

Project Group: 12

Juin 2024

# Édition d'images de visages par manipulation de codes latents

by

Nicolas Billy

Charlotte-Alicia Bouée

Matthieu Kaeppelin

Marie Pizzini

**Abstract:** À travers ce projet, nous avons appris à utiliser des réseaux profonds génératifs afin de faire de la synthèse d'images avancée. Pour cela, nous nous sommes aidés de 3 articles décrivant l'architecture StyleGAN 1 afin d'implémenter notre version de StyleGAN2. L'architecture StyleGAN [1], par exemple, a ainsi permis la représentation et la synthèse de visages. Des visages très réalistes mais totalement fictifs peuvent être générés simplement en échantillonnant l'espace latent de cette architecture. Le but du projet est donc de comprendre dans un premier temps cette approche et sa variante StyleGAN2, puis de tester InterfaceGan, une méthode supervisée de manipulation des codes latents permettant l'édition de visages selon des attributs sémantiques (lunettes, âge, sourire), par exemple comme proposé dans [2]. On testera ensuite la possibilité de structurer l'espace latent à partir d'outils élémentaires tel qu'une PCA (principal component analysis), une méthode non supervisée proposée dans [3].

## Contents

<b>1</b>	<b>Introduction au réseau pré-entraîné StyleGAN2-ADA</b>	<b>3</b>
1.1	Architecture Traditionnelle . . . . .	4
1.2	StyleGAN . . . . .	4
1.3	Implémentation de la génération d'image . . . . .	4
1.4	Obtention du vecteur latent à partir d'une image cible . . . . .	5
<b>2</b>	<b>Implémentation de méthodes non supervisées : manipulation de l'espace latent à l'aide de GANSpace</b>	<b>7</b>
2.1	PCA traditionnel . . . . .	7
2.2	Autre méthode : Kernel PCA . . . . .	9
2.3	Autoencodeur . . . . .	11
2.4	Comparaison des méthodes non supervisées . . . . .	12
<b>3</b>	<b>Implémentation de méthodes supervisées : Manipulation de l'espace latent à l'aide de la méthode InterfaceGAN</b>	<b>13</b>
3.1	Introduction à la méthode InterfaceGAN . . . . .	13
3.2	Prérequis . . . . .	14
3.3	Entrainement des SVM et analyse de la séparation des attributs par un SVM linéaire . . . . .	15
3.4	Manipulation conditionnelle . . . . .	20
3.5	Manipulation couche par couche . . . . .	23
3.6	Édition d'image . . . . .	26
<b>4</b>	<b>Sources</b>	<b>27</b>
<b>5</b>	<b>Annexes</b>	<b>28</b>
5.1	Annexe 1 . . . . .	28
5.2	Annexe 2 . . . . .	29
5.3	Annexe 3 . . . . .	30

# 1 Introduction au réseau pré-entraîné StyleGAN2-ADA

Les méthodes de génération d'images, notamment les réseaux GAN, ont considérablement amélioré la résolution et la qualité des images générées. Cependant, certains aspects de l'espace latent d'un GAN traditionnel restent difficiles à manipuler comme dit dans l'article [1]. C'est pourquoi l'article introduit une nouvelle approche, le StyleGAN. Au lieu de fournir le code latent directement à la première couche du générateur, StyleGAN commence par une entrée constante apprise et ajuste le "style" de l'image à chaque couche de convolution en fonction du code latent. Cette approche permet de contrôler certaines caractéristiques de l'image (couleur des cheveux, pose, genre, sourire...).

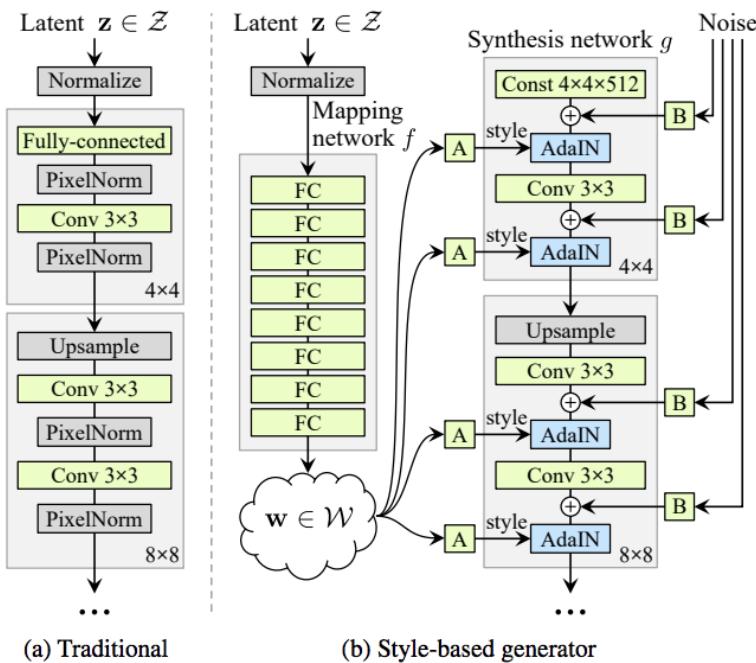


Figure 1: Comparaison de l'architecture d'un GAN traditionnel et d'un StyleGAN  
Ce qui nous intéresse donc dans ce projet est de pouvoir se déplacer dans certaines directions dans l'espace latent pour générer des images avec des attributs sémantiques particuliers.

Voici une petite comparaison que nous pouvons faire entre les deux architectures :

## 1.1 Architecture Traditionnelle

- Le code latent est fourni à la première couche du générateur.
- Chaque couche applique des convolutions traditionnelles avec normalisation et activation.
- La sortie finale est convertie en image RGB.

## 1.2 StyleGAN

- Le code latent est d'abord mappé à un espace latent intermédiaire via un réseau de mappage non linéaire.
- Utilisation de la normalisation d'instance adaptative (AdaIN) pour ajuster le style à chaque couche de convolution.
- Ajout de bruit gaussien à chaque couche pour introduire des détails stochastiques.

## 1.3 Implémentation de la génération d'image

Les expérimentations ont principalement été faites sur les datasets CELEBA-HQ et FFHQ. En quelques phrases, on commence avec une configuration de référence améliorée pour ensuite ajouter le réseau de mappage et les opérations AdaIN. Cela revient à commencer la synthèse d'images à partir d'un tenseur constant appris de  $4 \times 4 \times 512$  (dans StyleGAN 1, c'est un peu différent pour StyleGAN 2). L'introduction des entrées de bruit ( $E$ ) qui améliore encore les résultats, ainsi qu'une nouvelle régularisation qui décortille les styles voisins et permet un contrôle plus précis sur les images générées.

Dans notre code, nous avons utilisé la fonction **generate** contenue dans le fichier `generate.py`, pour générer nos images et tester nos différentes méthodes de manipulation des codes latents. Cette fonction permet :

- de générer des images à partir d'un vecteur latent aléatoire  $Z$  de dimension 512 et de seed
- de générer des images à partir d'un vecteur latent projeté  $W$  de dimension

18x512

Dans le cadre de StyleGAN 2, nous avons découvert le concept de désenchevêtrement qui permet au modèle de séparer et de représenter distinctement différents facteurs de variation dans les données générées. Contrairement aux GAN traditionnels où les attributs tels que la pose, l'identité, et d'autres attributs peuvent être mélangés dans l'espace latent, StyleGAN 2 propose une architecture qui permet une séparation plus nette de ces caractéristiques. Au lieu de manipuler ces caractéristiques ensemble, on peut maintenant les manipuler séparément.



Figure 2: Image générée à l'aide du réseau StyleGAN2-ADA pré-entraîné

## 1.4 Obtention du vecteur latent à partir d'une image cible

L'algorithme de projection latente permet de trouver le vecteur latent  $W$  qui génère une image la plus proche possible d'une image cible donnée.

- Pour cela on part d'une image générée aléatoirement, on utilise un réseau VGG16 pour extraire les features de l'image générées et de l'image cible, que l'on compare à l'aide d'une loss.
- Par descente de gradient, on modifie le vecteur latent  $W$  de l'image générée pour minimiser la loss.
- On obtient alors le vecteur latent  $W$  qui génère l'image la plus proche de l'image cible.

Dans cette première partie, nous avons découvert StyleGAN2-ADA et ses capacités de génération d'images de visages à travers la fonction **generate**. Nous avons également vu comment obtenir un vecteur latent  $W$  à partir d'une image cible en passant par un algorithme d'optimisation assez gourmand en ressources. Dans la partie suivante, nous allons voir comment manipuler les vecteurs latents  $W$  pour modifier les images générées au travers de plusieurs méthodes différentes.

## 2 Implémentation de méthodes non supervisées : manipulation de l'espace latent à l'aide de GANSpace

### 2.1 PCA traditionnel

L'idée dans cette partie a été de pouvoir identifier les principales directions de l'espace latent grâce à une analyse en composantes principales, et de se déplacer sur ces dernières pour identifier lesquelles permettent de modifier des attributs sémantiques sur des visages générés par la méthode décrite en première partie.

Dans notre code, on commence par générer des échantillons dans l'espace latent  $w$  à partir de vecteurs aléatoires  $z$ . Ensuite, on applique la PCA pour réduire la dimensionnalité de ces échantillons et identifier les composantes principales, réduisant les vecteurs. En utilisant ces composantes, le code explore les variations dans l'espace latent, (dont les vecteurs  $w$  sont de dimension 1x512), selon les directions des composantes principales :

$$w_0 = w + Vx \quad (1)$$

$x$  est un vecteur "paramètre de contrôle", c'est-à-dire que toutes ses entrées  $x_k$  sont égales à 0 jusqu'à ce qu'elles soient modifiées par l'utilisateur, ce sont des coordonnées dans l'espace du PCA.

$V$  est une matrice de passage de l'espace du PCA à l'espace latent, la quantité  $Vx$  constitue donc notre déplacement.

On a également créé un vecteur  $X$  dont  $x$  est une colonne, chaque ligne de  $X$  correspond à une direction et les colonnes représentent le nombre d'évolutions/étapes que l'on effectue, ici on a choisi de parcourir de  $-2\sigma$  à  $2\sigma$  sur l'axe de chaque direction par rapport à la position initiale de l'image générée, pour ne pas aller trop loin dans l'espace latent.

Nous avons commencé par un exemple assez large, en prenant en compte seulement la première composante obtenue avec la PCA.

Comme dit dans l'article [3], cela montre le phénomène d'intrication des caractéristiques de l'image. En effet, on remarque qu'en se déplaçant dans l'espace latent selon la première composante, cela modifie le genre, la pose du visage, la



Figure 3: Série d'images montrant le déplacement sur la composante 1

présence de lunettes, la quantité de cheveux sur le front...Notre but maintenant serait de modifier légèrement notre code pour tester le "layer wise edits" évoqué par l'article [3], c'est-à-dire le fait de se déplacer dans certaines directions mais en modifiant seulement certaines couches du réseau de neurones. Les modifications sont donc appliquées à w sur les couches spécifiées (de 1 à 18) par l'utilisateur en entrée de la fonction comme ceci :

```
#Gender
directions = [0]
layers =list(range(0, 18))
img_list_gender , X_list_gender = move_according_to_component(G, device,pca, 'output_directory_pca', directions=directions)

plot_results(img_list_gender, X_list_gender)
```

Python

Figure 4: Code pour l'obtention du déplacement dans l'espace latent selon certaines directions et en ne modifiant que les couches contenues dans "layers"

Nous avons donc testé plusieurs combinaisons de couches afin de déterminer lesquelles permettaient d'isoler totalement certaines caractéristiques notamment :

- En se déplaçant sur la composante 11 et en ne modifiant que les couches 0 à 4 du réseau sur les 18 existantes pour isoler l'attribut de la quantité de cheveux
- En se déplaçant sur la composante 4 et en ne modifiant que les couches 5 à 17 du réseau sur les 18 existantes pour isoler l'attribut de la barbe

et d'autres encore.



Figure 5: Série d'images montrant l'évolution de la quantité de cheveux



Figure 6: Série d'images montrant l'évolution de la barbe

## 2.2 Autre méthode : Kernel PCA

Après avoir réalisé une analyse en composantes principales traditionnelle, nous avons également implémenté une méthode similaire mais en utilisant un Kernel PCA cette fois ci avec un noyau RBF (Radial Basis Function). Nous voulions tester cette méthode car sans être sûrs du résultat, l'utilisation du KPCA permet souvent de capturer des structures enchevêtrées plus complexes que l'on peut séparer plus facilement en passant dans un autre espace grâce au noyau. La seule différence avec le code précédent concernant le PCA est que cette fois-ci nous avons transformé le vecteur latent  $w$  à l'aide du KPCA afin d'avoir ses coordonnées dans l'espace transformé.

Notre premier réflexe a été de regarder la différence en termes mathématiques. Nous avons donc calculé le cosinus et la norme euclidienne, de la différence entre le déplacement sur la direction 0 généré avec la méthode utilisant le KPCA et celui généré avec la méthode utilisant le PCA.

En faisant différents tests, avec des échantillons identiques au départ ou différents pour les deux méthodes, on remarque que plus l'on prend d'échantillons pour entraîner le PCA par rapport au KPCA, plus la différence entre les deux déplacements

générés est faible. Par exemple, en prenant seulement 1000 échantillons pour entraîner le KPCA et 100 000 pour entraîner le PCA, la norme euclidienne de leur différence est environ égale à 10 et leur cosinus à 0.78. Plus le cosinus se rapproche de 1 plus cela signifie que les deux déplacements sont proches et qu'ils vont dans la même direction. A l'inverse, en prenant 10 000 échantillons pour entraîner le PCA et le KPCA, on obtient cette fois-ci un cosinus de 0.41 ce qui signifie que leurs directions sont déjà plus décorrélées qu'avant. Comme leur cosinus est compris entre 0.78 et 0.41 selon nos tests, le PCA et le KPCA restent tout de même corrélés mais il est possible, parfois, de remarquer des différences entre les deux méthodes.

Nous avons également testé en réalisant deux PCA différents avec les mêmes échantillons, si nous pouvions obtenir des composantes principales légèrement différentes. Avec 10 000 échantillons, le cosinus et la norme euclidienne de la différence du déplacement généré par les deux pca sont respectivement égaux à 1 et 0, ce qui signifie que malgré l'aléatoire qu'il pourrait y avoir, les deux pca donnent les mêmes composantes principales.



Figure 7: Série d'images montrant l'évolution sur la direction 1 grâce au PCA



Figure 8: Série d'images montrant l'évolution sur la direction 1 grâce au KPCA

Par exemple, sur l'image ci dessus, le KPCA permet de conserver les lunettes sur toutes les images contrairement au PCA. Même si les différences visuelles ne sont pas très notables, le KPCA permet tout de même de rendre moins enchevêtré un attribut qui l'est souvent, c'est-à-dire le port de lunettes.

## 2.3 Autoencodeur

Une autre manière de réarranger l'espace  $\mathcal{W}$  des features est d'utiliser un autoencoder qui va reconstruire cet espace  $\mathcal{W}$  en lui imposant des contraintes. Nous allons donc utiliser la structure d'un Variational Autoencoder en modifiant des éléments de la loss. Nous avons choisi 50 dimensions pour l'espace latent pour forcer l'autoencoder à apprendre une représentation plus synthétique des données, et nous avons imposé des contraintes pour la reconstruction de l'espace  $\mathcal{W}$ . Nous avons utilisé une contrainte d'orthogonalité comme proposée dans l'article "Clustering with Orthogonal Autoencoders" [4]. Comme nous voulions faire des manipulations dans l'espace latent de cet autoencoder, nous avons tenté d'imposer une distribution dans cette espace latent, en particulier, en imposant une somme de deux gaussiennes dans chaque composante de l'espace latent. Pour cela nous avons approximé la KL Divergence comme étant la somme des KL divergence des deux gaussiennes.

En annexe 5.1, le modèle de l'autoencoder est détaillé avec les différentes dimensions, et couches ainsi que l'évolution des différentes loss avec l'annexe 5.2.

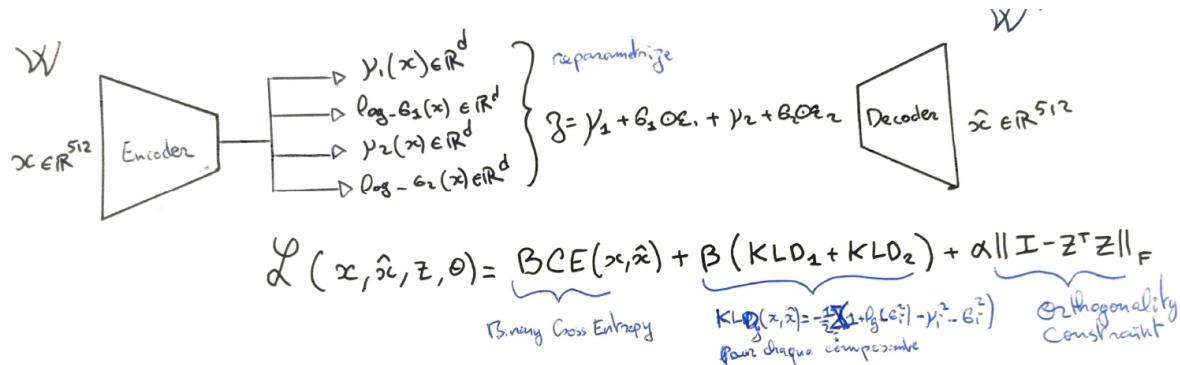


Figure 9: Description de l'autoencoder, et de la Loss utilisée

Sur les images ci dessous, on voit que l'autoencoder permet de reconstruire un sample à fournir au générateur qui donne une image relativement similaire à l'image originale.

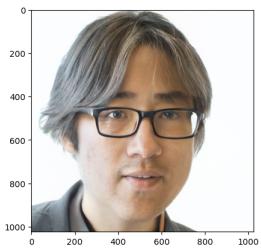


Figure 10: Image Original générée par un sample de l'espace latent

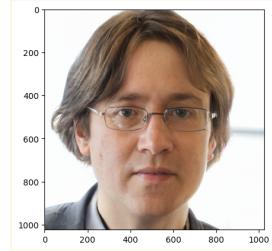


Figure 11: Image générée avec la projection du code latent dans l'espace reconstruit par l'autoencoder

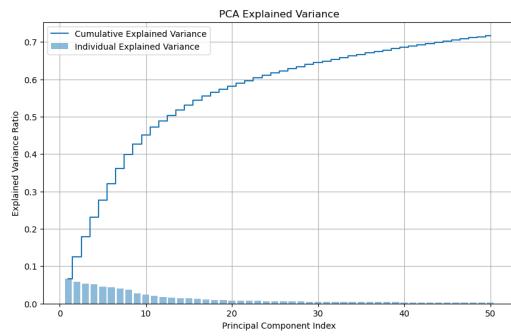


Figure 12: PCA in the original space

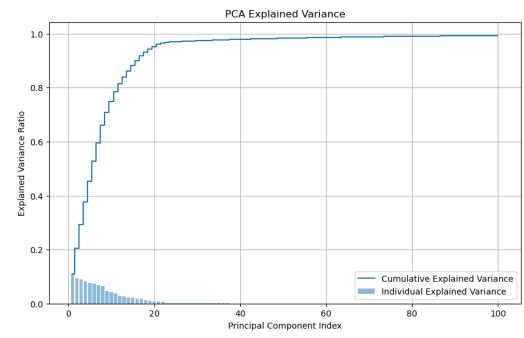


Figure 13: PCA in the reconstructed space by the VAE

Notons  $\mathcal{W}'$  l'espace latent  $\mathcal{W}$  reconstruit, il est intéressant de voir les différences de l'analyse par composante principale. Dans  $\mathcal{W}'$ , 20 composantes permettent d'expliquer plus de 95% de la variance totale alors que dans  $\mathcal{W}$ , 50 composantes expliquent seulement 70 % de la variance totale.

## 2.4 Comparaison des méthodes non supervisées

La figure 14 montre les résultats des déplacements dans différentes directions et choix de layers avec les différentes méthodes explorées ici. Les colonnes PCA et KPCA sont les images ou le déplacement est obtenu après analyse de l'espace latent  $\mathcal{W}$ . La colonne VAE montre l'image obtenue après avoir créé un déplacement dans l'espace  $\mathcal{W}'$  qu'on ajoute au code latent de l'image original dans  $\mathcal{W}$ . Pour

ces 3 colonnes, l'image de référence est celle originale de la figure 10. La colonne VAE Projected montre l'image obtenue après avoir créé un déplacement dans l'espace  $\mathcal{W}'$  qu'on ajoute au code latent que l'on a projeté dans  $\mathcal{W}'$ . L'image de base pour les déplacements est celle de la figure 11.

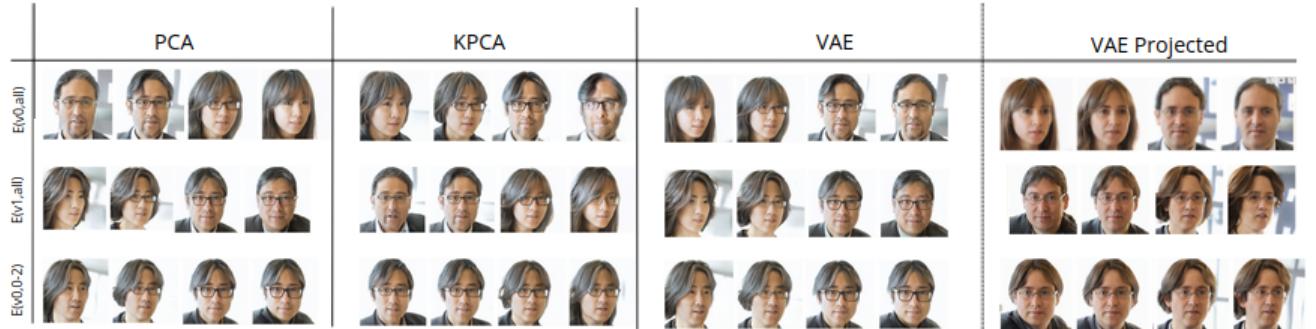


Figure 14: Comparaison des résultats de la méthode utilisée dans l'article avec les différentes méthodes

Les trois premières méthodes ont des résultats assez similaires, du fait qu'il y a des faibles différences entre le déplacement que l'on apporte au vecteur du code latent original de  $\mathcal{W}$ . La dernière méthode montre des résultats visuellement différents car ici on projette ce vecteur de  $\mathcal{W}$  dans  $\mathcal{W}'$ . On remarque toutefois qu'en modifiant la première direction (E(0,all)) on modifie bien le genre.

### 3 Implémentation de méthodes supervisées : Manipulation de l'espace latent à l'aide de la méthode InterfaceGAN

#### 3.1 Introduction à la méthode InterfaceGAN

L'objectif d'InterfaceGAN est l'exploration et la manipulation de l'espace latent des réseaux antagonistes génératifs (GAN), en particulier du modèle StyleGAN 2. Cette méthode décrite dans le papier [2] *InterFaceGAN: Interpreting the Disentangled Face Representation Learned by GANs*, publié par Yujun Shen, Ceyuan Yang, Xiaoou Tang, Fellow, IEEE, and Bolei Zhou, Member, IEEE, permet un contrôle précis et intuitif des attributs des images synthétiques générées.

Chaque image générée par styleGAN provient d'un vecteur Z de dimension 512, qui a été transformé par un réseau de neurones complètement connecté en une matrice W de dimension (1,18,512). C'est l'étape dite de mapping. InterfaceGAN vise à analyser cet espace W pour comprendre comment les différents points se traduisent en caractéristiques visuelles dans les images générées.

Pour identifier les directions correspondant à des attributs spécifiques (comme le genre, l'âge ou le sourire), InterfaceGAN utilise des classificateurs pré-entraînés. Ces classificateurs permettent de distinguer les images contenant certains attributs de celles qui ne les contiennent pas.

Les résultats des classificateurs permettent de voir l'espace latent comme un espace géométrique. Les points latents associés à un attribut particulier sont séparés des autres points par un hyperplan, ce qui permet d'identifier des directions spécifiques dans l'espace latent.

En déplaçant un point latent le long de la direction identifiée, il devient possible de modifier les attributs de l'image générée sans affecter ses autres caractéristiques. Par exemple, un déplacement dans la direction associée à l'âge permet de rendre un visage plus jeune ou plus âgé tout en conservant son identité.

Dans les sections suivantes seront décrites toutes les étapes de notre implémentation d'InterfaceGAN et de nos manipulations dans l'espace latent.

### 3.2 Prérequis

InterfaceGAN nécessite l'utilisation d'un classificateur afin de classifier sur différents attributs des images générées et de créer un dataset d'images classifiées. Nous avons récupéré pour cela un classificateur transmis par notre professeur encadrant.

Nous avons généré un dataset de 50 000 images classifiées, et avons enregistré leurs vecteurs latents W ainsi que la probabilité de chacun des attributs considérés. On retrouve :

- |                      |                      |                    |                    |
|----------------------|----------------------|--------------------|--------------------|
| 1. <b>Bald</b>       | 4. <b>Brown hair</b> | 7. <b>Gender</b>   | 10. <b>Smiling</b> |
| 2. <b>Black hair</b> | 5. <b>Eyeglasses</b> | 8. <b>Mustache</b> | 11. <b>Hat</b>     |
| 3. <b>Blond hair</b> | 6. <b>Make-up</b>    | 9. <b>Beard</b>    | 12. <b>Young</b>   |

### 3.3 Entrainement des SVM et analyse de la séparation des attributs par un SVM linéaire

Pour chacun des attributs précédents, nous avons entraîné un Support Vector Machine (SVM) sur des datasets comportant les 2000 images ayant la plus grande probabilité d'avoir cet attribut et les 2000 images ayant la plus petite probabilité de l'avoir. Le SVM nous a permis de calculer un hyperplan séparateur de ces deux groupes d'images et d'en déduire un vecteur normal. Ce vecteur normal nous donne la direction selon laquelle se déplacer pour pouvoir modifier l'attribut considéré.

Afin de pouvoir visualiser le travail du SVM, on peut projeter les vecteurs latents des images d'entraînements sur la direction du vecteur normal de l'attribut considéré. En traçant un histogramme de ces vecteurs projetés, on obtient les figures suivantes :

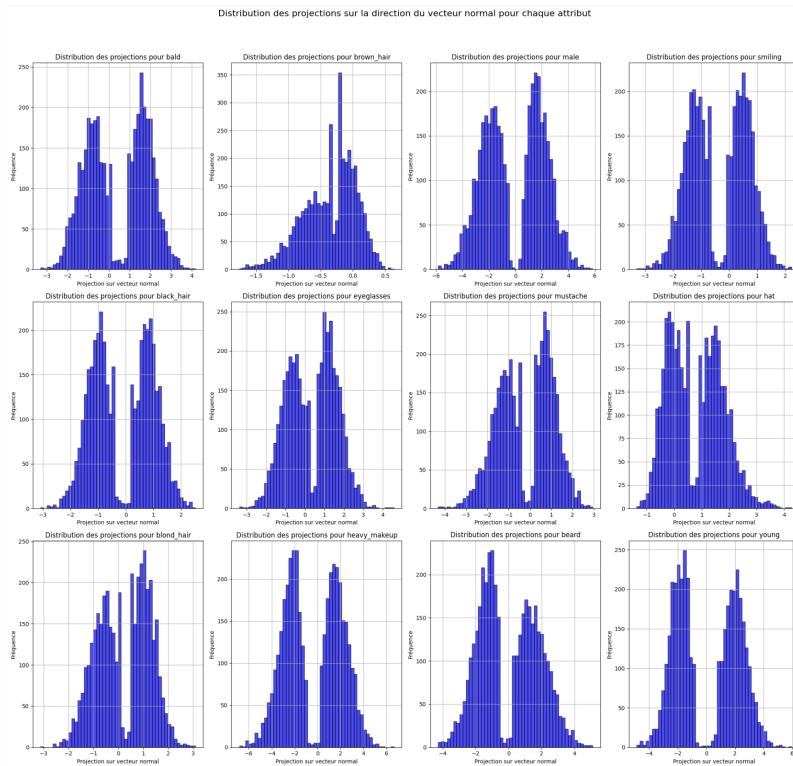


Figure 15: Histogrammes des projections des vecteurs latents  $W$  des images d'entraînements sur la direction de chaque attribut

On remarque bien que les distributions sont bien réparties de chaque côté de l'hyperplan pour la plupart des attributs. On voit que la frontière n'est pas très bien placée pour les attributs **brown hair** et **hat**. Cela peut s'expliquer de deux manières : ou bien il n'y a pas eu suffisamment d'images avec un chapeau ou sans chapeau (idem pour les cheveux marrons), ou bien la direction de ces attributs n'est pas linéaire ce qui expliquerait pourquoi le svm linéaire ne parvient pas à trouver de séparation pertinente.

On voit donc que le choix du SVM linéaire est pertinent puisque la séparation est très bonne pour la plupart des attributs. Nous nous sommes posés la question d'utiliser un noyau 'rbf'. Cependant, bien que la séparation soit très bonne, il est beaucoup plus compliqué de se déplacer dans la bonne direction car celle-ci n'est pas linéaire.

Un autre intérêt de ces projections est qu'elles nous permettent de voir les bornes entre lesquelles les images se situent sur les axes.

En testant le déplacement le long d'une direction on obtient par exemple :



Figure 16: Mouvement sur la composante du sourire



Figure 17: Mouvement sur la composante des lunettes

Après nos tests sur tous les attributs, nous sommes arrivés à ces conclusions :

- **Mouvement autour de la direction young:** On remarque ici que le mouvement autour de la direction young permet de rajeunir ou vieillir les personnes sur les images. Cela fonctionne très bien. On remarque que les attributs 'eyeglasses' et 'young' sont enchevêtrés, de même pour la couleur des cheveux et (légèrement) le teint de la peau.
- **Mouvement autour de la direction eyeglasses:** On remarque ici que le mouvement autour de la direction eyeglasses permet de mettre ou enlever des lunettes aux personnes sur les images (et la taille des lunettes grossit)
- **Mouvement autour de la direction beard:** On remarque ici que le mouvement autour de la direction beard permet de mettre ou enlever une barbe aux personnes sur les images. On remarque que les attributs 'beard' et 'male' sont enchevêtrés (ce qui est logique), et que 'beard' et 'eyeglasses' sont aussi enchevêtrés.
- **Mouvement autour de la direction mustache:** On remarque ici que le mouvement autour de la direction mustache permet de mettre ou enlever une moustache aux personnes sur les images. On remarque que les attributs 'mustache' et 'male' sont enchevêtrés.
- **Mouvement autour de la direction smiling:** On remarque ici que le mouvement autour de la direction smiling permet de faire sourire ou non les personnes sur les images. On remarque que les attributs 'smiling' et 'young' sont légèrement enchevêtrés. (On remarque ici que les directions 'eye-

glasses' et 'smiling' ne sont pas du tout enchevêtrés).

- **Mouvement autour de la direction male:** On remarque ici que le mouvement autour de la direction 'male' permet de changer le genre des personnes sur les images. On remarque que les attributs 'male' et 'beard' sont enchevêtrés, de même pour 'male' et 'mustache' et 'male' et 'eyeglasses'.

Les différentes observations faites sur l'enchevêtrement des attributs peuvent être confirmées en affichant sur les histogrammes précédents la projection des images au fur et à mesure que l'on se déplace le long d'une direction.



Figure 18: Mouvement le long de la composante Young

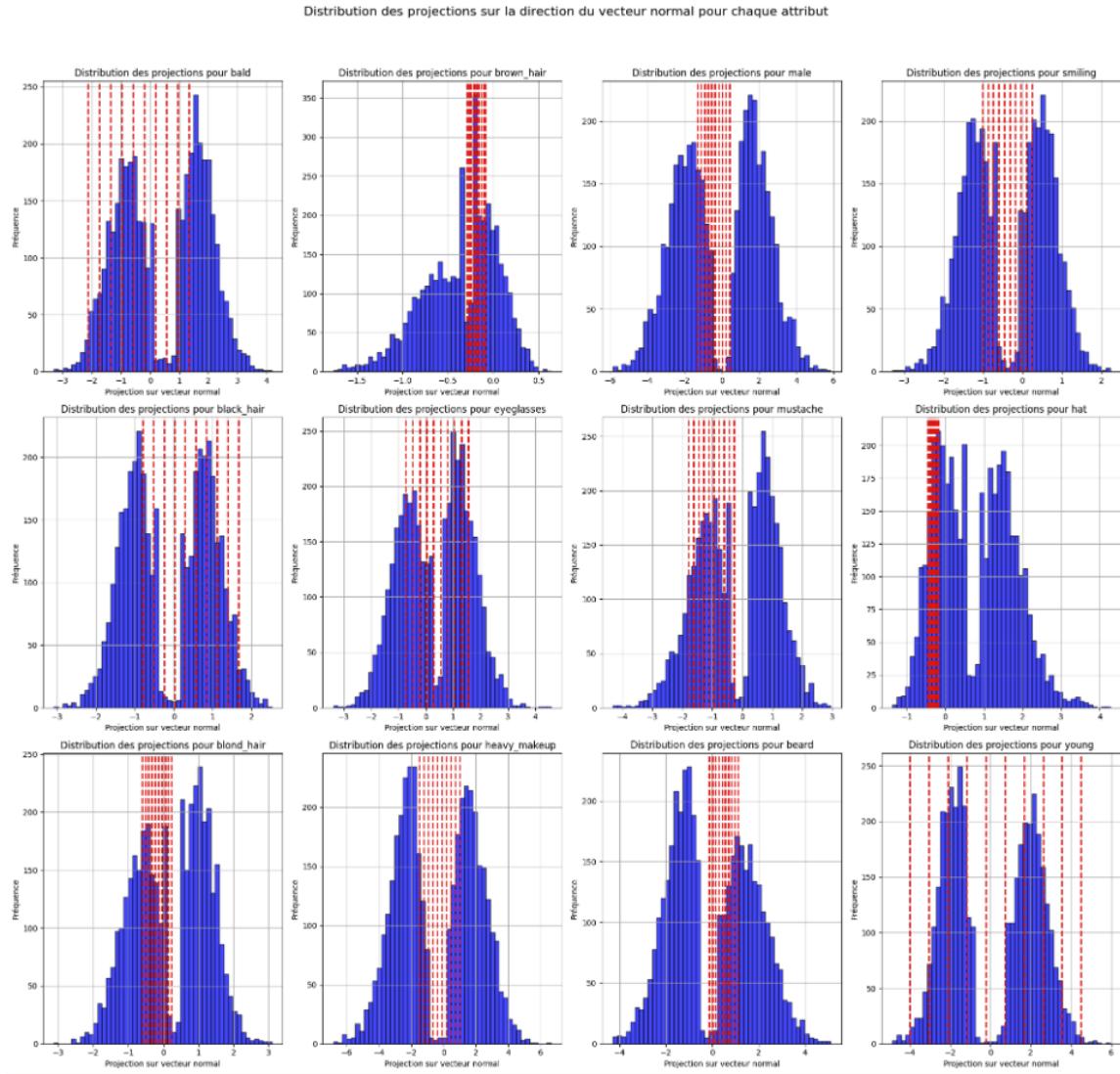


Figure 19: Histogramme sur la composante **young** avec les projections des 10 images générées

On observe ici très bien sur le dernier histogramme les projections régulières le long de la direction **young**. Théoriquement, si toutes les directions étaient perpendiculaire, c'est à dire si seul l'âge changeait sur la photo sans les lunettes ou le caractère chauve on devrait observer sur tous les autres histogrammes une seule ligne. Cependant ce n'est pas le cas. On voit ici très bien l'enchevêtrement

de l'attribut **young** avec les autres attributs. On a même un aperçu de "*l'intensité*" de cet enchevêtrement en voyant la largeur de la répartition des projections sur les histogrammes. Par exemple les attributs **bald** et **black hair** sont très enchevêtrés avec **young**. **Eyeglasses** et **smiling** le sont aussi pas mal. On voit aussi qu'au contraire, l'attribut **hat** et l'attribut **young** ne sont pas du tout enchevêtrés.

Pour continuer dans cette démarche d'analyse de l'enchevêtrement, on peut calculer la "semantic boundary correlation" qui permet de calculer à quel point deux directions sont proches à l'aide de la formule suivante :  $\cos(n_1, n_2) = n_1^T n_2$

On observe les résultats suivants :

Table 1: Tableau de corrélation entre les principaux attributs

	<b>bald</b>	<b>eyeglasses</b>	<b>male</b>	<b>smiling</b>	<b>young</b>	<b>heavy_makeup</b>
<b>bald</b>	1.0	0.18	0.24	0.07	-0.41	-0.30
<b>eyeglasses</b>	0.18	1.0	0.16	0.07	-0.27	-0.23
<b>male</b>	0.24	0.16	1.0	-0.22	-0.20	-0.68
<b>smiling</b>	0.07	0.07	-0.22	1.0	-0.15	0.17
<b>young</b>	-0.41	-0.27	-0.20	-0.15	1.0	0.30
<b>heavy_makeup</b>	-0.30	-0.23	-0.68	0.17	0.30	1.0

On voit très bien sur ce tableau la corrélation entre les différents attributs et l'on retrouve bien les résultats obtenus précédemment sur l'attribut **young**.

On remarque que ces corrélations entre les attributs ont très souvent du sens. En effet il y a beaucoup plus de chauves chez les personnes agées que chez les jeunes ( $\text{corr} = 0,41$ ), et un homme n'aura pas beaucoup de maquillage quand une femme aura plus de chance d'être fortement maquillée ( $\text{corr} = 0,68$ ).

### 3.4 Manipulation conditionnelle

Au vu de ces observations, il paraît intéressant d'essayer de déplacer un vecteur latent le long de la direction d'un attribut en empêchant le déplacement dans la direction d'un autre attribut. Dans le cas de l'âge par exemple, on voit que celle-ci est entre mêlée avec celle des lunettes. Il est donc intéressant de voir comment on peut déplacer un vecteur latent le long de la direction de l'âge en empêchant le déplacement dans la direction des lunettes.

Pour cela, il faut bien comprendre que si deux hyperplans ne sont pas perpen-

diculaires, faire évoluer le vecteur latent le long de la direction orthogonal à un de ces hyperplans affectera la projection par rapport à l'autre. Dès lors, on comprend qu'en projetant le vecteur normal de l'attribut A1 sur l'hyperplan séparateur du deuxième attribut A2 on aura un mouvement uniquement sur la direction de A1 et non plus selon A2.



Figure 20: Mouvement selon la composante young sans projection (au dessus) Mouvement selon la composante young projeté sur la direction eyeglasses (au dessous)



Figure 21: Mouvement selon la composante mustache (au dessus) Mouvement selon la composante mustache perpendiculairement à la composante beard (au dessous)

**- Mouvement sur la composante young perpendiculairement à la composante eyeglasses:** On remarque ici que l'on parvient à garder les lunettes plus longtemps en allant sur des personnes plus jeunes.

**- Mouvement sur la composante bald perpendiculairement à la composante male:** On remarque ici que l'on parvient à transformer une femme ayant des cheveux en une femme avec moins de cheveux (un front plus imposant). Encore une fois, il y a probablement très peu, voir pas de femme chauve dans le dataset.

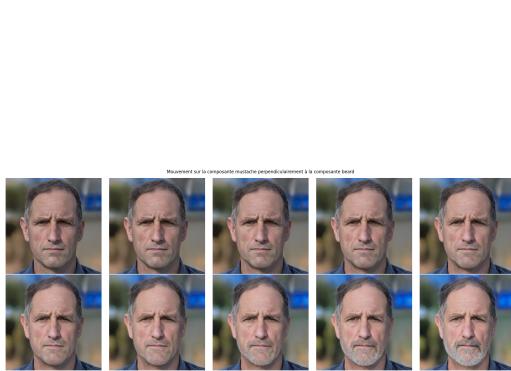


Figure 22: Mouvement sur la composante mustache perpendiculairement à la composante beard

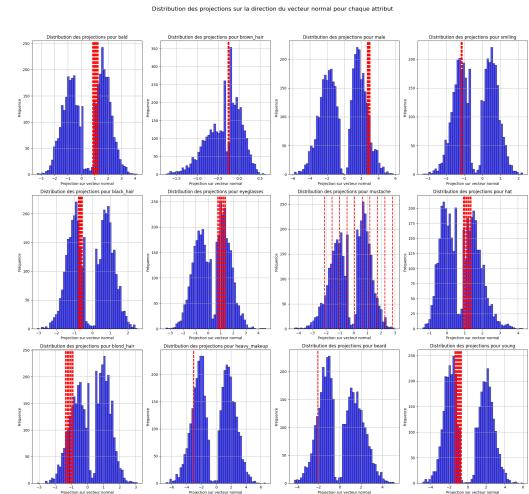


Figure 23: Projections sur le mouvement sur la composante mustache perpendiculairement à la composante beard

On voit ici très bien sur les histogrammes que la projection a permis de ne pas se déplacer dans la direction de l'attribut **beard**. On voit cependant une barbe apparaître, même si plus lentement qu'auparavant. Cela est probablement dû au fait que la direction 'beard' est très proche de la direction 'mustache', et que la séparation du SVM n'est pas parfaite.

Dans cette démarche d'évoluer le long d'un attribut sans pour autant modifier les autres, on peut aussi projeter selon tous les attributs sauf celui dont on cherche à modifier la valeur. On obtient les résultats suivants :

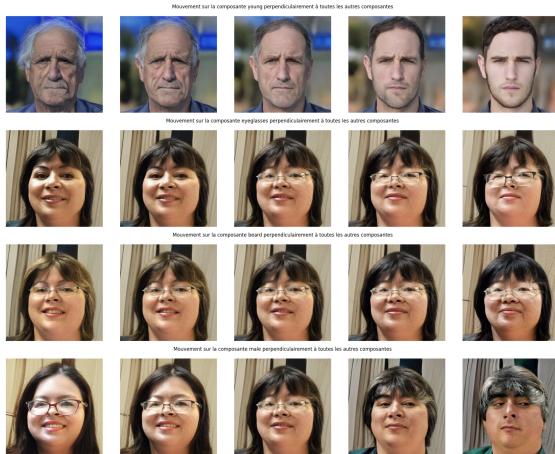


Figure 24: Mouvement le long de différents attributs perpendiculairement à tous les autres

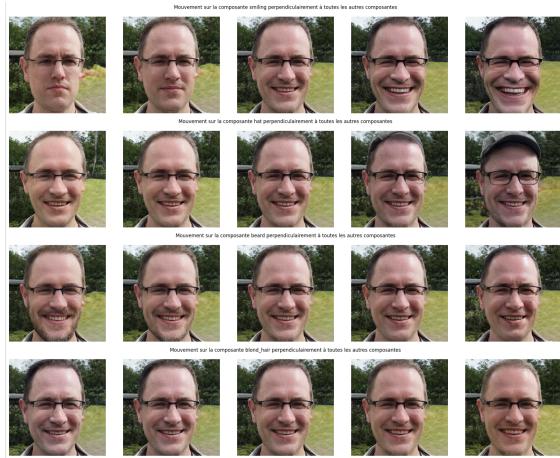


Figure 25: Mouvement le long de différents attributs perpendiculairement à tous les autres

On voit ici qu'on arrive à avoir des mouvements sur la direction de l'attribut visé en limitant au maximum les changements des autres attributs. Les résultats sont plutôt satisfaisant.

### 3.5 Manipulation couche par couche

L'architecture du générateur styleGAN permet d'incorporer des vecteurs de style à plusieurs niveaux de génération de l'image (au niveau des couches de convolution). En général, le même vecteur est envoyé à chaque couche. Il nous a semblé intéressant de modifier le vecteur latent qui serait envoyé dans le générateur seulement sur certaines couches.

En effet, plus on avance à travers les couches plus elles permettent de modifier des détails fins (plus la résolution de l'image générée est élevé). Ainsi, déplacer le vecteur latent selon une certaine direction peut avoir un impact différent selon la profondeur de la couche où nous le modifions.

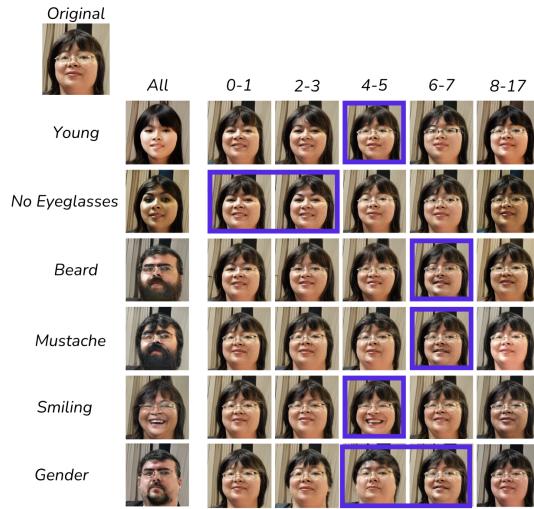


Figure 26: Série d'images montrant le déplacement pour chaque attributs et chaque couches

On observe sur la figure 12 que certaines couches contrôlent certains attributs mieux que d'autres.

Les couches 0 à 3 contrôlent de manière satisfaisante la présence ou non de lunettes. En se déplaçant dans ces couches, seul l'attribut lunette est modifié sans changer l'âge ou le genre.

Les couches 4 et 5 permettent de contrôler l'attribut âge : on réussit à obtenir une jeune fille qui a toujours des lunettes. Les couches 4 à 7 permettent de modifier le genre de la personne.



Figure 27: Série d'images montrant le déplacement sur la composante barbe dans les couches 6 et 7



Figure 28: Série d'images montrant le déplacement sur la composante barbe perpendiculairement à toutes les autres

Les couches plus élevées (de 6 à 17) contrôlent des détails de texture sur l'image. Ceci explique pourquoi les couches 6 à 7 contrôlent la moustache et la barbe. Le résultat pour ces attributs est remarquable puisque l'on réussit à obtenir une femme avec une moustache qui garde ses lunettes ce qui n'était pas possible en projetant simplement la direction moustache perpendiculairement aux autres (figure 13 et 14). On a donc réussi à démêler 2 attributs (genre et moustache) qui sont par définition très entremêlés. Cette méthode permet de modifier plus légèrement les attributs que lorsque l'on se déplace de la même manière sur toutes les couches mais permet de désentrelacer certaines composantes.

### 3.6 Édition d'image

Dans la continuité de ce qui a été réalisé jusque là, il nous semble désormais intéressant d'étudier les capacités d'édition d'image de StyleGAN avec la méthode InterfacGAN. Nous avons donc produit un algorithme capable, à partir d'une image de base, de centrer chacun des attributs sur l'hyperplan séparateur afin d'obtenir le visage "neutre" correspondant, puis de bouger sur chacune des directions voulues afin d'obtenir les attributs désirés.

Par exemple, en choisissant les valeurs suivantes pour chaque attribut :

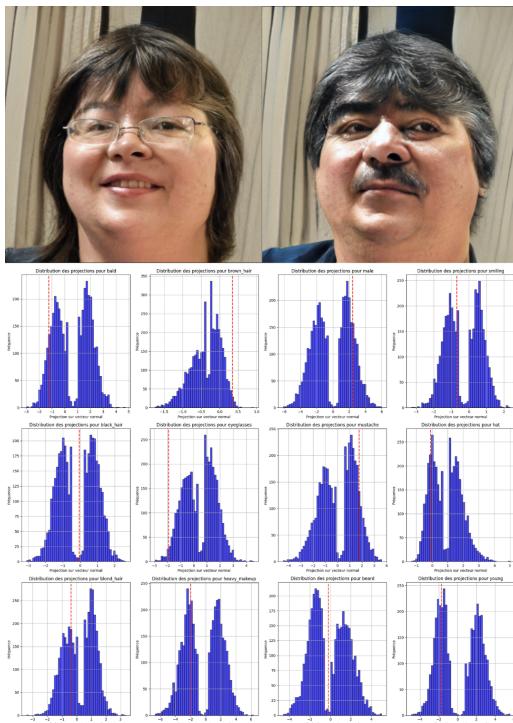


Figure 29: [En haut à gauche] Image originale [En haut à droite] Image éditée [En bas] Projection de l'image éditée sur les histogrammes

Attributs : bald = -3 ; eyeglasses = -3 ; male = 3 ; mustache = 2 ; young = -3

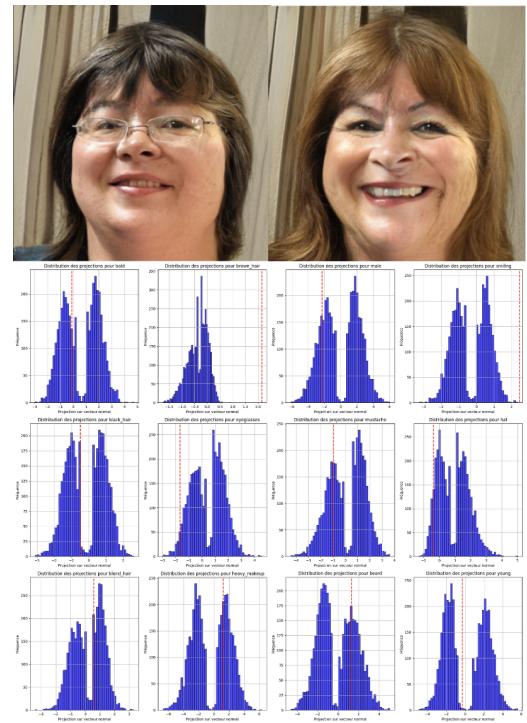


Figure 30: [En haut à gauche] Image originale [En haut à droite] Image éditée [En bas] Projection de l'image éditée sur les histogrammes

Attributs : brown hair = 2 ; eyeglasses = -2 ; male = -2 ; smiling = 2 ; young = -1

## 4 Sources

- [1] T. Karras, S. Laine, T. Aila, A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019, <https://arxiv.org/pdf/1812.04948.pdf>
- [2] Y. Shen, C. Yang, X. Tang, B. Zhou, Interfacegan: Interpreting the disentangled face representation learned by gans. IEEE transactions on pattern analysis and machine intelligence, 2020, 44(4), 2004-2018.
- [3] E. Häkkinen, A. Hertzmann, J. Lehtinen, S. Paris (2020). Ganspace: Discovering interpretable gan controls. Advances in neural information processing systems, 33, 9841-9850.
- [4] W. Wang, D. Yang, F. Chen, Y. Pang, S. Huang and Y. Ge, "Clustering With Orthogonal AutoEncoder," in IEEE Access, vol. 7, pp. 62421-62432, 2019, doi: 10.1109/ACCESS.2019.2916030. keywords: Clustering algorithms;Task analysis;Optimization;Logistics;Entropy;Stochastic processes;Prediction algorithms;Deep clustering;autoencoder;orthogonality,

## 5 Annexes

### 5.1 Annexe 1

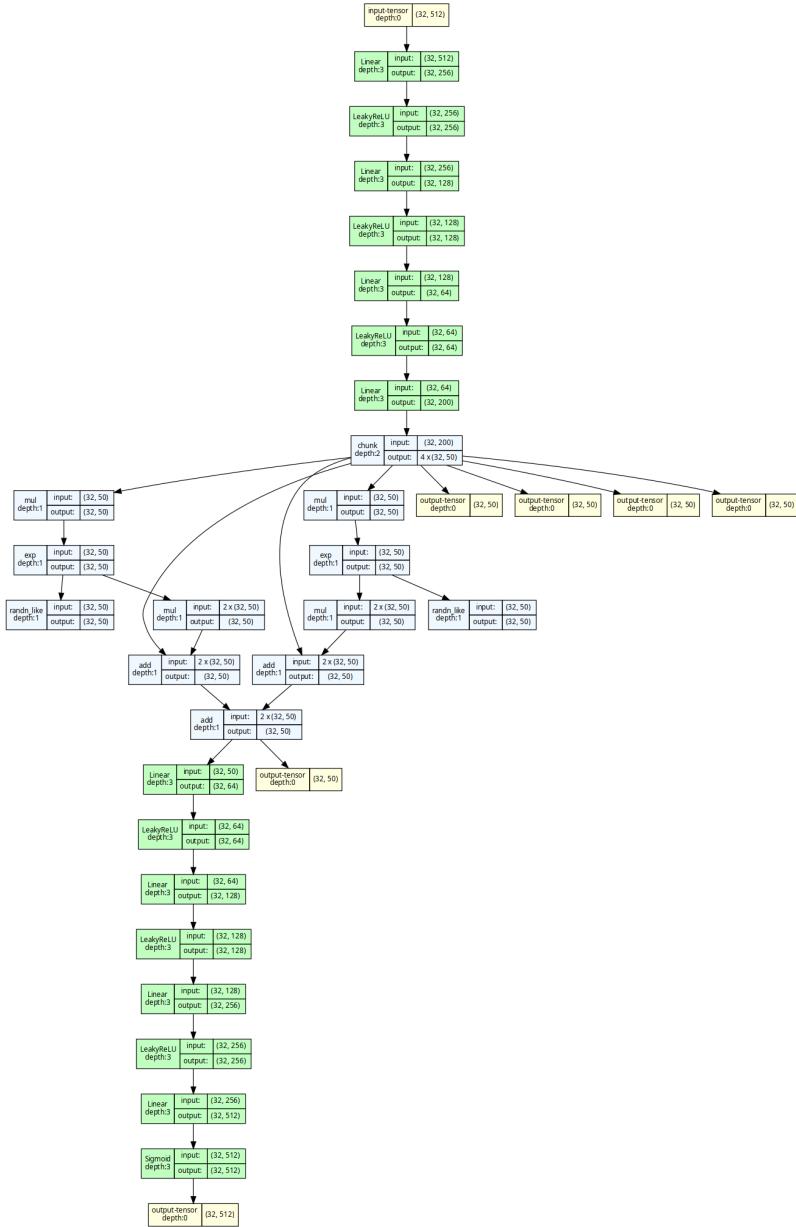


Figure 31: Représentation de l'autoencodeur, la sortie de l'encodeur est de taille  $4 \times$

## 5.2 Annexe 2

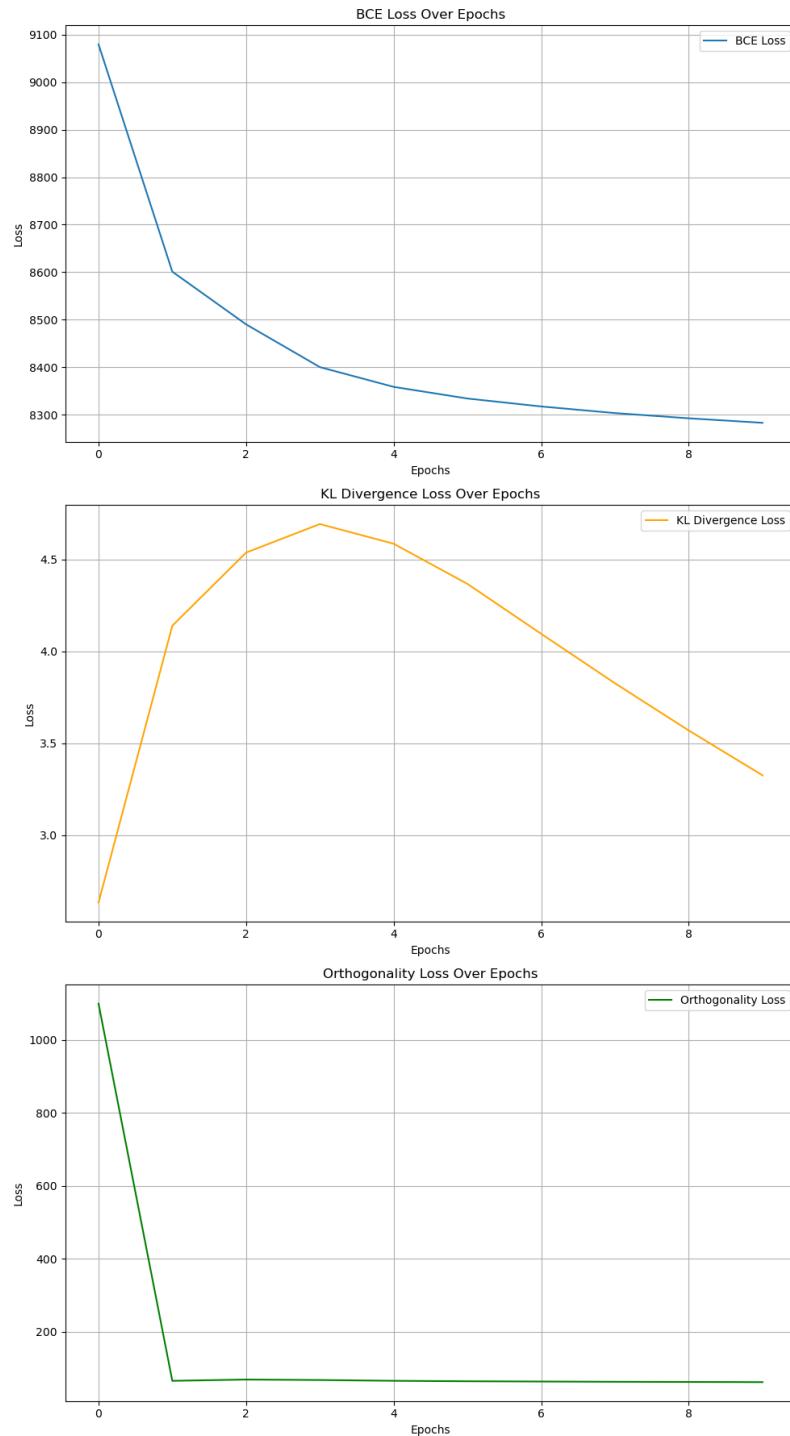


Figure 32: Evolution des différentes Loss au cours des epochs lors du training de l'autoencoder

### 5.3 Annexe 3

Analyse de l'enchevêtrement des attributs : La figure ci-dessous comprend 12 lignes et 12 colonnes. Chaque ligne contient la projection d'une image que l'on a déplacé le long de l'attribut de la ligne, affiché sur l'histogramme de chaque attribut (chaque colonne).

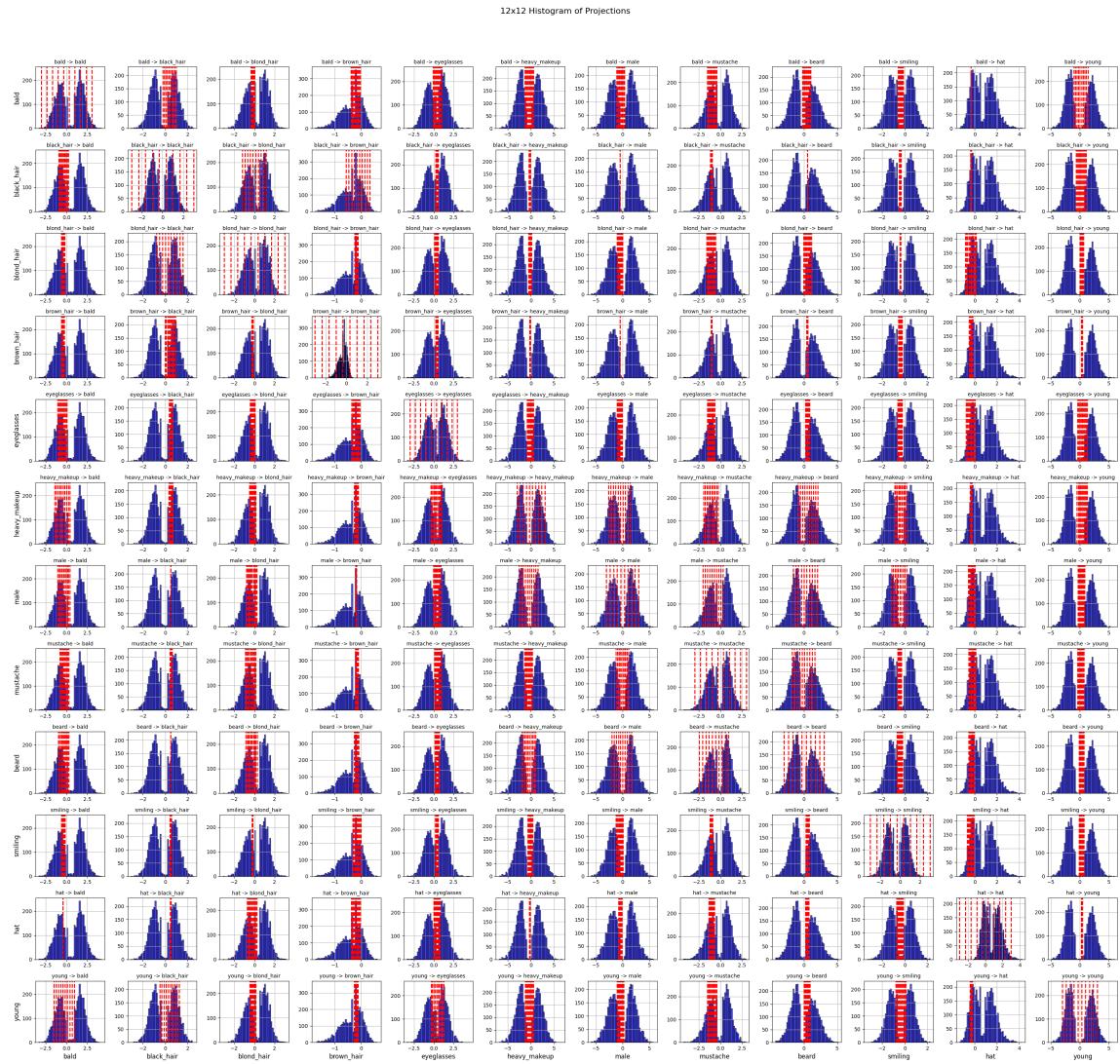


Figure 33: Projections d'images sur les histogrammes de chaque attribut