A Quantum Convolutional Neural Network for Image Classification

Milind Upadhyay

Abstract

From self-driving cars to medical diagnoses, machine learning (ML) has revolutionized our world in the last few decades. Additionally, Quantum Computing has considerable promise for the future, with superposition and entanglement making quantum algorithms much more efficient and effective than their classical counterparts. Quantum ML aims to apply these quantum principles to the groundbreaking field of ML, such as for image classification. Convolutional Neural Networks (CNNs) are very effective for classification, so we study a Quantum Convolutional Neural Network (QCNN) that is trained to distinguish between handwritten numbers in the MNIST dataset and compared to a similar-sized classical network. After tuning the QCNN and quantum encoding, the QCNN achieved comparable accuracy to the classical network.

Introduction

Quantum ML can be applied to many classification problems just like classical ML, using quantum neural networks. Quantum networks involve layers of rotation gates by parameterized angles. The optimal angle parameters for the given problem are solved for by a process such as gradient descent, similar to how weights and biases are solved for in classical networks. Farhi et al. *[4]* apply a simple, pure quantum network to classifying handwritten digits. Quantum networks can also be done as "hybrids", with some classical component of ML or feature detection, as done for medical image classification by Mathur et al. *[7]*

Classical CNN's are widely used for image classification. QCNN's are special quantum networks that are somewhat similar to classical ones in the sense that they have layers of convolution and pooling. Tensorflow Quantum *[2]* has applied a QCNN to classify if a cluster state of qubits *[8]* - a state with a highly-entangled group of qubits - is excited or not. This QCNN is a simpler version of the one presented by Cong et al. *[3]*

Classical inputs can be encoded into qubits in a multitude of ways, as described by LaRose et al. *[5]* Angle encoding involves using classical inputs as parameters for rotation gates on qubits, and simple versions of it use one qubit per classical input value. Angle Encoding is very simple, requiring only one gate to encode a value.

We apply Tensorflow Quantum's *[2]* QCNN, to distinguish handwritten digits in the same dataset as Farhi et al. *[4]* We entangle all the input qubits in a cluster state before plugging them into the QCNN.

Method

The input images are first downsized and stored in qubits with angle encoding. These qubits are then plugged into a QCNN which outputs the image class.

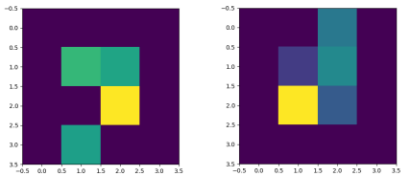
Dataset

We use the MNIST dataset *[6]*, which contains grayscale images of handwritten digits. Due to the limits on current quantum hardware, we heavily downsample this dataset. We filter the dataset into a binary classification between 3's and 6's, which results in 12,049 train images and 1968 test images.

We also downsample the images from 28 by 28 to 4 by 4 to avoid using too many qubits. This is around the minimum size to still be able to distinguish between the numbers.

Due to heavy downsampling, some test images become identical to train ones. We removed these 317 test images, resulting in 1597 testing images in the final dataset.

Figure 1: Downsized handwritten digits (3 and 6)



Encoding

We then encode the downsized images into the clusterstate qubits with Angle Encoding. For each pixel's grayscale value, we rotate a corresponding qubit around the X axis, using the pixel value as an angle. In other words, for a grayscale pixel at row i and column j, p_{ij}, the corresponding qubit has a $RX(p_{ij})$ gate performed on it.

The simple quantum network *[4]* applied to the MNIST dataset, binarized the pixel values, which means converting each grayscale pixel value (0 to 255) to either 0 or 1 based on if the value is greater than a certain threshold. We instead use the actual grayscale value (0 to 255) as a rotation, to make this extendable to complex images that would lose critical information when binarized.

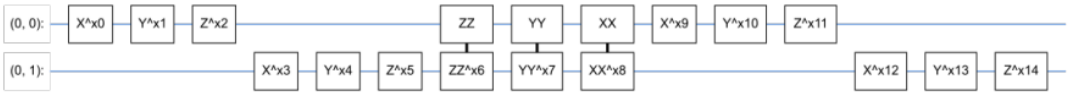
QCNN

Just like a classical CNN, a QCNN has layers of applying convolution on groups of inputs (qubits) and then pooling them together. We follow Tensorflow Quantum's *[2]* QCNN structure.

The convolution step involves performing parameterized two-qubit gates on consecutive qubits. The two qubit gates involve a set of parameterized gates done separately on each qubit, then a set of parameterized gates done on the two qubits together, and finally another set of separate single-qubit gates rotations. The parameterized single-qubit gates are X, Y, Z raised to the

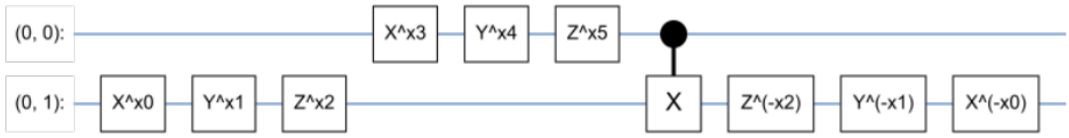
power of the parameters, and the two-qubit gates are ZZ, YY, XX raised to powers which are the parameters. The parameters will ideally become those that cause the rotations to check for important features in the qubits, which can then get pooled together.

Figure 2: Convolution Circuit Diagram



The pooling operation takes in two qubits and reduces them into one, transferring a "source" qubit into a "sink" one. First, it involves another set of one-qubit parameterized gates on both qubits, the same ones as the parameterized X, Y, Z gates in the convolution step. It then performs a CX gate from the source to the sink one, and performs the inverse of the same parameterized one-qubit gates on the sink qubit. This is different from the classical pooling which would normally find the maximum output of the convolution kernel, but both reduce the size of inputs by combining results.

Figure 3: Pooling Circuit Diagram



The overall QCNN circuit involves layers of convolution and pooling. Since the number of qubits is halved after each layer due to pooling, the number of qubits has to be a power of 2. The number is indeed a power of 2 because the 4 by 4 images result in 16 qubits.
For each layer, the convolution is run on the qubits, and then the first half of qubits are pooled into the second half. After this, the first half of qubits are thrown away, and the process is repeated with the second half until only 1 qubit is left. This is then read out with the Z operator to determine whether the input was a 3 or a 6.

Classical Network
The classical network is a simple non-convolutional network. After flattening the input into 16 pixel values, it has a dense layer of 5 neurons and then an output layer of 1 neuron. This results in 91 parameters, which is slightly over the quantum one which has 84 parameters. It is therefore a decent comparison to evaluate the quantum network. We did not use a classical convolutional network because it requires more parameters and higher-dimensional inputs.

Results

Using Tensorflow *[1]*, we trained the quantum and classical networks on the MNIST dataset and compared their performances.

QCNN
We achieved the best performance for the QCNN with 5 epochs and a batch size of 50. Even though this only uses a fraction of all training images, this was the best because the network started overfitting with more epochs or a higher batch size. The QCNN achieved an accuracy of around 87%. When we binarized the images (mentioned in the *Encoding* section) it went up to 90%, however we did not pursue that strategy further for reasons mentioned in that section. The accuracy for our QCNN is slightly above the accuracy of the simple quantum network *[4]* which performed the same task.

Comparison to Classical Network
With the same epochs and batch size, the classical network would not consistently converge to a high accuracy, ranging between 70% to 90%. However, when given 20 epochs and a batch size of 128, it consistently converged to around 90%. Therefore, when kept at the ideal parameters for the QCNN, the QCNN actually outperformed the classical network because it learned considerably faster. However, when each model was tuned to its ideal performance, they both performed similarly.

Conclusion

Ultimately, given the comparable performance of this QCNN on the MNIST dataset to a similar-sized classical network, a QCNN combined with angle encoding is a viable strategy for image classification. In the future, Quantum Computers will have many more qubits and encoding pixel values for large images may be viable. However, for current Quantum Computers, other techniques such as encoding features or keypoints may need to be used.

References

1. Abadi, M. *et al.* TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org (2015).
https://www.tensorflow.org/

2. Broughton, M. *et al.* TensorFlow Quantum: A Software Framework for Quantum Machine Learning (2020).
https://arxiv.org/abs/2003.02989

3. Cong, I., Choi, S. & Lukin, M. Quantum convolutional neural networks. *Nature Physics* (2019).
https://www.nature.com/articles/s41567-019-0648-8

4. Farhi, E. & Neven, H. Classification with Quantum Neural Networks on Near Term Processors (2018).
https://arxiv.org/pdf/1802.06002.pdf

5. LaRose, R. & Coyle, B. Robust data encodings for quantum classifiers (2020).
https://arxiv.org/pdf/2003.01695.pdf

6. LeCun, Y. & Cortes, C. MNIST handwritten digit database (2010).
http://yann.lecun.com/exdb/mnist/

7. Mathur, N. *et al.* Medical image classification via quantum neural networks (2021).
https://arxiv.org/pdf/2109.01831.pdf

8. Nielsen, M. CLUSTER-STATE QUANTUM COMPUTATION (2005).
https://arxiv.org/pdf/quant-ph/0504097.pdf