QCNN for Arbitrary Input Dimension

Si-Heon Park[1], Dong-Ha Kim[1], Ju-Young Park[1], Jung-Han Lee[2], and Prof. Daniel Kyungdeok Park*[2]

[1]KAIST
[2]Yonsei University

June 29, 2022

1 Introduction

The Convolutional Neural Network (CNN) has been utilized for various pattern recognition problems. Due to arising need for efficient Quantum Machine Learning (QML) algorithm, Quantum Convolutional Neural Network[1] has been proposed inspired by CNN, not only for classical data, but also quantum data. However, QCNN does not provide freedom in choosing the number of input qubits. For instance, when one out of two qubit is traced-out for qubit pooling process, the number of initial qubit should be in the form of 2^a where a is an integer. In addition, when amplitude encoding of classical data is considered, the dimension of the data should be 2^{2^a}. This constraint restrict applications of QCNN in machine learning.

In this work, we provide various padding methods on input data and qubits to overcome the addressed drawback. We then benchmarks those methods in terms of binary MNIST data classification to conclude that the number of additional ancillary qubits for padding can be only one even with very high accuracy. Finally, we provide application of our finding for Quantum Error Correction (QEC) compared with Shor's (bit-flip) QEC code.

2 QCNN

2.1 Construction of QCNN Circuits

As discussed later on, numerous QCNN models are used. We modularize our code implementing QCNN structure and corresponding padding methods. The module datasetloading.py is responsible for loading training and test data from .csv files and compressing data into desired one dimensional vector with the help of Principle Component Analysis (PCA).

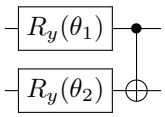
Among various ansatzes referred from Hur et al.[2] we have chosen ansatz composed of two R_y gates and a $CNOT$ gate (shown in Fig. 1a) as convolutional layer ansatz, for having relatively short circuit depth and the small number of parameters, while having competitive accuracy.[2] The pooling layer ansatz (shown in Fig. 1b) consist of two CR_y gates, so that one of two qubits propagates to the next layer. These ansatzes are implimented in ansatz.py module.

Our QCNN models in modules QCNN_circuit_only.py and QCNN_circuit_only_single_ancilla.py accept convolutional and pooling ansatz as inputs for construction to maximize convenience. Each QCNN model has unique arrangement of convolution and pooling layers. Since only two ansatz are being repeatedly used throughout the models, our code design provide benefits.

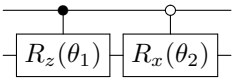
3 Result

3.1 QCNN results

In this learning process, first we generated 10000 train datas and 1000 validation datas from train MNIST data set, and 1000 test datas from test MNIST data set. Also, we only classified 0 labeled



(a) Convolutional Layer Ansatz



(b) Pooling Layer Ansatz

Figure 1: Ansatz Structures

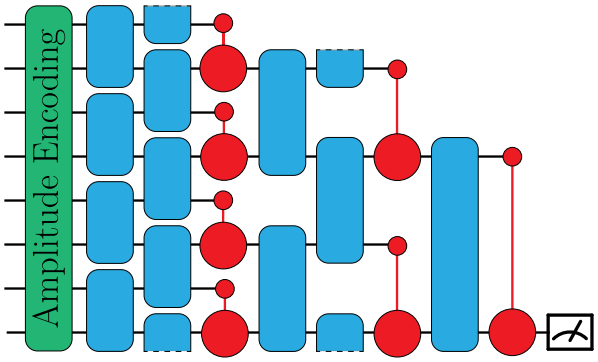


Figure 2: Quantum Convolutional Neural Network. The blue blocks are ansatzes for convolutional layers (in Fig. 1a and the reds are for pooling layers (in Fig. 1b)

and 1 labeled MNIST data set. Cost function of classification is Mean Square Error (MSE) between true label and estimated label with batch size 25. Expectation Z-measurement value on the last qubit of QCNN is the estimated label of the data. Optimization on the cost function has done by Adam optimizer with learning rate 0.01. We have compressed dimension of 28×28 MNIST dataset using PCA algorithm (to 256 or 30) and embedded that data to quantum state by Amplitude Encoding:

$$|\mathbf{x}\rangle = \frac{1}{\|\mathbf{x}\|} \sum_{i=0}^{2^N-1} x_i \, |i\rangle \tag{1}$$

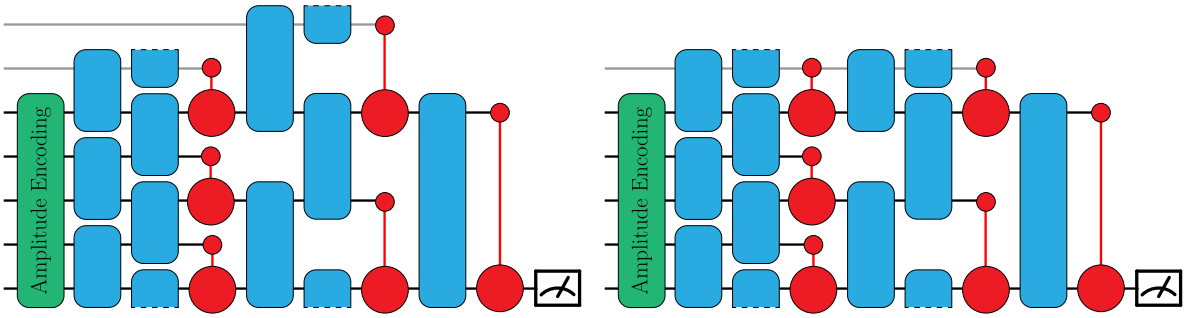
where N is the number of qubits.

Since QCNN reduces the number of qubits by half every layer, one needs an input state that can be encoded into a power of 2 number of qubits. For example a datapoint with dimension 256 can be amplitude encoded into an 8 qubit state. 8 qubits is a power of 2, so applying a QCNN is simple. However, consider a datapoint with dimension 30. We need to alter the QCNN architecture or embedding method to make the number of qubits in each layer even.

We propose two QCNN architectures to meet this criteria: layer-wise padding and single-ancilla padding. Layer-wise padding (Fig. 3a) simply adds an ancilla qubit every time a layer has an odd number of qubits. The convolution and padding is adjusted to take account for the ancilla. For example, in 5 qubit case, 5 is odd so we add one ancilla qubit and then layer became 6 qubit. By pooling, second layer became 3 qubit and 3 is also odd. Similarly, we add one ancilla qubit to make second layer 4 qubit. No additional ancillary qubits are needed since the number of qubits in the layer is power of 2. Single-ancilla padding (Fig. 3b) adds an ancilla qubit the first time if a layer has an odd number of qubits, and reuses this qubit whenever the number of qubits in the layer is odd.

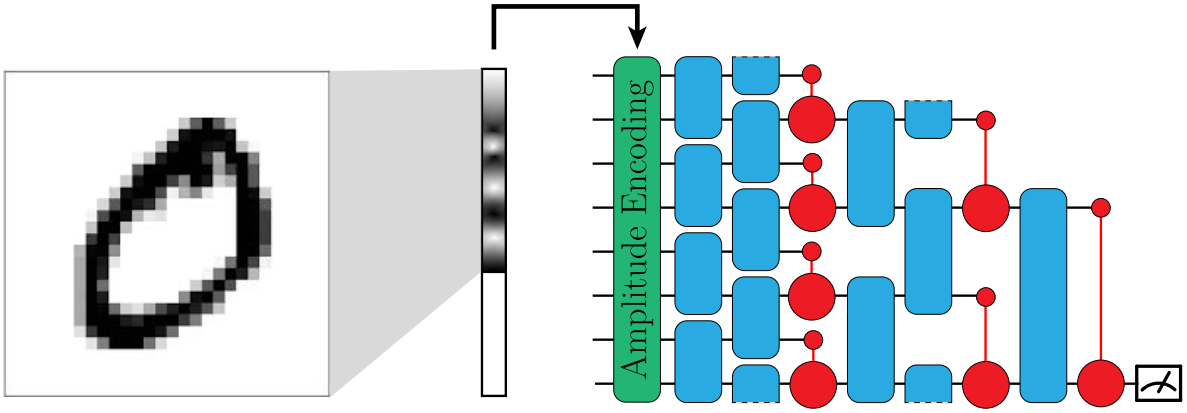
Fig. 5 displays dimension of feature, number of qubit to embed (initial qubit), used ancilla qubit number, total used qubit number, test accuracy for test data set. In fig. 5, single-ancilla qubit padding method used lowest total number of qubit, lowest number of ancilla qubit and highest test accuracy, this implies single-ancilla qubit padding is best performanced method. We propose two embedding methods to meet this criteria: zero-data padding and periodic data padding. In both methods, the input data has its dimension d increased to 2^{2^a} by adding additional elements, where $a = \lceil \log_2 \log_2 d \rceil$ is an integer. Thus using these methods the total number of qubits used is 2^a. For the zero-data padding (Fig. 3c), the additional elements are zero. For the periodic-data padding (Fig. 3d), the original data is repeated, like a 1-dimensional periodic boundary condition.

Fig. 4 displays graph of train loss and validation loss during learning process for 5 methods. Also,

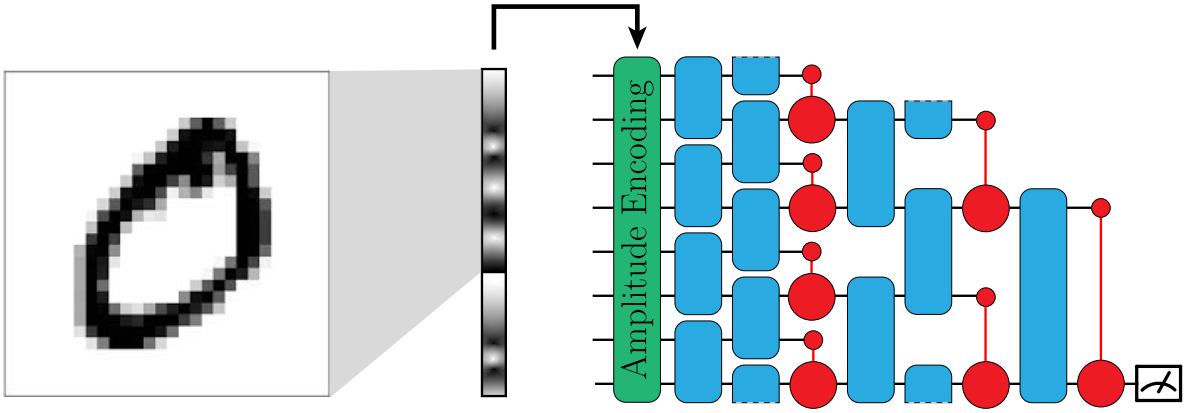


(a) Layer-wise padding

(b) Single-ancilla padding

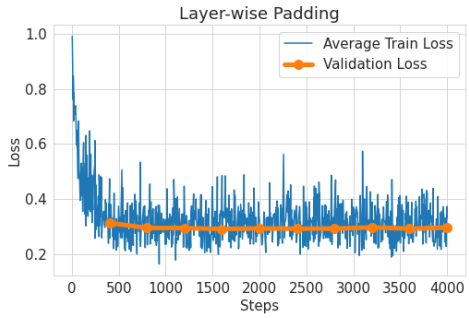


(c) Zero-data padding

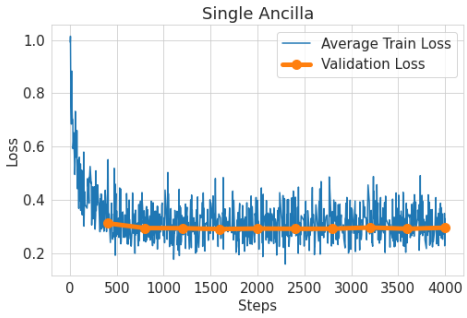


(d) Periodic-data padding

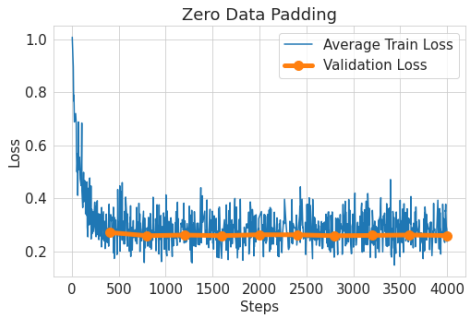
Figure 3: The padding methods tested in this work. a A new ancilla qubit is added for every layer with odd number of qubits. b An preexisting ancilla qubit for previous layer is reused for for every layer with odd number of qubits. c input data compressed to one-dimensional vector is padded with zero values to fullfile 2^{2^a}-dimension condition. d input data compressed to one-dimensional vector is padded with itself to fullfile 2^{2^a}-dimension condition.



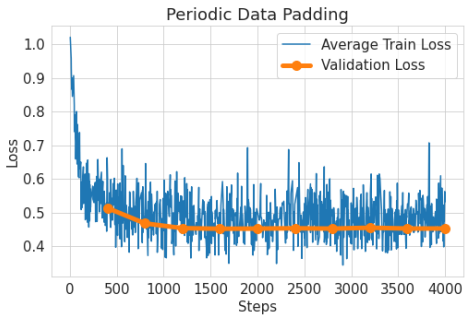
(a) Layer-wise Padding



(b) Single-Ancilla Padding



(c) Zero-data Padding



(d) Periodic-data Padding

Figure 4: Padding training results

Fig. 4 shows that every validation loss decreases during learning process. It can be interpreted that QCNN did not show overfitting in the training.

Fig. 5 displays dimension of feature, number of qubit to embed (initial qubit), used ancilla qubit number, total used qubit number, test accuracy for test data set. In fig. 5, single-ancilla qubit padding method used lowest total number of qubit, lowest number of ancilla qubit and highest test accuracy, this implies single-ancilla qubit padding is the most efficient performing method.

Method	Features	Initial Qubits M	Ancillary Qubits	Total Qubits	Accuracy
None	256	8	0	8	97%
Layer-wise	30	5	$\mathcal{O}(\log(M))$=2	7	97.4%
Zero Data Padding	30	5	$\mathcal{O}(M)$=3	8	98.4%
Periodic Data Padding	30	5	$\mathcal{O}(M)$=3	8	94.6%
Single-Ancilla	30	5	1	6	98.9%

Figure 5: Result Summary

4 Quantum Error Correction Using QCNN

4.1 Motivation

One proposed application of QCNNs is to optimize quantum error correction [1]. This method uses a QCNN as a decoder and its inverse as the encoder. In this section, this study focuses on training a QCNN quantum error correction decoder for an odd-qubit input.

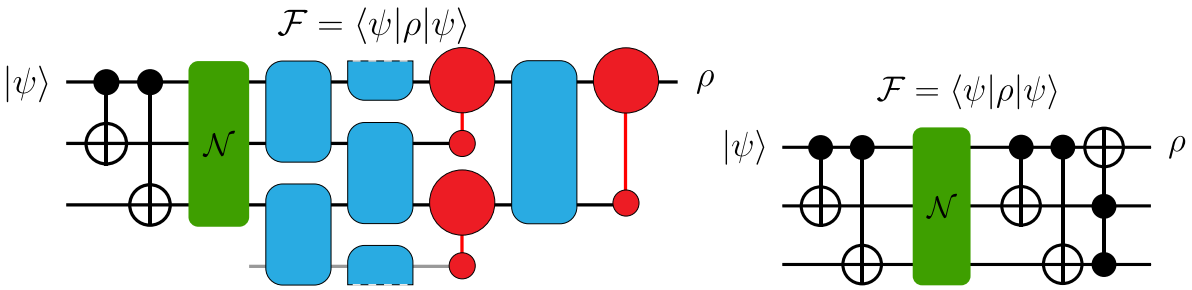
4.2 Methods

To create the decoder, a QCNN is used to minimize the following cost function:

$$C(\theta) = -\frac{1}{|S|}\sum_{|s\rangle\in S} F(|s\rangle, QCNN_\theta\{\mathcal{N}[U_E|s\rangle]\})$$

Here, F is the state fidelity while the training set is $S = \{|0\rangle, |1\rangle, |+\rangle, |-\rangle, |+i\rangle, |-i\rangle\}$. The second state in the fidelity function is the input state encoded into a larger state, passed through a quantum error channel and acted on by a QCNN. Thus if the QCNN is to be a decoder, the fidelity between the input and output states should be maximized. Here, $|0\rangle$ initialized ancillas are inserted during the encoding, decoding phase when necessary.

In this study, we used the Shor's bit flip code for the encoder and a 3-qubit depolarizing quantum error channel. The QCNN structure is shown in Fig. 6a, where the convolution and pooling ansatzes are the same in the previous sections. Note that this architecture is the same as layer-wise padding and single-ancilla padding discussed in the previous sections. All numerical experiments were done using Qiskit and the AerSimulator (shot number 1024). The QCNN decoder was trained in different noise parameter values, and the achieved fidelity was compared with that of the Shor's bit flip code decoder. The Shor's bit flip code is shown in Fig. 6b.



(a) QCNN Decoder Training Circuit (b) Shor's Bit Flip Code

Figure 6: QCNN Decoder Circuit and Shor's Bit Flip Code

4.2.1 Results

The result is summarized in the following graph (Fig. 7). The results showed that the QCNN decoder achieved slightly better fidelity results than the Shor's bit flip code decoder. Thus using an ancilla qubit, we were able to train a decoder QCNN that achieved similar (or arguably better) error correcting capabilites. Since the number of layers in a QCNN increases logarithmically as a function of the number of qubits and the QCNN does not suffer from the barren plateau problem, we postulate that this QCNN decoder method is a scalable way of constructing error correction decoders.

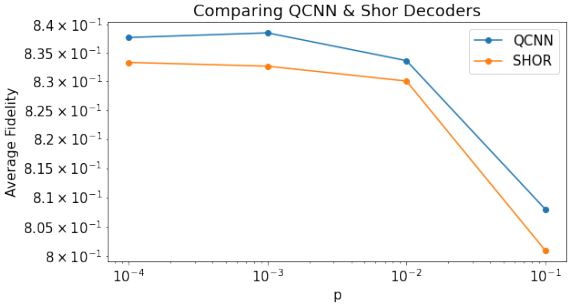


Figure 7: Comparison Between the QCNN Decoder and Shor's Code (Simulation)

5 Conclusion

QCNN is a promising quantum neural network structure, but suffers from constraints to the data input dimension. This study compares the effect of 2 different data padding and 2 different qubit padding methods applied on the binary classification of '0' and '1' digits of the MNIST dataset. The single-ancilla method outperformed all other methods. This method also has the advantage in qubit scaling; no matter how many qubits are used, only a maximum of one ancilla qubit is needed.

This method is also used to create a QCNN quantum decoder for an error correction code. By changing the error parameter for the error channel, the QCNN quantum decoder slightly outperformed the Shor bit flip code decoder. Since QCNN is scalable in terms of layer numbers and barren plateaus, we suggest that this method may be able to scale.

We expect that the proposed padding methods need to be benchmarked on multiple datasets, with different numbers of qubits. Also testing different ansatz structures should provide a more accurate picture of the QCNN performance.

References

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

[2] T. Hur, L. Kim, and D. K. Park, "Quantum convolutional neural network for classical data classification," *Quantum Machine Intelligence*, vol. 4, no. 1, pp. 1–18, 2022.