ANNÉE 2024/2025

Rapport du Projet

réalisé par

"HOUARI Yassine"

Intitulé du projet :

Diffusion Models :
Denoising Diffusion Probabilistic Models

Table des matières

Introduction génerale

Les modèles de diffusion, et en particulier les *Denoising Diffusion Probabilistic Models* (DDPM), représentent une avancée majeure dans le domaine des modèles génératifs en intelligence artificielle. Leur succès repose sur une approche originale consistant à corrompre progressivement les données par l'ajout de bruit gaussien, puis à apprendre un processus inverse capable de reconstruire les données initiales. Cette dynamique en deux étapes permet de générer de nouvelles instances réalistes, qu'il s'agisse d'images, de sons ou d'autres types de données complexes.

L'objectif de ce travail est d'apporter une vision globale des modèles de diffusion. Nous présentons d'abord les fondements théoriques et les formulations mathématiques qui régissent leur fonctionnement, en insistant sur le processus direct (diffusion) et le processus inverse (dénombrement probabiliste). Ensuite, nous proposons une mise en perspective en comparant les DDPM avec d'autres modèles génératifs tels que les *Generative Adversarial Networks* (GAN) ou les *Variational Autoencoders* (VAE), afin de souligner leurs différences méthodologiques, leurs avantages et leurs limites. Enfin, une discussion des résultats obtenus lors de l'application des DDPM permettra de mettre en évidence leur efficacité ainsi que les défis encore ouverts dans ce domaine en pleine évolution.

Diffusion Models : DDPM

1 Introduction aux Diffusion Models

Soit $p(x)$ une distribution de probabilité qui représente nos données d'entraînement.

— Cette distribution est tellement complexe qu'il est impossible de trouver une expression analytique simple pour la décrire complètement.

— En fait, il est probable qu'une telle expression n'existe même pas :

$$p(x) = ?$$

Le défi consiste donc à trouver un moyen de générer de nouveaux échantillons sans disposer d'une description complète et explicite de la distribution $p(x)$.

Les **diffusion models** relèvent ce défi en suivant un processus d'inversion du bruit gaussien ajouté aux images :

- Dans un premier temps, on corrompt progressivement les images en leur ajoutant du bruit gaussien.

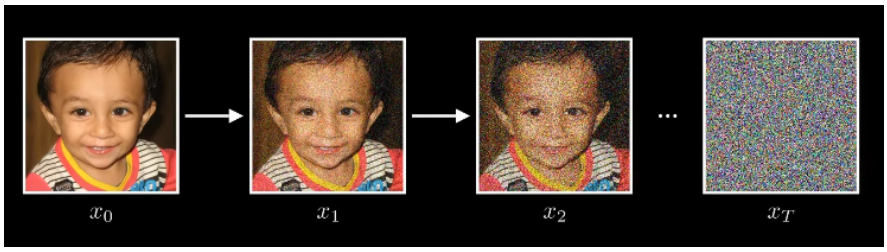


Fig. 1.1 L'ajout du bruit Gaussien

- Si l'on répète ce processus pendant un grand nombre d'étapes, l'image se transforme finalement en pur bruit.

L'idée clé est alors d'entraîner un modèle capable d'apprendre à inverser ce processus en supprimant le bruit étape par étape.
Si le modèle est efficace, il devrait être capable de partir d'un bruit aléatoire et de le raffiner progressivement pour obtenir une image cohérente et significative.

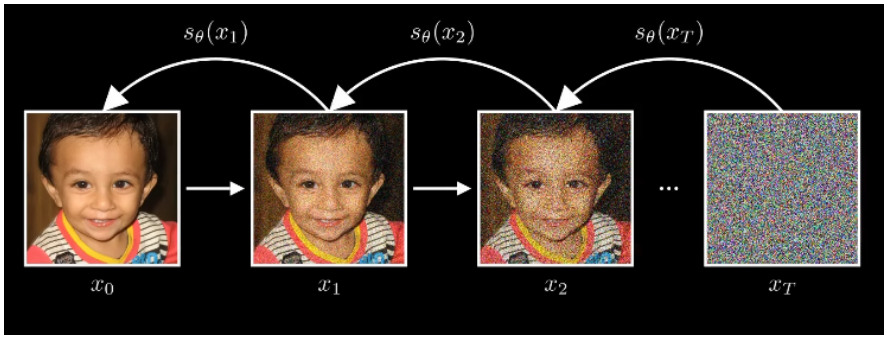


Fig. 1.2 Inversion du processus

2 Forward Process

Commençons par une image, que nous appelons x_0, puisqu'elle est le point de départ du processus de bruitage. Pour l'instant, cette image ne contient aucun bruit.

Pour générer l'image suivante x_1, nous utilisons une distribution conditionnelle $q(x_1 \mid x_0)$.

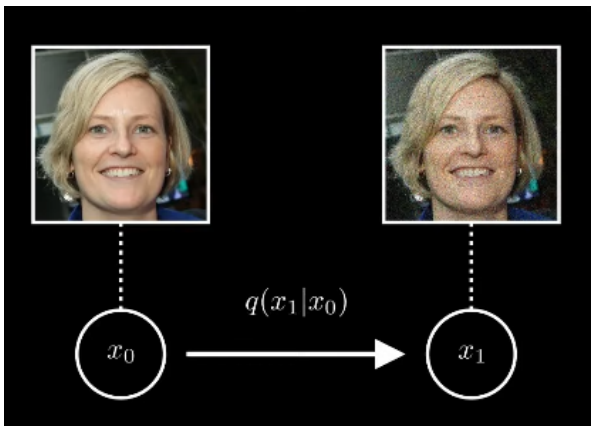


Fig. 1.3 Créeation de l'image x_1

Nous produisons x_1 en ajoutant une certaine quantité de bruit gaussien ϵ :

$$x_1 \;=\; x_0 \;+\; \beta\,\epsilon, \qquad \epsilon \sim \mathcal{N}(0,1).$$

La quantité de bruit est contrôlée par un scalaire $\beta > 0$ (la *variance*). Ainsi, la loi conditionnelle s'écrit

$$q(x_1 \mid x_0) \;=\; \mathcal{N}(x_0,\, \beta).$$

Pour passer à l'étape suivante, on applique la même opération :

$$x_2 \;=\; x_1 \;+\; \beta\,\epsilon, \qquad \epsilon \sim \mathcal{N}(0,1).$$

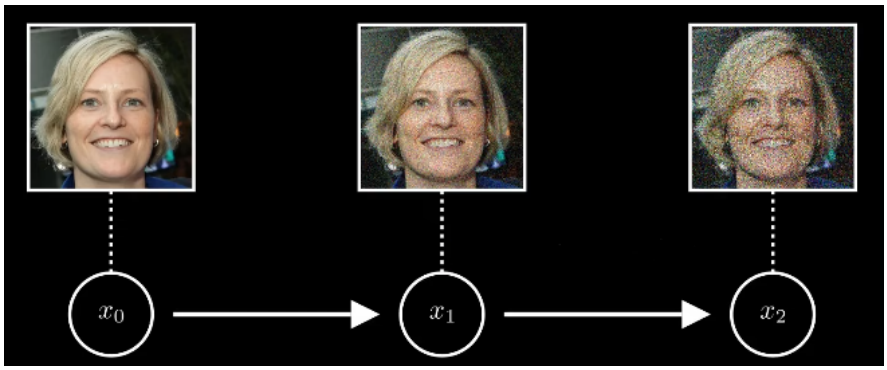


Fig. 1.4 Production de x_2 a partir de x_1

Par addition de bruits indépendants, on obtient

$$q(x_2 \mid x_0) \;\sim\; \mathcal{N}(x_0,\, 2\beta), \quad \text{et} \quad x_2 \;=\; x_0 \;+\; 2\beta\,\epsilon, \;\; \epsilon \sim \mathcal{N}(0,1).$$

En répétant ce processus jusqu'au temps t, on a en général

$$q(x_t \mid x_0) \sim \mathcal{N}(x_0, t\beta) \iff x_t = x_0 + t\beta\,\epsilon, \qquad \epsilon \sim \mathcal{N}(0,1).$$

Fig. 1.5 Production de x_2 a partir de x_0

Maintenant, nous avons un problème avec ce processus.

Problématique : Ce que nous voulions était un processus qui transforme graduellement nos données en une distribution normale standard. Clairement, ce n'est pas ce qui se passe ici :

— La moyenne reste toujours fixée à x_0 et n'est jamais modifiée par le processus.

— La variance, elle, continue d'augmenter indéfiniment.

Ainsi, dans cette configuration, il n'y a absolument aucune chance que notre processus de diffusion converge vers la distribution normale :

$$q(x_t \mid x_0) = \mathcal{N}(x_0, t \cdot \beta) \not\to \mathcal{N}(0,1).$$

2.1 Nouvel objectif

Définissons maintenant un processus de diffusion qui fonctionne réellement.

$$q(x_t \mid x_0) \longrightarrow \mathcal{N}(0,1) \quad \text{quand } t \to \infty$$

C'est-à-dire un processus qui transforme graduellement nos données en une loi normale standard. Pour cela, nous devons satisfaire deux conditions :

— La moyenne doit converger vers 0.

— La variance doit converger vers 1.

Pour y parvenir, nous allons modifier la façon de passer d'une étape à l'autre. Évidemment, il faut changer la moyenne de la distribution à chaque pas, et donc introduire un coefficient devant la moyenne afin qu'elle décroisse vers 0.

Ce nombre magique que nous allons utiliser est :

$$\sqrt{1-\beta}.$$

Ainsi, la formule devient :

$$q(x_t \mid x_{t-1}) = \sqrt{1-\beta}\, x_{t-1} \; + \; \beta\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I).$$

De cette façon, pour aller de l'entrée sans bruit x_0 jusqu'à un pas t donné du processus de diffusion, il suffit d'appliquer cette formule.

$$q(x_t \mid x_0) = \sqrt{\bar{\alpha}_t}\, x_0 \; + \; \sqrt{1-\bar{\alpha}_t}\, \epsilon,$$

Le résultat dépend alors du terme $\bar{\alpha}_t$ où :

$$\bar{\alpha}_t = (1-\beta)^t.$$

2.2 Explication : Pourquoi le choix de $\sqrt{1-\beta}$?

On aurait pu imaginer un processus plus général de la forme :

$$x_t = a\, x_{t-1} + b\, \epsilon_t, \qquad \epsilon_t \sim \mathcal{N}(0, I),$$

où a et b sont deux coefficients (constants ou dépendants de t).

1. Évolution de la moyenne

On a :

$$\mathbb{E}[x_t] = a\, \mathbb{E}[x_{t-1}] \quad \Longrightarrow \quad \mathbb{E}[x_t] = a^t\, x_0.$$

Ainsi, pour que la moyenne tende vers 0 lorsque $t \to \infty$, il faut :

$$|a| < 1.$$

2. Évolution de la variance

On calcule :

$$\mathrm{Var}(x_t) = a^2\, \mathrm{Var}(x_{t-1}) + b^2.$$

En itérant cette relation, on obtient à la limite :

$$\lim_{t\to\infty} \mathrm{Var}(x_t) = \frac{b^2}{1-a^2}.$$

Donc, pour que la variance converge vers 1, il faut :

$$\frac{b^2}{1-a^2} = 1 \quad \Longrightarrow \quad b^2 = 1 - a^2.$$

3. Choix standard dans les diffusion models

Dans les Diffusion Models, on paramètre :

$$a = \sqrt{1-\beta}, \qquad b = \sqrt{\beta}.$$

Ce choix satisfait clairement la contrainte :

$$a^2 + b^2 = (1-\beta) + \beta = 1,$$

et permet une écriture simple et analytique :

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, \qquad \bar{\alpha}_t = \prod_{s=1}^{t} (1-\beta_s).$$

Conclusion. En théorie, d'autres couples (a, b) pourraient convenir tant que $b^2 = 1 - a^2$ et $|a| < 1$. Mais le choix

$$(a, b) = (\sqrt{1-\beta}, \sqrt{\beta})$$

est le plus naturel : il garde la variance sous contrôle, rend les équations simples à manipuler et fournit une interprétation claire comme combinaison de *signal* et de *bruit*.

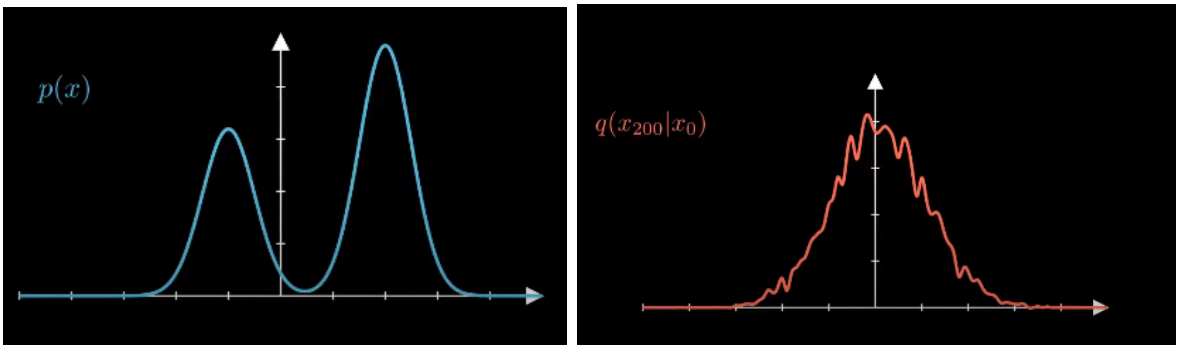
2.3 Convergence vers la loi normale

Est-ce que ce processus converge réellement vers la distribution normale ?

Si nous choisissons $0 < \beta < 1$, il est facile de voir que $\bar{\alpha}_t \to 0$, alors la moyenne de la distribution conditionnelle tend vers 0, tandis que la variance tend vers 1.

Ainsi, nous obtenons enfin un processus de diffusion qui préserve correctement la variance. Cette formule est censée prendre notre distribution de données $p(x)$ et la transformer progressivement en une loi normale standard.

Exemple 1D



(a) Distribution initiale : $p(x)$ **(b)** Distribution après diffusion : $\mathcal{N}(0, 1)$

Fig. 1.6 Illustration 1D : le processus de diffusion transforme progressivement une distribution complexe $p(x)$ en une loi normale standard $\mathcal{N}(0, 1)$.

En pratique, les diffusion models appliquent couramment des quantités de bruit variables à chaque étape, plutôt qu'un β fixe. En effet, n'importe quel *noise schedule* peut être choisi tant que chaque β_t reste compris entre 0 et 1.

Comment ce fait-il influencer notre formule pour passer de l'image sans bruit jusqu'à une étape donnée du processus ?

Lorsque β varie à chaque étape, $\bar{\alpha}_t$ devient simplement le produit de $1-\beta$ à chaque étape :

$$\bar{\alpha}_t = \prod_{s=1}^{t} (1 - \beta_s).$$

Nous avons ainsi défini un processus de diffusion correct, qui transformera progressivement n'importe quelle distribution de données en une distribution normale.

3 Processus inverse

Supposons maintenant qu'il existe un moyen d'inverser ces étapes individuelles, et présentons-le à l'aide d'une densité de probabilité p qui produit x_{t-1} à partir de x_t.

Tout comme nous avons défini des distributions conditionnelles q pour aller vers l'avant d'une étape à l'autre, nous définissons maintenant des distributions conditionnelles p pour aller dans la direction opposée.

Alors que dans le processus direct tous les paramètres sont fixés, pour le processus inverse, notre objectif est en réalité de trouver les meilleurs paramètres θ, qui nous permettent de retirer le bruit de manière efficace.

Ces paramètres θ correspondent aux poids d'un réseau de neurones.

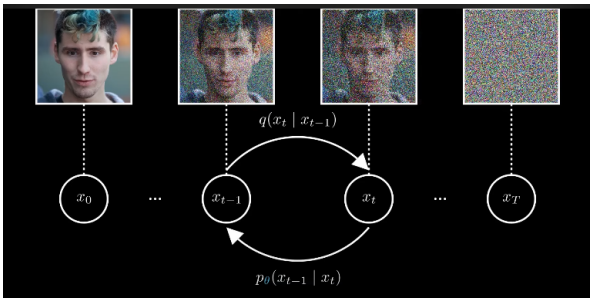


Fig. 1.7 Processus inverse

La questoin qui se pose : Comment entrainer ce réseau de neuronnes ?

Nous allons adopter une approche très standard en statistiques bayésiennes, qui consiste à minimiser la *log-vraisemblance négative* :

$$-\log p_\theta(x_0)$$

Avant d'aller plus loin, définissons clairement quelques notations utiles.

3.1 Distribution jointe du processus direct

La probabilité jointe décrivant la distribution de toutes les variables de x_0 à x_T, sachant x_0, représente notre processus direct complet. En utilisant le théorème de Bayes, on peut décomposer cette probabilité jointe en distributions conditionnelles individuelles :

$$q(x_1, \ldots, x_T \mid x_0) = q(x_1 \mid x_0)\, q(x_2 \mid x_1, x_0) \ldots q(x_T \mid x_{T-1}, \ldots, x_0)$$

Et l'avantage est que, puisque le processus direct est une chaîne de Markov :

$$q(x_1, \ldots, x_T \mid x_0) = q(x_1 \mid x_0)\, q(x_2 \mid x_1) \ldots q(x_T \mid x_{T-1}),$$

chaque étape de bruitage dépend uniquement de l'étape précédente, ce qui simplifie grandement ces probabilités conditionnelles.

Notation compacte

On peut alors écrire de manière compacte :

$$q(x_{1:T} \mid x_0) = q(x_1 \mid x_0) \ldots q(x_T \mid x_{T-1}) = \prod_{t=1}^{T} q(x_t \mid x_{t-1}).$$

3.2 Distribution jointe du processus inverse

On utilise la même notation compacte pour la distribution jointe décrivant notre processus inverse complet :

$$q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1}).\text{(processus direct complet)}$$

$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t).\text{(processus inverse complet)}$$

3.3 Log-vraisemblance négative (Negative log likelihood)

Maintenant que nous avons des expressions décrivant clairement nos processus direct et inverse complets, essayons de trouver une expression pour la log-vraisemblance négative.

De manière générale, lorsqu'on manipule des probabilités jointes, une première étape utile consiste souvent à marginaliser la distribution par rapport aux autres variables. En faisant cela, on obtient que $p_\theta(x_0)$ est simplement l'intégrale de la probabilité jointe par rapport à toutes les variables :

$$-\log p_\theta(x_0) \;=\; -\log \int p_\theta(x_{0:T})\, dx_{1:T}.$$

Peut-on calculer la vraisemblance en utilisant seulement cette formule ?

Pas vraiment. Ce que signifie cette intégrale, c'est que pour calculer la probabilité que notre réseau de neurones génère un échantillon x_0, il faudrait sommer sur tous les chemins possibles capables de le générer. Visualisons ca :

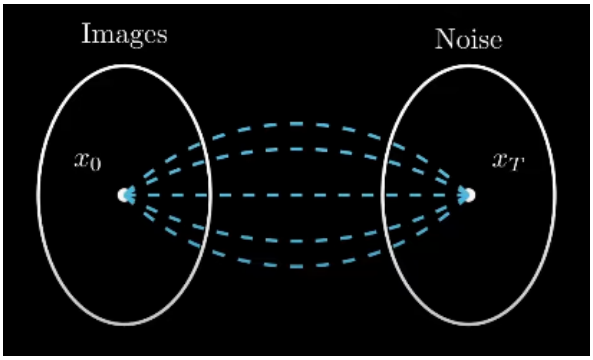


Fig. 1.8 Visualisation Images-Bruit

À gauche, nous avons la densité des images naturelles, et à droite la densité du bruit gaussien pur.

Nous savons qu'il existe des chemins reliant les images au bruit, comme notre processus direct, et nous supposons qu'il existe des chemins reliant le bruit aux images.

La formule nous dit que nous devrions sommer sur tous ces chemins possibles pour obtenir la log-vraisemblance négative. Bien sûr, il existe beaucoup trop de chemins pour que nous puissions calculer cette quantité.

On dit que cette quantité est *intractable*, ce qui signifie que nous ne pouvons pas la calculer exactement de manière réaliste. Mais cela ne veut pas dire que nous ne pouvons pas calculer la log-vraisemblance négative du tout !

3.4 De la log-vraisemblance à la borne variationnelle (ELBO)

Utilisons maintenant une astuce bien connue : nous multiplions et divisons par la densité représentant notre processus direct (forward process). On obtient :

$$-\log p_\theta(x_0) = -\log \int q(x_{1:T} \mid x_0)\, \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\, dx_{1:T}.$$

Nous pouvons réécrire cette intégrale comme une espérance, ce qui rend l'expression plus compacte :

$$-\log p_\theta(x_0) = -\log\ \mathbb{E}_{q(x_{1:T}|x_0)}\left[\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right].$$

Inégalité de Jensen

Énoncé : Soit X une variable aléatoire définie sur un espace de probabilité, et soit $\varphi : \mathbb{R} \to \mathbb{R}$ une fonction convexe. Alors :

$$\varphi(\mathbb{E}[X])\ \leq\ \mathbb{E}[\varphi(X)],$$

à condition que les deux membres soient finis.

Comme la fonction $-\log$ est convexe, nous pouvons faire entrer le logarithme à l'intérieur de l'espérance, au prix d'introduire une inégalité :

$$-\log p_\theta(x_0)\ \leq\ -\mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right].$$

Evidence Lower Bound (ELBO)

Nous obtenons ainsi une borne supérieure sur la log-vraisemblance négative, souvent appelée *Evidence Lower Bound* (ELBO) :

$$\mathcal{L}_{\text{ELBO}}(x_0) = -\mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right].$$

Réécriture de l'ELBO

Maintenant, nous allons essayer de donner un sens à cet objectif d'entraînement en modifiant et simplifiant l'expression de la log-vraisemblance négative.

Nous pouvons réécrire l'ELBO sous cette forme utile :

$$\mathcal{L} = \mathbb{E}_q[D_{\text{KL}}(q(x_T \mid x_0) \,||\, p(x_T)) + \sum_{t>1} D_{\text{KL}}(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)) - \log p_\theta(x_0 \mid x_1)]$$

Divergence de Kullback-Leibler et sa relation avec l'espérance

La **divergence de Kullback-Leibler ($\mathbf{D_{KL}}$)**, également appelée entropie relative, est une mesure de la différence entre deux distributions de probabilité P et Q. Elle quantifie à quel point la distribution P est différente de la distribution Q, souvent utilisée en théorie de l'information pour évaluer la perte d'information lorsqu'on utilise Q pour approximer P.

Définition formelle

Pour deux distributions de probabilité P et Q définies sur le même espace, la divergence de Kullback-Leibler est donnée par :

1. **Cas discret** :
$$D_{\mathrm{KL}}(P||Q) = \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$
où x parcourt tous les événements possibles, et $P(x)$ et $Q(x)$ sont les probabilités respectives sous P et Q.
2. **Cas continu** :
$$D_{\mathrm{KL}}(P||Q) = \int P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx$$
où $P(x)$ et $Q(x)$ sont les densités de probabilité.

Propriétés

— **Non-négativité** : $D_{\mathrm{KL}}(P||Q) \geq 0$, avec égalité si et seulement si $P = Q$ presque partout.

— **Non-symétrie** : $D_{\mathrm{KL}}(P||Q) \neq D_{\mathrm{KL}}(Q||P)$, ce qui signifie que ce n'est pas une distance au sens mathématique, mais une mesure de dissimilarité.

— **Interprétation** : La $\mathrm{D_{KL}}$ peut être vue comme la quantité d'information supplémentaire nécessaire pour coder des échantillons de P en utilisant un codage optimisé pour Q.

Relation avec l'espérance

La divergence de Kullback-Leibler peut être exprimée en termes d'espérance. Plus précisément, pour la distribution P, on peut réécrire la $\mathrm{D_{KL}}$ comme :

$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_P\left[\log\left(\frac{P(X)}{Q(X)}\right)\right]$$

où $\mathbb{E}_P$ désigne l'espérance par rapport à la distribution P. Cela signifie que la $\mathrm{D_{KL}}$ est l'espérance, sous P, du logarithme du rapport entre les probabilités (ou densités) de P et Q.

Démonstration :

Dans ce paragraphe on va montrer que :

$$\mathcal{L} = \mathbb{E}_q[D_{\mathrm{KL}}(q(x_T \mid x_0) \,||\, p(x_T)) + \sum_{t>1} D_{\mathrm{KL}}(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)) - \log p_\theta(x_0 \mid x_1)]$$

L'égalité initiale :

$$\mathcal{L} = \mathbb{E}_q\Big[-\log\frac{p(x_{0:T})}{q(x_{1:T} \mid x_0)}\Big] = \mathbb{E}_q\Big[-\log p(x_{0:T}) + \log q(x_{1:T} \mid x_0)\Big]$$

or :

$$p(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p(x_{t-1} \mid x_t), \quad q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1})$$

Ce qui implique :

$$\log p(x_{0:T}) = \log p(x_T) + \sum_{t=1}^{T} \log p(x_{t-1} \mid x_t), \quad \log q(x_{1:T} \mid x_0) = \sum_{t=1}^{T} \log q(x_t \mid x_{t-1})$$

Donc :

$$\mathcal{L} = \mathbb{E}_q\Big[-\log p(x_T) - \sum_{t=1}^{T} \log \frac{p(x_{t-1} \mid x_t)}{q(x_t \mid x_{t-1})}\Big] = \mathbb{E}_q\Big[-\log p(x_T) - \sum_{t=2}^{T} \log \frac{p(x_{t-1} \mid x_t)}{q(x_t \mid x_{t-1})} - \log \frac{p(x_0 \mid x_1)}{q(x_1 \mid x_0)}\Big]$$

On utilise la relation suivante pour la distribution conditionnelle q :

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

En remplaçant dans $\log \frac{p(x_{t-1}|x_t)}{q(x_t|x_{t-1})}$, on obtient :

$$\log \frac{p(x_{t-1} \mid x_t)}{q(x_t \mid x_{t-1})} = \log p(x_{t-1} \mid x_t) - \log q(x_{t-1} \mid x_0, x_t) - \log q(x_t \mid x_0) + \log q(x_{t-1} \mid x_0)$$

$$= \log \frac{p(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)} \cdot \frac{q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)}$$

Donc :

$$\mathcal{L} = \mathbb{E}_q\Big[- \log \frac{p(x_T)}{q(x_T \mid x_0)} - \sum_{t>1} \log \frac{p(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)} - \log p(x_0 \mid x_1)\Big]$$

Et donc :

$$\boxed{\mathcal{L} = \mathbb{E}_q[D_{\mathrm{KL}}(q(x_T \mid x_0) \,||\, p(x_T)) + \sum_{t>1} D_{\mathrm{KL}}(q(x_{t-1} \mid x_t, x_0) \,||\, p_\theta(x_{t-1} \mid x_t)) - \log p_\theta(x_0 \mid x_1)]}$$

Avant de voir ce que sont réellement ces distributions, nous pouvons simplifier encore cette expression.

- Le premier terme : $D_{\mathrm{KL}}(q(x_1 \mid x_0))$ ne dépend pas de nos paramètres de modèle, donc on peut l'ignorer lors de l'optimisation du modèle.

- Le dernier terme : $\log p_\theta(x_0 \mid x_1)$ dépend bien des poids du réseau, et il mesure à quel point le modèle est capable de reconstruire une image propre x_0 à partir d'une version légèrement bruitée x_1. Cependant, dans les modèles de diffusion, on prend généralement environ un millier d'étapes de débruitage. Ainsi, lorsqu'on passe de x_1 à x_0, il reste très peu de bruit, le modèle a déjà récupéré la majeure partie de l'image. En conséquence, ce terme est très petit et n'apporte pas beaucoup de signal d'apprentissage supplémentaire. En pratique, l'inclure dans la perte ne change presque rien, donc on le supprime généralement de l'objectif final.

Il reste alors la partie principale : une grande somme de divergences de Kullback-Leibler entre certaines distributions :

$$\mathbb{E}_q\left[\sum_{t>1} D_{\mathrm{KL}}(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t))\right]$$

Nous savons que $p_\theta(x_{t-1} \mid x_t)$ correspond à notre étape de diffusion inverse. Mais qu'est-ce que cette autre distribution $q(x_{t-1} \mid x_t, x_0)$?

Cette distribution est ce qu'on appelle le **véritable postérieur**.

Si nous avons notre processus direct $q(x_t \mid x_{t-1})$, nous savons qu'il existe un processus inverse que nous essayons d'approximer en ajustant les paramètres d'une distribution p. Or, si nous conditionnons aussi sur l'image originale x_0, nous pouvons en fait calculer exactement ce processus inverse. Ce véritable processus inverse, qui utilise l'image propre x_0, est ce qu'on appelle le **véritable postérieur**. Et l'avantage est que nous avons une expression en forme fermée de ce véritable postérieur.

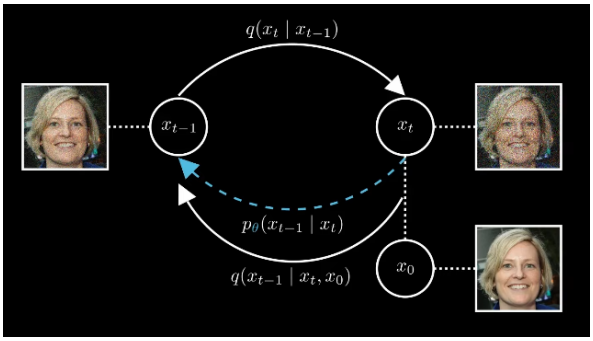


Fig. 1.9 Véritable postérieur

Mais alors, pourquoi avons-nous besoin d'un réseau de neurones si nous avons déjà une expression exacte pour inverser le bruit ? La réponse est : utiliser ce véritable postérieur serait de la triche, car il nécessite de connaître x_0, et pendant l'inférence, notre but est de générer x_0 en partant de pur bruit. Nous ne pouvons donc pas utiliser ce postérieur conditionné sur x_0 pour produire de nouveaux échantillons. Mais nous pouvons l'utiliser quand x_0 est connu, comme pendant l'entraînement, et c'est exactement ce que nous faisons ici.

Ainsi, ce que nous faisons réellement avec cette fonction de perte, c'est entraîner le réseau à **imiter le véritable postérieur** — celui que nous pourrions calculer si nous avions accès à x_0, même si en pratique le réseau ne voit que x_t.

En ce qui concerne l'expression du véritable postérieur, il se trouve qu'il est Gaussien, avec une moyenne $\tilde{\mu}_t$ et une variance $\tilde{\beta}_t$. Il est donc naturel de choisir aussi une forme Gaussienne pour notre postérieur approché :

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta, \sigma_\theta^2).$$

Deux raisons motivent ce choix :

1. Il existe un théorème qui dit que l'inverse d'une chaîne de Markov Gaussienne est également Gaussien.

2. Les calculs sont beaucoup plus simples avec des Gaussiennes.

Notre réseau prédit donc les paramètres de cette Gaussienne, à savoir sa moyenne μ_θ et sa variance σ_θ. Pour simplifier davantage, on peut fixer la variance σ_t à une constante non apprise. Ainsi, le réseau n'a qu'à apprendre la moyenne de la distribution.

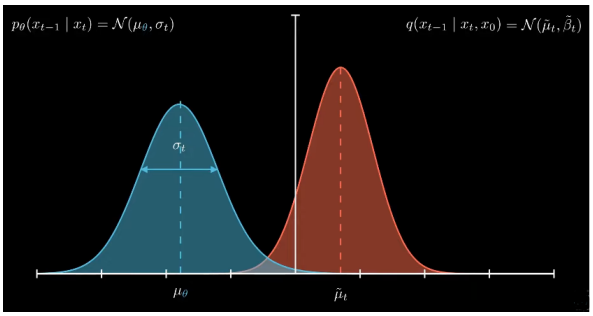


Fig. 1.1 Distrubution du véritable posterieur q & postérieur approché p

Puisque nous cherchons à faire correspondre deux Gaussiennes, mais que nous ne contrôlons que la moyenne de l'approximation, le mieux que nous puissions faire est de rapprocher cette moyenne de celle du véritable postérieur. En d'autres termes, nous cherchons à minimiser la distance entre les deux moyennes.

$$D_{KL}(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t)) = \tfrac{1}{2\sigma_t^2} \, \|\tilde{\mu}_t - \mu_\theta\|^2$$

Et en effet, dans ce cas, la divergence de KL se réduit à une simple distance au carré entre les moyennes.

Ainsi, minimiser la somme des divergences de KL dans notre objectif revient à rapprocher chaque postérieur approché de son véritable postérieur.

En pratique, cela nous ramène à minimiser une somme de distances L_2. Chaque terme mesure à quel point la moyenne prédite μ_θ est éloignée de la moyenne du véritable postérieur $\tilde{\mu}_t$, avec un poids dépendant de σ_t.

Nous avons une expression en forme fermée pour la moyenne du véritable postérieur :

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

On remarque qu'il s'agit d'une combinaison linéaire de x_0 et x_t, ce qui n'est pas très pratique à manipuler.

On peut écrire :

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

avec $\varepsilon \sim \mathcal{N}(0, I)$.

Réciproquement, avec un peu d'algèbre, on obtient aussi :

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

En injectant cela dans $\tilde{\mu}_t$, on obtient une expression dépendant uniquement de x_t et ε :

$$\tilde{\mu}_t = \frac{1}{\sqrt{\bar{\alpha}_t}} \Big(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon \Big).$$

Idée clé : puisque le débruiteur a aussi accès à x_t, on peut appliquer le même raisonnement au μ_θ prédit par le réseau et l'écrire comme :

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \Big(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta(x_t, t) \Big).$$

En insérant ces deux expressions dans la perte, les termes en x_t s'annulent et on obtient :

$$\mathcal{L} = \mathbb{E}_q \left[\sum_{t>1} \frac{\beta_t^2}{2\sigma_t^2 \bar{\alpha}_t (1-\bar{\alpha}_t)} \left\| \varepsilon - \varepsilon_\theta(x_t, t) \right\|^2 \right].$$

Nous obtenons donc une perte beaucoup plus simple : une distance au carré entre ε et ε_θ. Ainsi, ε_θ devient l'estimation par le réseau du bruit ajouté à x_0 pour produire x_t, et minimiser la perte revient à rendre cette prédiction la plus précise possible.

Enfin, nous pouvons encore simplifier : au lieu de sommer sur toutes les étapes t, ce qui est coûteux, nous choisissons un instant t aléatoire pour chaque échantillon, et en moyenne cela converge vers l'objectif initial :

$$\mathcal{L} = \mathbb{E}_{q,t} \left[\frac{\beta_t^2}{2\sigma_t^2 \bar{\alpha}_t (1-\bar{\alpha}_t)} \left\| \varepsilon - \varepsilon_\theta(x_t, t) \right\|^2 \right].$$

Vue d'ensemble et comparaison des architectures : VAE, GAN et modèles de diffusion

1 Variational Auto encoders (VAE)

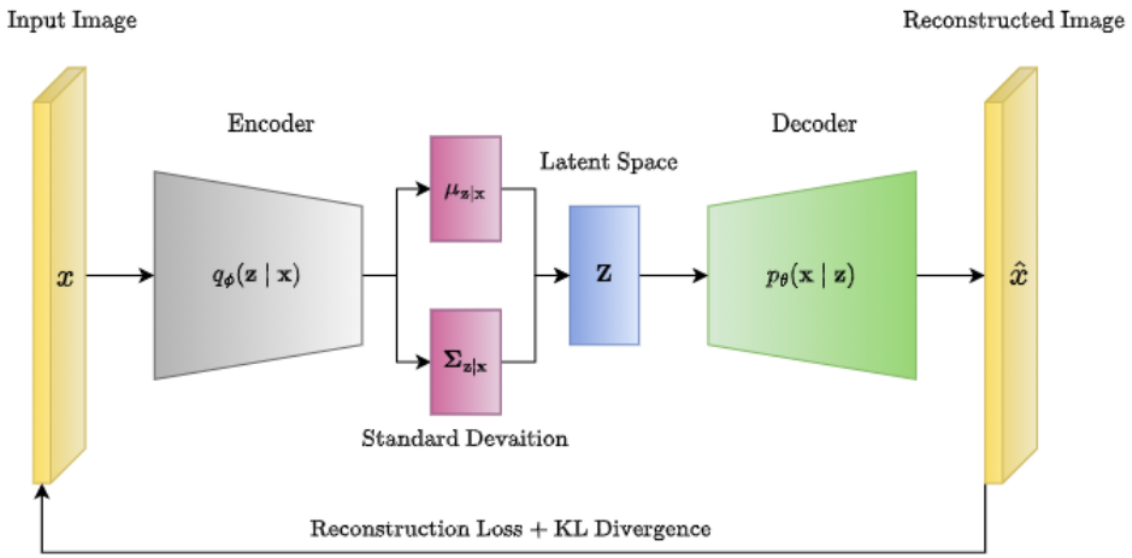
Les *Variational Autoencoders* (VAE) sont des modèles d'intelligence artificielle qui apprennent à générer de nouvelles données (images, textes, etc.) en imitant un ensemble de données existant. Ils compressent les données dans un espace latent (une représentation simplifiée) et les reconstruisent à partir de cet espace.

1.1 Objectif des VAE :

— **Reconstruction** : Reproduire les données d'entrée aussi fidèlement que possible.

— **Régularisation** : Structurer l'espace latent souvent sous forme une distribution gaussienne pour permettre la génération de nouvelles données cohérentes.

1.2 architecture de VAE



— **Architecture** : L'encodeur est généralement un réseau de neurones par exemple un réseau convolutif pour les images ou un réseau entièrement connecté pour d'autres types de données.

— **Sortie** : Pour chaque entrée x, l'encodeur produit deux vecteurs :

  — μ : La moyenne de la distribution latente, représentant le centre des représentations.

  — σ : L'écart-type (ou le logarithme de la variance, pour des raisons de stabilité numérique), représentant l'incertitude ou la variabilité autour de μ.

— Ces paramètres définissent la distribution $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu, \sigma^2)$, où ϕ représente les paramètres du réseau de l'encodeur.

Latent Space C'est le cœur probabiliste du VAE. Au lieu de prendre directement μ comme vecteur latent (ce qui rendrait le modèle déterministe), on échantillonne un vecteur z à partir de la distribution $\mathcal{N}(\mu(x), \sigma^2(x))$.

— **Décodeur** : C'est un réseau de neurones symétrique à l'encodeur (souvent un CNN déconvolutif ou transpose convolutif) qui prend le vecteur latent échantillonné z et le reconstruit en données d'entrée $\hat{x}$.

— **Sortie** : L'image reconstruite $\hat{x} = p_\theta(x|z)$, qui devrait ressembler à x.

— On sample $\epsilon \sim \mathcal{N}(0, 1)$ (bruit gaussien standard, indépendant des paramètres).

— Puis $z = \mu(x) + \sigma(x) \odot \epsilon$ (où $\odot$ est le produit Hadamard, élément par élément).

— **X** :Fournir les images x.

— **Encoder** : Obtenir μ(x) et σ(x)

— **Latent Space** :Obtenir $\hat{x}$

— **Décodeur** : d'apres z reconstruire $\hat{x}$.

— **Backpropagation** : Calculer la perte totale et backpropager pour updater les poids.

2 Generative Adversial Network (GAN)

Les *G*AN apprend à générer de nouvelles données qui ressemblent aux données réelles d'un dataset (e.g., images de visages). Il ne s'agit pas d'une simple reconstruction comme dans un VAE, mais d'une génération créative via une compétition entre deux modèles : le Générateur (G) crée des "fausses" données, et le Discriminateur (D) essaie de les distinguer des vraies. Au fil de l'entraînement, G devient expert à tromper D, et D devient expert à détecter les fakes. À la fin, G produit des données indistinguables des réelles.

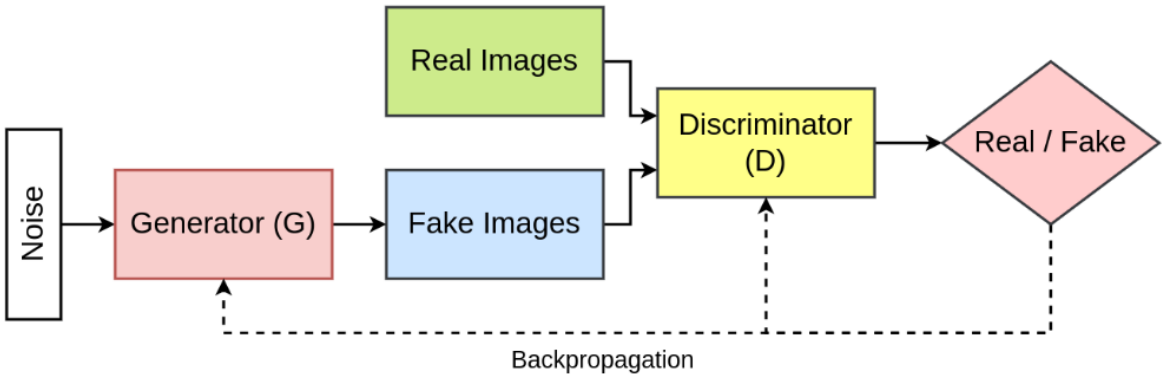
2.1 Objective de GAN

Un GAN est conçu comme un jeu à somme nulle :

— **Objectif du Discriminateur (D)** : Maximiser la probabilité de correctement classer les vraies images (réelles) comme "réelles" (1) et les fausses images (générées par G) comme "fausses" (0).

— **Objectif du Générateur (G)** : Minimiser la probabilité que D détecte ses fausses images comme fausses, c'est-à-dire maximiser $D(G(z))$ pour que D les prenne pour des vraies (proche de 1).

2.2 Architecture de GAN



— **Bruit (Noise)** : Entrée aléatoire pour G.

— **Générateur (G)** : Produit des Fake Images.

— **Images Réelles (Real Images)** : Données du dataset.

— **Discriminateur (D)** : Évalue à la fois les vraies et fausses images pour décider si elles sont Réal/Fake.

— **Backpropagation** : Lignes pointillées en entraînant G et D de manière antagoniste.

3 Avantages des Diffusion Models par rapport aux GAN

Les diffusion models surpassent souvent les GANs dans certains cas grâce à des avantages clés, bien que cela dépende du contexte :

— **Qualité et Diversité** : Les diffusion models produisent des images plus nettes et variées, car ils apprennent à débruiter progressivement à partir de bruit aléatoire, capturant mieux la distribution complète des données. Les GANs peuvent manquer de diversité et produire des artefacts.

— **Stabilité d'Entraînement** : Les diffusion models sont plus stables à entraîner grâce à une approche basée sur l'optimisation de la vraisemblance, sans la compétition instable entre générateur et discriminateur des GANs.

— **Contrôle et Flexibilité** : Ils permettent un meilleur contrôle (e.g., via guidance ou étapes de débruitage), utile pour des tâches comme la génération texte-image (e.g., Stable Diffusion). Les GANs sont moins adaptables à des conditions complexes.

— **Inconvénient Principal** : Les diffusion models sont beaucoup plus lents à l'inférence (plusieurs étapes de débruitage), tandis que les GANs génèrent des images en une seule passe.

4 Avantages des Diffusion Models par rapport aux VAE

Les **modèles de diffusion** surpassent souvent les **auto-encodeurs variationnels (VAE)** dans la génération de données, notamment pour des tâches comme la génération d'images de haute qualité. Voici une explication simple en quelques étapes des raisons pour lesquelles les modèles de diffusion sont généralement plus performants :

1. **Meilleure qualité des échantillons générés**
   - **Diffusion** : Les modèles de diffusion génèrent des données en raffinant progressivement du bruit aléatoire à travers plusieurs étapes de débruitage. Ce processus itératif capture des détails fins, produisant des échantillons (comme des images) plus nets et réalistes.
   - **VAE** : Les VAE optimisent une perte de reconstruction (souvent MSE), ce qui peut entraîner des échantillons flous, car ils lissent les détails pour minimiser l'erreur moyenne.
2. **Modélisation flexible des distributions complexes**
   - **Diffusion** : Ils modélisent la transformation du bruit vers les données sans imposer de contraintes fortes sur la distribution latente. Cela leur permet de capturer des distributions de données complexes et multimodales (par exemple, images avec des textures variées).
   - **VAE** : Les VAE imposent une distribution a priori (comme une gaussienne standard) via la divergence KL, ce qui peut limiter leur capacité à représenter des distributions complexes, réduisant la diversité des échantillons.
3. **Processus itératif vs espace latent contraint**
   - **Diffusion** : Le processus de débruitage itératif apprend à reconstruire les données étape par étape, offrant une grande flexibilité pour modéliser des structures complexes sans dépendre d'un espace latent fixe.
   - **VAE** : Les VAE s'appuient sur un espace latent structuré, régulé par la divergence KL, qui peut entraîner un effondrement postérieur ou une perte d'information, limitant la qualité des générations.
4. **Stabilité et robustesse**
   - **Diffusion** : Bien que l'entraînement et l'inférence soient plus lents (car itératifs), les modèles de diffusion sont souvent plus stables et moins sensibles aux problèmes d'équilibre entre termes de perte, contrairement aux VAE où le réglage de β (poids de la KL) est critique.
   - **VAE** : Les VAE peuvent souffrir d'un déséquilibre entre la perte de reconstruction et la divergence KL, ce qui peut dégrader la qualité des reconstructions ou des échantillons générés.

En résumé

Les modèles de diffusion surpassent les VAE car ils :

1. Produisent des échantillons plus nets et réalistes grâce à un processus itératif.
2. Capturent mieux les distributions complexes sans contraintes strictes sur l'espace latent.
3. Offrent une approche plus flexible et robuste, bien que plus coûteuse en calcul.

Cependant, les VAE restent utiles pour des tâches nécessitant un espace latent structuré ou une inférence rapide, mais pour la génération de données de haute qualité (images, audio), les modèles de diffusion sont souvent préférés.

U-net : architecture et utilsation

1 Introduction

U-Net est un réseau de neurones convolutifs (CNN) , il tire son nom de sa structure en forme de « U », combinant un chemin de contraction (encodeur) et un chemin d'expansion (décodeur). U-Net est particulièrement adapté aux tâches nécessitant des prédictions au niveau des pixels, comme la segmentation d'images ou, plus récemment, les modèles de diffusion pour la génération d'images.
Dans les modèles de diffusion, U-Net joue un rôle clé dans le processus de débruitage, permettant la génération d'images de haute qualité à partir de données bruitées.

2 Architecture de U-Net

L'architecture de U-Net est symétrique, composée de deux parties principales : le **chemin de contraction** (encodeur) et le **chemin d'expansion** (décodeur), reliés par des **connexions de saut**. Voici une description détaillée :

2.1 Chemin de contraction (Encodeur)

- **Objectif** : Extraire des caractéristiques abstraites en réduisant la résolution spatiale de l'image d'entrée.
- **Structure** :
  - Plusieurs couches convolutives (généralement 3×3) avec activations ReLU.
  - Couches de *pooling* (par exemple, max-pooling) pour réduire les dimensions spatiales (par exemple, division par 2).
  - Le nombre de filtres augmente progressivement (par exemple, $64 \rightarrow 128 \rightarrow 256$) pour capturer des caractéristiques plus abstraites.
- **Résultat** : Une représentation compacte de l'image, à faible résolution mais riche en informations abstraites.

2.2 Chemin d'expansion (Décodeur)

- **Objectif** : Restaurer la résolution spatiale pour produire une sortie de même taille que l'entrée.
- **Structure** :
  - Couches d'*upsampling* (par exemple, convolutions transposées) pour augmenter les dimensions spatiales.

- Couches convolutives pour affiner les caractéristiques.
- Le nombre de filtres diminue progressivement pour simplifier la représentation.

— **Résultat** : Une sortie à haute résolution (par exemple, une carte de segmentation ou une image débruitée).

2.3 Connexions de saut

— **Concept** : Les cartes de caractéristiques du chemin de contraction sont concaténées avec les couches correspondantes du chemin d'expansion.

— **Objectif** : Préserver les détails spatiaux fins (par exemple, contours, textures) perdus lors de la réduction de résolution.

— **Mécanisme** : Concaténation des cartes de caractéristiques avant les couches convolutives du décodeur.

2.4 Couche finale

Une couche convolutive 1×1 transforme la carte de caractéristiques finale en la sortie souhaitée (par exemple, une carte de segmentation ou une prédiction de bruit).

3 Rôle dans les modèles de diffusion

:

— U-Net agit comme le **modèle de débruitage** dans le processus inverse.

— À chaque étape t, U-Net prend en entrée :

- **Une image bruitée** x_t
- Un embedding temporel pour l'étape t.

— U-Net prédit le bruit ajouté à l'étape t ou l'image débruitée x_{t-1}.

— Cette prédiction guide le processus itératif de débruitage pour produire une image claire.

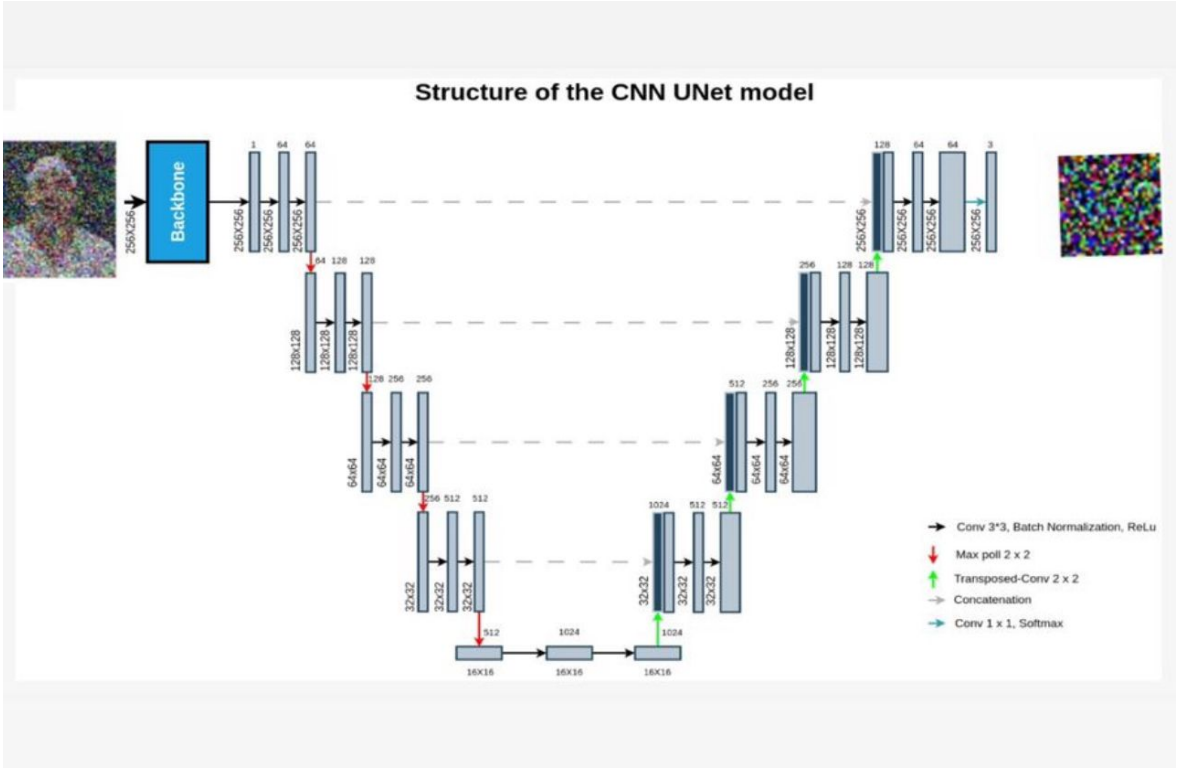


Fig. 3.1 Architecture du réseau U-net

3.1 Pourquoi U-Net ?

— **Préservation spatiale** : Les connexions de saut conservent les détails fins, essentiels pour des images de haute qualité.

— **Flexibilité** : Intègre facilement le conditionnement temporel et contextuel.

— **Efficacité** : Gère des images à haute résolution avec un coût computationnel relativement faible.

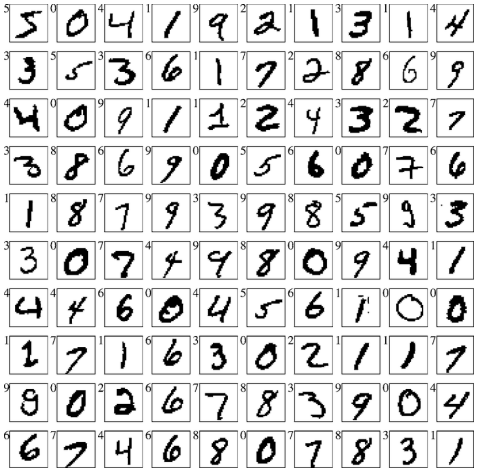
4 Conclusion

U-Net est une architecture puissante et polyvalente, idéale pour les modèles de diffusion grâce à sa capacité à gérer la reconstruction spatiale et à intégrer des informations contextuelles. Dans des modèles comme Stable Diffusion, U-Net permet la génération d'images de haute qualité en prédisant le bruit lors du processus de débruitage, en faisant un pilier des technologies modernes de génération d'images.

MNIST Diffusion

1 Introduction

Ce chapitre explique l'imlémentation d'un modèle de Denoising Diffusion Probabilistic Model (DDPM) pour générer des images du dataset **MNIST** en utilisant **PyTorch**. Il s'agit d'un modèle de diffusion probabiliste qui ajoute progressivement du bruit à des images (processus forward) et apprend à inverser ce processus pour générer de nouvelles images à partir de bruit pur (processus reverse).



2 Imports

Objectif

Cette cellule importe les bibliothèques nécessaires pour construire, entraîner, et évaluer un modèle de Denoising Diffusion Probabilistic Model (DDPM) sur le dataset MNIST.

Explication

— **torch** : Fournit les tenseurs et les outils pour les réseaux de neurones.

— **torch.nn** et **torch.nn.functional** : Définition des couches (convolutions, batch norm, etc.) et fonctions d'activation (ReLU, etc.).

— **torchvision** et **transforms** : Chargement et prétraitement du dataset MNIST.

— **DataLoader** : Création de batches pour l'entraînement et l'évaluation.

— **matplotlib.pyplot** et **numpy** : Visualisation des résultats et calculs numériques.

— **tqdm** : Barre de progression pour les boucles.

— **math** : Fonctions mathématiques (cos, log, etc.).

— **skimage.metrics.ssim** : Calcul de la métrique SSIM pour évaluer la qualité des images générées.

— **os** : Gestion des fichiers (sauvegarde/chargement du modèle).

Contexte

Ces imports préparent l'environnement pour toutes les étapes du DDPM : modélisation, données, entraînement, génération, et évaluation.
import torch import torch.nn as nn import torch.nn.functional as F import torchvision import torchvision.transforms as transforms from torch.utils.data import DataLoader import matplotlib.pyplot as plt import numpy as np from tqdm import tqdm import math from skimage.metrics import structural$_similarityasssimimportos$

3 Configuration du Device

Objectif

Détecter si un GPU est disponible pour accélérer les calculs, sinon utiliser le CPU.

Explication

La ligne `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")` vérifie la disponibilité de CUDA (GPU). Si disponible, `device="cuda"`, sinon `device="cpu"`.

Contexte

Le DDPM implique des calculs intensifs (convolutions, matrices). Utiliser un GPU (CUDA) accélère significativement l'entraînement et la génération.
device = torch.device("cuda" if torch.cuda.is$_available()else$"cpu")

4 Fonction `betas_for_alpha_bar`

Objectif

Calculer les paramètres β_t (variance du bruit) pour un schedule de diffusion cosinus, qui contrôle l'ajout progressif de bruit.

Explication

— La fonction prend :

  — `num_diffusion_timesteps` : Nombre de pas T.

  — `alpha_bar` : Fonction définissant $\bar{\alpha}_t$, le produit cumulatif des α_t.

  — `max_beta` : Valeur maximale de β_t (par défaut 0.999).

— Boucle sur i de 0 à $T-1$:

- Calcule $t_1 = i/T$ et $t_2 = (i+1)/T$.
- $\beta_i = \min(1 - \frac{\bar{\alpha}(t_2)}{\bar{\alpha}(t_1)}, \beta_{\max})$.

— Retourne un tenseur de β_t.

Formules mathématiques

Dans le DDPM, le schedule cosinus est défini par :

$$\bar{\alpha}_t = \cos^2\left(\frac{t+s}{T+s} \cdot \frac{\pi}{2}\right), \quad s = 0.008$$

$$\beta_t = \min\left(1 - \frac{\bar{\alpha}_{t+1}}{\bar{\alpha}_t}, \beta_{\max}\right)$$

Le paramètre s évite des β_t trop petits au début.

5 Configuration du Schedule de Diffusion

Objectif

Définir les paramètres du processus de diffusion ($T = 1000$, schedule cosinus) et précalculer les tenseurs pour optimiser le sampling.

Explication

— Définit $T = 1000$ (nombre de timesteps).

— Si `schedule_name="linear"`, utilise β_t linéaires de 0.0001 à 0.02.

— Sinon ("cosine"), utilise `betas_for_alpha_bar` avec :

$$\bar{\alpha}_t = \cos^2\left(\frac{t+0.008}{1.008} \cdot \frac{\pi}{2}\right)$$

— Calcule :

- $\alpha_t = 1 - \beta_t$
- $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$
- $\bar{\alpha}_{t-1}$ (décalé avec padding).
- $\sqrt{\bar{\alpha}_t}$, $\sqrt{1-\bar{\alpha}_t}$, $\sqrt{\frac{1}{\alpha_t}}$
- Variance postérieure : $\tilde{\beta}_t = \beta_t \cdot \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}$.

Formules mathématiques

— Processus forward :

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

— Variance postérieure :

$$\tilde{\beta}_t = \beta_t \cdot \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}$$

6 Fonction `extract`

Objectif

Extraire des valeurs d'un tenseur précalculé (comme $\bar{\alpha}_t$) pour un batch de timesteps t.

Explication

- Prend :
  - `a` : Tenseur précalculé (e.g., $\bar{\alpha}_t$).
  - `t` : Indices des timesteps.
  - `x_shape` : Forme de l'entrée pour broadcasting.
- Extrait `a[t]` et reshape pour correspondre à la forme de l'entrée (batch_size, 1, 1, ...).

Formules mathématiques

Aucune formule complexe ; utilisé pour broadcaster des scalaires comme $\sqrt{\bar{\alpha}_t}$ sur des tenseurs d'images.

Contexte

Simplifie le sampling forward/reverse en appliquant les coefficients à un batch d'images.

7 Fonction `q_sample` (Forward Diffusion)

Objectif

Ajouter du bruit à une image x_0 au timestep t selon le processus forward du DDPM.

Explication

- Si `noise=None`, génère $\epsilon \sim \mathcal{N}(0, I)$.
- Calcule :
$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$$
- Utilise `extract` pour obtenir $\sqrt{\bar{\alpha}_t}$ et $\sqrt{1 - \bar{\alpha}_t}$.

Formules mathématiques

$$q(x_t | x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t) I)$$

C'est une reparamétrisation pour sampler x_t directement sans itérer tous les timesteps.

8 Fonction `sinusoidal_time_embedding`

Objectif

Créer des embeddings positionnels pour les timesteps t, inspirés des transformers.

Explication

— Divise `dim` en deux pour sin et cos.
— Fréquences : $\exp(-\log(10000) \cdot \text{arange}(0, \text{half})/\text{half})$.
— Calcule $\sin(t \cdot \text{freqs})$ et $\cos(t \cdot \text{freqs})$, concatène.
— Pad si `dim` impair.

Formules mathématiques

$$\begin{cases} PE(t, 2i) = \sin\left(\frac{t}{10000^{2i/d}}\right), \\ PE(t, 2i+1) = \cos\left(\frac{t}{10000^{2i/d}}\right) \end{cases}$$

9 Classe `ResidualBlock`

Objectif

Définir un bloc résiduel avec conditionnement temporel pour le UNet.

Explication

— Structure :
  — Conv2d (3x3) + BatchNorm + ReLU.
  — Ajout de l'embedding temporel via une projection linéaire.
  — Conv2d (3x3) + BatchNorm + ReLU.
  — Connexion résiduelle (conv 1x1 si les canaux changent, sinon identité).

Formules mathématiques

$$h = \text{ReLU}(\text{BN}(\text{Conv}(x))) + \phi(t), \quad h = \text{ReLU}(\text{BN}(\text{Conv}(h))) + \text{res\_conv}(x)$$

où $\phi(t)$ est l'embedding temporel projeté.

10 Classe `SelfAttention`

Objectif

Ajouter un mécanisme de self-attention au bottleneck du UNet pour capturer des dépendances globales.

Explication

— Projection QKV via Conv2d 1x1.
— Reshape pour attention : [B, C, H, W] → [B, H*W, C].
— Attention multi-head : $Q@K^T$, softmax, @ V.
— Projection finale + résidu.

11 Classe UNet

Objectif

Définir le modèle principal : un UNet conditionné sur t pour prédire le bruit ϵ.

Explication

— **Time MLP** : Projette l'embedding temporel via Linear + SiLU + Linear.

— **Encoder** : ResidualBlock (1→128), MaxPool, ResidualBlock (128→256), MaxPool, ResidualBlock (256→512).

— **Bottleneck** : ResidualBlock + SelfAttention.

— **Decoder** : Up-conv + concaténation des skip connections + ResidualBlock.

— **Output** : Conv2d pour prédire le bruit (même forme que l'entrée).

Formules mathématiques

Le UNet approxime :

$$\epsilon_\theta(x_t, t)$$

Loss d'entraînement :

$$\mathbb{E}_{x_0,t,\epsilon}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2\right]$$

Contexte

Cœur du DDPM ; prédit le bruit pour débruiter itérativement les images.

12 Résultats

Résultats de la loss après 50 époques

Après l'entraînement du modèle sur 50 époques avec le dataset MNIST, la loss (MSE sur le bruit prédit) atteint une valeur de 0.03 à l'époque 50. Cela indique une bonne convergence, avec une diminution progressive de la loss au fil des époques.

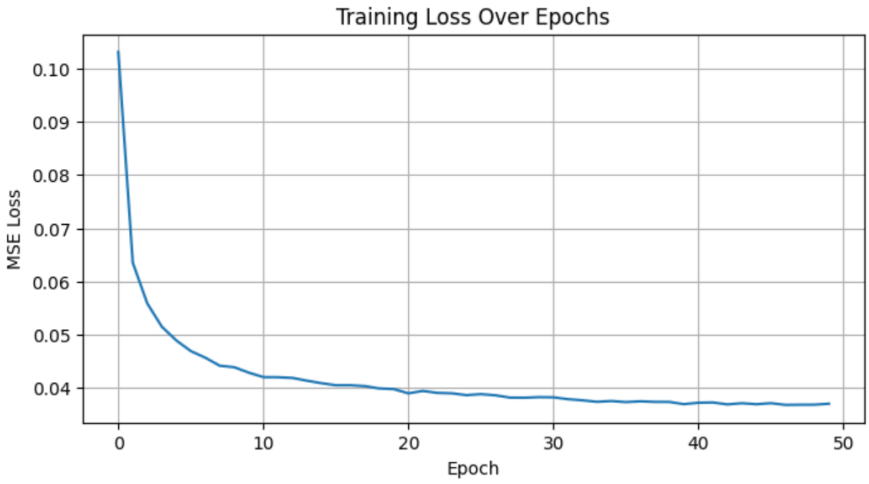


Fig. 4.1 Fonction loss

```
Training model...
Epoch 1/50: 100%|██████████| 469/469 [02:06<00:00,  3.72it/s]
Epoch 1/50, Loss: 0.1032
Epoch 2/50: 100%|██████████| 469/469 [02:10<00:00,  3.60it/s]
Epoch 2/50, Loss: 0.0635
Epoch 3/50: 100%|██████████| 469/469 [02:10<00:00,  3.58it/s]
```

Fig. 4.2 Epochs 1-3

```
Epoch 48/50: 100%|██████████| 469/469 [02:11<00:00,  3.57it/s]
Epoch 48/50, Loss: 0.0368
Epoch 49/50: 100%|██████████| 469/469 [02:11<00:00,  3.57it/s]
Epoch 49/50, Loss: 0.0368
Epoch 50/50: 100%|██████████| 469/469 [02:11<00:00,  3.57it/s]
```

Fig. 4.3 Epochs 48-50

12.1 Procédure backward des modèles de diffusion

La procédure backward, ou processus reverse, dans les modèles de diffusion probabilistes comme le DDPM, consiste à inverser le processus forward d'ajout de bruit. On commence par un bruit pur $x_T \sim \mathcal{N}(0, I)$ et on débruite itérativement pour obtenir une image x_0.
À chaque timestep t (de T à 1) :

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sqrt{\tilde{\beta}_t} z, \quad z \sim \mathcal{N}(0, I)$$

où $\epsilon_\theta(x_t, t)$ est le bruit prédit par le modèle UNet, et $\tilde{\beta}_t$ est la variance postérieure.
Ce processus permet de générer de nouvelles images à partir de bruit gaussien en apprenant à estimer le bruit ajouté à chaque étape.

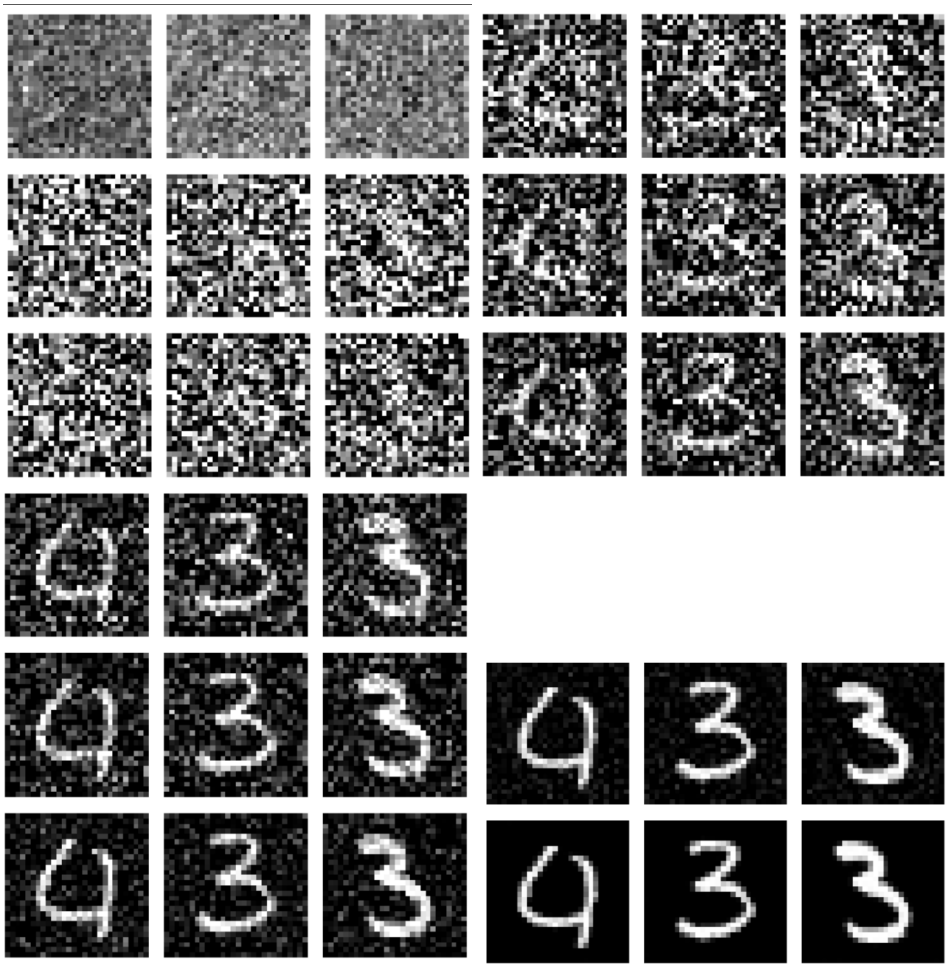


Fig. 4.4 Exemple de sorties obtenues

12.2 Exemples d'images générées par le modèle

Le modèle génère des images de chiffres manuscrits à partir de bruit pur. Voici un exemple de visualisation des étapes de génération (du bruit à l'image finale) pour 8 échantillons, avec des intervalles de 100 timesteps :

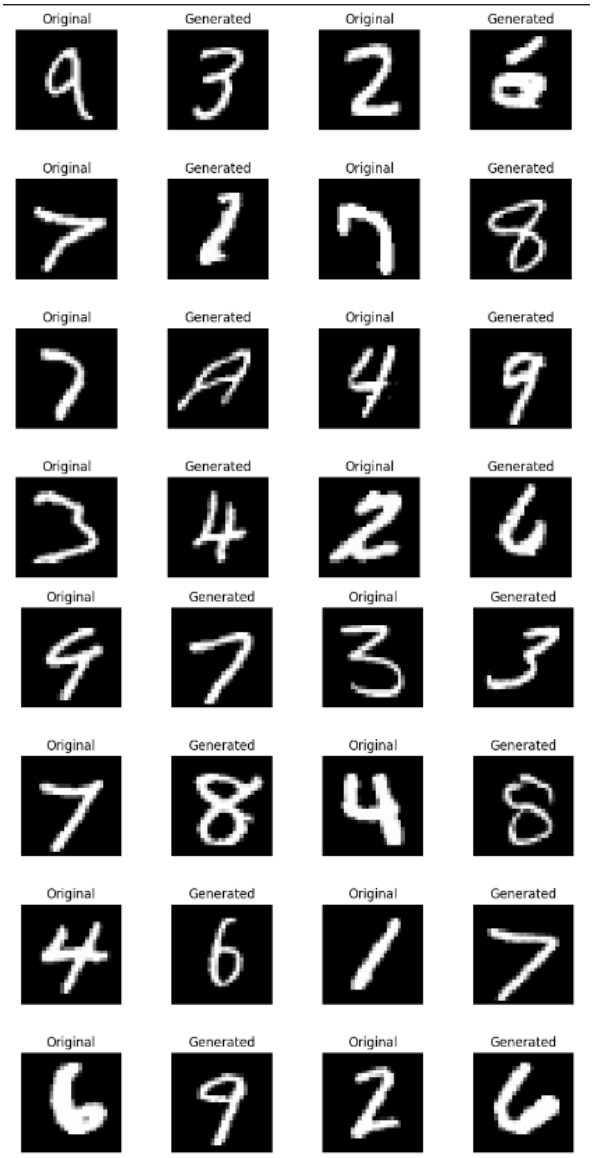


Fig. 4.5 Exemple des images générées

12.3 Résultats du calcul de SSIM

L'évaluation du modèle sur 64 images générées comparées à des images du dataset de test donne un score SSIM moyen de 0.3, ce qui suggère que la majorité des images générées ont un score SSIM dans cette plage. Cela indique que le modèle produit des images qui en moyenne similarité structurelle modérée avec les images réelles du dataset MNIST.

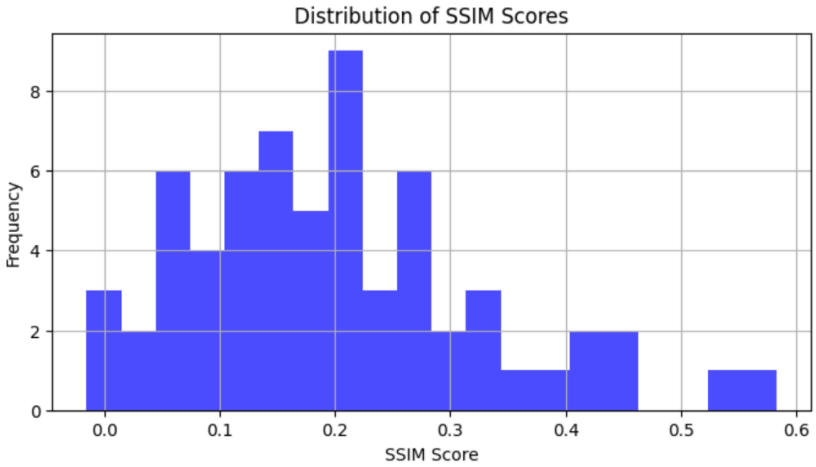


Fig. 4.6 Distribution de SSIM