ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dipartimento di Informatica - Scienza e Ingegneria

Corso di Laurea in Ingegneria e Scienze-Informatiche

Sviluppo ed analisi di modelli generativi di immagini basati su diffusione e autoencoder variazionali

Relatore:
Damiana Lazzaro

Presentata da:
Kevin Shimaj

Sessione Unica

Anno Accademico 2024/2025

Indice

Alla mia famiglia, per il vostro amore incondizionato e per tutti i sacrifici che mi hanno permesso di arrivare fin qui.

Introduzione

L'Intelligenza Artificiale Generativa si sta affermando come una delle tecnologie più rivoluzionarie degli ultimi anni, con un impatto crescente in ambiti quali automazione dei processi, creazione di contenuti, sviluppo software e supporto decisionale.[1] In particolare, nel 2024 una larga maggioranza di organizzazioni ha dichiarato di adottare sistemi di AI in almeno una delle proprie funzioni aziendali, segnando un significativo aumento rispetto all'anno precedente. Parallelamente, l'utilizzo di strumenti di AI generativa si è rapidamente diffuso in diverse aree di business, registrando una crescita importante su base annua. Tra gli utenti, questi modelli generativi hanno permesso di risparmiare mediamente una quantità apprezzabile di tempo nel totale delle ore lavorative settimanali [1], [2], [3].

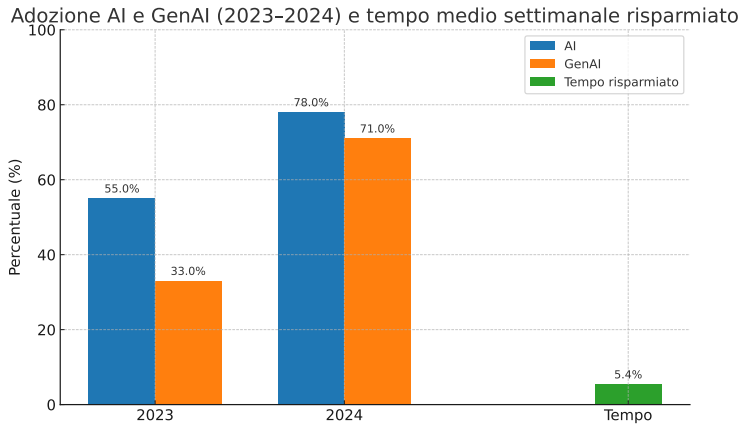

Figura 1: Adozione dell'AI e dell'AI generativa (2023 vs 2024), e tempo medio settimanale risparmiato grazie alla GenAI.

In questo contesto, la presente tesi si propone di contribuire allo studio dei modelli generativi e alla relativa implementazione. È stato innanzitutto realizzato un *Denoising Diffusion Probabilistic Model* (DDPM), affiancato dalla progettazione di un *Variational Autoencoder* (VAE), al fine di esplorare i punti di forza e i limiti di ciascun approccio. Su questa base è stata sviluppata un'architettura ibrida, denominata *Diffuse-VAE*, che combina la capacità dei VAE di apprendere spazi latenti compatti

con la potenza dei DDPM nel generare campioni ad alta fedeltà. L'analisi delle prestazioni è stata condotta sia in termini quantitativi, attraverso metriche standard quali PSNR, SSIM e MSE, sia in termini qualitativi, mediante l'ispezione visiva dei campioni generati. Questo duplice approccio ha permesso di valutare in maniera bilanciata efficacia, limiti e potenzialità delle soluzioni proposte. Il lavoro è stato sviluppato principalmente in `Python` utilizzando il framework `PyTorch`, scelto per la flessibilità del modello a tensori con *autograd*, il supporto GPU, l'ecosistema ricco e l'ampia diffusione nella comunità di ricerca [4].
La tesi è così organizzata:

- Il **Capitolo 1** introduce il contesto dell'intelligenza artificiale generativa e ne ripercorre l'evoluzione storica, evidenziando i principali approcci e le loro caratteristiche.
- Il **Capitolo 2** è dedicato ai modelli di diffusione,con una descrizione approfondita dei principi teorici e dei meccanismi che ne permettono la generazione progressiva.
- Il **Capitolo 3** approfondisce i *Variational Autoencoder* (VAE), illustrandone la formulazione probabilistica e le tecniche di addestramento.
- Il **Capitolo 4** presenta l'architettura ibrida **DiffuseVAE**, analizzando nel dettaglio come il modello di diffusione (DDPM) venga condizionato dalla ricostruzione generata dal VAE. Vengono, inoltre, esplorate le interazioni tra i due modelli e il loro impatto sulla qualità finale delle immagini generate
- Il **Capitolo 5** descrive le scelte implementative effettuate e le tecniche di ottimizzazione degli iperparametri utilizzate.
- Il **Capitolo 6** è dedicato all'analisi delle prestazioni, condotta attraverso metriche quantitative e confronti qualitativi tra i modelli.

Capitolo 1

L'AI Generativa

In questo capitolo viene presentata l'evoluzione dei modelli generativi di immagini, ripercorrendo le principali tappe che hanno condotto dalle prime architetture fino agli approcci più recenti [5], [6]. Oltre agli aspetti tecnici, viene inoltre offerta una panoramica sul quadro legislativo attualmente in vigore, che definisce limiti e responsabilità nell'impiego dell'AI generativa [7].

1.1 Panoramica ed evoluzione

Negli ultimi anni l'Intelligenza Artificiale Generativa ha compiuto progressi notevoli, passando da semplici modelli statistici a sofisticate architetture neurali in grado di generare immagini, audio e testo di qualità comparabile a quella prodotta da esseri umani. Questo sviluppo è stato guidato da una combinazione di fattori: la disponibilità di grandi dataset, l'aumento della potenza computazionale e il miglioramento degli algoritmi di apprendimento.

In ambito di *image synthesis*, i modelli generativi hanno seguito un percorso evolutivo ben definito:

1. **Generative Adversarial Networks (GAN)**, che hanno introdotto un approccio avversario capace di produrre immagini fotorealistiche, stabilendo per anni lo stato dell'arte in termini di qualità percepita.

2. **Modelli basati su verosimiglianza esplicita** (es. Variational Autoencoders e modelli autoregressivi), caratterizzati da una buona copertura della distribuzione dei dati ma qualità visiva limitata.

3. **Denoising Diffusion Probabilistic Models (DDPM)**, modelli basati sulla stima della distribuzione di probabilità tramite un processo inverso di

rimozione del rumore, che uniscono stabilità di addestramento, copertura della distribuzione e, con recenti miglioramenti, qualità visiva pari o superiore ai GAN.

1.2 Dai GAN ai DDPM

Le **Generative Adversarial Networks** (GAN) hanno rappresentato una svolta fondamentale nei modelli generativi, consentendo di ottenere immagini ad alta risoluzione. La loro architettura si basa su due componenti principali che competono in un processo di apprendimento avversariale:

- **Generatore (G)**: prende in input un vettore di rumore casuale $z \sim p_z(z)$, proveniente da una distribuzione nota (tipicamente gaussiana o uniforme), e lo trasforma in un campione sintetico $G(z)$ che mira a riprodurre le caratteristiche statistiche dei dati reali.
- **Discriminatore (D)**: riceve in ingresso sia campioni reali $x \sim p_{\text{data}}(x)$ che campioni sintetici $G(z)$, restituendo una probabilità che misura quanto l'input sembri reale.

L'addestramento è formalizzato come un *gioco minimax*:

$$\min_G \max_D \; \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

dove il Discriminatore cerca di massimizzare la capacità di distinguere tra dati reali e generati, mentre il Generatore cerca di minimizzare la capacità del Discriminatore di riconoscere i falsi, "ingannandolo" con campioni sempre più realistici. Questo processo iterativo porta entrambe le reti a migliorarsi reciprocamente, fino a raggiungere un equilibrio in cui le immagini sintetiche diventano difficilmente distinguibili da quelle reali.

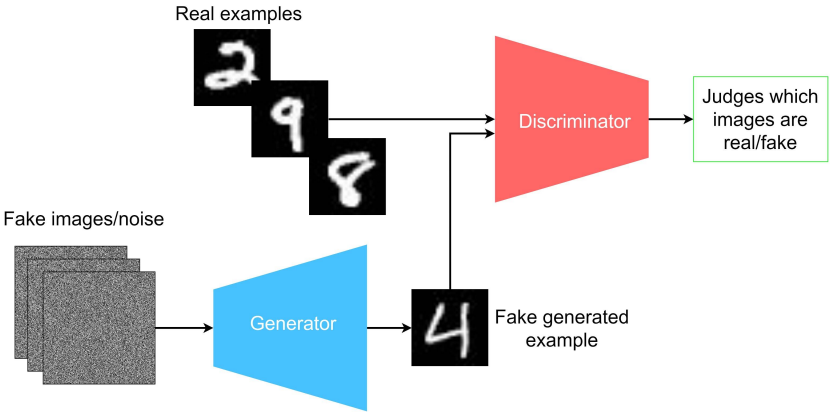

Figura 1.1: Schema generale di una GAN: il Generatore G trasforma un rumore casuale z in un campione sintetico, mentre il Discriminatore D valuta la probabilità che il campione sia reale.

Nonostante i numerosi successi ottenuti, le *Generative Adversarial Networks* (GAN) presentano alcuni limiti intrinseci che ne condizionano l'impiego in scenari complessi. L'addestramento, infatti, risulta spesso instabile a causa della natura avversaria del processo, che può causare divergenze o oscillazioni difficilmente controllabili. Un ulteriore problema ricorrente è il cosiddetto *mode collapse*, ovvero la tendenza del generatore a produrre soltanto un insieme ristretto di campioni, con conseguente riduzione della diversità. A ciò si aggiunge una marcata sensibilità alla scelta degli iperparametri, che rende l'ottimizzazione delicata e ne ostacola la scalabilità verso domini particolarmente complessi o significativamente differenti da quello di addestramento.

Per affrontare queste problematiche, si sono affermati i **Denoising Diffusion Probabilistic Models** (DDPM), appartenenti alla famiglia dei modelli *likelihood-based*. Diversamente dalle GAN, che apprendono a mappare direttamente un rumore in un'immagine, i DDPM definiscono un processo generativo inverso di *denoising* multi-step: partendo da un rumore gaussiano puro, rimuovono progressivamente il rumore stimando a ogni passo la distribuzione condizionata $p_\theta(x_{t-1} \mid x_t)$. Questi aspetti verranno ripresi e approfonditi nel Capitolo 2, dove verranno illustrati in modo più dettagliato i principi di funzionamento dei modelli di diffusione e i meccanismi che ne rendono possibile la fase generativa.

Rispetto alle GAN, i DDPM presentano vantaggi significativi:

- **Addestramento stabile**, poiché non dipendono dalla dinamica competitiva che spesso compromette la convergenza dei modelli avversari.
- Maggiore **copertura della distribuzione**, riducendo sensibilmente il rischio di *mode collapse*.

- **Scalabilità** a differenti risoluzioni e domini, riuscendo a preservare una buona consistenza qualitativa nei campioni generati.

Inizialmente, i DDPM non raggiungevano la qualità visiva delle GAN su dataset complessi. Tuttavia, miglioramenti architetturali come UNet potenziate, attenzione multi-scala, blocchi di up/downsampling derivati da BigGAN, Adaptive Group Normalization e tecniche di *guidance* basate su classificatori hanno consentito di ottenere un controllo più fine sul compromesso diversità/fedeltà, superando le GAN in metriche come il FID (Fréchet Inception Distance).

1.3 Legislazioni e limitazioni dell'AI generativa

L'Unione Europea è attualmente la giurisdizione con il quadro regolatorio più avanzato in materia di intelligenza artificiale. Con l'adozione dell'**AI Act**, l'UE ha introdotto una classificazione dei sistemi di AI basata sul livello di rischio (*minimo*, *limitato*, *alto*, *inaccettabile*), imponendo obblighi proporzionati alla categoria.

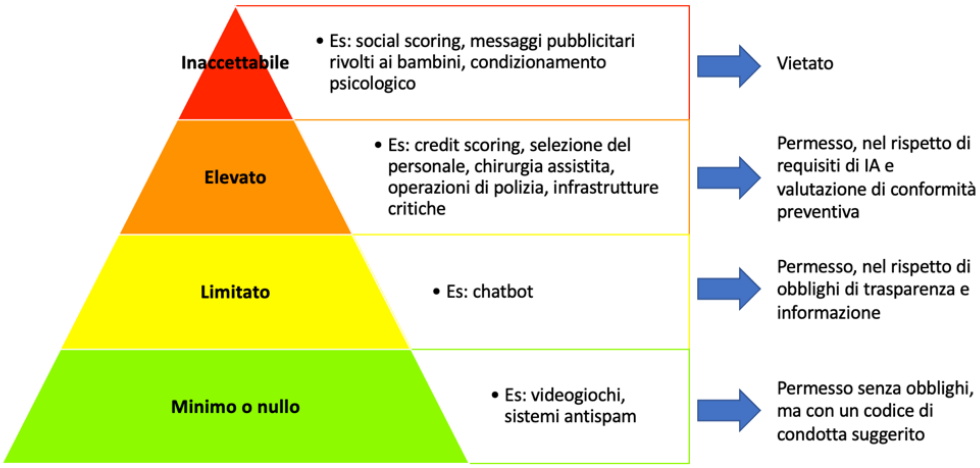

Figura 1.2: Classificazione dei sistemi di intelligenza artificiale secondo l'AI Act europeo, con esempi per ciascun livello di rischio e relativi obblighi normativi.

L'*AI generativa*, come i modelli di diffusione e i modelli linguistici di grandi dimensioni (LLM), può essere collocata in diverse fasce della piramide in funzione dell'uso previsto. In situazioni a basso impatto, come nei chatbot generici o negli strumenti creativi per la produzione di immagini, l'AI generativa può essere considerata a **rischio limitato**, purché vengano rispettati gli obblighi di trasparenza, ad esempio indicare chiaramente che il contenuto è stato generato artificialmente. In contesti più delicati, come la generazione di contenuti per processi decisionali

automatizzati, diagnosi mediche assistite o sistemi di monitoraggio, l'AI generativa può essere considerata a **rischio elevato**, poiché potrebbe incidere su diritti fondamentali o sulla sicurezza. Infine, se utilizzata per scopi vietati dall'AI Act, come nel caso di *social scoring*, manipolazione psicologica di soggetti vulnerabili o produzione di deepfake per scopi di disinformazione politica, l'AI generativa rientra nel **rischio inaccettabile**.

L'AI Act stabilisce specifici obblighi per l'uso dell'AI generativa, tra cui la **trasparenza**, che richiede di dichiarare chiaramente quando un contenuto è stato generato o manipolato da un sistema di AI. Inoltre, è necessario fornire **documentazione sui dati di addestramento**, specificando i dataset utilizzati, con particolare attenzione ai contenuti protetti da copyright. Infine, è fondamentale implementare misure per la **mitigazione dei rischi**, al fine di prevenire la generazione di contenuti illeciti o fuorvianti.

Sul piano della proprietà intellettuale, la Direttiva UE 2019/790 sul *text and data mining* consente l'estrazione di dati per scopi di ricerca, ma permette ai titolari dei diritti di escludere esplicitamente le loro opere. Questo pone limiti diretti all'uso di dataset contenenti materiale protetto.

Infine, il *Regolamento Generale sulla Protezione dei Dati* (GDPR) si applica quando i dataset di addestramento includono informazioni personali, imponendo obblighi di anonimizzazione o pseudonimizzazione e di tutela della privacy. Questi vincoli assumono un ruolo centrale nell'AI generativa, poiché i modelli di diffusione e gli autoencoder variazionali richiedono grandi quantità di dati per l'addestramento. L'impiego di dataset contenenti opere protette da copyright o informazioni personali deve pertanto essere gestito con attenzione, garantendo la conformità alle normative vigenti. In particolare, la trasparenza sulla provenienza dei dati e l'adozione di tecniche di anonimizzazione rappresentano condizioni essenziali per lo sviluppo responsabile di sistemi generativi.

Capitolo 2

Denoising Diffusion Probabilistic Models (DDPM)

In questo capitolo vengono analizzati i *Denoising Diffusion Probabilistic Models* (DDPM) [8], [9], [10], [11], una classe di modelli generativi basati su processi di diffusione che hanno recentemente ottenuto risultati allo stato dell'arte. L'esposizione parte dai principi fondamentali e dalle motivazioni teoriche che giustificano l'impiego della diffusione come meccanismo di generazione, approfondendo in particolare il legame tra entropia, termodinamica e dinamiche stocastiche. Successivamente viene presentata la formulazione matematica dei processi *forward* e *reverse*, insieme alla derivazione del bound alla log-likelihood e alla sua interpretazione. Un'attenzione specifica è dedicata al ruolo dei cosiddetti *variance scheduler*, che influenzano in maniera significativa la qualità dei campioni prodotti. Il capitolo discute inoltre l'architettura di riferimento basata su reti U-Net arricchite con meccanismi di *time-embedding*, fondamentali per modellare la dipendenza temporale dei passi di denoising. Infine, viene introdotto il meccanismo di *Classifier-Free Guidance*, che consente di controllare la generazione in maniera condizionata, ampliando le possibilità di utilizzo pratico dei modelli di diffusione.

2.1 Entropia e termodinamica

I modelli di diffusione prendono ispirazione dalla *non-equilibrium thermodynamics*: partendo da una distribuzione dei dati $q(x^{(0)})$ caratterizzata da struttura elevata e quindi *bassa entropia* si applica un processo di degradazione controllato, aggiungendo rumore in più passi fino a ottenere una distribuzione semplice π (forward process), che rappresenta uno stato di *alta entropia*. Il modello generativo impara quindi

a *invertire* questo processo, rimuovendo gradualmente il rumore e ripristinando la struttura originale (reverse process).

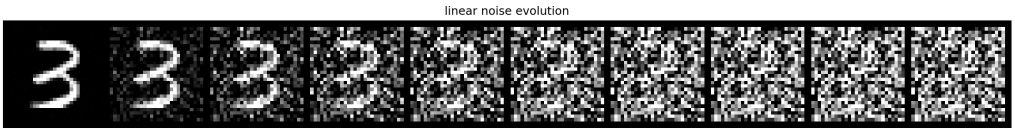

Figura 2.1: Esempi qualitativi di *bassa* entropia (alta struttura) e *alta* entropia (rumore casuale).

Forward process (aumento di entropia). Il *forward process* rappresenta la fase in cui i dati originali vengono progressivamente "degradati" aggiungendo rumore in più passi, fino a perdere quasi completamente la loro struttura. Indichiamo con $T_\pi(\cdot \mid \cdot; \beta_t)$ il *kernel di transizione* (cioè la regola probabilistica che descrive come passare da uno stato $x^{(t-1)}$ a uno stato $x^{(t)}$) al passo t, dove β_t controlla la quantità di rumore aggiunto in quel passo.

Matematicamente, la dinamica del forward si descrive come:

$$q(x^{(t)} \mid x^{(t-1)}) = T_\pi\Big(x^{(t)} \mid x^{(t-1)}; \beta_t\Big), \quad \pi(y) = \int T_\pi(y \mid y'; \beta)\, \pi(y')\, dy'. \tag{2.1}$$

La prima delle equazioni (2.1) dice che la probabilità di ottenere $x^{(t)}$ dipende solo dallo stato precedente $x^{(t-1)}$ e dal rumore iniettato. La seconda delle equazioni (2.1) descrive la *distribuzione stazionaria* π: se il processo di rumore venisse iterato per un numero infinito di passi, si convergerebbe a π (tipicamente una Gaussiana standard). Nel caso continuo, se i dati sono prima normalizzati a varianza unitaria, l'entropia[1] non può diminuire quando si aggiunge rumore gaussiano indipendente. Infatti, la convoluzione di una distribuzione qualsiasi con una gaussiana produce una distribuzione più "larga" e meno concentrata: le regioni di alta densità vengono smussate e quelle di bassa densità si riempiono, incrementando l'incertezza complessiva. In altre parole, il rumore agisce come un operatore di diffusione che disperde l'informazione, spingendo la distribuzione verso uno stato di massima entropia (rumore puro).

La relazione

$$H_q(X^{(t)}) \;\geq\; H_q(X^{(t-1)}) \tag{2.2}$$

significa che:

- $H_q(X^{(t)})$ è l'entropia dopo t passi di diffusione;
- $H_q(X^{(t-1)})$ è l'entropia dopo il passo precedente;

[1] In questo contesto, l'entropia di Shannon di una variabile casuale continua X è $H(X) = -\int p(x) \log p(x)\, dx$, e misura il "disordine" o l'incertezza della distribuzione.

- l'uguaglianza si verifica solo se $\beta_t = 0$, cioè non viene aggiunto rumore in quel passo.

In altre parole, ogni passo del *forward process* non riduce mai l'incertezza della distribuzione, e nella pratica la aumenta quasi sempre.

Intuitivamente:

- se β_t è piccolo, il rumore aggiunto è minimo e la crescita di entropia è lenta;
- se β_t è grande, la crescita di entropia è più rapida e la struttura dei dati si perde velocemente.

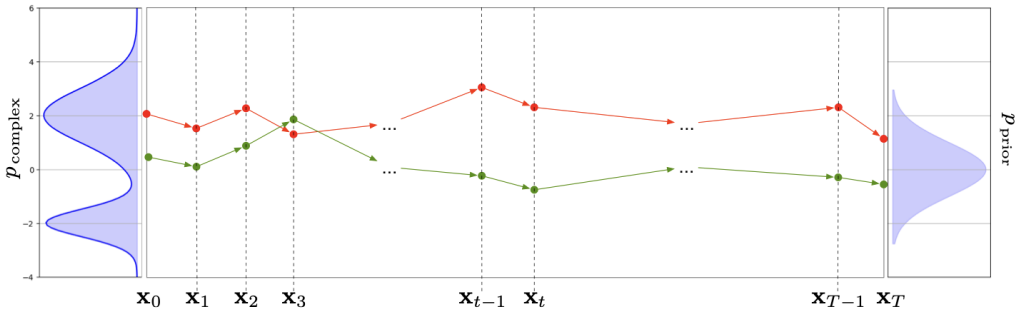

Figura 2.2: Rappresentazione schematica del *forward process*: partendo dai dati complessi p_{complex}, il processo di diffusione applica più passi di rumore fino a raggiungere il prior p_{prior}.

Reverse process (riduzione di entropia). Il *reverse process* è la fase generativa vera e propria: partendo da uno stato di rumore puro, il modello cerca di ricostruire gradualmente la struttura dei dati originali, riducendo passo dopo passo l'incertezza della distribuzione.

Formalmente, la probabilità di una traiettoria completa $(x^{(0)}, \ldots, x^{(T)})$ nel reverse si scrive come:

$$p(x^{(0\ldots T)}) = p(x^{(T)}) \prod_{t=1}^{T} p(x^{(t-1)} \mid x^{(t)}), \quad p(x^{(T)}) = \pi.$$

Qui:

- $p(x^{(T)}) = \pi$ indica la distribuzione iniziale del reverse, che è la stessa distribuzione di rumore raggiunta dal forward process dopo T passi;
- $p(x^{(t-1)} \mid x^{(t)})$ è il *kernel inverso* che descrive come passare dallo stato $x^{(t)}$ a quello $x^{(t-1)}$ ricostruendo parte del segnale perso;
- il prodotto $\prod_{t=1}^{T}$ combina tutti i passi, partendo dal rumore e tornando ai dati.

Se i passi β_t sono abbastanza piccoli, il reverse process ha *la stessa forma funzionale* del forward: ad esempio, se il forward aggiunge rumore gaussiano, il

reverse sarà anch'esso descritto da una distribuzione gaussiana, ma con media e covarianza scelte in modo da *togliere* il rumore anziché aggiungerlo.

In pratica, il modello deve apprendere i parametri della distribuzione inversa al passo t. La **media** $f_\mu(x^{(t)}, t)$ rappresenta il "punto centrale" attorno a cui è distribuito il campione $x^{(t-1)}$. Intuitivamente, questa media indica quale sia la versione più probabile del dato quando proviamo a rimuovere il rumore dal campione corrente $x^{(t)}$. D'altro canto, la **covarianza** $f_\Sigma(x^{(t)}, t)$ misura l'incertezza associata a questa ricostruzione. Se il rumore è stato iniettato in modo lieve (cioè per valori piccoli di β_t), la covarianza sarà ridotta e il modello potrà stimare $x^{(t-1)}$ con alta precisione. Al contrario, quando il campione è molto degradato (per valori elevati di t), la covarianza aumenta, poiché diverse configurazioni di $x^{(t-1)}$ sono compatibili con lo stesso $x^{(t)}$.

Dal punto di vista geometrico, possiamo immaginare $f_\mu(x^{(t)}, t)$ come la "direzione di denoising" che porta il campione verso lo spazio dei dati, mentre $f_\Sigma(x^{(t)}, t)$ controlla quanto il processo resti stocastico o, al contrario, deterministico. In altre parole, la media dice *"dove andare"*, mentre la covarianza dice *"quanto fidarsi"* di quella direzione.

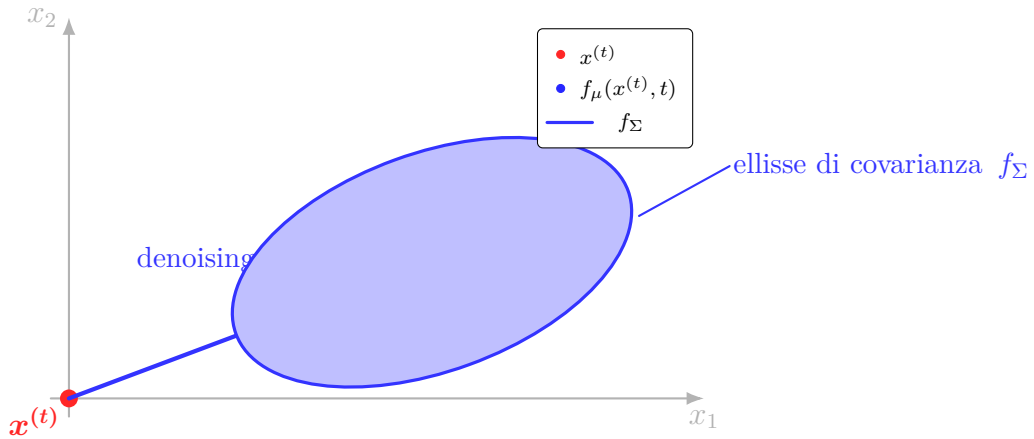

Figura 2.3: Media e covarianza nel reverse: la freccia indica la direzione di denoising verso la media $f_\mu(x^{(t)}, t)$; l'ellisse (1-σ) rappresenta l'incertezza modellata da $f_\Sigma(x^{(t)}, t)$.

Questi due parametri del kernel inverso vengono stimati da una rete neurale (tipicamente una UNet), addestrata in modo che la sequenza di passi inversi ricostruisca fedelmente la distribuzione dei dati originali.

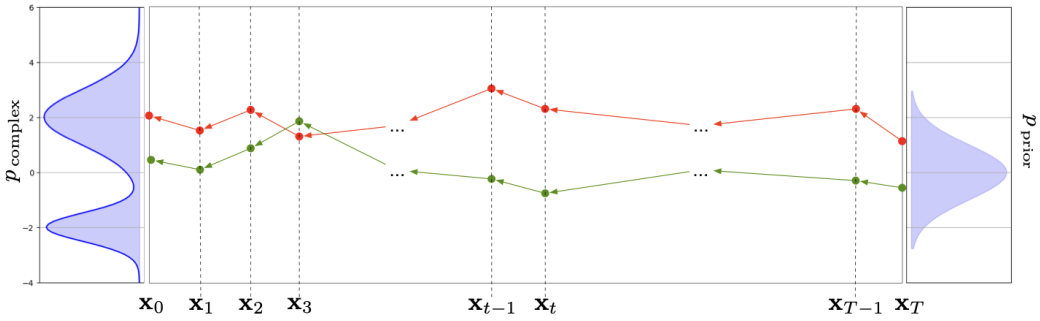

Figura 2.4: Rappresentazione schematica del *reverse process*: a partire dal prior p_{prior}, il modello ricostruisce gradualmente la distribuzione complessa dei dati p_{complex}.

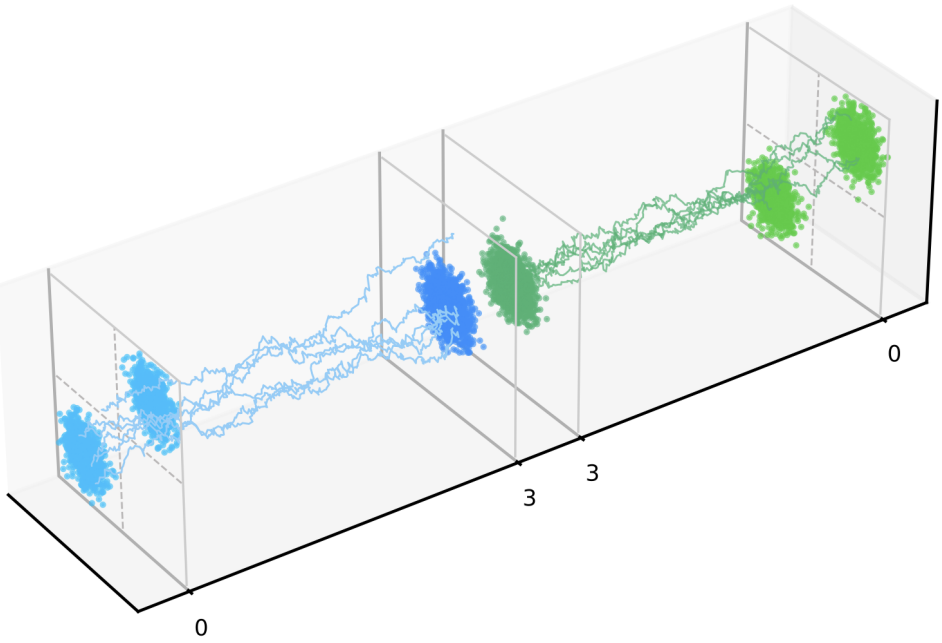

Figura 2.5: Esempio di traiettorie nel forward (in blu) e nel reverse process (in verde) nello spazio delle configurazioni.

Bound alla log-likelihood. L'obiettivo dell'addestramento è massimizzare la *log-likelihood*

$$L = \log p(x^{(0)}),$$

ossia il logaritmo della probabilità che il modello assegna a un campione reale $x^{(0)}$. L'impiego del logaritmo non è soltanto una convenzione, ma ha motivazioni pratiche: consente di trasformare prodotti di probabilità condizionate in somme, rendendo l'ottimizzazione più trattabile, e al tempo stesso stabilizza i calcoli quando si lavora con valori molto piccoli, tipici di distribuzioni ad alta dimensionalità. In termini intuitivi, durante il training massimizzare la log-likelihood significa aumentare la

coerenza tra i campioni generati e i dati osservati: un modello che assegna alta probabilità (quindi alta log-probabilità) a un'immagine reale avrà maggiori capacità di ricostruirla, e di conseguenza, nella fase di generazione, sarà in grado di produrre campioni che appartengono a distribuzioni simili a quelle del dataset. Nel caso dei modelli di diffusione, L non può essere calcolata in forma chiusa, ma possiamo ottenere un *lower bound* (limite inferiore) K che può essere massimizzato in pratica:

$$L \geq K = -\sum_{t=2}^{T} \mathbb{E}_{q(x^{(0)},x^{(t)})}\Big[\mathrm{KL}\Big(q(x^{(t-1)} \mid x^{(t)}, x^{(0)}) \,\|\, p(x^{(t-1)} \mid x^{(t)})\Big)\Big] + H_q(X^{(T)} \mid X^{(0)}) - H_q(X^{(1)} \mid X^{(0)}) - H_p(X^{(T)}). \tag{2.3}$$

Vediamo il significato di ogni termine:

- $\mathrm{KL}(\cdot \,\|\, \cdot)$ è la divergenza di Kullback–Leibler, che misura quanto due distribuzioni differiscono tra loro. Qui confronta:
  - $q(x^{(t-1)} \mid x^{(t)}, x^{(0)})$: distribuzione "vera" del passo inverso derivata dal forward;
  - $p(x^{(t-1)} \mid x^{(t)})$: distribuzione inversa appresa dal modello.

  Ogni termine di KL penalizza il modello quando il reverse appreso si discosta dal reverse "ideale".
- La sommatoria $\sum_{t=2}^{T}$ calcola la penalità totale lungo tutti i passi, partendo dal secondo fino all'ultimo.
- $H_q(X^{(T)} \mid X^{(0)})$: entropia condizionata dopo T passi, misura l'incertezza introdotta dal forward partendo dai dati reali.
- $H_q(X^{(1)} \mid X^{(0)})$: entropia condizionata dopo il primo passo, che viene sottratta per bilanciare la stima.
- $H_p(X^{(T)})$: entropia (non condizionata) dello stato iniziale del reverse process, cioè del rumore puro π.

Il bound K diventa *esatto* (cioè $K = L$) nel caso ideale in cui il reverse process appreso coincida perfettamente con quello "vero". In questo scenario, ogni termine di KL nella sommatoria si annulla:

$$\mathrm{KL}(q \,\|\, p) = 0 \quad \Rightarrow \quad L = K.$$

Questo caso limite corrisponde a un processo *quasi-statico*, in cui la generazione è perfettamente reversibile e non si perde informazione lungo i passi.

Seconda legge in forma discreta. Il punto chiave è che il *forward process* q è definito in modo esplicito: siamo noi a stabilire la sequenza di rumori $\{\beta_t\}$

da applicare, e quindi conosciamo analiticamente le distribuzioni $q(x^{(t)} \mid x^{(t-1)})$ e $q(x^{(t)} \mid x^{(0)})$. Questo non vale per il reverse process p, che deve invece essere appreso. Proprio grazie al fatto che q è noto, è possibile derivare dei *bound* sull'entropia condizionata del reverse p, portando alla relazione:

$$H_q(X^{(t)} \mid X^{(t-1)}) \;\geq\; H_q(X^{(t-1)} \mid X^{(t)}) \;\geq\; H_q(X^{(t)} \mid X^{(t-1)}) + H_q(X^{(t-1)} \mid X^{(0)}) - H_q(X^{(t)} \mid X^{(0)}).$$

si interpreta così:

- $H_q(X^{(t)} \mid X^{(t-1)})$ è l'incertezza introdotta andando da $t-1$ a t nel forward;
- $H_q(X^{(t-1)} \mid X^{(t)})$ è l'incertezza del reverse nel ricostruire $t-1$ conoscendo t;
- il termine $H_q(X^{(t-1)} \mid X^{(0)}) - H_q(X^{(t)} \mid X^{(0)})$ corregge il bound tenendo conto di quanta informazione residua sul dato originale si perde tra $t-1$ e t.

Il "divario" tra il lato sinistro e destro della disuguaglianza misura l'*irreversibilità* del processo: più è grande, maggiore è la quantità di informazione che è andata persa e non può essere recuperata. Nel limite di passi infinitesimi ($\beta_t \to 0$) questo divario si annulla e il processo diventa quasi-statico.

Scheduler β_t e dissipazione. Lo *scheduler* definisce come i valori β_t (quantità di rumore aggiunto a ogni passo del forward) variano nel tempo. La scelta della schedule influisce su:

- la **dissipazione**: quantità di "energia informativa" persa lungo il ciclo forward–reverse;
- la **precisione del bound** K: passi più piccoli e concentrati nelle fasi "critiche" riducono la divergenza KL tra forward e reverse, avvicinando K a L.

In sintesi: scheduler dolci (piccoli β_t) rendono il processo più vicino al quasi-statico ma aumentano la lunghezza della catena di passi; scheduler aggressivi (grandi β_t) riducono i il numero di passi ma aumentano la perdita di informazione.

2.2 Processi forward e reverse

In questa sezione formalizziamo i processi di *forward* (diffusione) e *reverse* (denoising) nei DDPM, esplicitando le distribuzioni coinvolte, le espressioni in forma chiusa utili in addestramento e campionamento, e la parametrizzazione pratica impiegata nel modello.

2.2.1 Forward: catena di Markov gaussiana

Il **forward process** (o *diffusion process*) è una catena di Markov che aggiunge rumore gaussiano ai dati $x^{(0)} \sim q(x^{(0)})$ secondo una sequenza di varianze $\{\beta_t\}_{t=1}^{T}$. Ad ogni passo t, lo stato corrente $x^{(t)}$ dipende unicamente dallo stato precedente $x^{(t-1)}$ e da un rumore gaussiano $\mathcal{N}(0, \mathbf{I})$:

$$q(x^{(1:T)} \mid x^{(0)}) \;=\; \prod_{t=1}^{T} q(x^{(t)} \mid x^{(t-1)}), \qquad q(x^{(t)} \mid x^{(t-1)}) \;=\; \mathcal{N}\Big(x^{(t)};\, \sqrt{1-\beta_t}\, x^{(t-1)},\, \beta_t \mathbf{I}\Big). \tag{2.4}$$

Derivazione intuitiva. Scrivendo $\alpha_t = 1 - \beta_t$, l'equazione (2.4) diventa:

$$x_t = \sqrt{\alpha_t}\, x_{t-1} + \sqrt{\beta_t}\, \mathcal{N}(0, I).$$

Se, ad esempio, $\alpha_t = 0.5$ e $\beta_t = 0.1$, la distribuzione di x_t è una combinazione di metà del segnale precedente e di rumore gaussiano, con un progressivo "appiattimento" verso una Gaussiana standard.

Caso estremo: se $\alpha_t = 0$ e $\beta_t = 1$, otteniamo immediatamente una distribuzione normale standard indipendente dal dato iniziale x_0.

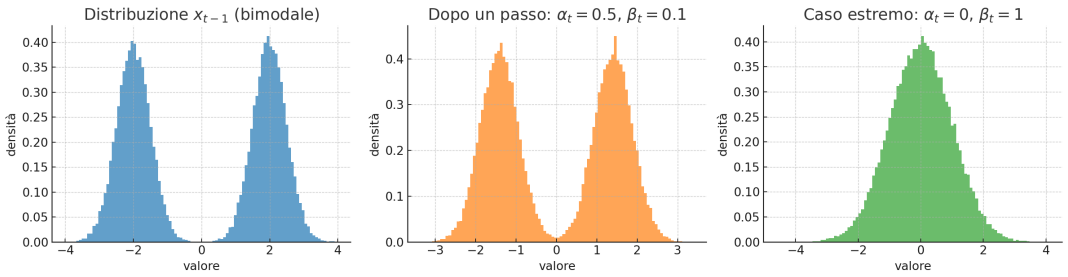

Figura 2.6: Evoluzione della distribuzione nel forward process: (sinistra) distribuzione strutturata iniziale x_{t-1}; (centro) dopo un passo con $\alpha_t = 0.5$, $\beta_t = 0.1$; (destra) caso estremo con $\alpha_t = 0$, $\beta_t = 1$, che produce immediatamente una Gaussiana standard.

Iterando la definizione per due passi:

$$x_t = \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{\beta_{t-1}}\, \mathcal{W}(0, I)\right) + \sqrt{\beta_t}\, \mathcal{N}(0, I),$$

da cui:

$$x_t = \alpha_t^{1/2} \alpha_{t-1}^{1/2}\, x_{t-2} + \sqrt{\alpha_t \beta_{t-1}}\, \mathcal{W}(0, I) + \sqrt{\beta_t}\, \mathcal{N}(0, I).$$

Proseguendo e scrivendo x_{t-2} in funzione di x_{t-3}, si ottiene:

$$x_t = (\sqrt{\alpha_t}\sqrt{\alpha_{t-1}}\sqrt{\alpha_{t-2}}) x_{t-3} + \cdots + \sqrt{\beta_t}\, \mathcal{N}(0, I) + (\text{termini di rumore pesati}).$$

Generalizzando:

$$x_t = \alpha^T x_0 + \sum_{k=0}^{T-1} \left(\sqrt{\beta_{t-k}} \prod_{j=0}^{k-1} \sqrt{\alpha_{t-j}} \right) \mathcal{W}(0, I),$$

dove i pesi dei termini di rumore dipendono dal prodotto delle α precedenti.

Per T sufficientemente grande, il termine $\alpha^T x_0$ tende a 0 e resta solo la somma dei rumori, la cui varianza totale tende a 1:

$$\beta \, \frac{1 - (1-\beta)^T}{1 - (1-\beta)} \xrightarrow[T \to \infty]{} 1.$$

Quindi la "transition function" porta automaticamente a una distribuzione con media 0 e varianza 1 (Gaussiana standard).

Forma chiusa. Definendo $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, si ottiene la forma chiusa:

$$q(x^{(t)} \mid x^{(0)}) = \mathcal{N}\Big(x^{(t)};\, \sqrt{\bar{\alpha}_t}\, x^{(0)},\, (1 - \bar{\alpha}_t)\mathbf{I}\Big), \tag{2.5}$$

cioè:

$$x^{(t)} = \sqrt{\bar{\alpha}_t}\, x^{(0)} + \sqrt{1 - \bar{\alpha}_t}\, \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \mathbf{I}). \tag{2.6}$$

Scheduler della varianza. La varianza β_t non è fissa: l'intuizione è usare uno *variance scheduler*, cioè una sequenza $\beta_1, \beta_2, \ldots, \beta_T$ che controlla l'entità del rumore aggiunto a ogni passo. Scheduler comuni sono:

- **Lineare**: β_t cresce linearmente da un valore iniziale (es. $\beta_1 = 0.0001$) a un valore finale (es. $\beta_T = 0.02$).
- **Cosine**: crescita secondo una curva coseno per distribuire il rumore in modo più uniforme.

Come euristica, vale il principio: *"Create chaos, but do it wisely"*, ovvero aggiungere rumore in modo graduale per rendere il processo più facilmente invertibile.

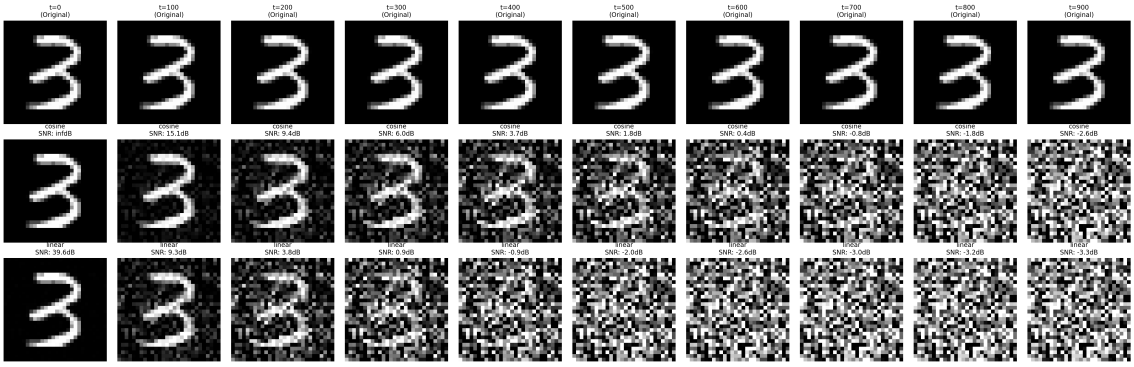

Figura 2.7: Confronto visivo tra Linear Scheduler e Cosine Scheduler nell'aggiunta di rumore a un'immagine MNIST (cifra 3) a diversi timestep t. Si nota come il Cosine Scheduler mantenga un SNR più alto nelle prime fasi rispetto al Linear Scheduler, preservando più a lungo la struttura dell'immagine.

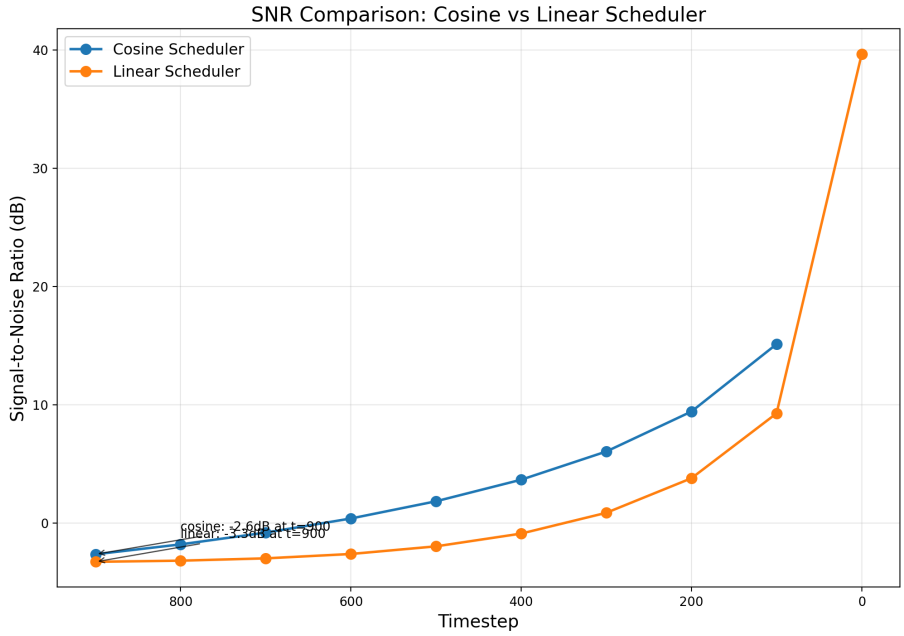

Figura 2.8: Andamento del rapporto Segnale-Rumore (SNR) in dB per Cosine e Linear Scheduler. Il Cosine Scheduler degrada il segnale più lentamente nelle prime fasi, garantendo una transizione più graduale verso il rumore puro.

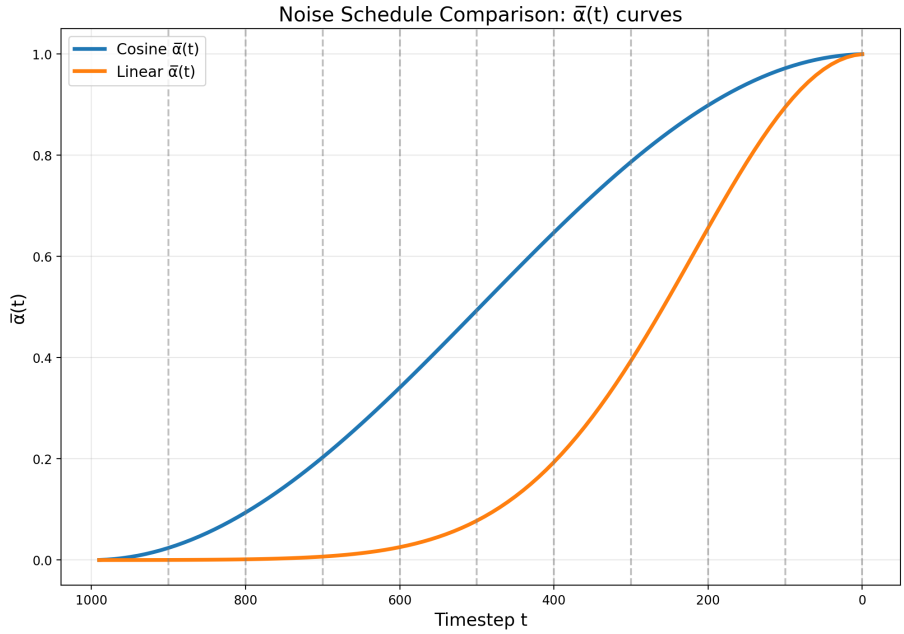

Figura 2.9: Confronto tra le curve di $\bar{\alpha}(t)$ nei due scheduler. Il Cosine Scheduler distribuisce la riduzione di $\bar{\alpha}$ in modo più uniforme lungo la catena di Markov, mentre il Linear Scheduler concentra la riduzione in pochi passi iniziali e finali.

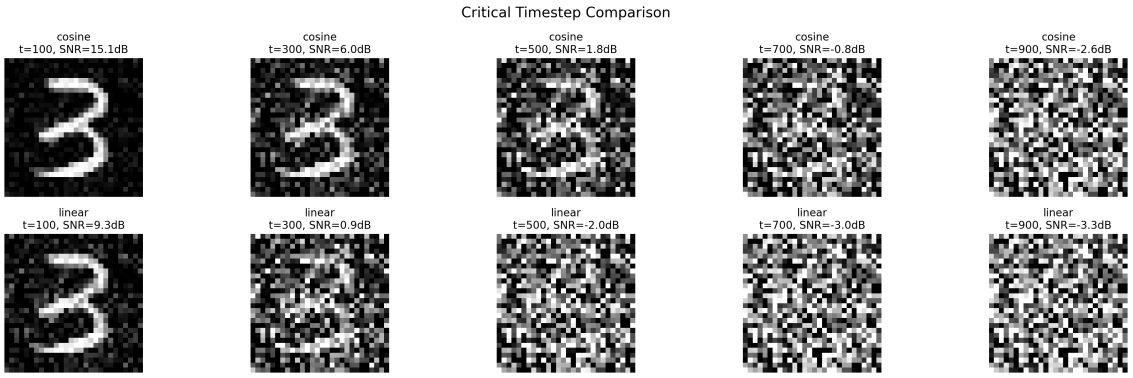

Figura 2.10: Confronto visivo agli istanti critici ($t = 100, 300, 500, 700, 900$) per Linear e Cosine Scheduler. Il Cosine mantiene dettagli visivi più a lungo, risultando in un processo di degradazione più controllato.

2.2.2 Reverse: modello generativo gaussiano

Per brevità adottiamo la notazione $x_t \equiv x^{(t)}$. Dopo T passi di diffusione, il *reverse process* è una catena di Markov che parte dal prior gaussiano e rimuove progressivamente il rumore:

$$p_\theta(x_{0:T}) \;=\; p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t), \qquad p(x_T) = \mathcal{N}(0, I), \tag{2.7}$$

dove, per ogni t, assumiamo transizioni gaussiane

$$p_\theta(x_{t-1} \mid x_t) \;=\; \mathcal{N}\Big(x_{t-1};\, \mu_\theta(x_t, t),\, \Sigma_\theta(x_t, t)\Big). \tag{2.8}$$

Perché non usiamo direttamente $q(x_{t-1} \mid x_t)$? In linea di principio vorremmo campionare dal *vero* reverse $q(x_{t-1} \mid x_t)$, ottenuto applicando Bayes a $q(x_t \mid x_{t-1})$. Tuttavia:

$$q(x_{t-1} \mid x_t) = \frac{q(x_t \mid x_{t-1})\, q(x_{t-1})}{q(x_t)} = \frac{q(x_t \mid x_{t-1}) \int q(x_{t-1} \mid x_0)\, q(x_0)\, dx_0}{\int q(x_t \mid x_0)\, q(x_0)\, dx_0},$$

e le marginali $q(x_{t-1})$ e $q(x_t)$ richiedono integrazioni sulla (sconosciuta) distribuzione dei dati $q(x_0)$. La via diretta è dunque intrattabile; introduciamo quindi il modello parametrico (2.8) da apprendere.

Posteriori del forward in forma chiusa. Nel forward gaussiano, con $\alpha_t = 1 - \beta_t$ e $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, valgono

$$q(x_t \mid x_0) = \mathcal{N}\Big(x_t;\, \sqrt{\bar{\alpha}_t}\, x_0,\, (1 - \bar{\alpha}_t) I\Big), \qquad q(x_t \mid x_{t-1}) = \mathcal{N}\Big(x_t;\, \sqrt{\alpha_t}\, x_{t-1},\, \beta_t I\Big).$$

Combinando il teorema di Bayes con la proprietà markoviana del processo di forward si ottiene, per ogni t, il *posteriore di forward* (condizionato a x_0), che è ancora una distribuzione gaussiana:

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\Big(x_{t-1};\, \tilde{\mu}_t(x_t, x_0),\, \tilde{\beta}_t I\Big), \tag{2.9}$$

con media e varianza note date da:

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\, x_0 \;+\; \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\, x_t, \qquad \tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\, \beta_t. \tag{2.10}$$

ELBO locale: riduzione a una MSE sulla media. Il VLB (ELBO) su $-\log p_\theta(x_0)$ si scompone in termini locali

$$L_{t-1} \;=\; \mathbb{E}_{q(x_t|x_0)}\Big[D_{\mathrm{KL}}\Big(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t)\Big)\Big], \quad 2 \le t \le T.$$

Fissando la varianza del processo reverse $\Sigma_\theta(x_t, t) = \sigma_t^2 I$ (uguale per p_θ e q al passo t), la KL tra gaussiane coincide (a costanti additive) con una MSE tra le medie:

$$L_{t-1} = \frac{1}{2\sigma_t^2}\, \mathbb{E}\Big[\Big\|\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)\Big\|^2\Big] + \text{cost}. \tag{2.11}$$

Scelte pratiche per σ_t^2 sono β_t (rumore del forward), $\tilde{\beta}_t$ (varianza del posteriore (2.10)) oppure 0 (caso deterministico DDIM).

Dalla forma chiusa del forward alla ε-prediction. Dalla forma chiusa del forward si ottiene

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I),$$

da cui

$$x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\Big(x_t - \sqrt{1-\bar{\alpha}_t}\, \varepsilon\Big). \tag{2.12}$$

Sostituendo l'espressione in (2.12) nella definizione della media $\tilde{\mu}(x_t, x_0)$ in (**??**), si ottiene:

$$\tilde{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}\left(x_t \; - \; \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\, \varepsilon\right). \tag{2.13}$$

Tale forma suggerisce di *parametrizzare* la media del reverse in funzione del *rumore* ϵ predetto dalla rete:

$$\boxed{\mu_\theta(x_t, t) \;=\; \frac{1}{\sqrt{\alpha_t}}\left(x_t \; - \; \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\, \varepsilon_\theta(x_t, t)\right).} \tag{2.14}$$

Sostituendo (2.14) dentro (2.11), il termine locale diventa (a pesi noti)

$$L_{t-1} \;\propto\; \mathbb{E}\Big[\big\|\varepsilon - \varepsilon_\theta(x_t, t)\big\|^2\Big],$$

ossia una *denoising MSE* su più livelli di rumore.

Score matching: legame concettuale. Dalla distribuzione gaussiana del forward si ricava

$$\nabla_{x_t} \log q(x_t \mid x_0) \;=\; -\,\frac{x_t - \sqrt{\bar{\alpha}_t} x_0}{1-\bar{\alpha}_t} \;=\; -\,\frac{\varepsilon}{\sqrt{1-\bar{\alpha}_t}}.$$

Quindi predire ε equivale, a un fattore noto, a stimare lo *score* $\nabla_{x_t} \log q(x_t)$. La rete ε_θ apprende dunque un campo di forza di denoising coerente con lo score matching.

Regola di campionamento (reverse step). Fissata $\sigma_t^2 \in \{\beta_t, \tilde{\beta}_t, 0\}$, la regola di campionamento del passo inverso è data da:

$$x_{t-1} \;=\; \mu_\theta(x_t, t) \;+\; \sigma_t z \;=\; \frac{1}{\sqrt{\alpha_t}}\left(x_t \; - \; \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\, \varepsilon_\theta(x_t, t)\right) \;+\; \sigma_t z, \qquad z \sim \mathcal{N}(0, I), \tag{2.15}$$

con $z = 0$ per $t = 1$. In pratica, se si imposta $\sigma_t = \sqrt{\beta_t}$, come nel DDPM classico, la varianza del processo inverso coincide con quella del forward al passo t. In questo caso, la componente stocastica $z \sim \mathcal{N}(0, I)$ riproduce fedelmente il disturbo aggiunto durante la diffusione, mantenendo così la coerenza statistica tra forward e reverse. Questa scelta consente di ottenere campioni molto variabili partendo dalla stessa condizione iniziale, ma introduce una componente di casualità nei risultati.

Nel caso in cui si imposti $\sigma_t = \sqrt{\tilde{\beta}_t}$, il processo inverso si allinea meglio all'ideale probabilistico previsto dal modello. Questo riduce la discrepanza statistica tra forward e reverse, producendo risultati con minore rumore residuo e traiettorie di denoising più stabili.

Infine, fissando $\sigma_t = 0$, si ottiene il metodo deterministico usato nei DDIM(Denoising Diffusion Implicit Models). In questo caso, la componente stocastica viene completamente eliminata e il campionamento diventa deterministico: fissato il valore iniziale x_T e il seme casuale usato, si ottiene sempre lo stesso x_0. Questo approccio permette di ridurre il numero di passi necessari per il campionamento, mantenendo alta la qualità visiva, ma a discapito della diversità nei campioni generati a parità di x_T.

Perché funziona nella pratica. Per valori piccoli di t , x_t conserva informazione rilevante su x_0 (varianza $(1 - \bar{\alpha}_t)I$ ridotta), fornendo un segnale di apprendimento forte per la predizione di ε. Per valori grandi di t , il segnale è più debole, ma la coerenza accumulata nei passi precedenti e la struttura markoviana della catena consentono traiettorie di denoising stabili fino a x_0.

2.3 Derivazione della loss function

Partiamo dal modello di transizione inverso:

$$p_\theta(x_{t-1} \mid x_t) \;=\; \mathcal{N}\Big(x_{t-1}\,;\, \mu_\theta(x_t, t),\, \Sigma_\theta(x_t, t)\Big), \tag{2.16}$$

dove μ_θ è la media predetta dalla rete neurale, mentre la varianza Σ_θ viene spesso fissata (tipicamente $\beta_t \mathbf{I}$), così che la rete non debba predirla.

Loss function di partenza. L'obiettivo teorico è massimizzare la log-likelihood:

$$\mathcal{L} = -\log p_\theta(x_0). \tag{2.17}$$

Tuttavia, questa quantità è difficile da calcolare esattamente, poiché dipende da tutti i passi t. Per questo motivo si introduce il *Variational Lower Bound* (VLB):

$$-\log p_\theta(x_0) \;\leq\; -\log p_\theta(x_0) + D_{\mathrm{KL}}\Big(q(x_{1:T} \mid x_0) \,\|\, p_\theta(x_{1:T} \mid x_0)\Big), \tag{2.18}$$

dove il termine di Kullback–Leibler misura la somiglianza tra la distribuzione di forward q e quella generata dal modello nel reverse p_θ.

Riformulazione. Usando la definizione di KL:

$$D_{\mathrm{KL}}\Big(q \,\|\, p\Big) = \mathbb{E}_q\left[\log \frac{q}{p}\right], \tag{2.19}$$

e applicando il teorema di Bayes per decomporre le probabilità condizionate, si ottiene:

$$-\log p_\theta(x_0) \leq \log \frac{q(x_{1:T} \mid x_0)}{p_\theta(x_{0:T})} \tag{2.20}$$

$$= -\log p_\theta(x_T) + \sum_{t=2}^{T} \log \frac{q(x_{t-1} \mid x_t, x_0)}{p_\theta(x_{t-1} \mid x_t)} + \sum_{t=2}^{T} \log \frac{q(x_t \mid x_0)}{q(x_{t-1} \mid x_0)}. \tag{2.21}$$

La seconda sommatoria può essere semplificata:

$$\sum_{t=2}^{T} \log \frac{q(x_t \mid x_0)}{q(x_{t-1} \mid x_0)} = \log \frac{q(x_T \mid x_0)}{q(x_1 \mid x_0)},$$

portando il bound nella forma:

$$-\log p_\theta(x_T) + \sum_{t=2}^{T} \log \frac{q(x_{t-1} \mid x_t, x_0)}{p_\theta(x_{t-1} \mid x_t)} + \log \frac{q(x_T \mid x_0)}{q(x_1 \mid x_0)}. \tag{2.22}$$

Riconoscendo nuovamente le definizioni di KL, si ottiene:

$$\mathcal{L}_{\mathrm{VLB}} = D_{\mathrm{KL}}\Big(q(x_T \mid x_0) \,\|\, p_\theta(x_T)\Big) + \sum_{t=2}^{T} D_{\mathrm{KL}}\Big(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t)\Big) - \log p_\theta(x_0 \mid x_1). \tag{2.23}$$

Osservazioni pratiche. Dal punto di vista applicativo, il primo termine di KL della Equation (2.23) risulta poco rilevante durante l'addestramento. Infatti, la distribuzione $q(x_T \mid x_0)$ non dipende da parametri da ottimizzare ed è completamente determinata dal processo di diffusione che abbiamo definito a priori; inoltre, il modello $p_\theta(x_T)$ coincide con un rumore gaussiano isotropo, cioè una distribuzione normale in cui la varianza è la stessa in tutte le direzioni e non privilegia alcun asse dello spazio

latente. In altre parole, non vi è alcun guadagno reale nell'ottimizzare questo termine. Un discorso analogo vale per l'ultimo termine, $-\log p_\theta(x_0 \mid x_1)$, che si riferisce al passo finale di ricostruzione diretta dal primo stato rumoroso. Nella pratica questo contributo viene spesso trascurato o trattato separatamente, poiché il suo impatto sull'ottimizzazione complessiva è marginale e non influisce in maniera significativa sulla capacità del modello di apprendere il processo di denoising lungo gli altri passi della catena.

Espressioni di q e p_θ

Sappiamo che:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\Big(x_{t-1};\, \mu_\theta(x_t, t),\, \beta_t \mathbf{I}\Big), \tag{2.24}$$

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\Big(x_{t-1};\, \tilde{\mu}_t(x_t, x_0),\, \tilde{\beta}_t \mathbf{I}\Big), \tag{2.25}$$

dove:

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\, x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\, x_t, \tag{2.26}$$

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\, \beta_t. \tag{2.27}$$

Dalla definizione del forward:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon,$$

ricaviamo:

$$x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \sqrt{1-\bar{\alpha}_t}\, \varepsilon \right).$$

Sostituendo in $\tilde{\mu}_t$ e semplificando:

$$\tilde{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\, \varepsilon \right). \tag{2.28}$$

Loss per passo

La KL gaussiana fra q e p_θ porta a:

$$\ell_t = \frac{1}{2\sigma_t^2} \left\| \tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t) \right\|^2. \tag{2.29}$$

Parametrizzando la media del reverse in funzione del rumore predetto:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \, \varepsilon_\theta(x_t, t) \right), \tag{2.30}$$

la loss diventa:

$$\ell_t = \frac{\beta_t^2}{2\sigma_t^2 \, \alpha_t (1-\bar{\alpha}_t)} \left\| \varepsilon - \varepsilon_\theta(x_t, t) \right\|^2. \tag{2.31}$$

Forma "simple" finale. Tralasciando costanti e pesi, la loss usata in pratica è:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t, x_0, \varepsilon} \left[\left\| \varepsilon - \varepsilon_\theta\left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t} \, \varepsilon, \, t \right) \right\|^2 \right], \tag{2.32}$$

dove $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$ e t è campionato uniformemente da $\{1, \ldots, T\}$.

Regola di campionamento nel reverse e procedure operative. Una volta addestrato il modello, la generazione avviene applicando ricorsivamente la regola di aggiornamento:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \, \varepsilon_\theta(x_t, t) \right) + \sigma_t z, \tag{2.33}$$

dove $z \sim \mathcal{N}(0, \mathbf{I})$ se $t > 1$ e $z = 0$ se $t = 1$. Qui σ_t è tipicamente scelto come $\sqrt{\beta_t}$ per rispettare la varianza del processo inverso.

2.4 Training e Sampling nei DDPM

Il funzionamento dei DDPM può essere sintetizzato in due procedure principali:

- **Training**, durante il quale la rete neurale impara a predire il rumore ε iniettato nei dati a diversi livelli di rumore;
- **Sampling**, che consiste nel partire da rumore puro e applicare il processo inverso per generare campioni realistici.

2.4.1 Procedura di Training

Algorithm 1 Training di un DDPM

repeat
  $x_0 \sim q(x_0)$ ▷ Campione reale dal dataset
  $t \sim \text{Uniform}(\{1, \dots, T\})$ ▷ Scelta casuale del timestep
  $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$ ▷ Rumore gaussiano standard
  Aggiorna θ minimizzando:

$$\left\| \varepsilon - \varepsilon_\theta\left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\, \varepsilon,\, t\right) \right\|^2$$

until convergenza

Durante l'addestramento:

1. Si campiona un'immagine reale x_0.
2. Si sceglie un timestep t casuale.
3. Si aggiunge rumore per ottenere x_t.
4. La rete ε_θ predice il rumore ε.
5. Si minimizza l'MSE tra rumore vero e predetto.

2.4.2 Procedura di Sampling

Algorithm 2 Sampling da un DDPM

$x_T \sim \mathcal{N}(0, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  if $t > 1$ **then**
    $z \sim \mathcal{N}(0, \mathbf{I})$
  else
    $z = 0$
  end if
  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta(x_t, t) \right) + \sigma_t z$
end for
return x_0

Durante il sampling:

1. Si parte da un rumore puro x_T.

2. Ad ogni passo, la rete predice il rumore in x_t.

3. Si rimuove il rumore e si aggiunge rumore stocastico z (tranne all'ultimo passo).

4. Dopo T passi si ottiene x_0.

2.5 Architettura del modello nei DDPM

In questa sezione descriviamo in dettaglio la struttura della **UNet** utilizzata nei modelli di diffusione e il meccanismo di **time embedding**, evidenziando come quest'ultimo venga integrato all'interno dell'architettura per condizionare ogni passo del processo di denoising.

2.5.1 Architettura UNet

La UNet nei modelli di diffusione è una rete convoluzionale encoder–decoder caratterizzata da *skip connections* tra i livelli di pari risoluzione. Questa architettura permette di combinare il **contesto globale**, ottenuto riducendo progressivamente la risoluzione delle feature map, con i **dettagli locali**, preservati e recuperati grazie ai collegamenti diretti tra encoder e decoder.

L'encoder, che segue un percorso di downsampling, riduce la risoluzione spaziale aumentando al contempo il numero di canali per estrarre caratteristiche sempre più astratte. Il bottleneck, ovvero il punto in cui la risoluzione è minima e il campo recettivo massimo, è spesso integrato con moduli di attenzione per catturare relazioni a lungo raggio tra le feature. Infine, il decoder, che segue il percorso di upsampling, ricostruisce la risoluzione originale combinando le informazioni provenienti dall'encoder tramite i collegamenti di skip connection.

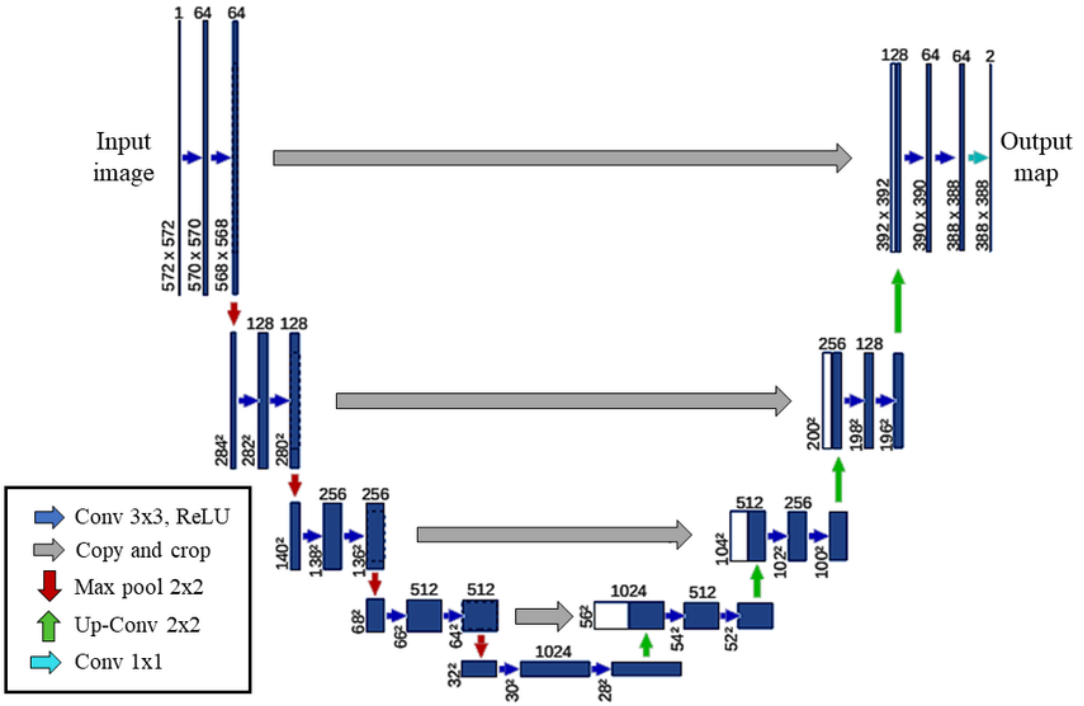

Figura 2.11: Architettura UNet classica con percorso encoder–decoder e connessioni *skip*. La riduzione della risoluzione avviene nel percorso di codifica tramite operazioni di downsampling, mentre il recupero della risoluzione originale è effettuato nel percorso di decodifica tramite upsampling.

Skip connections Le connessioni *skip* mettono in comunicazione i livelli dell'encoder e del decoder alla stessa risoluzione spaziale. Il loro scopo è preservare e trasferire informazioni locali ad alta frequenza, che andrebbero altrimenti perse nei passaggi di downsampling. In fase di ricostruzione, queste feature vengono concatenate alle mappe del decoder, consentendo di combinare dettagli strutturali con rappresentazioni semantiche più astratte. Questo meccanismo è cruciale per la qualità delle ricostruzioni nei modelli di diffusione, poiché permette di mantenere coerenza spaziale e nitidezza anche dopo molti strati convoluzionali.

Downsampling block

Ogni blocco di downsampling ha il compito di ridurre progressivamente la risoluzione spaziale delle feature map, aumentando al contempo la profondità e l'astrazione delle rappresentazioni. Il processo inizia con una serie di convoluzioni 3×3 seguite da funzioni di attivazione non lineari, come ReLU o sue varianti, che permettono di catturare pattern complessi nei dati.

Per garantire stabilità numerica e favorire la convergenza, le convoluzioni vengono accompagnate da meccanismi di normalizzazione, tipicamente la *Group Normalization*.

La riduzione della dimensione spaziale è realizzata tramite convoluzioni con stride pari a 2 o, in alternativa, operazioni di pooling, consentendo di comprimere gradualmente l'informazione e ampliare il campo recettivo della rete.

All'interno di questi blocchi sono spesso presenti anche *residual connections*, che facilitano il flusso del gradiente durante l'addestramento e permettono di preservare l'informazione utile evitando la degradazione delle feature. Infine, viene integrato il contributo del **time embedding**, proiettato sui canali della feature map: in questo modo, la rete incorpora in maniera esplicita la consapevolezza del passo temporale t, condizionando le rappresentazioni intermedie al contesto del processo di denoising.

Upsampling block

Ogni blocco di upsampling ha l'obiettivo di ricostruire progressivamente la risoluzione originale dell'input, integrando al tempo stesso i dettagli provenienti dall'encoder. La prima operazione consiste nell'aumentare la risoluzione spaziale, tipicamente tramite convoluzioni trasposte oppure con uno schema di interpolazione seguita da convoluzione, così da recuperare gradualmente la dimensione originaria dei dati senza compromettere la coerenza locale.

Successivamente, le feature map ottenute vengono arricchite concatenandole con quelle corrispondenti del percorso di codifica attraverso le *skip connections*. Questo passaggio è cruciale poiché consente al decoder di riutilizzare i dettagli locali che sarebbero andati persi nella fase di compressione, evitando che la ricostruzione si basi esclusivamente su rappresentazioni troppo astratte.

Le mappe combinate vengono quindi sottoposte ad un processo di affinamento, che include convoluzioni 3×3, funzioni di attivazione non lineari e meccanismi di normalizzazione. In questo modo, l'informazione globale e i dettagli locali vengono integrati in maniera armonica, producendo rappresentazioni stabili e coerenti.

Infine, anche nei blocchi di upsampling viene iniettato il contributo del **time embedding**.

Bottleneck e attenzione

Il bottleneck rappresenta il punto di minima risoluzione della rete, in cui le feature map hanno una dimensione spaziale ridotta ma un numero elevato di canali. In questa fase la rappresentazione è altamente compressa e astratta, e il campo recettivo della rete è al massimo. Per arricchire ulteriormente l'informazione, è comune introdurre meccanismi di **self-attention** o di **multi-head attention**.

Questi moduli consentono di modellare in maniera esplicita le dipendenze a lungo raggio tra regioni anche molto distanti dell'immagine, superando i limiti locali delle convoluzioni. In tal modo la rete integra un *contesto globale* che si rivela cruciale nella successiva fase di ricostruzione, permettendo al decoder di generare output più coerenti e consistenti dal punto di vista semantico.

2.5.2 Time embedding e iniezione nella UNet

Nei modelli di diffusione, il passo temporale t indica il livello di rumore nell'input x_t. Il **time embedding** fornisce alla rete un'informazione esplicita su questo livello, permettendo di modulare il comportamento di ogni blocco.

2.5.3 Codifica sinusoidale

Il passo t viene codificato come:

$$\mathrm{TE}(t)_{2k} = \sin\left(\frac{t}{10000^{2k/d}}\right), \qquad \mathrm{TE}(t)_{2k+1} = \cos\left(\frac{t}{10000^{2k/d}}\right), \tag{2.34}$$

dove d indica la dimensione dell'embedding.

Proiezione e iniezione

L'informazione temporale, inizialmente rappresentata tramite un embedding sinusoidale, viene trasformata attraverso uno o più strati fully-connected, spesso intervallati da funzioni di attivazione non lineari che ne arricchiscono la capacità rappresentativa. La proiezione così ottenuta viene poi rimodellata e aggiunta, tramite una somma con broadcast, ai canali delle feature map all'interno di ciascun *residual block*. In questo modo il contributo del time embedding si integra direttamente con le rappresentazioni intermedie della rete. L'iniezione dell'informazione temporale non è confinata a una singola parte della rete, ma viene applicata in maniera coerente sia nei blocchi di downsampling che in quelli di upsampling. Ciò assicura che la consapevolezza del passo temporale influenzi l'intero processo di denoising, guidando la rete a generare aggiornamenti consistenti lungo tutta la catena di trasformazioni.

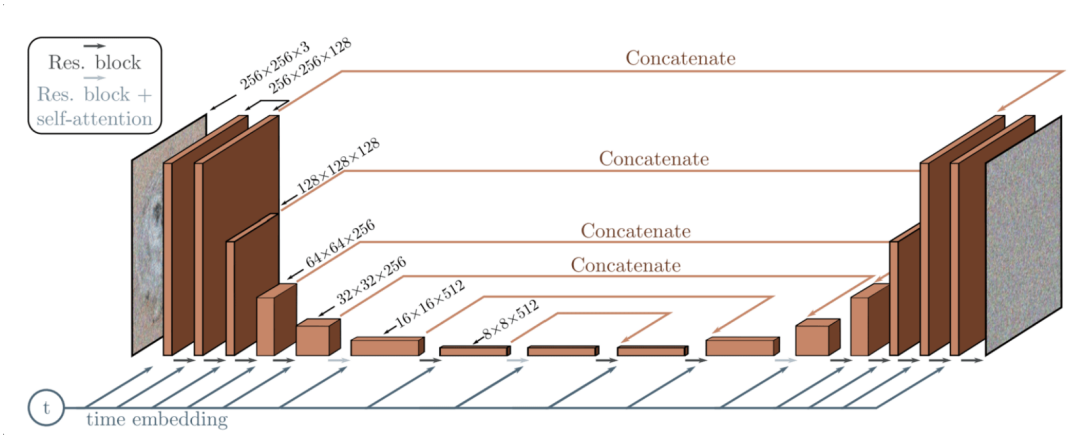

Figura 2.12: Architettura UNet adattata ai modelli di diffusione, con iniezione del *time embedding* in ciascun blocco e moduli di self-attention inseriti nei livelli a risoluzione intermedia e bassa.

Ruolo del time embedding

Il time embedding svolge un ruolo fondamentale nei modelli di diffusione. Nei primi passi (valori bassi di t, con rumore ridotto), il time embedding guida la rete a preservare i dettagli locali. Nei passi successivi (valori alti di t, con rumore più elevato), orienta la rete verso il recupero di strutture globali. Permette di usare un unico modello per tutti i timestep, evitando di addestrare reti distinte per ciascun valore di t.

Sintesi del flusso dati

Il flusso dati nel modello segue un percorso ben definito. L'input x_t attraversa il percorso dell'encoder, dove subisce una serie di convoluzioni e riduzioni di risoluzione. Allo stesso tempo, il *time embedding*, calcolato a partire dal passo temporale t, viene iniettato in ogni blocco della rete. Nel bottleneck, la rete utilizza moduli di attenzione per integrare il contesto globale, catturando relazioni a lungo raggio tra le feature. Nel decoder, le feature vengono upsampled e fuse con le corrispondenti informazioni provenienti dall'encoder, per ricostruire l'immagine. Alla fine, la rete restituisce $\varepsilon_\theta(x_t, t)$, che rappresenta una stima del rumore presente nell'input x_t.

2.6 Classifier-Free Guidance

Un aspetto fondamentale dei modelli di diffusione è la possibilità di introdurre un meccanismo di *guidance*, ossia un sistema che orienti la generazione verso campioni coerenti con un vincolo o una condizione esterna. Questo principio può essere

interpretato matematicamente come l'introduzione di un *campo esterno*, in grado di modificare la traiettoria media del reverse process senza alterarne l'incertezza intrinseca [12].

2.6.1 Posteriori come campi esterni

Moltiplicare la distribuzione generativa $p(x^{(0)})$ per una funzione $r(x^{(0)})$ (ad esempio una likelihood che codifica un vincolo o un'evidenza osservata) equivale a favorire i campioni compatibili con r. Dal punto di vista del reverse process, questo effetto si traduce in una forza esterna che devia la dinamica di denoising verso regioni dello spazio che rispettano il vincolo imposto.

Nel caso di transizioni gaussiane, la struttura della distribuzione rimane semplice. La **covarianza** $f_\Sigma(x^{(t)}, t)$ resta invariata, il che implica che il livello di incertezza del passo inverso non subisce modifiche. Al contrario, la **media** $f_\mu(x^{(t)}, t)$ viene traslata di una quantità proporzionale al gradiente $\nabla \log r(x^{(0)})$, che agisce come una forza orientante, guidando la ricostruzione verso una versione più precisa dell'input originale.

Formalmente:

$$\tilde{p}(x^{(t-1)} \mid x^{(t)}) \approx \mathcal{N}\Big(x^{(t-1)};\ f_\mu(x^{(t)}, t) + f_\Sigma(x^{(t)}, t)\, \nabla \log r,\ f_\Sigma(x^{(t)}, t)\Big).$$

Questa interpretazione fornisce una cornice unificata per comprendere tecniche come il *denoising* (dove r penalizza configurazioni rumorose), l'*inpainting* (dove r vincola solo le regioni osservate) e il *guidance condizionato*, in cui r rappresenta la coerenza con una condizione esterna (es. testo o etichetta di classe).

2.6.2 Dal classifier guidance al classifier-free guidance

Un primo approccio al guidance consiste nell'aggiungere al modello di diffusione un classificatore esterno, capace di fornire un gradiente $\nabla \log p(c \mid x)$ che spinga le traiettorie verso campioni compatibili con la classe desiderata. Questo metodo incrementa la qualità visiva ma richiede l'addestramento di un modello separato e presenta limiti di stabilità.

Il **classifier-free guidance (CFG)** supera queste difficoltà evitando completamente il classificatore. L'idea è di addestrare un'unica rete neurale sia in modalità condizionata che non condizionata: durante il training, con probabilità p_{uncond} si rimuove il condizionamento (impostando $c = \emptyset$), così che il modello apprenda entrambe le situazioni.

2.6.3 Formula di combinazione

In fase di campionamento, il modello produce due predizioni distinte. La prima, $\varepsilon_\theta(z_\lambda, c)$, rappresenta lo score condizionato da una certa informazione c (ad esempio, una classe o un prompt testuale). La seconda, $\varepsilon_\theta(z_\lambda)$, è lo score non condizionato, che riflette la dinamica "pura" del processo di diffusione senza alcun vincolo esterno.

Lo *score guidato* si ottiene combinando linearmente i due contributi:

$$\tilde{\varepsilon}_\theta(z_\lambda, c) \;=\; (1+w)\,\varepsilon_\theta(z_\lambda, c) \;-\; w\,\varepsilon_\theta(z_\lambda),$$

dove $w \geq 0$ è un iperparametro di controllo.

Dal punto di vista intuitivo, per $w = 0$ si ottiene semplicemente il modello condizionato standard. Aumentando w, la differenza tra lo score condizionato e quello non condizionato viene amplificata, spingendo il modello con maggiore decisione verso campioni che siano coerenti con la condizione c. Valori molto alti di w portano a immagini che sono visivamente molto fedeli al prompt o alla condizione, ma a scapito della diversità, poiché i campioni tendono a diventare simili fra loro.

In sintesi, il parametro w introduce un *trade-off controllabile* tra diversità e fedeltà condizionata: valori bassi privilegiano una copertura più ampia della distribuzione dei dati, mentre valori elevati enfatizzano la coerenza con il condizionamento a scapito della varietà.

2.6.4 Interpretazione e risultati

Il CFG può essere interpretato come un'applicazione pratica del concetto di campo esterno: la componente non condizionata agisce come termine “repulsivo”, mentre la componente condizionata spinge il campione verso la distribuzione desiderata. L'effetto complessivo è un bilanciamento tra fedeltà e diversità. In particolare, valori bassi di w privilegiano la diversità, mentre valori alti di w favoriscono la fedeltà percettiva. Questa semplice procedura ha reso il CFG una componente essenziale nei moderni modelli di diffusione condizionati, consentendo di ottenere risultati di alta qualità senza necessità di classificatori esterni.

Capitolo 3

Variational Autoencoder (VAE)

3.1 Introduzione e principi generali

I *Variational Autoencoder* (VAE) rappresentano una classe di modelli generativi basati sull'inferenza variazionale. L'idea di fondo è estendere la struttura classica degli *autoencoder* introducendo una formulazione probabilistica, così da permettere non solo la ricostruzione dei dati in ingresso, ma anche la generazione di nuovi campioni da una distribuzione latente appresa.

A differenza di un autoencoder standard, in cui l'encoder mappa il dato osservato in un codice latente deterministico, nel VAE l'encoder definisce una distribuzione approssimata $q_\phi(z|x)$ nello spazio latente. L'obiettivo è avvicinare questa distribuzione posteriori al vero $p_\theta(z|x)$, generalmente intrattabile, tramite il principio dell'*evidence lower bound* (ELBO). Il decoder, a sua volta, genera campioni nello spazio osservabile tramite la distribuzione $p_\theta(x|z)$.

Il vantaggio principale di questa formulazione è che lo spazio latente non funge più da semplice contenitore compressivo, ma acquisisce una struttura probabilistica capace di riflettere i fattori generativi sottostanti ai dati. In tale prospettiva, i VAE consentono di campionare nuovi dati realistici a partire da variabili latenti $z \sim p(z)$ e, al contempo, di apprendere rappresentazioni latenti interpretabili che, in alcune varianti, risultano anche *disentangled*, ossia sensibili a singoli fattori di variazione indipendenti. Inoltre, l'addestramento si mantiene stabile e scalabile grazie all'impiego del *reparameterization trick* e di obiettivi basati sulla *likelihood*, rendendo i VAE modelli versatili e solidi per la generazione e l'analisi dei dati [13], [14].

3.2 Formulazione probabilistica e reparameterization trick

Il *Variational Autoencoder* è un modello generativo che assume l'esistenza di variabili latenti non osservabili z, dalle quali i dati osservati x vengono generati secondo una distribuzione condizionata $p_\theta(x|z)$. La formulazione probabilistica completa è data da:

$$p_\theta(x, z) = p_\theta(x|z)\, p(z), \tag{3.1}$$

dove $p(z)$ è la distribuzione prior sullo spazio latente, tipicamente scelta come $\mathcal{N}(0, I)$ per ragioni di semplicità e regolarizzazione.

L'obiettivo è stimare la distribuzione marginale dei dati:

$$p_\theta(x) = \int p_\theta(x|z)\, p(z)\, dz, \tag{3.2}$$

ma questa integrazione risulta in generale intrattabile, soprattutto a causa dell'alta dimensionalità dello spazio latente. Per ovviare a questo problema, si introduce una distribuzione *variazionale* $q_\phi(z|x)$ che approssima il vero posteriore $p_\theta(z|x)$. L'addestramento del VAE consiste dunque nel massimizzare la log-likelihood dei dati tramite il bound variazionale (Evidence Lower Bound, ELBO), che verrà approfondito nella sezione successiva.

Sampling e difficoltà di backpropagation

Generare nuovi campioni dal modello prevede di estrarre prima un vettore latente $z \sim q_\phi(z|x)$ e quindi passare a un dato $x \sim p_\theta(x|z)$. Tuttavia, se z viene campionato in modo stocastico, il gradiente della loss rispetto ai parametri ϕ dell'encoder non può essere calcolato in maniera diretta, impedendo l'applicazione standard della backpropagation.

Reparameterization trick

Per risolvere questa problematica, si utilizza il **reparameterization trick**. L'idea è di esprimere il campione latente z come trasformazione deterministica di una variabile aleatoria ε indipendente:

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I), \tag{3.3}$$

dove $\mu_\phi(x)$ e $\sigma_\phi(x)$ sono le uscite dell'encoder e $\odot$ indica il prodotto elemento per elemento. In questo modo, la stocasticità viene confinata in ε, che non dipende dai parametri del modello, mentre la mappatura $\mu_\phi(x), \sigma_\phi(x)$ resta differenziabile rispetto a ϕ.

Questo trucco consente di applicare la backpropagation attraverso il campionamento, rendendo l'addestramento del VAE stabile ed efficiente. Inoltre, la parametrizzazione gaussiana dello spazio latente consente al modello di generare campioni continui e di interpolare agevolmente tra punti diversi nello spazio latente.

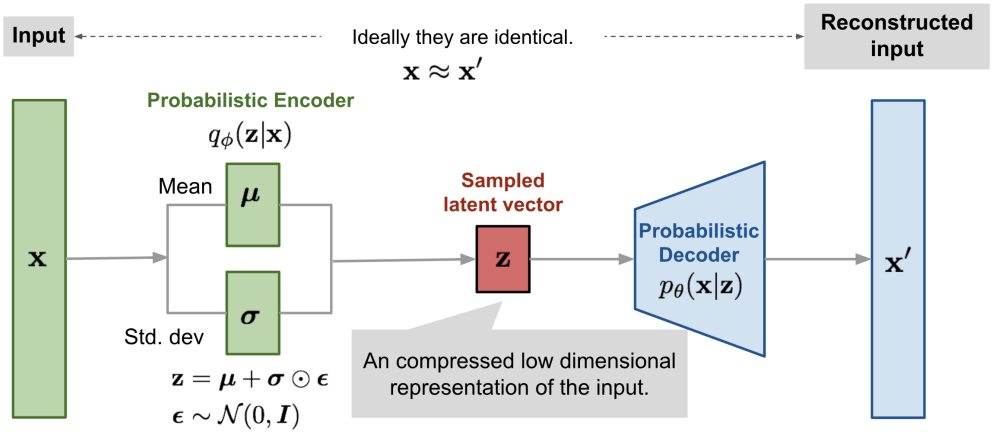

Figura 3.1: Schema del Variational Autoencoder (VAE). L'encoder probabilistico $q_\phi(z|x)$ mappa l'input x in una distribuzione gaussiana parametrizzata da media μ e deviazione standard σ. Il reparameterization trick permette di campionare il vettore latente $z = \mu + \sigma \odot \varepsilon$ con $\varepsilon \sim \mathcal{N}(0, I)$, garantendo la differenziabilità del processo. Il decoder $p_\theta(x|z)$ ricostruisce l'input x', che idealmente approssima l'input originale x.

3.3 Evidence Lower Bound (ELBO)

3.3.1 Derivazione matematica

L'obiettivo di un VAE è massimizzare la log-likelihood dei dati osservati x rispetto ai parametri θ del modello generativo:

$$\log p_\theta(x) = \log \int p_\theta(x \mid z)\, p(z)\, dz. \tag{3.4}$$

Tuttavia, l'integrale sullo spazio latente z è in generale intrattabile, poiché lo spazio può avere dimensionalità elevata e la funzione integranda non è nota in forma chiusa. Per superare questo problema si introduce una distribuzione variazionale $q_\phi(z \mid x)$, che approssima il vero posteriore $p_\theta(z \mid x)$. Questa scelta permette di riscrivere la

log-likelihood come:

$$\log p_\theta(x) = \log \int q_\phi(z \mid x)\, \frac{p_\theta(x \mid z)p(z)}{q_\phi(z \mid x)}\, dz \tag{3.5}$$

$$= \log\, \mathbb{E}_{q_\phi(z|x)}\left[\frac{p_\theta(x \mid z)p(z)}{q_\phi(z \mid x)}\right]. \tag{3.6}$$

Applicando la **disuguaglianza di Jensen**, che per una funzione convessa f e una variabile aleatoria X vale:

$$f\Big(\mathbb{E}[X]\Big) \;\leq\; \mathbb{E}[f(X)].$$

E considerando che il logaritmo è concavo (quindi la disuguaglianza si inverte),si ottiene:

$$\log \mathbb{E}[X] \;\geq\; \mathbb{E}[\log X].$$

Applicando questo principio all'espressione precedente si ricava un limite inferiore (lower bound) della log-likelihood:

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x \mid z)p(z)}{q_\phi(z \mid x)}\right]. \tag{3.7}$$

Separando i termini si arriva alla forma canonica della **Evidence Lower Bound (ELBO)**:

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}\Big[\log p_\theta(x \mid z)\Big] - D_{\mathrm{KL}}\Big(q_\phi(z \mid x) \,\|\, p(z)\Big) \tag{3.8}$$

$$=: \mathcal{L}_{\mathrm{ELBO}}(\theta, \phi; x). \tag{3.9}$$

La ELBO massimizza contemporaneamente due obiettivi:

- **Accuratezza di ricostruzione**

  $$\mathbb{E}_{q_\phi(z|x)}\Big[\log p_\theta(x \mid z)\Big],$$

  che misura quanto bene il decoder $p_\theta(x \mid z)$ riesce a ricostruire i dati.
- **Regolarizzazione del latente**

  $$-D_{\mathrm{KL}}\Big(q_\phi(z \mid x) \,\|\, p(z)\Big),$$

  che penalizza le deviazioni tra il posteriore approssimato e il prior scelto $p(z)$ (tipicamente $\mathcal{N}(0, I)$).

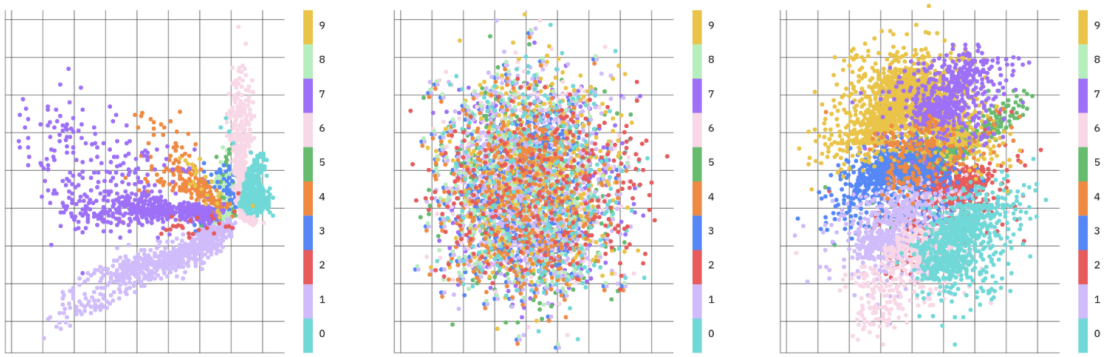

Figura 3.2: Effetto dei due termini della ELBO. A sinistra, con sola **ricostruzione**, lo spazio latente collassa in regioni disordinate e non regolarizzate. Al centro, con solo termine di **regolarizzazione KL**, i punti si dispongono secondo il prior ma senza struttura utile. A destra, la combinazione dei due termini produce uno spazio latente organizzato, in cui le classi sono separabili e coerenti con il prior.

3.3.2 Interpretazione pratica

Dal punto di vista operativo, la ELBO può essere interpretata come il risultato di un compromesso tra due esigenze contrastanti. Da un lato, il termine di **ricostruzione** assicura che l'autoencoder probabilistico mantenga l'informazione rilevante: dato un campione x, il decoder deve essere in grado di riprodurlo fedelmente a partire dalla corrispondente variabile latente z. Dall'altro, il termine di **regolarizzazione**, rappresentato dalla divergenza di Kullback–Leibler, forza la distribuzione latente $q_\phi(z \mid x)$ ad avvicinarsi al prior $\mathcal{N}(0, I)$. Questa spinta regolarizzante garantisce che lo spazio latente sia continuo, ben organizzato e adatto al campionamento di nuovi dati plausibili.

Se il termine di ricostruzione prevale eccessivamente, il modello rischia di adattarsi troppo ai dati di addestramento, arrivando a memorizzarli e producendo così uno spazio latente poco strutturato e irregolare. Al contrario, se domina il termine di regolarizzazione, lo spazio latente risulta ben organizzato e regolare, ma le ricostruzioni perdono fedeltà e dettaglio rispetto ai dati originali. Solo bilanciando correttamente queste due componenti si ottiene un modello capace, al tempo stesso, di ricostruire con buona qualità e di generare campioni nuovi e coerenti.

La ELBO rappresenta dunque non soltanto il criterio di addestramento del VAE, ma anche il principio che gli consente di unire capacità ricostruttiva e potere generativo in un unico quadro probabilistico.

3.4 Architettura del modello

Un *Variational Autoencoder* (VAE) è costituito da due componenti principali:

- un **encoder probabilistico** $q_\phi(z|x)$, che mappa un input x in una distribuzione latente gaussiana parametrizzata da media $\mu_\phi(x)$ e varianza $\sigma^2_\phi(x)$;
- un **decoder generativo** $p_\theta(x|z)$, che a partire da un campione z dallo spazio latente ricostruisce i dati nello spazio osservabile.

L'insieme encoder–decoder realizza una compressione e successiva ricostruzione dei dati, dove lo spazio latente ha una struttura probabilistica regolarizzata dal prior $p(z)$.

Encoder

L'encoder riduce progressivamente la dimensionalità dell'input tramite una sequenza di trasformazioni convoluzionali. Tipicamente, blocchi di convoluzioni sono arricchiti con normalizzazioni (ad esempio *Group Normalization*) e funzioni di attivazione non lineari (SiLU, ReLU o GELU). Per migliorare la stabilità dell'addestramento e la capacità di rappresentazione, si utilizzano spesso blocchi residuali, che aiutano a mantenere il flusso del gradiente nelle reti profonde. A risoluzioni intermedie possono essere inseriti meccanismi di attenzione, utili a cogliere dipendenze a lungo raggio e relazioni globali tra le feature estratte. Inoltre, lo schema prevede un downsampling gerarchico, realizzato tramite convoluzioni con stride o operazioni di pooling, che riducono progressivamente la risoluzione spaziale mentre aumentano la profondità delle feature.

Alla fine della catena di trasformazioni, l'encoder produce due mappe distinte, $\mu(x)$ e $\log \sigma^2(x)$, che definiscono i parametri della distribuzione latente gaussiana $q_\phi(z|x)$.

Decoder

Il decoder opera in direzione opposta: a partire dal campione latente z, proietta l'informazione verso lo spazio dei dati originali. La ricostruzione segue uno schema gerarchico di **upsampling**, in cui la risoluzione viene gradualmente aumentata tramite interpolazioni seguite da convoluzioni, oppure mediante convoluzioni trasposte con stride. Analogamente all'encoder, i blocchi di ricostruzione possono includere residual blocks, che stabilizzano il processo di generazione, e moduli di attenzione, che migliorano la coerenza semantica dell'immagine prodotta. Inoltre, l'uso di tecniche come la normalizzazione e il dropout regolarizza l'apprendimento. Il decoder termina

con una convoluzione finale che restituisce un output nello spazio dei dati originali, ad esempio un'immagine con lo stesso numero di canali dell'input.

Schema complessivo

L'architettura complessiva del VAE realizza dunque una mappatura $x \mapsto (\mu(x), \sigma^2(x)) \mapsto z \mapsto \hat{x}$, dove:

$$z = \mu(x) + \sigma(x) \odot \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I). \tag{3.10}$$

Il reparameterization trick garantisce la differenziabilità del passaggio stocastico, consentendo di addestrare congiuntamente encoder e decoder tramite backpropagation.

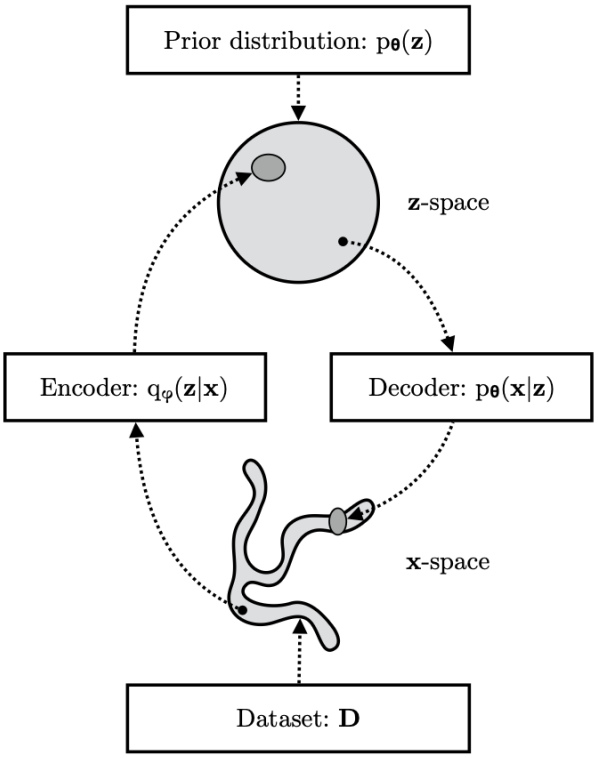

Figura 3.3: Schema concettuale di un VAE: l'encoder $q_\phi(z|x)$ proietta i dati nello spazio latente regolarizzato dal prior $p(z)$, mentre il decoder $p_\theta(x|z)$ ricostruisce l'input.

Dal punto di vista architetturale, l'adozione di blocchi residuali e meccanismi di attenzione si è affermato come pratica consolidata per migliorare sia la stabilità dell'ottimizzazione sia la qualità dei campioni generati. Queste soluzioni aiutano a bilanciare efficacemente la capacità rappresentativa del modello con la sua scalabilità, rendendo i VAE componenti fondamentali all'interno di modelli generativi più complessi, come ad esempio i modelli di diffusione condizionati su spazi latenti.

3.5 Pseudocodice di training e sampling

Per completezza, si riportano di seguito gli pseudocodici che descrivono in forma operativa le due fasi fondamentali di un **Variational Autoencoder (VAE)**. Il primo algoritmo illustra il procedimento di *training*, basato sulla minimizzazione della ELBO mediante il reparameterization trick. Il secondo algoritmo mostra invece la procedura di *sampling*, attraverso la quale è possibile generare nuovi dati a partire dal prior latente e dal decoder.

Algorithm 3 Training di un VAE

1: **repeat**
2: $x \sim q(x)$ ▷ Campione reale dal dataset
3: Ottieni parametri latenti: $\mu_\phi(x), \sigma_\phi(x)$
4: Campiona $\varepsilon \sim \mathcal{N}(0, \mathbf{I})$
5: Reparameterization: $z \leftarrow \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon$
6: Ricostruzione: $\hat{x} \sim p_\theta(x|z)$
7: Calcola la loss ELBO:

$$\mathcal{L}(x; \theta, \phi) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\mathrm{KL}}(q_\phi(z|x) \,\|\, p(z))$$

8: Aggiorna i parametri θ, ϕ tramite backpropagation
9: **until** convergenza

Algorithm 4 Sampling da un VAE

1: Campiona $z \sim p(z) = \mathcal{N}(0, \mathbf{I})$
2: Genera $\hat{x} \sim p_\theta(x|z)$
3: **return** $\hat{x}$ come nuovo campione generato

3.6 Limiti, varianti e confronto con i DDPM

3.6.1 Limiti principali

Nonostante la solidità della loro formulazione matematica, i VAE presentano alcune limitazioni nella pratica. Tra queste, le ricostruzioni tendono a risultare sfocate, soprattutto con immagini complesse, a causa dell'assunzione di una distribuzione gaussiana semplice nello spazio latente. Inoltre, in certi casi si verifica il posterior collapse, ovvero il decoder tende a ignorare la variabile latente z, trasformando di fatto il modello in un autoencoder deterministico. Infine, l'uso di un prior semplice e

isotropico, come la distribuzione normale standard N(0,I), può limitare la capacità del modello di rappresentare strutture latenti più complesse e articolate.

3.6.2 Varianti

Per superare i limiti dei VAE classici sono state sviluppate diverse varianti, ognuna pensata per risolvere specifici problemi. Tra queste, la **β-VAE** modifica la funzione obiettivo introducendo un fattore β che moltiplica il termine della divergenza di Kullback–Leibler. Quando β è maggiore di 1, la regolarizzazione sullo spazio latente diventa più severa, spingendo la distribuzione latente ad aderire meglio al prior e favorendo una separazione più netta dei fattori latenti, ovvero una *disentanglement* che rende le variabili più interpretabili e indipendenti. Va però considerato che questo miglioramento a livello di disentanglement porta spesso a ricostruzioni meno precise, meno fedeli ai dati originali.

Un altro approccio è rappresentato dal **VQ-VAE** (Vector Quantized VAE), che sostituisce lo spazio latente continuo con uno discreto, ottenuto tramite una tecnica di quantizzazione vettoriale. In questo modo, i codici latenti sono mappati su un dizionario finito di embedding discreti. Questa strategia risolve il problema della sovra-regolarizzazione gaussiana, migliora la nitidezza delle immagini generate e permette di combinare la compressione con modelli sequenziali potenti, come i Transformer o i modelli di diffusione applicati allo spazio latente discreto.

Infine, le varianti gerarchiche come **Hi-VAE** estendono il modello introducendo più livelli latenti disposti in modo gerarchico. Ogni livello cattura diversi fattori di variazione: quelli più alti modellano aspetti globali e semantici, come la forma complessiva di un oggetto, mentre i livelli più bassi si occupano dei dettagli locali. Questa struttura gerarchica consente di aumentare la capacità espressiva del modello e di rappresentare distribuzioni latenti più complesse e multimodali.

3.6.3 Confronto con i DDPM

I modelli di diffusione (*Denoising Diffusion Probabilistic Models*, DDPM) hanno recentemente superato i VAE in termini di qualità visiva, grazie alla capacità di modellare distribuzioni complesse senza assumere forme parametriche semplici per lo spazio latente.

(a) Campioni generati con un VAE: ricostruzioni tendono ad essere sfocate e meno realistiche.

(b) Campioni generati con un DDPM: le immagini risultano più nitide e fedeli alla distribuzione dei dati.

Figura 3.4: Confronto qualitativo tra campioni generati da un VAE e da un DDPM.

I VAE restano comunque competitivi in scenari in cui efficienza e compattezza della rappresentazione sono cruciali, mentre i DDPM eccellono nella generazione di immagini ad alta fedeltà, al costo di una maggiore complessità computazionale e tempi di campionamento più lunghi.

Capitolo 4

Diffuse-VAE

In questo capitolo viene presentato il **DiffuseVAE**, un modello ibrido che combina le proprietà dei VAE e dei DDPM. Dopo aver introdotto le motivazioni che sorreggono la sua definizione, ne verrà descritto il funzionamento generale e le principali formulazioni proposte in letteratura, con particolare attenzione alla *Formulation 1*, che costituisce il nucleo dell'implementazione sviluppata in questa tesi. Verranno quindi illustrate le scelte progettuali adottate, i vantaggi e le criticità del modello, per concludere con una discussione dei limiti emersi e delle possibili estensioni future [15].

4.1 Introduzione

I *Variational Autoencoder* (VAE) e i *Denoising Diffusion Probabilistic Models* (DDPM) presentano punti di forza complementari: i primi offrono uno spazio latente compatto e interpretabile, mentre i secondi garantiscono una qualità di campionamento superiore. Tuttavia, i VAE tendono a produrre ricostruzioni poco nitide, mentre i DDPM richiedono un numero elevato di passi di campionamento e non dispongono di una rappresentazione latente esplicita.

Il **DiffuseVAE** nasce con l'obiettivo di combinare i vantaggi di entrambi: utilizzare il VAE come meccanismo di codifica–decodifica che fornisce una ricostruzione preliminare e uno spazio latente strutturato, delegando al DDPM il compito di affinare progressivamente i dettagli e migliorare la qualità visiva dei campioni generati. In questo modo, il modello integra interpretabilità e compattezza con realismo e fedeltà, risultando particolarmente adatto come architettura generativa modulare e scalabile.

L'idea alla base di **DiffuseVAE** è quella di combinare i due approcci in una pipeline a due stadi:

1. uno **stadio VAE** che produce una ricostruzione preliminare dell'input;

2. uno **stadio DDPM** che agisce come *refiner*, migliorando la qualità del campione generato.

In questo modo, si ottiene un modello in grado di mantenere la struttura latente dei VAE e, al contempo, la qualità visiva dei DDPM.

4.2 Funzionamento generale

Il funzionamento di DiffuseVAE può essere riassunto in due fasi principali:

- **Stage 1 – VAE**: dato un input x_0, l'encoder produce i parametri della distribuzione latente, $\mu(x)$ e $\sigma^2(x)$, da cui si campiona un vettore z. Il decoder genera quindi una ricostruzione $\hat{x}_0 = p_\theta(x \mid z)$. Questa ricostruzione funge da *bozza iniziale*.
- **Stage 2 – DDPM**: il modello di diffusione viene addestrato a partire da rumore puro x_T e procede attraverso il reverse process $p(x_{t-1} \mid x_t, \hat{x}_0)$. Qui, il condizionamento avviene direttamente sulla ricostruzione del VAE, che guida il denoising verso campioni più realistici.

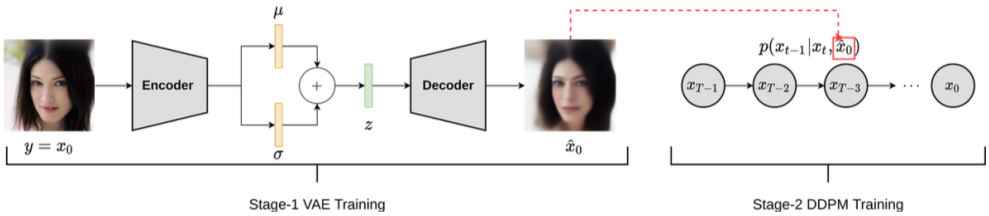

Figura 4.1: Schema del DiffuseVAE: nello **Stage 1** il VAE produce una ricostruzione preliminare $\hat{x}_0$; nello **Stage 2** il DDPM utilizza tale ricostruzione come condizionamento per affinare la generazione.

4.3 Formulazione 1

Nel paper originale vengono discusse diverse possibili formulazioni. Nella mia tesi è stata implementata la **Formulation 1**, caratterizzata da due assunzioni principali:

1. **Forward process indipendente**: le transizioni del processo forward non dipendono dal codice latente z né dalla ricostruzione $\hat{x}_0$, bensì solo dall'input originale:

$$q(x_{1:T} \mid z, x_0) \approx q(x_{1:T} \mid x_0).$$

2. **Reverse process condizionato**: le transizioni inverse dipendono unicamente dalla ricostruzione del VAE:

$$p(x_{0:T} \mid z) \approx p(x_{0:T} \mid \hat{x}_0).$$

In questo modo, il ruolo del VAE è quello di fornire una ricostruzione preliminare $\hat{x}_0$ che il DDPM utilizza come condizionamento durante tutte le fasi del reverse process. L'interpretazione intuitiva è che il VAE fornisce una bozza a bassa fedeltà, mentre il DDPM agisce come un raffinatore progressivo, correggendo i dettagli e portando il campione verso una distribuzione ad alta qualità visiva.

4.4 Implementazione nella tesi

Nella mia implementazione ho seguito la **Formulation 1**, integrando un VAE addestrato separatamente con un modello DDPM condizionato sulla sua ricostruzione. In particolare:

- il **VAE** è stato addestrato in modo classico, producendo ricostruzioni $\hat{x}_0$;
- il **DDPM** riceve in input lo stato rumoroso x_t concatenato alla ricostruzione $\hat{x}_0$, così da imparare a predire la componente rumorosa e guidare il denoising verso l'immagine pulita.

Questa scelta consente una chiara separazione tra la fase di rappresentazione latente (affidata al VAE) e la fase di raffinamento ad alta qualità (affidata al DDPM), ottenendo un modello più modulare e scalabile rispetto ai VAE o DDPM puri. Un ulteriore vantaggio di questa combinazione riguarda i **costi computazionali**: poiché il VAE condensa l'informazione in uno spazio latente strutturato e produce già una ricostruzione coerente, il DDPM non deve apprendere da zero la distribuzione globale dei dati, ma si concentra unicamente sul raffinamento dei dettagli locali. Questo comporta un addestramento più efficiente e una pipeline più semplice da scalare a dataset complessi.

4.5 Limiti e prospettive

Sebbene il DiffuseVAE rappresenti un passo significativo nella combinazione tra VAE e DDPM, non è privo di alcune criticità. La qualità complessiva del modello dipende fortemente dal VAE: se quest'ultimo produce ricostruzioni poco fedeli, il DDPM è in grado soltanto di attenuare parzialmente l'imprecisione, con un conseguente limite sulla qualità finale dei campioni generati. Inoltre, il condizionamento imposto dal

VAE introduce un compromesso tra fedeltà e diversità: da un lato la generazione risulta più stabile e coerente, dall'altro la varietà dei campioni tende a ridursi, poiché il modello rimane vincolato alla bozza iniziale. A questo si aggiunge il costo addestrativo più elevato, dovuto alla natura a due stadi della pipeline, che richiede prima il pre-addestramento del VAE e successivamente quello del DDPM. Infine, la scalabilità a domini più complessi, come dati ad alta risoluzione o multimodali, potrebbe richiedere architetture più sofisticate, ad esempio varianti gerarchiche di VAE o meccanismi di condizionamento avanzati.

Nonostante tali limitazioni, il DiffuseVAE si configura come un framework modulare e flessibile, che offre ampi margini di estensione. Tra le prospettive future più promettenti vi è l'impiego di VAE più espressivi, come i VQ-VAE o i β-VAE, in grado di fornire uno spazio latente meglio strutturato e più ricco di informazione. Un'altra direzione interessante è l'integrazione con tecniche di guidance condizionata, ad esempio mediante testo o etichette di classe, così da ottenere generazioni più controllabili e aderenti a vincoli esterni. Parallelamente, un obiettivo cruciale riguarda la riduzione dei tempi di campionamento, perseguibile attraverso reverse process accorciati, che permetterebbero di rendere la fase generativa molto più rapida ed efficiente.

In conclusione, il DiffuseVAE rappresenta un compromesso efficace tra efficienza e qualità visiva, ponendosi come una base solida su cui costruire sviluppi futuri. La sua natura modulare lo rende particolarmente adatto a essere adattato, potenziato e ottimizzato in funzione delle esigenze applicative.

Capitolo 5

Implementazione

In questo capitolo vengono presentati i dettagli implementativi delle architetture e dei modelli discussi nei capitoli precedenti. L'obiettivo non è fornire un'esposizione esaustiva del codice, ma mettere in evidenza le componenti essenziali come la struttura della UNet, i meccanismi di scheduling del rumore, i metodi di training e sampling, mostrando gli elementi chiave che collegano la teoria alla pratica.

5.1 DDPM e Diffuse-VAE

5.1.1 Architettura UNet

Timestep embedding. Il passo temporale t viene codificato tramite un embedding sinusoidale (Listing 5.1), che combina funzioni seno e coseno a frequenze diverse, in modo simile al *positional encoding* dei Transformer. Il risultato è un vettore denso di dimensione fissa che fornisce una rappresentazione continua e periodica del tempo. Per aumentarne l'espressività e adattarlo ai canali della rete, l'embedding viene ulteriormente elaborato da un multilayer perceptron (`self.t_proj`), composto da due trasformazioni lineari intervallate da un'attivazione `SiLU`. La funzione di attivazione `SiLU` (*Sigmoid Linear Unit*) è definita come

$$\mathrm{SiLU}(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}},$$

dove $\sigma(x)$ è la funzione sigmoid standard. Questa scelta introduce non linearità e modulazione proporzionale all'input, migliorando la capacità della rete di modellare relazioni complesse. In questo modo, l'embedding temporale può essere iniettato nei blocchi della U-Net, permettendo alla rete di modulare dinamicamente il processo di denoising in funzione del timestep.

```python
def timestep_embedding(timesteps: torch.Tensor, dim: int, max_period: int = 10000):
    assert dim % 2 == 0
    half = dim // 2
    freqs = torch.exp(-math.log(max_period) * torch.arange(0, half, device=timesteps.device, dtype=timesteps.dtype) / half)
    args = timesteps[:, None] * freqs[None, :]
    return torch.cat([torch.cos(args), torch.sin(args)], dim=-1)  # [B, dim]

self.t_proj = nn.Sequential(
    nn.Linear(self.t_emb_dim, self.t_emb_dim), nn.SiLU(),
    nn.Linear(self.t_emb_dim, self.t_emb_dim)
)
```

Listing 5.1: Embedding temporale sinusoidale e proiezione MLP

Downsampling: residual + time injection + attention. I blocchi di downsampling hanno il compito di ridurre progressivamente la risoluzione spaziale delle feature map, aumentando al tempo stesso la profondità del tensore e quindi la capacità rappresentativa della rete. Ogni blocco è costituito da due percorsi principali: il primo è un **residual path**, che applica convoluzioni 3×3 seguite da normalizzazione e attivazioni non lineari (`SiLU`); a questo viene aggiunto un collegamento residuo che preserva l'informazione di partenza e facilita la propagazione del gradiente. Il secondo contributo deriva dal **time embedding**, che viene proiettato in un vettore della stessa dimensionalità dei canali e iniettato additivamente nelle feature map. In questo modo, la rappresentazione rimane esplicitamente condizionata al passo temporale t del processo di diffusione.

Infine, all'interno di ciascun blocco viene integrato un modulo di **multi-head attention**, applicato sulle feature spazialmente appiattite: questa operazione consente di modellare dipendenze a lungo raggio tra diverse regioni dell'immagine, arricchendo la rappresentazione con un contesto globale che andrebbe altrimenti perso con sole convoluzioni locali. Dopo queste trasformazioni, una convoluzione con stride 2 (o un'operazione equivalente di pooling) effettua la riduzione di risoluzione vera e propria, comprimendo l'informazione spaziale e permettendo alla rete di ampliare progressivamente il proprio campo recettivo.

Il risultato è un blocco che non solo riduce dimensionalità e complessità spaziale, ma che integra dettagli locali, consapevolezza temporale e relazioni globali, fornendo feature profonde e strutturate utili per la fase di ricostruzione.

```
class DownBlock(nn.Module):
    def __init__(self, in_channels, out_channels, t_emb_dim,
    down_sample=True, num_heads=4, num_layers=1):
        ...
        self.resnet_conv_first = nn.Sequential(
            nn.GroupNorm(8, in_channels),
            nn.SiLU(),
            nn.Conv2d(in_channels, out_channels, kernel_size=3,
    padding=1)
        )
        self.t_emb_layer = nn.Linear(t_emb_dim, out_channels)
        self.attention = nn.MultiheadAttention(out_channels,
    num_heads, batch_first=True)
        self.down_sample_conv = nn.Conv2d(out_channels, out_channels,
    4, 2, 1)

    def forward(self, x, t_emb):
        out = self.resnet_conv_first(x)
        out = out + self.t_emb_layer(t_emb)[:, :, None, None]  #
    iniezione temporale
        b, c, h, w = out.shape
        attn_in = out.view(b, c, h*w).transpose(1, 2)
        out_attn, _ = self.attention(attn_in, attn_in, attn_in)
        out = out + out_attn.transpose(1, 2).view(b, c, h, w)
        out = self.down_sample_conv(out)  # riduzione risoluzione
        return out
```

Listing 5.2: Downsampling block con residual connections, time injection e attenzione

Mid-block con attenzione multi-head. Il *mid-block*, situato nel collo di bottiglia dell'architettura UNet, rappresenta il punto in cui le feature map hanno minima risoluzione spaziale ma profondità massima. In questa fase la rete dispone di un campo recettivo molto ampio, condizione ideale per integrare meccanismi in grado di catturare relazioni a lungo raggio tra regioni spazialmente distanti.

Il blocco alterna sequenze di convoluzioni residue e moduli di **multi-head self-attention**. I percorsi residui, costituiti da convoluzioni 3×3 seguite da attivazioni non lineari e normalizzazione, mantengono stabile il flusso dell'informazione e del gradiente, preservando i dettagli locali appresi nelle fasi precedenti. L'iniezione del

time embedding, aggiunta ai canali delle feature, assicura che la rappresentazione rimanga coerente con lo specifico passo temporale t del processo di diffusione.

In parallelo, i moduli di **self-attention** trasformano le feature map in sequenze di token e apprendono dipendenze contestuali globali, permettendo alla rete di correlare strutture visive anche molto distanti tra loro. Questa alternanza tra convoluzioni residue (per i dettagli locali) e attenzione (per il contesto globale) rende il mid-block un componente cruciale per combinare informazioni multi-scala e migliorare la capacità generativa della rete.

```
class MidBlock(nn.Module):
    def forward(self, x, t_emb):
        out = x
        # Primo percorso residuo
        res = out
        out = self.resnet_conv_first[0](out)
        out = out + self.t_emb_layers[0](t_emb)[:, :, None, None]
        out = self.resnet_conv_second[0](out)
        out = out + self.residual_input_conv[0](res)

        # Alternanza attenzione + residui
        for i in range(self.num_layers):
            # Self-attention multi-head
            b, c, h, w = out.shape
            tokens = out.view(b, c, h*w).transpose(1, 2)
            attn, _ = self.attentions[i](tokens, tokens, tokens)
            out = out + attn.transpose(1, 2).view(b, c, h, w)

            # Percorso residuo con iniezione temporale
            res = out
            out = self.resnet_conv_first[i+1](out)
            out = out + self.t_emb_layers[i+1](t_emb)[:, :, None,
    None]
            out = self.resnet_conv_second[i+1](out)
            out = out + self.residual_input_conv[i+1](res)
        return out
```

Listing 5.3: MidBlock: alternanza tra residui e attenzione multi-head

Upsampling e skip connections. La fase di ricostruzione dell'UNet avviene attraverso i blocchi di *upsampling*, che hanno il compito di riportare progressivamente

le feature map alla risoluzione originaria. Il processo inizia con un'operazione di upsampling, realizzata tramite convoluzioni trasposte o interpolazioni seguite da convoluzioni, che espandono la risoluzione spaziale preservando coerenza locale.

Un aspetto fondamentale è la presenza delle **skip connections**: le feature map prodotte dall'encoder, corrispondenti allo stesso livello di risoluzione, vengono concatenate con quelle del decoder. Questo meccanismo consente di reintegrare i dettagli locali persi nella fase di compressione, evitando che la ricostruzione si basi esclusivamente su rappresentazioni troppo astratte.

Dopo la fusione con le feature provenienti dall'encoder, le mappe risultanti attraversano percorsi residui costituiti da convoluzioni 3×3, arricchite dall'iniezione del **time embedding**. In questo modo la rete mantiene consapevolezza del passo temporale t durante tutto il processo di generazione.

Infine, anche nei blocchi di upsampling è integrato un meccanismo di **self-attention**, che consente di modellare relazioni a lungo raggio e garantisce coerenza globale nella ricostruzione. La combinazione di upsampling, skip connections, residui e attenzione rende questa fase essenziale per bilanciare dettagli locali e struttura complessiva.

```
class UpBlock(nn.Module):
    def forward(self, x, out_down, t_emb):
        # Upsampling tramite conv trasposta
        x = self.up_sample_conv(x)
        # Fusione con feature simmetriche dall'encoder
        x = torch.cat([x, out_down], dim=1)

        out = x
        for i in range(self.num_layers):
            # Percorso residuo con iniezione del time embedding
            res = out
            out = self.resnet_conv_first[i](out)
            out = out + self.t_emb_layers[i](t_emb)[:, :, None, None]
            out = self.resnet_conv_second[i](out)
            out = out + self.residual_input_conv[i](res)

            # Self-attention per catturare relazioni globali
            b, c, h, w = out.shape
            tokens = out.view(b, c, h*w).transpose(1, 2)
            attn, _ = self.attentions[i](tokens, tokens, tokens)
            out = out + attn.transpose(1, 2).view(b, c, h, w)
```

```
22          return out
```

Listing 5.4: UpBlock: upsampling, fusione con skip e attenzione

Condizionamento "Formulation-1". Nella pipeline Diffuse-VAE, la ricostruzione preliminare $\hat{x}_0$ generata dal VAE non viene utilizzata unicamente come informazione iniziale, ma viene integrata a più livelli della UNet. Questo approccio, noto come **Formulation-1**, permette al modello di sfruttare la bozza del VAE in maniera gerarchica, guidando il processo di denoising in modo più stabile ed efficace.

Il condizionamento avviene in tre punti distinti:

- all'ingresso della rete, dove x_t e $\hat{x}_0$ vengono concatenati e proiettati nello spazio dei canali originali tramite una convoluzione 1×1 (`fuse_in`);
- lungo il percorso di downsampling, dove le feature intermedie vengono arricchite aggiungendo la ricostruzione ridimensionata, fusa nuovamente con convoluzioni 1×1 (`fuse_down`);
- nelle skip connections durante l'upsampling, in cui la ricostruzione viene combinata con le feature provenienti dall'encoder per preservare i dettagli locali (`fuse_skip`).

Questo schema multi-scala garantisce che l'informazione del VAE permei l'intera architettura, fornendo un condizionamento coerente sia sui dettagli locali che sulla struttura globale. In altre parole, $\hat{x}_0$ funge da vincolo strutturale che accompagna il denoising ad ogni livello, evitando che il DDPM generi campioni incoerenti o troppo lontani dalla ricostruzione iniziale.

```
1  self.fuse_in   = nn.Conv2d(self.im_channels * 2, self.im_channels,
       kernel_size=1)
2
3  self.fuse_down = nn.ModuleList([
4      nn.Conv2d(ch + self.im_channels, ch, kernel_size=1)
5      for ch in self.down_skip_channels
6  ])
7
8  self.fuse_skip = nn.ModuleList([
9      nn.Conv2d(ch + self.im_channels, ch, kernel_size=1)
10     for ch in reversed(self.down_skip_channels)
11 ])
```

Listing 5.5: Layer di fusione per il condizionamento multi-scala

Outline del forward. Infine, uno schema semplificato mostra come il condizionamento viene applicato lungo il percorso.

```
def forward(self, x, t, cond=None):
    # Embedding temporale
    t_emb = self.t_proj(timestep_embedding(t, self.t_emb_dim))

    # Fusione iniziale
    if cond is not None:
        cond = F.interpolate(cond, size=x.shape[-2:],
    mode="bilinear", align_corners=False)
        x = self.fuse_in(torch.cat([x, cond], dim=1))
    out = self.conv_in(x)

    # Encoder con condizionamento
    skips = []
    for i, down in enumerate(self.downs):
        if cond is not None:
            cond_r = F.interpolate(cond, size=out.shape[-2:],
    mode="bilinear", align_corners=False)
            out = self.fuse_down[i](torch.cat([out, cond_r], dim=1))
        skips.append(out)
        out = down(out, t_emb)

    # Bottleneck
    for mid in self.mids:
        out = mid(out, t_emb)

    # Decoder con skip condizionati
    for i, up in enumerate(self.ups):
        skip = skips.pop()
        if cond is not None:
            cond_r = F.interpolate(cond, size=skip.shape[-2:],
    mode="bilinear", align_corners=False)
            skip = self.fuse_skip[i](torch.cat([skip, cond_r], dim=1))
        out = up(out, skip, t_emb)

    # Output finale
    return self.conv_out(nn.SiLU()(self.norm_out(out)))
```

Listing 5.6: Outline del forward con condizionamento multiscala

5.2 Noise scheduler

Il *noise scheduler* definisce la quantità di rumore iniettata ad ogni passo del processo di diffusione. La scelta della schedulazione influenza sia la stabilità dell'addestramento sia la qualità dei campioni generati, poiché controlla come la distribuzione dei dati viene progressivamente corrotta (forward process) e ricostruita (reverse process). Di seguito riportiamo due strategie comunemente adottate: lo **scheduler lineare** e lo **scheduler coseno**.

5.2.1 Linear Scheduler

Lo scheduler lineare introduce una progressione uniforme dei valori di rumore. I β_t crescono linearmente dal valore iniziale β_{start} fino a β_{end}, garantendo una corruzione graduale e stabile. Questo approccio è semplice ed efficace, ma può risultare meno ottimale nei passi iniziali o finali, dove la distribuzione del rumore influenza in maniera critica la dinamica del modello.

```
class LinearNoiseScheduler:
    def __init__(self, num_timesteps, beta_start, beta_end):
        self.num_timesteps = num_timesteps
        self.betas = torch.linspace(beta_start, beta_end,
    num_timesteps)
        self.alphas = 1.0 - self.betas
        self.alpha_bars = torch.cumprod(self.alphas, dim=0)

    def add_noise(self, x, noise, t):
        alpha_bar = self.alpha_bars[t.cpu()].to(x.device)
        return alpha_bar.sqrt().view(-1,1,1,1) * x + \
               (1 - alpha_bar).sqrt().view(-1,1,1,1) * noise
```

Listing 5.7: Linear scheduler

5.2.2 Cosine Scheduler

Lo scheduler coseno propone invece una schedulazione non lineare, in cui i valori $\bar{\alpha}_t$ seguono una curva a coseno. Questo schema, introdotto per ridurre la perdita di informazione nei primi passi, preserva meglio la struttura dei dati e tende a produrre campioni di qualità superiore. La definizione è:

$$\bar{\alpha}_t = \cos^2\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right),$$

normalizzata affinché $\bar{\alpha}_0 = 1$.

```
class CosineNoiseScheduler:
    def __init__(self, num_timesteps, s=0.008):
        self.num_timesteps = num_timesteps
        self.alpha_bars = self._compute_alpha_bars(num_timesteps, s)
        self.alphas = self.alpha_bars[1:] / self.alpha_bars[:-1]
        self.betas = 1.0 - self.alphas

    def _compute_alpha_bars(self, T, s):
        steps = torch.arange(0, T+1)
        f = torch.cos(((steps / T + s) / (1+s)) * math.pi / 2) ** 2
        return f / f[0]
```

Listing 5.8: Cosine scheduler

In sintesi, mentre lo scheduler lineare offre un controllo uniforme sulla diffusione, quello coseno gestisce in maniera più sofisticata la distribuzione della corruzione, migliorando la qualità della generazione soprattutto nelle prime fasi del processo.

5.2.3 Metodi forward e reverse

Forward process (DiffuseVAE). Durante la fase di addestramento il modello apprende a ricostruire il rumore iniettato nei dati. Dato un batch di immagini normalizzate $x \in [0, 1]$, si estrae un passo temporale t e un rumore gaussiano $\varepsilon \sim \mathcal{N}(0, I)$. Lo *noise scheduler* combina i due, generando la versione rumorosa $x_t = \sqrt{\bar{\alpha}_t}\, x + \sqrt{1 - \bar{\alpha}_t}\, \varepsilon$. La ricostruzione preliminare $\hat{x}_0$ ottenuta dal VAE viene utilizzata come condizionamento aggiuntivo per la U-Net, che in questo contesto non ricostruisce direttamente l'immagine, ma impara a predire il rumore ε responsabile della corruzione di x. In questo modo, il training forza la rete a modellare la distribuzione del rumore in funzione sia del passo temporale t sia della bozza $\hat{x}_0$, garantendo un legame stretto tra il processo di diffusione e lo spazio latente del VAE.

```
x = x.to(device)                    # [0,1]
with torch.no_grad():
    x_hat, _, _ = vae(x)            # condizionamento immagine
    x_hat = x_hat.clamp(0, 1)

B = x.size(0)
t = torch.randint(0, scheduler.num_timesteps, (B,),
    device=device).long()
eps = torch.randn_like(x)
```

```
x_t = scheduler.add_noise(x, eps, t)      # x_t = q(x_t | x, t)

eps_pred = unet(x_t, t, cond=x_hat)       # predizione rumore
loss = mse(eps_pred, eps)                  # obiettivo standard DDPM
```

Listing 5.9: Forward: add_noise e loss su rumore con condizionamento VAE

Reverse process (sampling guidato). Durante la generazione si parte da $x_T \sim \mathcal{N}(0, I)$ e si applica iterativamente il processo inverso per $t = T-1 \to 0$. Ad ogni passo la U-Net predice il rumore $\varepsilon_\theta(x_t, t, \hat{x}_0)$, che viene utilizzato dallo *noise scheduler* per stimare lo stato precedente x_{t-1}. La transizione può essere deterministica (schema DDIM) oppure includere una componente stocastica, producendo così campioni diversi a partire dalla stessa condizione iniziale.

```
@torch.no_grad()
def sample_grid(unet, vae, scheduler, x, device, cond_from_gt=True):
    unet.eval(); vae.eval()
    x_hat = vae(x)[0].clamp(0,1) if cond_from_gt else None

    def step_back(sched, x_t, eps_pred, t):
        try:    return sched.sample_prev_timestep(x_t, eps_pred, t,
    eta=0.0)
        except TypeError:
            return sched.sample_prev_timestep(x_t, eps_pred, t)

    x_t = torch.randn_like(x, device=device)
    for t_step in reversed(range(scheduler.num_timesteps)):
        t = torch.full((x_t.size(0),), t_step, device=device,
    dtype=torch.long)
        eps_pred = unet(x_t, t, cond=x_hat)
        x_t, _ = step_back(scheduler, x_t, eps_pred, t)
    return x_t.clamp(0, 1)
```

Listing 5.10: Sampling: loop reverse con condizionamento da VAE

5.2.4 Training

Passo di training (loss su rumore). Dato un campione reale $x \in [0, 1]$, durante l'addestramento si sceglie casualmente un passo temporale t e si genera rumore

gaussiano $\varepsilon \sim \mathcal{N}(0, I)$. Lo *noise scheduler* combina questi elementi costruendo

$$x_t = \sqrt{\bar{\alpha}_t}\, x + \sqrt{1 - \bar{\alpha}_t}\, \varepsilon,$$

che rappresenta la versione rumorosa di x al passo t.

La U-Net riceve in input x_t, insieme all'informazione temporale t e (nel caso di DiffuseVAE) alla ricostruzione $\hat{x}_0$ del VAE, e ha il compito di predire il rumore $\hat{\varepsilon}$. L'obiettivo del training consiste nel minimizzare la differenza tra il rumore predetto e quello reale attraverso una loss di tipo MSE, ossia

$$\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{x,\varepsilon,t}\Big[\|\hat{\varepsilon}(x_t, t, \hat{x}_0) - \varepsilon\|^2\Big].$$

In questo modo la rete impara a invertire progressivamente il processo di diffusione, acquisendo la capacità di rimuovere il rumore passo dopo passo fino a ricostruire un campione pulito.

```
model.train(); optimizer.zero_grad(set_to_none=True)

imgs = imgs.to(device)                          # [0,1]
t = torch.randint(0, scheduler.num_timesteps, (imgs.size(0),),
    device=device).long()
eps = torch.randn_like(imgs)
x_t = scheduler.add_noise(imgs, eps, t)     # x_t = sqrt(ab_t) * x +
    sqrt(1-ab_t) * eps

eps_pred = model(x_t, t)                        # U-Net predice il rumore
loss = F.mse_loss(eps_pred, eps)

loss.backward()
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)  #
    opzionale
optimizer.step()
```

Listing 5.11: Passo di training DDPM (loss su rumore)

Validazione (stessa loss, no grad).

```
model.eval()
with torch.no_grad():
    val_losses = []
    for imgs, _ in val_loader:
        imgs = imgs.to(device)
```

```
6           t = torch.randint(0, scheduler.num_timesteps,
    (imgs.size(0),), device=device).long()
7           eps = torch.randn_like(imgs)
8           x_t = scheduler.add_noise(imgs, eps, t)
9           eps_pred = model(x_t, t)
10          val_losses.append(F.mse_loss(eps_pred, eps).item())
11  avg_val = sum(val_losses)/len(val_losses)
```

Listing 5.12: Loop di validazione

5.2.5 Sampling

Denoising loop (DDPM/DDIM). Il processo di campionamento parte da $x_T \sim \mathcal{N}(0, I)$, cioè un rumore gaussiano puro. Si procede iterativamente dal passo T fino a 0: ad ogni iterazione la U-Net predice il rumore $\varepsilon_\theta(x_t, t, \hat{x}_0)$, e lo *scheduler* calcola lo stato precedente x_{t-1}. Il passo di reverse, in forma compatta, è descritto da:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta(x_t, t, \hat{x}_0) \right) + \sigma_t z, \quad z \sim \mathcal{N}(0, I),$$

dove σ_t controlla l'eventuale rumore aggiuntivo in fase di generazione.

Se si imposta $\eta = 0$, il termine stocastico $\sigma_t z$ scompare e si ottiene una dinamica deterministica (DDIM), in cui una stessa condizione iniziale produce sempre lo stesso campione. Con $\eta > 0$, invece, si mantiene una componente stocastica che aumenta la diversità dei campioni, come nel caso classico dei DDPM. In questo modo, il loop di denoising rappresenta il cuore della generazione: partendo da rumore casuale e guidati dal condizionamento $\hat{x}_0$, si ricostruiscono progressivamente immagini coerenti e di alta qualità.

```
1   model.eval()
2   x = torch.randn(num_samples, C, H, W, device=device)   # x_T
3   T = scheduler.num_timesteps
4
5   with torch.no_grad():
6       for t_step in reversed(range(T)):
7           t = torch.full((x.size(0),), t_step, device=device,
    dtype=torch.long)
8           eps_pred = model(x, t)
9           try:
10              x, _ = scheduler.sample_prev_timestep(x, eps_pred, t,
    eta=0.0)  # (x_{t-1}, x0_pred)
```

```
        except TypeError:
            x, _ = scheduler.sample_prev_timestep(x, eps_pred, t)

# denormalizzazione per salvataggio
x = torch.clamp(x, 0, 1)
save_image(x, "samples.png", nrow=int(math.sqrt(num_samples)))
```

Listing 5.13: Loop di sampling

5.3 VAE

5.3.1 Encoder e decoder

Blocchi residui. I *ResidualBlock* rappresentano una parte fondamentale sia nell'encoder che nel decoder. Ogni blocco applica due convoluzioni 3×3 intervallate da normalizzazione *GroupNorm* e attivazione *SiLU*, garantendo stabilità numerica e capacità di modellare relazioni non lineari. La skip connection somma direttamente l'input all'output del blocco, facilitando il flusso del gradiente ed evitando il degrado delle feature. Quando il numero di canali cambia, lo `shortcut` usa una convoluzione 1×1 (o 3×3) per adattare le dimensioni, mantenendo coerenza tra le rappresentazioni. In questo modo, i blocchi residui combinano apprendimento profondo e preservazione delle informazioni.

```
class ResidualBlock(nn.Module):
    def __init__(self, in_ch, out_ch=None, dropout=0.0, groups=8,
    use_conv_shortcut=False):
        super().__init__()
        out_ch = out_ch or in_ch
        self.norm1 = nn.GroupNorm(min(groups, in_ch), in_ch)
        self.conv1 = nn.Conv2d(in_ch, out_ch, 3, padding=1)
        self.norm2 = nn.GroupNorm(min(groups, out_ch), out_ch)
        self.dropout = nn.Dropout(dropout)
        self.conv2 = nn.Conv2d(out_ch, out_ch, 3, padding=1)
        self.shortcut = (
            nn.Identity() if in_ch == out_ch
            else nn.Conv2d(in_ch, out_ch, 1 if not use_conv_shortcut
    else 3, padding=0 if not use_conv_shortcut else 1)
        )

    def forward(self, x):
```

```
16          h = self.conv1(F.silu(self.norm1(x)))
17          h = self.conv2(self.dropout(F.silu(self.norm2(h))))
18          return h + self.shortcut(x)
```

Listing 5.14: ResidualBlock con GroupNorm e SiLU

Self-Attention. I blocchi di *self-attention* consentono di catturare dipendenze a lungo raggio tra regioni diverse dell'immagine, andando oltre il limitato campo recettivo delle convoluzioni locali. L'input viene normalizzato e trasformato in insiemi di query, key e value, sui quali si applica il meccanismo di attenzione scalato per ottenere una combinazione pesata delle feature spaziali. Questo consente alla rete di integrare informazioni globali e contestuali in ogni rappresentazione intermedia. La proiezione finale dei valori è inizializzata a zero per garantire stabilità durante le prime fasi di addestramento, evitando che l'attenzione perturbi eccessivamente l'ottimizzazione iniziale.

```
1  class AttentionBlock(nn.Module):
2      def __init__(self, ch, groups=8):
3          super().__init__()
4          self.norm = nn.GroupNorm(min(groups, ch), ch)
5          self.q = nn.Conv2d(ch, ch, 1); self.k = nn.Conv2d(ch, ch, 1);
       self.v = nn.Conv2d(ch, ch, 1)
6          self.proj_out = nn.Conv2d(ch, ch, 1)
7          nn.init.zeros_(self.proj_out.weight);
       nn.init.zeros_(self.proj_out.bias)
8
9      def forward(self, x):
10         b, c, h, w = x.shape
11         h_ = self.norm(x)
12         q = self.q(h_).reshape(b, c, h*w).permute(0, 2, 1)   # [B,
       HW, C]
13         k = self.k(h_).reshape(b, c, h*w)                    # [B, C,
       HW]
14         v = self.v(h_).reshape(b, c, h*w).permute(0, 2, 1)   # [B,
       HW, C]
15         attn = F.softmax(torch.bmm(q, k) / math.sqrt(c), dim=-1)
16         out  = torch.bmm(attn, v).permute(0, 2, 1).reshape(b, c, h, w)
17         return x + self.proj_out(out)
```

Listing 5.15: Self-attention 2D con proiezione zero-init

Encoder: compressione a $\mu, \log \sigma^2$. L'encoder ha il compito di comprimere l'immagine nello spazio latente. Dopo una convoluzione iniziale, la rappresentazione attraversa una sequenza di blocchi residui e, se previsto, moduli di self–attention che arricchiscono le feature con dipendenze globali. Ad ogni livello viene eseguito un downsampling con fattore 2, così da ridurre progressivamente la risoluzione e ampliare il campo recettivo. Nel collo di bottiglia, la rete stima i parametri della distribuzione gaussiana fattoriale $q_\phi(z \mid x) = \mathcal{N}(\mu, \sigma^2 I)$, dalla quale verrà campionato il vettore latente z. Per garantire stabilità numerica, i valori di $\log \sigma^2$ vengono vincolati a un intervallo predefinito, evitando esplosioni o degenerazioni durante l'addestramento.

```
class Encoder(nn.Module):
    def __init__(...):
        ...
        self.conv_in = nn.Conv2d(in_channels, base_channels, 3,
    padding=1)
        self.down_blocks = nn.ModuleList([...])  #
    ResidualBlock(+Attention)+Downsample ripetuti
        self.mid_block1 = ResidualBlock(in_ch, in_ch); self.mid_attn
    = AttentionBlock(in_ch)
        self.mid_block2 = ResidualBlock(in_ch, in_ch)
        self.norm_out = nn.GroupNorm(min(group_norm_groups, in_ch),
    in_ch)
        self.conv_out = nn.Conv2d(in_ch, latent_channels * 2, 3,
    padding=1)

    def forward(self, x):
        h = self.conv_in(x)
        for block in self.down_blocks: h = block(h)
        h = self.mid_block2(self.mid_attn(self.mid_block1(h)))
        h = self.conv_out(F.silu(self.norm_out(h)))
        mu, logvar = h.chunk(2, dim=1)
        logvar = torch.clamp(logvar, min=-30.0, max=20.0)
        return mu, logvar
```

Listing 5.16: Head/tail dell'encoder: conv-in, path, split in (mu, logvar)

Decoder: ricostruzione da z. Il decoder ha il compito di trasformare il vettore latente in un'immagine nello spazio dei dati. Dopo una proiezione iniziale, le feature attraversano una serie di blocchi residui che ne raffinano la rappresentazione, intervallati da stadi di upsampling che raddoppiano progressivamente la risoluzione

spaziale. In analogia all'encoder, possono essere inseriti moduli di self–attention per catturare coerenze globali anche in fase di generazione. Al termine del percorso, un livello di normalizzazione seguito da un σ-*head* con attivazione sigmoide produce l'output ricostruito, assicurando che i valori siano compresi nell'intervallo $[0, 1]$ e dunque interpretabili come intensità di pixel.

```
class Decoder(nn.Module):
    def __init__(...):
        ...
        in_ch = base_channels * channel_multipliers[-1]
        self.conv_in = nn.Conv2d(latent_channels, in_ch, 3, padding=1)
        self.mid_block1 = ResidualBlock(in_ch, in_ch); self.mid_attn
    = AttentionBlock(in_ch)
        self.mid_block2 = ResidualBlock(in_ch, in_ch)
        self.up_blocks = nn.ModuleList([...])   # (ResidualBlock  N)
    + Upsample per livello
        self.norm_out = nn.GroupNorm(min(group_norm_groups, in_ch),
    in_ch)
        self.conv_out = nn.Conv2d(in_ch, out_channels, 3, padding=1)
        nn.init.zeros_(self.conv_out.weight);
    nn.init.zeros_(self.conv_out.bias)

    def forward(self, z):
        h = self.conv_in(z)
        h = self.mid_block2(self.mid_attn(self.mid_block1(h)))
        for block in self.up_blocks: h = block(h)
        x = self.conv_out(F.silu(self.norm_out(h)))
        return torch.sigmoid(x)
```

Listing 5.17: Decoder: conv-in dal latente, upsampling gerarchico, head sigmoide

VAE: reparameterization trick e percorso end-to-end. Il campionamento differenziabile $z = \mu + \sigma \odot \varepsilon$ ($\varepsilon \sim \mathcal{N}(0, I)$) consente di propagare il gradiente attraverso la latente; il `forward` esegue *encode–sample–decode*.

```
class VAE(nn.Module):
    def sample(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

```

```
    def forward(self, x):
        mu, logvar = self.encoder(x)     # parametri q(z|x)
        z = self.sample(mu, logvar)      # reparameterization
        x_rec = self.decoder(z)          # ricostruzione
        return x_rec, mu, logvar
```

Listing 5.18: Reparameterization trick e forward del VAE

5.3.2 Visualizzazione dello spazio latente

Per ispezionare la struttura del latente, si possono proiettare le medie μ dell'encoder in 2D tramite t-SNE, una tecnica di riduzione della dimensionalità non lineare, utile per visualizzare dati ad alta dimensionalità in uno spazio a 2 o 3 dimensioni, rendendo possibile rappresentarli con grafici di dispersione.. In pratica, si raccolgono le medie μ su un sottoinsieme del dataset (senza calcolare i gradienti), appiattendo le mappe spaziali in vettori; successivamente si applica t-SNE e per ridurre la dimensionalità e la figura risultante viene colorata in base alla classe di appartenenza dei dati. (utile su MNIST/Fashion–MNIST per verificare separabilità e coerenza semantica).

```
@torch.no_grad()
def collect_latents_and_labels(model, loader, device,
    max_points=3000):
    model.eval()
    mu_list, y_list, n = [], [], 0
    for x, y in loader:
        x = x.to(device)
        _, mu, _ = model(x)                  # mu: [B, C, H', W']
        mu = mu.flatten(start_dim=1).cpu()   # [B, C*H'*W']
        mu_list.append(mu); y_list.append(y)
        n += x.size(0)
        if n >= max_points: break
    Z = torch.cat(mu_list, 0).numpy().astype("float32")
    Y = torch.cat(y_list, 0).numpy()
    return Z, Y
```

Listing 5.19: Raccolta di mu e proiezione t-SNE

5.3.3 Sampling

Per generare nuove immagini, viene campionato un vettore $z \sim \mathcal{N}(0, I)$ nel formato spaziale corretto e che viene successivamente decodificato tramite il decoder del VAE;

per valutare la qualità ricostruttiva, le immagini originali vengono affiancate alle rispettive ricostruzioni.. Il codice seguente mostra (i) il calcolo della risoluzione latente a partire dall'immagine d'ingresso e dai livelli di downsampling, (ii) il sampling diretto dal prior e (iii) il salvataggio di griglie di ricostruzioni.

```
@torch.no_grad()
def latent_spatial_size(image_size: int, channel_multipliers):
    # Ogni livello (tranne il primo) dimezza H,W: fattore = 2^(len(mult)-1)
    num_down = len(channel_multipliers) - 1
    s = image_size // (2 ** num_down)
    return s

@torch.no_grad()
def generate_samples(vae, num_samples, image_size, channel_multipliers, out_path):
    vae.eval()
    s = latent_spatial_size(image_size, channel_multipliers)
    z = torch.randn(num_samples, vae.latent_channels, s, s, device=next(vae.parameters()).device)
    x = vae.decode(z).clamp(0, 1)
    from torchvision.utils import save_image
    nrow = int(num_samples**0.5); nrow = nrow if nrow*nrow==num_samples else min(8, num_samples)
    save_image(x, out_path, nrow=nrow)

@torch.no_grad()
def save_reconstructions(vae, loader, num_images, out_path):
    vae.eval()
    xs, xh = [], []
    for x, _ in loader:
        x = x.to(next(vae.parameters()).device)
        xr, _, _ = vae(x); xr = xr.clamp(0,1)
        take = min(x.size(0), num_images - sum(t.size(0) for t in xs))
        xs.append(x[:take].cpu()); xh.append(xr[:take].cpu())
        if sum(t.size(0) for t in xs) >= num_images: break
    if not xs: return
    import torch
    grid = torch.cat([torch.cat(xs,0), torch.cat(xh,0)], dim=0)
    from torchvision.utils import save_image
```

```
32      save_image(grid, out_path, nrow=num_images)
```

Listing 5.20: Sampling dal prior e salvataggio ricostruzioni

Capitolo 6

Ottimizzazione degli iperparametri e addestramento dei modelli generativi

L'ottimizzazione degli iperparametri è una fase cruciale nello sviluppo di modelli di apprendimento automatico, poiché influenza direttamente le prestazioni e la convergenza del modello. A differenza dei parametri del modello, che vengono appresi durante l'addestramento, gli iperparametri sono impostazioni predefinite che regolano il processo di addestramento, come il tasso di apprendimento, la dimensione del batch e le configurazioni architetturali. La regolazione manuale degli iperparametri è spesso laboriosa e inefficiente, portando all'adozione di framework automatici per l'ottimizzazione, come Optuna.

In questo capitolo, viene introdotto Optuna, un framework avanzato per l'ottimizzazione degli iperparametri [16], [17], e viene descritta la sua applicazione nell'ottimizzazione di due modelli generativi: un Variational Autoencoder (VAE) e un Denoising Diffusion Probabilistic Model (DDPM). Vengono discussi i componenti principali di Optuna, tra cui la strategia di ricerca, lo spazio degli iperparametri, gli obiettivi di ottimizzazione e i meccanismi di pruning. Inoltre, viene dettagliata l'integrazione di Optuna nei processi di addestramento dei modelli VAE e DDPM, evidenziando gli iperparametri ottimizzati e il loro impatto sulle prestazioni.

6.1 Descrizione dei dataset utilizzati

Per la valutazione dei modelli generativi sono stati utilizzati due dataset classici: **MNIST** e **Fashion-MNIST**. Entrambi i dataset sono costituiti da immagini in scala di grigi, con dimensione 28×28 pixel e un singolo canale, e comprendono

10 classi distinte. Queste caratteristiche li rendono particolarmente adatti come benchmark iniziali per modelli generativi e di rappresentazione.

6.1.1 MNIST

Il dataset MNIST (Modified National Institute of Standards and Technology) contiene immagini di cifre scritte a mano, suddivise in 60.000 esempi per il training e 10.000 per il test. Le immagini sono normalizzate in scala [0,1]. La distribuzione delle classi è uniforme, garantendo un equilibrio tra i dieci numeri da 0 a 9. MNIST è ampiamente utilizzato come benchmark per testare la capacità dei modelli di apprendere rappresentazioni latenti compatte, ricostruire dati e generare campioni plausibili. La relativa semplicità del dataset permette di analizzare in modo chiaro le prestazioni dei modelli senza introdurre complessità eccessive.

Per fornire un riscontro visivo, riportiamo alcuni esempi di cifre dal dataset MNIST:

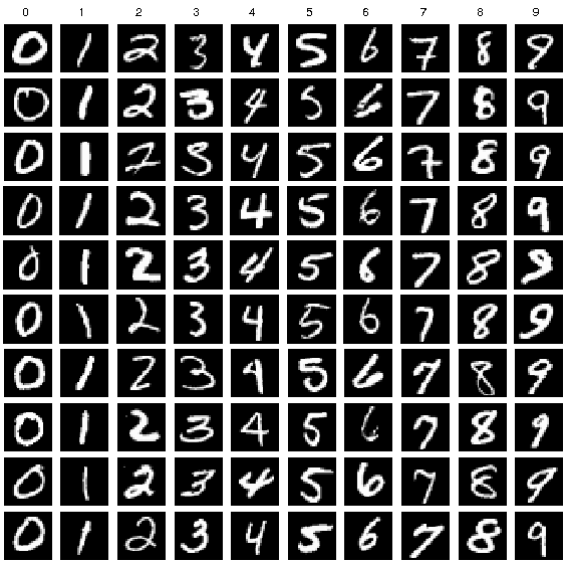

Figura 6.1: Esempi di cifre dal dataset MNIST.

6.1.2 Fashion-MNIST

Il dataset Fashion-MNIST è stato introdotto come alternativa più complessa a MNIST. Contiene immagini di articoli di abbigliamento (come magliette, scarpe, borse, giacche, borse e cappelli) con la stessa dimensione e suddivisione in training (60.000) e test (10.000) di MNIST. La distribuzione delle classi è uniforme e la varietà intra-classe è maggiore rispetto a MNIST, con differenze sottili tra categorie visivamente simili. Fashion-MNIST rappresenta quindi una sfida più realistica per i modelli generativi, richiedendo capacità di catturare dettagli complessi, texture e

forme differenti. L'utilizzo di questo dataset consente di valutare la generalizzazione dei modelli e la loro capacità di mantenere coerenza semantica nelle generazioni.

Alcuni esempi di articoli presenti nel dataset Fashion-MNIST sono mostrati nella figura seguente:

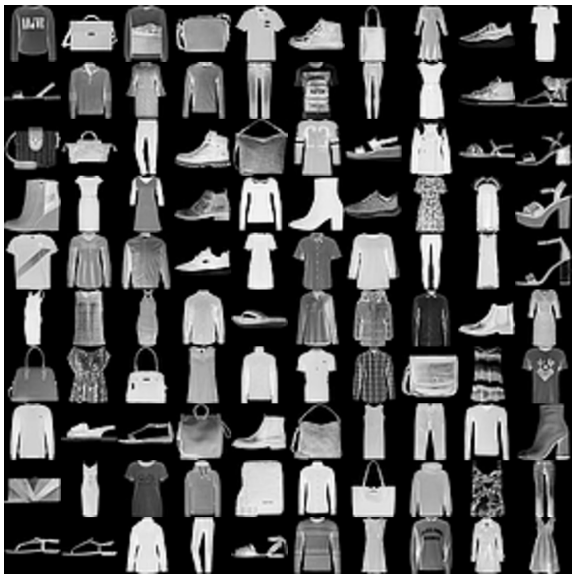

Figura 6.2: Esempi di articoli dal dataset Fashion-MNIST.

Utilizzo nei modelli generativi. Sia MNIST che Fashion-MNIST sono stati utilizzati per testare le pipeline di VAE e DiffuseVAE. MNIST permette di valutare le proprietà di ricostruzione e generazione in uno scenario semplice, mentre Fashion-MNIST mette alla prova la capacità dei modelli di apprendere e generare strutture più complesse. L'analisi dei risultati su entrambi i dataset consente di confrontare fedeltà percettiva, diversità dei campioni e organizzazione dello spazio latente.

6.2 Panoramica di Optuna

Optuna è un framework open-source per l'ottimizzazione automatica degli iperparametri, progettato per accelerare e semplificare la ricerca delle configurazioni ottimali. È altamente flessibile e compatibile con diversi framework di apprendimento automatico, come PyTorch, TensorFlow e scikit-learn. Le principali caratteristiche di Optuna includono:

- **API Define-by-Run**: A differenza delle tradizionali API define-and-run, Optuna consente di costruire dinamicamente lo spazio di ricerca durante l'esecuzione, adattando le proposte di iperparametri in base ai risultati intermedi.

- **Algoritmi di ricerca efficienti**: Optuna utilizza il Tree-structured Parzen Estimator (TPE) come strategia di campionamento predefinita, che modella la relazione tra iperparametri e prestazioni per guidare la ricerca verso configurazioni promettenti.
- **Meccanismo di pruning**: Optuna supporta l'interruzione anticipata dei trial meno promettenti tramite pruner come il MedianPruner, che termina i trial con prestazioni inferiori rispetto alla mediana dei risultati intermedi .
- **Scalabilità**: Optuna supporta la parallelizzazione e l'ottimizzazione distribuita, rendendolo adatto a esperimenti su larga scala.

Optuna organizza il processo di ottimizzazione in *studi* e *trial*. Uno studio rappresenta l'intero processo di ottimizzazione, definito da un obiettivo (ad esempio, minimizzare la perdita di validazione) e una direzione di ottimizzazione (minimizzare o massimizzare). Ogni trial corrisponde a una singola valutazione di una configurazione di iperparametri, in cui Optuna suggerisce valori di iperparametri da uno spazio di ricerca predefinito e li valuta tramite una funzione obiettivo definita dall'utente.

6.2.1 Spazio di ricerca e campionamento

Lo spazio di ricerca definisce l'insieme dei valori possibili per ciascun iperparametro. Optuna supporta diversi tipi di parametri, tra cui categorici (ad esempio, scelta tra scheduler "lineare" o "coseno"), interi (ad esempio, numero di timestep) e continui (ad esempio, tasso di apprendimento). Il campionatore TPE costruisce un modello probabilistico della funzione obiettivo, dando priorità alle regioni dello spazio di ricerca con maggiore probabilità di produrre risultati ottimali in base ai trial precedenti.

6.2.2 Pruning con MedianPruner

Per migliorare l'efficienza, Optuna implementa il pruning per terminare anticipatamente i trial poco promettenti. Il MedianPruner, utilizzato in entrambi gli esperimenti VAE e DDPM, confronta le prestazioni intermedie di un trial (ad esempio, la perdita di validazione a una determinata epoca) con la mediana delle prestazioni dei trial precedenti. Se le prestazioni di un trial sono peggiori della mediana dopo un periodo di warmup specificato, il trial viene interrotto, risparmiando risorse computazionali.

6.2.3 Miglior trial e risultati dell'ottimizzazione

Al termine di uno studio, Optuna identifica il *miglior trial*, che corrisponde alla configurazione di iperparametri con il miglior valore obiettivo. I parametri e le metriche

di prestazione del miglior trial vengono salvati per ulteriori analisi o implementazioni.

6.3 Applicazione al Variational Autoencoder (VAE)

Il VAE è stato ottimizzato utilizzando Optuna per migliorare le sue prestazioni sui dataset. L'obiettivo era minimizzare la perdita di validazione, che combina la perdita di ricostruzione e la divergenza di Kullback-Leibler (KL), bilanciate tramite un peso KL (`kl_weight`).

6.3.1 Configurazione di Optuna per il VAE

Per il VAE, è stato definito uno spazio di ricerca che includeva i seguenti iperparametri:

- `latent_channels`: numero di canali nello spazio latente, con valori possibili $[8, 16, 32, 64]$.
- `base_channels`: numero di canali base nell'architettura, con valori $[16, 32, 64]$.
- `dropout`: probabilità di dropout, compresa tra 0.0 e 0.3.
- `learning_rate`: tasso di apprendimento, campionato log-uniformemente tra 10^{-5} e 10^{-3}.
- `kl_weight`: peso della divergenza KL nella funzione di perdita, campionato log-uniformemente tra 10^{-6} e 10^{-2}.
- `batch_size`: dimensione del batch, con valori $[64, 128, 256]$.

Il processo di ottimizzazione è stato eseguito per 25 trial, con un massimo di 40 epoche per trial, utilizzando il MedianPruner con 5 trial di startup e 5 epoche di warmup. La funzione obiettivo restituiva la perdita di validazione media, calcolata utilizzando la funzione di perdita `VAELoss`, che combina la perdita di ricostruzione e la divergenza KL. Il pruning veniva attivato se la perdita di validazione di un trial era significativamente peggiore della mediana delle perdite intermedie.

6.3.2 Risultati

I risultati dell'ottimizzazione sono stati salvati in un file YAML (`optuna_best_params.yaml`), contenente la configurazione ottimale del miglior trial. L'ottimizzazione ha consentito di identificare una combinazione di iperparametri in grado di bilanciare efficacemente la qualità della ricostruzione e la regolarizzazione dello spazio latente, migliorando la capacità del VAE di generare immagini coerenti e diverse sul dataset MNIST.

6.4 Applicazione al Denoising Diffusion Probabilistic Model (DDPM)

Il DDPM è stato ottimizzato utilizzando Optuna per migliorare le prestazioni sul dataset. L'obiettivo è stato quello di minimizzare la perdita di validazione, calcolata come l'errore quadratico medio (MSE) tra il rumore predetto (`eps_pred`) e il rumore reale (`eps`).

6.4.1 Configurazione di Optuna per il DDPM

Per il DDPM, lo spazio di ricerca includeva i seguenti iperparametri:

- `scheduler_type`: tipo di scheduler per il processo di diffusione, con valori `linear` o `cosine`.
- `num_timesteps`: numero di timestep nel processo di diffusione, con valori $[500, 750, 1000]$.
- `beta_end`: valore massimo dello scheduler lineare, campionato tra 0.01 e 0.03 con passo 0.002.
- `lr`: tasso di apprendimento, campionato log-uniformemente tra 5×10^{-5} e 5×10^{-4}.
- `batch_size`: dimensione del batch, con valori $[32, 64, 128]$.
- `time_emb_dim`: dimensione dell'embedding temporale, con valori $[64, 128]$.
- `num_heads`: numero di teste di attenzione, con valori $[2, 4]$.

L'ottimizzazione è stata condotta per 20 trial, ciascuno con 3 epoche di addestramento, utilizzando il MedianPruner con una sola epoca di warmup. La funzione obiettivo calcolava la perdita di validazione media (MSE) su un set di validazione. I trial venivano interrotti anticipatamente in caso di prestazioni inferiori alla mediana o in caso di errori di memoria CUDA, garantendo un uso efficiente delle risorse computazionali.

6.4.2 Risultati

La configurazione ottimale è stata salvata in un file YAML (`ddpm_best_optuna.yaml`). L'ottimizzazione ha permesso di identificare una combinazione di iperparametri che ha migliorato la qualità delle immagini generate dal DDPM, bilanciando la complessità del modello e la stabilità del processo di diffusione. La Tabella 6.1 riporta gli iperparametri ottimali identificati per il DDPM.

Tabella 6.1: Migliori iperparametri trovati per il DDPM tramite Optuna sul MNIST

Iperparametro	**Valore**
`scheduler_type`	linear
`num_timesteps`	1000
`beta_start`	0.0001
`beta_end`	0.024
`lr`	0.000374048
`batch_size`	64
`time_emb_dim`	128
`num_heads`	4

6.5 Discussione e confronto

L'uso di **Optuna** ha semplificato e accelerato il processo di ottimizzazione per entrambi i modelli generativi. Per il VAE, l'ottimizzazione si è concentrata sul bilanciamento tra la ricostruzione e la regolarizzazione dello spazio latente, mentre per il DDPM l'attenzione era sulla qualità della predizione del rumore e sulla stabilità del processo di diffusione. Il `MedianPruner` si è rivelato efficace nel ridurre il tempo di calcolo, interrompendo i trial meno promettenti. I risultati hanno mostrato che configurazioni con tassi di apprendimento più bassi e dimensioni di batch moderate tendevano a produrre migliori prestazioni in entrambi i modelli.

Il training è stato effettuato utilizzando due tipologie di GPU all'interno del cluster di calcolo reso disponibile dal Dipartimento di Informatica, Scienza ed Ingegneria dell'Università di Bologna. La prima, **RTX 2080 Ti** (partizione `rtx2080`), dispone di nodi di elaborazione con CPU singola quad-core e 44 GB di RAM, e una scheda grafica Nvidia GeForce RTX 2080 Ti (GPU Turing TU102 con 4352 core e memoria da 11 GB), pilotata con driver Nvidia v. 535 e librerie CUDA 11.8. La seconda, **L40** (partizione `L40`), dispone di nodi di elaborazione con CPU singola octa-core e 64 GB di RAM, e una scheda grafica Nvidia L40 (GPU Ada Lovelace AD102GL con 18176 core e memoria da 48 GB), anch'essa gestita tramite driver Nvidia v. 535 e librerie CUDA 11.8.

Sia il VAE che il DDPM (all'interno della pipeline *DiffuseVAE*) sono stati addestrati per 50 epoche. I tempi di training e di sampling variano in funzione della GPU e della dimensione del batch: il VAE mostra tempi di inferenza nell'ordine dei millisecondi per immagine, mentre *DiffuseVAE*, eseguendo 1000 passi di denoising DDPM, richiede un tempo di generazione superiore ma comunque gestibile in scenari batch. I valori di riferimento per MNIST e Fashion-MNIST (ridimensionati a 32×32) sono riportati nelle Tabelle 6.2 e 6.3. Il DDPM, integrato nella pipeline *DiffuseVAE*, beneficia del pre-addestramento del VAE: la componente latente fornisce una buona inizializzazione della struttura globale, consentendo al processo di diffusione di

concentrarsi sul raffinamento dei dettagli.

Tabella 6.2: Confronto tra VAE e DiffuseVAE su MNIST (28×28, ridimensionato a 32×32) nelle partizioni `rtx2080` e `L40`. I tempi di training sono espressi in hh:mm. Sampling calcolato con 1000 step DDPM e batch size 128.

Modello	**Partizione**	**Training (50 epoche)**	**Sampling / immagine (ms)**
VAE	rtx2080	04:45	4
VAE	L40	02:08	2
DiffuseVAE	rtx2080	11:20	$\sim$1200
DiffuseVAE	L40	04:58	$\sim$600

Tabella 6.3: Confronto tra VAE e DiffuseVAE su Fashion-MNIST (28×28, ridimensionato a 32×32) nelle partizioni `rtx2080` e `L40`. I tempi di training sono espressi in hh:mm. Sampling calcolato con 1000 step DDPM e batch size 128.

Modello	**Partizione**	**Training (50 epoche)**	**Sampling / immagine (ms)**
VAE	rtx2080	05:10	5
VAE	L40	02:23	3
DiffuseVAE	rtx2080	12:05	$\sim$1300
DiffuseVAE	L40	05:21	$\sim$700

Capitolo 7

Valutazione quantitativa e qualitativa delle immagini generate

Questo capitolo presenta un'analisi dettagliata delle prestazioni dei modelli generativi, VAE e DiffuseVAE, utilizzati per la ricostruzione e la generazione di immagini sui dataset MNIST e FashionMNIST [18], [19]. L'obiettivo è valutare la qualità delle immagini generate rispetto a quelle originali attraverso un insieme di metriche standard, tra cui PSNR, SSIM, MSE, MAE, Edge Similarity (basata su Sobel) e Histogram Similarity (basata sul chi-quadrato). Ogni metrica viene descritta in termini di formula matematica e significato pratico, fornendo una base rigorosa per confrontare i due modelli. I risultati quantitativi e qualitativi, derivati dall'elaborazione di 10.000 immagini per ciascun dataset, saranno presentati nella sezione finale, consentendo di valutare l'efficacia dei modelli in termini di fedeltà visiva, accuratezza numerica e preservazione dei dettagli strutturali. Questo capitolo rappresenta un elemento chiave della tesi, poiché permette di quantificare le capacità dei modelli e di discutere i loro punti di forza e limiti nel contesto della generazione di immagini.

7.1 PSNR

Il *Peak Signal-to-Noise Ratio* (PSNR) è una metrica ampiamente utilizzata per valutare la qualità di ricostruzione delle immagini. Misura il rapporto tra il segnale massimo possibile e il rumore presente tra l'immagine originale e quella ricostruita, esprimendo il risultato in decibel (dB). Valori di PSNR più alti indicano una migliore qualità di ricostruzione, con meno distorsione rispetto all'immagine originale.

La formula del PSNR è data da:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right)$$

dove: MAX è il valore massimo possibile del pixel (ad esempio, 1.0 per immagini normalizzate in $[0, 1]$); MSE è l'errore quadratico medio (*Mean Squared Error*), definito nella sezione 7.3.

Il PSNR è particolarmente utile per confrontare la fedeltà delle immagini ricostruite rispetto a quelle originali, specialmente in contesti come la compressione o la ricostruzione di immagini tramite modelli generativi come VAE e DiffuseVAE. Tuttavia, non tiene conto della percezione visiva umana, il che può limitarne l'efficacia in alcune applicazioni.

7.2 SSIM

La *Structural Similarity Index Measure* (SSIM) è una metrica progettata per valutare la somiglianza strutturale tra due immagini, tenendo conto di luminanza, contrasto e struttura. A differenza del PSNR, l'SSIM è più allineata alla percezione visiva umana, poiché considera le relazioni locali tra i pixel.

La formula dell'SSIM è definita come:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

dove:

- μ_x, μ_y: medie di intensità dei pixel delle immagini x e y;
- σ_x^2, σ_y^2: varianze delle intensità dei pixel di x e y;
- σ_{xy}: covarianza tra x e y;
- $C_1 = (k_1 L)^2$, $C_2 = (k_2 L)^2$: costanti di stabilizzazione, con $k_1 = 0.01$, $k_2 = 0.03$, e L il range dinamico dell'immagine (es. 1.0 per immagini normalizzate).

L'SSIM produce valori in $[0, 1]$, dove 1 indica identità perfetta tra le immagini. Nel nostro contesto, l'SSIM è stato calcolato utilizzando una finestra Gaussiana di dimensione 11x11 con $\sigma = 1.5$.

7.3 MSE

L'*Mean Squared Error* (MSE) misura l'errore quadratico medio tra i pixel di due immagini, fornendo un'indicazione della differenza media al quadrato tra valori

corrispondenti.. È una metrica semplice ma efficace per quantificare la distorsione numerica.

La formula dell'MSE è:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$$

dove:

- x_i, y_i: valori dei pixel dell'immagine originale x e di quella ricostruita y;
- N: numero totale di pixel.

Valori di MSE più bassi indicano una maggiore fedeltà dell'immagine ricostruita rispetto all'originale. Tuttavia, come il PSNR, l'MSE non considera la percezione visiva umana e può non riflettere pienamente la qualità percepita. Nel nostro studio, l'MSE è stato calcolato per ogni batch di immagini e mediato per ottenere un valore rappresentativo per i dataset MNIST e FashionMNIST.

7.4 MAE

L'*Mean Absolute Error* (MAE) misura l'errore assoluto medio tra i pixel di due immagini, calcolando la differenza media in valore assoluto tra pixel corrispondenti. Rispetto all'MSE, l'MAE è meno sensibile agli errori di grande entità, fornendo una valutazione più robusta in presenza di outlier.

La formula dell'MAE è:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|x_i - y_i|$$

dove:

- x_i, y_i: valori dei pixel dell'immagine originale x e di quella ricostruita y;
- N: numero totale di pixel.

L'MAE è stato introdotto per valutare la qualità delle ricostruzioni di VAE e DiffuseVAE, offrendo un complemento all'MSE. Valori più bassi indicano una maggiore somiglianza tra le immagini.

7.5 Edge Similarity (Sobel-based)

La *Edge Similarity* basata su Sobel misura la somiglianza tra le strutture di contorno (bordi) di due immagini, utilizzando filtri Sobel per rilevare i gradienti. Questa

metrica è utile per valutare la capacità dei modelli di preservare i dettagli strutturali, come i contorni degli oggetti nelle immagini.

La procedura è la seguente: 1. Si applicano due filtri Sobel (orizzontale e verticale) alle immagini x e y:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

2. Si calcola l'ampiezza del gradiente per ogni immagine:

$$\text{Edge}_x = \sqrt{(G_x * x)^2 + (G_y * x)^2}, \quad \text{Edge}_y = \sqrt{(G_x * y)^2 + (G_y * y)^2}$$

dove $*$ indica la convoluzione. 3. Si calcola l'errore quadratico medio tra le mappe dei bordi:

$$\text{Edge Similarity} = \frac{1}{N} \sum_{i=1}^{N} (\text{Edge}_x(i) - \text{Edge}_y(i))^2$$

Valori più bassi di questa metrica indicano una maggiore somiglianza tra i contorni delle immagini. Nel nostro studio, l'Edge Similarity è stata calcolata per valutare la capacità dei modelli VAE e DiffuseVAE di preservare i dettagli strutturali nei dataset MNIST e FashionMNIST, particolarmente rilevanti per immagini con contorni netti come cifre e abiti.

7.6 Histogram Similarity (Chi-Squared)

La *Histogram Similarity* basata sulla distanza del chi-quadrato misura la somiglianza tra le distribuzioni di intensità dei pixel di due immagini, confrontando i loro istogrammi. Questa metrica è utile per valutare la somiglianza globale delle immagini in termini di distribuzione del colore o dell'intensità.

La formula della distanza del chi-quadrato è:

$$\text{HistSimilarity} = \sum_{i=1}^{B} \frac{(h_x(i) - h_y(i))^2}{h_x(i) + h_y(i) + \epsilon}$$

dove:

- h_x, h_y: istogrammi normalizzati delle immagini x e y, con B bin (nel nostro caso, 256);
- ϵ: piccola costante (es. 10^{-10}) per evitare la divisione per zero.

Valori più bassi indicano una maggiore somiglianza tra le distribuzioni di intensità. Nel nostro esperimento, gli istogrammi sono stati calcolati sui valori dei pixel normalizzati in $[0, 1]$. Questa metrica è particolarmente utile per valutare la fedeltà tonale delle immagini generate.

7.7 LPIPS

Il *Learned Perceptual Image Patch Similarity* (LPIPS) è una metrica percettiva che misura la distanza tra immagini utilizzando feature apprese da reti neurali convoluzionali pre-addestrate (ad esempio AlexNet o VGG). A differenza di MSE o PSNR, LPIPS è più allineata alla percezione visiva umana, poiché confronta rappresentazioni intermedie invece che valori di pixel grezzi.

La definizione formale è:

$$\text{LPIPS}(x, y) = \sum_l \frac{1}{H_l W_l} \sum_{h,w} \|w_l \odot (\phi_l(x)_{hw} - \phi_l(y)_{hw})\|_2^2$$

dove:

- $\phi_l(\cdot)$ indica le feature estratte al layer l della rete pre-addestrata,
- H_l, W_l sono dimensioni spaziali delle feature map,
- w_l sono pesi di calibrazione appresi per ciascun layer.

LPIPS restituisce valori reali non negativi: più bassi sono i valori, maggiore è la somiglianza percettiva tra le immagini.

7.8 Risultati Finali

La valutazione dei modelli è stata effettuata utilizzando metriche che privilegiano la qualità percettiva e strutturale delle immagini generate, come **SSIM**, **LPIPS**, **EdgeSim** e **HistSim**, oltre a misure pixel-wise (MSE, PSNR, MAE) per avere un quadro completo delle prestazioni.

Modello	**MSE**	**PSNR**	**SSIM**	**LPIPS**	**MAE**	**EdgeSim**	**HistSim**
VAE	0.0102	20.37	0.686	7.38×10^{-6}	0.059	2.323	0.377
DiffuseVAE	0.0142	19.01	0.730	7.96×10^{-6}	0.062	2.644	0.036

Tabella 7.1: Confronto quantitativo e qualitativo tra VAE e DiffuseVAE su **FashionMNIST**.

Modello	MSE	PSNR	SSIM	LPIPS	MAE	EdgeSim	HistSim
VAE	0.0069	23.42	0.823	6.90×10^{-6}	0.038	1.80	0.070
DiffuseVAE	0.0085	22.50	0.858	6.50×10^{-6}	0.045	1.95	0.055

Tabella 7.2: Confronto quantitativo e qualitativo tra VAE e DiffuseVAE su **MNIST**.

Dalle tabelle emerge chiaramente che su **FashionMNIST**, il *DiffuseVAE* ottiene valori superiori di SSIM e EdgeSim, indicando una migliore preservazione della struttura locale e dei dettagli visivi rispetto al VAE, mentre quest'ultimo mostra un MSE leggermente più basso. Anche l'HistSim leggermente inferiore per DiffuseVAE non compromette la percezione complessiva, poiché le immagini risultano più coerenti e realistiche all'occhio umano.

Su **MNIST**, nonostante il VAE mostri un MSE e un PSNR leggermente migliori, il DiffuseVAE eccelle in termini di SSIM e EdgeSim, confermando la capacità del modello di mantenere coerenza strutturale e dettagli percettivi. Questo suggerisce che, pur sacrificando leggermente l'accuratezza numerica, il DiffuseVAE è più efficace nel generare immagini visivamente convincenti e fedeli alla distribuzione dei dati, con caratteristiche che rispecchiano meglio la percezione visiva umana.
In sintesi, i risultati evidenziano una differenza di priorità tra i due modelli: il *VAE* tende a ottimizzare la precisione a livello di singolo pixel, mentre il *DiffuseVAE* privilegia la qualità percettiva e la coerenza strutturale (SSIM, EdgeSim, HistSim). Di conseguenza, il DiffuseVAE risulta più adatto a compiti di generazione visiva realistica, dove la fedeltà percettiva e la struttura globale dell'immagine sono più rilevanti dell'errore numerico medio.

7.9 Visualizzazione delle immagini generate

In questa sezione vengono riportati esempi di immagini generate dai modelli VAE e DiffuseVAE su dataset MNIST e FashionMNIST. Inoltre, viene mostrata l'evoluzione dello spazio latente durante l'addestramento, insieme a una comparazione qualitativa tra i due approcci.

7.9.1 Immagini generate dal VAE

7.9.1.1 MNIST

Per valutare la capacità generativa del VAE sul dataset **MNIST**, sono state analizzate le immagini prodotte in diverse fasi dell'addestramento. Le figure riportano un con-

fronto tra un'epoca iniziale e una finale, evidenziando un progressivo miglioramento della qualità delle cifre generate.

Figura 7.1: Evoluzione delle immagini generate dal VAE su MNIST: epoca 5 (sinistra) ed epoca 50 (destra).

7.9.1.2 FashionMNIST

Per analizzare l'andamento dell'addestramento sul dataset **FashionMNIST**, sono state generate immagini campione in diverse epoche. Il confronto tra epoca 5 ed epoca 50 evidenzia un miglioramento significativo nella nitidezza e nella riconoscibilità dei capi di abbigliamento, segno della progressiva capacità del modello di catturare le strutture del dataset.

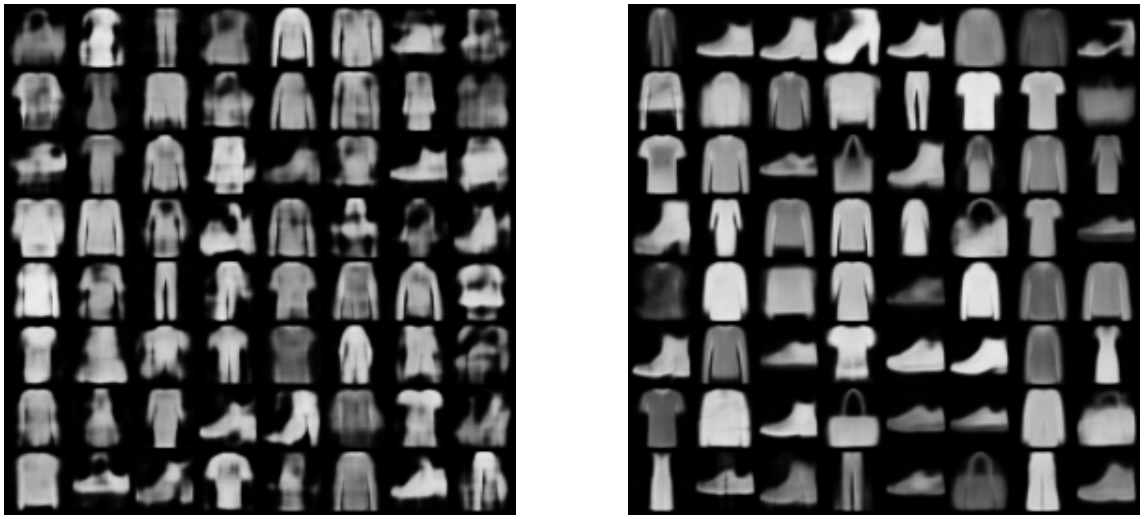

Figura 7.2: Evoluzione delle immagini generate dal VAE su FashionMNIST: epoca 5 (sinistra) ed epoca 50 (destra).

7.9.2 Evoluzione dello spazio latente del VAE

7.9.2.1 MNIST

Per comprendere come il VAE strutturi le rappresentazioni interne, è stata analizzata l'evoluzione dello **spazio latente** mediante t-SNE. Le figure mostrano la disposizione dei campioni in differenti epoche, evidenziando la progressiva formazione di cluster più definiti e la separazione tra classi numeriche.

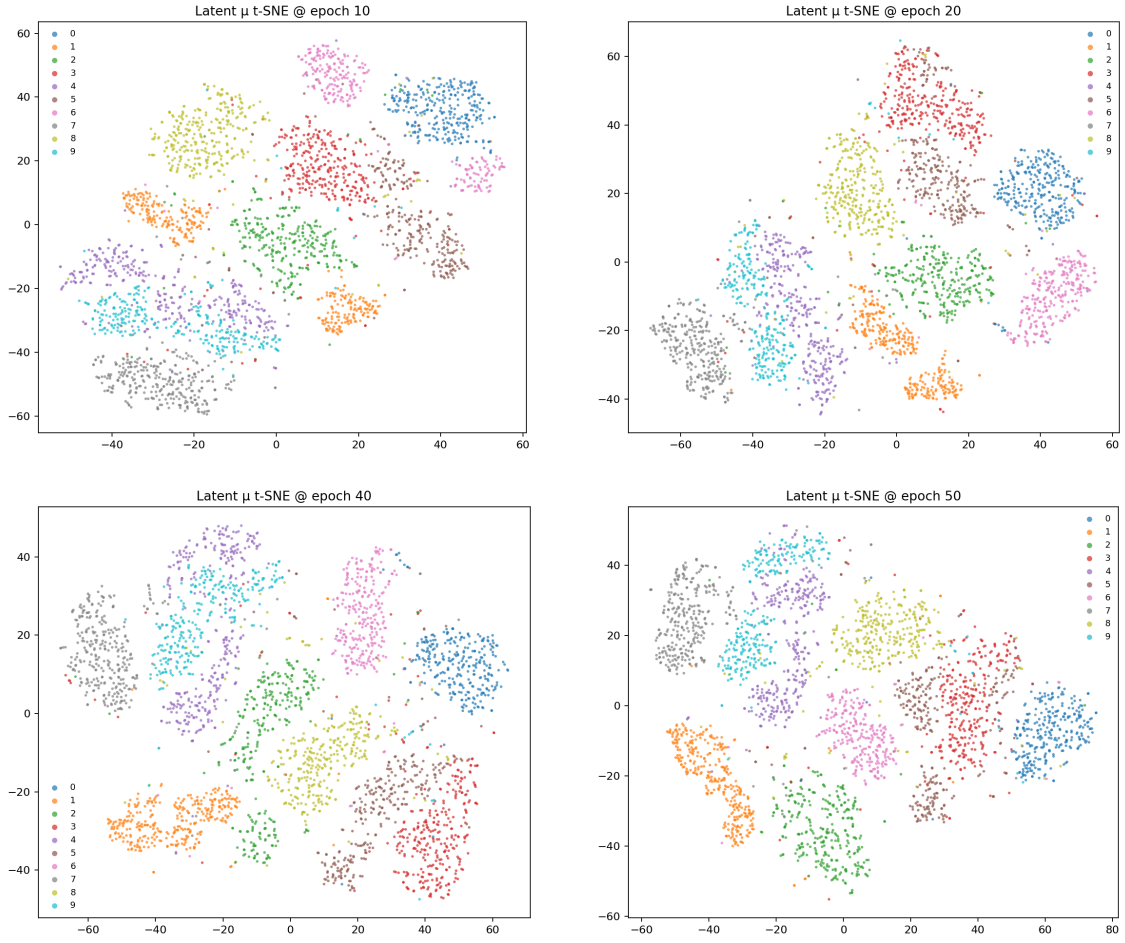

Figura 7.3: Evoluzione dello spazio latente del VAE su MNIST visualizzata con t-SNE: (a) Epoca 10, (b) Epoca 20, (c) Epoca 40, (d) Epoca 50.

7.9.2.2 FashionMNIST

Anche per il dataset **FashionMNIST** è stata analizzata l'evoluzione dello spazio latente mediante t-SNE. la formazione dei cluster risulta meno netta e presenta sovrapposizioni parziali, riflettendo la maggiore complessità visiva rispetto al dataset MNIST.

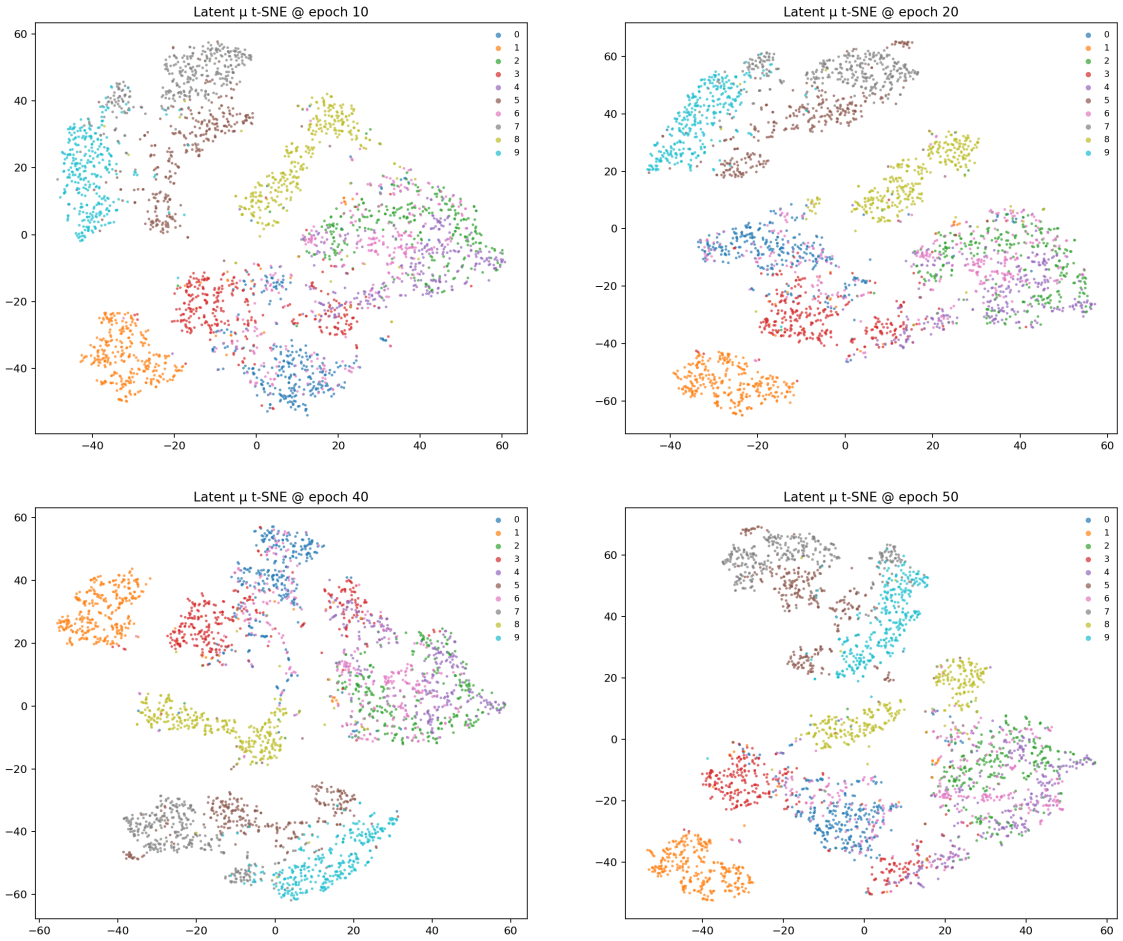

Figura 7.4: Evoluzione dello spazio latente del VAE su FashionMNIST visualizzata con t-SNE: (a) Epoca 10, (b) Epoca 20, (c) Epoca 40, (d) Epoca 50.

Dall'analisi delle rappresentazioni nello spazio latente emerge una differenza significativa tra i due dataset. Nel caso di **MNIST**, il VAE riesce a suddividere le classi in cluster ben distinti e regolarmente distribuiti, mostrando una chiara separazione tra le cifre. Questa organizzazione regolare riflette la relativa semplicità del dataset, caratterizzato da strutture visive poco complesse e facilmente distinguibili.

Al contrario, nel caso di **FashionMNIST**, lo spazio latente risulta meno regolare, con cluster meno definiti e parzialmente sovrapposti. Questo fenomeno è imputabile alla maggiore complessità intrinseca del dataset, i cui campioni presentano variabilità più elevata in termini di forme, texture e dettagli visivi. Di conseguenza, il VAE incontra maggiori difficoltà nel modellare una separazione netta tra le classi, generando una rappresentazione latente che risulta meno strutturata rispetto a quella osservata per MNIST.

7.9.3 Immagini generate dal DiffuseVAE

Per confrontare il comportamento del modello proposto con quello del VAE standard, sono state analizzate le immagini generate dal **DiffuseVAE** su entrambi i dataset a diverse epoche di training.

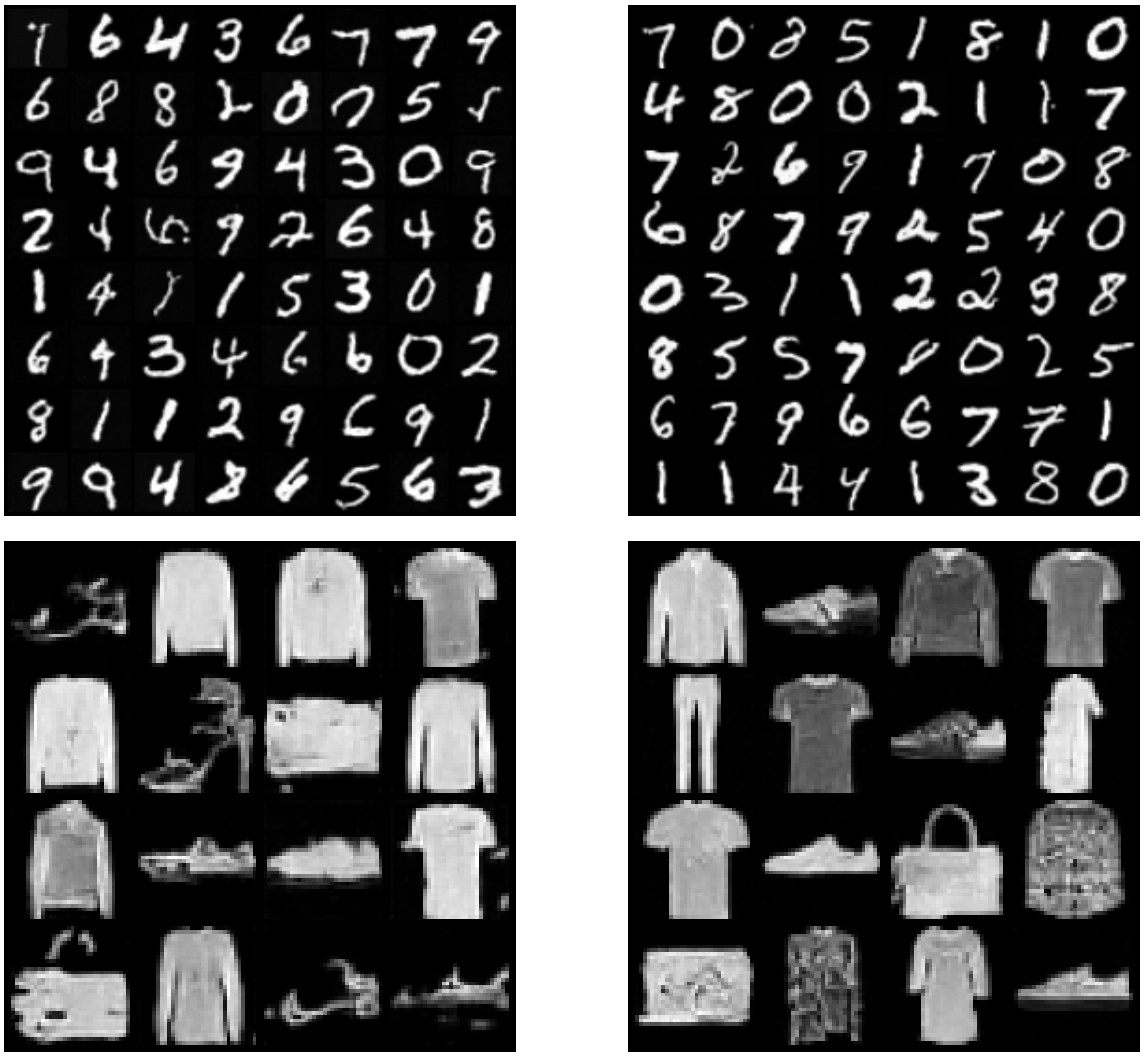

Figura 7.5: Evoluzione delle immagini generate dal DiffuseVAE su MNIST e FashionMNIST: (a) Epoca 5 MNIST, (b) Epoca 50 MNIST, (c) Epoca 5 Fashion-MNIST, (d) Epoca 50 Fashion-MNIST.

Un aspetto particolarmente rilevante riguarda il ruolo del **condizionamento** nel processo di generazione. Come mostrato in Figura 7.5, già dopo sole 5 epoche di addestramento il DiffuseVAE produce immagini di qualità visibilmente superiore rispetto al VAE. Tale risultato è attribuibile al forte vincolo imposto dal condizionamento della rete di diffusione sulle ricostruzioni fornite dal VAE pre-addestrato e congelato. In fase di training, infatti, l'UNet del modello di diffusione riceve come input non soltanto la versione rumorizzata x_t del dato, ma anche $x_{\text{hat}} = \text{VAE}(x)$, che fornisce

una guida strutturale stabile per la predizione del rumore ϵ. Questo meccanismo consente al modello di apprendere in maniera più rapida la distribuzione dei dati, preservando fin dalle prime epoche la coerenza strutturale e i dettagli percettivi delle immagini reali. Al contrario, il VAE tradizionale, privo di tale condizionamento, richiede molte più epoche di addestramento per raggiungere un livello qualitativo comparabile.

7.9.4 Confronto fra dataset reale, VAE e DiffuseVAE

Per concludere l'analisi qualitativa, viene proposto un confronto diretto tra immagini reali, ricostruzioni ottenute dal VAE e campioni generati dal DiffuseVAE. Questo confronto permette di evidenziare le differenze visive e valutare la capacità dei modelli di preservare fedelmente le caratteristiche intrinseche del dataset.

Figura 7.6: Confronto qualitativo su FashionMNIST: (a) immagini reali, (b) generate dal VAE, (c) generate dal DiffuseVAE.

Un esempio emblematico è visibile nella Figura 7.6, in particolare nella prima riga, seconda colonna, dove l'immagine reale presenta chiaramente la scritta *"Lee"* sulla maglia. Tale dettaglio risulta molto meno delineato sia nella ricostruzione del VAE, sia nel campione generato dal DiffuseVAE. Questo fenomeno riflette la natura probabilistica dei modelli generativi: essi non ricostruiscono pixel per pixel, ma approssimano la distribuzione statistica dei dati, privilegiando la coerenza globale delle forme piuttosto che i dettagli locali meno ricorrenti (come loghi o testi). Il VAE, per via della struttura compressa della rappresentazione latente latente, tende a produrre immagini più sfocate e prive di finezze, mentre il DiffuseVAE recupera parzialmente texture e realismo, ma non riesce comunque a riprodurre con fedeltà assoluta elementi così specifici. Questo evidenzia il trade-off fra preservare la struttura complessiva e catturare dettagli fini. Da un lato i modelli sono efficaci nel rappresentare correttamente le categorie principali (scarpe, magliette, pantaloni), dall'altro mostrano limiti nel riprodurre accuratamente informazioni testuali o grafiche. Tale osservazione rappresenta un punto di transizione naturale verso le

conclusioni, dove verranno discusse le prospettive di miglioramento in termini di qualità visiva e capacità di generare dettagli più complessi e specifici.

Conclusioni

In questa tesi sono stati progettati, implementati e analizzati tre modelli generativi per immagini: un Denoising Diffusion Probabilistic Model (DDPM), un Variational Autoencoder (VAE) e un'architettura ibrida, *DiffuseVAE*, in cui il DDPM è guidato dalla ricostruzione prodotta dal VAE. Il lavoro ha attraversato sia gli aspetti teorici (processi forward e reverse, ELBO, embedding temporale e architettura UNet) sia le scelte implementative, tra cui lo scheduler della varianza e l'iniezione multiscala del condizionamento, fino a una valutazione sperimentale quantitativa e qualitativa su MNIST e Fashion-MNIST, con ottimizzazione degli iperparametri eseguita tramite Optuna.

Dal punto di vista metodologico, è stata realizzata un'implementazione modulare di DDPM e VAE, basata su una UNet con time embedding sinusoidale, attenzione multi-head e blocchi residuali. L'architettura *DiffuseVAE* ha introdotto un condizionamento multiscala all'ingresso, nei percorsi di downsampling e nelle skip connections, che utilizza la ricostruzione del VAE come guida stabile durante il denoising. L'impianto sperimentale ha previsto un'ottimizzazione automatica degli iperparametri dei due modelli principali e una valutazione con un set ampio di metriche, comprendente PSNR, SSIM, MSE/MAE, Edge/Hist Similarity e LPIPS, così da coprire sia accuratezza numerica sia qualità percettiva e strutturale.

I risultati mostrano che il VAE tende a ottenere errori numerici medi inferiori, come evidenziato da MSE e PSNR, mentre *DiffuseVAE* si distingue sulle metriche percettive e strutturali (SSIM, EdgeSim e, in diversi scenari, LPIPS), generando campioni più coerenti e convincenti all'occhio umano. Queste evidenze confermano la natura complementare dei due paradigmi: il VAE offre compattezza latente e buona ricostruzione pixel-wise, i modelli di diffusione garantiscono stabilità di training e fedeltà visiva. Nel complesso, l'ibrido si rivela una soluzione efficace per conciliare rappresentazione latente strutturata e alta qualità percettiva.

Il lavoro presenta tuttavia alcuni limiti. La qualità finale dipende dalla bontà della ricostruzione del VAE, le cui imperfezioni possono propagarsi nel processo di denoising. Inoltre, il reverse process multi-step comporta un costo computazionale e

tempi di campionamento più elevati rispetto a un decoder VAE puro, soprattutto all'aumentare dei timestep. Infine, la batteria di metriche non include ancora FID e KID, indicatori utili per confronti più diretti con la letteratura.

Queste considerazioni orientano le direzioni future.

Dal punto di vista valutativo può essere opportuno integrare FID e KID. Il *Fréchet Inception Distance* (FID) confronta la distribuzione dei descrittori estratti con Inception v3 per immagini reali e generate, approssimandole con gaussiane multivariate e calcolando la distanza di Fréchet; valori più bassi indicano campioni più realistici e migliore copertura della distribuzione. Il *Kernel Inception Distance* (KID) misura invece la *Maximum Mean Discrepancy* con kernel polinomiale sugli stessi descrittori e impiega uno stimatore non polarizzato, più affidabile su campioni di dimensione ridotta e adatto a riportare intervalli di confidenza. Riportare entrambi rende i risultati comparabili con la letteratura e più robusti rispetto alla dimensione del campione e ai dettagli di implementazione.

Per il condizionamento, è possibile impiegare il meccanismo di *cross-attention*: le feature dell'immagine (query) prestano attenzione a chiavi e valori derivati dall'embedding di condizione (testo, classe o altra modalità), così da iniettare informazione in modo selettivo e localmente consapevole lungo la UNet. In continuità con la *classifier-free guidance* già discussa, per *guidance testuale* e *guidance di classe* intendiamo l'uso, rispettivamente, dell'embedding di un prompt testuale o della label di classe come condizione c; durante il campionamento si combinano le predizioni condizionate e non condizionate con un fattore di scala s, ottenendo un controllo continuo dell'allineamento semantico (testuale) o categoriale (di classe).

Parallelamente, si possono estendere i benchmark a dataset più sfidanti, come CIFAR-10, CelebA o immagini a risoluzione superiore; migliorando l'efficienza del campionamento tramite l'adozione di sampler avanzati (DDIM, DPM-Solver, consistency o rectified-flow) o tecniche di distillazione per ridurre il numero di passi a parità di qualità; esploreremo spazi latenti più espressivi, ad esempio VQ-VAE o β-VAE, e condizionamenti alternativi che traggano vantaggio dalla cross-attention e dalla guidance sopra descritte.

In conclusione, *DiffuseVAE* rappresenta un compromesso pratico tra efficienza, controllo e qualità visiva. L'integrazione tra una rappresentazione latente informativa e un processo di diffusione robusto fornisce una base solida e flessibile su cui innestare ulteriori miglioramenti, con l'obiettivo di elevare la qualità percettiva, accelerare la generazione e aumentare la controllabilità del contenuto.

Bibliografia

[1] AI Index Steering Committee, *The 2025 AI Index Report — Economy: Use of AI climbs to unprecedented levels*, Stanford HAI, 2025. https://hai.stanford.edu/ai-index/2025-ai-index-report/economy

[2] A. Bick, A. Blandin, D. Deming, *The Impact of Generative AI on Work Productivity*, Federal Reserve Bank of St. Louis, Febbraio 2025. https://www.stlouisfed.org/on-the-economy/2025/feb/impact-generative-ai-work-productivity

[3] McKinsey & Company, *The State of AI: Global Survey*, 2025. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai

[4] PyTorch Team, *2024 Year in Review*, Dicembre 2024. https://pytorch.org/blog/2024-year-in-review/

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *arXiv:1406.2661*, 2014. https://arxiv.org/abs/1406.2661

[6] P. Dhariwal and A. Nichol, "Diffusion Models Beat GANs on Image Synthesis," *arXiv:2105.05233*, 2021. https://arxiv.org/abs/2105.05233

[7] A. Novelli, M. Pasetti, et al., "Generative AI in EU Law: Liability, Privacy, Intellectual Property, and Cybersecurity," *arXiv:2401.07348*, 2024. https://arxiv.org/abs/2401.07348

[8] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep Unsupervised Learning using Nonequilibrium Thermodynamics," *arXiv:1503.03585*, 2015. https://arxiv.org/abs/1503.03585

[9] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models," *Advances in Neural Information Processing Systems (NeurIPS)*, 33:6840–6851, 2020. https://arxiv.org/abs/2006.11239.

[10] Lilian Weng, *What are Diffusion Models?*, Lil'Log, 2022. https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

[11] I. Strümke and H. Langseth, *Lecture Notes in Probabilistic Diffusion Models*, 2023. https://arxiv.org/abs/2312.10393

[12] J. Ho and T. Salimans, *Classifier-Free Diffusion Guidance*, 2022. https://arxiv.org/abs/2207.12598

[13] D. P. Kingma and M. Welling, *An Introduction to Variational Autoencoders*, 2019. https://arxiv.org/abs/1906.02691

[14] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, *beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework*, 2017. https://arxiv.org/abs/1611.02731

[15] K. Pandey, A. Mukherjee, P. Rai, and A. Kumar, *DiffuseVAE: Efficient, Controllable and High-Fidelity Generation from Low-Dimensional Latents*, Transactions on Machine Learning Research (TMLR), 2022. https://arxiv.org/abs/2201.00308

[16] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A Next-generation Hyperparameter Optimization Framework*, 2019. https://arxiv.org/abs/1907.10902

[17] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, *Algorithms for Hyper-Parameter Optimization*, 2011. https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86(11):2278–2324, 1998. http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

[19] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*, arXiv:1708.07747, 2017. https://arxiv.org/abs/1708.07747

Ringraziamenti

Giunto alla conclusione di questo fantastico percorso, ci tengo a ringraziare tutti voi per il supporto e la vicinanza mostrata in questi 3 anni.
In ambito accademico ringrazio la prof.ssa Lazzaro, per avermi ispirato fin dal corso di Metodi Numerici, per la disponibilità e la serietà che ha messo nel seguirmi nella stesura di questa tesi. A lei devo anche la mia passione per l'Intelligenza Artificiale.
Ringrazio la mia famiglia: Mamma, Babbo e Meli per essere stati, da sempre, il più grande esempio di come nella vita, se si vuole, si può fare tutto ciò che ci passa per la testa. La condizione necessaria è, però, metterci impegno, dedizione e passione.
Mamma, Babbo, non mi scorderò mai di tutti i sacrifici che avete fatto per me, spero di avervi resi orgogliosi.
Meli, a te auguro di raggiungere obiettivi sempre più grandi, anche più grandi di tutti quelli che io posso raggiungere... so che ne sei in grado.
Ringrazio i nonni per aver sempre creduto in me, per aver sempre sostenuto le mie idee, per non aver mai dubitato anche solo un secondo che Kevin un giorno sarebbe riuscito a raggiungere tutto questo.
Ringrazio Zia, Dori e zio Sajmiri per avermi sempre insegnato che lo studio, la costanza e il duro lavoro alla fine dei conti ripagano sempre.
Ringrazio la famiglia Bombardi: Lele, Monica e Angi. Voglio ringraziarvi di cuore per questi anni in cui mi avete sempre accolto e ascoltato come uno di famiglia. Non è mai scontato trovare persone così disponibili e affettuose, e io sono davvero grato di avervi accanto.
Ringrazio i miei amici di sempre: Checco, Biso, Marco, Nico, Ele, Leo, Canna, il Mister, Ghello, Dodo... vedere ciascuno di noi realizzarsi mi rende sempre più felice, anche per quello che sarà il nostro futuro. Nonostante vari momenti difficili abbiamo raggiunto, a mio avviso, una grande maturità... alla fine dei conti, mi duole dirlo, non potevo trovare amici migliori.
Ringrazio tutti i compagni incontrati in questo percorso: Fede, Giammi, Lisa, Sapo, Paggio, Qiu, Balza, Fronzo e l'Oraz... mi ricorderò di tutti i momenti spensierati passati a lezione, ma con ancora più gratitudine, tutti i momenti in cui ci siamo

aiutati e sostenuti. Giungere alla fine tutti insieme è un traguardo indimenticabile.
Ringrazio la persona che ha dato una svolta alla mia vita: Lucia.
Possiamo dire di essere cresciuti e maturati insieme e di aver costruito qualcosa di davvero grande, mai me lo sarei immaginato, ma tutto questo è grazie a te.
Sei sempre stata due passi avanti a tutti, e questo mi ha permesso di dare sempre il massimo, in tutti i momenti, sia belli che brutti. Mi hai insegnato ad apprezzare le piccole cose, a vedere le giornate più difficili con un occhio diverso, a credere in me stesso e tanto altro.
Sei stata la spalla più bella su cui piangere e il sorriso più bello che mi ha accompagnato e dato forza in questo percorso.
Spero di averti accanto a me per ancora tanto tempo e ancor di più spero di poter raggiungere altri innumerevoli traguardi insieme.
Faccio un piccolo ringraziamento al Kevin di tanti anni fa, il quale aveva ben chiaro di intraprendere questo percorso, anche se inconsapevole del fatto che senza la famiglia e le persone giuste al proprio fianco è davvero difficile raggiungere grandi obiettivi.