
DIFFUSION MODELS

Raffaele Cerizza

Università di Milano-Bicocca
Matricola 845512

r.cerizza1@campus.unimib.it

Giacomo Savazzi

Università di Milano-Bicocca
Matricola 845372

g.savazzi1@campus.unimib.it

Thomas Barbera

Università di Milano-Bicocca
Matricola 845538

t.barbera@campus.unimib.it

February 17, 2023

ABSTRACT

I Diffusion Models stanno emergendo negli ultimi anni come nuova tipologia di Modelli Generativi grazie alla notorietà acquisita sia in ambito scientifico che commerciale. La teoria ad essi sottostante offre nuove prospettive per lo sviluppo di temi di ricerca collegati, come la correzione di immagini degradate. Questo lavoro ha per oggetto l'analisi di alcune tipologie di Diffusion Models, a partire dall'introduzione dei fondamenti teorici alla loro base. In particolare vengono analizzati i Denoising Diffusion Probabilistic Models, i Denoising Diffusion Implicit Models, le Noise Conditional Score Networks, e gli Score-Based Generative Models tramite Equazioni Differenziali Stocastiche.

1 Introduzione

I Diffusion Models appartengono alla categoria dei Deep Generative Models. Questi modelli sfruttano reti neurali per generare campioni appartenenti a distribuzioni complesse [1]. Tra questi modelli si possono ricordare anche le Generative Adversarial Networks (GAN), i Variational Autoencoders (VAE) e i Normalizing Flows [2]. I Diffusion Models hanno recentemente acquisito notorietà grazie al successo di alcune applicazioni basate su questi modelli. Un esempio di queste applicazioni è DALL·E, che permette di generare immagini a partire da descrizioni testuali [3]. Nella Figura 1 sono mostrati alcuni esempi di immagini generate con DALL·E 2. Il successo di queste applicazioni è stato affiancato da una prolifica e repentina ricerca scientifica [2].

Esistono diverse tipologie di Diffusion Model. In questo lavoro sono state analizzati:

- **Denoising Diffusion Probabilistic Models** (in seguito: DDPM): questi modelli utilizzano (*i*) una prima catena di Markov per aggiungere progressivamente rumore ai dati secondo un processo di diffusione [5]; e (*ii*) una seconda catena di Markov parametrizzata, addestrata sfruttando l'inferenza variazionale per invertire il processo di diffusione [6];



Figure 1: Esempi di immagini generate con DALL·E 2. Questi esempi sono tratti da [4].

- **Denoising Diffusion Implicit Models** (in seguito: DDIM): questi modelli utilizzano processi di diffusione non Markoviani [7];
- **Noise Conditional Score Networks** (in seguito: NCSN): questi modelli si basano sulla stima dello score tramite il processo di score-matching, e utilizzano la dinamica di Langevin per il campionamento [8, p. 5–7];
- **Modelli Score-Based basati su Stochastic Differential Equations**: anche questi modelli si basano sulla stima dello score, e rappresentano una generalizzazione dei modelli Score-Based e DDPM [9, 2].

L'esposizione del presente lavoro seguirà questo ordine:

1. Descrizione dei fondamenti teorici dei Diffusion Models;
2. Presentazione dell'architettura U-Net quale tipica rete neurale utilizzata per l'addestramento dei Diffusion Models;
3. Approfondimento delle diverse tipologie di Diffusion Models appena introdotte;
4. Applicazione pratica e confronto di questi modelli.

2 Fondamenti teorici

In questa sezione verranno presentati i fondamenti teorici dei Diffusion Models. In particolare, verranno affrontati i seguenti argomenti:

1. Verranno fornite alcune nozioni di base in merito alle catene di Markov, in quanto fondamentali per la comprensione del processo diffusivo;
2. Verrà descritto il forward diffusion process;
3. Verrà descritto il reverse diffusion process;
4. Verrà esposto il tipico procedimento di addestramento dei Diffusion Models, con particolare attenzione alla definizione della loss function.

2.1 Catene di Markov

Una catena di Markov è un processo stocastico di Markov con determinate caratteristiche. In particolare:

- Un processo stocastico $\{X_t, t \in T\}$ è un insieme di variabili aleatorie che evolvono nel tempo. In questo caso T denota l'insieme degli indici temporali; t è l'istante di tempo considerato; e X_t è il valore della variabile aleatoria che descrive lo stato del sistema all'istante di tempo t . L'insieme dei valori assumibili dalle variabili aleatorie viene detto “spazio degli stati”. Lo spazio degli stati può essere continuo o discreto. Parimenti, il tempo può essere continuo o discreto;
- Un processo stocastico di Markov è un processo stocastico che gode della proprietà di Markov. Questa proprietà richiede che la distribuzione di probabilità del processo in ogni istante di tempo futuro dipenda (non da tutta la storia dei valori passati, ma) solo dal valore corrente. In questo caso si parla di processi Markoviani del primo ordine¹. Più formalmente la proprietà di Markov viene espressa attraverso la seguente equazione:

$$P(X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, \dots, X_0 = x_0) = P(X_{t+1} = x_{t+1} | X_t = x_t), \quad (1)$$

dove con le lettere minuscole si specificano le realizzazioni delle variabili aleatorie. Il simbolo “|” mette in evidenza che P specifica una distribuzione di probabilità condizionata;

- Una catena di Markov è un processo stocastico di Markov caratterizzato da:
 - uno spazio degli stati S ;
 - una matrice P_{mat} delle probabilità di transizione tra stati. Questa matrice definisce la probabilità di passare da uno stato S_i a uno stato S_j . Gli elementi di questa matrice sono definiti come:

$$p_{mat,ij} = P(X_{t+1} = S_j | X_t = S_i), \quad (2)$$

con $p_{mat,ij} \in [0, 1]$. La somma degli elementi di ciascuna riga di P_{mat} è pari a 1. Pertanto ogni transizione comporta certamente il passaggio da uno stato di partenza a uno dei possibili stati di arrivo, eventualmente rimanendo nello stesso stato di partenza. La matrice delle probabilità di transizione tra stati in una catena di Markov è un Markov kernel [10, p. 865];

¹Quando la distribuzione di probabilità del processo in ogni istante di tempo futuro dipende sia dal valore corrente che da quello immediatamente precedente si parla di processo Markoviano del secondo ordine. Pertanto l'ordine del processo Markoviano è dato dal numero di valori da cui dipende il valore futuro.

- una distribuzione iniziale delle probabilità di transizione tra gli stati π_0 . In questo caso π_0 può essere rappresentato come un vettore che specifica la probabilità di ogni stato di S all’istante di tempo 0. In particolare, dati gli stati $\{S_1, S_2, \dots, S_N\}$, $\pi_0(i) = P\{X_0 = S_i\}$. Pertanto l’ i -esimo elemento di π_0 specifica la probabilità che X_0 valga S_i all’istante di tempo iniziale 0.

Una catena di Markov si definisce **stazionaria** quando la probabilità di passare da uno stato ad un altro è indipendente dal tempo. Questo significa che, se la distribuzione π_t al tempo t è stazionaria, allora sia la distribuzione π_{t+1} al tempo $t+1$ sia le distribuzioni successive sono uguali a π_t . In questo quadro una distribuzione di probabilità π è stazionaria quando risulta soddisfatta la seguente equazione:

$$\pi = \pi P_{mat}. \quad (3)$$

2.2 Forward diffusion process

Sin qui sono state descritte le catene di Markov. La conoscenza delle catene di Markov è essenziale per comprendere il funzionamento dei Diffusion Models, e in particolare dei DDPM. Infatti, questi modelli sfruttano le catene di Markov per operare (e invertire) un processo diffusivo ispirato dalla fisica [5]. Il processo generativo dei Diffusion Models si può scindere in due parti: (i) una prima parte denominata **forward diffusion process**, e (ii) una seconda parte denominata **reverse diffusion process**. Il forward diffusion process viene descritto in questo paragrafo. Il reverse diffusion process sarà invece descritto nel prossimo paragrafo (vedi Paragrafo 2.3).

Descrizione. Sia data la distribuzione dei dati reali $q(x_0)$. Il forward diffusion process converte gradualmente un dato appartenente a questa distribuzione in un nuovo dato appartenente a una distribuzione che sia semplice e trattabile analiticamente, tipicamente una distribuzione Gaussiana [5]. Il procedimento è il seguente [11]:

- viene considerato un campione x_0 appartenente alla distribuzione dei dati reali $x_0 \sim q(x_0)$;
- a questo campione viene progressivamente aggiunto rumore (tipicamente Gaussiano²) in T passi;
- quindi, ad ogni passo si ottiene un nuovo campione più rumoroso che presenta le stesse dimensioni di x_0 , e che rappresenta una variabile latente;
- la sequenza di questi campioni rumorosi viene denotata con x_1, x_2, \dots, x_T .

Si è detto che a ogni passo del procedimento viene aggiunto del rumore ai dati. In questo paragrafo e nei successivi verrà preso in considerazione il rumore Gaussiano. In particolare, il rumore viene aggiunto attraverso un processo stocastico che soddisfa la proprietà di Markov e che è qualificabile come catena di Markov. Più precisamente, viene utilizzato un kernel di transizione $q(x_t | x_{t-1})$ definito come segue:

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t \mathbf{I}), \quad (4)$$

dove \mathbf{I} è la matrice di covarianza dell’identità (che ha le stesse dimensioni del dato di partenza x_0) e $\beta_t \in (0, 1)$ è il valore della *variance schedule* al tempo t [12]. La matrice \mathbf{I} comporta che ogni dimensione dell’input avrà la stessa varianza β_t . Inoltre, in merito alla notazione utilizzata [2]:

- \mathcal{N} indica una distribuzione normale;
- x_t rappresenta l’output della distribuzione;
- $\sqrt{1 - \beta_t} x_{t-1}$ rappresenta la media della distribuzione, che ad ogni passo viene diminuita in modo da avvicinarsi sempre di più allo 0;
- $\beta_t \mathbf{I}$ rappresenta la covarianza, che a sua volta specifica il livello del rumore aggiunto.

In questo quadro l’Equazione 4 specifica che a ogni passo del forward diffusion process viene generato un nuovo campione partendo da una distribuzione normale avente come media il dato precedente e come varianza il rumore aggiunto. I valori della *variance schedule* β_1, \dots, β_T possono essere oggetto di apprendimento oppure possono essere costanti [6].

²Il rumore Gaussiano è un rumore statistico che ha una funzione di densità di probabilità che segue una distribuzione normale. Nell’ambito delle immagini, il rumore si manifesta come una variazione casuale del colore.

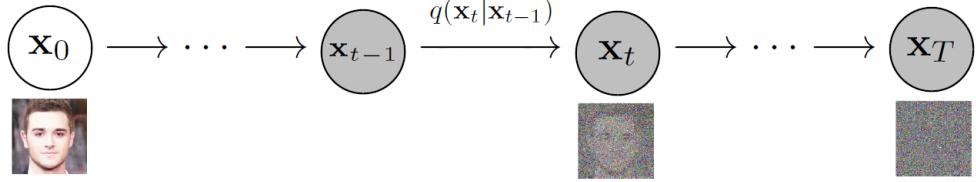


Figure 2: Rappresentazione del forward diffusion process. Al dato di partenza x_0 viene progressivamente aggiunto rumore Gaussiano. Questo porta alla distruzione dei suoi tratti distintivi all'ultimo istante di tempo T . Questa rappresentazione riprende e modifica la Figura 2 mostrata in [6].

Proprietà. Una proprietà del forward diffusion process definito in questo modo consiste nella possibilità di campionare x_t in forma chiusa per un arbitrario istante di tempo t . In particolare:

- sia $\alpha_t = 1 - \beta_t$;
- sia $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$ il prodotto di tutti i valori α_t dal primo istante di tempo fino a quello corrente;
- allora:

$$q(x_t | x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t) \mathbf{I})[6]. \quad (5)$$

Pertanto x_t può essere calcolato semplicemente attraverso la seguente equazione:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, \quad (6)$$

dove $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ rappresenta il rumore Gaussiano [12].

Inoltre, la distribuzione $q(x_t | x_0)$ converge a una distribuzione stazionaria quando t tende a infinito, e questo indipendentemente dal valore iniziale x_0 [13]. Pertanto, se la distribuzione ricercata attraverso l'aggiunta di rumore Gaussiano è una distribuzione Gaussiana, allora x_∞ apparirà a questa distribuzione.

Conclusioni. In questo quadro il forward diffusion process aggiunge progressivamente rumore ai dati iniziali finché la loro struttura non è completamente distrutta [12]. Questo significa che i dati perdono progressivamente i loro tratti distintivi. All'ultimo passo del processo il dato assume una distribuzione Gaussiana isotropa³ [14]. Pertanto $q(x_T) = \mathcal{N}(x_T; 0, \mathbf{I})$ [12]. Il forward diffusion process può essere rappresentato come mostrato in Figura 2.

2.3 Reverse diffusion process

Sin qui è stato descritto il forward diffusion process. Ora verrà dettagliato il reverse diffusion process.

Descrizione. L'obiettivo del reverse diffusion process consiste nell'invertire il processo di diffusione descritto al paragrafo precedente. In particolare:

- il reverse diffusion process considera inizialmente un dato rumoroso appartenente alla distribuzione cercata (e ottenuta) con il forward diffusion process. Nel caso di distribuzione Gaussiana il dato di partenza sarà $p(x_T) = \mathcal{N}(x_T; 0, \mathbf{I})$;
- successivamente, a ogni passo del reverse diffusion process viene rimossa una parte del rumore. Anche in questo caso i nuovi campioni hanno le stesse dimensioni del dato originale;
- alla fine si vuole ottenere un dato appartenente alla distribuzione originaria $q(x_0)$.

Problema. Ora, il forward diffusion process è fisso (non presenta parametri da ottimizzare). Questo comporta che tale processo può essere invertito. In particolare l'esistenza del processo di diffusione inversa è garantita per il caso di diffusione Gaussiana. Infatti, nel caso di diffusione Gaussiana, il kernel di transizione inverso a quello del forward diffusion process ne ha la stessa forma funzionale quando il valore di β è piccolo. Quindi, se $q(x_t | x_{t-1})$ ha forma Gaussiana, anche $q(x_{t-1} | x_t)$ avrà forma Gaussiana [5].

Tuttavia, il calcolo del kernel di transizione inverso $q(x_{t-1} | x_t)$ è intrattabile [7]. Infatti, per arrivare alla distribuzione iniziale $q(x_0)$ partendo dal campione rumoroso x_T è necessario marginalizzare sopra tutti i modi con cui è possibile arrivare a x_0 . Questo significa riuscire a calcolare $\int q(x_{0:T}) dx_{1:T}$, che però è intrattabile [5]. Pertanto occorre individuare un kernel di transizione parametrizzato che possa approssimare il kernel di transizione cercato.

³Una distribuzione isotropa presenta gli stessi valori in tutte le direzioni.

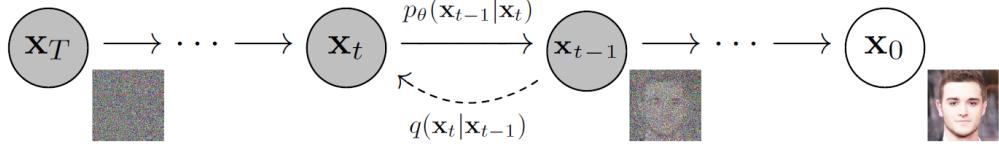


Figure 3: Rappresentazione del reverse diffusion process. In questo caso il dato di partenza x_T è un dato rumoroso. Durante il reverse diffusion process il rumore viene gradualmente rimosso. Infine si ottiene nuovamente il dato originale x_0 degradato attraverso il forward diffusion process. Questa rappresentazione riprende la Figura 2 mostrata in [6].

Soluzione. Una soluzione a questo problema consiste nell'utilizzare un kernel di transizione parametrizzato $p_\theta(x_{t-1} | x_t)$ definito come segue:

$$p_\theta(x_{t-1} | x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)), [12] \quad (7)$$

dove θ denota i parametri da ottimizzare. Naturalmente questo kernel di transizione avrà forma Gaussiana dato che anche $q(x_{t-1} | x_t)$ ha forma Gaussiana. In questo caso la media e la covarianza della distribuzione risultano parametrizzate. E questi parametri possono essere ottimizzati attraverso l'addestramento di un modello.

Condizionamento a x_0 . Sebbene $q(x_{t-1} | x_t)$ sia intrattabile, è stato dimostrato che $q(x_{t-1} | x_t, x_0)$ è invece trattabile [6]. In particolare, $q(x_{t-1} | x_t, x_0)$ può essere definito come segue:

$$q(x_{t-1} | x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t \mathbf{I}), \quad (8)$$

dove

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\alpha_{t-1}} \beta_t}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t} (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t, \quad (9)$$

e

$$\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t. \quad (10)$$

Questa informazione sarà rilevante per la definizione della loss function da ottimizzare durante l'addestramento del modello parametrizzato.

Conclusione. Il forward diffusion process aggiunge progressivamente rumore ai dati. Il reverse diffusion process rimuove progressivamente rumore dai dati in modo da ottenere nuovamente i dati originali. Il reverse diffusion process avviene utilizzando un modello parametrizzato che può essere addestrato. L'addestramento verrà illustrato nel prossimo paragrafo. Il reverse diffusion process può essere rappresentato come mostrato in Figura 3.

2.4 Addestramento

Nel paragrafo precedente è stato descritto il reverse diffusion process. Questo processo è caratterizzato da un kernel di transizione parametrizzato. I parametri di questo kernel di transizione possono essere ottimizzati attraverso l'addestramento di uno specifico modello. In questo paragrafo verrà definita la forma generale della loss function utilizzata per questo addestramento.

Parametri e obiettivo. L'obiettivo primario sarebbe ottimizzare la seguente verosimiglianza:

$$p_\theta(x_0) = \int p_\theta(x_{0:T-1} | x_T) \cdot p(x_t) dx_{1:T}^4. \quad (11)$$

In questo modo si vorrebbe massimizzare la verosimiglianza rispetto al dato iniziale x_0 . Questo consentirebbe al reverse diffusion process di eliminare correttamente il rumore, per ottenere dati appartenenti alla distribuzione originaria. Tuttavia il calcolo di questa verosimiglianza è intrattabile [14].

⁴Si usa la notazione $p_\theta(x_0)$ per la verosimiglianza $p(x_0 | \theta)$ in modo da avere una notazione consistente (i) rispetto al resto dell'esposizione e (ii) rispetto alla letteratura citata. Naturalmente $p_\theta(x_0)$ è una funzione di verosimiglianza in quanto si vogliono trovare i parametri θ che rendono più probabile il raggiungimento di x_0 .

Evidence Lower Bound. Una soluzione a questo problema di intrattabilità consiste nell'utilizzare un *Evidence Lower Bound* (in seguito: ELBO). Il nome stesso suggerisce che si tratta di un limite inferiore rispetto a un valore da stimare. La definizione formale dell'ELBO è la seguente:

- siano date due variabili aleatorie X e Z ;
- sia $x \sim p_\theta$ un campione della distribuzione p_θ ;
- sia q_ϕ una distribuzione diversa da p_θ ;
- allora vale che:

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x, z)}{q_\phi(z | x)} \right], \quad (12)$$

dove il membro sinistro della disequazione rappresenta l'evidenza logaritmica di x , mentre il membro destro rappresenta il limite inferiore dell'evidenza logaritmica di x (ovvero l'ELBO) [15].

In particolare, l'ELBO permette di trasformare un problema di inferenza intrattabile in un problema di ottimizzazione che può essere risolto utilizzando algoritmi di ottimizzazione noti, come il metodo di discesa del gradiente. In questo caso si parla di inferenza variazionale [16].

Ora, nel caso di specie è possibile definire l'ELBO sulla *negative log-likelihood* $-\log p_\theta(x_0)$. Più precisamente è possibile definire la seguente disequazione:

$$\mathbb{E}[-\log p_\theta(x_0)] \leq \mathbb{E}_q \left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} | x_0)} \right], \quad (13)$$

dove il membro destro della disequazione è l'ELBO [6]. Si noti che in questo caso i membri hanno segno negativo. Quindi il verso della disequazione è mutato rispetto alla definizione dell'ELBO riportata nella Disequazione 12. Pertanto in questo caso l'ELBO è (non un limite inferiore, ma) un limite superiore.

L'ELBO appena definito può essere manipolato come segue:

$$\mathbb{E}_q \left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} | x_0)} \right] = \mathbb{E}_q \left[-\log p(x_T) - \sum_{t \geq 1} \log \frac{p_\theta(x_{t-1} | x_t)}{q(x_t | x_{t-1})} \right] = L. \quad (14)$$

In questo quadro l'Equazione 14 rappresenta una prima versione di loss function che può essere utilizzata per ottimizzare i parametri θ durante l'addestramento.

Quando si utilizza la definizione canonica dell'ELBO (con segno positivo come nella Disequazione 12), l'ottimizzazione consiste tipicamente in una massimizzazione [17]. Tuttavia, poiché l'ELBO definito con l'Equazione 14 ha segno negativo, in questo caso l'ottimizzazione consiste nella minimizzazione. Infatti, la minimizzazione dell'ELBO comporta certamente anche la minimizzazione della *negative log-likelihood* $-\log p_\theta(x_0)$, dato che questa è sempre minore o uguale all'ELBO.

Divergenza di Kullback-Leibler. La loss function definita con l'Equazione 14 può essere ulteriormente manipolata sfruttando la divergenza di Kullback-Leibler (in seguito anche: D_{KL}). In particolare, la riformulazione dell'ELBO in termini di divergenza di Kullback-Leibler è utile in quanto le due catene di Markov definite per il processo di diffusione (in avanti e inversa) sono Gaussiane. E la divergenza di Kullback-Leibler tra due distribuzioni Gaussiane può essere calcolata analiticamente in forma chiusa [2]. Conviene dunque (i) prima fornire una definizione della divergenza di Kullback-Leibler e (ii) poi ridefinire la loss function.

La divergenza di Kullback-Leibler (detta anche entropia relativa) è una misura non simmetrica della differenza tra due distribuzioni di probabilità⁵. La divergenza di Kullback-Leibler può essere definita come segue [18]. Siano p e q due distribuzioni discrete. Allora:

$$D_{KL}(p || q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}. \quad (15)$$

In questo modo il valore di D_{KL} misura l'informazione persa quando q è usata per approssimare p . Inoltre il valore di D_{KL} è sempre non negativo ed è pari a 0 solo quando $p = q$. La divergenza di Kullback-Leibler può essere rappresentata come mostrato in Figura 4.

⁵La divergenza di Kullback-Leibler non è simmetrica in quanto il valore di D_{KL} dalla distribuzione p alla distribuzione q può essere diverso dal valore di D_{KL} dalla distribuzione q alla distribuzione p . Per questo motivo si parla (non di "distanza", ma) di "divergenza".

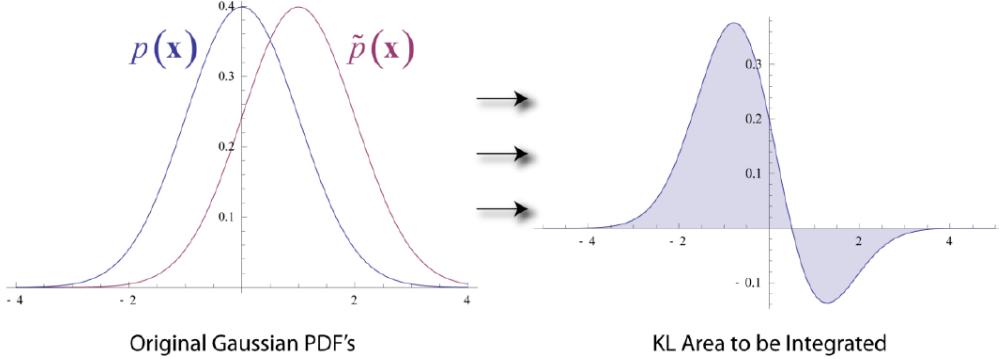


Figure 4: Rappresentazione della divergenza di Kullback-Leibler. A sinistra sono mostrate due distribuzioni di probabilità Gaussiane $p(x)$ e $\tilde{p}(x)$. A destra è mostrata la divergenza di Kullback-Leibler tra le due distribuzioni di probabilità. Il valore della divergenza è dato dall'area sottesa alla curva. Questa rappresentazione è tratta da [19].

Loss function Chiarito il significato della divergenza di Kullback-Leibler, è possibile riformulare l'Equazione 14. La riformulazione è la seguente:

$$L = \mathbb{E}_q \left[\underbrace{D_{\text{KL}}(q(x_T | x_0) || p(x_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\text{KL}}(q(x_{t-1} | x_t, x_0) || p_\theta(x_{t-1} | x_t))}_{L_{t-1}} - \underbrace{\log p_\theta(x_0 | x_1)}_{L_0} \right] [6]. \quad (16)$$

Questa equazione presenta diversi termini. In particolare:

- Il primo termine L_T è: $D_{\text{KL}}(q(x_T | x_0) || p(x_T))$. Questo termine specifica quanto $p(x_T)$ sia prossimo a una distribuzione Gaussiana isotropa. Questo termine non presenta parametri θ da addestrare. Infatti, q non ha parametri e x_T è semplicemente un campione di rumore Gaussiano. Pertanto questo primo termine può essere ignorato durante l'addestramento. Il calcolo di $q(x_T | x_0)$ può comunque essere ricavato dall'Equazione 5.
- Il secondo termine L_{t-1} è: $D_{\text{KL}}(q(x_{t-1} | x_t, x_0) || p_\theta(x_{t-1} | x_t))$. Questo termine comporta che, per ogni istante di tempo t , il modello deve essere addestrato in modo che $p_\theta(x_{t-1} | x_t)$ sia il più possibile vicino alla vera probabilità a posteriori del forward diffusion process condizionata al dato originale (ovvero $q(x_{t-1} | x_t, x_0)$) [2]. Pertanto i parametri θ sono ottimizzati in modo tale che la distribuzione congiunta della catena di Markov inversa $p_\theta(x_0, x_1, \dots, x_T)$ approssimi il più possibile quella della catena di Markov del forward diffusion process $q(x_0, x_1, \dots, x_T)$. E questo permette al reverse diffusion process di eliminare correttamente il rumore al fine di ottenere dati appartenenti alla distribuzione originaria. In questo caso il calcolo di $q(x_{t-1} | x_t, x_0)$ può essere ricavato dall'Equazione 8; il calcolo di $p_\theta(x_{t-1} | x_t)$ invece può essere ricavato dall'Equazione 7.
- Il terzo termine L_0 è: $-\log p_\theta(x_0 | x_1)$. Questo termine viene denominato *reconstruction term*. Questo termine misura la probabilità logaritmica del dato originale x_0 data la prima variabile latente x_1 [20]. Anche il calcolo di questo termine può essere ricavato dall'Equazione 7.

Conclusione. È possibile addestrare un modello al fine di ottimizzare i parametri della media e della covarianza del kernel di transizione del reverse diffusion process. Il modello di addestramento può essere rappresentato da una rete neurale [11]. Nella prossima Sezione verrà descritta una tipica architettura di rete neurale utilizzata per il predetto addestramento.

3 Architettura: U-Net

La Sezione 2 è stata dedicata alla descrizione della teoria sottostante i Diffusion Models. In particolare nel Paragrafo 2.4 è stato descritto l'addestramento dei Diffusion Models. Per questo addestramento può essere utilizzata una rete neurale. Il modello tipicamente utilizzato per l'addestramento dei Diffusion Models è quello di una rete U-Net. In questo paragrafo viene descritta questa architettura.

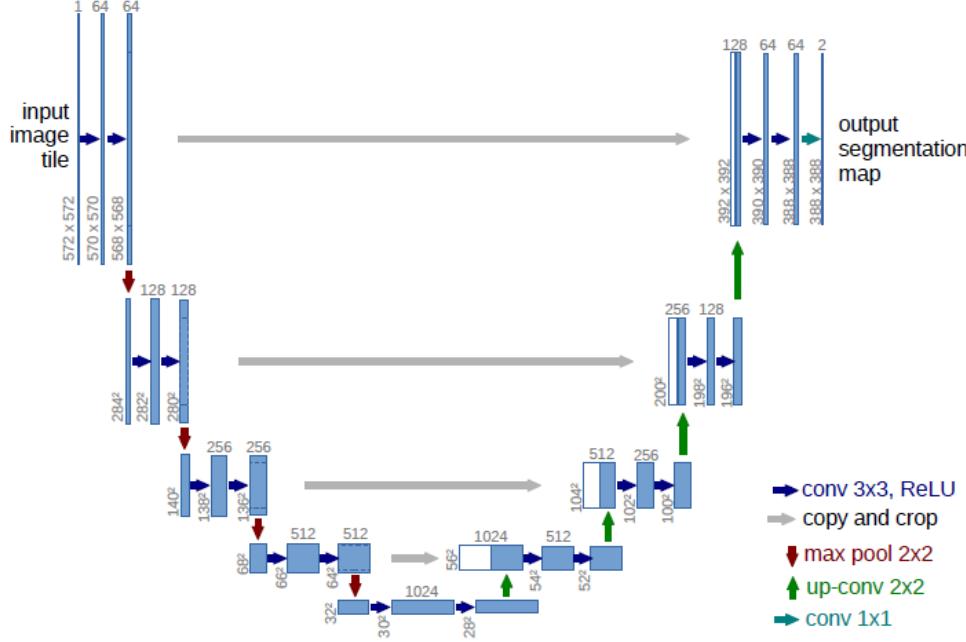


Figure 5: Rappresentazione dell’architettura di U-Net. La prima parte della rete riduce le dimensioni spaziali dell’immagine fornita in input estraendo le informazioni rilevanti. La seconda parte della rete invece espande nuovamente le dimensioni spaziali dell’immagine localizzando le informazioni estratte. La prima e la seconda parte dell’architettura sono simmetriche e la struttura nata dalla loro combinazione assume una forma a “U” da cui il nome della rete trae ispirazione. Questa rappresentazione è tratta da [21].

Prima versione. La prima versione di U-Net risale al 2015 [21]. Questa rete trae a sua volta ispirazione dall’architettura denominata Fully Convolutional Network [22]. La versione originaria di U-Net è stata pensata per la segmentazione di immagini, dove viene associata una label a ogni pixel. Tuttavia ha trovato larga applicazione anche in altri ambiti, come appunto la generazione di immagini [6, 7]. Una rappresentazione dell’architettura di U-Net è mostrata nella Figura 5. L’architettura di U-Net è simmetrica ed è composta da due parti: una prima parte riduttiva e una seconda parte espansiva. In particolare:

- La prima parte persegue l’obiettivo di catturare le feature più rilevanti dalle immagini fornite in input. Per raggiungere questo obiettivo utilizza una serie di layer convoluzionali. In particolare vengono utilizzati blocchi composti da:
 - due layer convoluzionali di dimensioni 3x3, senza *padding* e con funzione di attivazione ReLU;
 - seguiti da un layer di max pooling con dimensioni 2x2 e con un valore di *stride* pari a 2.

L’ultimo blocco della prima parte presenta solo due layer convoluzionali, non seguiti da un layer di max pooling. Il numero di feature channels viene raddoppiato dopo ogni blocco. In particolare dai 64 canali del primo blocco si arriva ai 1024 canali dell’ultimo blocco.

- La seconda parte persegue invece l’obiettivo di ricostruire le immagini localizzando le informazioni estratte. In questo caso vengono utilizzati blocchi composti da:
 - un primo layer di upsampling⁶ per incrementare le dimensioni spaziali di altezza e larghezza;
 - un successivo layer di up-convolution di dimensioni 2x2 per dimezzare il numero di feature channels⁷;

⁶I layer di upsampling incrementano semplicemente le dimensioni spaziali dell’immagine mediante interpolazione. Questi layer non hanno parametri da addestrare. Pertanto i layer di upsampling non sono layer di convoluzione trasposta.

⁷La combinazione del layer di upsampling e del layer di up-convolution produce un’immagine avente il doppio delle dimensioni spaziali e la metà dei canali rispetto al blocco precedente.

- una concatenazione del risultato di questa up-convolution con la corrispondente mappa di feature avente lo stesso numero di canali nella prima parte (riduttiva) della rete⁸. Questa concatenazione avviene attraverso skip connections;
- e infine due layer convoluzionali di dimensioni 3x3 con funzione di attivazione ReLU⁹.

Come ultimo layer viene utilizzata una convoluzione di dimensioni 1x1 per mappare ogni vettore di feature nel numero di classi desiderato. In questo modo viene associata la label predetta a ogni pixel dell'immagine.

La rete così descritta presenta complessivamente 23 layer convoluzionali e nessun layer completamente connesso [21]. L'utilizzo di un *padding* pari a 0 comporta che l'output della rete abbia dimensioni spaziali inferiori rispetto all'immagine fornita in input. Tuttavia utilizzando un diverso *padding* nei layer convoluzionali è possibile ottenere un output avente le stesse dimensioni dell'immagine fornita in input. Questo risulta particolarmente utile per i Diffusion Models. Infatti sia il forward diffusion process che il reverse diffusion process producono a ogni passo un output avente le stesse dimensioni del dato fornito in input, ma con diversi livelli di rumore. Per questo motivo l'architettura di U-Net ha trovato largo impiego nell'ambito dei Diffusion Models.

Versione migliorata. Una nuova versione di U-Net è stata proposta in [23]. Questa versione consente di ottenere performance migliori nella generazione di immagini. In particolare le modifiche principali di questa nuova versione sono le seguenti:

- viene aggiunto un global attention layer con teste multiple di 64 canali nei blocchi relativi alle risoluzioni 32x32, 16x16 e 8x8¹⁰;
- viene aggiunto un layer denominato Adaptive Group Normalization in ogni blocco. In particolare questo layer incorpora l'istante di tempo e l'embedding della classe dopo un'operazione di normalizzazione di gruppo [23]¹¹;
- vengono modificati i blocchi della rete utilizzando la struttura dei blocchi di BigGAN [25]. Questa struttura prevede il seguente ordine di layer: (i) un layer di normalizzazione; (ii) un layer convoluzionale; (iii) un altro layer di normalizzazione; e (iv) un altro layer convoluzionale;
- vengono riscalate le skip connections con un valore di $\frac{1}{\sqrt{2}}$.

4 Diffusion Models

Sin qui sono stati presentati alcuni concetti generali che interessano i Diffusion Models ed è stata descritta la principale architettura di rete neurale utilizzata per l'addestramento di questi modelli. Ora verranno approfonditi alcuni specifici Diffusion Models. In particolare verranno presentati: (i) i Denoising Diffusion Probabilistic Models (DDPM); (ii) i Denoising Diffusion Implicit Models (DDIM); (iii) le Noise Conditional Score Networks (NCSN); e (iv) i modelli *score-based* basati su Stochastic Differential Equations (SBGM-SDE).

4.1 Denoising Diffusion Probabilistic Models

Le radici dei DDPM risalgono al 2015 [5]. Tuttavia la loro formulazione più compiuta è arrivata nel 2020 [6]. Molti concetti generali esposti nella Sezione 2 trovano applicazione anche per i DDPM. In questa Sezione verranno specificate le peculiarità e le novità dei DDPM rispetto a quei concetti generali.

⁸La concatenazione con informazioni provenienti da layer precedenti permette alla rete di migliorare la precisione delle predizioni. Infatti nella prima parte la rete individua le informazioni più rilevanti presenti nelle immagini, ma perde la localizzazione di queste informazioni. Nella seconda parte la rete riesce a recuperare la localizzazione spaziale delle informazioni associando il risultato dell'up-convolution alle informazioni estratte dai dati aventi le stesse dimensioni spaziali nella prima parte della rete.

⁹I layer convoluzionali nella parte espansiva della rete sono addestrati per registrare la combinazione delle informazioni della prima parte della rete con il risultato dell'up-convolution. In questo modo si vuole ottenere un output più preciso.

¹⁰Gli attention layer sono layer tipici dell'architettura dei Transformer [24]. L'obiettivo degli attention layer è quello di concentrare l'attenzione sulle informazioni più importanti, trascurando quelle meno importanti. In particolare gli attention layer utilizzano dei pesi per evidenziare l'importanza delle informazioni. Nel caso dei Diffusion Models viene considerato anche l'istante di tempo attraverso l'inserimento del *sinusoidal position embedding* dei Transformer all'interno di ogni blocco [6].

¹¹In particolare questo layer è definito come:

$$\text{AdaGN}(h, y) = y_s \text{GroupNorm}(h) + y_b, \quad (17)$$

dove h è la funzione di attivazione del blocco dopo la prima convoluzione e $y = [y_s, y_b]$ è ottenuto come proiezione lineare dell'istante di tempo e dell'embedding della classe [23]. La normalizzazione di gruppo divide i canali in gruppi e normalizza le feature di ciascun gruppo.

Varianza β_t . La prima peculiarità riguarda i valori β_t della *variance schedule*. Al Paragrafo 2.2 si è anticipato che questi valori possono essere fissati o possono costituire oggetto di ottimizzazione. In [6] si è scelto di utilizzare valori β_t fissati e linearmente crescenti nel tempo. In particolare si è scelto un valore iniziale $\beta_1 = 10^{-4}$ e un valore finale è $\beta_T = 0.02$. Questa scelta è stata fatta in modo che il termine L_T nell'Equazione 16 della loss function non avesse parametri da ottimizzare e quindi potesse essere ignorato durante l'addestramento.

In successive implementazioni dei DDPM è stato proposto di utilizzare (non una *linear schedule*, ma) una *cosine schedule* per la determinazione dei valori β_t ¹². Infatti si è osservato che la *linear schedule* risulta sub-ottimale per immagini a bassa risoluzione [26].

Covarianza Σ_θ . In [6] anche la covarianza viene fissata in modo da essere costante. In particolare il valore di $\Sigma_\theta(x_t, t)$ è fissato pari a $\sigma^2 \mathbf{I}$. In questo caso si è osservato che $\sigma^2 = \beta_t$ e $\sigma^2 = \tilde{\beta}_t$ conducono a risultati simili¹³. Tuttavia successivamente è stato verificato che anche la covarianza fissata in modo costante porta a risultati sub-ottimali [23]. Pertanto è stato proposto di parametrizzare anche la covarianza con una rete neurale. Più precisamente Σ_θ è stata definita come:

$$\Sigma_\theta(x_t, t) = \exp(v \log \beta_t + (1 - v) \log \tilde{\beta}_t), \quad (20)$$

dove v è l'output di una rete neurale. In questo modo la covarianza è data dall'interpolazione tra β_t e $\tilde{\beta}_t$ [26].

Media μ_θ . Al Paragrafo 2.4 è stata definita la loss function con l'Equazione 16. Questa loss function presenta diversi termini da ottimizzare. Tra questi termini vi è il termine L_{t-1} definito come segue:

$$L_{t-1} = D_{\text{KL}}(q(x_{t-1} | x_t, x_0) \| p_\theta(x_{t-1} | x_t)). \quad (21)$$

L'obiettivo dell'ottimizzazione è minimizzare la divergenza di Kullback-Leibler tra $q(x_{t-1} | x_t, x_0)$ e $p_\theta(x_{t-1} | x_t)$. In particolare la media del primo elemento può essere definita come:

$$\tilde{\mu}_t(x_t) = \frac{1}{\sqrt{\alpha_t}} (x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon), \quad (22)$$

sfruttando (i) la definizione di $\tilde{\mu}_t(x_t, x_0)$ di cui all'Equazione 9 e (ii) la definizione di x_0 ricavabile dall'Equazione 6. In questo quadro si vuole ottimizzare la media del secondo elemento μ_θ in modo da predire $\tilde{\mu}_t$. Questo porta a definire $\mu_\theta(x_t, t)$ come:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} (x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t)), \quad (23)$$

dove ϵ_θ è una funzione che approssima ϵ da x_t [6].

Loss function. La ridefinizione dei termini Σ_θ e μ_θ porta a una semplificazione della loss function definita all'Equazione 16. In particolare la loss function può essere semplificata come:

$$L_{\text{simple}} = \mathbb{E}_{t, x_0, \epsilon} [\|\epsilon - \epsilon_\theta(x_t, t)\|^2] [6, 26]. \quad (24)$$

In questo quadro la loss function si riduce al semplice calcolo dell'errore quadratico medio tra il vero rumore Gaussiano ϵ e quello predetto ϵ_θ .

Qualora si consideri la covarianza Σ_θ (non costante, ma) parametrizzata, allora la loss function può essere ridefinita come:

$$L_{\text{hybrid}} = L_{\text{simple}} + \lambda L_{\text{vlb}}, \quad (25)$$

dove L_{vlb} è la loss function di cui all'Equazione 16 e λ è una costante di valore inferiore a 1 per evitare che L_{vlb} risulti predominante rispetto a L_{simple} . Tipicamente viene utilizzato $\lambda = 0.001$ [26].

Algoritmi. Gli algoritmi di addestramento e campionamento possono essere sintetizzati come mostrato in Figura 6. In particolare l'addestramento avviene con la seguente procedura:

¹²In particolare per la *cosine schedule* β_t è definito come $\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$. In questo caso:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos(\frac{t/T + s}{1 + s} \cdot \frac{\pi}{2})^2, \quad (18)$$

dove s è un piccolo offset pari a 0.008 [26].

¹³La definizione di $\tilde{\beta}_t$ è riportata nell'Equazione 10. Tuttavia si riporta anche qui per una lettura più agevole:

$$\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t. \quad (19)$$

Algorithm 1 Training	Algorithm 2 Sampling
<pre> 1: repeat 2: $x_0 \sim q(x_0)$ 3: $t \sim \text{Uniform}(\{1, \dots, T\})$ 4: $\epsilon \sim \mathcal{N}(0, I)$ 5: Take gradient descent step on $\nabla_{\theta} \ \epsilon - \epsilon_{\theta}(\sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon, t)\ ^2$ 6: until converged </pre>	<pre> 1: $x_T \sim \mathcal{N}(0, I)$ 2: for $t = T, \dots, 1$ do 3: $z \sim \mathcal{N}(0, I)$ if $t > 1$, else $z = 0$ 4: $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}} \epsilon_{\theta}(x_t, t) \right) + \sigma_t z$ 5: end for 6: return x_0 </pre>

Figure 6: Algoritmi di addestramento e campionamento dei DDPM. Questa Figura è tratta da [6].

- viene campionato un dato x_0 (per esempio un’immagine) dalla distribuzione dei dati reali;
- viene campionato un istante di tempo arbitrario t da una distribuzione uniforme compresa tra 1 e T ;
- viene campionato il rumore ϵ da una distribuzione Gaussiana;
- viene applicato il rumore ϵ al dato x_0 ottenendo il dato rumoroso x_t (che può essere calcolato per ogni istante di tempo arbitrario in base all’Equazione 6) tramite forward diffusion process;
- la rete neurale predice il rumore ϵ_{θ} in base al dato x_t e al tempo t ;
- viene calcolato il valore della loss function in base al rumore vero ϵ e a quello predetto ϵ_{θ} ;
- vengono aggiornati i pesi della rete neurale utilizzando il metodo di discesa del gradiente.

L’algoritmo di campionamento viene utilizzato per generare nuovi campioni (appartenenti alla distribuzione cercata) partendo da rumore Gaussiano puro. In particolare:

- viene campionato il rumore Gaussiano puro corrispondente al tempo T ;
- per ogni istante di tempo compreso fra T e 1:
 1. viene predetto il rumore ϵ_{θ} dell’immagine x_t tramite la rete neurale;
 2. viene rimosso il rumore predetto dall’immagine x_t ;
 3. viene calcolata una nuova quantità di rumore da aggiungere ($\sigma_t z$);
 4. viene prodotta una nuova immagine x_{t-1} leggermente meno rumorosa di x_t aggiungendo la nuova quantità di rumore;
- infine viene restituita l’immagine x_0 ottenuta dopo T passi di graduale rimozione del rumore.

Per i DDPM il numero di passi T deve essere alto al fine di (i) ottenere una distribuzione $q(x_T | x_0)$ prossima alla distribuzione Gaussiana e (ii) generare quindi immagini fedeli alla distribuzione dei dati reali. Tipicamente viene utilizzato un numero di passi T pari a 1000. Tuttavia poiché solitamente T rappresenta anche il numero di iterazioni del procedimento di campionamento, un alto valore di T si traduce anche in un costo computazionale elevato [27].

4.2 Denoising Diffusion Implicit Models

I DDIM sono stati presentati per la prima volta nel 2020 [7]. Traggono ispirazione dai DDPM. E se ne differenziano rendendo il forward diffusion process non Markoviano¹⁴. Le peculiarità principali dei DDIM sono le seguenti.

Forward diffusion process. Anzitutto come anticipato il forward diffusion process non soddisfa più la proprietà di Markov specificata nell’Equazione 1. Infatti viene definito un kernel di transizione che dipende (non più solo dall’ultimo valore calcolato, ma) anche dal valore iniziale x_0 . Più precisamente:

$$q(x_t | x_{t-1}, x_0) = \frac{q(x_{t-1} | x_t, x_0)q(x_t | x_0)}{q(x_{t-1} | x_0)}, \quad (26)$$

dove

$$q(x_{t-1} | x_t, x_0) = \mathcal{N}(\sqrt{\alpha_{t-1}}x_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \frac{x_t - \sqrt{\alpha_t}x_0}{\sqrt{1-\alpha_t}}, \sigma_t^2 I)[7]. \quad (27)$$

¹⁴L’uso dell’aggettivo “Diffusion” per modelli non Markoviani ha suscitato perplessità nella comunità scientifica. Infatti il processo di diffusione è tipicamente Markoviano. Tuttavia l’uso dell’aggettivo “Diffusion” può essere giustificato per la connessione che i DDIM presentano con i precedenti modelli di diffusione proposti, tra cui i DDPM [28].

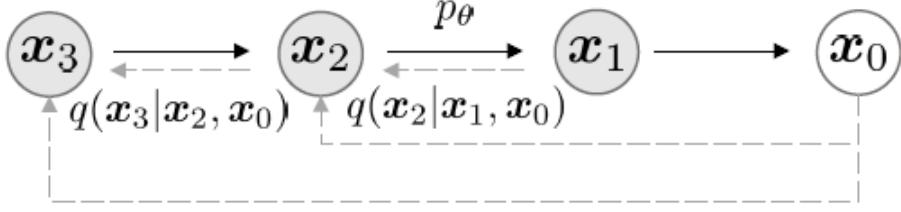


Figure 7: Rappresentazione grafica del processo diffusivo dei DDIM. Questa rappresentazione è tratta da [7].

Reverse diffusion process. Anche il reverse diffusion process viene ridefinito sulla base della ridefinizione del forward diffusion process. In particolare il kernel di transizione viene ridefinito come:

$$p_\theta(x_{t-1} | x_t) = \begin{cases} \mathcal{N}(f_\theta(x_t), \sigma_1^2 \mathbf{I}) & \text{se } t = 1 \\ q(x_{t-1} | x_t, f_\theta(x_t)) & \text{altrimenti} \end{cases} \quad (28)$$

dove

$$f_\theta(x_t) = \frac{(x_t - \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon_\theta(x_t))}{\sqrt{\bar{\alpha}_t}} [7]. \quad (29)$$

In questo caso $f_\theta(x_t)$ rappresenta la predizione del dato privo di rumore (x_0) sulla base di x_t . Il processo di diffusione può essere sintetizzato come mostrato in Figura 7.

Loss function. La loss function viene ridefinita come segue:

$$J_\theta(\epsilon_\theta) = \mathbb{E}_{x_{0:T} \sim q(x_{0:T})} \left[\log q(x_T | x_0) + \sum_{t=2}^T \log q(x_{t-1} | x_t, x_0) - \sum_{t=1}^T \log p_\theta(x_{t-1} | x_t) - \log p_\theta(x_T) \right] [7]. \quad (30)$$

Tuttavia è stato dimostrato che questa loss function è equivalente alla loss function L_{simple} definita per i DDPM con l'Equazione 24 [7].

Campionamento. Infine viene ridefinito anche il campionamento di x_{t-1} a partire da x_t . In particolare:

$$x_{t-1} = \underbrace{\sqrt{\bar{\alpha}_{t-1}} \left(\frac{x_t - \sqrt{1 - \bar{\alpha}_t} \epsilon_\theta(x_t)}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{predicted } x_0} + \underbrace{\sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2} \cdot \epsilon_\theta(x_t)}_{\text{direction pointing to } x_t} + \underbrace{\sigma_t \epsilon_t}_{\text{random noise}} [7]. \quad (31)$$

Quando σ_t è pari a 0 (o molto piccolo) per ogni t , il procedimento di campionamento diventa deterministico. In questo caso i campioni sono generati dalle variabili latenti (da x_T a x_0) con una procedura fissa.

Vantaggi. I vantaggi dei DDIM rispetto ai DDPM possono essere riassunti come segue:

- i DDIM generano campioni di qualità comparabile ai DDPM ma con una velocità fino a 100 volte superiore;
- i DDIM consentono di effettuare interpolazioni di immagini semanticamente significative nello spazio latente delle feature¹⁵;
- i DDIM utilizzano un processo di campionamento (non stocastico come i DDPM, ma) deterministico che permette di ricostruire i dati con un tasso di errore basso [7].

4.3 Noise Conditional Score Network

Le Noise Conditional Score Networks (NCSN) [8] vengono introdotte nel 2019 come miglioramento dei modelli generativi Score-Based (vedi Appendice A). I modelli generativi Score-Based si basano sulla stima e il campionamento

¹⁵L'interpolazione avviene con la seguente procedura. Anzitutto vengono campionate due distinte immagini x_T da una distribuzione Gaussiana. Dopodiché viene utilizzata un'interpolazione sferica lineare per generare immagini intermedie tra queste due immagini di partenza. Infine viene applicato il procedimento di campionamento dei DDIM per ottenere le corrispondenti immagini x_0 .

tramite (*Stein*) *Score* [29]. Lo *Score* rappresenta un vettore che punta nella direzione dove si ha più alta densità dei dati. Esso viene definito come $\nabla_x \log p(x)$ (vedi Equazione 40). La generazione si basa sull’addestramento di una rete neurale tramite Score-Matching [30] per apprendere il vettore dello *Score* a partire dai dati di train. Saranno prodotti campioni che si adattano alla distribuzione definita dal training set utilizzando la *dinamica di Langevin*: questo algoritmo opera muovendo gradualmente un dato random di partenza verso aree di alta densità, spostandosi lungo il vettore stimato per lo *Score*. I modelli generativi Score-Based vengono introdotti per cercare di superare le limitazioni dei metodi Likelihood-Based e delle GANs. Infatti, i metodi Likelihood-Based necessitano di architetture specializzate per definire un modello probabilistico da usare nell’addestramento. Le GANs, invece, presentano una procedura di addestramento instabile a causa del training “contradditorio”. Inoltre, la funzione obiettivo definita per le GANs non è utilizzabile per confrontare differenti modelli.

L’approccio basato sulla stima dello *Score* ha una serie di vantaggi: (*i*) la funzione obiettivo è trattabile per praticamente tutte le possibili parametrizzazioni della score network; (*ii*) la score network può essere ottimizzata senza addestramento adversarial; e (*iii*) la funzione obiettivo può essere utilizzata per comparare differenti modelli definiti sullo stesso dataset. Sperimentalmente, è stata dimostrata l’efficacia di questo approccio sui dataset MNIST, CelebA e CIFAR-10. Si è visto come gli esempi generati sono comparabili con quelli prodotti da moderni modelli Likelihood-Based o GANs. In particolare, su CIFAR-10 il modello ha ottenuto una *FID* (vedi Paragrafo 5.3) di 25.32.

Definizione. Analizzando i classici modelli generativi Score-Based (vedi appendice A), si è osservato che perturbando i dati con rumore Gaussiano casuale, la distribuzione dei dati diventa più facilmente trattabile dalla rete per la stima dello *Score*. Questo perché:

- I dati perturbati non risultano più confinati in regioni di bassa molteplicità. La stima dello score risulta così più precisa;
- Un grande rumore Gaussiano applicato ai dati originali ha come effetto il riempimento delle regioni a bassa densità. Il processo di Score-Matching avrà allora a disposizione più evidenze per migliorare la stima dello score;
- Utilizzando diversi livelli di rumore si può ottenere una sequenza di distribuzioni perturbate, che convergono verso la distribuzione reale. Si potrà allora migliorare la dinamica di Langevin andando a considerare non solo lo score della distribuzione finale, ma anche quello delle distribuzioni intermedie. Questo procedimento si ispira al Simulated Annealing [31].

A partire da questa intuizione, quello che viene proposto in [8, p. 5–7] è un miglioramento del metodo generativo Score-Based, basato su: (*i*) perturbazione dei dati utilizzando diversi livelli di rumore, e (*ii*) stima dello score rispetto a tutti i livelli di rumore. Quando si utilizzerà la dinamica di Langevin per generare campioni, si partirà con lo score relativo alla distribuzione con più alto livello di rumore, e gradualmente si andrà a ridurre il rumore, fino ad arrivare a considerare lo score della reale distribuzione dei dati. Questo aiuterà a trasferire gradualmente i benefici dati dai livelli più alti di rumore (come il riempimento delle regioni a bassa densità, quindi migliore stima dello score) sui livelli più bassi di rumore, dove i dati perturbati sono praticamente indistinguibili da quelli reali. In quanto segue si discuterà dell’architettura della rete per la stima dello *Score*, della funzione obiettivo considerata, e di come viene eseguita la dinamica di Langevin nell’ambito delle NCSN.

Architettura della Rete. Si consideri una sequenza di livelli di rumore $\{\sigma_i\}_{i=1}^L$, e si denoti con $q_\sigma(x)$ la distribuzione dei dati perturbata con livello di rumore $\sigma \in \{\sigma_i\}_{i=1}^L$. La sequenza di livelli di rumore deve essere scelta in modo tale che σ_1 sia abbastanza grande da permettere di evitare le difficoltà dei metodi Score-Based (vedi Appendice A), e σ_L sia abbastanza piccolo da rendere $q_{\sigma_L}(x) \sim q(x)$, quindi minimizzare l’effetto del rumore sulla distribuzione originale. L’obiettivo sarà quello di addestrare una rete neurale che permetta di stimare congiuntamente lo score di tutte le distribuzioni di dati perturbate, cioè $\forall \sigma \in \{\sigma_i\}_{i=1}^L : s_\theta(x, \sigma) \sim \nabla_x \log q_\sigma(x)$.

Definiamo $s_\theta(x, \sigma)$ come *Noise Conditional Score Network* (NCSN). Si noti che $s_\theta(x, \sigma) \in \mathbb{R}^D$ se $x \in \mathbb{R}^D$. Per quanto riguarda l’architettura della rete, visto che l’output di una NCSN $s_\theta(x, \sigma)$ ha stessa forma rispetto all’input x , si è presa ispirazione dalle architetture più di successo nell’ambito della predizione di immagini. In particolare, nell’esperimento proposto da [8] (vedi Paragrafo 5.2.3), il modello impiegato combina l’architettura di U-Net (vedi Sezione 3) con l’utilizzo di layer di convoluzione Dilated [32]. Inoltre, viene adottata la *Instance Normalization*, ispirandosi ai buoni risultati che questa ha ottenuto in altri task di generazione di immagini. Per maggiori dettagli si veda 5.2.3.

Apprendimento. Per l’addestramento di una NCSN si può utilizzare lo Score-Matching sia nella versione Denoising che nella versione Sliced (vedi Appendice A). In [8, p. 6] si propone l’utilizzo del Denoising Score-Matching, perché più veloce e per sua natura adatto alla stima dello score di una distribuzione perturbata. La distribuzione del rumore

viene definita come $q_\sigma(\tilde{x}, x) = \mathcal{N}(\tilde{x}|x, \sigma^2 I)$. Fissato il livello di rumore σ , la funzione obiettivo sarà:

$$l(\theta; \sigma) = \frac{1}{2} \mathbb{E}_{p_{data}(x)} \mathbb{E}_{\tilde{x} \sim \mathcal{N}(x, \sigma^2 I)} \left[\|s_\theta(\tilde{x}, \sigma) + \frac{\tilde{x} - x}{\sigma^2}\|_2^2 \right], \quad (32)$$

Combinando tale funzione per ogni livello di rumore $\sigma \in \{\sigma_i\}_{i=1}^L$, si ottiene una funzione obiettivo unificata del tipo:

$$\mathcal{L}(\theta; \{\sigma_i\}_{i=1}^L) = \frac{1}{L} \sum_{i=1}^L \lambda(\sigma_i) l(\theta; \sigma_i). \quad (33)$$

dove $\lambda(\sigma_i) > 0$ è un termine regolatore dipendente da σ_i . Il valore di $\lambda(\cdot)$ può essere sia fissato a priori che considerato come parametro nell'addestramento. Solitamente viene posto pari a $\lambda(\sigma_i) = \sigma^2$ [8, p. 6]. Si noti che la funzione obiettivo così definita non richiede addestramento *adversarial*, né funzioni surrogate, per poter essere minimizzata. Inoltre, per $\lambda(\cdot)$ e $\{\sigma_i\}_{i=1}^L$ fissati, può essere utilizzata per comparare differenti NCSN. Per come viene definita la funzione obiettivo, si può dire che $s_{\theta^*}(x, \sigma)$ la minimizza se e solo se si ha che $s_{\theta^*}(x, \sigma_i) = \nabla_x \log q_{\sigma_i}(x) \quad \forall i \in \{1, 2, \dots, L\}$.

Inferenza. Una volta addestrata la NCSN, per eseguire il campionamento viene proposta una versione *annealed* della dinamica di Langevin, ispirata al Simulated Annealing [31]. Come mostrato in Figura 8 (sinistra), l'algoritmo parte da una istanza estratta da una distribuzione fissata a priori, che può essere ad esempio una distribuzione uniforme. A questo punto si esegue la dinamica di Langevin per campionare dalla distribuzione $q_{\sigma_1}(x)$ con dimensione dello step α_1 . Allo step successivo si campiona considerando la distribuzione $q_{\sigma_2}(x)$, partendo dall'esempio prodotto al passo precedente. Si utilizza una dimensione dello step α_2 minore rispetto allo step precedente. Si va avanti in questo modo fino a che, all'ultima iterazione, si esegue la dinamica di Langevin per campionare dalla distribuzione $q_{\sigma_L}(x)$, che sarà abbastanza vicina alla distribuzione reale per $\sigma_L \sim 0$.

Visto che le distribuzioni $\{q_{\sigma_i}\}_{i=1}^L$ sotto tutte perturbate da rumore Gaussiano, queste saranno in grado di coprire tutto lo spazio considerato. Quindi, il loro Score sarà ben definito, evitando le difficoltà introdotte dall'ipotesi della Molteplicità. In particolare, per σ_1 abbastanza grande, le regioni a bassa densità di dati in $q_{\sigma_1}(x)$ saranno molto poche. Questo renderà la stima dello Score più accurata, e l'esecuzione della dinamica di Langevin più rapida. Inoltre, assumendo che la dinamica di Langevin produca un buon campione per $q_{\sigma_1}(x)$ si sarà abbastanza certi che questo campione risiederà in una regione ad alta densità anche in $q_{\sigma_2}(x)$, ipotizzando il fatto che le due distribuzioni siano solamente leggermente differenti. Questo vuol dire che il campione generato per $q_{\sigma_1}(x)$ sarà un buon punto di partenza per il campionamento su $q_{\sigma_2}(x)$. L'idea è proprio questa: partire dall'applicazione della dinamica di Langevin ad una distribuzione altamente perturbata, che permetterà di ottenere quasi sicuramente un campione appartenente ad una regione di alta densità. Successivamente si sposterà gradualmente questo campione verso distribuzioni meno perturbate, avendo come vantaggio il fatto che il campione generato per la distribuzione precedente molto probabilmente giacerà in una regione ad alta densità anche nella distribuzione successiva.

Per dimostrare l'efficacia di questa versione annealed della dinamica di Langevin, in [8] viene proposto un esperimento. L'obiettivo è riuscire a campionare da una distribuzione Gaussiana mista del tipo $p_{data} = \frac{1}{5}\mathcal{N}((-5, -5), I) + \frac{4}{5}\mathcal{N}((5, 5), I)$. In Figura 25 (c) si può vedere il risultato ottenuto utilizzando la versione annealed della dinamica di Langevin, ponendo $L = 10$, $\sigma_1 = 10$ e $\sigma_{10} = 0.1$. Si nota che la distribuzione dei campioni generati si adatta bene alla distribuzione reale, mostrata in Figura 25 (a), a differenza di quanto ottenuto dalla versione classica della dinamica.

Conclusioni. Sono diversi gli aspetti strutturali che differenziano l'approccio Score-Based [8] dagli altri metodi generativi, tra cui:

- Non si ha la necessità di campionare da una catena di Markov durante l'addestramento. L'unica cosa che serve predire è lo score. Questa differenza rende tale approccio più facilmente scalabile all'addestramento di reti profonde;
- La fase di addestramento è del tutto slegata da quella di campionamento. Questo vuol dire che è possibile definire una qualunque combinazione di metodi per la stima dello score (denoising score matching, sliced score matching etc.) e metodi per il campionamento basati sul gradiente (dinamica di Langevin, Hamiltonian Monte Carlo [33] etc.), a differenza di altri modelli strettamente legati ad una specifica catena di Markov.

L'approccio Score-Based classico (vedi Appendice A) ai modelli generativi, basato su una prima fase di stima del gradiente della densità dei dati tramite Score-Matching, e una seconda fase di campionamento tramite dinamica di Langevin (o simili), presenta molte sfide da gestire. In [8] viene proposta come soluzione a queste sfide l'utilizzo di una NCSN per la stima dello score, e di una versione annealed della dinamica di Langevin. Questo approccio non richiede

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^L, \epsilon, T$.

- 1: Initialize \tilde{x}_0
- 2: **for** $i \leftarrow 1$ to L **do**
- 3: $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ α_i is the step size.
- 4: **for** $t \leftarrow 1$ to T **do**
- 5: Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
- 6: $\tilde{x}_t \leftarrow \tilde{x}_{t-1} + \frac{\alpha_i}{2} \mathbf{s}_\theta(\tilde{x}_{t-1}, \sigma_i) + \sqrt{\alpha_i} \mathbf{z}_t$
- 7: **end for**
- 8: $\tilde{x}_0 \leftarrow \tilde{x}_T$
- 9: **end for**
- 10: **return** \tilde{x}_T

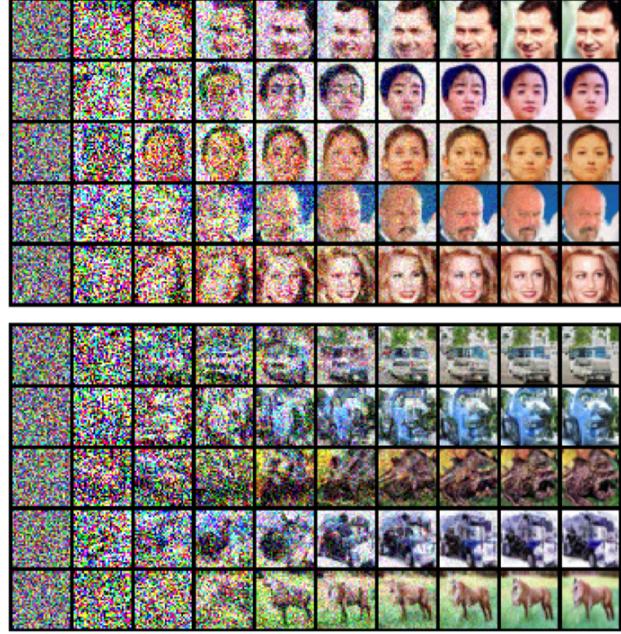


Figure 8: *Sinistra*: pseudo-codice della versione annealed della dinamica di Langevin. *Destra*: campioni intermedi prodotti dalla applicazione di questo algoritmo ai dataset CelebA e CIFAR-10. Queste immagini sono tratte da [8, p. 6-7].

addestramento "contraddittorio", non richiede campionamento durante l'addestramento, e non necessita di architetture particolari. Sperimentalmente, si è dimostrato che tale approccio è in grado di generare immagini di alta qualità, con un valore di FID precedentemente ottenibile solo con l'impiego dei migliori modelli Likelihood-Based o GAN.

Anche le NCSN vengono considerate Diffusion Model. Infatti, come processo di forward diffusion si ha l'addestramento della rete per la stima dello Score rispetto ai diversi livelli di rumore. Come processo di reverse diffusion si ha il campionamento tramite dinamica di Langevin [2].

4.4 Score-Based Generative Model tramite Equazioni Differenziali Stocastiche

I modelli fino ad ora analizzati prevedono una prima fase di corruzione incrementale dei dati di training —aumentando a mano a mano il rumore per arrivare ad una distribuzione nota— e una seconda fase di apprendimento della modalità con cui invertire questa corruzione. Si ottengono dunque dei modelli generativi che a partire da sequenze randomiche di valori (rumore), sono in grado di produrre dati che ben si amalgano con la distribuzione a cui appartengono i dati di training.

Considerando due tra i modelli generativi probabilistici di maggiore successo è possibile trovare:

- Modelli di *Score Matching* tramite dinamica di *Langevin* (SMLD) [8]: questo metodo prevede la stima di uno *score* (definito dall'Equazione 40) ad ogni scala di rumore, e il successivo uso della dinamica di *Langevin* per effettuare il campionamento da una sequenza di scale di rumore decrescente durante la fase di generazione (vedi Appendice A);
- *Denoising Diffusion Probabilistic Modeling* (DDPM): questo metodo prevede l'addestramento di una sequenza di modelli probabilistici in grado di invertire ogni passo della prima fase di corruzione, sfruttando la conoscenza delle distribuzioni inverse per rendere l'addestramento trattabile. Si noti che, nel caso di spazi degli stati continui, la funzione obiettivo definita per i DDPM già implicitamente calcola uno *score* ad ogni scala di rumore.

Entrambi questi metodi prevedono la computazione di score, quindi con *Score-Based Generative Models* (SBGM) si fa riferimento alla categoria che comprende questi due modelli. In particolare, in questa sezione si analizza come è

possibile definire nuovi metodi di campionamento ed estendere le capacità degli SGBM, tramite l'utilizzo di equazioni differenziali stocastiche (*Stochastic Differential Equations*, SDE¹⁶). Questa generalizzazione è denominata SBGM-SDE.

Invece di perturbare i dati con un numero finito di distribuzioni di rumore, si considera uno spazio continuo di distribuzioni, che evolvono nel tempo secondo un certo processo di diffusione: questo processo trasforma progressivamente un dato in rumore randomico ed è descritto da una SDE che non dipende dal dato e non è legata a parametri da addestrare. Invertendo questo processo, è possibile gradualmente trasformare sequenze di valori randomici in dati che ben si adattano alla distribuzione definita dal training set.

È cruciale che il processo inverso soddisfi una SDE *reverse-time*, derivata dalla SDE *forward* dato lo *score* calcolato come visto nell'Equazione 40. È quindi possibile approssimare la SDE *reverse-time* tramite l'addestramento di una rete neurale *time-dependent* per la stima degli *score*, e successivamente produrre campioni andando a risolvere questa SDE tramite risolutori noti. L'idea viene riassunta in Figura 9.

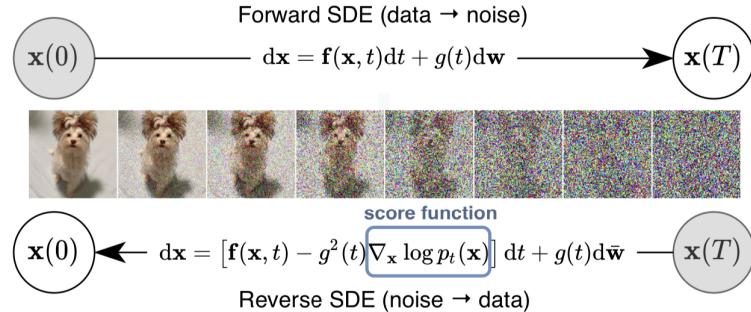


Figure 9: Risolvere una *reverse-time* SDE opportunamente definita corrisponde a definire un modello generativo Score-Based. Questa immagine è tratta da [9].

Caratteristiche. Le principali caratteristiche di questo metodo sono:

- **Campionamento flessibile:** è possibile utilizzare qualunque algoritmo *general-purpose* per la risoluzione di SDE in modo da effettuare il campionamento. In aggiunta, è possibile utilizzare ulteriori metodi specifici come i *predictor-corrector samplers*.
- **Generazione Controllabile:** è possibile modulare il processo di generazione dei campioni, condizionando il processo tramite informazioni non disponibili durante l'addestramento. Questo permette l'applicazione degli SBGM-SDE per task come la generazione *class-conditional*, *image inpainting* e colorizzazione senza la necessità di effettuare più addestramenti.
- **Framework Unificato:** il metodo analizzato fornisce una modalità unificata per esplorare e adattare diverse SDE per migliorare le prestazioni degli SBGM. I metodi definiti da SMLD e DDPM possono essere visti come specializzazioni di questo framework tramite la discretizzazione di due distinte SDE.

Di seguito vengono descritti i concetti base degli SBGM-SDE. Uno schema che li raccoglie schematicamente è mostrato nella Figura 10.

4.4.1 Generazione del rumore

I modelli SMLD e DDPM prevedono la perturbazione dei dati con più scale di rumore, mentre gli SBGM-SDE generalizzano questo concetto introducendo una scala infinita di rumore.

Si vuole costruire un processo di diffusione $\{\mathbf{x}(t)\}_{t=0}^T$, dove t è una variabile continua nel range $[0, T]$ e tale per cui $\mathbf{x}(0) \sim p_0$ e $\mathbf{x}(T) \sim p_T$.

In altre parole, p_0 è la distribuzione dei dati di training e p_T è una distribuzione nota a priori. Quest'ultima non contiene alcuna informazione riguardo p_0 e può essere ad esempio una distribuzione Gaussiana con media e varianza predeterminate.

Il processo di diffusione è dunque modellato come la soluzione di una SDE:

$$dx = f(\mathbf{x}, t)dt + g(t)d\mathbf{w}. \quad (34)$$

¹⁶Una equazione differenziale stocastica è una equazione differenziale in cui uno o più termini sono processi stocastici, portando la soluzione ad essere anch'essa un processo stocastico (vedi sezione 2.1).

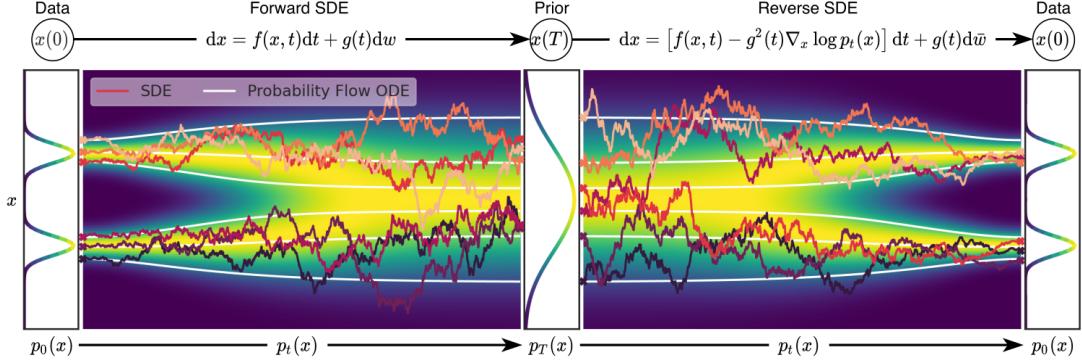


Figure 10: Schema di funzionamento degli SBGM tramite SDE. Vengono mappati i dati su una distribuzione di rumore nota tramite l'uso di una SDE e si definisce un modello generativo con la SDE opposta *reverse-time*. Quest'ultima si ottiene stimando lo score (Equazione 40). Questa immagine è tratta da [9].

dove \mathbf{w} è il processo di Wiener¹⁷ standard, $f(\cdot, t) : \mathbb{R}^d \rightarrow \mathbb{R}^d$ è una funzione detta "coefficiente di drift di $\mathbf{x}(t)$ " e $g(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$ è una funzione detta "coefficiente di diffusione di $\mathbf{x}(t)$ ".

Con $p_t(\mathbf{x})$ si denota la densità di probabilità di $\mathbf{x}(t)$.

4.4.2 Generazione dei campioni

A partire da campioni di $\mathbf{x}(T) \sim p_T$ si possono ottenere dei campioni $\mathbf{x}(0) \sim p_0$ con il processo inverso a quello di degradazione. Un risultato notevole definito in [34] è che l'opposto di un processo di diffusione è anch'esso un processo di diffusione che procede indietro nel tempo, ed è descritto da una *reverse-time* SDE:

$$dx = [f(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})] dt + g(t)d\bar{\mathbf{w}}. \quad (35)$$

dove $\bar{\mathbf{w}}$ è il processo di Wiener in cui il tempo fluisce al contrario da T a 0 e dt è uno step infinitesimale nel tempo (negativo). Una volta che lo score di ogni distribuzione marginale, $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, è noto per ogni t , è possibile derivare il processo di diffusione inverso e simularlo per effettuare dei campionamenti da p_0 .

4.4.3 Stima degli score

Lo score di una distribuzione può essere stimato addestrando un modello score-based su campioni score-matching [30]. Per effettuare la stima di $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ si può addestrare un modello *score-based* e *time-dependent* $s_{\theta}(x, t)$:

$$\theta^* = \underset{x}{\operatorname{argmin}} \mathbb{E}_t \{ \lambda(t) \mathbb{E}_{\mathbf{x}(0)} \mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)} [\| s_{\theta}(\mathbf{x}(t), t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t)|\mathbf{x}(0)) \|_2^2] \}. \quad (36)$$

dove $\lambda : [0, T] \rightarrow \mathbb{R}_{>0}$ è una funzione di peso positiva; t è campionato uniformemente nell'intervallo $[0, T]$; $\mathbf{x}(0) \sim p_0(\mathbf{x})$ e $\mathbf{x}(t) \sim p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))$.

Con una quantità di dati sufficiente e con una capacità del modello adeguata viene garantita la soluzione ottimale per la suddetta equazione, denotata con $s_{\theta^*}(\mathbf{x}, t)$.

4.4.4 Risoluzione della SDE inversa

Dopo aver addestrato un modello *time-dependent* e *score-based* s_{θ} lo si può usare per costruire la *reverse-time* SDE utilizzata per generare campioni. A questo scopo possono essere usati diversi risolutori general-purpose, come ad esempio il metodo di Euler–Maruyama.

4.4.5 Campionamenti *predictor-corrector*

Si può notare che nel dominio analizzato non si fa uso di SDE generiche, ma si possiedono informazioni ulteriori che possono essere sfruttate per migliorare le soluzioni. In particolare è possibile utilizzare degli approcci *Markov chain Monte Carlo* (MCMC) *score-based* per correggere le soluzioni dei risolutori di SDE impiegati. Con questa tecnica ad

¹⁷Un processo di Wiener (o moto browniano) è un processo stocastico in tempo continuo. In diversi ambiti applicativi è usato per rappresentare l'integrale del rumore bianco.

ogni step il risolutore di SDE fornisce una stima del campione per il prossimo step ("predictor") e un approccio MCMC corregge la distribuzione marginale del campione predetto ("corrector").

Una schematizzazione di questo algoritmo è disponibile nella figura 11.

Algorithm Predictor-Corrector (PC) sampling

Require:

N : Number of discretization steps for the reverse-time SDE

M : Number of corrector steps

```

1: Initialize  $\mathbf{x}_N \sim p_T(\mathbf{x})$ 
2: for  $i = N - 1$  to 0 do
3:    $\mathbf{x}_i \leftarrow \text{Predictor}(\mathbf{x}_{i+1})$ 
4:   for  $j = 1$  to  $M$  do
5:      $\mathbf{x}_i \leftarrow \text{Corrector}(\mathbf{x}_i)$ 
6: return  $\mathbf{x}_0$ 
```

Figure 11: Schema dell'algoritmo *predictor-corrector*.

5 Esperimenti

In questa sezione verranno descritti gli esperimenti svolti. Per questi esperimenti è stato utilizzato il dataset CelebA-HQ. In particolare, questi esperimenti hanno avuto per oggetto l'addestramento e la generazione di immagini attraverso i Diffusion Models presentati nei paragrafi precedenti. Le performance dei modelli verranno confrontate in base alla metrica FID (vedi Paragrafo 5.3), nonché al tempo di addestramento e di generazione di campioni. Infine verranno illustrati i risultati di operazioni di *similarity retrieval* per fornire una ulteriore valutazione qualitativa delle immagini generate.

Non si è potuto confrontare i modelli NCSN e SBGM-SDE con DDPM e DDIM perché:

- Sia per NCSN che SBGM-SDE, a causa della scarsa potenza computazionale a disposizione non è stato possibile definire un modello che ottenessi risultati apprezzabili. Si è invece deciso di considerare modelli pre-addestrati;
- Il modello NCSN considerato è pre-addestrato su una versione di CelebA diversa da quella considerata dagli altri esperimenti. Per tale motivo le NCSN non saranno oggetto di confronto con gli altri modelli;
- Il modello SBGM-SDE è pre-addestrato sul dataset CelebA-HQ (vedi Paragrafo 5.1). Nonostante ciò, non è possibile confrontarlo con DDPM e DDIM perché i passi di inferenza dettati dallo scheduler sono notevolmente maggiori.

5.1 Dataset CelebA-HQ

Il dataset utilizzato per gli esperimenti è stato CelebA-HQ. Questo dataset è stato recuperato da Kaggle¹⁸. In particolare:

- il dataset è composto da immagini di volti di celebrità;
- le immagini sono a colori e hanno dimensioni 256x256;
- le immagini presenti nel dataset sono 30000 e sono denominate con un numero crescente da 0 a 29999;
- le immagini non presentano né attributi né label;
- l'ordine delle immagini nel dataset non è significativo.

Su questo dataset sono state svolte alcune operazioni di preprocessing limitatamente agli esperimenti in cui si è proceduto all'addestramento dei modelli. In particolare a causa delle limitate risorse computazionali a disposizione è stato necessario (*i*) ridurre il dataset alle sole prime 3000 immagini e (*ii*) ridurre le dimensioni delle immagini a 128x128.

Nella Figura 12 si riportano alcuni esempi di immagini del dataset.

¹⁸<https://www.kaggle.com/datasets/badasstechie/celebahq-resized-256x256>.



Figure 12: Esempi di immagini tratte dal dataset CelebA-HQ.

5.2 Modelli analizzati

I modelli analizzati negli esperimenti sono: DDPM, DDIM, NCSN e SBGM-SDE. Nei prossimi paragrafi verranno specificati i dettagli implementativi.

5.2.1 Denoising Diffusion Probabilistic Models

I DDPM sono stati implementati sfruttando le librerie sui Diffusion Models offerte da Hugging Face [35]. In particolare per l’implementazione si è fatto riferimento alla documentazione e alle guide ivi messe a disposizione.

Architettura della rete. L’architettura della rete U-Net utilizzata segue le specifiche menzionate in [6]. Rispetto alla struttura descritta nella Sezione 3 risulta incrementata la profondità della rete. Inoltre trovano applicazione layer di normalizzazione di gruppo e attention layer. Infine come funzione di attivazione è stata utilizzata la funzione SiLU¹⁹.

Preprocessing. I valori delle immagini sono stati scalati nell’intervallo $[-1, 1]$ analogamente a quanto effettuato negli esperimenti menzionati in [6].

Iperparametri. Per gli esperimenti in cui si è proceduto all’addestramento dei DDPM sono state utilizzate 50 epoche di addestramento. Per i primi 500 passi dell’addestramento è stata utilizzata una fase di *warm-up* del learning rate che lo ha portato a crescere linearmente da un valore iniziale di 0.00004 al valore di 0.0001. Dopodiché il learning rate è stato decrementato linearmente nei restanti passi dell’addestramento. Come algoritmo di ottimizzazione è stato utilizzato *Adam*. La dimensione del batch è stata fissata a 16.

In merito agli iperparametri più attinenti ai DDPM si precisa quanto segue. Anzitutto sono stati effettuati esperimenti utilizzando sia la *linear schedule* della varianza menzionata in [6] sia la *cosine schedule* menzionata in [26]. Inoltre sono stati effettuati esperimenti utilizzando sia un campionamento con 1000 passi di inferenza sia un campionamento con 50 passi di inferenza.

Pre-addestramento. Per alcuni esperimenti di campionamento è stato utilizzato un modello di DDPM pre-addestrato da Google e reso disponibile da Hugging Face²⁰. Questo modello è stato addestrato con una *linear schedule*.

5.2.2 Denoising Diffusion Implicit Models

Anche per l’implementazione dei DDIM sono state utilizzate le librerie di Hugging Face [35].

Architettura della rete. L’architettura della rete U-Net utilizzata per i DDIM è la stessa utilizzata per i DDPM.

Preprocessing. Anche per l’addestramento dei DDIM è stato effettuato il preprocessing menzionato nel Paragrafo 5.2.1.

¹⁹SiLU è l’abbreviazione di Sigmoid Linear Unit. Questa funzione è definita come:

$$\text{SiLU}(x) = x \cdot \sigma(x), \quad (37)$$

dove $\sigma(x)$ è la funzione sigmoide.

²⁰<https://huggingface.co/google/ddpm-celebahq-256>.

Iperparametri. Gli iperparametri utilizzati per l’addestramento dei DDIM sono gli stessi menzionati per i DDPM. Anche in questo caso sono stati effettuati esperimenti con diverse *variance schedule* e con diversi passi di inferenza per il campionamento.

Pre-addestramento. Per alcuni esperimenti di campionamento è stato utilizzato il modello di DDPM pre-addestrato da Google già menzionato. Ora, l’utilizzo di un modello pre-addestrato di DDPM non è incompatibile con il campionamento tramite DDIM. Infatti le loss function delle due tipologie di modelli sono equivalenti, come già specificato al Paragrafo 4.2. E anche in letteratura sono riportati esperimenti di campionamento tramite DDIM per i quali vengono utilizzati modelli pre-addestrati di DDPM [7].

5.2.3 Noise Conditional Score Networks

Per l’implementazione di una NCSN si è considerato il codice definito dagli autori di [8]²¹. In particolare, si è fatto riferimento al modello pre-addestrato sul dataset CelebA [36]. La versione di CelebA considerata si compone di 202.599 immagini di dimensione 178x218. Queste vengono ritagliate al centro, ottenendo immagini di dimensione 140x140. Infine, vengono ridimensionate in immagini 32x32, e i loro pixel vengono riscalati nel range [0, 1], per velocizzare l’addestramento. Il campionamento, a sua volta, produrrà immagini di dimensione 32x32.

Architettura della Rete. L’architettura della NCSN utilizzata presenta tre componenti fondamentali: (i) **Instance normalization**, utilizzata al posto di batch normalization [37]²²; (ii) **Convoluzione Dilated**, utilizzata in tutti i livelli di subsampling con un fattore di dilatazione pari a 2 [32]²³; e (iii) **U-Net**: si è considerata l’architettura di RefineNet [38], moderna variazione di U-Net. Si sono sostituiti tutti i livelli di Max Pooling con livelli di Average Pooling, perché Average Pooling si dimostra in generale migliore nella generazione di immagini più regolari. Come funzione di attivazione si è utilizzata la ELU²⁴.

Setup. Come sequenza geometrica di deviazioni standard $\{\sigma_i\}_{i=1}^L$, viene considerato $L = 10$, $\sigma_1 = 1$, e $\sigma_{10} = 0.01$. Si noti che un rumore Gaussiano con $\sigma = 0.01$ è praticamente invisibile ad occhio nudo. Per quanto riguarda la dinamica di Langevin annealed (vedi Figura 8), viene considerato come numero di iterazioni per l’esecuzione di ogni dinamica un $T = 100$, e una dimensione dello step $\epsilon = 2 * 10^{-5}$. Come campione di partenza x_0 viene considerato del rumore uniforme casuale. Il modello viene addestrato utilizzando *Adam* come algoritmo di ottimizzazione. Viene considerato un tasso di apprendimento di 0.001, per un totale di 200.000 iterazioni. La dimensione dei batch è stata fissata a 128.

5.2.4 Score-Based Generative Models attraverso SDE

Il modello SBGM-SDE è stato anch’esso testato tramite le librerie sui Diffusion Models offerte da Hugging Face [35]. Per questa famiglia di modelli sono stati svolti esperimenti di inferenza, tramite un modello pre-addestrato da Google e disponibile sull’hub di Hugging Face²⁵. Il modello è stato trattato come una *black box* per la generazione dei campioni.

Architettura della rete. L’architettura usata è una U-Net analoga a quelle precedentemente descritte. La dimensione dei sample è di 256x256 e sono presenti sette blocchi di layer riduttivi e sette blocchi di layer espansivi. La tipologia di modello appartiene alla categoria *predictor-corrector* definita in 4.4.5.

Iperparametri. Il modello è caratterizzato da un numero molto elevato di passi di inferenza: 2000. Ciò non permette un confronto equo con i modelli preaddestrati DDPM e DDIM in quanto essi sono stati addestrati per funzionare

²¹<https://github.com/ermongroup/ncsn>.

²²Instance Normalization, a differenza di Batch Normalization, agisce sulle singole istanze. Batch Normalization, per dimensione dei batch abbastanza elevata, permette alla rete di convergere più velocemente. Questo perché Batch Normalization considera gruppi di istanze, quindi è in grado di ottenere stime più accurate per media e varianza del gruppo. Se, però, non si dispone di abbastanza potenza computazionale, quindi la dimensione dei singoli batch è limitata, in generale risulta migliore l’approccio Instance Normalization.

²³La Convoluzione Dilated espande il filtro da applicare all’input, andandoci ad inserire degli spazi vuoti. Questa tecnica permette di avere una visione globale dell’immagine, ma utilizzando meno parametri. Inoltre, è molto efficiente.

²⁴La ELU (Exponential Linear Unit) è una funzione di attivazione che, a differenza della ReLU, può produrre valori negativi. Viene così definita:

$$f(x) = \begin{cases} x, & \text{se } x > 0 \\ \alpha(\exp(x) - 1), & \text{se } x \leq 0 \end{cases} \quad \text{con } \alpha > 0 \quad (38)$$

²⁵<https://huggingface.co/google/ncsnpp-celebahq-256>.

con un minor quantitativo di passi di inferenza. Gli altri iperparametri (come ad esempio il numero delle epoche di addestramento) non sono disponibili, in quanto non forniti dagli autori del modello pre-addestrato.

5.3 Fréchet Inception Distance (FID)

Per confrontare le prestazioni ottenute dai diversi modelli è stata utilizzata una metrica di valutazione oggettiva. In particolare si è scelto di utilizzare la *Fréchet Inception Distance* (in seguito: FID) in quanto tipicamente impiegata per valutare le prestazioni di modelli generativi [39, p. 11-15]. Il motivo di questo impiego risiede nel fatto che la FID permette di valutare sia la fedeltà dei campioni generati rispetto alla distribuzione dei dati reali, sia la diversità all'interno dei campioni generati [23].

Definizione. La definizione della FID si basa sulla nozione di distanza di Fréchet [40, 41], utilizzata per misurare la somiglianza tra due curve²⁶. La FID viene definita come la distanza di Fréchet $d(\cdot, \cdot)$ tra una Gaussiana con media e covarianza (m, C) , ottenuta dalla distribuzione dei campioni generati $p(\cdot)$, e una Gaussiana con media e covarianza (m_w, C_w) , ottenuta dalla distribuzione dei dati reali $p_w(\cdot)$. La FID è calcolata come:

$$d^2((m, C), (m_w, C_w)) = \|m - m_w\|_2^2 + Tr(C + C_w - 2(CC_w)^{1/2}), \quad (39)$$

dove la funzione Tr restituisce la traccia di una matrice quadrata, cioè la somma degli elementi sulla sua diagonale. Poiché la FID misura la distanza tra la distribuzione dei dati reali e la distribuzione dei dati generati, quanto più il suo valore sarà alto tanto più le due distribuzioni saranno dissimili tra loro.

Il motivo per cui il termine “Inception” compare nel nome della FID deriva dalle modalità concrete di calcolo di questa distanza. In particolare viene utilizzato l'ultimo layer della rete Inception-V3 pre-addestrata su ImageNet per estrarre vettori di feature dal dataset di dati reali e dal dataset generato. E su questi vettori di feature vengono calcolate la media e la covarianza utilizzate per il calcolo della FID [27].

Per il calcolo della FID si è impiegata la libreria *pytorch-fid*²⁷, riadattamento in PyTorch dell'algoritmo ufficiale definito in [39].

A livello pratico, si vuole far notare che nei diversi esperimenti la FID è stata calcolata mettendo a confronto le istanze del dataset di partenza con un centinaio di campioni generati. Solitamente viene consigliato di utilizzare al minimo 2048 istanze per ognuno dei dataset. Questo in modo da ottenere dall'ultimo layer di Inception-V3 (che ritorna 2048 feature) una stima accurata. A causa della scarsa potenza computazionale a disposizione non è stato possibile generare così tanti campioni per ogni esperimento. La FID ottenuta su ogni esperimento, quindi, è da considerare come una stima approssimativa. Comunque, i valori ottenuti sono sembrati coerenti con valori di FID ottenuti da altri modelli generativi sullo stesso dataset [39, Appendice A1].

Confronto con Inception Score. La FID rappresenta un miglioramento dell'*Inception Score* nella valutazione dei modelli generativi [43]. Infatti, mentre l'*Inception Score* non considera la distribuzione definita dagli esempi reali a confronto con la distribuzione definita dagli esempi generati, la FID viene pensata proprio come distanza tra la distribuzione dei dati reali e la distribuzione dei dati generati.

Inoltre la FID risulta più consistente rispetto all'*Inception Score* nella valutazione di livelli crescenti di rumore nelle immagini. Questa maggiore consistenza della FID può essere dimostrata empiricamente [39]. In particolare la dimostrazione è la seguente:

- anzitutto si procede alla degradazione di immagini con diversi tipi di rumore e con diversi livelli di intensità per ciascun tipo di rumore;
- dopodiché si procede alla misurazione della FID (tra le immagini originali e quelle degradate) e dell'*Inception Distance* (in seguito: IND)²⁸ per ogni tipo di rumore e per ogni livello di intensità di ciascun rumore;
- infine si verifica quale metrica coglie meglio l'incremento del livello di rumore nelle immagini.

I risultati di questa dimostrazione empirica sono mostrati in [39]. In particolare emerge chiaramente come la FID colga meglio l'incremento dei livelli di rumore rispetto all'IND.

²⁶Si immagini una persona che attraversa un percorso curvo di lunghezza finita mentre porta a spasso il proprio cane al guinzaglio. Il cane attraversa un percorso curvo di lunghezza finita separato. L'obiettivo è fare in modo che i due riescano a percorrere ognuno il suo percorso fino alla fine, mantenendo il guinzaglio sempre allentato. Quello che possono fare è variare la propria velocità, ma non possono mai tornare indietro. La distanza di Fréchet tra le due curve è la lunghezza minima sufficiente per il guinzaglio affinché entrambi possano attraversare le loro curve dall'inizio alla fine. Questa definizione è simmetrica rispetto alle due curve [42].

²⁷<https://github.com/mseitzer/pytorch-fid>

²⁸Con IND si intende la trasformazione dell'*Inception Score* in una distanza, in modo tale da poterlo comparare con la FID. In particolare, la IND viene calcolata come $IND = m - InceptionScore$, con m numero di esempi considerati [39].

5.4 Analisi

5.4.1 Confronto addestramento DDPM e DDIM

Il primo confronto ha riguardato l’addestramento di DDPM e DDIM con diverse *variance schedule*. Tutti i modelli confrontati sono stati addestrati per 50 epochhe sul dataset CelebA-HQ ridotto a 3000 immagini scalate alle dimensioni 128x128. In particolare sono stati confrontati: (i) DDPM addestrato con *linear schedule*; (ii) DDPM addestrato con *cosine schedule*; (iii) DDIM addestrato con *linear schedule*; e (iv) DDIM addestrato con *cosine schedule*. L’oggetto del confronto ha riguardato la FID, il tempo di addestramento e il tempo di generazione di 100 campioni utilizzando il modello addestrato. In questo caso per il campionamento sono stati utilizzati 1000 passi per DDPM e 50 per DDIM. Gli esperimenti per questo confronto sono stati condotti utilizzando Google Colaboratory [44]. I risultati del confronto sono mostrati nella Tabella 1.

Table 1: Confronto addestramento DDPM e DDIM

Modello	FID	Tempo addestramento	Tempo campionamento
DDPM linear	99.802	138 minuti	80 minuti
DDPM cosine	115.659	140 minuti	75 minuti
DDIM linear	103.322	127 minuti	4 minuti
DDIM cosine	113.433	125 minuti	4 minuti

FID. Il modello che ha ottenuto la FID migliore è DDPM con *linear schedule*. Tuttavia la FID registrata da DDIM con *linear schedule* è più alta di soli 4 punti. Pertanto il peggioramento è modesto. Inoltre sia per i DDPM che per i DDIM si nota come la *linear schedule* abbia portato a risultati migliori rispetto alla *cosine schedule*. Questo risultato non contraddice la tesi avanzata in letteratura secondo cui la *linear schedule* sarebbe sub-ottimale rispetto alla *cosine schedule* [26]. Infatti questa tesi riguarda solo le risoluzioni 64x64 e 32x32, mentre in questo caso le immagini del dataset avevano dimensioni 128x128.

Tempo di addestramento. Il modello con il minor tempo di addestramento è stato DDIM con *cosine schedule*. Tuttavia la differenza rispetto al tempo di addestramento di DDIM con *linear schedule* è poco significativa. In ogni caso i tempi di addestramento dei modelli DDPM sono stati superiori rispetto a quelli dei modelli DDIM.

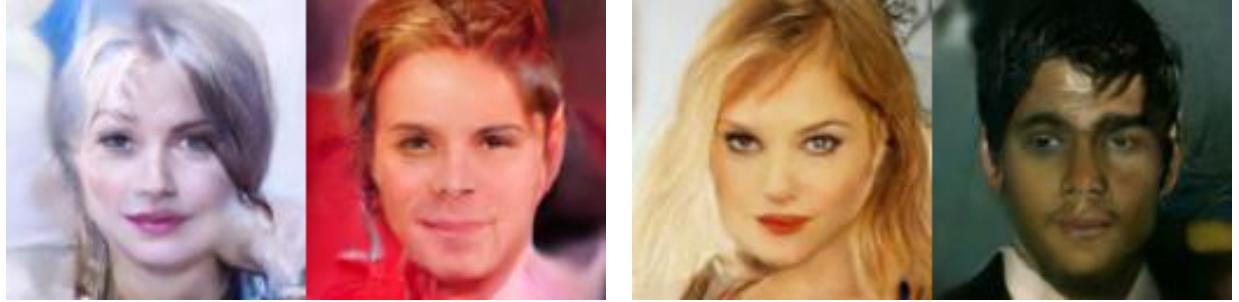
Tempo di campionamento. I modelli con il minor tempo di campionamento sono stati i DDIM. In questo caso i modelli DDIM hanno generato campioni con una velocità superiore ai DDPM di un fattore 20. Tuttavia occorre segnalare che questa maggiore velocità dei DDIM è stata certamente influenzata dal fatto che il campionamento per i DDIM è avvenuto con 50 passi di inferenza, mentre il campionamento per i DDPM è avvenuto con 1000 passi di inferenza.

Conclusioni. Sia per DDPM che per DDIM l’addestramento con la *linear schedule* ha condotto a una FID migliore rispetto all’addestramento con la *cosine schedule*. La *variance schedule* non ha invece influenzato significativamente il tempo di addestramento. A parità di *variance schedule*, le differenze tra DDPM e DDIM non sono state significative. Con la *linear schedule* DDPM ha ottenuto una FID leggermente migliore. Con la *cosine schedule* è stata invece DDIM a ottenere una FID leggermente migliore. Nella Figura 13 si riportano alcuni esempi di campioni generati dai DDPM con diverse *variance schedule*.

5.4.2 Confronto inferenza DDPM e DDIM

Il secondo confronto ha riguardato le performance dei modelli DDPM e DDIM pre-addestrati da Google in merito alla procedura di inferenza. In particolare sono stati confrontati: (i) DDPM con 50 passi di inferenza; (ii) DDPM con 1000 passi di inferenza; (iii) DDIM con 50 passi di inferenza; e (iv) DDIM con 1000 passi di inferenza. In questo caso i passi di inferenza sono i passi di graduale rimozione del rumore che compongono la procedura di campionamento. L’oggetto del confronto ha riguardato la FID e il tempo di generazione di 100 campioni utilizzando il modello pre-addestrato. Gli esperimenti per questo confronto sono stati condotti utilizzando Google Colaboratory [44]. I risultati del confronto sono mostrati nella Tabella 2.

FID. Il modello che ha ottenuto la FID migliore è DDPM con 1000 passi di inferenza. Tuttavia la FID registrata da DDIM con 50 passi di inferenza è di poco più alta. Il modello che ha ottenuto la FID peggiore è DDPM con 50 passi



(a) Esempi di immagini generate da DDPM con linear schedule. (b) Esempi di immagini generate da DDPM con cosine schedule.

Figure 13: Confronto tra immagini generate da (a) DDPM con *linear schedule* e (b) DDPM con *cosine schedule*. I modelli considerati per la generazione di queste immagini sono stati addestrati per 50 epoche.

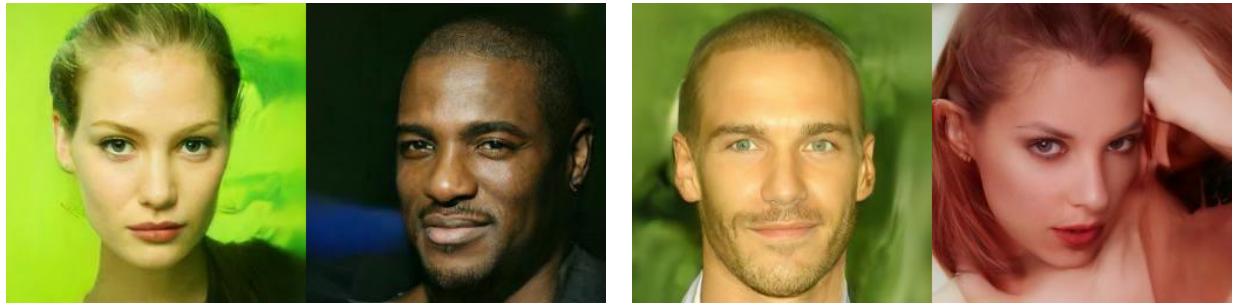
Table 2: Confronto inferenza DDPM e DDIM

Modello	Passi	FID	Tempo campionamento
DDPM	50	459.211	10 minuti
DDPM	1000	77.659	213 minuti
DDIM	50	81.006	10 minuti
DDIM	1000	94.843	213 minuti

di inferenza. In questo caso la FID registrata è particolarmente elevata. Infatti con soli 50 passi di inferenza DDPM ha prodotto immagini che si avvicinano al rumore Gaussiano puro. Infine si può notare come DDIM con 1000 passi di inferenza abbia registrato una FID peggiore rispetto a DDIM con soli 50 passi di inferenza. Questo risultato può sembrare sorprendente. In realtà si tratta di un fenomeno già osservato in letteratura [26, p. 6].

Tempo di campionamento. Naturalmente i modelli con il minor tempo di campionamento sono stati i modelli con il minor numero di passi di inferenza. In questo caso si nota come il tempo di campionamento sia cresciuto linearmente con il numero di passi di inferenza. Infatti passando da 50 passi di inferenza a 1000 passi di inferenza, anche il tempo di campionamento è cresciuto di un fattore 20.

Conclusioni. I modelli DDIM sono stati in grado di generare immagini con qualità comparabile ai DDPM, ma con una velocità superiore di almeno 20 volte. Tuttavia la qualità delle immagini generate con i DDIM sembra peggiorare quando si utilizza un numero di passi di inferenza elevato. Nella Figura 14 si riportano alcuni esempi di campioni generati. Nella Figura 15 e nella Figura 16 si mostrano i risultati del processo di campionamento per diversi passi di inferenza ottenuti rispettivamente con i DDPM e con i DDIM.



(a) Esempi di immagini generate con 1000 passi di inferenza utilizzando un modello DDPM pre-addestrato. (b) Esempi di immagini generate con 50 passi di inferenza utilizzando un modello DDIM pre-addestrato.

Figure 14: Confronto tra immagini generate con modelli pre-addestrati di (a) DDPM e (b) DDIM.

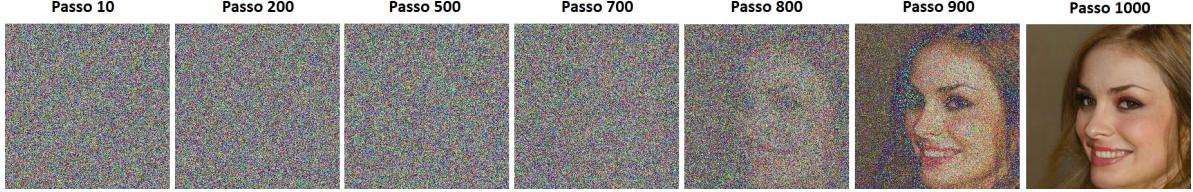


Figure 15: Risultati del processo di campionamento di DDPM con 1000 passi di inferenza.

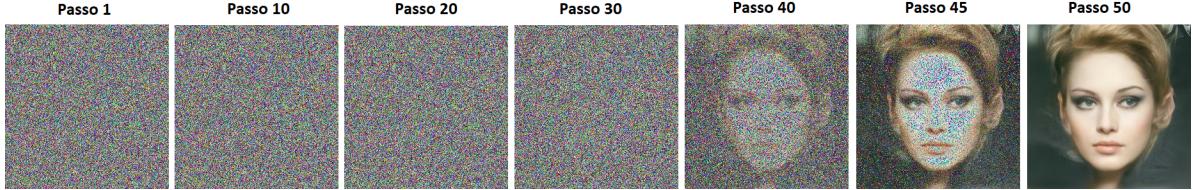


Figure 16: Risultati del processo di campionamento di DDIM con 50 passi di inferenza.

5.4.3 Analisi NCSN pre-addestrata

Sono stati generati 100 diversi campioni. In Figura 17 se ne vedono alcuni. La generazione di ogni campione ha impiegato in media 100 secondi. La FID è stata calcolata considerando le istanze del dataset CelebA [36] messe a confronto con i 100 campioni generati. Prima di eseguire il calcolo della FID, le immagini del dataset sono state ritagliate al centro, e ridimensionate a 32x32. Questo è stato fatto in modo da mettere a confronto i campioni generati con la stessa tipologia di istanze usate per il train. Come risultato si è ottenuto 110,163.

Conclusioni. La NCSN analizzata permette di generare campioni con un tempo di inferenza molto basso, proprio perché le immagini generate sono relativamente piccole, 32x32. Il valore di FID ottenuto è il più alto tra gli esperimenti eseguiti. L'esperimento è stato proposto soprattutto con lo scopo di mostrare gli effettivi miglioramenti introdotti da DDPM e SBGM-SDE.



Figure 17: (**Sinistra**) Alcuni dei campioni generati. (**Destra**) Esempi di campioni intermedi.

5.4.4 Analisi SBGM-SDE pre-addestrato

Il modello analizzato è stato preaddestrato da Google sul dataset CelebA-HQ ed è stato reso disponibile sull'hub di Hugging Face [35]. Esso è quindi stato usato per la generazione di 100 campioni tramite Google Colaboratory. Alcuni di questi campioni sono mostrati in figura 18. In aggiunta, nella figura 19 vengono mostrati i passi intermedi di un processo di campionamento.

FID. I 100 campioni generati sono stati usati —assieme al dataset CelebA-HQ— per il calcolo della FID, come descritto in 5.3. Il risultato ottenuto è stato di 63,514.

Tempo di campionamento. Per la generazione di un singolo campione 256x256 il modello impiega 12 minuti con l'uso di una GPU. Il tempo elevato è dovuto alla quantità di passi di inferenza da effettuare (2000).

Conclusioni. I risultati che questo modello permette di ottenere sono di buona qualità, come è possibile riscontrare sia visivamente che con l'uso della FID. Purtroppo, però, il costo computazionale della fase di inferenza per la generazione di un campione è decisamente elevato. La soluzione più naturale per affrontare il problema è quella di addestrare una



Figure 18: Alcuni esempi di campioni ottenuti dal modello pre-addestrato.

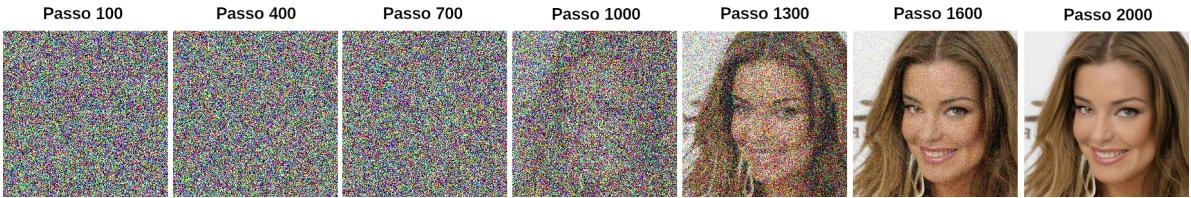


Figure 19: Passi intermedi del processo di campionamento di SBGM-SDE con 2000 passi di inferenza.

vriante del modello con un numero di passi di inferenza inferiore, sacrificando la qualità del risultato finale. In base al dominio applicativo è quindi necessario stabilire tale compromesso.

5.5 Similarity retrieval

L'ultimo esperimento svolto ha riguardato un *task* di *similarity retrieval*. In particolare questo esperimento ha avuto per oggetto l'individuazione delle immagini del dataset reale più simili a un campione selezionato di immagini generate. L'obiettivo di questo esperimento è stato verificare che i modelli addestrati non si fossero limitati a campionare immagini memorizzate dal dataset reale. La valutazione della *similarity retrieval* è stata puramente soggettiva. In particolare ci si è limitati a verificare che le immagini generate non fossero identiche a quelle del dataset delle immagini reali.

L'esperimento è stato condotto come segue. Anzitutto è stato utilizzato l'ultimo layer della rete Inception-V3 pre-addestrata su ImageNet per estrarre vettori di feature sia dal dataset di dati reali sia dalle immagini generate²⁹. Dopo di che si è proceduto a calcolare la distanza tra le feature delle immagini generate e le feature di ognuna delle immagini del dataset. Per calcolare questa distanza è stato utilizzato un algoritmo K-Nearest Neighbor con K pari a 5³⁰. La distanza considerata è stata quella Euclidea. Infine sono state recuperate le 5 immagini del dataset le cui feature sono risultate più vicine a quelle delle immagini generate.

I risultati per ciascun modello considerato sono mostrati nelle Figure 20, 21, 22 e 23. È agevole riscontrare che le immagini generate sono sufficientemente diverse dalle immagini del dataset. Si può dunque concludere che i Diffusion Models non si sono limitati semplicemente a campionare immagini dal dataset.



Figure 20: Risultati dell'operazione di *similarity retrieval* per il modello DDPM con 1000 passi di inferenza. A sinistra è mostrata l'immagine generata dal modello. A destra sono mostrate le 5 immagini più simili del dataset CelebA-HQ.

²⁹La rete Inception-V3 utilizzata è quella fornita dalla libreria Keras [45].

³⁰Per l'implementazione del Nearest Neighbor è stata utilizzata la libreria scikit-learn [46].



Figure 21: Risultati dell’operazione di *similarity retrieval* per il modello DDIM con 50 passi di inferenza. A sinistra è mostrata l’immagine generata dal modello. A destra sono mostrate le 5 immagini più simili del dataset CelebA-HQ.



Figure 22: Risultati dell’operazione di *similarity retrieval* per il modello NCSN pre-addestrato. A sinistra è mostrata l’immagine generata dal modello. A destra sono mostrate le 5 immagini più simili del dataset CelebA [36].



Figure 23: Risultati dell’operazione di *similarity retrieval* per il modello SBGM-SDE pre-addestrato. A sinistra è mostrata l’immagine generata dal modello. A destra sono mostrate le 5 immagini più simili del dataset CelebA-HQ.

6 Conclusioni

In questo lavoro si sono analizzati i Diffusion Models, con particolare attenzione a modelli non condizionali come DDPM, DDIM, NCSN e SBGM-SDE. Non sono stati approfonditi i modelli condizionali.

L’analisi svolta in questo lavoro si è concentrata sulle tipologie di Diffusion Models. Non si è estesa al confronto con altri tipi di modelli generativi. Tuttavia, per un miglior inquadramento, si riportano brevemente vantaggi e svantaggi rispetto alle GAN, che si collocano nello stato dell’arte:

- vantaggi dei Diffusion Models rispetto alle GAN sono: (i) non soffrono di mode collapse; (ii) sono meno sensibili all’ottimizzazione degli iperparametri; (iii) non necessitano di addestramento contraddittorio; (iv) offrono una funzione obiettivo adatta al confronto; e (v) non soffrono del problema del vanishing gradient;
- principale svantaggio dei Diffusion Models è rappresentato dall’elevato tempo computazione del processo di inferenza.

Per quanto riguarda gli esperimenti svolti, si sono tratte le seguenti conclusioni:

- I DDIM hanno registrato il miglior compromesso tra qualità dei campioni generati e tempo computazionale;
- I SBGM-SDE hanno prodotto i campioni di migliore qualità, al costo di un tempo computazionale molto elevato;

In questo lavoro sono state vagilate le potenzialità generative dei Diffusion Models analizzati. Tuttavia, questi modelli si prestano ad applicazioni più ampie e variegate. Tra queste si citano, a titolo meramente esemplificativo: (i) deblurring; (ii) image inpainting; (iii) image colorization; (iv) sintesi di audio; e (v) denoising [47, 48, 49, 14].

References

- [1] Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling. *arXiv preprint arXiv:2103.05180v2*, 2021.
- [2] Florinel-Alin Croitoru, Vlad Hondu, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. *arXiv preprint arXiv:2209.04747v3*, 2022.
- [3] Dall-e. <https://openai.com/dall-e-2>.
- [4] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125v1*, 2022.
- [5] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *Proceedings of the 32nd International Conference on Machine Learning*, pages 2256–2265, 2015.
- [6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems 33*, pages 6840–6851, 2020.
- [7] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502v4*, 2020.
- [8] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *arXiv preprint arXiv:1907.05600v3*, 2019.
- [9] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456v2*, 2020.
- [10] Hélène Soubaras. An evidential measure of risk in evidential markov chains. In *Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU ’09*, pages 863—874, Berlin, Heidelberg, 2009. Springer-Verlag.
- [11] Anwaar Ulhaq, Naveed Akhtar, and Ganna Pogrebna. Efficient diffusion models for vision: A survey. *arXiv preprint arXiv:2210.09292v2*, 2022.
- [12] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Yingxia Shao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *arXiv preprint arXiv:2209.00796v9*, 2022.
- [13] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *arXiv preprint arXiv:2107.03006v2*, 2021.
- [14] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. *arXiv preprint arXiv:2009.09761v3*, 2020.
- [15] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691v3*, 2019.
- [16] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *arXiv preprint arXiv:1711.05597v3*, 2018.
- [17] Liqun Chen, Chenyang Tao, Ruiyi Zhang, Ricardo Henao, and Lawrence Carin Duke. Variational inference and model selection with generalized evidence bounds. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 893–902. PMLR, 2018.
- [18] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.
- [19] T. Nathan Mundhenk, Rashmi Sundareswaraa, David R. Gerwe, and Yang Chen. High precision object segmentation and tracking for use in super resolution video reconstruction. *Imaging and Applied Optics*, 2011.
- [20] Calvin Luo. Understanding diffusion models: A unified perspective. *arXiv preprint arXiv:2208.11970v1*, 2022.
- [21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv preprint arXiv:1505.04597v1*, 2015.
- [22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *arXiv preprint arXiv:1411.4038v2*, 2015.
- [23] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *arXiv preprint arXiv:2105.05233v4*, 2021.

- [24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762v5*, 2017.
- [25] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096v2*, 2019.
- [26] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. *arXiv preprint arXiv:2102.09672v1*, 2021.
- [27] Zhifeng Kong and Wei Ping. On fast sampling of diffusion probabilistic models. *arXiv preprint arXiv:2106.00132v2*, 2021.
- [28] Openreview. <https://openreview.net/forum?id=St1giarCHLP>.
- [29] J. Lee Q. Liu and M. Jordan. A kernelized stein discrepancy for goodness-of-fit tests. *International Conference on Machine Learning*, pages 276–284, 2016.
- [30] A. Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 2005.
- [31] C. D. Gelatt S. Kirkpatrick and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, pages 671–680, 1983.
- [32] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *International Conference on Learning Representations*, 2016.
- [33] R. M. Neal. Mcmc using hamiltonian dynamics. *ParXiv preprint arXiv*, 2012.
- [34] Brian D.O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, pages 313–326, 1982.
- [35] Hugging face. <https://huggingface.co/>.
- [36] Celeba dataset. <http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html>.
- [37] J. Shlens V. Dumoulin and M. Kudlur. A learned representation for artistic style. *International Conference on Learning Representations*, 2017.
- [38] C. Shen G. Lin, A. Milan and I. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.
- [39] Hubert; Unterthiner Thomas; Nessler Bernhard; Hochreiter Sepp Heusel, Martin; Ramsauer. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in Neural Information Processing Systems*, 2017.
- [40] M. Fréchet. Sur la distance de deux lois de probabilité. *C. R. Acad. Sci. Paris*, 1957.
- [41] D. C. Dowson and B. V. Landau. The fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 1982.
- [42] Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. *Technical report CD-TR 94/64*, 1994.
- [43] W. Zaremba V. Cheung A. Radford T. Salimans, I. J. Goodfellow and X. Chen. Improved techniques for training gans. *Advances in Neural Information Processing Systems*, 2016.
- [44] Google colaboratory. <https://colab.research.google.com/>.
- [45] Keras. <https://keras.io/>.
- [46] scikit-learn. <https://scikit-learn.org/stable/>.
- [47] Bahjat Kawar, Michael Elad, Stefano Ermon, and Jiaming Song. Denoising diffusion restoration models. *arXiv preprint arXiv:2201.11793v3*, 2022.
- [48] Nithin Gopalakrishnan Nair, Kangfu Mei, and Vishal M. Patel. At-ddpm: Restoring faces degraded by atmospheric turbulence using denoising diffusion probabilistic models. *arXiv preprint arXiv:2208.11284v2*, 2022.
- [49] Ozan Özdenizci and Robert Legenstein. Restoring vision in adverse weather conditions with patch-based denoising diffusion models. *arXiv preprint arXiv:2207.14626v2*, 2022.
- [50] J. Shi Y. Song, S. Garg and S. Ermon. Sliced score matching: A scalable approach to density and score estimation. *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence*, 2019.
- [51] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 2000.

Appendice A Modelli Generativi Score-Based

Si supponga di avere a disposizione un dataset $\{x_i \in \mathbb{R}^D\}_{i=1}^N$ composto da dati identicamente distribuiti e tra loro indipendenti (i.i.d.), generati da una distribuzione non nota $p_{data}(x)$. Lo *score* per una distribuzione di probabilità $p(x)$ viene definito come:

$$\nabla_x \log p(x). \quad (40)$$

La *score network*, cioè la rete per la stima dello score, viene definita come $s_\theta : \mathbb{R}^D \rightarrow \mathbb{R}^D$. Quindi, s_θ è una rete neurale parametrizzata rispetto a θ , addestrata per calcolare lo score della distribuzione $p_{data}(x)$. I due ingredienti alla base di un modello generativo Score-Based sono:

- **Score Matching per la Stima dello Score:** tramite Score-Matching [30] si può addestrare una score network $s_\theta : \mathbb{R}^D \rightarrow \mathbb{R}^D$ per la stima dello score della distribuzione dei dati $p_{data}(x)$, senza dover prima addestrare un modello per stimare $p_{data}(x)$. In particolare, la funzione obiettivo da minimizzare viene definita come:

$$\mathbb{E}_{p_{data}(x)} \left[\text{tr}(\nabla_x s_\theta(x)) + \frac{1}{2} \|s_\theta(x)\|_2^2 \right], \quad (41)$$

La funzione tr ritorna la traccia di una matrice quadrata, cioè la somma degli elementi sulla sua diagonale. Nella pratica, la funzione obiettivo così definita non è considerabile nel caso di reti Deep e dataset di grandi dimensioni, a causa della computazione di $\text{tr}(\nabla_x s_\theta(x))$. Due dei metodi più popolari per l'applicazione dello Score-Matching in ambito large scale sono:

- *Denoising Score Matching*: variante dello Score-Matching che circumnaviga completamente il fattore $\text{tr}(\nabla_x s_\theta(x))$. Per fare ciò, prima perturba i dati x con una distribuzione di rumore definita a priori, e poi applica lo Score-Matching per stimare lo score della distribuzione perturbata;
- *Sliced Score Matching* [50]: lo sliced score matching utilizza proiezioni random dei dati per approssimare $\text{tr}(\nabla_x s_\theta(x))$. La funzione obiettivo diventa:

$$\mathbb{E}_{p_v} \mathbb{E}_{p_{data}(x)} \left[v^T \nabla_x s_\theta(x) v + \frac{1}{2} \|s_\theta(x)\|_2^2 \right], \quad (42)$$

dove p_v è una semplice distribuzione di vettori random. A differenza del Denoising Score Matching, che stima lo score di una distribuzione perturbata, lo sliced score matching fornisce una stima per lo score della distribuzione reale. In generale, richiede fino a quattro volte più tempo computazionale rispetto al Denoising Score Matching.

Si dimostra che la minimizzazione di tale funzione obiettivo, il cui risultato ottimo lo denotiamo con $s_{\theta^*}(x)$, soddisfa $s_{\theta^*}(x) = \nabla_x \log p(x)$ quasi sicuramente [50];

- **Langevin dynamics:** utilizzata per campionare da una certa distribuzione di probabilità $p(x)$ conoscendo solamente il suo valore di score $\nabla_x \log p_t(x)$. Data una dimensione per lo step $\epsilon > 0$, e un valore di partenza $\tilde{x}_0 \sim \pi(x)$, con π distribuzione a priori, il metodo di Langevin calcola il valore al passo successivo come:

$$\tilde{x}_t = \tilde{x}_{t-1} + \frac{\epsilon}{2} \nabla_x \log p(\tilde{x}_{t-1}) + \sqrt{\epsilon} z_t. \quad (43)$$

con $z_t \sim \mathcal{N}(0, I)$. Si ha che la distribuzione di \tilde{x}_T sarà uguale a $p(x)$ per $\epsilon \rightarrow 0$ e $T \rightarrow \inf$, con \tilde{x}_T che diventa un esempio esatto da $p(x)$.

Si noti che per campionare tramite l'utilizzo della dinamica di Langevin basta calcolare lo score $\nabla_x \log p_t(x)$. Quindi, per ottenere nuovi campioni sempre appartenenti alla distribuzione $p_{data}(x)$ definita dal training set, si deve prima di tutto addestrare la score network in modo da ottenere $s_\theta(x) \sim \nabla_x \log p(x)$. Successivamente si andrà ad utilizzare $s_\theta(x)$ e, tramite applicazione del metodo di Langevin, si potranno ottenere nuovi campioni a partire da valori casuali.

Sfide dei Modelli Generativi Score-Based I due principali ostacoli che impediscono l'applicazione naive dei modelli generativi Score-Based sono:

- **Manifold Hypothesis** (Ipotesi della Molteplicità): questa ipotesi dice che i dati, nel mondo reale, tendono a concentrarsi in regioni di bassa dimensionalità, presenti all'interno di spazi di dimensione molto più elevata. Si è dimostrato empiricamente, che questa ipotesi sussiste su molti dataset [51]. Intuitivamente, quello che succede è che i valori presenti all'interno di molti dataset non si distribuiscono uniformemente su tutto lo spazio considerato dal dataset, ma si vanno invece a concentrare solo in certe regioni di bassa dimensionalità. Sotto questa ipotesi, i modelli Score-Based si troveranno ad affrontare due problemi principali: primo, (i)

lo score (calcolato come mostrato dall'Equazione 40) non sarà definibile se il valore di x è confinato in una regione a bassa dimensionalità; inoltre, (ii) la funzione obiettivo per lo score matching (definita dall'Equazione 41) fornisce una buona stima dello score solo quando la distribuzione dei dati considera l'intero spazio. Sarà inconsistente se i dati risiedono in regioni di bassa dimensionalità. L'effetto negativo sullo score dato dall'ipotesi della Molteplicità può essere ben visto in Figura 24. Qui si vede (*sinistra*) il risultato che si ottiene addestrando una rete per stimare lo score sul dataset CIFAR-10: si noti come inizialmente la funzione di loss diminuisca, per poi iniziare a fluttuare irregolarmente. Invece, sulla destra possiamo vedere il risultato che si ottiene con la stessa rete, andando però a perturbare preventivamente i dati con un piccolo rumore Gaussiano: in questo caso si riesce a convergere. Il rumore Gaussiano qui considerato è un $\mathcal{N}(0, 0.0001)$ (vedi [8]), quindi abbastanza piccolo da non essere notato ad occhio nudo, ma adeguato per poter distribuire le immagini in maniera più uniforme rispetto allo spazio considerato;

- **Regioni a bassa densità di dati:** la presenza di pochi dati in queste regioni sparse può causare difficoltà sia nel processo di stima dello score, che nel campionamento tramite Langevin dynamics. Questo perché:

- Nelle regioni a bassa densità di dati, lo Score-Matching molto spesso non dispone di abbastanza evidenze per stimare in modo corretto lo score. Questo accade perché, ricordando l'Equazione 41, lo Score-Matching va a minimizzare l'errore quadratico atteso per la stima dello score. Nella pratica, quello che si fa è andare a stimare la distribuzione dei dati $p_{data}(x)$ utilizzando i campioni forniti dal dataset $\{x_i\}_{i=1}^N \sim p_{data}(x)$. Se si considerano le regioni a bassa densità $R \subset \mathbb{R}^D$, cioè quelle regioni tali per cui $p_{data}(R) \sim 0$, quindi porzioni di spazio dove giacciono pochi dati, molto probabilmente queste non saranno coperte dai campioni presenti nel dataset, cioè si avrà $\{x_i\}_{i=1}^N \cap R = \emptyset$. Non si sarà allora in grado di stimare lo score in maniera precisa per queste regioni;
- Quando due concentrazioni della distribuzione dei dati sono separate da regioni a bassa densità, la dinamica di Langevin classica non sempre è in grado di oltrepassarle in tempi ragionevoli, non riuscendo a convergere verso la vera distribuzione. Per dimostrare empiricamente questo fatto, in [8, p. 4-5] viene proposto un esperimento che prevede il test della dinamica di Langevin classica per il campionamento da una distribuzione Gaussiana mista $p_{data} = \frac{1}{5}\mathcal{N}((-5, -5), I) + \frac{4}{5}\mathcal{N}((5, 5), I)$, andando ad utilizzare lo score calcolato in modo esatto. In Figura 25 si può vedere il risultato ottenuto: si nota ad occhio nudo come la dinamica di Langevin abbia prodotto campioni con distribuzione che non si adatta alla densità delle due sezioni di cui si compone la distribuzione reale, anche se si è utilizzato lo score esatto. Invece, la versione annealed della dinamica di Langevin, su cui si basano le NCSN, ha generato campioni che meglio si adattano alla distribuzione reale;

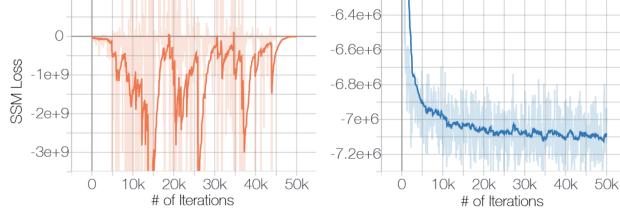


Figure 24: Sulla sinistra si può vedere l'evoluzione della loss nel caso di stima dello score sui dati reali, mentre, sulla destra si può l'evoluzione della loss nel caso di stima dello score sui dati perturbati con rumore Gaussiano. Questa figura è tratta da [8, p. 3].

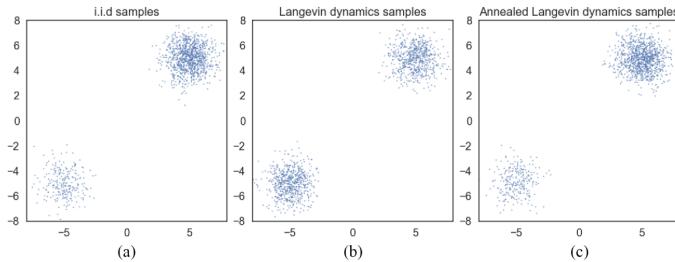


Figure 25: Campioni della distribuzione $p_{data} = \frac{1}{5}\mathcal{N}((-5, -5), I) + \frac{4}{5}\mathcal{N}((5, 5), I)$. (a) Campionamento esatto, (b) Campionamento con dinamica di Langevin con score esatto, e (c) Campionamento con versione annealed della dinamica di Langevin e score esatto. Questa figura è tratta da [8, p. 5].