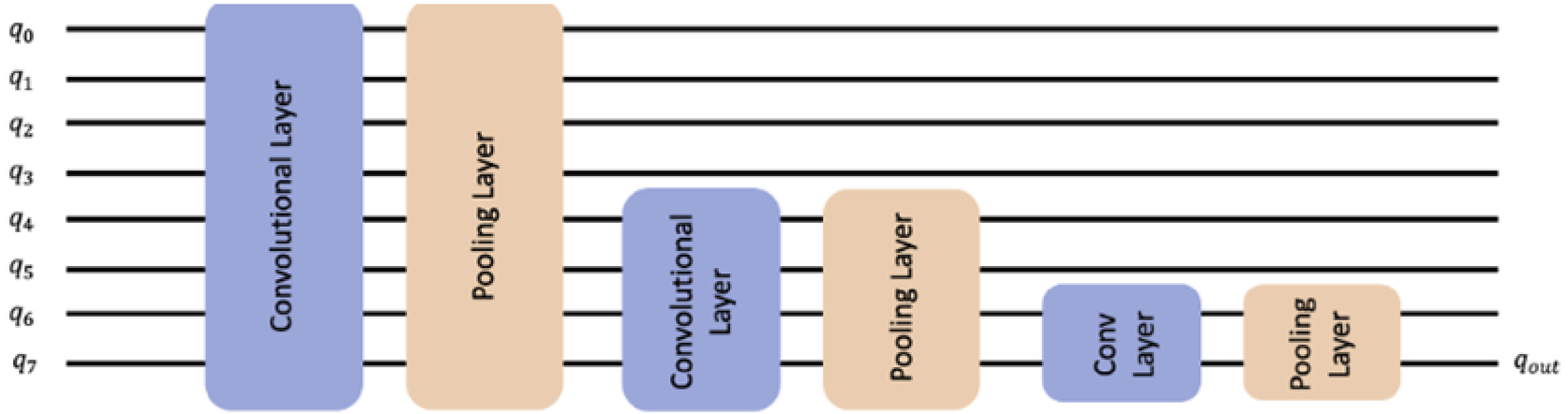
IMPLEMENTAZIONE ED ANALISI DI UNA QUANTUM CONVOLUTIONAL NEURAL NETWORK PER LA CLASSIFICAZIONE DELLE IMMAGINI MNIST CON AMPLITUDE ENCODING

Progetto di Elena Azzi e Arianna Cella

CONTENTS:

QUANTUM CONVOLUTIONAL NEURAL NETWORK (QCNN)

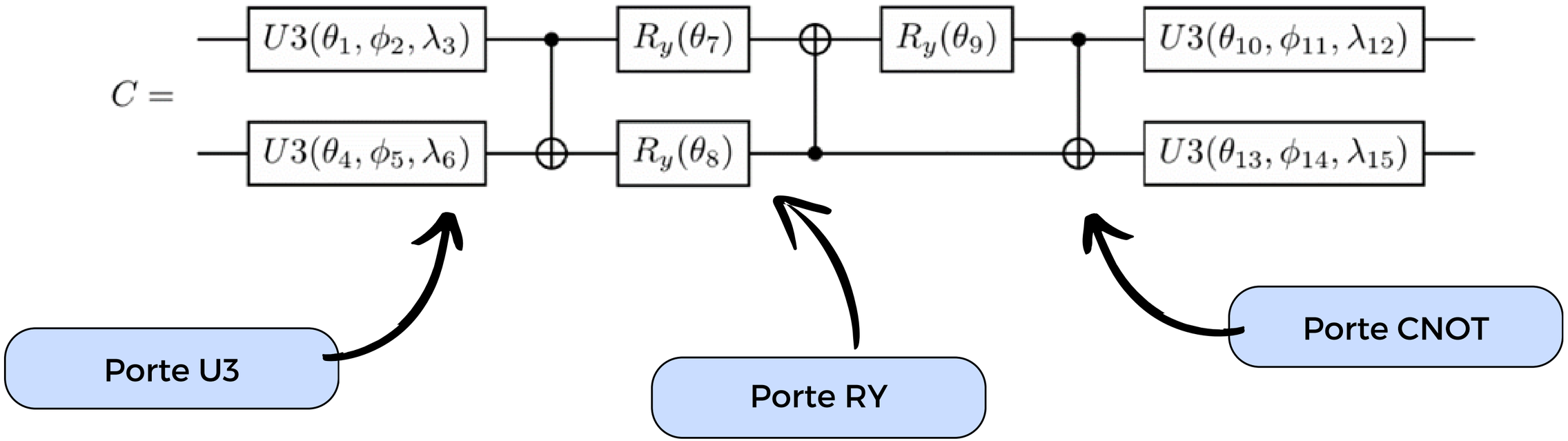
Le QCNN sono una classe di reti neurali quantistiche che combinano il deep learning con i principi della meccanica quantistica, impiegando circuiti quantistici per eseguire le operazioni di convoluzione e pooling.



Durante l'addestramento della QCNN, i parametri (ovvero gli angoli di rotazione) vengono ottimizzati per minimizzare la funzione di perdita della rete.

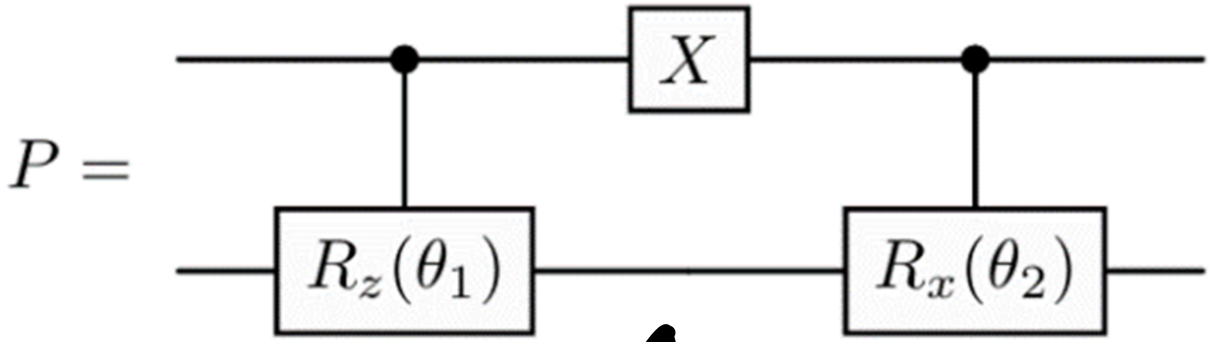
Circuito di Convoluzione quantistica (C)

Il Circuito di Convoluzione Quantistica (C) è responsabile dell'estrazione delle caratteristiche dai dati di input.



Circuito di Pooling (P)

Il Circuito P implementa l'operazione di pooling, che riduce la dimensionalità dei dati mantenendo le informazioni più rilevanti.



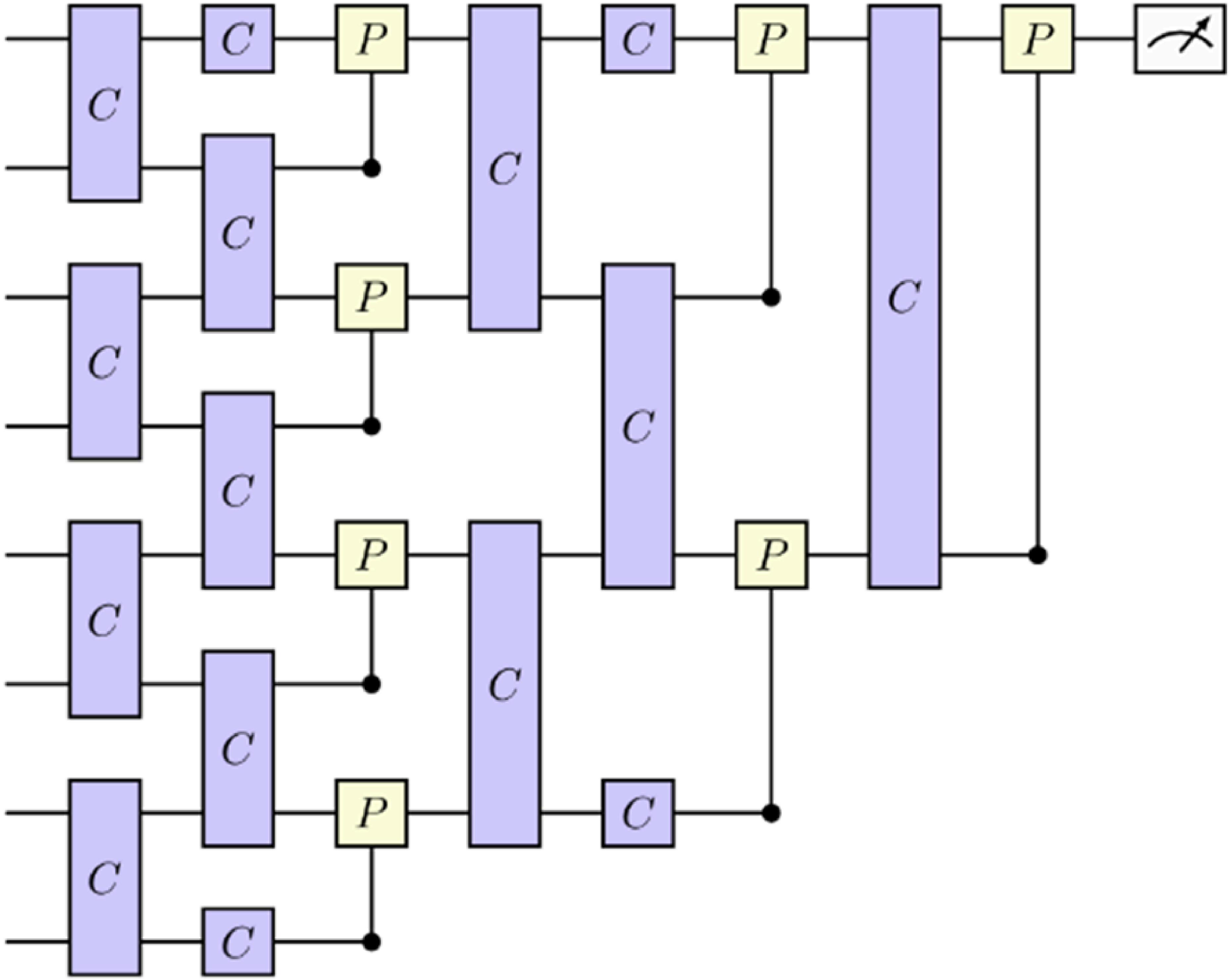
Porta CRZ

Porta Pauli-X

Porta CRX

CIRCUITO PRINCIPALE (8 QUBIT)

Il circuito QCNN si ottiene combinando il circuito C e il circuito P in una struttura gerarchica in grado di ridurre le dimensioni dei dati quantistici e conservare le informazioni rilevanti.



AMPLITUDE ENCODING

L'amplitude encoding è una tecnica di codifica dei dati in cui i valori dei dati classici vengono rappresentati come le ampiezze dei coefficienti di uno stato quantistico.

La funzione **qcnn_circuit** è stata creata per eseguire il circuito quantistico principale dopo aver effettuato l'**Amplitude Embedding** dei dati di input, che codifica le informazioni classiche nei qubit del circuito

```python
# Dispositivo quantistico per circuito principale
n_qubit = 8
dev = qml.device('default.qubit', wires=n_qubit)

@qml.qnode(dev)
def qcnn_circuit(theta, thetaP, phi, lamb, x):
    # Amplitude Encoding per inserire i dati nel circuito quantistico
    qml.AmplitudeEmbedding(x, wires=range(8), normalize=True)

    return circuit(theta, thetaP, phi, lamb), qml.state()
```

Questo passaggio è essenziale per preparare il sistema quantistico a eseguire le operazioni successive di convoluzione e pooling, che estrarranno le caratteristiche rilevanti per la classificazione.

PREPARAZIONE DEI DATI

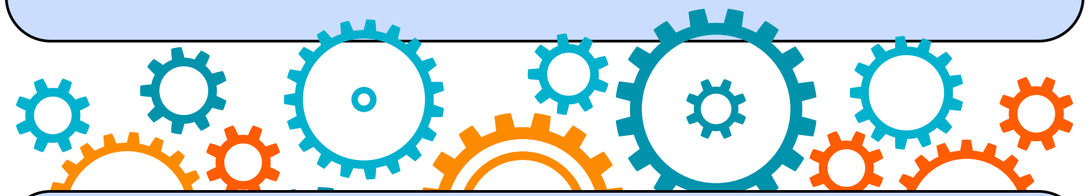
Abbiamo lavorato sul **dataset MNIST** che contiene un ampio numero di immagini di cifre scritte a mano da 0 a 9, ciascuna di dimensioni 28x28 pixel in scala di grigi.

```python
training_data = datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=transforms.Compose([
        transforms.Resize((16, 16)),    # Ridimensiona le immagini a 16x16
        transforms.ToTensor(),          # Converti le immagini in tensori
        transforms.Normalize((0.1307,), (0.3081,))  # Normalizza le immagini
    ])

)

# separa le immagini di 0 e 1
class_0 = [(img, label) for img, label in training_data if label == 0]
class_1 = [(img, label) for img, label in training_data if label == 1]

# numero massimo di campioni da selezionare per ogni classe
num_campioni = min(len(class_0), len(class_1), 1000)

# numero uguale di campioni da ciascuna classe
dati_bilanciati = class_0[:num_campioni] + class_1[:num_campioni]

np.random.shuffle(dati_bilanciati)

train_loader = torch.utils.data.DataLoader(dati_bilanciati, batch_size=64, shuffle=True)
```

1. Importare dataset MNIST
2. Ridimensionare immagini
3. Conversione in tensori
4. Normalizzazione tra 0 e 1
5. Filtraggio immagini di 0 e 1

ADDESTRAMENTO DEL MODELLO

In ogni epoca di addestramento, il modello **elabora** un batch di immagini del training set e **ottimizza** i parametri che rappresentano le rotazioni *(theta, thetaP, phi, lamb)* attraverso il metodo **step_and_cost**.



La funzione **cost_fn**, che rappresenta la *loss function*, utilizza il circuito quantistico per generare le previsioni del modello e applica la **cross_entropy** per valutare quanto queste previsioni si discostano dalle etichette reali.
Tale processo aiuta ad aggiornare i parametri del modello per minimizzare la perdita media su tutti i batch.

```
# ottimizzatore
opt = qml.AdamOptimizer(stepsize=0.01)


# ciclo di addestramento
for epoch in range(2):
    total_loss = 0
    for i, (images, labels) in enumerate(train_loader):
        images = images.view(-1, 16*16).numpy()  # vettore di lunghezza 256
        labels = labels.numpy()  # etichette in numpy array

        def cost_fn(theta, thetaP, phi, lamb):
            predictions = []
            for x in images:
                probs, _ = qcnn_circuit(theta, thetaP, phi,lamb, x)
                predictions.append(probs)
            return cross_entropy(labels, predictions, 2)

        # ottimizzazione
        (theta, thetaP, phi, lamb), loss = opt.step_and_cost(cost_fn, theta, thetaP, phi, lamb)

        total_loss += loss

    # media della perdita
    avg_loss = total_loss / len(train_loader)
    print(f'Epoca {epoch+1}: Loss = {avg_loss:.4f}')

print("Training completato.")
```

VALUTAZIONE DEL MODELLO

Per valutare le prestazioni del modello, è stata implementata la funzione **evaluate_model,** in grado di raccogliere le predizioni del modello confrontandole con le etichette corrette e analizzare come il modello classifica le immagini del test set.

Metriche utilizzate

- Accuratezza
- Matrice di Confusione
- Fedeltà quantistica

ACCURATEZZA E MATRICE DI CONFUSIONE

L**'accuratezza** fornisce una misura globale di quanto bene il modello sta performando su tutti i campioni del test set

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

La **matrice di confusione** è in grado di restituire una rappresentazione grafica immediata e dettagliata delle prestazioni del modello su ciascuna classe del dataset

		Predicted	
		0	**1**
Actual	**0**	TN	FP
	1	FN	TP

FEDELTÀ QUANTISTICA

La fedeltà è una misura di somiglianza tra due stati quantistici che restituisce un valore tra 0 e 1.

Essa fornisce un valore numerico che varia da 0 a 1 per indicare la ”vicinanza” tra gli stati:

- **1** : gli stati sono identici
- **0** : i due stati sono completamente distinti o ortogonali

$$F(\rho,\sigma) = \left(\mathrm{Tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)^2$$

UTILIZZATA PER:

1. Trovare lo stato ideale, ovvero lo stato quantistico che ha la minima distanza media rispetto a tutti gli altri stati
2. Misurare la distanza media degli stati quantistici classificati erroneamente e di quelli classificati correttamente rispetto allo stato ideale

ANALISI DEI RISULTATI

Per ogni esecuzione possiamo visualizzare:

Andamento della loss ad ogni epoca di addestramento

Accuratezza

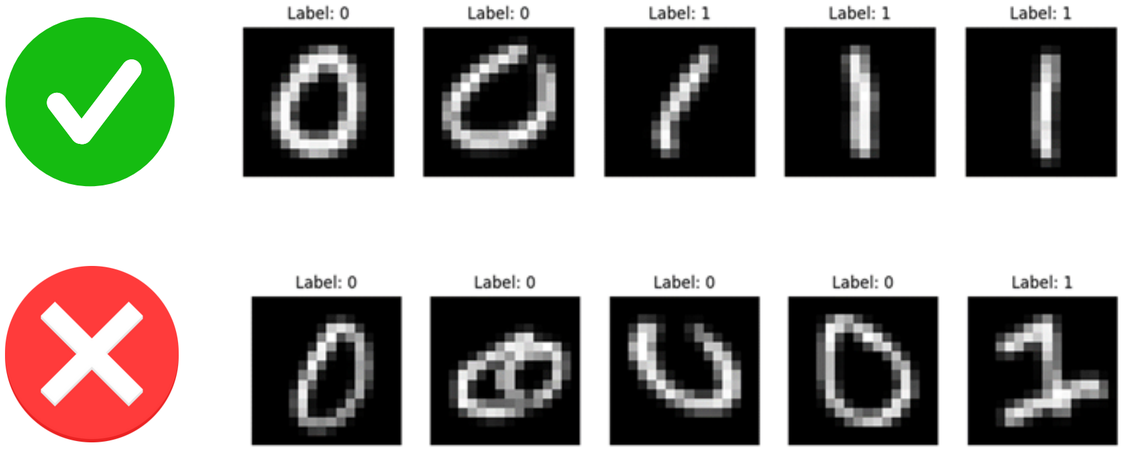
Matrice di confusione

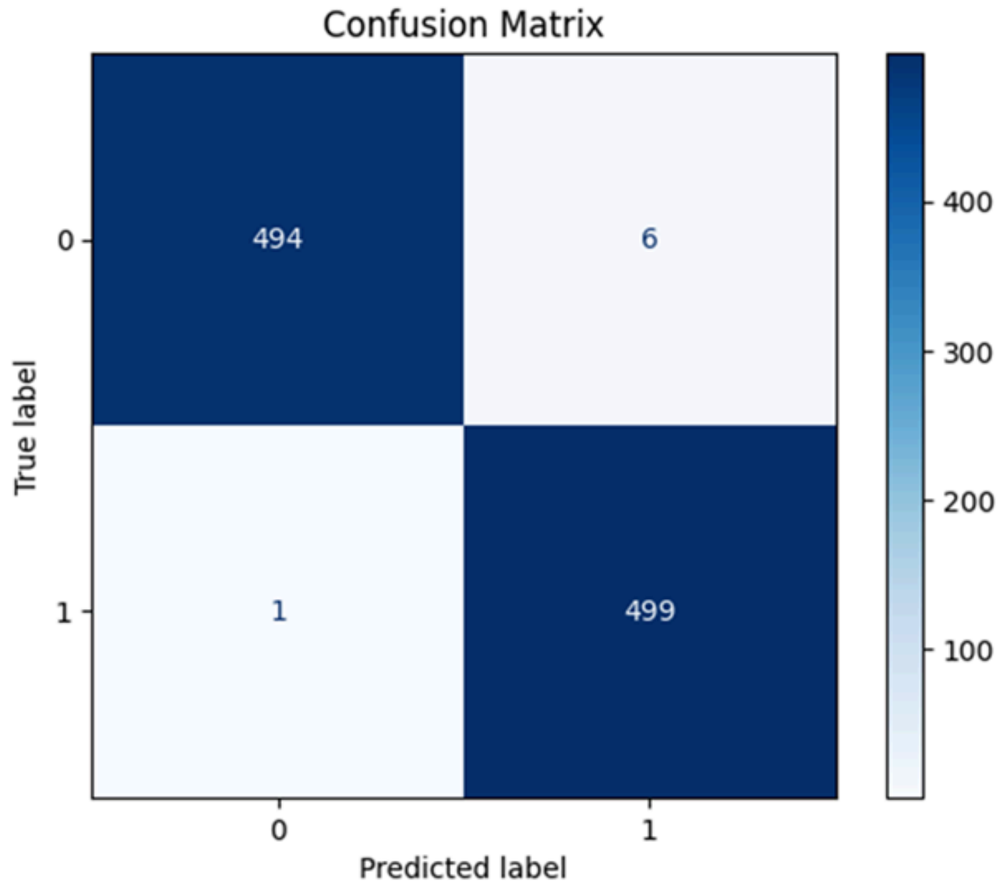
Visualizzazione di un istogramma della fedeltà di ciascuno stato con tutti gli altri

Visualizzazione delle immagini predette correttamente ed erroneamente

ANALISI DEI RISULTATI (2 EPOCHE DI ADDESTRAMENTO)

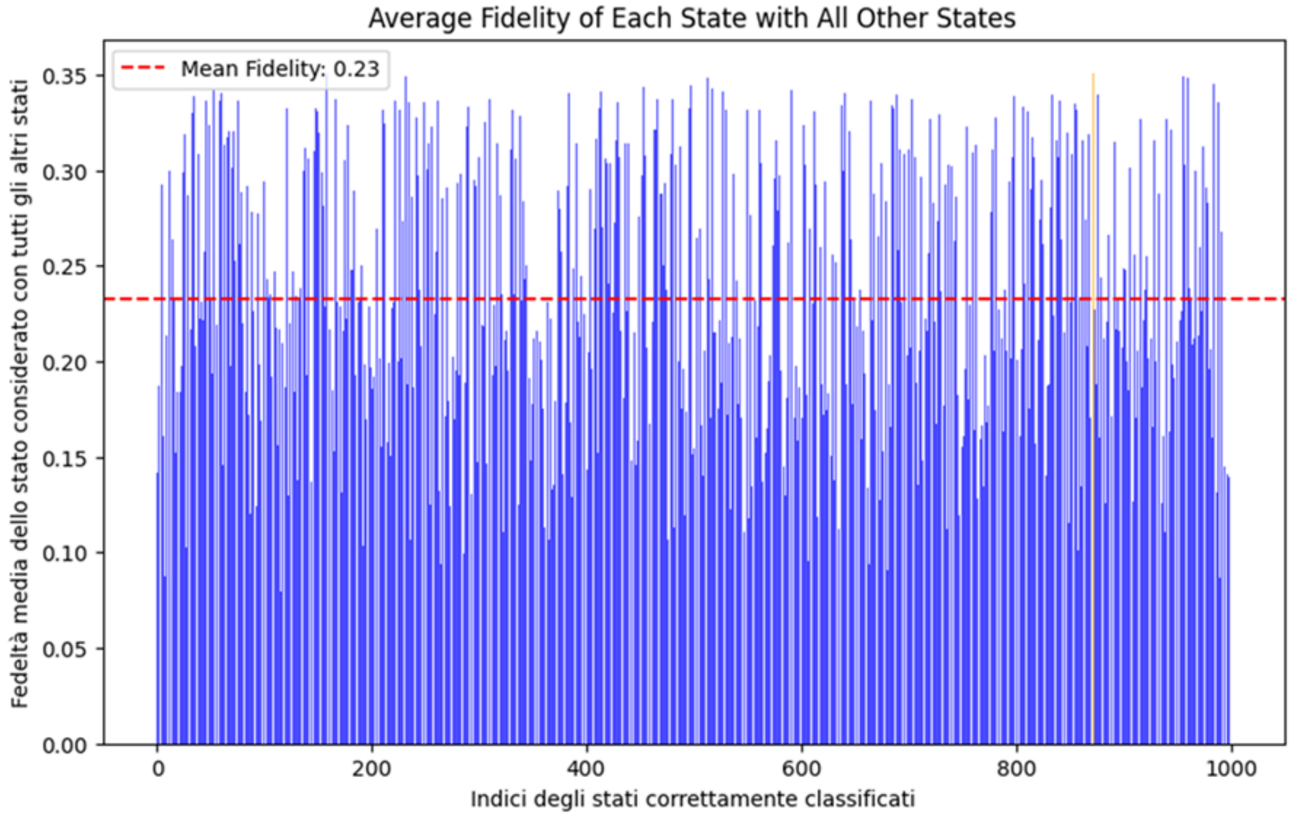
Epoca 1: Loss = 0.5863
Epoca 2: Loss = 0.5284
Accuratezza = 99.30%
7 classificate in modo sbagliato
0 sbagliati : 6
1 sbagliati : 1





ANALISI DEI RISULTATI (2 EPOCHE DI ADDESTRAMENTO)

max_fidelity: 0.3508998989578559
Fidelity media degli stati erroneamente classificati : 0.07799564102404624
Fidelity media degli stati correttamente classificati : 0.35347737138346713



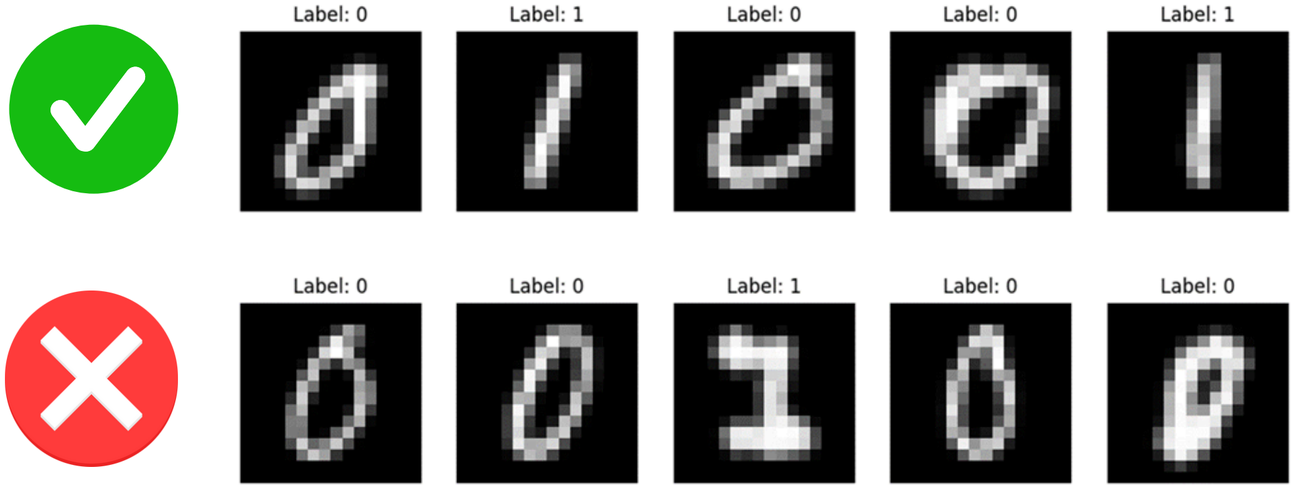
ANALISI DEI RISULTATI (2 EPOCHE DI ADDESTRAMENTO)

Run	Loss Finale	Accuracy (%)	TP and TN	FP (0 failed)	FN (1 failed)
Run 1	0.5284	99.30	993	6	1
Run 2	0.5116	98.00	980	17	3
Run 3	0.4820	99.20	992	5	3
Run 4	0.5485	98.70	987	12	1

Run	max Fidelity (Ideal State)	Avg Fidelity correctly classified	Avg Fidelity incorrectly classified
Run 1	0.3509	0.3535	0.0780
Run 2	0.3508	0.3575	0.0593
Run 3	0.3509	0.3529	0.1773
Run 4	0.3508	0.3543	0.1412

Tabelle riassuntive dei risultati ottenuti

ANALISI DEI RISULTATI (1 EPOCA DI ADDESTRAMENTO)



Epoca 1: Loss = 0.6078
Accuracy = 98.10%
19 classificate in modo sbagliato
0 sbagliati : 18
1 sbagliati : 1

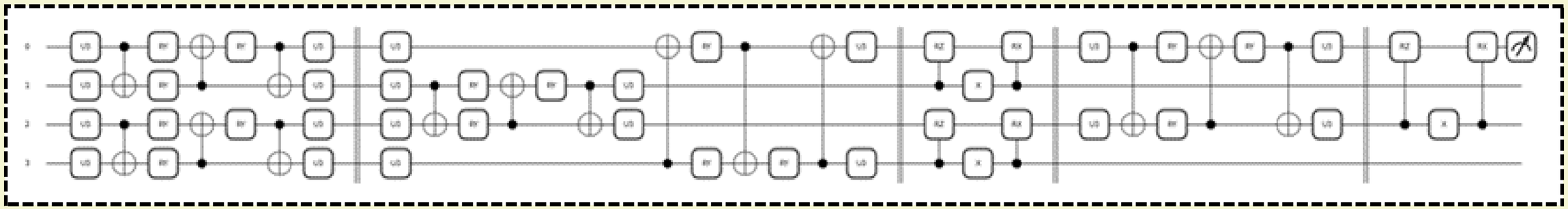
max_fidelity: 0.3508998989578592
Fidelity media degli stati erroneamente classificati : 0.1395674819583596
Fidelity media degli stati correttamente classificati : 0.3556546555695673

MODELLO A 4 QUBIT

Vengono eliminati il primo layer di convoluzione e di pooling per adattare l'architettura alla riduzione del numero di qubit.

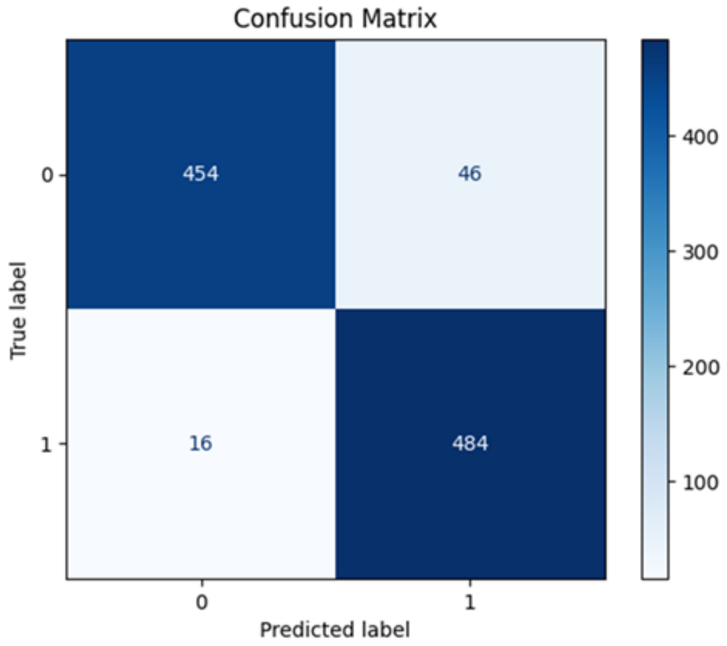
Le semplificazioni rispetto al circuito a 8 qubit hanno permesso l'implementazione di un'architettura più compatta.

```python
# Circuito quantistico principale 4 qubit
def circuit(theta, thetaP, phi, lamb):

    for i in range(0, n_qubit, 2):
        circuitC(theta[0:7], phi[0:4], lamb[0:4],[i,i+1])

    qml.Barrier(wires=range(n_qubit))

    for i in range(1, n_qubit, 2):
        circuitC(theta[0:7], phi[0:4], lamb[0:4],[i,((i+1) % n_qubit)])

    qml.Barrier(wires=range(n_qubit))

    for i in range(0, n_qubit, 2):
        circuitP(thetaP[0:2], i+1, i)

    qml.Barrier(wires=range(n_qubit))

    circuitC(theta[7:14], phi[4:8], lamb[4:8],[0,2])

    qml.Barrier(wires=range(n_qubit))

    circuitP(thetaP[2:4], 2, 0)

    return qml.probs(wires=0)
```

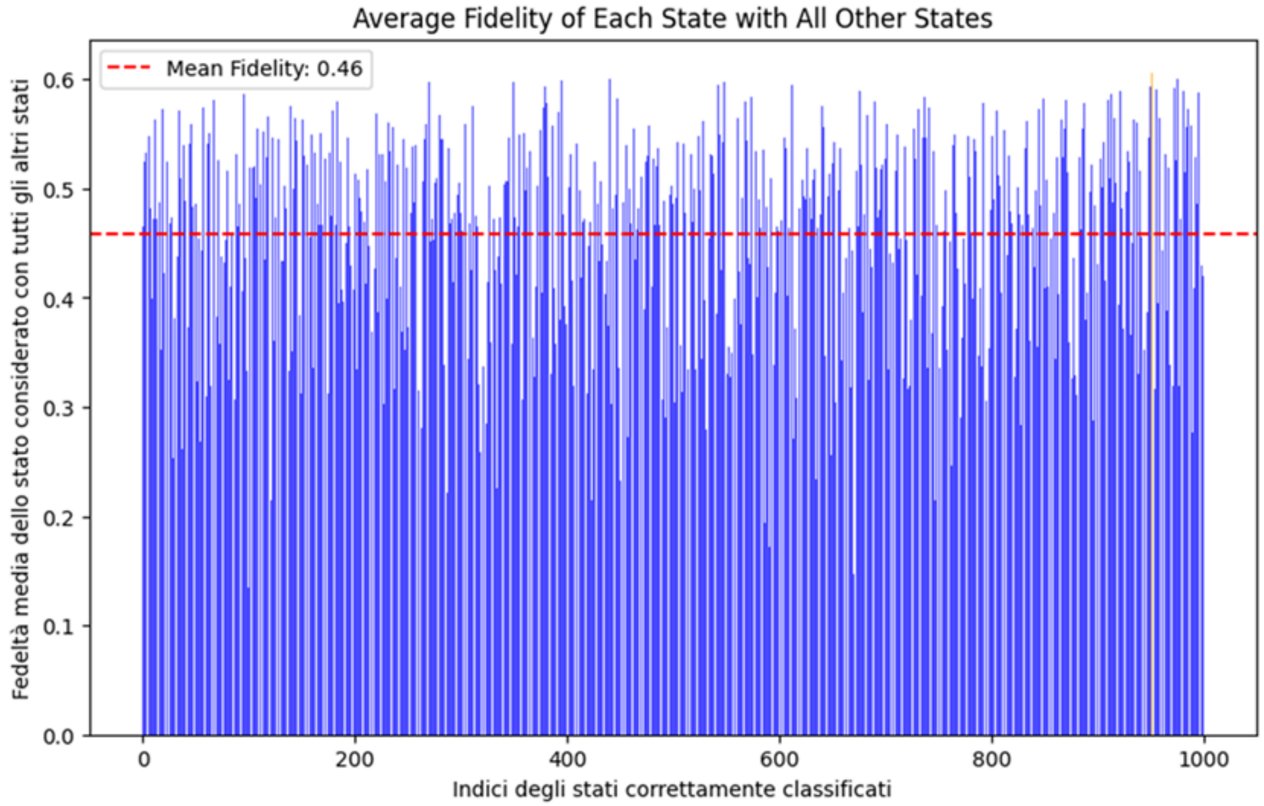


ANALISI DEI RISULTATI (2 EPOCHE DI ADDESTRAMENTO)

Epoca 1: Loss = 0.5630
Epoca 2: Loss = 0.3831
Accuracy = 93.80%
62 classificate in modo sbagliato
0 sbagliati : 46
1 sbagliati : 16



max_fidelity: 0.6062557127983451
Fidelity media degli stati erroneamente classificati : 0.7276497644283052
Fidelity media degli stati correttamente classificati : 0.5986515690795089



CONFRONTO DEL MODELLO A 8 QUBIT CON QUELLO A 4 QUBIT

Il modello a 8 qubit riduce gli errori ma mostra una bassa fedeltà tra lo stato ideale e gli stati erronei.
Al contrario, il modello a 4 qubit, sebbene con un numero maggiore di errori, mantiene una fedeltà superiore.

Questo suggerisce che una minore complessità può ancora garantire risultati di alta qualità, mentre una maggiore complessità potrebbe essere vantaggiosa per migliorare le tecniche di correzione degli errori.

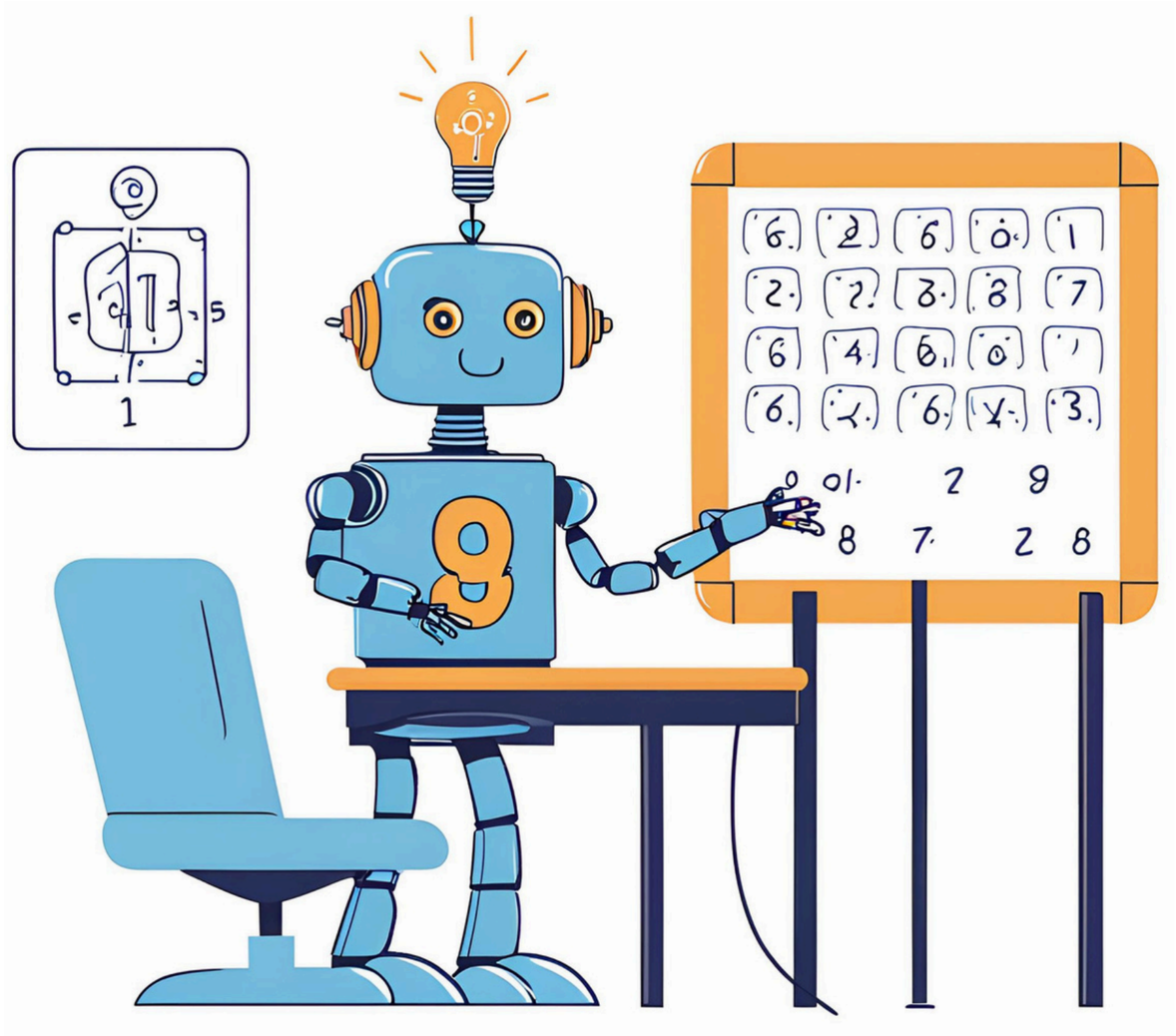
4 QUBIT

- maggiore frequenza di errori
- perdita di dettagli nella visualizzazione delle immagini
- alta fedeltà degli stati errati
- minore complessità

VS

8 QUBIT

- maggiore accuratezza
- migliore visualizzazione delle immagini
- bassa fedeltà degli stati errati
- maggiore complessità



GRAZIE PER L'ATTENZIONE