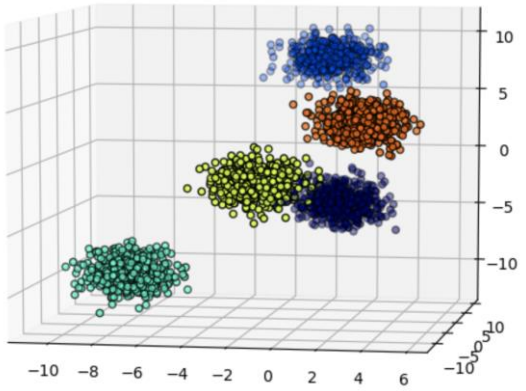
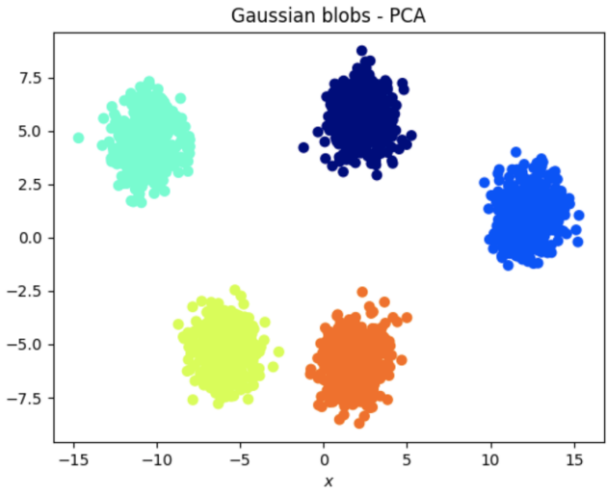
Les autoencoders, les variational autoencoders et les eenerative adversarial networks : tour d'horizon

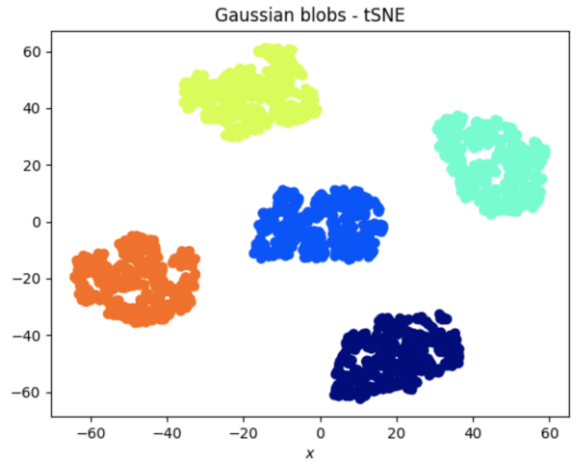
Pascal Poncelet
LIRMM
Pascal.Poncelet@lirmm.fr
http://www.lirmm.fr/~poncelet

Un problème de dimensionnalité

- Comment réduire la dimensionnalité ?
  - ACP
  - T-SNE
  - UMAP
  - LDA
  - Etc.





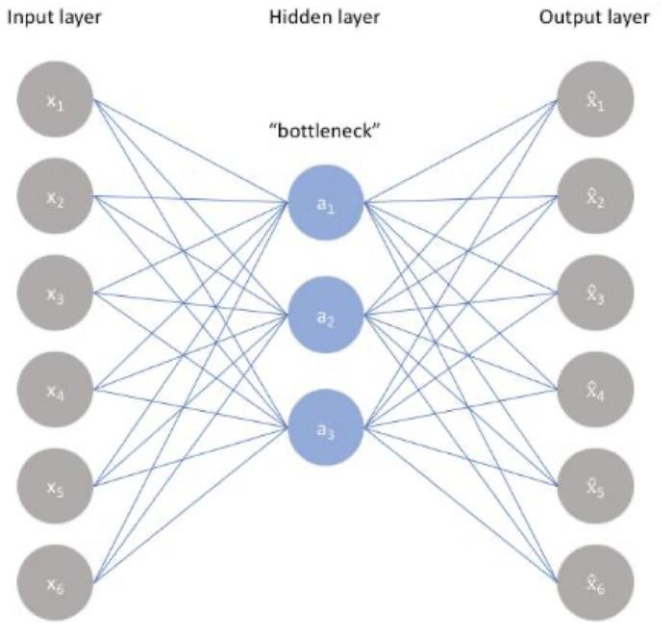


Un problème de données labelisées

- Difficile d'avoir des données labelisées
- Que faire si l'on a beaucoup de données mais sans label ?
- Apprentissage non supervisé
  - Classification non supervisé
- Evaluation perte :
  - Supervisé : confrontation avec données de test labelisées
  - Non supervisé : erreur basée sur la reconstruction de l'entrée

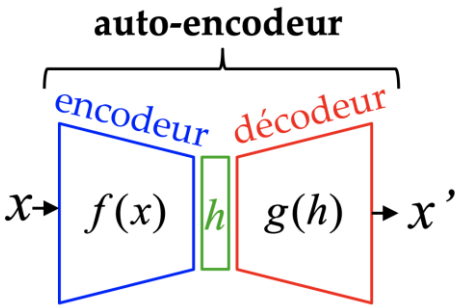
L’autoencoder

- C’est un réseau dont la couche centrale constitue un goulot d’étranglement
- Les sorties sont identiques aux entrées !



Deux parties

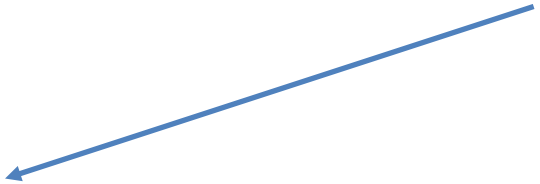
- Un encoder : encode les données d'entrée vers l'espace latent
  - l'encoder doit trouver une projection vers un espace de plus petite dimension
- Un decoder : décode à partir de l'espace latent les données pour retrouver les données d'origine
- Préférer taille x > taille h ! Sinon recopie!
- Remarque : si f, g sont linéaires : proche de PCA



Composition de l'encoder

- N'importe quel type de réseau :
  - Multilayer perceptron
  - CNN
  - Récursif (exemple texte)

Exemple avec des convolutions



```
# Partie encodeur
model.add(Conv2D(30, 3, activation= 'relu', padding='same',
input_shape = (28,28)
model.add(MaxPooling2D(2, padding= 'same'))
model.add(Conv2D(15, 3, activation= 'relu', padding='same'))
model.add(MaxPooling2D(2, padding= 'same'))
```

Composition du decoder

- L’entrée doit correspondre à la sortie de l’encoder

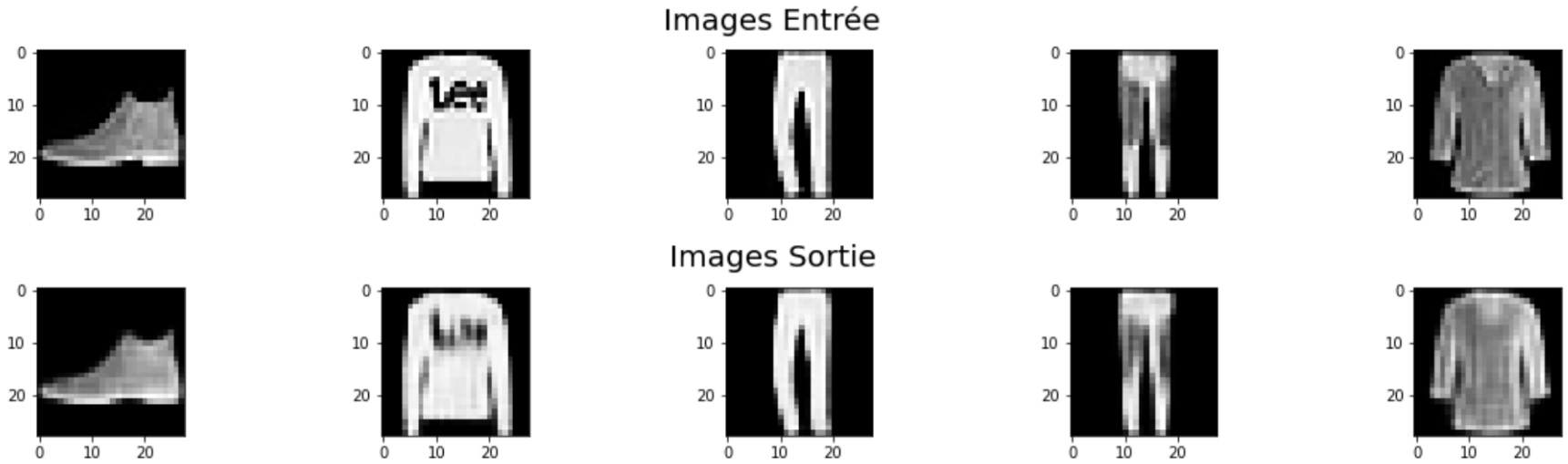
```
# Partie décodeur
model.add(Conv2D(15, 3, activation= 'relu’, padding='same'))
model.add(UpSampling2D(2))
model.add(Conv2D(30, 3, activation= 'relu', padding='same'))
model.add(UpSampling2D(2))
```

Upsampling fonctionne en répétant les lignes et les colonnes de l'entrée. Ainsi une image de 2x2 donnera avec un UpSampling (2) une image de 4x4.

Suppression du decoder

- Il est possible d'enlever le decoder et de lui mettre une partie classification
  - C'est ce que nous faisions pour les CNN !

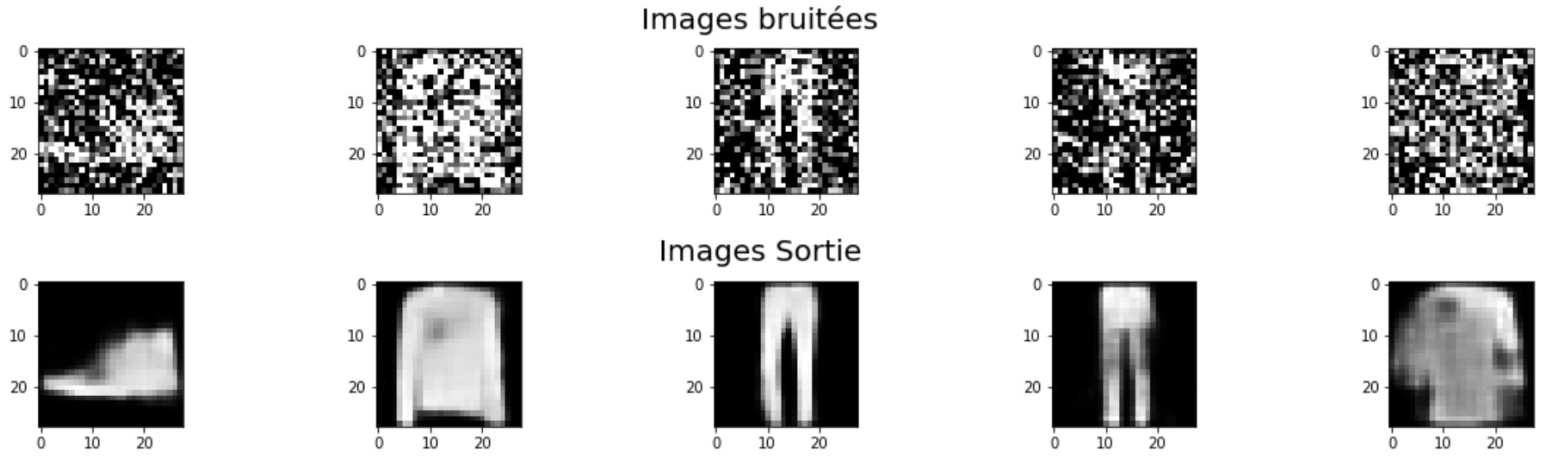
Exemples



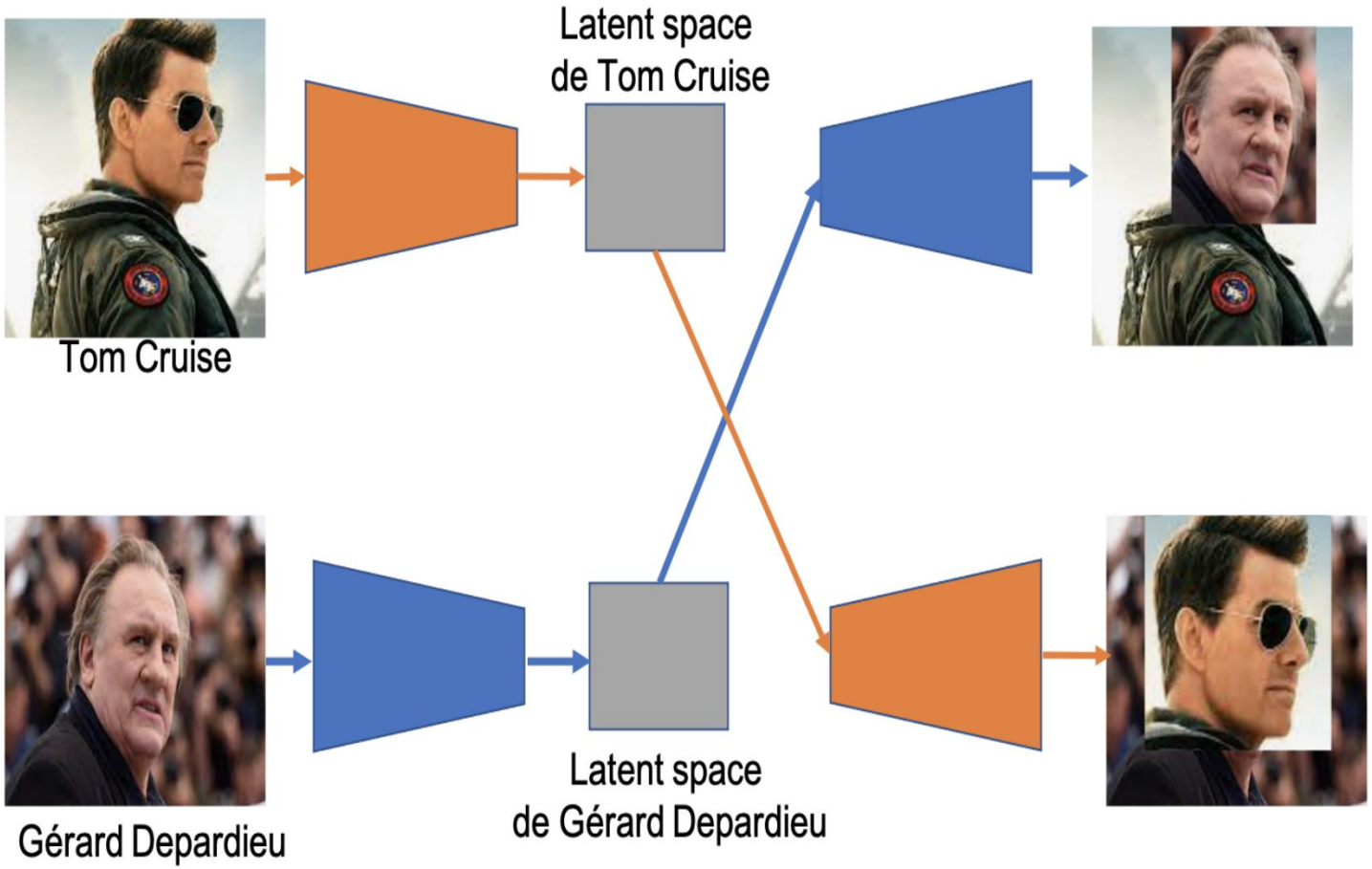
MNIST - epochs=15, batch_size=128

Enlever du bruit, colorier, …

- En entrée mettre une image bruitée (resp. noir et blanc) et en sortie l'image non bruitée (resp. couleur)



Principe du DeepFake



https://archive.org/details/github.com-aerophile-awesome-deepfakes_-_2020-03-27_06-52-56

Rappel : Rôle d'un autoencoder

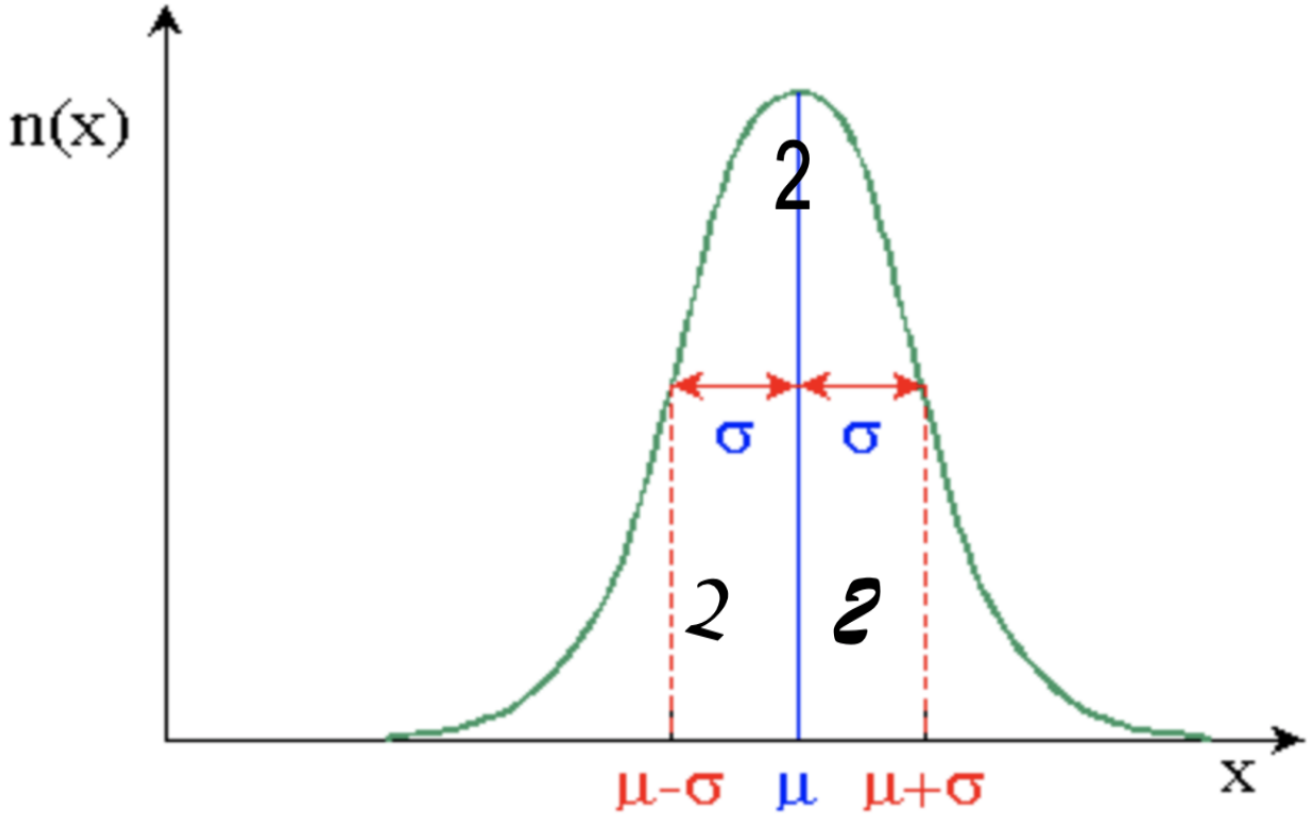
- Il peut débruiter, générer, colorier des images … mais son premier rôle = réduire les données d'origines dans un espace (espace latent)

- Générer ?
  - Utiliser l'espace latent pour générer à l'aide du decoder !

Générer : principe

- Echantillonner le latent space à partir d'une distribution normale
  - comme le latent space représente les images d'origines compressées, nous tirons un vecteur aléatoire en respectant une distribution normale
  - le fait de prendre une distribution normale permet de localiser une zone du latent space que nous espérons continue !

Distribution normale des 2



On espère que tous les 2 se situent dans la même zone

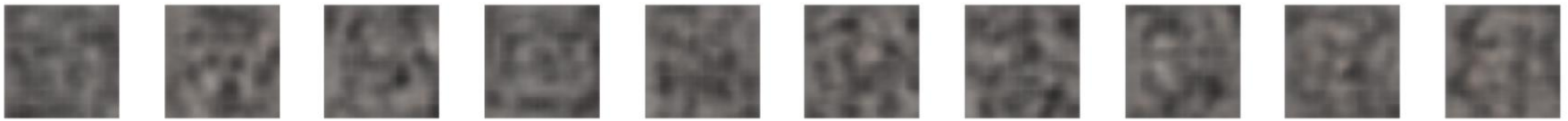
En prenant un échantillon dans cette zone on espère générer un 2

Concrètement

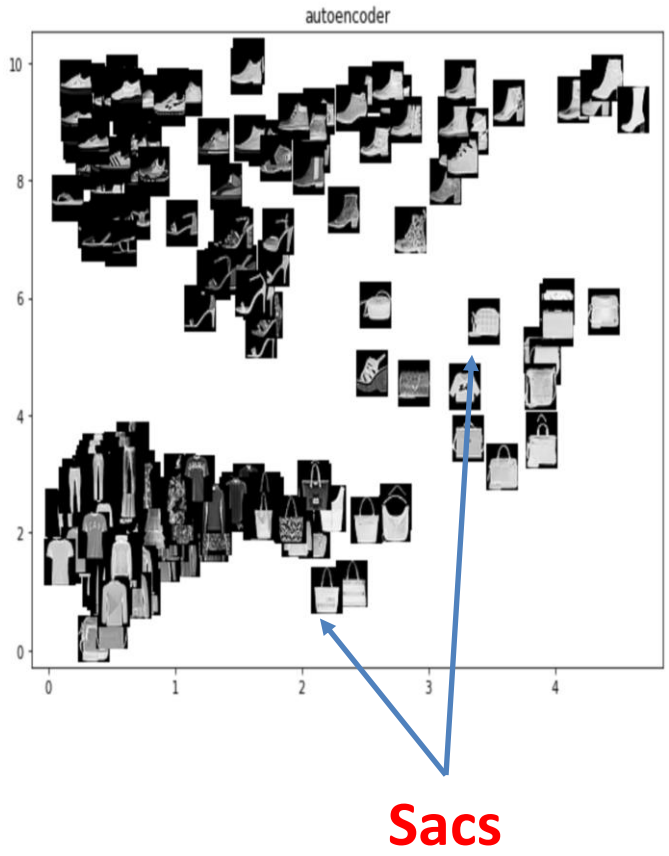
```
normal_distribution_sample=
np.random.normal(0,1,size=(nb_images,taille_latent
))


# Application de la partie decoder
  reconst_images =
decoder.predict(normal_distribution_sample)
```

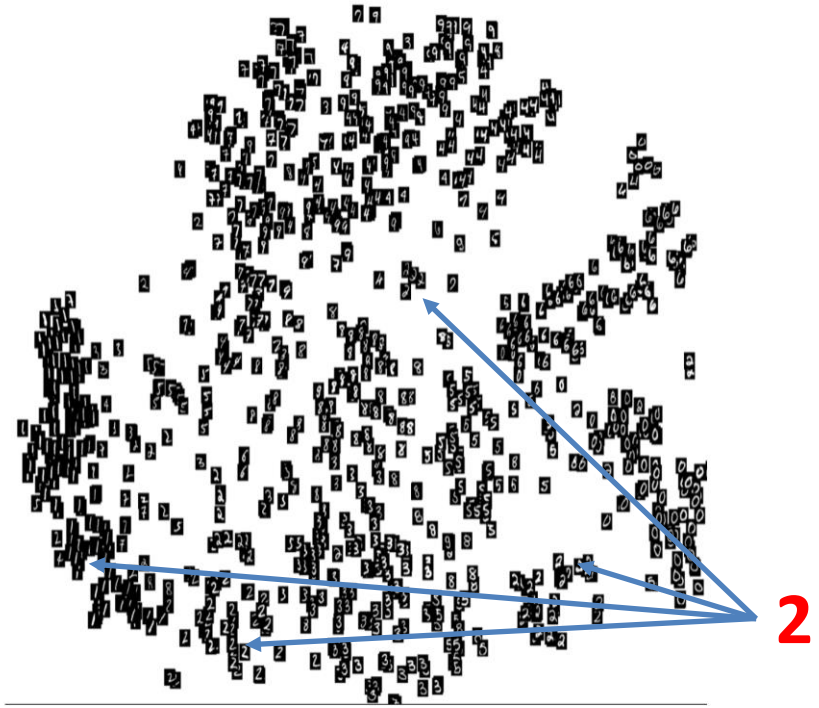
Résultat

```
1/1 [==============================] - 0s 191ms/step
```

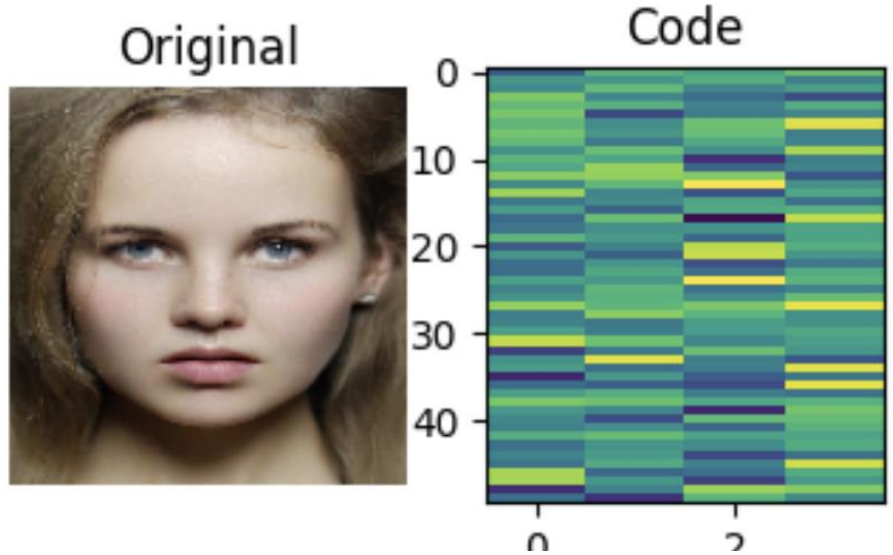


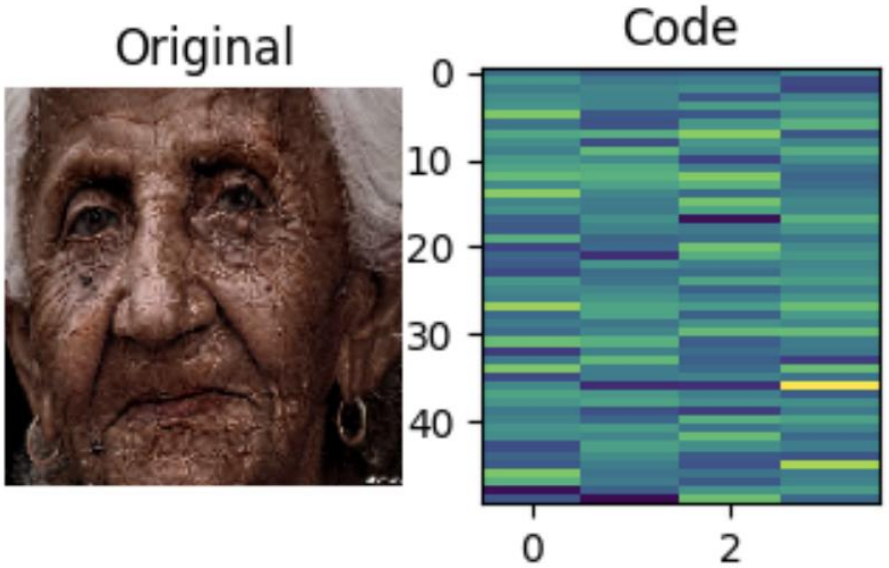
Pourquoi ?





Pourquoi ?

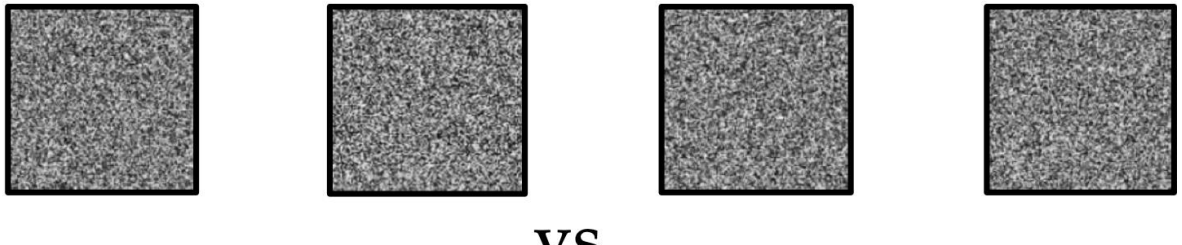




Pourquoi ?

- L'espace latent dans un auto encodeur est souvent de dimension réduite par rapport à l'espace d'entrée
  - plusieurs images d'entrées peuvent être projetées sur la même région de l'espace latent
  - pas possible de générer de nouvelles images uniquement à partir de l'espace latent car plusieurs images peuvent correspondre au même point
  - Même si l'espace latent est continu, il n'est pas garanti que n'importe quel point de cet espace puisse être décodé de manière significative en une image reconnaissable

Pourquoi ?



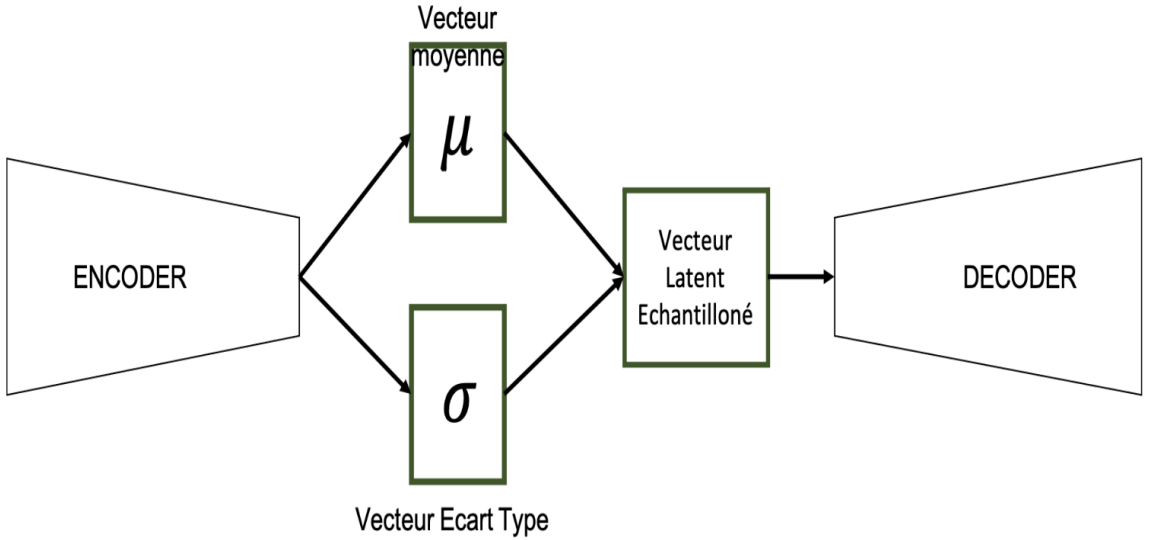
vs.

Les variational autoencoders

- Alors qu'un auto encodeur simple apprend à mapper chaque image sur un point fixe dans le latent space, l'encodeur d'un auto encodeur variationnel (VAE) mappe chaque image sur une distribution normale standard à z-dimensions (la taille du latent space)

Les variational autoencoders

- Le VAE étend l'auto encodeur classique en introduisant une composante probabiliste dans le processus d'encodage (la partie vecteur moyenne et vecteur écart type) afin de mieux considérer la distribution des données



Mise en œuvre

- Seule la partie encoder doit être réécrite
- L'encoder doit être capable de calculer le vecteur moyen et écart type pour générer un échantillon du vecteur latent

- Générer l'échantillon : utilisation de la divergence de Kullback-Leigler pour trouver la distribution la plus proche des données

L'objectif de KL est de voir celles qui sont les plus proches … ne pas hésiter à approfondir

Mise en œuvre

- Le corps de l'encoder reste le même +

```
# Différence par rapport à avant

# créer une couche pour la moyenne
mean_mu = Dense(output_dim, name = 'mu')(x)
# créer une couche pour l'écart type
log_var = Dense(output_dim, name = 'log_var')(x)
# Définir une fonction pour l'échantillonnage
def sampling(args):
  mean_mu, log_var = args
  epsilon = K.random_normal(shape=K.shape(mean_mu), mean=0.,
stddev=1.)
  return mean_mu + K.exp(log_var/2)*epsilon
```

En faire une couche

```
# Le lambda layer de Keras permet d'inclure la fonction d'échantillonnage comme une couche.
# c'est important pour la descente de gradient

vae_encoder_output = Lambda(sampling, name='vae_encoder_output')([mean_mu, log_var])
```

- Il y a pleins d’autres manières de faire ! Il s’agit juste d’un exemple d’implémentation !

Quid de la fonction de coût ?

- pour l'autoencoder initial :
  - MSE - Mean Square Error, binary_crossentropy, etc. Ces fonctions permettent de mettre à jour les poids dans le modèle.
- Par contre, à présent, nous avons un autre élément qui doit être mis à jour : bien choisir la distribution pour l'échantillonnage !
- Nécessité de revoir la fonction de coût qui doit intégrer les deux !

Quid de la fonction de coût ?

C'est juste une illustration

```
def kl_loss(y_true, y_pred):
    kl_loss =  -0.5 * K.sum(1 + log_var - K.square(mean_mu)
K.exp(log_var), ax
    return kl_loss
```

Coût pour KL

```
def r_loss(y_true, y_pred):
    return K.mean(K.square(y_true - y_pred), axis = [1,2,3])
```

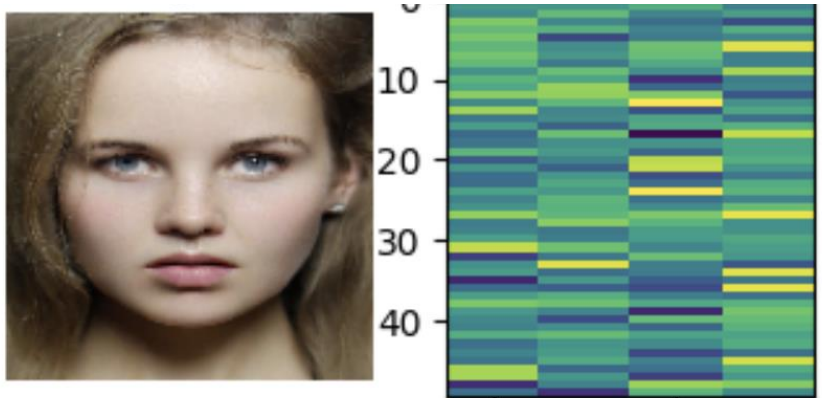
Coût pour les poids

```
def total_loss(y_true, y_pred):
    return B*r_loss(y_true, y_pred) + kl_loss(y_true, y_pred)
```

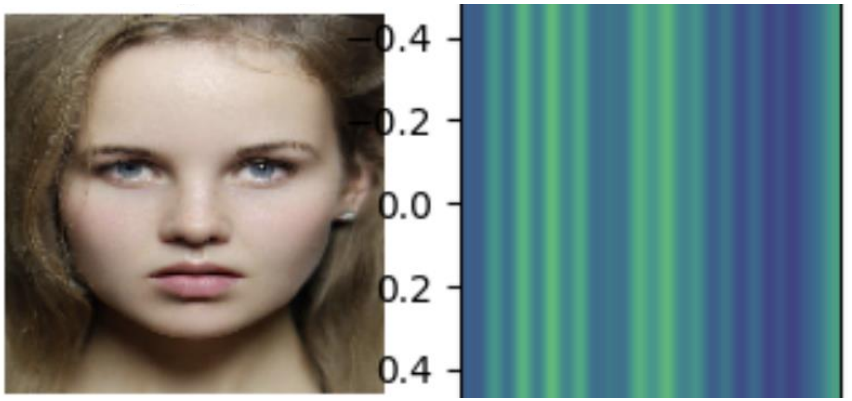
Coût total

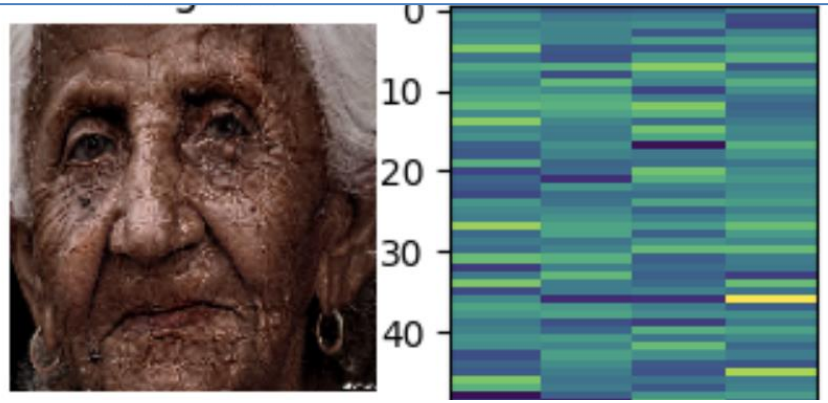
L’espace Latent : AE vs VAE

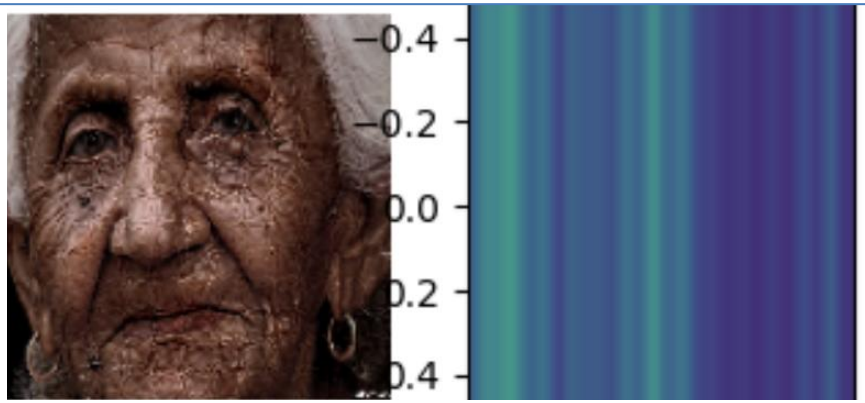
AE



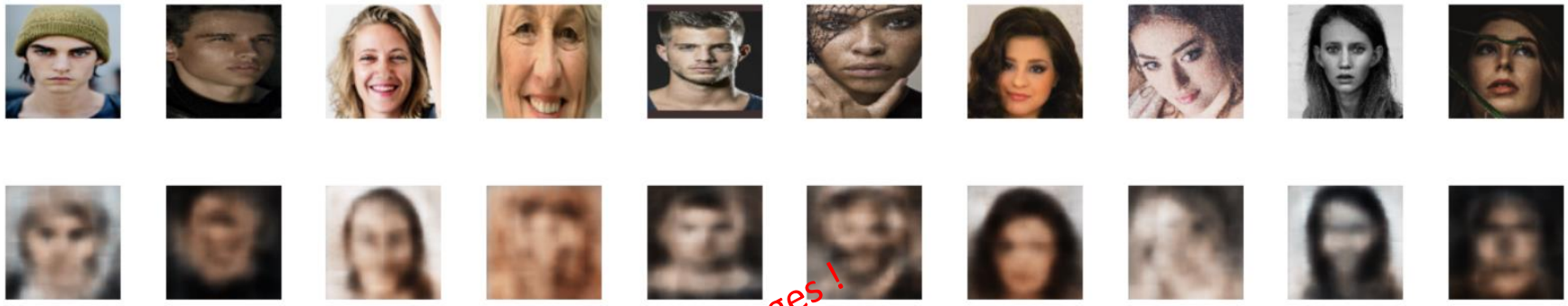
VAE







Un exemple de génération



Affichage de la sortie du VAE

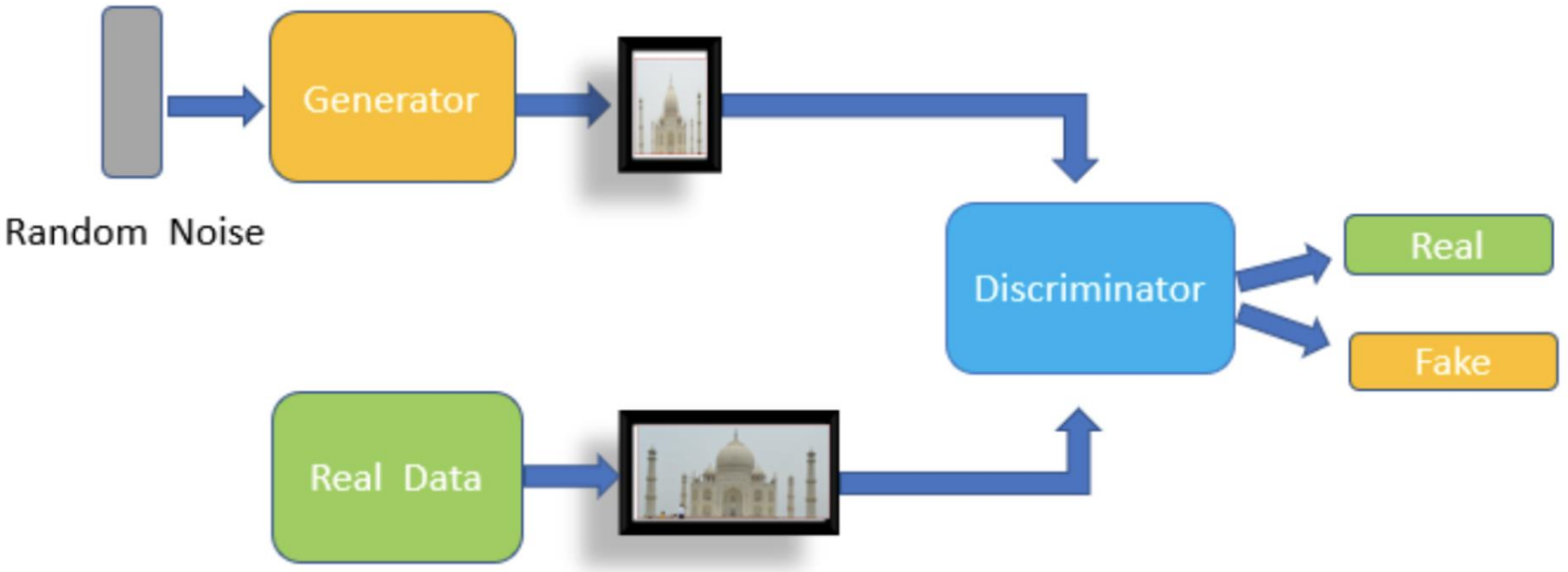
Attention : peu d'entrainement sur peu d'images !

Génération d'images

Générer : Generative Adversarial Networks

- Les Generative Adversarial Networks (GAN) permettent de générer des données
- Le GAN est un échange entre deux réseaux :
  - Le générateur : son objectif est à partir de données aléatoires de générer des données qui sont le plus similaires possibles d'un jeu de données
  - Le discriminateur : son objectif est de parer les fausses images générées et d'éviter que celles-ci soient considérées comme vraies.

GAN



GAN : principes

- A partir de données aléatoires, le générateur va générer des données. Ces dernières sont combinées à des données réelles et sont envoyées dans le discriminateur
- L'objectif du discriminateur est de déterminer s'il s'agit de données réelles ou de fausses données.
  - classification binaire : données réelles ou données fausses

GAN : propagation

- L'erreur à la fin du discriminateur est reportée au niveau du générateur afin qu'il mette ses poids à jour ... donc au fur et à mesure des epochs, la loss est projetée vers le générateur

- Il faut faire attention à ne pas projeter l'erreur du générateur à l'issue du modèle GAN au niveau du discriminateur !

KERAS : méthode train_on_batch

Exemple

MNIST – 100 epochs

Extensions

- Il existe de nombreuses extensions :

  AE denoising : ajoute du bruit aléatoire à l'entrée

  AE contractive : désensibilise l'encodeur à certaines directions

  CVAE : Conditional Variational Autoencoders – prise en compte des classes

  GAN+VAE

  etc.

  Domaine où les nouveautés apparaissent toutes les semaines !

Conclusion

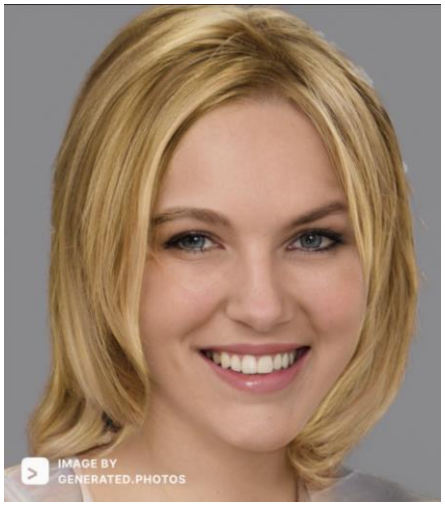
- Voir les notebooks :

  Autoencoder

  Variational Autoencoder

  Gan

Conclusion



Génération :
Beaucoup d'images
Beaucoup de temps de calcul
…. mais le principe est basé sur les approches présentées

- Des questions ?