RÉDUCTION DE DIMENSION SUR DES DONNÉES DE GÉNÉRATION D'IMAGES DE CANARDS À L'AIDE DE PIPELINES VAE"

Margaux MOUTON
M1 Machine Learning
Université de Lille
margaux.mouton.etu@univ-lille.fr

François MULLER
M1 Machine Learning
Université de Lille
francois.muller.etu@univ-lille.fr

08/10/2024

RÉSUMÉ

Nous étudirons les Variational Auto Encoders (VAE), une classe de modèles génératifs utilisés pour la réduction de dimension et la génération de données. Nous nous concentrons sur l'espace latent des VAE et leur nature probabiliste, en soulignant sa capacité à représenter de manière efficace et structurée les informations importantes des données. Nous verrons les applications pratiques de l'espace latent, notamment sa capacité à générer de nouvelles données et à débruiter des images. Nous explorons l'espace latent des VAE, mettant en évidence son rôle central dans la représentation et la manipulation efficace des données dans le contexte de la réduction de dimension et du traitement d'image.

1 Introduction à la théorie derrière les VAE

Les VAE (*Variational Auto-Encoder*) sont une variante des auto-encodeurs (*AE*). Les auto-encodeurs sont un type de réseau de neurones qui ont comme particularité que la taille de la couche de sortie est égale à celle de l'entrée. Un AE dispose de deux composants, un encodeur et un décodeur. L'encodeur apprend à réduire la dimension de l'espace d'entrée dans un espace dit *latent*. Le décodeur lui reconstruit l'image à partir de l'espace latent. Nous appellerons f la fonction qui prend en entrée x et qui donne en sortie z, qui représente l'espace latent. On appelle g est la fonction qui prend en entrée z et qui retourne $\hat{x}$. [1] [2]

Nous avons donc :

$$x \xrightarrow{f} z \xrightarrow{g} \hat{x}$$

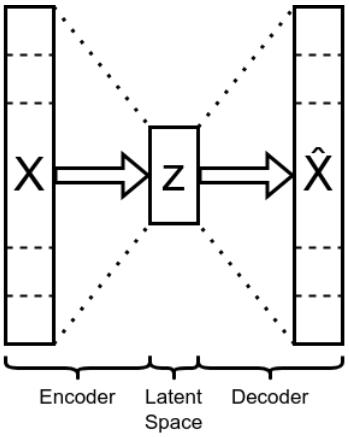


FIGURE 1 – Structure d'un auto-encodeur

L'objectif quand nous entraînons ce réseau est de réduire la différence entre x, la donnée d'entrée et sa reconstruction $\hat{x}$.

$$L(x, \hat{x}) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \tag{1}$$

Les auto-encodeurs apprennent à générer des représentations compactes et à bien reconstruire leurs entrées, mais à part quelques applications comme les auto-encodeurs de débruitage, leurs capacités sont assez limitées. Le problème fondamental des auto-encodeurs pour la génération est que l'espace latent vers lequel ils convertissent leurs entrées et où se trouvent leurs vecteurs encodés peut ne pas être continu ou permettre une interpolation facile. Cela signifie que, même si deux points de données sont très similaires, l'encodeur peut choisir de les positionner relativement loin l'un de l'autre dans l'espace latent si cela minimise la perte de reconstruction. Nous pouvons faire liens avec d'autres techniques de réductions qui ne conserves pas le voisinage comme PCA (*Principal Component Analysis*) contrairement à T-SNE ou U-map qui elles conservent le voisinage.. La sortie de la fonction d'encodage f dans une telle architecture génère un espace latent très discret. Par conséquent, bien qu'un AE puisse former un espace latent qui lui permette d'être très précis dans sa tâche, nous ne pouvons pas supposer grand-chose sur la distribution et la topologie de l'espace latent qu'il génère, ni sur la façon dont les données y sont organisées.

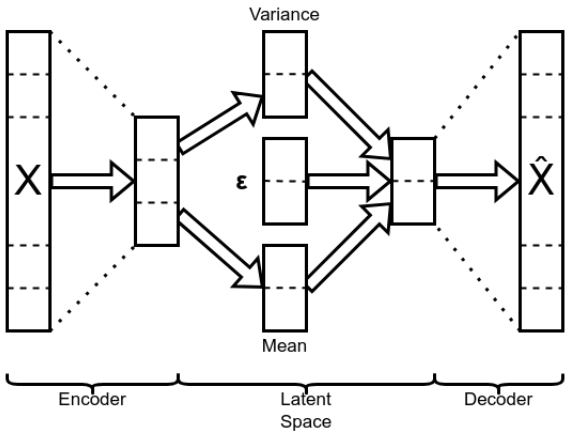


FIGURE 2 – Structure d'un auto-encodeur variationnel

L'espace latent d'un VAE (*Variational Autoencoder*) est échantillonné à partir d'une distribution que l'encodeur apprend pour chaque caractéristique latente. Pour avoir un espace latent plus interprétable qui représente mieux les données, nous devons, dans la fonction de perte, ajouter une contrainte sur celle-ci. Un VAE utilise une fonction de perte composée de deux éléments :

1. un composant de perte de reconstruction — qui oblige l'encodeur à générer des caractéristiques latentes minimisant cette perte de reconstruction, tout comme avec un AE (*Auto-encodeur*), sous peine d'être pénalisé ;
2. un composant de perte KL (*Kullback-Leibler*) — qui contraint la distribution générée par l'encodeur à être similaire à la probabilité *a priori* du vecteur d'entrée, supposée être normale, poussant ainsi l'espace des caractéristiques latentes vers une loi normale.

$$L(x, \hat{x}, z) = ||x - x'||^2 + D_{\text{KL}}(q(z|x)||p(z)) \tag{2}$$

Où $q(z|x)$ représente l'approximation de la distribution de probabilité qui projette les données dans l'espace latent et $p(z)$ la distribution de l'espace latent.
Si l'espace présente des discontinuités (par exemple, des lacunes entre les clusters) et que nous échantillonnons une variation à partir de l'un d'elles, le décodeur produira simplement une sortie irréaliste. Lorsque nous échantillonnons une variation à partir d'une région de l'espace latent qui n'est pas continue ou qui n'a pas été vue pendant l'entraînement, le décodeur peut produire une sortie irréaliste ou incohérente. Cela se produit parce que le décodeur a appris à reconstruire les données d'entrée à partir des vecteurs encodés dans l'espace latent, mais il n'a pas appris à traiter les régions de l'espace latent qui n'ont pas été vues pendant l'entraînement.

Pour résoudre ce problème, il est important de s'assurer que l'espace latent est suffisamment régulier et que les données d'entrée sont bien représentées dans l'espace latent. Les VAE (Variational Autoencoders) sont une technique couramment utilisée pour apprendre un espace latent régulier en ajoutant une contrainte de régularisation à la fonction de perte de l'auto-encodeur. Les VAE utilisent une distribution de probabilité, généralement une distribution gaussienne,

pour modéliser la distribution des vecteurs encodés dans l'espace latent. Cela permet d'échantillonner facilement des variations à partir de l'espace latent et de générer des données cohérentes et réalistes.

Comment une image est générée à partir d'un :

AE	VAE
1. D'abord, l'entrée est encodée comme un vecteur z dans l'espace latent.	1. L'entrée est encodée comme une distribution dans l'espace latent.
2. Le point vecteur de l'espace latent est décodé.	2. Un point de l'espace latent est échantillonné à partir de cette distribution ($moyenne + variance \cdot \epsilon$).
	3. Le point échantillonné est décodé.

La sortie d'un *AE* est déterministe alors que celle d'un *VAE* est probabiliste. Pour s'entraîner, le VAE utilise un mécanisme d'apprentissage supervisé où il tente de reproduire les données d'entrée en sortie, tout en obtenant une représentation latente efficace des données. La clef pour entraîner le réseau et faire fonctionner la rétro-propagation à travers la couche d'échantillonnage aléatoire dans un VAE est l'astuce de reparamétrisation (*reparameterization trick*). Cette astuce permet d'exprimer l'opération d'échantillonnage comme une transformation différentiable des sorties de l'encodeur (la moyenne μ et l'écart-type σ de la distribution latente). Cela permet aux gradients de circuler à travers la couche d'échantillonnage lors de la rétro-propagation. Voici comment cela fonctionne :

1. le réseau encodeur produit les paramètres de la distribution latente, généralement la moyenne (μ) et l'écart-type (σ) d'une distribution gaussienne ;
2. au lieu d'échantillonner directement à partir de cette distribution, l'astuce de reparamétrisation introduit une nouvelle variable aléatoire ϵ, qui est un échantillon d'une distribution normale standard ($\mathcal{N}(0, 1)$) ;
3. l'échantillon latent z est ensuite calculé comme suit :

$$z = \mu + \sigma \cdot \epsilon \tag{3}$$

Pour générer un échantillon à partir de l'espace latent, nous utilisons la formule ($\mathsf{moyenne} + \mathsf{variance} \times \epsilon$). Ici, ϵ joue un rôle crucial dans le maintien de la continuité de l'espace latent. Étant donné son caractère aléatoire, ϵ garantit que les points proches de l'emplacement encodé dans l'espace latent peuvent être décodés pour produire des résultats similaires au vecteur d'entrée. Ce mécanisme contraint l'espace latent à rester continu et significatif. Dans ce contexte, des points situés à proximité dans l'espace latent produiront des images très similaires. La combinaison de la continuité et de la faible dimensionnalité de l'espace latent garantit que chaque direction au sein de cet espace représente un axe de variation important des données. Cette opération d'échantillonnage reparamétrisée est différentiable par rapport aux sorties de l'encodeur (μ et σ), permettant aux gradients de circuler à travers la couche d'échantillonnage lors de la rétro-propagation. L'idée clef est que le hasard introduit par l'échantillonnage est maintenant isolé dans le terme ϵ, qui est indépendant des sorties de l'encodeur. Cela signifie que les gradients peuvent être calculés par rapport aux sorties de l'encodeur, même si l'échantillon latent final z implique une étape d'échantillonnage aléatoire. L'astuce de reparamétrisation est cruciale pour permettre un entraînement efficace des VAE en utilisant des méthodes d'optimisation basées sur les gradients, comme la descente de gradient stochastique. Sans elle, l'opération d'échantillonnage non différentiable bloquerait les gradients, rendant impossible l'entraînement de l'encodeur par rétro-propagation.

L'astuce de reparamétrisation utilisée dans les VAE garantit que ce processus d'échantillonnage est différentiable, permettant ainsi un entraînement de bout en bout du modèle VAE. La reparamétrisation consiste à réécrire l'échantillonnage stochastique en une fonction déterministe de la variable de bruit et des paramètres du modèle. Dans le cas d'une distribution gaussienne, nous pouvons échantillonner à partir d'une distribution gaussienne standard (moyenne de 0 et variance de 1) et ajouter la moyenne prédite par le modèle et multiplier par l'écart-type prédit par le modèle. Cela permet de déplacer la source de stochasticité en dehors de la fonction d'échantillonnage et de permettre la propagation rétrograde des gradients à travers le modèle.

Les échantillons de données générés auront des propriétés statistiques similaires aux données d'entraînement originales, car le VAE a appris à capturer la distribution sous-jacente des données pendant l'entraînement. Ce processus d'échantillonnage depuis l'espace latent et de passage à travers le décodeur permet de générer un nombre arbitraire de nouveaux échantillons de données cohérents avec la distribution des données d'entraînement.

2 Jeu de données

Nous avons créé notre propre jeu de données à l'aide du paquet `TikZducks`, qui permet de créer des canards en TikZ. Ce paquet permet de générer une diversité de canards avec différents accessoires tels que des lunettes, des chapeaux,

etc. Bien que MNIST soit un jeu de données utilisé de façon récurrente (à tort) pour l'entraînement de Variational Autoencoders (VAEs), notre jeu de données se révèle être un choix plus avantageux pour plusieurs raisons.

1. **Uniformité et Stabilité :** contrairement à MNIST, où les variations entre les chiffres sont nombreuses, notre jeu de données présente une uniformité avec des images centrées. Cette uniformité assure un modèle plus stable et cohérent.
2. **Diversité contrôlée et subtile :** l'outil TikZducks permet de créer un jeu de données avec une diversité contrôlée de canards, incluant des accessoires. Bien que les canards conservent une forme globale similaire, seuls les couleurs et les accessoires varient. Ces altérations offrent la possibilité d'explorer et d'apprendre des caractéristiques plus complexes et variées.
3. **Représentation plus réaliste :** notre jeu de données reflète une distribution plus réaliste que celle de MNIST. Il y a une légère variabilité est essentielle pour entraîner un VAE capable de généraliser à des données réelles.

En conclusion, notre jeu de données généré avec TikZducks se présente comme une excellente alternative. Il combine uniformité et stabilité tout en offrant une diversité contrôlée et subtile, permettant ainsi de capturer des représentations latentes plus expressives et générales.

Pour constituer notre jeu de données de 5000 canards, nous avons sélectionné des attributs particuliers et en extrait des données aléatoirement selon ceux-ci. Les critères sur lesquels les canards vont changer sont :

— la couleur de la tête ;
— la couleur du corps ;
— la couleur de l'aile ;
— la couleur du bec ;
— la présence des boutons ainsi que leurs couleurs ;
— la forme du bec (d'émotion "grincheuse" ou en train de rire ou la forme de bec de perroquet) ;
— la présence d'une baguette ;
— la présence de lunettes de soleil.

Avec le choix des couleurs, on arrive à avoir 57 attributs différents. Pour chaque canard, on attribue quatre de ces paramètres de manière aléatoire. Si on choisit un attribut modulable (par exemple l'aile avec ses couleurs ou la forme du bec), on ne va plus pouvoir rechoisir ce même attribut avec une autre couleur, renforçant ainsi le fait que chaque canard est défini par ces quatre attributs (par exemple, si on choisit que l'aile sera rose dans un premier temps, on ne choisira plus la personnification de l'aile par la suite). Ensuite, nous déterminons la taille souhaitée pour cet ensemble. Chaque entrée de cet ensemble est convertie en une image PNG, ajustée à une résolution de 28 par 28 pixels, similaire au format du jeu de données MNIST. Ces images sont ensuite transformées en nuances de gris, normalisées et intégrées dans un fichier CSV.

3 Génération d'images

Les *VAE* sont donc utilisés pour générer d'autres données. Nous les avons utilisés pour générer de nouveaux canards en utilisant notre propre jeu de données (comme décrit précedemment).

Notre modèle de VAE :

```
VAE(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=400, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=400, out_features=200, bias=True)
    (3): LeakyReLU(negative_slope=0.2)
  )
  (mean_layer): Linear(in_features=200, out_features=2, bias=True)
  (logvar_layer): Linear(in_features=200, out_features=2, bias=True)
  (decoder): Sequential(
    (0): Linear(in_features=2, out_features=200, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=200, out_features=400, bias=True)
    (3): LeakyReLU(negative_slope=0.2)
    (4): Linear(in_features=400, out_features=784, bias=True)
    (5): Sigmoid()
```

```
        )
    )
```

L'architecture de notre Variational Autoencoder (VAE) est bien structurée et suit les conventions typiques d'un VAE standard. Voici une explication détaillée de chaque composant de notre code :

Initialisation : nous définissons les dimensions d'entrée (`input_dim`), de la couche cachée (`hidden_dim`), et de l'espace latent (`latent_dim`), ainsi que le dispositif (`device`) sur lequel le modèle sera exécuté. l'encodeur est une séquence de couches linéaires suivies de fonctions d'activation LeakyReLU. Il réduit la dimensionnalité des données d'entrée pour produire des caractéristiques latentes. Les couches `mean_layer` et `logvar_layer` calculent respectivement la moyenne et le logarithme de la variance de la distribution latente.

fonction `encode` : cette fonction prend en entrée x (les données) et les fait passer à travers l'encodeur. Elle retourne la moyenne et le logarithme de la variance de la distribution latente.

fonction `reparameterization` : cette fonction implémente la reparamétrisation de la distribution latente pour permettre la différentiation par rapport aux paramètres de la distribution. Elle prend la moyenne et le logarithme de la variance (`var`) en entrée pour générer un échantillon latent z à partir duquel l'échantillon est tiré de manière stochastique.

fonction `decode` : l'encodeur est suivi d'un décodeur qui essaie de reconstruire les données originales à partir des caractéristiques latentes. Le décodeur est également une séquence de couches linéaires suivies de `LeakyReLU`, et se termine par une couche Sigmoid pour s'assurer que les valeurs de sortie sont comprises entre 0 et 1 (idéal pour les images en niveaux de gris).

fonction `forward` : cette fonction combine les étapes d'encodage, de reparamétrisation et de décodage. Elle prend les données en entrée et produit une reconstruction (`x_hat`) ainsi que la moyenne et le logarithme de la variance de la distribution latente.

Faux faire du sampling

Pour la génération d'images, nous avons introduit en entrée des images de canards comportant de nouvelles caractéristiques qui n'avaient pas été vues lors de l'entraînement. L'objectif était d'évaluer la réaction des VAE face à ces données inédites et de déterminer si les modèles étaient capables de produire quelque chose de cohérent. Dans les résultats observés, nous n'avons pas obtenu une reconstruction fidèle du canard avec son chapeau, mais le VAE a généré une image de canard qui demeurait cohérente et reconnaissable, démontrant ainsi sa capacité à généraliser et à produire des sorties pertinentes même avec des entrées non conformes à celles du jeu de données d'entraînement. Nous constatons également que pour les attributs qu'il identifie et avec lesquels il a été entrainé, comme les lunettes, la baguette ou les ailes foncées, le VAE produit une image qui inclut ces caractéristiques.

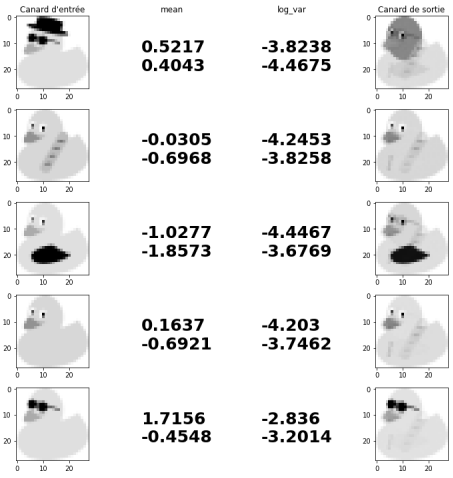


FIGURE 3 – Génération d'image à partir d'une image donnée

Dans la figure 3, nous avons voulu comparer le comportement d'un VAE sur les données vues à l'entrainement ou non. Les quatre derniers canards (baguette, ail noir, basique, lunette de soleil) ont été vus à l'entrainement. L'espace latent

est affiché au centre de la figure. Les images générées sont très proches des images d'origines. En revanche le canard avec chapeau qui n'a pas été vu à l'entrainement lui n'a pas une représentation similaire, mais qui reste cohérente et sensée puisque le canard est tout de même reconnaissable. Les canards 2 et 4 ont un espace latent proche ce qui donne en sorti un canard proche.

4 Espace latent interprétable

Dans notre cas, la zone latente de la moyenne et de la variance sont des vecteurs de taille 2, et la reconstruction avec le ϵ renvoie aussi un vecteur de taille 2. Ce qui équivaut à dire que, à partir d'une image de deux pixels, le décodeur peut reconstruire une image semblable aux images qu'il a vues durant son entraînement, tout en restant sensible aux changements dans ces deux valeurs. Ce sont ces variations des deux valeurs qui permettent d'attribuer plus de poids sur certain aspect (est-ce que le canard a une baguette, couleur des ailes, etc). Le choix de mettre seulement deux pixels provient du fait qu'il est plus simple d'interpréter l'influence de la zone latente sur la reconstruction par le décodeur en deux dimensions, comme si qui va être vu plus tard dans cette partie. Nous pensions qu'en augmentant le nombre de dimensions au nombre de groupes de critères (10 dans notre cas, pour les ailes, tête, baguette, etc, sans prendre en compte les couleurs), nous aurions une dimension représentant chacun de ses attributs. C'est-à-dire, par exemple, la valeur trois du vecteur de la zone latente change à quel point l'aile est foncé. Malheureusement ce ne fut pas le cas, donc nous sommes restés sur deux dimensions pour une meilleure interprétation de l'espace latent.

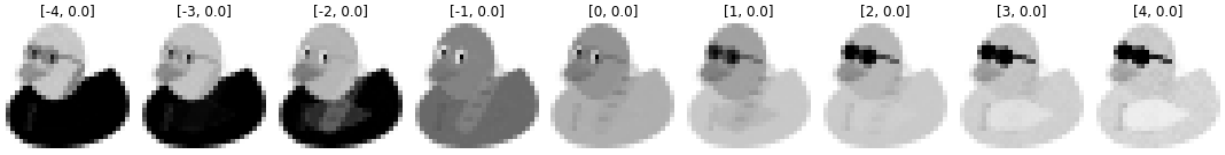


FIGURE 4 – Variation de l'espace latent sur la première valeur

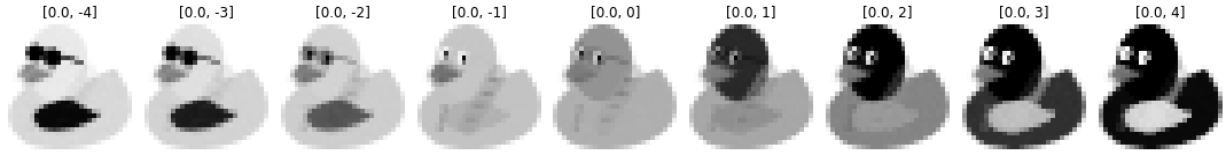


FIGURE 5 – Variation de l'espace latent sur la deuxième valeur

On peut simuler le décodage du VAE en passant des données directement dans la reconstruction de la zone latente, à laquelle on peut observer le résultat du décodage en fonction des données entrées. Comme notre zone latente d'entrée du décodeur est un vecteur comportant deux dimensions, il suffit de donner à une dimension une valeur arbitraire (ici, on a donné 0) et faire varier l'autre. On pourra alors observer le comportement du décodeur en fonction de la zone latente. On peut voir les évolutions en Figure 4 et 5. Si on interprète ces évolutions de l'espace latent, on peut voir que lorsque la première valeur du vecteur grandit, le canard passe d'un corps de couleurs foncé de manière graduelle à un corps plus clair, avec l'apparition des lunettes de soleil et d'une aile blanche. On remarque aussi que la présence de la baguette est la plus forte dans les alentours de 0. Dans la variation de la deuxième valeur (ici représenté dans la Figure 5), on débute avec un canard à l'aile foncée et des lunettes à un canard ayant la tête et le corps foncé et une aile blanche. La présence de la baguette est aussi bien plus forte dans les alentours de 0, comme dans la première variation.

Il faut aussi noter que cette interprétation change à chaque fois qu'on réentraîne le VAE. Par exemple, les Figures 4 et 5 proviennent du même VAE, mais celui de la Figure 6 provient d'un autre, ce qui peut altérer les interprétations de l'espace latent. Le but ici n'est pas de montrer ce qu'une valeur de l'espace latent peut donner, mais de bien montrer l'évolution de la sortie du décodeur en fonction de l'évolution des valeurs de l'espace latent.

On a analysé sur une seule dimension, mais comme notre espace latent est sur deux dimensions, on peut afficher les évolutions de celles-ci en deux dimensions, comme en Figure 6. Ici, on voit qu'il y a des groupements en fonction des caractéristiques de canard. Par exemple, on voit que les canards à lunettes sont dans le coin en bas à gauche et les canards aux corps foncé sont en haut à droite. Cette représentation est bien plus parlante, car on peut voir ici les évolutions des deux valeurs de l'espace latent, ce qui permet d'afficher presque toutes les possibilités de canard.

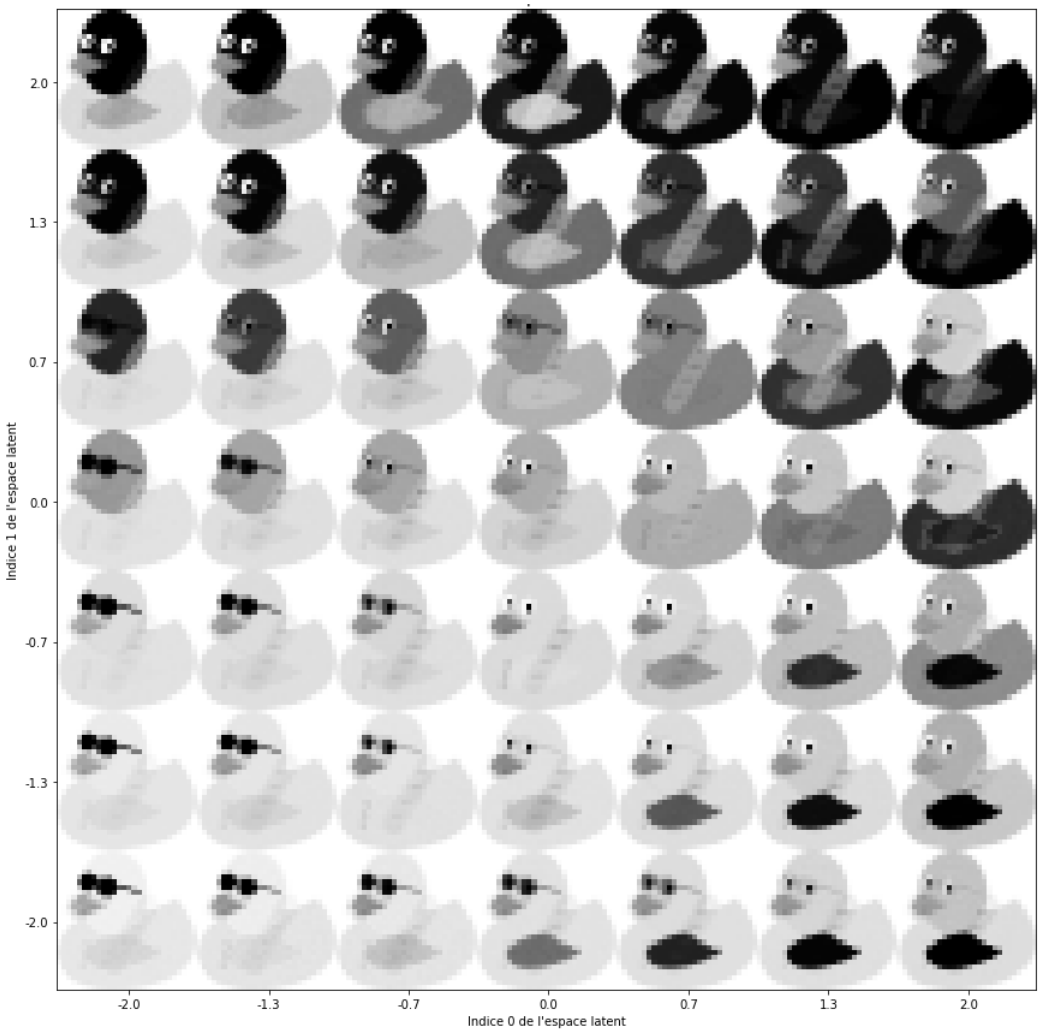


FIGURE 6 – Variation de l'espace latent sur les deux dimensions

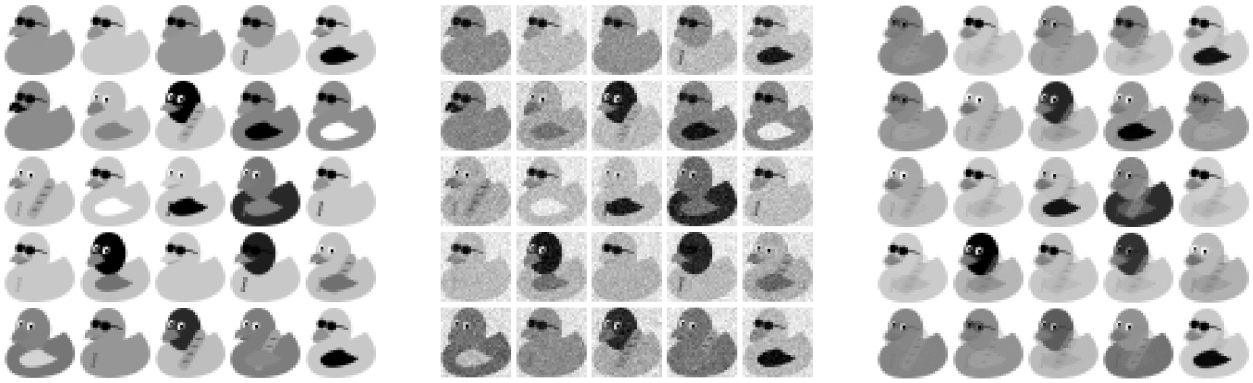
5 Débruitage d'image avec l'aide des VAE

Pour utiliser les VAE comme débruiteur, nous avons récupéré notre modèle initialement entraîné pour la génération d'images. En ajoutant du bruit gaussien à nos images, nous avons simulé des perturbations aléatoires qui peuvent être dues à diverses sources, telles que des imperfections de capteur ou des distorsions lors de la capture de l'image. Pour cela, nous avons généré une grille de bruit gaussien de la taille de l'image cible, ici 28x28 pixels, en échantillonnant une distribution normale centrée autour de zéro avec une variance de 0,1. Si on augmente la variance de l'épsilon, cela va effectivement augmenter la plage de valeurs possibles pour la variable rééchantillonnée. Cela peut être utile pour explorer un espace de solutions plus large, mais cela peut également rendre le modèle plus aléatoire et donc plus difficile à contrôler.

En effet, si la variance de l'épsilon est trop élevée, cela peut entraîner des valeurs rééchantillonnées qui sont très éloignées de la valeur prédite par le modèle. Cela peut rendre l'apprentissage du modèle plus difficile, car les gradients peuvent devenir très instables et donc difficiles à optimiser.

D'un autre côté, si la variance de l'épsilon est trop faible, cela peut limiter la capacité du modèle à explorer l'espace de solutions et à trouver des solutions optimales. Cela peut également rendre le modèle plus sensible aux valeurs aberrantes et aux erreurs de mesure.

Il est donc important de trouver une valeur de variance pour l'épsilon qui soit adaptée à la tâche à accomplir et aux données disponibles. En général, on utilise des valeurs de variance comprises entre 0,1 et 1.Cette grille, une fois créée, a été ajoutée pixel par pixel à l'image originale. Après cette étape, les valeurs des pixels de l'image résultante peuvent



(a) Images originales (b) Images bruitées (c) Images débruitées

FIGURE 7 – Débruitage des images

dépasser la plage de valeurs originale, souvent entre 0 et 255 pour une image en niveaux de gris. Afin de corriger cela, nous avons normalisé l'image, ajustant les valeurs minimale et maximale des pixels pour les ramener dans la plage originale. Le résultat final est une image bruitée, qui peut présenter une apparence plus granuleuse ou floue en fonction de l'intensité du bruit ajouté. Cette méthode d'ajout de bruit gaussien nous offre un moyen simple, mais efficace d'introduire des variations aléatoires et contrôlables dans les images.

En appliquant ce bruit aux images, nous avons constaté que le VAE était capable de reconstruire des versions débruitées des images originales. Grâce à sa capacité à apprendre les structures sous-jacentes des données et à les représenter de manière compacte dans l'espace latent, le VAE a pu éliminer efficacement le bruit gaussien tout en préservant les caractéristiques essentielles des images de canards. Ainsi, notre VAE s'est révélé être un outil efficace pour le débruitage d'images, démontrant sa polyvalence et sa capacité à traiter différents types de tâches de traitement d'images.

Nous avons utilisé le modèle VAE classique sans le réentraîner spécifiquement pour le débruitage. Cependant, nous avons constaté que la représentation latente est suffisamment robuste pour le bruit gaussien. En revanche, si le bruit est plus prononcé, comme une occultation par exemple, cet espace latent pourrait ne plus être adéquat. Dans de tels cas, il serait nécessaire de réentraîner le modèle pour obtenir une représentation latente plus adaptée au type spécifique de bruit ou de corruption rencontré. Afin de calculer à quel point le VAE corrige les images bruitées, on utilise la métrique des erreurs quadratiques moyenne (mean squared error ou MSE). On importe une implémentation du module `scikit-learn`, et qui va calculer la distance de chaque pixel à son pixel correspondant de l'autre image. C'est-à-dire, on va donner deux images à MSE, et la métrique nous renvoie la distance moyenne de chaque pixel qui sont au même indice. On va donc mesurer la différence entre les images propres et leurs équivalents bruitées, et ensuite des images propres et des reconstructions des images bruitées par le VAE. Si la distance entre les images propres et des images bruitées est plus grande que celle des images propres et de leurs reconstructions, alors on peut considérer que le VAE peut débruiter des images.

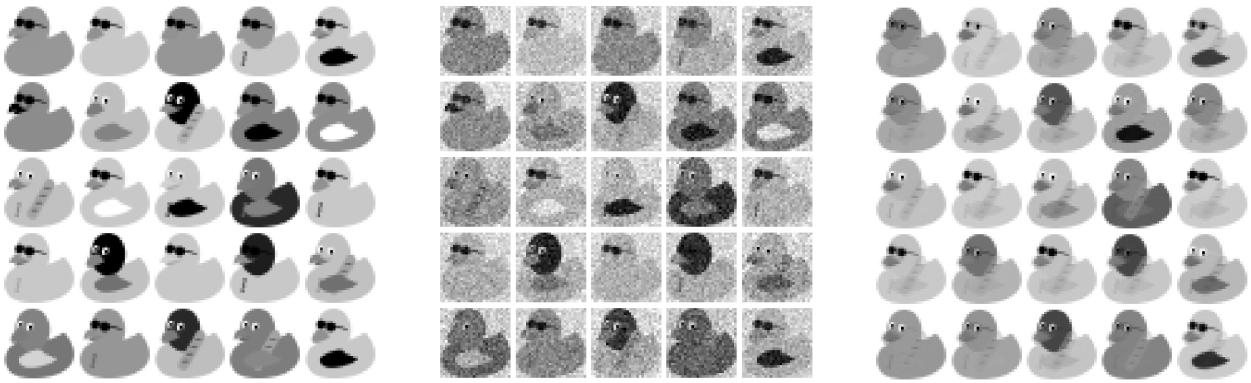
Dans la figure 7, les images (a) ont été passées dans un bruit, donnant les images (b), et le VAE débruite les images (b) d'une intensité gaussienne de 15, ici montré par les images (c). Quand on calcule la MSE entre (a) et (b) et on obtient 2.03×10^{-6}, et la MSE entre (a) et (c) est de 1.12×10^{-6}. On peut donc considérer que le VAE peut débruiter des images.

On cherche maintenant à savoir si le VAE peut aussi débruiter les images dont le bruit est bien plus fort. Ici, on va utiliser un bruit gaussien d'intensité de 30, comme ce qui a été fait dans Figure 8.

Les scores des MSE sont :
- entre les images originales et les images bruitées sont de 5.816×10^{-6} ;
- entre les images originales et les images reconstruites sont de 2.213×10^{-6} ;

On remarque qu'un bruitage fort des données ne pose pas de problème au VAE quant à la reconstruction des données.

L'efficacité des VAE provient de leur côté générateur, mais cela peut poser problème lorsque le bruit n'est pas assez fort. Si, par exemple, on donne une image en entrée qui n'est pas bruitée, le VAE peut reconstruire une autre image, car les VAE sont probabilistes et non déterministe. Il se peut aussi que lorsque le bruit n'est pas assez fort, le score MSE entre l'image originale et l'image bruité est plus petit que celui de l'image originale et de la reconstruction.



(a) Images originales (b) Images bruitées (c) Images débruitées

FIGURE 8 – Débruitage des images sous un bruit fort

Références

[1] Ian GOODFELLOW, Yoshua BENGIO et Aaron COURVILLE. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[2] Diederik P KINGMA et Max WELLING. *Auto-Encoding Variational Bayes*. 2022. arXiv : `1312.6114 [stat.ML]`.