UNIVERSITÉ PARIS DESCARTES

# Apprentissage Profond pour la Reduction de Dimension


**Réalisé par :**

- Ikram KAROUCHE
- Majda OULAARBI

**Responsable :**

- Pr. Mohamed NADIF

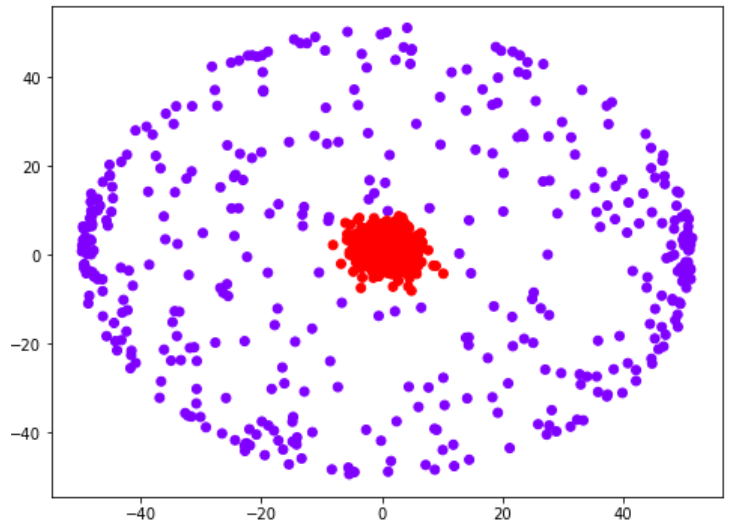
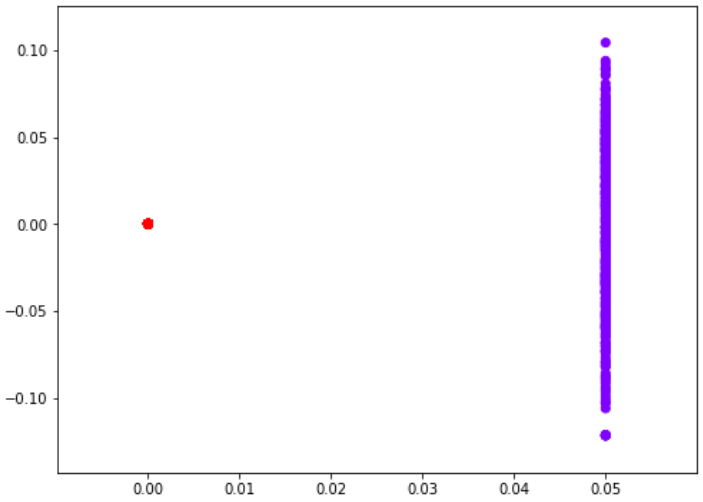
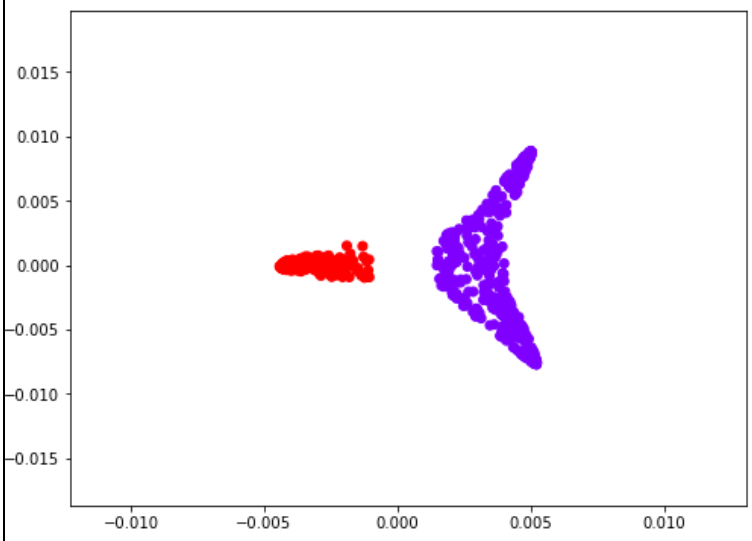
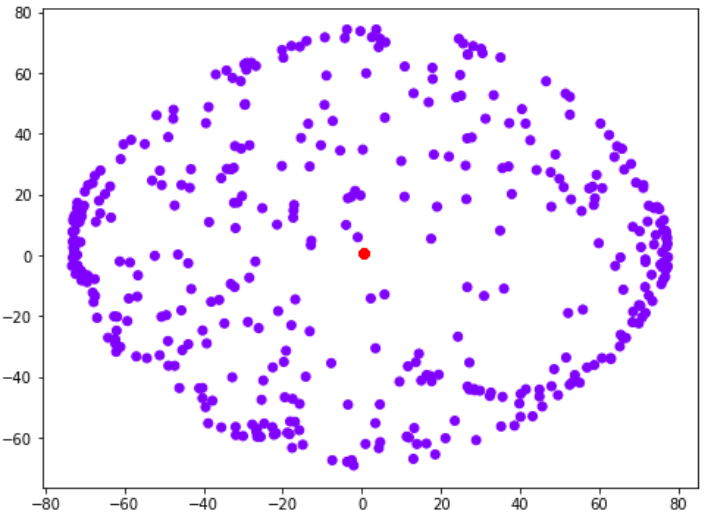

**Master Machine Learning pour la science de données**

**Année 2019-2020**

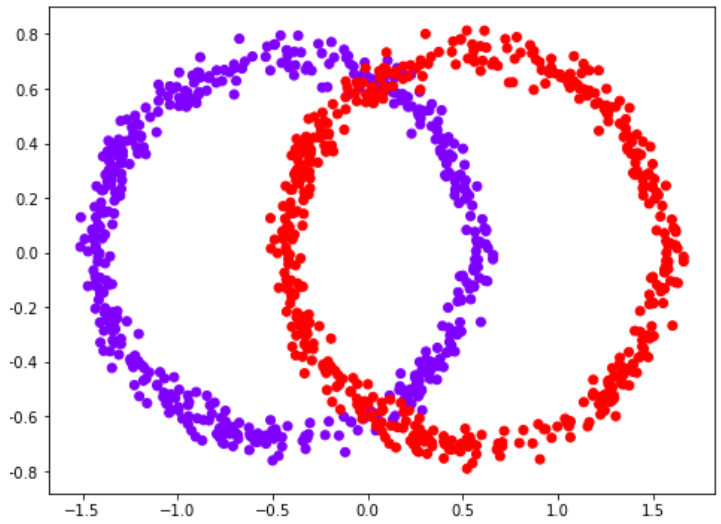
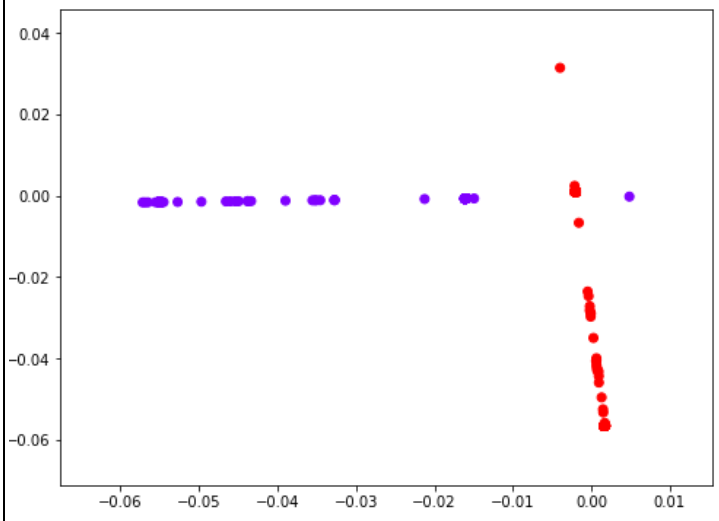
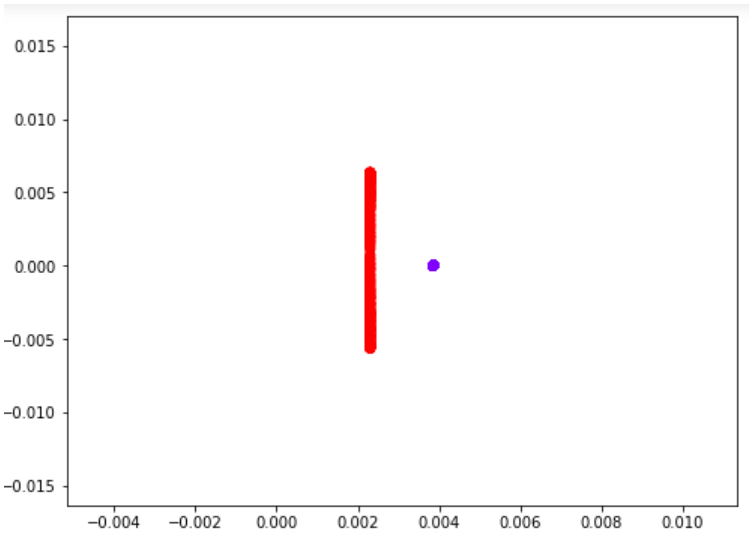
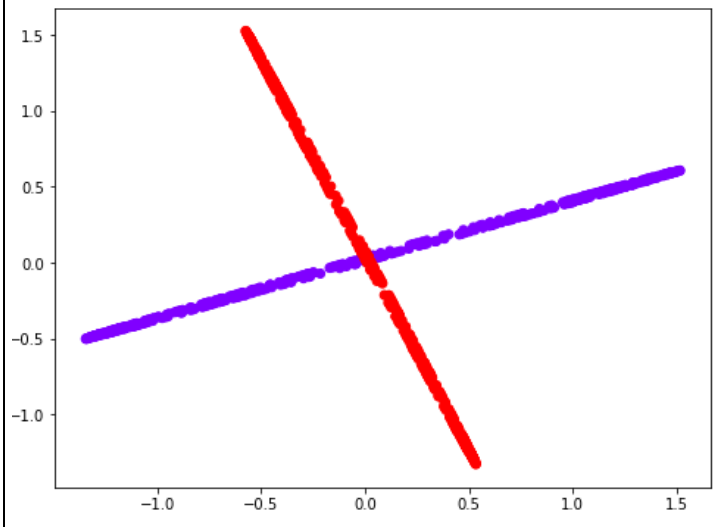
# I. Données FCPS

Dans cette partie, nous présentons les résultats des méthodes de réduction de dimensions : ACP, LLE, Spectral embedding et ISOMAP.

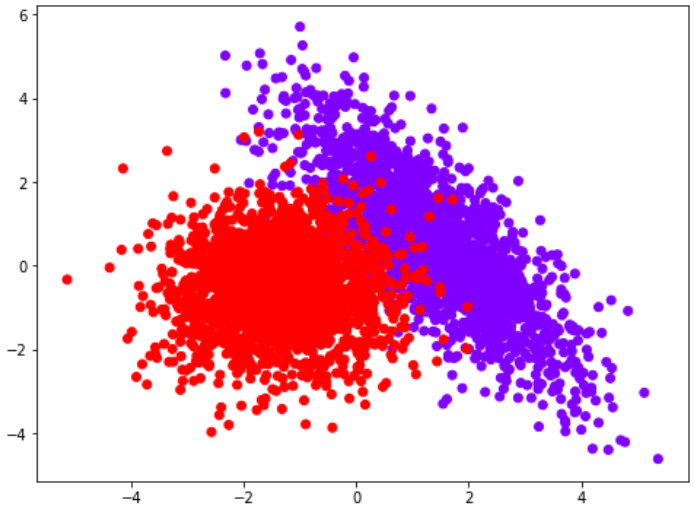
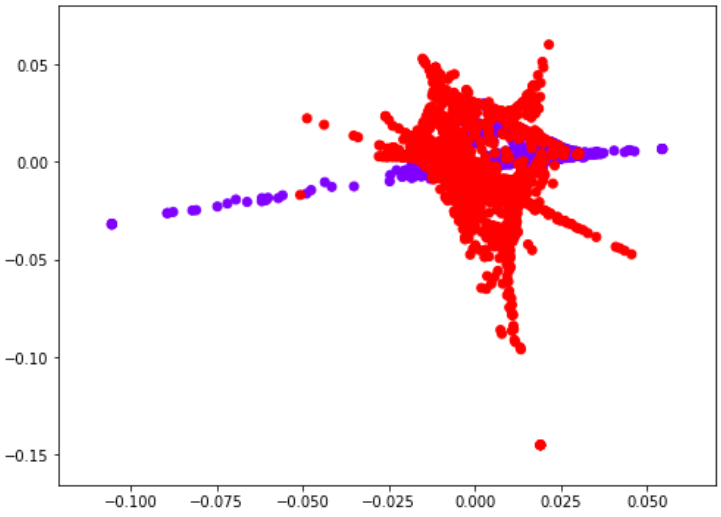
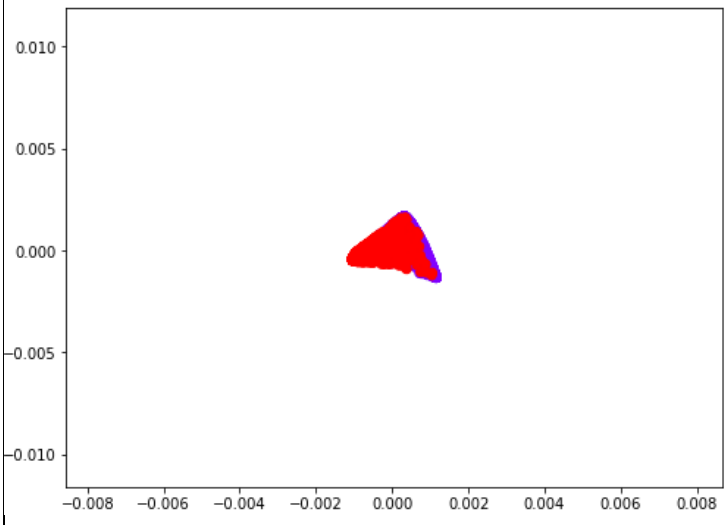
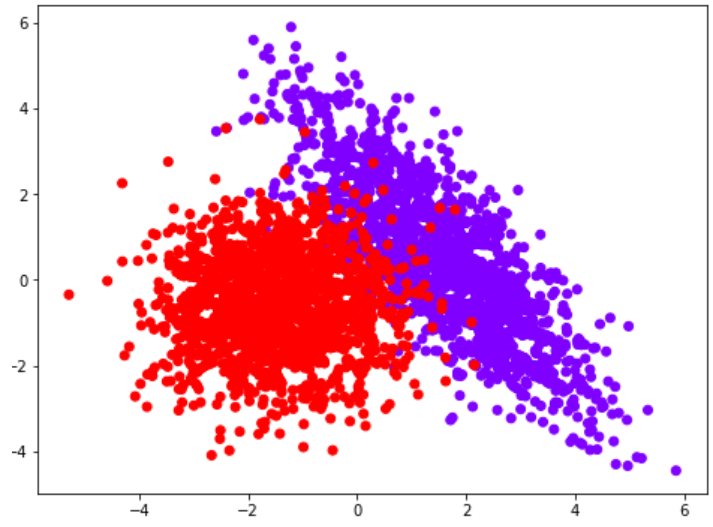
## A. Atom data

| ACP | Local Linear Embedding (LLE) |
|---|---|
|  |  |
| **Spectral Embedding** | **ISOMAP** |
|  |  |

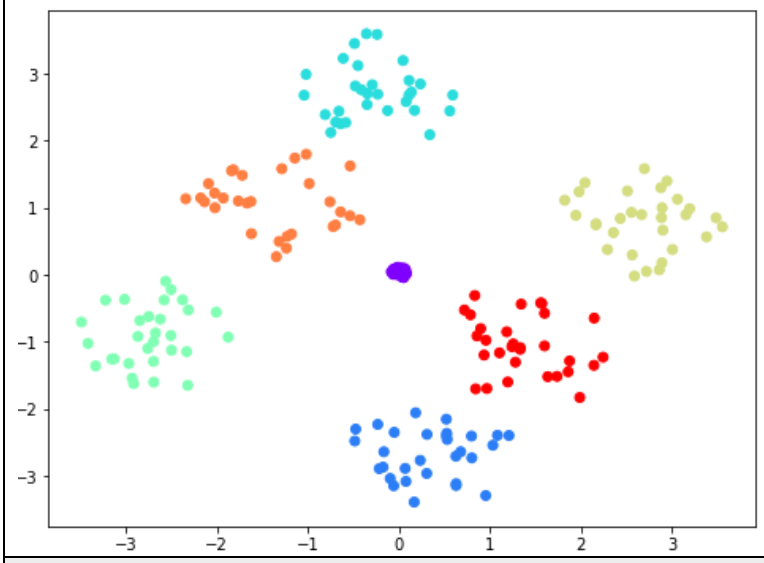
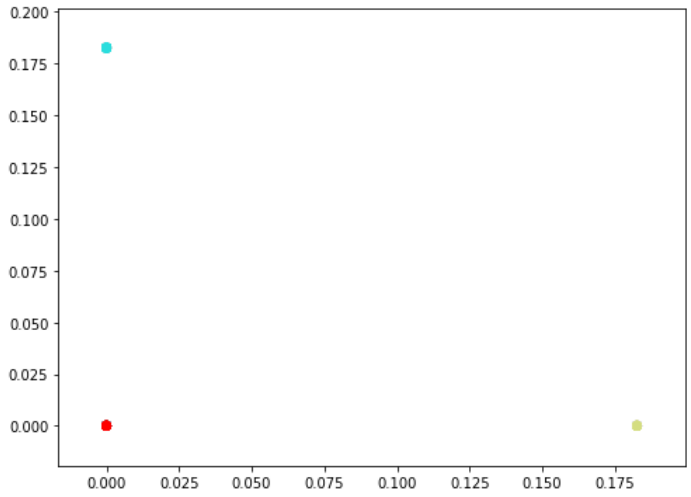
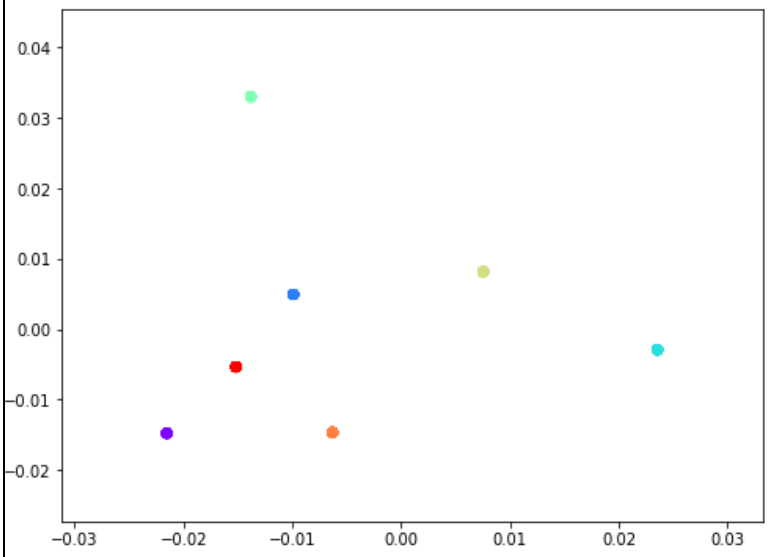
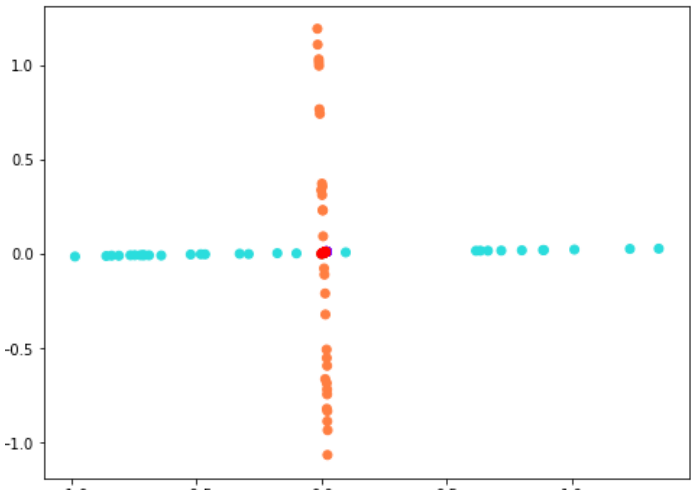
## B. ChainLink data

| ACP | Local Linear Embedding (LLE) |
|---|---|
|  |  |
| **Spectral Embedding** | **ISOMAP** |
|  |  |

## C. EngyTime data

| ACP | Local Linear Embedding (LLE) |
|---|---|
|  |  |
| **Spectral Embedding** | **ISOMAP** |
|  |  |

## D. Hepta data

| ACP | Local Linear Embedding (LLE) |
|---|---|
|  |  |
| **Spectral Embedding** | **ISOMAP** |
|  |  |

# II. Combinaisaon Auto-encodeur & LLE

L'algorithme que nous avons implémenté consiste à combiner le deep autoencodeur et la méthode Local Linear Embedding (LLE) afin d'avoir la meilleure représentation possible (avec Deep AE) et un graphe de similarité caractérisant au mieux la proximité entre les points (avec LLE). La fonction objective à optimiser est la suivante :

$$\min_{\theta_1,\theta_2,\mathbf{S}} \|\mathbf{X} - g_{\theta_2}(f_{\theta_1}(\mathbf{X}))\|^2 + \lambda \|f_{\theta_1}(\mathbf{X}) - \mathbf{S} f_{\theta_1}(\mathbf{X})\|^2$$

Nous avons évalué cet algorithme sur trois ensembles de données : Coil 20, Coil100 et MNIST.

Le paramètre de régularisation lambda est égal à 0.02. L'algorithme converge lorsque la fonction objective atteint un seuil 0.0001 ou bien lorsque le pas d'amélioration ne dépasse pas 0.0001. Le nombre maximum d'itérations est égal à 30.

# III. Evaluation de l'algorithme

## A. Les données Coil20

Il s'agit de 720 images en noir et blanc (niveaux de gris) de 20 objets. Les objets ont été placés sur une plaque tournante. La table tournante a été tournée de 360 degrés pour varier la pose des objets par rapport à la caméra fixe. Les images des objets ont été prises à un intervalle de pose de 5 degrés, ce qui correspond à 72 images par objet. En total ça fait 1440 images de dimension (128 x 128).

| Type | Nombre d'objets | Dimensions | Features |
|---|---|---|---|
| Gray-scale Images | 1440 | (128 x 128) | 16384 |

Nous avons subdivisé les données en échantillons d'apprentissages (64%), échantillons de validation (16%) et échantillons de test (20%).

| Apprentissage | Validation | Test |
|---|---|---|
| 921 | 1231 | 288 |

### 1. Le paramétrage de l'algorithme

L'auto-encodeur appliqué sur ces données est composé de 2 couches comme le montre la figure suivante.

Après plusieurs tests de l'algorithme LLE, nous avons choisi le nombre de voisins (k = 30).

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_8 (InputLayer) | (None, 16384) | 0 |
| dense_19 (Dense) | (None, 4096) | 67112960 |
| dense_20 (Dense) | (None, 1024) | 4195328 |
| dense_21 (Dense) | (None, 4096) | 4198400 |
| dense_22 (Dense) | (None, 16384) | 67125248 |

Total params: 142,631,936
Trainable params: 142,631,936
Non-trainable params: 0

*Figure 1 : Architecture de l'auto-encodeur utilisé pour la compression des données coil20*

## 2. Evaluation de l'algorithme

Une première application de l'auto-encodeur (epochs = 50 et batch = 256) sur les données de test a donnés une qualité de visualisation faible comme le montre la figure suivante.

*Figure 2 : Le résultat de visualisation des images reconstruites avec l'auto-encodeur*

Nous avons injecté la représentation encodée à l'algorithme LLE. La figure suivante illustre la distribution des images selon les deux premières composantes :

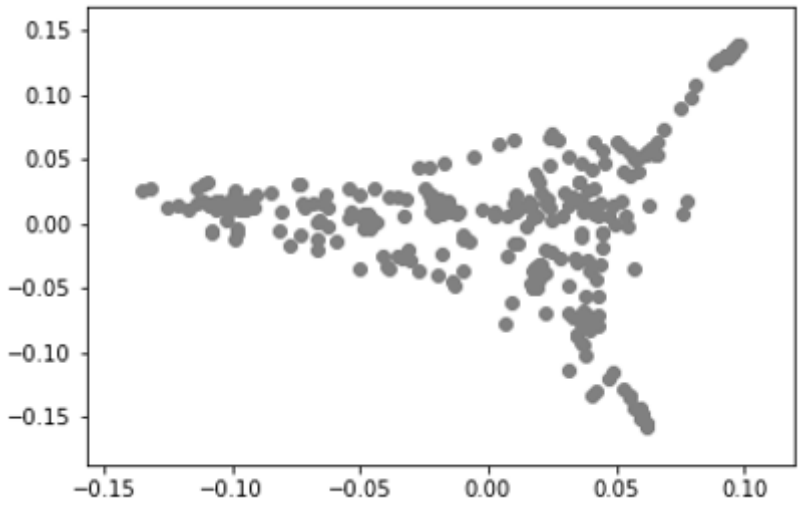



*Figure 3 : La distribution des images coil20 selon les deux premières composantes principales LLE*

La projection des images de test sur les deux premières composantes donne le résultat présenté dans la figure suivante

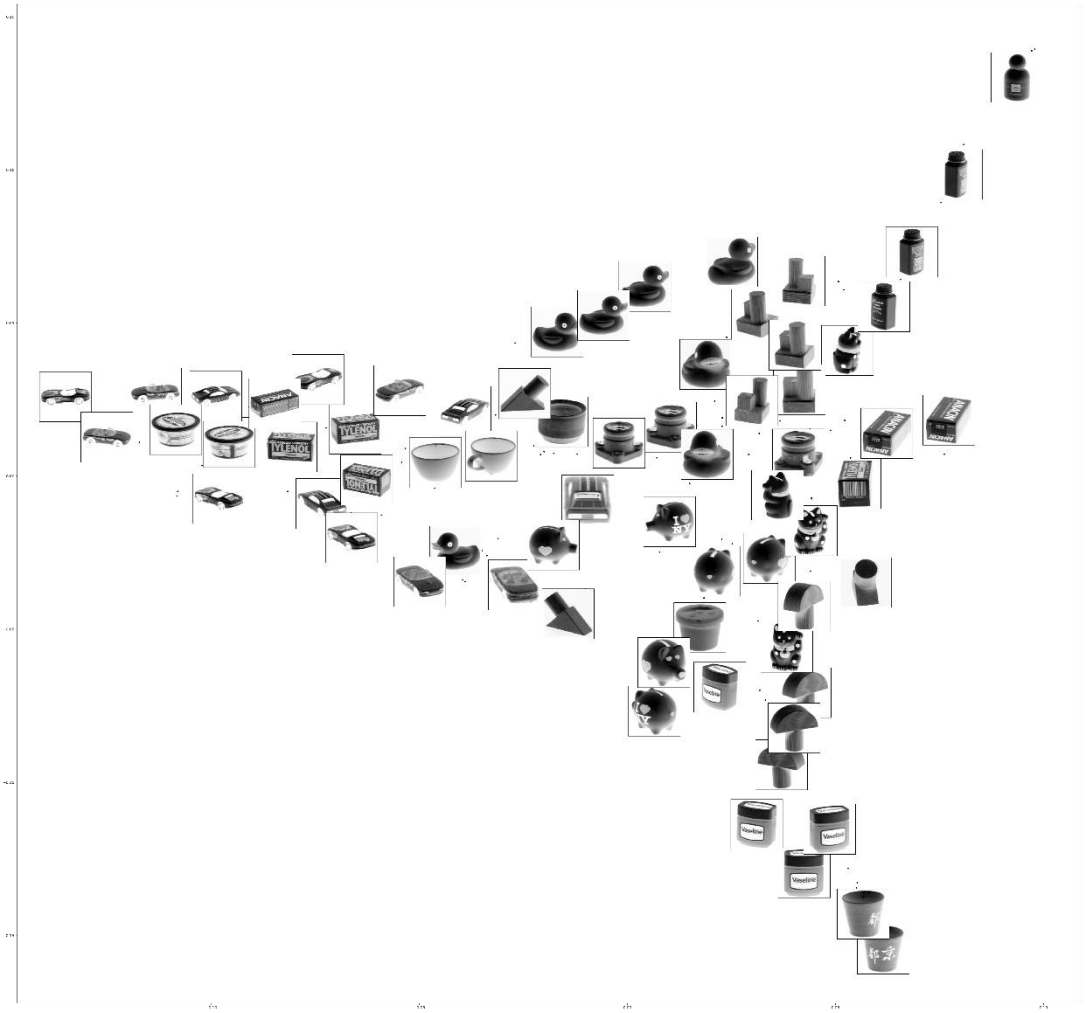


*Figure 4 : Le résultat initial du clustering avec LLE sur l'encodage des images*

Nous remarquons que LLE a pu détecter quelques clusters d'objets à partir de la représentation encodée des images. L'erreur global après une première itération est égale à : 1.61%

Après 7 itérations, l'algorithme a convergé avec une valeur de MSE égale à 0.74%. La figure suivante montre l'évolution de la fonction objective au cours des itérations :

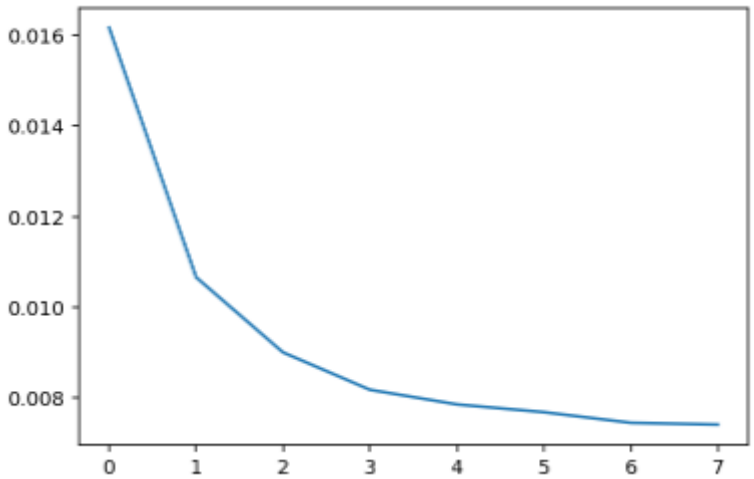



*Figure 5 : L'évolution de la fonction de perte globale au cours des itérations*

Le résultat de visualisation des images reconstruites est montré dans la figure suivante :

*Figure 6 : La qualité de visualisation des images reconstruites après la convergence de l'algorithme*

Nous remarquons que la qualité de visualisation a beaucoup amélioré après la convergence de l'algorithme. De même pour la qualité du clustering comme le montre la figure suivante.

*Figure 7 : Le résultat final du clustering avec LLE sur l'encodage des images*

## B. Les données MNIST

MNIST es une base de données de chiffres écrits à la main. Elle comprend un lot 60 000 images dédiées à l'apprentissage et 10 000 images pour le test set.

| Type | Nombre d'objets | Dimensions | Features |
|---|---|---|---|
| Gray-scale Images | - 60000 pour l'apprentissage<br>- 10000 pour le test | (28 x 28) | 784 |

Nous avons subdivisé les données d'apprentissages en échantillon de validation (40%) et échantillons d'apprentissage (60%).

| Apprentissage | Validation | Test |
|---|---|---|
| 36000 | 24000 | 10000 |

### 1. Le paramétrage de l'algorithme

L'auto-encodeur appliqué sur ces données est composé d'une seule couche comme le montre la figure suivante.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 784)               25872
=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
_________________________________________________________________
```

*Figure 8 : Architecture de l'auto-encodeur utilisé pour la compression des données MNIST*

Après plusieurs testes de l'algorithme LLE, nous avons choisi le nombre de voisins (k = 20).

### 2. Evaluation de l'algorithme

Une première application de l'auto-encodeur (epochs = 50 et batch = 256) sur les données de test a donnés une qualité de visualisation faible comme le montre la figure suivante.

*Figure 9 : Le résultat initial de visualisation des images reconstruites avec l'auto-encodeur*

Nous avons injecté la représentation encodée à l'algorithme LLE. La figure suivante illustre la distribution des images selon les deux premières composantes :

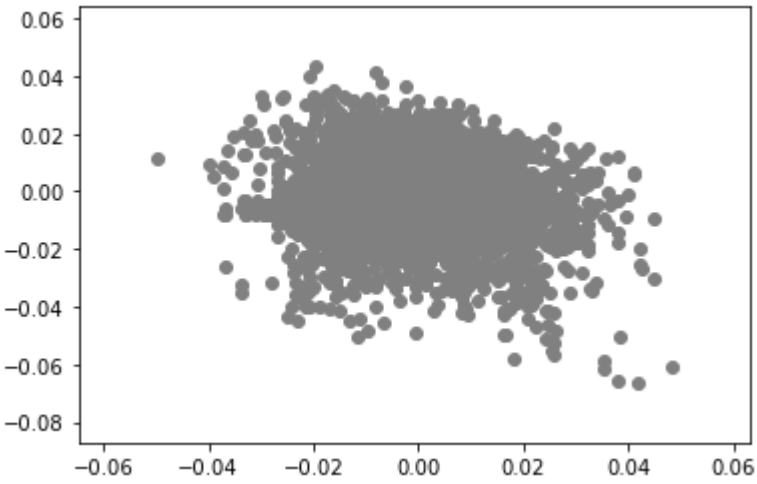



*Figure 10 : La distribution des images MNIST selon les deux premières composantes principales LLE*

La projection des images de test set sur les deux premières composantes donne le résultat présenté dans la figure suivante

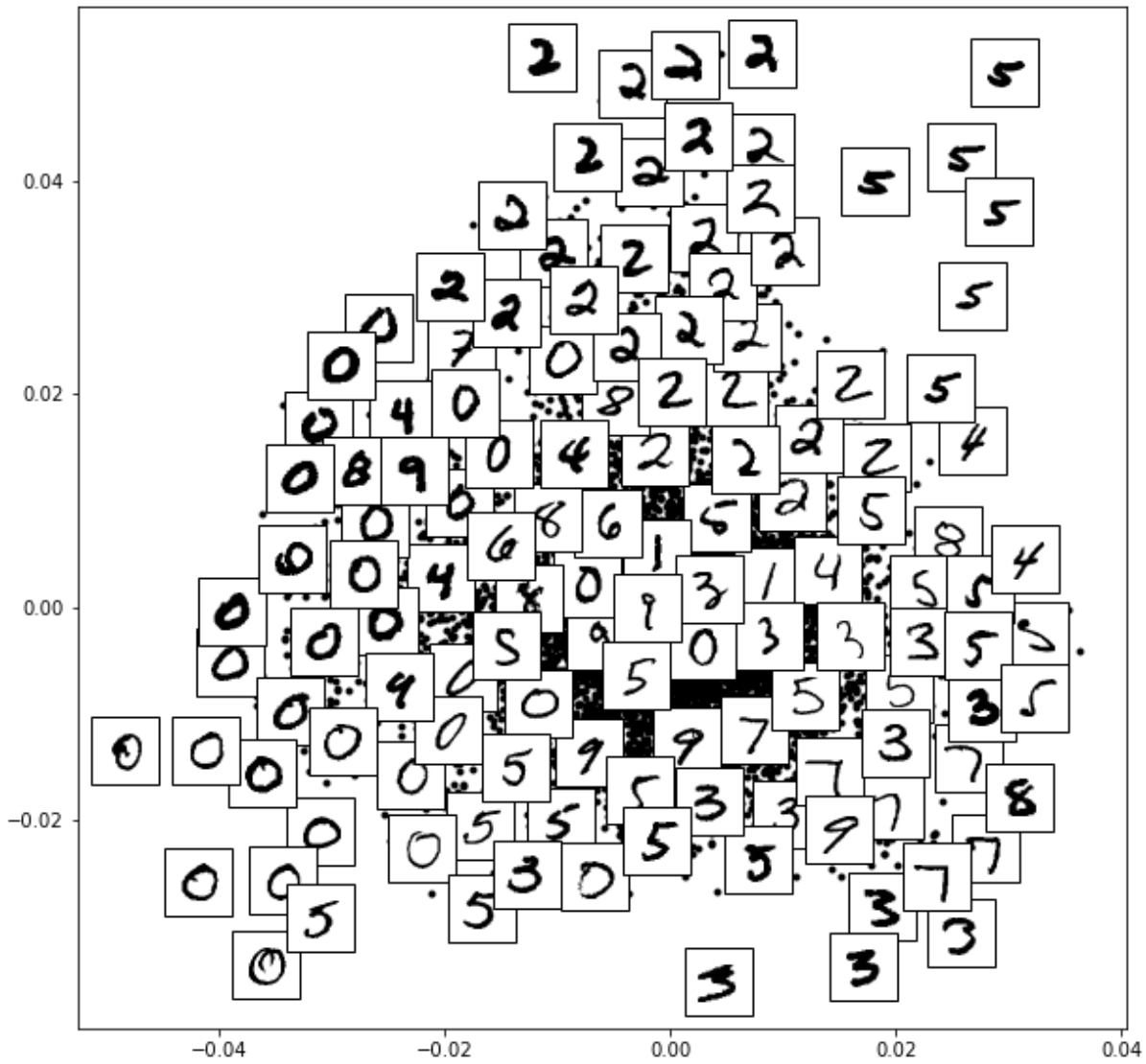



*Figure 11 : Le résultat initial du clustering avec LLE sur l'encodage des images*

Nous remarquons que LLE a pu la forme des chiffres à partir de la représentation encodée des images. L'erreur global après une première itération est égale à : 2.98%

Après 30 itérations, l'algorithme a convergé avec une valeur de MSE égale à 1.08%. La figure suivante montre l'évolution de la fonction objective au cours des itérations :

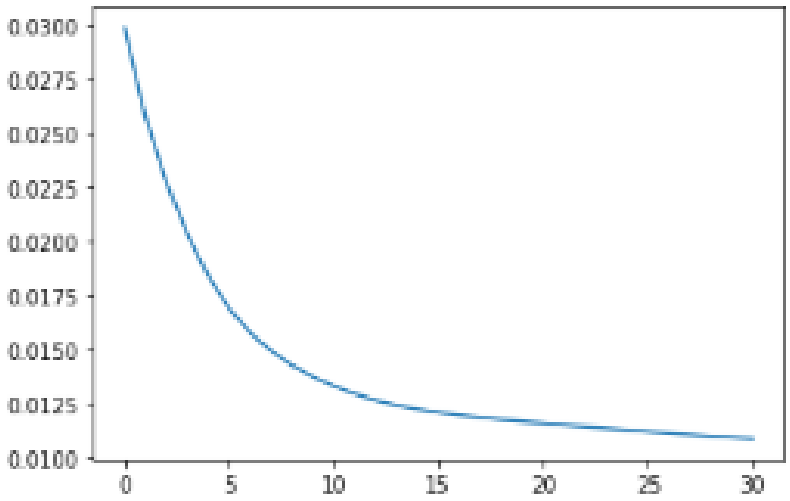



*Figure 12 : L'évolution de la fonction de perte globale au cours des itérations*

Le résultat de visualisation des images reconstruites est montré dans la figure suivante :

*Figure 13 : La qualité de visualisation des images reconstruites après la convergence de l'algorithme*

Nous remarquons que la qualité de visualisation a beaucoup amélioré après la convergence de l'algorithme. La qualité du clustering aussi. La figure suivante illustre le clustering des images après la convergence.

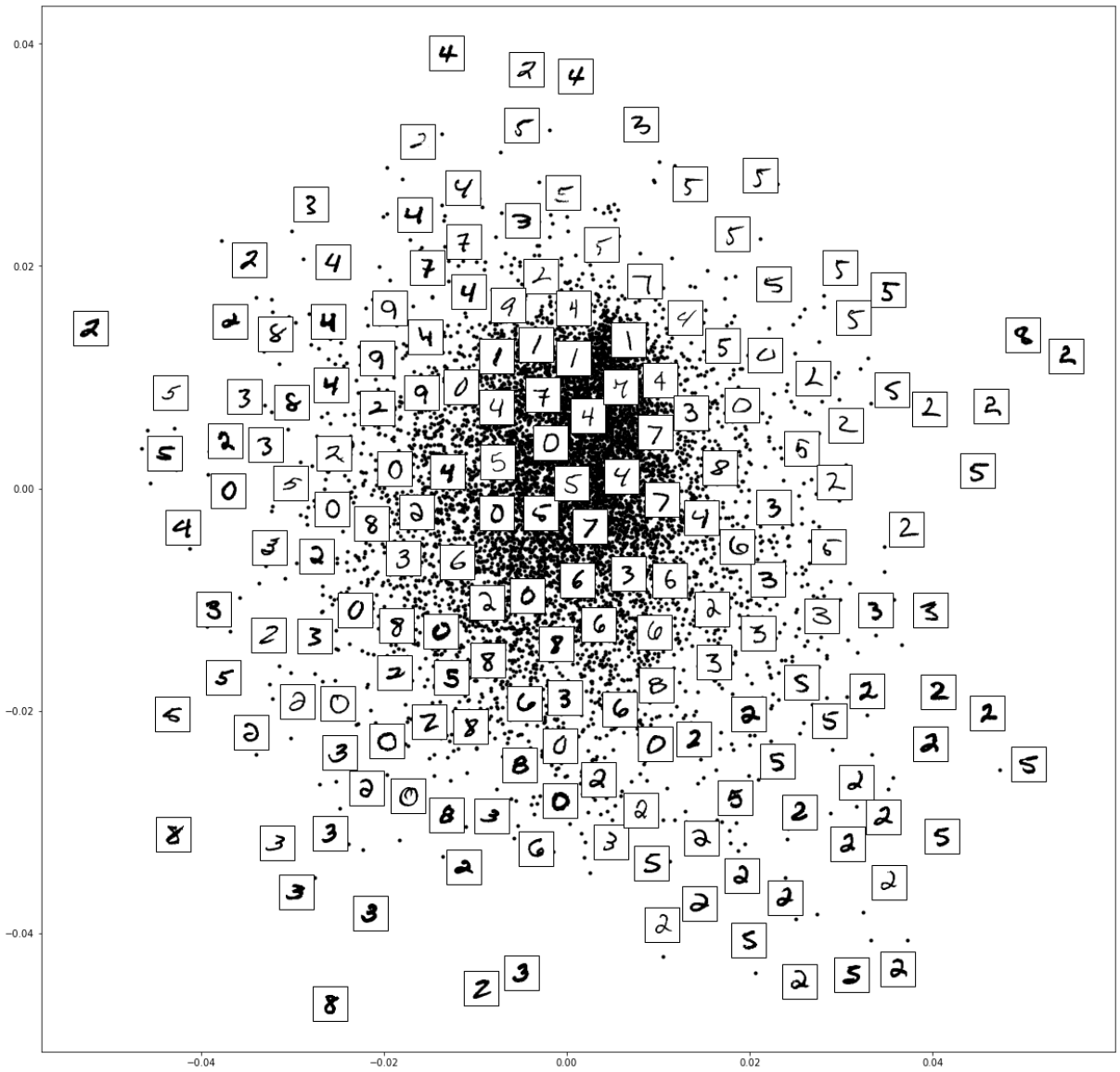



*Figure 14 : Le résultat final du clustering avec LLE sur l'encodage des images*

### C. Les données Coil100

Coil100 est une base de données de 7200 images en couleur de 100 objets (72 images par objet) Les objets ont été placés sur une plaque tournante. La table tournante a été tournée de 360 degrés pour varier la pose des objets par rapport à la caméra fixe. Les images des objets ont été prises à un intervalle de pose de 5 degrés.

| Type | Nombre d'objets | Dimensions | Features |
|---|---|---|---|
| En couleur | 7200 | (28 x 28) | 784 |

Nous avons subdivisé les données en échantillons d'apprentissages (85%) et échantillons de test (15%).

| Apprentissage | Test |
|---|---|
| 6120 | 1080 |

### 1. Le paramétrage de l'algorithme

L'auto-encodeur appliqué sur ces données est composé d'une seule couche comme le montre la figure suivante.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 128, 128, 3)       0
_________________________________________________________________
sequential_3 (Sequential)    (None, 1024)              50332672
_________________________________________________________________
sequential_4 (Sequential)    (None, 128, 128, 3)       50380800
=================================================================
Total params: 100,713,472
Trainable params: 100,713,472
Non-trainable params: 0
_________________________________________________________________
None
```

*Figure 15 : Architecture de l'auto-encodeur utilisé pour la compression des données Coil100*

Après plusieurs testes de l'algorithme LLE, nous avons choisi le nombre de voisins (k = 30).

### 2. Evaluation de l'algorithme

Une première application de l'auto-encodeur (epochs = 20) sur les données de test a donnés une qualité de visualisation très faible comme le montre la figure suivante.

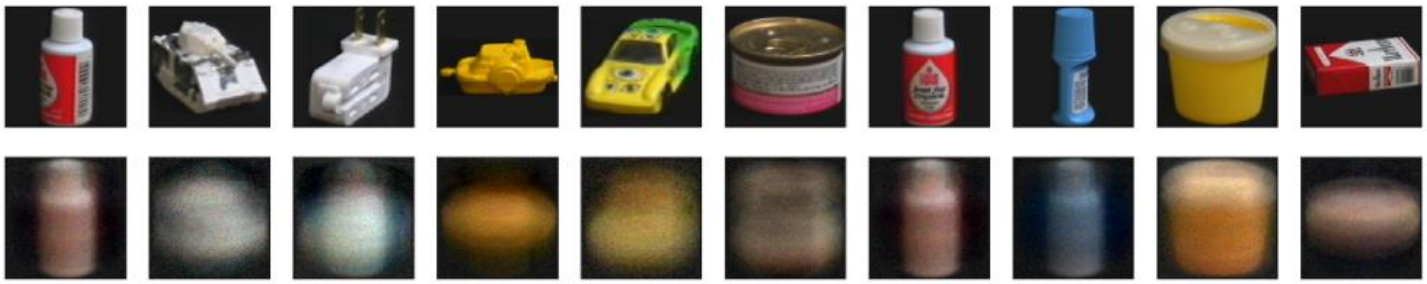


*Figure 16 : Le résultat initial de visualisation des images reconstruites avec l'auto-encodeur*

Nous avons injecté la représentation encodée à l'algorithme LLE. La figure suivante illustre la distribution des images selon les deux premières composantes :

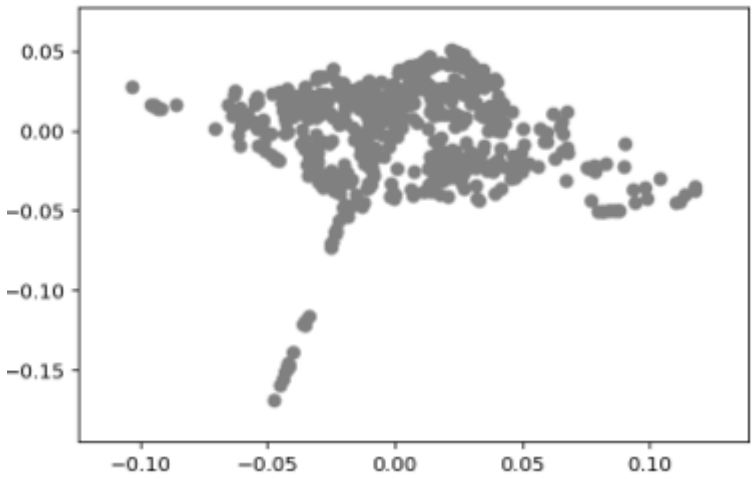



*Figure 17 : La distribution des images Coil100 selon les deux premières composantes principales*

La projection des images de test set sur les deux premières composantes donne le résultat présenté dans la figure suivante.

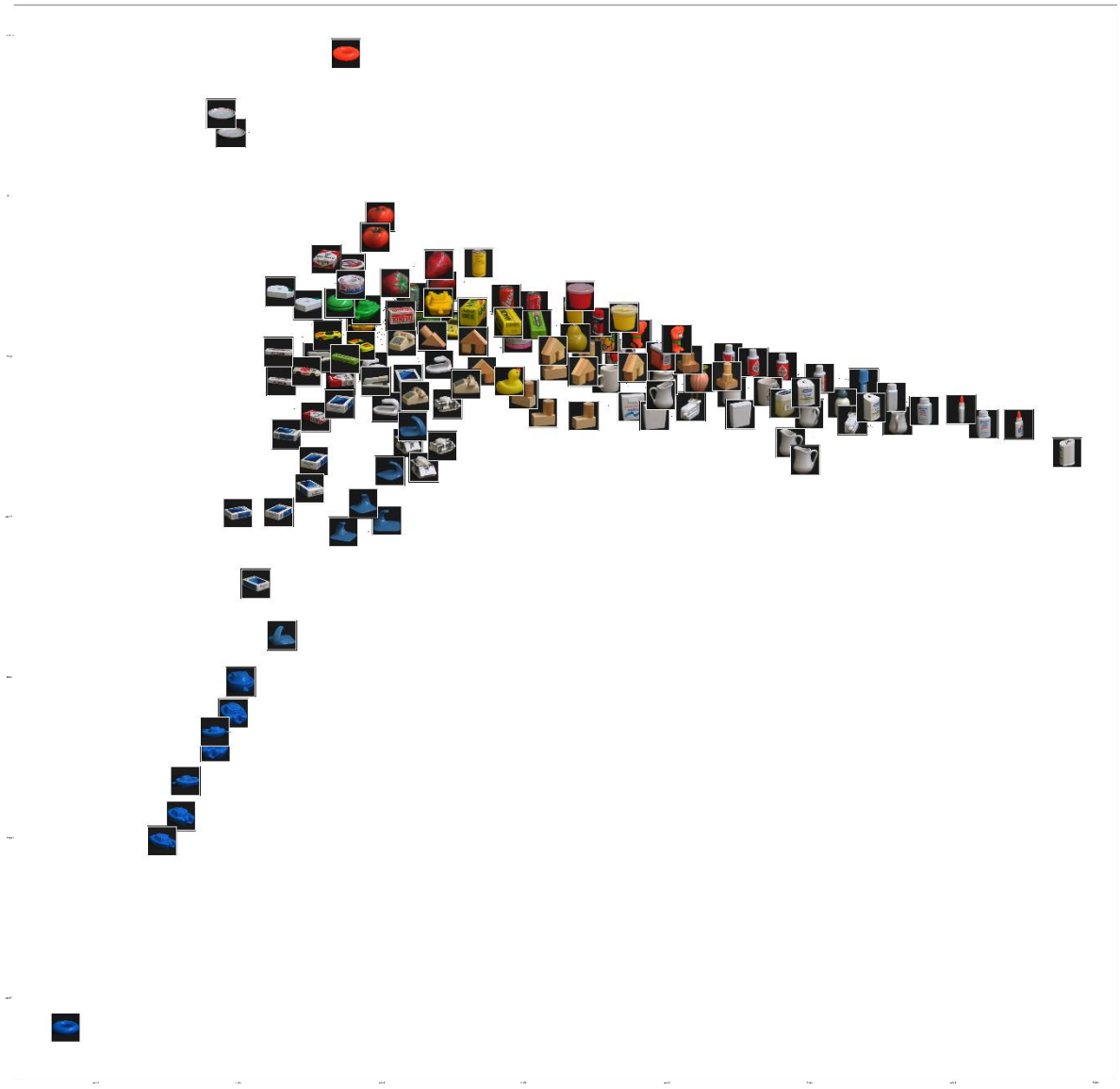


*Figure 18 : Le résultat initial du clustering avec LLE sur l'encodage des images*

Nous remarquons que LLE a pu détecter les clusters des objets à partir de la représentation encodée des images. L'erreur global après une première itération est égale à : 1.78%

Après 5 itérations, l'algorithme a convergé avec une valeur de MSE égale à 0.19%. La figure suivante montre l'évolution de la fonction objective au cours des itérations :

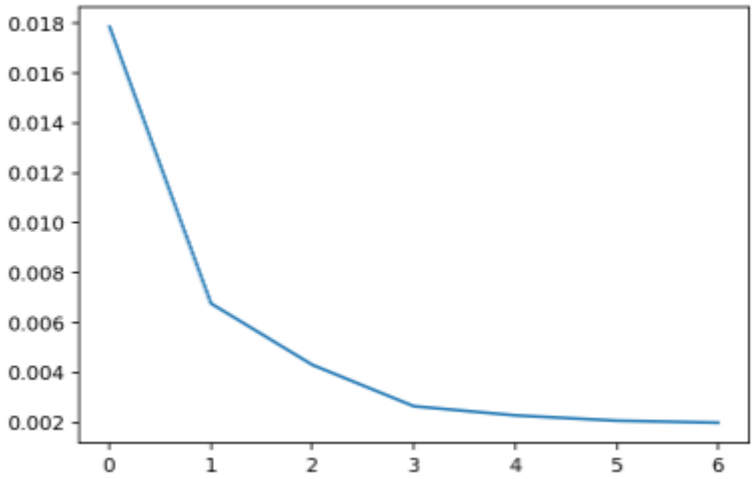



*Figure 19 : L'évolution de la fonction de perte globale au cours des itérations*

Le résultat de visualisation des images reconstruites est montré dans la figure suivante :

*Figure 20 : La qualité de visualisation des images reconstruites après la convergence de l'algorithme*

Nous remarquons que la qualité de visualisation a beaucoup amélioré après la convergence de l'algorithme. De même pour le clustering comme le montre la figure suivante.

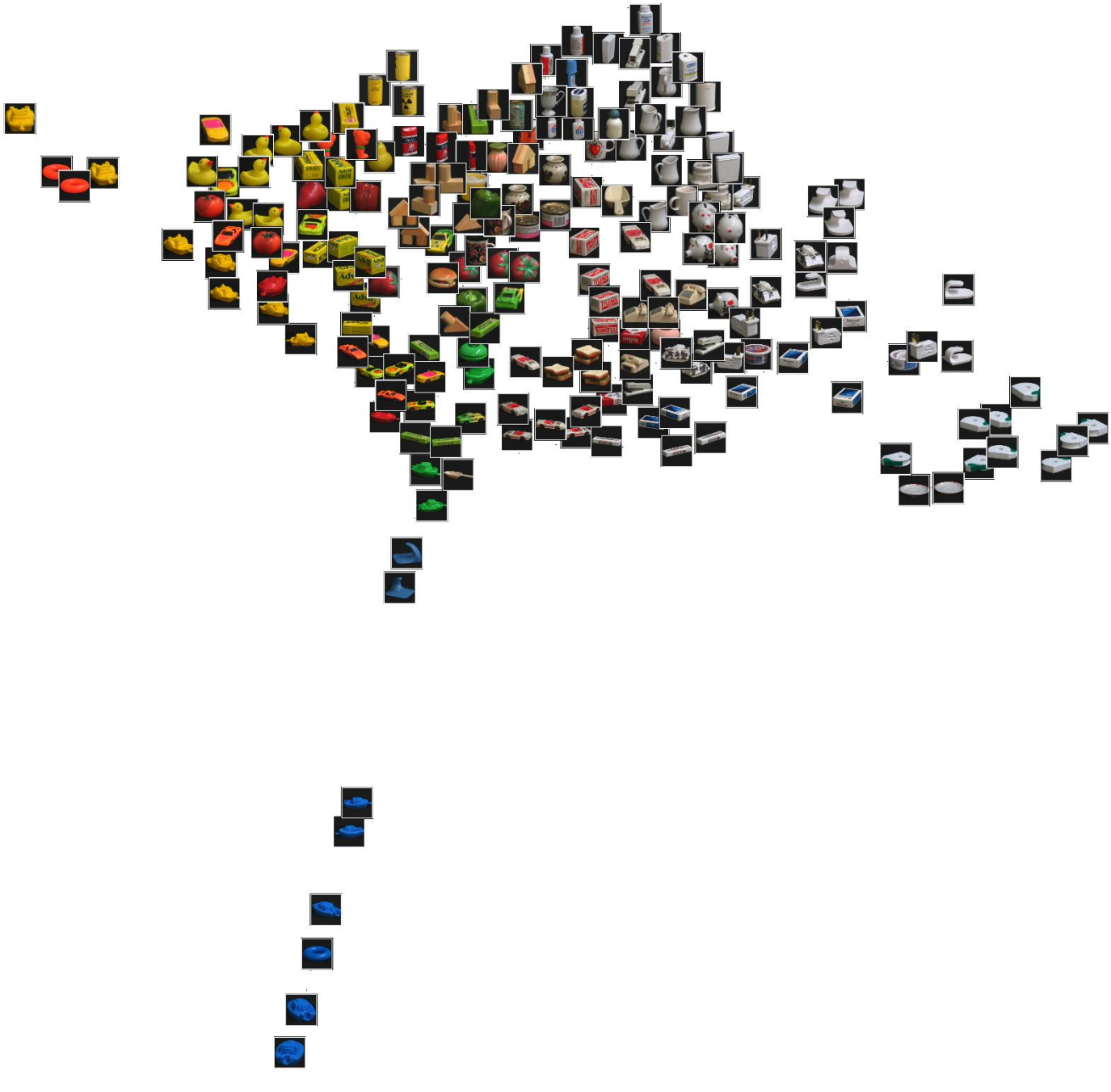


*Figure 21 : Le résultat du clustering avec LLE sur l'encodage des images*

# IV. Conclusion

Le deep autoencodeur est un outil puissant pour la compression des images. Sa combinaison avec LLE a permis d'améliorer la qualité de représentation encodée ainsi que le clustering. L'inconvénient majeur à noter est le temp nécessaire pour la convergence de l'algorithme qui dépend fortement des ressources utilisés ainsi que la taille des objets.