

# **Edition d'attributs avec des GANs**

Hanna Bekkare, Ines Besbes, Tristan Gay,

Clément Gris & Sara Rool

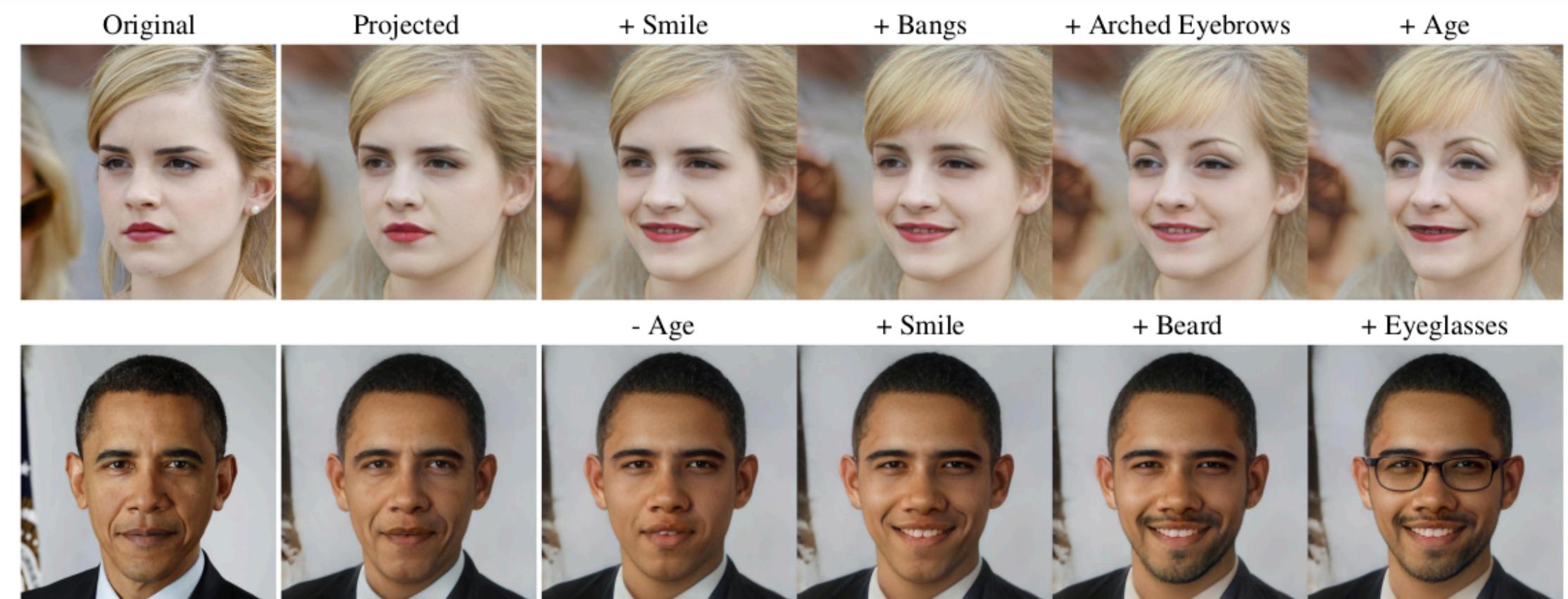
4 ModIA, 2023-2024

# Sommaire

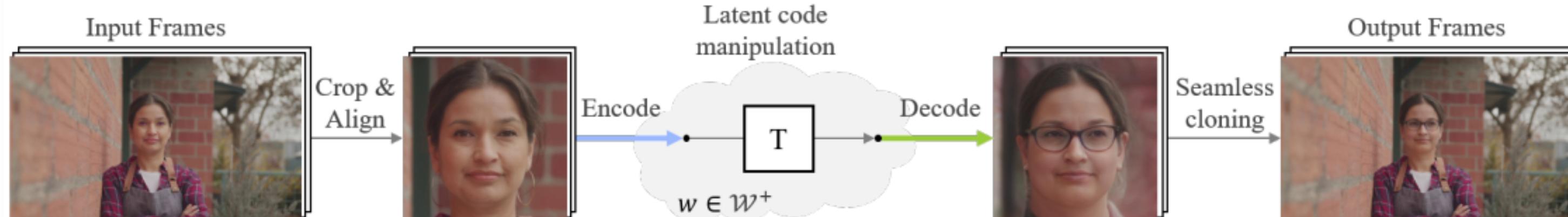
▶ Introduction	03
▶ Structure de l'édition d'attributs	04
▶ Auto-encodeur, GAN et StyleGAN	10
▶ Première modification d'une image	16
▶ Construction d'un classifieur	17
▶ Le transformeur	20
▶ Résultats	24
▶ Conclusion	28

# Introduction

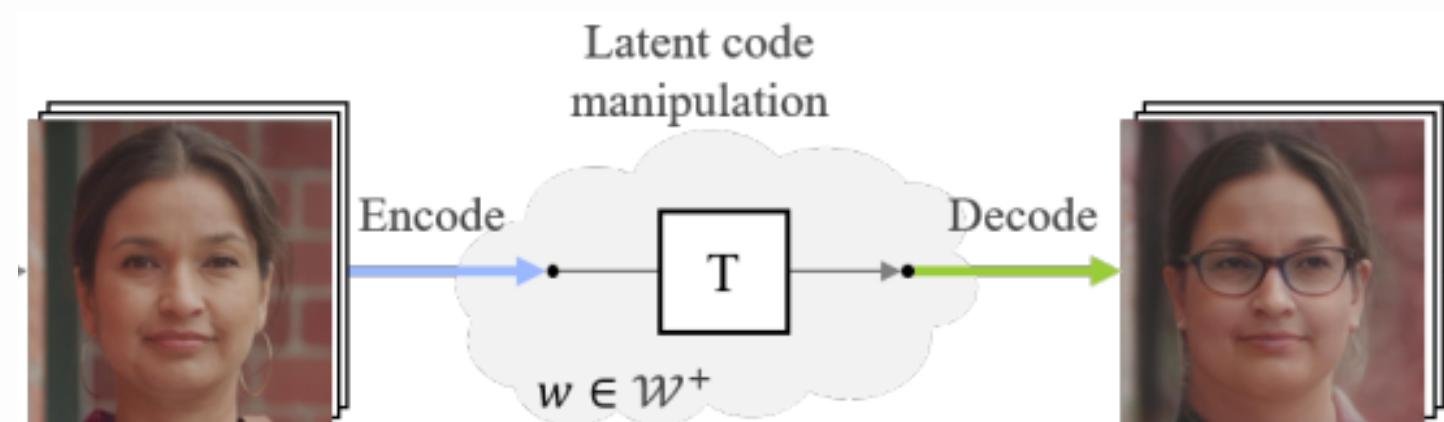
- **Motivation** : retouche photo et la post-production cinématographique
- **Limites actuelles** : fonctionne pour des images à faibles résolution, difficile de modifier un attribut sans affecter les autres ou altérer l'identité, hypothèse de linéarité qui réduit l'efficacité des transformations et performances sur images réelles décevantes
- **Proposition d'amélioration de l'article** : Modifications distinctes et contrôlées des attributs faciaux, fonctionne efficacement sur des images et vidéos réelles, qualité et stabilité pour des images de haute résolution



# Structure de l'édition d'attributs (vidéo)



# Structure de l'édition d'attributs (vidéo)



# Structure de l'édition d'attributs



**Définition :** Un espace latent est une représentation compressée des données d'entrée. Plutôt que de représenter les données de manière brute, on les transforme en un ensemble de variables (ou dimensions) qui capturent les caractéristiques essentielles des données de manière plus abstraite et compacte.

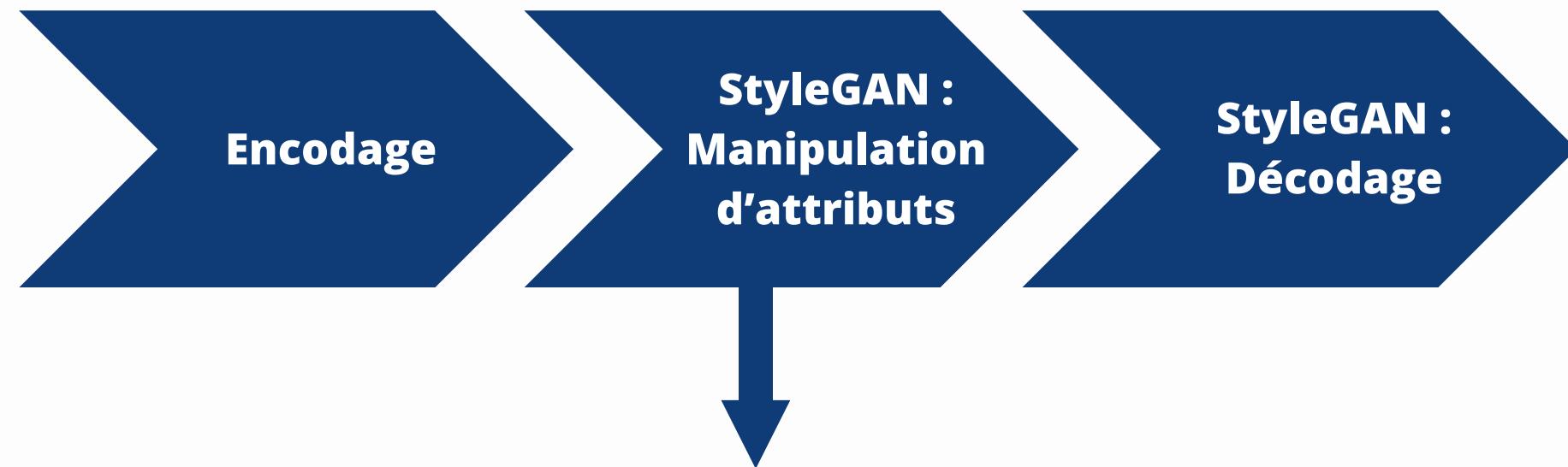
# Structure de l'édition d'attributs



- Transformation d'une image en un vecteur latent dans un espace  $\mathbf{Z}$
- Les dimensions de l'espace latent représentent les caractéristiques de l'image
- Un StyleGAN a un encodeur unique
- L'encodage aide à préserver les détails importants de l'image
- Encodage déjà effectué dans notre cas

**Définition :** Un espace latent est une représentation compressée des données d'entrée. Plutôt que de représenter les données de manière brute, on les transforme en un ensemble de variables (ou dimensions) qui capturent les caractéristiques essentielles des données de manière plus abstraite et compacte.

# Structure de l'édition d'attributs



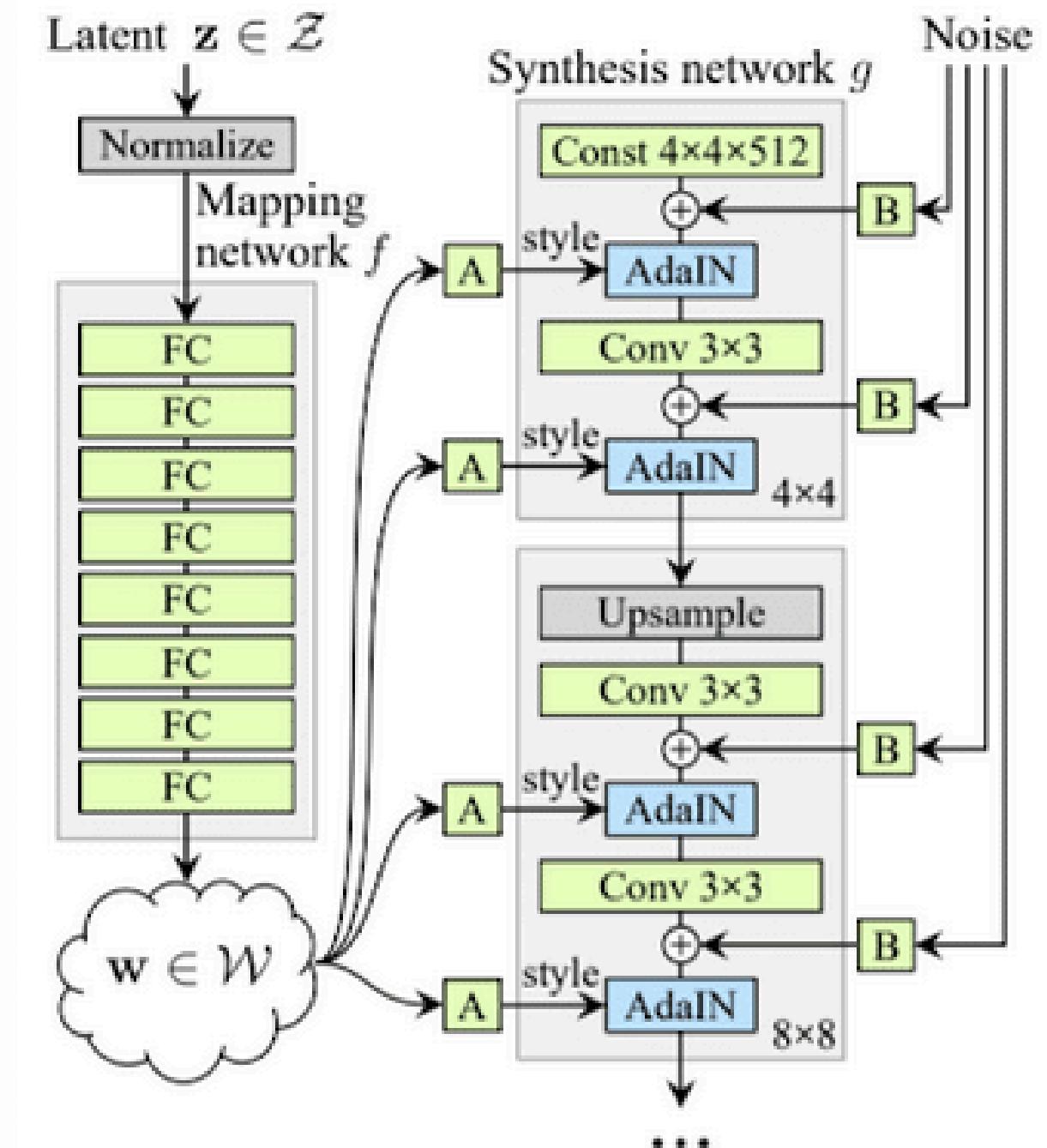
- Manipulation des attributs dans un nouvel espace latent  $\mathbf{W}$  (1 direction = 1 attribut)
- L'espace  $\mathbf{W}$  est une transformation non linéaire de l'espace latent  $\mathbf{Z}$
- Contrôle plus fin des attributs grâce à la non linéarité
- Optimiser la génération d'images de haute qualité

**Définition :** Un StyleGAN contrairement à un GAN classique permet le contrôle et la manipulation du style des images générées, comme les détails des textures, la pose et les attributs spécifiques

# Structure de l'édition d'attributs



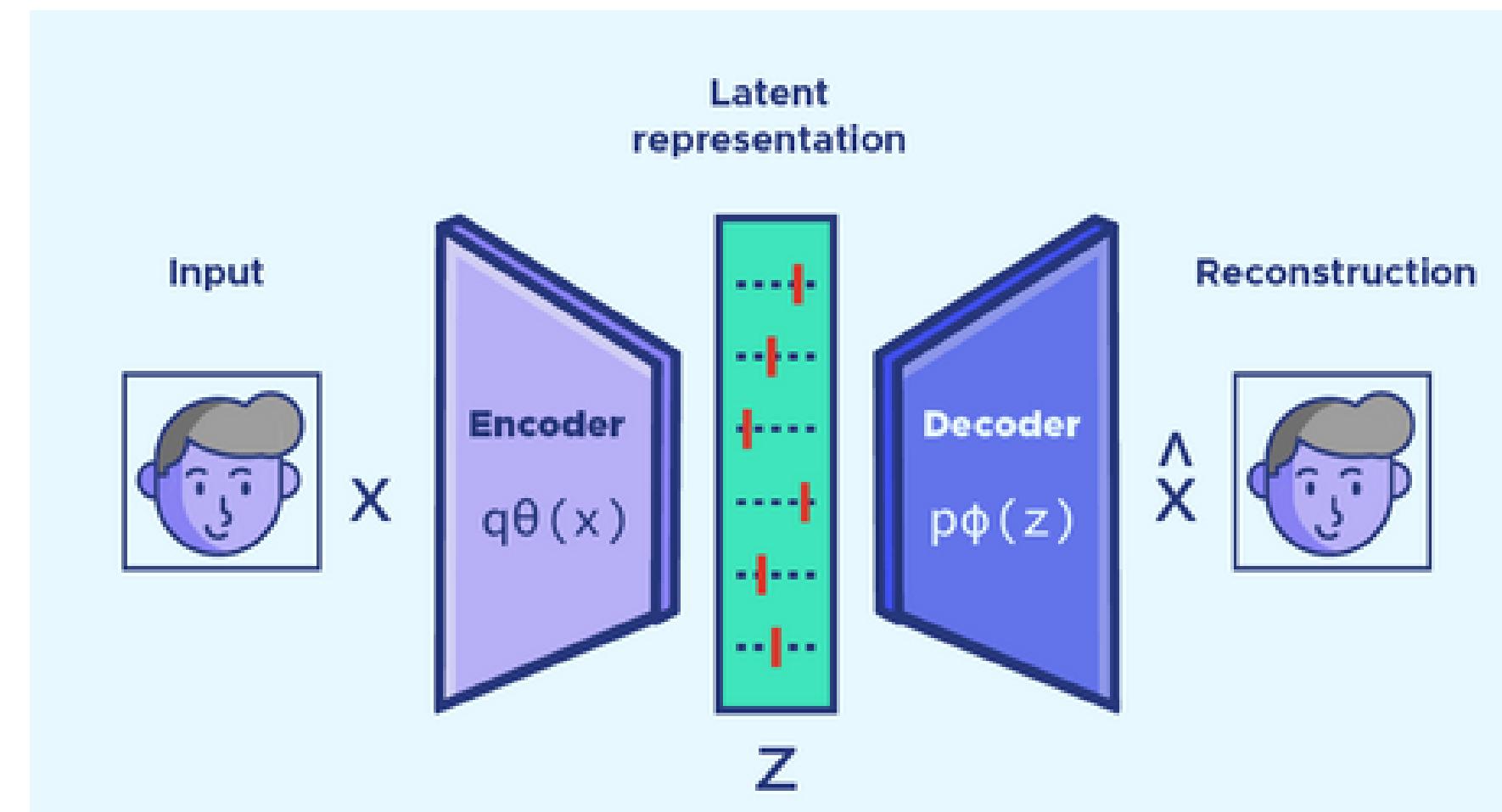
- Etape comprise dans le StyleGAN dans notre structure
- Le vecteur latent modifié  $w$  est utilisé comme entrée
- Génération d'une nouvelle image directement



# Auto-encodeurs

## Deux parties principales

- *Encodeur*
  - Comprime l'entrée en une représentation de dimension inférieure (espace latent)
- *Décodeur*
  - Reconstruit l'entrée à partir de la représentation latente

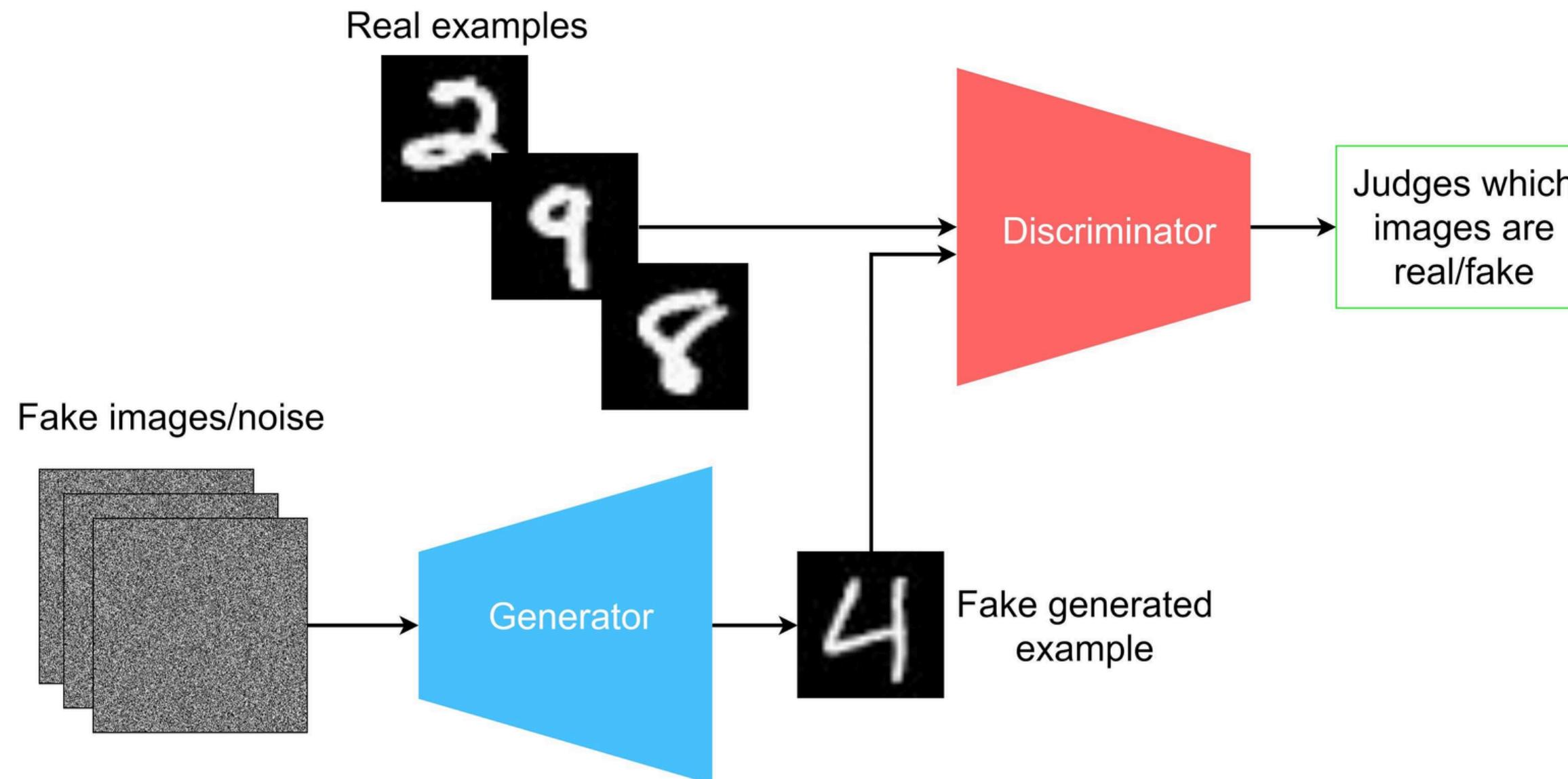


*Schéma inspiré d'un cours de Fabien Moutarde - Mines de Paris (2021)*

## Processus d'apprentissage

- L'*encodeur* apprend à conserver les informations essentielles
- Le *décodeur* utilise ces informations pour reconstruire fidèlement l'entrée

# GAN (Generative Adversarial Networks)



- **Générateur :**

- Crée des données similaires à des données réelles à partir de bruits aléatoires

- **Discriminateur :**

- Évalue la plausibilité des données, distinguant les vraies données des fausses

# Applications des GANs - en médecine

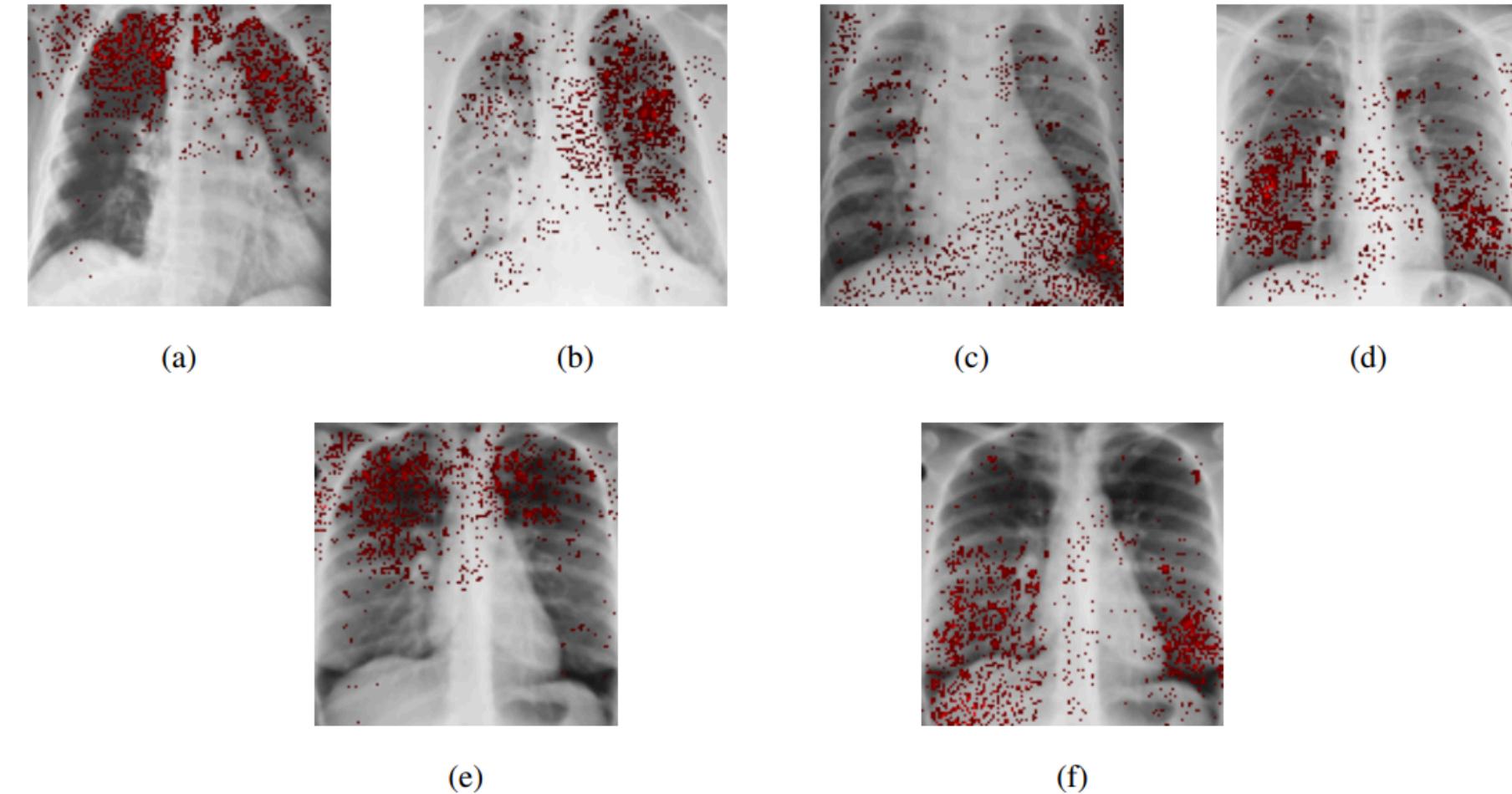
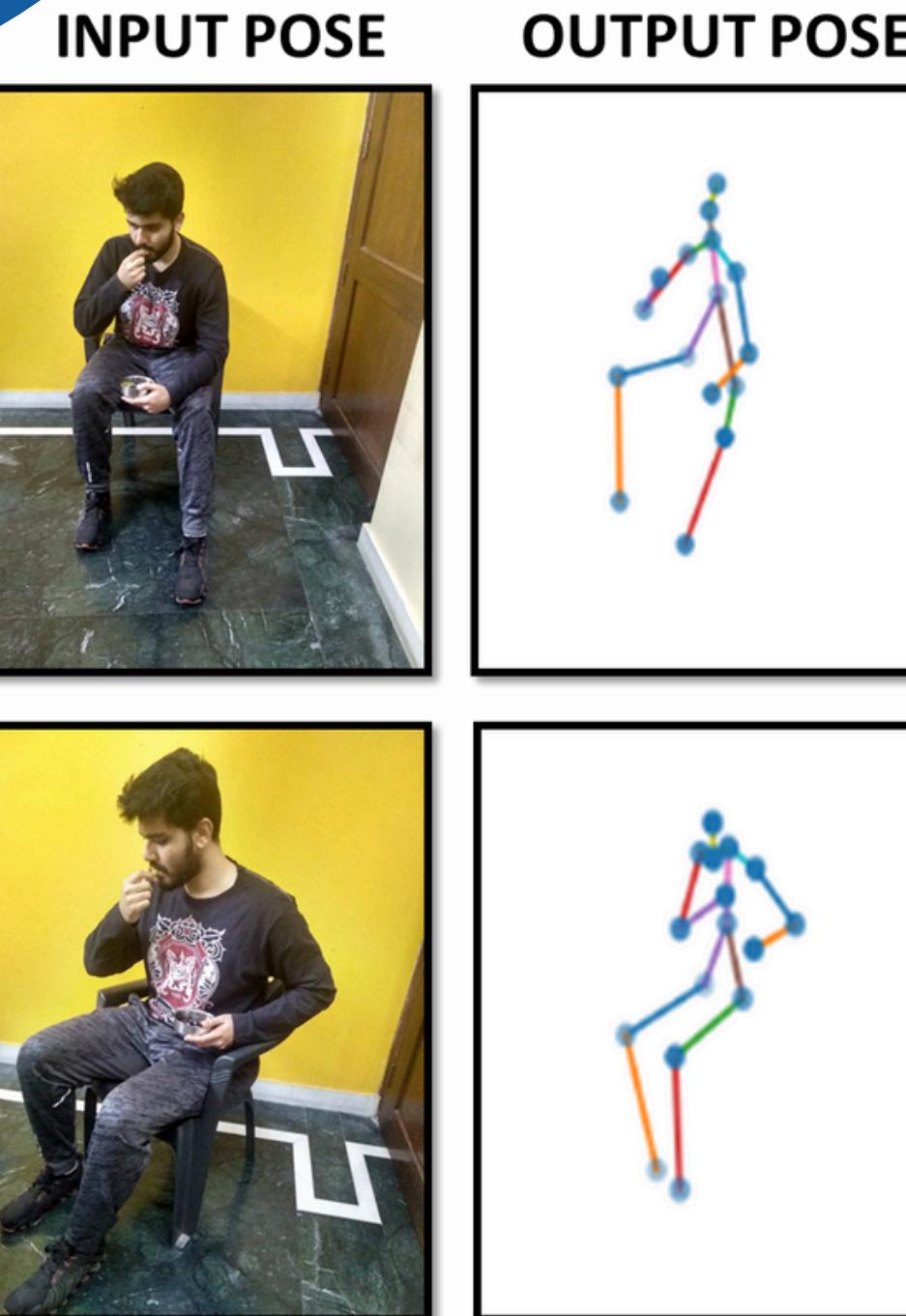
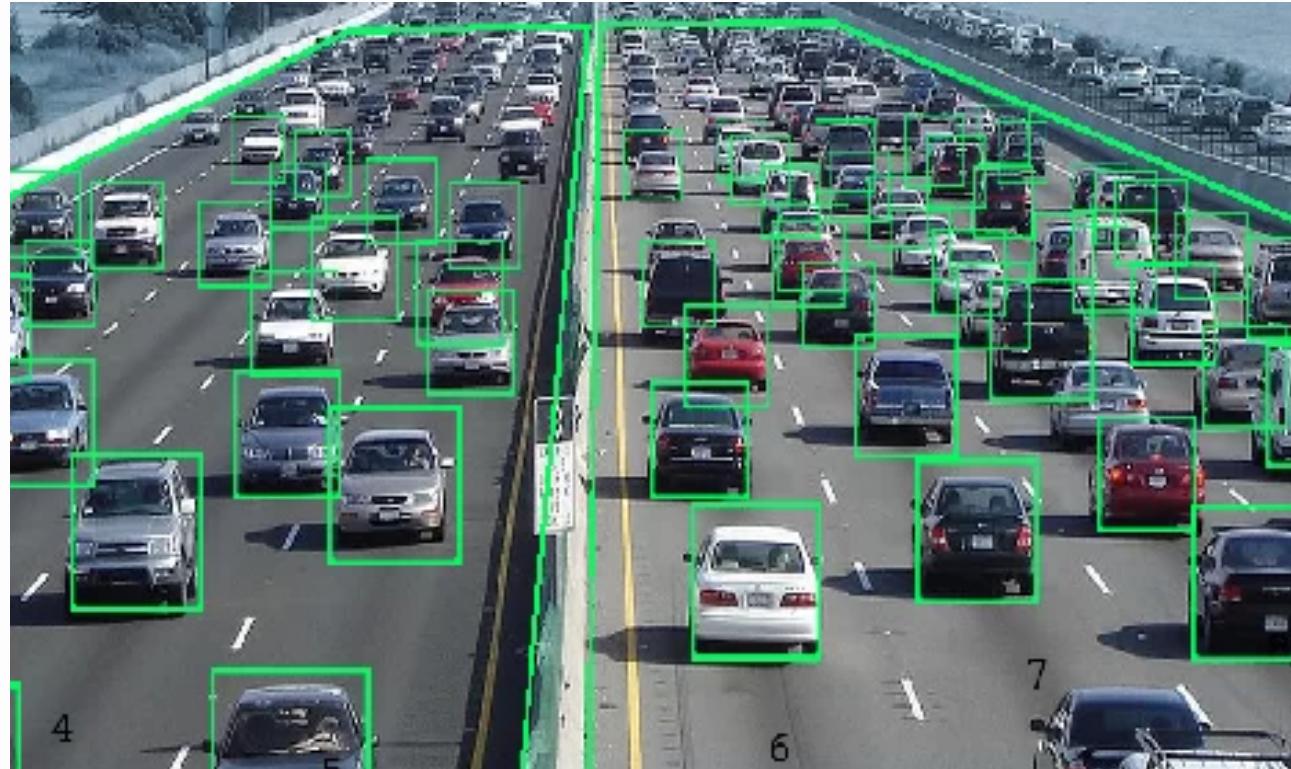


FIG. 12. Top row - examples of the image-specific saliency maps of the predicted class on the correctly classified images: a) covid, b) lung opacity, c) viral pneumonia, d) healthy lungs. Bottom row - example of saliency maps for incorrectly classified COVID-19 positive image as a healthy one: a) saliency map for COVID-19 class prediction, b) saliency map for healthy class prediction.

**GAN Poser: la prédiction des mouvements humains à partir d'une image 3D du squelette humain**  
<https://github.com/abhishek-kathuria/GAN-Poser?tab=readme-ov-file>

**Performance of GAN-based Augmentation for Deep Learning COVID-19 Image Classification**  
<https://arxiv.org/abs/2304.09067>

# Autres applications des GANs



**GE-GAN : traffic road estimation framework using deep learning where dual street systems of two cities are been used as a case study**

<https://www.sciencedirect.com/science/article/abs/pii/S0968090X19312409>

[https://github.com/wcc961129/GE-GAN/tree/main/GE\\_GAN](https://github.com/wcc961129/GE-GAN/tree/main/GE_GAN)

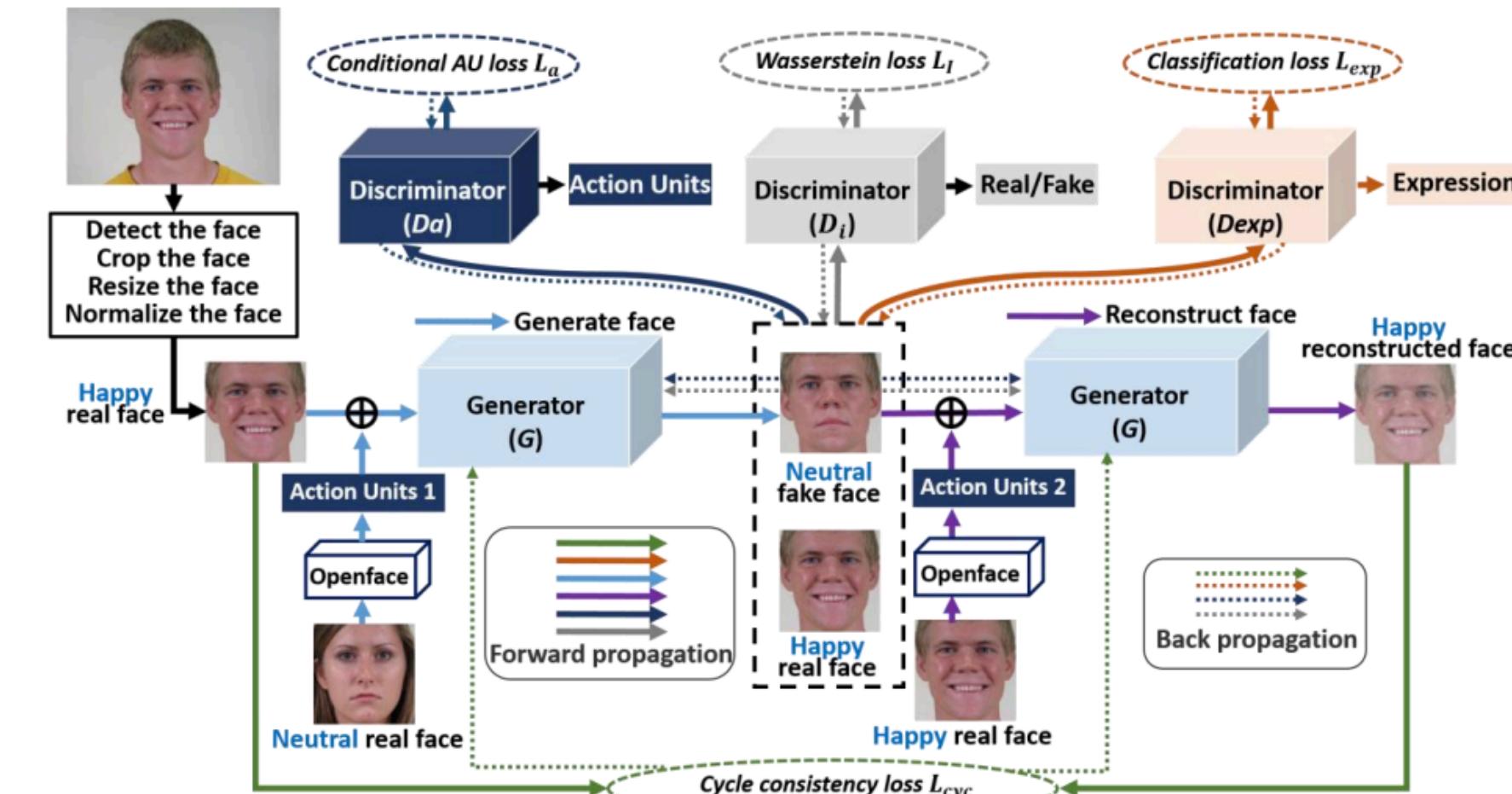


FIGURE 3. The overall architecture of the proposed model, which consists of a generator G and three discriminators ( $D_l$ ,  $D_a$ , and  $D_{exp}$ ). The generator G is applied twice: G transforms any given query face image to another facial expression image and then renders it back. Openface [53] is applied to extract the AUs.

**CGAN : Face detection applications**

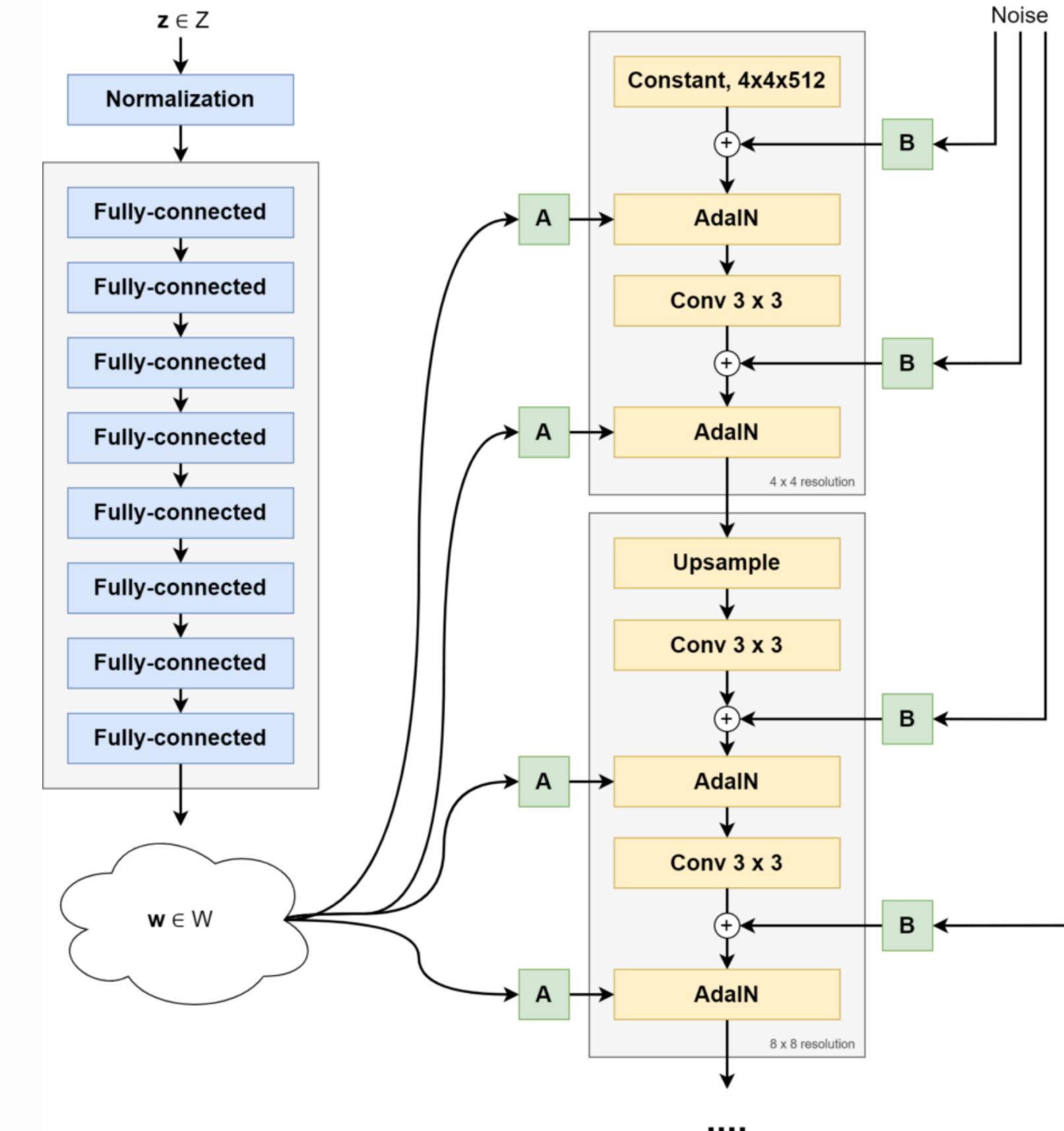
[https://www.researchgate.net/publication/330294861\\_CGAN\\_Based\\_Facial\\_Expression\\_Recognition\\_for\\_Human-Robot\\_Interaction](https://www.researchgate.net/publication/330294861_CGAN_Based_Facial_Expression_Recognition_for_Human-Robot_Interaction)

# StyleGAN

Generate a person : <https://thispersondoesnotexist.com/>

- Utilisation d'un réseau de mapping pour transformer le bruit d'entrée en un vecteur latent  $w$ .
- Générateur composé de couches de convolutions et de couches Adaptive Instance Normalization (AdaIN) modulées par le vecteur  $w$ .
- Convolutions 3x3 pour traiter les données à différentes échelles.
- Ajout de bruit gaussien à chaque étape pour améliorer la variabilité et la réalisme des images.

$$AdaIN(x_i, y) = y_{s,i} \frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$$



# Première modification d'une image

```
### nous avons comme data les codes latents, on en prends un aléatoirement  
z = torch.from_numpy(np.random.RandomState(seed).randn(1, G.z_dim)).to(device)  
### on reconstruit l'image originale  
img = G(z, label, truncation_psi=truncation_psi, noise_mode=noise_mode)  
### on map le vecteur latent z dans W : l'espace Z transformé non linéairement  
w = G.mapping(z,label)  
### on modifie w aléatoirement pour modifier l'image  
w = w + torch.randn(1,18,512).to(device)  
### on reconstruit l'image modifiée à partir de w  
img_mod = G.synthesis(w, noise_mode=noise_mode)
```

Image Originale



Image Modifiée



Image Modifiée



# Les différents attributs

Attribut 0: {0: 25509, 1: 4491}  
Attribut 1: {0: 18980, 1: 11020}  
Attribut 2: {0: 12782, 1: 17218}  
Attribut 3: {0: 21366, 1: 8634}  
Attribut 4: {0: 29288, 1: 712}  
Attribut 5: {0: 24575, 1: 5425}  
Attribut 6: {0: 19110, 1: 10890}  
...  
Attribut 32: {0: 23556, 1: 6444}  
Attribut 33: {0: 19277, 1: 10723}  
Attribut 34: {0: 22056, 1: 7944}  
Attribut 35: {0: 28930, 1: 1070}  
Attribut 36: {0: 13141, 1: 16859}  
Attribut 37: {0: 24915, 1: 5085}  
Attribut 38: {0: 27838, 1: 2162}  
Attribut 39: {0: 6632, 1: 23368}

```
attr_dict = {'5_o_Clock_Shadow': 0, 'Arched_Eyebrows': 1, 'Attractive': 2, 'Bags_Under_Eyes': 3, 'Bald': 4, 'Bangs': 5, 'Big_Lips': 6, 'Big_Nose': 7, 'Black_Hair': 8, 'Blond_Hair': 9, 'Blurry': 10, 'Brown_Hair': 11, 'Bushy_Eyebrows': 12, 'Chubby': 13, 'Double_Chin': 14, 'Eyeglasses': 15, 'Goatee': 16, 'Gray_Hair': 17, 'Heavy_Makeup': 18, 'High_Cheekbones': 19, 'Male': 20, 'Mouth_Slightly_Open': 21, 'Mustache': 22, 'Narrow_Eyes': 23, 'No_Beard': 24, 'Oval_Face': 25, 'Pale_Skin': 26, 'Pointy_Nose': 27, 'Receding_Hairline': 28, 'Rosy_Cheeks': 29, 'Sideburns': 30, 'Smiling': 31, 'Straight_Hair': 32, 'Wavy_Hair': 33, 'Wearing_Earrings': 34, 'Wearing_Hat': 35, 'Wearing_Lipstick': 36, 'Wearing_Necklace': 37, 'Wearing_Necktie': 38, 'Young': 39}
```

*Dictionnaire des attributs*

- Correspondance entre indice et attributs
- Répartition inégale des différents attributs
- Choix d'un premier attribut

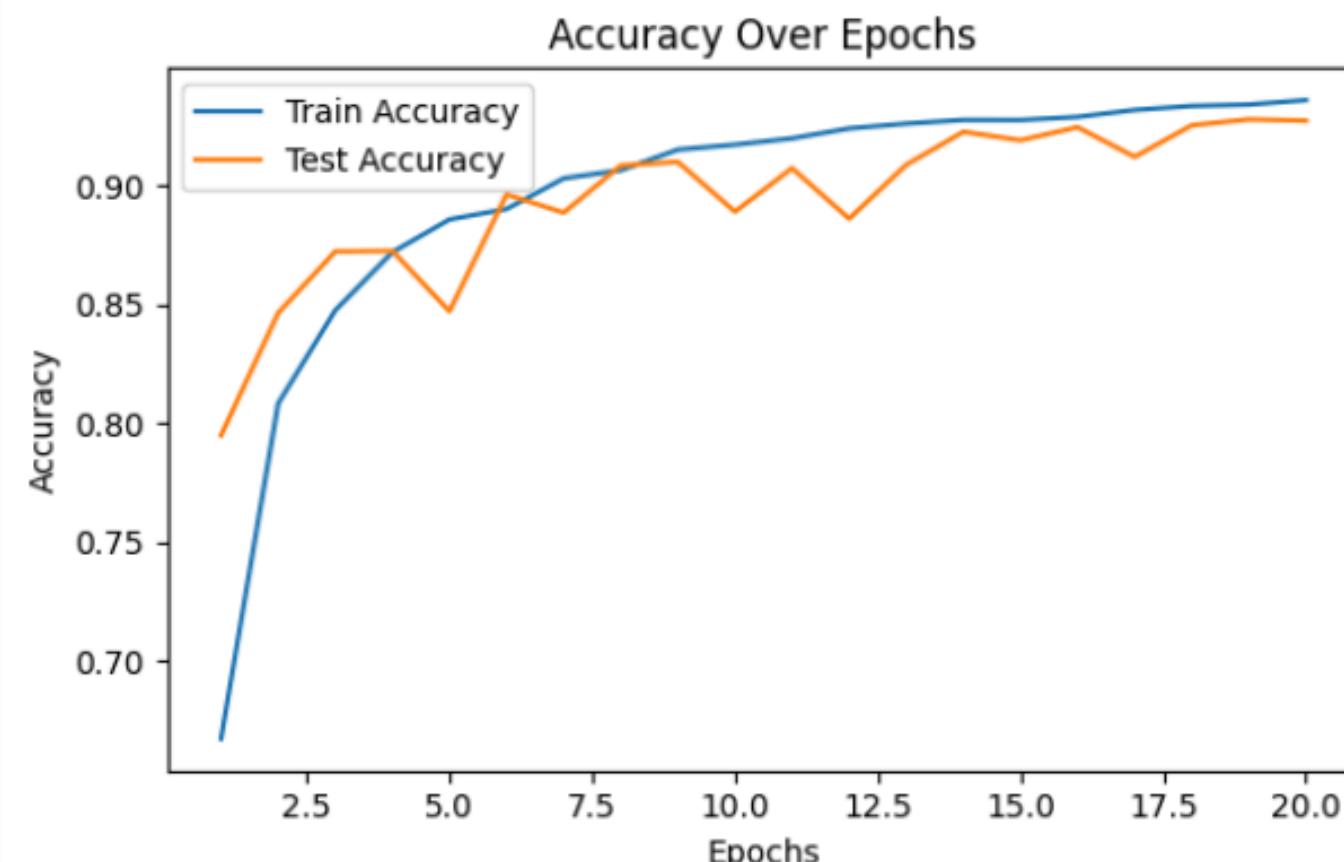
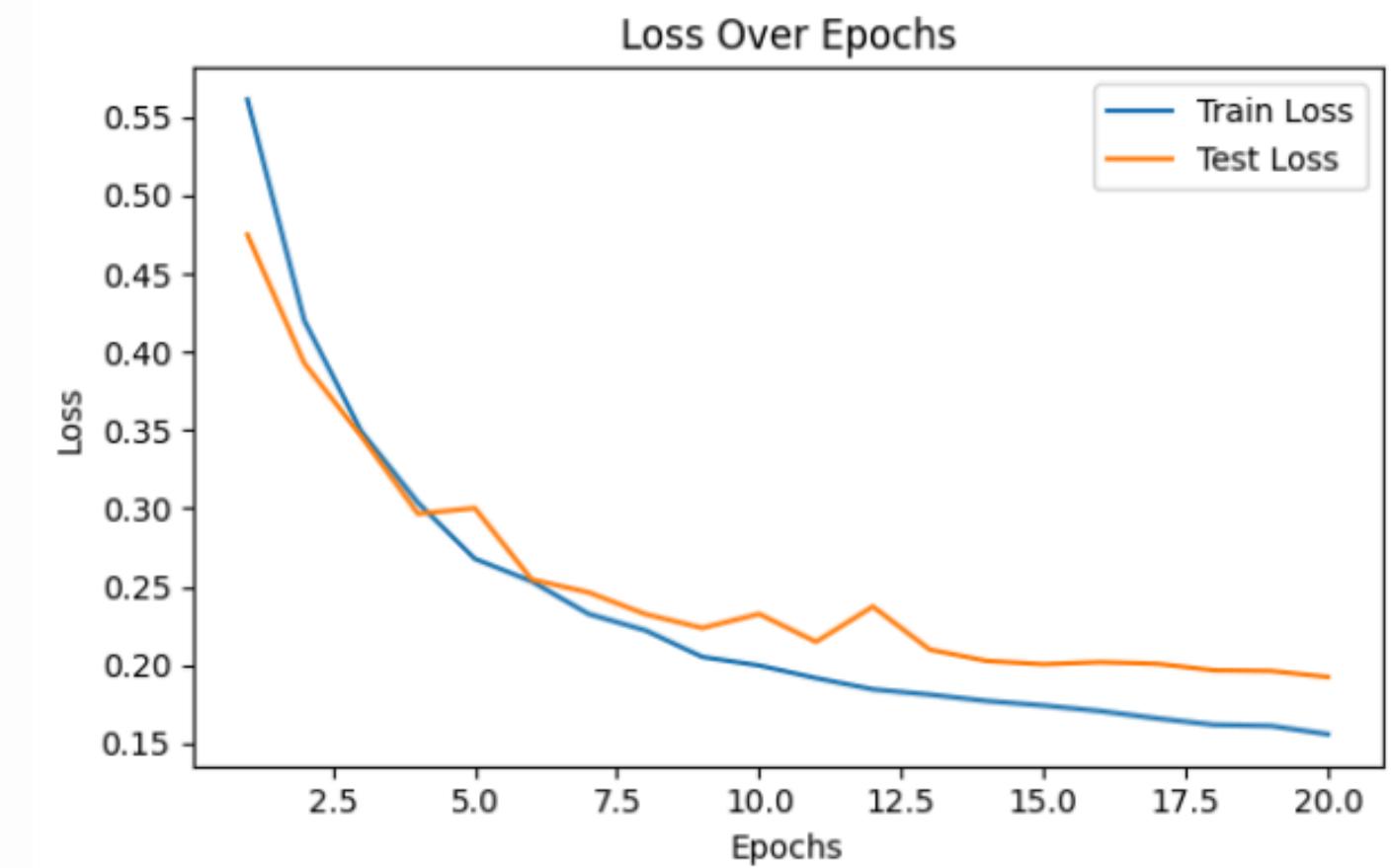
*Valeurs des attributs dans les codes latents W*

# Construction d'un classifieur : pour un attribut

```
class NNClassificationSimple(torch.nn.Module):
    def __init__(self):
        super(NNClassificationSimple, self).__init__()
        self.network = torch.nn.Sequential(
            torch.nn.Linear(9216, 1024),
            torch.nn.ReLU(),
            torch.nn.Linear(1024, 256),
            torch.nn.ReLU(),
            torch.nn.Linear(256, 1),
        )

    def forward(self, xb):
        return self.network(xb)
```

- Estime la probabilité que l'image contienne un certain attribut (la proba est combiné avec la loss : BCEWithLogitsLoss)
- Convergence de la loss vers 0.17 et accuracy de 0.92

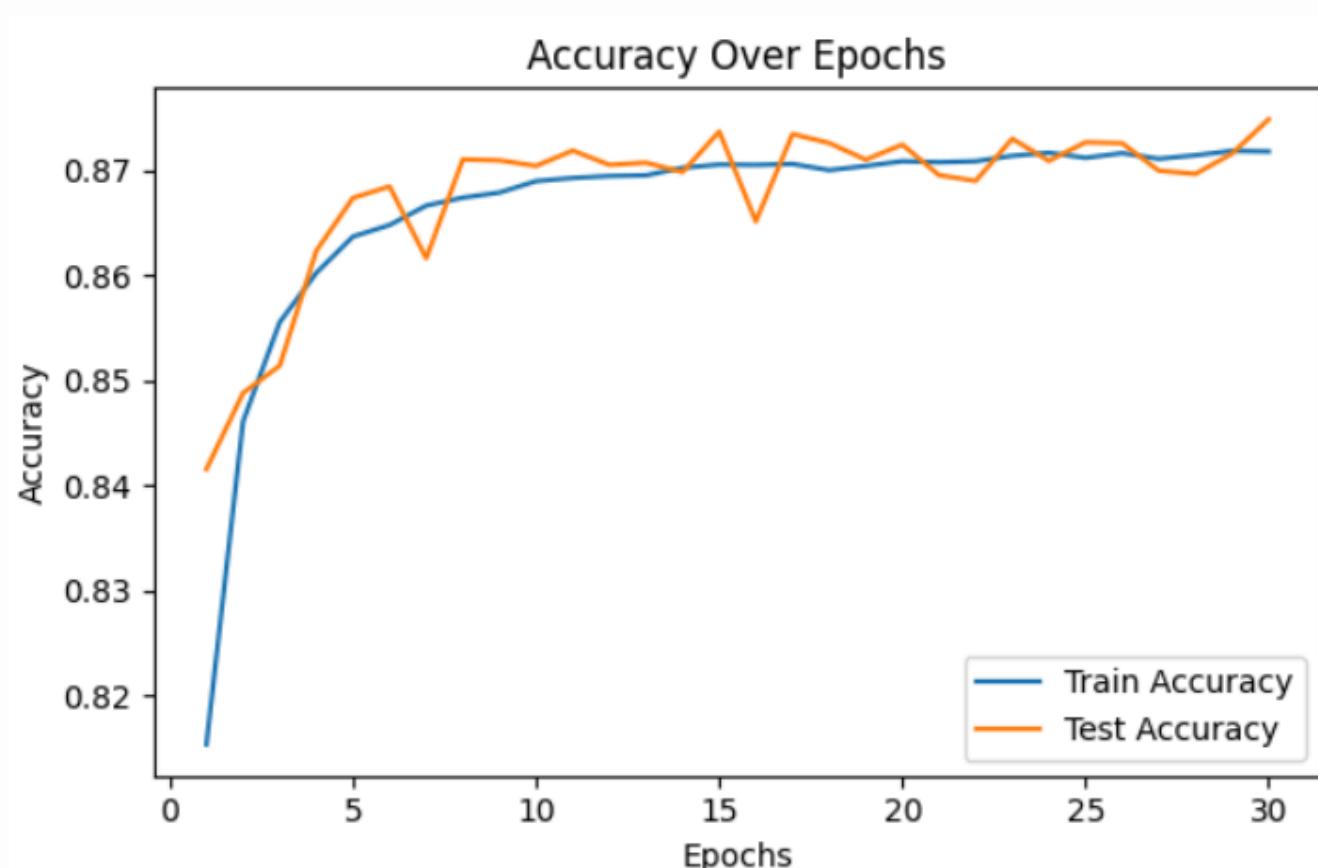
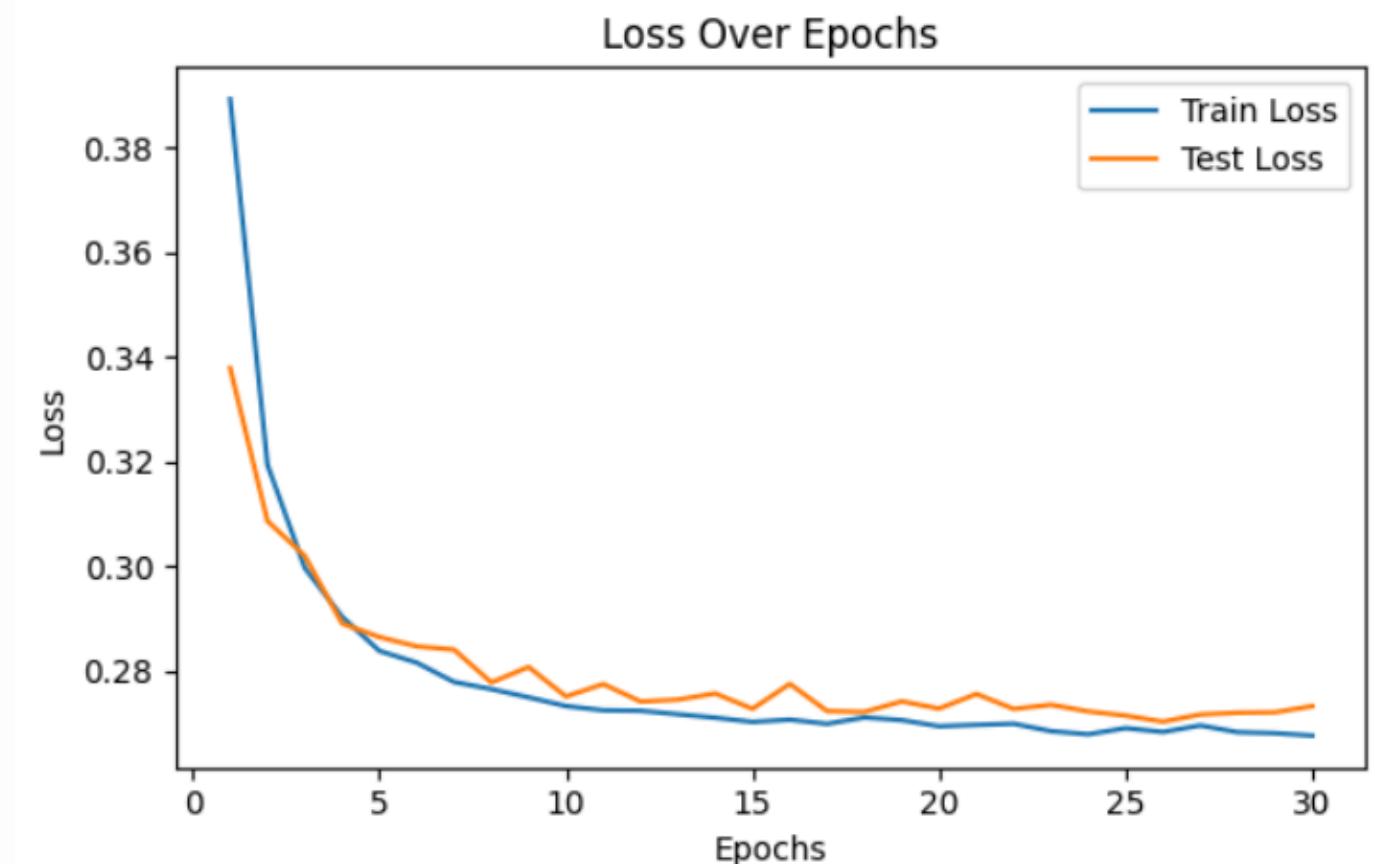


# Construction d'un classifieur pour les 40 attributs

```
class NNClassification(torch.nn.Module):
    def __init__(self):
        super(NNClassification, self).__init__()
        self.network = torch.nn.Sequential(
            torch.nn.Linear(9216, 1024),
            torch.nn.ReLU(),
            torch.nn.Linear(1024, 256),
            torch.nn.ReLU(),
            torch.nn.Linear(256, 40),
        )

    def forward(self, xb):
        return self.network(xb)
```

- Convergence de la loss vers 0.27 et de l'accuracy vers 0.87
- Résultats légèrement inférieur que pour 1 attribut mais très bon quand même
- Dans les deux cas, ajout de régularisation pour éviter l'overfitting



# Transformeur

- $G \rightarrow$  notre générateur et  $w$  un code latent de  $W$
- $T$  transformeur latent  $\rightarrow$  éditer un seul attribut de l'image projetée  $G(w)$
- L'image modifiée devient  $G(T(w))$
- Un transformeur par attribut  $\rightarrow Tk$
- Entrainement avec une somme de 3 loss
- $\alpha \rightarrow$  contrôle le montant des modifications
- $T$  est un réseau avec une seule couche linéaire

$$T(w, \alpha) = w + \alpha \cdot f(w)$$

Train

$$\alpha = \begin{cases} 1 - p & \text{pour } p < 0.5, \\ -p & \text{pour } p > 0.5. \end{cases}$$

Test

$$\alpha \in [-1, 1]$$

# Transformeur

## Implémentation de la classe Transformeur :

```
class Transform(torch.nn.Module):
    def __init__(self, input_size, output_size,attribute):
        super(Transform, self).__init__()
        self.weight = torch.nn.Parameter(torch.Tensor(output_size, input_size))
        self.bias = torch.nn.Parameter(torch.Tensor(output_size))
        self.k=attribute
        torch.nn.init.xavier_uniform_(self.weight)
        torch.nn.init.zeros_(self.bias)

    def forward(self, x, alpha):
        weight=self.weight
        fx = F.linear(x, weight, bias=self.bias)
        return x+alpha*fx
```

# Transformeur : les loss functions

$$\mathcal{L}_{\text{cls}} = -y_k \log (\mathbf{p}_k) - (1 - y_k) \log (1 - \mathbf{p}_k)$$

Loss de classification binaire pour l'attribut  $k$  souhaité

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{\mathbf{w}}[||\mathbf{T}(\mathbf{w}) - \mathbf{w}||_2]$$

Terme de régularisation pour préserver l'identité générale de la personne

$$\mathcal{L}_{\text{attr}} = \sum_{i \neq k} (1 - \gamma_{ik}) \mathbb{E}_{\mathbf{w}, i}[||\mathbf{p}_i - \mathbf{C}(\mathbf{w})[i]||_2],$$

Terme de régularisation pour pas modifier les autres attributs, prends en compte la corrélation entre attribut  
Ex : "chubby" et "double chin"

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \lambda_{\text{attr}} \mathcal{L}_{\text{attr}} + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}},$$

Somme des 3 loss pondérées par des poids

# Transformeur : les loss functions

## Implémentation de la fonction loss :

```
def calcul_loss(k,gamma_ik,initial_attributes,alphas, initial_latent_codes, transformed_latent_codes,transformed_attributes):  
    # Paramètres pour la perte  
    lambda_attr = 1  
    lambda_rec = 10  
  
    attr_k=initial_attributes[:,k]  
    pk=transformed_attributes[:, k]  
  
    # L_cls  
    target_pb = torch.clamp(attr_k + alphas, 0, 1).round()  
  
    cls_loss = nn.BCEWithLogitsLoss()(pk, target_pb)|  
  
    cls_loss = cls_loss.mean()  
    # L_attr  
    mask=gamma_ik.repeat(transformed_attributes.size(0),1)  
    attr_loss = nn.MSELoss()(transformed_attributes*mask,initial_attributes*mask)  
  
    # L_rec  
    rec_loss = nn.MSELoss()(transformed_latent_codes,initial_latent_codes)  
    # Total loss  
    total_loss = cls_loss + lambda_attr * attr_loss + lambda_rec * rec_loss  
    return total_loss
```

# Résultats



**Image Original**

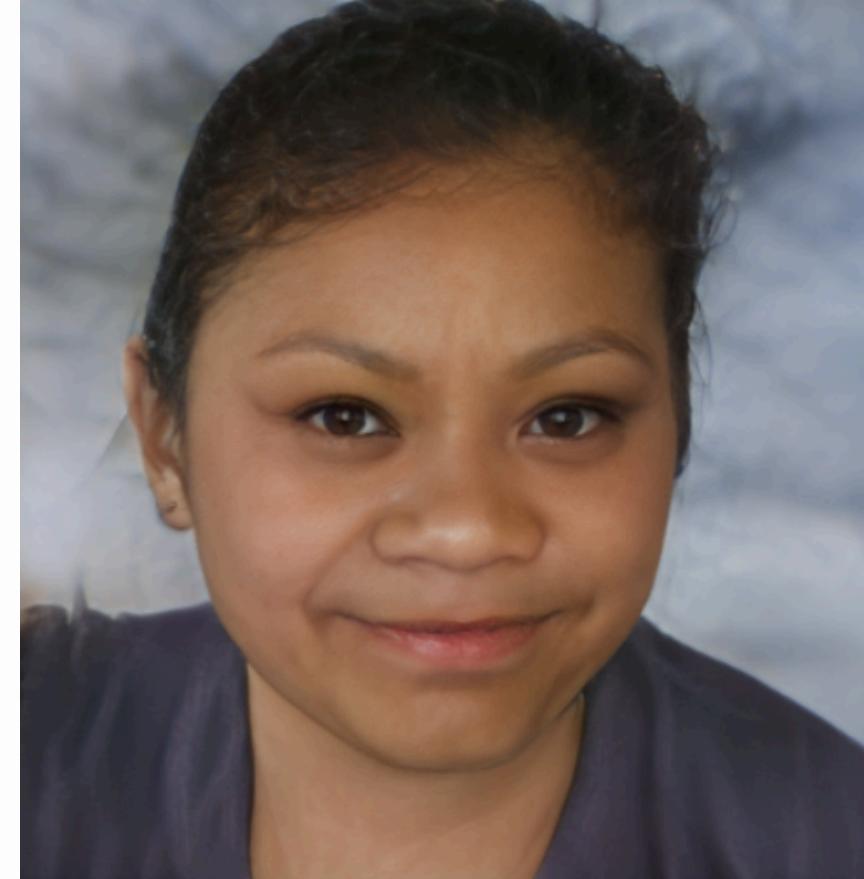
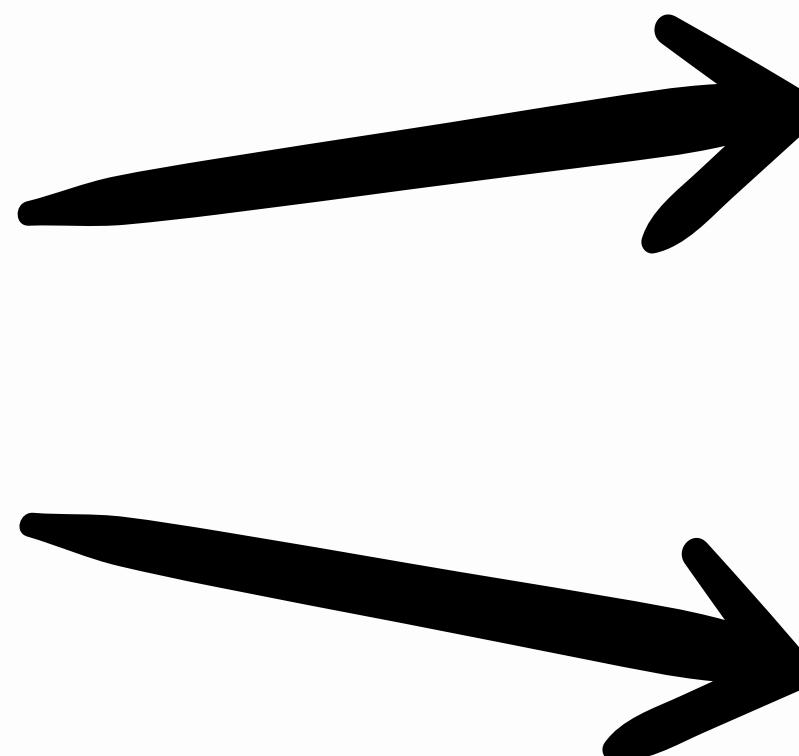


**Projection dans l'espace latent**

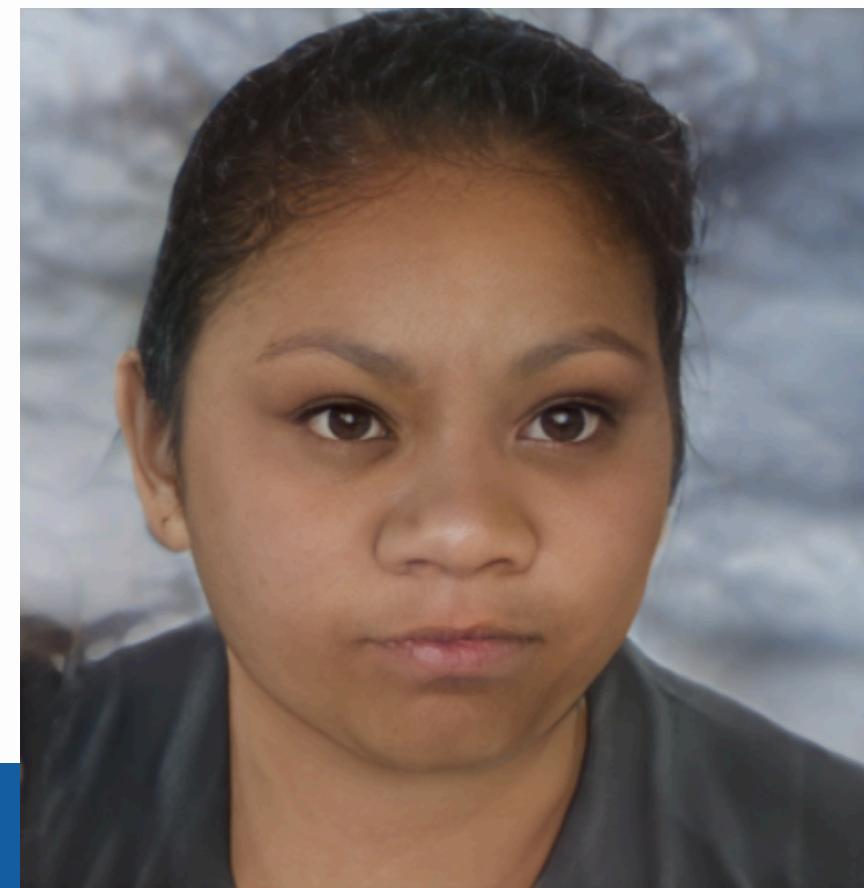
# Résultats : sourire



Projection dans l'espace latent



$\alpha = 1$

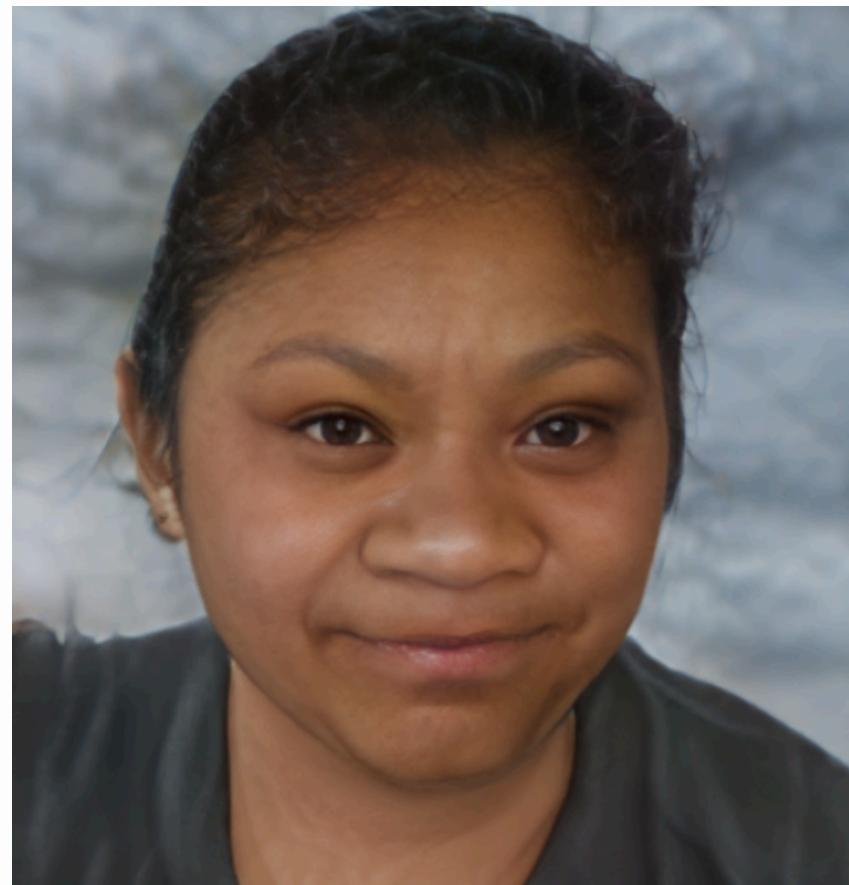
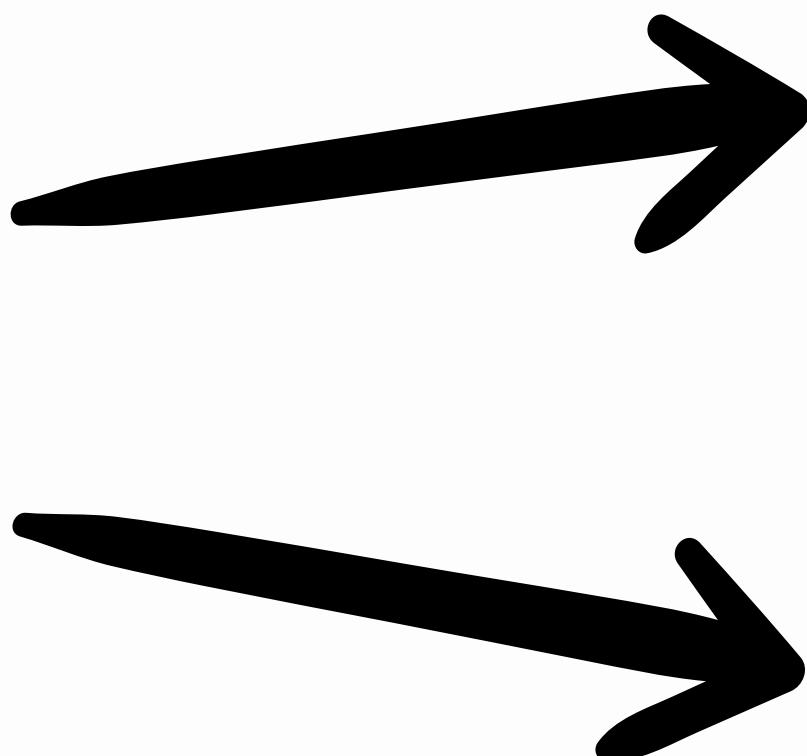


$\alpha = -1$

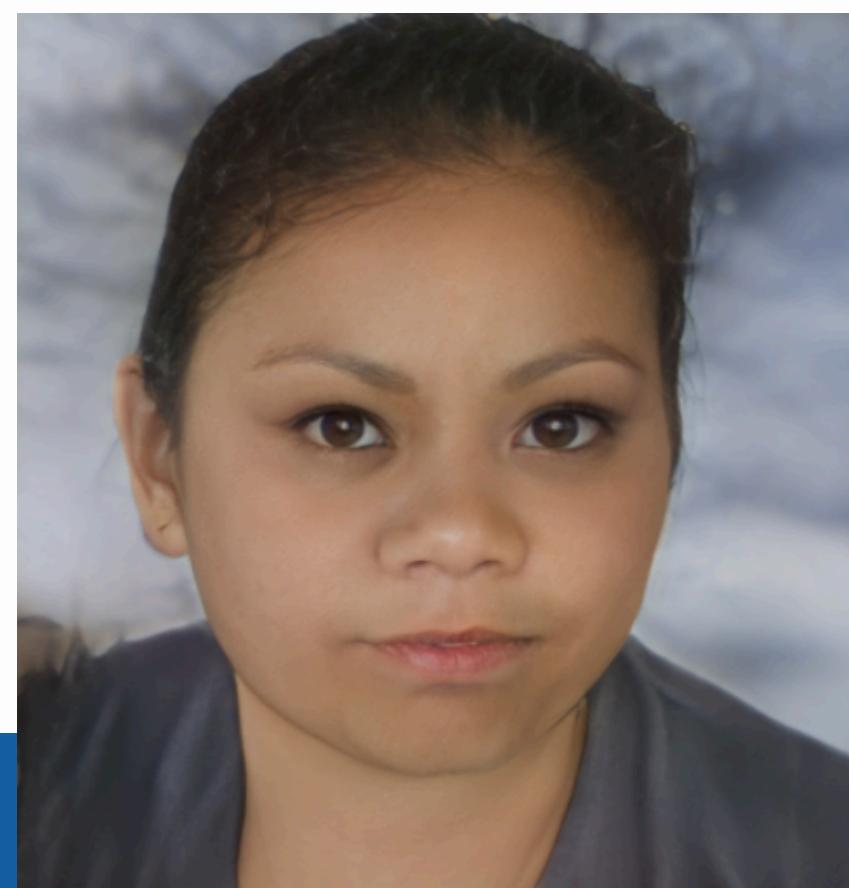
# Résultats : gros nez



Projection dans l'espace latent



**alpha = 1**

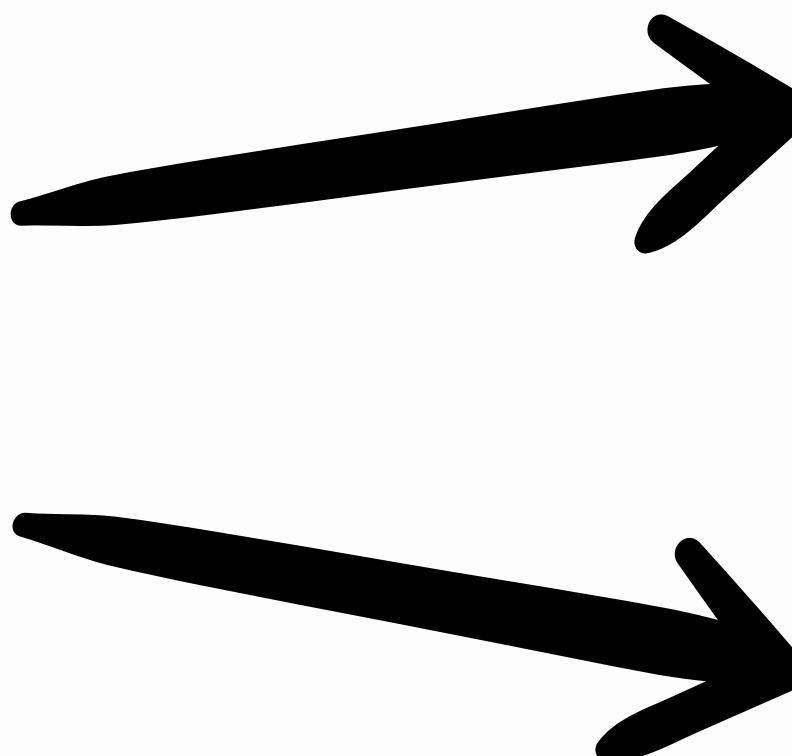


**alpha = -1**

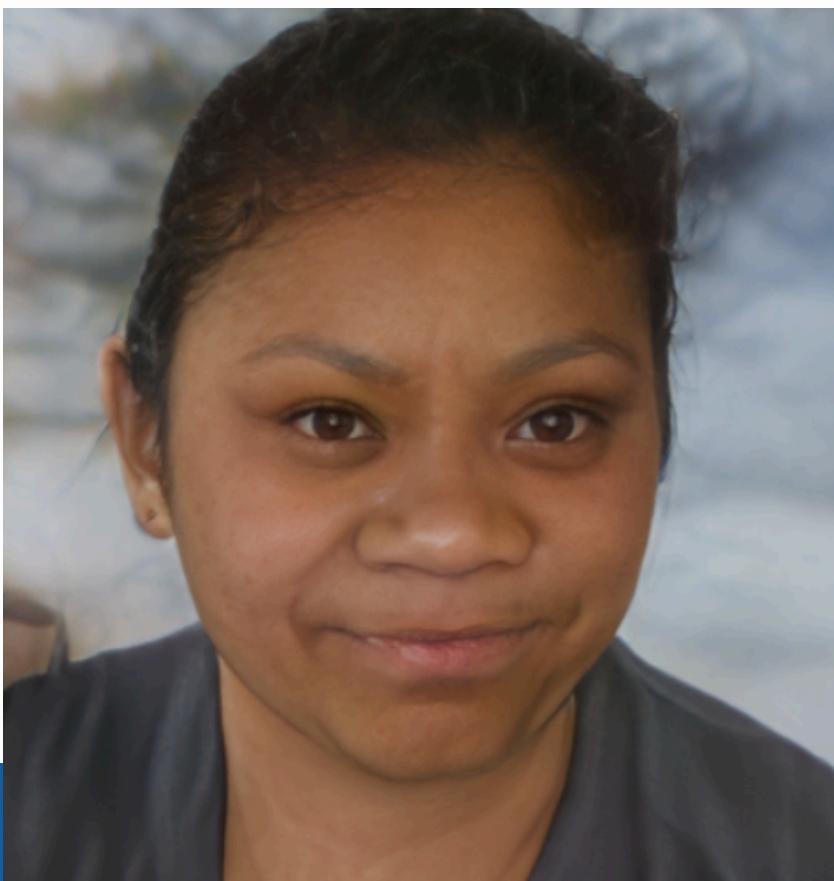
# Résultats : Peau claire



Projection dans l'espace latent



**alpha = 3**



**alpha = -2**

# CONCLUSION

- **Objectif :** Modification d'un attribut d'une image
- **Résultats :** Conception d'un Classifieur et d'un Transformer qui semblent fonctionner (convergence des loss) et modifications d'images
- **Limites :** Temps, différentes versions de StyleGAN & Pas de maîtrise totale des modifications des images



# Sources

## Github :

- <https://github.com/NVlabs/stylegan3>
- <https://github.com/InterDigitalInc/latent-transformer>
- <https://github.com/eladrich/pixel2style2pixel>
- <https://github.com/csxmli2016/w-plus-adapter>

## Articles :

- Yao, X., Newson, A., Gousseau, Y., & Hellier, P.. A Latent Transformer for Disentangled Face Editing in Images and Videos. LTCI, Télécom Paris, Institut Polytechnique de Paris, France & InterDigital R&I, 975 avenue des Champs Blancs, Cesson-Sévigné, France
- Abdal, R., Qin, Y., & Wonka, P.. Image2StyleGAN: How to Embed Images Into the StyleGAN Latent Space? KAUST.
- Shen, Y., Gu, J., Tang, X., & Zhou, B.. Interpreting the Latent Space of GANs for Semantic Face Editing. The Chinese University of Hong Kong & The Chinese University of Hong Kong, Shenzhen.

