

Norayda N'SIEMO

Kamelia SIAM

Idriss RHERMINI

Rapport Projet Unsupervised Learning

Plan

1. Présentation des modèles
2. Implémentation et résultats
3. Répartition des tâches et conclusion

1. Présentation des modèles

Il nous a été demandé d'implémenter from scratch les algorithmes suivants :

- Kmeans
- PCA
- AutoEncoder
- GAN
- VAE
- SOM

Sur cette sélection de modèle nous avons réussi à implémenter l'algorithme des Kmeans, le PCA et AutoEncoder. Le Variational autoencoder ainsi que les autres modèles nous les avons commencés mais les résultats sont peu pertinents.

Pour chacun de ces modèles nous avons appliqué deux datasets différents le MNIST vu en cours et un dataset personnalisé au choix. Nous avons opté pour le dataset Simpson dont le lien est le suivant : <https://www.kaggle.com/datasets/alexattia/the-simpsons-characters-dataset?resource=download>

Il contient **20 933** images labelisés représentant les personnages de la série animé Les Simpson.

Un extrait de ce dataset serait :



Nous allons donc faire abstraction des labels de ce dataset et appliquer de la classification sur ces données.

I. KMeans

Pour réaliser l'implémentation du modèle Kmeans on s'est aidé du pseudo code suivant :

ALGORITHM

1. First, initialize the number of clusters, K (Elbow method is generally used in selecting the number of clusters)
2. Randomly select the k data points for centroid. A centroid is the imaginary or real location representing the center of the cluster.
3. Categorize each data items to its closest centroid and update the centroid coordinates calculating the average of items coordinates categorized in that group so far
4. Repeat the process for a number of iterations till successive iterations clusters data items into the same group

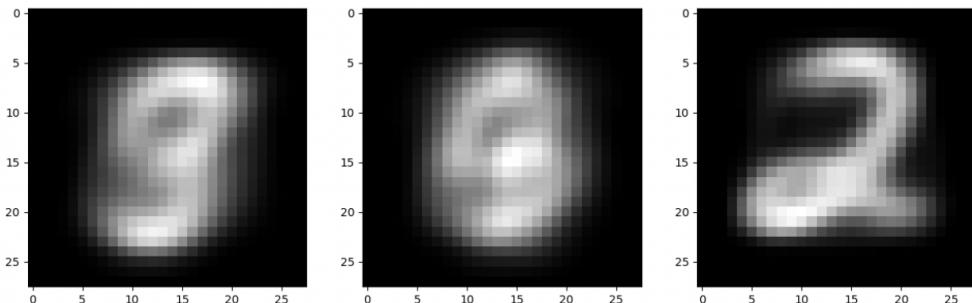
Un cluster est constitué de ce qu'on appelle un centroïde qui est au centre du cluster et de ses plus proches voisins. Le nombre de centroïdes est fixé au départ par la constante K.

Pour l'initialisation des clusters nous avons réalisé une randomisation des index choisissant aléatoirement les centroïdes. Puis pour appliquer l'algorithme on calcule la distance de tous les point avec les centroïdes et on choisit un nouveau centroïde qui sera la moyenne de tous les points les plus proches. On réalise cette boucle un certain nombre de fois jusqu'à obtenir une liste de centroïdes fixe.

Nous avons fait varier le nombre de centroïdes afin de déterminer le nombre moyen à partir duquel la représentation est optimale c'est-à-dire, les clusters contiennent une majorité remarquable d'une seule catégorie.

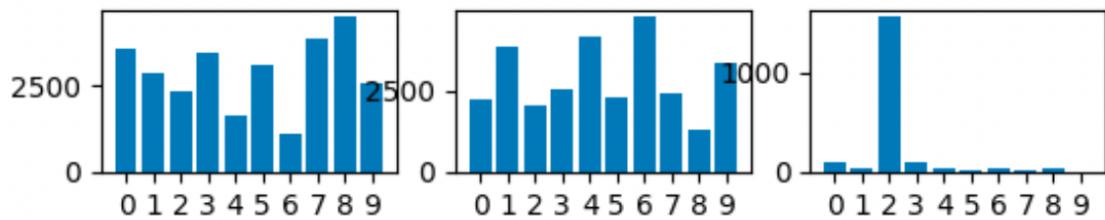
Pour K= 3 on obtient les représentant suivant

Représentants générés



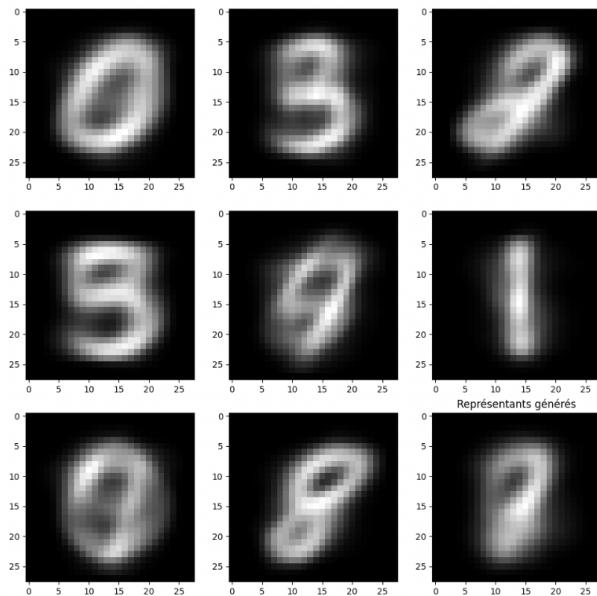
Le fait qu'il soit flou nous informe déjà que dans chaque cluster la répartition des catégories est grande. On peut visualiser cette répartition et on obtient :

Repartition du dataset pour chaque représentant



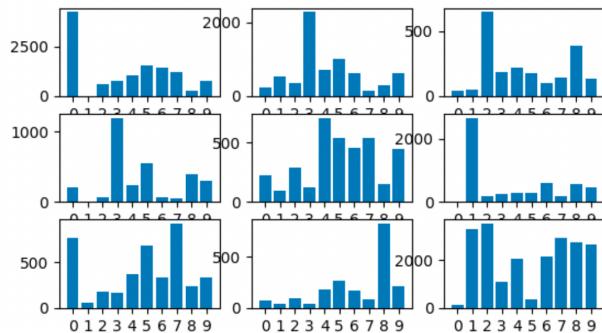
On remarque que pour le dernier représentant on a une image un peu moins flou que les autres et ça se voit dans la répartitions des catégories par cluster.

Pour K = 9 on obtient les représentant suivant



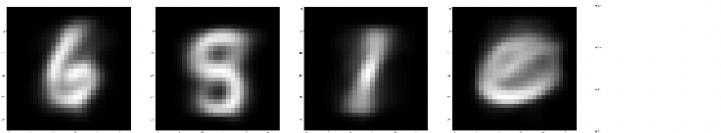
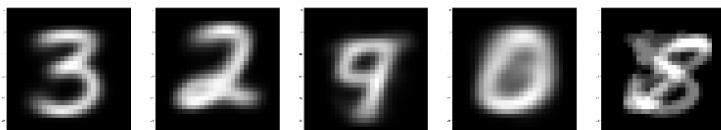
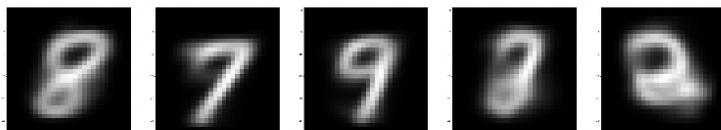
Étant donné que le choix des premiers centroïdes est aléatoire, on est tombé peu de fois avec K=9 à plus de 3 représentant significatif. Pour ces représentant là nous avons :

Repartition du dataset pour chaque représentant



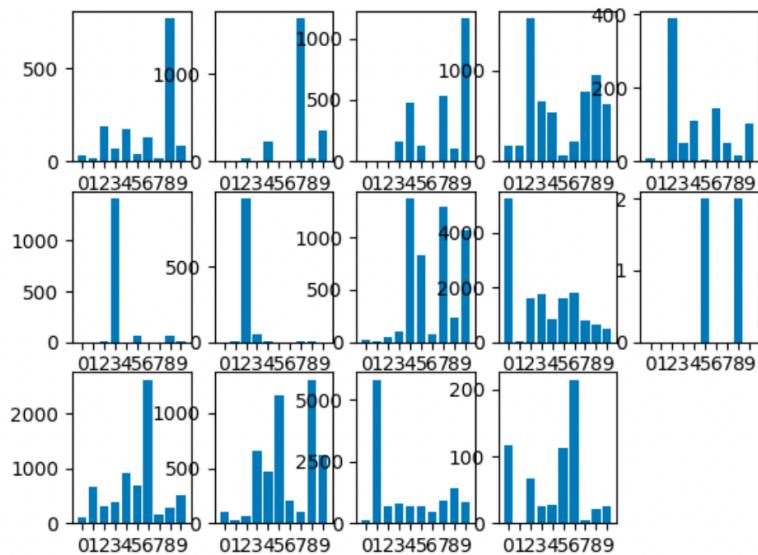
Après des tests sur différent K on obtient pour K = 14 les résultats les plus intéressants sur ce dataset

Représentant K=14



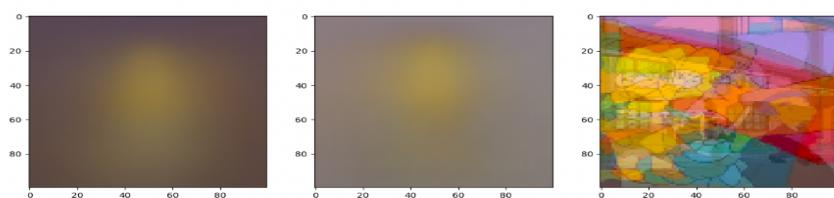
Les images des représentants sont plus claire et ça se voit dans la répartition des catégories

Repartition du dataset pour chaque représentant

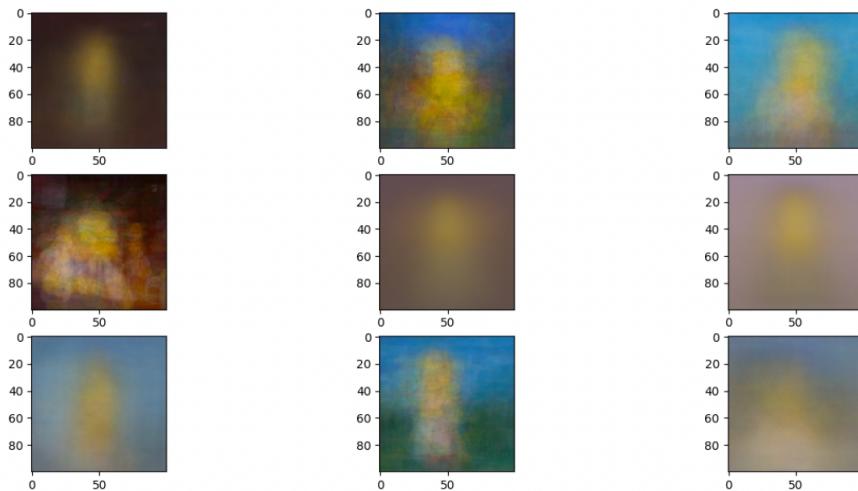


Pour $K>14$ on retrouve plusieurs représentant avec une majorité de 2 catégories dans le cluster.

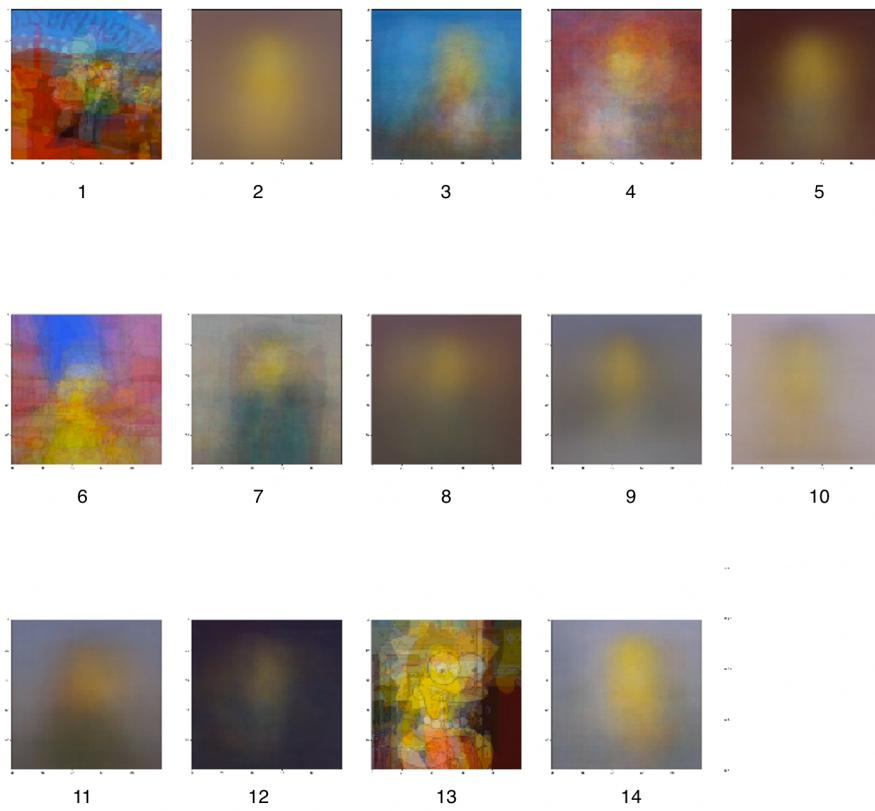
Sur notre dataset personnalisé, on retrouve par exemple pour $K=3$



Pour K= 9

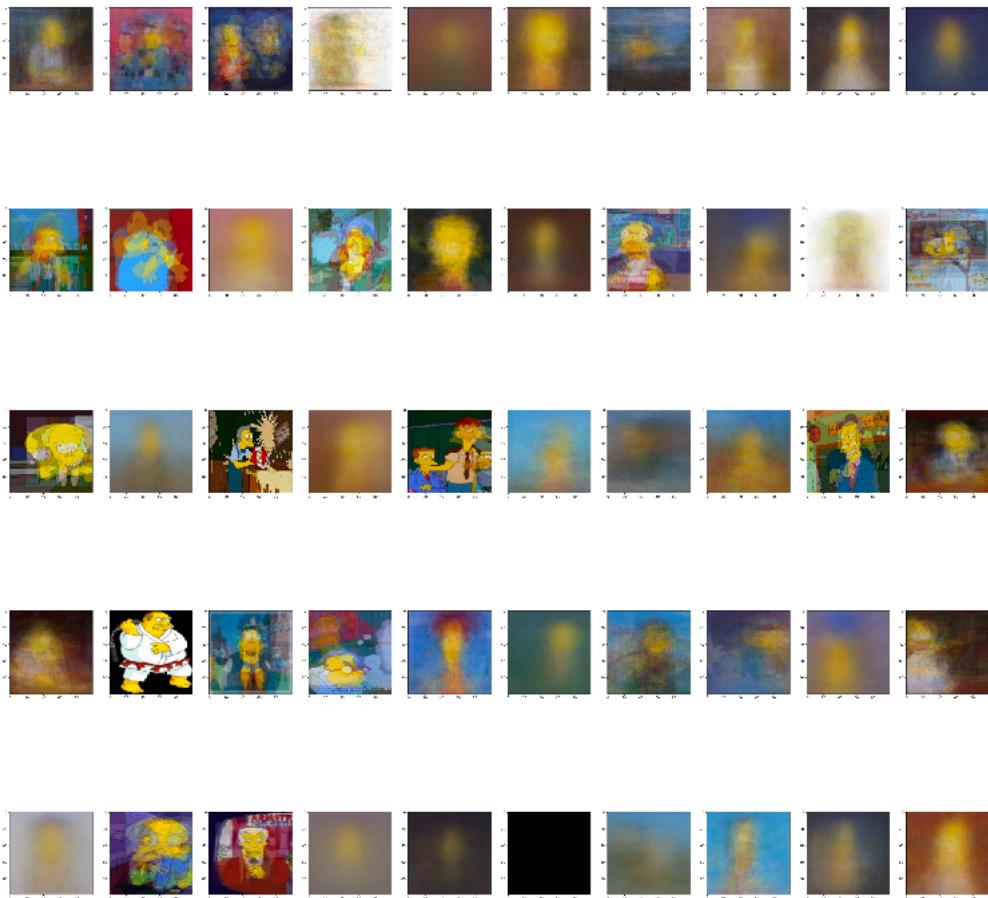


Pour K= 14



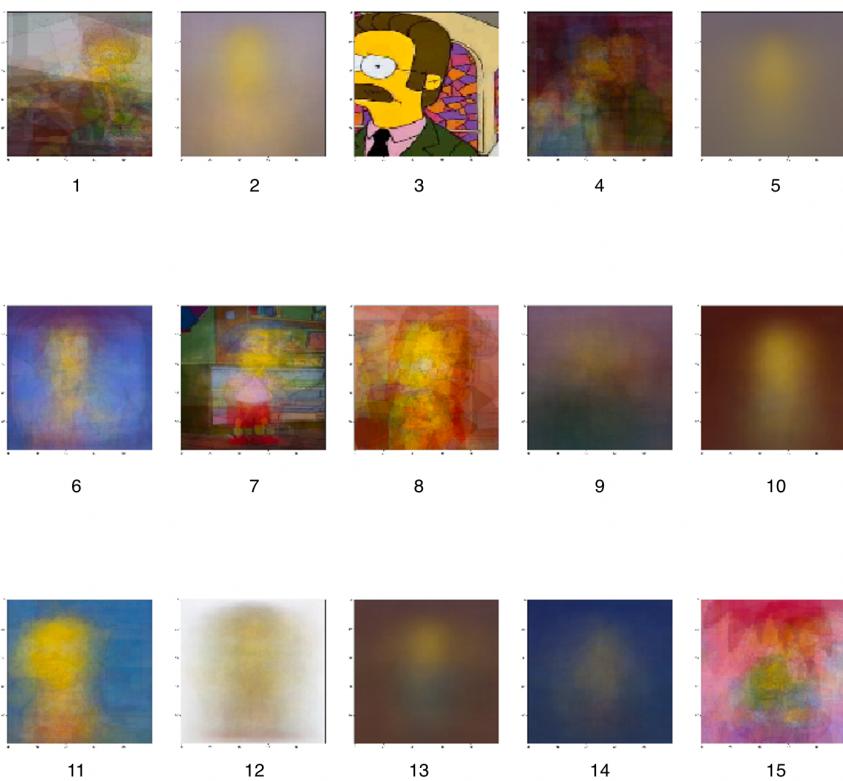
On peut déjà supposé que dans le cluster 13, la majorité des images représente Lisa Simpson et dans le cluster 6 Marge Simpson est en majorité.

On peut supposer que pour avoir une meilleure clusterisation il nous faut un plus grand nombre de cluster. Pour K = 50 on remarque déjà que l'entraînement prend plus de temps, pour les précédents résultats les centroïdes se stabilisaient à moins de 5 epochs. Pour ce test, après 200 epochs sur 200 on obtient les représentants suivants :



On voit tout de suite qu'on reconnaît beaucoup plus de personnages que pour un nombre de centroïdes plus bas. On pourrait se demander ce qu'on obtiendrait avec un nombre d'epoch et de cluster plus important.

Nous avons essayé de voir le pourcentage de représentation de la catégorie maximale dans un cluster afin de voir si le cluster était pertinent. On trouve par exemple pour $K = 15$ les résultats suivants



On retrouve bien dans le tableau ci-dessous les personnages correspondant(Ned flanders = représentant 3 (index 2 sur le tableau), Milhouse = représentant 7 (index6sur le tableau) notre clustering fonctionne plutôt bien.

	personnage	pourcentage
0	edna_krabappel	25.0
1	homer_simpson	22.48062015503876
2	ned_flanders	100.0
3	sideshow_bob	28.57142857142857
4	homer_simpson	10.773734838979507
5	sideshow_bob	33.33333333333333
6	milhouse_van_houten	60.71428571428571
7	bart_simpson	36.36363636363637
8	charles_montgomery_burns	14.018691588785046
9	moe_szyslak	14.22855721393035
10	lisa_simpson	34.54545454545455
11	lisa_simpson	19.46107784431138
12	charles_montgomery_burns	12.126642771804063
13	marge_simpson	14.713896457765669
14	sideshow_bob	21.73913043478261

II. PCA

a. Étapes de l'algo :

- Centrer les x
- Calculer les valeurs propres et vecteurs propres

- Aligner les vecteurs et les valeurs propres
- Prendre les x les plus importants
- Appliquer sur un dataset

b. Résultats sur MNIST

Après avoir essayé d'encoder et décoder on obtient :



c. Résultat sur nos données

III. AutoEncoder

a. Résultats sur MNIST

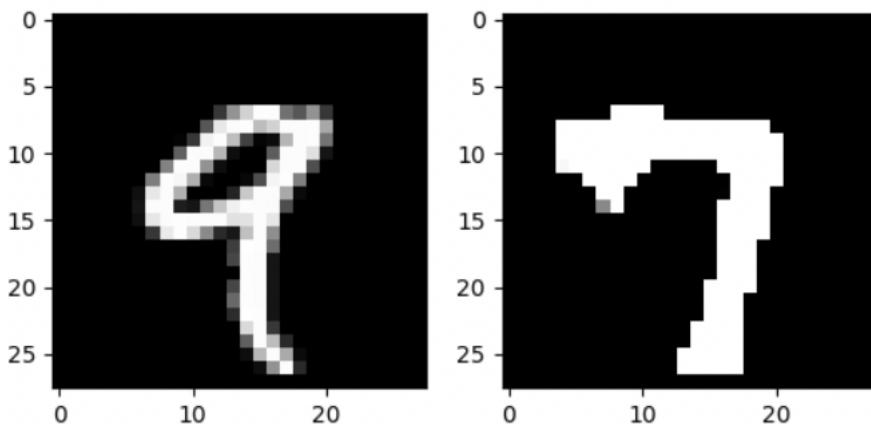
Nous avons premièrement entrainé un autoencoder avec les paramètres suivants :

Units des couches Dense de l'encoder =[52,16,8]

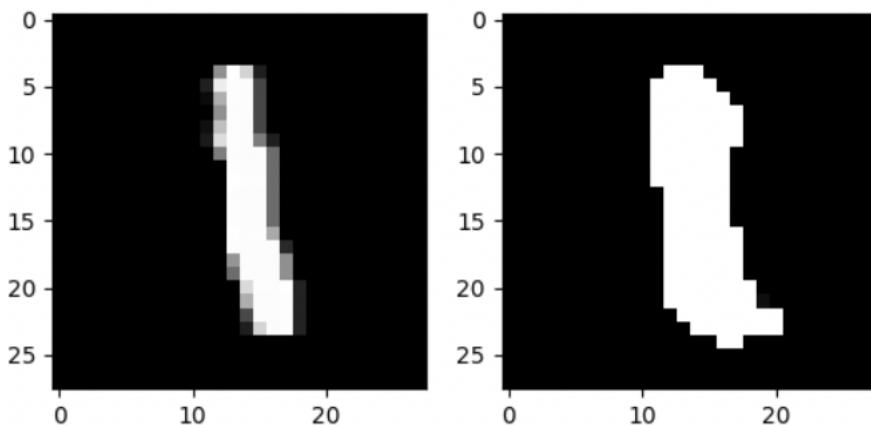
Espace latent = 2

Les units des couches Dense du decoder étant l'inverse de celles de l'encoder

On trouve l'output ci-dessous (à droite l'image dans le dataset, à droite l'image décodé par notre modèle)



En modifiant la dimension de l'espace latent à 100 avec les units [512,324,64] on obtient :



Ayant pris un léger retard nous n'avons pas pu faire plus de tests et tous les tests ont été réalisé avec un epochs = 10, lr = 0.01, loss = MSE, activation= 'relu' (sauf la dernière couche du decoder qui a une activation= 'sigmoid '). Mais nous avons bien saisie le fonctionnement et l'utilité de ce modèle.

b. Résultat sur le dataset personnalisé

Afin de terminer le flow d'entraînement de tous les modèles et faire le maximum de tests possible nous avons réduit le dataset aux 3 premières catégories ce qui nous fait un total de 1056 images sur notre dataset personnalisé.

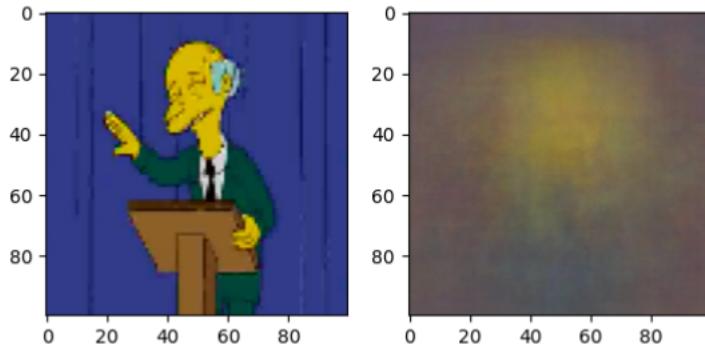
Nous avons également adapté le modèle initial afin qu'il s'adapte aux différentes taille de données.

Sur notre dataset personnalisé on retrouve les résultats suivants :

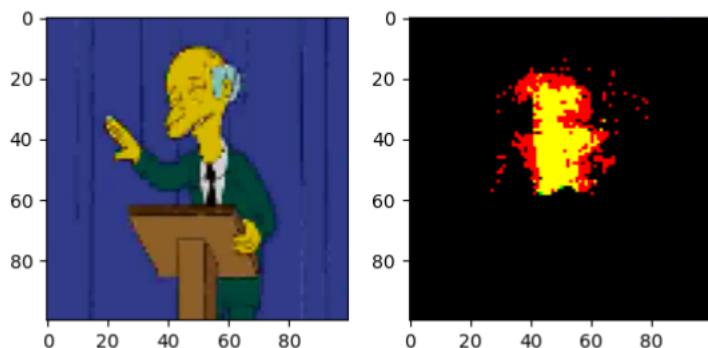
Avec

Dimension espace latent = 10 ,

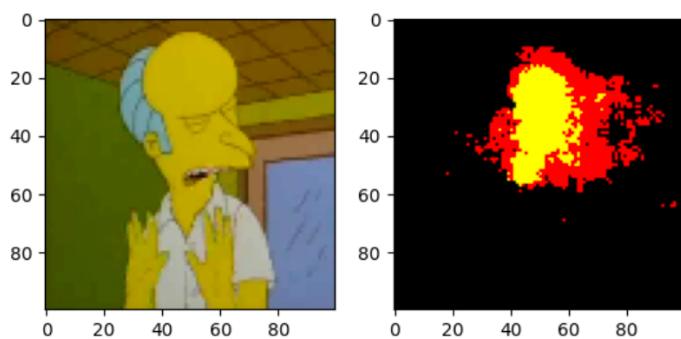
10 epochs,
lr = 0,01,
batch_size =32
et units = [58, 36, 28]



On constate que l'output est très flou, en augmentant considérablement la dimension de l'espace latent et les units on obtient :
Pour esp_latent = 150 et Units = [580, 360, 280]



On a fait le test avec la même configuration pour une image différente on obtient :



Pour la dimension de l'espace latent = 25 on obtient

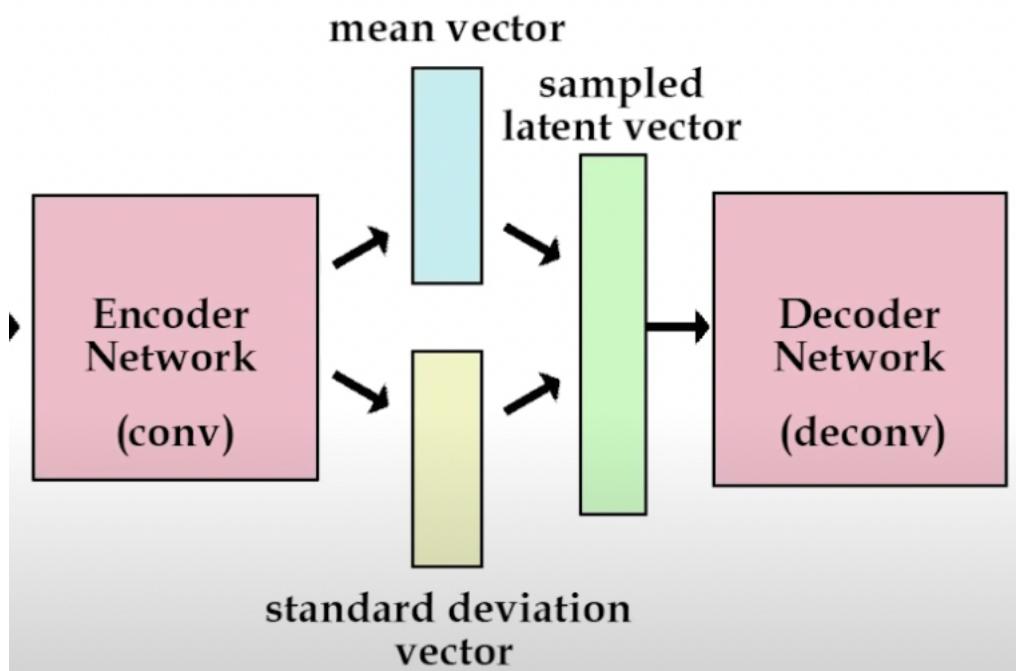


On pourrait au vu de nos résultats précédent penser que la dimension idéale pour l'espace latent est strictement inférieure à 100 ou que ce modèle n'est juste pas adapté pour des images de plusieurs couleurs. Etant donné que les résultats sur le dataset Mnist (noir et blanc) étaient plutôt intéressant.

A présent nous allons vous mettre notre compréhension de l'algorithme à faire vous trouverez dans le rendu l'implementation de chaque modèle par classe.

IV. VAE

La différence entre un autoencoder et un Variational autoencoder se rouve au niveau de l'architecture, le VAE possède une étape supplémentaire :



On part donc du code effectué pour l'Auto encoder et on remplace le layer correspondant à l'espace latent par 2 layers Dense:

```
means = tf.keras.layers.Dense(self.latent_dim, activation="linear", name="means")
logvar = tf.keras.layers.Dense(self.latent_dim, activation="linear", name="logvar")
```

Ces deux layers seront par la suite concaténer dans notre decoder.

L'algorithme a été correctement implémenté, le training se lance et se termine mais nous avons des erreurs au moment d'encoder les images.

V. SOM

Comment fonctionne SOM?

Disons une donnée d'entrée de taille (m, n) où m est le nombre d'exemples d'apprentissage et n est le nombre d'entités dans chaque exemple. Premièrement, il initialise les poids de taille (n, c) où c est le nombre de clusters. Ensuite, en itérant sur les données d'entrée, pour chaque exemple d'entraînement, il met à jour le vecteur gagnant (vecteur de poids avec la distance la plus courte de l'exemple d'entraînement).

L'objectif est de produire un regroupement de manière que les individus situés dans la même case soient semblables, les individus situés dans des cases différentes soient différents.

Dans le cas du dataset du MNIST, on obtient le résultat suivant sur 1000 epochs

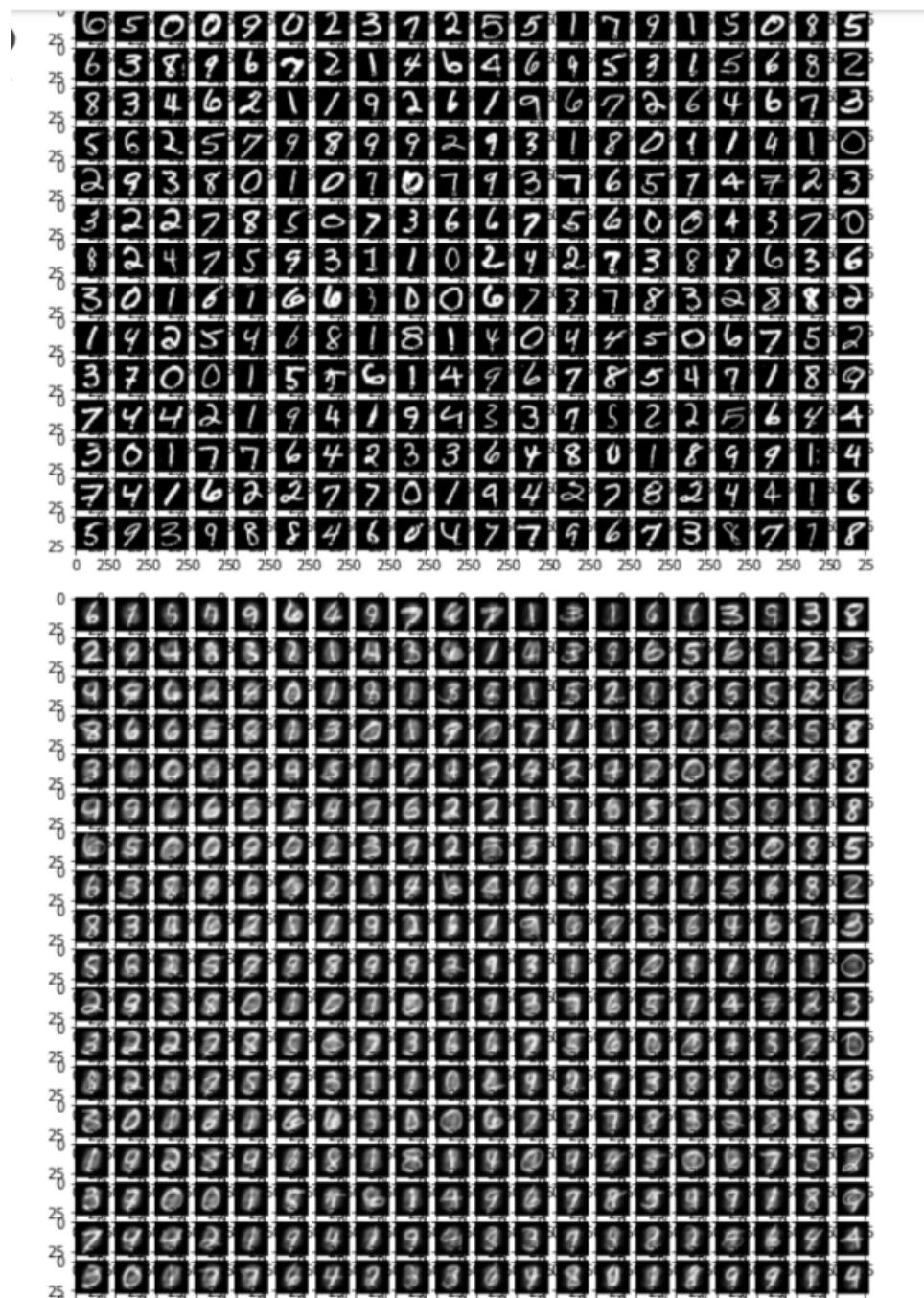
model.display_grid((28,28), float, True)

9	6	8	0	1	4	9	6	3	0	6	6	9	9	1	2	7	9	7	2
2	1	4	8	3	5	8	0	4	2	1	4	3	8	1	3	9	9	7	6
0	6	9	8	9	5	0	2	2	2	2	1	1	6	9	4	7	0	9	6
4	4	3	8	1	7	2	6	3	7	9	2	3	9	3	1	1	1	5	3
0	1	0	9	2	3	7	0	7	5	5	7	3	5	9	9	2	7	0	+
7	7	5	2	3	5	6	6	4	1	7	7	4	8	0	2	6	2	8	0
6	1	7	8	8	7	8	9	9	3	7	5	5	6	4	7	3	2	6	2
8	1	6	0	7	4	7	6	2	1	8	7	3	5	7	0	8	3	8	8
4	0	8	1	3	5	0	2	7	9	3	5	0	9	0	1	1	5	0	4
0	5	6	0	3	9	2	4	9	0	7	8	7	3	6	5	7	2	6	2
2	1	6	7	9	7	8	4	0	4	3	2	5	6	3	8	2	5	8	5
5	2	4	9	0	8	1	2	0	5	8	2	5	6	5	8	3	3	4	9
8	6	3	7	8	4	9	8	3	7	7	5	1	2	7	3	2	7	0	5
2	2	8	5	8	7	0	7	1	7	8	5	2	3	5	2	0	1	6	2
1	7	7	2	2	1	1	0	3	9	1	0	1	5	2	3	6	1	1	3
1	9	1	1	2	0	5	3	0	0	6	3	9	4	3	8	9	7	1	9
3	2	9	9	7	1	1	4	6	5	6	2	8	1	2	0	1	0	3	5
0	4	1	0	1	5	2	1	7	8	4	2	2	8	0	7	0	1	6	0
4	3	0	7	9	7	2	8	6	3	6	9	5	9	2	8	6	0	1	8
9	7	8	2	2	6	4	2	3	8	6	5	9	0	9	0	8	9	9	9
9	6	8	0	1	4	9	6	3	0	6	6	9	9	1	2	7	9	7	2
2	1	4	8	3	5	8	0	4	2	1	4	3	8	1	3	9	9	7	6
0	6	9	8	9	5	0	2	2	2	2	1	1	6	9	4	7	0	9	6
4	4	3	8	1	7	2	6	3	7	9	2	3	9	3	1	1	1	5	3
0	1	0	9	2	3	7	0	7	5	5	7	3	5	9	9	2	7	0	+
7	7	5	2	3	5	6	6	4	1	7	7	4	8	0	2	6	2	8	0
6	1	7	8	8	7	8	9	9	3	7	5	5	6	4	7	3	2	6	2
8	1	6	0	7	4	7	6	2	1	8	7	3	5	7	0	8	3	8	8
4	0	8	1	3	5	0	2	7	9	3	5	0	9	0	1	1	5	0	4
0	5	6	0	3	9	2	4	9	0	7	8	7	3	6	5	7	2	6	2
2	1	6	5	0	7	5	4	0	4	3	5	6	3	8	2	6	5	6	0

On remarque que dans la sortie sur 1000 epochs les représentants ne sont pas semblables entre eux. Ensuite, on va augmenter le nombre d'epochs pour avoir des résultats mieux.

Vous trouverez l'ensemble des résultats et le code effectué pour les avoir sur le notebook KOHONEN.ipynb

Rechercher + Texte

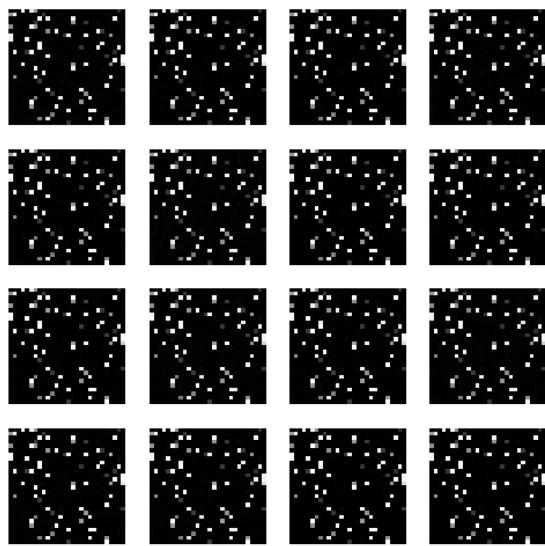


Comme vous constatez, sur l'image ci-dessus, on voit bien qu'on a un peu d'améliorations. Certes que l'exécution a pris un peu plus de temps, c'est à cause epochs = 3000. Mais on a des résultats meilleurs.

VI. GAN

GAN Algo :

- Initialiser le generator
- Initialiser le discriminator
- Définir leur loss sur des vrai et fausses données
- Entrainement • Generation d'images



3. Répartition des tâches et conclusion

Tout au long de cette semaine thématique nous avons pu mettre en application différents algorithmes que nous ne connaissions pas, à première vue ils nous ont donné du fil à retordre, ils nous ont fait peur même, mais en se répartissant les différents modèles à implémenter on a pu optimiser notre temps et essayer de toucher à tout.