Novel Quantum Image Representation & Data Compression for Classifying Images via Quantum Classifiers & Neural Networks

Avi Veeramoothoo & Munia Humaira
QIC 890 / ECE 730: Quantum Machine Learning,
University of Waterloo, Waterloo, ON

***Abstract*—This project explores novel quantum image representation (QIR) and data compression techniques for classifying digital images using quantum classifiers and neural networks. The goal of our work is to compare classical machine learning models with their quantum counterparts and benchmark their performances. This work uses the Improved Novel Enhanced Quantum Representation (INEQR) technique for effective quantum statevector encoding and the Quantum Hadamard Edge detection (QHED) technique to extract the outlines of such images for data compression. We have used the MNIST, CIFAR-10, and mock datasets with various pre-processing steps, including image resizing and grayscale conversion. Our study compares Support Vector Machines (SVM) vs. Quantum SVM (QSVM) and Convolutional Neural Networks (CNN) vs. Quantum CNN (QCNN). Results showed that QSVM achieved high training accuracy (100%) but suffered from overfitting and significantly longer computation times than classical SVM. While with a toy dataset QSVM followed by INEQR showed promising results in training and testing accuracies. QCNN exhibited a lower accuracy of $\sim 18\%$ than classical CNN for multiclass classification, primarily due to necessary image downsizing. This research highlights the potential of quantum machine learning techniques in image classification while also identifying current limitations and areas for future investigation, such as optimizing quantum methods for larger datasets and higher-resolution images.**

***Index Terms*—Quantum Machine Learning, Quantum Image Representations, INEQR, Quantum Edge Detection, Hybrid quantum-classical architecture.**

I. Introduction

ML models solve various problems, including classification. Classification is particularly useful in safety-critical applications (e.g. self-driving cars) and recommender systems (e.g. social networks). Both of these applications require high accuracy and fast computational speed to ensure safety, drive user engagement and help monetization. Many times, they ingest a high volume of feature-rich images but only need to consider a few features to make their decisions. Thus, we are interested in combining quantum-enabled data compression techniques with quantum ML models to investigate if any advantage emerges. We consider data compression methods like reducing images to their edges and vector embeddings. Then, we explore how they can assist Quantum Support Vector Machine (QSVM) and Quantum Convolutional Neural Network (QCNN) models. The initial setup of some trials may not work well due to constraints of current technologies (e.g. number of qubits available on IBMQ), so we will adjust some complexity variables and retry with a more feasible setup. Ultimately, we report our findings and suggest setups for future trials when/if those constraints are eventually overcome.

II. Theory

We would like to start our report by giving a brief overview of the machine learning models we have used in our work both in classical and in quantum regimes.

A. SVM vs. QSVM

One of the most crucial machine learning tools is data classification, which can be used to find, classify, and analyze new data related to a wide range of use cases, including computer vision issues, medical imaging, drug development, handwriting recognition, object classification, and many more. Support-vector machines (SVM) are one of the most widely used data classification techniques in machine learning. SVM is a supervised learning technique that is especially helpful since it uses a hyperplane to divide training data into two classes, allowing classification into one of the two categories. In essence, the hyperplane and support vectors are data points employed to maximize the margin between classes.

In real world scenarios, the data are not linearly separable, in those cases one needs to use different kernel methods to effectively classify data. It enables SVMs to handle non-linearity in data by mapping the input features into higher-dimensional space. A kernel can be defined as a function $k(\vec{x_i}, \vec{x_j})$ that determines the dot product of two data points in the higher-dimensional feature space. Direct computation in the new space is made possible via the kernel function rather than explicit data transformation. It can also be expressed in matrix representation as ,

$$K_{ij} = k(\vec{x_i}, \vec{x_j}) \tag{1}$$

In classical SVM notable kernel functions are polynomial, radial basis function, Gaussian functions, etc.

The QSVM can be thought of as the quantum counterpart of the classical SVM. This algorithm was first proposed by Rebentrost et al. [1]. Rebentrost, however, suggests that the evaluation of inner products in the dual form of SVM can be completed more quickly on a quantum computer due to the inherent ability of quantum computers to map data to higher dimensional spaces. This concept is expanded by Havelicek

et al. [2], who provide kernel methods and quantum feature maps that help estimate the necessary kernel function by non-linearly mapping data inputs to higher-dimensional quantum states. In order to encode features into the amplitudes or phases of a quantum system, a sequence of quantum gates is usually applied. The overlap between quantum states in the Hilbert space is measured to calculate the kernel function. In QSVM, two digital data points x_i, x_j are converted into quantum state vectors $\phi(\vec{x_i}), \phi(\vec{x_j})$ and the quantum kernel functions becomes,

$$K_{ij} = |\langle \phi(\vec{x_i}) | \phi(\vec{x_j}) \rangle|^2 \tag{2}$$

Although quantum systems operate in increasingly huge spaces by nature, feature mappings may become more expressive. Research is still ongoing to create efficient quantum feature maps that perform better than their classical equivalents.

B. CNN vs. QCNN

Classical convolutional neural network (CNN) applies linear convolutional filters and nonlinear activation functions to extract hierarchical features from data. For an input $\mathbf{x} \in \mathbb{R}^N$, a single convolutional layer may be represented as

$$f^{(l)} = \sigma(\mathbf{W} * \mathbf{x}^{(l-1)} + \mathbf{b}), \tag{3}$$

where $\mathbf{W}$ is a learnable convolutional kernel, $\sigma(\cdot)$ is a nonlinear activation function, and $*$ denotes the convolution operation. In contrast, a quantum convolutional neural network (QCNN) applies parameterized unitary transformations $U(\boldsymbol{\theta})$ on subsets of qubits, followed by measurements or partial traces analogous to pooling. By exploiting quantum superposition and entanglement, QCNNs can encode and process correlations that are intractable for classical networks, potentially offering exponential advantages for certain quantum information processing tasks. The encoding methods include amplitude, angle, phase and entangled encoding. For our experiments, we proceed with the default angle encoding that Pennylane provides.

III. Quantum Image representation

This section will discuss the techniques we have employed in our research to compress the data of the classical images and to represent them in the quantum realm.

A. Improved Novel Enhanced Quantum Representation

For the objective of representing digital images on quantum devices, QIR is a data embedding approach that offers a bridge between classical and quantum platforms. Certain unitary transforms can be used to encode characteristics of digital images, such as pixel position and color information, onto a quantum circuit. For our work we have chosen the INEQR method introduced by Nan Jiang et al. [3]. This method enables representing non-square images with support for image resizing and scaling. However, the image dimensions must be powers of 2, such as $2, 4, 8, 16$, and so on. Using this method one can represent a $2^{n_1} \times 2^{n_2}$ grayscale image as,

$$|I\rangle = \frac{1}{2^{\frac{n_1+n_2}{2}}} \sum_{y=0}^{2^{n_1}-1} \sum_{x=0}^{2^{n_2}-1} |f(y,x)\rangle |yx\rangle \tag{4}$$

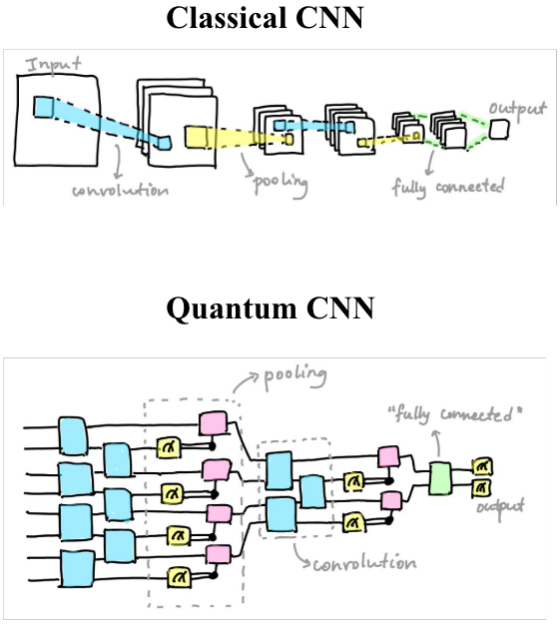

Fig. 1. CNN and QCNN architecture from Pennylane glossary website [8]

.

$$= \frac{1}{2^{\frac{n_1+n_2}{2}}} \sum_{y=0}^{2^{n_1}-1} \sum_{x=0}^{2^{n_2}-1} \bigotimes_{(i=0)}^{(q-1)} |C_{yx}^i\rangle |yx\rangle \tag{5}$$

where the grayscale value are encoded by *multi-CX* transform $f(y,x) = C_{yx}^{q-1}, C_{yx}^{q-2}, C_{yx}^{q-3} ... C_{yx}^{i}$ and the pixel positions are encoded using multiple *Hadamard* gates as,

$$|yx\rangle = |y\rangle |x\rangle = |y_0 y_1 ... y_{n_1} - 1\rangle |x_0 x_1 ... x_{n_1} - 1\rangle \tag{6}$$

where $y_i, x_i \in [0, 1]$.

This method uses in total $n_1 + n_2 + q$ qubits to denote the y-coordinate, x-coordinate and grayscale values of a quantum image. For example, an image of a digit 5 from the MNIST dataset has been shown in Fig.2 to be embedded into a quantum circuit using the python package piQture [4] that implements this method.

For our purpose, MNIST dataset is not appropriate to work with. Hence, we focused on the CIFAR-10 dataset that have $50,000$ training images and $10,000$ test images with pixel size of 32×32. Next section will focus on the experiment methods and results of our work.

B. Quantum Hadamard Edge Detection (QHED)

Yao et al. [5] developed quantum image processing for edge detection. Quantum Hadamard Edge Detection (QHED) is a flavor of their generalized technique, implemented by Bulkapuram et al. [6], by using specifically Hadamard gates – illustrated in Fig. 2. The steps of QHED are as follows:

1) Normalize the pixel intensities.

$$c_i = \frac{I_{xy}}{\sqrt{\sum I_{xy}^2}} \tag{7}$$

where I_{xy} denotes the intensity of the pixel at coordinates (x, y).

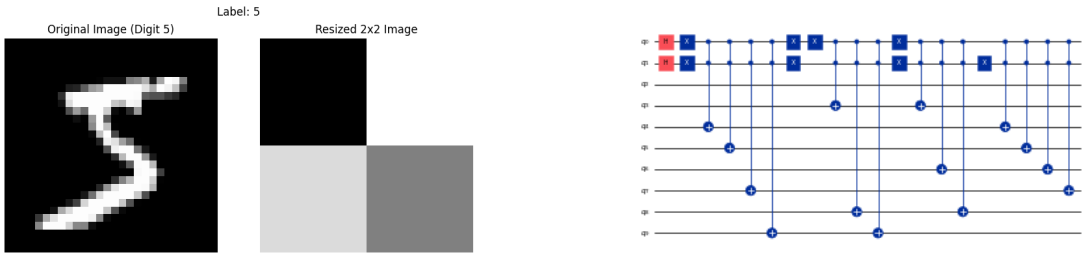

Fig. 2. (a) A digital image of digit $'5'$ converted into 2×2 pixel sized grayscale image and (b) embedded into quantum circuit using 2 Hadamard gates and 8 CNOT gates for pixel values $[[53, 67], [74, 60]]$

.

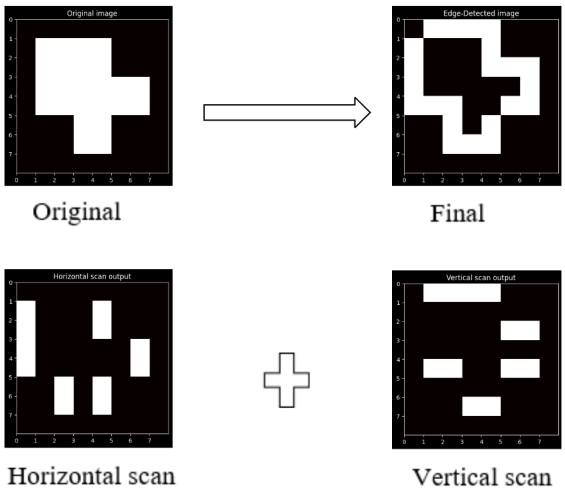

Fig. 3. Steps to perform QHED

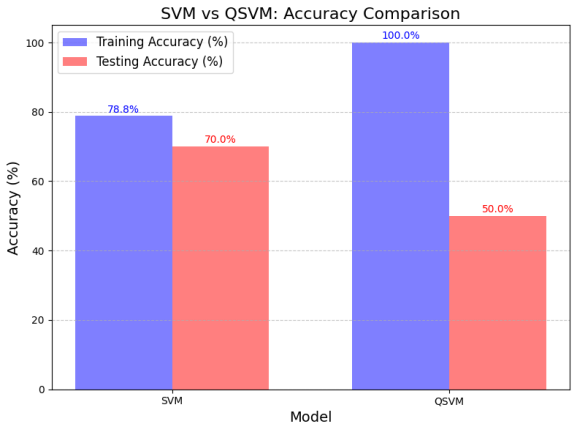

Fig. 4. Model accuracy comparison of SVM and QSVM.

2) Encode the pixel intensity using Quantum Probability Image Encoding (QPIE) technique.

$$|\text{Img}\rangle = \sum_{i=0}^{2^n-1} c_i |i\rangle \tag{8}$$

3) Create a quantum circuit that applies H gate and decrement gate (i.e., amplitude permutation unitary gate).

$$|\text{Img}\rangle \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{bmatrix} c_0 \\ c_0 \\ c_1 \\ c_1 \\ \vdots \\ c_{N-2} \\ c_{N-2} \\ c_{N-1} \\ c_{N-1} \end{bmatrix} \tag{9}$$

4) Perform a horizontal scan to get vertical edges.
5) Perform a vertical scan to get horizontal edges.
6) Combine the results of the two scans to construct the image outline.

IV. Methods & Results

A. SVM & QSVM

Our first objective was to use IBM's QSVM and Sci-Kit Learn's SVM to benchmark and compare the performance of both the classical SVM and QSVM with the same amount of data. However, we faced some bottlenecks while working with the large amount of data and 32×32 pixel size images. We have chosen 500 training data points, 10 test data points and 2 classes out of 10 and the images are resized to 8 pixels due to the lack of computing resources we had. Also another problem being the quantum simulators are constrained in terms of qubit size it can work with.

SVM was implemented in code following the Sci-Kit Learn's convention and for QSVM the training data is encoded into a quantum state using a parameterized quantum circuit, commonly referred to as a feature map. A data point from the training dataset is represented by each of the circuit's parameters. We used a particular kind of parameterized quantum circuit called the *ZZFeatureMap* to efficiently encode the data points into the feature map. Both of these approaches' accuracy as well as their training and testing time comparisons are displayed in Fig.3 and Fig.4.

From Fig.4 we can see that with the small amount of data classical SVM shows a reliable result of training and testing accuracies of 78.8% and 70%, respectively. Whereas, QSVM

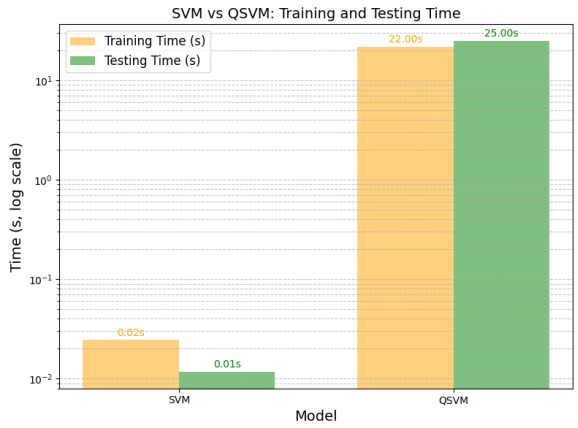

Fig. 5. Training and Testing Time Comparison of SVM and QSVM

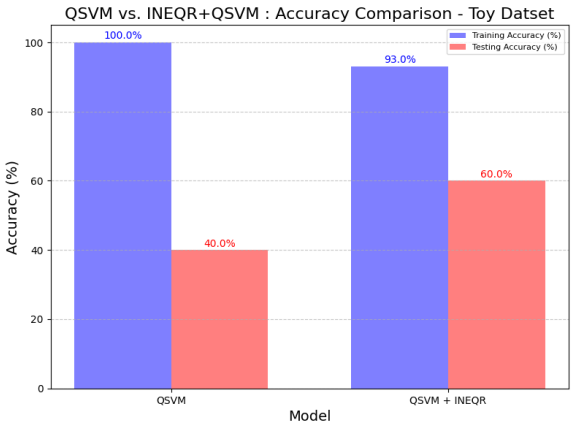

Fig. 7. Model performance comparison of training and testing accuracies of QSVM and QSVM+INEQR

shows excellent training accuracy of 100% but the testing accuracy halves.

In Fig.5 we can see that QSVM has much higher training $(22\ s$ vs. $0.02\ s)$ and testing time $(25\ s$ vs. $0.01\ s)$ than its classical counterpart.

B. QSVM & INEQR-based QSVM

To compare which strategy produces better outcomes, our study's next logical step is to compare the performance of QSVM and the INEQR-based QSVM. However, qiskit's qubit requirements, which are restricted to 2^{16}, place limitations on how the circuits can be encoded and built. In order to overcome these constraints, we chose a toy dataset with two classes out of ten, five test data points, fifteen training data points, and images scaled to 4×4 pixels.

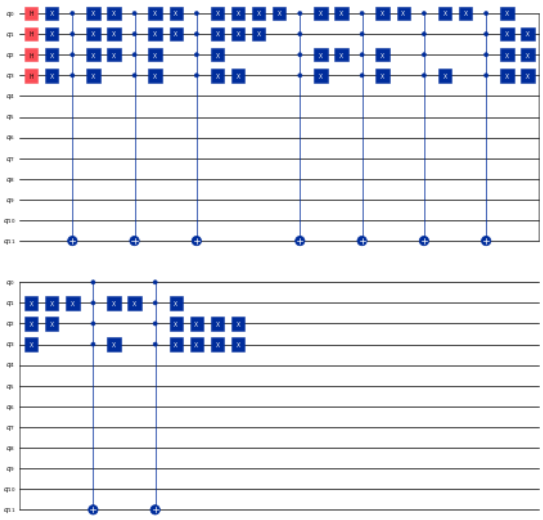

Fig. 6. A train grayscale image of 4×4 pixel embedded into quantum circuit using piQture's INEQR implementation

Additionally, encoding each image using the INEQR method is computationally intensive, further justifying the choice of a simplified dataset for this experiment. In Fig.6 a quantum circuit of one embedded image is shown with a depth of 36 and size of 77.

Comparing qiskit's QSVM using the same *ZZFeatureMap* and piQture's INEQR implementation to encode the digital images in quantum statevectors and then performing the machine learning techniques on these data points give us the results shown in Fig.7 and Fig. 8. It is evident from Fig.5 that although the QSVM exhibits a training accuracy of 100%, the testing accuracy of 40% falls short. On the other hand, the QSVM based on INEQR exhibits balanced training and testing accuracies of 93% and 60%.

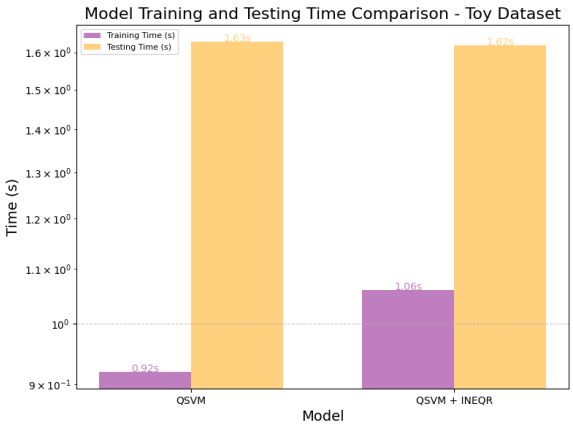

Fig. 8. Time taken by QSVM and QSVM+INEQR to train and test the toy dataset

.

The training and testing times for both models are nearly equal, as shown in Fig.8. This result indicates that we can observe some progress in classification accuracy even though we may not have much control over training and testing time on INEQR-based QSVM.

C. QCNN & QHED

CNN/QCNN operates as a network of neurons built as a stack of layers. Our experiment seeks to combine QHED as a starter layer in the QCNN.

1) Original setup: We wanted to test the full potential of the QCNN to ensure a fair comparison with the classical counterparts. This meant using the complete Cifar-10 dataset with its 50,000 datapoints of 32x32 color images from 10 classes. However, Pennylane and Qiskit could not handle more than 2x2 images. It would exceed the free 10-min limit on IBMQ. On the local simulator, it would run beyond 1 hour, which would already be in excess of the classical versions which took a less than 5 mins.

2) Adjusted setup: Instead of solely focusing on benchmarking against classical CNN, we adjusted our approach to see if QHED and QCNN would even work. Thus, we reduced the Cifar-10 images to dimension 2x2. Moreover, the implementation of QHED currently works only with black-white images, so we converted the Cifar-10 images to grayscale. Since the simulator was still taking long, we performed the QHED and QCNN separately, and then passed their data through a pipeline. This should not bear any effect on the accuracy, only the time.

After all preliminary adjustments, we trained the model on a 80-20 train-test split, using Pennylane, TensorFlow and Scikit-learn. Another challenge came up. TensorFlow acts as a framework to abstract some tedious ML operations for the end user. Behind the scenes, it performs ML quantisation, which is a form of data typecasting, aimed at classical ML operations. However, when using it for QCNN, it did not properly handle the case for complex numbers – common in Quantum computing and Hadamard gates. The built-in functions were truncating the imaginary parts of the values.

Consequently, our final results for QHED+QCNN were significantly affected by image downsizing and unwanted typecasting of complex numbers. Ultimately, we observe that QHED-based QCNN took 3 times longer than classical CNN alone for both training time and prediction time (Fig. 8), but performed 4 times worse in terms of accuracy (Fig. 9).

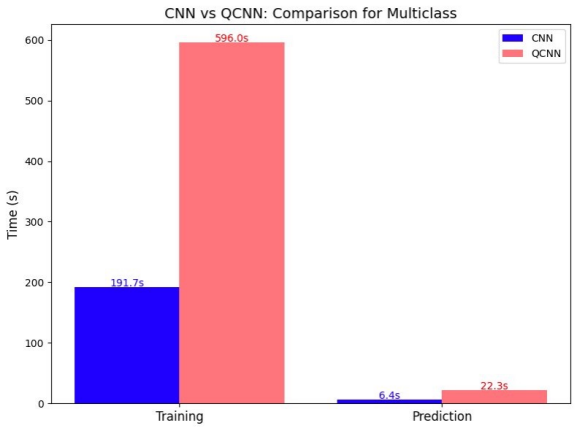

Fig. 9. Time taken by CNN and QCNN/QHED to train and test the toy dataset

.

V. Conclusion

The results obtained show the advantages and disadvantages of different quantum machine learning techniques when applied to low-resolution photos and short datasets. The QSVM showed a propensity to over-fit by achieving perfect training accuracy (100%) while struggling with testing accuracy (50%). On the other hand, the classical SVM consistently performed well, confirming its dependability for these kinds of datasets. Quantum embeddings have the potential to balance accuracy and resource utilization, as demonstrated by the integration of the INEQR technique with QSVM. A higher compromise between generalization and over-fitting was demonstrated by QSVM+INEQR, which improved testing accuracy to 60% despite achieving somewhat lower training accuracy (93%). According to these results, QSVM may have trouble with over-fitting, but by optimizing performance and computational resource consumption, INEQR improves its usefulness in real-world scenarios. Even with limited resources, this development highlights the potential of quantum embeddings to advance machine learning techniques. To increase accuracy and efficiency even further, more research might look into scalability and parameter adjustment.

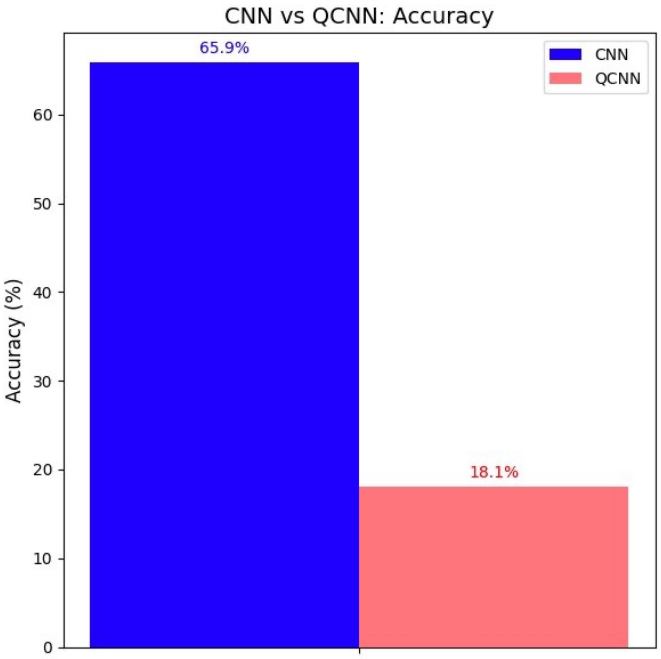

Fig. 10. Model performance comparison of training and testing accuracies of CNN and QCNN/QHED

.

Our QCNN experiments focused on multi-class classification, which is a more complex case than binary ones. The accuracy is 18.1%. This result proves that the QCNN is still no better than a random guess and has quite a large room for imporvement. As IBMQ matures, we should retry the experiments with more qubits, less preprocessing and all ran on real hardware. For the longer term, we recommend a case of applying QHED to reduce an image to its edges, then INEQR to embed it on a vector and finally training it on QCNN.

References

[1] Rebentrost, Patrick and Mohseni, Masoud and Lloyd, Seth *Quantum Support Vector Machine for Big Data Classification*. American Physical Society, Physical Review Letters, 113(13):130503, 2014.

[2] Vojtech Havlicek, Antonio D. C´orcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta*Supervised learning with quantum enhanced feature spaces*. Nature, 567(7747):209–212, 2019.
[3] Nan Jiang, Jian Wang, and Yue Mu., *Quantum image scaling up based on nearest-neighbor interpolation with integer scaling ratio.*, Quantum Information Processing, 14(11):4001–4026, 2015.
[4] Sasha Joshi, *piQture*, https://github.com/SaashaJoshi/piQture.
[5] Xi-Wei Yao, Hengyan Wang, Zeyang Liao, Ming-Cheng Chen, Jian Pan, Jun Li, Kechao Zhang, Xingcheng Lin, Zhehui Wang, Zhihuang Luo, Wenqiang Zheng, Jianzhong Li, Meisheng Zhao, Xinhua Peng, and Dieter Suter, *Quantum Image Processing and Its Application to Edge Detection: Theory and Experiment*. Phys. Rev. X, 7, 031041 (2017). doi:10.1103/PhysRevX.7.031041.
[6] Akshara Bulkapuram, *Quantum-Hadamard-Edge-detection*, GitHub repository, 2021. Available at: https://github.com/Akshara-Bulkapuram/Quantum-Hadamard-Edge-detection
[7] AnkRaw, *Quantum Convolutional Neural Network*, GitHub repository, 2024. Available at: https://github.com/AnkRaw/Quantum-Convolutional-Neural-Network
[8] PennyLane, *Quantum Convolutional Neural Networks*, 2024. Available at: https://pennylane.ai/qml/glossary/qcnn

VI. Acknowledgment

ChatGPT has been used to construct well-structured sentences in some parts of the report and Grammerly has been used to check for grammatical errors.

Appendix

Munia worked on the SVM, QSVM and QSVM+INEQR topics. Avi worked on the CNN, QCNN, QCNN+QHED topics. We have our code base has been uploaded in a repository in GitHub: $https : //github.com/Avi - Veeramoothoo/QIC890_Project$