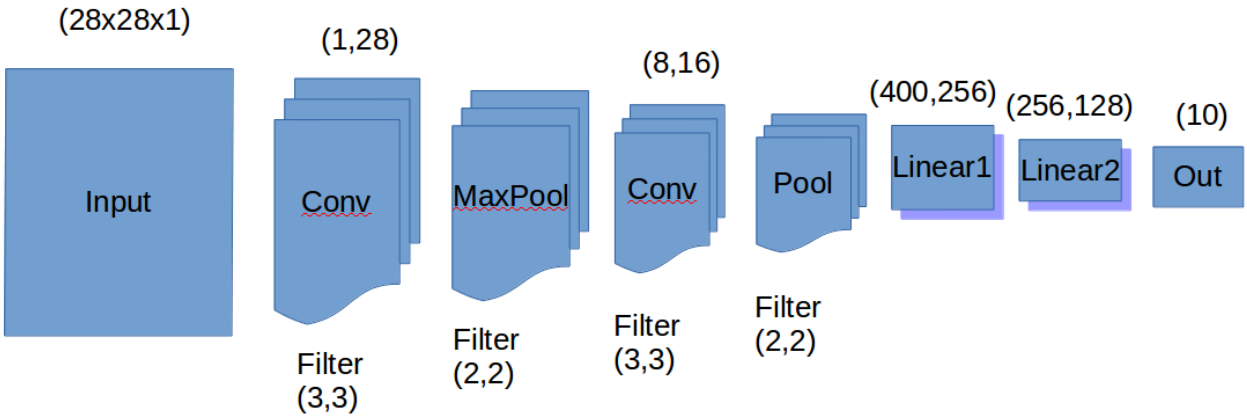


Figure 10: NN Architecture with convolution

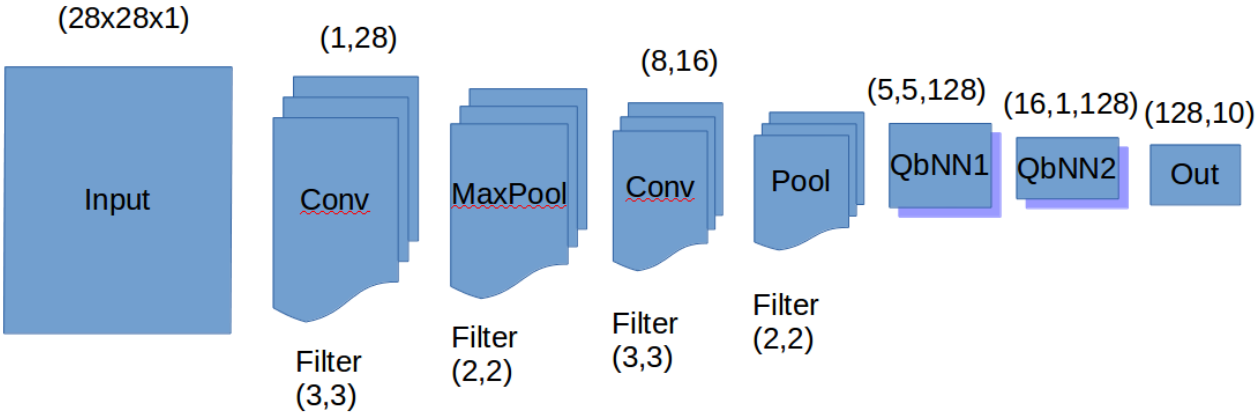


Figure 11: NN Architecture with 2d Conv + Qubit NN

As experiment performed on both of the neural networks result was very appealing. Here some interesting results we found in experiment, total number trainable parameters in in neural network with Qubit NN(13802) is 90% less than neural network without Qubit NN(138090) and still same accuracy is achievable in same number of epochs. Neural network without Qubit NN architecture has 0.0112 best batch loss in 9th epoch and 0.0717 average loss during training where neural network with Qubit NN has 0.0305 best batch loss and 0.1342 average loss during training. Neural network with Qubit NN and without Qubit NN architectures have 96.08 and 97.80 accuracy respectively.

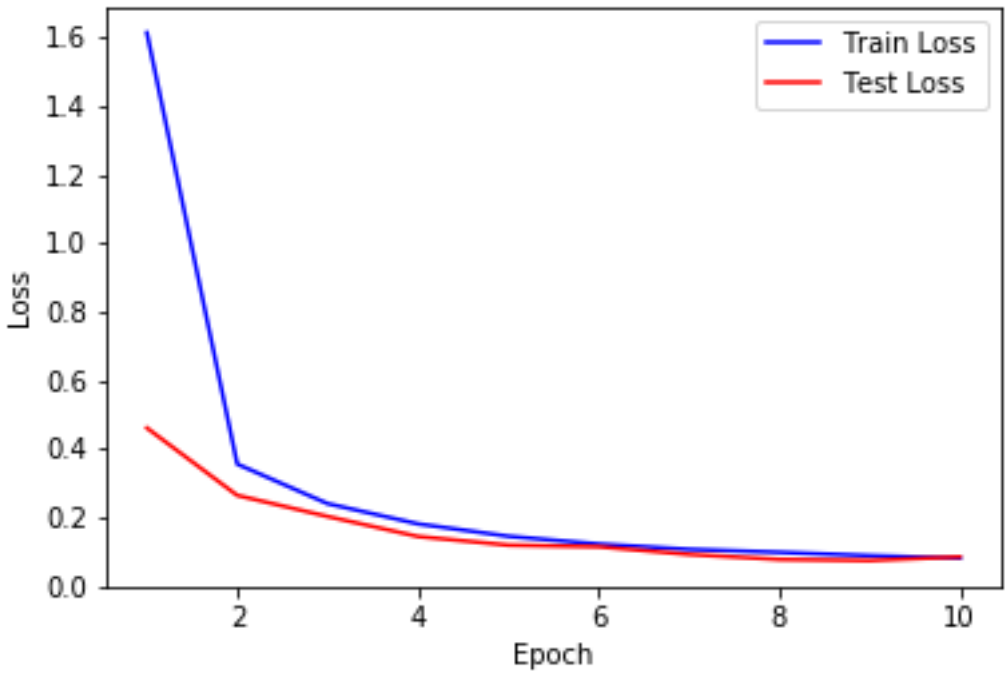


Figure 12: Neural network with CNN without Qubit NN, traing vs test loss

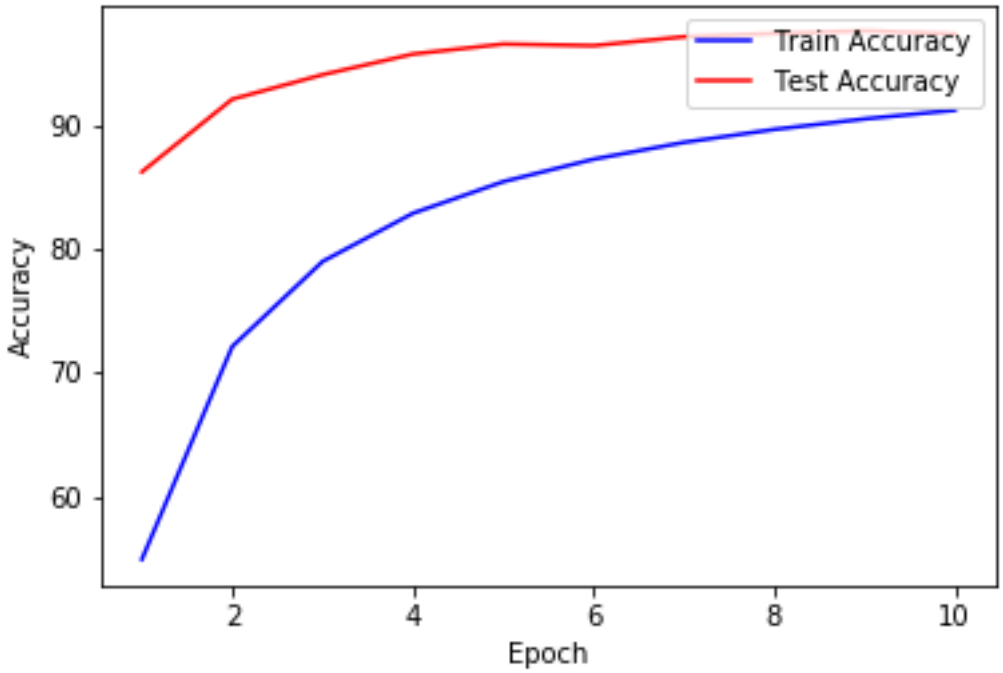


Figure 13: Neural network with CNN with Qubit NN, traing vs test loss

3.1.3 Qubit NN with CV-CNN layers for MNIST dataset classification task

If we follow up the previous results, we found while mixing Qubit NN with CNN, it's increasing the accuracy with 90% less number of parameters. As from theory we all know the Qubit NN is complex-valued by nature. So, we decided to explore and test Qubit NN with proposed CV-CNN and complex pooling. We already discussed CV-CNN, complex pooling and complex ReLU above in theory part. So per architecture, It's quite same as the neural network with Qubit NN and CNN architecture. This experiment architecture has 2 CV-CNN layers followed by complex ReLU and 2 complex pooling layers and 1 Qubit NN layer as hidden layer and 1 Qubit NN layer as output layer. We use zero-padding and stride 1 in CV-CNNs. We used same MNIST dataset with batch size 128 and input size 28x28x1. Neural network without Qubit NN and CV-CNN has 0.8447 best batch loss in 10^{th} epoch and 0.8881 average loss during training. Neural network with Qubit NN and CNN has 97.33% accuracy and 60% less number of parameters than previous neural network architecture .

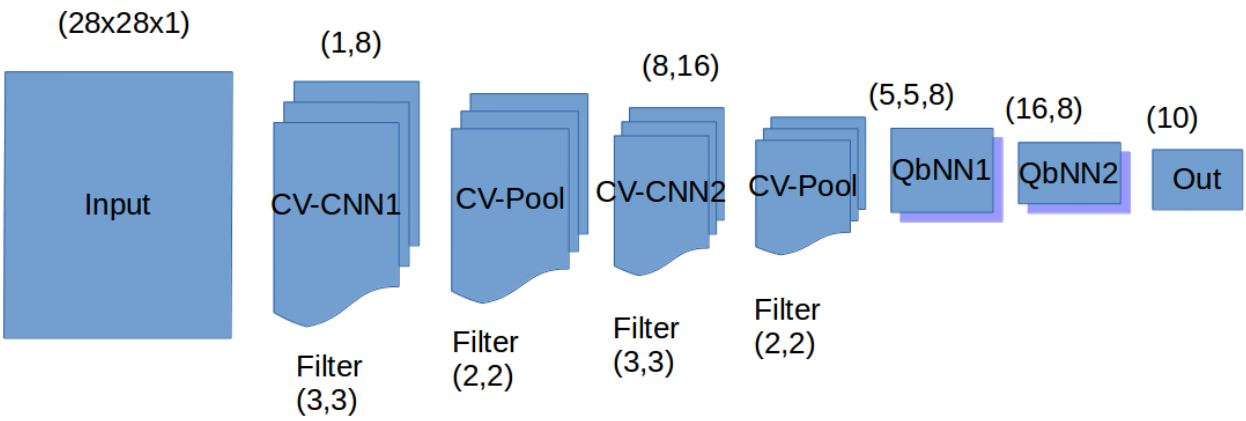


Figure 14: Neural Network with Qubit NN and CV-CNN

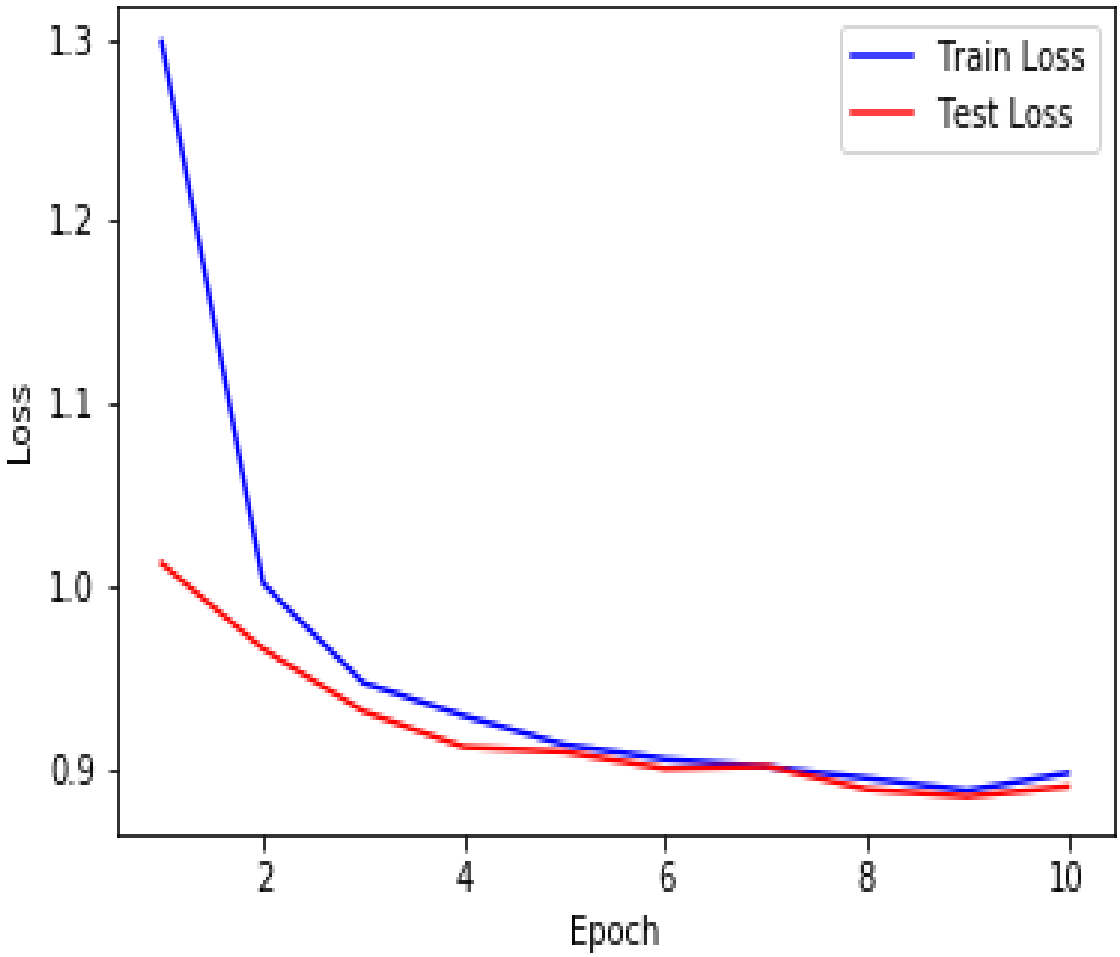


Figure 15: Neural network with CV-CNN and Qubit NN, traing, test loss and Accuracy

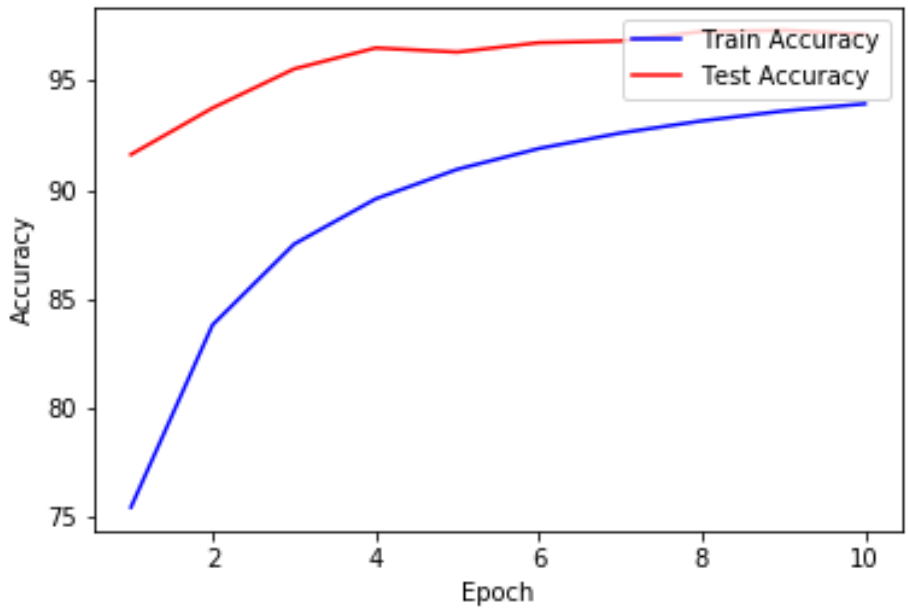


Figure 16: Neural network with CV-CNN and Qubit NN, traing vs test loss

3.2 Qubit NN with CV-CNN layers for Pneumonia infection type classification

Originally X-Ray pneumonia classification competition has only 2 classes to classify, first noraml and pneumonia. There is already many more neural networks, with accuracy, So we modified the problem statement a little bit. Now this model is classifying 3 classes, 0. Normal 1. Bacterial Pneumonia and 2. Virus Pneumonia. I found out data a bit dis-balance for this task but we wanted to test efficiency of newly developed network. This neural network in same in architecture but differ in input, because of different dataset from previous. So for the network architecture you can see figure number [15] and figure number [20]. We didn't work more on data augmentation part, because just trying to find the network efficiency and performance, bit in my opinion, if we augment dataset little better then network will outperform. Number of parameters is 90% less than neural network without Qubit NN and still same accuracy is achievable in same number of epochs. Neural network without Qubit NN architecture has 0.1756 best batch loss and 0.44 average loss during training where neural network with Qubit NN has 0.3816 best batch loss in 6th epoch and 0.5953 average loss during

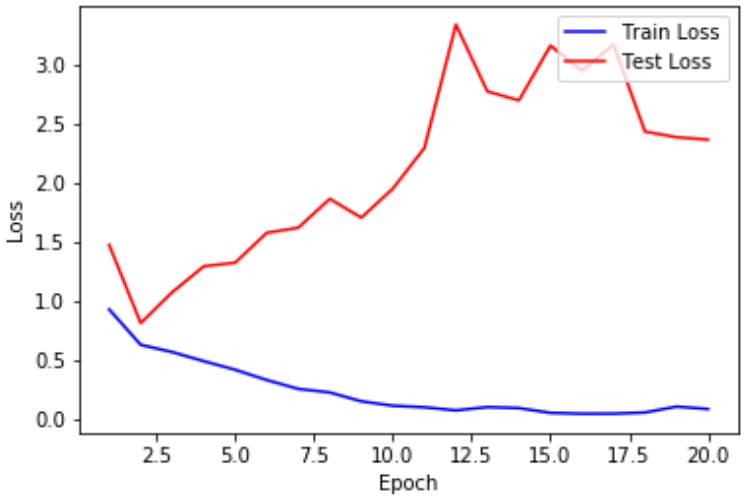


Figure 17: Neural network with QubitNN and CV-CNN, train vs test loss

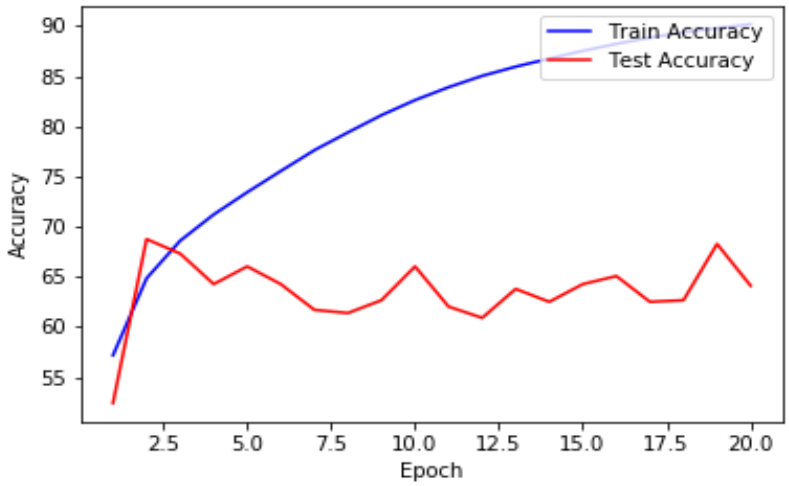


Figure 18: Neural network with CNN, train vs test loss

3.3 Qubit NN with CV-CNN layers for BraTS segmentation task

Qubit NN is not applicable for image segmentation task. It has dimensionality reduction behavior, like Linear NN layers.

3.4 Brain Tumor classification tasks from 2d-segmented MRI images

Brain Tumor dataset contains 3064 images of size 512x512. In this experiment, we implemented the same network which we used in 3.13 experiment with different input output size. This class contains 3 possible output labels, 1. Meningioma, 2. Glioma and 3. Pituitary. We did a little augmentation on dataset and compressed dataset into 256x256 input size. During the training loss was decreasing but learning of model found very slow and test accuracy found very much constant.

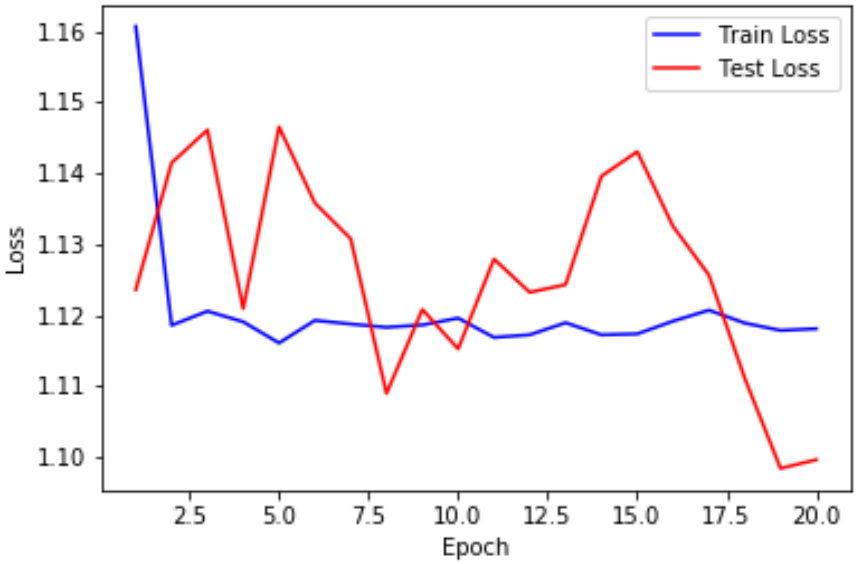


Figure 19: Brain Tumor classification loss

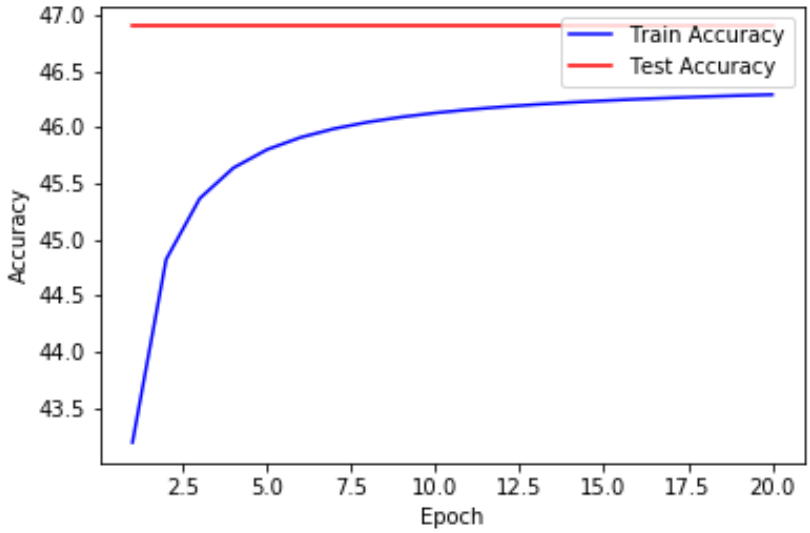


Figure 20: Brain Tumor Classification accuracy

3.5 Experiment Summary

If we are summarize all the experiments in single word then we can say there is many aspects to deep dive into Qubit NN and CV-CNN experiments. This summary table is providing more clarity.

Model	Dataset	Epochs	Avg. Loss(Cross Entropy)	Accuracy(%)	Number of Trainable Parameters
Simple NN	MNIST	10	1.5	88.9	4336
Qubit NN	MNIST	10	2.15	38.9	203678
CNN	MNIST	10	0.0717	97	138090
Qubit NN + CNN	MNIST	10	0.132	96	13802
Qubit NN + CV-CNN	MNIST	10	0.889	97	5510
CNN	X-Ray	20	2.36	68	31504855
Qubit NN + CV-CNN	X-Ray	20	0.9244	68	127385
Qubit NN + CV-CNN	Brain Tumor Classification	20	1.097	48	127612

4 Conclusion

Qubit NN is found very successful, if you mix with CV-CNN and in classification tasks, it is outperforming real-valued NN. If your want to use Qubit NN alone in computer vision tasks then it's not recommendable. My excrements was very limited and focused in computer vision and medical data analysis, but still there is many area to explore for many computer task like object detection. Further researchers can explore the implementation of Qubit NN in forecasting problem and I think, it will be very useful in time series forecasting. Hope further researcher will explore this area.

5 Acknowledgement

I would like to thank my supervisor, Pavlovsky E.N., who had introduced me to the exciting field of computer vision and Quantum Computing, and guided me throughout this research. I would also like to express my gratitude for my peers. They have made this experience much more meaningful, and enjoyable.

Finally, and most importantly, I am greatly thankful to my family and my friends for their support. My gratitude for their help and support is beyond words.

6 References

1: Stanford, Outpacing Moore's Law, 2019

2: K. Mori, T. Isokawa, N. Kouda, N. Matsui, and H. Nishimura, Qubit Inspired Neural Network towards Its Practical Applications, 2006

3: D. Deutsch, Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer, 1985

4: P. Shor, Polynomial-Time Algorithms for Prime Factorization and Discrete Algorithms on a Quantum Computer, 1994

5: M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information, 2000

6: Y. R. Zheng, C. Song, M.-C. Chen, B. Xia, W. Liu, Q. Guo, L. Zhang, D. Xu, H. Deng, K. Huang, Y. Wu, Z. Yan, D. Zheng, L. Lu, J.-W. Pan, H. Wang, C.-Y. Lu, and X. B. Zhu, Solving Systems of Linear Equations with a Superconducting Quantum Processor, 2017

7: X. H. Peng, Z.-Y. Liao, N. Xu, G. Qin, X. Zhou, D. Suter, and J. Du., Quantum Adiabatic Algorithm for Factorization and Its Experimental Implementation, 2008

8: L. K. Grover, Quantum Mechanics Helps in Searching for a Needle in a Haystack, 1997

9: Xi-Wei Y, H Wang, Z Liao, MC Chen, J Pan, J Li, K Zhang, X Lin, Z Wang, Z Luo, W Zheng, J Li, M Zhao, X Peng, and D Suter, Quantum Image Processing and Its Application to Edge Detection: Theory and Experiment, 2017

10: D. Riste, M. Silva, C. Ryan, A. Cross, A. Córcoles, J. Smolin, J. Gambetta, J. Chow, and B Johnson, Demonstration of Quantum Advantage in Machine Learning, 2017

11: A. W. Harrow, A. Hassidim, and S. Lloyd, Quantum Algorithm for Linear Systems of Equations, 2009

12: X. Cai, D. Wu, Z. Su, M.-C. Chen, X.-L. Wang, L. Li, N. Liu, C.-Y. Lu, and J.-W. Pan, Entanglement-Based Machine Learning on a Quantum Computer, 2015

13: P. Rebentrost, M. Mohseni, and S. Lloyd, Quantum Support Vector Machine for Big Data Classification, 2014

14: Y. Zhang, K. Lu, Y. Gao, and M. Wang, NEQR: A Novel Enhanced Quantum Representation of Digital Images, 2013

15: Ga. Gedda, O. Logoa, Yueh-Yun Yao, Si-Han Chen, Anil V. Ghule, Yong-Chien Ling and Jia-Yaw Chang, Facile synthesis of gold/gadolinium-doped carbon quantum dot nanocomposites for magnetic resonance imaging and photothermal ablation therapy, 2017

16: T. Nitta, On the critical points of the complex-valued neural network, 2002

17: Akira Hirose and Shotaro Yoshida, Generalization characteristics of complex-valued feedforward neural net-works in relation to signal coherence, 2012

18: Martin Arjovsky, Amar Shah, and Yoshua Bengio, Unitary evolution recurrent neural networks, 2015

19: Nitzan Guberman, On Complex Valued Convolutional NeuralNetworks, 2016

20: C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio & C. J. Pal, D EEP C OMPLEX N ETWORKS, 2018

21: Z. Zhang, H. Wang, Feng Xu and Ya-Qiu Jin, Complex-Valued Convolutional Neural Network and Its Application in Polarimetric SAR Image Classification, 2017

Qubit NN and Neural Network Code References

```
import numpy as np
import torch

class QubitComplex(torch.nn.Module):
    def __init__( self, in_shape=None, out_units=None, magnitude=.1, no_bias=False, out_layer=False,
power = 2,
                 variant = 1, out_trainable=False, layer_num=1, complex_input=False ):
        super( QubitComplex, self).__init__()
        self.eps = 0.001
        self.layer_num = layer_num
        self.shape1 = None
        self.shape2 = None
        self.no_bias = no_bias
        self.out_layer = out_layer
        self.variant   = variant
        self.power     = power
        self.out_trainable = out_trainable
        self.magnitude = magnitude
        if in_shape:
            self.shape1 = in_shape
        if out_units:
            if isinstance(out_units, tuple):
                self.shape2 = out_units
            else:
                self.shape2 = (out_units,)
        #print(self.shape1,self.shape2)
        theta_re = torch.empty( (self.shape1), requires_grad=False)
        self.theta_re = torch.nn.Parameter(torch.nn.init.xavier_uniform_(theta_re, gain=self.magnitude))
        theta_im = torch.empty( (self.shape1), requires_grad=False)
        self.theta_im = torch.nn.Parameter(torch.nn.init.xavier_uniform_(theta_im, gain=self.magnitude))

        # n x 1
        lambda_re = torch.empty( (self.shape2), requires_grad=False )
        self.lambda_re = torch.nn.Parameter(torch.nn.init.uniform_( lambda_re , a=0.0, b=0.1 ))
        lambda_im = torch.empty( (self.shape2), requires_grad=False)
        self.lambda_im = torch.nn.Parameter(torch.nn.init.uniform_( lambda_im , a=0.0, b=0.1 ))

        # 1 x 1
        #delta = mx.sym.Variable('delta%d' % layer_num, shape = shape2, init = mx.init.Uniform(scale=
1.5))
        delta = torch.empty( (self.shape2), requires_grad=False)
        self.delta = torch.nn.Parameter(torch.nn.init.uniform_( delta , b=1.5 ))
        if no_bias:
            self.lambda_c = 0.0
        else:
            self.lambda_c = 1.0
        self.complex_input = complex_input
```

```
def forward(self, data):
        fi_in = data #QubitPhaseLimitLayer(data)
        if self.complex_input:
            x_re, x_im = fi_in[:][0], fi_in[:][1]
        else:
            x_re, x_im = get_re_im_from_fi(fi_in,1.0)
        # k x n             # k x m  # m x n
        x_re_theta_re = torch.tensordot(x_re, self.theta_re, dims=2)
        x_re_theta_im = torch.tensordot(x_re, self.theta_im, dims=2)
        x_im_theta_re = torch.tensordot(x_im, self.theta_re, dims=2)
        x_im_theta_im = torch.tensordot(x_im, self.theta_im, dims=2)
        # k x n
        x_theta_re = x_re_theta_re - x_im_theta_im
        x_theta_im = x_re_theta_im + x_im_theta_re
        # k x n           # n x 1
        u_re = torch.add(x_theta_re, - self.lambda_c * self.lambda_re)
        u_im = torch.add(x_theta_im, - self.lambda_c * self.lambda_im)
#         print(type(u_re))
        # k x n
        if self.variant == 1: # Tricky arg
            arg = -1 * torch.atan( u_re ) #/ ((u_re) + sign_u_re * eps))

            # n x 1
            sigma = (np.pi / 2) * sigmoid(self.delta)

            # k x n
            #y = torch.broadcast_tensors(torch.add(arg, sigma))
            y = torch.add(arg, sigma)
        elif self.variant == 5: # Honest arg with Gaussian
            sign_u_re = torch.sign(u_re)
            u_im_not_zero = torch.abs(torch.sign(u_im))
            u_re_sign = torch.sign(u_re)
            numerator = torch.sqrt(torch.pow(u_im, 2) + torch.pow(u_re, 2)) - u_re
            arg = -1 * (2 * u_im_not_zero * torch.atan(numerator / u_im) + (np.pi/2 - np.pi/2 *
u_re_sign) * (1-u_im_not_zero) )

            # n x 1
            # Gaussian here is from the paper: "Qubit Neural Tree Network With Applicationsin
Nonlinear System Modeling", doi:10.1109/ACCESS.2018.2869894
            sigma = (np.pi / 2) * gaussian(self.delta)

            # k x n
            #y = torch.broadcast_tensors(torch.add(arg, sigma))
            y = torch.add(arg, sigma)
        # k x n
        if self.out_layer:
            fi_out = torch.pow(torch.sin(y), self.power) * out_coeff
        else:
            fi_out = y # mx.sym.cos(y).__pow__(power) * 1
        return fi_out
```

```
def sigmoid(X):
    return 1 / (1 + torch.exp(-X))

def gaussian(X,a=.0,b=1.0):
    return torch.exp(-((X-a)/b)**2)

def get_re_im_from_fi(fi,coeff = 1):
    re = coeff * torch.cos(fi)
    im = coeff * torch.sin(fi)
    return re, im

class ComplexConv(torch.nn.Module):
    def __init__(self, in_channel, out_channel, kernel_size, stride=1, padding=0, dilation=1,
groups=1, bias=True, complex_input=False):
        super(ComplexConv,self).__init__()
        self.padding = padding
        self.complex_input = complex_input
        ## Model components
        self.conv_re = torch.nn.Conv2d(in_channel, out_channel, kernel_size, stride=stride,
padding=padding, dilation=dilation, groups=groups, bias=bias)
        self.conv_im = torch.nn.Conv2d(in_channel, out_channel, kernel_size, stride=stride,
padding=padding, dilation=dilation, groups=groups, bias=bias)

    def forward(self, x): # shpae of x : [batch,2,channel,axis1,axis2]
#         real = self.conv_re(x[:,0]) - self.conv_im(x[:,1])
#         imaginary = self.conv_re(x[:,1]) + self.conv_im(x[:,0])
        if self.complex_input:
            x_re, x_im = x[:][0], x[:][1]
        else:
            x_re, x_im = get_re_im_from_fi(x,1.0)
        real = self.conv_re(x_re) - self.conv_im(x_im)
        imaginary = self.conv_re(x_re) + self.conv_im(x_im)
        output = torch.stack((real,imaginary),dim=0)
        return output
```