



UNIVERSITÀ DI PARMA

Dipartimento di Ingegneria e Architettura

Corso di Laurea in Ingegneria delle Tecnologie Informatiche

Implementazione e Analisi di Quantum GAN ad Architettura Patch-based per la Generazione di Immagini

Implementation and Analysis of Patch-based Quantum GANs for
Image Generation

Relatore:

Prof. Michele Amoretti

Correlatore:

Dott. Marco Mordacci

Tesi di Laurea di:

Emanuele Maffezzoli

ANNO ACCADEMICO 2024-2025

Alla mia famiglia.

Indice

Introduzione	1
Obiettivi della Tesi	2
Organizzazione della Tesi	2
1 Stato dell'arte	4
1.1 Machine Learning	4
1.1.1 Deep Learning e Reti Neurali Artificiali	5
1.1.2 Training e Ottimizzazione	7
1.1.3 Architetture Specializzate	9
1.2 Generative Adversarial Networks	10
1.2.1 Architettura e Principio di Funzionamento	11
1.2.2 Fase di Training	12
1.2.3 Batch vs. Patch	13
1.3 Fondamenti di Quantum Computing	14
1.3.1 Qubit	14
1.3.2 Entanglement	15
1.3.3 Gates	16
Gate a Singolo Qubit	16
Gate a Doppio Qubit	18
1.3.4 Misura Quantistica	18

1.4	Quantum Machine Learning	19
1.4.1	Parameter-Shift Rule	21
1.5	Quantum Generative Adversarial Networks	22
1.5.1	Componenti	23
1.5.2	Qubit Ancillari	24
1.5.3	Dataset MNIST e Modelli State-of-the-art	24
1.6	Metriche di Valutazione	27
1.6.1	Fréchet Inception Distance	27
1.6.2	Learned Perceptual Image Patch Similarity	28
2	Architettura del Modello QGAN	29
2.1	Componenti dell'Architettura	30
2.1.1	Generatore Quantistico	30
2.1.2	Circuito Quantistico Parametrizzato (PQC)	31
2.1.3	Discriminatore Classico	33
2.1.4	Varianti Ibride del Generatore	34
	Generatore QGAN-CNN	34
	Generatore QGAN-MLP	35
2.2	Generazione e Addestramento	35
2.3	Preparazione Immagini per le Metriche	36
3	Implementazione	38
3.1	Framework e Tecnologie Principali	38
3.2	Gestione dei Dati e Monitoraggio	39
3.3	Implementazione Pratica	40
3.3.1	Definizione del Discriminatore	40
3.3.2	Definizione del PQC	41
3.3.3	Definizione del Generatore	42

3.3.4	Ciclo di Addestramento	45
4	Risultati	48
4.1	Configurazioni Sperimentali	48
4.2	Risultati Qualitativi	50
4.3	Risultati Quantitativi	54
	Conclusioni	60
	Bibliografia	63

Introduzione

L'intelligenza artificiale e il machine learning hanno rivoluzionato numerosi settori negli ultimi decenni, con le reti neurali generative che rappresentano uno degli sviluppi più promettenti per l'elaborazione del testo e la generazione autonoma di contenuti. Parallelamente, il quantum computing sta emergendo come tecnologia in grado di offrire vantaggi significativi per specifiche classi di problemi. Il Quantum Machine Learning (QML) è considerato una delle prime applicazioni pratiche dei dispositivi quantistici near-term.

Le Generative Adversarial Networks (GAN) hanno dimostrato eccellenti capacità nella generazione di immagini realistiche attraverso un processo di "adversarial training" tra due reti neurali: un generatore che produce dati sintetici e un discriminatore che distingue tra dati reali e dati generati. Tuttavia, le GAN classiche presentano diverse limitazioni, tra cui l'instabilità durante il training, il collasso dell'abilità generativa e costi computazionali molto elevati.

In questo contesto, l'integrazione di principi quantistici in queste architetture rappresenta una frontiera di ricerca particolarmente interessante. Le Quantum Generative Adversarial Networks (QGAN) propongono di sostituire il generatore classico con un circuito quantistico parametrizzato, sfruttando la natura probabilistica della meccanica quantistica per la generazione di dati. I lavori teorici suggeriscono che le QGAN possano avere potenziali

vantaggi esponenziali rispetto alle GAN classiche; tuttavia, rimane ancora da chiarire se le QGAN implementate su dispositivi quantistici near-term possano effettivamente risolvere compiti di apprendimento del mondo reale.

Obiettivi della Tesi

Questo lavoro di Tesi intende sviluppare e valutare un'implementazione pratica di quantum GAN per la generazione di immagini. Il primo obiettivo consiste nella realizzazione di un'architettura funzionale, integrando il framework PennyLane per la gestione dei circuiti quantistici e PyTorch per la componente discriminativa classica. Le immagini vengono generate tramite la tecnica "patch" per superare le limitazioni dimensionali imposte dal numero limitato di qubit disponibili nei simulatori quantistici attuali. Invece di tentare di generare un'intera immagine con un singolo circuito quantistico, il sistema proposto suddivide il problema in sotto-generatori, ciascuno responsabile della generazione di una specifica porzione dell'immagine finale. Dal punto di vista della valutazione, vengono utilizzati i punteggi Fréchet Inception Distance (FID) e Learned Perceptual Image Patch Similarity (LPIPS) per quantificare obiettivamente la qualità delle immagini generate e la diversità tra di esse. Vengono infine testate diverse configurazioni parametriche per analizzare le proprietà dei circuiti e delle reti, fornendo così indicazioni pratiche per future implementazioni e ottimizzazioni.

Organizzazione della Tesi

Il Capitolo 1 presenta un'analisi dello stato dell'arte delle GAN e delle QGAN. Il Capitolo 2 illustra l'architettura del modello QGAN progettata e realizzata.

Il Capitolo 3 descrive l'implementazione del modello. Il Capitolo 4 presenta i risultati ottenuti tramite simulazioni. Infine, le Conclusioni riassumono il lavoro svolto e propongono alcuni sviluppi futuri.

Capitolo 1

Stato dell'arte

1.1 Machine Learning

Il **Machine Learning** è un settore fondamentale dell'intelligenza artificiale che studia metodi per l'apprendimento automatico dai dati e dalle esperienze. Aniché seguire istruzioni programmate esplicitamente per ogni specifica situazione, i modelli di machine learning identificano pattern e relazioni complesse all'interno di dataset di addestramento, migliorando iterativamente le proprie prestazioni in riferimento a un determinato compito.

Negli ultimi decenni, questo campo si è evoluto radicalmente, spinto dalla crescente disponibilità di grandi moli di dati (Big Data) e dalla potenza computazionale. Si è passati da applicazioni relativamente semplici, come modelli di regressione e classificazione, a complesse architetture di deep learning capaci di affrontare problemi più avanzati (come il riconoscimento delle immagini, l'elaborazione del linguaggio naturale e la guida autonoma), precedentemente considerati al di là delle capacità computazionali.

1.1.1 Deep Learning e Reti Neurali Artificiali

Le reti neurali artificiali traggono ispirazione dalla struttura interconnessa del cervello biologico. Esse organizzano unità computazionali elementari, note come neuroni artificiali, in strati (layers) sequenziali. Il modello di un singolo neurone artificiale, proposto da McCulloch e Pitts [1] e perfezionato con il Perceptron di Rosenblatt [2], è alla base di queste architetture.

Ogni neurone calcola un'aggregazione ponderata dei segnali che riceve in input. Questa operazione è tipicamente una combinazione lineare, in cui a ciascun input x_i è associato un peso sinaptico w_i , che ne determina l'importanza. A questa somma pesata viene aggiunto un termine di bias b , che agisce come un offset, permettendo alla funzione di attivazione di traslare. L'output del neurone y è quindi il risultato dell'applicazione di una funzione di attivazione non lineare σ come descritto dalla seguente formula:

$$y = \sigma\left(\sum_i (w_i \cdot x_i) + b\right) \quad (1.1)$$

L'introduzione della non-linearità tramite la funzione di attivazione è cruciale: senza di essa, una rete neurale composta da molti strati collasserebbe matematicamente a un singolo strato lineare, perdendo la capacità di modellare relazioni complesse. La scelta della funzione di attivazione influisce significativamente sulla dinamica di apprendimento.

Storicamente, la funzione **Sigmoid**, definita come:

$$\sigma(x) = 1/(1 + e^{-x}) \quad (1.2)$$

è stata ampiamente utilizzata. Essa comprime l'intero asse reale in un intervallo di output $[0, 1]$, rendendola idonea per l'output di problemi di classificazione binaria, dove il risultato può essere interpretato come una probabilità. Tuttavia, soffre del problema del vanishing gradient: per valori di input molto

grandi o molto piccoli, la sua derivata tende a zero, rallentando o bloccando l'apprendimento nelle reti profonde.

Per ovviare a ciò, la funzione **ReLU** (Rectified Linear Unit):

$$\sigma(x) = \max(0, x), \quad (1.3)$$

è diventata la scelta predominante per gli hidden layers [3]. La sua definizione è computazionalmente molto efficiente (un semplice confronto con zero). Fondamentalmente, per input positivi, la sua derivata è costante e pari a 1, mitigando drasticamente il problema del vanishing gradient e permettendo un flusso di errore più robusto durante la backpropagation.

Altre funzioni rilevanti includono la **Tanh** (tangente iperbolica), che mappa l'output in $[-1, 1]$ ed è centrata sullo zero, e varianti della ReLU come Leaky ReLU, $\sigma(x) = \max(\alpha x, x)$ con α piccolo, che previene il fenomeno dei "neuroni morti" (neuroni che si bloccano in uno stato di output nullo).

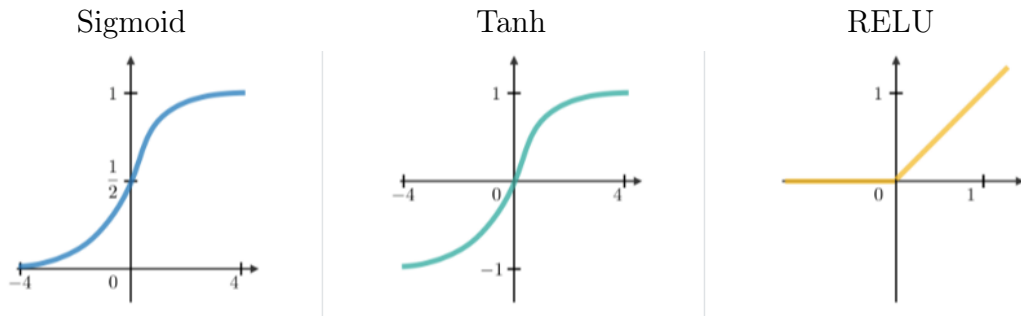


Figura 1.1: Funzioni di attivazione

Le architetture profonde (Deep Learning), caratterizzate da molteplici strati nascosti, sfruttano questa struttura per costruire rappresentazioni gerarchiche dei dati. Nei primi layer, la rete impara a riconoscere feature di basso livello (es. contorni, texture), mentre nei layer più profondi queste

vengono composte per comprendere concetti semantici via via più complessi (es. oggetti, volti).

1.1.2 Training e Ottimizzazione

L'obiettivo dell'addestramento (training) di una rete neurale è trovare l'insieme ottimale di parametri θ (l'insieme di tutti i pesi w_i e i bias b della rete) che consenta di mappare correttamente gli input agli output desiderati.

La **funzione di loss** (o funzione di costo) L è la misura quantitativa dell'errore commesso dalla rete. Quest'ultima calcola la discrepanza tra le previsioni della rete e i valori reali presenti nel dataset di addestramento. L'intero processo di training si configura quindi come un problema di ottimizzazione: minimizzare la funzione di loss $L(\theta)$ rispetto ai parametri θ .

Per problemi di classificazione binaria, dove l'obiettivo è predire una probabilità $p \in [0, 1]$ che un esempio appartenga a una determinata classe, la funzione di loss più comunemente utilizzata è la **Binary Cross Entropy** (BCE) [4]. Questa funzione misura la divergenza tra la distribuzione di probabilità predetta dal modello e la distribuzione reale dei dati.

Data una singola osservazione con etichetta vera $y \in \{0, 1\}$ e probabilità predetta $\hat{y} \in [0, 1]$, la BCE è definita come:

$$\text{BCE}(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \quad (1.4)$$

Quando $y = 1$, il primo termine domina e la loss penalizza le predizioni \hat{y} lontane da 1. Viceversa, quando $y = 0$, il secondo termine penalizza le predizioni \hat{y} lontane da 0. Per un dataset di N esempi, la BCE totale è calcolata come media:

$$L_{BCE}(\theta) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \quad (1.5)$$

Non essendo possibile trovare una soluzione esatta per la minimizzazione della loss in architetture complesse, si ricorre a metodi iterativi basati sul gradiente ($\nabla L(\theta)$). Questo è un vettore che indica la direzione di massima pendenza (massimo incremento) della funzione di loss e, per minimizzare l'errore, è necessario aggiornare i parametri muovendosi nella direzione opposta a quella del gradiente.

Il calcolo efficiente di questo gradiente in reti ampie è reso possibile dall'algoritmo di retropropagazione (backpropagation) dell'errore [5]. Sfruttando la chain rule del calcolo differenziale, la backpropagation calcola i gradienti della loss rispetto a ciascun parametro della rete, procedendo a ritroso dall'output verso l'input per ogni strato. Una volta calcolati, l'algoritmo di ottimizzazione li utilizza per aggiornare i parametri θ .

Un ottimizzatore fondamentale è lo Stochastic Gradient Descent (**SGD**) [6]. Nella sua forma pura (Batch Gradient Descent), calcolerebbe il gradiente sull'intero dataset prima di ogni aggiornamento, un processo computazionalmente molto costoso. La variante "stocastica" risolve questo problema stimando il gradiente su piccoli sottoinsiemi casuali del dataset, chiamati mini-batch. Questo approccio risulta più efficiente e introduce un elemento di rumore stocastico nel processo di ottimizzazione, il quale può evitare la convergenza a minimi locali non ottimali. L'aggiornamento dei parametri θ al tempo $t + 1$ segue la regola:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t) \quad (1.6)$$

dove η è il learning rate, un parametro essenziale che determina l'ampiezza del passo di aggiornamento.

1.1.3 Architetture Specializzate

Multi-Layer Perceptron (MLP): Il Multi-Layer Perceptron rappresenta l'architettura classica di rete neurale feedforward. È composto da una sequenza di strati: uno strato di input, uno o più strati nascosti (hidden layers) e uno strato di output. La caratteristica distintiva dell'MLP è che i suoi strati sono densi o fully-connected: ogni neurone di uno strato è connesso a ogni neurone dello strato successivo.

Grazie al teorema dell'approssimazione universale, un MLP con un solo strato nascosto (opportunamente dimensionato) e attivazioni non-lineari può, in teoria, approssimare qualsiasi funzione continua. Sebbene potenti come approssimatori generici, gli MLP non scalano bene ai dati ad alta dimensionalità, come le immagini. Essendo ogni neurone connesso a ciascun input, il numero di parametri (pesi) aumenta rapidamente. Ad esempio, un'immagine 100x100 ha 10000 pixel; un primo strato nascosto di soli 1000 neuroni richiederebbe già $10000 \times 1000 = 10$ milioni di pesi. Per questo motivo, gli MLP sono impiegati principalmente su dati già strutturati o come componenti finali all'interno di architetture più complesse.

Convolutional Neural Networks (CNN) [7]: Le Reti Neurali Convolutionali sono architetture progettate specificamente per elaborare dati che presentano una struttura a griglia, come le immagini o i video. La loro efficacia deriva dall'uso di operazioni di convoluzione nei loro strati iniziali: uno strato convoluzionale applica un insieme di filtri (o kernel), che sono piccoli tensori di parametri, i quali vengono traslati sistematicamente sull'input. Ogni filtro è specializzato nel rilevare un pattern locale specifico (ad esempio, un bordo o un colore).

Due principi chiave rendono le CNN estremamente potenti ed efficienti per i dati visivi: la condivisione dei parametri (parameter sharing), poiché lo

stesso filtro viene utilizzato su tutte le porzioni dell'immagine, riducendo drasticamente il numero totale di parametri rispetto a una rete fully-connected; e la località (o invarianza traslazionale), poiché la rete impara a riconoscere i pattern indipendentemente dalla loro posizione esatta.

Tipicamente, le architetture CNN alternano strati convoluzionali a strati di pooling, i quali riducono la dimensionalità spaziale (down-sampling) e aumentano il "campo ricettivo", permettendo alla rete di costruire una comprensione stratificata dell'immagine. Infine, l'output di questi strati viene solitamente appiattito e processato da uno o più strati fully-connected per la classificazione o regressione finale.

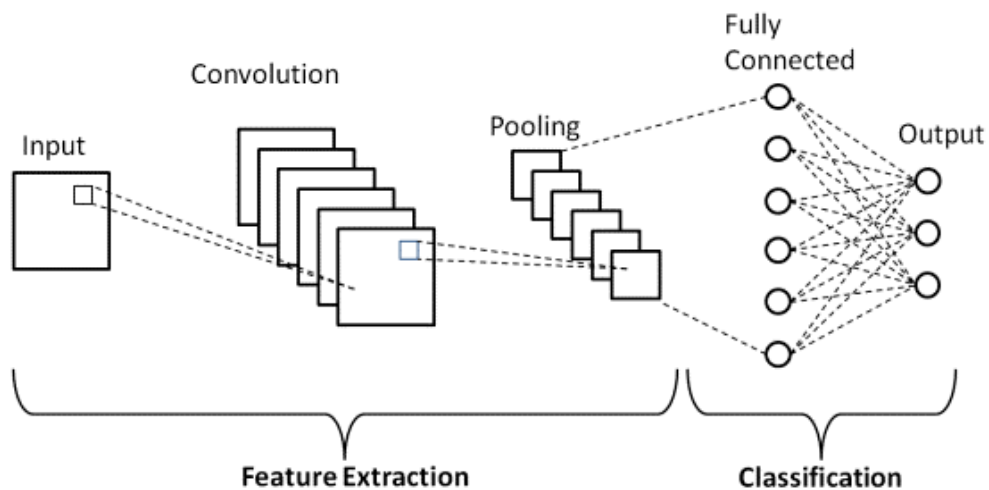


Figura 1.2: Schema strutturale di CNN [8]

1.2 Generative Adversarial Networks

Le **Generative Adversarial Networks (GAN)**, introdotte da Ian Goodfellow e collaboratori [9] nel 2014, rappresentano uno dei contributi più rilevanti e innovativi nell'ambito del machine learning dell'ultimo decennio. L'in-

tuizione alla base di questo modello consiste in un processo di apprendimento avversariale, in cui due reti neurali vengono poste in competizione all'interno di un gioco a somma zero, con l'obiettivo di migliorarsi reciprocamente durante la fase di addestramento.

1.2.1 Architettura e Principio di Funzionamento

L'architettura GAN comprende due componenti principali che operano in opposizione:

- **Generatore (G):** Una rete neurale che mappa vettori di rumore random z (tipicamente estratti da una distribuzione uniforme o gaussiana) nello spazio dei dati. Il generatore cerca di produrre campioni $G(z)$ che siano indistinguibili dai dati reali.
- **Discriminatore (D):** Una rete neurale che agisce come classificatore binario, cercando di distinguere tra campioni reali x provenienti dal dataset e campioni sintetici $G(z)$ prodotti dal generatore. È tipicamente implementato come rete convoluzionale standard per immagini o come rete fully-connected per dati strutturati.

Il processo di training può essere formalizzato come un gioco minimax descritto dalla seguente funzione obiettivo $V(D, G)$, dove il Discriminatore cerca di massimizzare il valore e il Generatore di minimizzarlo:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (1.7)$$

Il Discriminatore D cerca di massimizzare questa funzione: il primo termine, $\mathbb{E}_{x \sim p_{data}(x)} [\log D(x)]$, lo spinge ad assegnare una probabilità alta (vicina a 1) ai dati reali x . Il secondo termine, $\mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$, lo spinge ad

assegnare una probabilità bassa (vicina a 0) ai dati falsi $G(z)$. Contemporaneamente, il Generatore G cerca di minimizzare la funzione, ovvero tenta di produrre $G(z)$ tali che $D(G(z))$ sia vicino a 1 (ingannando D). Idealmente, all'equilibrio, il generatore produce campioni perfettamente realistici e il discriminatore non può fare meglio di un'ipotesi casuale ($D(x) = 0.5$ per ogni x).

1.2.2 Fase di Training

Il training delle GAN è un processo dinamico che richiede un'attenta calibrazione, poiché i due modelli vengono addestrati simultaneamente. La procedura avviene tipicamente alternando gli aggiornamenti dei due avversari. In un tipico passo di addestramento, si campiona un mini-batch di dati reali x dal dataset e si genera un mini-batch di campioni sintetici $G(z)$ (utilizzando z campioni dalla distribuzione latente).

Successivamente, si esegue la fase del Discriminatore: entrambi i batch (reale e sintetico) vengono passati al Discriminatore D . Si calcola la loss L_D e si aggiornano i parametri di D tramite backpropagation, con l'obiettivo di migliorare la propria capacità di distinguere il reale dal falso. Segue la fase del Generatore, nella quale si genera un nuovo mini-batch di campioni sintetici $G(z)$ che vengono passati al Discriminatore, i cui parametri sono ora invariati. Si calcola la loss del Generatore L_G , che misura quanto bene G stia ingannando D . I gradienti di questa loss vengono quindi propagati all'indietro attraverso il Discriminatore (che rimane fisso) fino ai parametri del Generatore G , che vengono aggiornati.

Questo equilibrio è notoriamente difficile da raggiungere. Uno dei problemi principali è il vanishing gradient precedentemente accennato: se il Discriminatore diventa troppo efficace, la sua funzione di loss satura e il gra-

diente che ritorna al Generatore tende a zero. Il Generatore, non ricevendo alcun segnale di errore, smette di migliorare. Un altro fallimento comune è il “mode collapse” [10], il quale si verifica quando il Generatore converge prematuramente verso un sottospazio limitato dello spazio di output. Anziché apprendere l'intera distribuzione dei dati, si identifica che la minimizzazione della propria funzione di loss è possibile producendo campioni con variabilità molto bassa, o addirittura un singolo output, che il Discriminatore interpreta come valido. Di conseguenza, il Generatore cessa di esplorare l'intero spazio latente e la diversità della distribuzione target viene persa, risultando in una generazione di campioni ripetitivi.

1.2.3 Batch vs. Patch

Il processo di training descritto in precedenza, in cui il Discriminatore D riceve un intero campione e produce un singolo output di probabilità, è definito approccio **Batch** o Globale. Il Generatore G viene eseguito k volte (dove k è la dimensione del batch) per generare k nuovi campioni. Questi k campioni vengono aggregati e presentati al Discriminatore insieme a un batch di k campioni reali, e la loss viene calcolata sulla media di queste valutazioni globali.

Un'alternativa influente, ispirata ad architetture note come PatchGAN [11], è il metodo **Patch**. In questa configurazione, il Discriminatore non valuta l'intero campione. Al contrario, D è progettato per analizzare l'input e restituire una matrice di output in cui ogni elemento corrisponde alla valutazione (vera/falsa) di una diversa sotto-regione, o patch, dell'immagine (es. blocchi $N \times N$). Il Discriminatore classifica quindi ogni patch individualmente e la loss finale viene calcolata come la media delle decisioni su tutte le patch.

1.3 Fondamenti di Quantum Computing

Il quantum computing rappresenta un approccio computazionale radicalmente diverso dal calcolo classico, sfruttando principi della meccanica quantistica (come sovrapposizione, entanglement e interferenza) per elaborare informazioni.

1.3.1 Qubit

L'unità fondamentale dell'informazione quantistica è il **qubit** (quantum bit). A differenza di un bit classico, che può essere solo 0 o 1, un qubit può esistere in una sovrapposizione di entrambi gli stati:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (1.8)$$

dove α e β sono ampiezze di probabilità complesse che soddisfano la condizione di normalizzazione $|\alpha|^2 + |\beta|^2 = 1$. Le ampiezze non sono direttamente osservabili; per caratterizzarle sono necessarie molte misurazioni del qubit. Quando si misura un qubit, il suo stato collassa in $|0\rangle$ con probabilità $|\alpha|^2$ o in $|1\rangle$ con probabilità $|\beta|^2$.

Un sistema di n qubit può esistere in una sovrapposizione di 2^n stati classici simultaneamente:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \quad (1.9)$$

Questa espansione esponenziale dello spazio degli stati rappresenta la causa del potenziale vantaggio quantistico per certi algoritmi: mentre un sistema classico di n bit ha 2^n possibili configurazioni, di cui solo una è "attiva" in un dato momento, un sistema quantistico di n qubit può codificare informazioni su tutte le 2^n configurazioni simultaneamente.

Sfera di Bloch: Per un singolo qubit, lo stato può essere visualizzato come un punto sulla superficie di una sfera unitaria:

$$|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\phi} \sin(\theta/2)|1\rangle \quad (1.10)$$

dove $\theta \in [0, \pi]$ e $\phi \in [0, 2\pi]$ sono coordinate sferiche. Questa rappresentazione geometrica viene utilizzata per visualizzare operazioni su singoli qubit.

1.3.2 Entanglement

L'**entanglement** rappresenta una delle proprietà fondamentali della meccanica quantistica, descrivendo una correlazione tra due o più sistemi che non ha alcun analogo nel mondo classico.

Per comprendere l'entanglement, è utile prima definire il suo opposto: uno stato separabile. In un sistema classico, o in un sistema quantistico separabile, le proprietà dei singoli componenti sono indipendenti. Lo stato complessivo di un sistema separabile può essere descritto come il prodotto tensoriale degli stati individuali (ad esempio, se il qubit A è nello stato $|a\rangle$ e il qubit B nello stato $|b\rangle$, lo stato congiunto è $|\psi\rangle = |a\rangle \otimes |b\rangle$ o, più semplicemente, $|ab\rangle$). Al contrario, un sistema è entangled quando i suoi componenti non possiedono più stati individuali ben definiti; solo lo stato globale del sistema è definito e non può essere fattorizzato come prodotto tensoriale di stati individuali.

L'esempio canonico di uno stato entangled a due qubit è lo **stato di Bell** $|\Phi^+\rangle$:

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \quad (1.11)$$

Questo stato descrive una sovrapposizione equiprobabile di due configurazioni: "entrambi i qubit nello stato $|0\rangle$ " oppure "entrambi nello stato $|1\rangle$ ". I

due qubit sono perfettamente correlati: se misuriamo il primo qubit e otteniamo $|0\rangle$, lo stato collassa istantaneamente e sappiamo con certezza che, misurando il secondo qubit (nella stessa base), otterremo anch'esso $|0\rangle$. Questa correlazione persiste indipendentemente dalla distanza fisica che separa i qubit [12].

1.3.3 Gates

Le operazioni sui qubit sono implementate attraverso gate quantistici [13], ovvero trasformazioni unitarie che preservano la normalizzazione (e quindi la natura probabilistica) degli stati quantistici. Ogni gate quantistico corrisponde matematicamente a una matrice unitaria U , tale che $UU^\dagger = I$, dove I è la matrice identità e U^\dagger è la trasposta coniugata.

Gate a Singolo Qubit

Questi gate agiscono su un singolo qubit, modificandone lo stato. Tra i più importanti vi sono i Gate di Pauli, che costituiscono una base per le operazioni single-qubit:

- Il gate **X** è l'analogo quantistico della porta NOT classica. Esegue un bit-flip, scambiando le ampiezze degli stati base: $|0\rangle \leftrightarrow |1\rangle$.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (1.12)$$

- Il gate **Z** esegue un'operazione di phase-flip. Lascia invariato lo stato $|0\rangle$, ma applica una fase di -1 (equivalente a una rotazione di π radianti attorno all'asse Z) allo stato $|1\rangle$, mappando quindi $|1\rangle \rightarrow -|1\rangle$.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad (1.13)$$

- Il gate **Y** applica una combinazione di un bit-flip e un phase-flip.

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad (1.14)$$

Un altro gate fondamentale per il calcolo quantistico è il **gate di Hadamard (H)**. La sua funzione primaria è generare la sovrapposizione. Se applicato a un qubit in uno stato della base computazionale ($|0\rangle$ o $|1\rangle$), il gate di Hadamard lo trasforma in una sovrapposizione uniforme (equiprobabile) dei due stati:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle \quad (1.15)$$

$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle \quad (1.16)$$

Gli stati risultanti, $|+\rangle$ e $|-\rangle$, sono anch'essi ortogonali e formano una base alternativa nota come base diagonale.

Infine, i gate di rotazione (R_X, R_Y, R_Z) sono operazioni parametrizzate che ruotano lo stato del qubit attorno ai rispettivi assi della Sfera di Bloch di un angolo θ . Ad esempio, la rotazione attorno all'asse Y ($R_Y(\theta)$) è cruciale per il machine learning quantistico ed è definita dalla matrice:

$$R_Y(\theta) = \exp(-i\theta Y/2) = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix} \quad (1.17)$$

Quando questo gate viene applicato allo stato $|0\rangle$, permette di creare una sovrapposizione arbitraria e controllata:

$$R_Y(\theta)|0\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle \quad (1.18)$$

La natura parametrizzata di questi gate è essenziale per gli algoritmi quantistici variazionali (come le QGAN), poiché l'angolo θ può essere trattato come un peso addestrabile, analogo ai pesi di una rete neurale classica.

Gate a Doppio Qubit

Per eseguire calcoli complessi e sfruttare appieno la potenza del calcolo quantistico, sono necessari gate che agiscano su più qubit, permettendo la creazione di entanglement.

Il gate a due qubit più noto è il **CNOT** (Controlled-NOT). Questo gate opera su un qubit di controllo e un qubit target. Applica un gate X (NOT) al qubit target se e solo se il qubit di controllo è nello stato $|1\rangle$. Al contrario, se il controllo è $|0\rangle$, il target rimane invariato:

$$\text{CNOT}|00\rangle = |00\rangle \quad (1.19)$$

$$\text{CNOT}|01\rangle = |01\rangle \quad (1.20)$$

$$\text{CNOT}|10\rangle = |11\rangle \quad (1.21)$$

$$\text{CNOT}|11\rangle = |10\rangle \quad (1.22)$$

Quando il CNOT viene applicato a un qubit di controllo che si trova in sovrapposizione (ad esempio, lo stato $|+\rangle$ ottenuto da $H|0\rangle$), il risultato è uno stato entangled in cui i due qubit sono correlati in modo non classico.

Un altro gate comune di questo tipo è il **CZ** (Controlled-Z), che applica un gate Z (phase-flip) al target solo quando il qubit di controllo è $|1\rangle$. Il suo unico effetto sulla base computazionale è applicare una fase di -1 allo stato $|11\rangle$:

$$\text{CZ}|11\rangle = -|11\rangle \quad (1.23)$$

1.3.4 Misura Quantistica

La misura è un processo intrinsecamente probabilistico che provoca il collasso dello stato quantistico in uno stato classico. Nel caso di una misura nella base

computazionale, il sistema collassa in uno dei due stati di base, secondo le seguenti probabilità:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \rightarrow \begin{cases} |0\rangle & \text{con prob } |\alpha|^2 \\ |1\rangle & \text{con prob } |\beta|^2 \end{cases} \quad (1.24)$$

Dopo la misura, lo stato collassa irreversibilmente nel risultato misurato. Questa irreversibilità distingue la misura dai gate unitari.

Per i sistemi multi-qubit, la misurazione di n qubit produce una stringa di n bit classici. Il risultato i compare con una probabilità $|\alpha_i|^2$. In quantum machine learning, spesso utilizziamo misure di probabilità che restituiscono la distribuzione completa $\{|\alpha_i|^2\}$ invece di un singolo outcome.

Misura Parziale: È possibile misurare solo un sottoinsieme di qubit, lasciando gli altri in sovrapposizione (ridotta). Questo è utile per tecniche relative all'utilizzo di qubit ancillari, che verranno approfondite nella prossima sezione.

1.4 Quantum Machine Learning

Il **Quantum Machine Learning (QML)** trae la sua motivazione principale dall'ipotesi che i sistemi quantistici possano offrire vantaggi computazionali e rappresentazionali rispetto ai metodi di apprendimento classici [14]. Sebbene tali vantaggi siano in gran parte teorici e non ancora dimostrati sperimentalmente, essi delineano direzioni di ricerca promettenti. Parallelamente, il campo è caratterizzato da limitazioni tecniche e teoriche che ne condizionano l'applicabilità pratica, specialmente nell'attuale tecnologia NISQ (Noisy Intermediate-Scale Quantum).

Potenziati vantaggi:

- **Vantaggio computazionale:** Si ipotizza che, per alcune classi di funzioni, i circuiti quantistici possano fornire una rappresentazione esponenzialmente più compatta rispetto ai modelli classici. In altre parole, un sistema quantistico potrebbe essere in grado di descrivere relazioni complesse utilizzando un numero di risorse, come porte e numero di parametri, significativamente inferiore.
- **Espressività:** Lo spazio di Hilbert associato a un sistema quantistico cresce esponenzialmente con il numero di qubit (2^n dimensioni per n qubit). Questa caratteristica permette ai modelli quantistici di rappresentare funzioni e distribuzioni di probabilità estremamente complesse, potenzialmente non raggiungibili da reti neurali classiche di dimensioni simili.
- **Efficienza di Training:** Grazie al parallelismo intrinseco della meccanica quantistica, alcuni aspetti del processo di training, come la valutazione simultanea di stati o funzioni, potrebbero essere accelerati.

Sfide e Limitazioni:

- **Bottleneck:** La codifica dei dati classici in stati quantistici rappresenta un ostacolo difficile. In molti casi, la preparazione degli stati quantistici richiede circuiti molto profondi o con un numero elevato di qubit [15], rischiando di annullare i benefici computazionali del calcolo quantistico. Anche l'estrazione di informazione classica dagli stati quantistici è un'operazione costosa. Per ottenere stime accurate delle probabilità, è spesso necessario ripetere un grande numero di misurazioni, con conseguente aumento del tempo di esecuzione.

- **Limitazione Hardware:** I dispositivi attuali appartenenti alla classe NISQ sono limitati dal rumore e da un numero ridotto di qubit affidabili. Questi vincoli restringono la dimensione e la complessità dei problemi che possono essere affrontati concretamente oggi.
- **Barren Plateaus:** Questo è un fenomeno critico che limita la scalabilità del QML, un problema che emerge quando si addestrano circuiti quantistici profondi con inizializzazione casuale dei parametri [16]. In questi circuiti, gli stati generati da parametri casuali tendono a distribuirsi uniformemente. Questa uniformità implica che piccole variazioni dei parametri causano solo cambiamenti infinitesimali nelle quantità osservabili, portando i gradienti a scalare come $\exp(-n)$, dove n rappresenta il numero di qubit o la profondità del circuito. Per valori anche moderati di n (tipicamente 20-30 qubit), i gradienti diventano talmente piccoli da essere indistinguibili dal rumore statistico delle misure, rendendo di fatto impossibile l'ottimizzazione tramite gradient descent.

1.4.1 Parameter-Shift Rule

Nei circuiti quantistici parametrizzati, il calcolo dei gradienti rispetto ai parametri variazionali è fondamentale per l'ottimizzazione, analogamente alla backpropagation nelle reti neurali classiche. Tuttavia, la natura probabilistica della meccanica quantistica rende inutilizzabile l'uso dei metodi tradizionali. La **Parameter-Shift Rule** [17] risolve questo problema fornendo un metodo analitico per calcolare gradienti esatti di aspettative quantistiche.

Consideriamo un circuito quantistico parametrizzato $U(\theta)$ e un'osservabile O . L'obiettivo è calcolare il gradiente della quantità attesa $f(\theta) =$

$\langle \psi(\theta) | O | \psi(\theta) \rangle$ rispetto a un parametro θ_i . Per gate di rotazione della forma $R_i(\theta_i) = \exp(-i\theta_i P_i/2)$, dove P_i è un generatore (tipicamente un gate di Pauli), la Parameter-Shift Rule stabilisce che:

$$\frac{\partial f(\theta)}{\partial \theta_i} = r [f(\theta + s) - f(\theta - s)] \quad (1.25)$$

dove $s = \pi/(4r)$ è lo shift del parametro e r dipende dallo spettro del generatore. Per i gate di Pauli standard (X, Y, Z), si ha $r = 1/2$ e quindi $s = \pi/2$, ottenendo la formula:

$$\frac{\partial f(\theta)}{\partial \theta_i} = \frac{1}{2} [f(\theta_i + \pi/2) - f(\theta_i - \pi/2)] \quad (1.26)$$

Questa regola ha un'interpretazione operativa molto pratica: per calcolare il gradiente rispetto a θ_i , è sufficiente eseguire il circuito quantistico due volte, una con il parametro "shiftato" di $+\pi/2$ e una con $-\pi/2$, e calcolare la differenza delle aspettative risultanti.

Questo presenta vantaggi cruciali per il QML, in quanto fornisce gradienti ed è hardware-efficient (richiede solo esecuzioni aggiuntive del circuito senza modifiche strutturali). Nonostante ciò, presenta anche limitazioni per circuiti profondi, dove il numero totale di valutazioni può diventare proibitivo [18].

1.5 Quantum Generative Adversarial Networks

Le Quantum GAN, teorizzate per la prima volta nel 2018 da Lloyd e Weedbrook [19], rappresentano l'estensione naturale delle GAN classiche al dominio quantistico. La loro implementazione avviene sostituendo uno o entrambi i componenti con circuiti quantistici parametrizzati.

1.5.1 Componenti

Data la limitata disponibilità e la natura "rumorosa" dei processori quantistici attuali, l'approccio che prevede entrambi i componenti quantistici è di difficile implementazione. Pertanto, l'approccio ibrido è il più studiato e implementato nella letteratura corrente. Nello specifico, il modello ibrido Quantum-Classical GAN impiega un Generatore quantistico e un Discriminatore classico. Il discriminatore viene implementato in maniera classica, in modo tale da non avere bisogno di un circuito di codifica dei dati classici reali in stati quantistici, poiché esso genererebbe molto rumore se venisse eseguito su hardware reale. Questa configurazione cerca di bilanciare lo sfruttamento delle capacità generative quantistiche con la solida capacità di apprendimento discriminativo già dimostrata dalle reti neurali classiche. Il funzionamento del generatore segue tre fasi principali:

1. **Codifica dell'Input:** Un vettore di rumore classico z , campionato da una distribuzione normale standard, viene utilizzato per inizializzare lo stato del circuito. Questo vettore z viene codificato nello stato quantistico iniziale, tipicamente $|\psi_0\rangle = \text{Encode}(z)|0\rangle^{\otimes n}$, dove n è il numero di qubit e $\text{Encode}(z)$ è un circuito di encoding (ad esempio, una serie di rotazioni R_Y i cui angoli dipendono dai valori in z).
2. **Circuito Variazionale:** Lo stato iniziale evolve attraverso il circuito parametrizzato $U(\theta)$. Questo circuito è composto da una sequenza di strati (layers) di gate quantistici. Ogni strato è tipicamente costituito da gate di rotazione single-qubit (es. $R_X(\theta_i), R_Y(\theta_j)$) e da gate di entanglement multi-qubit (es. CNOT, CZ). I parametri θ di questi gate (gli angoli di rotazione) sono i pesi del Generatore, analoghi ai pesi di una rete neurale classica. $U(\theta) = U_L(\theta_L) \cdot \dots \cdot U_2(\theta_2) \cdot U_1(\theta_1)$

3. **Misurazione:** Dopo l'evoluzione attraverso $U(\theta)$, si ottiene lo stato finale $|\psi(\theta, z)\rangle$. Per estrarre dati classici (es. un'immagine) da questo stato, si effettua una misurazione.

Il Discriminatore D rimane classico ed è tipicamente una rete neurale classica standard, come MLP o CNN precedentemente trattati.

1.5.2 Qubit Ancillari

Per aumentare l'espressività del Generatore quantistico senza necessariamente aumentare il numero di qubit di output, è possibile introdurre i qubit di ancilla (o ausiliari).

In questa configurazione, il sistema totale è composto da $n = n_s + n_a$ qubit, dove n_s sono i qubit di "sistema" (che produrranno l'output finale) e n_a sono i qubit ancillari. Il circuito $U(\theta)$ agisce sull'intero registro di n qubit:

$$|\psi_{\text{tot}}\rangle = U(\theta) (|0\rangle^{\otimes n_s} \otimes |0\rangle^{\otimes n_a}) \quad (1.27)$$

permettendo la creazione di entanglement tra i qubit di sistema e quelli ancillari. Questi ultimi agiscono come un registro quantistico, consentendo al circuito di implementare trasformazioni unitarie più complesse sui soli n_s qubit di sistema.

Al termine dell'evoluzione, si effettua una misurazione parziale solo sui n_s qubit di sistema. Lo stato ridotto risultante è:

$$\rho_s = \text{Tr}_a[|\psi_{\text{tot}}\rangle\langle\psi_{\text{tot}}|] \quad (1.28)$$

1.5.3 Dataset MNIST e Modelli State-of-the-art

Il **dataset MNIST** comprende una raccolta di cifre scritte a mano ed è ampiamente utilizzato come benchmark. È composto da circa 70000 cifre

scritte a mano (cifre da 0 a 9 comprese) con ogni esempio rappresentato come un'immagine in scala di grigi di 28×28 pixel, suddiviso in 60.000 immagini di addestramento e 10.000 immagini di test [20].

I seguenti modelli sono stati ottimizzati e addestrati su quest'ultimo e possono essere considerati lo stato dell'arte attuale nell'ambito QGAN.

- **MosaiQ** [21]: MosaiQ si concentra sulla qualità dell'immagine e sulla stabilità del training, introducendo due innovazioni tecniche principali. In primo luogo, si differenzia nell'elaborazione dell'input: invece di un approccio pixel-per-pixel, implementa una tecnica di ridistribuzione delle feature. Questa tecnica sfrutta la principal component analysis (PCA) dei dati dell'immagine. In questo modo, il modello non apprende da singoli pixel, ma da una rappresentazione più compatta e significativa, permettendo al circuito quantistico di operare in modo più efficiente. In secondo luogo, per affrontare direttamente il mode collapse, MosaiQ non utilizza un rumore di input statico, come una semplice distribuzione gaussiana, ma impiega un metodo di generazione di rumore di input adattivo. Questo meccanismo regola dinamicamente l'input del generatore per incoraggiare l'esplorazione di diverse modalità dei dati. L'efficacia di questa combinazione è dimostrata da un miglioramento di oltre 100 punti nel punteggio della metrica FID (descritta nel capitolo successivo).
- **QINR-QGAN** [22]: L'obiettivo principale di questo modello è l'efficienza parametrica, e la sua differenza fondamentale è l'integrazione della "Quantum Implicit Neural Representation (QINR)". Mentre una QGAN standard mappa un vettore di rumore a un'immagine intera (richiedendo molti parametri), un approccio QINR addestra il modello

quantistico a funzionare come una funzione continua. Questo approccio riduce drasticamente il numero di parametri quantistici addestrabili, rendendoli fino a 10 volte inferiori rispetto a PQWGAN [23]. Per quanto riguarda la stabilità, QINR-QGAN non introduce nuovi stabilizzatori quantistici, ma adatta efficacemente tecniche già consolidate, come l'applicazione della distanza di Wasserstein [24] come funzione di costo.

- **VAE-QWGAN** [25]: È un modello ibrido che combina l'architettura di un Variational Autoencoder (VAE) [26] con una QGAN. La differenza tecnica principale riguarda la gestione dello spazio latente, in quanto un encoder VAE classico viene prima addestrato per comprimere le immagini MNIST in una rappresentazione strutturata (spazio che segue una distribuzione approssimativamente Gaussiana). Questa rappresentazione latente viene quindi utilizzata come distribuzione a priori per il generatore quantistico. A livello architetturale, il decoder del VAE e il generatore sono fusi in un singolo modello quantistico con parametri condivisi. Per massimizzare la diversità, il modello non campiona da una semplice gaussiana, ma da un Gaussian Mixture Model (GMM), che è stato a sua volta addestrato sui vettori latenti prodotti dall'encoder durante il training. Questo campionamento assicura che il generatore attinga a tutte le modalità identificate dal VAE, mitigando in modo efficiente il mode collapse.

L'architettura alla base che verrà utilizzata per i successivi test si basa su un modello proposto da Huang e collaboratori [27], il quale verrà descritto più nel dettaglio nelle sezioni successive.

1.6 Metriche di Valutazione

La valutazione dei modelli proposti è un compito intrinsecamente complesso, in quanto mira a quantificare concetti qualitativi come il "realismo" e la "diversità". Per superare la valutazione soggettiva umana, sono state sviluppate metriche quantitative che analizzano le distribuzioni dei dati generati.

1.6.1 Fréchet Inception Distance

Il **FID** si è affermato come standard per la valutazione della qualità delle immagini generate dalle GAN [28]. Aniché operare un confronto diretto pixel per pixel, il FID misura la distanza statistica tra la distribuzione delle immagini reali e quella delle immagini generate nello spazio delle feature (caratteristiche).

La procedura prevede l'impiego della rete Inception-V3 [29], pre-addestrata sul dataset ImageNet, come estrattore di feature. Un vasto campione di immagini reali e un numero equivalente di immagini generate vengono elaborati dalla rete. Da queste elaborazioni si estraggono le attivazioni di uno strato intermedio profondo (tipicamente lo strato pool3, che produce un vettore di 2048 dimensioni) in base al numero di parametri.

Queste due collezioni di vettori di feature vengono quindi modellate come distribuzioni gaussiane multivariate. Si calcolano i rispettivi vettori delle medie (μ_r, μ_g) e le matrici di covarianza (Σ_r, Σ_g). Il FID è, infine, la distanza di Fréchet calcolata tra queste due distribuzioni:

$$FID = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \cdot \Sigma_g)^{1/2}) \quad (1.29)$$

dove Tr indica la traccia della matrice, cioè la somma dei suoi elementi diagonali. Un valore di FID più basso indica una maggiore somiglianza tra la di-

istribuzione delle immagini generate e quella delle immagini reali, suggerendo una migliore qualità e diversità complessiva.

È importante notare che il punteggio FID assoluto non è una metrica universale, poiché il suo valore è sensibile a molti fattori, tra cui il dataset di riferimento, il numero di campioni utilizzati e il pre-processing applicato alle immagini.

1.6.2 Learned Perceptual Image Patch Similarity

A differenza del FID, che confronta intere distribuzioni di dataset, la metrica **LPIPS** [30] è progettata per misurare la distanza percettiva tra due immagini specifiche, x e x' . Per ottenere questo risultato, LPIPS utilizza le attivazioni estratte da una rete neurale profonda pre-addestrata (come AlexNet, VGG o SqueezeNet).

Le due immagini x e x' vengono passate attraverso la rete, estraendo le attivazioni $\{F_l\}$ da più strati l . Per ogni strato, si calcola la distanza L2 tra le attivazioni, la quale viene ponderata (tramite prodotto elemento per elemento \odot) da un vettore di pesi w_l . Questi pesi sono stati a loro volta addestrati su un dataset di giudizi umani, permettendo alla metrica di identificare quali strati e quali feature siano più rilevanti per la percezione visiva.

La distanza d_l per lo strato l è calcolata come:

$$d_l = ||w_l \odot (F_l(x) - F_l(x'))||^2 \quad (1.30)$$

Il punteggio LPIPS finale è la somma di queste distanze d_l attraverso tutti gli strati considerati: $LPIPS = \sum_l d_l$. Un valore LPIPS più basso indica che le due immagini sono considerate percettivamente più simili. Sebbene sia efficace nel catturare le differenze di texture e stile a livello di patch, questa metrica rimane meno sensibile agli aspetti di tipo globale.

Capitolo 2

Architettura del Modello QGAN

L'architettura sviluppata prende come riferimento principale il lavoro di Huang et al. [27] relativo a QGAN sperimentali per la generazione di immagini, adattandone i principi e il modello. L'obiettivo principale è addestrare un modello capace di generare immagini di 16×16 pixel che rappresentano la cifra "0", basandosi su un sottoinsieme filtrato del dataset MNIST. In aggiunta a questa configurazione di base, è stata implementata e testata una seconda architettura per la generazione di immagini a risoluzione maggiore, 32×32 pixel, per valutare la scalabilità dell'approccio.

L'architettura si fonda sull'interazione avversaria di un generatore quantistico e di un discriminatore classico. Questi due componenti vengono addestrati simultaneamente, come descritto dettagliatamente nelle sezioni successive.

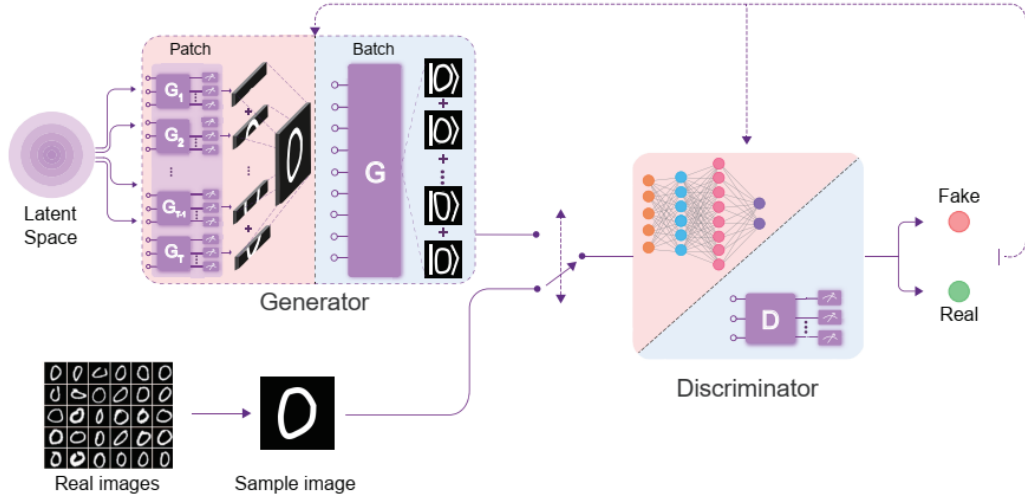


Figura 2.1: Schema strutturale di QGAN [27]

2.1 Componenti dell'Architettura

2.1.1 Generatore Quantistico

Il Generatore è il componente responsabile della creazione di nuove immagini. Per generare un'immagine completa di 16×16 (256 pixel) sarebbe necessario un circuito quantistico capace di modellare una distribuzione di probabilità su 256 stati, richiedendo $\log_2(256) = 8$ qubit di dati. Tuttavia, la complessità computazionale della simulazione non dipende solo dal numero di qubit, ma anche dalla profondità totale del circuito. Basandosi sulla struttura dell'ansatz variazionale che sarà dettagliata nella Sezione 2.1.2, un circuito completo che opera su $n = 9$ qubit totali (8 di dati più 1 ancilla-re) raggiungerebbe una profondità totale di $D_{\text{circ}} = 12$. L'addestramento di un circuito che combina $n = 9$ qubit con una tale profondità su simulatori classici è computazionalmente molto oneroso.

Per superare questa limitazione, è stata implementata un'architettura "patch-based". L'immagine finale non è quindi generata da un unico circuito,

ma è composta assemblando gli output di N_g sub-generatori indipendenti.

La mappatura per la configurazione base 16×16 avviene come segue:

- Pixel totali: $16 \times 16 = 256$
- Numero di generatori: 4
- Pixel per generatore (dimensione della patch): $256/4 = 64$

Per l'architettura 32×32 (1024 pixel), è stata mantenuta una logica analoga con quattro generatori:

- Pixel totali: $32 \times 32 = 1024$
- Numero di generatori: 4
- Pixel per generatore (dimensione della patch): $1024/4 = 256$

Oltre alla configurazione di base a 4 generatori, per entrambe le risoluzioni è stata testata anche una configurazione alternativa con 8 generatori, dimezzando dunque le dimensioni della patch. In entrambi i casi, ciascun sub-generatore è un circuito quantistico indipendente che riceve in input un vettore di rumore e produce in output la patch di pixel corrispondente. I vettori di output vengono infine concatenati per formare l'immagine completa.

2.1.2 Circuito Quantistico Parametrizzato (PQC)

Ciascuno dei sub-generatori è implementato come un Circuito Quantistico Parametrizzato (**PQC**). Per la configurazione base (16×16 con patch da 64 pixel), il circuito necessita di $\log_2(64) = 6$ qubit di dati. A questi si aggiunge $n_a = 1$ qubit ancillare, portando il numero totale di qubit per PQC a:

$$n_{\text{qubits}} = \log_2(\text{pixel}_{\text{patch}}) + n_a = 6 + 1 = 7 \text{ qubits} \quad (2.1)$$

Sono state condotte anche sperimentazioni variando il numero di qubit ancillari (es. $n_a = 2$), sebbene $n_a = 1$ sia rimasta la configurazione di riferimento per bilanciare l'espressività e il costo computazionale.

La struttura di ogni PQC è definita come segue:

1. **Codifica del Rumore:** Il circuito riceve in input un vettore di rumore classico z (campionato da una distribuzione uniforme $U(0, \pi/2)$) di dimensione $2 \times n_{\text{qubits}}$. Questo rumore viene "caricato" nello stato quantistico iniziale (tipicamente $|0\rangle^{\otimes n_{\text{qubits}}}$) applicando rotazioni $R_Y(z_i)$ e $R_X(z_{i+n})$ a ciascun qubit i .
2. **Struttura Variazionale (Ansatz):** Segue l'ansatz, ovvero il corpo principale del circuito i cui parametri θ sono oggetto dell'addestramento. L'ansatz è composto da sei strati identici. Ogni strato è formato da due blocchi:
 - (a) Blocco di Rotazione: Applicazione di rotazioni $R_X(\theta_{i,j,0})$ e $R_Y(\theta_{i,j,1})$ a ogni qubit j nello strato i . (Test preliminari hanno indicato questa combinazione come più efficace rispetto ad altre, es. R_Y/R_Z).
 - (b) Blocco di Entanglement: Applicazione di porte C_Z (Control-Z) a coppie di qubit adiacenti. Questo passo è fondamentale per creare correlazioni complesse. Come alternativa al C_Z , è stata testata una variante con blocchi di entanglement parametrizzati, utilizzando porte $C_RX(\phi_{i,j})$, dove ϕ è un ulteriore parametro addestrabile.
3. **Output:** L'output del circuito è la distribuzione di probabilità sull'intero spazio di Hilbert. Eseguendo il circuito, si ottiene un vettore p di

$2^{n_{\text{qubits}}}$ probabilità, $p_k = |\langle k | \psi(\theta, z) \rangle|^2$, che rappresenta la probabilità di misurare ciascuno degli stati k della base computazionale.

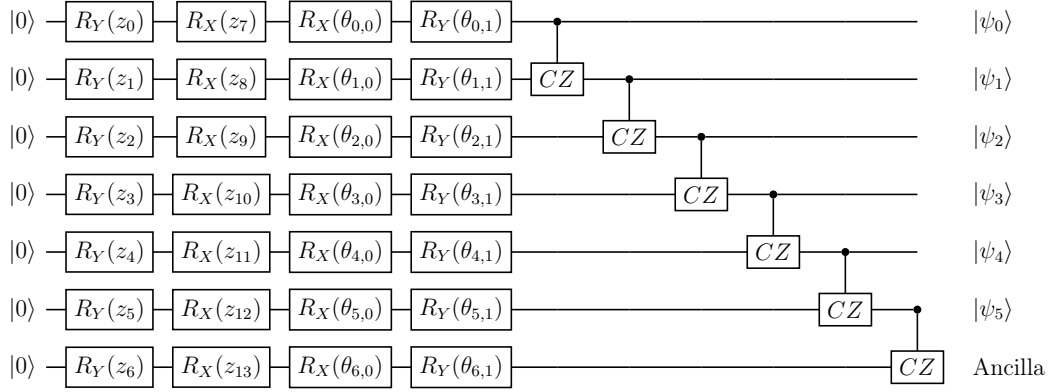


Figura 2.2: Singolo layer del PQC nella configurazione base

2.1.3 Discriminatore Classico

Il Discriminatore è un modello classico che agisce come classificatore binario. Si tratta di una Rete Neurale fully-connected (Multi-Layer Perceptron) la cui architettura varia solo in base alla dimensione dell'immagine in input.

Per i test 16×16 , l'immagine viene "appiattita" (flattened) in un singolo vettore di 256 elementi. Questo vettore alimenta una sequenza di strati lineari (con dimensioni $256 \rightarrow 64$, $64 \rightarrow 16$, $16 \rightarrow 1$), intervallati da funzioni di attivazione ReLU. Per i test 32×32 , l'approccio è identico ma scalato: l'input è un vettore di 1024 elementi e la struttura MLP è proporzionalmente più grande, con strati intermedi più ampi (es. $1024 \rightarrow 512 \rightarrow 128 \rightarrow 1$) prima dell'output. Oltre a questo modello di base, per il discriminatore 32×32 sono state testate anche due architetture MLP più profonde: un modello a 5 strati nascosti e un modello più complesso a 7 strati nascosti, entrambi con una riduzione progressiva della dimensionalità a ogni strato.

In tutti i casi, lo strato di output, composto da un singolo neurone, utilizza una funzione di attivazione Sigmoid, che comprime l'output nell'intervallo $[0, 1]$. Questo valore scalare rappresenta la probabilità determinata dal discriminatore che l'immagine in input sia "reale" (valore vicino a 1) o "falsa" (valore vicino a 0).

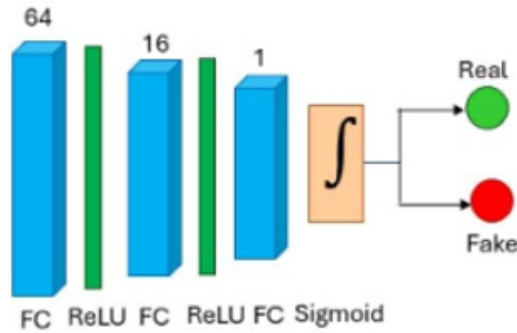


Figura 2.3: Struttura concettuale del Discriminatore 16x16 Classico [27]

2.1.4 Varianti Ibride del Generatore

Oltre al modello QGAN "puro", in cui l'output dei PQC viene concatenato e inviato al discriminatore, sono state valutate due architetture ibride alternative per il generatore. In queste varianti, l'immagine assemblata dalle patch quantistiche subisce un ulteriore passaggio di post-processing classico (tramite una CNN o un MLP) prima di essere considerata l'output finale del generatore.

Generatore QGAN-CNN

In questa architettura, l'output quantistico viene prima rimodellato e successivamente trattato attraverso una Rete Neurale Convoluzionale composta da strati Conv2d [31] ($1 \rightarrow 16$ canali, $16 \rightarrow 32$, $32 \rightarrow 1$) e attivazioni Re-

LU. L'output "raffinato" della CNN viene infine appiattito e restituito come output finale del generatore. Lo scopo di questa CNN interna è agire come un post-processore addestrabile, imparando a raffinare gli artefatti e a migliorare le correlazioni spaziali dell'output quantistico.

Generatore QGAN-MLP

Questa variante segue una logica simile, ma utilizza un Multi-Layer Perceptron (MLP) per il post-processing. Il vettore di patch concatenate (es. 1×256) viene passato direttamente a un MLP (es. $256 \rightarrow 500$, ReLU, $500 \rightarrow 500$, ReLU, $500 \rightarrow 256$). Il vettore di output di questo MLP viene quindi restituito come output finale del generatore.

In entrambi i casi ibridi, i gradienti della loss del generatore vengono retro-propagati sia attraverso gli strati classici di post-processing (addestrandoli), sia attraverso i PQC per addestrare i parametri quantistici.

2.2 Generazione e Addestramento

L'addestramento segue il ciclo GAN standard, già descritto nel capitolo precedente 1.2.2 e formalizzato dall'Eq. (1.7), alternando l'ottimizzazione del discriminatore e del generatore. Per l'aggiornamento dei parametri di entrambe le reti è stato utilizzato l'algoritmo di ottimizzazione Stochastic Gradient Descent (**SGD**), applicato iterativamente per minimizzare le rispettive funzioni di costo. Nella fase di addestramento del Discriminatore (D), i suoi pesi classici vengono aggiornati utilizzando la Binary Cross-Entropy (BCE) (1.1.2) su batch di immagini reali (etichetta 1.0) e false (etichetta 0.0). Nella fase di addestramento del Generatore (G), i parametri del discriminatore rimangono invariati. Il generatore viene ottimizzato utiliz-

zando la loss "non-saturating" (massimizzando $\log D(G(z))$), associando alle immagini false l'etichetta "reale".

L'aspetto cruciale di questa architettura ibrida risiede nella retropropagazione verso il generatore. Il gradiente ∇_{θ} della loss, calcolato classicamente attraverso D , viene applicato ai parametri θ dei PQC (i sub-generatori) utilizzando la parameter-shift rule (1.4.1). Questo ciclo viene ripetuto per un numero prefissato di iterazioni, portando G a produrre immagini sempre più indistinguibili da quelle reali.

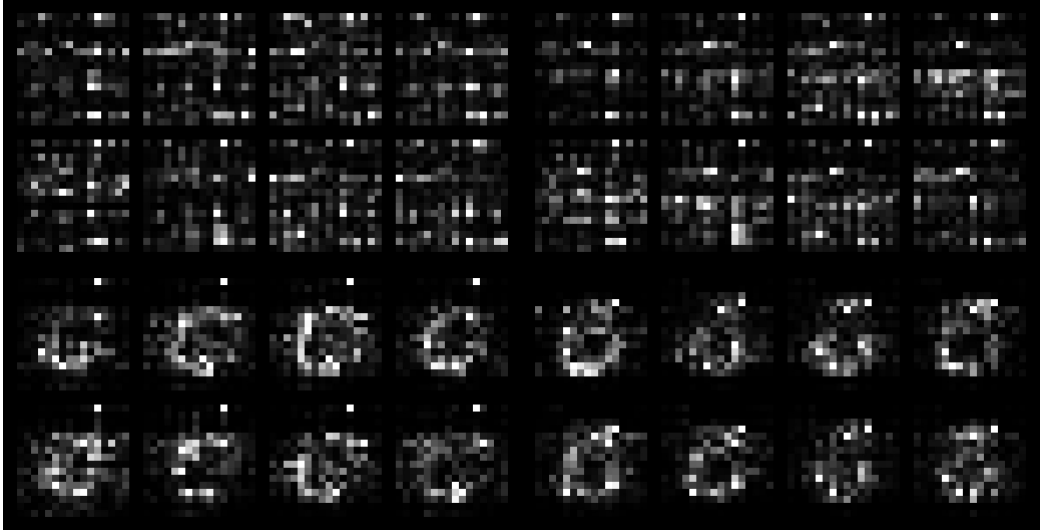


Figura 2.4: Evoluzione dell'addestramento per una configurazione generica con 8 immagini per ogni esempio (iterazioni 50, 150, 300, 500).

2.3 Preparazione Immagini per le Metriche

Le immagini provenienti dal dataset MNIST, aventi una dimensione nativa di 28x28 pixel, vengono ridimensionate, in fase di caricamento del dataset,

alla dimensione target dell'esperimento, la stessa che il generatore andrà a produrre.

Per valutare quantitativamente la qualità e la diversità delle immagini generate, a intervalli regolari durante l'addestramento viene eseguita una procedura di valutazione statistica. Per questa procedura, vengono generate 5000 immagini false, un campione statisticamente significativo necessario per garantire il corretto calcolo dei valori. Per LPIPS, che qui è utilizzato per misurare la diversità dei campioni, viene utilizzato un sottoinsieme ridotto di queste ultime (B_1 e B_2 di 500 esemplari ciascuno).

Le immagini generate e reali richiedono un pre-processing specifico per essere compatibili con le metriche standard. Un passaggio comune a entrambe le metriche (FID e LPIPS) è il rimodellamento delle immagini e la conversione in 3 canali (RGB), replicando il canale monocromatico. Da qui, la preparazione si differenzia:

- Per FID, le immagini a 3 canali vengono scalate nell'intervallo $[0, 255]$ e convertite in interi (tipo byte). È importante notare che per le immagini 16×16 è stato utilizzato lo strato di feature 192 del modello Inception, mentre per le immagini 32×32 è stato impiegato lo strato 2048, in quanto standard per risoluzioni maggiori.
- Per LPIPS, le immagini a 3 canali vengono mantenute nell'intervallo $[0, 1]$ (tipo float). Per i test 16×16 , le immagini sono state ingrandite a 32×32 tramite interpolazione bilineare; per i test 32×32 , sono state utilizzate nella loro dimensione nativa di 32×32 per adattarsi al modello percettivo.

Capitolo 3

Implementazione

Questo capitolo descrive i principali framework, le librerie ausiliarie e le classi specifiche che compongono il sistema complessivo.

3.1 Framework e Tecnologie Principali

La natura ibrida del modello ha richiesto l'integrazione di due framework software distinti:

- **PennyLane** viene utilizzato per la simulazione quantistica [32]. Il suo ruolo primario è definire il circuito quantistico che implementa il generatore e permetterne il calcolo dei gradient per l'ottimizzazione. I circuiti quantistici sono definiti come "QNodes", ovvero funzioni che incapsulano la sequenza di porte (rotazioni ed entanglement) e specificano il dispositivo di esecuzione. Nell'implementazione è stato utilizzato il simulatore "default.qubit", che permette di eseguire e testare i circuiti quantistici mediante hardware classico.
- **PyTorch** agisce come il "cervello" classico del sistema, gestendo tutti gli aspetti computazionali non quantistici [33]. Il Discriminatore

è implementato interamente come un modulo standard di PyTorch, ereditando dalla classe base dei modelli. La sua architettura MLP è definita impilando strati lineari e funzioni di attivazione fornite dalla libreria. Inoltre, PyTorch gestisce l'intero ciclo di ottimizzazione. Sia gli ottimizzatori che le funzioni di costo sono componenti nativi del framework, utilizzati per aggiornare i parametri classici del discriminatore e quelli quantistici del generatore.

Questa metodologia risulta efficace grazie all'integrazione dei due framework. La classe che definisce il Generatore Quantistico è strutturata anch'essa come un modulo nativo di PyTorch e i parametri addestrabili (gli angoli delle rotazioni) sono dichiarati in modo tale che quest'ultimo possa riconoscerli e aggiornarli durante l'ottimizzazione.

3.2 Gestione dei Dati e Monitoraggio

Per supportare l'addestramento e la valutazione, sono state utilizzate diverse librerie ausiliarie.

La gestione del dataset MNIST è affidata a **TorchVision** [34]. Questa libreria gestisce il caricamento, il download automatico e le trasformazioni preliminari dei dati. Le trasformazioni includono la conversione delle immagini in tensori e il ridimensionamento alla dimensione target. Infine, i dati vengono forniti in un "batch" e mescolati casualmente ad ogni epoca.

Per garantire la flessibilità e la riproducibilità degli esperimenti, l'implementazione utilizza un parser per gli argomenti da riga di comando. Questo permette di definire iperparametri cruciali (come il learning rate, il numero di iterazioni e il tipo di ottimizzatore) al momento dell'esecuzione, senza la necessità di modificare il codice sorgente.

Il monitoraggio dell'addestramento e la valutazione quantitativa sono gestiti da due componenti principali:

- **TensorBoard** [35]: Utilizzato per il logging e la visualizzazione in tempo reale dal terminale. L'implementazione registra l'andamento delle funzioni di costo e, a intervalli regolari, salva una griglia (grid) di immagini generate.
- **TorchMetrics** [36]: Per le metriche quantitative (FID e LPIPS), viene utilizzata questa libreria. Vengono utilizzate due delle sue classi specifiche per calcolare i punteggi. Una funzione ausiliaria di preparazione, descritta nel Capitolo 2.3, si occupa di formattare le immagini generate secondo i requisiti specifici di queste metriche.

3.3 Implementazione Pratica

Le sezioni seguenti mostrano i frammenti di codice fondamentali che realizzano l'architettura ibrida. La base del codice iniziale è stata tratta dal lavoro di Ellis [37], successivamente adattata ed estesa per l'implementazione di un nuovo dataset e le specifiche configurazioni sperimentali di questa tesi.

3.3.1 Definizione del Discriminatore

Il Discriminatore è un modulo PyTorch standard (`nn.Module`) che incapsula una semplice rete MLP. L'architettura è definita utilizzando `nn.Sequential`, ponendo in sequenza strati lineari (`nn.Linear`) e attivazioni ReLU, con una Sigmoid finale per produrre l'output di probabilità.

```
1 class Discriminator(nn.Module):  
2
```

```
3     def __init__(self):
4         super().__init__()
5
6         self.model = nn.Sequential(
7             nn.Linear(image_size * image_size, 64),
8             nn.ReLU(),
9             nn.Linear(64, 16),
10            nn.ReLU(),
11            nn.Linear(16, 1),
12            nn.Sigmoid(),
13        )
14
15    def forward(self, x):
16        return self.model(x)
```

Listato 3.1: Implementazione del Discriminatore classico.

3.3.2 Definizione del PQC

Il cuore quantistico del generatore è definito dalla funzione `create_quantum_circuit` (Listato 3.2). Questa funzione utilizza PennyLane per definire il dispositivo (`qml.device`) e il "QNode" (`@qml.qnode`).

Il QNode è un circuito parametrizzato che accetta due input: il rumore (noise) e i pesi addestrabili (weights). Il decoratore `@qml.qnode` specifica il dispositivo di esecuzione e il metodo di differenziazione che abilita il calcolo del gradiente per la retropropagazione.

```
1 def create_quantum_circuit(n_qubits, n_a_qubits, q_depth):
2     # Quantum simulator
3     dev = qml.device("default.qubit", wires=n_qubits)
4
5     @qml.qnode(dev, diff_method="parameter-shift")
```

```

6     def quantum_circuit(noise, weights):
7         weights = weights.reshape(q_depth, n_qubits, 2)
8
9         # Initialise latent vectors
10        for i in range(n_qubits):
11            qml.RY(noise[i], wires=i)
12            qml.RX(noise[i + n_qubits], wires=i)
13
14        # Repeated layer
15        for i in range(q_depth):
16            # Parameterised layer
17            for y in range(n_qubits):
18                qml.RX(weights[i][y][0], wires=y)
19                qml.RY(weights[i][y][1], wires=y)
20
21            # Entanglement
22            for y in range(n_qubits - 1):
23                qml.CZ(wires=[y, y + 1])
24
25        return qml.probs(wires=list(range(n_qubits)))
26
27    return quantum_circuit

```

Listato 3.2: Definizione del PQC (QNode).

3.3.3 Definizione del Generatore

L'integrazione tra i due framework principali avviene all'interno della classe `PatchQuantumGenerator` nel Listato 3.3. Questa classe eredita da `nn.Module` di PyTorch, comportandosi come qualsiasi strato di rete neurale classica.

La funzione `partial_measure` nel Listato 3.4 funge da ponte: esegue il QNode di PennyLane e poi processa l'array di probabilità risultante utiliz-

zando operazioni tensoriali di PyTorch (slicing, `torch.sum`, `torch.max`) per implementare la misurazione parziale.

```
1 class PatchQuantumGenerator(nn.Module):
2
3     def __init__(self, n_generators, n_a_qubits, q_delta=1):
4
5         super().__init__()
6
7         total_pixels = image_size * image_size
8         if total_pixels % n_generators != 0:
9             raise ValueError(f"n_generators ({n_generators})
10         ↪ must divide total_pixels")
11
12         pixels_per_generator = total_pixels // n_generators
13         n_qubits = int(math.log2(pixels_per_generator)) +
14         ↪ n_a_qubits
15
16         self.n_generators = n_generators
17         self.n_qubits = n_qubits
18         self.n_a_qubits = n_a_qubits
19         self.q_depth = 6
20
21         self.quantum_circuit_fn = create_quantum_circuit(
22         ↪ n_qubits, n_a_qubits, self.q_depth)
23
24         self.q_params = nn.ParameterList(
25             [
26                 nn.Parameter(q_delta * torch.rand(self.
27                 ↪ q_depth * n_qubits * 2), requires_grad=True)
28                 for _ in range(n_generators)
29             ]
30         )
```

```
27
28     def forward(self, x):
29         patch_size = 2 ** (self.n_qubits - self.n_a_qubits)
30
31         images_list = []
32         for params in self.q_params:
33             patch_list = []
34             for elem in x:
35                 q_out = partial_measure(elem, params, self.
↪ quantum_circuit_fn,
36                                     self.n_qubits, self.
↪ n_a_qubits).float().unsqueeze(0)
37                 patch_list.append(q_out)
38                 patches = torch.cat(patch_list, dim=0)
39
40                 images_list.append(patches)
41
42         images = torch.cat(images_list, dim=1)
43         return images
```

Listato 3.3: Implementazione del Generatore.

```
1 def partial_measure(noise, weights, quantum_circuit_fn,
↪ n_qubits, n_a_qubits):
2     probs = quantum_circuit_fn(noise, weights)
3     probsgiven0 = probs[: (2 ** (n_qubits - n_a_qubits))]
4     probsgiven0 /= torch.sum(probs)
5     probsgiven = probsgiven0 / torch.max(probsgiven0)
6     return probsgiven
```

Listato 3.4: Funzione ausiliaria per la misurazione parziale.

Nel costruttore (`__init__`) del generatore, i parametri addestrabili (`self.q_params`) vengono definiti come `nn.ParameterList`. Questa procedura registra gli angoli di rotazione tramite il sistema di ottimizzazione di PyTorch.

Successivamente, il metodo `forward` definisce la logica di esecuzione: itera sui `q_params` (un set di parametri per ogni patch), chiama la funzione `partial_measure` e infine assembla le patch risultanti in un'unica immagine utilizzando `torch.cat`.

3.3.4 Ciclo di Addestramento

Infine, viene definito il ciclo di addestramento (Listato 3.5). Questo codice, gestito interamente da PyTorch, coordina il training avversario.

Durante le iterazioni si distinguono due fasi:

1. Addestramento del Discriminatore: Vengono calcolate le loss sui dati reali (`errD_real`) e falsi (`errD_fake`). È fondamentale l'uso di `fake_data.detach()`, in quanto questa funzione interrompe il tracciamento del gradiente all'indietro verso il generatore, assicurando che l'ottimizzatore `optD` aggiorni solo i pesi del discriminatore.
2. Addestramento del Generatore: Viene calcolata una nuova loss per il generatore, confrontando l'output del discriminatore sulle immagini false con le etichette "reali". Quando viene chiamata `errG.backward()`, viene propagato il gradiente all'indietro. Quando quest'ultimo raggiunge i parametri quantistici, PyTorch invoca automaticamente la regola "parameter-shift" definita tramite PennyLane per calcolare il gradiente quantistico. Successivamente, `optG.step()` aggiorna i parametri del PQC.

```
1 while True:
2     for i, (data, _) in enumerate(dataloader):
3
4         # Data for training the discriminator
5         data = data.reshape(-1, image_size * image_size)
6         real_data = data.to(device)
7
8         # Noise following a uniform distribution in range [0,
9         ↪ pi/2)
10        noise = torch.rand(batch_size, noise_dim, device=
11        ↪ device) * math.pi / 2
12        fake_data = generator(noise)
13
14        # --- Training the discriminator ---
15        discriminator.zero_grad()
16        outD_real = discriminator(real_data).view(-1)
17        outD_fake = discriminator(fake_data.detach()).view
18        ↪ (-1)
19
20        errD_real = criterion(outD_real, real_labels)
21        errD_fake = criterion(outD_fake, fake_labels)
22        errD_real.backward()
23        errD_fake.backward()
24
25        errD = errD_real + errD_fake
26        optD.step()
27
28        # --- Training the generator ---
29        generator.zero_grad()
30        outD_fake = discriminator(fake_data).view(-1)
31        errG = criterion(outD_fake, real_labels)
32        errG.backward()
```

```
30     optG.step()
31
32     counter += 1
33
34     # ... (Logging and metric calculation) ...
35
36     if counter == args.num_iter:
37         break
38 if counter == args.num_iter:
39     break
```

Listato 3.5: Ciclo di addestramento (estratto dalla funzione main).

Capitolo 4

Risultati

In questo capitolo vengono presentati i risultati sperimentali ottenuti dall'addestramento delle diverse configurazioni architetture della QGAN. L'analisi è organizzata in due sezioni: la prima mostra esempi visivi delle immagini generate durante il training, mentre la seconda presenta l'andamento quantitativo delle metriche FID e LPIPS per le varie configurazioni testate. L'architettura di "Default" per le immagini 16×16 è già presentata nel Capitolo 2 e analizzata nel Capitolo 3, mentre quella per le immagini 32×32 è semplicemente una versione scalata di quest'ultima (2.1.1). Le altre configurazioni testate sono variazioni di questi modelli di base, come specificato di seguito.

4.1 Configurazioni Sperimentali

Per i test su immagini 16×16 pixel, sono state confrontate le seguenti varianti:

- **Default:** Configurazione di riferimento: 4 sub-generatori, 1 ancilla, entanglement C_Z e rotazioni R_X e R_Y .
- **8 generatori:** Configurazione con 8 sub-generatori invece di 4.

- **2 Qubit Ancillari:** Configurazione "Default" ma con 2 qubit ancillari invece di 1, aumentando dunque il numero di qubit utilizzati.
- **Entanglement-CNOT:** Variante con entanglement *CNOT* al posto del gate C_Z nell'ansatz del PQC.
- **Entanglement Parametrico:** Variante con entanglement parametrico C_{RX} al posto del gate C_Z .
- **MLP:** Generatore ibrido QGAN-MLP con post-processing MLP applicato all'output quantistico (Capitolo 2.1.4).
- **CNN:** Generatore ibrido QGAN-CNN, con post-processing convoluzionale applicato all'output quantistico (2.1.4).

Per i test su immagini 32×32 pixel, le configurazioni sono:

- **Default:** Configurazione di riferimento, ma scalata per renderla compatibile.
- **Discriminatore-5L:** Variante con Discriminatore a 5 layer invece di 3.
- **Discriminatore-7L:** Variante con Discriminatore a 7 layer.
- **8 Generatori:** Configurazione con 8 sub-generatori invece di 4, con dimensioni scalate rispetto al corrispettivo 16×16 .
- **2 Qubit Ancillari:** Configurazione con 2 Qubit Ancillari (PQC a 10 qubit)
- **MLP:** Configurazione con Generatore ibrido QGAN-MLP (Capitolo 2.1.4).

4.2 Risultati Qualitativi

La prima analisi è qualitativa, basata su un'ispezione visiva dei campioni generati. Prima di presentare i risultati, è utile visualizzare un esempio di immagine reale utilizzata per l'addestramento, che costituisce il target di riferimento per la generazione:

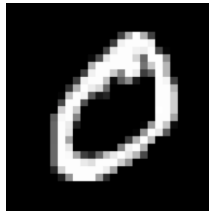


Figura 4.1: Esempio di cifra reale dal dataset MNIST.

Per i test con immagini 16×16 , tutte le immagini mostrate sono state campionate al termine dell'addestramento (500 iterazioni). La Figura 4.2 riporta i risultati ottenuti dalla configurazione Default, che ha prodotto le immagini graficamente più riconoscibili. Le Figure 4.3, 4.4, 4.5, 4.6 illustrano ulteriori varianti a 16×16 , anch'esse capaci di generare strutture definite, seppur con livelli diversi di coerenza. I risultati delle configurazioni selezionate sono presentati tramite griglie composte da 8 campioni ciascuna.

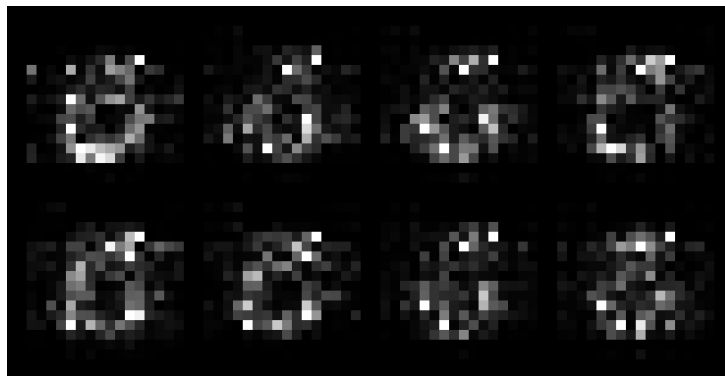


Figura 4.2: Immagini generate dalla configurazione Default 16x16.

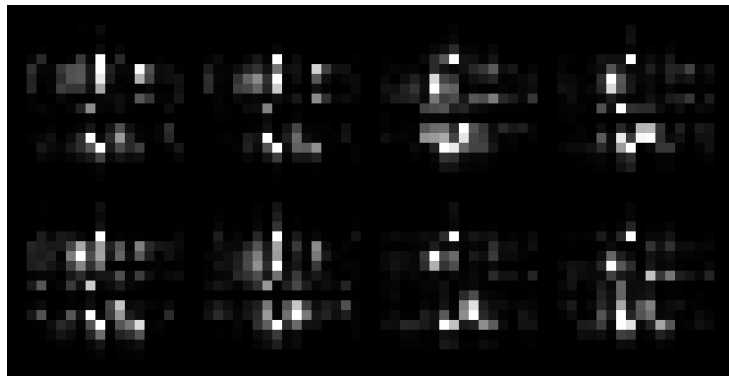


Figura 4.3: Immagini generate con entanglement parametrico CRX.

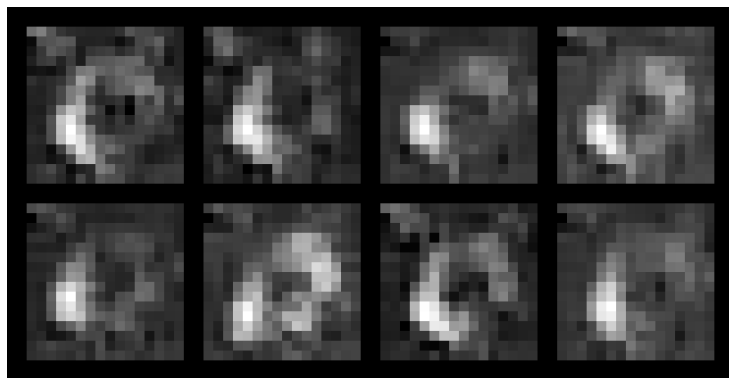


Figura 4.4: Immagini generate con generatore QGAN-CNN.

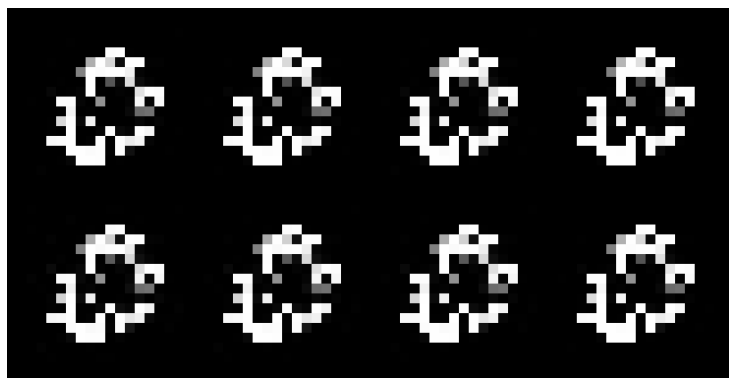


Figura 4.5: Immagini generate con generatore QGAN-MLP.

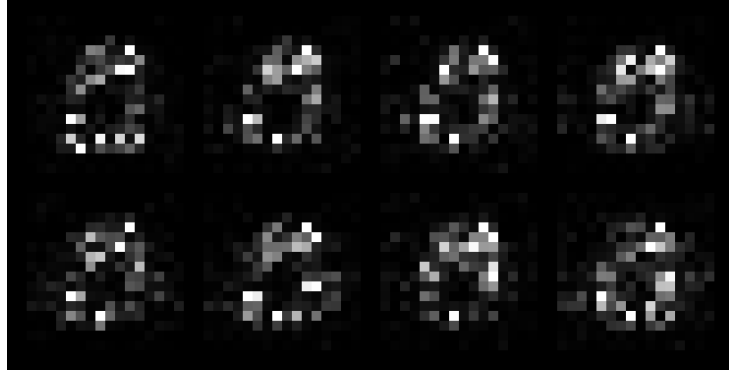


Figura 4.6: Immagini generate con entanglement CNOT.

Successivamente, vengono presentati i risultati visivi per alcuni test a 32×32 pixel. A questa risoluzione più elevata, la configurazione Default (Figura 4.7) non ha replicato le stesse performance ottenute nelle 16×16 e alcune varianti architetturali (Figure 4.8, 4.9, 4.10, 4.11) hanno invece mostrato risultati più promettenti. Le immagini delle configurazioni Default, con 8 generatori e con 2 qubit ancillari, sono state campionate all'iterazione 300 anziché 500, in quanto mostrano una convergenza migliore rispetto a quelle successive.

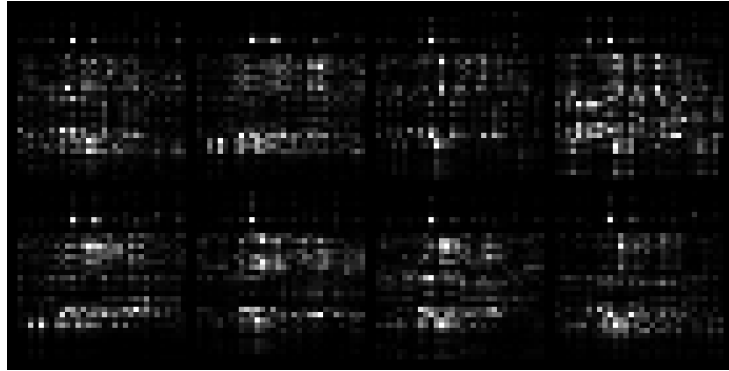


Figura 4.7: Immagini generate dalla configurazione Default 32x32.

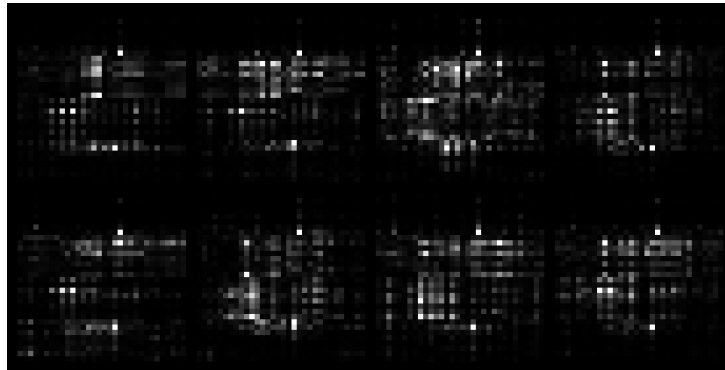


Figura 4.8: Immagini generate con discriminatore a 5 layer.



Figura 4.9: Immagini generate con generatore ibrido QGAN-MLP.

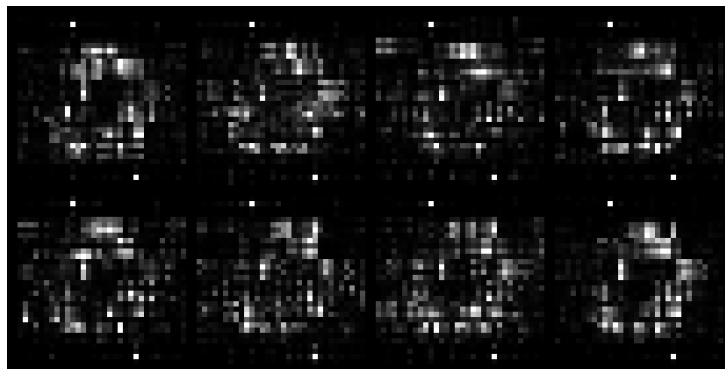


Figura 4.10: Immagini generate con 8 patch.

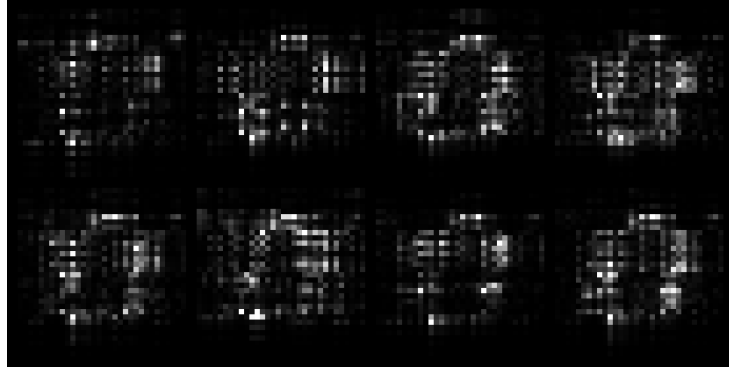


Figura 4.11: Immagini generate con 2 qubit ancillari.

4.3 Risultati Quantitativi

L'analisi quantitativa si concentra sull'evoluzione delle metriche FID (valori più bassi indicano una qualità superiore) e LPIPS (valori più alti corrispondono a una maggiore diversità percettiva), calcolate ogni 50 iterazioni durante il processo di addestramento.

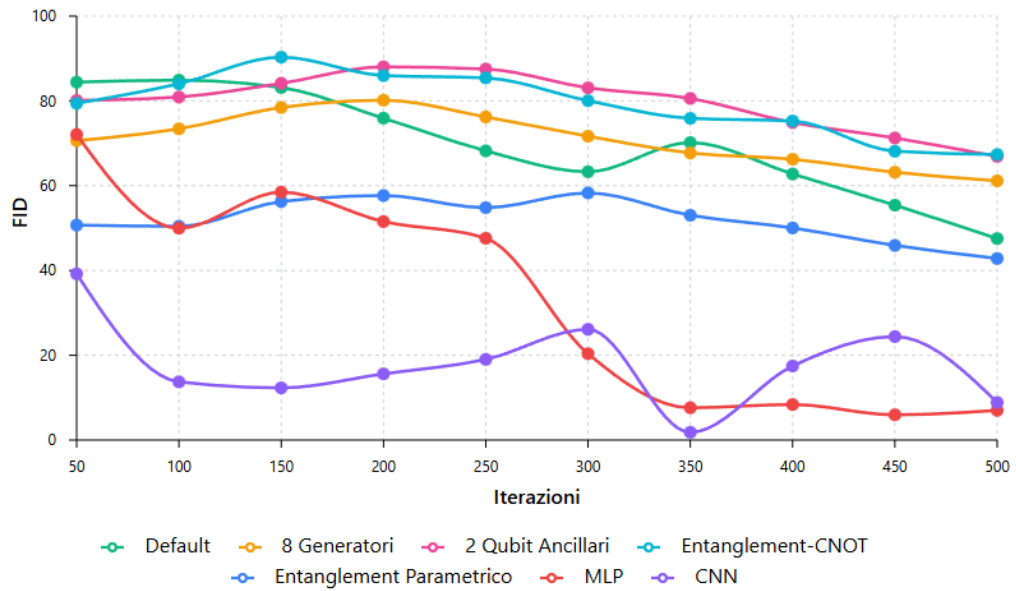


Figura 4.12: Andamento del FID Score in funzione delle iterazioni di addestramento per immagini 16×16 pixel.

Come visibile nella Figura 4.12, l'andamento del FID per i modelli 16×16 mostra un chiaro trend decrescente per quasi tutte le configurazioni, indicando che il generatore sta effettivamente imparando a replicare la distribuzione dei dati reali.

Le varianti ibride QGAN-CNN e QGAN-MLP raggiungono i punteggi più bassi, indicando che il post-processing tramite reti neurali classiche consente di raffinare efficacemente l'output dei circuiti quantistici. Questo vantaggio quantitativo deriva dalla capacità dei layer classici di apprendere trasformazioni non lineari complesse che mappano l'output quantistico verso distribuzioni più simili ai dati reali, riducendo le distanze statistiche misurate dal FID.

La configurazione Default si posiziona comunque a livelli competitivi, raggiungendo un buon equilibrio tra performance metriche e qualità visiva percettiva. Come già osservato qualitativamente, questa configurazione produce le immagini più riconoscibili e coerenti, dimostrando che un FID leggermente superiore non implica necessariamente una qualità percettiva inferiore.

Le configurazioni con entanglement CNOT ed entanglement parametrico C_{RX} presentano andamenti differenti: come ci si può aspettare, avendo un numero di parametri addestrabili maggiore, l'entanglement parametrico ottiene un FID migliore nonostante la qualità grafica percepita sia leggermente inferiore. Questo disallineamento suggerisce che la maggiore flessibilità del gate C_{RX} permette di avvicinarsi alla distribuzione target senza migliorare la qualità visiva. La configurazione con 2 qubit ancillari e quella con 8 sub-generatori producono FID scores lievemente superiori rispetto al Default, confermando che l'aumento di complessità del circuito non è stato sfruttato efficacemente dall'ottimizzatore e che la frammentazione in patch più piccole limita la capacità del modello di catturare le feature necessarie per una

generazione ottimale.

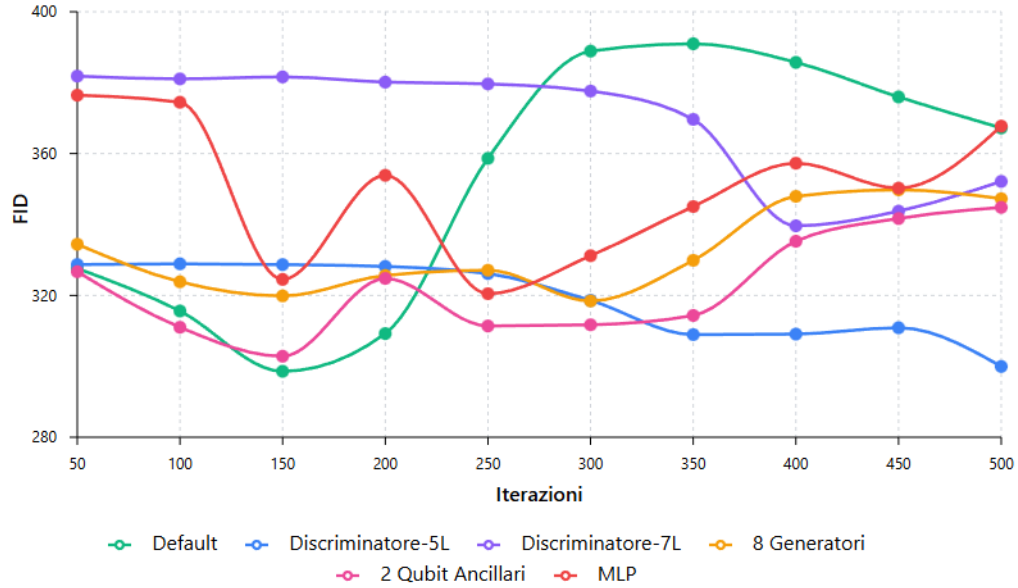


Figura 4.13: Andamento del FID Score in funzione delle iterazioni di addestramento per immagini 32×32 pixel.

Nei test 32×32 (Figura 4.13), i punteggi FID sono generalmente più alti. Analizzandone i risultati, emerge un panorama simile ma con alcune differenze significative. A differenza dei test a 16×16 , l'andamento del FID per le configurazioni 32×32 non mostra generalmente un trend in discesa continua: dopo una lieve diminuzione iniziale, i valori tendono ad aumentare superata una certa soglia di iterazioni, indicando difficoltà nel mantenere la convergenza. La configurazione Default, che aveva ottenuto buoni risultati a risoluzione inferiore, è affetta da questo problema e ha punteggi decisamente peggiori. Questo evidenzia che il semplice scaling dell'architettura non è sufficiente per gestire la maggiore complessità spaziale delle immagini 32×32 . Le configurazioni con discriminatori più profondi mostrano comportamenti contrastanti. Il discriminatore a 5 layer ottiene i migliori risultati tra tutte

le configurazioni testate, raggiungendo un buon equilibrio tra qualità grafica e metrica. Il discriminatore a 7 layer, invece, non migliora rispetto al 5 layer o lo fa troppo lentamente: l'eccessiva capacità discriminativa sembra rendere il training più difficile, suggerendo che un discriminatore troppo potente possa ostacolare l'apprendimento del generatore. La configurazione ibrida MLP, nonostante le immagini generate siano graficamente buone, ha un FID score elevato e non raggiunge i livelli del discriminatore a 5 layer. Le configurazioni con 2 qubit ancillari e 8 sub-generatori confermano le difficoltà precedentemente osservate

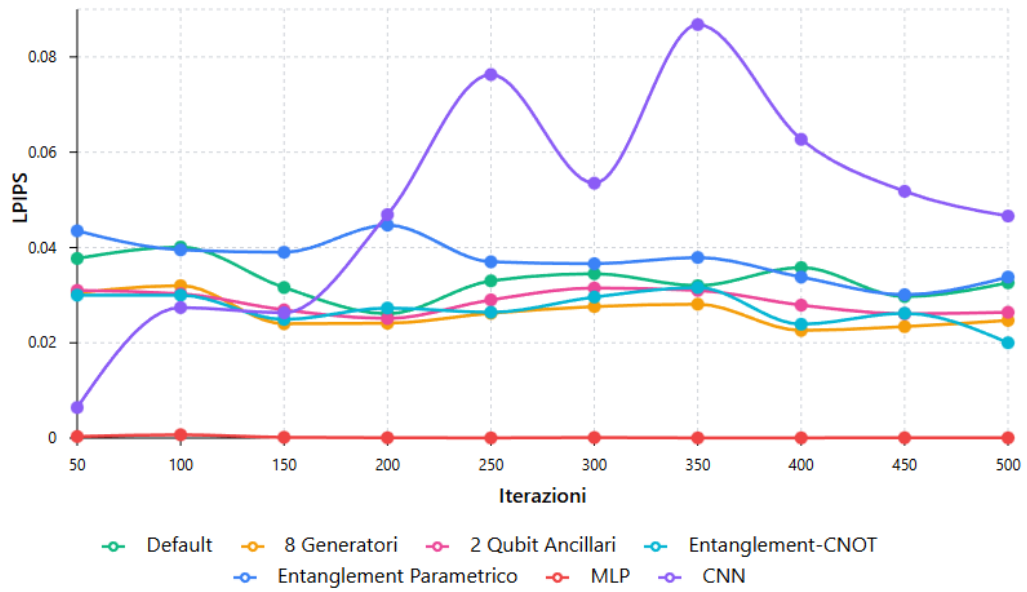


Figura 4.14: Andamento del LPIPS Score in funzione delle iterazioni di addestramento per immagini 16×16 pixel.

La metrica LPIPS (Figura 4.14) rivela una caratteristica comune a tutte le configurazioni testate: i valori sono complessivamente molto bassi e tendono a diminuire durante l'addestramento, indicando che i generatori producono campioni con diversità molto limitata. La configurazione CNN mostra valori

LPIPS leggermente più alti rispetto alle altre varianti, ma anch'essa presenta un andamento calante nel corso del training. Tutte le altre configurazioni presentano valori molto simili tra loro e ugualmente bassi. Questi risultati evidenziano che la capacità di generare campioni diversificati rimane ridotta anche quando le performance in termini di FID migliorano. Questo disaccoppiamento tra qualità statistica e diversità percettiva rappresenta un aspetto critico da considerare nella valutazione complessiva.

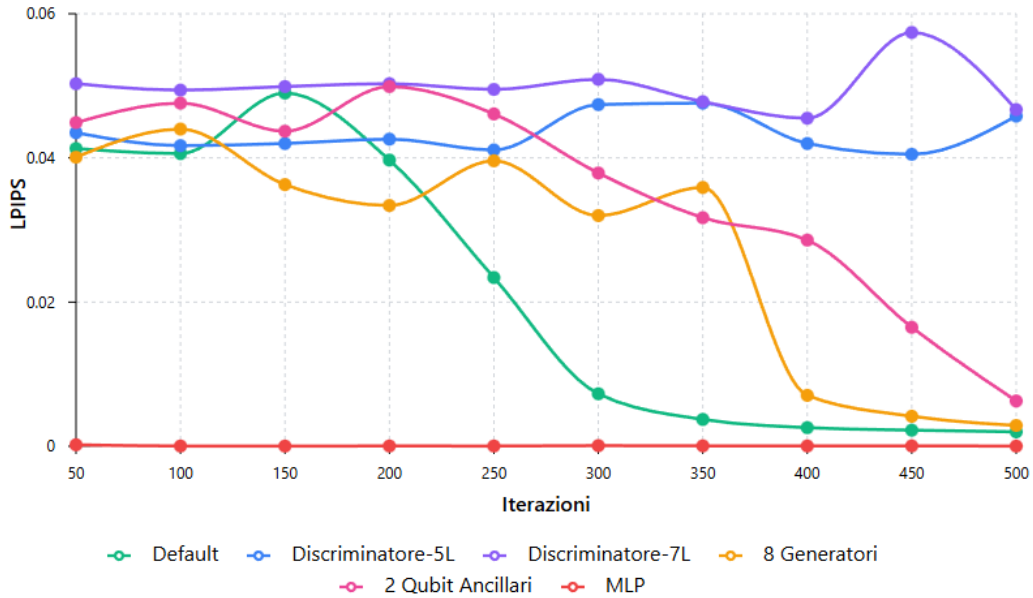


Figura 4.15: Andamento del LPIPS Score in funzione delle iterazioni di addestramento per immagini 32×32 pixel.

Anche con una risoluzione maggiore, si osserva in Figura 4.15 un comportamento simile a quello riscontrato nelle configurazioni 16×16 , con valori generalmente bassi. I discriminatori più profondi mantengono punteggi LPIPS superiori rispetto alle altre configurazioni, sebbene i valori rimangano comunque contenuti. Le configurazioni Default, 2 ancilla e 8 generatori mostrano invece un calo troppo marcato della diversità percettiva durante il

training. In sintesi, i risultati a 32x32 evidenziano che:

- L'aumento della capacità discriminativa è fondamentale per gestire risoluzioni superiori, ma esiste un punto ottimale oltre il quale ulteriori incrementi non portano benefici o rallentano l'apprendimento.
- Il semplice scaling delle architetture quantistiche non è sufficiente, come dimostrato dal deterioramento delle performance del Default dopo 300 iterazioni.
- L'aumento della complessità dei circuiti (più qubit o più patch) non garantisce miglioramenti senza adeguate strategie di ottimizzazione.
- La diversità percettiva rimane una sfida aperta per tutte le configurazioni testate, con le architetture più performanti in termini di FID che riescono comunque a mantenere una varietà relativamente superiore.

Conclusioni

In questa Tesi è stata progettata, implementata e valutata un'architettura ibrida di Quantum Generative Adversarial Network (QGAN) applicata alla generazione di immagini, utilizzando il dataset MNIST. L'obiettivo principale era esplorare la fattibilità e le prestazioni di un sistema ibrido che integra framework di calcolo quantistico e classico, addestrando un modello generativo basato su Parameterized Quantum Circuit (PQC). Per superare le limitazioni dei simulatori attuali, è stato adottato un approccio "patch-based" per la generazione di immagini a risoluzioni di 16×16 e 32×32 pixel, in cui l'output finale viene assemblato combinando i risultati di multipli sub-generatori quantistici. La sperimentazione ha coinvolto diverse varianti architetturali, testando modifiche al numero di generatori, ai qubit ancillari e alla tipologia di entanglement, oltre all'introduzione di reti neurali classiche (CNN e MLP) per raffinare il generatore e i discriminatori a diversa profondità.

L'analisi dei risultati ha fornito un quadro chiaro delle potenzialità e dei limiti di questo approccio. Il risultato più significativo è che l'apprendimento è stato dimostrato con successo: l'andamento della metrica FID ha mostrato un chiaro trend decrescente in quasi tutti i test 16×16 , indicando che il generatore quantistico stava effettivamente imparando a replicare la distribuzione statistica dei dati reali. Le configurazioni ibride (QGAN-CNN e MLP) hanno ottenuto i punteggi migliori, dimostrando la validità del sistema e la

precisione delle reti classiche.

Tuttavia, sono emerse due criticità principali. La prima riguarda la scalabilità: il passaggio alla risoluzione 32×32 si è rivelato complesso, evidenziando come il semplice scaling dell'architettura non fosse sufficiente. La configurazione di base, efficace a risoluzione inferiore, ha subito un deterioramento delle prestazioni, mitigato solo parzialmente dall'adozione di un discriminatore più profondo (5 layer), il quale si è rivelato necessario per guidare l'addestramento in uno spazio delle feature più complesso. La seconda e più rilevante criticità riguarda la diversità percettiva dei campioni generati. I punteggi LPIPS sono risultati sistematicamente molto bassi per tutte le configurazioni testate, indicando che i generatori, anche quelli che producono immagini con FID competitivo, convergono verso regioni ristrette dello spazio latente, generando campioni con scarsa variabilità. Tale fenomeno è risultato particolarmente evidente nella configurazione QGAN-MLP, con la generazione di campioni praticamente identici. Inoltre, l'aumento della complessità del circuito quantistico (tramite l'aggiunta di qubit ancillari o la frammentazione in otto patch) non ha comportato benefici tangibili, suggerendo che l'espansione del circuito introduca difficoltà di ottimizzazione che non vengono automaticamente compensate da una maggiore espressività.

Questo lavoro può fornire una base per ulteriori ricerche, evidenziando le sfide da affrontare. Nel futuro, le sperimentazioni in questo campo dovrebbero indirizzarsi verso:

- **Miglioramento della diversità:** Per superare la variabilità limitata nei campioni generati, sarà necessaria la sperimentazione di funzioni di costo alternative o strategie di addestramento differenti, al fine di aumentare la diversificazione dell'output.
- **Scalabilità e Generalizzazione:** L'applicazione del metodo a dataset più

complessi e a risoluzioni superiori richiederà lo studio di tecniche più efficienti per gestire l'elevata dimensionalità dei dati visivi.

- Validazione su Hardware Reale: Il passo più significativo sarà il testing di queste architetture su processori quantistici reali. Questo passaggio è fondamentale per verificare se, al netto del rumore hardware, il generatore quantistico possa offrire un vantaggio computazionale o qualitativo concreto rispetto alle controparti classiche.

In conclusione, questo lavoro ha dimostrato che le QGAN possono generare immagini riconoscibili con metriche di qualità competitive, ma ha anche evidenziato alcune sfide legate alla diversità dei campioni e alla scalabilità a risoluzioni superiori. Il superamento delle sfide emerse rappresenta la direzione chiave per rendere il Quantum Machine Learning un'alternativa concreta per la generazione di contenuti sintetici.

Bibliografia

- [1] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [2] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [3] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
- [4] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 20(2):215–232, 1958.
- [5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [6] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT’2010: 19th International Con-*

- ference on Computational Statistics Paris France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.
- [7] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.
- [8] Sai Balaji. Binary image classifier cnn using tensorflow.
- [9] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.
- [10] Youssef Kossale, Mohammed Airaj, and Aziz Darouichi. Mode collapse in generative adversarial networks: An overview. In *2022 8th International Conference on Optimization and Applications (ICOA)*, pages 1–6. IEEE, 2022.
- [11] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.
- [12] Ryszard Horodecki, Paweł Horodecki, Michał Horodecki, and Karol Horodecki. Quantum entanglement. *Reviews of modern physics*, 81(2):865–942, 2009.
- [13] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin,

- and Harald Weinfurter. Elementary gates for quantum computation. *Physical review A*, 52(5):3457, 1995.
- [14] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015.
- [15] Manuela Weigold, Johanna Barzen, Frank Leymann, and Marie Salm. Encoding patterns for quantum algorithms. *IET Quantum Communication*, 2(4):141–152, 2021.
- [16] Martin Larocca, Supanut Thanasilp, Samson Wang, Kunal Sharma, Jacob Biamonte, Patrick J Coles, Lukasz Cincio, Jarrod R McClean, Zoë Holmes, and Marco Cerezo. Barren plateaus in variational quantum computing. *Nature Reviews Physics*, pages 1–16, 2025.
- [17] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3):032331, 2019.
- [18] Andrea Mari, Thomas R Bromley, and Nathan Killoran. Estimating the gradient and higher-order derivatives on quantum hardware. *Physical Review A*, 103(1):012405, 2021.
- [19] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. *Physical review letters*, 121(4):040502, 2018.
- [20] Yann LeCun. The mnist database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>, 1998.
- [21] Daniel Silver, Tirthak Patel, William Cutler, Aditya Ranjan, Harshitta Gandhi, and Devesh Tiwari. Mosaiq: Quantum generative adversarial

- networks for image generation on nisc computers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7030–7039, 2023.
- [22] Quangong Ma, Chaolong Hao, NianWen Si, Geng Chen, Jiale Zhang, and Dan Qu. Quantum adversarial generation of high-resolution images. *EPJ Quantum Technology*, 12(1):3, 2025.
- [23] Shu Lok Tsang, Maxwell T West, Sarah M Erfani, and Muhammad Usman. Hybrid quantum–classical generative adversarial network for high-resolution image generation. *IEEE Transactions on Quantum Engineering*, 4:1–19, 2023.
- [24] Shouvanik Chakrabarti, Huang Yiming, Tongyang Li, Soheil Feizi, and Xiaodi Wu. Quantum wasserstein generative adversarial networks. *Advances in Neural Information Processing Systems*, 32, 2019.
- [25] Aaron Mark Thomas, Harry Youel, and Sharu Theresa Jose. Vae-qwgan: addressing mode collapse in quantum gans via autoencoding priors. *Quantum Machine Intelligence*, 7(2):91, 2025.
- [26] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [27] H. Huang, Y. Du, M. Gong, Y. Zhao, Y. Wu, C. Wang, S. Li, F. Liang, J. Lin, Y. Xu, R. Yang, T. Liu, M. Hsieh, H. Deng, H. Rong, C. Peng, C. Lu, Y. Chen, D. Tao, X. Zhu, and J. Pan. Experimental quantum generative adversarial networks for image generation. *arXiv:2010.06201*, 2021.
- [28] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule

- converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.
- [29] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
- [30] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.
- [31] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 1998.
- [32] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahna-waz Ahmed, Vishnu Ajith, M. Sohaib Alam, Guillermo Alonso-Linaje, B. AkashNarayanan, Ali Asadi, et al. PennyLane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018. Version 0.41.1.
- [33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035, 2019. Version 2.7.1.

-
- [34] TorchVision maintainers and contributors. Torchvision: Pytorch’s computer vision library. <https://github.com/pytorch/vision>, 2016. Version 0.22.1.
 - [35] TensorBoard Development Team. Tensorboard: Tensorflow’s visualization toolkit. <https://www.tensorflow.org/tensorboard>, 2015. Version 2.20.0.
 - [36] TorchMetrics Contributors. Torchmetrics: Machine learning metrics for distributed, scalable pytorch applications. <https://github.com/Lightning-AI/torchmetrics>, 2020. Version 1.7.4.
 - [37] J. Ellis. Quantum gans, 2022.
 - [38] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, and K. Li. A Parallel Random Forest Algorithm for Big Data in a Spark Cloud Computing Environment. *IEEE Transactions on Parallel and Distributed Systems*, 28(4):919–933, 2017.
 - [39] F. Brandao and K. Svore. Quantum speed-ups for semidefinite programming. In *Proc. of the IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 415–426, 2017.
 - [40] G. Winskel and M. Nielsen. *Handbook of Logic in Computer Science (Vol. 4)*, chapter Models for Concurrency, pages 1–148. Oxford University Press, 1995.