Exploration des espaces latents issus des Réseaux Antagonistes Génératifs et des Autoencodeurs Variationnels

Augustin BARRUOL, Ilan BASTA, Lamia SALHI, Andrey SOBOLEVSKY
INF8225, Polytechnique Montréal
Intelligence Artificielle - Techniques probabilistes et apprentissage

Abstract

Dans ce rapport, nous nous proposons d'effectuer une recherche sur les espaces latents issus des Réseaux Antagonistes Génératifs (GANs) et des Autoencodeurs Variationnels (VAEs). Au fil de votre lecture, vous pourrez notamment retrouver la construction de chacun de ces modèles ainsi qu'une exploration en profondeur de leur espace latent pour comparer les meilleurs d'entre eux et comprendre leur organisation. Notre implémentation est disponible sur le dépôt suivant https://github.com/dysoxor/INF8225_projet.

1 Introduction

1.1 Intérêt pour le sujet

Le projet soumis à notre attention a pour objectif de nous faire apprendre plus en profondeur un aspect de l'intelligence artificielle suscitant notre intérêt.

De ce fait, de nombreux sujets s'offraient à nous. Au cours de nos discussions et des sujets abordés en cours, nous avons porté notre attention sur la génération d'images par ordinateur. En effet, c'est un sujet récent et très populaire dans le domaine de l'intelligence artificielle pour lequel de nombreuses méthodes ont vu le jour. Nous pensons notamment au domaine de l'art avec *OpenAI* et Dall-E, un modèle générant une image à partir d'une simple description. Au cours de notre projet, nous nous intéresserons à deux méthodes : la première correspond aux *Réseaux Antagonistes Génératifs (GAN)* tandis que la seconde est l'*Auto-Encodeur Variationnel (VAE)*.

1.2 Travaux antérieurs

Différents travaux ont partiellement exploré la question de l'interprétation de l'espace latent des GANs et des VAEs.

La première chose importante à noter est que la structure d'un espace latent vient du générateur et n'a pas de sens en lui-même [2]. La plupart des recherches déjà effectuées se basent sur des explorations non-supervisées, comme par exemple, l'interpolation entre deux échantillons de l'espace latent afin d'observer la transition visuelle dans l'espace des données ou aussi, l'arithmétique de vecteurs dans l'espace latent pour observer un transfert de style dans l'espace des données [1].

Cependant, les articles s'accordent à dire qu'il existe des limitations dans la diversité des sémantiques interprétables. En effet, la malléabilité de l'espace latent dépend grandement du modèle entraîné et généralement, plus le modèle est complexe, plus son espace latent permet des opérations variées mais avec le défaut d'être souvent plus difficile à entraîner (c'est notamment le cas des GANs) [2].

1.3 Objectif du projet

Dans le cadre de la génération d'images, un VAE récupère un vecteur issu de l'espace latent qu'il a encodé dans le but de générer une nouvelle image. Dans un premier temps, nous nous intéresserons à l'exploration de cet espace latent. En effet, il est en général admis que l'espace latent est difficilement interprétable et qu'il est donc complexe d'en comprendre la structure. Cependant, il a été observé qu'en variant des points de l'espace latent, il est possible d'en conclure certaines propriétés. L'objectif de notre projet sera alors d'explorer les espaces latents issus du VAE et du GAN que nous aurons implémentés et entraînés.

Ces modèles étant difficiles à entraîner, nous prendrons un jeu de données avec des images en faible résolution pour obtenir un temps d'entraînement raisonnable. Nous effectuerons alors une étude de cas pour comparer les observations que nous ressortirons de nos meilleurs modèles à base d'interpolations, d'opérations arithmétiques et de visualisations sur l'espace latent.

2 Approche théorique

2.1 Auto-Encodeurs

Pour présenter les autoencodeurs variationnels, nous nous devons d'abord de faire un récapitulatif succinct de ce qu'est un autoencodeur. Ceux-ci sont utilisés pour l'apprentissage non-supervisé. En effet, l'autoencodeur est un réseau qui apprend un encodage de ses instances en entrée.

La structure d'apprentissage de l'autoencodeur consiste à reconstruire l'information en entrée dans une dimension beaucoup plus réduite. Par exemple, dans le cas d'images en entrée, on peut passer d'une dimension N avec N un nombre conséquent correspondant à la taille de l'image, à 2 dimensions. Cet espace réduit est appelé *espace latent*, il représente l'élément clé dans le fonctionnement de l'autoencodeur.

En effet, après cette phase d'encodage qui permet de générer l'espace latent vient le décodage. Celui-ci consiste, à partir de l'espace latent, à reconstruire la dimension initiale de notre représentation et ainsi de générer une nouvelle instance, basée sur l'apprentissage que nous en avons fait dans l'encodeur. Somme toute, le modèle est entrainé par descente de gradient pour prédire sa propre entrée. En considérant la sortie comme une probabilité, nous pouvons représenter le fonctionnement de l'autoencodeur tel que nous cherchons à optimiser

$$p(x = \hat{x}|\tilde{x}) \tag{1}$$

où $\hat{x}$ est la valeur en sortie du modèle sachant l'observation $\tilde{x}$ en entrée. L'objectif de l'autoencodeur est alors de garder au mieux l'information lors de la phase d'encodage et de réduire au mieux l'erreur ou l'écart de reconstruction lors de la phase de décodage, et donc de création de la nouvelle instance.

Cependant, la régularité de l'espace latent qui permet de générer notre nouvelle instance dépend de nombreux facteurs, notamment les données initiales, la dimensionnalité de notre espace latent et la structure de l'encodeur. De ce fait, nous n'avons pas de réelle garantie que l'espace latent sera structuré de manière à pouvoir générer correctement notre nouvelle instance. Pour pallier à cela, les autoencodeurs variationnels ont été créés.

2.2 Auto-Encodeurs variationnels

Conceptuellement, les autoencodeurs variationnels peuvent être interprétés comme une classe fille ayant hérité des principes de l'autoencodeur classique mais spécialisant l'espace latent.

En effet, l'autoencodeur variationnel émet des hypothèses concernant la distribution des variables latentes. Sans cette hypothèse, l'autoencodeur serait simplement entraîné à encoder et décoder avec le moins de perte possible sur la reconstruction, mais sans égard quant à la structure de l'espace latent. De fait, des phénomènes de surapprentissage seraient observés.

Avec notre hypothèse de distribution, nous passons d'un modèle *déterministe* pour l'autoencodeur à *stochastique* pour le VAE. En effet, plutôt que d'encoder directement en une représentation, on encode nos entrées comme une distribution autour de l'espace latent puis nous effectuons un *échantillonnage* de cette distribution qui servira d'entrée au décodeur pour la reconstruction.

En résumé, nous avons $p(x = \hat{x}|\tilde{x})$ notre distribution sur l'espace latent et une variable $Z \sim p(x = \hat{x}|\tilde{x})$ telle que Z est décodée pour nous donner notre variable générée et nous continuons ainsi l'entraînement par rétropropagation de l'erreur.

Notons que la distribution sur l'espace latent est supposée **Gaussienne** et permet donc d'encoder nos entrées selon une distribution normale avec μ notre moyenne et σ la variance.

Disposant dorénavant d'une distribution pour la régularisation, la fonction de perte d'un VAE est alors composée d'un terme de *reconstruction* évalué sur la couche finale et d'un terme de *régularisation* évalué sur la couche latente, appelé *Kullback-Leibler divergence*:

$$L_{\theta,\phi}(x) = D_{KL}(q_\phi(z|x)||p_\theta(z)) - E_{q_\phi(z|x)}(logp_\theta(x|z)) \tag{2}$$

2.3 Réseaux antagonistes génératifs à convolutions denses

Le second modèle étudié dans ce travail est le réseau antagoniste génératif à convolutions denses (DCGAN).

Tout d'abord, il faut savoir que les réseaux antagonistes génératifs (GANs) sont des modèles non-supervisés qui étudient l'entrée, qui correspond à l'espace latent de notre modèle, et génèrent de nouveaux exemples qui ressemblent à ceux de l'ensemble de données utilisé. Pour ce qui est de la tâche étudiée dans ce projet, nous voulons générer des images à partir de l'espace latent.

Conceptuellement, le modèle est constitué de deux réseaux : le *générateur* et le *discriminateur*.

Le *générateur* est le réseau de neurones qui prend en entrée un espace latent et génère une image. Il est à noter que souvent, l'espace latent est une quantité fixe de chiffres aléatoires car on veut que l'image générée soit originale.

Le *discriminateur* quant à lui est un réseau dont le but est, à partir d'une entrée qui est une image, de classifier l'image comme étant soit une image réelle, c'est à dire provenant de la base de données, soit comme étant une image fausse, c'est à dire générée. C'est grâce à ce réseau que ce modèle est non-supervisé car il peut dire au générateur si l'image qu'il a généré est réaliste ou non.

Il est intéressant de remarquer que la différence du DCGAN par rapport au GAN est que celui-ci utilise des réseaux à convolutions denses au sein même du *générateur* et du *discriminateur* ce qui donne de meilleurs résultats lorsqu'on veut générer des images.

Lorsque nous entraînons ce modèle, à chaque époque, nous commençons par générer un espace latent aléatoire que l'on donne au générateur. Ensuite, on va passer l'image générée dans le *discriminateur* qui va prédire si c'est une image réelle (= 1) ou fausse (= 0). Nous lui donnons également une image issue de la base de données. Suite à cela, l'époque se finalise en calculant la perte pour le discriminateur et le générateur ainsi qu'en propageant l'erreur dans le but d'améliorer les réseaux.

L'apprentissage est régit en propageant l'erreur sur le discriminateur et le générateur obtenu avec la fonction suivante:

$$L = \min_G \max_D [\log(D(x)) + \log(1 - D(G(z)))] \tag{3}$$

2.4 Les espaces latents

Comme énoncé précédemment, le domaine de la génération d'images est très en vogue puisqu'il est lié à de nombreuses applications variées aujourd'hui (art, deep fakes, augmentation de données, ...). Les modèles qui produisent ces images utilisent la réduction de dimension : un entraînement sur les données permet de les caractériser pour créer un nouvel espace, où les éléments principaux de l'image sont gardés. Cet espace est nommé espace latent. Chaque point de cet espace peut ensuite être ramené dans l'espace des données, afin de correspondre à une nouvelle image.

Les espaces latents dans les GANs et les VAEs sont extrêmement intéressants car non seulement ils constituent un espace compressé des images que l'ont souhaite générer mais c'est en plus un espace qui est interprétable. L'interprétabilité

dans le domaine de l'intelligence artificielle est rare et est très recherchée. Un exemple d'un domaine où l'on aurait besoin d'interpréter l'espace latent serait le clustering des types de cellules en génétique et génomique. Nous pourrions comprendre comment les images générés (par exemple ici les gènes) changent d'un type de cellule à un autre (d'un cluster à un autre) dans l'espace latent. Il est alors important d'avoir un bon clustering et une bonne interpolation. Dans le cadre de l'utilisation du jeu de données MNIST on peut considérer que les types de cellules sont les différentes classes (0, 1, 2, ..., 9). Il est donc intéressant de comprendre comment est construit cet espace et comment se font les transitions d'un cluster à un autre.

Pour projeter les espaces latents, nous utiliserons, sauf mention du contraire, la projection t-SNE qui tente de préserver la topologie des données et qui révèle extrêmement bien les clusters et sous-clusters des données. La t-SNE est une méthode non-supervisée et non-linéaire qui permet de représenter des données multi-dimensionnelles dans un espace à deux (ou trois) dimensions, ce qui facilite l'interprétation. Dans l'application, elle se rapproche de l'analyse en composante principale (ACP) mais diffère dans le principe par le fait qu'elle s'intéresse aux similarités locales tandis que l'ACP cherche à maximiser la variance en préservant les grandes distances entre les paires de points.

3 Méthodologie

3.1 Choix du jeu de données

Pour notre jeu de données, nous avons fait le choix de partir du jeu de données MNIST et ses chiffres écrits à la main, issu du site de Yann Lecun [3]. En effet, avec sa définition 28x28 pixels et plus de 60 000 images labellisées, nous trouvons le bon compromis entre des images de petites tailles et un jeu de données conséquent. La petite taille des images nous permettra ainsi un temps d'entraînement rapide de nos modèles. Cependant, les images étant relativement basiques, lors de la présentation des résultats, nous aurons moins de profondeur d'interprétation, notamment pour les opérations d'interpolation.

3.2 Entraînement du VAE

Initialement, nous souhaitions récupérer un modèle pré-entraîné pour ensuite pouvoir faire notre exploration de l'espace latent. Néanmoins, partant du jeu de données MNIST, nous sommes arrivés à la conclusion qu'il était possible pour nous, dans les temps impartis, d'entraîner nos propres modèles. Pour ce faire, nous avons récupérer la structure de code issu de [4] sur laquelle nous nous sommes basés tout au long du projet.

En effet, nous récupérons notamment l'implémentation du VAE et des fonctions d'affichage et reconstruction de l'espace latent encodé. La structure du VAE reprend l'organisation attendu avec un Encodeur et un Décodeur comme illustré sur le schéma suivant :

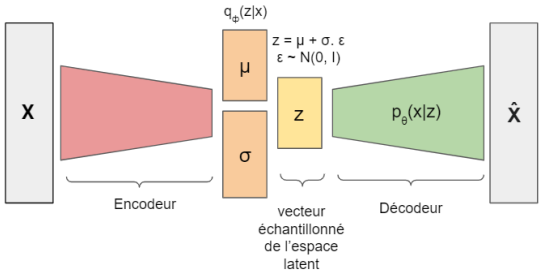


Figure 1: Schéma de l'AutoEncodeur Variationnel

Dans cette configuration, X représente notre image réelle et $\hat{X}$ représente notre image reconstruite basée sur l'échantillonnage de l'espace latent passé par le décodeur. L'encodeur est structuré en 3 couches linéaires, la première effectue une transformation d'une dimension 784 à 512, puis les deux dernières de 512 à la dimension de l'espace latent exigée (mettons 2 pour un espace en *2D*, voir *Annexe 6.1*). Notons que les deux dernières couches sont identiques et serviront chacune à générer respectivement μ et σ telles que représentées sur le schéma, en sortie de l'encodage.

Dès lors, en initialisant une variable ϵ selon une distribution normale d'espérance 0 et de variance 1, nous pouvons établir nos échantillons **z** en sortie de l'espace latent tel que

$$z = \mu + \sigma.\epsilon \tag{4}$$

A cet endroit précis du processus d'encodage, nous calculons la divergence *Kulback-Leibler* pour accumuler sa quantité et la propager dans notre fonction de perte lors de la rétropropagation.

Finalement, le décodeur consiste en deux couches linéaires permettant de remettre notre image encodée dans sa dimension initiale. Pour la première couche linéaire, une fonction d'activation *sigmoïde* est initialement appliquée sur la transformation linéaire, de même pour la couche de sortie vers l'image reconstruite. Au cours de nos explorations, nous ferons varier la fonction d'activation de notre couche linéaire précédent la couche de sortie pour voir l'influence d'icelle sur les images générées.

3.3 Entraînement du DCGAN

Dans le but de réaliser un DCGAN qui est capable de générer des images, nous sommes partis d'un modèle pré-entraîné que l'on a pu également entraîner en reprenant la même architecture implémentée sur le répertoire [6]. L'implémentation est faite sous la librairie Pytorch.

L'architecture du modèle est reprise ci-dessous avec toutes les composantes qui ont été expliquées à la section **2.3**.

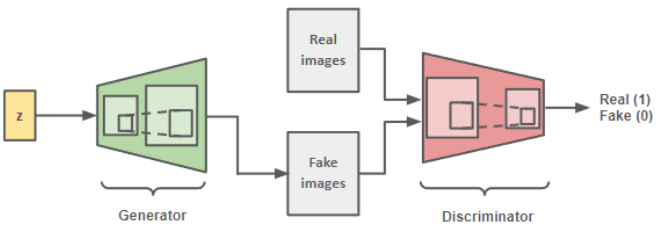


Figure 2: Schéma du reseau antagoniste génératif à convolutions denses

Dans cette architecture, le Z représente l'espace latent qui est uni-dimensionnel et de longueur 100 dans le modèle pré-entraîné.

Le générateur est constitué d'une séquence de 4 convolutions, batch normalization et fonction d'activation Rectified Linear Unit (ReLU) pour, à la fin, finir avec une convolution finale et une fonction d'activation Tangente Hyperbolique (Tanh). Nous pouvons d'ailleurs observer sur la description du générateur en annexe **6.2 : Générateur DCGAN** le résumé plus complet de l'architecture où l'on voit explicitement les dimensions à chaque étape.

Pour ce qui est du discriminateur, celui-ci est tout d'abord construit à l'aide d'une convolution puis d'une fonction d'activation LeakyReLU. Il s'en suit une séquence de 3 convolutions, batch normalization et LeakyReLU. Enfin, nous terminons avec une convolution et une fonction d'activation sigmoïde pour pouvoir classer si l'image est réelle, c'est à dire égale à 1, ou générée (fausse), c'est à dire égale à 0. Nous pouvons une nouvelle fois observer l'architecture complète du discriminateur en annexe **6.3 : Discriminateur DCGAN**.

Lors de l'entraînement, nous enregistrons les modèles à chaque époque car le meilleur modèle n'est pas forcément le tout dernier. De plus, il faut faire attention à deux fonctions de pertes et pas seulement une comme c'est traditionnellement le cas. Si nous observons la Figure 3, qui représente les fonctions de pertes pour les deux réseaux lors de l'entraînement, ce graphe ne montre pas une amélioration du modèle à chaque époque mais plutôt un combat entre les deux réseaux. En effet, lorsque le générateur a une fonction de perte qui diminue, le discriminateur a une fonction de perte qui augmente et c'est la même chose inversement. Le meilleur modèle est celui qui a la fonction de perte la plus petite possible pour les deux réseaux. A la fin de l'entraînement, nous observons que les dernières époques donnent de très mauvais résultats, reprenant bien l'affirmation selon laquelle le meilleur modèle n'est pas forcément le tout dernier.

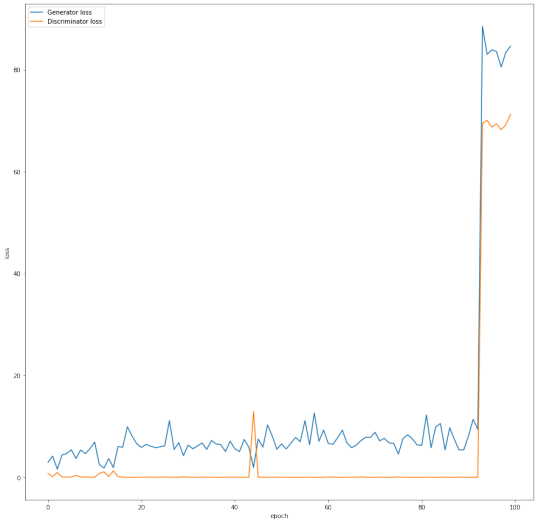


Figure 3: Représentation des fonctions de perte du DCGAN (avec fonction d'activation ReLU) pour le générateur en bleu et le discriminateur en orange.

3.4 Opérations appliquées aux espaces latents

Pour les opérations appliquées sur l'espace latent, nous ne nous baserons pas sur la base de code proposée par [4] mais plutôt sur l'article [5] qui effectue une revue des opérations possibles sur un espace latent appliquées sur le dataset *CelebA*.

Ainsi, nous nous intéresserons aux :

- Interpolations linéaires
- Interpolations sphériques
- Opérations arithmétiques sur les vecteurs représentatifs
- Représentations de l'espace latent par interpolation

D'un côté, **l'interpolation linéaire** entre deux points permet d'effectuer des reconstructions partant d'un point ciblé à un autre point ciblé. Entre le départ et l'arrivée, on retrouve alors les différentes reconstructions et transitions au sein de l'espace latent en traversant les différents points jusqu'au point ciblé. Ainsi, on a une idée des transitions effectuées et du parcours au sein de l'espace latent pour arriver à l'état final. Nous retrouvons ainsi l'équation suivante :

$$v = (1-t)v_0 + tv_1 \tag{5}$$

où t correspond au nombre de transitions.

De l'autre côté, **l'interpolation sphérique** entre deux points permet de prendre en compte l'aspect courbé de l'espace. Cette opération sera en particulier utile sur les espaces latents de grandes dimensions, elle est régie par l'équation suivante :

$$v = \frac{sin[(1-t)\theta]}{sin\theta}v_0 + \frac{sin[t\theta]}{sin\theta}v_1 \tag{6}$$

où t correspond au nombre de transitions et θ l'angle sous-tendu par l'arc de cercle dans l'espace.

De plus, nous explorerons les **opérations arithmétiques** qui consistent à récupérer des vecteurs représentatifs de l'espace latent et à les additionner ou les soustraire afin d'identifier des patterns propre à l'espace latent généré.

Enfin, nous explorerons **la représentation** de notre espace latent **en interpolant** des points sur une grille de taille $n \times n$ pour pouvoir analyser la structure et l'organisation de celui-ci.

Dans un dernier temps, nous effectuerons de **la visualisation de nos espaces latents**, puis nous tenterons d'examiner les frontières représentant la transition entre deux zones (*typiquement, la transition entre le chiffre 3 et le chiffre 5*), et de visualiser à partir de quel moment nous commençons, en tant qu'utilisateur, à interpréter un changement.

4 Résultats

Après avoir présenté nos modèles et leur architecture, nous pouvons maintenant passer à la présentation des résultats. Dans cette partie, vous pourrez tout d'abord retrouver individuellement l'analyse des résultats sur le VAE puis le DCGAN. Les expérimentations sont en deux parties, la première est une étude de l'influence de la fonction d'activation sur les performances de nos modèles, et la deuxième est une étude de l'influence de la dimensionnalité de l'espace latent.

Dans un second temps, vous pourrez retrouver une analyse comparative entre nos deux meilleurs modèles du VAE et du DCGAN issus de nos expérimentations.

4.1 Résultats du VAE

Fonction d'activation

Dans un premier temps, nous avons fait varier la fonction d'activation au niveau du décodeur pour déterminer son influence sur la génération d'images mais aussi choisir celle qui nous donnait le meilleur résultat. Nous avons donc entraîné le VAE avec, en plus de la fonction d'activation Sigmoïde proposée par défaut, ReLU et Tanh.

D'après les résultats mis en avant par la Figure 5 (*Annexe page 10*), nous observons que globalement, les résultats sont plutôt similaires pour chacune de nos fonctions d'activation. En effet, pour la reconstruction des images en entrée, nous constatons à travers nos trois grilles d'assez bons résultats avec la grande majorité des chiffres que nous pouvons distinguer. Néanmoins, ils sont globalement flous et nous avons quelques chiffres difficilement reconnaissables.

Cependant, lorsque l'on s'attarde sur l'interpolation linéaire puis sphérique de 3 vers 5, nous constatons que typiquement, pour la fonction *Sigmoïde*, notre modèle part bien de 3 mais tend difficilement vers 5, à tel point qu'à l'arrivée, nous avons l'impression d'être au point de départ (i.e avoir un 3). De même, le modèle avec *Tanh* part bien de 3 mais nous affiche un résultat final qui ressemble à une fusion entre un 5 et un 0, difficilement interprétable. *ReLU* semble tout de même plus se distinguer, bien que l'interpolation soit largement représentée par des 3, les deux dernières transitions nous amène vers un squelette de 5. Ces résultats semblent cohérents au vu de l'organisation des espaces latents car nous voyons à travers la visualisation que pour la *Sigmoïde* et *Tanh* les clusters de 3 et 5 ne semblent pas très homogènes et éparpillés au centre, ce qui expliquerait la transition difficile. Néanmoins, avec *ReLU*, on constate une certaine homogénéité de ces deux clusters qui semblent bien séparé par le 8 et le 6.

Si les résultats sont tout de même très similaires, dans la suite de nos résultats, nous nous baserons sur la fonction d'activation ReLU.

Comparaison de dimensions d'espace latent

Dans un second temps, nous avons comparé les résultats en faisant varier la dimension de l'espace latent. Comme dit précédemment, nous nous fixons comme fonction d'activation ReLU et nous effectuons des entraînements pour des dimensions de l'espace latent de 2, 50 et 100 (Figure 6 *Annexe page 10*). Nous avons choisi d'effectuer une projection t-SNE pour chacune des représentations ainsi qu'une comparaison de différents échantillons et une interpolation linéaire et sphérique.

Nous observons pour la dimension 2 que les différents clusters sont facilement différentiables. Les classes sont bien séparées sauf quelques unes comme le 3 en rouge ou le 8 en jaune. On retrouve un aspect de "vague" qui est propre à la projection t-SNE. Pour les dimensions supérieures (50 et 100), on remarque que les résultats de la projection du t-SNE sont très approximatifs : les clusters ne sont pas clairement différentiables, ils s'entrecoupent, même si l'on observe que chaque classes reste majoritaire dans sa zone.

Pour les échantillons de chaque modèle, on observe que plus le nombre de dimension augmente, plus il y a des incohérences. Par exemple, à 100 dimensions, on observe des chiffres 8 très aplatis ou alors parfois même des formes dont il est difficile de déterminer le chiffre correspondant. En dimension 2, tous les chiffres sont distinguables même si certains sont légèrement flous.

Pour les interpolations linéaire et sphérique, on remarque encore une fois une incohérence en dimension 100, où l'interpolation donne une résultat mixé entre un 3 et un 2, difficilement distinguable. Pour les dimensions 2 et 50, les résultats sont très corrects.

Avec ces résultats, on peut arriver à la conclusion qu'à ce modèle de VAE convient mieux un espace latent de faible dimension, ce qui facilite l'interprétation (comme on peut l'observer sur les projections t-SNE) tout en gardant des résultats très corrects (au vu des échantillons et des interpolations).

4.2 Résultats du DCGAN

Fonction d'activation

De la même manière que pour le VAE nous avons tenté différentes fonctions d'activation pour le générateur du DCGAN. Trois modèles avec les fonctions ReLU, Tanh et Sigmoide ont été entraînés.

Pour la fonction sigmoïde, les tests n'ont pas été concluants car dès que le modèle est en présence de grand nombre positifs/négatifs, la fonction sature et empêche la mise à jour des poids avec un gradient toujours à zéro. Par contre, pour la fonction tanh (qui a ses valeurs centrées en 0 contrairement à la fonction sigmoïde) on obtient des résultats plus intéressants mais assez médiocres.

En effet, sur la Figure 7 (*Annexe page 11*) on observe que certains nombres sont très peu lisibles et que les images générées sont peu diversifiées, ce qui est en accord avec la projection de l'espace latent où certains nombres sont quasiment inexistants comme le 0 *(couleur bleu foncé)* et le 3 qui est peu présent *(couleur rouge)*. Les points de même couleur sont peu étalés sur la projection et il y a beaucoup d'espaces blancs ce qui montre une faible diversification des images générées, comme observé plus tôt.

Par ailleurs, avec l'interpolation linéaire, on observe que le passage du 3 au 5 n'est pas fluide avec un 3 et un 5 peu lisibles tandis qu'avec l'interpolation sphérique le modèle génère un chiffre entre le 5 et le 9. En comparaison, le modèle avec ReLU offre des interpolations linéaires et sphériques fluides et des images diversifiées. La projection t-SNE nous montre une bonne diversification et la plupart des nombres sont bien clusterisés. Cependant, nous pouvons observer que le 4 et le 9 se mélangent et que les points du chiffre 6 sont séparés, donnant ainsi des clusters hétérogènes. Sudipto Mukherjee, Himanshu Asnani, Eugene Lin et Sreeram Kannan proposent un nouveau modèle, le ClusterGAN, pour remédier à ce problème et obtenir un meilleur clustering avec les GANs [10].

Comparaison de dimensions d'espaces latent
Ensuite, pour ce qui est de l'analyse sur la dimensionnalité, il est à noter que les dimensions 2, 100 et 200 ont été comparées en utilisant la fonction ReLU pour le générateur.

Sur la Figure 8 (*Annexe page 11*), nous observons qu'un espace de dimension 2 nous donne déjà d'excellents résultats mais nous observons un manque de diversité à la fois dans les images générées et dans la projection t-SNE. Tous les points de même couleur sont localisés au même endroit et ne sont pas éparpillés contrairement à la dimensions 100.

Ivana Marin and co., dans leur papier de recherche [7], étudient la dimension optimale à choisir pour un GAN dans le cadre de la génération d'images sur le jeu de données CelebA. Selon leurs expériences, c'est la dimension 100 qui donne les meilleurs résultats et c'est d'ailleurs ce qui est utilisé en général dans la littérature. Parfois, de plus grandes dimensions sont utilisées comme 128 et 512, la seule condition est qu'il faut que l'espace latent reste petit devant les dimensions de notre sortie. Ici, nous avons essayé avec 200 dimensions et on peut voir que notre modèle a tendance à ne générer que les chiffres 1, 0, 2 et 6 mais presque jamais de 5 et peu de 3 et c'est ce qui se retranscrit sur les interpolations linéaires et sphériques de 3 à 5 qui sont très mauvaises.

De plus, sur nos représentations, nous constatons que les chiffres qui sont de même classe sont aussi quasiment identiques et la projection t-SNE nous le confirme avec beaucoup d'espace blancs et des clusters peu distribués. Tous les points d'une même classe sont collés. Par conséquent, la dimension 100 reste la meilleure dimension comme ce qu'on peut lire dans la littérature.

4.3 Comparaison des espaces latents du VAE et du DCGAN

En nous basant sur les résultats mis en avant dans les parties précédentes, nous avons pu identifier que le VAE nous donnait de bons résultats en étant encodé dans de faibles dimensions (2D) tandis que la tendance est contraire avec le GAN, il a besoin de plus de dimensions pour performer, dans notre cas 100 dimensions. Dans cette partie, nous nous attarderons sur une comparaison de nos des espaces latents issus de ces deux modèles.

Tout d'abord, comme vous pouvez le voir dans la Figure 4 à la page suivante, il apparaît tout d'abord que les images générées par le GAN sont plus nettes que celles générées par le VAE. En effet, nous retrouvons là une différence connue entre les deux qui est que les réseaux antagonistes génératifs produisent des images moins floues que le VAE. Cela est notamment dû à la différence dans la fonction objective puisque le VAE minimise la différence entre l'entrée et l'image générée selon l'erreur carrée tandis que le DCGAN est entraîné pour produire des échantillons qualitatifs ressemblant à nos échantillons d'entraînement.

De plus, lorsque l'on compare les interpolations linéaires et sphériques de nos deux modèles, nous nous rendons compte que l'interpolation proposée par le DCGAN est beaucoup plus nette que celle proposée par le VAE. En effet, non seulement les images représentées pendant la transition sont claires, mais la transition du 3 vers le 5 est aussi beaucoup plus rapide puisque l'on constate la présence du 5 dès la moitié de l'interpolation tandis qu'avec le VAE, nous devons attendre les deux dernières étapes pour distinguer un début de squelette ressemblant à un 5. Dès lors, la supériorité des performances du DCGAN par rapport au VAE par rapport aux interpolations est sans appel.

Nous pouvons par ailleurs tenter comparer les modèles du point de vue des opérations arithmétiques. En effet, nous souhaitons déterminer s'il y a une logique dans nos espaces latents. Une opération arithmétique consiste à prendre l'espace latent de chiffres choisis au préalable et d'opérer des opérations arithmétiques telles que les additions et les soustractions. Notons que l'on ne va pas utiliser la multiplication et la division car, dans le premier cas, les valeurs des espaces latents peuvent avoir des valeurs qui atteignent des valeurs trop extrêmes et dans le second cas il se pourrait alors que des division par 0 surviennent. Nous sommes donc restés sur les additions et les soustractions et en observant les résultats à la Figure 10 (*Annexe page 12*) il semblerait que les différents espaces latents issus de chacun de nos deux modèles n'ont aucun lien arithmétique entre eux. Plus de résultats ont été générés dans les codes et peuvent être retrouvés pour appuyer ce constat. Notons que cette absence de résultats semble cohérente avec le fait que le jeu de données MNIST met en avant des images basiques à partir desquelles il apparaît difficile de déceler des liens complexes entre les espaces latents.

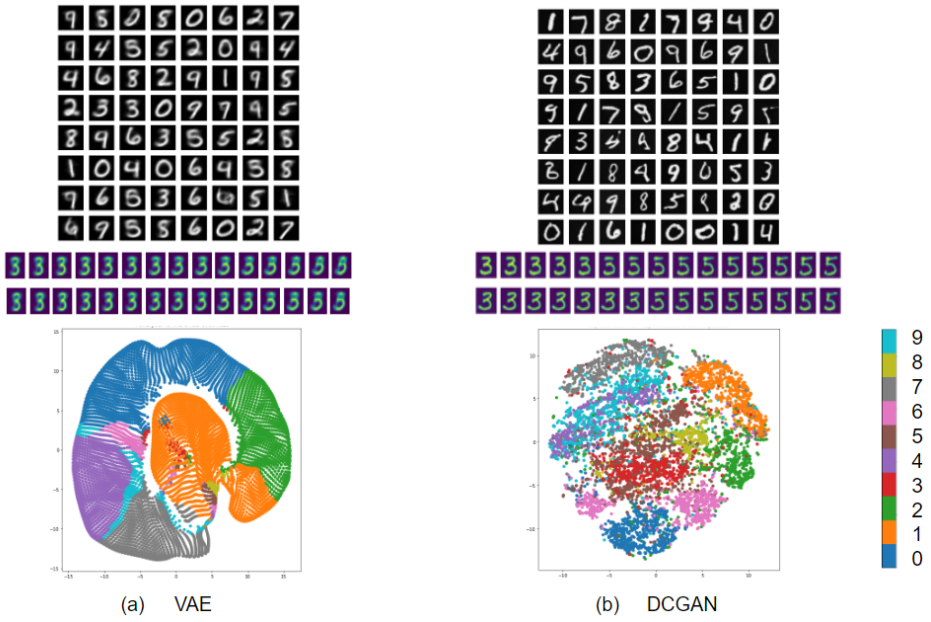
Enfin, lorsque l'on s'attarde sur les représentations de nos deux espaces latents par la projection t-SNE, il apparaît que pour nos deux modèles, les clusters semblent bien regroupés bien que loin d'être totalement homogènes. Nous remarquerons dans les deux cas que certains clusters sont scindés voire eparpillés en prenant notamment pour exemple le 6 et le 9 dans le cas du VAE (*en bleu clair et rose*) de même que le 6 pour le DCGAN. De plus, on remarquera que le 9 et le 4 sont entremêlés dans la représentation du DCGAN et que c'est le cas pour de nombreux groupes dans le VAE.

Cependant, les résultats de la projection sont à relativiser puisque nos modèles n'ont pas été entraînés dans le but d'effectuer une bonne clusterisation dans la représentation de nos données. En effet, comme mentionné dans [10], il est possible de changer la structure de nos modèles pour pouvoir les spécialiser dans la tâche d'une bonne clusterisation et nous pouvons justement prendre en exemple le *ClusterGAN* mis en avant dans l'article cité.

Pour néanmoins mieux comprendre à la fois la logique de nos interpolations mais aussi la représentation de nos modèles, nous avons travaillé sur une implémentation interactive représentant les images générées à chaque instant dans l'espace que nous présentons dans la partie suivante.

4.4 Visualisation dynamique

Nous avons aussi pu effectuer une représentation interactive de l'espace latent du VAE (Figure 9 , *Annexe page 12*). Cette représentation est construite à partir d'un quadrillage de $n \times n$ points dans l'espace latent. À partir de chaque point, nous pouvons reconstruire une image associée. Ainsi en glissant la souris sur ce point nous pouvons observer l'image reconstruite correspondante. Dès lors, cette image est ensuite classifiée par un réseau de neurones convolutif (CNN) classique



(a) VAE (b) DCGAN

Figure 4: Visualisation des résultats du DCGAN et du VAE selon nos meilleurs Pour (a) notre VAE encodé en 2 dimensions et (b) notre DCGAN pour une dimension 100. La première ligne correspond à des images générés par les modèles, la deuxième ligne à une interpolation linéaire du chiffre 3 au chiffre 5, la 3ème ligne à une interpolation sphérique du chiffre 3 au chiffre 5 et la 4 ligne correspond à la visualisation de l'espace latent en 2D par projection t-SNE de l'espace latent.

qui permet de classifier MNIST avec une précision moyenne de 99.7%, afin de lui donner un label. Ce type de visualisation est très pratique pour explorer facilement l'espace latent et comprendre les transitions d'un cluster à un autre. Cependant, cette représentation ne fonctionne dans notre cas que pour une représentation à deux dimensions et donc uniquement pour le VAE, qui donne des résultats interprétables pour un espace latent à deux dimensions.

5 Conclusion

5.1 Discussion

Au cours de notre projet, nous avons étudié l'effet des choix de conception sur les modèles du VAE et du DCGAN. Nos expérimentations nous ont permis de comprendre que la dimension d'un espace latent peut améliorer ou dégrader la qualité des images générées et nous pouvons également l'observer sur la projection de l'espace latent t-SNE. Ce résultat est toutefois à nuancer car la projection t-SNE a ses limites et demande parfois un grand nombre de points pour avoir une réelle représentation des clusters.

De même, nous avons observé les effets de la variation de la fonction d'activation sur les performances de nos modèles et il apparaît que la fonction d'activation *ReLU*, qui ne sature pas dans les régions positives, nous offre les meilleures performances pour nos deux modèles.

Somme toute, à travers la représentation t-SNE, les différentes opérations sur l'espace latent (interpolation linaire, sphérique et opération arithmétique) ainsi que la visualisation interactive des images générées en interpolant les points de notre espace latent, nous disposions de nombreux outils pour comparer correctement notre VAE avec notre DCGAN. Comme discuté dans la présentation des résultats, le DCGAN nous offre les meilleures performances avec des images générées beaucoup plus nettes et des transitions sans aucune trace de bruit dans les différentes interpolations, contrairement au VAE pour lequel les résultats sont plus flous et les transitions moins interprétables qualitativement.

Ces opérations nous auront permis d'explorer l'espace latent généré par nos deux modèles et la visualisation interactive Figure 9 nous aura aussi permis de mieux comprendre à partir de quel moment la transition d'un chiffre vers un autre est effectuée de manière spatiale, en transitant notamment sur les frontières de nos clusters. Pour pouvoir manipuler cette fonction comme il se doit, nous vous renvoyons aux instructions de notre Github.

5.2 Perspective d'études

Tout au long du projet, nous nous sommes concentrés sur le dataset MNIST notamment car le faible nombre de classes dans ce jeu de données nous permettait d'observer des espaces latents simples. Cependant, les opérations d'interpolations et opérations arithmétiques sont limitées et difficilement interprétables (notamment pour les opérations arithmétiques). De ce fait, dans la suite de ce projet, il serait intéressant d'élargir à un jeu de données comme Fashion-MNIST ou CelebA qui donnent de meilleures visualisations pour les opérations d'interpolations et arithmétiques.

De plus, il serait intéressant d'approfondir les recherches sur les opérations possibles que nous pourrions effectuer sur l'espace latent de manière à explorer la structure de celui-ci d'une manière encore différente.

Aussi, il est à noter que dans notre projet, nous avons utiliser un modèle simple de VAE tandis qu'il existe d'autres variante supposées être plus performantes telles que Cond-VAE et CSVAE [2]. Enfin, au cours de nos recherches, nous avons découvert l'existence du modèle VAE-GAN [11] qui combine le VAE et le GAN en un modèle génératif non-supervisé qui apprend à encoder, générer et comparer des échantillons de manière simultanée et il pourrait alors être intéressant d'étendre notre étude à ce modèle.

References

[1] Jack Klys, Jake Snell, Richard Zemel - Learning Latent Subspaces in Variational Autoencoders, University of Toronto, Vector Institute (2018)

[2] Toan Pham Van, Tam Minh Nguyen, Ngoc N. Tran, Hoai Viet Nguyen, Linh Bao Doan, Huy Quang Dao, Thanh Ta Minh - Interpreting the Latent Space of Generative Adversarial Networks using Supervised Learning (2021)

[3] Yann Lecun (Courant Institute, NYU), Corinna Cortes (Google Labs, New York) et Christopher JC Burges (Microsoft Research, Redmond) - MNIST Database of handwritten digits, sur http://yann.lecun.com/exdb/mnist/

[4] Alexander Van de Kleut, Variational AutoEncoders with Pytorch, sur https://avandekleut.github.io/vae/ (2020)

[5] How to Explore the GAN Latent Space When Generating Faces , (2019) - Jason Brownlee PhD. Website : https://machinelearningmastery.com

[6] Architecture de base du DCGAN récupérée sur Github - https://github.com/csinva/gan-vae-pretrained-pytorch/tree/master/mnist_dcgan

[7] Ivana Marin, Sven Gotovac, Mladen Russo, and Dunja Bozic-Stulic, The Effect of Latent Space Dimension on the Quality of Synthesized Human Face Images

[8] Jason Brownlee, How to Explore the GAN Latent Space When Generating Faces, July 3, 2019

[9] Christopher Olah, Visualizing MNIST: An Exploration of Dimensionality Reduction, October 9, 2014, https://colah.github.io/posts/2014-10-Visualizing-MNIST/

[10] Sudipto Mukherjee,Himanshu Asnani,Eugene Lin, Sreeram Kannan, ClusterGAN : Latent Space Clustering in Generative Adversarial Networks

[11] Shambhavi Mihsra, An Introduction to VAE-GAN https://wandb.ai/shambhavicodes/vae-gan (2021)

[12] Joseph Rocca, Théorie des autoencodeurs variationnels https://towardsdatascience.com (2019)

6 Annexe

6.1 Structure du VAE

```
VariationalAutoencoder(
  (encoder): VariationalEncoder(
    (linear1): Linear(in_features=784,
               out_features=512, bias=True)
    (linear2): Linear(in_features=512,
               out_features=2, bias=True)
    (linear3): Linear(in_features=512,
               out_features=2, bias=True)
  )
  (decoder): Decoder(
    (linear1): Linear(in_features=2,
               out_features=512, bias=True)
    (linear2): Linear(in_features=512,
               out_features=784, bias=True)
  )
)
```

6.2 Générateur DCGAN

```
Generator_relu(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512,
        kernel_size=(4, 4), stride=(1, 1),
        bias=False)
    (1): BatchNorm2d(512, eps=1e-05,
        momentum=0.1, affine=True,
        track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256,
        kernel_size=(4, 4), stride=(2, 2),
        padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05,
        momentum=0.1, affine=True,
        track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128,
        kernel_size=(4, 4), stride=(2, 2),
        padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05,
        momentum=0.1, affine=True,
        track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64,
        kernel_size=(4, 4), stride=(2, 2),
        padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05,
        momentum=0.1, affine=True,
        track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 1,
        kernel_size=(4, 4), stride=(2, 2),
        padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

6.3 Discriminateur DCGAN

```
Discriminator(
  (main): Sequential(
    (0): Conv2d(1, 64,
        kernel_size=(4, 4),
        stride=(2, 2), padding=(1, 1),
        bias=False)
    (1): LeakyReLU(negative_slope=0.2,
        inplace=True)
    (2): Conv2d(64, 128,
        kernel_size=(4, 4),
        stride=(2, 2), padding=(1, 1),
        bias=False)
    (3): BatchNorm2d(128, eps=1e-05,
        momentum=0.1, affine=True,
        track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2,
        inplace=True)
    (5): Conv2d(128, 256,
        kernel_size=(4, 4),
        stride=(2, 2), padding=(1, 1),
        bias=False)
    (6): BatchNorm2d(256, eps=1e-05,
        momentum=0.1, affine=True,
        track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2,
        inplace=True)
    (8): Conv2d(256, 512,
        kernel_size=(4, 4),
        stride=(2, 2), padding=(1, 1),
        bias=False)
    (9): BatchNorm2d(512, eps=1e-05,
        momentum=0.1, affine=True,
        track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2,
        inplace=True)
    (11): Conv2d(512, 1,
        kernel_size=(4, 4),
        stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Illustrations des résultats de nos différentes expériences

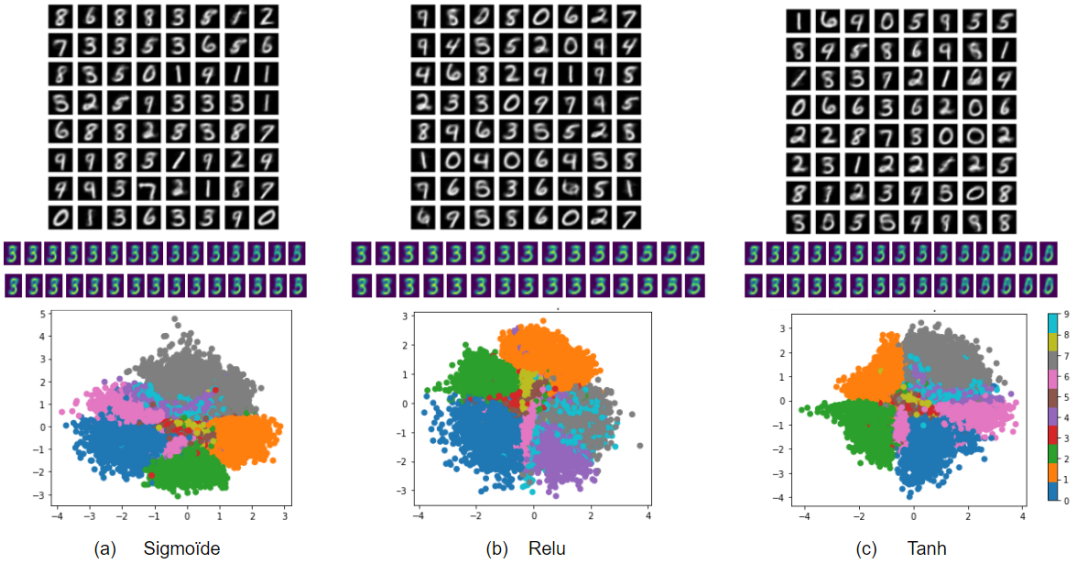


Figure 5: Visualization des résultats du VAE (espace latent de dimension 2) pour (a) l'activation Sigmoïde, (b) l'activation ReLU et (c) Tanh. La première ligne correspond à des images générées par le modèle, la deuxième ligne à une interpolation linéaire du chiffre 3 au chiffre 5, la 3ème ligne à une interpolation sphérique du chiffre 3 au chiffre 5 et la 4 ligne correspond à la projection de l'espace latent en deux dimensions avec 6400 points.

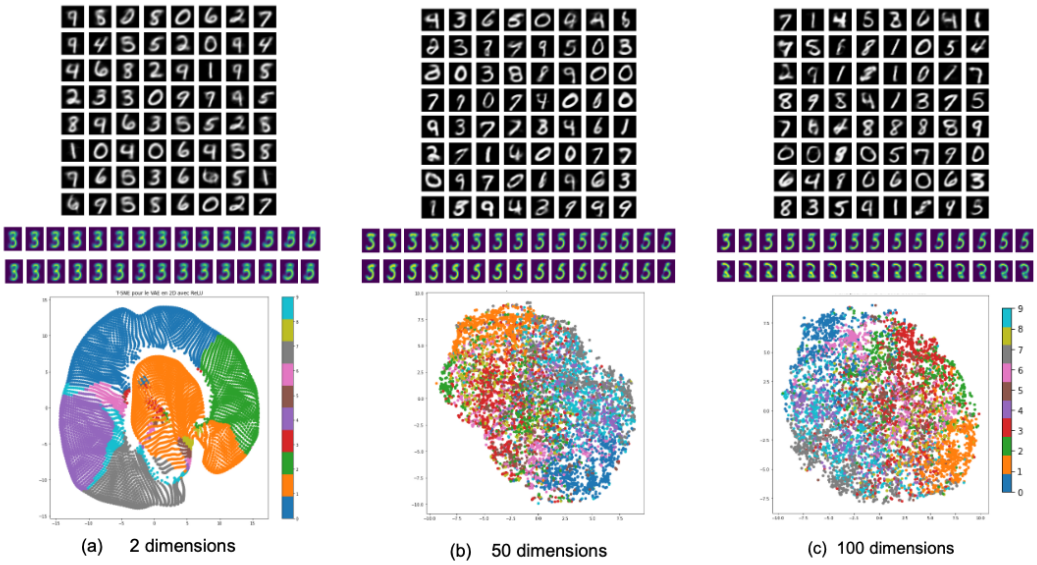
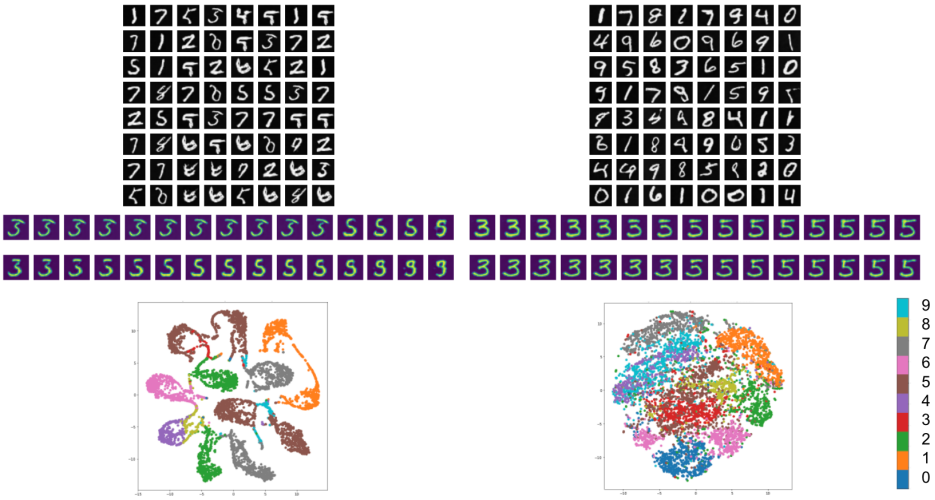
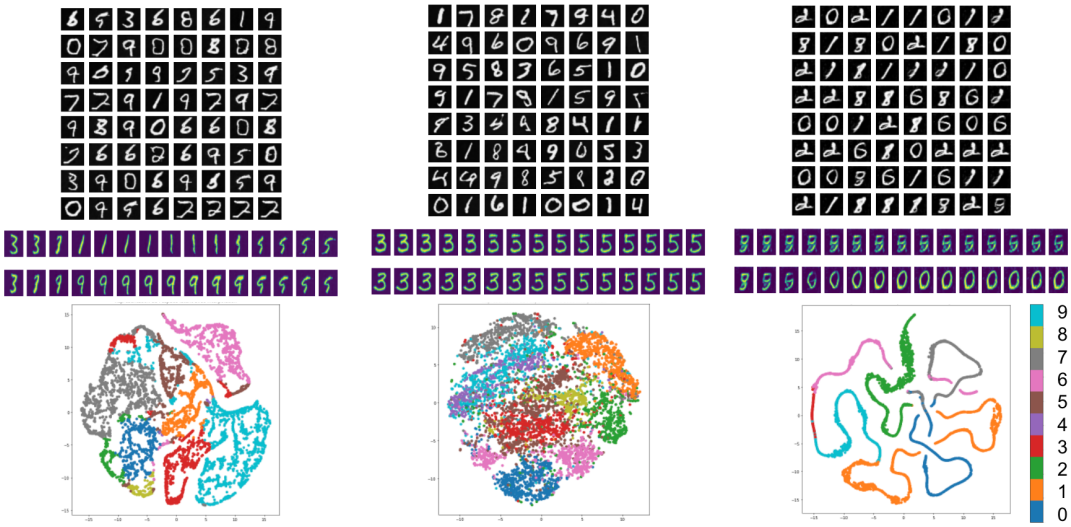


Figure 6: Visualisation des résultats du VAE pour une dimension de l'espace latent de (a) 2 et (b) 100, (c) 200. La première ligne correspond à des images générés par le modèle, la deuxième ligne à une interpolation linéaire du chiffre 3 au chiffre 5, la 3ème ligne à une interpolation sphérique du chiffre 3 au chiffre 5 et la 4ème ligne correspond à la projection t-SNE de l'espace latent sur 6400 points.



(a) Tanh (b) ReLU

Figure 7: Visualisation des résultats du DCGAN (espace latent de dimension 100) pour (a) l'activation Tanh et (b) l'activation ReLU. La première ligne correspond à des images générés par le modèle, la deuxième ligne à une interpolation linéaire du chiffre 3 au chiffre 5, la 3ème ligne à une interpolation sphérique du chiffre 3 au chiffre 5 et la 4 ligne correspond à la projection t-SNE de l'espace latent sur 6400 points.



(a) 2 dimensions (b) 100 dimensions (c) 200 dimensions

Figure 8: Visualisation des résultats du DCGAN pour une dimension de l'espace latent de (a) 2 et (b) 100, (c) 200. La première ligne correspond à des images générés par le modèle, la deuxième ligne à une interpolation linéaire du chiffre 3 au chiffre 5, la 3ème ligne à une interpolation sphérique du chiffre 3 au chiffre 5 et la 4 ligne correspond à la projection t-SNE de l'espace latent sur 6400 points.

Illustrations des résultats du modèle interactif

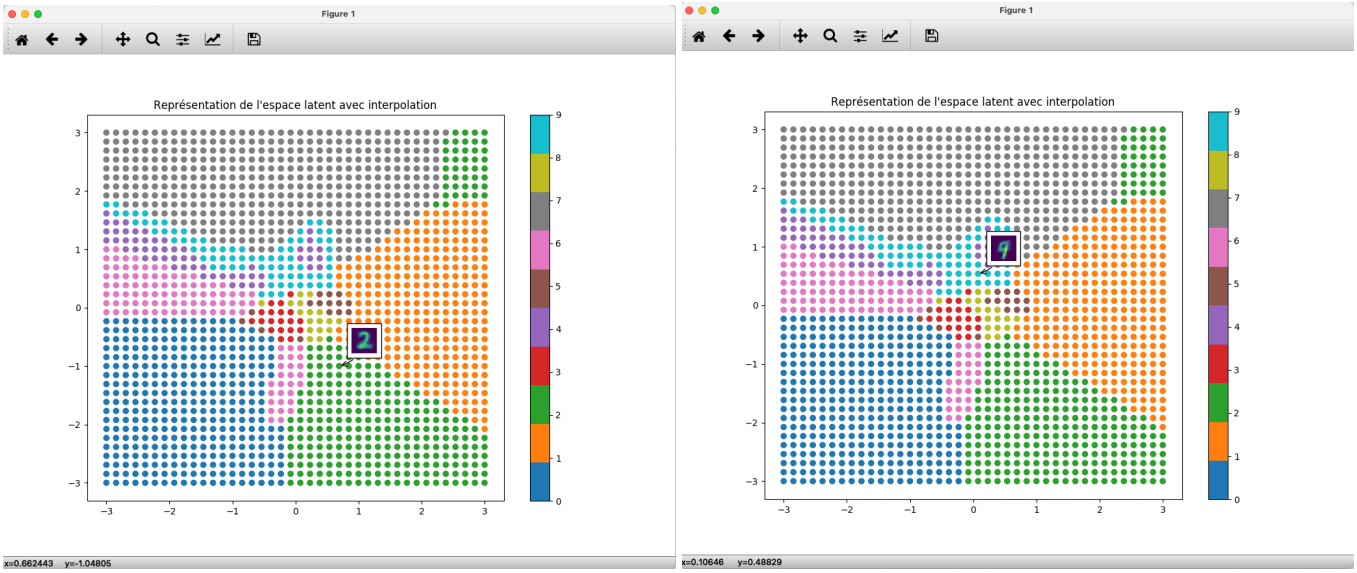


Figure 9: Représentation interactive de l'espace latent par un quadrillage de points (ici, 40 points entre -3 et 3). À chaque point est associé son image reconstruite par le décodeur du VAE.

Illustrations des résultats des opérations arithmétiques

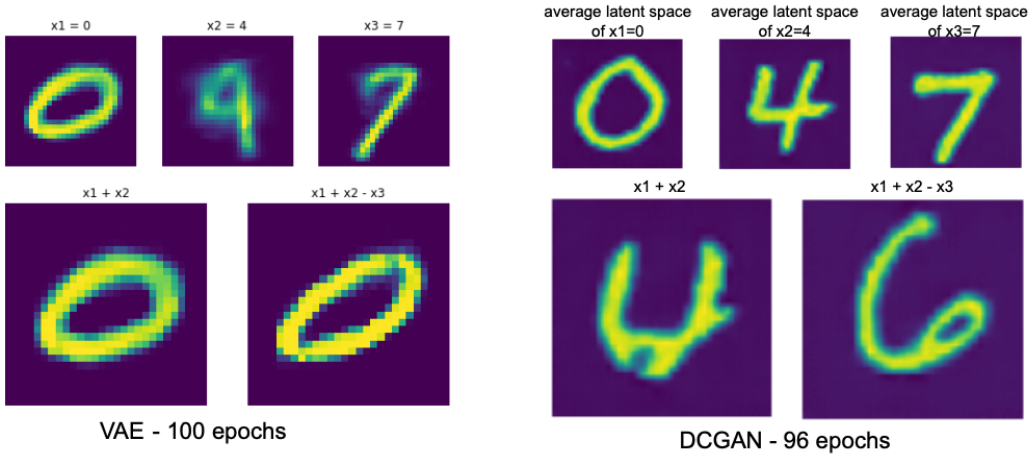


Figure 10: Résultats des opérations arithmétiques sur les espaces latents pour le VAE à gauche et le GAN à droite. Le chiffre en haut à gauche représente $x_1 = 0$, au milieu en haut représente $x_2 = 4$ et en haut à droite représente $x_3 = 7$. On peut ensuite observer des opérations sur les espaces latent de ces chiffres, tout d'abords en bas à gauche il y a le résultat pour $x_1 + x_2$ et en bas à droite il y a $x_1 + x_2 - x_3$.