Analyse exploratoire de modèle génératif d'image de chat

Nicolas Breault , Arnaud Dalbec-Constant , Thomas Ricard

Abstract

Ce rapport porte sur une analyse exploratoire de trois méthodes classiques de génération d'images: les Variational AutoEncoders (VAE), les Deep Convolutional Generative Adversial Networks (DCGAN) et les Denoising Diffusion Probalistic Model (DDPM). Les trois méthodes sont présentées brièvement à partir du cadre mathématique proposé dans leur article respectif et des modifications ou améliorations sont présentées pour certaines. Des résultats des trois méthodes sont ensuite présentés sur un ensemble de donnée d'images 64x64 pixel de figure de chat, puis une discussion est faite sur les différences de résultats entre les méthodes.

1 Introduction

Parmi les premiers modèles génératifs d'images, on retrouve les variational AutoEncoder (VAE). Plus précisément, l'article "Auto-Encoding Variational Bayes" publié en 2013 présente une architecture très près des autoencoder classiques, mais l'ajout de bruits dans l'espace latent permet une reconstruction d'image originale. Cependant, la qualité des images est souvent inférieure aux données d'entrainement. Peu de temps par la suite, les Generative Adversial Network (GAN) sont présentés et deviennent des modèles de premiers choix pour la génération d'images après leur publication en 2014. Avec leur architecture unique formant un générateur et discriminateur, ceux-ci offrent une reconstruction des nouvelles images en dehors des données d'entrainement, tout en conservant la qualité initiale des données. Au fils des années, les GAN ont pris le dessus en termes de popularité et plusieurs variantes sont apparues, dont les Deep Convolutional GAN (DCGAN). Seulement quelques années plus tard qu'une architecture rivalisant avec les GAN apparait: les Denoising Diffusion Probalistic Models (DDPM). En 2020, l'architecture du DDPM est publié, utilisant des processus de diffusion de bruit de façon consécutive ainsi que leur processus inverse pour créer des images. À ce jour, les PPDM donne des résultats supérieurs en termes de qualité par rapport aux GANs.

Du côté du grand public, les modèles génératifs profonds d'images sont de plus en plus populaires pour générer du contenu digital à faible coût. Cet intérêt est notamment expliqué avec les avancements et innovations des dernières années qui rendent la génération d'images de plus en plus réalistes, en plus d'avoir un grand degré de liberté. Avec des outils tels que DALL-E, tous peuvent obtenir une image de qualité tout en respectant un ensemble de contrainte donné. Dans le but d'explorer les techniques utilisées pour la génération d'images, ce projet se concentre sur quelques techniques classiques ainsi qu'évaluer leurs performances sur la génération de figure de chats. Le but principal est de faire une analyse exploratoire des trois méthodes mentionnées plus haut: VAE, DCGAN et DDPM. Leurs différences, forces, faiblesses et hyperparamètres seront discutés et comparés dans la suite de ce rapport.

2 Methodologie

2.1 Variational autoencoder (VAE)

Le modèle VAE est basé sur l'article 'Auto-Encoding Variational Bayes' [Kingma and Welling2022], dans lequel les auteurs proposent une nouvelle approche d'autoencodeur, où l'espace latent présente des caractéristiques reflétant la distribution des données d'entraînements. Le VAE a été construit à partir d'un exemple adapté aux données MNIST trouvé sur Internet. L'architecture de celui-ci a été modifiée en ajoutant plusieurs couches convolutives et en adaptant les entrées et sorties des encodeurs et décodeurs. Les données sont séparées en batchs qui sont mélangées après chaque epoch pour favoriser la généralisation de l'apprentissage. De plus, la fonction de perte est composée d'un terme sur l'encodeur (divergence de Kullback-Leibler) et d'un terme sur le decodeur (log-vraisemblance):

$$L_i(\theta, \phi) = -[E_{z\sim q_\theta(z|x_i)}[\log(p_\phi(x|z)] + \beta * KL(q_\theta(z|x_i)||p(z))] \tag{1}$$

Où $q_\theta(z|x_i)$ représente la partie encodeur et $(p_\phi(x|z)$ représente la partie décodeur. Une pondération β est utilisée pour mettre ces deux pertes à la même échelle. Celle-ci commence avec une valeur nulle pour les 10 première epoch et augmente à une valeur fixe pour le reste de l'entraînement. Plusieurs hyperparamètres vont être testés pour améliorer les performances, telles que β et la taille de la batch.

2.2 Deep convolutional generative adversial network (DCGAN)

Les modèles GAN sont présentés dans l'article Generative Adversarial Nets [Goodfellow *et al.*2014] et utilisent en fait deux réseaux différents, un premier pour générer des images à partir d'un échantillon z pris d'une distribution connu $p_z(z)$, puis un deuxième qui prend en entrée une image et donne en sortie une probabilité entre 0 et 1 que l'image soit réelle. Le but de ces deux réseaux est donc de s'entraîner en parallèle et d'être en compétition. En effet, le premier tente d'apprendre à générer des images de plus en plus réalistes tandis que le deuxième apprend à discriminer de mieux en mieux les fausses images des vraies. Pour apprendre l'un de l'autre, les auteurs proposent une adversial loss qui permet de prendre les sorties des deux réseaux en jeux tel que :

$$L_D = -\left(E_{x\sim P_{data}}[\log(D(x))] + E_{z\sim P_z}[\log(1 - D(G(z)))]\right) \tag{2}$$

On voit donc que le modèle utilise une première loss sur la réponse du discriminateur à de vraies images et une deuxième loss sur la réponse à des images provenant du générateur. Plus précisément, durant l'entraînement, on va mettre à jour le discriminateur avec seulement la partie gauche de la loss (avec de vraies images) puis ensuite le mettre à jour avec la partie droite (fausses images). En théorie, le modèle devrait converger vers un générateur qui génère des images provenant de la distribution des données d'entraînements (pas de manière parfaite puisqu'on optimise pour θ_g et non p_g) de manière aléatoire grâce aux vecteurs d'entrées aléatoires, puis un modèle discriminateur qui n'est plus capable de différencier les vraies images des fausses et donne alors en sortie une probabilité constante de 50%. Dans le cadre du projet, on utilise une variante du GAN de base qui est le Deep convolutional GAN qui, tout simplement, utilise des réseaux convolutifs pour le générateur et le discriminateur les rendant plus puissants. Les modèles GAN sont cependant très sensibles aux hyperparamètres puisque si l'un des deux réseaux est plus puissant ou apprend plus rapidement qu'un autre (souvent le discriminateur), alors le deuxième n'est plus en mesure d'apprendre. Il faut donc calibrer soigneusement les hyperparamètres pour que les deux apprennent conjointement. Pour se faire, quelques modifications et tests ont été faits dans ce projet. Le code de base est basé sur celui présenté sur le site de Pytorch : https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html. Premièrement, une recherche a été faite sur les hyperparamètres de bases comme le batch size, la latent space du vecteur d'entrée du générateur et les dimensions des maps de features pour les réseaux. Ensuite, l'ajout de la spectral normalization a été testé puisque c'est un ajout simple qui devrait, selon l'article Spectral Normalization For Generative Adversial Networks [Miyato *et al.*2018], améliorer les résultats en régularisant les poids sur le discriminateur. Finalement, une learning rate schedule a été implémenté pour tenter de réguler le discriminateur qui devenait souvent meilleur que le générateur après un certain nombre d'epochs.

2.3 Modèle de diffusion

Les modèles DDPM sont présentés dans l'article "Denoising Diffusion Probabilistic Models" [Ho *et al.*2020]. L'idée générale est d'utiliser un processus stochastique gaussien pour T steps, bruité l'image jusqu'au bruit blanc, et ensuite reconstruire celle-ci à partir du processus inverse. De façon plus théorique, soit un point échantillonné d'une distribution de données réelles $x_0 \sim q(x)$, on ajoute de façon récursive un petit bruit gaussien à l'échantillon afin d'obtenir l'ensemble d'échantillons bruités $x_1, ..., x_T$. La variance de chaque gaussienne β_t étant contrôlée en fonction des steps, on a que le forward process est décrit par:

$$q(x_t|x_{t-1}) = N(x_t; \sqrt{1-\beta_T}x_{t-1}, \beta_t I) \tag{3}$$

$$q(x_{1:T}|x_0) = \Pi_{t=1}^{T} q(x_t|x_{t-1}) \tag{4}$$

Une fois que le forward process ou diffusion process a été appliqué, on inverse le processus. S'il était possible d'échantillonner à partir de $q(x_{t-1}|x_t)$, cela serait possible. L'expression $q(x_{t-1}|x_t)$ n'était pas facile à obtenir, on apprend un modèle p_θ qui approximera l'expression. On a alors:

$$p_\theta(x_{t-1}|x_t) = N(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{5}$$

$$p_\theta(x_{0:T}) = p(x_T)\Pi_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{6}$$

L'entraînement est par la suite effectué en optimisation la borne variationnelle du negative log likelihood:

$$L_{VLB} = E_{q(x_{0:T})}\left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\right] \geq -E_{q(x_0)} \log p_\theta(x_0) \tag{7}$$

Il est important de noter que les β_t peuvent être appris par reparamétrisation ou être des hyperparamètres. En développant et simplifiant l'expression de L_{VLB} basé sur la divergence KL, les auteurs présentent une expression plus simple et efficace pour l'entraînement:

$$L_{simple}(\theta) = L_t^{simple} + C \tag{8}$$

$$L_t^{simple} = E_{x_0,t,\epsilon_t}\left[||\epsilon_t - \epsilon_\theta(\sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon_t, t||^2\right] \tag{9}$$

C est une constante indépendante de θ, $\alpha_t = \Pi_{i=1}^{t}(1-\beta_i)$, $\epsilon_t \sim N(0, I)$ et ϵ_θ essaie de prédire ϵ_t à partir de x_t. L'optimisation des paramètres θ se fait via la descente de gradient. Un résumé de l'algorithme est présenté à la figure suivante.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure 1: Résumé de l'algorithme DDPM

Dans l'article original, les auteurs avaient conçu un calendrier pour β_t grossissant linéairement entre 10^{-4} et 0.02.

Dans l'article "Improved Denoising Diffusion Probabilistic Models" publié en 2021, les auteurs présentent de nouvelles techniques, dont un calendrier en cosinus pour β_t afin d'améliorer les résultats. Voici les modifications:

$$\beta_t = \text{clip}(1 - \frac{\alpha_t}{\alpha_{t-1}}, 0.999)$$

$$\alpha_t = \frac{f(t)}{f(0)}, \text{ où } f(t) = \cos(\frac{t/T + s}{1+s}\frac{\pi}{2})^2$$

Pour ce qui est de l'implémentation d'un modèle de diffusion, nous avons utilisé la librairie denoising-diffusion-pytorch (Github: https://github.com/lucidrains/denoising-diffusion-pytorch?tab=readme-ov-file). L'algorithme présenté a été implémenté ainsi que les principales modifications discutées dans [Nichol and P.Dhariwal2021]. L'architecture pour le modèle appris est un U-net et si le paramètre flash_attn = True, le modèle utilise également un transformer pour l'attention (flash attention).

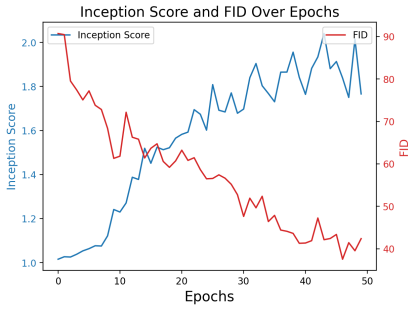
3 Résultats

Les modèles seront analysés en utilisant le métrique d'inception score (IS) qui permet de mesurer la qualité des images sans faire référence aux dataset utilisé et la métrique FID qui permet de mesurer la similarité des images simulées à celles du dataset. Cependant, ces deux métriques ne sont pas parfaites, IS utilise un modèle pré-entrainé sur une grande quantité d'image pour faire ces calculs et donc, si ce modèle n'a pas utilisé des données ressemblantes aux nôtres, ces résultats peuvent être mauvais. Pour le FID, plusieurs papiers ont montré que les résultats peuvent être très différents en fonction du réseau utilisé, de la normalisation ou de la résolution des images. Ce ne sont donc pas des méthodes parfaites, c'est pourquoi ils seront utilisés pour comparer les différents hyperparamètres, mais pas les méthodes entre elles. Pour les comparer, nous allons simplement regarder les images et donner notre avis.

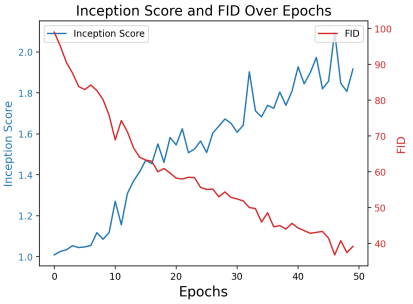
3.1 Variational autoencoder (VAE)

Initialement, plusieurs valeurs de β ont été testé soit 2, 5, 10 et 15 avec un batchsize de 128 sur 50 epochs avec un taux d'apprentissage de 0.001. Le score d'inception est très similaire sur tous les tests avec des score finaux variants entre 90 et 100. Cependant, pour le score FID, les modèles avec un plus petit β performent mieux. Ainsi, les tests sur le batch size se feront avec des valeurs de β de 2 et 5 pour des batchs de 32, 64, 128 et 256 images. La figure 2 montre qu'une taille plus grande de batch permet d'augmenter la performance.

Cependant, cette tendance ne s'étend pas à une batch de 256 images et le meilleur modèle testé est celui possédant une batch de 128 images avec un β = 2. Les images générées par ce modèle sont présentées à la figure 3.



(a) Métriques avec batch size de 64



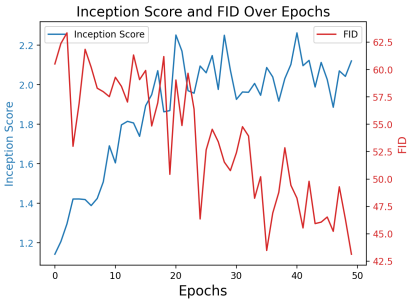
(b) Métriques avec batch size de 128

Figure 2: Résultats des modèles VAE avec β = 2 avec différentes batch size

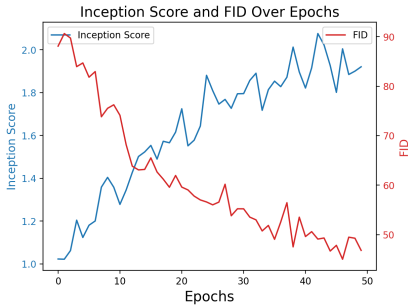
Figure 3: Images générées par le meilleur modèle VAE, avec β = 2, batchsize = 128 et dimension = 64

Finalement, en gardant ces paramètres fixes, des tests ont été fait en modifiant le nombre de dimensions de l'encodage avec 16,32,64 et 128 dimensions. Les résultats présentés à la figure 4 présentent l'impact de l'augmentation des dimensions sur les scores. Il est possible de remarquer que cette modification a principalement un impact sur l'IS, et agit négativement sur le FID. Le meilleur modèle est donc celui avec une dimension latente de 64, car il offre un bon compromis pour les métriques. Les résultats générés par ce modèle sont présentés

à la figure 3.



(a) Métriques pour une dimension d'encodage de 16

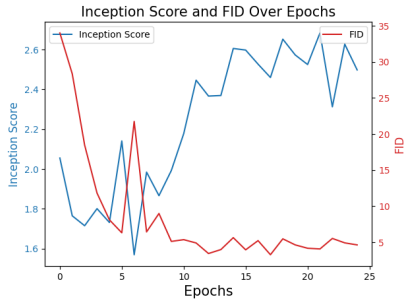


(b) Métriques pour une dimension d'encodage de 128

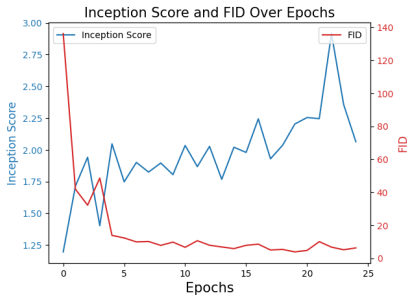
Figure 4: Résultats du modèle VAE avec des espaces latents de 16 et 128 dimensions

3.2 Deep convolutional generative adversial network (DCGAN)

Tout d'abord, en jouant avec le batch size on a pu remarquer que différentes combinaisons donnaient des modèles où les réseaux n'apprenaient rien dès le départ, on a donc choisit des batch size de 64 et 128 qui fonctionnait (voir figure 5).
On remarque donc qu'il n'y a pas de grandes différences et puisque ce sont des procédés aléatoires, en refaisant les expériences, les résultats sont aussi modifiés un peu. C'est pourquoi pour le reste des résultats, nous avons travaillé avec une batch size de 128 pour réduire le temps de calcul. Nous avons ensuite tenté d'augmenter les features map à 128 au lieu de 64 pour le générateur et le discriminateur. Cependant, en faisant cela, le discriminateur apprenait trop vite pour le générateur et le modèle s'était effondré.
Pour régler le problème, nous avons ajouté de la spectral normalisation dans sur les couches de convolutions du discriminateur pour régulariser les poids, ce qui a permis au modèle de s'entraîner. Nous avons alors testé avec des features map de taille 128 et 196.
On voit à la figure 7 qu'avec une taille de 196, les valeurs de IS sont un peu plus faibles, mais elles semblent montrer une tendance vers l'amélioration, tandis que le modèle avec une taille de 128 semble arrêter de s'améliorer. Le score FID tant qu'à lui donne environ les mêmes valeurs. Visuellement (Annexe figure 11), les deux modèles génèrent encore beaucoup d'images avec des artefacts, un problème qui devrait se réduire en augmentant le nombre d'epoch. On test ensuite



(a) Métriques avec batch size de 64



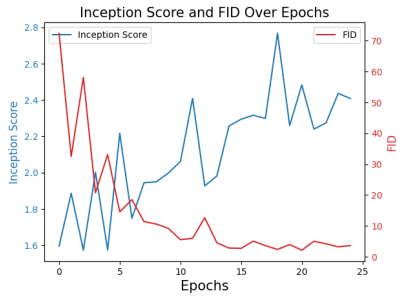
(b) Métriques avec batch size de 128

Figure 5: Résultats des modèles avec différentes batch size pour le DCGAN

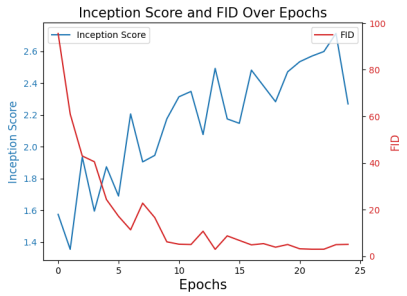
l'augmentation de la taille du vecteur d'entrée du générateur en passant de 100 à 150 à la figure 6.
On voit qu'on obtient environ les mêmes valeurs de métriques IS et FID que le modèle avec une longueur d'entrée de 100, cependant les images sont nettement mieux avec beaucoup moins d'artefacts et de bruit. On a donc finalement augmenté le nombre d'epoch pour notre dernier modèle. Cependant, on a vu rapidement que le discriminateur était toujours un peu meilleur que le générateur, ce qui rendait l'entraînement peu fructueux. Pour améliorer cela, nous avons ajouté une learning rate schedule pour diminuer le learning rate du discriminateur à chaque bond de 25 epochs tout en ayant un cycle durant les 25 epochs ou le learning rate augmente et diminue autour d'une valeur centrale. Cela a permis d'avoir un entraînement en continu et qui ne stagnait pas autant.
La figure 8 permet de voir qu'après environ 50 epochs, le modèle semble stagner si l'on se fie aux métriques. Cependant, on a remarqué que, visuellement, le modèle continuait à s'améliorer un peu, c'est pourquoi nous avons poussé à 100 epochs. On obtient finalement un modèle qui génère des images très réalistes avec très peu de bruit, mais il reste encore plusieurs images qui ne font aucun sens.

3.3 Modèle de diffusion

On a remarqué que les modèles diffusions prennent énormément de temps pour l'entraînement. En comparaison aux VAE et aux GAN, la génération d'échantillons est lente. Malgré le fait que plusieurs ajustements ont été implémentés dans le but d'accélérer le processus dans la librairie utilisée, le temps n'est pas comparable aux 2 autres modèles étudiés. En conséquence, dans le cas du modèle de diffusion, nous



(a) Métriques avec features map de taille 128



(b) Métriques avec features map de taille 196

Figure 6: Résultats des modèles avec différentes tailles de features map pour le DCGAn

n'avons pas assez de temps (ni de ressources sur Colab) pour faire une exploration des hyperparamètres.

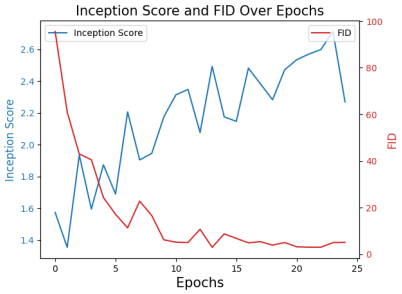
Les modèles essayés étaient testés sur Colab afin d'avoir des entraînements plus rapides. Cependant, il a eu beaucoup de difficultés techniques entre la librairie, Colab et l'importation des données. Plusieurs architectures ont été essayées, mais il arrivait régulièrement qu'on ait à arrêter l'entraînement , car le temps d'entraînement était irréaliste (dizaines d'heures). En ce sens, un seul modèle a été effectué, et celui-ci a pris plus de 10h d'entraînement. Le U-net utilise les paramètres suivants: dim = 32, dim_mults = (1, 2,4, 8), flash_attn = True. Pour la diffusion, les paramètres sont le U-net, image_size = 64, sampling = 1000, batch_size = 32, learning rate = 8e-5. Le modèle a été entraîné avec 30 000 itérations. Il est important de mentionner que cela est inférieur aux valeurs présentes de base (70 000), mais par souci de temps et de ressources, il n'avait pas d'autres options. À la figure 9 on peut observer les résultats de génération de figure de chats pour le DDPM pour deux nombres différents d'itérations. Du côté des métriques, aucune n'a été calculée pour ce modèle. L'implémentation des métriques n'avait pas été effectuée au moment de l'entraînement et ultérieurement, certains bugs avec la librairie nous ont empêché de récupérer les anciens modèles sauvegardés.

4 Discussion

4.1 Jeu de données

Initialement, le premier jeu de données utilisé contenait des images de chat dans divers contextes et centrés de manière aléatoire. Pour des modèles plus simples de génération

(a) Image généré



(b) Métriques

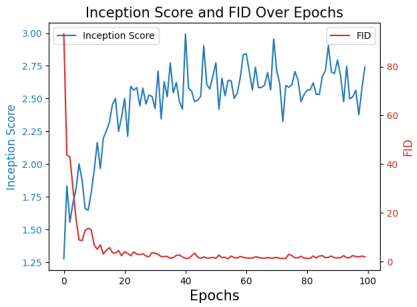
Figure 7: Résultats du modèle avec un vecteur d'entrer de longueur 150 pour le DCGAN

d'image, cela causait des incohérences au niveau de la reconstruction, car les caractéristiques du chat n'étaient pas toujours mise de l'avant. Ainsi, un nouveau jeu de données de 30 000 images de visage de chat a été utilisé pour faciliter le travail. Les caractéristiques des chats sont beaucoup plus présentes, mais le jeu de données possède encore des images où les chats sont moins mis de l'avant tel que présenté dans la figure 10:

4.2 Modèles

Puisque les métriques utilisées ne permettent pas de comparer les modèles entre eux, ceux-ci devront être comparés par la qualité des images générées à l'oeil. Premièrement, pour le VAE, on remarque avec la figure 3 que le chat de type blanc et noir semble se répéter. Également, les images sont floues, et elles semblent être le résultat d'une moyenne. Pour ce qui est du GAN à la figure 8, on voit une amélioration de qualité par rapport au VAE. Cependant, quelques images ne semblent pas être un chat, mais plus tôt une image difficile à cerner. Certaines couleurs de chats se répètent beaucoup, et en regardant plus précisemment, certains détails sont flous. Finalement du côté du DDPM à la figure 9. Pour les "beaux" chats, le DDPM surpasse les deux autres méthodes dans le cas des images détaillées. Pour le cas de DDPM, les meilleures images n'ont pas d'artefact. Cependant, on remarque une fréquence plus nombreuse d'images difficiles à cerner. En ce sens, en termes de qualité, le DDPM sem-

(a) Images générées du modèle final sur 100 epochs



(b) Métriques du modèle final sur 100 epochs

Figure 8: Résultats du modèle final pour le DCGAN

blent meilleures, mais sa fréquence d'échec est également plus élevé que les deux autres méthodes. On en conclut alors qu'avec les modèles présentés, le GAN performe mieux en moyenne que le DDPM. Il est cependant important de noter que le DDPM est partiellement entraîné. En entraînant davantage, il serait attendu que les lacunes du DDPM soit améliorées, surpressant surement le GAN.
Comme retrouvé dans la littérature, on obtient que les VAE ont une vitesse rapide de génération, mais la technique ne produit pas des images de grandes qualités. Pour les GAN, leur vitesse de génération est, elle aussi, rapide et il y a la possibilité de créer des images de grandes qualités. Cependant, cette technique peut parfois être restreinte et terme de diversité de génération. Finalement, le DDPM a une génération d'images très lente, mais la qualité des images est égale ou supérieure aux GAN, ainsi qu'une grande diversité de production d'images.

5 Conclusion

Pour conclure, les métriques utilisées ont permis d'optimiser les architectures proposées afin de générer 64 images sur lesquelles une analyse qualitative a été faite. Cette analyse a permis de conclure qu'a l'échelle des modèles proposés, avec seulement 30 000 images, l'architecture performant le mieux est le DCGAN, suivi du modèle de diffusion avec l'autoencodeur variationnel en dernière place. Effectivement, la recherche d'hyperparamètres ainsi que la taille des modèles furent grandement limitées par les ressources de calcul ainsi

(a) Génération après 22 500 itérations

(b) Génération après 30 000 itérations

Figure 9: Résultats du DDPM pour la génération de 32 images de chats

que la quantité de données disponibles. Finalement, il est a noté que les résultats obtenus avec le VAE sont beaucoup plus uniformes, mais il s'agit d'une des propriétés du modèle (erreur de reconstruction tend vers une solution "chat moyen"), ce pourquoi les deux autres architectures peuvent sembler plus intéressantes avec des images de couleur et de formes variées.

(a) cat-10025 (b) cat-10026 (c) cat-10250 (d) cat-10436

Figure 10: Exemples d'images de chat non centrées sur le visage

6 Annexes

(a) Images générées avec features map de taille 128

(b) Images générées avec features map de taille 196

Figure 11: Résultats des modèles avec différentes tailles de features map pour le DCGAN

References

[Goodfellow *et al.*, 2014] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[Ho *et al.*, 2020] J. Ho, A.Jain, and P. Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020.

[Kingma and Welling, 2022] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[Miyato *et al.*, 2018] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018.

[Nichol and P.Dhariwal, 2021] A. Nichol and P.Dhariwal. Improved denoising diffusion probabilistic models. *arXiv preprint arXiv:2102.09672*, 2021.