ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

ĐỒ ÁN TỔNG HỢP (CO3101)

Báo cáo đồ án

Quantum Fourier Convolutional Neural Network (QFCNN)

GIẢNG VIÊN HƯỚNG DẪN: THOẠI NAM

STT	Họ tên SV	MSSV
1	Dương Hồ Hoàng Phúc	2212607
2	Huỳnh Thị Minh Tâm	2213013
3	Võ Nguyễn Gia Huy	2252270

Mục lục

1 Giới thiệu

1.1 Giới thiệu tổng quan

Trong những năm gần đây, lĩnh vực học máy đã có những sự phát triển vượt bậc. Vì đây là một trong những lĩnh vực đòi hỏi lượng tài nguyên tính toán rất lớn, yêu cầu dành cho các hệ thống tính toán hiệu năng cao cũng nhanh chóng tăng theo. Chính vì điều này, học máy lượng tử (Quantum Machine Learning - QML) là một trong những vấn đề đang rất được quan tâm hiện nay.

Một trong những hướng đi phổ biến của QML là phát triển các thuật toán QML từ các thuật toán học máy cổ điển sẵn có. Mạng neuron tích chập (Convolution Neural Network - CNN) là một kiến trúc học máy thường được sử dụng trong các tác vụ nhận biết và phân loại hình ảnh. Chính vì vậy, nhiều công trình nghiên cứu đã tập trung vào việc xây dựng các mạng CNN lượng tử như QCNN hay Quanvolution Neural Network (QNN). Trong đồ án này, chúng em sẽ hiện thực và kiểm chứng Quantum Fourier Convolution Neural Network - QFCNN, một mạng CNN lượng tử sử dụng biến đổi Fourier lượng tử (Quantum Fourier Transform - QFT) cho phép tính tích chập [1]. QFT là một trong những thuật toán lượng tử được biết đến rộng rải nhất nhờ vào ưu thế về độ phức tạp thời gian so với thuật toán biến đổi Fourier nhanh (Fast Fourier Transform - FFT) trong tính toán cổ điển. Chính vì vậy, về mặt lý thuyết, tốc độ xử lý phép tính tích chập có thể được cải thiện đáng kể khi được hiện thực trên máy tính lượng tử, từ đó cải thiện tốc độ xử lý chung của toàn bộ mạng neuron.

Trong báo cáo này, chúng em sẽ tập trung vào việc hiện thực mạch lượng tử giả lập thực hiện chức năng của lớp tích chập lượng tử của mạng QFCNN bằng simulator Cirq. Bên cạnh đó, chúng em cũng sẽ hiện thực giả lập số (numerical simulation) để có ước lượng chung về độ chính xác của QFCNN khi được huấn luyện trên bộ dữ liệu MNIST.

1.2 Giới thiệu về Cirq

Cirq là một thư viện mã nguồn mở cho Python do Google phát triển. Thư viện ban đầu được thiết kế để lập trình và chạy các thuật toán lượng tử trên các bộ xử lý lượng tử (Quantum processor). Sau đó, thư viện đã được thiết kế để có thể mô phỏng được các mạch lượng tử (Quantum circuit) trên máy tính cổ điển để các nhà nghiên cứu, kỹ sư và các nhà phát triển dễ dàng xây dựng và tối ưu hệ thống, thuật toán.

Tương tự với các thư viện mô phỏng lượng tử khác (Qikits,Pennylane,...) Cirq cho phép người dùng điều khiển chi tiết các mạch lượng tử để thiết kế thuật toán và tối ưu. Không chỉ vậy Cirq còn có thể hỗ trợ với các vấn đề thực tế từ bộ xử lý lượng tử như crosstalk, decoherence và noise. Thậm chí Cirq còn thể hiện khả năng vượt trội ở tính linh hoạt cho phép điều chỉnh nhiều tham số.

Tuy nhiên, Cirq chưa phải là công cụ mô phỏng lượng tử tốt nhất với nhiều vấn đề hạn chế. Khác với Qikits của IBM có tích hợp sẵn các thuật toán cấp cao (Quantum convolution neural network, Grover's search,..), người dùng Cirq phải lập trình công cụ từ đầu, theo đó mà làm tăng thời gian xử lí thuật toán cũng như tối ưu. Không chỉ vậy, Cirq không thực sự tốt ưu tốt khả năng tính toán ở quy mô rộng (nhiều qubit) gây khó khăn cho việc thiết kế và tính toán hiệu suất.

2 Cơ sở lý thuyết

2.1 Tổng quan về tính toán lượng tử và quy ước ký hiệu

Tính toán lượng tử là một nhánh nghiên cứu kết hợp giữa khoa học máy tính và cơ học lượng tử. Tương tự với bit trong tính toán cổ điển, qubit là đơn vị tính toán trong tính toán lượng tử. Một qubit được cấu thành từ hai phần là các thái riêng (eigenstate) và các biên độ ứng với các trạng thái riêng đó.

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

Trạng thái $|\psi\rangle$ của qubit trên bao gồm các trạng thái riêng $|0\rangle$ và $|1\rangle$ và các biên độ α và β. Ta có α^2 và β^2 là các xác suất để việc đo đạc giá trị qubit cho ra các kết quả 0 hoặc 1.

Đối với các hệ có nhiều qubit, các trạng thái riêng của hệ sẽ được kí hiệu bằng tổ hợp các chuỗi giá trị 0 và 1. Một hệ gồm n qubit sẽ có 2^n trạng thái riêng.

$$|\psi\rangle = \alpha_0\,|00..00\rangle + \alpha_1\,|00..01\rangle + ... + \alpha_{2^n-1}\,|11..11\rangle$$

Nếu ta coi các chuỗi 0 và 1 là các số nhị phân, ta có thể chuyển đổi chúng về giá trị thập phân tương được và viết chúng dưới dạng:

$$|\psi\rangle = \alpha_0\,|0\rangle + \alpha_1\,|1\rangle + ... + \alpha_{2^n-1}\,|2^n-1\rangle$$

Dĩ nhiên, ta có $\sum_{i=0}^{2^n-1}\alpha_i^2 = 1$, vì α_i^2 chính là xác suất đo được giá trị bằng với trạng thái riêng tương ứng $|i\rangle$. Trong tính toán lượng tử, một trạng thái của một hệ qubit thường được biểu diễn dưới dạng một vector thuộc không gian Hilbert, với tập hợp các trạng thái riêng là cơ sở cho không gian vector đó.

Các phép biến đổi trên các qubit (operator) thường được thể hiện bằng các cổng lượng tử (quantum gate). Về mặt bản chất, chúng là những phép biến đổi tuyến tính và thường được biểu diễn bằng các ma trận Hermitian (ma trận bằng với ma trận liên hợp chuyển vị của chính nó). Một phép biến đổi chỉ có thể được hiện thực nếu nó có thể được biểu diễn bằng một ma trận Hermitian. Điều này là lý do các thuật toán cổ điển không thể dễ dàng được chuyển đổi thành một thuật toán lượng tử. Ở phần dưới, ta sẽ có một ví dụ về một phép tính cơ bản trong tính toán cổ điển nhưng lại không thể hiện thực được trong tính toán lượng tử.

2.2 Mã hóa biên độ (Amplitude Embedding)

Mã hóa biên độ là quá trình xử lí các dữ liệu cổ điển đầu vào thành biên độ trạng thái của các qubits dựa trên công thức:

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

Trong đó:

$|\psi\rangle$ là trạng thái lượng tử,

$\|\mathbf{x}\|$ là chuẩn của vectơ $\mathbf{x}$,

x_i là thành phần thứ i của $\mathbf{x}$,

$|i\rangle$ là Qubit thứ $\mathbf{i}$.

Ví dụ: Ta có 1 vecto

$$\mathbf{X} = [1, 2, 3, 4]$$

. Hãy mã hóa dữ liệu X thành biên độ Qubit.

Lời giải

Để mã hóa vector $\mathbf{X}$ thành biên độ Qubit, trước tiên cần chuẩn hóa vector $\mathbf{X}$ bằng cách tính chuẩn $\|\mathbf{X}\|$. Công thức chuẩn hóa như sau:

$$\|\mathbf{X}\| = \sqrt{\sum_{i=1}^{4} x_i^2}.$$

Thay các giá trị của $\mathbf{X}$:

$$\|\mathbf{X}\| = \sqrt{1^2 + 2^2 + 3^2 + 4^2} = \sqrt{30}.$$

Sau khi chuẩn hóa, trạng thái Qubit mã hóa biên độ của $\mathbf{X}$ sẽ là:

$$|\psi\rangle = \frac{1}{\sqrt{30}} \left(1|0\rangle + 2|1\rangle + 3|2\rangle + 4|3\rangle\right).$$

2.3 Biến đổi Fourier lượng tử

Phép biến đổi Fourier rời rạc (DFT) là phép biến đổi một dãy số x_j có chiều dài N thành dãy số y_k có độ dài N sao cho:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi ijk/N}$$

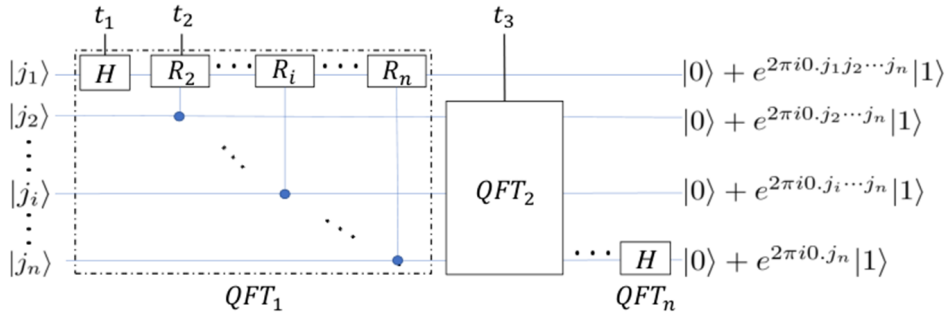
Phép biến đổi DFT có độ phức tạp là $O(N^2)$. Một biến thể của DFT là phép biến đổi Fourier nhanh (FFT) có độ phức tạp đã được cải thiện còn $O(NlogN)$. Đây là một trong những phép biến đổi mà người ta đã tìm ra một thuật toán lượng tử tương đương với nó. Tương tự như DFT (biến đổi Fourier rời rạc) cổ điển, biến đổi Fourier lượng tử hay gọi tắt là QFT hiện thực trên một trạng thái lượng tử $|X\rangle = \sum_{j=0}^{N-1} x_i |j\rangle$ và ánh xạ vào trạng thái $|Y\rangle = \sum_{j=0}^{N-1} y_k |j\rangle$ hay nói một cách khác:

$$\sum_{j=0}^{N-1} x_j |j\rangle \xrightarrow{\text{QFT}} \sum_{k=0}^{N-1} y_k |k\rangle$$

Với dãy số y_k là biến đổi Fourier của dãy số x_j

Một cách viết tương đương của phép cách biểu diễn trên là:

$$|j\rangle \xrightarrow{\text{QFT}} \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle$$



Hình 1: Bảng mạch QFT

$$\textit{Cổng điều khiển } R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix}$$

Gọi $n = log_2 N$, ta có giải thuật QFT đã cải thiện độ phức tạp của phép biến đổi Fourier xuống còn $O(n^2)$. Chính vì điểm mạnh này, phần tiếp theo sẽ tập trung vào việc xây dựng phiên bản lượng tử sử dụng QFT của một trong số những ứng dụng của FFT trong tính toán cổ điển.

2.4 Định lý tích chập và tích chập lượng tử

Tích chập rời rạc của hai dãy số f[n] và g[n] có N phần tử với n chạy từ 0 tới N-1 được định nghĩa là dãy số $(f * g)[n]$ sao cho:

$$(f * g)[n] = \sum_{m=0}^{N-1} f[m]g[n-m]$$

Từ định nghĩa trên, ta xây dựng định nghĩa phép tích chập lượng tử là phép biến đổi P như sao:

$$\sum_{t=0}^{N-1} f(t) \sum_{j=0}^{N-1} g(j) \, |ij\rangle \rightarrow \lambda \sum_{w=0}^{N-1} \sum_{j=0}^{N-1} f(t)g(w-t) \, |w\rangle$$

Trong đó, λ là hệ số chuẩn hóa để đảm bảo các biên độ thỏa mãn điều kiện tổng bình phương các biên độ đều bằng 1.

Định lý tích chập là một công cụ giúp ta có thể thực hiện phép tính tích chập bằng phép biến đổi Fourier cổ điển.

Định lý 1 *(Định lý tích chập) Cho hai dãy số u[n] và v[n] với biến đổi Fourier lần lượt là U[n] và V[n]. Ta gọi tích chập của u[n] và v[n] là r[n]. Khi đó, ta có biến đổi Fourier của r[n] là R[n] bằng với tích của U và V.*

$$R[m] = U[m]V[m]$$

Vì vậy, trên một hệ thống tính toán cổ điển, ta có giải thuật tính tích chập sử dụng biến đổi Fourier:

```
Algorithm: Fourier Convolution
Input: Input sequence I, Kernel K
Output: Convolution of I and K
    TransformedInput = FourierTransform(I)
```

```
    TransformedKernel = FourierTransform(K)
    TransformedConvol = TransformedInput*TransformedKernel
    return InverseFourierTransform(TransformedConvol)
```

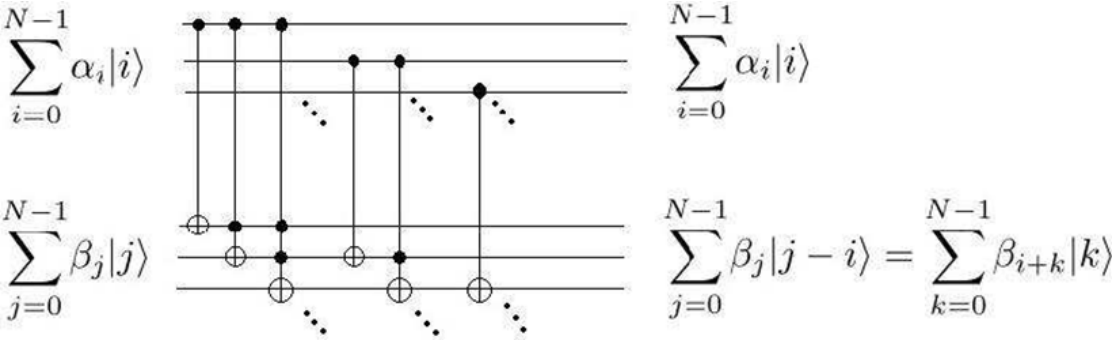
Giải thuật tính tích chập bằng biến đổi Fourier, mà cụ thể là phép biến đổi Fourier nhanh có độ phức tạm là O(MNlog(MN)) với M và N lần lượt là kích thước chiều dài và chiều rộng của ảnh đầu vào. Trong khi đó, độ phức tạp của phép tính tích chập thông thường là O(MNmn) với m và n là kích thước của kernel. Ta thấy, giải thuật tính tích chập bằng biến đổi Fourier sẽ có lợi thế hơn ở các trường hợp có kernel có kích thước lớn và ngược lại.

Tuy nhiên, giải thuật này đã được chứng minh là không thể hiện thực trực tiếp trên một hệ thống lượng tử. Điều này là do phép tính tích của hai dãy số, ở đây được hiểu là phép tính tích từng phần tử, là một phép biến đổi phi tuyến. Do vậy, không có một quá trình P nào có thể thực hiện được phép biến đổi trên [4].

Để xử lý vấn đề trên, ta sẽ thay thế phép tính tích từng phần tử bằng quá trình M thực hiện phép biến đổi sau:

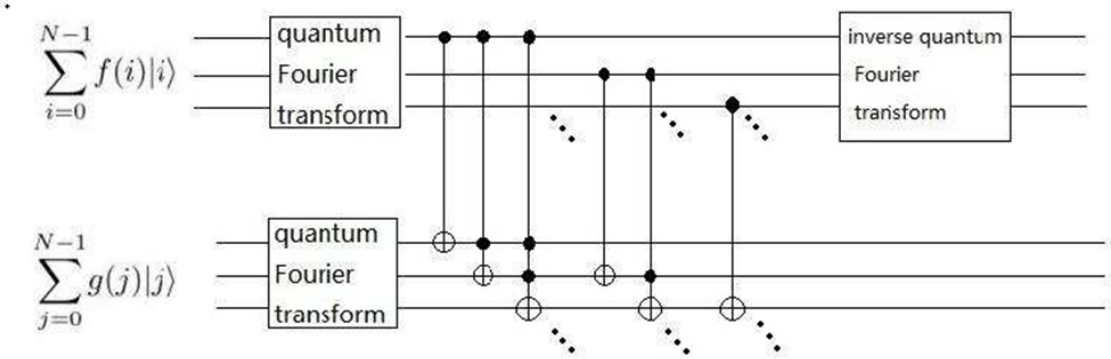
$$\sum_{i,j=0} f(i)g(j)\,|ij\rangle \rightarrow \sum_{i,j=0} f(i)g(i+j)\,|ij\rangle$$

Ta có mạch lượng tử có thể thực hiện được quá trình M



Hình 2: Mạch lượng tử của quá trình M

Kết hợp M với cách mạch lượng tử của QFT và IQFT nêu trên, ta thu được một mạch có thể thực hiện công việc tính tích chập [5].



Hình 3: Mạch lượng tử tính tích chập

Kết quả thu được sau khi chạy mạch lượng tử trên là:

$$\sum_{i,j=0}^{N-1} f(j)g(j)\,|ij\rangle \rightarrow \frac{1}{N\sqrt{N}} \sum_{w,k,t,j,l=0}^{N-1} f(t)g(j)e^{2\pi i(kt-kw+kj+jl)}\,|wl\rangle$$

Mặt khác, ta có thể viết tích chập lượng tử của f và g dưới dạng:

$$\lambda \sum_{w,j=0}^{N-1} f(t)g(w-t)\,|w\rangle = \frac{\lambda}{N} \sum_{w,k,t,j=0}^{N-1} f(t)g(j)e^{2\pi i(kt-kw+kj)}\,|w\rangle$$

Ta có thể kiểm chứng điều này bằng cách áp dụng lần lượt QFT và IQFT lên vế trái

So sánh hai biểu thức trên, ta thấy ta có tích chập (đã được chuẩn hóa) của f và g là $r[n]$ ở phần *biên độ* của trạng thái riêng ứng với $w = n$ và $l = 0$. Ví dụ, ta có giá trị của phần tử có index $= 0$ của r tại biên độ của trạng thái riêng ứng với $w = 0$ và $l = 0$. Ta biểu diễn lại trạng thái sau khi chạy qua mạch.

$$|\rho\rangle = r[0]\,|w = 0, l = 0\rangle + r[1]\,|w = 1, l = 0\rangle + ... + r[N-1]\,|w = N-1, l = 0\rangle + \rho_0\,|w = 0, l = 1\rangle + ...$$

Thuật toán tính tích chập bằng QFT ở trên có độ phức tạp là $O(n^2)$ với $n = log_2(MN)$ (n chính là số qubit). Ta thấy thuật toán sử dụng QFT có độ phức tạp nhỏ hơn thuật toán FFT. Vì vậy, trên lý thuyết, ta có thể cải thiện hiệu suất xử lý bằng sử dụng thuật toán tính tích chập bằng QFT.

Tuy nhiên, trên thực tế, một trong những vấn đề với phương pháp tính tích chập ở trên chính là việc kết quả nằm ở phần biên độ của trạng thái qubit. Đây là yếu tố ta không thể trực tiếp thực hiện đo đạc được mà phải sử dụng đến một số kỹ thuật thuật như quantum tomography hay ước lượng biên độ để thu được kết quả. Các tác vụ này có thể gây thêm chi phí trong quá trình tính toán, cũng như khiến các sai số xuất hiện làm giảm độ chính xác của giải thuật. Ở phần giả lập kết quả, ta sẽ giới thiệu một kỹ thuật ước lượng biên độ phù hợp để có thể thu được kết quả mong muốn.

3 Tiền xử lí dữ liệu

3.1 Xử lí dữ liệu đầu vào cổ điển

Chương trình sử dụng 1 phần file "mnist.csv" (200 mẫu) làm dữ liệu đầu vào.
Có bao gồm 2 hàm dùng để xử lí dữ liệu đầu vào cổ điển:

```
def LoadData(file_path):
    data = pd.read_csv(file_path)
    y_data = data.iloc[:, 0].values
    x_data = data.iloc[:, 1:].values
    x_data = tf.convert_to_tensor(x_data, dtype=tf.float32)
    y_data = tf.convert_to_tensor(y_data, dtype=tf.int32)
    return x_data, y_data

def SplitData(x_data, y_data, train_percent_size):
    train_size = int(len(x_data) * train_percent_size)
    x_train = x_data[:train_size]
    y_train = y_data[:train_size]
    x_test = x_data[train_size:]
    y_test = y_data[train_size:]
    return x_train, y_train, x_test, y_test
```

- Đầu tiên, hàm `LoadData()` sẽ lấy dữ liệu từ file và chia thành hai biến x_data (dữ liệu ảnh) và y_data (nhãn).
- Tiếp theo, hàm `SplitData()` sẽ tách dữ liệu ra làm hai phần (cho huấn luyện và kiểm tra) dựa theo trọng số yêu cầu.

3.2 Mã hóa biên độ (Amplitude Embedding)

Để đưa các dữ liệu đầu vào vào Quantum Circuit ta phải mã hóa dữ liệu sang các dữ liệu trạng thái của các Qubit.
Dưới đây là các hàm hỗ trợ và hàm mã hóa biên độ:

```
def pad_to_power_of_two(arr):
    """
    Pad the array to the nearest power of 2 length.
    """
    target_length = 2**int(np.ceil(np.log2(len(arr))))
    return np.pad(arr, (0, target_length - len(arr)), 'constant')

def amplitude_embedding(x_train, qubits):
    """
    Encode classical data into quantum states using amplitude embedding.
    Args:
        x_train: Classical data to be encoded.
        qubits: List of qubits to encode the data.
    """
    circuit = cirq.Circuit()
    for sample in x_train:
        # Normalize the sample to create a valid quantum state
```

```
        sample = pad_to_power_of_two(sample)
        norm = np.linalg.norm(sample)
        normalized_sample = sample / norm if norm != 0 else sample
        circuit.append(cirq.StatePreparationChannel(normalized_sample)(*qubits))
    return circuit

def qubits_data(x_train):
    # Assuming the number of qubits needed is the same as the length of each sample
    num_qubits = int(np.ceil(np.log2(len(x_train[0]))))
    return [cirq.GridQubit(0, i) for i in range(num_qubits)]

def n_qubits(x_train):
    return int(np.ceil(np.log2(len(x_train[0]))))
```

Giải thích các chức năng hàm:

- **pad_to_power_of_two()**:

Về cơ bản, kích thước ảnh của mẫu thường là 28x28 (784 điểm dữ liệu) và nó làm cho kết quả tính số lượng Qubit không phải là số nguyên ($\log_2(784) \approx 9.61$) và dẫn tới lỗi. Vì thế hàm này sẽ làm cho mảng có độ dài bằng với lũy thừa gần nhất của 2.

- **amplitude_embedding()**:

Đây là hàm chính để mã hóa dữ liệu từ cổ điển sang trạng thái lượng tử theo công thức

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

- **qubits_data()**:

Dùng để lấy danh sách các qubit được dùng để mã hóa dữ liệu.

- **n_qubits()**:

Trả về số lượng Qubit được sử đụng để mã hóa dữ liệu.

4 Hiện thực lớp Quantum Fourier Convolutional

Flow giải thuật:

- Đầu vào cho lớp sẽ là các *quantum state* đã có được ở bước mã hóa dữ liệu. Bao gồm cả state của hình ảnh (*input state*) và kernel (với kiểu dữ liệu có thể train được).
- Tiếp theo, ta chuyển đổi các state sang miền tần số thông qua giải thuật Quantum Fourier Transform (QFT). QFT giúp ánh xạ các dữ liệu từ miền không gian (*spatial domain*) sang miền tần số (*frequency domain*).
- Sau đó, ta thực hiện các phép tính chập tương tác giữa input và kernel trong miền tần số. Tương tác này được hiện thực thông qua cổng điều khiển R_k, với việc thêm tham số để điều chỉnh kernel trainable.
- Cuối cùng, ta áp dụng giải thuật Inverse Quantum Fourier Transform (iQFT) để đưa các state từ miền tần số quay trở lại miền không gian. Các state kết quả này sẽ được sử dụng ở bước đo đạc (*measurement*) để thu được kết quả cuối cùng.

4.1 Các thành phần chính

4.1.1 Hàm QFT và iQFT

```python
def qft(qubits):
    """Quantum Fourier Transform on the given list of qubits."""
    circuit = cirq.Circuit()

    n = len(qubits)
    for i in range(n):
        circuit.append(cirq.H(qubits[i]))
        for j in range(i + 1, n):
            angle = np.pi / (2 ** (j - i))
            circuit.append(cirq.CZ(qubits[j], qubits[i]) ** (angle / np.pi))
    return circuit


def inverse_qft(qubits):
    circuit = cirq.Circuit()
    n = len(qubits)
    for i in reversed(range(n)):
        for j in reversed(range(i + 1, n)):
            angle = -np.pi / (2 ** (j - i))
            circuit.append(cirq.CZ(qubits[j], qubits[i]) ** (angle / np.pi))
        circuit.append(cirq.H(qubits[i]))
    return circuit
```

Hàm QFT và iQFT được xây dựng dựa trên:

- Cổng Hadamard (H): Cổng này tạo trạng thái chồng chất (*superposition*) cho từng qubit.
- Cổng điều khiển pha (CZ hoặc R_k): Được sử dụng để thêm các pha điều khiển giữa các qubit, giúp hiện thực hóa các góc quay cần thiết. Góc quay (*angle*) cho CZ được xác định

bởi khoảng cách giữa các qubit, với công thức:

$$\text{angle} = \frac{\pi}{2^{j-i}}$$

Cụ thể:

- Trong QFT, các cổng H và CZ được áp dụng lần lượt từ qubit thấp đến qubit cao, thêm các pha tương ứng.
- Trong iQFT, thứ tự các cổng được áp dụng ngược lại (từ qubit cao đến thấp) và các góc quay được đổi dấu (*negative phase shift*).

4.1.2 Hàm ánh xạ trong miền tần số (Mapping)

```
def mapping_m(input_qubits, kernel_qubits, params):
    circuit = cirq.Circuit()
    for i_q, k_q, param in zip(input_qubits, kernel_qubits, params):
        # Replace CZ with parameterized controlled-RY gates
        circuit.append(cirq.ry(param).controlled()(i_q, k_q))
    return circuit
```

Hàm ánh xạ (*Mapping Function*) thực hiện phép nhân tương ứng giữa các trạng thái lượng tử (*quantum states*) của input và kernel trong miền tần số. Điều này mô phỏng phép tích chập (*convolution*) bằng cách:

- Áp dụng các cổng xoay (RY) có tham số lên các qubit kernel. Các tham số này có thể được huấn luyện (*trainable parameters*).
- Sử dụng các cổng điều khiển xoay (*controlled RY*) để kết hợp giữa input và kernel. Điều này cho phép kernel tương tác với input theo cách train và tối ưu.
- Từng cặp qubit (một từ input, một từ kernel) được ánh xạ thông qua các cổng này.

4.1.3 Hàm iQFT

Cuối cùng, hàm iQFT được thực thi để đưa trạng thái lượng tử từ miền tần số quay trở lại miền không gian. Hàm này đảm bảo rằng các trạng thái kết quả có thể được đo (*measured*) ở bước kế tiếp. Quá trình đo giúp thu thập kết quả tích chập và chuyển đổi chúng sang định dạng cổ điển để xử lý thêm.

4.2 Những vấn đề trong cách hiện thực

Mặc dù giải thuật trên là đúng về mặt lý thuyết, nhưng việc hiện thực hóa lớp tích chập lượng tử hiện tại gặp rất nhiều vấn đề, cụ thể:

- **Tăng trưởng theo hàm mũ (Exponential Growth)**: - Để giả lập một hệ có n qubit, ta cần một vector trạng thái với 2^n số phức. Điều này khiến việc lưu trữ và xử lý dữ liệu trở nên không khả thi khi n lớn (ví dụ, $n \geq 20$).
- **Tốn kém trong bước tính toán**: Mỗi cổng lượng tử được biểu diễn bằng một ma trận $2^n \times 2^n$, và việc nhân ma trận này với vector trạng thái có độ phức tạp $O(2^{2n})$. Điều này làm tăng nhanh thời gian xử lý với các hệ phức tạp.

- **Hạn chế của bộ giả lập**: Bộ giả lập Cirq và các công cụ tương tự phải tải toàn bộ trạng thái lượng tử vào bộ nhớ mỗi khi thực thi một phép tính, dẫn đến việc tiêu tốn tài nguyên (RAM) rất lớn. Điều này gây khó khăn cho việc chạy các mạch phức tạp trên phần cứng thông thường.
- **Khả năng thực thi thực tế**: Việc giả lập lớp QFCNN hiện tại chỉ mang tính lý thuyết và minh họa. Để ứng dụng thực tế, cần các hệ thống máy tính lượng tử với số lượng qubit và độ ổn định cao hơn.

5 Kết quả chương trình theo dữ liệu mẫu

5.1 Dữ liệu đầu vào

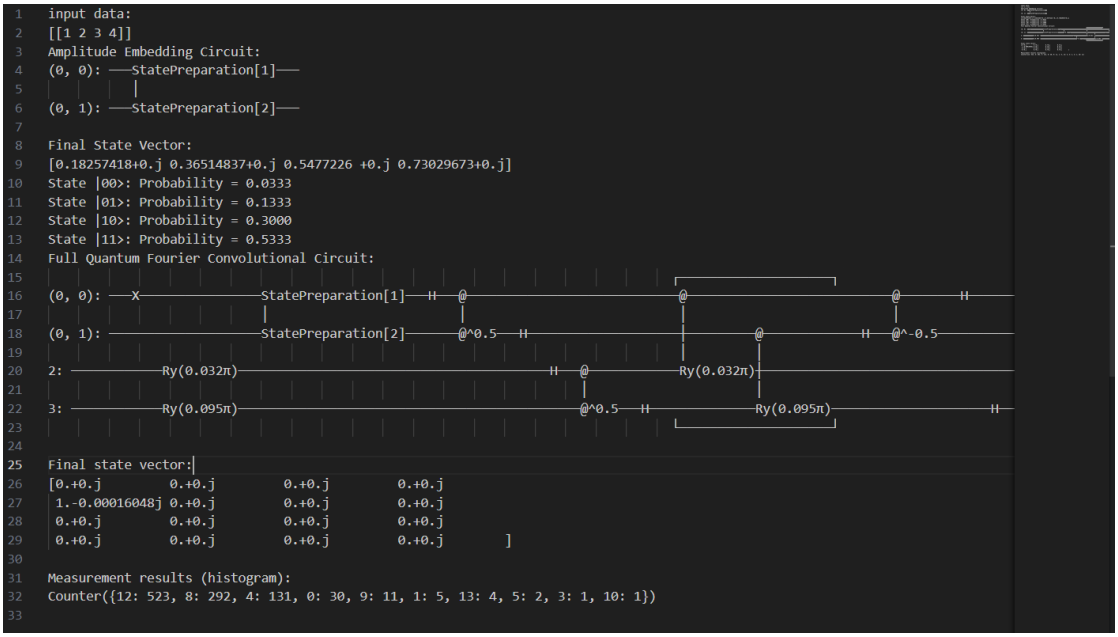
Theo lý thuyết, chương trình sẽ có thể xử lí được mọi kích thước dữ liệu nhưng theo thực tế, sử dụng lượng lớn Qubit gây tiêu tốn rất lớn tài nguyên (kiểu dữ liệu ảnh "Mnist.csv" 28x28 yêu cầu hệ thống tối thiểu 16 GB Ram). Nên để xác nhận tính đúng đắn của mô hình và dễ dàng quan sát kết quả, bài báo cáo sẽ sử dụng 1 vector mẫu có 4 dữ liệu (tương tự ảnh 2x2) như sau:

$$\mathbf{X} = [1, 2, 3, 4]$$

Sở dĩ bài báo cáo sử dụng vector dữ liệu nhỏ, như vector $\mathbf{X}$, là vì nó chỉ đòi hỏi tối thiểu 2 qubit. Điều này thuận lợi hơn cho việc minh họa thiết kế mạch lượng tử (Quantum Circuit) và quan sát các trạng thái kết quả.

5.2 Kết quả chương trình

Dưới đây là kết quả của chương trình theo từng bước bao gồm:
- Sau khi mã hóa biên độ (Amplitude Embedding).
- Sau khi thiết kế xong mạch lượng tử (kết quả thiết kết mạch).
- Sau khi cho dữ liệu đi qua mạch lượng tử.



Hình 4: Các kết quả chi tiết sau khi thực hiện xong từng quá trình.

Theo như kết quả, ta thấy dữ liệu đầu vào chỉ cần 2 Qubit để lưu trữ trạng thái nhưng mô hình sẽ cần tới 4 Qubit để xử lí dữ liệu. Như vậy với **n** Qubit dữ liệu đầu vào sẽ cần tới **2n** Qubit cho cả mạch gây tiêu tốn rất lớn tài nguyên (1 tấm ảnh 28x28 chỉ cần 10 qubit để lưu trữ nhưng cần tới 20 qubit để xử lí). Đây cùng là hạn chế lớn nhất khi cố gắng mô phỏng thuật toán lượng tử trên các máy tính cổ điển hiện tại.

6 Giả lập kết quả

Bởi vì việc sử dụng các quantum simulator tiêu tốn rất nhiều tài nguyên, việc giả lập một mạng QFCNN có thể huấn luyện được và cho ra kết quả trên máy tính cá nhân là một việc rất khó khăn. Vì vậy, ở phần này, lớp tích chập của QFCNN sẽ được giả lập bằng cách tính tích chập bằng thuật toán cổ điển kết hợp với tham số độ nhiễu xuất hiện trong quá trình đo đạc. Việc hiện thực được dựa trên mạng QCNN được xây dựng trong [2].

6.1 Ước lượng biên độ (Amplitude Estimation)

Bởi vì kết quả tính tích chập từ mạch lượng tử xây dựng được ở các phần trước nằm ở phần biên độ của các trạng thái riêng, ta cần một phương pháp ước lượng biên độ để có thể thu được kết quả. Ở đây ta sử dụng phương pháp được giới thiệu trong [3].

Định lý 2 *(Ước lượng tham số) Cho một thuật toán lượng tử:*

$$A : |0\rangle \rightarrow \sqrt{p}\,|y, 1\rangle + \sqrt{1-p}\,|G, 0\rangle$$

Trong đó $|G\rangle$ là một trạng thái rác bất kì, khi đó với một số nguyên M bất kì, thuật toán ước lượng biên độ cho ra giá trị $\tilde{p}$ sao cho

$$|\,\tilde{p} - p\,| \leq 2\pi \frac{\sqrt{p(1-p)}}{M} + (\frac{\pi}{M})^2$$

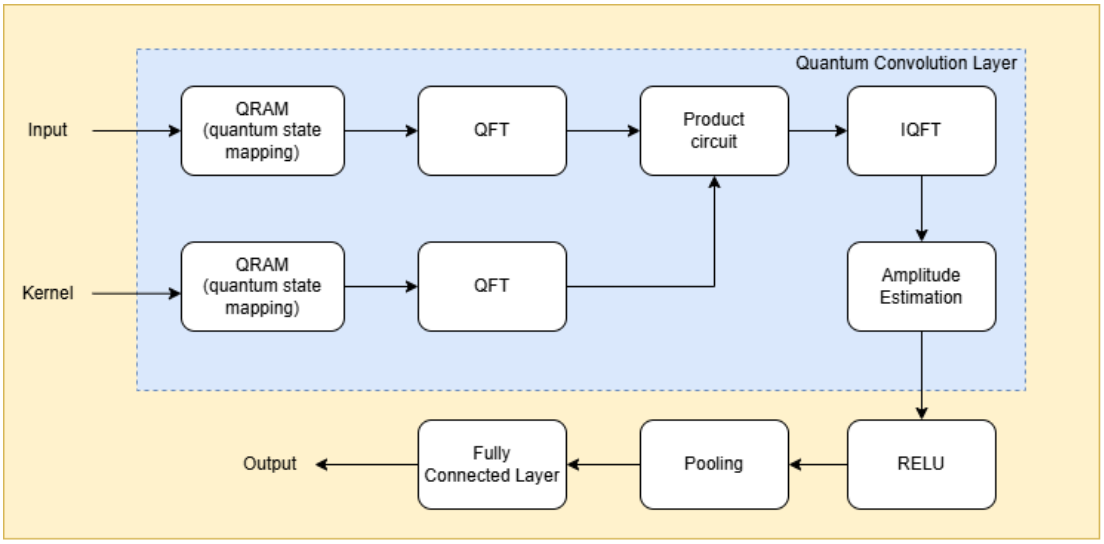
với xác suất nhỏ nhất là $\frac{8}{\pi^2}$. Thuật toán sự dụng M lần lặp của thuật toán A. Nếu $p = 0$ thì chắc chắn $\tilde{p} = 0$. Nếu $p = 1$ và p chẵn thì chắc chắn $\tilde{p} = 1$.

Bởi vì ở phần này ta chỉ tiến hành numerical simulation, ta không quan tâm đến chi tiết mạch lượng tử cũng như các bước thực hiện của giải thuật trên mà chỉ tập trung vào mô phỏng kết quả có thể thu được sau khi thực hiện giải thuật. Tuy nhiên, nhìn vào biểu thức sai số, ta có thể thấy rằng để đạt được độ nhiễu thấp, ta sẽ phải thực hiện đo

Dựa vào định lý trên, ta thêm tham số nhiễu với phân phối Gaussian có độ lệch chuẩn ϵ.

```
1  #Add gaussian noise to simulate the amplitude estimation step
2  def add_gaussian_noise(self, input_tensor, epsilon):
3      noise = torch.Tensor(input_tensor.data.new(input_tensor.size()).normal_(0,
           epsilon))
4      output_tensor = input_tensor + 2*torch.mul(noise, torch.max(input_tensor *
           (1-input_tensor)), out=None) #2*noise*max(sqrt(p*(1-p)))
5      output_tensor = torch.clamp(output_tensor, min=0, max=Cap) #no value should be
           less than 0 or more than Cap!
6      return output_tensor
```

Bởi vì đây là một hệ thống hybrid, sau khi thu được kết quả tính tích chập, ta sẽ thực hiện các lớp còn lại của mạng CNN bằng máy tính cổ điển, sử dụng các framework học máy có sẵn như PyTorch hay TensorFlow.



Hình 5: Sơ đồ hệ thống hybrid. Phần màu xanh được thực hiện trên máy tính lượng tử, phần màu vàng được thực hiện trên máy tính cổ điển.

6.2 Hiện thực và kết quả

Ta sử dụng framework PyTorch để giả lập mạng QFCNN dựa trên source code từ **??**. Ta thử nghiệm trên bộ dữ liệu MNIST với 60000 ảnh huấn luyện và 10000 ảnh kiểm thử. Hệ số nhiễu của quá trình chuẩn bị trạng thái và đo đạc là $\epsilon = 0.01$. Hệ số Cap sử dụng trong CapRELU là 10.

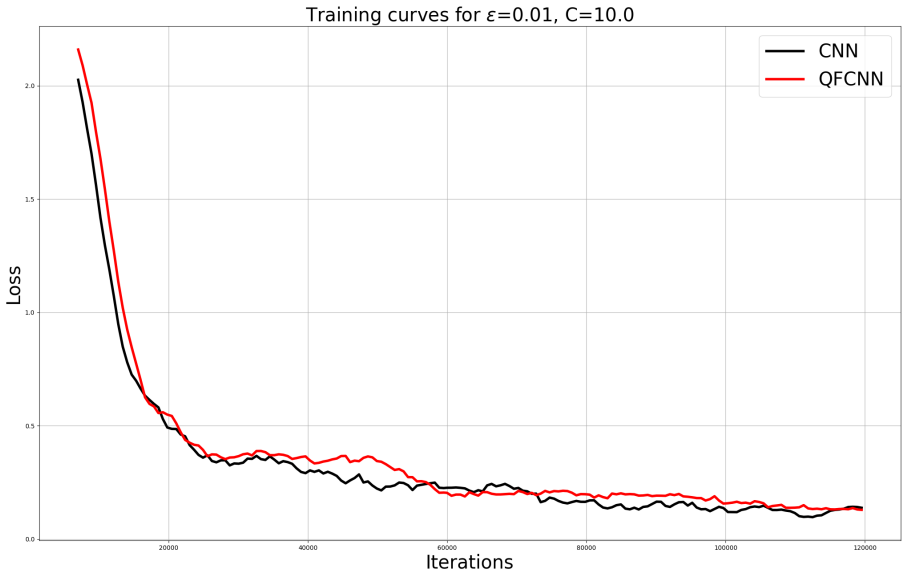
Kết quả cho thấy mạng QFCNN đạt được độ chính xác cao hơn mạng QCNN sử dụng trong **??**. Điều này có thể được lí giải bằng việc hệ thống hybrid của ta chỉ có lớp tích chập là được tính toán bằng hệ thống lượng tử, trong khi mạng QCNN được thiết kế để thực hiện cả bước RELU và back propagation trên máy tính lượng tử, từ đó dẫn đến việc xuất hiện nhiều yếu tố nhiễu hơn.

Từ training curve thu được, ta thấy không có sự khác biệt nhiều giữa độ chính xác giữa QFCNN và CNN trong môi trường giả lập mà ta đã xây dựng. Bởi vì phép tính tích chập bằng QFT có độ phức tạp thời gian nhỏ hơn phép tính tích chập thông thường đáng kể, đặc biệt là với kích cỡ kernel lớn, về mặt lý thuyết, mạng QFCNN sẽ có tốc độ xử lý tốt hơn mạng CNN cổ điển.

Tuy nhiên, ta cũng cần phải xét đến thời gian thực hiện quá trình chuyển dữ liệu xử lý từ môi trường lượng tử sang cổ điển và ngược lại. Hiện tại, chỉ bằng việc giả lập bằng máy tính cổ điển, ta không thể kiểm chứng độ trễ gây ra bởi các giai đoạn này.

Ta có bảng so sánh độ chính xác giữa CNN, QCNN và QFCNN.

	Lost	Accuracy
CNN	0.116	96,2%
QCNN	0.142	95.4%
QFCNN	0.121	96,1%



Hình 6: Training curve của CNN và QFCNN qua 2 epochs (120000 iterations)

7 Kết luận

Báo cáo đã hoàn thành việc trình bày các vấn đề liên quan đến việc tìm hiểu và hiện thực QFCNN theo bài báo gốc. Trong quá trình thực hiện, chúng em đã có cơ hội tìm hiểu những kiến thức tổng quát liên quan đến tính toán lượng tử, bao gồm các thành phần chính của một hệ thống lượng tử, cũng như cách sử dụng một quantum simulator để hiện thực một mạch lượng tử.

Về những phần kiến thức mang tính cụ thể hơn, chúng em đã tìm hiểu về phép biến đổi Fourier, tích chập cũng như mạng CNN, từ đó đi đến phiên bản lượng tử của chúng, bằng cả hai phương pháp là sử dụng simulator để giả lập mạch lượng tử cũng như numerical simulation để thu được kết quả và ước lượng độ chính xác của một mạng QFCNN hoàn chỉnh.

Bên cạnh đó, qua quá trình tìm hiểu và sử dụng quantum simulator, cụ thể là Cirq, chúng em rút ra kết luận rằng việc giả lập một mạch lượng tử phức tạp trên máy tính cổ điển tiêu tốn rất nhiều chi phí tính toán và bộ nhớ. Lượng chi phí này tăng theo hàm mũ so với số qubit. Vấn đề trên có thể phần nào được khắc phục bằng các kỹ thuật tối ưu hóa cũng như circuit cutting để làm giảm độ phức tạp của mạch lượng tử. Tuy nhiên, những phần này không nằm trong giới hạn đề tài của đồ án này. Vì vậy, chúng em minh họa khả năng tính tích chập của mạch được trình bày bằng các vector có kích cỡ tương đối nhỏ.

Qua báo cáo, ta có thể thấy rằng mạng QFCNN trên lý thuyết có thể hiện thực được. Với kiến trúc hybrid tận dụng cả tính toán cổ điển lẫn tính toán lượng tử, QFCNN đạt được độ chính xác cao hơn một mạng QCNN được hiện thực thuần túy bằng tính toán lượng tử. Tuy vậy, nó vẫn còn một số khuyết điểm như chi phí tăng thêm cho quá trình mã hóa từ môi trường cổ điển sang môi trường lượng tử cũng như đo đạt kết quả từ môi trường lượng tử sang môi trường cổ điển. Vì vậy, dựa trên những hiểu biết của chúng em về tài nguyên phần cứng của các máy tính lượng tử hiện tại, chúng em đánh giá rằng giá trị thực tiễn của kiến trúc QFCNN không quá cao nếu so với các mạng QCNN khác.

Tài liệu

[1] F. Shen and J. Liu. "Quantum Fourier Convolutional Network." *ACM Transactions on Multimedia Computing, Communications and Applications* 19.1 (2023): 1-14.

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

[3] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, "Quantum amplitude amplification and estimation," *Contemporary Mathematics,* vol. 305, pp. 5374, 2002.

[4] C. Lomont, "Quantum convolution and quantum correlation algorithms are physically impossible," *arXiv preprint quantph/0309070,* 2003.

[5] G. Ma, H. Li, and J. Zhaom "Windowed fourier transform and general wavelet algorithms in quantum computation," *Quantum Inf. Comput.,* 19(3&4):237–251, 2019.

[6] Michael A Nielsen and Isaac Chuang. *Quantum computation and quantum information,* 2002