Date: 2 Mai, 23h

Instructions

- Pour toutes les questions qui ne sont pas notées uniquement sur la réponse, montrez votre travail ! Tout problème dont le travail n'est pas montré ne sera pas noté, quelle que soit l'exactitude de la réponse finale.
- Utilisez de préférence un document de rendu comme LaTeX. Si vous écrivez vos réponses à la main, sachez que vous risquez de perdre des points si votre écriture est illisible, sans possibilité de reclassement, à la discrétion du correcteur.
- Soumettez vos réponses par voie électronique via le GradeScope du cours. Les réponses attribuées de manière incorrecte peuvent se voir attribuer automatiquement la note 0, à la discrétion de l'évaluateur. Pour attribuer les réponses correctement, veuillez ::
  - Veillez à ce que le haut de la première page soit consacré à la question évaluée.
  - N'incluez aucune partie de réponse à d'autres questions dans les pages assignées.
  - Les pages assignées doivent être placées dans l'ordre.
  - Pour les questions comportant plusieurs parties, les réponses doivent être rédigées dans l'ordre des parties de la question. de la question.
- Les questions appelant des réponses écrites doivent être courtes et concises lorsque cela est nécessaire. Les questions inutilement verbeuses seront pénalisées à la discrétion du correcteur.
- Svp, signez la décharge ci dessous.
- Il est de votre responsabilité de suivre les mises à jour de l'affectation après sa publication. Tous les changements seront visibles sur Overleaf et Piazza.
- Toute question doit être adressée aux assistants pour ce travail. (partie pratique): *Vitória Barin Pacela, Philippe Martin.*

Pour ces travaux, le lien github est le suivant:

`https://github.com/philmar1/Teaching_IFT6135---Assignment-3---H25`

Je reconnais avoir lu les instructions ci-dessus et je m'engage à les respecter tout au long de ce travail. Je reconnais également que tout travail soumis sans que le formulaire suivant ait été rempli ne sera pas noté pour cette partie du travail..

Signature: OA

Prénom Nom: OUAKIB Amine

UdeM Student ID: 20300393

1 VAE (68 points)

Les autoencodeurs variationnels (VAEs) sont des modèles génératifs probabilistes permettant de modéliser la distribution des données $p(x)$. Dans cette question, vous devrez entraîner un VAE sur l'ensemble de données Binarised MNIST, en utilisant la perte ELBO négative comme vu en cours. Notez que chaque pixel de cet ensemble de données d'images est binaire : le pixel est soit noir, soit blanc, ce qui signifie que chaque point de données (image) est une collection de valeurs binaires, et que chaque image a une taille de 28×28. Vous devez modéliser la vraisemblance $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, c'est-à-dire le décodeur, comme un produit de distributions de Bernoulli.

Dans cette question, vous devez joindre votre code pour l'évaluation qualitative (questions marquées avec `report`) au rapport.

1. **(`unittest`, 6 pts)** Implémentez la fonction '`log_likelihood_bernoulli`' dans '`q1_vae.py`' pour calculer la log-vraisemblance $\log p(\boldsymbol{x})$ pour un échantillon binaire donné $\boldsymbol{x}$ et une distribution de Bernoulli $p(\boldsymbol{x})$. $p(\boldsymbol{x})$ sera paramétrisé par la moyenne de la distribution $p(\boldsymbol{x} = 1)$, et cette valeur sera fournie en entrée à la fonction.

2. **(`unittest`, 6 pts)** Implémentez la fonction '`log_likelihood_normal`' dans '`q1_vae.py`' pour calculer la log-vraisemblance $\log p(\boldsymbol{x})$ pour un vecteur flottant donné $\boldsymbol{x}$ et une distribution normale isotrope $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$. Notez que $\boldsymbol{\mu}$ et $\log(\boldsymbol{\sigma}^2)$ seront donnés pour les distributions normales.

3. **(`unittest`, 8 pts)** Implémentez la fonction '`log_mean_exp`' dans '`q1_vae.py`' pour calculer l'équation suivante[1] pour chaque $\boldsymbol{y}_i$ dans un ensemble donné $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_i, \ldots \boldsymbol{y}_M\}$;

$$\log \frac{1}{K} \sum_{k=1}^{K} \exp\left(y_i^{(k)} - a_i\right) + a_i,$$

   où $a_i = \max_k y_i^{(k)}$. Notez que $\boldsymbol{y}_i = [y_i^{(1)}, y_i^{(2)}, \ldots, y_i^{(k)}, \ldots, y_i^{(K)}]$.

4. **(`unittest`, 6 pts)** Calculez la solution analytique de la divergence KL $D_{\mathrm{KL}}\left(q(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})\right)$ pour les distributions p et q, où p et q sont des distributions normales multivariées avec covariance diagonale. Puis, implémentez la fonction '`kl_gaussian_gaussian_analytic`' dans '`q1_vae.py`'.

5. **(`unittest`, 6 pts)** Implémentez la fonction '`kl_gaussian_gaussian_mc`' dans '`q1_vae.py`' pour calculer la divergence KL $D_{\mathrm{KL}}\left(q(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})\right)$ en utilisant une estimation de Monte Carlo pour les distributions p et q.

6. **(`report`, 8 pts)** Entraînez un VAE en utilisant l'architecture de réseau et les hyperparamètres fournis dans '`q1_train_vae.py`'.

[1] Ceci est une variante de la technique log-sum-exp pour gérer les problèmes de sous-débordement numérique : la génération d'un nombre trop petit pour être représenté par l'appareil destiné à le stocker. Par exemple, les probabilités des pixels d'une image peuvent devenir très petites. Pour plus de détails sur le sous-débordement numérique dans le calcul de la log-probabilité, voir `http://blog.smola.org/post/987977550/log-probabilities-semirings-and-floating-point`.

Remplissez la fonction '`loss_function.py`' dans '`q1_train_vae.py`' en réutilisant votre code de '`q1_vae.py`'.

Optimisez avec Adam, avec un taux d'apprentissage de 10^{-3}, et entraînez pendant 20 époques.

Ensuite, évaluez le modèle sur le jeu de validation.

Le modèle doit atteindre une perte moyenne ≤ 104 sur le jeu de validation.

Rapportez la perte finale de validation de votre modèle et tracez les pertes d'entraînement et de validation.

Réponse :

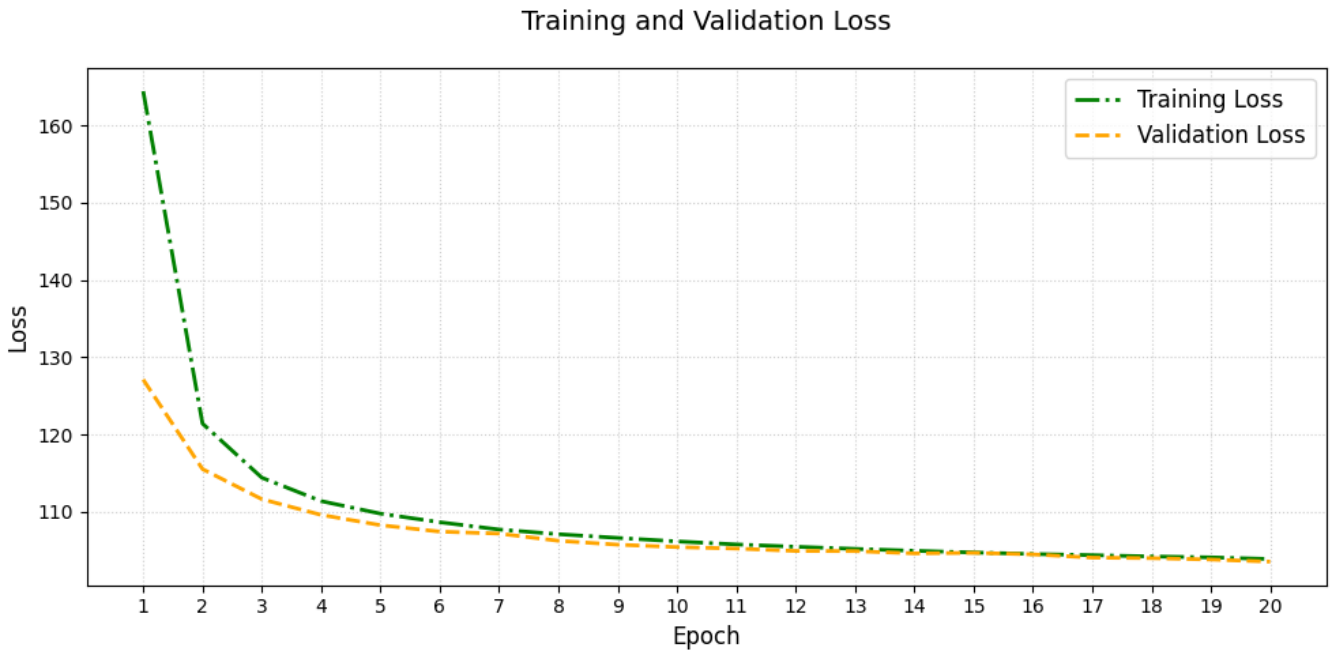

Figure 1: les pertes d'entraînement et de validation

```
====> Epoch: 20 Average loss: 103.8828
====> Validation Loss: 103.5406
Training completed. Model and loss saved in ./results_q1_vae/
```

Figure 2: une perte moyenne ≤ 104 sur le jeu de validation.

7. **(`report`, 6 pts) Fournissez des exemples visuels générés par le modèle.** Commentez la qualité des exemples (ex. flou, diversité, réalisme).
Réponse :

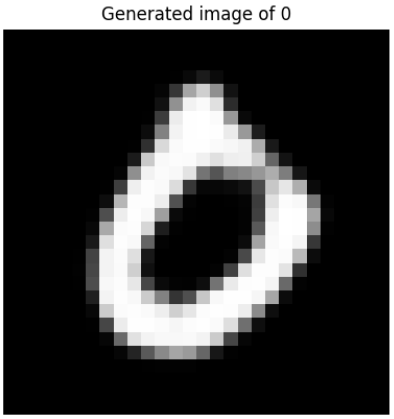

Figure 3: generated image 0

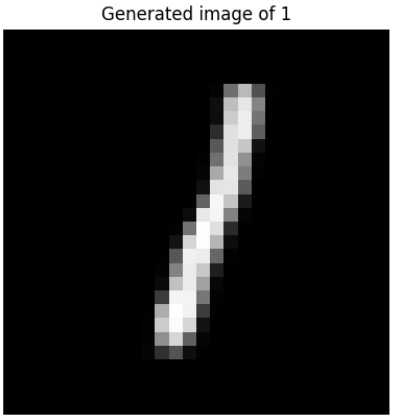

Figure 4: generated image 1

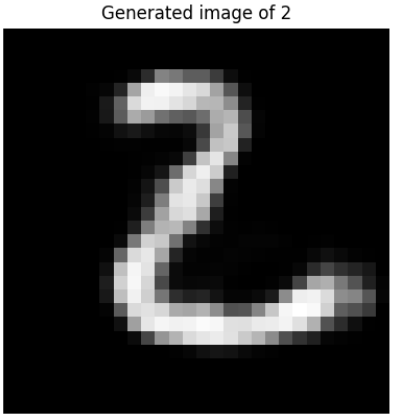

Figure 5: generated image 2

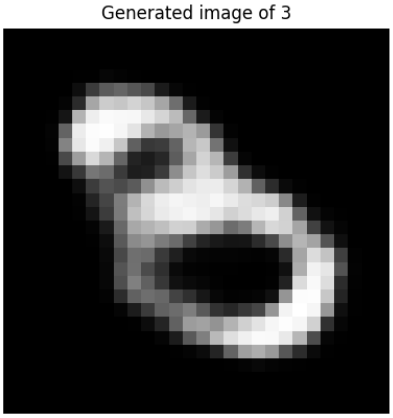

Figure 6: generated image 3

Qualité visuelle de l'exemple généré

Netteté : L'image est légèrement floue, ce qui est typique des générations par des VAE. Contrairement aux autoencodeurs classiques ou aux GANs, les VAE ont une tendance à produire des images un peu plus douces, car ils introduisent une distribution probabiliste dans l'espace latent.

Reconnaissance du chiffre : Le chiffre **6** est globalement reconnaissable. On observe bien la courbure caractéristique du haut et la boucle du bas. Cela montre que le modèle a bien capturé les caractéristiques principales des chiffres manuscrits.

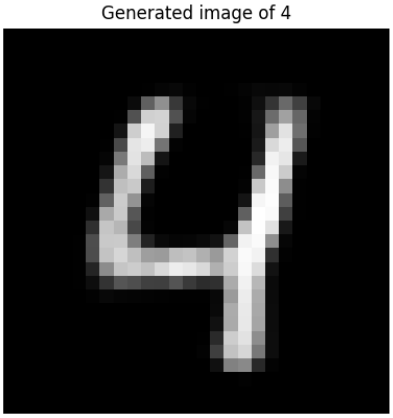

Figure 7: generated image 4

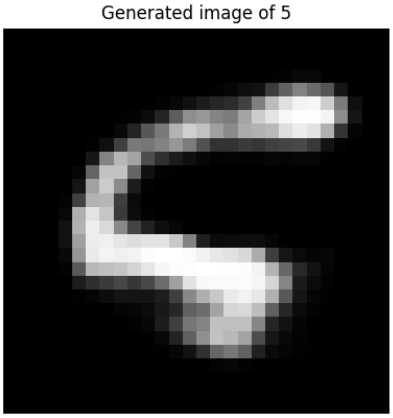

Figure 8: generated image 5

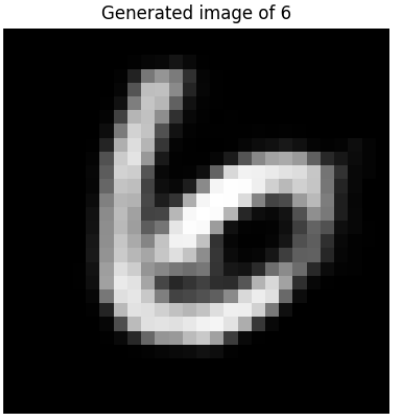

Figure 9: generated image 6

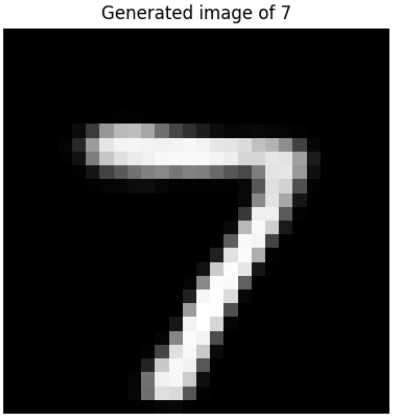

Figure 10: generated image 7

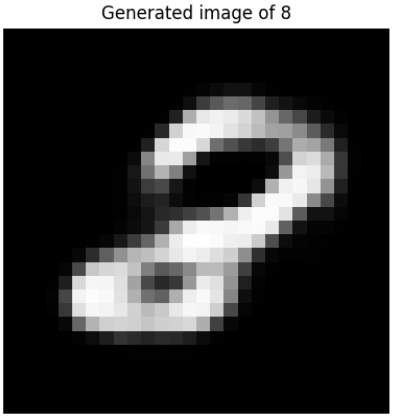

Figure 11: generated image 8

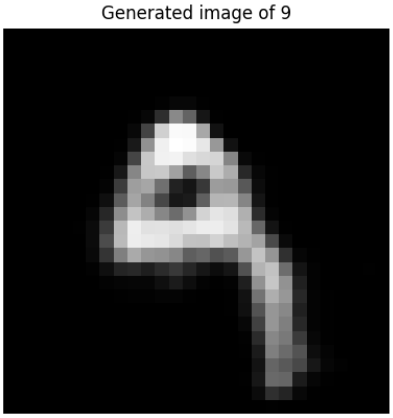

Figure 12: generated image 9

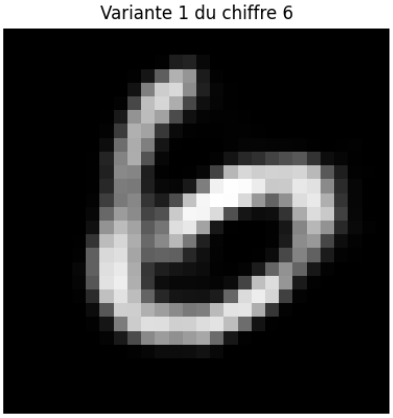

Figure 13: generated image 6 variation 1

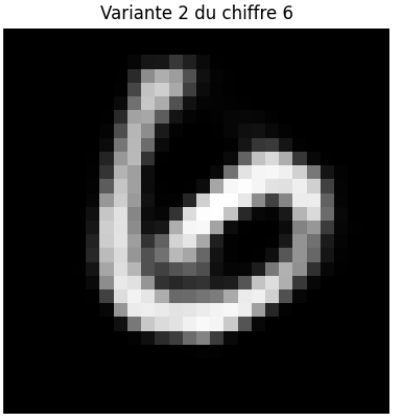

Figure 14: generated image 6 variation 2

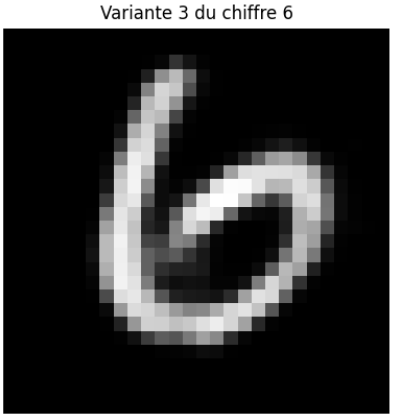

Figure 15: generated image 6 variation 3

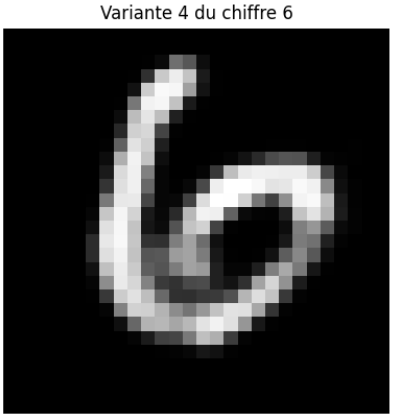

Figure 16: generated image 6 variation 4

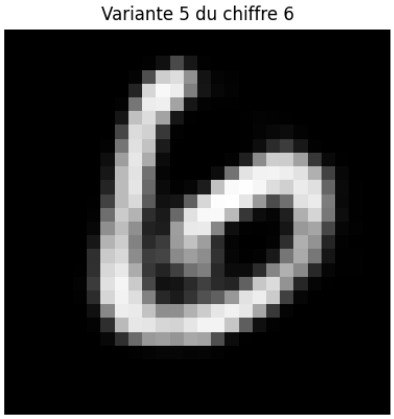

Figure 17: generated image 6 variation 5

Diversité : Comme un seul chiffre est affiché ici, on ne peut pas encore juger la diversité. Il faudrait comparer plusieurs exemples pour voir si le modèle génère différentes variantes du même chiffre (par exemple, un *6* plus serré ou plus large).

Réaliste par rapport à MNIST :

- Chiffre centré sur fond noir
- Taille et tracé cohérents
- Style manuscrit respecté

Cela indique que le modèle a bien appris la distribution des chiffres dans MNIST.

En résumé :

- Les images générées sont plutôt réalistes et reconnaissables.
- Un peu floues, ce qui est normal pour un VAE.
- Bonne fidélité au style MNIST.
- Il reste à tester d'autres chiffres (*0* à *9*) pour commenter la diversité complète.

Les images générées sont globalement réalistes et fidèles au style des chiffres manuscrits du dataset **MNIST**. Chaque chiffre (de **0** à **9**) est bien centré, tracé sur fond noir, et respecte le style manuscrit d'origine.

Malgré une légère perte de netteté — typique des modèles variationnels autoencodeurs (VAE) — les chiffres sont facilement reconnaissables. Ce flou est attendu et provient de l'échantillonnage probabiliste dans l'espace latent.

Le modèle semble donc avoir correctement appris la distribution sous-jacente des données. Enfin, on observe une bonne diversité : pour un même chiffre, plusieurs variantes sont générées, reflétant la variabilité présente dans les données réelles.

8. **(`report`, 12 pts)** Nous souhaitons vérifier si le modèle a appris une représentation démêlée dans l'espace latent. La VAE fournie produit 20 facteurs latents. Représentez graphiquement les images des parcours latents avec 5 échantillons par facteur latent. Que pouvez-vous dire sur la démêlée des facteurs latents ?

   Pour effectuer les parcours, échantillonnez z de votre distribution (qui, dans ce cas, est une gaussienne standard). Apportez de petites perturbations à votre échantillon z pour chaque dimension (par exemple, pour une dimension i, $z'_i = z_i + \epsilon$). ϵ doit être suffisamment grand pour observer une différence visuelle. Pour chaque dimension, observez si les changements entraînent des variations visuelles (c'est-à-dire des variations de $g(z)$ – où g est le décodeur).
   Réponse :

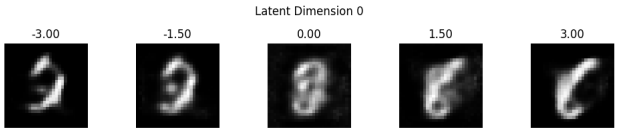

Figure 18: latent dim 0

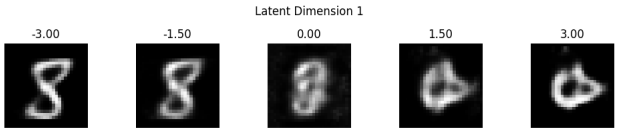

Figure 19: latent dim 1

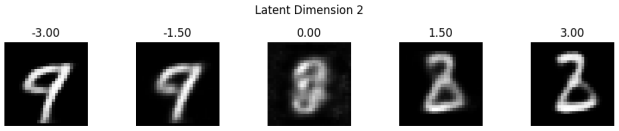

Figure 20: latent dim 2

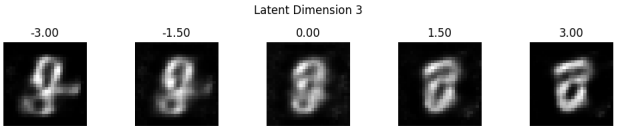

Figure 21: latent dim 3

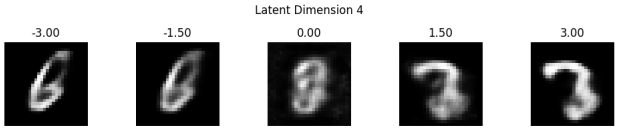

Figure 22: latent dim 4

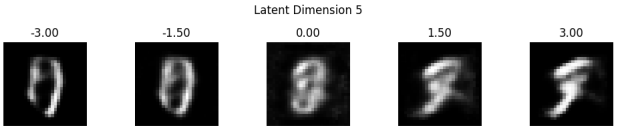

Figure 23: latent dim 5

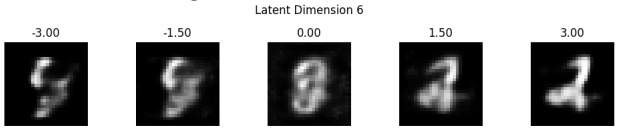

Figure 24: latent dim 6

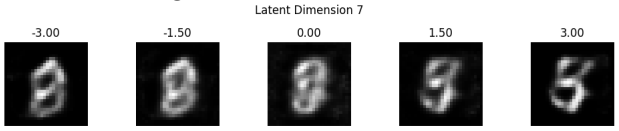

Figure 25: latent dim 7

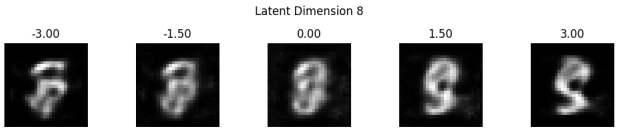

Figure 26: latent dim 8

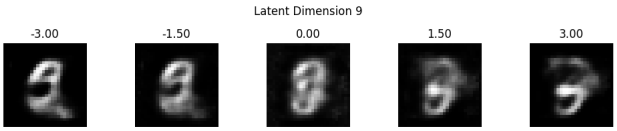

Figure 27: latent dim 9

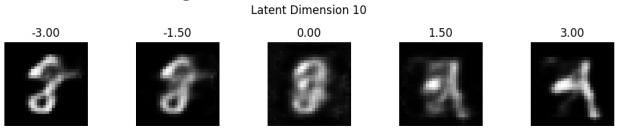

Figure 28: latent dim 10

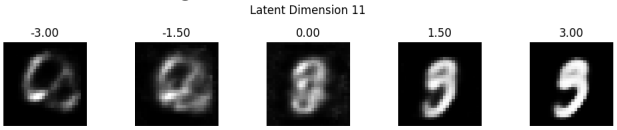

Figure 29: latent dim 11

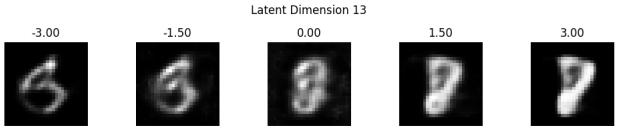

Figure 30: latent dim 13

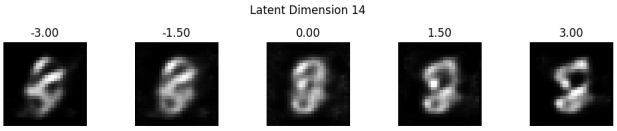

Figure 31: latent dim 14

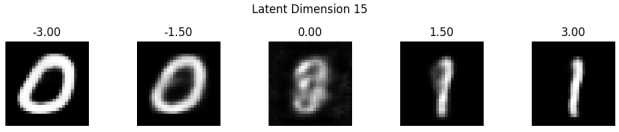

Figure 32: latent dim 15

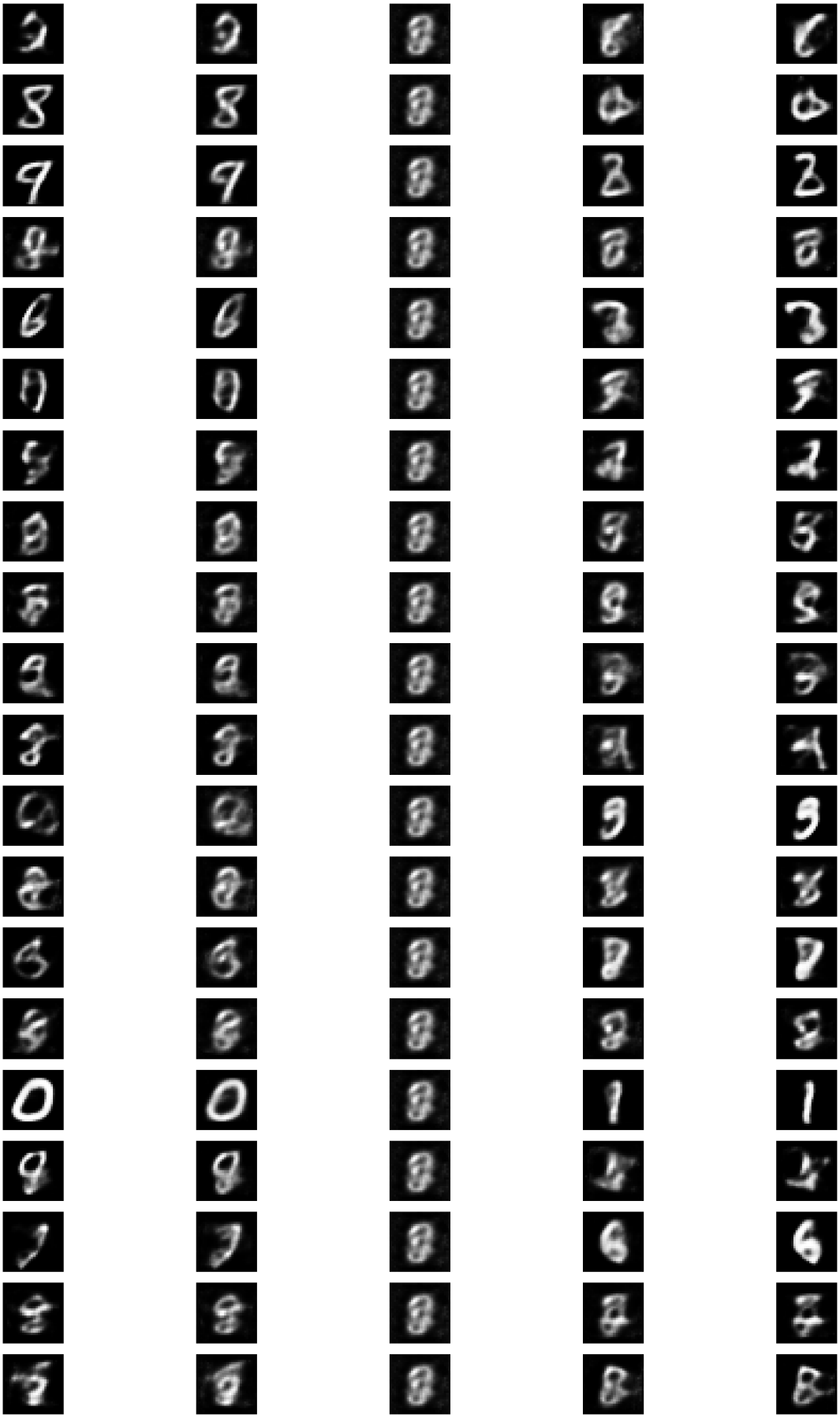

Figure 33: latent dimension pour all

9. **(`report`, 10 pts) Comparer l'interpolation dans l'espace de données et dans l'espace latent.** Choisir deux points aléatoires z_0 et z_1 dans l'espace latent échantillonné à partir de la loi a priori.

   (a) Pour $\alpha = 0, 0, 1, 0, 2 \ldots 1$, calculer $z'_\alpha = \alpha z_0 + (1 - \alpha) z_1$ et représenter graphiquement les échantillons résultants $x'_\alpha = g(z'_\alpha)$.

   (b) En utilisant les échantillons de données $x_0 = g(z_0)$ et $x_1 = g(z_1)$ et pour $\alpha = 0, 0, 1, 0, 2 \ldots 1$, tracez les échantillons $\hat{x}_\alpha = \alpha x_0 + (1 - \alpha) x_1$.

   Expliquez la différence entre les deux schémas d'interpolation entre images.
   Réponse :

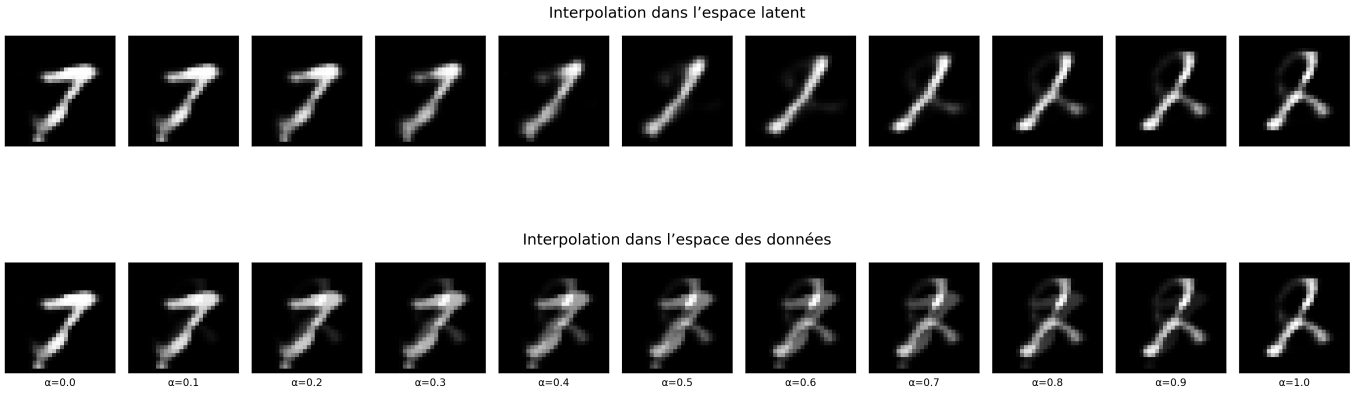

Figure 34: comparaison entre les deux interpolations

Interpolation dans l'espace latent

L'interpolation dans l'espace latent se fait entre deux vecteurs latents z_0 et z_1. Chaque image générée à partir de $z'_\alpha = \alpha z_0 + (1 - \alpha) z_1$ est un mélange des caractéristiques latentes des deux images originales.

Comme l'espace latent est continu, l'interpolation entre ces points latents produit des transitions fluides entre les deux images. En d'autres termes, l'interpolation dans l'espace latent permet de varier progressivement les caractéristiques des images sans ruptures brusques.

Interpolation dans l'espace des données

L'interpolation dans l'espace des données est effectuée directement entre les images x_0 et x_1, générées à partir des vecteurs latents z_0 et z_1. Ici, on effectue une simple combinaison linéaire des pixels des deux images, ce qui peut entraîner des artefacts visuels si les images ne sont pas suffisamment similaires.

Cette interpolation peut être moins fluide que dans l'espace latent, car les images résultantes sont une simple combinaison de pixels et ne respectent pas la structure continue et lisse de l'espace latent.

Différence entre les deux types d'interpolation

- **Espace latent :** L'interpolation dans l'espace latent produit des transitions plus douces et naturelles entre les images, car les vecteurs latents suivent une distribution probabiliste et capturent mieux les variations continues des caractéristiques des images.
- **Espace des données :** L'interpolation dans l'espace des données peut être plus brute, car elle ne tient pas compte des relations complexes entre les pixels, ce qui peut entraîner des artefacts visuels lors de l'interpolation. De plus, cela peut ne pas être aussi fluide, surtout si les images sont très différentes l'une de l'autre.

Conclusion

L'interpolation dans l'espace latent permet de créer des transitions plus cohérentes et fluides, tandis que l'interpolation dans l'espace des données peut être moins réaliste en raison de la combinaison directe des images. L'espace latent représente une structure plus profonde et continue des données, rendant l'interpolation plus précise et visuellement agréable.

Interpolation dans l'espace des données (Data Space)

Dans ce cas, l'interpolation est effectuée **pixel par pixel** entre deux images générées x_0 et x_1.

Le résultat est un *fondu linéaire* entre les deux chiffres, similaire à un effet de superposition.

Problèmes observés :

- Il n'y a **aucune garantie** que les images intermédiaires soient réalistes.
- Par exemple, pour $\alpha = 0.33$ ou $\alpha = 0.56$, l'image interpolée ressemble à un *mélange incohérent* de deux chiffres.
- Les bords sont souvent flous, et des *parties de chiffres en double* peuvent apparaître.

Conclusion : les images interpolées ressemblent à des *moyennes visuelles*, sans signification sémantique claire.

Interpolation dans l'espace latent (Latent Space)

Dans ce cas, on interpole entre deux vecteurs latents z_0 et z_1 issus de la distribution a priori (souvent $\mathcal{N}(0, I)$) :

$$z'_\alpha = \alpha z_0 + (1 - \alpha) z_1$$

On applique ensuite le générateur g pour obtenir l'image correspondante :

$$x'_\alpha = g(z'_\alpha)$$

Observations :

- La transition entre les deux chiffres est **fluide et continue**.
- Chaque image intermédiaire est **cohérente**, bien formée et visuellement plausible.
- Cela fonctionne car le modèle a appris une *structure sémantique* dans l'espace latent.

2 Diffusion models

Implémentation de Modèles Probabilistes de Diffusion pour la Réduction du Bruit (DDPM) et de Modèles à Guidage sans Classificateur (CFG) pour générer des images de style MNIST (100 points) .

1. Génération inconditionnelle (45 pts.)
Dans ce problème, vous allez implémenter une classe DDPM (menter une classe DDPM (\url{https://arxiv.org/ sur le jeu de données MNIST en utilisant PyTorch selon les instructions fournies. L'objectif est de minimiser la fonction de perte et d'entraîner le modèle à générer des images MNIST.

Les classes Train et UNet sont déjà implémentées pour vous. Vous devez implémenter la classe DDPM (voir les détails ci-dessous). Les images générées par le modèle seront automatiquement affichées selon l'implémentation de la classe Trainer. Assurez-vous que les images générées sont affichées dans la sortie, elles seront évaluées.

Si vous avez besoin de créer votre propre environnement virtuel, veuillez créer un environnement python==3.11 et installer les dépendances depuis requirements.txt

Nous vous recommandons de suivre le notebook DDPM.ipynb sur votre propre ordinateur. Une fois que vous avez complété les méthodes pour les classes DenoiseDiffusion et Trainer, vous pouvez les copier-coller dans ddpm.py et trainer.py, ces fichiers seront utilisés par Gradescope. Ensuite, lorsque vous serez prêt à effectuer un entraînement, vous pourrez importer ce notebook dans Google Colab si vous avez besoin d'un accès gratuit à un GPU.

Gradescope:

Les fichiers que vous devez compléter et soumettre via Gradescope pour l'évaluation automatique sont `q2_trainer_ddpm.py`, `q2_ddpm.py`. Vous devez également soumettre votre rapport (PDF) sur Gradescope. Votre rapport doit contenir les réponses au problème. Vous n'avez pas besoin de soumettre du code pour ces questions ; le rapport suffira.

Grade:

- Implémentez la classe DenoiseDiffusion (15 points).
- Complétez les méthodes *sample* et *generate_ intermediate_ samples* de la class Trainer (10 points).
- Entraînez le modèle pour générer des images MNIST raisonnables en 20 époques (10 points)
- Rédigez un rapport pour expliquer:
  - Les images générées à chaque epoch et donnez des recommandations sur comment améliorer la qualité des images générées (5 points).
    Réponse :

Analyse par époque

Époque 0

* **Observation :** Les images sont très bruitées, sombres et peu informatives. Les structures visibles sont floues ou absentes.
* **Interprétation :** Le modèle n'a pas encore appris à retirer correctement le bruit ou à former des structures cohérentes. Ce comportement est typique du début d'apprentissage dans un DDPM.

Époque 2

* **Observation :** Les images commencent à montrer des formes lumineuses plus définies. La structure commence à émerger, mais de nombreux artefacts sont encore présents.
* **Interprétation :** Le modèle commence à apprendre les motifs de la distribution réelle, mais la génération n'est pas encore stable.

Époque 4

* **Observation :** Les formes sont beaucoup plus nettes et complexes. Le fond est bien noir et les structures blanches présentent des contours définis.
* **Interprétation :** Le modèle a clairement progressé dans la génération de données réalistes. On observe des objets bien formés, comparables à des vraies instances de la distribution cible.

subsection*Époque 14

* **Observation :** Certaines images sont floues ou ambigües (`(0,0)`, `(0,1)`).
* Des chiffres sont mal formés ou ambivalents (`(2,0)` pourrait être un 6 ou un 0).
* Plusieurs chiffres "4" apparaissent très souvent, ce qui pourrait indiquer un déséquilibre ou une mauvaise diversité.

Note approximative : 2.5 / 5

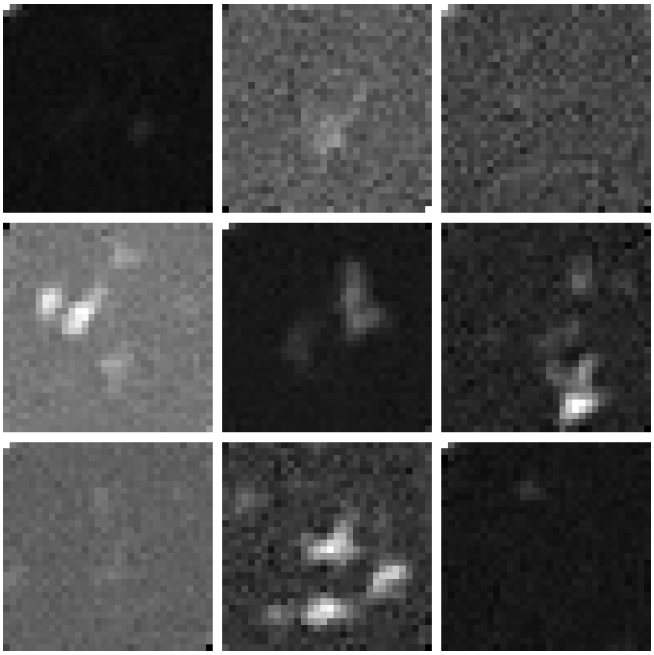

Figure 35: DDPM epoch 0

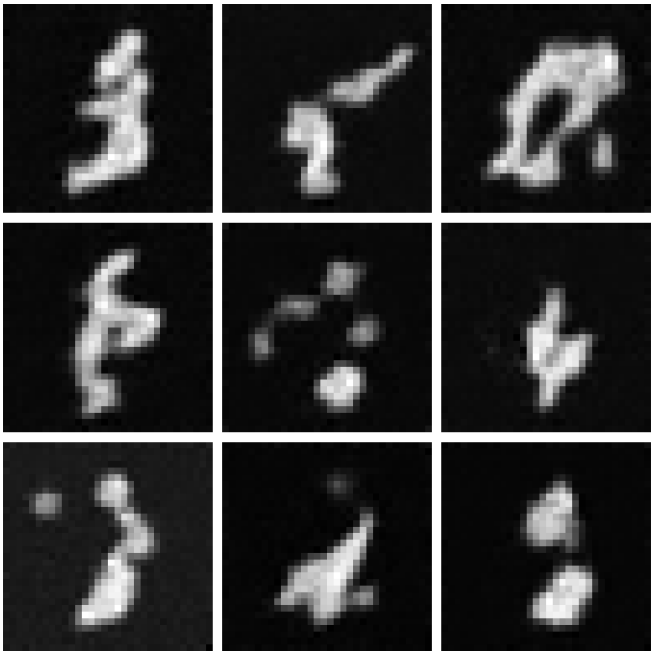

Figure 36: DDPM epoch 2

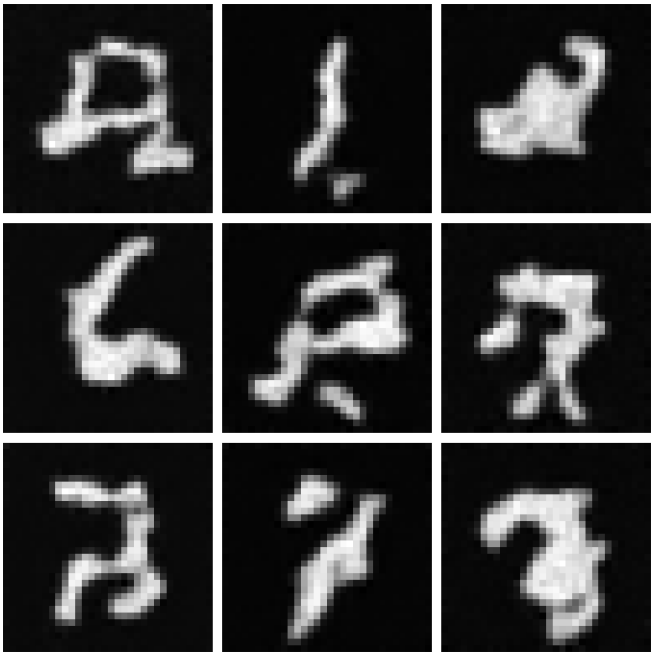

Figure 37: DDPM epoch 4

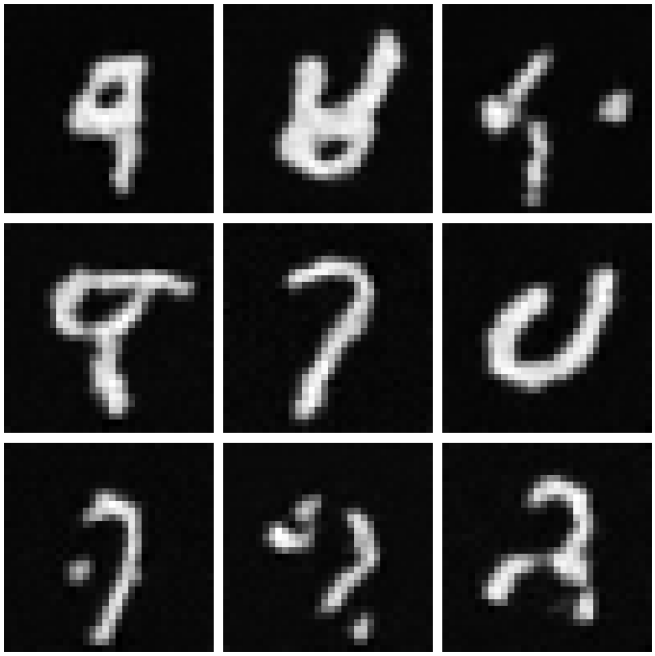

Figure 38: DDPM epoch 6

Époque 16

* **Observation :** Qualité globale en légère amélioration.
* Quelques chiffres sont bien formés (`(0,0)` est un 5 clair, `(1,1)` est un 1 net).
* Plusieurs chiffres sont encore distordus ou ambigus (`(0,1)`, `(1,0)`, `(2,1)`).

Note approximative : 3 / 5

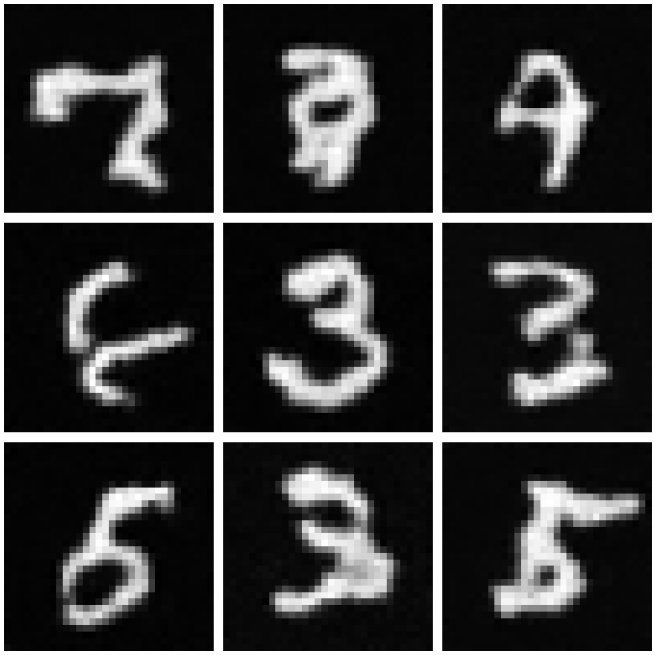

Figure 39: DDPM epoch 8

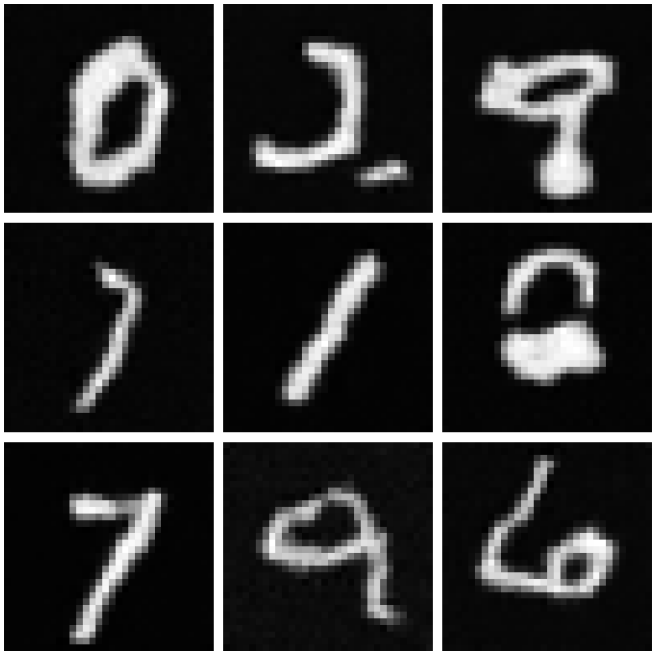

Figure 40: DDPM epoch 10

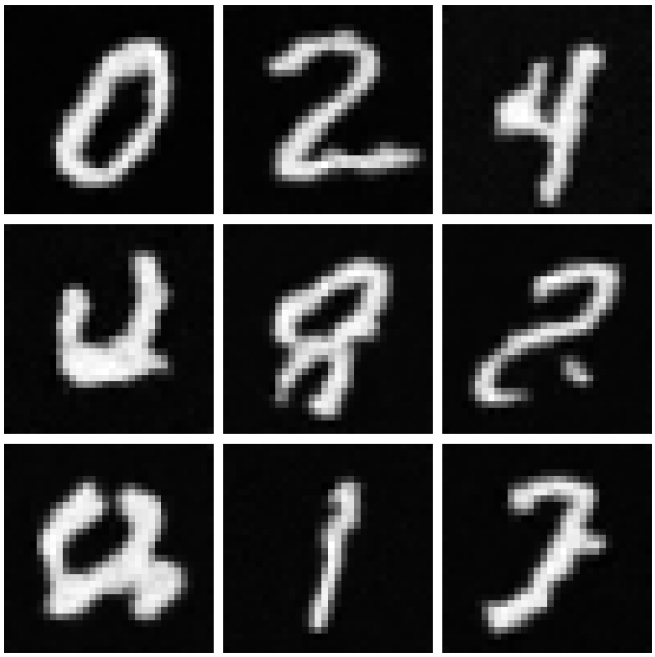

Figure 41: DDPM epoch 12

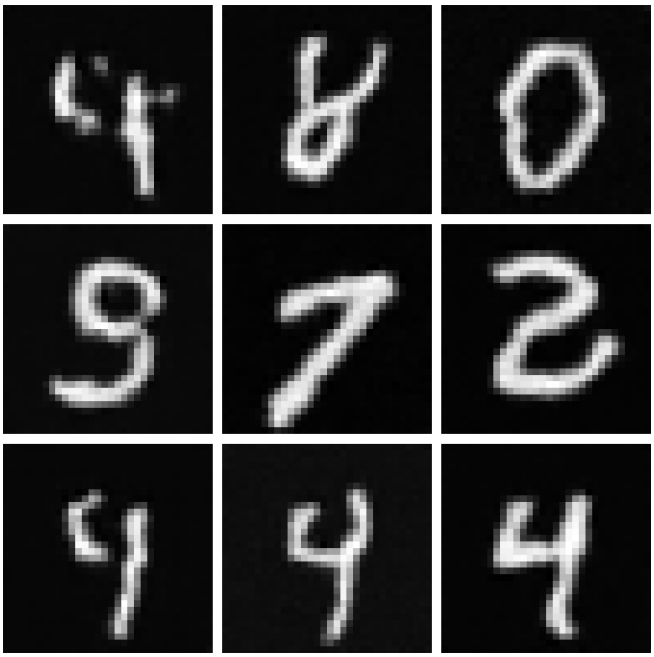

Figure 42: DDPM epoch 14

Époque 18

* **Observation :** La qualité visuelle et la netteté des chiffres augmentent.
* Davantage de chiffres sont reconnaissables (`(1,0)` est un 3, `(1,1)` est un 9).
* Encore quelques chiffres flous ou difficiles à lire (`(0,1)`, `(0,2)`, `(2,1)`).

Note approximative : 3.5 / 5

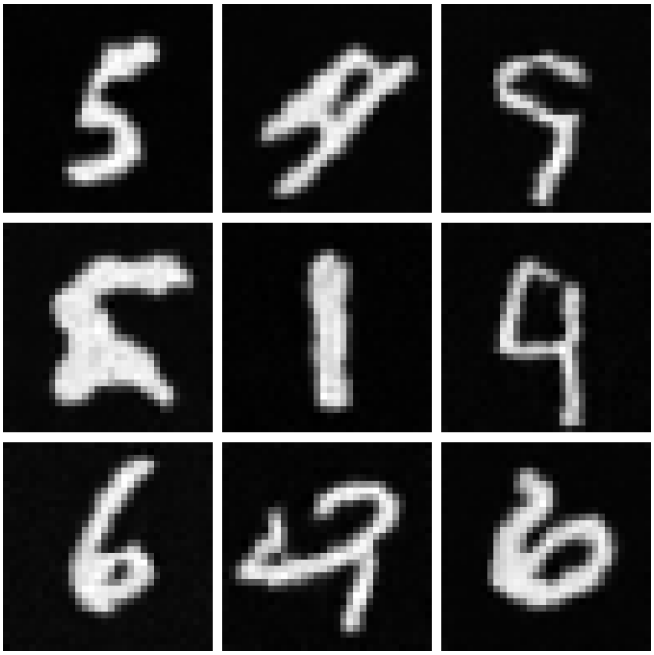

Figure 43: DDPM epoch 16

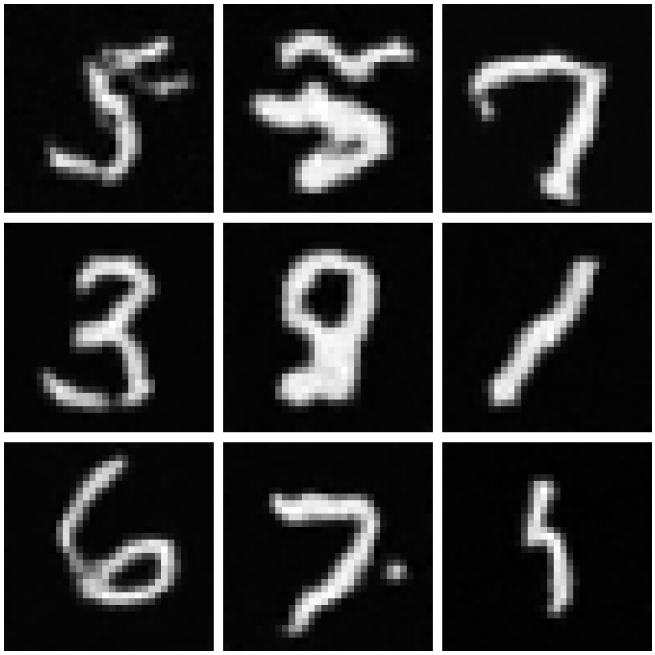

Figure 44: DDPM epoch 18

Recommandations pour améliorer la qualité

1. **Augmenter le nombre d'époques :** Le modèle montre une amélioration significative à chaque étape ; continuer l'entraînement pourrait stabiliser et affiner davantage les détails.
2. **Utiliser un scheduler de bruit plus adapté :** Par exemple, le *cosine noise schedule* (au lieu d'un planning linéaire) permet souvent de mieux préserver les détails visuels.
3. **Augmenter la capacité du réseau :** Un U-Net plus profond ou l'ajout de blocs résiduels peut aider à mieux modéliser les détails fins.
4. **Améliorer le dataset ou le prétraitement :** Si les données d'entraînement sont bruitées ou déséquilibrées, cela affecte la qualité des générations. Une normalisation plus soignée ou un meilleur contraste peut améliorer les résultats.
5. **Ajouter du conditionnement ou de la guidance :** L'implémentation de la *classifier-free guidance*, par exemple, permet d'orienter la génération vers des échantillons plus réalistes.

– Les images générées à différents pas lors de l'échantillonnage inverse par le modèle entrainé, en utilisant la fonction *generate_intermediate_samples*() (5 points).
Réponse :

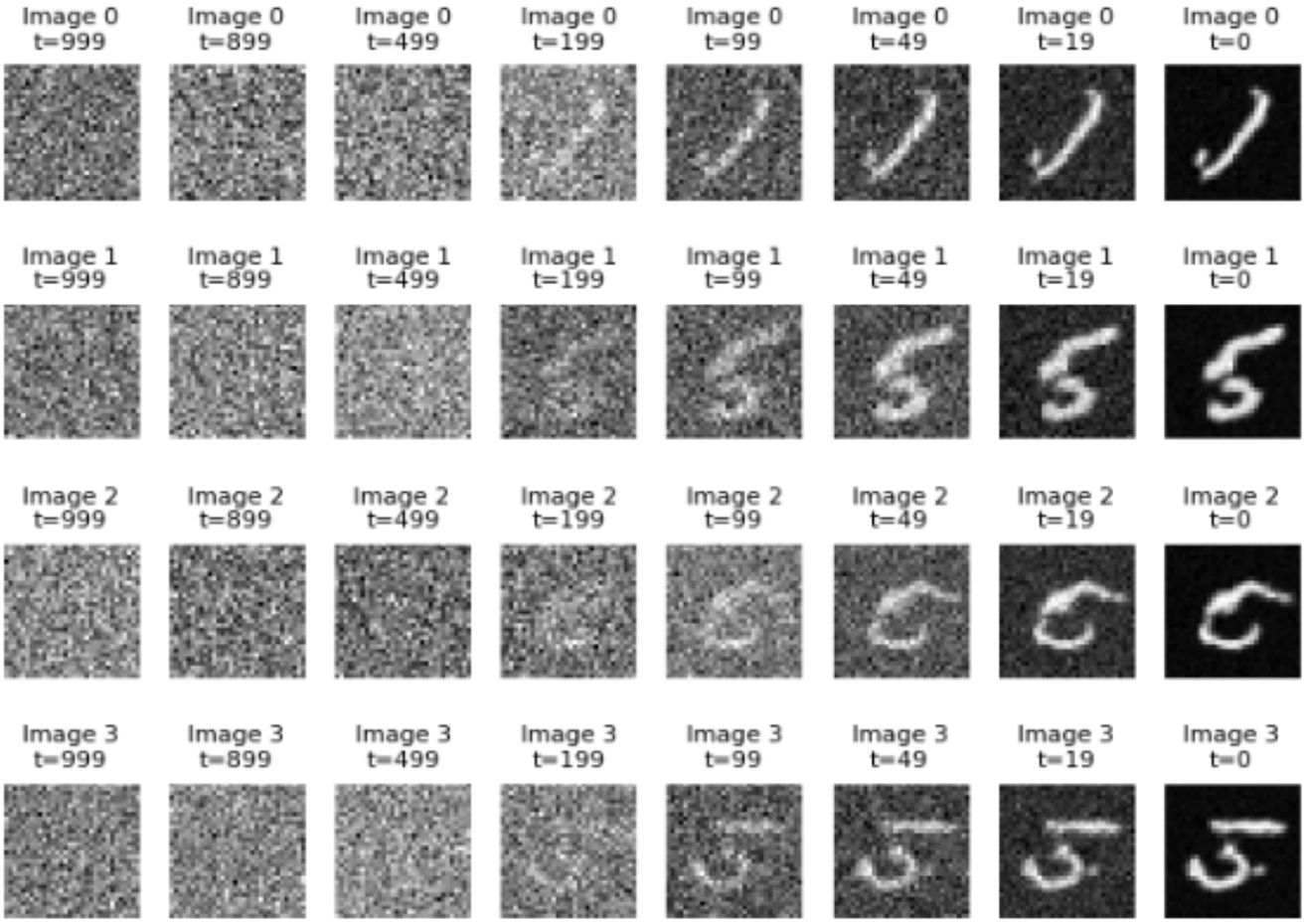

Figure 45: ddpm generative

Veuillez noter que la fonction permettant de générer les images est déjà fournie.

2. Génération conditionnelle (55 pts.)

Dans la partie suivante du problème, vous modifierez le code pour permettre à l'utilisateur de spécifier le modèle que la classe doit générer. À cette fin, le modèle doit être entraîné en tenant compte des étiquettes des données d'entraînement. Nous utiliserons l'approche Classifier Free Guidance (`https://arxiv.org/pdf/2207.12598`). La classe UNet a été mise à jour pour prendre en compte les étiquettes en entrée. Dans ce tutoriel, l'algorithme 1 échantillonnera le temps t uniformément, tout comme dans l'implémentation précédente du DDPM.

Une fois de plus, nous vous recommandons de suivre le notebook ClassifierFreeGuidance.ipynb sur votre propre ordinateur. Une fois que vous avez rempli les méthodes pour les classes CFGDiffusion

et Trainer, vous pouvez les copier-coller dans cfg_diffusion.py et trainer.py, qui seront utilisées par Gradescope. Ensuite, lorsque vous êtes prêt à faire une formation, vous pouvez importer ce cahier dans Google Colab si vous avez besoin d'un accès gratuit à un GPU.

Gradescope: Les fichiers que vous devez compléter et soumettre via Gradescope pour la notation automatique sont `q3_trainer_cfg.py`, `q3_cfg_diffusion.py`. Vous devez également soumettre votre rapport (PDF) sur Gradescope. Votre rapport doit contenir les réponses au problème. Il n'est pas nécessaire de soumettre le code pour ces questions ; le rapport suffira.

Grade:

- Expliquez pourquoi le modèle est appelé Classifier Free et pourquoi Guidance (10 points).
  Réponse :

Pourquoi le modèle est-il appelé *Classifier-Free Guidance* ?

Le terme **Classifier-Free Guidance** fait référence à une méthode de génération conditionnelle qui ne nécessite pas l'utilisation d'un classificateur externe.

Classifier-Free

Traditionnellement, certains modèles de diffusion conditionnels utilisent un *classificateur pré-entraîné* pour guider la génération vers une classe cible.

Dans **Classifier-Free Guidance**, il n'y a **pas de classificateur externe**. À la place, le modèle de bruit (souvent un U-Net) est entraîné à la fois :

- **Conditionnellement** : en prenant en compte les étiquettes de classe.
- **Non-conditionnellement** : en ignorant les étiquettes avec une certaine probabilité.

Ce double apprentissage permet au modèle de générer des images **avec ou sans condition**, d'où le nom *classifier-free*.

Guidance

Lors de la génération, on combine les prédictions du modèle conditionné et non conditionné pour orienter la génération :

$$\epsilon_{\text{guided}} = (1 + w) \cdot \epsilon_{\text{cond}} - w \cdot \epsilon_{\text{uncond}}$$

où w est un hyperparamètre appelé *guidance scale*.

Cela permet de renforcer la cohérence entre l'image générée et la classe cible, **sans utiliser de classificateur explicite**.

- D'après le document, quelle serait une alternative au modèle sans classificateur ? Expliquez comment la fonction de coût changerait dans cette alternative par rapport à celle DDPM originale. (10 points)
  Réponse :

Alternative au modèle sans classificateur : Classifier-Free Guidance

Dans l'article Elucidating the Design Space of Diffusion-Based Generative Models (Karras et al., 2022), une alternative aux modèles guidés par un classificateur est proposée : il s'agit de la méthode appelée **Classifier-Free Guidance (CFG)**.

Principe du CFG

Au lieu d'utiliser un classificateur externe pour guider la génération conditionnelle, le modèle de bruit ϵ_θ est entraîné à la fois avec et sans condition (par exemple une étiquette de classe) :

- Pendant l'entraînement, on fournit parfois la condition c (ex. : le label) et parfois on l'omet (en pratique, on met $c = \emptyset$ avec une probabilité de 10%).
- À l'inférence, la prédiction du bruit est interpolée entre la version conditionnée et non-conditionnée :

$$\epsilon_{\text{CFG}} = (1 + w) \cdot \epsilon_\theta(z_t, c) - w \cdot \epsilon_\theta(z_t, \emptyset)$$

où w est un facteur d'intensité du guidage (souvent noté `cfg_scale`).

Changement dans la fonction de coût

Fonction de coût du DDPM original (non conditionnel) :

$$\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon_\theta(z_t, t) - \epsilon\|^2\right]$$

Fonction de coût avec Classifier-Free Guidance :

$$\mathcal{L}_{\text{CFG}} = \mathbb{E}_{t,x_0,\epsilon,c}\left[\|\epsilon_\theta(z_t, c) - \epsilon\|^2\right] + \mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon_\theta(z_t, \emptyset) - \epsilon\|^2\right]$$

Autrement dit, on garde la même structure de perte (erreur quadratique entre le bruit réel et le bruit prédit), mais on introduit des cas où la condition est absente pour permettre au modèle d'apprendre une version non conditionnelle.

Résumé comparatif

Approche	Guidage	Coût suppl.	Fonction de coût
DDPM original	Non	Non	$\|\epsilon_\theta - \epsilon\|^2$
Avec classificateur (DDPM+CLF)	Oui (externe)	Oui (classif. + gradient)	MSE + guidance par gradient
Classifier-Free Guidance	Oui (interne)	Non (pas de classif.)	Même MSE, avec cas $c = \emptyset$

Alternative au modèle sans classificateur et changement de la fonction de coût

Une alternative à l'approche *Classifier-Free Guidance* est celle des modèles guidés par **classificateur externe**, comme proposé dans le travail de Dhariwal et Nichol (2021). Dans cette approche, un classificateur $p(y \mid x_t)$ est entraîné pour prédire la probabilité qu'une image bruitée x_t appartienne à une classe y, à chaque étape t de la diffusion.

Lors de la génération, ce classificateur est utilisé pour guider la trajectoire de débruitage. L'estimation du bruit est alors corrigée selon le gradient de la log-probabilité de la classe cible :

$$\tilde{\epsilon}(x_t, t) = \epsilon_\theta(x_t, t) - s \cdot \nabla_{x_t} \log p(y \mid x_t),$$

où s est un coefficient d'échelle (guidance scale). Ce terme pousse le modèle à générer des échantillons qui correspondent mieux à la classe souhaitée. Toutefois, cette méthode dépend fortement d'un classificateur externe bien entraîné, ce qui ajoute de la complexité et peut introduire des erreurs si le classificateur est peu fiable sur les images bruitées.

Dans le modèle DDPM original, la fonction de coût est définie comme une erreur quadratique moyenne entre le bruit réel ϵ et le bruit prédit ϵ_θ :

$$\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon_\theta(z_t, t) - \epsilon\|^2\right]$$

Avec l'approche **Classifier-Free Guidance (CFG)**, le modèle est entraîné à la fois avec et sans condition c (par exemple, l'étiquette de classe). Ainsi, la fonction de coût devient :

$$\mathcal{L}_{\text{CFG}} = \mathbb{E}_{t,x_0,\epsilon,c}\left[\|\epsilon_\theta(z_t, t, c) - \epsilon\|^2\right] + \mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon_\theta(z_t, t, \emptyset) - \epsilon\|^2\right]$$

Cette nouvelle fonction de coût incorpore deux cas :

- Lorsque la condition c est présente (conditionnel).
- Lorsque la condition est absente ($c = \emptyset$), pour apprendre une prédiction non conditionnelle.

L'objectif est de permettre à un seul modèle ϵ_θ de produire à la fois des sorties conditionnelles et non conditionnelles, ce qui rend possible l'interpolation contrôlée entre les deux pendant l'échantillonnage :

$$\epsilon_{\text{CFG}} = (1 + w) \cdot \epsilon_\theta(z_t, c) - w \cdot \epsilon_\theta(z_t, \emptyset)$$

où w est un coefficient qui règle la force du guidage.

- Implémenter la classe CFGDiffusion (20 points)
- Compléter la méthode Trainer.sample() (10 points)
- Rédigez un rapport décrivant les images échantillonnées générées par chaque période (5 points).
  Réponse :
  Entraînement avec 16 époques :

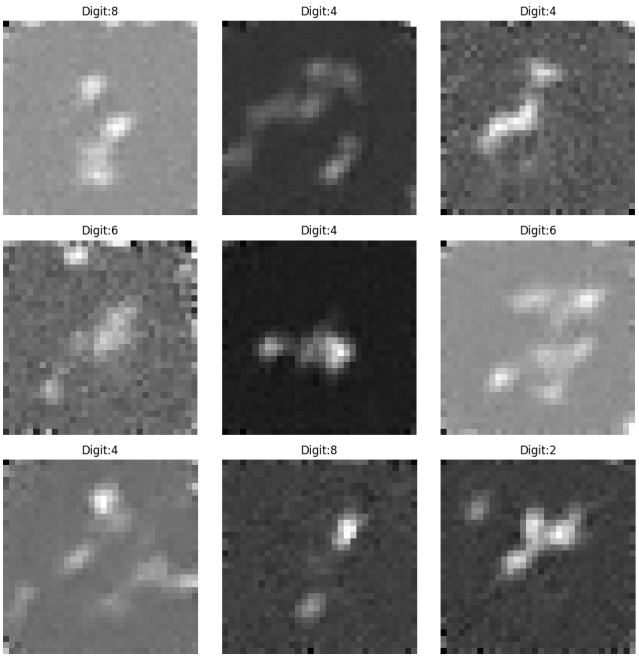

Figure 46: DDPM epoch 0

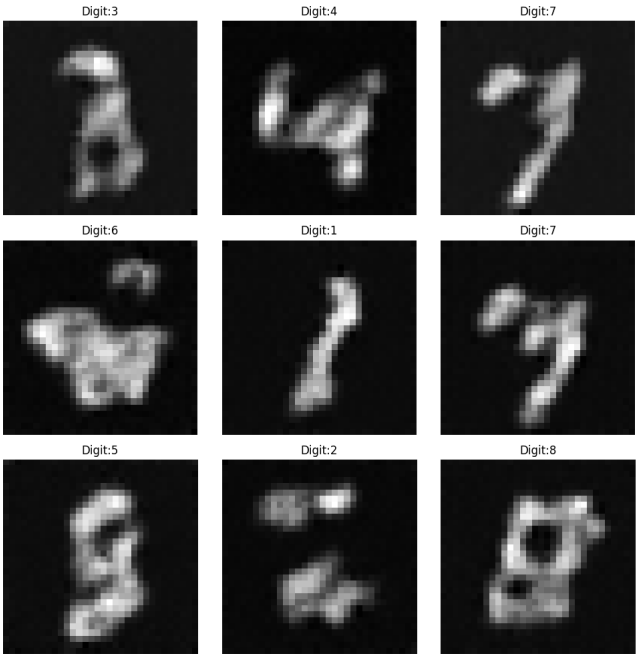

Figure 47: DDPM epoch 2

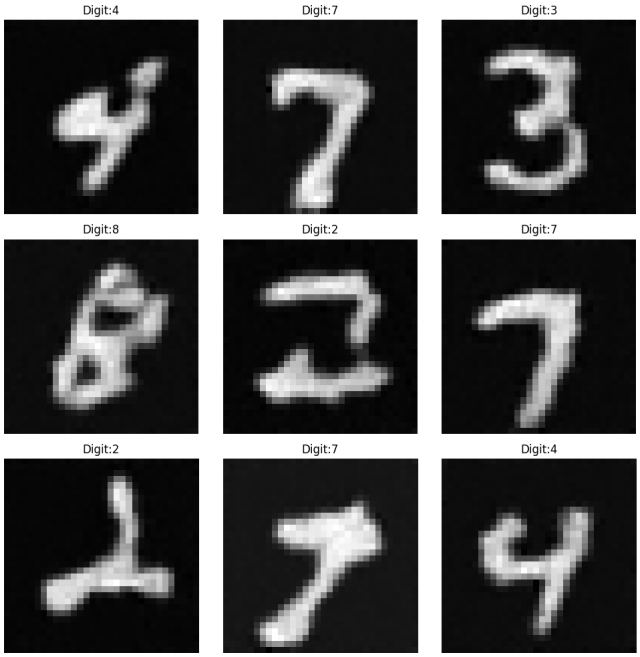

Figure 48: DDPM epoch 4

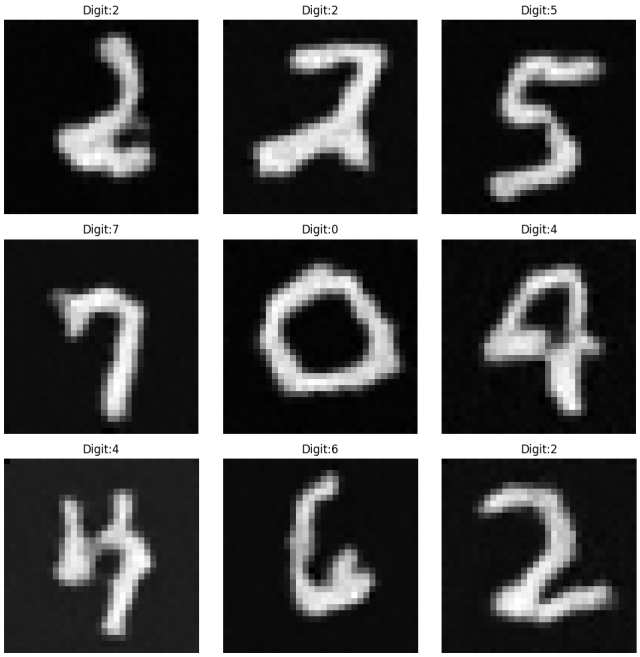

Figure 49: DDPM epoch 6

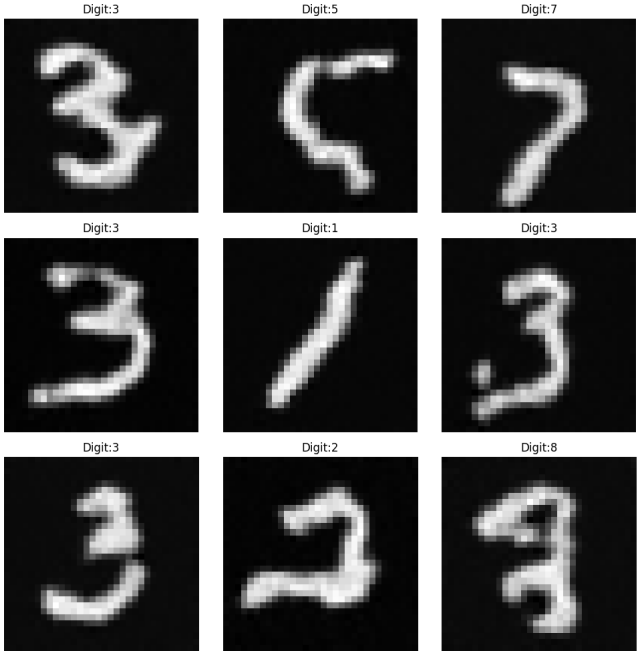

Figure 50: DDPM epoch 8

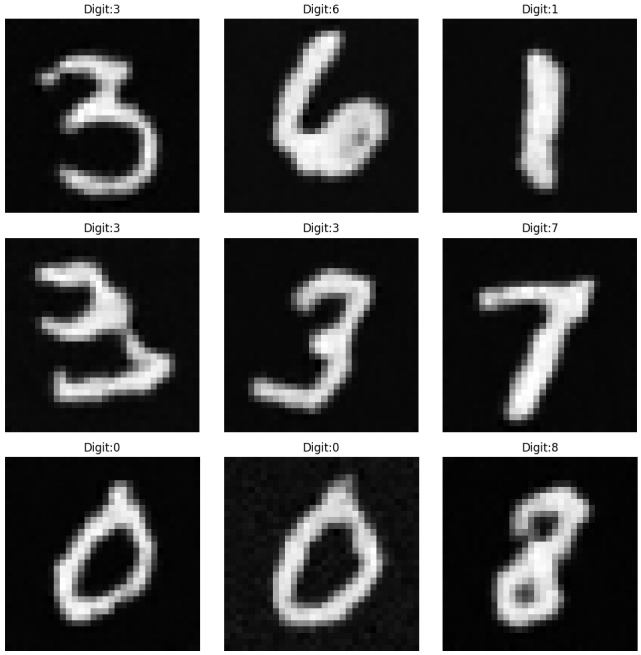

Figure 51: DDPM epoch 10

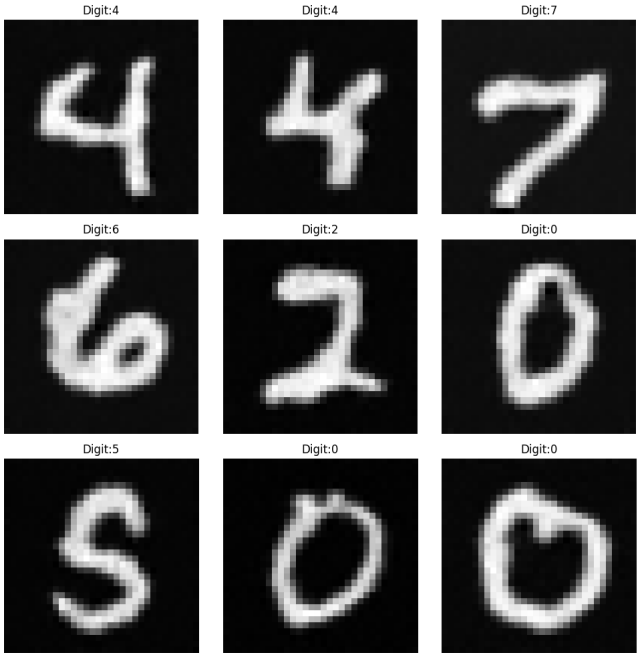

Figure 52: DDPM epoch 12

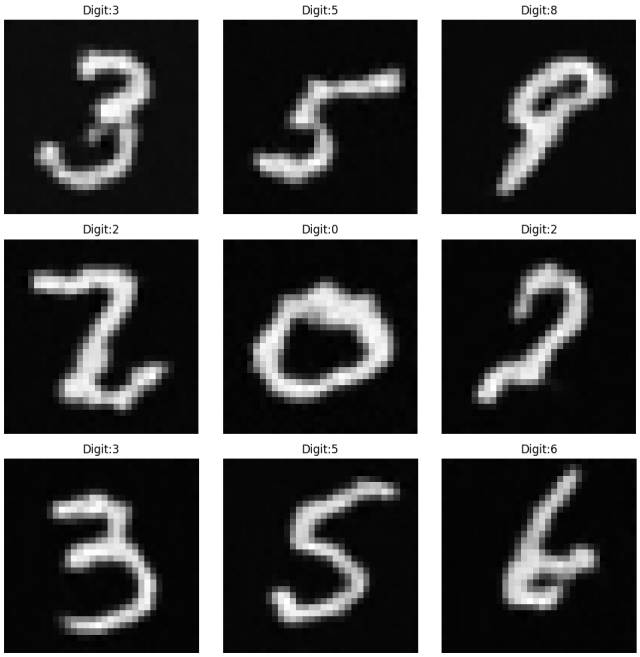

Figure 53: DDPM epoch 14

Analyse qualitative des images générées

Époque 0 (image 1)

- **Qualité visuelle :** Les chiffres sont très bruités, flous et difficilement reconnaissables.
- **Forme globale :** On devine vaguement des structures numériques, mais la majorité des pixels semblent aléatoires.

– **Conditionnement :** L'étiquette cible est affichée, mais les images ne correspondent pas aux chiffres attendus, ce qui indique que le modèle n'a pas encore appris à conditionner correctement la génération.

Époque 6 (image 3)

– **Qualité visuelle :** Les chiffres sont globalement bien formés, avec des contours nets et une bonne lisibilité.

– **Correspondance aux étiquettes :** La plupart des chiffres générés correspondent bien aux étiquettes affichées. Quelques cas (ex. `Digit:2` en haut à gauche) peuvent prêter à confusion, mais globalement le conditionnement fonctionne.

– **Stabilité du modèle :** Le modèle montre une amélioration visible par rapport à l'époque 0, mais reste légèrement en dessous des résultats observés à l'époque 14.

Époque 14 (image 2)

– **Amélioration significative :** Les chiffres sont nets, bien formés et facilement identifiables.

– **Correspondance aux étiquettes :** La majorité des images générées correspondent à la classe cible indiquée (par exemple, `Digit: 5` affiche bien un "5").

– **Guidance efficace :** Le conditionnement par étiquette semble fonctionner correctement à ce stade de l'entraînement.

Recommandations pour améliorer la qualité des images générées

– **Augmenter le nombre d'époques d'entraînement :** Même si les résultats à l'époque 14 sont bons, un entraînement plus long pourrait encore améliorer la fidélité et la diversité des chiffres générés.

– **Ajuster le coefficient de guidance (`w`) :** Un `w` trop faible peut nuire au conditionnement, tandis qu'un `w` trop élevé peut réduire la diversité. Il est recommandé de tester plusieurs valeurs (par exemple `w = 1.5`, `2.0`, `3.0`) pour trouver un bon compromis.

– **Utiliser un meilleur réseau de score (UNet) :** Un modèle plus profond, avec plus de filtres ou des mécanismes d'attention, pourrait améliorer la capacité du réseau à modéliser des chiffres cohérents.

– **Réduire le bruit initial (β_t) :** Un meilleur calendrier de bruit peut aider à une convergence plus rapide et plus stable.

– **Appliquer de la data augmentation sur MNIST :** Même pour des données simples, des transformations comme des rotations légères ou du jittering peuvent aider le modèle à mieux généraliser.

Entraînement avec 20 époques :

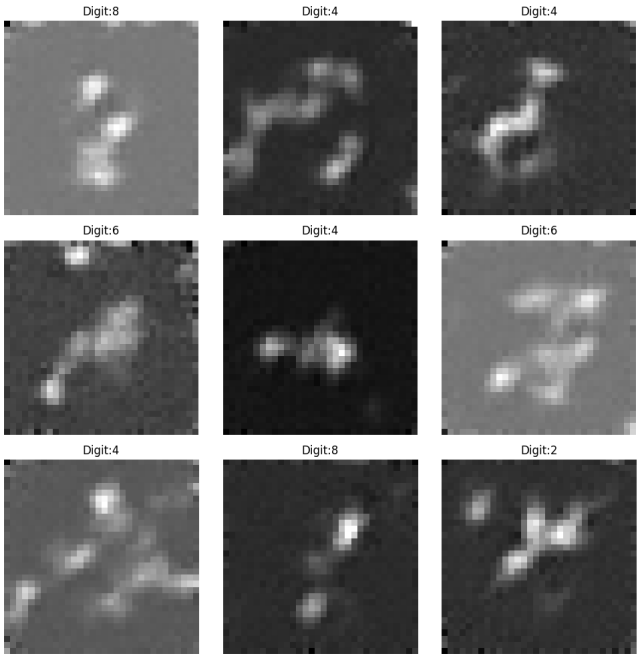

Figure 54: DDPM epoch 0

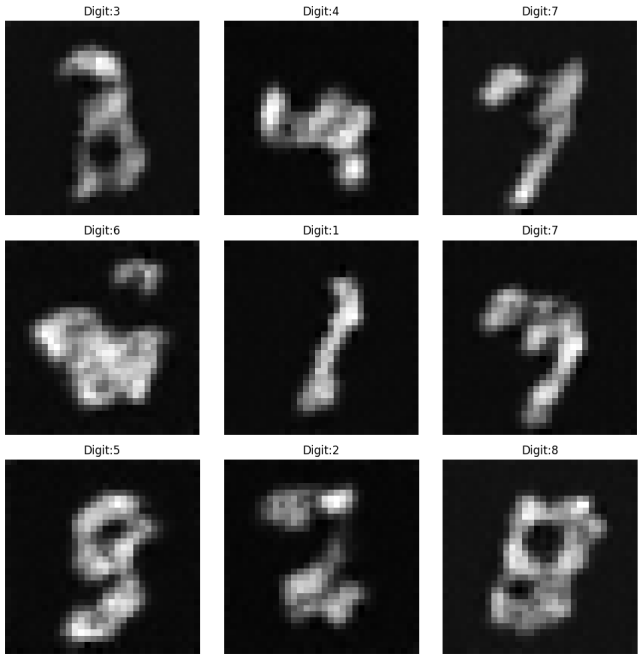

Figure 55: DDPM epoch 2

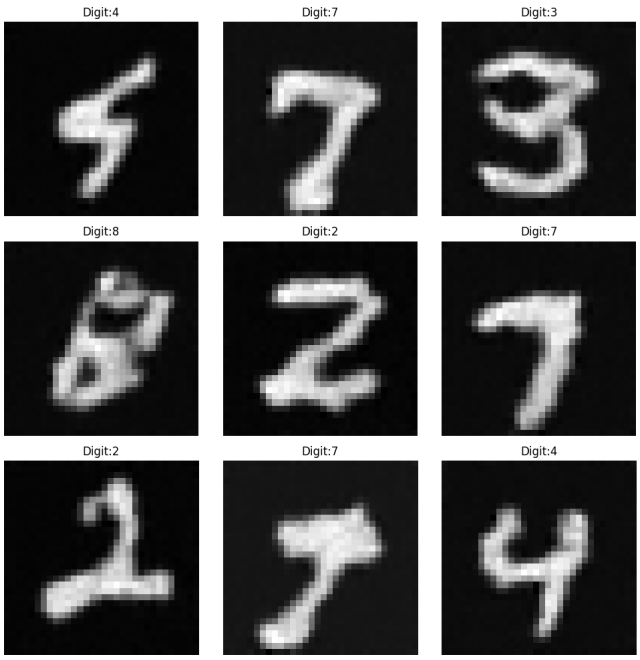

Figure 56: DDPM epoch 4

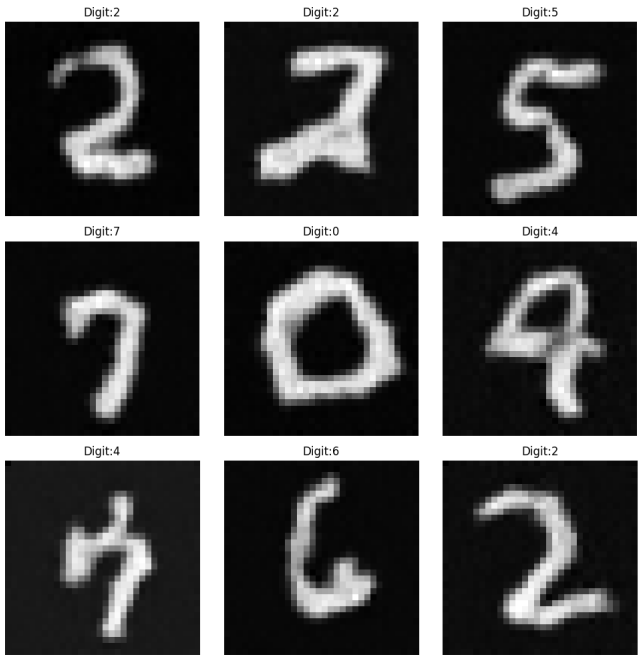

Figure 57: DDPM epoch 6

Pour plus de détails, veuillez consulter les instructions fournies dans le fichier notebook.

Veuillez NE PAS modifier le code fourni, mais seulement ajouter votre propre code aux endroits indiqués. Il est recommandé d'utiliser la session CPU (par exemple à partir de VSCode) pour déboguer lorsque le GPU n'est pas nécessaire, car Colab n'offre que 12 heures d'accès gratuit au

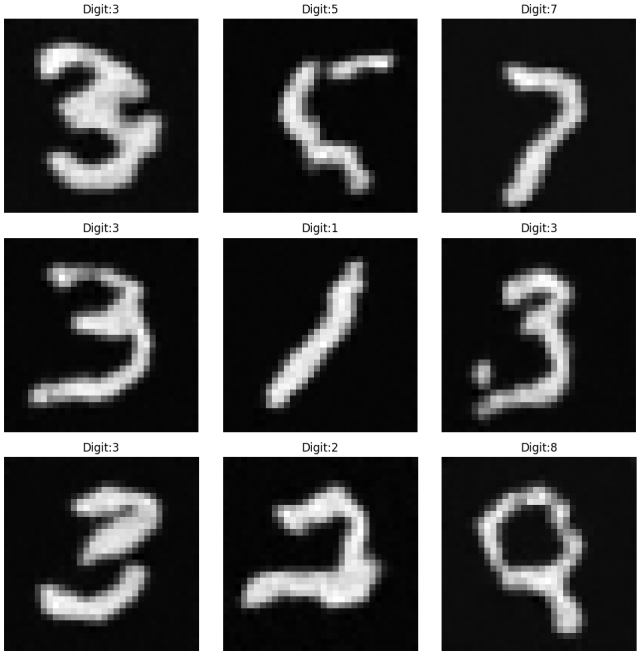

Figure 58: DDPM epoch 8

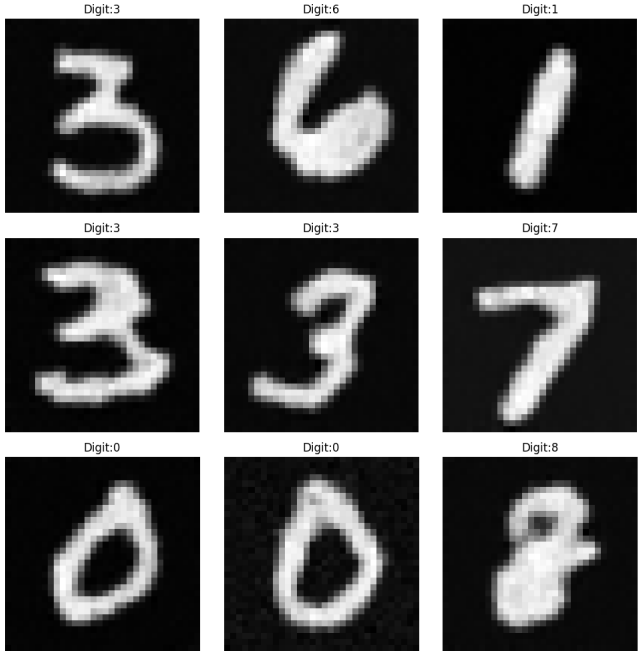

Figure 59: DDPM epoch 10

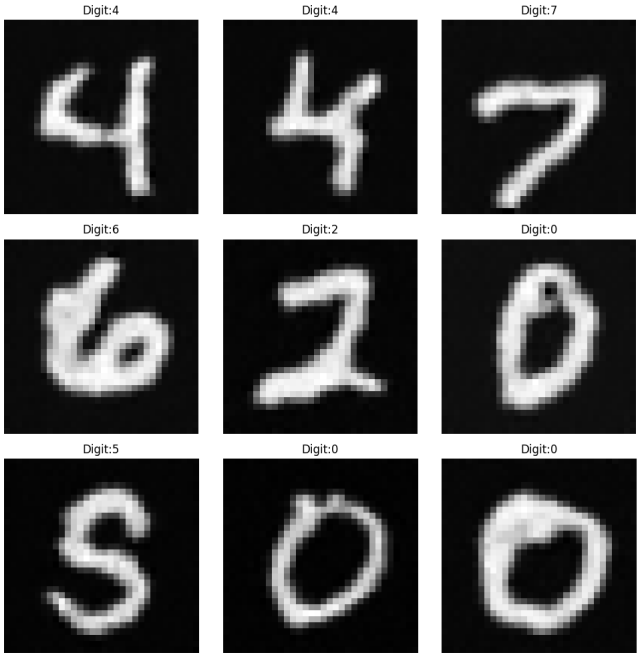

Figure 60: DDPM epoch 12

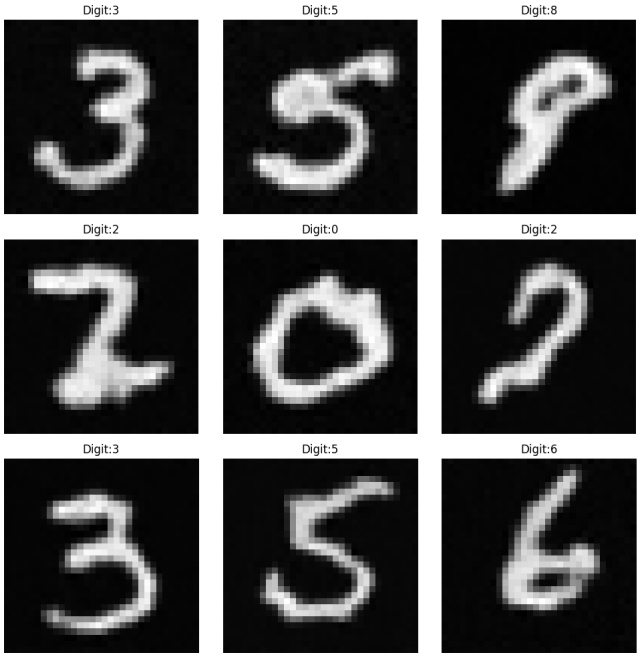

Figure 61: DDPM epoch 14

GPU à la fois. Si vous utilisez la ressource GPU, vous pouvez envisager d'utiliser la ressource GPU de Kaggle. Merci et bonne chance !

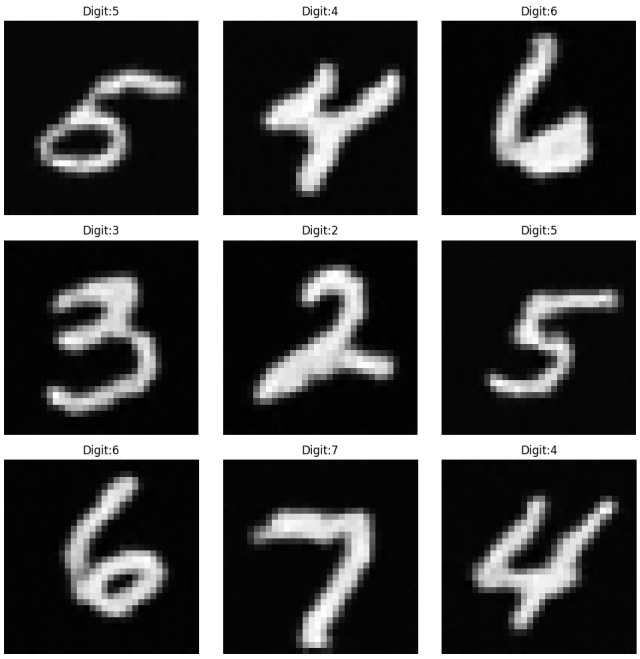

Figure 62: DDPM epoch 16

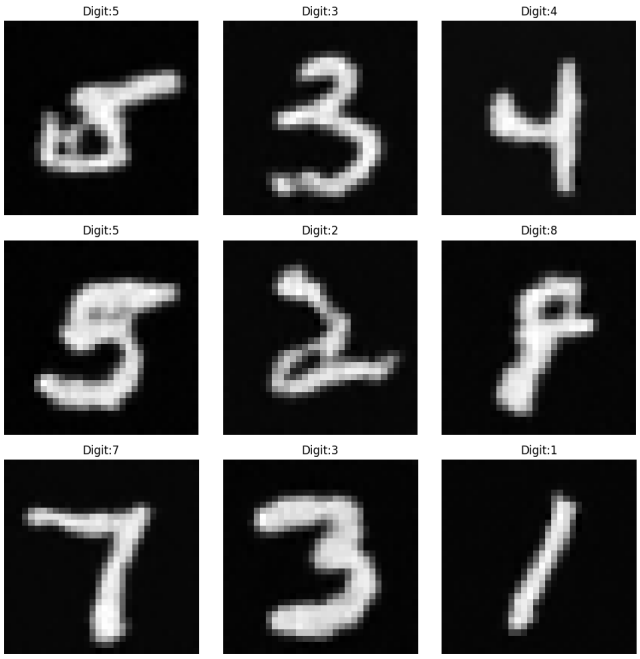

Figure 63: DDPM epoch 18