Quantum-Classical Hybrid Machine Learning for Image Classification

(ICCAD Special Session Paper)

Mahabubul Alam[1], Satwik Kundu[1], Rasit Onur Topaloglu[2], Swaroop Ghosh[1]
[1]*School of Electrical Engineering and Computer Science, Penn State University, University Park*
[2]*IBM Corporation*
mxa890@psu.edu, satwik@psu.edu, rasit@us.ibm.com, szg212@psu.edu

2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 978-1-6654-4507-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCAD51958.2021.9643516

***Abstract*—Image classification is a major application domain for conventional deep learning (DL). Quantum machine learning (QML) has the potential to revolutionize image classification. In any typical DL-based image classification, we use convolutional neural network (CNN) to extract features from the image and multi-layer perceptron network (MLP) to create the actual decision boundaries. QML models can be useful in both of these tasks. On one hand, convolution with parameterized quantum circuits (Quanvolution) can extract rich features from the images. On the other hand, quantum neural network (QNN) models can create complex decision boundaries. Therefore, Quanvolution and QNN can be used to create an end-to-end QML model for image classification. Alternatively, we can extract image features separately using classical dimension reduction techniques such as, Principal Components Analysis (PCA) or Convolutional Autoencoder (CAE) and use the extracted features to train a QNN. We review two proposals on quantum-classical hybrid ML models for image classification namely, Quanvolutional Neural Network and dimension reduction using a classical algorithm followed by QNN. Particularly, we make a case for trainable filters in Quanvolution and CAE-based feature extraction for image datasets (instead of dimension reduction using linear transformations such as, PCA). We discuss various design choices, potential opportunities, and drawbacks of these models. We also release a Python-based framework to create and explore these hybrid models with a variety of design choices.**

I. Introduction

Quantum computing is a new computing paradigm with tremendous future potential. Even though the technology is still in a nascent stage, the community is seeking computational advantage from quantum computers (i.e., quantum supremacy) for practical applications. Recently, Google claimed quantum supremacy can be achieved even with a 53-qubit quantum processor by completing a specific computation in 200 seconds that might take 10K years [1] (later rectified to 2.5 days [2]) on a state-of-the-art supercomputer.

The near-term quantum devices have limited number of qubits. Moreover, they suffer from various types of noises (decoherence, gate errors, measurement errors, crosstalk, etc.). Due to these limitations, these machines are not yet perfectly suitable to execute quantum algorithms that rely on high orders of error correction (e.g., Shor's factorization, Grover's search). Quantum machine learning (QML) promises to achieve quantum advantage with near-term machines because it is based on a variational principle (similar to other near-term algorithms such as, Quantum Approximate Optimization Algorithm or QAOA [3], Variational Quantum Eigensolver or VQE [4] and so on) that does not necessitate error correction [5].

Image classification is one of the most useful ML task having wide applications in autonomous driving [7], [8], medical diagnostics [9], [10], biometric security [11], [12], to name a few. An image classification ML pipeline generally consists of two stages: (i) feature extraction, and (ii) classification based on the extracted features. Before the rise of convolutional neural networks (CNN), various statistical techniques dominated feature extraction from images (e.g., SIFT, SURF, FAST, etc.) commonly referred to as feature engineering [13]. Later, these extracted features are used as inputs to a classifier (e.g., KNN, SVM, Decision Tree, Naive Bayes, MLP, etc.) [14]. CNN can extract the features and learn the classification decision boundaries simultaneously, and thus, eliminates the tedious step of feature engineering. As a result, CNN has become the ML algorithm of choice for image classification in recent years. It has also achieved human-level accuracy in many image recognition tasks [9], [10], [15].

Several QML models have been proposed for image classification to exploit quantum computers in practical use cases [6], [16]–[22]. In [6], the authors proposed Quanvolutional Neural Networks where parametric quantum circuits are used as filters/kernels to extract features from images. These quantum filters take image segments as inputs and produce output feature maps by transforming the data in the quantum space. The output features are used as inputs to an MLP network. In [17], the authors extended the classical transfer learning approach to the quantum domain. Here, the trained convolutional layers in a classical deep neural network are used to extract image features. Later, a Quantum Neural Network (QNN) is trained separately to learn the classification decision boundaries from these features. Several works have used classical dimension reduction techniques (e.g., Principal Component Analysis or PCA) to extract image features and later, used them as inputs to a QNN [23], [24]. In [16], the authors propose Quantum Convolutional Neural Network (QCNN) which is motivated by CNN. Here, convolutions are multi-qubit operations performed on neighboring pairs of qubits. These convolutions are followed by pooling layers, which are implemented by measuring a subset of the qubits, and using the results to control subsequent operations. The network ends with pairwise operations on the remaining qubits before measurement.

In this paper, we review two promising hybrid architectures for image classification, (i) Quanvolutional Neural Network (Quanvolution + MLP), and (ii) classical dimension reduction + QNN. We discuss their design choices, characteristics, enhancements, and potential drawbacks. Particularly, we advocate for trainable quantum filters in Quanvolution, and classical Convolutional Autoencoder (CAE) for image feature extraction in quantum-classical hybrid image classification models.

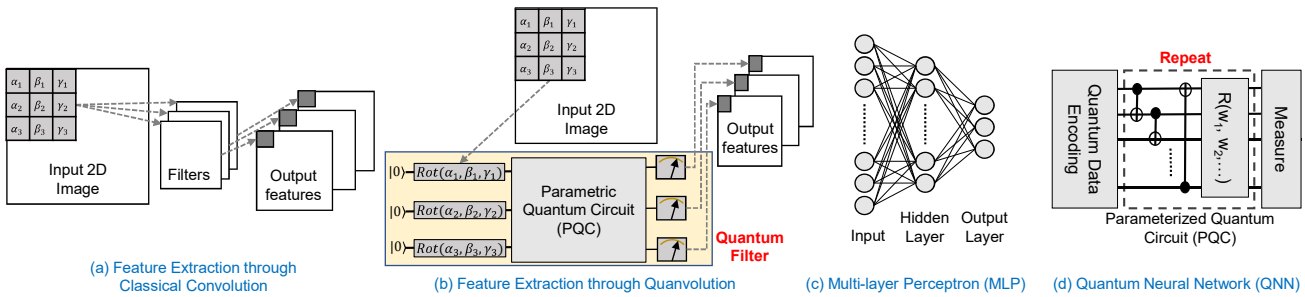

Fig. 1. (a) shows a classical convolution operation. (b) shows a toy Quanvolution operation proposed in [6]. In Quanvolution, a quantum circuit (also referred to as quantum filter) encodes an image segment as a quantum state, and produces output features corresponding to that segment through state transformation using a parameterized circuit and subsequent measurement operations. (c) shows the network diagram of a toy Multi-layer Perceptron (MLP) network. A conventional Quantum Neural Network (QNN) is shown in (d). It consists of a data encoding circuit, a parameterized circuit, and measurement operations.

A QNN/quantum filter has a myriad of design choices in terms of encoding methods, parametric circuits, and measurement operations. However, in this work, we only use two configurations for demonstration. The accompanying Python-based framework supports a wide variety of QNN/quantum filter design choices (6 encoding circuits, 19 parametric circuits, and 6 measurement circuits). Interested readers can utilize/extend this framework to explore the design space.

In the remaining paper, we cover basics on quantum computing and QNN in Section II, discuss the hybrid architectures in Section III, present relevant results in Section IV, and draw the conclusions in Section V.

II. Preliminaries

Qubits, Quantum Gates, State Vector, & Measurements: Qubit is analogous to classical bits. However, unlike a classical bit, a qubit can be in a superposition state i.e., a combination of $|0\rangle$ and $|1\rangle$ at the same time. A variety of qubit technologies exists, e.g., superconducting qubits, trapped-ions, neutral atoms, silicon spin qubits, to name a few [25]. Quantum gates such as, single qubit (e.g., Pauli-X (σ_x) gate) or multiple qubit (e.g., 2-qubit CNOT gate) gates modulate the state of qubits to perform computations. These gates can either perform a fixed or tunable computation e.g., an X gate flips a qubit state while the RY(θ) gate rotates the qubit along the Y-axis by θ. A two-qubit gate changes the state of one qubit (*target qubit*) based on the current state of the other qubit (*control qubit*). A quantum circuit can contain many gate operations. Qubits are measured in a desired basis to retrieve the final state of a quantum program. In physical quantum computers, measurements are generally restricted to a computational basis, e.g., Z-basis in IBM quantum computers.

Expectation Value of an Operator: Expectation value is the average of the eigenvalues, weighted by the probabilities that the state is measured to be in the corresponding eigenstate. Mathematically, expectation value of an operator (σ) is defined as $\langle\psi|\sigma|\psi\rangle$ where $|\psi\rangle$ is the qubit state vector. It varies between the minimum and maximum eigenvalues of the operator. For example, the Pauli-Z (σ_z) operator has two eigenvalues: +1 and -1. Therefore, the Pauli-Z expectation value of a qubit will vary in the range of [-1, 1] depending on the qubit state.

Quantum Neural Network: QNN involves parameter optimization of a PQC to obtain a desired input-output relationship. QNN generally consists of three segments: (i) a classical to quantum data encoding (or embedding) circuit, (ii) a parameterized circuit, and (iii) measurement operations. A variety of encoding methods are available in the literature [26]. For continuous variables, the most widely used encoding scheme is angle encoding where a continuous variable input classical feature is encoded as a rotation of a qubit along the desired axis (X/Y/Z) [26]–[29]. For 'n' classical features, we require 'n' qubits. For example, RZ(f1) on a qubit in superposition (Hadamard - H gate is used to put the qubit in superposition) is used to encode a classical feature 'f1' in Fig. 2(b). We can also encode multiple continuous variables in a single qubit using sequential rotations. For example, 'f1', 'f2', 'f3', and 'f4' are encoded using consecutive RZ(f1), RX(f2), RZ(f2), and RX(f4) rotations on a single qubit in Fig. 2(c). As the states produced by a qubit rotation along any axis will repeat in 2π intervals (Fig. 2(a)), features are generally scaled within 0 to 2π (or -π to π) in a data pre-processing step.

The parametric circuit has two components: entangling operations and parameterized single-qubit rotations. The entanglement operations are a set of multi-qubit operations between all the qubits to generate correlated states [29]. The following parametric single-qubit operations are used to search through the solution space. This combination of entangling and single-qubit rotation operations is referred to as a parametric layer in QNN. A widely used parametric layer architecture is shown in Fig. 2(d) [27], [30]. Here, CRZ(θ) gates between neighboring qubits create the entanglement, which is followed by rotations along Y-axis using RY(θ). Normally, these layers are repeated multiple times to extend the search space [27], [28].

QNN Cost Functions: Qubits in a QNN circuit are measured in the computational basis to retrieve the output state. A cost function is derived from the measurements to train the network [27], [28], [31]. For example, in a binary classification problem, the authors measured all the qubits in the QNN model in Pauli-Z basis and associated class 0 with the probability of obtaining even parity, and class 1 with odd parity [27]. Then, the model is trained using binary cross-entropy loss. In [32], the authors used the Pauli-Z expectation value of a single qubit (-1 associated with class 1 and +1 associated with class 0) for a binary classifier and trained it using mean squared error (MSE) loss.In [23], the authors fed the outputs of the

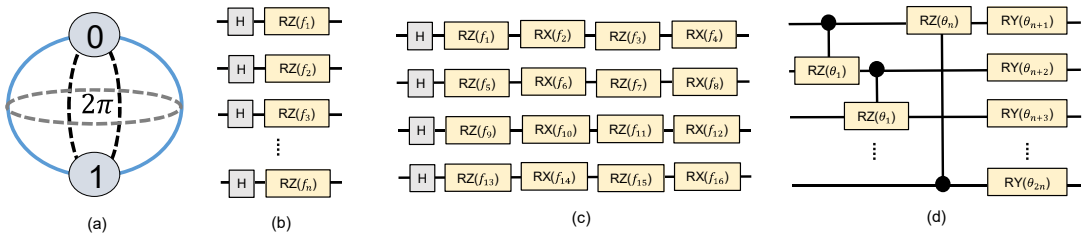

Fig. 2. (a) Bloch sphere representation of a qubit. At any given step, a qubit can be rotated along the X, Y, or Z axis by applying a gate. The states will repeat in 2π intervals. (b) Angle encoding 1:1 (1 continuous variable encoded in a single qubit state using RZ rotation, n qubits are required to encode n continuous variables as an n-qubit state). (c) Angle encoding 4:1 (4 continuous variables encoded in a single qubit state using alternating RZ and RX rotations, 4 qubits encode 16 continuous variables as a 4-qubit state). (d) Parametric layer used in this work. Parametric CRZ gates entangle the qubits; this is followed by single qubit RY rotations. Each n-qubit parametric layer has 2n circuit parameters.

QNN to a classical neural network and trained it using the binary cross-entropy loss function.

Training QNN: QNN's can be trained using any gradient-based optimization algorithm such as, Adam [33] or Adagrad [34]. To apply these methods, we need to compute the gradients [35], [36] of the QNN outputs with respect to the circuit parameters. The parameter-shift rule is a known method to compute the gradients [35], [36]. Conceptually, parameter-shift rule is very similar to the age-old finite difference method which uses two evaluations of a target function at close proximity to compute the gradients with respect to a parameter. Unlike finite difference, the two data points can be far from each other in parameter-shift rule. As a result, it shows greater resilience to shot noise and measurement errors compared to finite difference [36]. Alternatively, one can also use gradient free optimizer such as Nelder-Mead to train a QNN [37]. However, a gradient-free optimizer may perform poorly when the network has lots of parameters.

III. Hybrid Architectures for Image Classification

A. *Quanvolution + MLP*

Quanvolution is simply an extension of classical convolution where small local regions of an image are sequentially processed with the same kernel/filter. A kernel is a small 2D matrix. The dot product between the kernel and the image segment is used to generate an output feature. For 3D RGB images, separate kernels are applied across the channels (2D planes), and they are collectively referred to as filters. For 2D images, filters and kernels are synonymous. The results obtained for each region are usually associated to different channels of a single output pixel. The union of all the output pixels produces a new image-like object, which can be further processed by additional layers. A toy convolutional layer operation is shown in Fig. 1(a). In Quanvolution, quantum circuits mimic the behavior of classical CNN filters.

Quantum Filters: The quantum filters encode image segments as input state of a quantum circuit. The state is transformed using a parameterized quantum circuit and subsequent measurement operations produce output features corresponding to that segment. Fig. 1(b) shows a toy Quanvolution layer. Here, a 3-qubit quantum circuit is used as a filter. It encodes 3x3 image segments as a 3-qubit quantum state using 3 Rot(α, β, γ) rotations (Rot is an arbitrary quantum gate that takes three rotation parameters). Similar to CNN, these quantum filters are moved across the 2D plane in finite steps (strides) to generate the complete output feature map of the image.

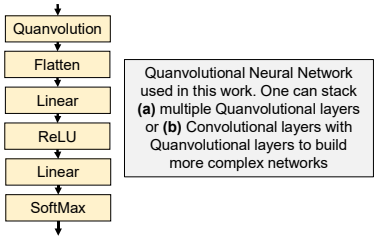

Fig. 3. A quantum-classical hybrid neural network based on the Quanvolutional Neural Network architecture [6] with a single Quanvolutional layer.

The qubit size of a filter depends on the chosen encoding method and the kernel size. For example, if we use the 1 variable/qubit encoding method (Fig. 2(b)), the resulting quantum filter will be a 16-qubit circuit for kernel size of 4x4. It will reduce to a 4-qubit circuit if we choose the 4 variables/qubit encoding method (Fig. 2(c)). Concepts like data re-uploading [38] can be used to encode an arbitrary number of variables in an arbitrary number of qubits. Choice of encoding methods will most likely be dictated by the availability of quantum resources [21]. In this work, we use the 4 variables/qubit encoding method for 4x4 kernels. We use the circuit architecture shown in Fig. 2(d) (Circuit 13 of [30]) as our preferred PQC in the quantum filters. We also use the Pauli-Z expectation values of the qubits as output features (an n-qubit filter generates n-features/image segment). Increasing the number of filters increases the number of extracted features for the downstream classifier. Similar to a CNN with high number of stages, a large number of quantum filters improve performance (lower cost/higher accuracy/faster training) [6].

Filter Trainability: In the original work in [6], the quantum filters did not have any trainable parameters. However, in CNN, filters have trainable weights, and they are learned during the training. Similarly, quantum filters can too have trainable parameters. For example, we can either initialize the PQC parameters ($\theta_1 - \theta_{2n}$ in Fig. 2(d)) randomly and keep it constant throughout the training, or we can update them during the training alongside other parameters in the network.

Trainable filters will result in many-fold increase in quantum circuit execution during the training. For example, if a

quantum filter has p trainable parameters, it will add 2xp more quantum circuit executions for each image segment to compute the required gradients using parameter-shift rule [36]. Note that, feature generation using non-trainable quantum filters is equivalent to random transformation of the image segments. A classical random function can replace such filters. In fact, the work in [6] showed that the performance of classical random transformations of the image segments matches the performance of random transformations with non-trainable quantum filters. However, the trainable quantum circuits are hard to simulate classically [39]. If they exhibit significant performance benefits over their non-trainable counterparts, it will be worthwhile for the research community to explore them for possible quantum advantage [40].

Network Design: Similar to CNN, a Quanvolutional layer can have many filters and multiple Quanvolutional layers can be stacked upon each other to develop a deep Quanvolutional Neural Network [6], [21]. The outputs from the final Quanvolutional layer can be fed to a an MLP (or a QNN). One can also apply classical non-linear activation functions (for additional non-linearity) and maxpooling (downsampling) at the output of a Quanvolutional layer. One can create separate filters by initializing the same PQC with different random seeds. Alternatively, we can use different PQC architectures, encoding methods, and measurement operations altogether to create different filters. One can also stack classical convolutional layers with Quanvolutional layers. Fig. 3 shows the network diagram used in this work with a single Quanvolutional layer followed by two fully connected classical layers.

Number of Circuit Executions: The number of quantum circuit executions per sample during training/inference depends on the kernel size, image size, and the stride (the amount of movement of the kernel in terms of pixels). For a 28x28 image and 4x4 kernel size, we need to execute a total of 7x7 quantum circuits when stride = 4 (a single non-trainable quantum filter). If this filter has 10 trainable parameters, the total number of circuit execution becomes 7x7 + 2x10x7x7 where the later 2x10x7x7 circuit executions are necessary to compute the gradients using parameter-shift rule (2 extra circuit executions for each parameters [36]). If we take 50 samples per batch, we will need 50x(7x7 + 2x10x7x7) filter execution for each batches during training. This is in fact a prohibitively large number for a single batch and a filter. However, all these circuits are independent of each other. Hence, one can argue that all these computations can be done simultaneously if one has access to multiple quantum computing resources.

B. Classical Dimension Reduction + QNN

Another popular hybrid QML model targeted for smaller quantum devices uses classical algorithm (e.g., Principal Component Analysis or PCA, Linear Discriminant Analysis or LDA, etc.) to reduce data dimension to a level that is tractable for a small QNN model [23], [24], [41]. Although PCA/LDA work quite well to extract most salient features from small tabular data, they are not suitable to extract features from large images. Autoencoders (AE), particularly Convolutional

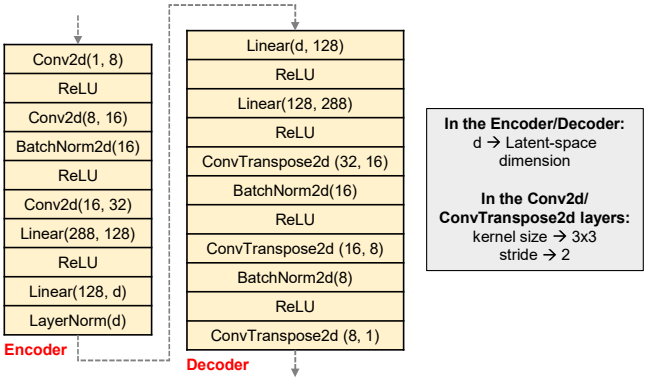

Fig. 4. Convolutional Autoencoder (CAE) architecture used in this work to extract image features (for both the MNIST and Fashion-MNIST datasets).

Autoencoders (CAE) are much more powerful tools for image feature extraction/dimension reduction [42], [43]. PCA is a linear transformation of the data whereas AE/CAE can model more complex non-linear relationships in the data using non-linear activation functions and regularization [43].

AE/CAE: AE's are a specific type of feedforward neural networks. They compress the input into a lower-dimensional code using an encoder network and then reconstruct the output from this representation through a decoder network. The code is a compact representation of the input, also called the latent-space representation. The distance between the input and reconstructed output (e.g., MSE loss) is used as the feedback signal to train the network. Both the encoder and decoder in a simple AE consist of several fully connected layers. CAE provides a better architecture than AE to extract the textural features of images. In CAE, the encoder block starts with one or more successive convolutional layers. The decoder block ends with convolutional transpose/deconvolutional layers. In the middle there is a fully connected AE whose innermost layer is composed of a small number of neurons. Once trained, the encoder block can be used as a standalone entity to extract lower dimensional representation of the input data.

Fig. 4 shows the CAE network architecture used in this work (for both MNIST and Fashion-MNIST datasets). The final ConvTranspose2d layer uses Sigmoid activation where 'd' is the dimension of the latent-space.

Network Design: The hybrid network (Fig. 5) consists of two separate networks - a CAE and a QNN. The CAE is trained with the original image dataset to learn a lower dimensional representation of the data. The trained encoder network is used to extract image features. A conventional QNN is trained with these extracted features and image labels to perform final classification. When both of these networks are trained, the encoder block and the QNN block is used together to classify data samples. We refer to this architecture as CAE+QNN.

QNN Design-Space: As mentioned earlier, numerous choices exist for the encoding circuits, PQC, and measurement circuits to build a QNN model. The accompanying Python framework to this work supports a wide variety of these choices which will impact the learnability of the QNN [30]. However, in this work we only use the single feature/qubit encoding

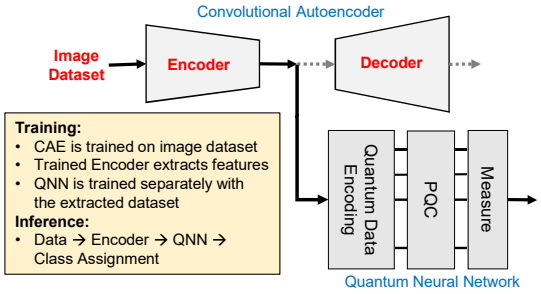

Fig. 5. The CAE + QNN network architecture. The trained CAE Encoder block creates a lower-dimensional representation of the image for the QNN.

method (Fig. 2(b)), the PQC layer of Fig. 2(d), and Z-basis measurements of the qubits in the QNN. We also restrict the number of parametric layers to 3. Following [23], we feed the QNN outputs to a fully-connected layer. The number of output neurons is equal to the number of classes in the dataset.

CAE+QNN Vs. Transfer Learning: Although, the CAE+QNN network has some similarities with transfer learning [17], there are some noteworthy differences as well. Both of these approaches extract image features using a classical network. In transfer learning, the convolutional layers of a classical CNN network, trained with a different dataset (e.g., AlexNet trained to classify the ImageNet dataset), is used to extract features for a target dataset. In contrast, the CAE in CAE+QNN network is trained separately to extract features from the target dataset. Therefore, the CAE extracted features may capture more variance in the target dataset compared to transfer learning, and thus, it may provide better performance (lower training cost/higher accuracy). The features extracted through transfer learning can be more generic [17], and it eliminates the need to train the classical network separately (as it is already trained).

IV. Evaluation

In this Section, we compare the performance differences between (a) Quanvolutional Neural Networks with trainable filters, and non-trainable filters, and (b) CAE + QNN and PCA-based approach for a variety of datasets.

Datasets: We pick the MNIST and Fashion-MNIST datasets for this work which are widely used in contemporary works in QML research (each pixel value scaled within 0-1) [44], [45]. Both of these datasets have 60,000 training samples, and 10,000 test samples of 2D images (28x28 pixels) that belong to 10 different classes. For empirical evaluation of the Quanvolution approach, we have picked 1,200 samples from three different classes to create 6 smaller classification datasets - MNIST_179, MNIST_246, MNIST_358, Fashion_012, Fashion_345, and Fashion_678. MNIST_179 has ≈400 samples of digits 1, 7, and 9 each. Similarly, Fashion_012 has ≈400 samples of classes 0 (t-shirt/top), 1 (trouser/pant), and 2 (pullover shirt) each. We have reduced the dimension of the samples from 28x28 to 14x14 using maxpooling to lower the simulation time. To compare CAE and PCA, we have trained the corresponding CAE (with latent dimension of 5 and 10) and PCA models with the entire MNIST and Fashion-MNIST datasets. Later, we have created 6 smaller classification datasets as before using the trained models with various latent dimensions (5/10) and principal component (10).

Metrics: We have divided the datasets into two equal sets for training and validation (600 samples/set). We use the average loss and accuracy over the entire training and validation datasets to measure the performance of the QML models [46].

Training Setup: We use the gradient-based Adagrad, SGD, and Adam optimizers to train these models [47], [48]. We use the same set of hyper-parameters across all the runs (learning rate = 0.5 for all the quantum/hybrid models).

Trainable Vs. Non-trainable Filters in Quanvolution: We trained quanvolutional neural networks with a single quanvolutional layer (Fig. 3) for six 3-class classification problems with a trainable and a non-trainable quantum filter (stride = 4). We used 4-qubit circuits as Quanvolutional filters. We used the 4 variables/qubit encoding method shown in Fig. 2(c) to encode 4x4 pixels into a 4-qubit state. The input pixels were scaled to 0-2π (originally 0-1). The PQC architecture in Fig. 2(d) was used with 3 parametric layers (3x2x4 parameters). In Quanvolution with trainable filters, these PQC parameters were trained alongside other network parameters using gradient descent. In the non-trainable filter, we had set the PQC parameters randomly (-π to π) at the beginning, and kept them constant throughout the training. Pauli-Z expectation values of qubits were used as the output features. The results are tabulated in Table I (performance after 10 training epochs). Each training epoch took ≈195 seconds with the trainable filter compared to ≈57 with the non-trainable filter on a single Core i7-10750H machine with 16 GB RAM.

On average, Quanvolution with trainable filter provided 15.98% lower training loss, 7.49% lower validation loss, 3.46% higher training accuracy, and 3.32% higher validation accuracy after 10 training epochs. In some cases, the Quanvolution with non-trainable filter performed at a similar level as its trainable counterpart. For example, MNIST_179 and MNIST_358 provided similar performance in both these approaches. However, in all other cases, there was a noticeable performance gap between these two approaches. We repeated the experiments 5 times with the MNIST_179 and MNIST_358 dataset with different random initialization. However, the performance remained at similar levels in both these approaches. In fact, both these models performed poorly on these two datasets compared to the others (average training loss of 0.535 against 0.266 overall with the trainable filter). The overall results indicate potential benefits of trainable filters which can be worthwhile to explore in the future.

CAE + QNN: We trained the CAE in Fig. 4 with 60000 training samples of the MNIST and Fashion-MNIST datasets with latent-dimension of 5 and 10 (optimizer: Adam, learning rate: 0.001, weight-decay: e^{-5}, epochs: 30, batch-size: 50). The extracted datasets (5/10 features) were used to train a QNN. In the QNN, we used 1 variable/qubit encoding method as shown in Fig. 2(b). We used 5 and 10 qubits for the 5-feature and 10-feature datasets, respectively. The QNN shared same PQC architectures and output measurements as the quantum

TABLE I
QUANVOLUTIONAL NEURAL NETWORK PERFORMANCE AFTER 10 EPOCHS OF TRAINING (OPTIMIZER: ADAGRAD, LEARNING RATE: 0.5)

	Quanvolution - Non-trainable Filters				Quanvolution - Trainable Filters			
Datasets	Training Set Loss	Validation Set Loss	Training Set Accuracy	Validation Set Accuracy	Training Set Loss	Validation Set Loss	Training Set Accuracy	Validation Set Accuracy
MNIST_179	0.3717	0.5537	0.8416	0.783	0.3881	0.5338	0.8500	0.7733
MNIST_246	0.3562	0.4607	0.8733	0.8333	0.2684	0.4879	0.9100	0.8583
MNIST_358	0.6577	0.8453	0.6916	0.6350	0.6825	0.9452	0.7083	0.6300
Fashion_012	0.1149	0.4213	0.9533	0.8833	0.0560	0.3813	0.9783	0.9200
Fashion_345	0.1416	0.2788	0.9433	0.8983	0.0827	0.1670	0.9666	0.9366
Fashion_678	0.2626	0.4434	0.8950	0.8450	0.1226	0.2631	0.9650	0.9216

TABLE II
CAE + QNN NETWORK PERFORMANCE AFTER 20 EPOCHS OF TRAINING OF THE QNN (OPTIMIZER: SGD, LEARNING RATE: 0.5)

	CAE + QNN: Latent Dimension = 5				CAE + QNN: Latent Dimension = 10			
Datasets	Training Set Loss	Validation Set Loss	Training Set Accuracy	Validation Set Accuracy	Training Set Loss	Validation Set Loss	Training Set Accuracy	Validation Set Accuracy
MNIST_179	0.1479	0.1676	0.9633	0.9533	0.1385	0.1120	0.9633	0.9683
MNIST_246	0.0574	0.0793	0.9900	0.9883	0.0676	0.0791	0.9833	0.9816
MNIST_358	0.3630	0.3281	0.8917	0.9183	0.2000	0.1866	0.9350	0.9383
Fashion_012	0.3620	0.2888	0.9083	0.9200	0.2154	0.1758	0.9266	0.9483
Fashion_345	0.3876	0.2084	0.8500	0.7533	0.1793	0.1541	0.9300	0.9466
Fashion_678	0.2468	0.1962	0.9233	0.9483	0.2968	0.2662	0.9033	0.9550

filters (Fig. 2(d)). We restricted the parametric layers to 3 in the 10-qubit model (3x2x10 parameters). To match the number of trainable circuit parameters, we restricted the parametric layers to 6 in the 5-qubit models (6x2x5 parameters).

The results are tabulated in Table II (performance after 20 epochs of training). All these models were trainable as evident from their loss and accuracy values. In the CAE+QNN model, the chosen number of latent-dimension (d) dictates the QNN architecture. It also affects the overall network performance. A higher value of d means more input features for the QNN model that generally translates to better training performance of the QNN. On average, the CAE + QNN model with d = 10 provided 29.85% lower training loss, 23.23% lower validation loss, 2.08% higher training accuracy, and 4.68% higher validation accuracy after 20 training epochs compared to d = 5. Therefore, a higher d (at the cost of larger QNN) may provide better performance in practical applications.

CAE + QNN Vs. PCA + QNN: As PCA uses linear transformation, the extracted image features are expected to be poor which may translate to poor training performance of the QNN. To perform this comparison, we trained a PCA model with the 60000 MNIST training samples and extracted 4000 samples (≈400/class) as before with 10 principal components as the feature variables. We also extracted another 4000 samples from trained CAE with d = 10. Later, we trained 10-qubit QNN models (parametric layers set to 3) with these datasets for 20 epochs using the same set of training hyper-parameters (optimizer: Adagrad, learning rate: 0.5). The results are shown in Table III. As expected, the CAE+QNN approach outperformed the PCA+QNN approach by significant margin. The CAE + QNN model provided 48.47% lower training loss, 49.2% lower validation loss, 14.1% higher training accuracy, and 25.3% higher validation accuracy.

TABLE III
CAE + QNN AND PCA + QNN NETWORK PERFORMANCE AFTER 20 EPOCHS OF TRAINING ON MNIST DATASET (4000 SAMPLES, 10 CLASSES)

Approach	Training Loss	Validation Loss	Training Accuracy	Validation Accuracy
PCA(10) + QNN	0.6496	0.6724	0.7875	0.7175
CAE(10) + QNN	0.3336	0.3463	0.8965	0.8980

Python Framework Supports: Numerous choices exist for the QNN/quantum filter design using various encoding methods, parametric circuit architectures, and measurements. In this work, we only explored a limited set (Fig. 2). However, we also release a Python-based framework (created using PennyLane, TensorFlow, and PyTorch packages [47]–[49]) that supports 19 parametric circuit architectures from [30], 6 encoding techniques, and 6 measurement circuits [50]. We also make the datasets available through the repository. Interested readers can utilize this repository for further exploration of these models on any chosen dataset.

V. CONCLUSION

In this article, using empirical evidence we argue that trainable quantum filters in Quanvolution may provide performance benefits over the non-trainable filters, and thus, it can be worthwhile to explore for potential quantum advantage on image classification tasks. We also show that in the later architecture, dimension reduction with convolutional autoencoder (CAE) can be more useful compared to the linear transformation-based approaches such as, PCA for image datasets.

Acknowledgements: The work is supported in parts by NSF (CNS-1722557, CCF-1718474, OIA-2040667, DGE-1723687 and DGE-1821766) and seed grants from Penn State ICDS and Huck Institute of the Life Sciences.

REFERENCES

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[2] E. Pednault, J. Gunnels, D. Maslov, and J. Gambetta, "On "quantum supremacy"," *IBM Research Blog*, vol. 21, 2019.

[3] E. Farhi and H. Neven, "Classification with quantum neural networks on near term processors," *arXiv preprint arXiv:1802.06002*, 2018.

[4] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.

[5] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, p. 023023, 2016.

[6] M. Henderson, S. Shakya, S. Pradhan, and T. Cook, "Quanvolutional neural networks: powering image recognition with quantum circuits," *Quantum Machine Intelligence*, vol. 2, no. 1, pp. 1–9, 2020.

[7] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE CVPR*, 2017, pp. 1907–1915.

[8] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 922–928.

[9] S. M. McKinney, M. Sieniek, V. Godbole, J. Godwin, N. Antropova, H. Ashrafian, T. Back, M. Chesus, G. S. Corrado, A. Darzi *et al.*, "International evaluation of an ai system for breast cancer screening," *Nature*, vol. 577, no. 7788, pp. 89–94, 2020.

[10] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[11] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE CVPR*, 2014, pp. 1701–1708.

[12] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE CVPR*, 2015, pp. 815–823.

[13] M. Nixon and A. Aguado, *Feature extraction and image processing for computer vision*. Academic press, 2019.

[14] J. Friedman, T. Hastie, R. Tibshirani *et al.*, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1, no. 10.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[16] I. Cong, S. Choi, and M. D. Lukin, "Quantum convolutional neural networks," *Nature Physics*, vol. 15, no. 12, pp. 1273–1278, 2019.

[17] A. Mari, T. R. Bromley, J. Izaac, M. Schuld, and N. Killoran, "Transfer learning in hybrid classical-quantum neural networks," *Quantum*, vol. 4, p. 340, 2020.

[18] Y. Li, R.-G. Zhou, R. Xu, J. Luo, and W. Hu, "A quantum deep convolutional neural network for image recognition," *Quantum Science and Technology*, vol. 5, no. 4, p. 044003, 2020.

[19] I. Kerenidis, J. Landman, and A. Prakash, "Quantum algorithms for deep convolutional neural networks," *arXiv preprint arXiv:1911.01117*, 2019.

[20] Y. Dang, N. Jiang, H. Hu, Z. Ji, and W. Zhang, "Image classification based on quantum k-nearest-neighbor algorithm," *Quantum Information Processing*, vol. 17, no. 9, pp. 1–18, 2018.

[21] M. Henderson, J. Gallina, and M. Brett, "Methods for accelerating geospatial data processing using quantum computers," *Quantum Machine Intelligence*, vol. 3, no. 1, pp. 1–9, 2021.

[22] J. Li, M. Alam, C. M. Sha, J. Wang, N. V. Dokholyan, and S. Ghosh, "Drug discovery approaches using quantum machine learning," *arXiv preprint arXiv:2104.00746*, 2021.

[23] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, and J. R. McClean, "Power of data in quantum machine learning," *Nature communications*, vol. 12, no. 1, pp. 1–9, 2021.

[24] E. Grant, M. Benedetti, S. Cao, A. Hallam, J. Lockhart, V. Stojevic, A. G. Green, and S. Severini, "Hierarchical quantum classifiers," *npj Quantum Information*, vol. 4, no. 1, pp. 1–8, 2018.

[25] M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*. American Association of Physics Teachers, 2002.

[26] M. Schuld, R. Sweke, and J. J. Meyer, "Effect of data encoding on the expressive power of variational quantum-machine-learning models," *Physical Review A*, vol. 103, no. 3, p. 032430, 2021.

[27] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, and S. Woerner, "The power of quantum neural networks," *Nature Computational Science*, vol. 1, no. 6, pp. 403–409, 2021.

[28] M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe, "Circuit-centric quantum classifiers," *Physical Review A*, 2020.

[29] S. Lloyd, M. Schuld, A. Ijaz, J. Izaac, and N. Killoran, "Quantum embeddings for machine learning," *arXiv preprint arXiv:2001.03622*, 2020.

[30] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, "Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms," *Advanced Quantum Technologies*, 2019.

[31] M. Schuld and N. Killoran, "Quantum machine learning in feature hilbert spaces," *Physical review letters*, vol. 122, no. 4, p. 040504, 2019.

[32] M. Alam, A. Ash-Saki, and S. Ghosh, "Addressing temporal variations in qubit quality metrics for parameterized quantum circuits," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2019, pp. 1–6.

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

[34] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[35] L. Banchi and G. E. Crooks, "Measuring analytic gradients of general quantum evolution with the stochastic parameter shift rule," *Quantum*, vol. 5, p. 386, 2021.

[36] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, "Evaluating analytic gradients on quantum hardware," *Physical Review A*, vol. 99, no. 3, p. 032331, 2019.

[37] W. Lavrijsen, A. Tudor, J. Müller, C. Iancu, and W. de Jong, "Classical optimizers for noisy intermediate-scale quantum devices," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2020, pp. 267–277.

[38] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, and J. I. Latorre, "Data re-uploading for a universal quantum classifier," *Quantum*, vol. 4, p. 226, 2020.

[39] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, no. 7747, pp. 209–212, 2019.

[40] P. Atchade-Adelomou and G. Alonso-Linaje, "Quantum enhanced filter: Qfilter," *arXiv preprint arXiv:2104.03418*, 2021.

[41] K. Batra, K. M. Zorn, D. H. Foil, E. Minerali, V. O. Gawriljuk, T. R. Lane, and S. Ekins, "Quantum machine learning algorithms for drug discovery applications," *Journal of Chemical Information and Modeling*, 2021.

[42] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *International conference on artificial neural networks*. Springer, 2011, pp. 52–59.

[43] M. Chen, X. Shi, Y. Zhang, D. Wu, and M. Guizani, "Deep features learning for medical image analysis with convolutional autoencoder neural network," *IEEE Transactions on Big Data*, 2017.

[44] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[45] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[46] M. Anthony and P. L. Bartlett, *Neural network learning: Theoretical foundations*. cambridge university press, 2009.

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[48] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[49] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, S. Ahmed, J. M. Arrazola, C. Blank, A. Delgado, S. Jahangiri *et al.*, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2018.

[50] https://github.com/mahabubul-alam/iccad_2021_invited_QML.