Enhancing QML Classifiers with Multiple Quantum Kernels

By Sunil Vittal

Introduction

In recent years, Quantum Machine Learning (QML) has risen in popularity, due to its potential to provide training advantages over classical models in regards to model training/optimization and data efficiency. While classical classifier models such as Support Vector Machine, Decision Trees, Logistic Regression, etc, have been well studied, corresponding generalizations to QML remain slightly inferior to the classical counterparts (e.g. https://www.sciencedirect.com/science/article/abs/pii/S1568494623003265?utm_source=chatgpt.com, and https://pubmed.ncbi.nlm.nih.gov/39202100/), leading us to wonder "on what kinds of problems can we realize a quantum advantage and how?"

One interesting application of QML is preprocessing. Quanvolutional neural networks are quantum-classical hybrid convolutional neural networks (CNNs) containing a quantum kernel preprocessing layer, effectively performing the same action as a classical convolution layer. These have been used for 2D images, and the quanvolutions segment the image, allowing for quantum feature extraction.(http://arxiv.org/abs/1904.04767). Recent work have developed on this, with Quantum U-nets (https://arxiv.org/pdf/2401.07049v1) and Quantum Feature Extractors (https://arxiv.org/pdf/2501.13165). Nonetheless, these developments focus on image processing, and fundamental classification questions involve 1D numerical data, so how can we apply these to 1D binary classification problems? In this project, I use 1D quanvolutions to create a quantum-classical hybrid classifier on the wine dataset (specifically red wine).

Methods + Model Architecture

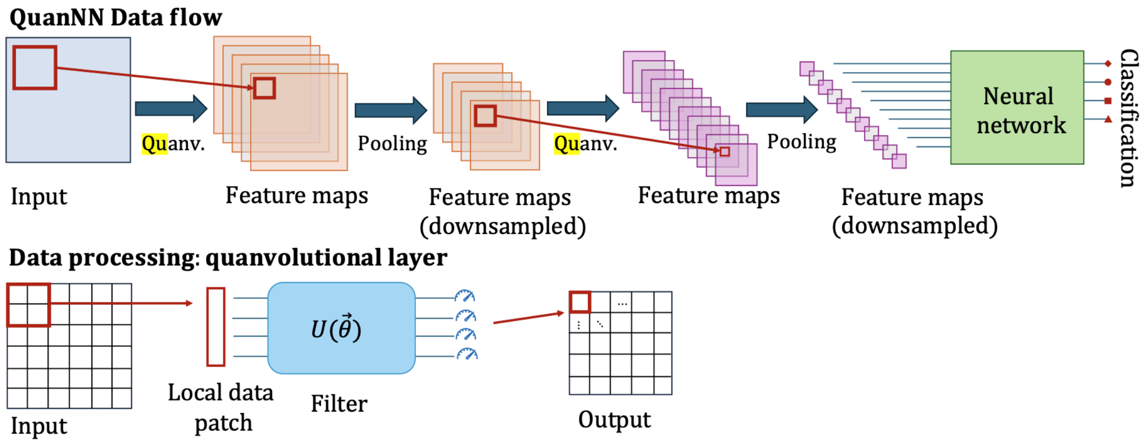
Quanvolutional layers embed small, local data patches into shallow variational quantum circuits (VQCs). In our 1D setting, each sliding-window of width ww is encoded into mm qubits (m≪nm≪n), run through a brief, learnable VQC, and then measured to produce a low-dimensional quantum feature vector. We process each input with multiple parallel VQCs each acting on its own partition of the features and concatenate their outputs into a classical CNN. Simulation cost per forward pass scales as O(P×2^m) where P≈N/w is the number of patches. Because m and circuit depth remain small (on the order of tens of gates), even large datasets stay tractable.

Since I'm working with 1D data, we simply have a 1D kernel (aka a 'sliding window'). I add onto this framework by creating multiple filters that work sequentially, partitioning the features into three different feature spaces. Afterwards, I pass the output tensor into a classical CNN. We will report the outputs of these hybrid neural networks and a classical CNN of similar power to provide a baseline. Moreover, this will allow us to empirically observe whether we have attained quantum advantage.

We employ the Fast Gradient Sign Method (FGSM) as a one-step adversarial attack. FGSM perturbs each input in the direction of the loss gradient, scaled by ε, requiring only a single backpropagation pass. By plotting model accuracy against ε, we obtain a robustness curve. To boost F1-score, recall, and precision, we partition the dataset into k folds and train an independent "expert" model on each. At inference, we average the experts' probability outputs to form the ensemble prediction. This approach smooths out individual over- or under-fitting and gives us roughly a **10 % improvement** in F1-score and recall versus single-model baselines. The trade-off is a k-fold increase in training time, memory footprint, and inference latency, justified in contexts where maximal accuracy is critical.

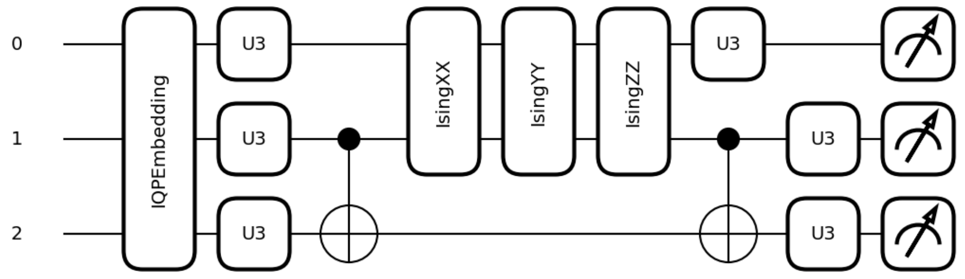
We also run explainability and interpretability testing via SHAP, LIME, and counterfactual identification on a random subset of the test dataset to determine data points and features of interest for this problem and model.

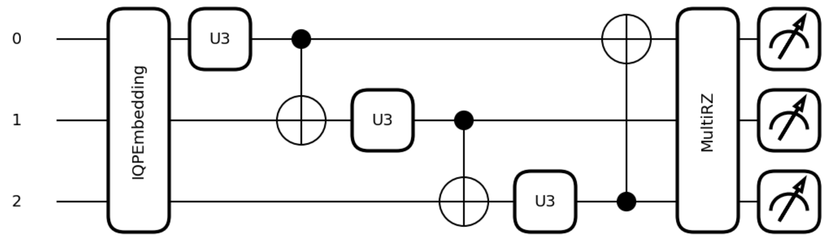
Now, I'll give diagrams of the high-level overview of the neural architectures I made, as well as circuits I used in my VQCs.

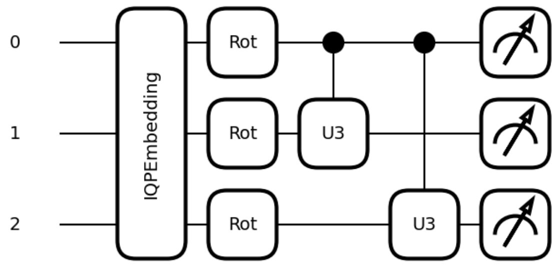


Quanvolutional NNs on 2D data

Normally, we stride by 1 value in the convolution, but instead I elected for 3 different kernels of 3 qubit VQCs, operating simultaneously, but each one has a stride of 3. In the breast cancer diagnosis dataset, we start with a 30 dim feature vector, so under this process we get a (9,10) tensor. This is then fed into a typical classification CNN (2 Convolutional layers, each with batchnorm, relu, and pooling. Then 2 dense layers to get the prediction). For the wine dataset, I used only one 3-qubit kernel since the number of features was prime (11), preventing any neat combination of kernels (aside from 11 different ones I guess but I couldn't think of 11 different enough circuits), so I used a size 3 sliding window, yielding a (3,8) tensor, which is fed into a CNN.







The three quanvolutional kernel circuits I used. The top one is used in both the breast cancer and wine datasets.

Results

All models were training using the Adam optimizer with learning rate 0.0002.

Breast Cancer Diagnosis Dataset (80% train, 20% test)

Classical baseline: classical CNN with 16193 trainable parameters. After training for 25 epochs, this classical model achieved 91.23% test accuracy, 0.8636 precision, 0.9048 recall, and a 0.8837 F-1 score.

Quanvolutional k-fold CNN: hybrid CNN with 21355 trainable parameters. After training each expert for 25 epochs, this hybrid model achieved 94.74% test accuracy, 0.9762 precision, 0.8913 recall, and a 0.9318 F-1 score. This model surpassed its classical counterpart in multiple metrics so we can conclude that the quanvolutional layers contributed to a better model. While there are five thousand more trainable parameters, this is mainly due to the ensemble nature. Each individual expert has around 4250 trainable parameters. We see a comparison of the robustness to FGSM attacks between the models in the below table. The classical model appears to be much less robust to noise, indicating some overfitting, which can be confirmed by the training accuracy consistently reaching 100%.

	Epsilon = 0	Epsilon = 0.01	Epsilon = 0.05	Epsilon = 0.1	Epsilon = 0.2
Hybrid Model	94.74%	94.74%	88.57%	83.33%	66.67%
Classical Model	91.23%	90.34%	76.32%	61.40%	41.23%

Model accuracy on FGSM Attacks on varying epsilons

Red Wine Quality Dataset (80% train, 20% test)

This model's quality attribute takes 11 different integer values, ranging from 0 to 10, inclusive. Further analysis on the dataset shows that mainly values from 3 to 9 show up, with 6 being the mode. As a result, we turn this into a binary classification problem by labelling all wines with quality >=6 as 1 and 0 otherwise.

Classical baseline: classical CNN with 3841 rainable parameters. After training for 30 epochs, this classical model achieved 66.25% test accuracy, 0.6761 precision, 0.7 recall, and a 0.6879 F-1 score.

Quanvolutional k-fold CNN: hybrid CNN with 19915 trainable parameters. After training each expert for 25 epochs, this hybrid model achieved 68.13% test accuracy, 0.7171 precision, 0.6488 recall, and a 0.68125 F-1 score. While the hybrid model achieved a better test accuracy and precision, the models seem comparable An interesting thing is that the precision of the hybrid model is higher than the precision of the classical model, but the opposite is true for recall, meaning the classical model predicts +1 correctly more often than the hybrid model, but also predicts +1 more often in general. Below we see another table capturing the resistance to FGSM attacks, and the hybrid model appears to not be robust at all. Consider the two major changes here are the quantum kernels and the dataset, further testing is needed to verify whether using more kernels makes your data more robust. The classical model also didn't perform well on this dataset either, however, so the dataset could be difficult to do binary classification with. Other options include a multinomial logistic regression, but I don’t have time to train that.

	Epsilon = 0	Epsilon = 0.01	Epsilon = 0.05	Epsilon = 0.1	Epsilon = 0.2
Hybrid Model	68.13%	63.44%	41.25%	23.13%	10.94%
Classical Model	66.25%	64.06%	54.38%	44.69%	29.69%

Model accuracy on FGSM Attacks on varying epsilons

Additional Tweaks if I had time or if I was using quantum hardware

The most obvious constraint is being limited by simulation times for quantum circuits, forcing my circuits to be 6 qubits max if I want the circuit to finish within an hour. This combined with the fact that some datasets had many data points limited areas of creativity, especially in constructing a quantum circuit in my opinion. Aside from constructing multiple circuits to execute sequentially, I wanted to play around with the quantums singular value transform (QSVT), which allows you to implement nonlinear transformations on quantum circuits. In the case of QML, we could implement nonlinear activation functions, thus enhancing a quantum CNN. But back to the circuit size limitation, it’s difficult to make a deep qCNN and incorporate QSVT without one epoch taking upwards of 20 minutes (yes I tried).