Image Classification with Quantum Convolution Neural Network

Naqeeb Ahmed Khan

(Report of Project Work in Machine Learning for Computer Vision)

1. Introduction

The evolution of computing technology has seen remarkable advancements, starting from the CPU, which serves as the brain of traditional computers, executing instructions in a sequential manner. As computational demands grew, the GPU emerged, offering parallel processing capabilities that dramatically accelerated tasks like image rendering and complex calculations, particularly in fields like gaming and scientific research. During this period, computer vision also began to flourish, initially tackling basic classification tasks and gradually advancing to more complex challenges such as object detection and image segmentation. Simultaneously, the datasets used in these tasks expanded in size and complexity, driving significant improvements in accuracy and performance, further fueling the growth and application of computer vision technologies.At the same time, quantum computers began to develop, leveraging the principles of quantum mechanics to offer unique advantages over classical computers, particularly in terms of speed and scalability. Intrigued by the potential of quantum computing, the student embarked on an exploration of how these machines could be applied to computer vision tasks.

Image classification is a core task in computer vision, which involves categorizing images into predefined categories based on their visual content. Traditional approaches using convolutional neural networks (CNNs) have achieved significant success in this field due to their ability to capture spatial hierarchical structures in image data. This report explores the application of quantum convolutional neural networks (QCNNs) to image classification by studying several papers and reproducing the results. This project trained and tested several proposed QCNNs to evaluate their performance and compare with classical CNNs, using the MNIST dataset as a benchmark. This research used a quantum simulator provided by IBM and the PennyLane platform, which allowed for a comprehensive comparison of these architectures in a controlled environment.

2. Methodology

Many researchers have different definitions of quantum convolutional neural networks. First, a novel quantum machine learning model inspired by classical CNNs has been introduced [1]. They proposed a fully parameterized quantum circuit (PQC) architecture and demonstrated its success for some quantum many-body problems. Not long after, Kerenidis et al. proposed a quantum algorithm for applying and training deep CNNs with the same name of QCNN, which mimics the behavior of a Quantum CNN [2]. In addition, others have proposed a new type of transformational layer called a quantum convolution, or quanvolutional layer [3]. In this project, we mainly study the above algorithms, quantum circuits and quantum convolutional layers.

2.1. Quantum Convolutional Neural Networks

The architecture proposed in the paper [1], leverages the Multi-Scale Entanglement Renormalization Ansatz (MERA) and quantum error correction techniques, using only (O(log(N))) variational parameters for input sizes of (N) qubits. This makes it efficient for training and implementation on near-term quantum devices. The paper demonstrates the QCNN's potential through applications in quantum state recognition and quantum error correction, significantly outperforming known quantum codes of comparable complexity. Although the article did not directly discuss quantum computer vision and no specific application examples, the analogy with classical CNN allows us to see the possibility of QCNNs as it shown in Figure 1, highlighting the model's promise in enhancing quantum machine learning and quantum error correction.

Based on this idea, IBM implemented such a QCNN on Qiskit by modeling both the convolutional layers and pooling layers using a specific quantum circuit, also trained it to differentiate horizontal and vertical lines from a pixelated image [4]. Figure 2 is an example of QCNN model for 4 input qubits. We can see similar with classical CNN, it has convolutional layers and pooling layers. By applying these alternating layers, it reduces the dimensionality of the circuit until only one qubit is left, then classify the input image by measuring the output of this one remaining qubit.

The Quantum Convolutional Layer will consist of a series of two qubit unitary operators, which recognize and determine relationships between the qubits in our circuit. The chosen circuit in Figure 3 has been proved as an optimal quantum circuit for a general two-qubit gate that requires at most 3 CNOT gates and 15 elementary one-qubit gates [5]. This two qubit unitary will be applied to all neighboring qubits in the feature map.

For the Quantum Pooling Layer, differently from the classical method, it reduce the number of qubits by performing operations upon each until a specific point and then disregard certain qubits in a specific layer. The specific pooling circuit is shown in the Figure 4.

The classical dataset can be encoded into QCNN by applying a Qiskit's built in feature map, such as ZFeatureMap or ZZFeatureMap. In the QCNN, each layer contains parametrized circuits, meaning we alter our output result by adjusting the parameters of each layer. When training our QCNN, it is these parameters that are adjusted to reduce the loss function of our QCNN.

2.2. Quanvolutional layer

The quanvolutional layer proposed in the paper [3], operates on input data by locally transforming it using random quantum circuits, similar to the transformations performed by random convolutional filter layers in CNNs. The authors empirically evaluate the potential benefits of these quantum transformations by comparing models built on the MNIST dataset. Their results show that QNNs achieve higher test set accuracy and faster training compared to purely classical CNNs, highlighting the promise of integrating quantum circuits into image recognition tasks.

As PennyLane[6] implemented the QNN on MNIST dataset shown in Figure 5, a small region of the input image, such as a 2×2 square, is embedded into a quantum circuit using parametrized rotations applied to qubits initialized in the ground state. A quantum

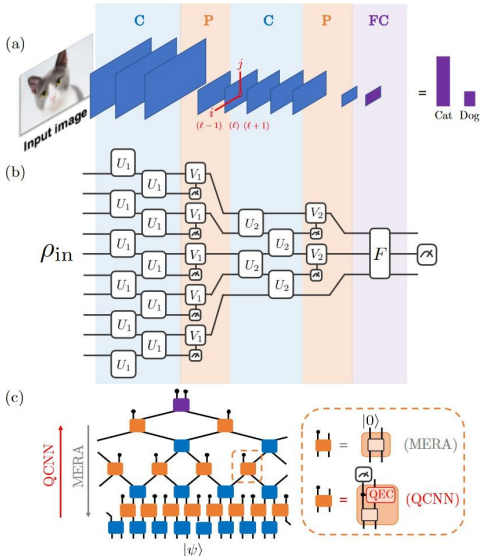

Figure 1. (a) Simplified illustration of CNNs. A sequence of image processing layers—convolution (C), pooling (P), and fully connected (FC)—transforms an input image into a series of feature maps (blue rectangles), and finally into an output probability distribution (purple bars). (b) QCNNs inherit a similar layered structure. (c) QCNN and MERA share the same circuit structure, but run in reverse directions.

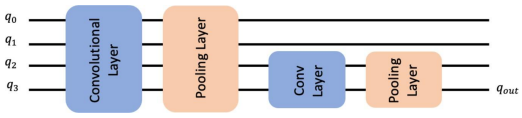

Figure 2. Example QCNN containing four qubits. The first Convolutional Layer acts on all the qubits. This is followed by the first pooling layer, which reduces the dimensionality of the QCNN from four qubits to two qubits by disregarding the first two. The second Convolutional layer then detects features between the two qubits still in use in the QCNN, followed by another pooling layer, which reduces the dimensionality from two qubits to one, which will be our output qubit.

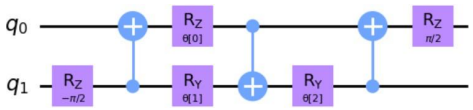

Figure 3. Parametrized two qubit unitary circuit

computation, associated with a unitary (U), is then performed on the system. This unitary can be generated by a variational quantum circuit or a random circuit as proposed in [3]. The quantum system is measured to obtain a list of classical expectation values, which can be directly used or classically post-processed. Each expectation value is mapped to a different channel of a single output pixel, similar to a classical convolution layer. By iterating this procedure over different regions, the full input image can be scanned, producing an output object structured as a multi-channel image. The quantum convolution

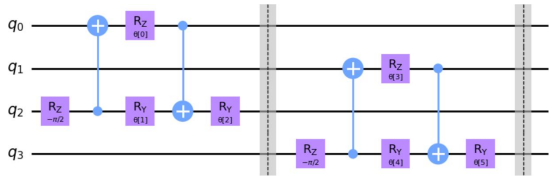

Figure 4. Pooling layer reduces the dimensionality from four qubits to the last two qubits.

can be followed by additional quantum layers or classical layers.

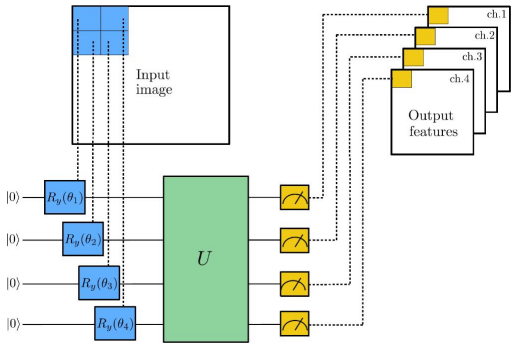

Figure 5. Quanvolutional Neural Network

2.3. QCNN

This experiment demonstrates the training of a small CNN enhanced with quantum elements using PyTorch [2]. A quantum sampling process, replacing traditional precision parameters with a more intuitive sampling ratio, is introduced after each convolution layer. Noise simulating amplitude estimation and quantum techniques like capReLU activation are integrated into the model. The core innovation is a quantum algorithm that efficiently computes convolution products between tensors, offering significant speedup, especially in high-dimensional tasks like time series or video processing. The approach uses quantum tomography to sample the most relevant information, reducing complexity while maintaining modularity for deep architectures. Although quantum non-linearity and tomography prevent exponential speedup, the technique enables effective training of CNNs, achieving similar accuracy to classical methods on the MNIST dataset. The backpropagation algorithm is also implemented as a quantum circuit. Overall, this work shows the potential of quantum CNNs for accelerating complex tasks in signal processing and machine learning.

3. Experiments and Results

3.1. Datasets

Horizontal or vertical line: A dataset of 2 x 4 pixel images to train a QCNN to distinguish between horizontal and vertical lines has been created. Each image contains either a horizontal or vertical line represented by a specific pixel value, while the remaining pixels have random values between 0 and 1 to introduce noise as shown in Figure6. The

dataset is divided into a training set and a testing set to evaluate the QCNN's performance. Images with a horizontal line are labeled -1, and those with a vertical line are labeled +1, enabling the QCNN to learn and differentiate these patterns amidst the noisy background.

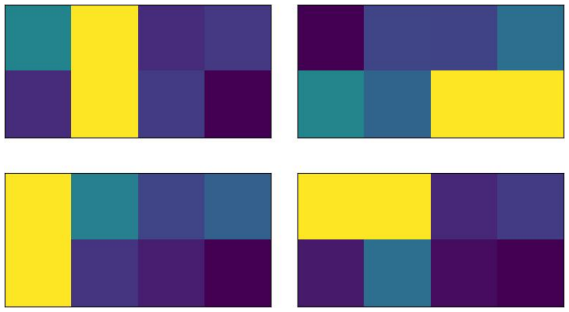

Figure 6. Dataset with vertical or horizontal line

MNIST: MNIST is a widely used dataset for machine learning, particularly for image classification tasks. It consists of 70,000 grayscale images of handwritten digits, with 60,000 images for training and 10,000 images for testing. Each image is 28 x 28 pixels and contains a single digit ranging from 0 to 9. MNIST is a standard benchmark dataset in the field of deep learning and is commonly used to test and evaluate various image recognition and classification algorithms. Our benchmark focuses on binary classification, and hence we select classes 0 and 1 of the dataset.

3.2. QCNN

3.2.1. Horizontal or vertical line dataset

We initially applied a specific QCNN quantum circuit proposed by IBM to the dataset featuring horizontal and vertical lines. Given that the dataset consists of 8 pixels, the quantum circuit employed has the same number of qubits as inputs. The design of the circuit is illustrated in Figure 7.

We utilized the Qiskit NeuralNetworkClassifier with the Constrained Optimization By Linear Approximation (COBYLA) optimizer and squared error loss function. The model was trained for 400 iterations on 35 training samples and evaluated on 15 test samples. The total training time was 107 seconds. The accuracy achieved on the training data was 91.43%, as shown in Figure 8, while the accuracy on the test set was lower, at 60.0%, as depicted in Figure 9.

We evaluated the performance of different optimizers and loss functions on the dataset with horizontal and vertical lines, as summarized in Table 1. For the Adam optimizer, we set the termination tolerance to 1×10^{4} and the learning rate to 0.01.

Among the COBYLA optimizer results, using cross-entropy loss yielded the best performance with a training accuracy of 94.29% and a test accuracy of 80.00%, indicating better generalization compared to squared error loss, which had a test accuracy of 60.00%. The absolute error loss showed the highest test accuracy of 86.67%, suggesting it was the most effective in preventing overfitting. The Adam optimizer, despite requiring a significantly longer training time of 38.11 minutes, achieved a notable test accuracy of

86.67% with a training accuracy of 91.43%. The Gaussian-smoothed Line Search (GSLS) optimizer performed poorly with both low training and test accuracies. In contrast, the Sequential Least SQuares Programming optimizer (SLSQP) optimizer delivered the best results overall, with perfect training accuracy of 100.00% and a high test accuracy of 93.33%, making it the most effective and efficient choice among the optimizers tested.

Table 1. Summary of QCNN results on horizontal or vertical line dataset

Optimizers	Iterations	Training Accuracy	Test Accuracy	Time (minutes)
COBYLA(squared_error_loss)	400	91.43	60.00	1.07
COBYLA(cross_entropy_loss)	400	94.29	80.00	1.48
COBYLA(absolute_error_loss)	400	85.71	86.67	1.43
ADAM(squared_error_loss)	100	91.43	86.67	38.11
GSLS(squared_error_loss)	100	60.00	40.00	7.35
SLSQP(squared_error_loss)	100	100.00	93.33	38.57

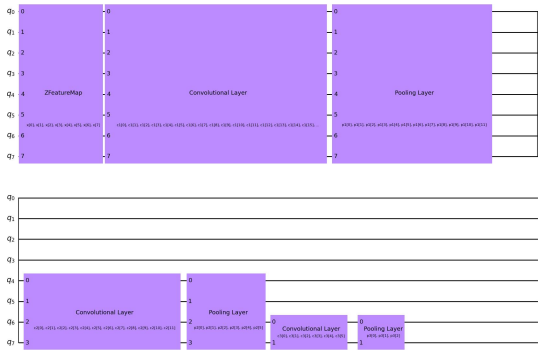

Figure 7. QCNN circuit for 8 qubits

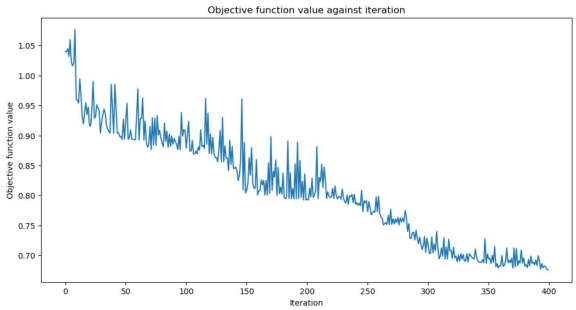

Figure 8. QCNN training results for horizontal or vertical line dataset

3.2.2. MNIST 0 and 1 dataset

When applying the model to the MNIST dataset, we first preprocessed the data to accommodate the QCNN model's input size, which is constrained by the quantum circuit's depth and simulation limits. To ensure compatibility and maintain a consistent network

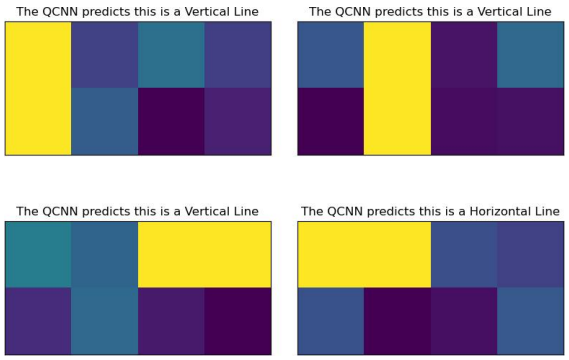

Figure 9. QCNN test results for horizontal or vertical line dataset

structure, we reduced the dimensions of the original 28x28 image data. The preprocessing steps are as follows:

- **Filter 0 and 1:**
  Filter out only the data with class labels 0 and 1.
- **Downsample:**
  Simply resize the images to 4x2 and 4x4 dimensions, then flatten them. Images after downsample are shown as in Figure12 and 13. Alternatively, apply Principal Component Analysis (PCA) to reduce dimensionality while retaining the most 8 important features, ensuring that the data remained representative after compression.
- **Normalize:**
  Normalize pixel values from the [0, 255] range to the [0.0, 1.0] range.
- **Remove duplicate or erroneous:**
  Eliminate any duplicate or erroneous data after dimensionality reduction.
- **Label adjustment:**
  Adjust the labels, mapping 0 and 1 to -1 and 1, respectively.
- **Shuffle:**
  Randomize the order of the selected dataset to prevent training biases.

Once the data is preprocessed, it is fed into the same quantum convolution circuit, where we train and test the model using various optimizers and loss functions, results details are shown in Table 2.

The analysis of the results reveals that using the COBYLA optimizer with cross-entropy loss yields the best performance, especially with longer training periods, where it achieved a test accuracy of 92.5% and a training accuracy of 98.0% after 400 iterations, shown in Figure 10 and 11. In contrast, squared error loss produced mixed results, showing high accuracy initially but suffering from overfitting with extended training. Absolute error loss performed poorly overall. The SLSQP optimizer with cross-entropy loss demonstrated strong performance even with fewer iterations. Larger input sizes, such as 4x4, led to lower accuracies and significantly longer training times compared to the 4x2 input, suggesting that a smaller input size may be more efficient for this dataset.

3.3. QNN

To further assess the performance characteristics of the QCNN with a specialized quantum circuit, we used the same dataset as in previous experiments, we constructed a QNN with

Table 2. Summary of QCNN results on 4x2 MNIST 0 & 1 dataset

Optimizers	Iterations	Training Accuracy	Test Accuracy	Time (minutes)
COBYLA(squared_error_loss)	100	88.0	92.5	1.90
COBYLA(squared_error_loss)	200	76.0	85.0	2.48
COBYLA(cross_entropy_loss)	100	92.0	87.5	1.20
COBYLA(cross_entropy_loss)	400	98.0	92.5	5.55
COBYLA(absolute_error_loss)	100	62.0	72.5	1.16
COBYLA(absolute_error_loss)	400	76.0	87.5	6.40
ADAM(cross_entropy_loss)	100	75.0	82.5	143.46
SLSQP(cross_entropy_loss)	20	84.0	92.5	5.26
COBYLA(se_loss)(4x4 input)	200	79.0	82.0	149.10

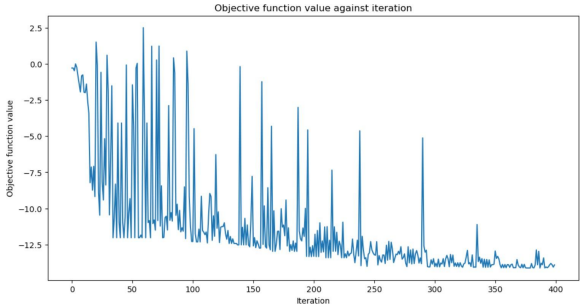

Figure 10. QCNN training results for 4x2 MNIST 0 & 1 dataset

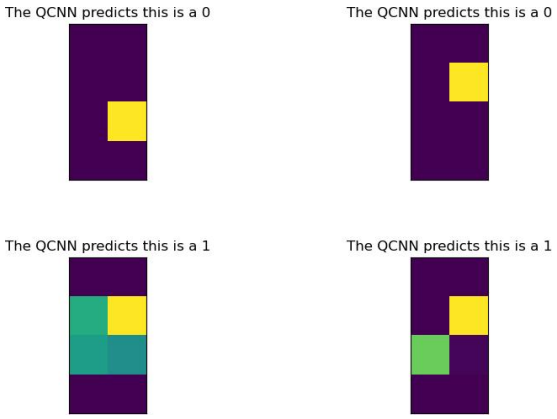

Figure 11. QCNN test results for 4x2 MNIST 0 & 1 dataset

the default ZZFeatureMap and RealAmplitudes ansatz from the QNNCircuit as in Figure 14. Under the same training conditions, the training results of QNN are not as accurate as those of QCNN, which proves the advantage of QCNN. Results details are shown in the Table 4 below.

3.4. Quanvolutional layer

As metioned above, the quantum convolutional layer, which can also called quanvolutional filter, processes 2x2 pixel blocks of the image using random quantum circuits to achieve similar functionality as a classical convolutional layer. Therefore, if each qubit

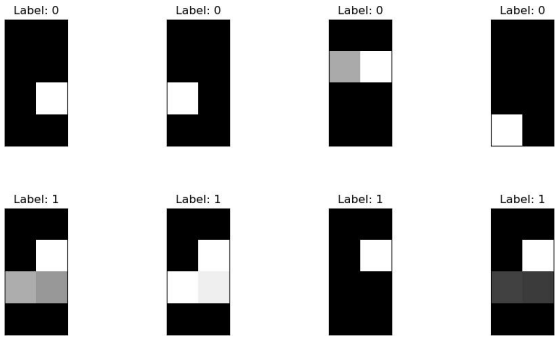

Figure 12. 4x2 MNIST 0 & 1 dataset

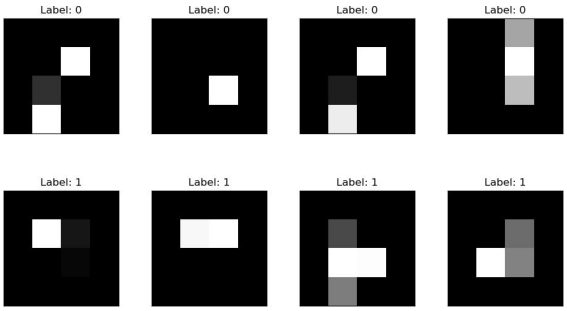

Figure 13. 4x4 MNIST 0 & 1 dataset

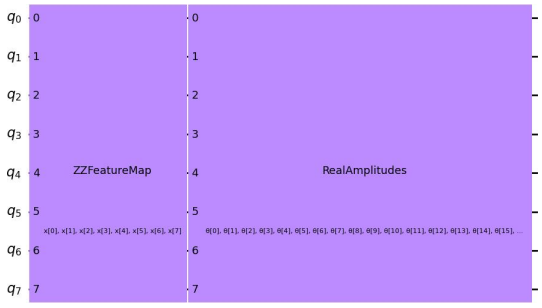

Figure 14. QNN circuit

processes the image as a separate channel, an input image of size 28x28 will yield a four-channel output of size 14x14, as illustrated in the Figure 15. The output results retain the majority of the data features. We then use these outputs as new training data by flattening them and passing them through a simple fully connected layer. This approach is compared to the classical model, which uses the original image data as input, employing the same straightforward neural network architecture.

The defined simple neural network model using Keras, it consists of a flatten layer to convert the input into a 1D array and a Dense layer with 10 units and a softmax activation function to output class probabilities. The model is compiled with the Adam optimizer and uses sparse categorical cross-entropy as the loss function, suitable for multi-class classification since the MNIST dataset has 10 classes. The model will be trained for 30 epochs, and the dataset sizes are set to 50 for training and 30 for testing. The results are

Table 3. Summary of QCNN results on PCA8 MNIST 0 & 1 dataset

Optimizers	Iterations	Training Accuracy	Test Accuracy	Time (minutes)
COBYLA(squared_error_loss)	100	73.0	80.0	1.23
COBYLA(squared_error_loss)	200	73.0	75.0	2.27
COBYLA(cross_entropy_loss)	100	79.0	77.5	1.15
COBYLA(cross_entropy_loss)	400	87.0	65.0	5.20
SLSQP(squared_error_loss)	100	85.0	80.0	126.24
SLSQP(cross_entropy_loss)	14	73.0	70.0	5.19

Table 4. Summary of QNN results on PCA8 MNIST 0 & 1 dataset

Optimizers	Iterations	Training Accuracy	Test Accuracy	Time (minutes)
COBYLA(squared_error_loss)	100	72.0	62.5	1.44
COBYLA(cross_entropy_loss)	400	55.0	55.0	7.16

shown in Figure 16, indicate that the performance curves for both quantum and classical inputs are closely aligned. Even after expanding the training set, the results remain consistent, demonstrating that the quantum layer preserves the data nearly as effectively as the original model. However, these transformations require additional computation time, and no clear quantum advantage over classical non-linear methods was observed.

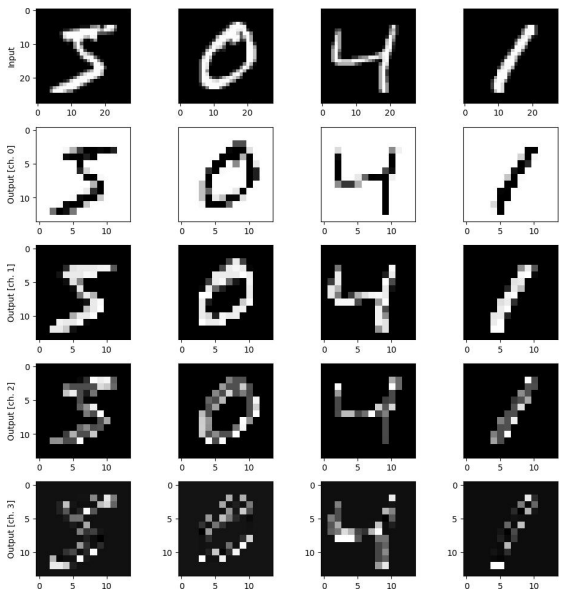

Figure 15. Input and outputs of quanvolutional layer

4. Conclusion

In this study, we explored the effectiveness of QCNNs for image classification tasks, comparing their performance against classical neural networks and evaluating various optimizers and loss functions. The experiments were conducted on two distinct datasets:

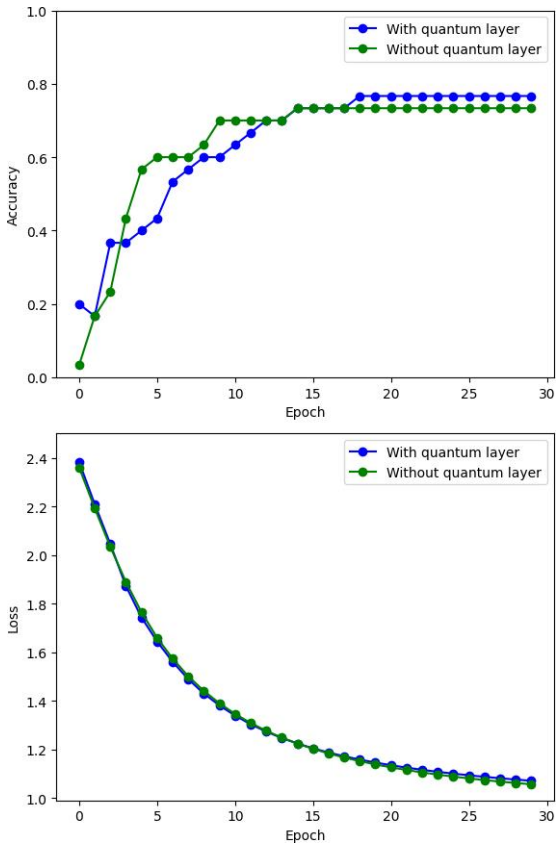

Figure 16. Results of quanvolutional network and classical network

a synthetic dataset with horizontal and vertical lines, and a subset of the MNIST dataset containing only digits 0 and 1.

The results demonstrated that QCNNs can effectively classify simple patterns, with performance varying based on the choice of optimizer and loss function. Specifically, the COBYLA optimizer gave decent results in a very short time and the SLSQP optimizer with cross-entropy loss proved to be the most effective configuration for the horizontal and vertical lines dataset, achieving high accuracy on both training and test sets. Conversely, the Adam optimizer, while time-consuming, also showed strong results, particularly for more complex data.

For the MNIST dataset, preprocessing steps such as downsampling and dimensionality reduction were crucial for adapting the data to the QCNN model. The use of the COBYLA optimizer with cross-entropy loss resulted in the best performance on the downsampled 4x2 images, highlighting the effectiveness of this configuration. In contrast, larger input sizes led to diminished accuracy and increased training times, underscoring the trade-offs between model complexity and computational efficiency.

We also compared the QCNN to a QNN using the same dataset. The QCNN outperformed the QNN, validating the advantage of specialized quantum circuits in extracting and learning features from the data. However, the comparison between the quantum convolutional layer input and classical input revealed that, while the quantum layer preserves data features effectively, there was no significant quantum advantage over classical

methods in terms of performance or computational efficiency.

In conclusion, QCNNs present a promising approach for image classification tasks, especially when tailored to the specific characteristics of quantum circuits and datasets. Despite the current limitations in terms of computational resources and complexity, QCNNs show potential for future advancements in quantum machine learning, particularly as quantum hardware and algorithms continue to evolve. Further research and experimentation are necessary to fully realize the advantages of quantum computing in practical applications and to overcome existing challenges in scalability and efficiency. In addition to advancing the design and optimization of quantum circuits, future research should focus on the challenges associated with encoding classical data into quantum circuits. Effective data encoding is crucial for bridging the gap between traditional machine learning and quantum machine learning.

References

[1] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum Convolutional Neural Networks. 15(12):1273–1278.

[2] Iordanis Kerenidis, Jonas Landman, and Anupam Prakash. Quantum Algorithms for Deep Convolutional Neural Networks.

[3] Maxwell Henderson, Samriddhi Shakya, Shashindra Pradhan, and Tristan Cook. Quanvolutional neural networks: Powering image recognition with quantum circuits. 2(1):2.

[4] Qiskit Community. Quantum convolutional neural networks, 2024.

[5] Farrokh Vatan and Colin Williams. Optimal Quantum Circuits for General Two-Qubit Gates. 69(3):032315.

[6] Pennylane. Quanvolutional neural networks, 2024.