Implementing Residual Connection on QCNN architecture

Mini Project - *CS-2-14(PO)*

By

Krishiv Trivedi [2023UMA0218]
Piyush Kumar [2023UMA0227]
Krishna [2023UEE0140]
Jyoti Yadav [2023UEE0139]

Supervisor:
Sumit Kumar Pandey

12^{th}April, 2025

Introduction

Quantum computing is a new computing paradigm based on quantum mechanics that utilizes qubits instead of classical bits to store and process information. Quantum nature like superposition and entanglement aren't just theoretical figments, now with the capacity to actually engineer these quantum systems, this nature can be meaningfully utilized and harnessed to speed up NP Algorithms like "*Brute-Force* ***Knapsack***"[1] and the "***Travelling Salesman Problem***" (TSP) and improve our modern AI models.

While the majority of our journey includes learning about theory of quantum circuits, learning and developing our own quantum algorithms for solving TSP etc. Our final aim of this paper will be the upcoming, *Quantum Convolutional Neural Networks* (**QCNNs**)[2]. We will demonstrate not only of how we can use the 'Quantum' to develop better convolutional neural networks but also show an implimentation of a quantum circuit-based algorithm, *quantum residual neural networks*[3].

Finally, we will explain what we have learnt so far from this vast theory and then display the different results produced by a Classical CNN and a Quantum CNN. The results will on a fixed dataset for both the networks, and display the differences in the important parameters like accuracy, cost function trends and our current engineering limitations on implemetation of QCNNs.

Motivation

In this new quantum era, as the number of qubits increase and our engineering catches up to our theory, the security of our existing cryptographic methods is starting to diminish. The main motivation for this project was '**Post-Quantum Cryptography**'. Later, we transposed from "How to make a problem which cannot be efficiently cracked by a quantum computer..." to "All such problems which can be efficiently cracked by a quantum computer..." where we encountered many NP hard problems but first

[1] [https://en.wikipedia.org/wiki/Knapsack_problem]
[2] [https://pennylane.ai/qml/glossary/qcnn]
[3] [https://doi.org/10.1038/s42005-024-01719-1]

we focused on a known solved problem of 'Knapsack' after which we started to develop our own algorithm with all that we have learnt for TSP.

Q. How does a Quantum Computer solve such "*Brute-force*" problems '*faster*'?

→ We will not elaborate on the theory as to keep this report concise. While we have to classically iterate over all solutions to check, in quantum we use the ***Hadamard***[4] Gate to check all such superpositions simultaneously.

This is used extensively in quantum algorithms. Our main focus was on how to make the brute-force solution $O(2^n)$ more optimal, which we did using ***Grovers' Algorithm***[5] which reduced the complexity to $O(2^{n/2})$. In general, we can say that in such brute-force approaches, Grovers provides a well-known $\sqrt{N}$ "quadratic-speedup".

Q. Why did we choose *QCNNs/QRNNs*?

→ We think that making a hands-on quantum circuit in a domain which has practical implications in the future of machine learnning and AI will not only show how we can turn our theoritical knowledge into practice, which is the best way to display what we have learnt, but also simultaneously work on the edge of scientific innovation happening in the field of quantum computing as we speak. This is why we will be implementing a very recent paper published in 2024 as our contribution to this field of research.

Q. What do we actually use, software or anything to implement and run these quantum algorithms and get results?

→ While we do not have a quantum computer, *IBM's* quantum cloud computing provides us few qubits to work with. For this we are further using ***Qiskit*** to simulate qubits, even if due to the nature of qubits it gets difficult to simulate more than 14-15 qubits at a time on a classical computer, we can simulate for our use case.

Methodology & Implementation Idea

For our methodology, first we'll make a Quantum CNN on a particular dataset, after which we will make an equivalent Classical CNN and compare the results to see which one performs better and their cost function trends. After this, we will implement the 'Residual-block' into our QCNN to see the difference it produces.

- **Dataset** consists of all tic-tac-toe games which have not ended in a draw. This dataset is actually very complex compared to all the dataset which a QCNN is typically trained on, i.e. detecting horizontal/vertical stripes on a 2x2 or a 4x2 grid, but we decided to push the limits of current standards. This is because we wanted to make a QCNN model never made before.

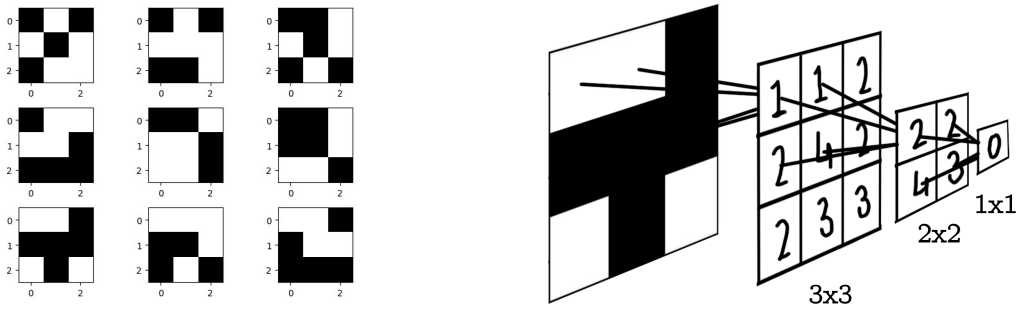


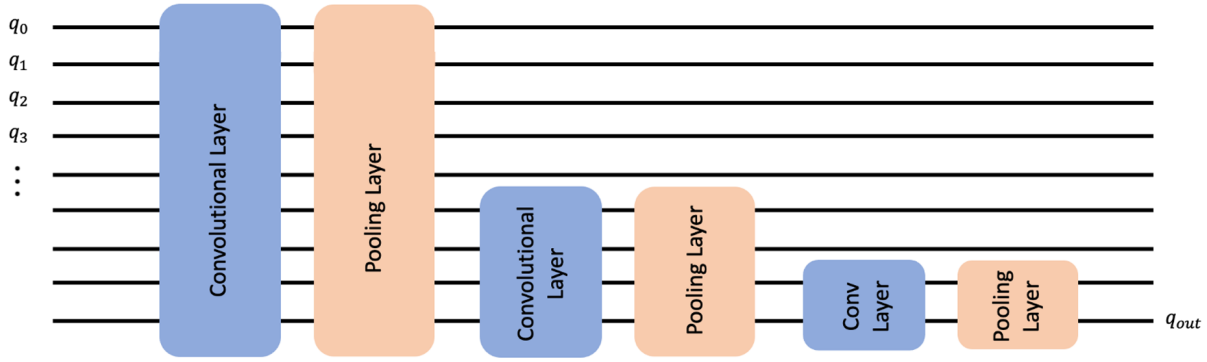
Figure 1: Our generated dataset and the overarching convolution model for both variations.

[4] [https://en.wikipedia.org/wiki/Quantum_logic_gate]
[5] [https://en.wikipedia.org/wiki/Grover%27s_algorithm]

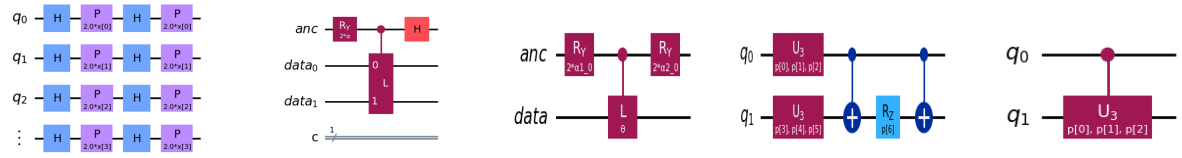
- **Layers** of pure convolutional networks are used. First is a convolution of a 3×3 to another 3×3 matrix, we'll call it $C_{3\times3}$. Next is a pooling layer from 3×3 to 2×2, we'll call it $P_{3\times3,2\times2}$. Similarly $C_{2\times2}$, $P_{2\times2,1\times1}$. The only difference is that the QCNN model uses an additional $P_{2\times2,2\times1}$ and $C_{2\times1}$ before pooling back via $P_{2\times1,1\times1}$.

$$\textbf{\textit{CCNN}}: \text{Input} \to C_{3\times3} \to P_{3\times3,2\times2} \to C_{2\times2} \to P_{2\times2,1\times1} \to \text{Output}$$

$$\textbf{\textit{QCNN}}: \text{Input} \to C_{3\times3} \to P_{3\times3,2\times2} \to C_{2\times2} \to P_{2\times2,2\times1} \to C_{2\times1} \to P_{2\times1,1\times1} \to \text{Output}$$



- **Training** of CCNN was done with Adam optimizer [*PyTorch*], Cost-function: Cross Entropy. QCNN was with COBYLA optimizer [*Qiskit*][6], Cost-function: Squared Error (Default).



(a) ZFeatureMap (b) $\mathcal{R}(x)$ (c) $\mathcal{R}_2(x)$ (d) C Layer element (e) P Layer element

- **Additional** blocks were added like ZFeatureMap[7] before sending them to the layers to encode the data into qubits. And then we built and added our 'Residual-Blocks' from the 2024 paper.

This is the method we have chosen to implement this quantum circuit. Our main focus in this paper is to show a detaied and practical implementation so we'll skip the intuition and theory which comes with this. Next is to report the results by both of these models [*Now uploaded on GitHub*].[8]

Results

For our Classical CNN, we understand that with our cross entropy function and Adams Optimizer which are all well developed systems when it comes to CNN gives our Classical model a huge advantage.

⇒ **Accuracy**: $81.5 \pm 1\%$

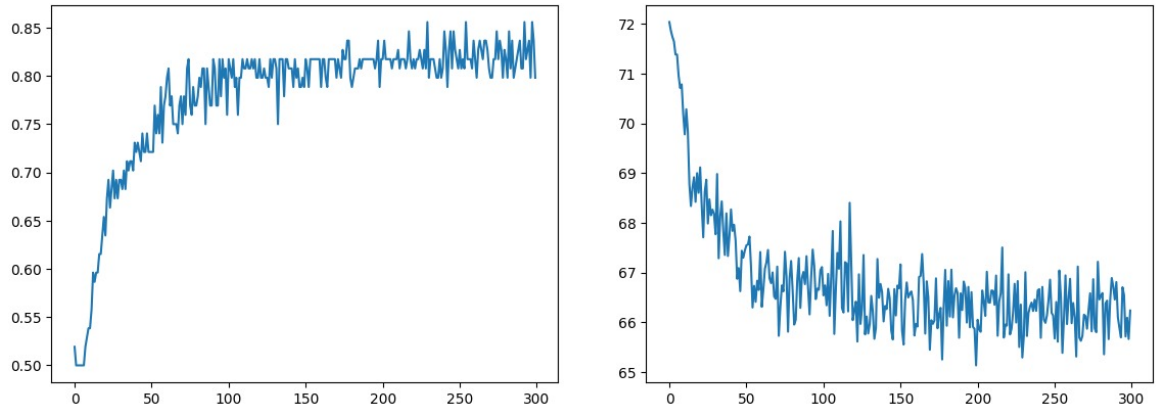
⇒ **Epochs**: 300

Similarly for the same dataset and same architecture for our QCNN model. Note that this comparision is purely with our convolutional QCNN and is without the use of $\mathcal{R}(x)$ and $\mathcal{R}_2(x)$ blocks.

⇒ **Accuracy**: 77%

⇒ **Epochs**: 200

[6] [https://docs.quantum.ibm.com/api/qiskit/0.27/qiskit.algorithms.optimizers.COBYLA]
[7] [https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.ZFeatureMap]
[8] Please see the references for the theoritical aspect, you won't be dissapointed!



(a) CCNN Accuracy

(b) CCNN Cross Entropy Loss

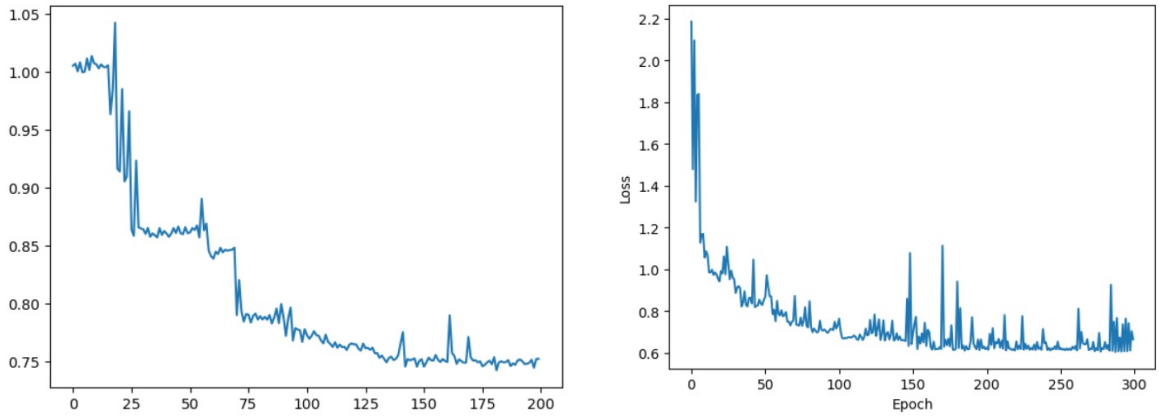
Figure 3: Results of Classical CNN

Now, we use the 'Residual-Blocks' to see the difference between QCNN with and without the residual connection architecture.

We also made another improvement in our residual QCNN, that is, diagonally entangling the qubits to also capture the diagonal features of this dataset. You can see the full circuit in the GitHub repo, rest all the components of the QCNN are compeletely the same. Now the results of just adding one layer of Residual Blocks is as follows.

⇒ **Accuracy**: 81.73%

⇒ **Epochs**: 300



(a) QCNN Sqrd.Error Loss wo/Residual Blocks

(b) QCNN w/Residual Blocks

Figure 4: Results of Quantum CNN

Conclusion & Future

Our Non-residual QCNN model reaches almost the same accuracy the CCNN reaches in 100 epochs. With more computing and time we might be able to replace CCNN with negligible decrease in performance. Also, the point to be noted is that our code is just simulating the qubits which makes it slower than if we had a actual quantum computer.

Meanwhile, our Improved-Residual-QCNN challenges our current Classical CNN! This small improvement and adding residual-blocks has made a significant breakthrough!

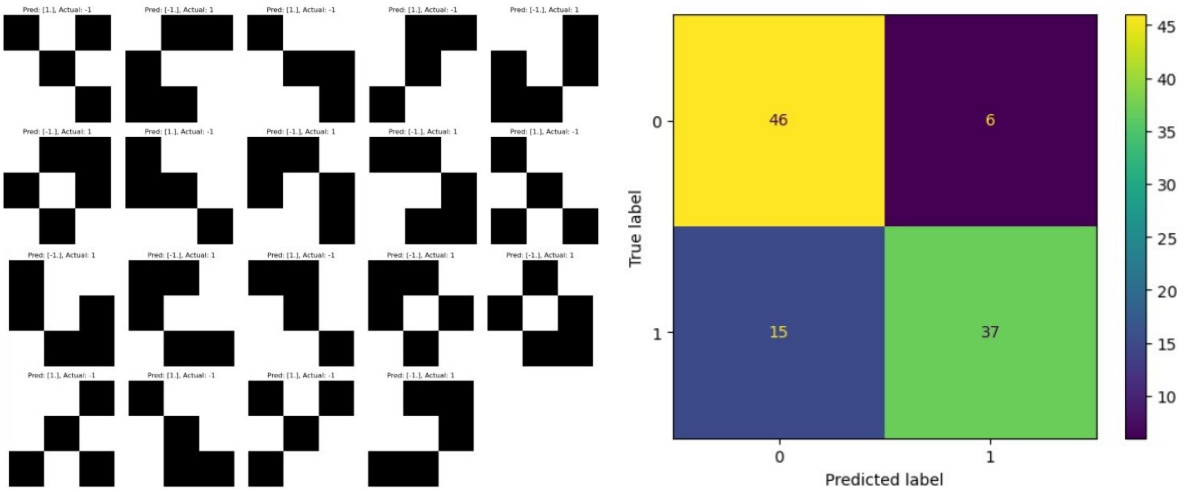
Q. ***So, What improved***?

→ We hypothesized that it is mostly to do with the "*Diagonal wins of a player*", this is because, when we analyzed all the games that our quantum model got wrong, we see that it was only the games where the wins were diagonal.

That is, the model correctly predicted all horizontal and vertical wins correctly. In case of diagonal cases...

Correct: 32/56 **Incorrect**: 24/56

So we think the diagonalization due to the "Residual-Blocks" and convolution layers made it so that our new QCNN was good at identifying those extra features. Our new and improved model however, has only 19 incorrect diagonals.



Our future prospects include making a multi-classification model. For example, a model which detects if a game is won by white or black or is a draw. We already partially have the multimodel ready on hand if you need for demonstration.

Literature Review & References:

- Wen, J., Huang, Z., Cai, D. et al. Enhancing the expressivity of quantum neural networks with residual connections. Commun Phys 7, 220 (2024).
- https://pennylane.ai/qml/glossary/qcnn
- https://docs.quantum.ibm.com/api/qiskit/0.27/qiskit.algorithms.optimizers.COBYLA
- https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.ZFeatureMap
- https://en.wikipedia.org/wiki/Grover%27s_algorithm
- https://en.wikipedia.org/wiki/Knapsack problem
- https://en.wikipedia.org/wiki/Quantum_logic_gate