UNIVERSITE DE PARIS
UFR MATHEMATIQUES ET INFORMATIQUE

Génération de l'art moderne en utilisant les GAN's

Master 1 Vision Machine Intelligente

Billal IHADDADEN – Fatima KHANTAR

Année universitaire 2021 – 2022

Table des matières

1. Introduction

Dans ce projet nous allons nous intéresser à l'art moderne, un art né dans le XVIIIe siècle initiée par Edouard Manet, plus précisément nous allons nous intéresser au cubisme, ce type d'art propose une déconstruction conceptuelle du réel et démultiplie les points de vue sur l'objet.

Plusieurs tableaux connus qui décrivent ce type d'art :

Figure 2 Portrait de Georges Braque, 1909–1910

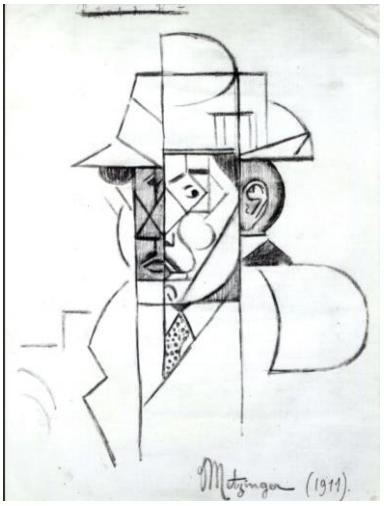


Figure 1 Portrait d'Albert Gleizes, 1911

Figure 4 Portrait de André Lhote, 1930

Figure 3 Portrait de Pablo Picasso, 1912

Nous allons essayer de reproduire ce type d'art en utilisant les gan's.

2. Generative Adversarial Network

Appelés GAN, le réseau antagoniste génératif est une approche d'apprentissage profond non supervisée utilisée pour la génération des données en se référents aux statistiques des données utilisées pour l'entrainement, en 2014 le premier article est publié par Lan Goodfellow [1] l'auteur de ce type d'architecture d'apprentissage profond.

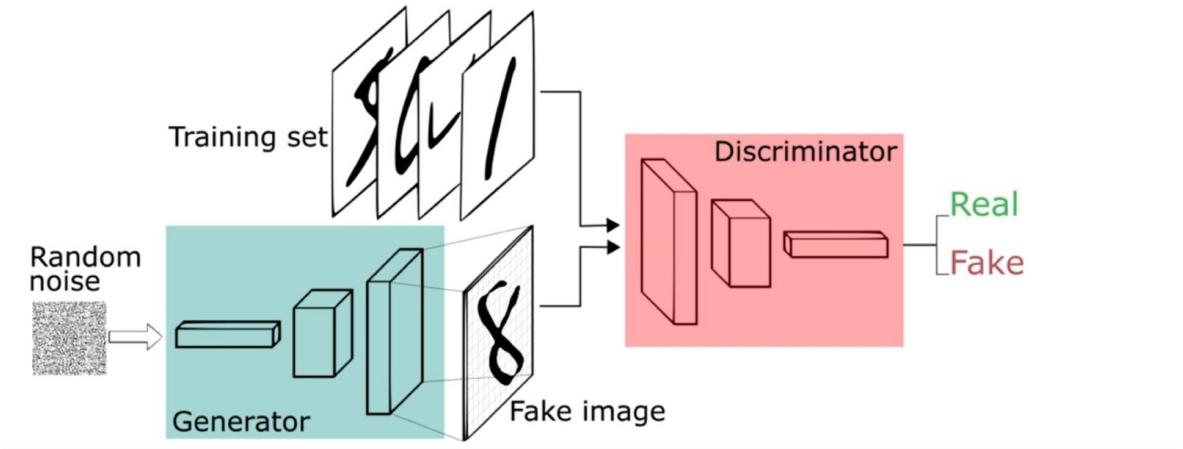


Figure 5 Architecture générale des Gan's

Un gan a deux réseaux de neurones, le Generator et le Discriminator. Au début les deux réseaux ne sont pas entrainés.

2.1 Generator

Le générateur est un réseau de neurones, généralement convolutif.

Dans une architecture GAN le générateur génère des images en prenant en entrée un vecteur de données normalisées appelé espace latent, au fur et à mesure le générateur génère de plus en plus des images réalistes et devient plus efficace.

2.2 Discriminator

Le discriminateur est un réseau de neurones, généralement convolutif.

Dans une architecture GAN le discriminateur est confronté à la fois à des images réelles issues du jeu de données et à des images fakes provenant du générateur.

Le discriminateur a pour mission de dire qu'une image réelle est réelle et une image fake est fake. Au fur et à mesure il rencontre des difficultés pour dire qu'une image est fake.

2.3 Fonction de loss

Pour une architecture GAN, elle contient deux réseaux de neurones de type CNN

- Générateur
- Discriminateur

Puisque c'est un apprentissage non supervisé, car l'apprentissage se fait entre les deux CNN donc ils doivent être connectées, la fonction de loss est une métrique d'échange entre le générateur et le discriminateur afin d'apprendre et d'améliorer les résultats.

Le but du générateur est de générer des images réalistes et le but du discriminateur est de distinguer entre les images générées et les images réelles.

On calcule la loss du discriminateur comme suit :

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

- Log D(xi) est la loss par rapport à une image réel xi
- Log (1 – D(G(zi))) est la loss par rapport à une image générée zi
- La soustraction 1 – D(G(zi)) signifie qu'on veut minimiser qu'une image fausse soit fausse.

Cette loss est appelée Binary cross entropy, elle est utilisée dans la classification binaire « dans notre cas Fake, Reel »

Ensuite on calcule la loss du générateur comme suit :

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

Le but c'est de maximiser qu'un image fake soit réelle.

3. Nos modèles

Nous avons testé deux architectures une pour une sortie d'images de 64x64 pixels et une autre pour une sortie d'image de 256x256 et les architectures sont faites comme suit :

3.1 Architecture 1

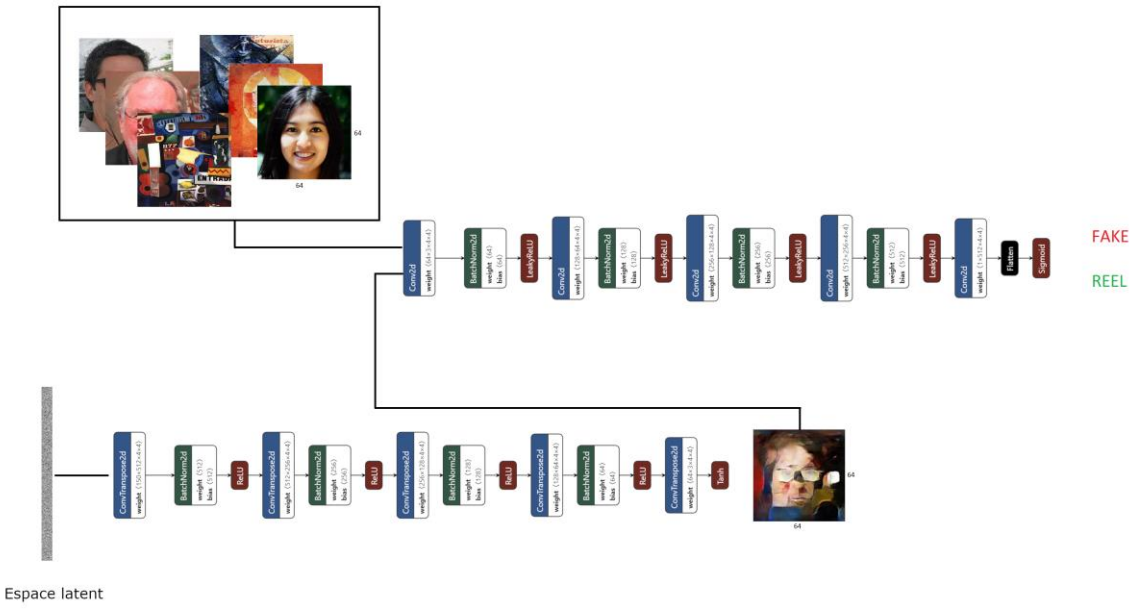


Figure 6 Architecture 1

Cette architecture représente notre premier modèle, le générateur contient 5 couches de convolutions transposées entre chaque couche nous faisant une Batch normalisation qui permet de faire une normalisation des poids de sortie de chaque convolution transposée. Ensuite après chaque batch normalisation nous appliquant la fonction d'activation Relu. Sauf pour la dernière couche après la convolution nous appliquons la fonction d'activation TanH.

Le discriminateur contient 5 couches de convolutions chaque convolution est d'une batch normalisation ensuite suivies par une fonction d'activation LeakyRelu , sauf pour la dernière convolution on fait un flattent ensuite une fonction d'activation sigmoïde .

3.2 Architecture 2

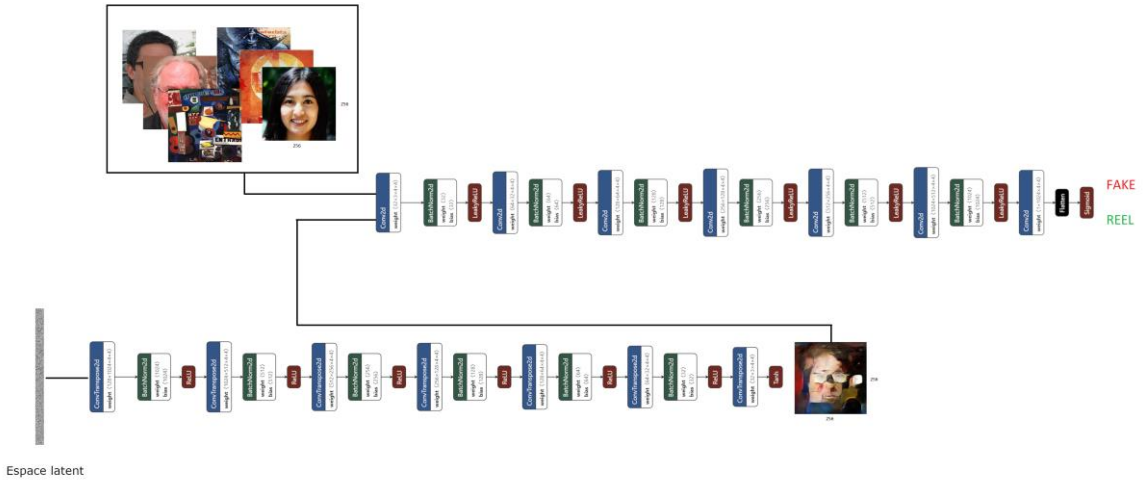


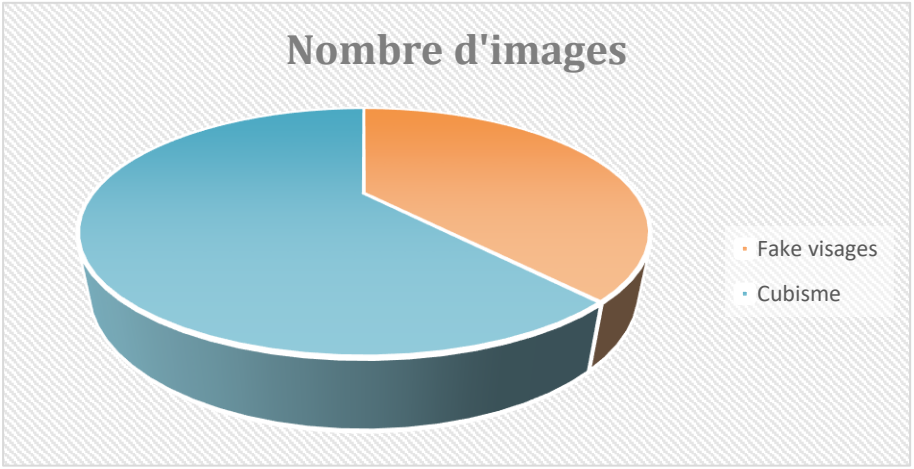
Figure 7 Architecture 2

La deuxième architecture est presque la même, la seule différence c'est l'ajout de deux layers pour faire une extraction de caractéristiques plus détaillée et bien évidemment pour générer une image plus grande 256x256 pixels.

4. Jeu de données

Afin d'entrainer nos modèles, nous avons utilisé un jeu de données de fake visage [2] et un autre jeu de données de tableau d'art cubisme.

Les statistiques de notre jeu de données sont comme suit



Voici quelques exemples de visages fake :

Figure 8 Fake visages

Voici quelques exemples de Cubisme

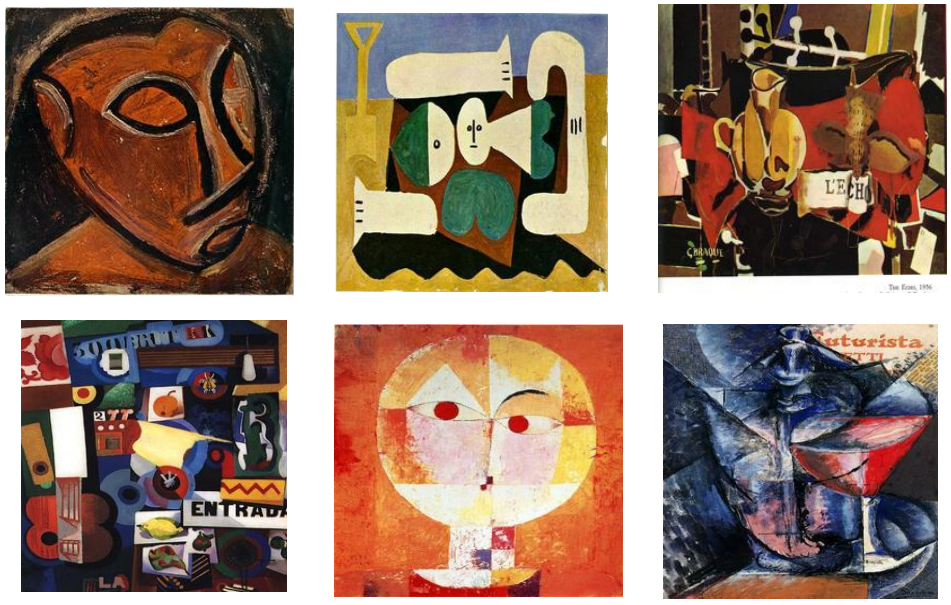


Figure 9 Art cubisme

5. Implémentation

Afin d'implémenter nos deux architectures nous avons utilisé le langage python et la bibliothèque pytorch.

Nous avons fait 4 expériences :

	Epoch	Batch size	Momentum	LR	Input size	Output size
Expérience 1	150	128	0,999	0,001	128	64x64
Expérience 2	700	128	0,999	<300 : 0,01 Sinon 0,0001	128	64x64
Expérience 3	630	128	0,999	0,0001	128	256x256
Expérience 4	1000	128	0,999	<100 : 0,001 Sinon 0,001	128	256x256

Pour les deux premières expériences nous avons appliqué les hyperparamètres pour l'architecture 1 et pour les deux dernière nous avons appliqué les hyperparamètres pour l'architecture 2.

Nous avons choisi la technique de changement du taux d'apprentissage « LR » pour que les modèles sur les expériences 2 et 4, pour l'expérience 2 nous avons mis le LR à 0,01 ensuite après l'époque 300 nous avons changé le LR à 0,0001. Nous avons fait la même chose pour l'expérience 4 où nous avons changé le LR de 0,001 à 0,0001 après l'époque 100.

6. Résultats

Après l'entrainement nous avons obtenu des résultats pour chaque expérience que nous illustrons ci-dessous :

- **Expérience 1**

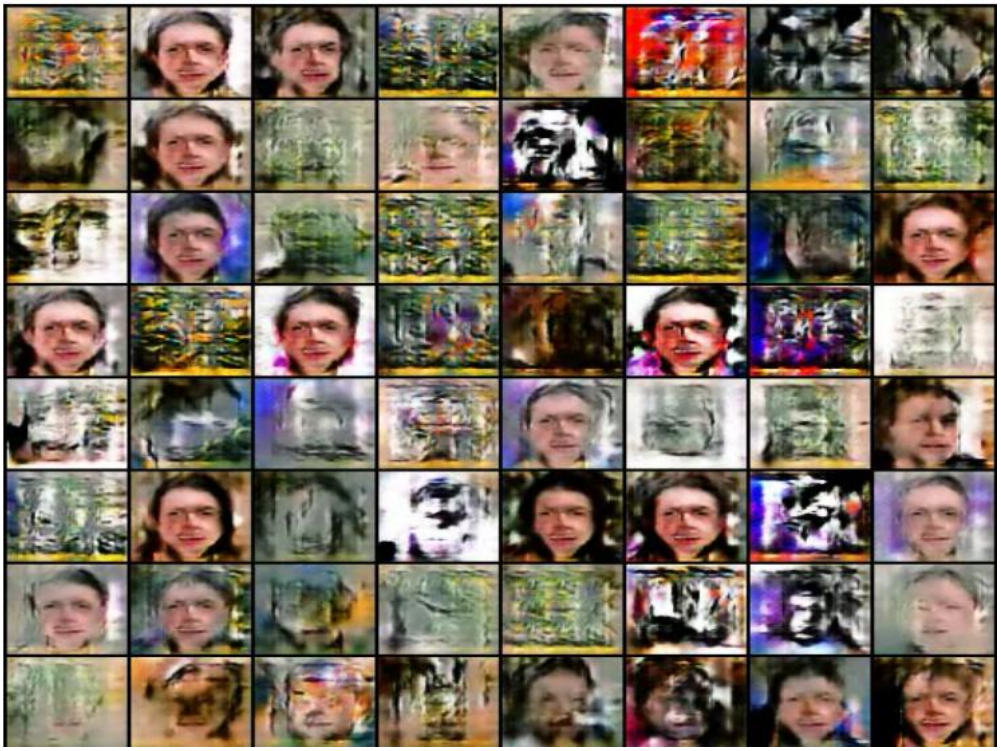


Figure 10 Résultats de l'expérience 1

- **Expérience 2**

Figure 11 Résultats de l'expérience 2

- **Expérience 3**

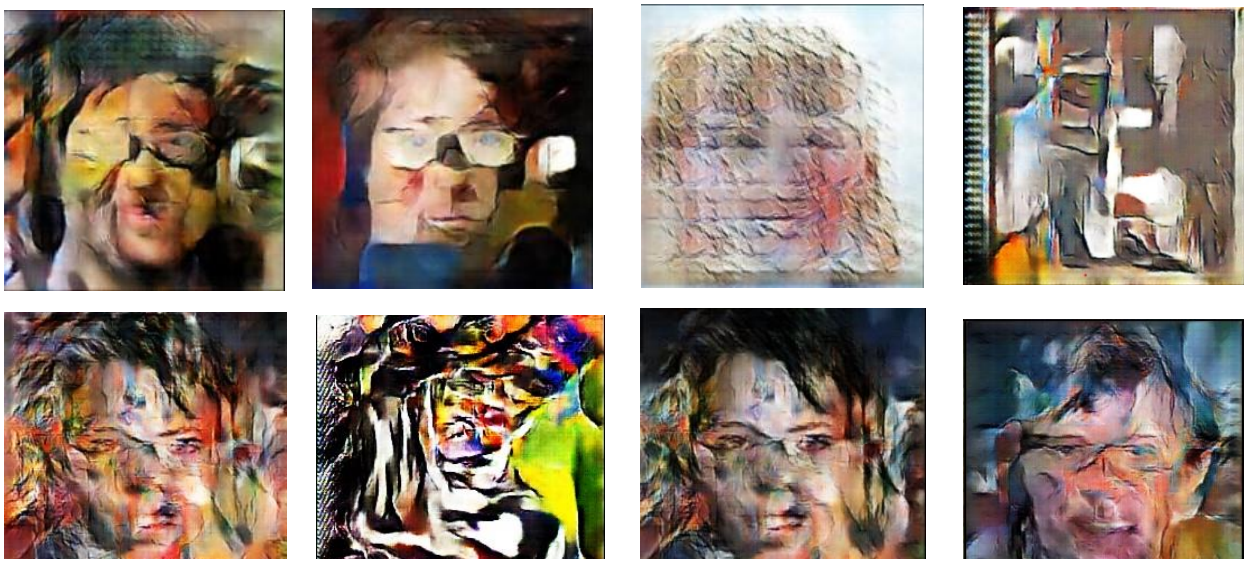


Figure 12 Résultats de l'expérience 3

- **Expérience 4**

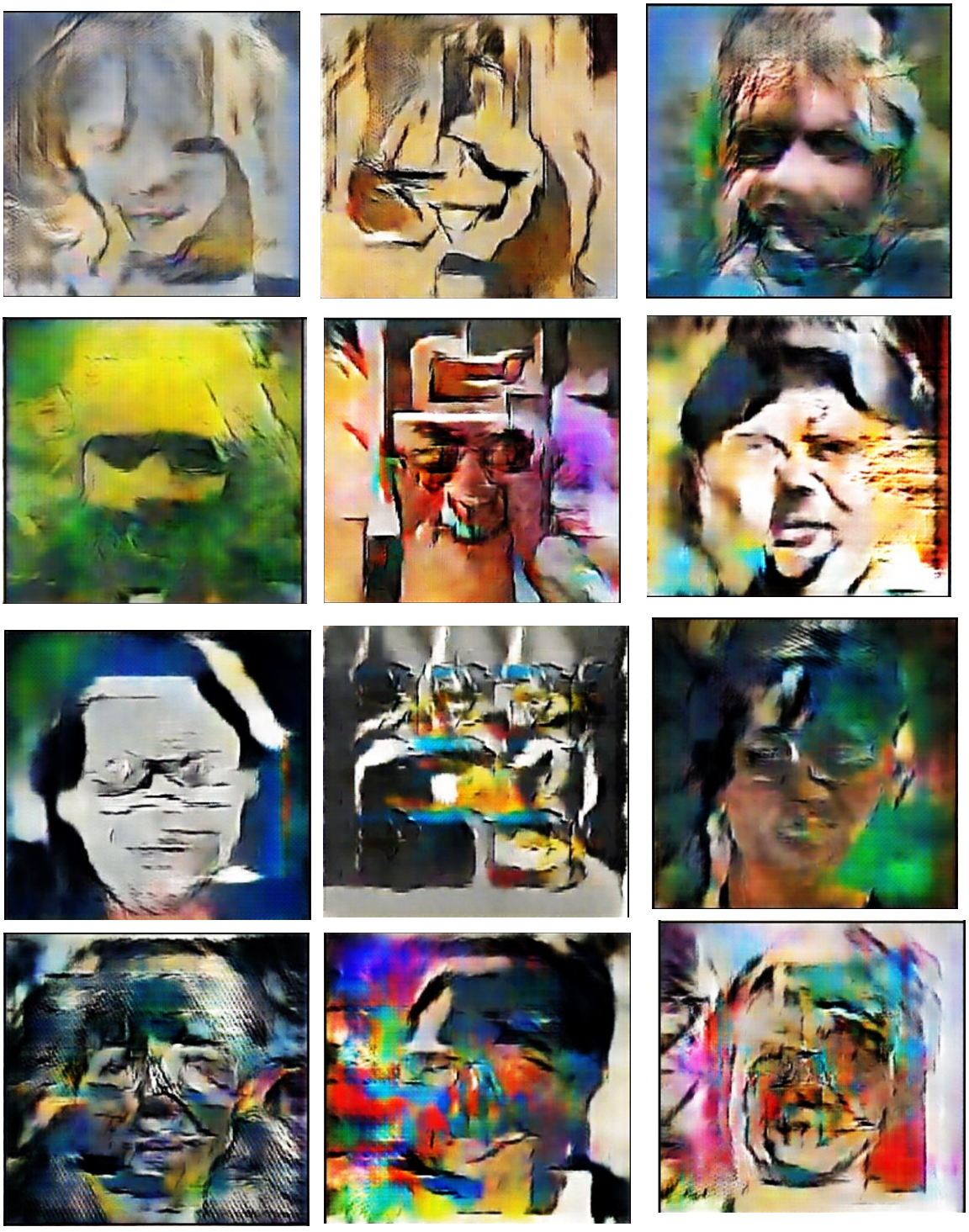


Figure 13 Résultats de l'expérience 4

Après avoir obtenu les résultats de chaque expérience, nous nous sommes focalisé sur la 4eme expérience qui nous donne de bon résultat, nous allons afficher les graphe de loss et de score de cette expérience.

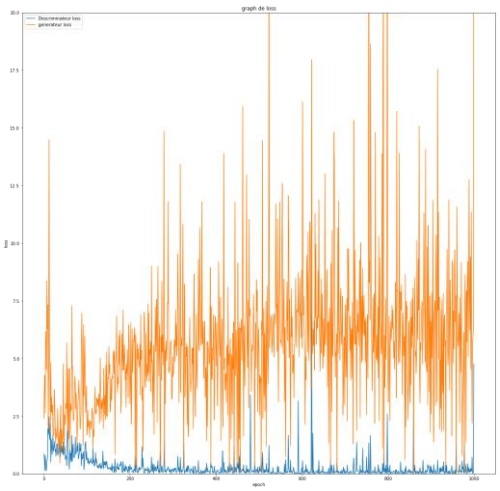


Figure 15 Graphe de loss de l'expérience 4

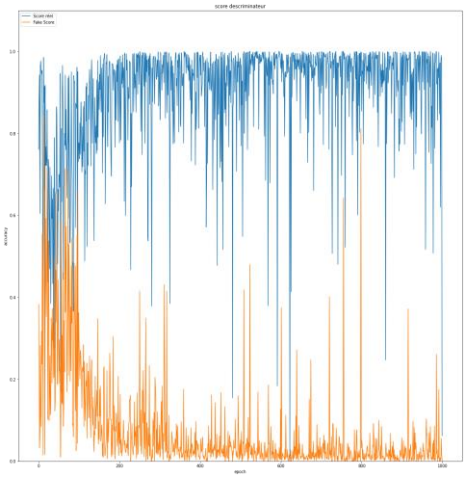


Figure 14 Graphe de score du discriminateur de l'expérience 4

7. Discussion

L'évaluation des GAN est souvent difficile à effectuer, une solution naïve existe, c'est de visualiser les images de sorti de chaque époque afin de faire une sélection visuelle.

Prenant exemple de la détection d'objet, nous évaluons par rapport à la vérité terrain en utilisant des métriques « IoU, F Score… » ou par exemple pour la classification en utilisant la matrice de confusion. Cependant les GAN's prennent en entrée un vecteur de bruit généré aléatoirement pour enfin obtenir une image générée, et le but c'est d'avoir une image réaliste.

Graphe de score du discriminateur : nous remarquons que de l'époque 0 à 150 nous avons un score qui est très perturber, ensuite nous remarquons que le score des images réelles est élevé et le score des images fausses générés est bas, le score c'est la moyenne du verdict du discriminateur, nous rappelons que le but c'est de maximiser qu'une image réelle soit réelle et une image fausse soit fausse.

8. Conclusion

Enfin pour conclure, nous avons pu comprendre l'architecture de base est utilisée dans les GAN's, nous avons implémenté cela sous python en utilisant pytorch et cela a été bénéfique pour nous afin d'enrichir nos connaissances.

Après l'entrainement nous avons obtenu des résultats satisfaisants qui ont été à la hauteur de nos attentes, malgré que nous avons eu des difficultés sur les ressources hardware pour l'entrainement de notre modèle a besoin de beaucoup d'époques afin de converger.

Pour améliorer notre travail nous proposant d'augmenter le nombre de layers afin d'augmenter les détails et d'avoir une image plus réaliste et également cela implique l'augmentation de la taille des images en sortie. Et également utiliser le Sliced Iterative Generator afin d'évaluer le modèle.

9. Bibliographie

1. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. « *Generative Adversarial Networks* » [2014]. arXiv:1406.2661

2. https://www.kaggle.com/datasets/tunguz/1-million-fake-faces

Table des illustrations