Comparative Analysis of Machine Learning and Quantum Machine Learning: Assessing the Viability of Quantum Approaches in Real-World Problem Solving

Md. Shahriar Islam Bhuiyan
Dept. of Computer Science and Engineering
Islamic University of Technology
Gazipur, Bangladesh
190041124

Shaikh Faiyaz Karim
Dept. of Computer Science and Engineering
Islamic University of Technology
Gazipur, Bangladesh
190041116

Sajid Ahmed Chowdhury
Dept. of Computer Science and Engineering
Islamic University of Technology
Gazipur, Bangladesh
190041140

***Abstract*—Classical Computers and Classical Machine Learning algorithms have been tried and tested on a lot of real-world problems. But they suffer from a lot of training time and extensive computing resources to be able to give useful predictions. Quantum Computing and Quantum Machine Learning algorithms show the promise to solve these drawbacks and solve problems more efficiently. In this paper, we experimented with Variational Quantum Circuits (VQC), Quanvolutional Neural Networks (QNN), Quantum Convolutional Neural Networks (QCNN) and compared them against their classical counterparts. Our experiments showed that VQCs and QCNNs work best with a higher number of features and QNNs work better with an increasing number of training samples. But QML algorithms have a lot of grounds to cover to catch up to their classical counterparts on tasks that don't require the quantum advantages QCs bring.**

***Index Terms*—Machine Learning, Quantum Computing, Quantum Machine Learning, ML, QML, SVC, VQC, QNN, QCNN**

I. Introduction

Classical machine learning (henceforth referred to as CML) algorithms have been researched for decades and have reached enough maturity to be used in real-world problem solving. But due to the design and implementation of classical algorithms, the main problem CML algorithms face is speed. Furthermore, training CML models require a lot of data to obtain desirable accuracy in complex problems. Quantum Computers are a promising technology that offers exponential speedup in complex problem-solving compared to super-computers. Even if Quantum Computers are in a very early stage in development, quantum algorithms are maturing very fast. Amongst which, Quantum machine learning (henceforth referred to as QML) algorithms have seen extensive research in recent times. There are 4 ways of combining machine learning and quantum computers depending on the input data source and the whether the processes used are classical or quantum:

1) CC: Classical Data, Classical Algorithm
2) CQ: Classical Data, Quantum Algorithm
3) QC: Quantum Data, Classical Algorithm
4) QQ: Quantum Data, Quantum Algorithm

For this work, we'll be focusing on CQ, i.e., applying quantum algorithms and circuits on classical datasets such as MNIST and Iris. The paper will focus on highlighting differences between CML algorithms and their quantum counterparts on a simulated environment.

II. Literature Review

As we emerge into the future of quantum computers it is often wondered how it will change the application of machine learning and pattern recognition. As such it is imminent to know the viability of QML in solving real-life problems. To serve that purpose this paper gives a comparative analysis of CML and QML to answer that query.

Classical Algorithm lacks memory capacity and also have very high cost learning [1]. In paper [1] they tried to tackle the training cost of CML. This was successful due to the principle of superposition and a nonlinear quantum operator.

In another work, they compare SVM and the QML counterpart Quantum SVM on the basis of Big Data. It was seen in the research that the QSVM used on a real quantum has several benefits including data privacy and as a component to be used in a larger network [2]. This further proves that QML is something that is to be explored and can be a significant tool in the future.

Quantum-Inspired Binary Classifier (QIBC) model, is a model that takes inspiration from quantum principles and techniques

and applies them in binary classification. This QIBC is shown to outperform all baselines for the MNIST Handwritten Image Dataset [3]. However, in this paper, it is the model works really good in some examples but not so well in some other. To understand why this happens micro-analysis is needed.
In [4] they show some comprehensive comparative analysis of several CML and QML. Among those the first comparison was shown in SVM and QSVM [5]. It is seen that the SVM has a accuracy of around 85 percent whereas the QSVM outperforms it with an accuracy of around 90 percent.
Furthermore, a comparison on QNN and CNN is seen. Here it is observed that QNN performs slightly better. The MNIST Dataset of Handwritten letters was used. Still, the results could be better.

Algorithm	Loss Function	Test Accuracy
CNN	1.0757	1.0757
QNN	0.9882	0.7000

Inspecting these results and studies a decent potential of QML was recognized in the real life application. Through this paper we tried to give more comparison between other CML and respective QML to see how does the QML compare to CML and hence deduce the viability of QML and give an analysis of QML in the future of machine learning.

III. Methodology

A. SVC and VQC

For the first part of the comparative analysis we have chosen a classification algorithm both in the classical paradigm and the quantum paradigm. For the classification algorithm SVC (support vector classification) is used whereas VQC (Variational Quantum Classifier) is used as the classification algorithm in the quantum realm.

1) Data Acquisition: The dataset that we used for the classification algorithms is the IRIS data. The IRIS dataset contains four different attributes of an iris flower, namely sepal length, sepal width, petal length and petal width. According to these attributes the iris is classified into Iris-Setosa, Iris-Versicolour and Iis-Virginica. [6]. It is our goal to train the model so that it can classify the IRIS based on the attributes given to it.
The data set contains four attributes of an iris, and the goal is to classify the class of iris based on these four attributes.

2) Preprocessing: It is almost a necessity that before training the data set with SVC, the data set needs to be scaled. Features scaling can be done many common techniques including normalization or standardization. For the purpose of this work the MinMaxScaler from Sklearn library was used.
For the VQC(Variational Quantum Classifier) it was required to transform the bits into quantum bits or qubits for the quantum model to work. This step is crucial in order to build an effective quantum model. The process of mapping from bits to qubits is called encoding. In order to transform into qubits we use the feature mapping. Particularly the ZZFeatureMap of the qiskit library is used. The ZZFeatureMap will take in the number of feature i.e. will have 4 qubits as there are 4 feature attributes.
During the second round of evaluation the features were reduced to see how that would affect both the models. Here the feature were reduced to two feature from four feature.

3) Proposed method: SVC (Support Vector Classification) is a popular machine learning algorithm used for classification tasks. [7] It operates under the supervised learning algorithm where the answers or labels are already given to us. In SVC the data is a set of feature with their corresponding label and can be represented as a vector in a feature space. The goal of the SVC is to find the hyperplane that separates the classes clearly.

The outcome of SVC is to find an optimal hyperplane for proper distinction of the classes. For a linearly separable case such as the IRIS dataset, the hyperplane equation can be defined as $w^T \cdot x + b = 0$ where w is the weight vector and b being the bias values. The distance between the hyperplane and the closest data points from each class is margin. We can calculate the margin by $\gamma = \min\left(\frac{y_i \cdot y_j}{\|w\|}\right)$.

The objective function of SVC can be defined as: $\min \frac{1}{2}\|w\|^2 + C\sum \xi_i$ subject to $y_i \cdot (w^T \cdot x_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$. The goal is to minimize the objective function.
In terms of the VQC (Variational Quantum Classifier) the methodology starts off with the conversion of the bits to qubits using feature mapping through ZZFeatureMap. In this case the parameter used was the number of feature which entailed to using four qubits corresponding to four features.

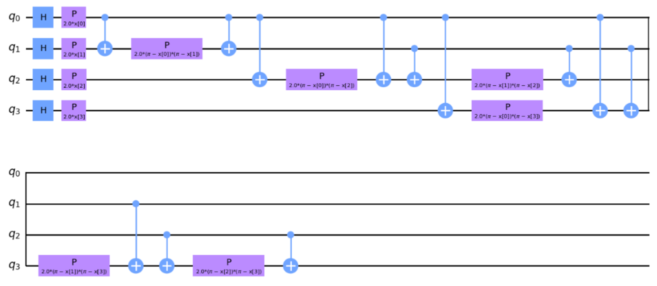


Fig. 1. Feature Mapping

The next step of VQC was to create the quantum circuit. For the purpose of this work we used a parameterized quantum circuit called ansatz. Ansatz is one of the many quantum circuits that have adjustable parameters which are optimized during training leading to an optimal solution to the classification problem. For the Ansatz we use a rep of 3 so it has 3 repetitions of the layers to parameterize.

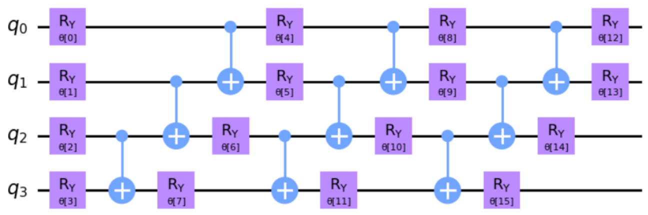


Fig. 2. Ansatz circuit

In this case we firstly RealAmplitudes and then EfficientSU2 class of Qiskit circuit library were used to construct the Ansatz. RealAmplitudes was used on all 4 features whereas EfficientSU2 was used on the reduced feature so see assess the changes in the scores.

After the results are in we make a comparative analysis of the SVC and VQC based on test score and the train score and make an assessment of the viability of VQC in real-world problem solving as a Quantum Machine learning algorithm.

B. Quanvolutional Neural Network

We now shift our focus towards Quanvolutional Neural Network, which is the quantum counterpart of Convolutional Neural Network.

1) Data Acquisition: For the simplicity of implementation and experimentation we created random images of size 2x6 using pixels ranging from 6 colors.

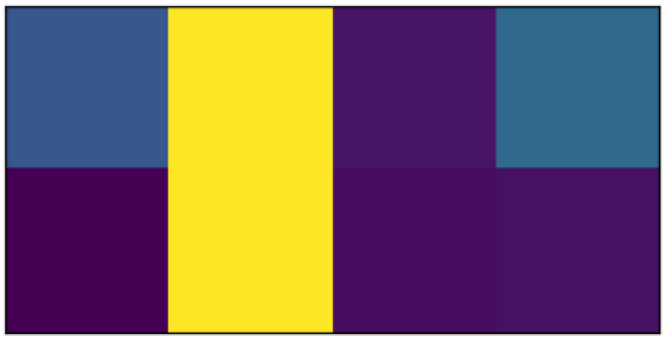


Fig. 3. Example data

2) Proposed method: To perform the Convolution operation we need a Convolution layer and a Pooling layer. For both of these layers we had to make two unitary gates that performs the convolution and pooling operation as defined by [8].

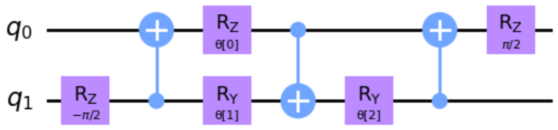


Fig. 4. Convolution Unitary Circuit

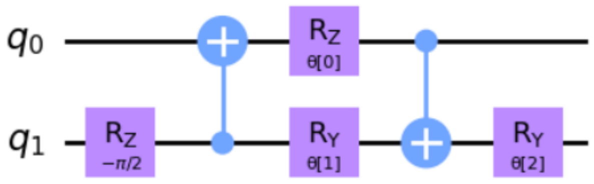


Fig. 5. Pooling Unitary Circuit

We then made the Convolution layer and the Pooling layer using these unitary circuits. In case of the Convolution layer, the convolution unitary circuits are applied to all even pairs of qubits and all odd pairs of qubits in a circular coupling manner. The first and last qubits are also coupled using a convolution unitary circuit. In case of the Pooling layer, unlike Classical CNN, it's impossible to reduce the number of qubits. The pooling unitary circuits combine two qubits' results into one and ultimately disregards the other qubit. This is how the Pooling layer reduces the number of effective qubits to reduce the dimension of an N qubit quantum circuit to N/2 qubit quantum circuits.

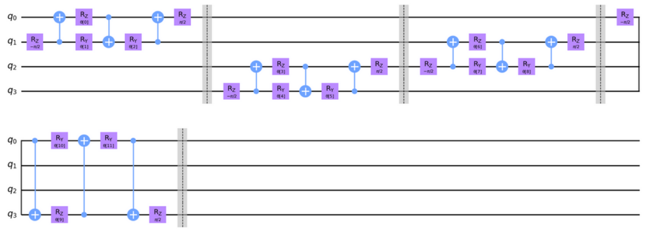


Fig. 6. Convolution Layer

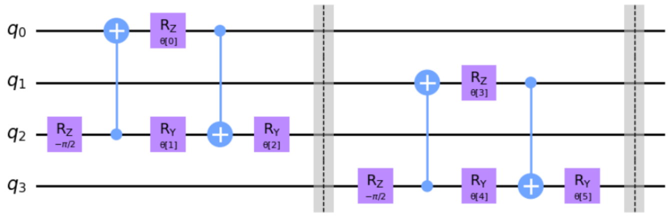


Fig. 7. Pooling Layer

The final QCNN model starts with a ZFeatureMap to encode the classical data to quantum states. Then several alternating convolution and pooling layers are placed.

C. CNN and QNN

For the final part of the analysis, we have also chosen a classification problem, the problem in this case being image classification. To tackle this, we have used a Classical Convolutional Neural Network, which we will call CNN, and a Quantum Neural Network, which we will call QNN.

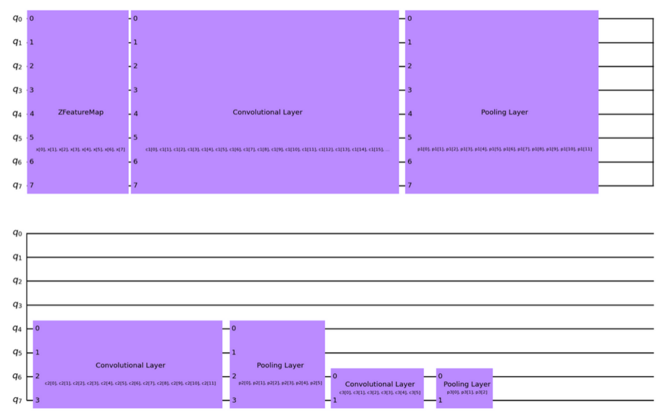
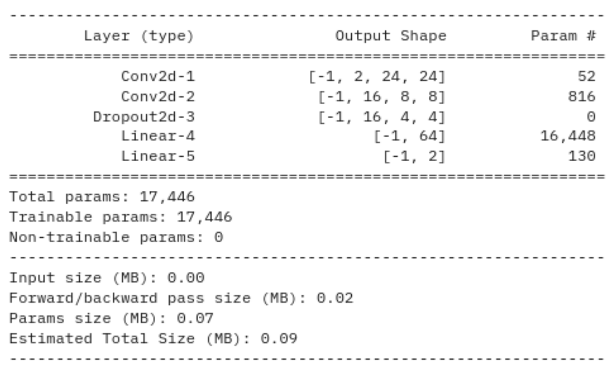


Fig. 8. QCNN Model



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 2, 24, 24]              52
            Conv2d-2            [-1, 16, 8, 8]              816
         Dropout2d-3            [-1, 16, 4, 4]                0
            Linear-4                   [-1, 64]          16,448
            Linear-5                    [-1, 2]             130
================================================================
Total params: 17,446
Trainable params: 17,446
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.07
Estimated Total Size (MB): 0.09
----------------------------------------------------------------
```

Fig. 9. CNN

However, in this case, we have used a unique implementation of a QNN which is called a Hybrid Neural Network.

1) Data Acquisition: The dataset used for both the CNN and the QNN was the MNIST dataset, a popular image classification dataset which consists of a collection of 28×28 grayscale images of handwritten digits ranging from 0 to 9 [9]. The dataset contains a total of 70,000 images, with 60,000 images split for training and 10,0000 images split for testing.
In using this dataset, the goal of our classification algorithms would be to correctly identify the digit from a picture given as input. This barometer would be used as the main factor in calculating the accuracy score of the models. To load the MNIST dataset, we used a pre-defined torchvision function. Torchvision is a package that mainly consists of popular datasets and architectures.

2) Preprocessing: Preprocessing was a necessity for both our models. To reduce the number of computations necessary, we decided to filter out the labels, resulting in the training and testing sets of the CNN and QNN having the labels 0 and 1. For training, we decided to use 20,000 samples chosen randomly from the dataset. We decided on 20,000 rather than all of the data in the sample because we wanted to save computation time, and we believed that 20,000 samples would give us a good estimate with our models.

3) Proposed method: As stated before, we wanted to showcase classification using CNN and QNN. For the CNN, we created the model with two convolution layers, one dropout layer to prevent overfitting and two fully connected layers.

For our QNN, things were a bit different. First, we had to set up the Quantum Circuit using a combination of a ZZFeatureMap and an ansatz. The ZZFeatureMap was used for encoding the bits in classical data to qubits in quantum data. The parameter used was the number of features, which in this case was 2 (0 and 1). After the features were converted into 2 qubits, they were sent into a parameterized circuit called an ansatz, which used the RealAmplitudes class of the Qiskit library [4]. The ansatz helped to define a set of operations that could be carried out on the 2 qubits. Through this, it was possible to get the required quantum states which were basically a superposition of the quantum bases.

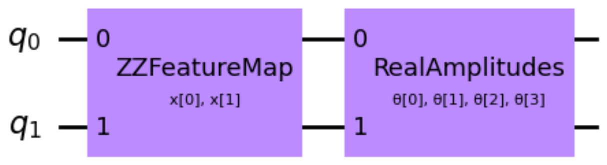


Fig. 10. Quantum Circuit

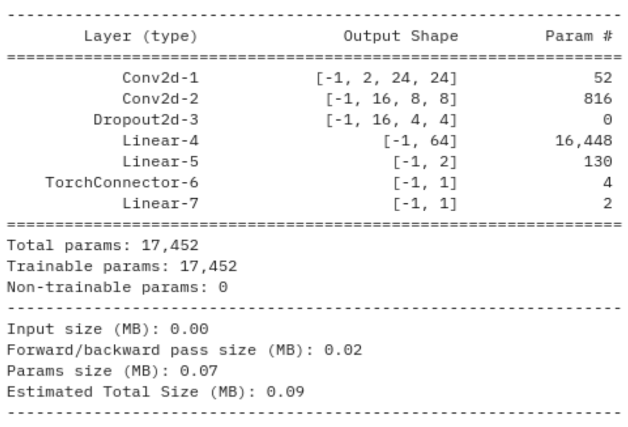
For the implementation of the QNN, we created a Hybrid Neural Network. This means that we used our quantum circuit as a layer in the original convolutional neural network. So, for this model, we had the 2 convolution layers, a dropout layer, 3 fully connected layers and one torchconnector layer. The torchconnector layer was the layer responsible for connecting to our quantum circuit, allowing quantum computations to take place.
Using the results from our experiments, we made a comparative analysis between the two models based on Average Loss and Test Accuracy.

IV. Experimental Setup

Lack of access to quantum computers led us to use simulations on classical computers. We leveraged both cloud and offline solutions for the simulations.

A. Cloud

Our cloud setup involved experimenting in Google Colab and IBM Quantum Lab. For the majority of the experiment in



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 2, 24, 24]              52
            Conv2d-2             [-1, 16, 8, 8]             816
         Dropout2d-3             [-1, 16, 4, 4]               0
            Linear-4                   [-1, 64]          16,448
            Linear-5                    [-1, 2]             130
    TorchConnector-6                    [-1, 1]               4
            Linear-7                    [-1, 1]               2
================================================================
Total params: 17,452
Trainable params: 17,452
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.07
Estimated Total Size (MB): 0.09
----------------------------------------------------------------
```

Fig. 11. QNN

Colab, we used a runtime with GPU (Tesla K80) support. Unfortunately we couldn't leverage the QPUs available through IBM Quantum Lab due to API restrictions.

B. Offline

We created a nested environment using mini-conda and then python to prepare our experimental setup. The mini-conda environment acted as a wrapper for all the core libraries and the python environment held all the Quantum support libraries, e.g. Qiskit, PennyLane etc. The runtime had CUDA 11.8 support backed by a GTX 1070.

V. Result analysis

insert introductory word here

A. SVC and VQC

After the training of the SVC and VQC model the result that were observed mostly favored the classical model over the quantum model. However, the quantum model did show some promise. A summary of the results is shown as follows:

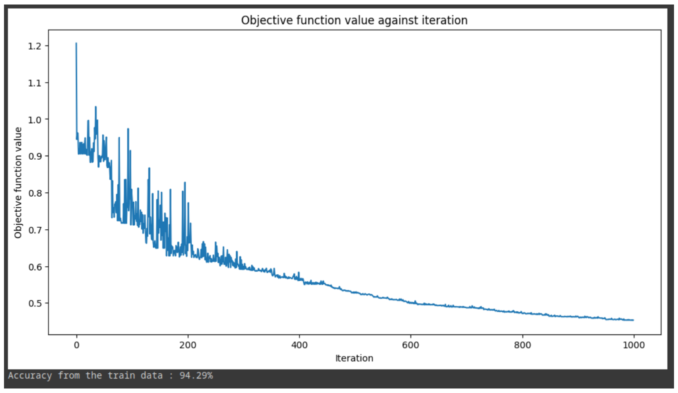
Model	Test Accuracy	Train Accuracy
SVC, 4 features	0.99	0.97
VQC, 4 features, RealAmplitude	0.85	0.87
SVC, 2 features	0.97	0.90
VQC, 2 features, EfficientSU2	0.79	0.77

Considering the results and the experimentation done with the VQC model the following findings can be assessed:

- Classical model performed better than Quantum model as of now
- Quantum model did well when we used all 4 features but the performance decreased when we reduced the feature. So if we have the hardware support and resources we should run the model on all the features available otherwise there may be compromise between dataset size, training time, and score.
- Sometimes using different ansatz change can lead to better results
- Observing the graphs we can see that sometimes if the graphs don't plateau we can increase the iterations to improve the model's score slightly

In terms of the comparative analysis of SVC and VQC it can be deduced that Quantum Machine learning is viable in terms of solving the real world problems and even might be the future of machine learning.

B. QCNN



```
Accuracy from the train data : 94.29%
```

Fig. 12. QCNN Object Function Value against iteration

Model	Test Accuracy	Train Accuracy
QCNN, 200 epochs	0.47	0.91
QCNN, 1000 epochs	0.60	0.94

The following observations can be made through our experiments:

- The QCNN model takes a very long time to reach a stable convergence as seen by Fig 12. This is likely because the data we used to train it didn't have high level features for the model to learn.
- Increasing number of training epochs have shown a significant increase in test accuracy and only a slight increase in training accuracy. This is likely because of the low number of features the model trained on.

Our experiments were conducted on a simulated environment. An actual QC will be able to converge the QCNN model more efficiently. But the data needs to have a high number of features to actually make use of the advantages a QC offers.

C. CNN and QNN

These were the results of our experiments. We carried out simulations for different numbers of samples and plotted down the average accuracy and average loss values accordingly.

The following things were observed after looking at the data:

- Firstly, both models used different Loss Functions, hence they had disparity in the values. The CNN used the Cross

Model	Average Loss	Average Accuracy
CNN, 100 samples	0.0000	100.0000
QNN, 100 samples	-5.2261	83.5000
CNN, 1000 samples	0.0148	99.7980
QNN, 1000 samples	-31.2987	85.4040
CNN, 10000 samples	0.0139	99.8109
QNN, 10000 samples	-40.6285	84.3972
CNN, 20000 samples	0.0139	99.8109
QNN, 20000 samples	-45.3290	84.3972

Entropy Loss function and QNN used the Negative Log Likelihood Loss function. For CNN, the average loss was 0 at 100 samples because there were too few samples and so the model could classify the data easily. As the number of samples increased, we could see that the average loss values for both models decreased, indicating the fact that the models were becoming better at predicting outputs.

- The accuracy for CNN remained relatively constant as the number of samples increased. On the other hand, the accuracy increased for QNN as the number of samples increased up to a certain point after which, the accuracy started to decrease. This showed that the QNN became worse at generalizing the data at higher numbers of samples.
- We also had to take into account that CNN consistently performed at a higher accuracy compared to QNN. This may be due to the fact the QNN model is a hybrid model and we were implementing a quantum circuit in a traditional neural network. If this was a quantum model built from scratch, maybe the QNN could have given better results.
- Overall, we came to the conclusion that there would be good scope for improvement in our our implementation of a QNN in the future.

VI. Conclusion and Future Work

QCs and QMLs are still very young. Regardless of the lack of maturity compared to their classical counterparts, QML algorithms are showing a lot of promise. Specially with very fast training times on actual Quantum Computers. The major hurdle QCs and QMLs need to overcome is the hypersensitivity to noise on a hardware level. Industry leaders on Quantum Computing i.e. IBM, Microsoft etc. are already working on solutions to mitigate noise and make QCs viable to solve real world problems. We can safely say that with more research and maturity, QCs and QMLs have the capability to solve problems that supercomputers and CML algorithms have a hard time with. But as of now, QMLs are not suitable for tasks that CML algorithms efficiently solve.

VII. Repository Link

https://github.com/T-A-L-O-S/CvQML

References

[1] A. J. da Silva, T. B. Ludermir, and W. R. de Oliveira, "Quantum perceptron over a field and neural network architecture selection in a quantum computer," *Neural Networks*, vol. 76, pp. 55–64, 2016.

[2] P. Rebentrost, M. Mohseni, and S. Lloyd, "Quantum support vector machine for big data classification," *Physical review letters*, vol. 113, no. 13, p. 130503, 2014.

[3] P. Tiwari and M. Melucci, "Towards a quantum-inspired binary classifier," *IEEE Access*, vol. 7, pp. 42 354–42 372, 2019.

[4] A. Zeguendry, Z. Jarir, and M. Quafafou, "Quantum machine learning: A review and case studies," *Entropy*, vol. 25, no. 2, p. 287, 2023.

[5] C. Ding, T.-Y. Bao, and H.-L. Huang, "Quantum-inspired support vector machine," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 7210–7222, 2021.

[6] S. R. Gunn *et al.*, "Support vector machines for classification and regression," *ISIS technical report*, vol. 14, no. 1, pp. 5–16, 1998.

[7] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[8] F. Vatan and C. Williams, "Optimal quantum circuits for general two-qubit gates," *Physical Review A*, vol. 69, no. 3, p. 032315, 2004.

[9] S. S. Kadam, A. C. Adamuthe, and A. B. Patil, "Cnn model for image classification on mnist and fashion-mnist dataset," *Journal of scientific research*, vol. 64, no. 2, pp. 374–384, 2020.