Rapport de stage
Études des "Generative Adversarial Networks"

Lucas Goareguer
Aix-Marseille Université
Lucas.Goareguer@etu.univ-amu.fr

Superviseur : Laurent Perrinet
Institut de Neurosciences de la Timone
Laurent.Perrinet@univ-amu.fr

Résumé

Ce rapport a pour objectif de décrire le travail effectué durant le stage final du Master *Intelligence Artificielle et Apprentissage Automatique*. Ce stage de six mois s'est déroulé à l'Institut de Neurosciences de La Timone et a été encadré par Laurent Perrinet, chercheur CNRS à l'université d'Aix-Marseille. Le principal objectif du stage était la compréhension des *Generative Adversarial Networks* (GAN) appliquée à la génération d'images. Un apport majeur de ce travail a été le développement d'outils facilitant la création et le test de GAN, notamment une librairie consacrée à cette tâche ainsi qu'un outil d'exploration de l'espace paramétrique.

1 Motivation

1.1 Pourquoi les GANs ?

Les *réseaux génératifs adverses* connus en anglais sous le terme Generative Adversarial Networks (GAN) sont des modèles qui permettent, par un apprentissage non supervisé, de générer des données cohérentes par rapport à un jeu de données.

Une bonne analogie pour présenter les GANs est la suivante : imaginons un faussaire G qui a pour objectif de créer de faux billets utilisables et un policier D qui a pour objectif de distinguer les faux billets des vrais. Lorsque G verra le policier repérer ses contrefaçons, il va modifier et améliorer sa technique. D quant à lui en comparant les faux billets à des vrais finira par mieux les repérer. Nos deux protagonistes vont alors se livrer à un duel de compétences. Là où G gagnera, il fera perdre D et inversement. Nos deux compères se retrouvent alors dans ce que l'on appelle un jeu à somme nulle. Contrairement à la plupart des autres modèles génératifs qui résolvent des problèmes d'optimisation standard, on cherche avec les GANs à se rapprocher d'un équilibre de Nash. C'est à dire un point où G et D prévoient parfaitement les choix de l'adversaire et prennent les bonnes décisions en conséquence. C'est dans cet état d'équilibre que le modèle converge et que le faussaire génère des billets indiscernables (pour le policier) des vrais.

Ces modèles ont été décrits pour la première fois en 2014 dans l'article de Ian Goodfellow (Goodfellow et al., 2014), depuis ils ont connu un engouement grandissant et de nombreux chercheurs s'y sont intéressés, comme en témoigne cette page référençant de nombreux articles : The Gan Zoo [1].

L'usage des GANs comme un outil de compréhension de notre cerveau est tentant en neurosciences. Le pas a notamment été franchi concernant le parallèle entre imagination et modèles génératifs via les travaux de (Palazzo et al., 2017) qui tendent notamment à montrer un lien entre la façon dont les images sont générées dans notre cerveau et dans les GANs.

Cette technologie étant très intéressante à bien des égards, la tâche qui m'a été confiée lors de mon stage a été d'étudier les GANs et de développer des outils permettant de les utiliser, de les entraîner mais aussi de tester le rôle des hyperparamètres.

1. The Gan Zoo : https ://github.com/hindupuravinash/the-gan-zoo

1.2 Espace latent

Pour une fonction $f : \mathbb{R}^a \to \mathbb{R}^b$, on appelle espace latent la représention dans $\mathbb{R}^b$ des données de $\mathbb{R}^a$. Tous vecteurs de dimension b pourront donc être plongés dans un espace latent de dimension $\mathbb{R}^b$.

1.3 Generative Adversarial Networks (GAN)

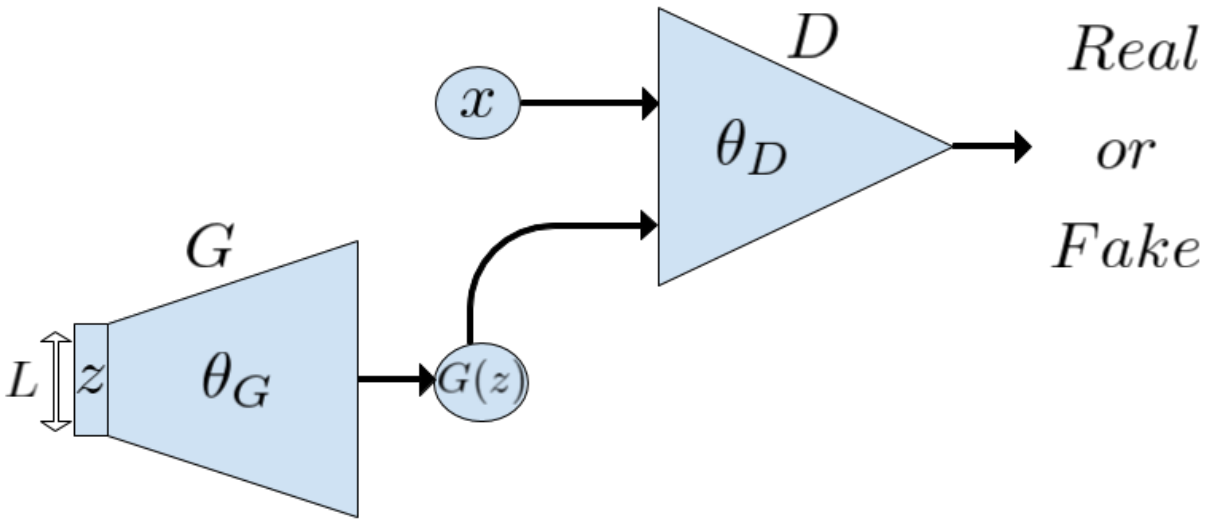


FIGURE 1 – Architecture d'un GAN.

Les *Generative Adversarial Networks* (GAN) sont des modèles qui permettent de générer des données approchant une loi de probabilité cible $P(x)$, où x est l'ensemble des données que l'on cherche à approcher. Ces modèles dans la version d'origine présentée par I. Goodfellow (Goodfellow et al., 2014) se présente sous la forme de deux réseaux distincts :

— Un générateur G qui prend en entrée un vecteur z dans un espace latent de dimension L et produit en sortie des données qui, après un entraînement réussi, doivent approcher la loi de probabilité $P(x, z)$. Ici l'espace de représentation des données est a l'entrée de G, il s'agit d'un cas spécifique aux GANs. Les poids de G sont notés θ_G (c.f. figure 1) ;
— Un discriminateur D qui a pour objectif de calculer la probabilité de y sachant x, avec y le label et x les données. Dans notre cas, le label est soit 0 pour des données générées, soit 1 pour des données appartenant a x. Son objectif est donc de déterminer si x appartient au jeu de données ou non. Les poids de D sont notés θ_D (c.f. figure 1) .

Ces deux réseaux vont être entraînés en parallèle par rétro-propagation du gradient en minimisant les fonctions de coûts suivantes, pour le discriminateur :

$$\mathcal{L}_D = -\log(D(x)) - \log(1 - D(G(z)))$$

et le générateur :

$$\mathcal{L}_G = -\log(D(G(z)))$$

Ces deux fonctions de coût se rapportent dans notre cas à la fonction de *Binary Cross Entropy* qui mesure la distance (en bits) à l'optimum de chaque composant tel que :

$$BCE(a, b) = -[b * \log(a) + (1 - b) * \log(1 - a)]$$

Avec a les réponses données par D pour des images et b les réponses attendues pour ces images.

Pour entraîner les deux réseaux, on suit l'algorithme suivant :

Algorithme 1 : Entraînement d'un GAN

Entrées : Nombre d'itération de l'entraînement I, la taille des batchs B, le jeu de données $data$
Sortie : G un générateur entraîné

Pour I étapes, faire :

$z \leftarrow$ ensemble de B vecteur de taille $L \sim \mathcal{N}(0, 1)$
$imagesGenerees \leftarrow G(z)$
$imagesReelles \leftarrow$ échantillon de B images de $data$
$Fake \leftarrow$ vecteur de taille B initialisé à 0
$Valid \leftarrow$ vecteur de taille B initialisé à 1

Entraînement de D
$d_x \leftarrow D(imagesReelles)$
$d_{g_z} \leftarrow D(imagesGenerees)$
$\mathcal{L}_D \leftarrow BCE(d_x, Valid) + BCE(d_{g_z}, Fake)$
Utilisation de $\mathcal{L}_D$ pour mettre à jour θ_D par descente de gradient stochastique

Entraînement de G
$\mathcal{L}_G \leftarrow BCE(d_{g_z}, Valid)$
Utilisation de $\mathcal{L}_G$ pour mettre à jour θ_G par descente de gradient stochastique

retourner G

L'algorithme utilisé pour effectuer les descentes de gradient lors de nos expériences est ADAM (Kingma and Ba, 2014). Des tests ont été réalisés avec une descente de gradient stochastique standard sans changement qualitatif notable dans l'entraînement. ADAM est tout de même plus rapide et il a donc été choisi par défaut.

1.4 Limites théoriques des GANs

Une part importante de nos recherches a consisté à essayer de comprendre les barrières théoriques qui entourent les GANs et leurs apprentissages. Les difficultés rencontrées lors de l'apprentissage d'un GAN sont importantes notamment à cause du grand nombre d'hyper-paramètres (voir section 2.1) ou encore de la difficulté qu'il y a de mesurer la qualité de l'apprentissage durant l'entraînent. En effet, pour les GANs standard aucune mesure ne permet directement de savoir si les images générées sont cohérentes au vu du jeu de données. L'objectif pratique des GANs n'est pas d'obtenir un générateur qui produise exactement les images du jeu de données, mais plutôt de produire des images à la fois diverses et proches des données. Cette notion de distance entre les images générées et celles du jeu de données est difficile à mesurer. C'est pourquoi la plupart du temps, il est nécessaire d'observer les images générées pour juger de la qualité de l'entraînement.

Nous nous sommes également penchés sur le problème dit de *mode collapse*, qui fait référence à un apprentissage lors duquel le générateur se met à produire des images peu diversifiées et souvent incohérentes, compte tenu du jeu de données (c.f. figure 8). Ce problème a notamment été étudié dans la section 3.2 de (Salimans et al., 2016) et dans la section 2.4 de (Metz et al., 2016).

La difficulté qui existe pour atteindre un point de convergence, dû au fait que l'équilibre de Nash entre les deux réseaux n'est pas garanti, fait également partie des problèmes importants rencontrés lors de l'entraînement d'un GAN. Ces questions seront discutées dans les sections 2.1, 3.2, 3.3.

1.5 Adversarial Auto-Encoder (AAE)

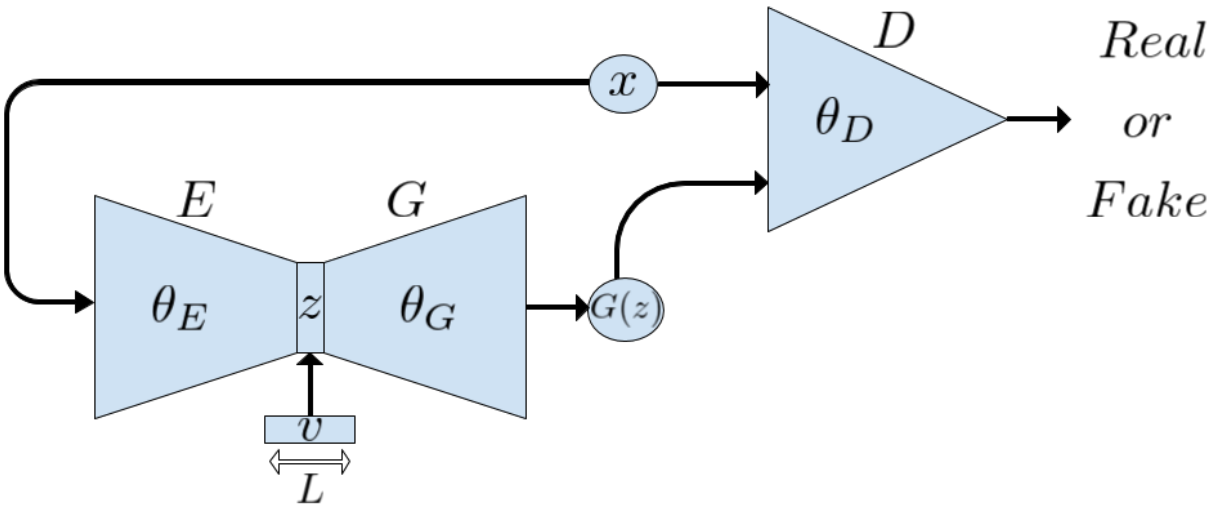


FIGURE 2 – Architecture d'un AAE. Le vecteur v représente une entrée directe de G sans passer par E.

Pour les expériences sur l'étude des espaces latents que nous allons détailler par la suite (c.f. section 2.5) nous avons choisi d'utiliser une architecture proche de celle présentée dans l'article de (Allen and Li,). Cette architecture nous donne un contrôle sur l'espace latent. On trouve dans la littérature différentes formes d'auto-encodeurs adversaires, comme par exemple la version de (Makhzani et al., 2015).

Dans le modèle que nous avons choisi, on trouve dans un premier temps un auto-encodeur (AE), qui est un réseau composer de deux parties symétriques avec en entrée un encodeur E qui réduit la dimension des données jusqu'à une taille L et en sortie un décodeur G qui augmente la taille de ce vecteur de taille L jusqu'à la taille d'origine des données.

Nous avons donc dans notre réseau (c.f. figure 1) :

— un AE comme présenté ci-dessus, avec un encodeur E (dont les poids seront notés θ_E), un décodeur G qui sert également de générateur (dont les poids seront notés θ_G). À la sortie de E et a l'entrée de G, on trouve un espace latent de taille L ;

— On associe ce réseau à un discriminateur D chargé de déterminer si les images qui lui sont présentées viennent du décodeur ou du jeu de données. Les poids de D sont notés θ_D

Les fonctions de coûts utilisées pour entraîner ce réseau sont :

$$\mathcal{L}_D = -\log(D(x)) - \log(1 - D(G(z)))$$

$$\mathcal{L}_G = -\log(D(G(z)))$$

$$\mathcal{L}_{EG} = MSE(x, G(E(x))) + MSE(z_{imgs}, z_{zeros}) + MSE(z^2_{imgs}, z^2_{ones})^{0.5}$$

Avec MSE définie comme l'erreur quadratique moyenne. Le $\mathcal{L}_{EG}$ sera détaillé dans la sous-section suivante 1.6.

L'intérêt d'un tel dispositif est qu'il permet d'avoir un accès direct à l'espace latent. Ainsi, on pourra par la suite, mener des expériences sur la façon dont est construit l'espace latent.

On suit un algorithme d'entraînement légèrement diffèrent que pour les GANs standard que voici :

Algorithme 2 : Entraînement d'un AAE

Entrées : Nombre d'itération de l'entraînement I, la taille des batchs B, le jeu de données $data$
Sortie G : un générateur entraîné

Pour I étapes, faire :

$z \leftarrow$ ensemble de B vecteur de taille $L \sim \mathcal{N}(0, 1)$
$imagesGenerees \leftarrow G(z)$
$imagesReelles \leftarrow$ échantillon aléatoire de B images de $data$
$Fake \leftarrow$ vecteur de taille B initialisé à 0
$Valid \leftarrow$ vecteur de taille B initialisé à 1

Entraînement de EG
$z_{zeros} \leftarrow$ ensemble de B vecteur de taille L initialisé à 0
$z_{ones} \leftarrow$ ensemble de B vecteur de taille L initialisé à 1
$z_{imgs} \leftarrow E(imagesReelles)$
$decoded_{imgs} \leftarrow G(z_{imgs})$
$\mathcal{L}_{EG} \leftarrow MSE(imagesReelles, decoded_{imgs}) + MSE(z_{imgs}, z_{zeros}) + MSE(z^2_{imgs}, z^2_{ones})^{0.5}$
Utilisation de $\mathcal{L}_{EG}$ pour mettre à jour θ_E et θ_G par descente de gradient stochastique

Entraînement de D
$d_x \leftarrow D(imagesReelles)$
$d_{g_z} \leftarrow D(imagesGenerees)$
$\mathcal{L}_D \leftarrow BCE(d_x, Valid) + BCE(d_{g_z}, Fake)$
Utilisation de $\mathcal{L}_D$ pour mettre à jour θ_D par descente de gradient stochastique

Entraînement de G
$\mathcal{L}_G \leftarrow BCE(d_{g_z}, Valid)$
Utilisation de $\mathcal{L}_G$ pour mettre à jour θ_G par descente de gradient stochastique

retourner G

1.6 Contraindre l'espace latent de l'auto-encodeur

Le $\mathcal{L}_{EG}$ s'inspire d'un article de Kingma (Kingma and Welling, 2013) dans lequel est détaillé une méthode permettant notamment de contraindre une loi de probabilité quelconque à suivre une loi Gaussienne. Grâce à cet outil, on s'assure que l'espace latent z au centre de l'auto-encodeur suive une loi Gaussienne de moyenne nulle et de variance égale à 1. Ceci nous permettra par la suite d'utiliser le générateur (qui est le décodeur de l'AE) en lui donnant des vecteurs suivants une loi normale. Ainsi, l'espace latent de l'AE et celui du générateur correspondront. En créant cette correspondance entre les deux espaces latents, on pourra les étudier en détail, ce qui sera le sujet de l'une des expériences que nous présentons dans la section 2.5.

2 Méthodes

2.1 Exploration des hyper-paramètres

L'ensemble des paramètres d'ajustement des algorithmes d'apprentissage et d'optimisation seront appelés hyper-paramètres. Il ne s'agit donc pas des poids du réseau que l'on optimise durant l'entraînement, mais des paramètres qui définiront la façon dont seront optimisés ces poids.
Pour bien comprendre le gigantisme de l'espace des paramètres : on peut vouloir tester le taux d'apprentissage (*learning rate*) du générateur pour des valeurs allant de 0.00005 à 0.005 avec un pas de 0.00005. Ce qui nous ferait une centaine de modèles à entraîner sans même compter tous les autres hyper-paramètres tels que le niveaux de normalisation par batch (*batchNormalization*), le nombre de

convolution ou encore leurs tailles.
Le nombre important d'hyper-paramètres présent dans les GANs nous a poussé a développer un outil permettant de passer en revue un nombre important de paramètres pour un GAN donné. Ainsi nous avons pu parcourir de manière efficace ces immenses espaces de paramètres. Nous avons par la suite très largement utilisé cet outil pour régler nos GANs durant toutes les expériences qui seront décrites plus bas.

2.2 Simpsons Dataset

Pour certaines de nos expériences, nous avons utilisé le jeu de données Simpsons Faces [2]. Ce jeu de données contient 9877 images de visage des personnages des Simpsons extraites des épisodes de manière automatique. Ce processus a introduit un certain nombre d'erreurs dans les données (aucun personnage sur l'image, des visages coupés, ect...), pour cette raison, j'ai effectué une passe manuelle pour retirer au maximum ces mauvaises images du jeu de données et nous avons finalement utilisé 9629 images pour nos expériences. Vous pouvez voir quelques exemples d'images de ce jeu de données sur la figure 3.
Après des tests avec des résolutions plus faibles, nous avons choisi d'utiliser ces images en taille 128x128 pixels pour nos expériences. Le procédé fonctionne peu importe la taille des images, mais 128x128 pixels a été un bon compromis entre résolution des images et temps de calcul.

FIGURE 3 – Exemples d'images du jeu de données Simpsons Faces utilisées pour certaines expériences.

2.3 Simpsons Generator

La première partie du stage a consisté en une exploration du monde des GANs. Nous nous sommes donc fixé comme objectif, la recherche, l'étude et l'entraînement d'un modèle capable de générer des visages de personnages des Simpson. Après avoir choisi notre jeu de données, nous avons exploré l'immense monde des GANs et nous avons rapidement dû faire des choix étant donné la quantité astronomique d'articles présentant des variantes de GAN (c.f. The Gan Zoo 1). Nous avons donc focalisé notre attention sur les systèmes les plus éprouvés afin de comprendre au mieux la théorie derrière ces modèles.

Nous avons particulièrement utilisé les DCGAN (Radford et al., 2015), qui sont notamment composés de couches de convolution très adaptées au traitement des images. Puis, dans un second temps, nous nous sommes penchés sur les *Adversarial Auto-Encoders* (AAE) qui nous ont servi par la suite pour l'expérience présentée en section 2.5.

2.4 Fractal Dream Dataset

Pour l'une des expériences que nous souhaitions mener (c.f. section 2.5), nous avions besoin d'un jeu de données paramétrique. En effet, il nous fallait un jeu de données dont chaque image puisse être associées à un point dans l'espace de manière cohérente. Pour ce faire nous avons décidé de construire un nouveau jeu de données à partir d'un outil mathématique : les attracteurs étranges (Pickover, 1995). Ces objets permettent notamment de générer des figures variées à partir d'un nombre donné de paramètres. Les Fractal Dream (Pickover, 1995) sont un type particulier d'attracteur étrange que nous avons choisi pour notre jeu de données. À partir de six paramètres, la formule disponible en annexe A permet de calculer de manière itérative (pour un nombre d'itérations données) un ensemble de points qui constitue la trajectoire de l'attracteur. Une fois ces points placés dans un quadrillage de taille finie (128x128 dans notre cas) on obtient une image comme celle visible dans la figure 4. Dans les images, les nuances de

2. Simpsons Faces : https ://www.kaggle.com/kostastokis/simpsons-faces

couleurs des pixels correspondent aux nombres de fois ou l'attracteur est passé dans la case. Le jeu de données est constitué de trois classes d'images correspondant aux trois palettes de couleurs choisies (bleue, orange et verte), pour nos expériences, nous n'avons utilisé qu'une seule de ces trois palettes pour faciliter l'entraînement.
J'ai mis à disposition sur le site Kaggle ce jeu de données construit pour l'occasion ainsi que les codes utilisés : Fractal Dream Dataset [3].

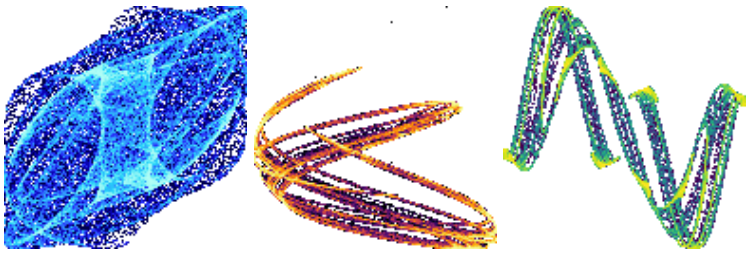


FIGURE 4 – Exemples d'images du jeu de données Fractal Dream Dataset utilisé pour certaines expériences. Vous pourrez trouver la liste des paramètres utilisés pour générer ces trois figures en annexes A.

2.5 Correspondances des espaces latents

Nous nous sommes posés de nombreuses questions concernant la façon dont est construit l'espace latent des GANs. Nous avons notamment voulu savoir si (étant donnés une base de données $data$ composée d'images x, chacune associée à un unique vecteur v, de dimension L, répartie de manière cohérente dans l'espace) un générateur G entraîné avec un AAE pourrait générer une image x' proche de celle d'une image x pour un vecteur v identique. Autrement dit : le générateur serait-il capable de reconstruire, par un entraînement adversaire, la fonction qui relie les images de $data$ au vecteur v qui leur est associé. Pour mettre en place cette expérience nous avons entraîné un AAE avec le jeu de données FDD présenté dans la section 2.4. Ensuite, nous avons pu comparer les images de FDD aux images générées par G pour un même vecteur v. La figure 5 décrit cette expérience.

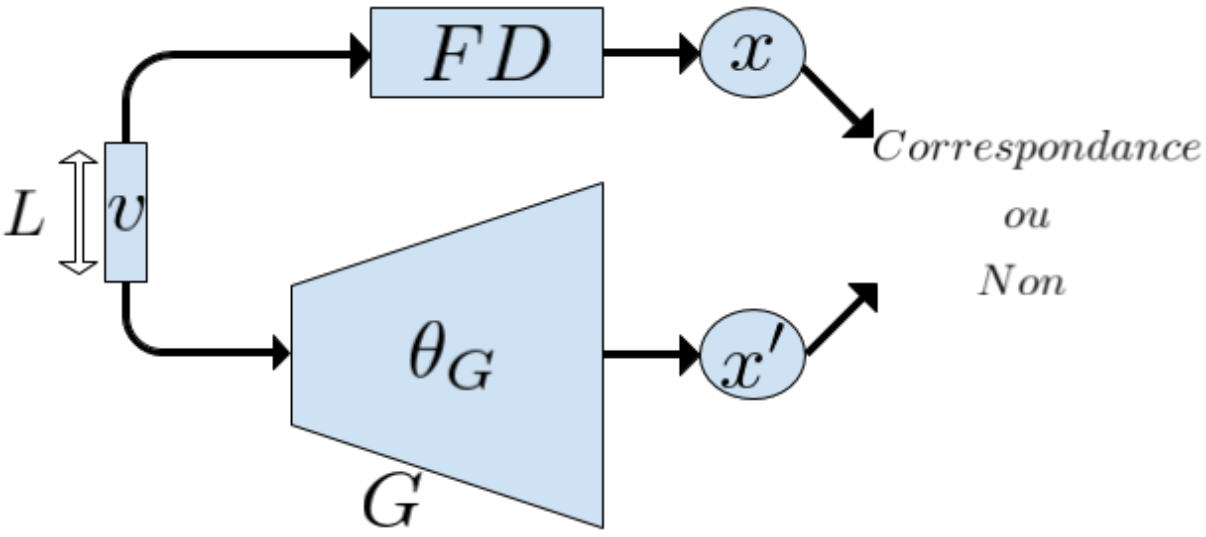


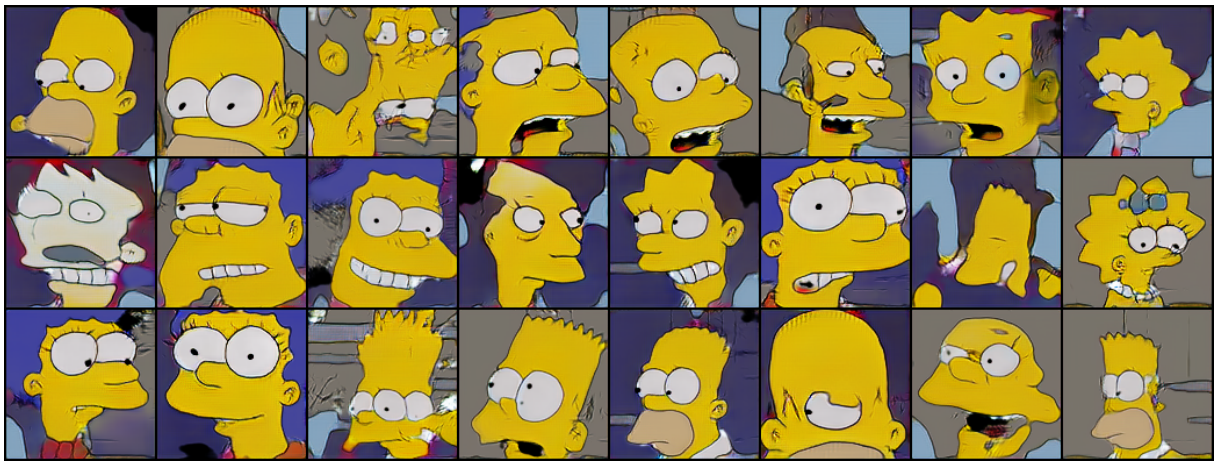
FIGURE 5 – Description de l'expérience de comparaison des espaces latents. Ici FD correspond à l'opération permettant de calculer une image de type Fractal Dream.

3. Fractal Dream Dataset : https ://www.kaggle.com/lgoareguer/fractal-dream-dataset

3 Résultats

3.1 Simpsons Cohérent

Le balayage des hyper-paramètres ainsi que l'affinage des modèles au grès de nos recherches ont permis d'obtenir de très bons résultats. Vous pouvez voir sur la figure 6a les résultats obtenus avec un DCGAN et sur la figure 6b les résultats obtenus avec un AAE. Les hyper-paramètres ainsi que l'architecture détaillée des deux réseaux sont disponibles en annexes B et C.
Il faut noter que les deux modèles étant diffèrents, ils disposent d'hyper-paramètres adaptés, mais nous avons cherché à leurs donner des capacités similaires (même nombre de couches pour G et D, même nombre de filtres par convolution, etc...). L'objectif était de pouvoir comparer les deux architectures sur le plan de la génération d'images et sur ce point les AAE montrent des capacités moindres que les DCGAN. On constate ici et dans chacune de nos expériences que les images obtenues avec l'AAE présente un aspect flou. Ceci vient certainement du fait que la fonction de coût de l'AE rentre en opposition (ou du moins n'est pas en totale adéquation) avec la fonction de coût du générateur qui est affecté par les deux.



(a) Images générées avec le générateur d'un DCGAN.

(b) Images générées avec le générateur d'un AAE.

FIGURE 6 – Images générées par deux générateurs entraînés différemment.

3.2 Mesure de la qualité des résultats

Malgré l'importante quantité de travaux sur le sujet, les GANs laissent encore de nombreuses questions en suspens. Durant nos expériences, nous nous sommes confrontés à de nombreuses barrières théoriques qui ne sont pas encore tombées.

L'une des principales faiblesses des GANs est l'absence de mesure de la qualité des résultats. En effet, même si dans la section 4.2 de l'article (Goodfellow et al., 2014), G est sensé converger vers un point où les données qu'il génère suive la loi de probabilités du jeu de données, dans la pratique aucun point de convergence n'est atteint entre G et D. Ceci est dû notamment au fait que G n'apprend que par le biais de D, il n'a donc pas pour objectif de se rapprocher des données du jeu de données, mais de générer des données qui trompent D. On ne peut donc pas ce fier au fonction de coût de G pour évaluer son niveau d'apprentissage. Certains articles se sont focalisés sur la recherche d'une métrique adaptée aux GANs comme dans (Salimans et al., 2016) qui utilise ce qu'ils appellent l'Inception score pour évaluer les images ou encore (Berthelot et al., 2017) qui introduit une nouvelle architecture de GAN ainsi qu'une métrique associée.

3.3 Équilibre de Nash et *mode collapse*

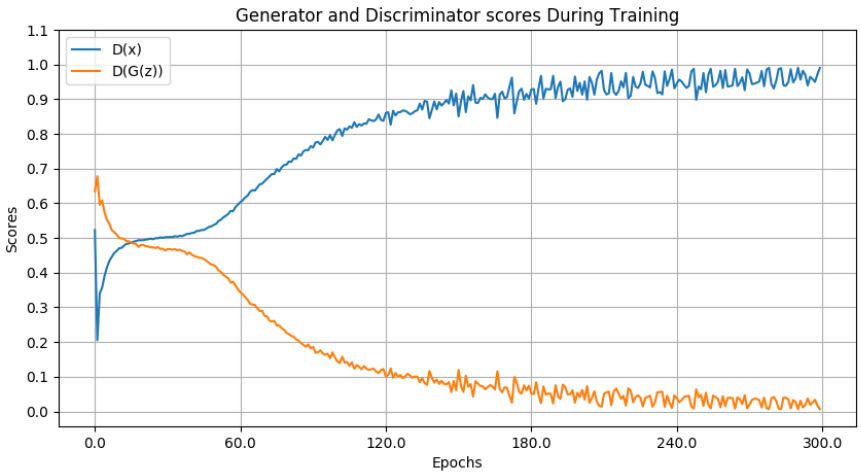
À ce problème de mesure s'ajoute un problème d'équilibre. En effet, comme dit précédemment, on peut voir l'apprentissage de notre réseau comme un jeu à somme nulle dans lequel ce que G gagne D le perd et inversement.

L'objectif de ce jeu est donc d'atteindre le point d'équilibre tel que G produit un résultat suivant la loi de probabilité $P(x)$. Dans ce cas D devient incapable de différencier les images générées par G des images du jeu de données. Ici, l'entraînement est terminé et G produit les images attendues. L'équilibre de Nash est atteint.

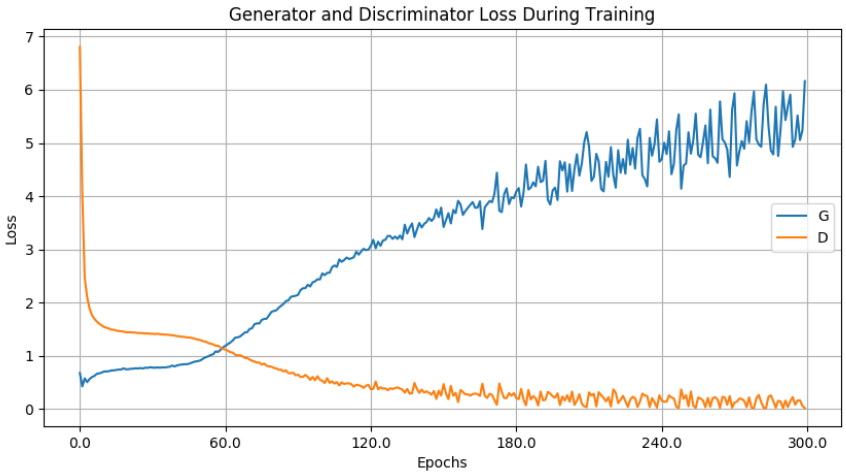
Malheureusement, ce que nos expériences nous ont apprises c'est que cet état d'équilibre, s'il existe, n'est jamais atteint (dans le cadre de nos expériences en tout cas). Ce problème est régulièrement rapporté dans la littérature des GANs et des cas où l'équilibre est atteint sont très rarement mentionnés.

Comme vous pouvez le voir sur la figure 7a la moyenne des réponses du discriminateur pour les images réelles et pour les images générées tend vers un point où D ne fait presque plus aucune erreur. Étant donné que G apprend via les réponses de D, on constate que la fonction de coût de G devient de plus en plus proche de l'infini ce qui à tendance à créer une forte instabilité dans l'apprentissage. Cette instabilité peut être observée sur la figure 7b où l'on constate que plus l'apprentissage progresse, plus la fonction de coût de G devient variable. En effet G, du fait de sa fonction de coût 1.3, a besoin que D réussisse à discriminer les images générées pour pouvoir se corriger et s'adapter. Mais plus D réussit à classer correctement les images, plus G obtient une fonction de coût importante ce qui le pousse au déséquilibre. C'est ce que l'on appelle dans le monde des GANs un *mode collapse*. Vous pouvez voir sur la figure 8 un exemple d'images générées lors du *mode collapse* d'un générateur. On constate que la diversité des images devient pauvre et totalement incohérente. Ce phénomène s'explique par le fait que D finit par réussir tellement bien à classer les images qu'il pousse G à se focaliser sur les quelques images qui réussissent encore à le tromper D. Ainsi, G se met a générer une seule (ou très peu) d'images différentes ce qui veut dire que l'espace latent de G devient très uniforme. De nombreux articles ont étudié le *mode collapse* comme par exemple (Goodfellow et al., 2014).

Fort heureusement, cette absence d'équilibre n'empêche pas l'entraînement du générateur. Il est vrai que D finit toujours (dans nos expériences) par prendre l'avantage et ne fait presque plus aucune erreur, mais G réussit encore, rarement, à flouer D. Le fait que G réussisse à gagner face à un générateur bien meilleur que lui luit permet de continuer son apprentissage. Les victoires de G, même si elles sont rares, permettent à D de s'améliorer — puis les victoires de D permettent à leurs tours à G de s'améliorer.



(a) Moyenne des réponses de D pour les images réelles en bleu et les images générées en orange, au cours de l'entraînement.



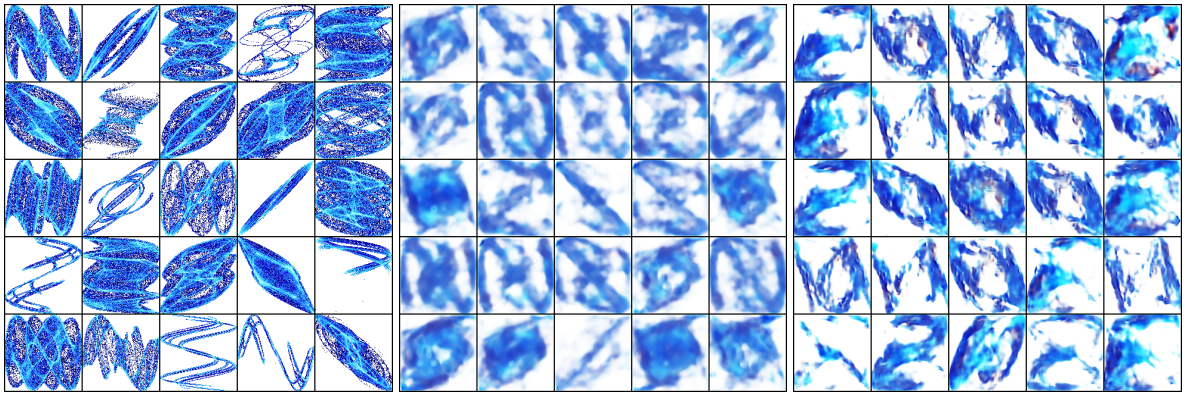
(b) Valeurs des fonctions de coût de G (en bleu) et D (en orange), au cours de l'entraînement.

FIGURE 7 – Courbes des coûts et des scores au cours de l'entraînement.

FIGURE 8 – Exemples d'images générées lors d'un *mode collapse* d'un générateur. On constate une faible diversité ainsi qu'une incohérence des images par rapport au jeu de données.

3.4 Comparaison des espaces latents connus et générés

Vous pouvez voir sur la gauche de la figure 9 des images extraites du jeu de données FDD. Après avoir entraîné et paramétré différents modèles à générer des images cohérentes par rapport à celles du jeu de données, nous avons pu générer les images visibles à droite et au centre de la figure 9. En comparant ces images avec celles de gauche, on ne constate aucune correspondance entre celles générer par G et celles construite en utilisant les formules Fractal Dream 2.4. Ce résultat tend à montrer que l'espace latent de G n'a pas développé de correspondance avec l'espace des paramètres de FDD durant sont entraînement.



(a) Des éléments extraits du jeu de données *Fractal Dream Dataset*.

(b) Des images générées par un générateur entraîné par un AAE.

(c) Des images générées par un autre générateur entraîné par un AAE.

FIGURE 9 – Images générées en utilisant des vecteurs latents identiques.

4 Conclusions et perspectives

4.1 Apprentissage des GANs

L'apprentissage des GANs est une tâche ardue et nous avons eu besoin de beaucoup de tests et de paramétrisation avant d'arriver à un résultat un tant soit peu convaincant. Le grand nombre d'hyper-paramètres ainsi que les vides théoriques présentés section 1.4 posent de nombreuses difficultés. Néanmoins, les résultats obtenus avec les Simpsons, notamment sont finalement tout à fait convaincants.

4.2 Découverte et prise en mains de nombreux outils

Durant le stage, j'ai été amené à me former à de nombreuses technologies et outils. On peut citer parmi les plus importants Pytorch (Paszke et al., 2017) ou encore Tensorboard (Abadi et al., 2015), à ceci s'ajoute également un usage quotidien de Linux et de Python qui m'ont permis de découvrir de nombreux logiciels et bibliothèques. Ensembles, ces outils s'agrègent à ceux utilisés lors de mon Master et me permettront, je l'espère, de commencer ma vie professionnelle sur de bonnes bases.

4.3 Premiers pas dans le monde professionnel

Les laboratoires de recherche sont un monde particulier que j'avais déjà pu approcher lors de mes études, ce stage m'a permis de les découvrir en détails. J'ai pu apprendre tout ce qui fait le monde professionnel : horaires fixes, journées complètes, réunions de travail, etc... J'ai aussi pu apprendre à gérer l'avancement de mon projet avec une méthode agile centrée sur l'utilisation de GitHub pour l'écriture de rapports d'expérience (cahier de laboratoire électronique) qui suivait mes expériences à avancées et points à améliorer. Ces nouvelles expériences forment le début de mon parcours professionnel.

Remerciements

Je tiens à remercier Laurent Perrinet pour m'avoir encadré et guidé durant ce stage et ainsi que Victor Boutin pour ces précieux conseils lors de la rédaction de ce rapport.

References

[Abadi et al.2015] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow : Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Allen and Li] Avery Allen and Wenchen Li. Generative adversarial denoising autoencoder for face completion. *UNKNOW*.

[Berthelot et al.2017] David Berthelot, Thomas Schumm, and Luke Metz. 2017. Began : Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv :1703.10717*.

[Goodfellow et al.2014] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

[Kingma and Ba2014] Diederik P Kingma and Jimmy Ba. 2014. Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*.

[Kingma and Welling2013] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv :1312.6114*.

[Makhzani et al.2015] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. 2015. Adversarial autoencoders. *arXiv preprint arXiv :1511.05644*.

[Metz et al.2016] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. 2016. Unrolled generative adversarial networks. *CoRR*, abs/1611.02163.

[Palazzo et al.2017] S. Palazzo, C. Spampinato, I. Kavasidis, D. Giordano, and M. Shah. 2017. Generative adversarial networks conditioned by brain signals. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3430–3438, Oct.

[Paszke et al.2017] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. *UNKNOW*.

[Pickover1995] Clifford A. Pickover. 1995. *Chaos in Wonderland : Visual Adventures in a Fractal World*. St. Martin's Press, Inc., New York, NY, USA, 1st edition.

[Radford et al.2015] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv :1511.06434*.

[Salimans et al.2016] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242.

5 Annexes

A - Formule des Fractal Dream

La formule ci-dessous permet de calculer le point suivant d'un attracteur de type Fractal Dream à partir des coordonnées du point actuel et de quatre paramètres données. En utilisant cette formule successivement un grand nombre de fois on peut calculer l'image correspondant à l'attracteur pour des paramètres et un point de départ choisi.

Algorithme 3 : Calcul du point suivant d'un Fractal Dream

Entrées : (x, y) coordonnées du point courant, (a, b, c, d) paramètres de l'attracteur
Sortie : (x', y') les coordonnées du point suivant

$x' \leftarrow sin(y * b) + c * sin(x * b)$
$y' \leftarrow sin(x * a) + d * sin(y * a)$

return (x', y')

Pour l'exemple les paramètres utiliser pour générer les trois images de la figure 4 sont respectivement : 1 : $x = -0,00545687187361743$ $y = -0,5312304575586382$ $a = 1,8197494653032407$ $b = 2,422111729355857$ $c = 1,4944898296889655$ $d = 2,487490210145214$
2 : $x = -0,319459804135373$ $y = -0,0981842500805612$ $a = 0,42291442353441633$ $b = 1,7253282321258185$ $c = 1,4272325555244099$ $d = 1,3701336149016234$
3 : $x = -0,9333646250089$ $y = -0,3698044331245828$ $a = -1,9514116449042127$ $b = -0,376713271698188$ $c = -1,1759178511809651$ $d = -1,7157801385982319$

B - Détail des hyper-paramètres des réseaux

Les réseaux on était entraîner avec les hyper-paramètres suivants pour les expériences sur les Simpson :

- DCGAN :
  - LeakyReLU : factor=1e-6
  - BatchNorm2d : epsilon=0.5
  - BatchSize : 16
  - Learning rate D : 5e-5
  - Learning rate G : 2.5e-4
- AAE :
  - LeakyReLU : factor=0.1
  - BatchNorm2d : epsilon=0.01
  - BatchSize : 16
  - Learning rate D : 4e-5
  - Learning rate G : 4e-4
  - Learning rate E : 4e-4

C - Détail de l'architecture des réseaux

Vous trouverez ci-dessous un détail couche par couche de l'architecture des réseaux DCGAN et AAE ayant servi pour nos expériences. Les noms des fonctions utilisées sont ceux définis par la librairie Pytorch en version 1.1.0.

Input : (16x32) (BatchSize x Latent space size)
Linear : in :32 out :64x512 LeakyReLU
UpsamplingNearest2d : scale factor 2 Conv2d : filtres (512)→(256) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) BatchNorm2d LeakyReLU
UpsamplingNearest2d : scale factor 2 Conv2d : filtres (256)→(128) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) BatchNorm2d LeakyReLU
UpsamplingNearest2d : scale factor 2 Conv2d : filtres (128)→(64) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) BatchNorm2d LeakyReLU
Conv2d : filtres (64)→(3) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) Tanh
Output : (16x3x128x128) (BatchSize x channels x Image size x Image size)

TABLE 1 – Détail de l'architecture du générateur du DCGAN et de L'AAE.

Input : (16x3x128x128) (BatchSize x channels x Image size x Image size)
Conv2d : filtres (3)→(64) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) LeakyReLU
Conv2d : filtres (64)→(128) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) LeakyReLU BatchNorm2d
Conv2d : filtres (128)→(256) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) LeakyReLU BatchNorm2d
Conv2d : filtres (256)→(512) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) LeakyReLU BatchNorm2d
Linear : in :64x512 out :1
Output : (16x1) (BatchSize x D answer size)

TABLE 2 – Détail de l'architecture du discriminateur du DCGAN et de L'AAE.

Input : (16x3x128x128) (BatchSize x channels x Image size x Image size)
Conv2d : filtres (3)→(64) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) LeakyReLU
Conv2d : filtres (64)→(128) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) LeakyReLU BatchNorm2d
Conv2d : filtres (128)→(256) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) LeakyReLU BatchNorm2d
Conv2d : filtres (256)→(512) kernel size=(9, 9), stride=(2, 2), padding=(4, 4) LeakyReLU BatchNorm2d
Linear : in :64x512 out :32
Output : (16x32) (BatchSize x Latent space size)

TABLE 3 – Détail de l'architecture de l'encoder de L'AAE.