

Réimplémentation de Glow sur MNIST

Modèles génératifs à flot (Normalizing Flows)

Lucas Collemare, Cyprien Deruelle, Antoine Malmezac
Leo Lopes, Yanis, Charles Prioux

30 janvier 2026

Résumé

Ce projet consiste à réimplémenter et à expliquer les fondements théoriques de Glow [7], une famille de modèles génératifs par flots (normalizing flows). Contrairement aux modèles adversariaux, Glow maximise une vraisemblance exacte : l'évaluation de $p_\theta(x)$ et l'inférence des variables latentes z sont toutes deux tractables et exactes. Nous détaillons la formule du changement de variable, la conception de transformations inversibles à jacobien triangulaire, puis les trois briques clés de Glow (ActNorm, convolution 1×1 inversible, couplage affine) ainsi que l'architecture multi-échelle. Nous présentons enfin une implémentation « Glow Mini » adaptée à MNIST (images $1 \times 28 \times 28$), et comparons deux stratégies de mélange de canaux : permutation aléatoire (baseline) versus convolution 1×1 inversible.

1 Introduction

Les modèles génératifs cherchent à apprendre une distribution de probabilité $p_\theta(x)$ sur des données x de très grande dimension (images, audio, texte), de façon à pouvoir évaluer la plausibilité d'un exemple, générer de nouveaux échantillons, ou encore encoder des données dans un espace latent utile pour l'analyse et l'édition. Cette tâche est difficile car les dépendances entre variables (pixels, échantillons audio, tokens) sont complexes, fortement non linéaires, et la dimension D de x est très élevée.

Les approches se répartissent classiquement entre :

- les modèles autoregressifs qui factorisent $p(x)$ en produit conditionnel et optimisent une vraisemblance exacte mais au prix d'une génération séquentielle (peu parallélisable) [11];
- les variational auto-encoders (VAE) qui maximisent une borne inférieure (ELBO) de la log-vraisemblance et offrent une génération parallélisable [8];
- les GAN qui apprennent par un jeu adversarial et produisent souvent de très bons échantillons visuels, mais sans densité explicite (pas de log-vraisemblance exacte) [4].

Les normalizing flows constituent une autre voie : ils construisent une bijection paramétrée f_θ entre l'espace des données et un espace latent gaussien simple. La clé est que la densité se transporte par la formule du changement de variable, ce qui rend possible l'optimisation directe et exacte de la log-vraisemblance, ainsi qu'une inférence de latents $z = f_\theta(x)$ exacte [9]. Glow [7] est une architecture de flot conçue pour être stable à entraîner et efficace, notamment grâce à ActNorm, une convolution 1×1 inversible qui généralise la permutation de canaux, et une architecture multi-échelle qui factorise progressivement les variables latentes.

Dans ce rapport, nous présentons d'abord les fondements théoriques des flots (Section 4), puis les composants de Glow (Section 5). Nous décrivons ensuite notre implémentation « Glow Mini » pour MNIST (Section ??) et discutons des résultats (Section 8).

2 Travaux connexes et positionnement

Il est utile de situer les flots parmi les grandes familles de modèles génératifs, car ils incarnent un compromis particulier entre qualité visuelle, vraisemblance et coût de génération.

2.1 Autoregressifs

Les modèles autoregressifs factorisent une distribution jointe en produit conditionnel :

$$p_{\theta}(x) = \prod_{d=1}^D p_{\theta}(x_d \mid x_{<d}).$$

Ils optimisent une vraisemblance exacte et obtiennent souvent d'excellents scores de log-vraisemblance. En revanche, la génération requiert l'échantillonnage séquentiel des D dimensions : pour des images, cela implique typiquement une boucle sur les pixels, difficile à paralléliser. PixelCNN et ses variantes illustrent ce paradigme [11].

2.2 VAE

Les VAE introduisent une variable latente z et maximisent une borne inférieure (ELBO) de la log-vraisemblance [8]. L'inférence de z est approximée par un encodeur $q_{\phi}(z \mid x)$, et l'optimisation conjugue reconstruction et régularisation (divergence au prior). Ces modèles sont parallélisables à l'échantillonnage, mais la borne peut être lâche, et l'équilibre entre termes dépend de choix de paramétrisation, de prior et de capacités du décodeur.

2.3 GAN

Les GAN apprennent par discrimination entre données réelles et synthétiques [4]. Ils produisent fréquemment des images très réalistes et des échantillons nets, mais la densité $p_{\theta}(x)$ n'est pas explicitement disponible : l'évaluation quantitative par log-vraisemblance est difficile, et l'inférence (encoder une image en latent) nécessite des architectures spécifiques. De plus, l'entraînement adversarial peut être instable (mode collapse, oscillations), ce qui a motivé de nombreuses variantes.

2.4 Flows

Les flows (NICE, RealNVP, Glow) maximisent une vraisemblance exacte, avec une inférence exacte $z = f_{\theta}(x)$ grâce à l'inversibilité [1, 2, 7]. Ils sont parallélisables à la fois en entraînement et en génération : une fois z tiré du prior, on applique une suite de transformations inversibles pour obtenir x . Le prix à payer est structurel : contraindre chaque transformation à être bijective et à jacobien tractable limite l'espace des architectures possibles, et, dans la pratique, les flows peuvent nécessiter davantage de calcul pour atteindre une qualité perceptive comparable aux meilleurs GAN. Des travaux ultérieurs (p.ex. Flow++ [5]) étudient des améliorations de la déquantification et des couplages pour mieux modéliser des distributions d'images discrètes.

3 Modélisation probabiliste et maximum de vraisemblance

Soit une distribution de données inconnue $p^\star(x)$ et un jeu de données i.i.d. $\mathcal{D} = \{x^{(i)}\}_{i=1}^N$. L'apprentissage par maximum de vraisemblance consiste à choisir des paramètres θ maximisant

$$\frac{1}{N} \sum_{i=1}^N \log p_\theta(x^{(i)}),$$

ou, de manière équivalente, minimisant la negative log-likelihood (NLL)

$$\mathcal{L}(\mathcal{D}) = -\frac{1}{N} \sum_{i=1}^N \log p_\theta(x^{(i)}).$$

Ce critère a plusieurs interprétations : il correspond à la minimisation de la divergence de Kullback–Leibler $\text{KL}(p^\star \| p_\theta)$ (à constante près), et il fournit une mesure quantitative de l'adéquation du modèle au jeu de données.

Dans les modèles génératifs à variables latentes, on introduit une variable z avec un prior simple $p(z)$ et un décodeur $p_\theta(x | z)$. Les flows adoptent une perspective particulière : au lieu de spécifier $p_\theta(x | z)$, on définit une transformation bijective $x = g_\theta(z)$ (et donc $z = f_\theta(x) = g_\theta^{-1}(x)$) qui transporte le prior $p(z)$ vers une densité sur x . La vraisemblance peut alors être évaluée exactement grâce à la formule du changement de variable.

4 Normalizing flows et changement de variable

4.1 Changement de variable

Considérons une transformation bijective et différentiable $f : \mathbb{R}^D \rightarrow \mathbb{R}^D$. Si $z = f(x)$ et si $p(z)$ est connu, alors la densité induite sur x est donnée par

$$p(x) = p(z) \left| \det \left(\frac{\partial z}{\partial x} \right) \right|. \quad (1)$$

En log, on obtient

$$\log p(x) = \log p(z) + \log \left| \det \left(\frac{\partial z}{\partial x} \right) \right|. \quad (2)$$

Le déterminant du jacobien mesure la variation locale de volume induite par f : une transformation qui « étire » l'espace augmente/diminue la densité en conséquence. Cette intuition relie les flows à l'idée de normalisation progressive : on part d'une distribution complexe $p^*(x)$, on applique f_θ pour obtenir une variable z qui se rapproche d'un prior simple (souvent gaussien), et le log-déterminant comptabilise précisément la « correction de volume » nécessaire pour conserver une densité normalisée.

Dans un cadre différentiable, la bijectivité impose aussi une contrainte topologique : la transformation f_θ ne peut pas « déchirer » l'espace, ni envoyer une région de volume non nul vers un ensemble de mesure nulle. C'est l'une des raisons pour lesquelles les flows privilégient des transformations lisses et inversibles (par ex. affines par morceaux via des couplages), et utilisent des astuces comme l'architecture multi-échelle pour gérer efficacement la dimension.

4.2 Flot composé

Un flow construit f comme composition de transformations inversibles

$$f = f_K \circ f_{K-1} \circ \dots \circ f_1.$$

En posant $h_0 = x$ et $h_k = f_k(h_{k-1})$ pour $k = 1, \dots, K$, on a $z = h_K$. La log-vraisemblance s'écrit alors comme somme de contributions :

$$\log p_\theta(x) = \log p(z) + \sum_{k=1}^K \log \left| \det \left(\frac{\partial h_k}{\partial h_{k-1}} \right) \right|. \quad (3)$$

Cette écriture est cruciale : elle permet de concevoir des briques f_k dont (i) l'inverse est facile à calculer, et (ii) le log-déterminant est peu coûteux.

4.3 Jacobien triangulaire et couplages

Un obstacle immédiat est que calculer un déterminant en dimension D coûte généralement $\mathcal{O}(D^3)$. Les flows contournent ce problème en utilisant des transformations dont le jacobien est triangulaire (ou bloc-triangulaire), auquel cas le déterminant est simplement le produit des éléments diagonaux et donc

$$\log |\det J| = \sum_{d=1}^D \log |J_{dd}|.$$

Les couches de couplage introduites dans NICE [1] et RealNVP [2] reposent précisément sur cette idée. Elles sont devenues un motif standard dans de nombreux flows modernes, car elles offrent trois propriétés essentielles :

Inversion facile : l'inverse se calcule par des additions et multiplications point-à-point

Log-déterminant simple : grâce à la structure triangulaire

Expressivité : la non-linéarité et la capacité du modèle proviennent du réseau NN interne, dont l'architecture peut être riche (convolutions, résidus, attention) tant qu'elle ne casse pas la forme de couplage.

Cette expressivité n'est toutefois pas « gratuite » : une seule couche de couplage modifie uniquement une partie des dimensions. Pour que toutes les dimensions puissent être transformées au fil des couches, on intercale des opérations de permutation/mélange (Section 5.2) et on empile de nombreux pas.

5 Briques de Glow

Glow [7] propose une architecture de flot profonde et stable en combinant trois types d'opérations au sein d'un flow step :

- une normalisation apprise par canal (ActNorm)
- un mélange réversible des canaux (permutation ou convolution 1×1 inversible)
- une couche de couplage (souvent affine)

Ces pas sont ensuite empilés dans une architecture multi-scale (Section 5.4).

5.1 ActNorm

ActNorm est une transformation affine par canal, appliquée à chaque position spatiale $y = (x + b) \odot \exp(s)$ où b et s sont des paramètres apprises (un par canal) et \odot désigne le produit terme à terme.

Le jacobien est diagonal par canal et répété pour toutes les positions (h, w) ; le log-déterminant s'obtient donc directement :

$$\log |\det J| = (H \cdot W) \sum_{c=1}^C s_c. \quad (4)$$

La difficulté pratique de Glow est l'entraînement de flots profonds sur de grandes images, où la taille de batch par GPU peut être très petite ce qui rend batch normalization instable. Glow remplace batch norm par ActNorm avec initialisation dépendante des données : les paramètres (b, s) sont initialisés de façon à ce que, sur un premier mini-batch, la sortie soit approximativement centrée et de variance unitaire [7, 10]. Ensuite, ActNorm se comporte comme une couche standard avec paramètres appris.

Sur le plan numérique, ActNorm est particulièrement adapté aux flows : il est strictement inversible, il évite l'introduction de bruit d'estimation de moments, et il fournit un contrôle direct sur les échelles internes, ce qui améliore le conditionnement des jacobiens et la stabilité des gradients dans des compositions profondes.

5.2 Convolution 1×1 inversible

Les couches de couplage modifient seulement une partie des canaux à la fois ; sans opération de mélange entre deux couplages, certaines dimensions resteraient isolées. Les flots précédents utilisaient une permutation de canaux fixe (par ex. inversion de l'ordre ou permutation aléatoire) [2]. Glow généralise cette idée en apprenant un mélange linéaire réversible via une convolution 1×1 : $\forall(h, w), \quad y_{:,h,w} = W x_{:,h,w}$ où $W \in \mathbb{R}^{C \times C}$ est une matrice inversible partagée pour toutes les positions spatiales. Cette opération est bijective dès que W l'est, et son inverse s'obtient par W^{-1} . Le log-déterminant s'écrit simplement car le jacobien global est bloc-diagonal (un bloc W par position spatiale) :

$$\log |\det J| = (H \cdot W) \log |\det(W)|. \quad (5)$$

Une manière formelle de le voir est de vectoriser l'entrée $x \in \mathbb{R}^{C \times H \times W}$ en un vecteur de dimension $C \cdot H \cdot W$. La transformation devient alors linéaire avec une matrice jacobienne J égale à un produit de Kronecker $J = I_{H \cdot W} \otimes W$ où $I_{H \cdot W}$ est l'identité de taille $(H \cdot W)$. On en déduit $\det(J) = \det(W)^{H \cdot W}$, d'où l'Equation 5.

Glow propose en outre une paramétrisation LU de W pour réduire le coût du déterminant (de $\mathcal{O}(C^3)$ à $\mathcal{O}(C)$) [7]. Dans notre implémentation MNIST, C reste faible (4 ou 8) et nous pouvons calculer $\log |\det(W)|$ directement.

Enfin, Glow initialise W comme une matrice de rotation (orthogonale) afin d'éviter des expansions/ contractions trop fortes au départ : $\log |\det(W)| = 0$ à l'initialisation. Dans notre version, nous utilisons aussi une initialisation orthogonale (QR), ce qui simplifie l'optimisation en début d'entraînement.

5.3 Couplage affine

Une couche de couplage affine partitionne les canaux en deux blocs (x_a, x_b) . Un petit réseau neuronal (souvent convolutionnel) calcule à partir de x_a deux tenseurs (s, t) , puis transforme seulement x_b :

$$(s, t) = \text{NN}(x_a), \quad (6)$$

$$y_a = x_a, \quad (7)$$

$$y_b = x_b \odot \exp(s) + t. \quad (8)$$

Cette forme implique que l'inverse est immédiat :

$$x_b = (y_b - t) \odot \exp(-s), \quad x_a = y_a.$$

Le jacobien est triangulaire et sa diagonale contient $\exp(s)$ sur la partie transformée, on obtient donc

$$\log |\det J| = \sum s, \quad (9)$$

où la somme porte sur toutes les dimensions de s (canaux et positions). Le couplage additif de NICE est un cas particulier ($s = 0$), dont le log-déterminant est nul [1].

Glow propose une initialisation à zéro du dernier bloc de la fonction NN afin que la couche de couplage démarre proche de l'identité, ce qui stabilise l'entraînement de réseaux très profonds [7]. Dans notre implémentation, la dernière convolution de NN est initialisée à zéro, et nous « bornons » s via une non-linéarité de type tanh (clamp) pour éviter des échelles extrêmes en début d'entraînement. Ce type de bornage est courant en pratique : puisque $y_b = x_b \odot \exp(s) + t$, des valeurs grandes de s peuvent provoquer des exponentielles extrêmes, amplifier les activations et déstabiliser le calcul du log-déterminant. Limiter s contraint temporairement la dynamique de volume et favorise une optimisation plus régulière.

5.4 Architecture multi-échelle et squeeze

Glow combine des pas de flot avec une architecture dite multiscale [2, 7]. L'idée est de ne pas conserver toute la représentation jusqu'au bout : après un certain nombre de transformations, on « factorise » une partie des variables comme latents z_1 et l'on continue à transformer le reste. Cette stratégie réduit le coût de calcul, améliore le conditionnement, et rend l'espace latent plus structuré.

Une opération centrale est le squeeze $(B, C, H, W) \rightarrow (B, 4C, H/2, W/2)$ qui réorganise chaque bloc spatial 2×2 en canaux. Elle est parfaitement réversible (une simple permutation d'indices), donc son déterminant a un module 1 et ne contribue pas au log-déterminant.

Conceptuellement, le squeeze change la « granularité » spatiale : les dépendances locales sur une grille fine peuvent être réexprimées comme des dépendances entre canaux sur une grille plus grossière. Cela rend le couplage plus efficace car un réseau convolutionnel de petite taille de noyau (par ex. 3×3) voit un voisinage spatial plus large en termes de pixels originaux après quelques squeeze.

Prior et factorisation. Dans Glow, la factorisation multi-échelle s'accompagne d'un prior conditionnel : lorsqu'on « sort » une partie des variables en tant que z_ℓ , on peut paramétrer une gaussienne dont les paramètres dépendent des activations restantes. Cela enrichit le prior sans casser la tractabilité. Dans « Glow Mini », nous adoptons un prior standard $\mathcal{N}(0, I)$ indépendant sur (z_1, z_2) afin de garder une implémentation compacte. Cette simplification est adaptée à MNIST, mais pourrait être assouplie pour des données plus complexes.

5.5 Complexité, parallélisme et mémoire

Un aspect central des flows est l'équilibre entre tractabilité et coût. Chaque pas de flot doit fournir trois opérations : une transformation avant, son inverse, et un log-déterminant. Cela impose des contraintes d'architecture qui ont un impact direct sur la complexité.

Coût des briques. Sur une activation de taille $B \times C \times H \times W$:

- ActNorm effectue une opération affine par canal et a un coût $\mathcal{O}(BCHW)$; le log-déterminant se calcule en $\mathcal{O}(C)$ (puis multiplié par $H \cdot W$).
- la convolution 1×1 inversible a un coût de convolution $\mathcal{O}(BHW C^2)$, car chaque position applique une matrice dense $C \times C$; le calcul de $\log |\det(W)|$ coûte $\mathcal{O}(C^3)$ si l'on utilise un déterminant direct, mais peut être ramené à $\mathcal{O}(C)$ avec la paramétrisation LU [7].
- le couplage affine a un coût dominé par le réseau NN, typiquement $\mathcal{O}(BHW C \cdot C_{\text{hidden}})$ selon l'architecture, tandis que le log-déterminant est une simple somme de s .

Sur MNIST, C reste faible et le coût est dominé par les convolutions du réseau de couplage. Sur des images couleur haute résolution, l'intérêt du multi-scale et des optimisations (LU, architecture de NN) devient beaucoup plus marqué.

Contrairement aux modèles autoregressifs, l'échantillonnage d'un flow est parallélisable : toutes les opérations d'un pas de flot s'exécutent en une passe (convolutions, opérations point-à-point), puis on répète pour K pas. Le coût de génération est donc proportionnel à la profondeur (nombre de pas), mais pas à la dimension D de manière séquentielle. C'est l'un des arguments majeurs de Glow : obtenir une log-vraisemblance exacte tout en conservant une synthèse efficace sur matériel parallèle [7].

Les réseaux inversibles offrent aussi un avantage potentiel en mémoire : au lieu de stocker toutes les activations intermédiaires pour le backpropagation, on peut les « reconstruire » à la volée en inversant les transformations, ce qui rend la mémoire asymptotiquement indépendante de la profondeur [3]. En pratique, exploiter pleinement cette propriété demande une implémentation attentive (contrôle des opérations non inversibles, gestion des états, stabilité numérique). Même sans cette optimisation, l'inversibilité fournit un outil de diagnostic puissant : on peut vérifier systématiquement que la composition inverse restitue bien l'entrée, et localiser les erreurs si une brique casse la bijection.

6 Pré-traitement des images discrètes et métrique en bits/dim

Les flots sont définis sur des variables continues. Or une image MNIST est discrète (pixels entiers $0, \dots, 255$). Comme dans les travaux de flows sur images [2, 7], on approxime ce problème en déquantifiant puis en appliquant une transformation bijective vers \mathbb{R} .

6.1 Déquantification uniforme

Une première étape consiste à transformer un pixel discret $x \in \{0, \dots, 255\}$ en une variable continue sur un petit intervalle. En pratique, on normalise d’abord dans $[0, 1]$, puis on ajoute un bruit uniforme :

$$\tilde{x} = \frac{x + u}{256}, \quad u \sim \mathcal{U}(0, 1).$$

Cela remplace une masse discrète par une densité continue. Dans l’évaluation bits/dim, la constante de quantification apparaît sous la forme d’un terme $-D \log 256$ (voir ci-dessous). Intuitivement, cette correction vient du fait qu’une densité continue $p(\tilde{x})$ est une quantité « par unité de volume ». Pour obtenir la probabilité d’un bin discret (un pixel entier), on intègre la densité sur un intervalle de taille $1/256$; pour des pixels indépendants, cela induit un facteur $(1/256)^D$ et donc un terme $-D \log 256$ en log. La déquantification uniforme est une approximation simple ; des variantes apprises (déquantification variée/conditionnelle) peuvent améliorer la modélisation des images discrètes, comme étudié dans Flow++ [5].

6.2 Transformation logit

Pour travailler sur \mathbb{R} , on applique ensuite une transformation logit (bijection $(0, 1) \rightarrow \mathbb{R}$), en évitant les bords 0 et 1 via un petit α :

$$x' = \alpha + (1 - 2\alpha)\tilde{x}, \quad y = \log \frac{x'}{1 - x'}.$$

Le log-déterminant associé se calcule explicitement et s’ajoute à la log-vraisemblance du flot (Equation 3). Le paramètre α évite l’évaluation de $\log(0)$ et contrôle le compromis entre « fidélité » aux bords (pixels proches de 0 ou 1) et stabilité numérique. Dans notre expérimentation, $\alpha = 0.01$ suit une pratique courante des implémentations Glow : suffisamment petit pour ne pas écraser les extrêmes, suffisamment grand pour empêcher des jacobiens trop singuliers.

6.3 Bits par dimension

La métrique bits per dimension (bpd) est une NLL normalisée et exprimée en bits :

$$\text{bpd}(x) = -\frac{1}{D \log 2} (\log p_\theta(x) - D \log 256),$$

où $D = C \cdot H \cdot W$. Le terme $D \log 256$ correspond à la conversion entre une densité continue sur $[0, 1]^D$ et une vraisemblance discrète sur 256 valeurs par pixel. Cette métrique permet de comparer les modèles sur des jeux d’images standards.

7 Implémentation : Glow Mini sur MNIST

7.1 Données

Nous utilisons MNIST (`torchvision.datasets.MNIST`) avec une séparation train/validation de 92%/8% (environ 55k/5k). Les images sont converties en tenseurs dans $[0, 1]$ puis pré-traitées par déquantification uniforme et transformation logit (Section 6) avec $\alpha = 0.01$.

7.2 Architecture « Glow Mini »

Pour MNIST, nous adoptons une architecture multi-échelle à deux niveaux :

- **Niveau 1** : squeeze $(1, 28, 28) \rightarrow (4, 14, 14)$, puis K pas de flot. Ensuite on sépare z_1 (la moitié des canaux) et on continue avec le reste.
- **Niveau 2** : squeeze du reste $(2, 14, 14) \rightarrow (8, 7, 7)$, puis K pas de flot. Le tenseur final constitue z_2 .

Le prior factorise $p(z) = \mathcal{N}(0, I)$ sur (z_1, z_2) . Dans notre configuration, $z_1 \in \mathbb{R}^{2 \times 14 \times 14}$ et $z_2 \in \mathbb{R}^{8 \times 7 \times 7}$, ce qui représente au total $2 \cdot 14 \cdot 14 + 8 \cdot 7 \cdot 7 = 784$ variables, cohérent avec les $28 \cdot 28$ pixels.

Un pas de flot (`FlowStep`) est défini comme :

$$\text{ActNorm} \rightarrow \text{Mixer} \rightarrow \text{AffineCoupling}.$$

Le `Mixer` est soit une permutation aléatoire fixe des canaux, soit une convolution 1×1 inversible (Equation 5).

Chaque brique (`ActNorm`, `mixer`, `couplage`) renvoie son propre log-déterminant, qui s'additionne selon l'Equation 3. Le modèle complet renvoie donc $\log p_\theta(y)$ pour l'entrée pré-traitée $y \in \mathbb{R}^{1 \times 28 \times 28}$. La log-vraisemblance finale sur les pixels $x \in [0, 1]$ s'obtient en ajoutant le log-déterminant du pré-traitement logit. Cette modularité est une des forces des flows : on peut vérifier l'inversibilité « brique par brique » et tracer d'où provient le gain/perte de densité.

7.3 Réseau de couplage

La fonction NN utilisée dans le couplage affine est un petit réseau convolutionnel :

$$\text{Conv}(3 \times 3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(1 \times 1) \rightarrow \text{ReLU} \rightarrow \text{Conv}(3 \times 3).$$

La dernière convolution est initialisée à zéro pour commencer proche de l'identité. Pour stabiliser l'échelle s , nous appliquons un clamp via

$$s \leftarrow c \tanh\left(\frac{s}{c}\right),$$

avec $c = 2$.

7.4 Entraînement et EMA

Nous entraînons les modèles en maximisant la log-vraisemblance exacte (équivalent à minimiser la NLL), en utilisant Adam [6] avec un warmup linéaire du taux d'apprentissage sur 1000 itérations, et un gradient clipping (norme ≤ 5). Les hyperparamètres utilisés sont :

Paramètre	Valeur
Profondeur par niveau (K)	16
Largeur du réseau de couplage	256
Batch size	256
Époques	35
Taux d'apprentissage	10^{-3} (warmup 1000 pas)
Weight decay	10^{-5}
EMA (décroissance)	0.999
Température d'échantillonnage	0.7

Nous appliquons donc une EMA, qui agit comme un lissage temporel de l'optimisation et améliore la qualité des échantillons, lors de l'évaluation et des visualisations.

Pour échantillonner, on tire $z \sim \mathcal{N}(0, I)$ puis on applique l'inverse g_θ . On introduit parfois une température τ en échantillonnant $z \sim \mathcal{N}(0, \tau^2 I)$. Réduire τ concentre les latents près de l'origine, ce qui tend à produire des échantillons plus « moyens » et souvent plus lisibles pour MNIST ; augmenter τ accroît la diversité mais peut aussi générer des formes atypiques. Nous utilisons $\tau = 0.7$.

Notre objectif étant pédagogique et centré sur MNIST, plusieurs simplifications sont faites par rapport à Glow sur grandes images [7] :

- Seulement deux niveaux multi-échelle (au lieu de plusieurs)
- Prior standard gaussien indépendant sur (z_1, z_2) , sans prior conditionnel paramétré
- Calcul direct du déterminant de W (sans paramétrisation LU), car C est petit
- Réseau de couplage relativement léger (quelques convolutions et ReLU), suffisant pour MNIST

Ces choix réduisent le coût de calcul et clarifient les mécanismes, tout en conservant l'essence de Glow.

8 Résultats et analyse

Nous comparons deux modèles entraînés avec la même architecture et les mêmes hyperparamètres, mais avec un mélange de canaux différent :

- **Glow Mini (conv1x1)** : convolution 1×1 inversible apprise
- **Glow Mini (permute)** : permutation aléatoire fixe des canaux

8.1 Comparaison quantitative (bpd)

Sur le split de validation (environ 5k images), nous obtenons :

Modèle	bpd (validation, EMA)	Gain
Permutation	1.2192	–
Conv 1×1 inversible	1.1917	–0.0275

La convolution 1×1 inversible améliore donc la vraisemblance (bpd plus faible), ce qui est cohérent avec l’objectif de Glow : apprendre un mélange riche des canaux plutôt qu’une permutation fixe [7]. L’écart de ≈ 0.0275 bpd peut sembler modeste, mais il est significatif car la bpd est une moyenne sur $D = 784$ dimensions : il correspond à une différence moyenne de log-vraisemblance de l’ordre de $0.0275 \times D \log 2 \approx 14.9$ nats par image (ordre de grandeur). Autrement dit, le modèle conv 1×1 assigne en moyenne une densité notablement plus élevée aux images de validation.

8.2 Échantillonnage

La Figure 1 montre des échantillons générés par les deux variantes (température 0.7, EMA).

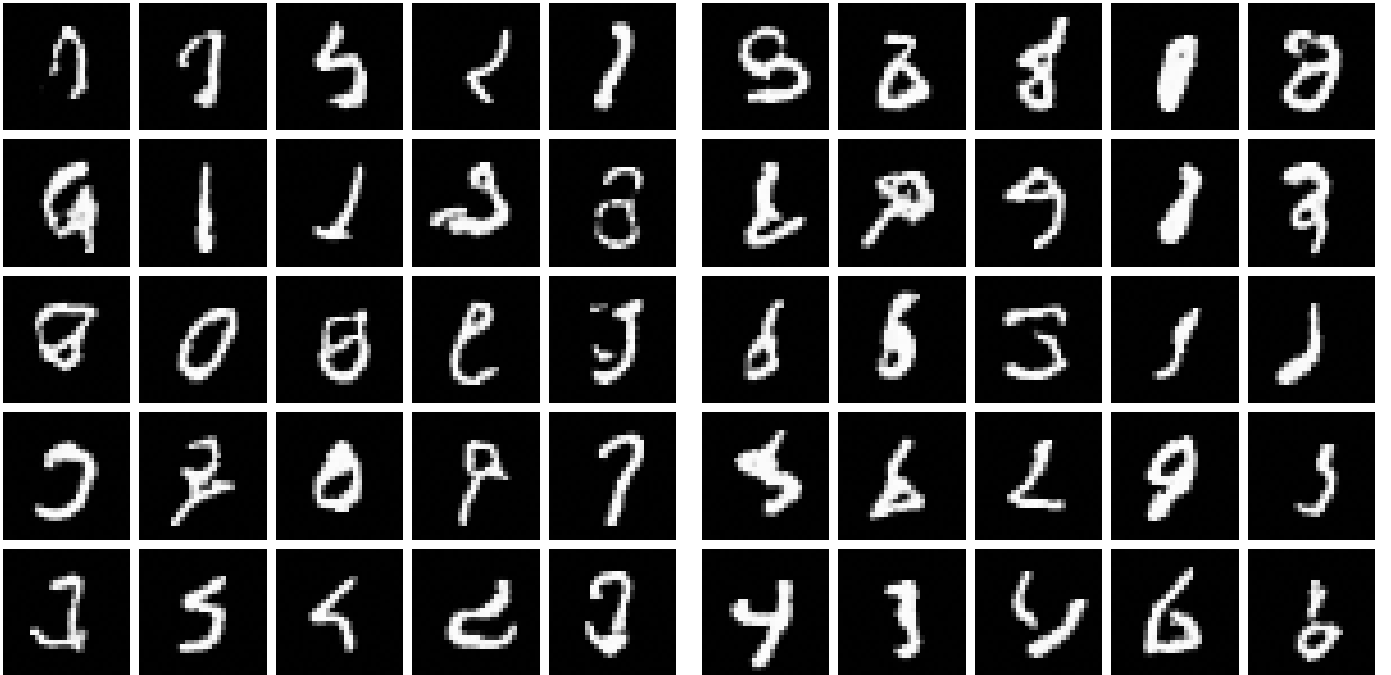


Figure 1: Échantillons MNIST générés (gauche : conv 1×1 inversible, droite : permutation).

Qualitativement, les deux modèles produisent des chiffres plausibles ; la variante conv 1×1 tend à mieux couvrir certaines variations d’épaisseur et de style, ce qui reflète son meilleur score bpd. Il faut toutefois rappeler qu’un score de log-vraisemblance optimal ne garantit pas toujours la meilleure « qualité perceptive » sur des datasets complexes : sur MNIST, cette tension est limitée, mais sur des images naturelles, des modèles différents peuvent échanger « finesse » contre « couverture ». Glow est intéressant précisément parce qu’il montre qu’un flow entraîné à la vraisemblance peut néanmoins produire des échantillons réalistes sur des datasets d’images plus difficiles [7].

8.3 Reconstruction et inversibilité

Un avantage majeur des flows est l’inversibilité exacte : encoder puis décoder doit reconstruire l’entrée (à la précision numérique et au bruit de déquantification près). La Figure 2 empile une ligne d’images originales et une ligne de reconstructions correspondantes.



Figure 2: Reconstructions (haut : originaux, bas : reconstructions). Gauche : conv 1×1 inversible. Droite : permutation.

On observe que les reconstructions sont très proches, confirmant que l’implémentation inverse bien chacune des briques (ActNorm, mélange, couplage, squeeze). En pratique, la reconstruction n’est pas strictement parfaite pour deux raisons :

1. la déquantification ajoute un bruit uniforme, ce qui rend le pré-traitement non strictement inversible au niveau des pixels discrets ;
2. les opérations flottantes (exp/log, inversion de matrice) introduisent de petites erreurs numériques.

Malgré cela, le fait que les reconstructions soient visuellement quasi identiques est un excellent test de cohérence de l’inversion.

8.4 Interpolation dans l’espace latent

L’espace latent d’un flow est « aligné » avec le prior : une interpolation linéaire entre latents correspond à un chemin continu dans l’espace des images décodées. La Figure 3 présente une interpolation linéaire entre deux images de validation, via interpolation des deux niveaux (z_1, z_2).

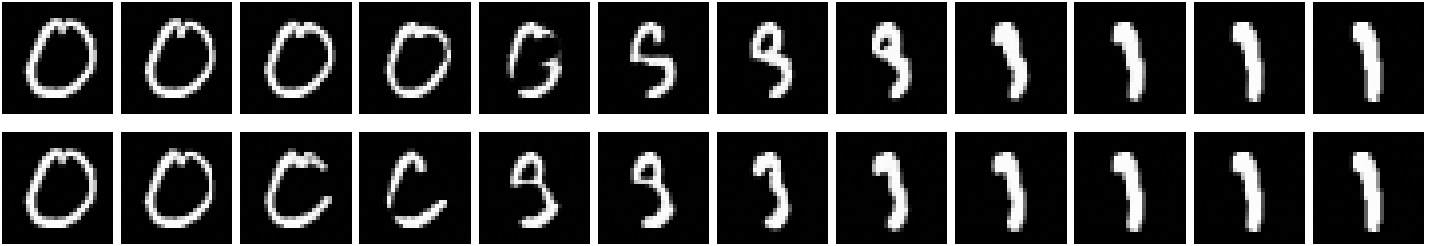


Figure 3: Interpolation linéaire dans l’espace latent (haut : conv 1×1 inversible, bas : permutation).

Le multi-scale suggère une interprétation : z_1 (niveau 1) capture des aspects plus « globaux » (structure grossière), tandis que z_2 (niveau 2) encode des détails plus fins. Une analyse plus approfondie peut interpoler séparément z_1 et z_2 pour visualiser ces rôles. Dans Glow sur images naturelles, cette structuration multi-échelle est utilisée pour des manipulations d’attributs : modifier certaines directions dans l’espace latent peut correspondre à changer des facteurs sémantiques (expression du visage, couleur des cheveux, etc.) [7]. Sur MNIST, l’analogie serait la modification de l’épaisseur, de l’inclinaison ou du style d’écriture.

8.5 Pourquoi le mélange conv 1×1 aide-t-il ?

Dans un couplage, seule une moitié des canaux est transformée ; l’autre moitié sert d’entrée à NN. Sans mélange, la même partition des canaux se répéterait et limiterait la capacité expressive. Une permutation fixe mélange déjà l’information, mais de manière non apprenable : elle impose une structure arbitraire. La convolution 1×1 inversible, au contraire, apprend un changement de base optimal (au sens de la vraisemblance) à chaque pas de flot, ce qui facilite la propagation de l’information et la modélisation de corrélations inter-canaux [7]. Dans notre cas, l’effet est d’autant plus visible que les canaux sont peu nombreux : après un squeeze, on n’a que 4 canaux, et un simple shuffle peut ne pas suffire à « désentrelacer » les dépendances utiles pour le couplage. Une matrice W apprise peut mettre en évidence des combinaisons linéaires (par exemple des contrastes locaux) qui rendent la tâche de NN plus simple.

8.6 Limites expérimentales

Nos expériences se limitent à MNIST et à une architecture compacte. Bien que cela suffise pour illustrer les mécanismes de Glow, plusieurs facteurs peuvent influencer les scores :

- la taille du réseau de couplage et la profondeur K (capacité) ;
- le choix de α et la stratégie de déquantification (uniforme vs apprise) ;
- l’usage d’un prior conditionnel (au lieu d’une gaussienne standard) ;
- des détails d’optimisation (schedule de LR, EMA, normalisation, clamp).

Sur des datasets plus difficiles (CIFAR-10, CelebA-HQ), ces choix deviennent critiques, et Glow inclut des raffinements supplémentaires (plus de niveaux, paramétrisation LU, réseau de prior) [7].

8.7 Exploration de l’espace latent

Pour toute image x , on calcule directement $z = f_\theta(x)$, puis toute modification $z \mapsto z'$ peut être visualisée en décodant $x' = g_\theta(z')$. Nous exploitons cette propriété pour analyser la structure des latents (z_1, z_2) du modèle conv 1×1 (avec EMA), en travaillant sur un sous-ensemble de validation de taille $N = 4800$.

8.7.1 Projections 2D/3D (PCA, UMAP)

On commence par concaténer les latents en un vecteur de dimension 784 afin d’appliquer des méthodes de projection. La Figure 4 montre deux projections 2D des latents colorées par label MNIST : PCA (linéaire) et UMAP (non linéaire). La PCA révèle une structure globale relativement continue, UMAP met davantage en évidence des voisinages locaux (styles proches) mais ne produit pas nécessairement des clusters parfaitement séparés sur MNIST.

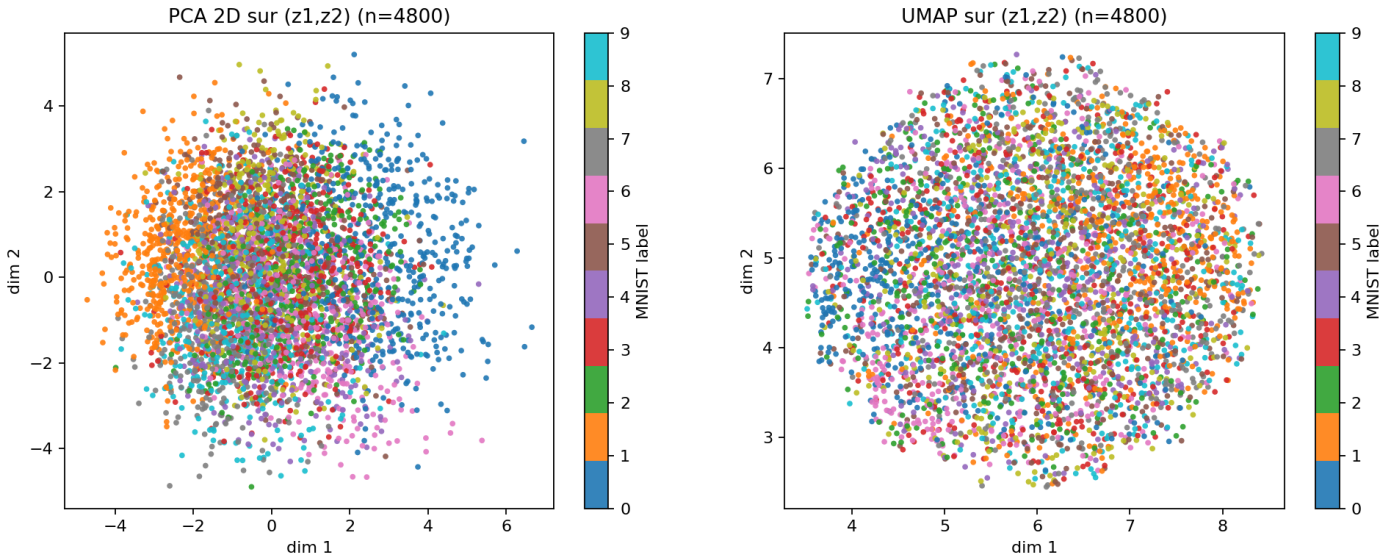


Figure 4: Projections 2D des latents concaténés (gauche : PCA 2D ; droite : UMAP 2D), colorées par label MNIST.

La Figure 5 montre la PCA en 3 dimensions ; elle est utile pour vérifier si la structure observée en 2D résulte d’une perte d’information trop forte ou si elle se maintient dans un espace de dimension légèrement supérieure.

PCA 3D sur (z1,z2) (n=4800)

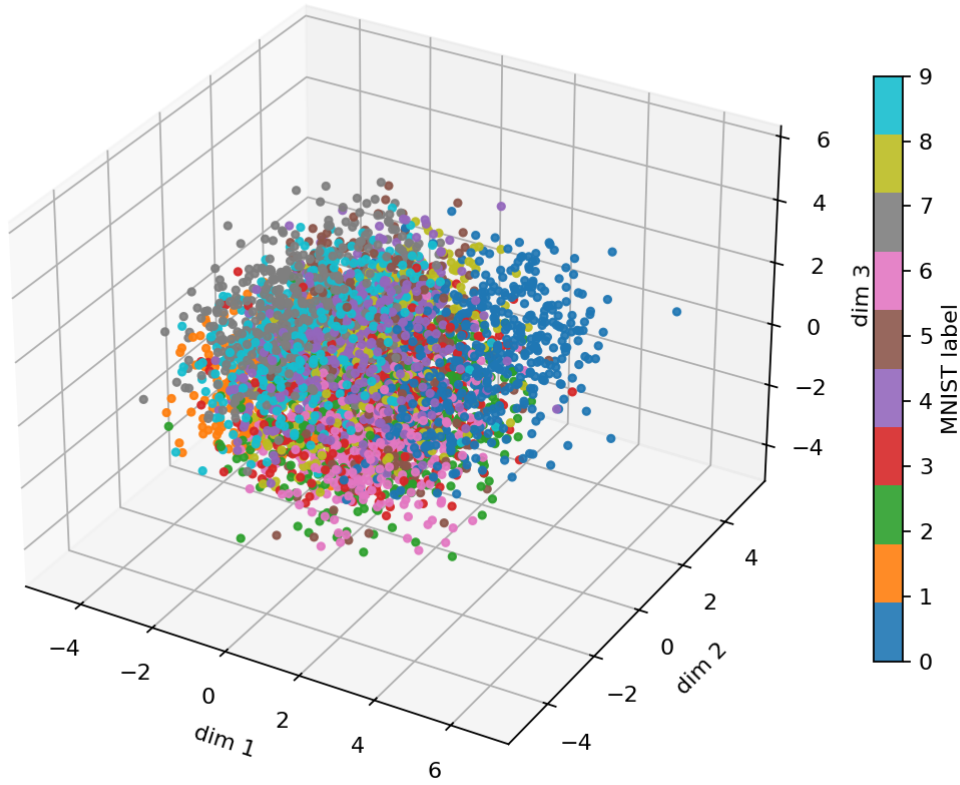


Figure 5: Projection PCA 3D des latents concaténés, colorée par label MNIST.

8.7.2 Adéquation au prior : moyennes et variances

Comme le prior est gaussien standard, une vérification simple consiste à regarder la moyenne et l'écart-type par dimension des latents inférés. Dans notre cas, la moyenne globale est proche de 0 et l'écart-type global proche de 1 ; néanmoins, certaines dimensions peuvent être légèrement décalées ou plus/moins dispersées. La Figure 6 montre les histogrammes des moyennes et des écarts-types par dimension.

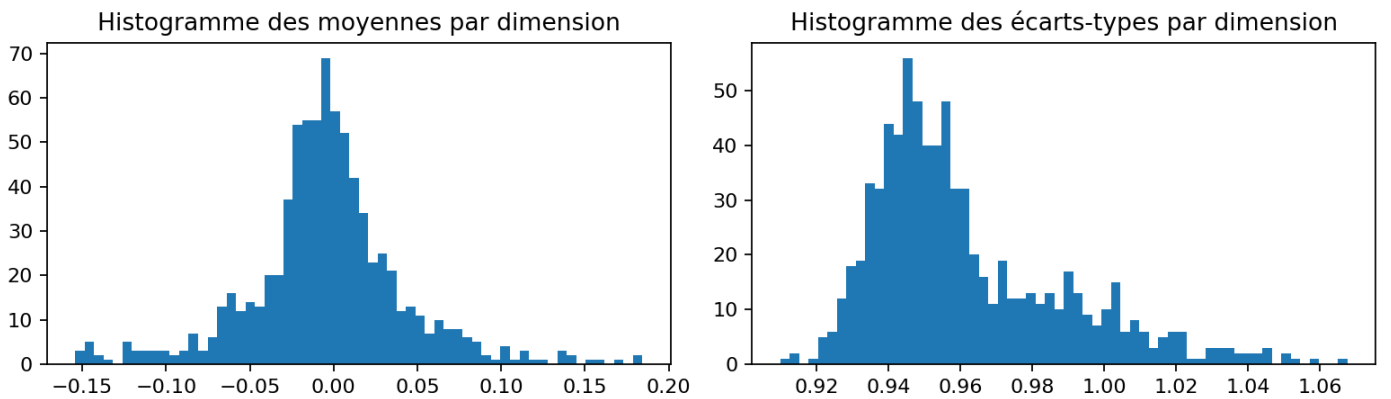


Figure 6: Histogrammes des moyennes et des écarts-types par dimension dans l'espace latent concaténé.

8.7.3 Traversées directionnelles (axes PCA)

Pour relier des directions latentes à des variations visuelles, on peut effectuer des traversées le long d'axes de PCA : on part du latent moyen μ et on déplace $z = \mu + \lambda v$ pour $\lambda \in [-3\sigma, 3\sigma]$, où v est une composante principale et σ la racine de la variance expliquée associée. Les Figures 7 et 8 illustrent deux traversées : sur MNIST, elles correspondent typiquement à des changements continus de style.

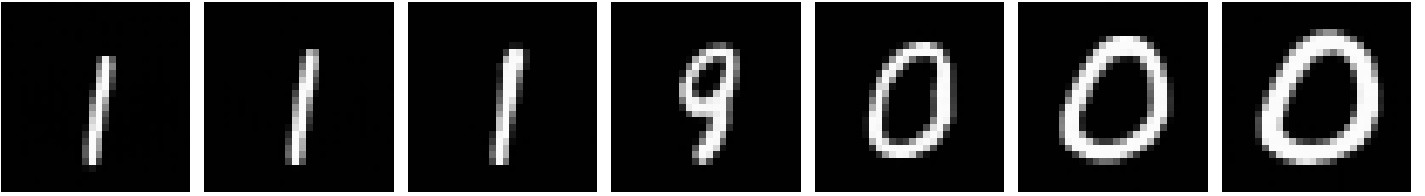


Figure 7: Traversée le long de l'axe PCA 0 (de $\mu - 3\sigma$ à $\mu + 3\sigma$).

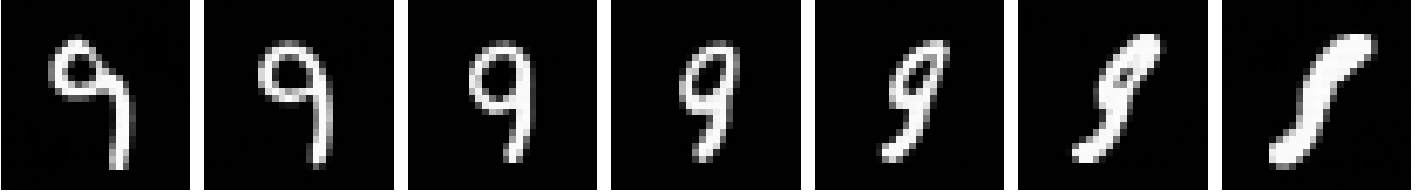


Figure 8: Traversée le long de l'axe PCA 1 (de $\mu - 3\sigma$ à $\mu + 3\sigma$).

8.7.4 Interpolations : linéaire, SLERP, et multi-échelle

Une autre exploration naturelle est d'interpoler entre deux images en espace latent. La Figure 9 rappelle les deux images de départ (réelles).

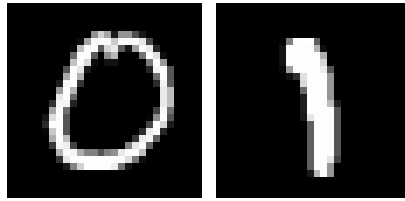


Figure 9: Deux images de validation utilisées comme points extrêmes pour les interpolations latentes.

La Figure 10 compare une interpolation linéaire et une interpolation sphérique (SLERP) sur (z_1, z_2) . L'interpolation linéaire est simple mais tend à réduire la norme au milieu ; SLERP maintient une norme plus constante et suit mieux la géométrie du prior gaussien.



Figure 10: Interpolation latente entre deux exemples (gauche : linéaire ; droite : SLERP) sur (z_1, z_2) .

Enfin, l'architecture multi-scale suggère d'interpoler séparément z_1 et z_2 . La Figure 11 illustre ce point : en interpolant seulement z_1 (en gardant z_2 fixe), on obtient des variations plus « grossières » et stables ; en interpolant seulement z_2 , on obtient des changements plus fins.



Figure 11: Interpolation multi-échelle (gauche : interpolation sur z_1 seul ; droite : interpolation sur z_2 seul).

9 Conclusion

Nous avons présenté les fondements théoriques des modèles génératifs par flots, puis les innovations de Glow : Act-Norm (stabilisation via initialisation dépendante des données), convolution 1×1 inversible (mélange apprenable et bijectif), et empilement dans une architecture multi-échelle. Nous avons ensuite implémenté une version « Glow Mini » adaptée à MNIST et comparé deux stratégies de mélange. Sur la validation, la convolution 1×1 inversible obtient une meilleure vraisemblance (bpd ≈ 1.1917 contre ≈ 1.2192 pour la permutation), et les visualisations confirment la capacité du modèle à générer, reconstruire et interpoler de manière cohérente.

Comme perspectives, on peut augmenter le nombre de niveaux multi-scale, adopter la paramétrisation LU pour W lorsque C devient grand, comparer différentes stratégies de déquantification (uniforme vs. déquantification apprise), et analyser plus finement l'espace latent (PCA/t-SNE/UMAP, manipulations directionnelles) pour relier dimensions latentes et attributs visuels.

Références

- [1] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. In *International Conference on Learning Representations (ICLR)*, 2015.
- [2] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *International Conference on Learning Representations (ICLR)*, 2017.
- [3] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Back-propagation without storing activations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.
- [5] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning (ICML)*, 2019.
- [6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.
- [7] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1×1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.
- [8] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.
- [9] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning (ICML)*, 2015.
- [10] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.
- [11] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.