ENSTA Paris *Mars 2022*

Projet IA
Génération d'images

Groupe 6 : Mouïn Ben Ammar, Lisa Giordani,
Yoldoz Tabei, Ilias Harkati
Tuteur : Nacim Belkhir

Contexte du projet

Le besoin d’avoir un dataset de très grande taille pose un problème dans le Deep Learning

Solution: data augmentation à travers la génération d’images

Il existe 4 stratégies principales:

- Les GANs (generative adversarial networks)
- Les VAEs (variational autoencoders)
- Les flow-based generative models
- Les modèles de diffusion

Applications de la génération d’images:

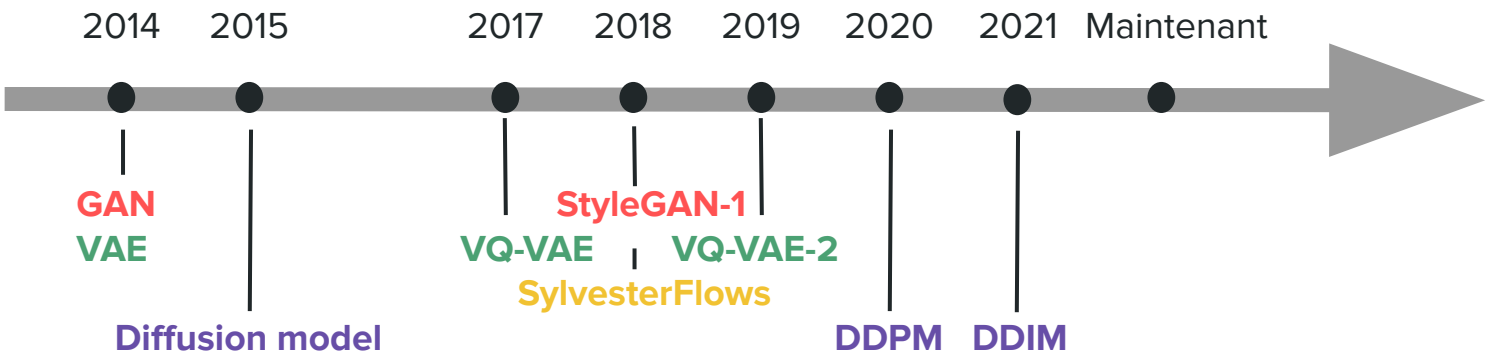
- Génération des personnages à partir de dessins animés
- Génération des photographies réalistes
- Traduction texte-image
- Génération de la vue frontale du visage
- etc...

Datasets utilisés:

- CIFAR-10
- Stanford Dogs
- MNIST
- Freyfaces

Sommaire

Etat de l’art


2014
2015
2017
2018
2019
2020
2021
Maintenant
GAN
VAE
Diffusion model
VQ-VAE
StyleGAN-1
VQ-VAE-2
SylvesterFlows
DDPM
DDIM

Indicateurs de performances

Un grand défi pour les réseaux génératifs

Inception score

Facilement manipulable

$$IS(\mathbb{P}_g) = e^{E_x \sim \mathbb{P}_g [KL(\mathbb{P}_M(y|x)||\mathbb{P}_M(y)]}$$

FID

- Lent
- Features peuvent ne pas être optimales
- Biaisé

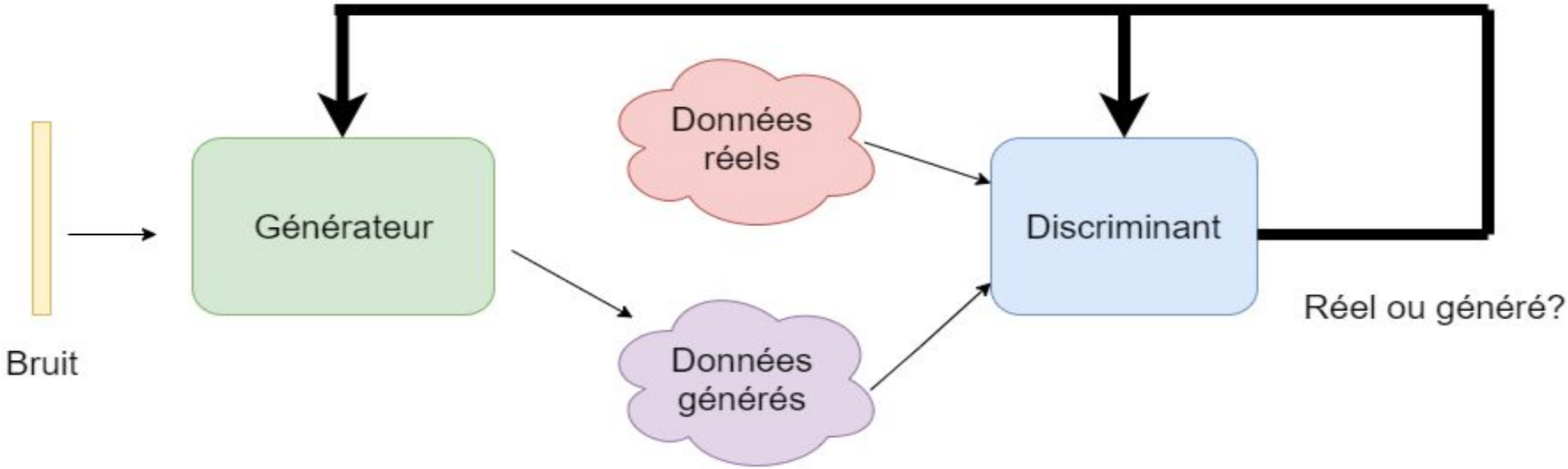
$$FID(rels, faux) = \|\mu_r - \mu_f\|^2 + Tr(\Sigma_r + \Sigma_f - 2 * \sqrt{\Sigma_r * \Sigma_f})$$

HYPE

- Meilleur indicateur
- Demande beaucoup de ressources

I. GAN (Generative adversarial networks)

GANs

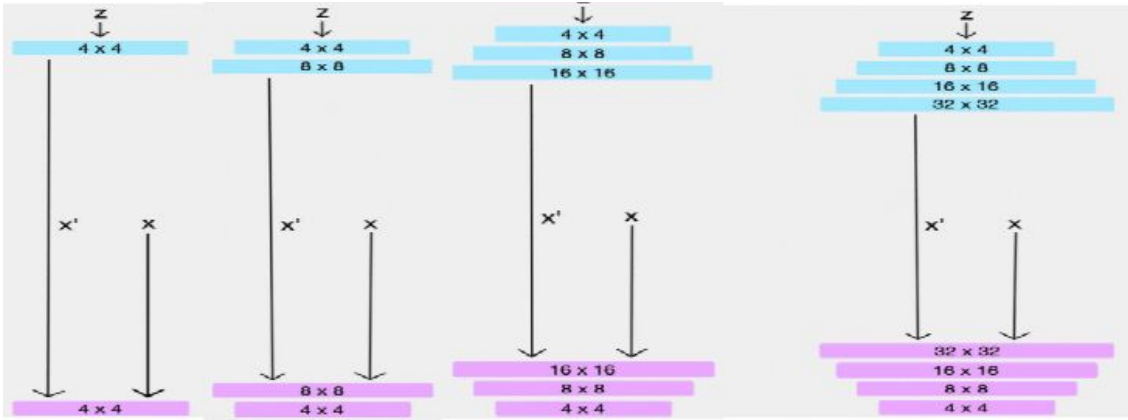


$$E(G,D) = \frac{1}{2}(E(x)_{x\sim p_t}[1 - D(x)] + E(x)_{x\sim p_g}[D(x)])$$

$$\underset{\mathbf{G}}{Max}(\underset{\mathbf{D}}{min}\, E(G,D)$$

StyleGANs

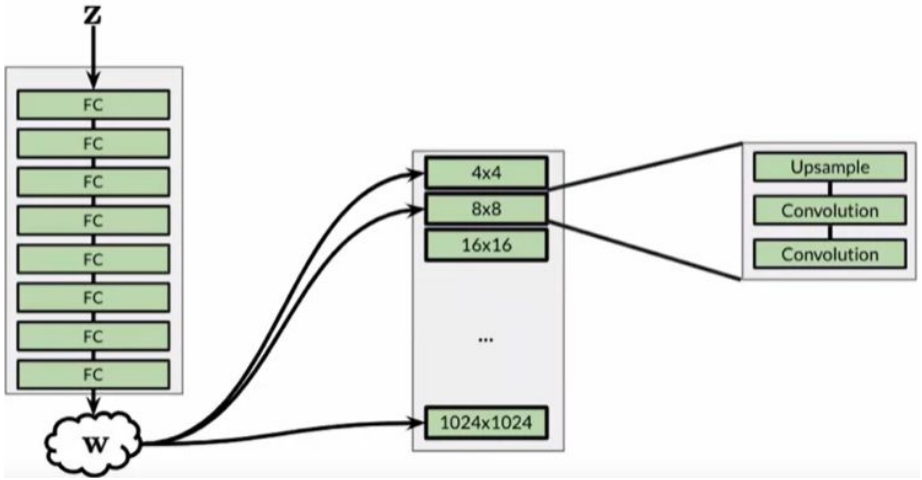
- Meilleur qualité de donné généré
- Plus de contrôle sur la génération



Architecture

1/ Réseau de cartographie

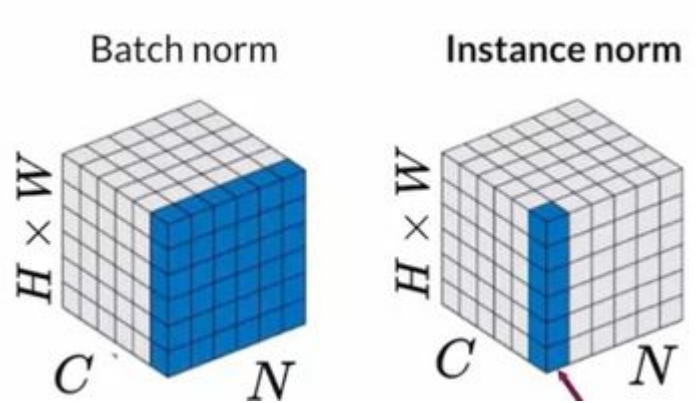
- Réduire l’enchevêtrement des caractéristiques
- Le vecteur W est injecté dans tous les résolutions



Architecture

2/ AdaIN

- Normalise chaque instance (enlève le style)
- Transforme le vecteur W, par une échelle et un bias (ajoute de style)
- injecte le vecteur transformé après chaque convolution



3/ Entrée de bruit constant

- Facilite l'entraînement
- Diminue l'enchevêtrement.

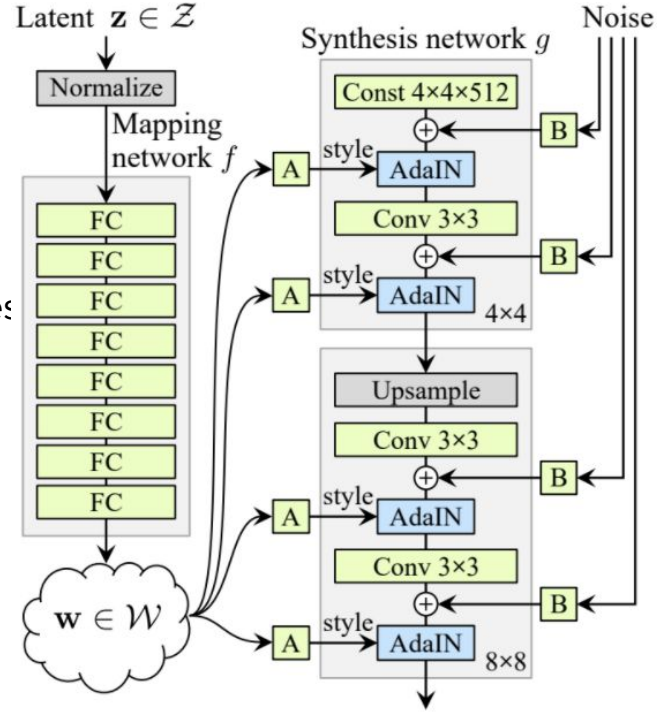
4/ Troncature

- Utiliser à l'étape de l'inférence
- Évite de générer de mauvaises images

Architecture

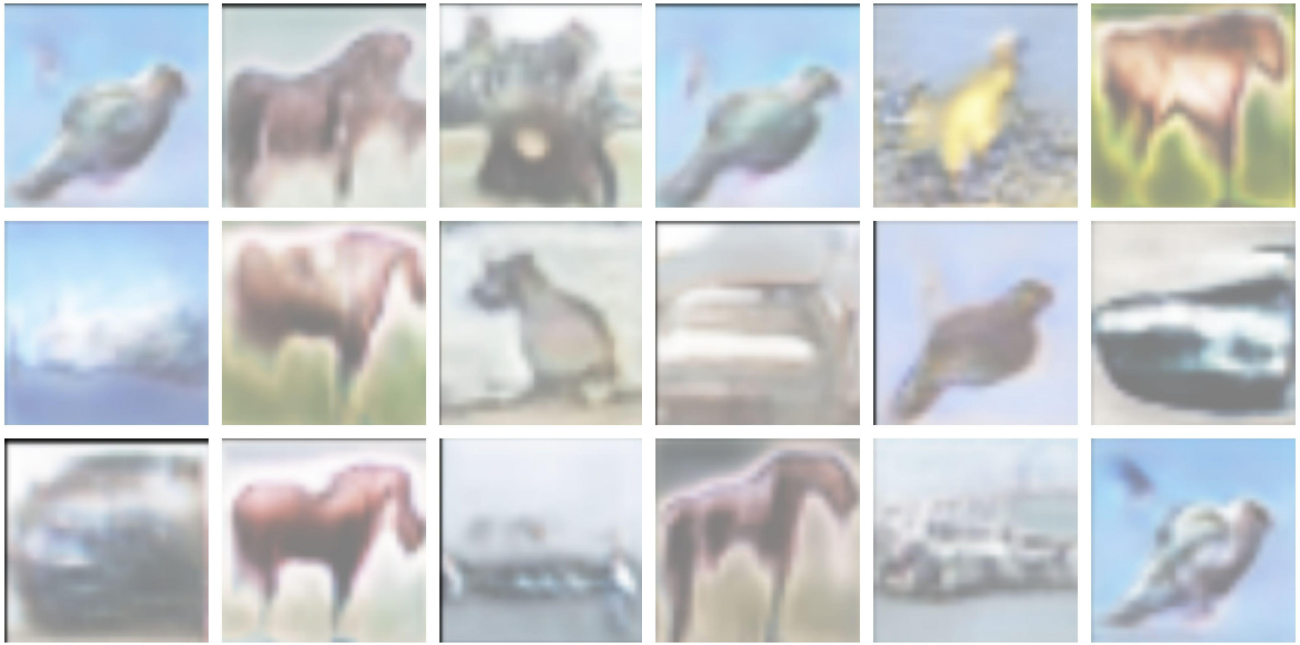
5/ Mélange de styles et bruit stochastique

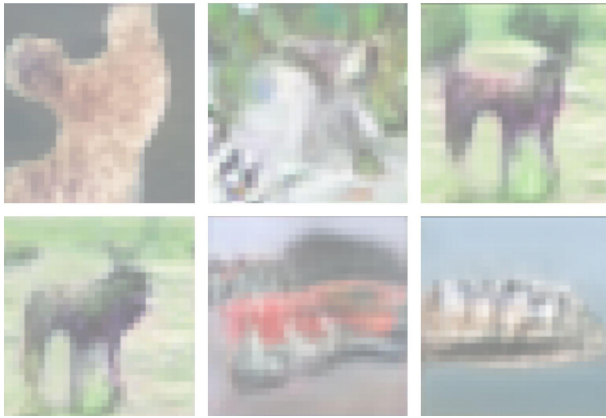
- Injecte un bruit avant AdaIN
- Ajoute divers petits caractéristiques au images qui les rend réaliste.



- On utilise deux vecteur de bruit intermédiaire (W)
- A partir d'un niveau aléatoire, on utilise le deuxième vecteur.
- Réduit la corrélation
- Mélange des caractéristiques du différent images.

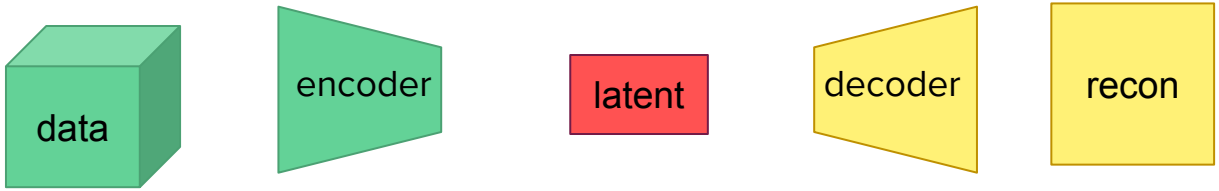
Résultats





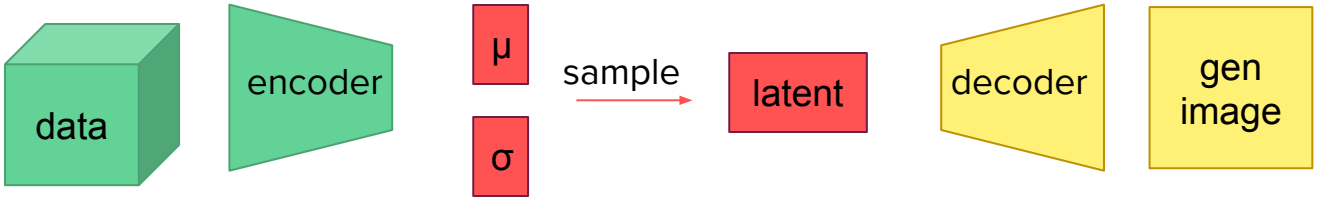
II. VAE (Variational Auto-Encoder)

Auto-encoder



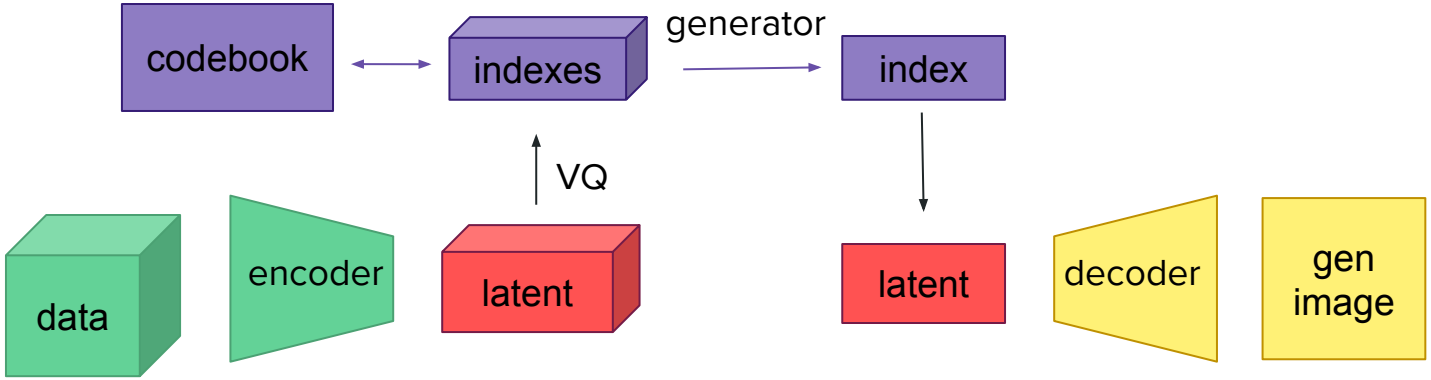
reconstruit seulement les images d'entrée
➜ génération de nouvelles images impossible

VAE



apprend une **distribution jointe gaussienne** à partir des données d'entrée

VQ-VAE (vector quantised auto-encoder)

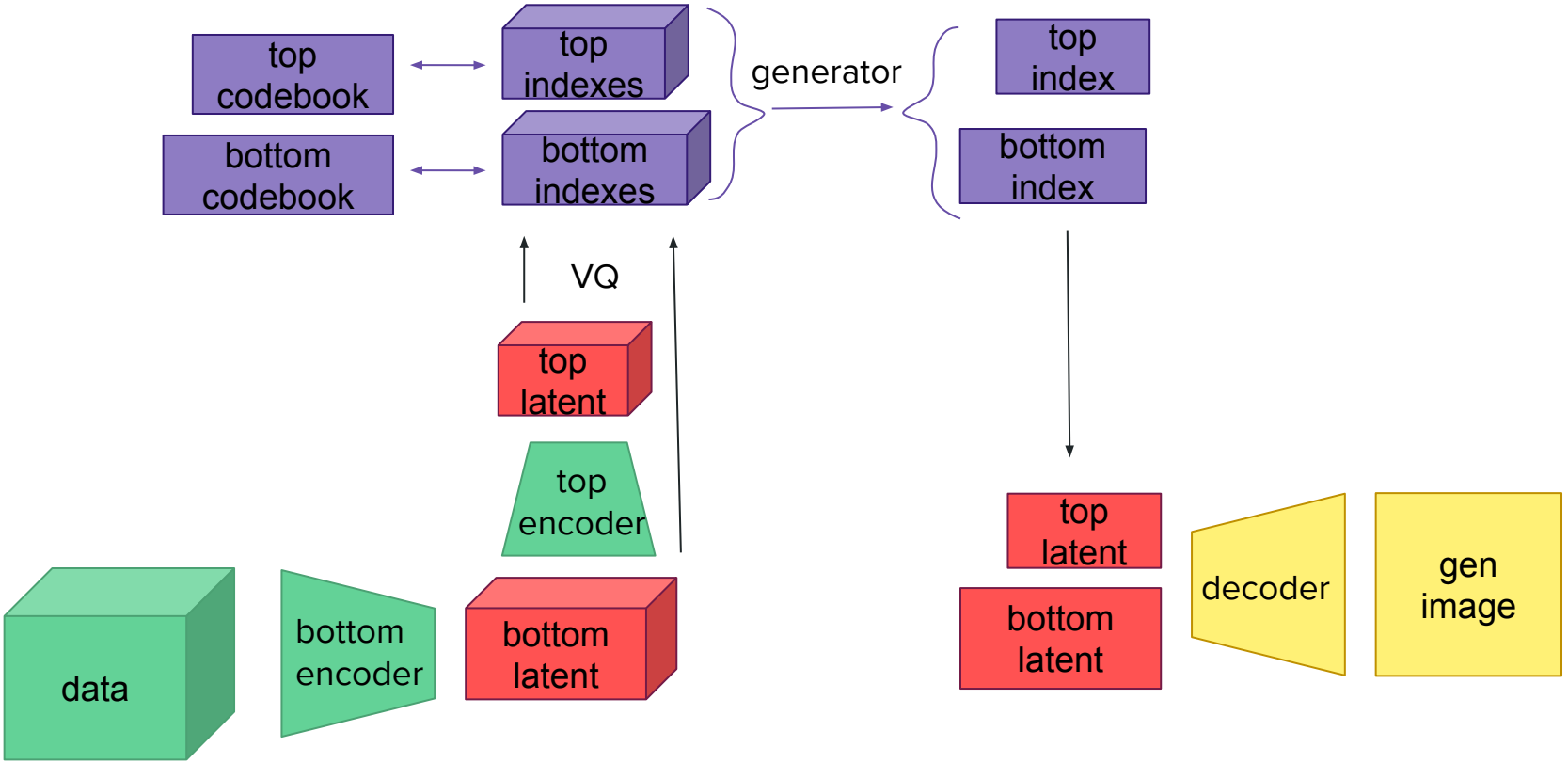


Quantification vectorielle → vecteurs latent **discrets**

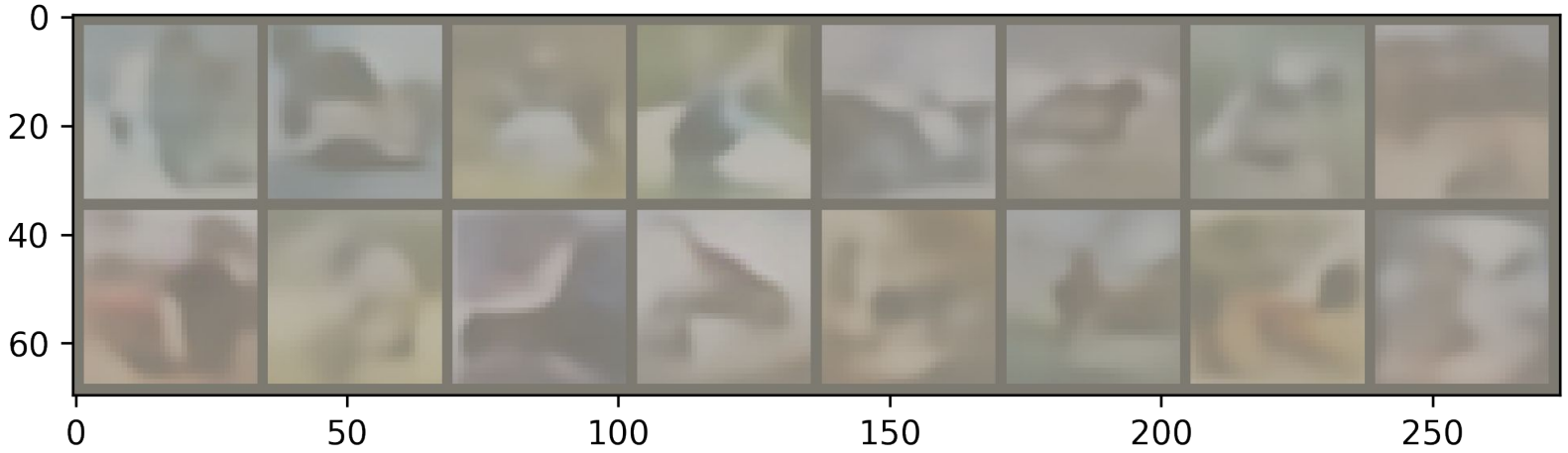
2 entraînements :

- **Modèle** : encodeur, décodeur et codebook
- **Générateur** : PixelCNN, transformer GPT2

VQ-VAE-2


top codebook
bottom codebook
top indexes
bottom indexes
generator
top index
bottom index
VQ
top latent
top encoder
data
bottom encoder
bottom latent
top latent
bottom latent
decoder
gen image

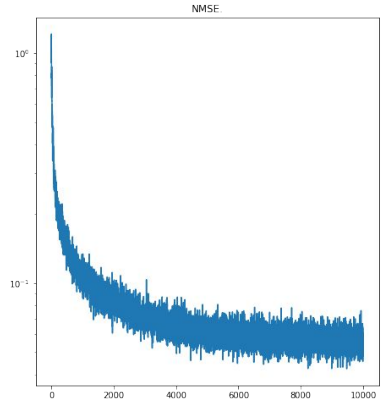
Résultats : génération d'images avec un VAE



Entraînement sur CIFAR-10 pendant 30 epochs (2h40)

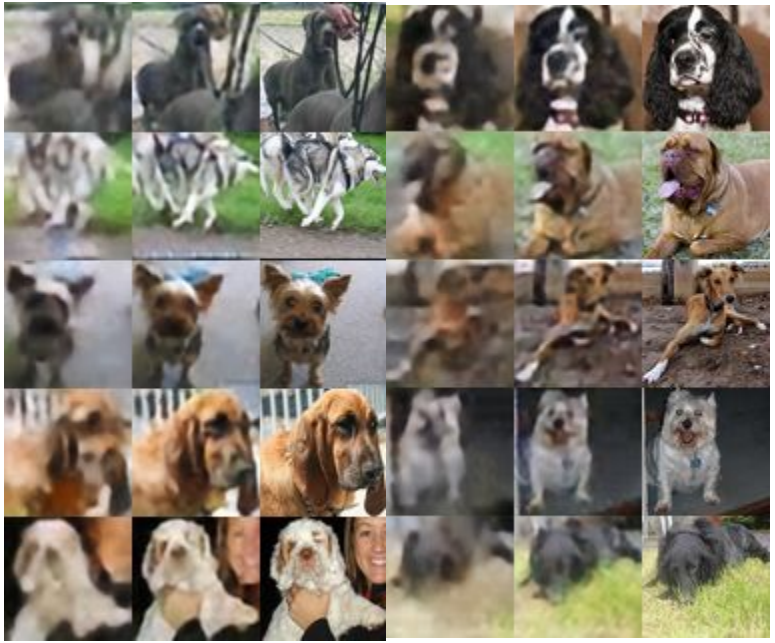
Résultats : reconstruction d'images avec un VQ-VAE

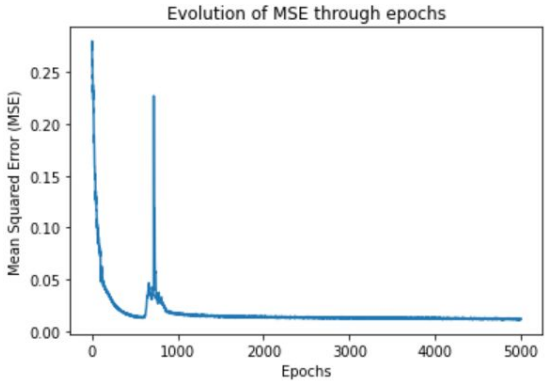
Entraînement sur CIFAR-10 pendant 8 epochs (8 min)



Courbe de loss (RMSE)

Résultats : reconstruction d'images avec un VQ-VAE-2

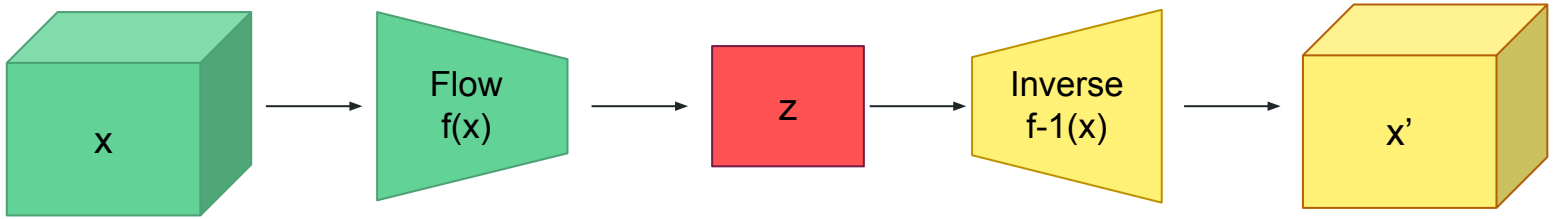




Courbe de loss (RMSE)

Entraînement sur *Stanford Dogs* pendant 5000 epochs (10 min)

III. Flow-based generative models



- Plus besoin d'entraîner le décodeur

⇒ Apprendre un réseau unique, qui est un **flux**, lui-même une composition de fonctions qui se transforment en différents blocs collés ensemble.

L'encodeur s'appelle un flux
Le décodeur s'appelle un flux inverse

Selon la règle de la chaîne de variables: $$p_x(x) = p_z(z)|det\frac{\partial f}{\partial x}|$$

- Le flux est une **composition de fonctions** qui transforment progressivement la distribution complexe en une distribution simple.

→ Il suffit donc que chacune des fonctions intermédiaires vérifient quelques conditions pour que tout le flux nous permette de générer des fonctions complexes à partir de fonctions simples.

Objectif: $$max[log(p(x))] = max[log(p(x)) + \sum_{i=1}^{K} log|det\frac{\partial f_i}{\partial x}|]$$

Sylvester Normalizing Flows for Variational Inference

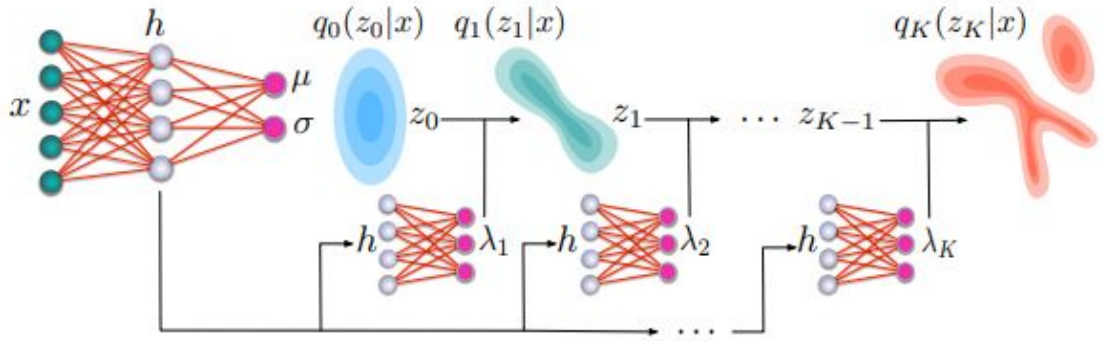
- Le déterminant jacobien de cette transformation peut être obtenu en utilisant l'**identité du déterminant de Sylvester**.

$$\det\left(\mathbf{I}_D + \mathbf{AB}\right) = \det\left(\mathbf{I}_M + \mathbf{BA}\right)$$

- Le réseau d'inférence consiste en plusieurs couches de "gated convolution" qui produisent un vecteur d'unité cachée, et des couches de convolutions transposées pour le décodeur.

Le déterminant Jacobien peut être calculé plus facilement en faisant cette transformation:

$$\mathbf{z}' = \mathbf{z} + \mathbf{Q}\mathbf{R}h(\tilde{\mathbf{R}}\mathbf{Q}^T\mathbf{z} + \mathbf{b}) = \phi(\mathbf{z})$$



- R and R˜ are upper triangular M × M matrices
- Q = (q1 . . . qM) with the columns qm ∈ R, D forming an orthonormal set of vector
- h is a suitable smooth activation function

- Il existe 3 types des Sylvester Normalizing flows (pour préserver l'orthogonalité de Q):
  - ***Les flux de Sylvester orthogonaux*** : utilisent une procédure itérative pour maintenir l'orthogonalité des matrices de paramètres

$$\mathbf{Q}^{(k+1)} = \mathbf{Q}^{(k)} \left(\mathbf{I} + \frac{1}{2} \left(\mathbf{I} - \mathbf{Q}^{(k)\top} \mathbf{Q}^{(k)} \right) \right)$$

  - ***Les flux de Sylvester de Householder*** : utilisent des réflexions de Householder pour construire des matrices orthogonales

$$H(\mathbf{z}) = \mathbf{z} - 2 \frac{\mathbf{v}\mathbf{v}^T}{\|\mathbf{v}\|^2} \mathbf{z}$$

  - ***Les flux triangulaires de Sylvester*** : alternent entre des matrices de permutation et les matrices d'identité pour les matrices orthogonales

Résultats

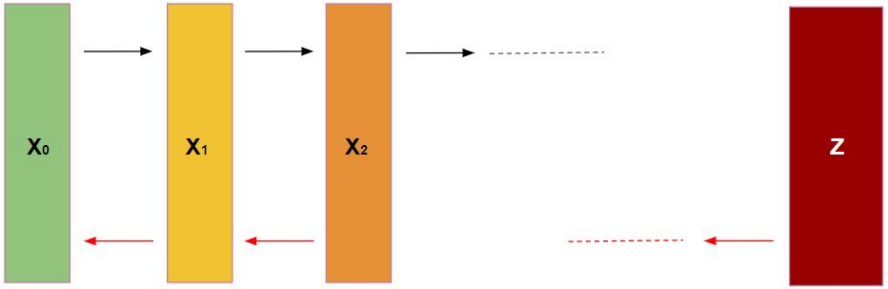
Modèle	NLL	-ELBO
Orthogonal Sylvester flows	82.1059	86.4930
Householder Sylvester flows	82.7201	84.8506
Triangular Sylvester flows	83.0452	84.7938

Modèle	NLL	-ELBO
Orthogonal Sylvester flows	4.6364	4.7239
Householder Sylvester flows	4.4800	4.6026
Triangular Sylvester flows	4.3856	4.5147

- Negative evidence lower bound (-ELBO)
- Negative log-likehood (NLL)

IV. Diffusion models

Principe



- Classe de modèle génératif
- Processus de bruitage : $q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \quad q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$
- Apprendre à inverser ce processus via un modèle d'apprentissage :

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \quad p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

Implémentation

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \mathbf{z}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

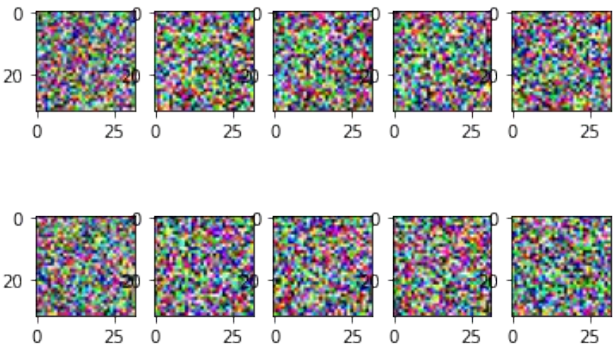
Algorithm 2 Sampling

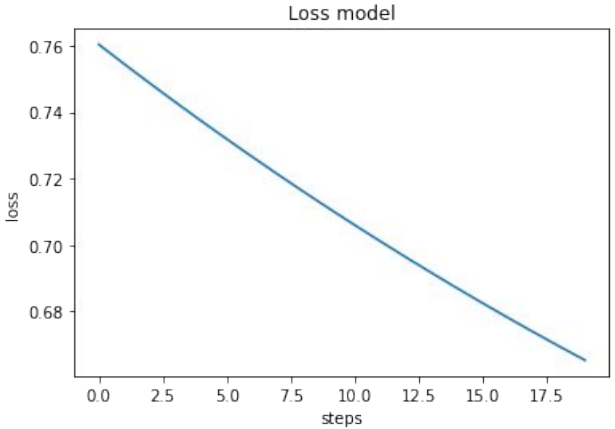
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \mathbf{z}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

- Modèle DDPM (Ho et al. (2020))
- Utilisation d'un réseau UNet
- Améliorations tirées des DDIM (Nichol & Dhariwal (2021))

Résultats et interprétation

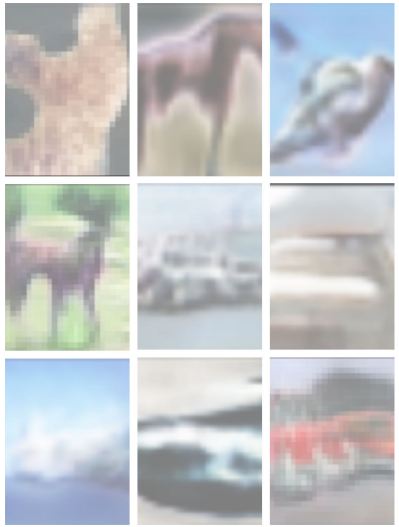
- Base de donnée : CIFAR-10 (32 x 32)
- Taille dataset : 20000
- Epochs: 20
- Temps d’entraînement: 1h~





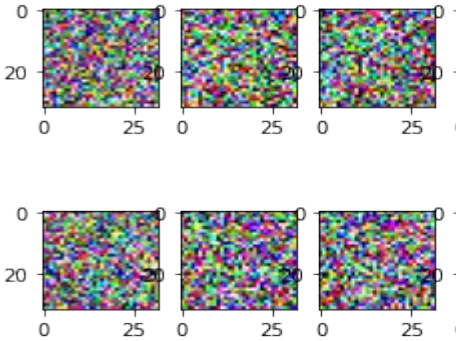
V. Comparaison des modèles génératifs

Qualité des images générées



StyleGAN

VAE



Diffusion (DDPM)

Modèle	Netteté	Cohérence	Diversité
GAN	+	++	++
VAE	-	+	++
VQ-VAE-2 (PixelCNN)	- -	- -	- -
Diffusion	o	- -	- -

TABLE 5 – Comparaison de la qualité des images générées (échelle : - -, -, o, +, ++)

Temps d'entraînement

Modèle	Temps d'apprentissage	Nombre d'epochs	Dataset	Taille du dataset
GAN	5h	30x4	CIFAR10 (32x32)	60 000
VAE	2h40	30	CIFAR10 (32x32)	60 000
VQ-VAE	8 min	8	CIFAR10 (32x32)	40 000
VQ-VAE-2	10 min	5 000	Stanford Dogs (64x64)	20 580
Flow-based	1h	300	MNIST (28x28)	60 000
	15 min	500	Freyfaces (20x28)	1 965
Diffusion	1h	20	CIFAR10 (32x32)	20 000

TABLE 3 – Comparaison des modèles dans la phase d'apprentissage

Modèle	Temps d'apprentissage du générateur
GAN	/
VAE	/
VQ-VAE-2 (PixelCNN)	> 6h pour 9 000 epochs
VQ-VAE-2 (GPT2)	15 min pour 100 epochs
Flow-based	/
Diffusion	/

TABLE 4 – Comparaison des modèles dans la phase de génération

Conclusion

Modèle	Ressources (GPU)	Qualité des images
StyleGAN		
VAE		
VQ-VAE-2 (PixelCNN)		
VQ-VAE-2 (GPT2)		erreur
Sylvester Flows		erreur
Denoising Diffusion Probabilistic Model		

Merci pour votre attention

Bibliographie

Generative Adversarial Networks, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio
A Style-Based Generator Architecture for Generative Adversarial Networks, Tero Karras, Samuli Laine, Timo Aila

Auto-Encoding Variational Bayes, Diederik P. Kingma, Max Welling (2014)
Vector Quantised-Variational AutoEncoder, Van Den Oord et al. (2017)
Generating Diverse High-Fidelity Images with VQ-VAE-2, Ali Razavi et al. (2019)

Self Normalizing Flows, T. Anderson Keller, Jorn W.T. Peters, Priyank Jaini, Emiel Hoogeboom, Patrick Forré, and Max Welling (2020)
Sylvester Normalizing Flows for Variational Inference, Rianne van den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling (2018)

Denoising Diffusion Probabilistic Models, Ho et al. (2020)
Improved Denoising Diffusion Probabilistic Models, Nichol & Dhariwal (2021)

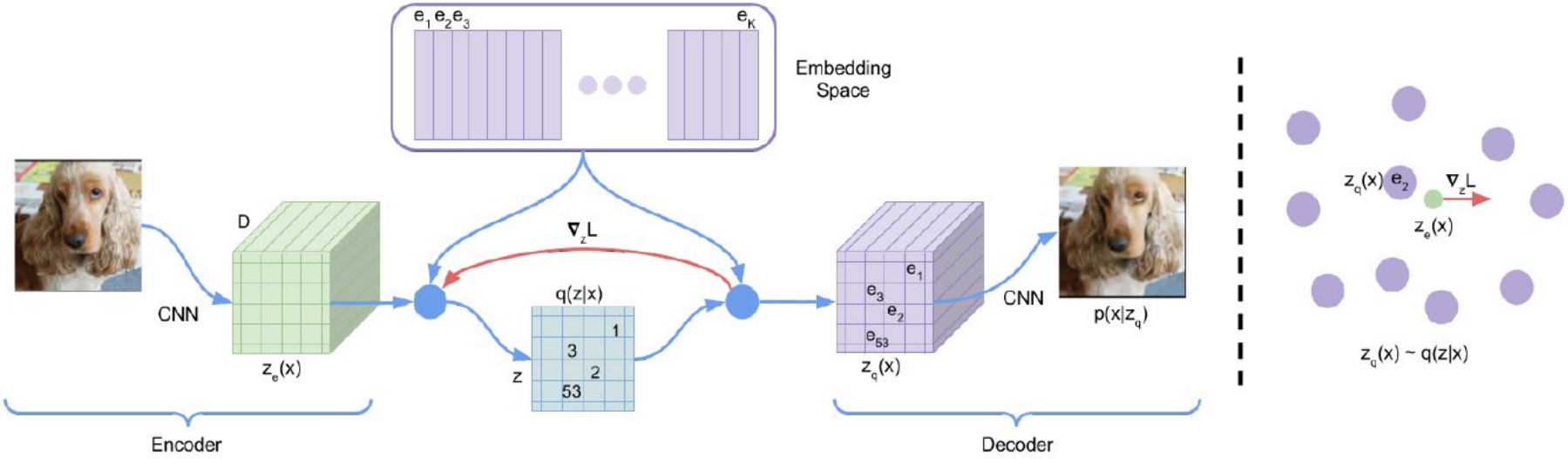
Annexes : Formules du VAE

$$z \sim q(z|x) = \mu + \sigma * N(0,1)$$

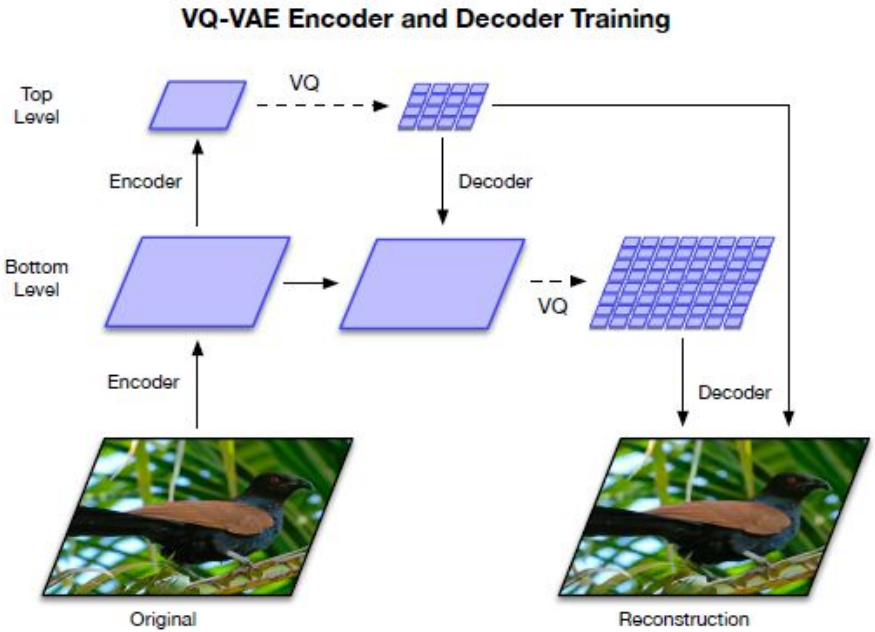
$$\underbrace{D_{KL}(q(z|x)||p(z|x))}_{divergence\ KL} + \underbrace{E_q(log(p(x|z))) - E_q(log(\frac{q(z|x)}{p(z)}))}_{ELBO} = \underbrace{log(p(x))}_{constante}$$

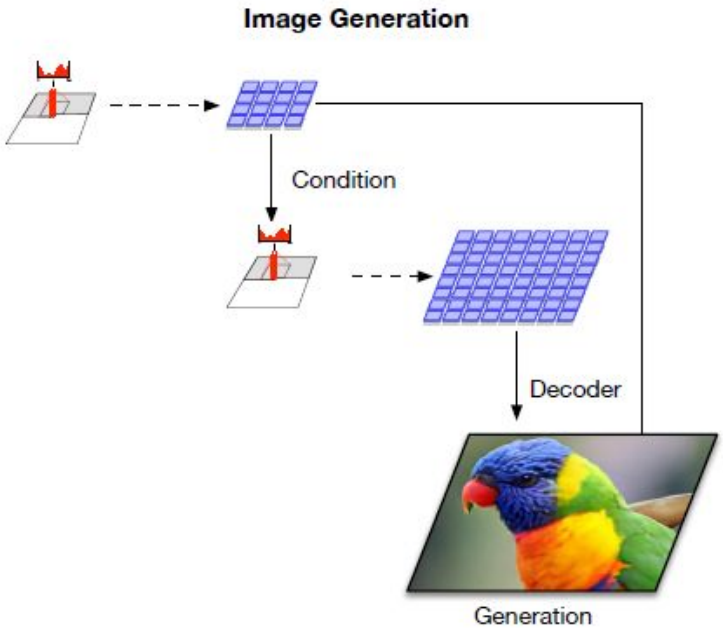
$$E_q = \frac{1}{N} \sum q(x)$$

Annexes : VQ-VAE (papier fondateur)

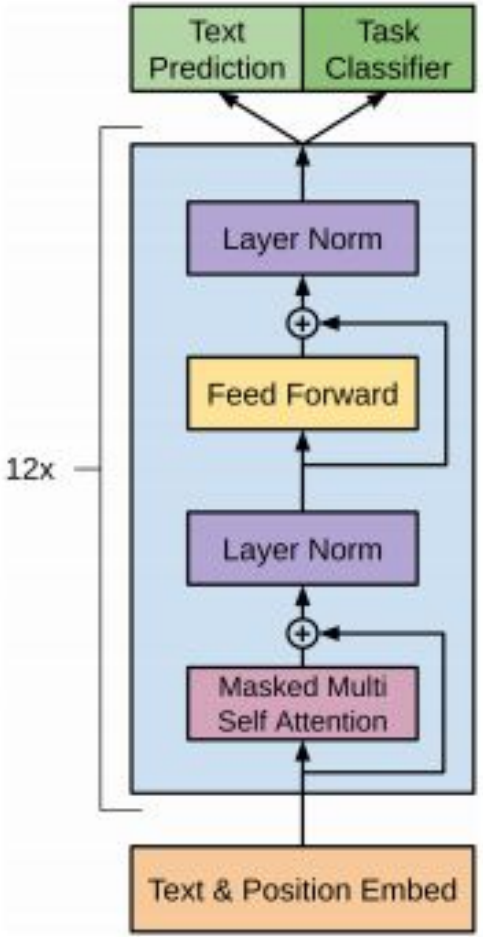


Annexes : VQ-VAE-2 (papier fondateur)

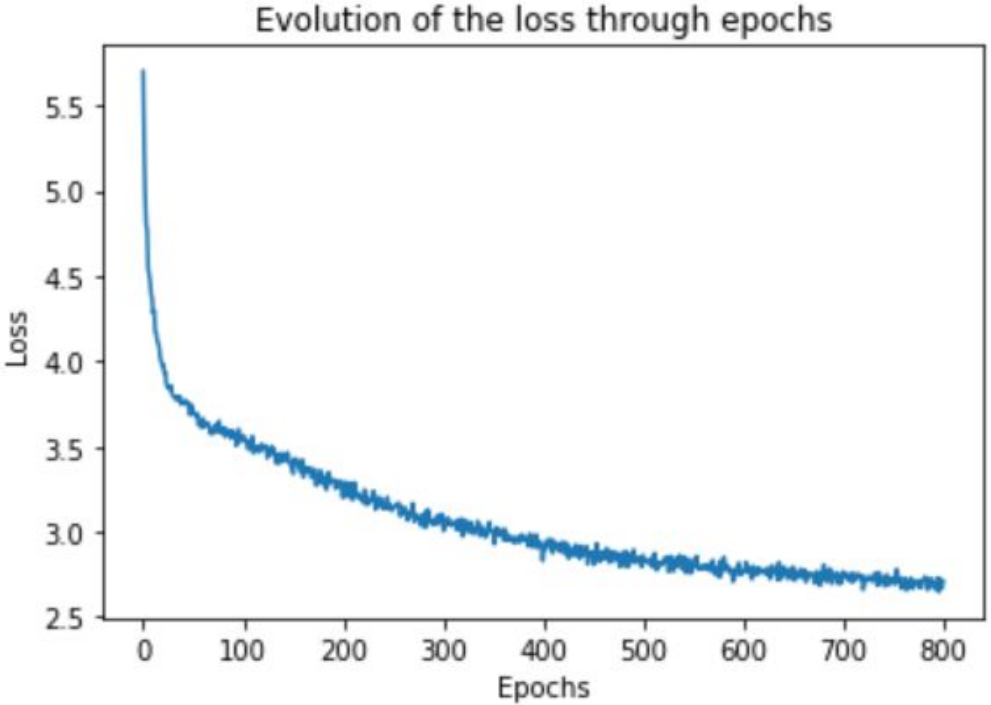




Annexes : GPT2



Annexes : Entraînement GPT2 sur les indices du VQ-VAE-2



Erreur lors de la génération des images :

CUDA error: device-side assert triggered
CUDA kernel errors might be asynchronously reported at some other API call

Annexes : VQ-VAE : difficultés et alternatives

- Réalisation des calculs sur le **CPU** pour essayer d'obtenir un message d'erreur plus explicite
- Libération de la **mémoire** du GPU dès qu'il était possible
- Sauvegarde des paramètres des modèles dans des **checkpoints** pour libérer de l'espace mémoire sur le GPU
- Modification des **hyperparamètres** (taille des embeddings, taille des batch, nombre de layers, nombre de codebooks utilisés...) du transformer
- Changement d'en**vironnement de développement** : Google Colab, Kaggle et le cluster de l'ENSTA
- Changement du **dataset** d'entrainement : passage du dataset Stanford Dogs (images en 64x64) au dataset CIFAR-10 (images en 32x32)
- Changement de **modèle** : passage du VQ-VAE-2 au VQ-VAE
- Changement de **framework** : passage de Pytorch à Tensorflow

Annexes : DDIM

Figure 3. Latent samples from linear (top) and cosine (bottom) schedules respectively at linearly spaced values of t from 0 to T. The latents in the last quarter of the linear schedule are almost purely noise, whereas the cosine schedule adds noise more slowly