



## PROJET MACHINE LEARNING

MOUBARAK Carine

HUANG Tian

24 mai 2024

# Table des matières

<b>I</b>	<b>Introduction</b>	<b>3</b>
<b>II</b>	<b>Mon premier est . . . linéaire !</b>	<b>3</b>
II.1	Expérimentations : Régression . . . . .	3
II.2	Expérimentations : Classification . . . . .	4
<b>III</b>	<b>Mon second est . . . non-linéaire !</b>	<b>5</b>
III.1	Classification Sans Bruit . . . . .	6
III.2	Classification Avec Bruit . . . . .	7
<b>IV</b>	<b>Mon troisième est un encapsulage</b>	<b>9</b>
IV.1	Comparaison des tailles de batch . . . . .	9
IV.2	Comparaison des taux d'apprentissage . . . . .	11
<b>V</b>	<b>Mon quatrième est multi-classe</b>	<b>12</b>
V.1	Classification multi-classe avec le jeu de données USPS . . . . .	12
V.2	Exploration supplémentaire avec le jeu de données MNIST . . . . .	15
<b>VI</b>	<b>Mon cinquième se compresse</b>	<b>15</b>
VI.1	Entraînement de l'autoencodeur avec différents hyperparamètres . . . . .	16
VI.2	Utilisation des données encodées pour la classification . . . . .	18
VI.3	Utilisation de K-Means pour la classification . . . . .	20
VI.4	Utilisation de l'autoencodeur pour supprimer le bruit . . . . .	21
<b>VII</b>	<b>Mon sixième se convole et mon tout s'améliore</b>	<b>23</b>
VII.1	Convolution 1D . . . . .	23
VII.2	Convolution 2D . . . . .	25
<b>VIII</b>	<b>Conclusion</b>	<b>26</b>

# I Introduction

Ce projet vise à créer un réseau de neurones en s'inspirant des anciennes versions de PyTorch, avant l'introduction d'Autograd. L'approche adoptée est modulaire et générique, permettant la construction flexible de réseaux complexes.

Les différentes étapes comprennent la mise en place d'une régression linéaire, considérée comme une base fondamentale des réseaux de neurones. Ensuite, une expérimentation est prévue avec des architectures non linéaires, telles qu'un réseau à deux couches linéaires avec des activations tangentielles et sigmoïdes, adapté à des problèmes de classification binaire.

Dans le cadre de ce projet, une classe Séquentielle sera implémentée pour encapsuler les différentes couches de manière séquentielle, simplifiant ainsi les procédures de propagation avant (forward) et de rétro-propagation (backward). L'extension vers la classification multi-classe sera également explorée, où chaque classe est représentée par une dimension de sortie et le vecteur de supervision est encodé en one-hot.

De plus, ce projet inclura une exploration des auto-encodeurs, un type d'apprentissage non supervisé visant à apprendre une représentation compressée des données. Enfin, les couches convolutionnelles seront mises en œuvre, particulièrement adaptées à la classification d'images, permettant de reconnaître des motifs horizontaux et verticaux.

L'objectif principal est d'offrir une exploration complète des concepts clés des réseaux de neurones, depuis la régression linéaire jusqu'aux architectures convolutionnelles, tout en encourageant la modularité et la flexibilité dans l'implémentation.

# II Mon premier est . . . linéaire !

Dans cette partie, nous avons implémenté deux classes essentielles pour effectuer une régression linéaire. Tout d'abord, nous avons créé une classe `MSELoss` destinée à calculer la fonction de coût Mean Squared Error (MSE), qui est essentielle pour mesurer l'efficacité du modèle. Ensuite, une classe `Linear` a été mise en place pour représenter un réseau neuronal à une couche linéaire. Après avoir établi ces bases, nous avons mené des expériences non seulement en régression linéaire mais aussi en classification binaire pour tester la polyvalence du modèle. Ces tests ont démontré que le modèle conserve une haute précision même avec l'augmentation du niveau de bruit, révélant ainsi sa robustesse et sa capacité à généraliser efficacement dans diverses conditions de bruit.

## II.1 Expérimentations : Régression

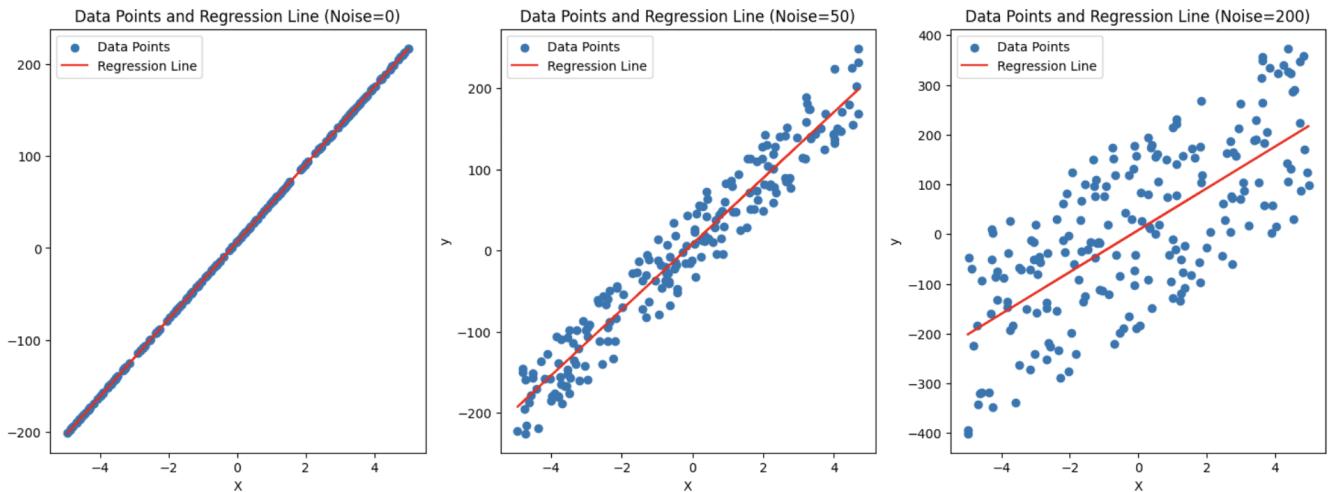


FIGURE 1 – Comparaison des lignes de régression avec différents niveaux de bruit

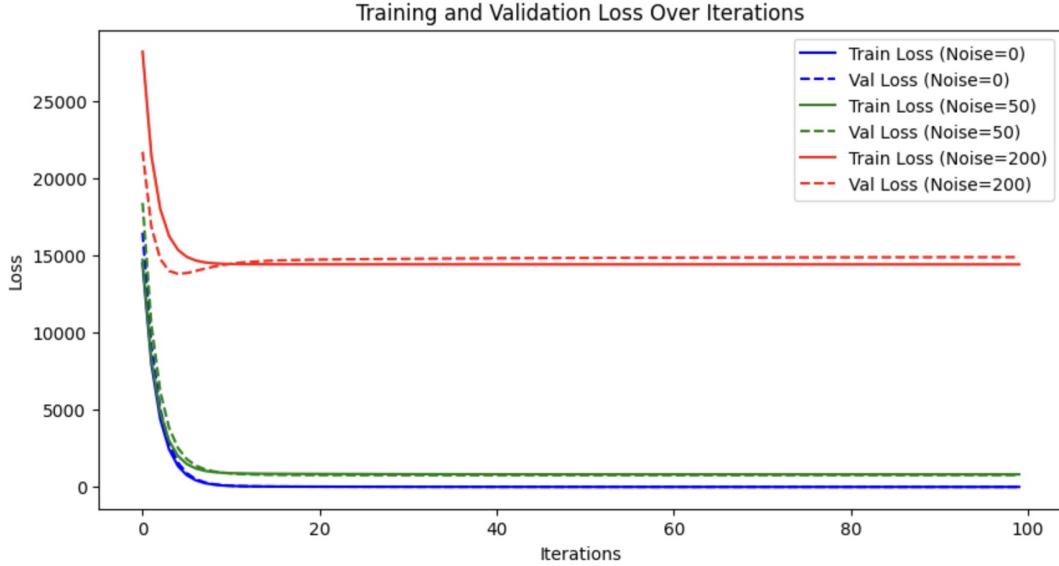


FIGURE 2 – Évolution des pertes d’entraînement et de validation à travers les itérations

Le dataset utilisé dans ces expériences a été généré en appliquant un modèle linéaire simple avec des coefficients fixes ( $a = 42$ ,  $b = 7$ ), et en introduisant des niveaux de bruit variés (0, 50, 200) pour observer l’impact du bruit sur la performance du modèle. Les résultats obtenus montrent que, même si le niveau de bruit augmente, entraînant une hausse de la perte, les lignes de régression restent de haute qualité et bien ajustées aux points de données. Les graphiques illustrent clairement que les lignes de régression tracées sont en adéquation avec les tendances des données, malgré la présence de bruit. En outre, il n’y a pas de différence significative entre les pertes d’entraînement et de validation, ce qui indique une bonne généralisation du modèle sur le jeu de données de validation, même dans des conditions de forte perturbation. Ces observations sont cruciales pour démontrer la robustesse du modèle face à des variations significatives de l’environnement d’entrée.

## II.2 Expérimentations : Classification

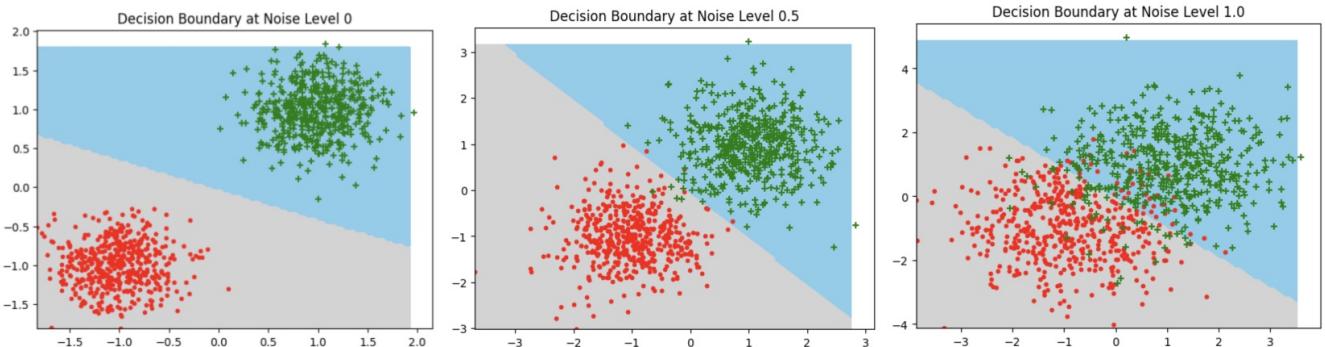


FIGURE 3 – Comparaison des frontières de décision pour différents niveaux de bruit

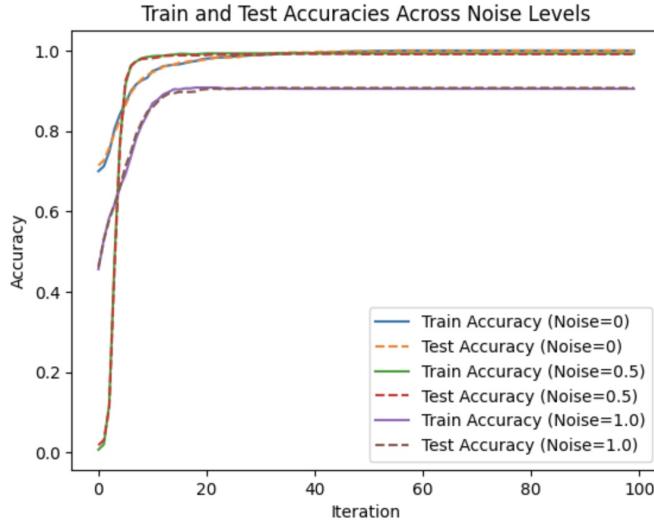


FIGURE 4 – Précision d’entraînement et de test à travers les itérations pour différents niveaux de bruit

Les expériences démontrent que, malgré l’augmentation du bruit, les frontières de décision du modèle restent robustes, illustrant sa capacité à distinguer les catégories sous différentes conditions. Pour des niveaux de bruit de 0 et 0,5, le modèle atteint rapidement une précision proche de 100 %. Toutefois, avec un niveau de bruit de 1,0, bien que la précision chute à environ 90 %, elle se stabilise promptement, reflétant la résilience du modèle face aux variations environnementales. L’absence de différence significative entre les précisions d’entraînement et de test confirme une excellente généralisation et régularisation, indiquant que le modèle est efficace et bien adapté, même dans des conditions bruyantes.

### III Mon second est . . . non-linéaire !

L’objectif de cette partie est d’implémenter des modules de fonctions d’activation non-linéaires, notamment la fonction Tangente Hyperbolique (Tanh) et la fonction Sigmoïde, afin d’améliorer la capacité du réseau de neurones à modéliser des relations complexes dans les données. La classe TanH implémente la fonction d’activation Tangente Hyperbolique, qui transforme les entrées en valeurs comprises entre -1 et 1. La classe Sigmoïde implémente la fonction d’activation Sigmoïde, qui transforme les entrées en valeurs comprises entre 0 et 1.

Nous avons varié le nombre de neurones dans notre réseau de neurones linéaires à deux couches pour expérimenter son impact sur les performances de notre modèle. En ajustant les dimensions des couches cachées, nous avons exploré comment la complexité du modèle influence sa capacité à apprendre et à généraliser à partir des données. Nous avons également effectué des tests avec et sans bruit pour observer l’effet des perturbations sur les performances du modèle.

### III.1 Classification Sans Bruit

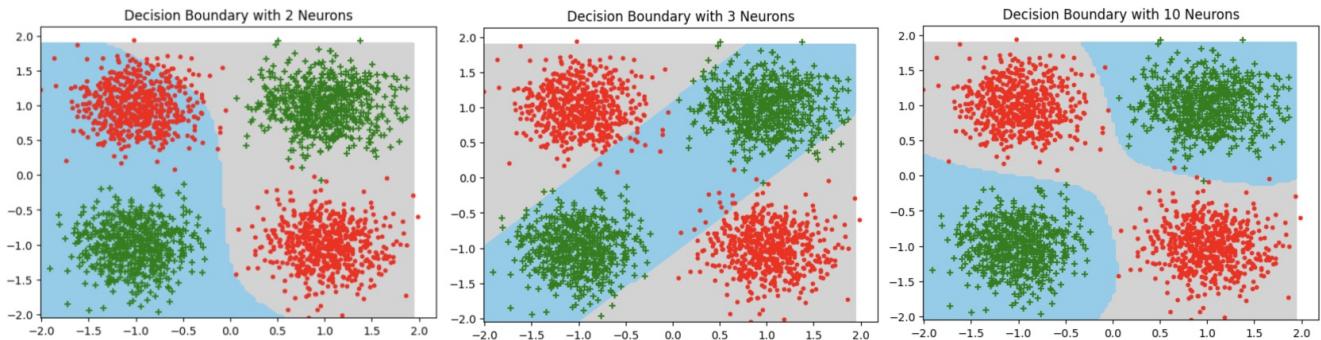


FIGURE 5 – Décision boundary pour les configurations de 2, 3 et 10 neurones (sans bruit)

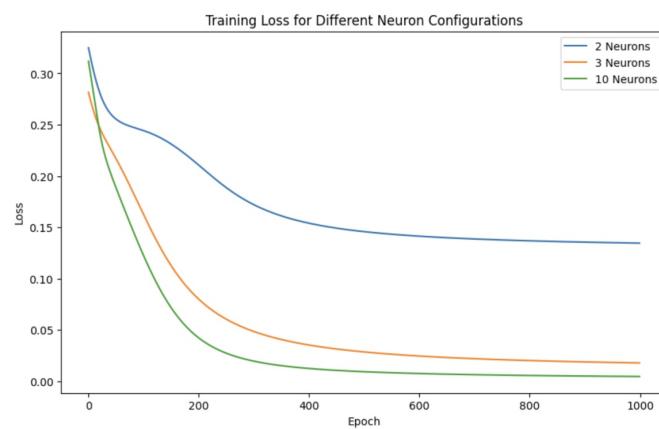


FIGURE 6 – Perte d'entraînement pour les configurations de 2, 3 et 10 neurones (sans bruit)

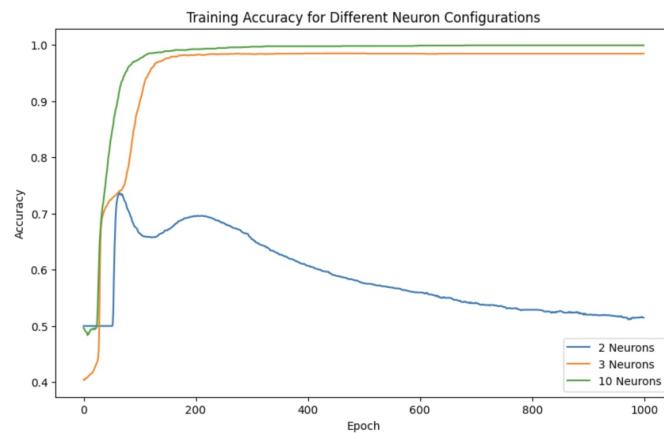


FIGURE 7 – Précision d'entraînement pour les configurations de 2, 3 et 10 neurones (sans bruit)

Les résultats des expérimentations sans bruit sont résumés comme suit :

Configuration	Meilleure précision (Entraînement)	Précision (Test)
2 neurones	0.7368	0.5136
3 neurones	0.9852	0.9792
10 neurones	0.9992	0.9984

TABLE 1 – Précision d’entraînement et de test pour les configurations de 2, 3 et 10 neurones (sans bruit)

Les résultats des expérimentations sans bruit montrent des tendances intéressantes. Pour la configuration avec 2 neurones, la précision augmente initialement mais commence à diminuer autour de l’epoch 200, tombant de 0.7 à 0.5 à la fin des 1000 epochs. Cela suggère que le modèle avec 2 neurones souffre d’un manque de capacité pour modéliser les relations complexes dans les données, conduisant à un sous-ajustement.

En revanche, les configurations avec 3 et 10 neurones convergent rapidement à une précision supérieure à 95% en moins de 200 epochs, montrant une capacité beaucoup plus élevée à apprendre des données. Cependant, le modèle avec 10 neurones montre une meilleure performance globale, atteignant presque 100% de précision, ce qui indique une capacité supérieure à capturer les nuances des données.

La différence entre les résultats de 3 et 10 neurones est notable. Bien que les deux modèles performent bien, le modèle avec 10 neurones présente une légère supériorité en termes de stabilité et de performance finale.

En conclusion, bien que les modèles avec 3 et 10 neurones soient performants, celui avec 10 neurones offre une précision et une stabilité légèrement supérieures.

### III.2 Classification Avec Bruit

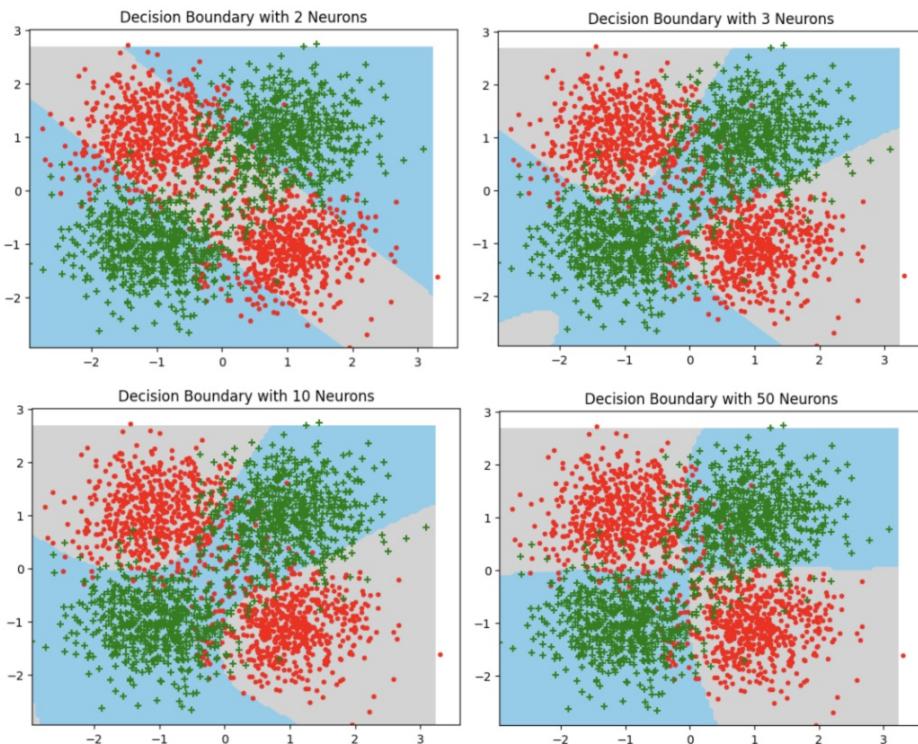


FIGURE 8 – Décision boundary pour les configurations de 2, 3, 10 et 20 neurones (avec bruit)

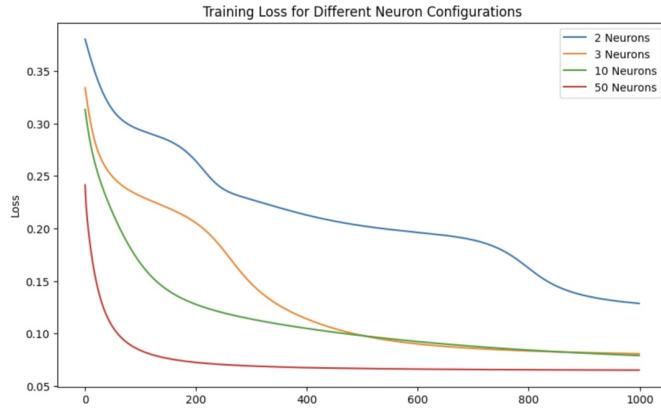


FIGURE 9 – Perte d’entraînement pour les configurations de 2, 3, 10 et 20 neurones (avec bruit)

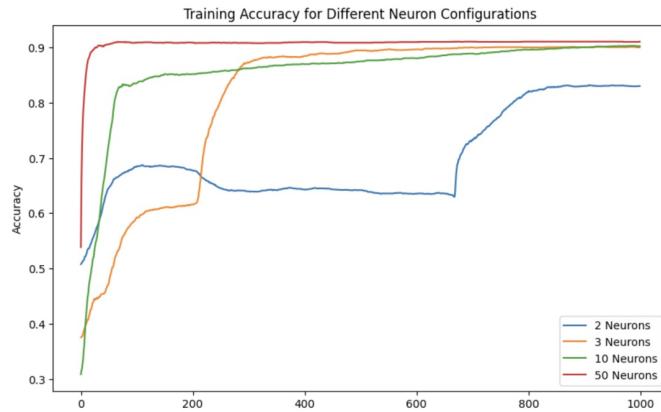


FIGURE 10 – Précision d’entraînement pour les configurations de 2, 3, 10 et 20 neurones (avec bruit)

Les résultats des expérimentations avec bruit ( $\epsilon=0.5$ ) sont résumés dans le tableau ci-dessous :

Configuration	Meilleure précision (Entraînement)	Précision (Test)
2 neurones	0.8316	0.8424
3 neurones	0.9008	0.8928
10 neurones	0.9028	0.8968
50 neurones	0.9104	0.9164

TABLE 2 – Précision d’entraînement et de test pour les configurations de 2, 3, 10 et 50 neurones (avec bruit)

En analysant les résultats, il est évident que, comme dans le cas sans bruit, la configuration avec 2 neurones n'est pas optimale. La précision est significativement inférieure comparée aux configurations avec plus de neurones. Dès que nous augmentons à 3 neurones, les performances s'améliorent considérablement.

Les configurations avec 3 et 10 neurones montrent des performances similaires, bien que le modèle avec 10 neurones offre une légère amélioration. En revanche, l'ajout de plus de neurones, comme dans la configuration avec 50 neurones, apporte une amélioration visible mais moindre comparée au saut de 2 à 3 neurones. Cette tendance indique que l'augmentation du nombre de neurones au-delà de 10 n'apporte que des gains marginaux en précision.

En termes de convergence, le modèle avec 50 neurones converge le plus rapidement, atteignant une précision stable avant 25 epochs. Le modèle avec 10 neurones converge en moins de 100 epochs, tandis que celui avec 3 neurones prend environ 300 epochs pour se stabiliser. Le modèle avec 2 neurones, quant à lui, met près de 800 epochs pour converger, avec une précision finale autour de 80%. De plus, les configurations avec 3 et 10 neurones ont des courbes de perte très similaires, tandis que le modèle avec 50 neurones présente une légère amélioration.

Des tests supplémentaires avec des configurations de 100 et 500 neurones montrent des résultats similaires à ceux avec 50 neurones, suggérant que l'ajout de neurones au-delà de ce point n'apporte que des améliorations marginales.

En conclusion, bien que l'augmentation du nombre de neurones améliore les performances du modèle, les gains diminuent après un certain point. Le modèle avec 10 neurones semble offrir un bon compromis entre complexité et performance, tandis que des configurations plus grandes comme celle avec 50 neurones montrent des améliorations mais à un coût de complexité accru.

## IV Mon troisième est un encapsulage

Dans cette partie, nous avons simplifié et automatisé les opérations de chaînage entre modules lors de la descente de gradient en implémentant une classe `Sequentiel`. Cette classe permet d'ajouter des modules en série et d'automatiser les procédures de *forward* et *backward*, quel que soit le nombre de modules.

Nous avons également développé une classe `Optim` pour encapsuler une itération de gradient. En prenant en entrée un réseau, une fonction de coût et un pas de gradient, elle gère automatiquement les mises à jour des paramètres du modèle via la méthode `step`.

Enfin, nous avons implémenté une fonction `SGD` (Stochastic Gradient Descent) pour diviser les données en batches et entraîner le réseau sur un nombre spécifié d'itérations, en utilisant la classe `Optim`.

Ces outils améliorent l'efficacité et la modularité de notre implémentation de réseaux de neurones, facilitant ainsi l'exploration de différentes architectures et hyperparamètres.

### IV.1 Comparaison des tailles de batch

Nous avons fixé le taux d'apprentissage à 0.001 et comparé les performances pour trois tailles de batch différentes : 20, 100 et 500. Les résultats de ces expériences sont présentés ci-dessous.

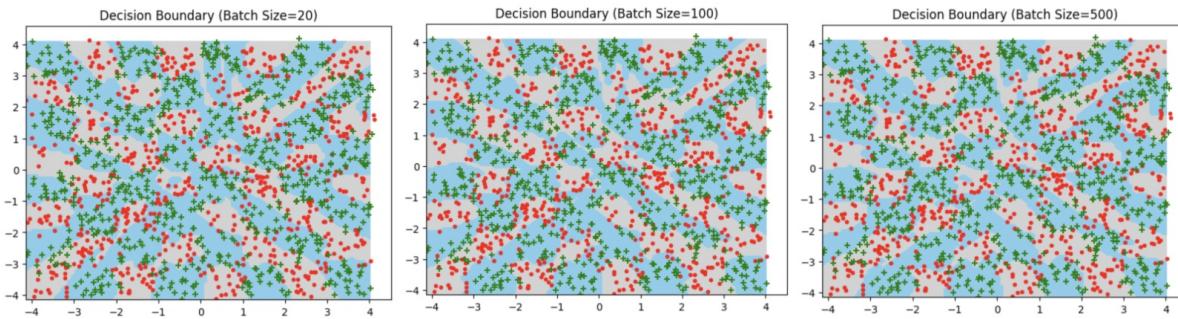


FIGURE 11 – Frontières de décision pour différentes tailles de batch (20, 100, 500) avec un taux d'apprentissage de 0.001

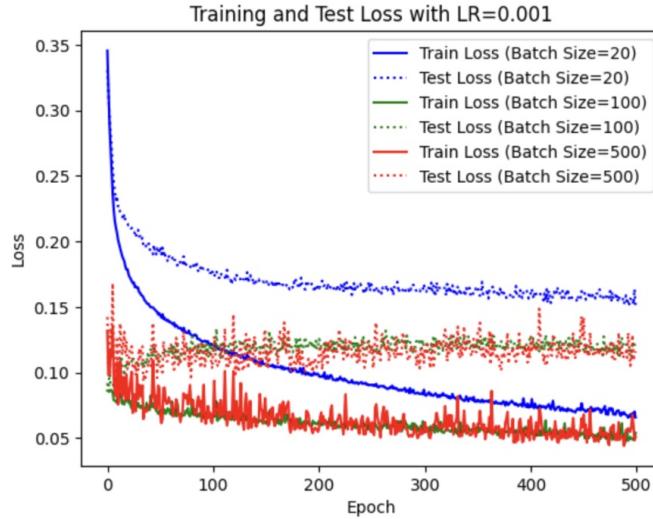


FIGURE 12 – Courbes de perte d’entraînement et de test pour différentes tailles de batch avec un taux d’apprentissage de 0.001

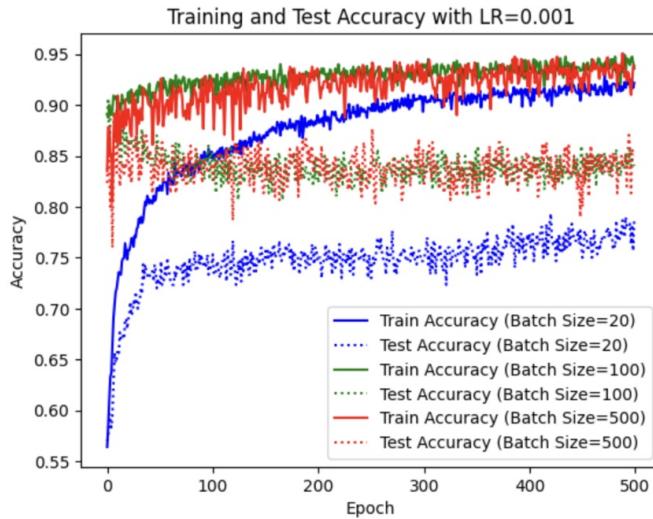


FIGURE 13 – Courbes de précision d’entraînement et de test pour différentes tailles de batch avec un taux d’apprentissage de 0.001

Batch size	Meilleure précision (Entraînement)	Précision (Test)
20	0.9317	0.7933
100	0.9492	0.9067
500	0.9508	0.8767

TABLE 3 – Résultats de précision pour différentes tailles de batch avec un taux d’apprentissage de 0.001

En analysant les résultats pour différentes tailles de batch avec un taux d’apprentissage fixe de 0.001, nous observons que les tailles de batch intermédiaire (100) et grande (500) montrent des tendances similaires en termes de précision et de perte. Les courbes de perte et de précision convergent de manière plus rapide et stable comparées à celles avec une petite taille de batch (20), qui convergent plus lentement et présentent une performance inférieure. Malgré cela, les frontières de décision pour les trois tailles de batch ne montrent pas de différences significatives à l’œil nu. Il est également important de noter qu’au-delà d’une certaine taille de batch, l’augmentation de celle-ci n’améliore plus significativement la qualité du modèle.

## IV.2 Comparaison des taux d'apprentissage

Nous avons fixé la taille du batch à 100 et comparé les performances pour trois taux d'apprentissage différents : 0.0001, 0.001 et 0.1. Les résultats de ces expériences sont présentés ci-dessous.

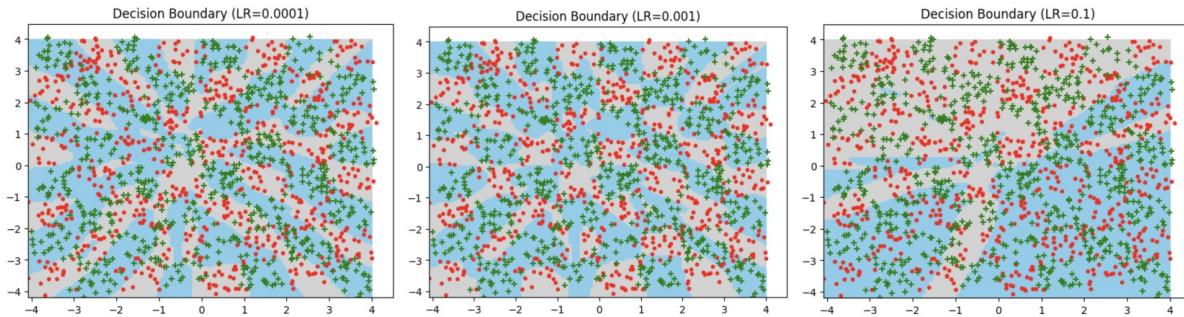


FIGURE 14 – Frontières de décision pour différents taux d'apprentissage (0.0001, 0.001, 0.1) avec une taille de batch de 100

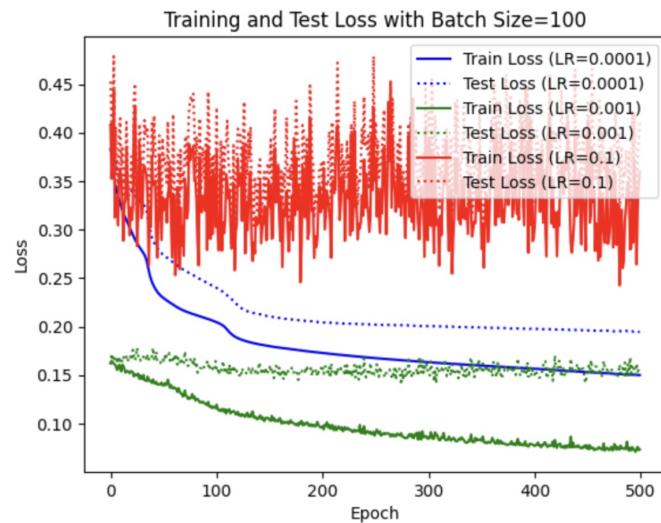


FIGURE 15 – Courbes de perte d'entraînement et de test pour différents taux d'apprentissage avec une taille de batch de 100

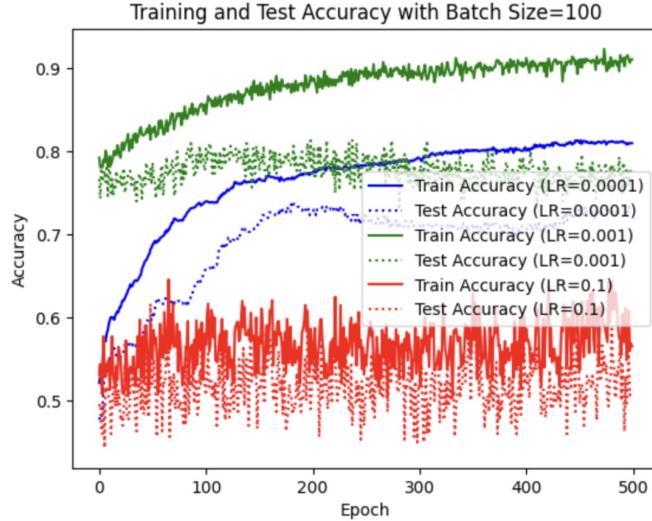


FIGURE 16 – Courbes de précision d’entraînement et de test pour différents taux d’apprentissage avec une taille de batch de 100

Learning rate	Meilleure précision (Entraînement)	Précision (Test)
0.0001	0.8133	0.7367
0.001	0.9233	0.8133
0.1	0.6458	0.6167

TABLE 4 – Résultats de précision pour différents taux d’apprentissage avec une taille de batch de 100

En analysant les résultats pour différents taux d’apprentissage avec une taille de batch fixe de 100, nous observons que le choix du taux d’apprentissage a un impact significatif sur les performances du modèle. Un taux d’apprentissage élevé (0.1) conduit à une frontière de décision visiblement moins performante, avec des courbes de perte et de précision très instables, indiquant une difficulté à converger. Un taux d’apprentissage trop faible (0.0001) entraîne une convergence lente et une performance initiale inférieure, bien que les courbes de perte et de précision montrent une tendance similaire entre l’entraînement et le test. Cela est également reflété dans le tableau 4, où nous voyons que le taux d’apprentissage de 0.1 donne les pires résultats, tandis que 0.0001 converge lentement.

Un taux d’apprentissage modéré (0.001) permet généralement d’atteindre un bon compromis entre vitesse de convergence et stabilité du modèle. Les courbes de perte et de précision pour ce taux montrent une convergence plus rapide et stable, avec une meilleure précision tant sur les données d’entraînement que de test, comme le montre également le tableau 4.

En conclusion, choisir un taux d’apprentissage adéquat est crucial pour obtenir de bonnes performances. Un taux trop élevé peut causer des oscillations et une divergence, tandis qu’un taux trop faible peut ralentir la convergence. Un taux modéré, comme 0.001, semble offrir le meilleur équilibre en termes de performance et de stabilité du modèle.

## V Mon quatrième est multi-classe

Dans cette section, nous abordons le problème de classification multi-classe en utilisant un réseau de neurones. La classification multi-classe nécessite une adaptation des sorties du réseau pour représenter les probabilités des différentes classes. Pour cela, nous utilisons la transformation Softmax à la dernière couche du réseau pour convertir les scores en une distribution de probabilités. De plus, nous utilisons une fonction de coût adaptée aux distributions de probabilités, comme la cross-entropie, pour entraîner le modèle.

### V.1 Classification multi-classe avec le jeu de données USPS

Nous avons utilisé le jeu de données USPS pour entraîner et évaluer notre modèle de classification multi-classe. Voici les principales étapes et résultats de notre expérimentation :

### Configuration du modèle :

- Nombre de couches cachées : 2
- Neurones dans la première couche cachée : 150, activation TanH
- Neurones dans la deuxième couche cachée : 60, activation TanH
- Fonction d'activation de la couche de sortie : Softmax
- Fonction de coût : Cross-Entropy (CELogSoftMax)
- Optimiseur : Stochastic Gradient Descent (SGD)
- Taille de batch : 100
- Nombre d'epochs : 100

### Résultats :

- Meilleure précision sur les données d'entraînement : 0.9233
- Meilleure précision sur les données de test : 0.8133

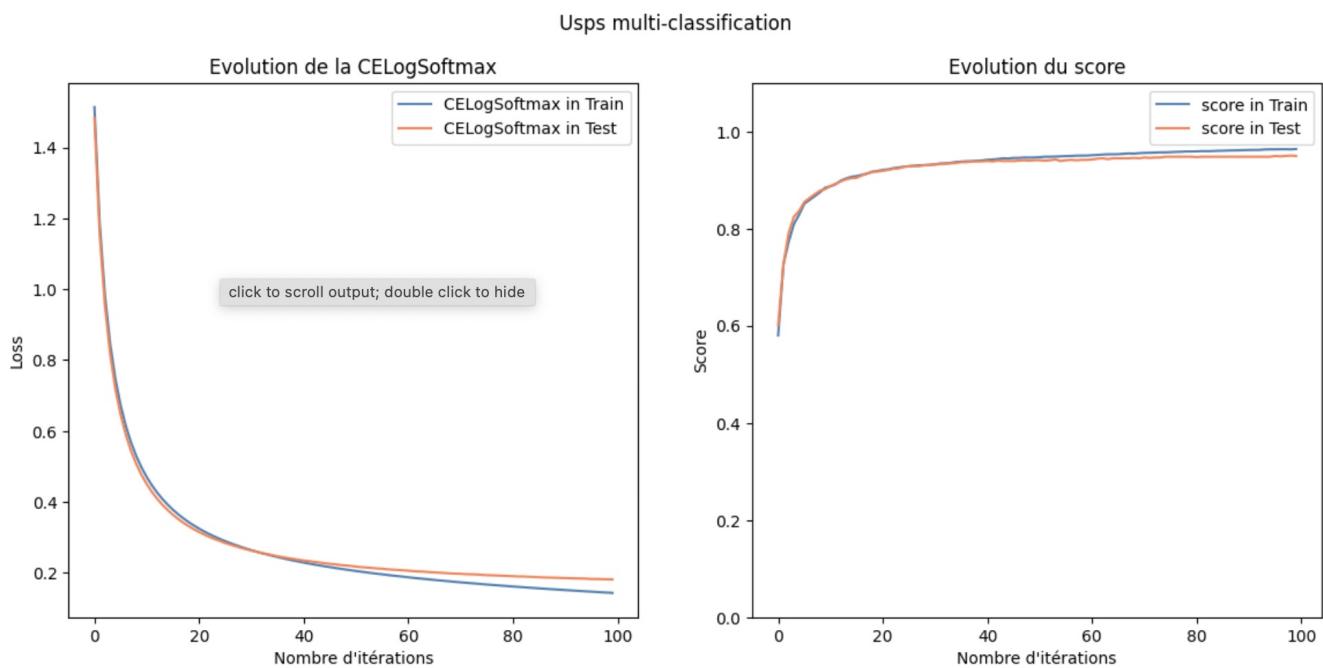


FIGURE 17 – Évolution de la loss et de la précision pendant l'entraînement et le test sur le jeu de données USPS

Matrice de confusion											
Labels	0	1	2	3	4	5	6	7	8	9	
	474	0	0	1	4	3	4	0	1	1	
	1	0	406	0	2	1	0	0	0	1	2
	2	3	0	290	5	6	2	2	2	1	0
	3	2	0	7	241	1	6	0	1	2	0
	4	2	2	10	0	279	0	4	1	2	6
	5	5	0	1	8	6	217	1	0	3	3
	6	0	0	5	0	4	1	251	0	0	0
	7	0	0	2	0	4	1	0	269	1	5
	8	4	3	6	5	5	2	0	3	193	3
	9	0	1	0	1	4	0	0	8	1	266
		0	1	2	3	4	5	6	7	8	9

FIGURE 18 – Matrice de confusion sur le jeu de données USPS

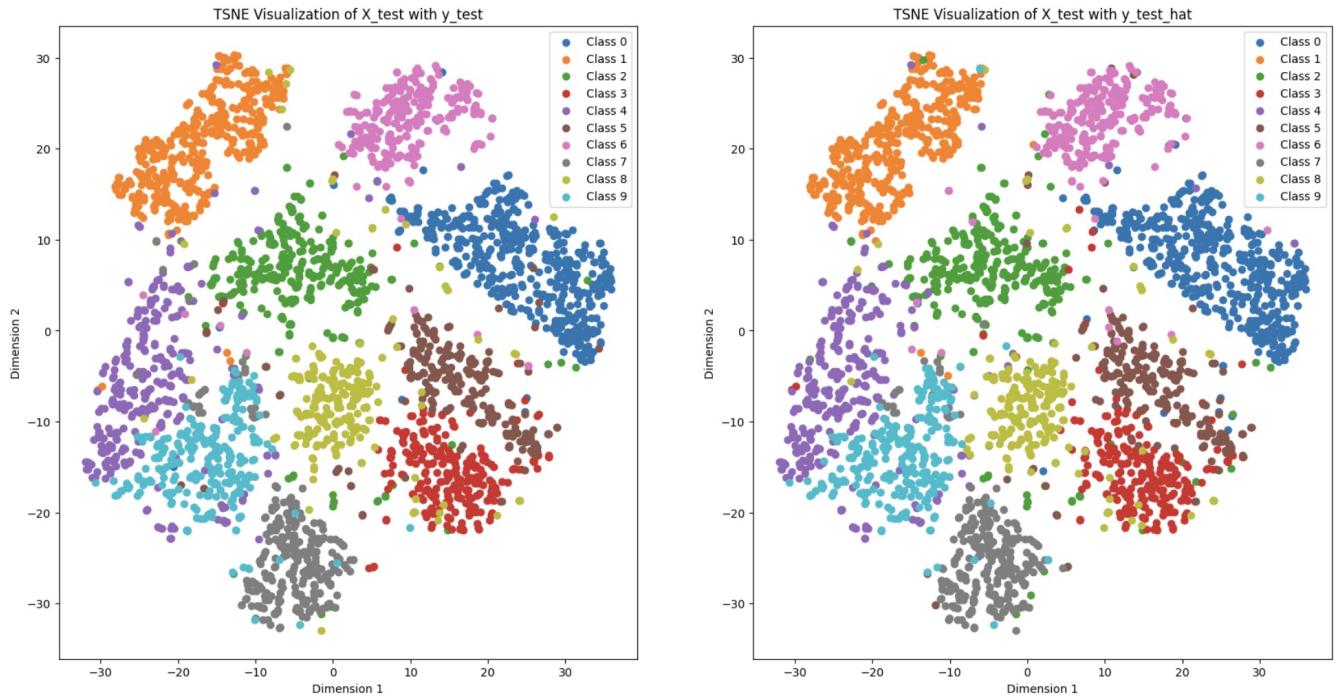


FIGURE 19 – Visualisation TSNE de X\_test avec y\_test et y\_test\_hat sur le jeu de données USPS

**Visualisations :** Le modèle de classification multi-classe sur le jeu de données USPS montre une performance solide avec une précision de 94% sur les données de test. Cette haute précision indique que le modèle est bien capable de généraliser à partir des données d'entraînement.

L'analyse de la matrice de confusion révèle que la plupart des classes sont bien distinguées, bien que certaines confusions subsistent entre des classes visuellement similaires. Cela pourrait être exploré davantage pour comprendre les limitations du modèle actuel.

La visualisation TSNE montre une bonne séparation des classes dans l'espace de projection, ce qui corrobore la haute précision du modèle. Les points colorés représentant les classes prédictives se regroupent de manière cohérente avec les points des classes réelles, indiquant que le modèle fait des prédictions fiables.

En conclusion, le modèle atteint déjà un niveau de précision élevé. Toutefois, des améliorations pourraient être apportées en affinant les hyperparamètres, en augmentant la taille de l'ensemble de données, ou en utilisant des techniques d'augmentation des données pour améliorer encore plus les performances.

## V.2 Exploration supplémentaire avec le jeu de données MNIST

Dans cette partie, nous allons plus loin en utilisant le jeu de données MNIST pour la classification multi-classe. Le jeu de données MNIST est un ensemble de 60 000 images de chiffres manuscrits (0 à 9) pour l'entraînement et 10 000 images pour les tests. Chaque image est de taille 28x28 pixels en niveaux de gris.

### Configuration du modèle :

- Nombre de couches cachées : 3
- Neurones dans la première couche cachée : 128, activation TanH
- Neurones dans la deuxième couche cachée : 64, activation TanH
- Neurones dans la troisième couche cachée : 10, activation Softmax
- Fonction de coût : Cross-Entropy (CELogSoftMax)
- Optimiseur : Stochastic Gradient Descent (SGD)
- Taille de batch : 10
- Nombre d'epochs : 50

### Résultats :

- Précision sur l'ensemble d'entraînement : 0.9692
- Précision sur l'ensemble de test : 0.9200

**Visualisation :** La figure ci-dessous montre quelques exemples d'images de chiffres manuscrits tirées au hasard du jeu de données MNIST avec leurs prédictions correctes :

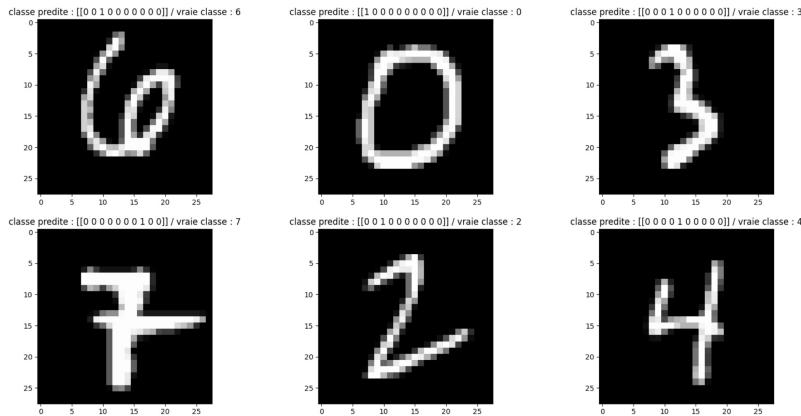


FIGURE 20 – Prédictions correctes sur des exemples de chiffres manuscrits tirés au hasard du jeu de données MNIST

Le modèle de classification multi-classe sur le jeu de données MNIST montre des performances solides avec une précision de 96.92% sur l'ensemble d'entraînement et de 92.00% sur l'ensemble de test. Ces résultats indiquent que le modèle est bien capable de généraliser à partir des données d'entraînement et de faire des prédictions précises sur de nouvelles données.

## VI Mon cinquième se compresse

Nous sommes actuellement sur la partie encodeur/décodeur de notre projet. Un autoencodeur est un type de réseau de neurones utilisé pour apprendre une représentation compressée des données, composé d'un encodeur qui compresses les données d'entrée en une représentation de plus faible dimension, et d'un décodeur qui reconstruit les données d'origine à partir de cette représentation comprimée. Nous allons effectuer plusieurs tâches : l'entraînement

de l'autoencodeur avec différents hyperparamètres, l'utilisation des données encodées pour la classification, l'application de K-Means pour la classification et l'utilisation de l'autoencodeur pour supprimer le bruit.

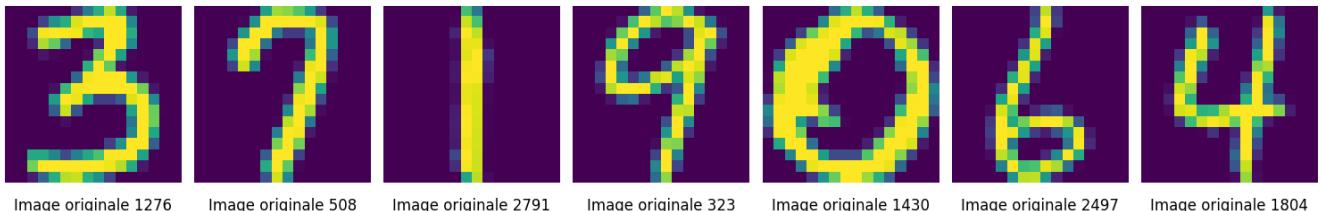
L'architecture de notre autoencodeur se compose de deux parties. L'encodeur commence par une couche d'entrée de 256 neurones, suivie par des couches cachées de 160, 120, 60 et 10 neurones, chacune utilisant la fonction d'activation TanH. Cette séquence de couches réduit progressivement la dimension des données, compressant ainsi l'information. Le décodeur prend ensuite cette représentation compressée et la passe par des couches de 10, 60, 120, et 160 neurones, également avec des activations TanH, pour finalement atteindre une couche de sortie de 256 neurones avec une activation Sigmoid. Cette structure permet à notre autoencodeur de compresser les données d'entrée en une représentation de 10 dimensions et de reconstruire les données originales à partir de cette représentation, optimisant ainsi le traitement et la réduction du bruit des données.

## VI.1 Entrainement de l'autoencodeur avec différents hyperparamètres

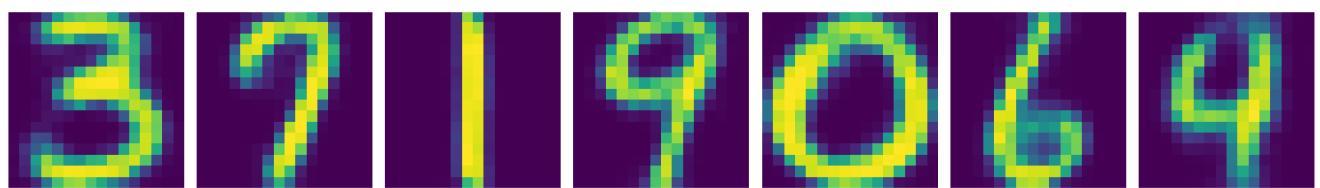
Dans cette étape, nous avons testé plusieurs configurations d'entraînement de l'autoencodeur pour optimiser sa performance dans la classification. Les paramètres que nous avons variés incluent l'initialisation des poids, la taille du lot (batch size) et le taux d'apprentissage (learning rate). Les configurations testées étaient les suivantes :

- "batch size" : 100, "learning rate" : 1e-4
- "batch size" : 5, "learning rate" : 1e-4
- "batch size" : 20, "learning rate" : 1e-4
- "batch size" : 20, "learning rate" : 1e-3
- "batch size" : 20, "learning rate" : 1e-5

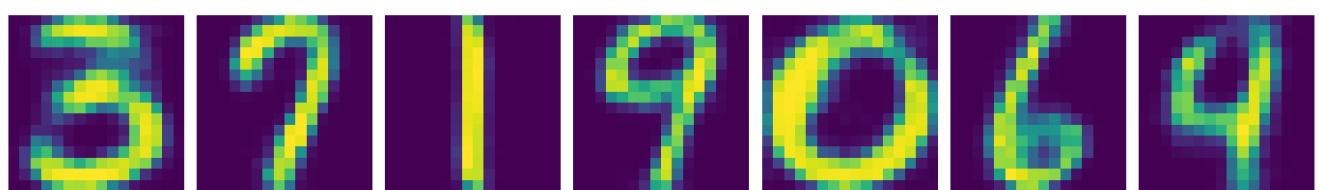
Les images obtenues pour chacune de ces configurations sont les suivantes :



Auto Encodeur avec initialisation he\_normal et batch de size 100 de learning rate 0.0001



Auto Encodeur avec initialisation he\_normal et batch de size 5 de learning rate 0.0001



Auto Encodur avec initialisation he\_normal et batch de size 20 de learning rate 0.0001

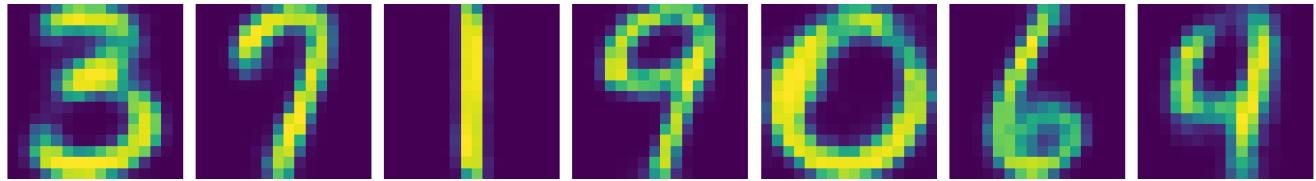


Image reconstruite 1276 Image reconstruite 508 Image reconstruite 2791 Image reconstruite 323 Image reconstruite 1430 Image reconstruite 2497 Image reconstruite 1804

Auto Encodur avec initialisation he\_normal et batch de size 20 de learning rate 0.001

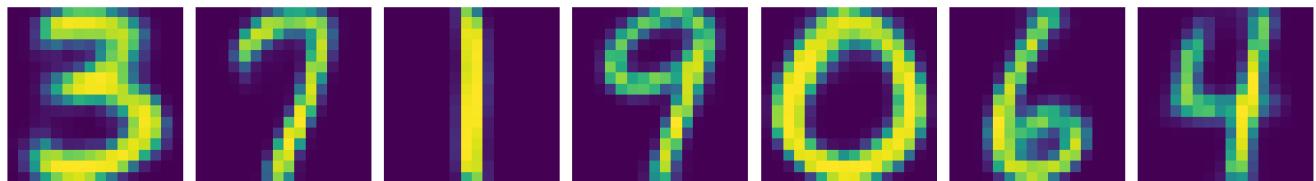


Image reconstruite 1276 Image reconstruite 508 Image reconstruite 2791 Image reconstruite 323 Image reconstruite 1430 Image reconstruite 2497 Image reconstruite 1804

Auto Encodur avec initialisation he\_normal et batch de size 20 de learning rate 1e-05

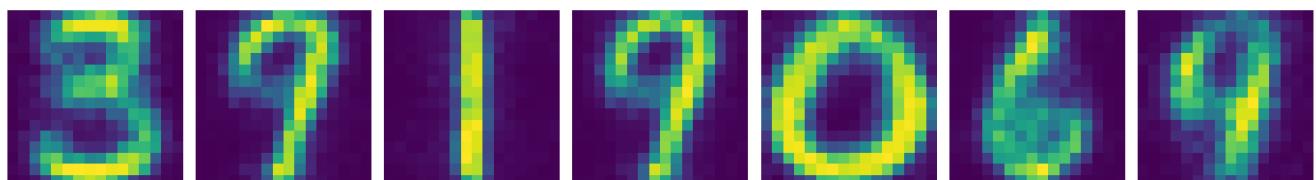


Image reconstruite 1276 Image reconstruite 508 Image reconstruite 2791 Image reconstruite 323 Image reconstruite 1430 Image reconstruite 2497 Image reconstruite 1804

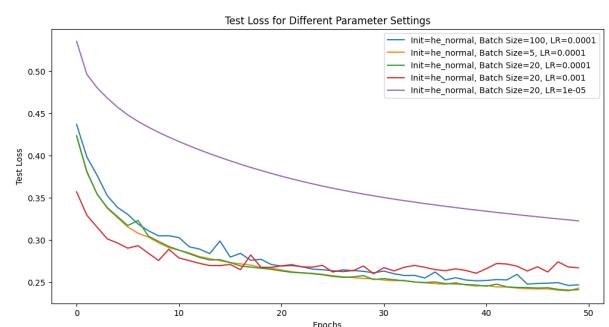
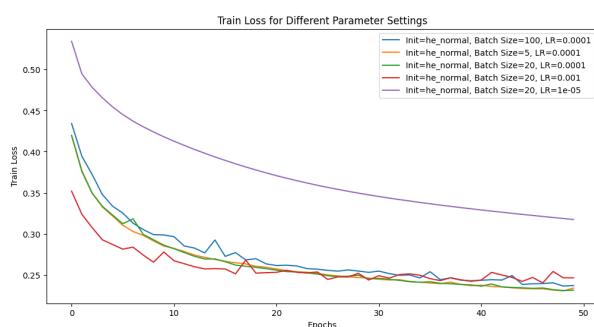


FIGURE 21 – Train and Test Loss des Différentes Configurations

#### L'évaluation des performances des configurations :

- La configuration avec un lot de taille 20 et un taux d'apprentissage de 0.0001 (courbe verte) semble offrir un bon équilibre entre stabilité et rapidité de convergence, aboutissant à la meilleure performance globale dans cette série d'expérimentations.
- Une taille de lot trop petite (5) ou trop grande (100) peut introduire du bruit ou ralentir l'apprentissage respectivement.
- Des taux d'apprentissage trop élevés (0.001) ou trop faibles (0.00001) peuvent entraîner des problèmes de convergence ou ralentir l'apprentissage.

## VI.2 Utilisation des données encodées pour la classification

Pour la classification, nous avons choisi d'utiliser une taille de lot (batch size) de 20 et un taux d'apprentissage (learning rate) de 0.0001, car d'après les expérimentations précédentes, cette configuration s'est révélée être la plus efficace.

En utilisant ces configurations, nous entraînons d'abord l'autoencodeur pour encoder les données, puis nous utilisons ces données encodées pour entraîner un réseau de classification. Nous visualisons ensuite les performances du réseau de classification en termes de précision et de perte. Cette approche nous permet d'évaluer l'efficacité de l'autoencodeur et de la classification multicouche dans notre projet.

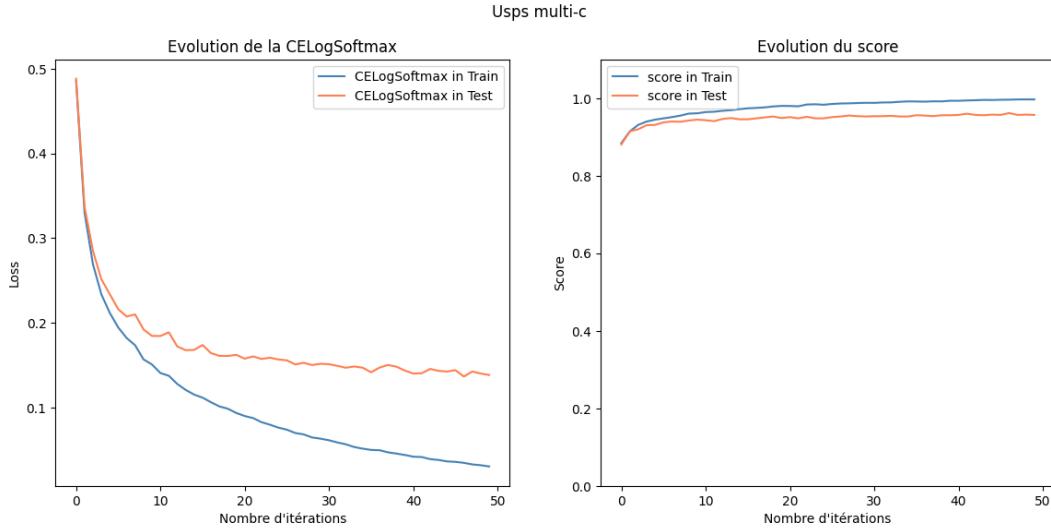


FIGURE 22 – Loss et Score du multiclass

Pour compléter notre analyse et évaluer l'efficacité de notre autoencodeur, nous avons utilisé des techniques de visualisation afin d'observer les données encodées et leur reconstruction. L'objectif est de vérifier si l'autoencodeur a correctement capturé la structure sous-jacente des données et si les données encodées sont bien séparables pour les tâches de classification. Nous avons implémenté une fonction de visualisation qui utilise des techniques de réduction de dimensionnalité telles que t-SNE et PCA .

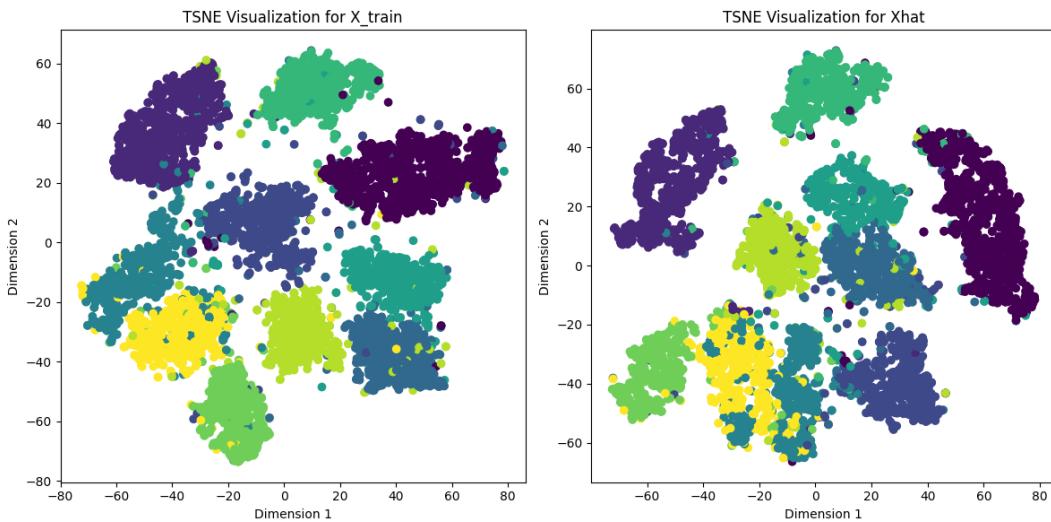


FIGURE 23 – TSNE train

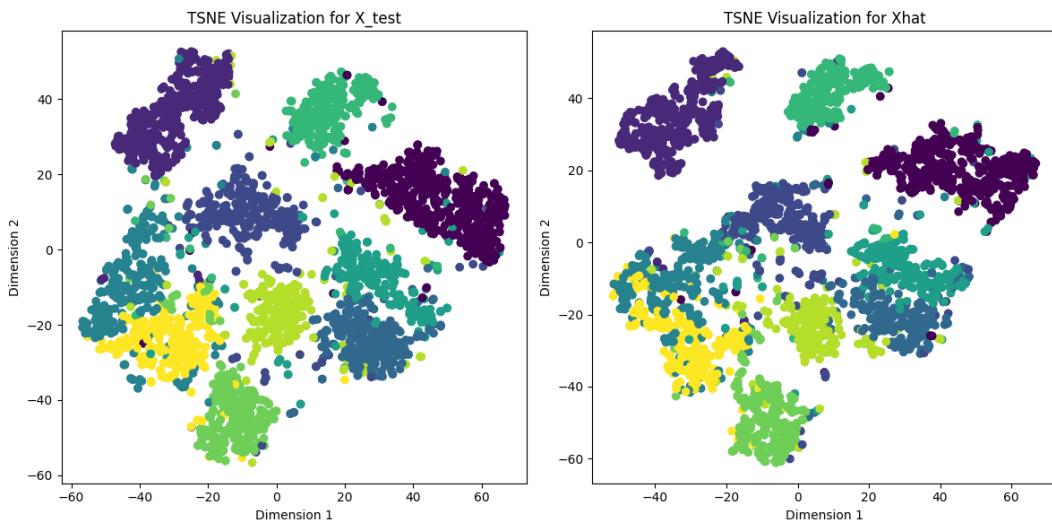


FIGURE 24 – TSNE test

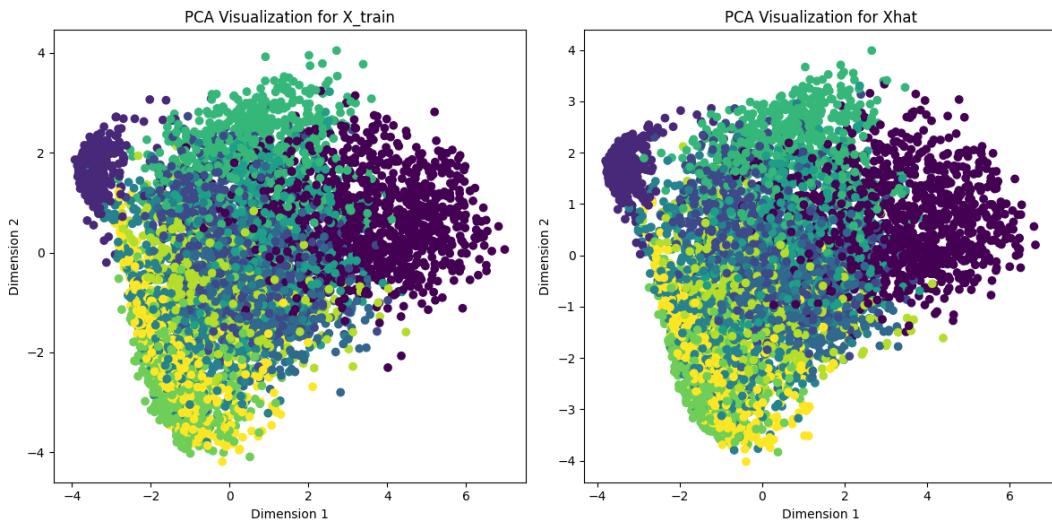


FIGURE 25 – PCA train

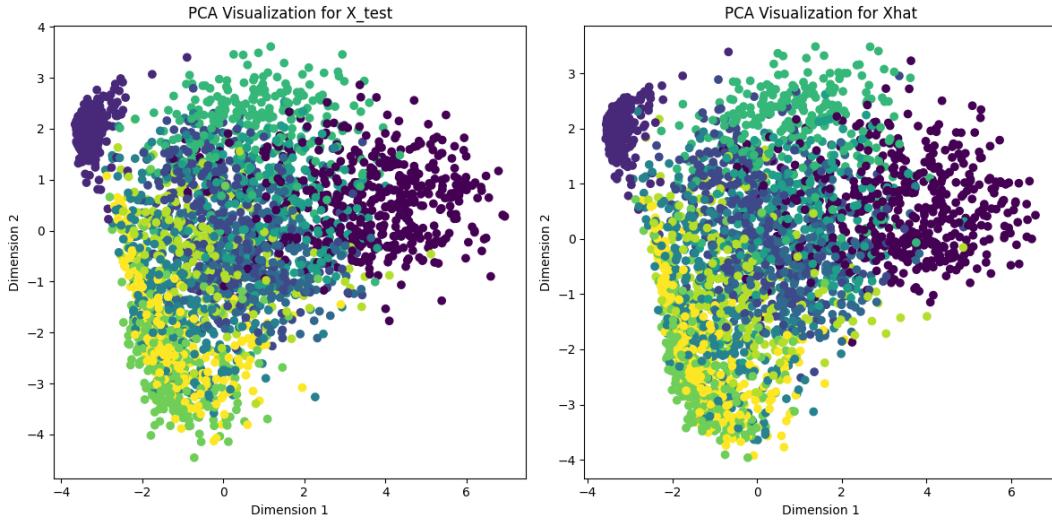


FIGURE 26 – PCA test

### VI.3 Utilisation de K-Means pour la classification

Pour évaluer l'efficacité de notre autoencodeur et explorer les données encodées, nous avons appliqué le clustering K-Means et la visualisation t-SNE. Nous avons transformé les données d'entraînement avec l'encodeur et appliqué K-Means pour les regrouper en 10 clusters. Ensuite, nous avons utilisé t-SNE pour réduire la dimensionnalité des données encodées et les visualiser en 2D.

La visualisation montre que les données encodées sont bien séparées selon leurs classes réelles. Nous avons attribué à chaque cluster l'étiquette de la classe majoritaire et évalué les performances de clustering en calculant les métriques suivantes :

- Accuracy : 0.7895
- Precision : 0.7963
- Recall : 0.7852
- F1 Score : 0.7901

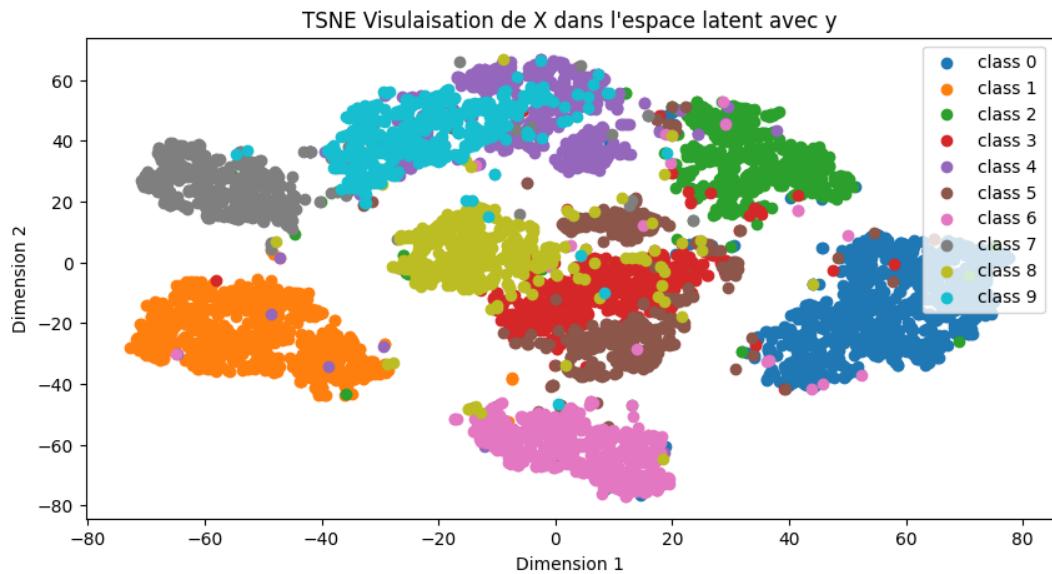


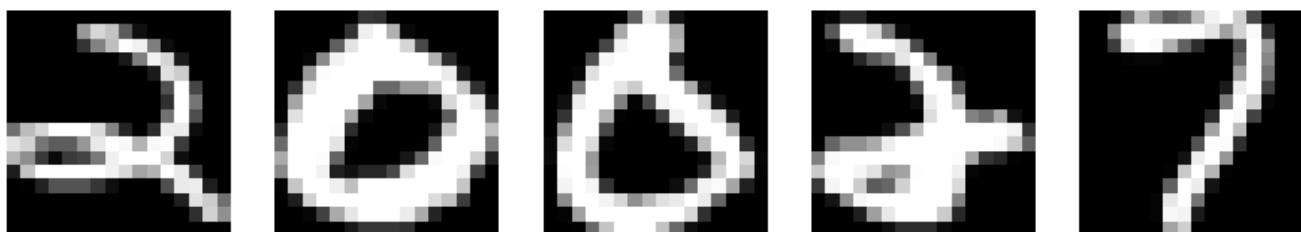
FIGURE 27 – t-SNE

## VI.4 Utilisation de l'autoencodeur pour supprimer le bruit

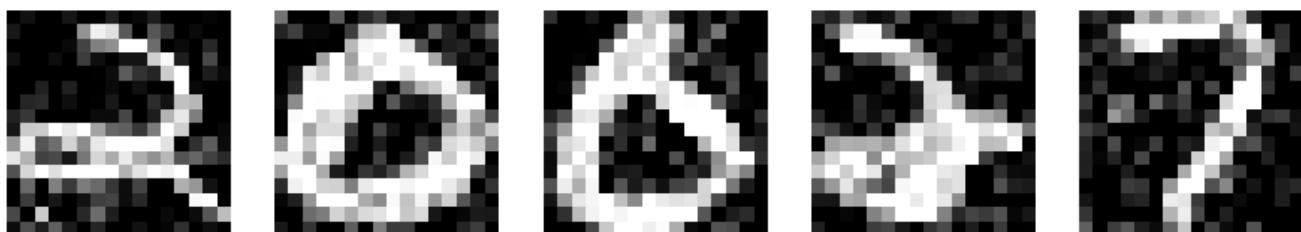
Nous avons exploré l'utilisation d'un autoencodeur pour débruiter des images et améliorer la classification des données. Voici les étapes principales et les résultats obtenus :

- **Ajout de bruit et débruitage des images :**
  - Du bruit gaussien a été ajouté aux images d'entraînement et de test.
  - L'autoencodeur a été entraîné pour débruiter ces images.
- **Architecture de l'autoencodeur :**
  - **Encodeur :**  $256 \rightarrow 300 \rightarrow 200 \rightarrow 100 \rightarrow 50 \rightarrow 10$  neurones, avec activation ReLU.
  - **Décodeur :**  $10 \rightarrow 50 \rightarrow 100 \rightarrow 200 \rightarrow 300 \rightarrow 256$  neurones, avec activation ReLU et Sigmoid.
- **Entraînement de l'autoencodeur :**
  - Entraîné pour 200 époques avec une perte de type binaire cross-entropy et un taux d'apprentissage de 0.00001, optimisé avec SGD et une taille de lot de 50.
- **Classification :**
  - Un classificateur de réseau de neurones a été entraîné sur les images débruitées avec une architecture comprenant trois couches cachées (200, 150, 100 neurones) et une sortie multi-classe.
  - Entraîné pour 200 époques avec une perte de type cross-entropy logarithmique et optimisé avec SGD.
- **Visualisation et évaluation :**
  - Visualisation des images d'origine, bruitées, et débruitées.
  - Le débruitage par l'autoencodeur a amélioré la qualité des images et la performance de classification.
  - Utilisation des métriques de précision, rappel, et score F1 pour évaluer le classificateur.

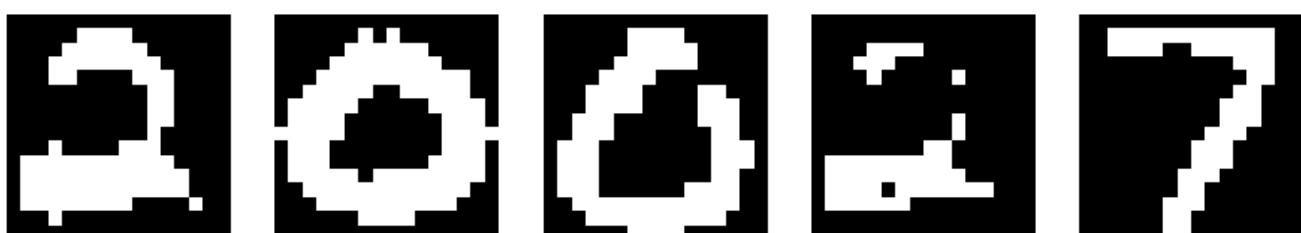
Images d'origine



Images bruitées

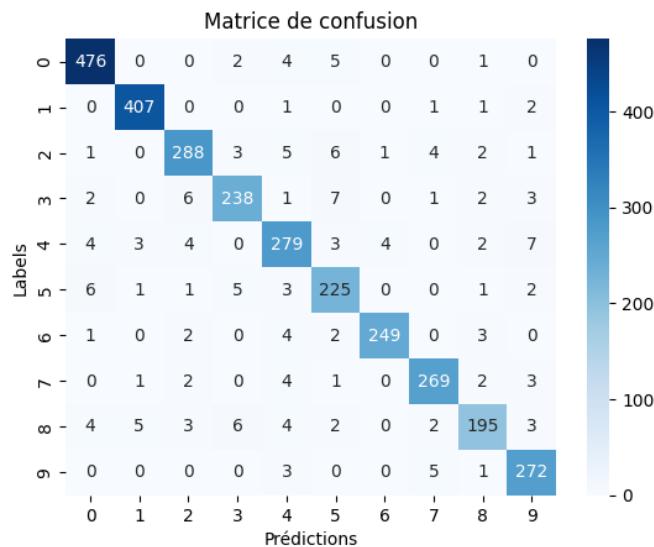


Images débruitées



Les résultats obtenus sont les suivants :

- **Précision de validation** : 95.99%
- **Précision de test** : 94.43%
- **Matrice de confusion pour l'ensemble de validation** :



Par la suite, nous avons appliqué le K-Means sur les données encodées pour analyser les clusters après débruitage des données. Les étapes sont les suivantes :

- **Application du K-Means sur les données encodées** :
  - Nous avons appliqué l'algorithme de K-Means avec 10 clusters sur les données de test encodées.
- **Visualisation des clusters avec t-SNE** :
  - Nous avons utilisé t-SNE pour réduire les dimensions des données encodées à deux dimensions afin de visualiser les clusters formés par K-Means.
  - Chaque cluster a été visualisé dans un graphique, avec une couleur différente pour chaque cluster.

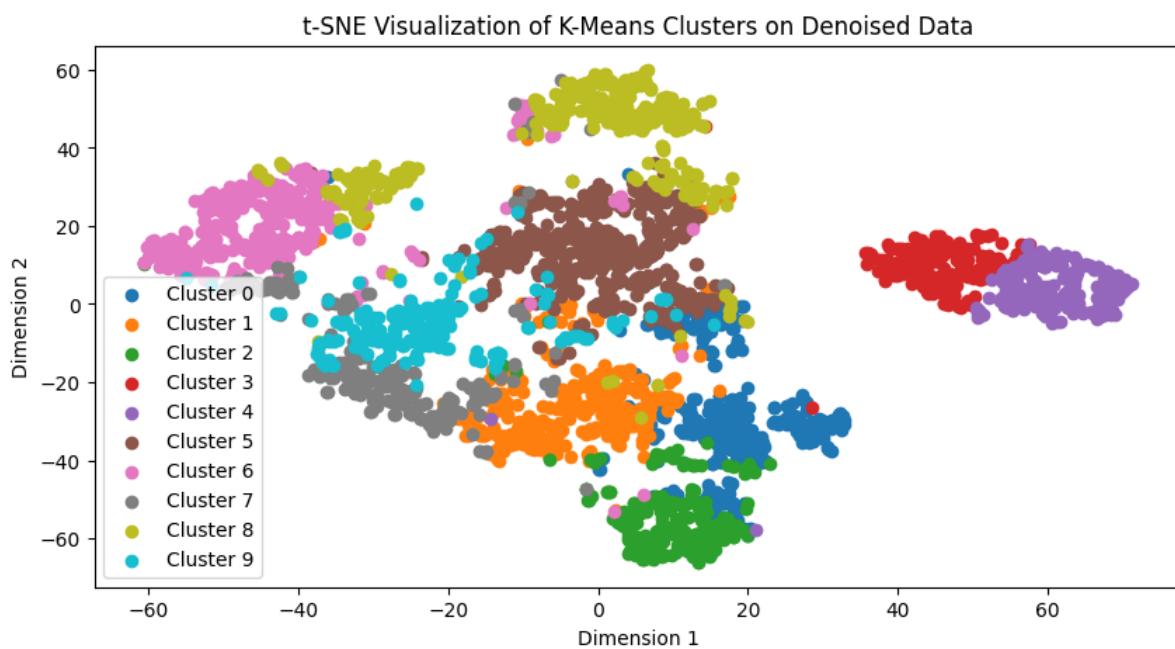


FIGURE 28 – Visualisation t-SNE des clusters K-Means sur les données débruitées

## VII Mon sixième se convole et mon tout s'améliore

Dans cette section, nous explorons l'utilisation des réseaux de neurones convolutionnels (CNN) pour la classification d'images. Nous commençons par l'implémentation et l'entraînement d'un modèle CNN en utilisant principalement des opérations de convolution 1D sur le dataset USPS. Ensuite, nous étendons notre analyse en implémentant une couche de convolution 2D pour traiter les données d'images. Nous évaluons ensuite les performances de ces deux approches.

### VII.1 Convolution 1D

Dans cette partie nous avons implémenté et entraîné un réseau de neurones convolutionnel (CNN) pour la classification d'images, en utilisant principalement des opérations de convolution 1D. Tout d'abord, nous avons utilisé les données du dataset USPS, et construit un réseau de neurones convolutionnel 1D composé des couches suivantes :

- **Conv1D** : taille de filtre = 3, canaux d'entrée = 1, canaux de sortie = 32, stride = 1
- **MaxPool1D** : taille du kernel = 2, stride = 2
- **Flatten**
- **Linear (Fully Connected)** : entrée = 4064, sortie = 100
- **ReLU**
- **Linear (Fully Connected)** : entrée = 100, sortie = 10
- **Softmax**

Pour l'entraînement du modèle, nous avons utilisé une descente de gradient stochastique (SGD) pour entraîner notre modèle :

- **Fonction de Perte** : La fonction de perte utilisée est l'entropie croisée (Cross-Entropy Loss)
- **Optimisation**
- **Batch Processing** : Les données sont divisées en petits lots (mini-batchs)
- **Epochs** : 300

Après l'entraînement, nous avons évalué les performances du modèle sur les données de test et visualisé les résultats.

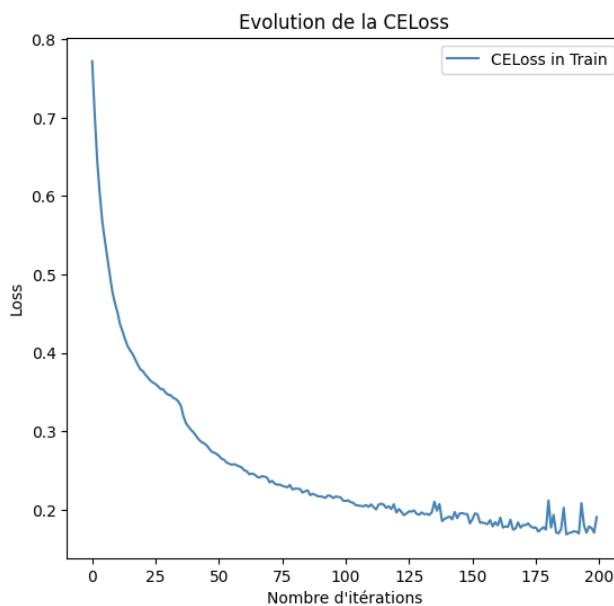


FIGURE 29 – CE-loss du train

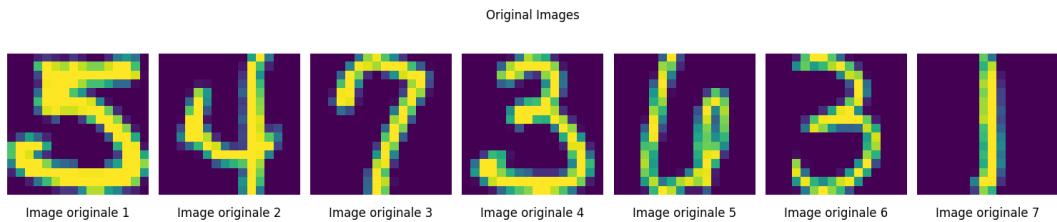


FIGURE 30 – Exemples de données

Nous avons obtenu les prédictions suivantes pour chacune : 5, 4, 9, 3, 0, 3, 1.

Nous présentons ci-dessous la matrice de confusion ainsi que le rapport de classification pour évaluer les performances de notre modèle :

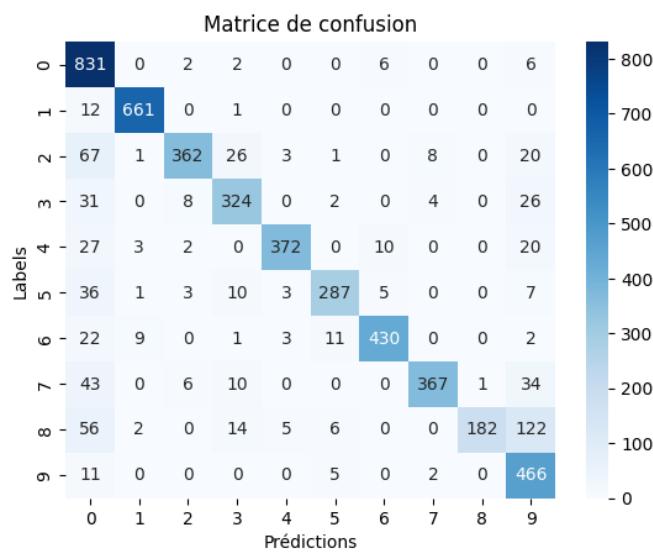


FIGURE 31 – Matrice de confusion du classification

Class	Precision	Recall	F1-score	Support
0	0.731514	0.98111	0.838124	847
1	0.976366	0.980712	0.978534	674
2	0.945170	0.741803	0.831228	488
3	0.835052	0.820253	0.827586	395
4	0.963731	0.857143	0.907317	434
5	0.919872	0.815341	0.864458	352
6	0.953437	0.899582	0.925727	478
7	0.963255	0.796095	0.871734	461
8	0.994536	0.470284	0.638596	387
9	0.662873	0.96281	0.785173	484
<b>Accuracy</b>	0.856400			5000

TABLE 5 – Matrice de confusion et rapport de classification

Les résultats indiquent des performances globalement solides du modèle, avec des scores de précision, de rappel et de F1 élevés pour plusieurs classes, et une accuracy globale de 85,64%, suggérant une capacité de classification fiable malgré quelques écarts dans la précision pour certaines classes spécifiques.

## VII.2 Convolution 2D

Dans cette partie, nous avons implémenté la couche de convolution 2D (Conv2D) pour traiter les données d'images.

L'architecture du modèle Conv2D est définie comme suit :

- **Conv2D** : filtres de taille 3x3 avec 32 canaux de sortie, pas de stride spécifié
- **MaxPool2D** : fenêtre de pooling 2x2 avec un stride de 2
- **Flatten**
- **Linear (Fully Connected)** :  $32 * 7 * 7$  entrées vers 100 sorties
- **ReLU**
- **Linear (Fully Connected)** : 100 entrées vers 10 sorties (nombre de classes)
- **Softmax**

Nous visualiserons également la fonction de perte Cross-Entropy pour évaluer l'apprentissage du modèle.

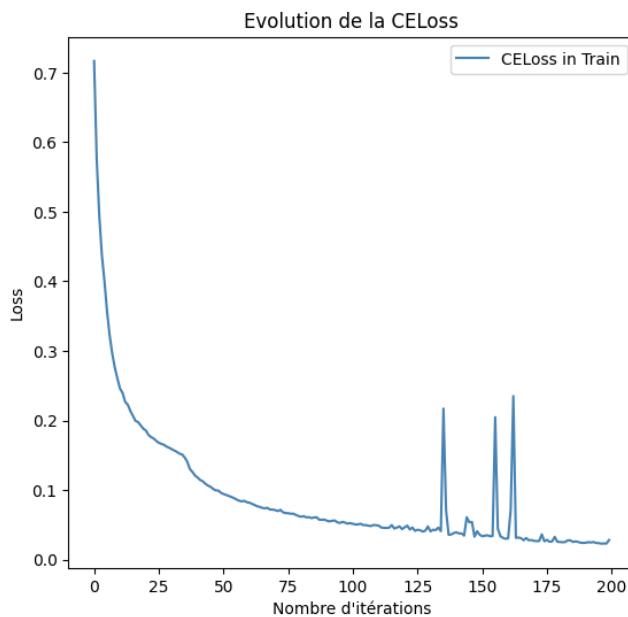


FIGURE 32 – CE-loss du train

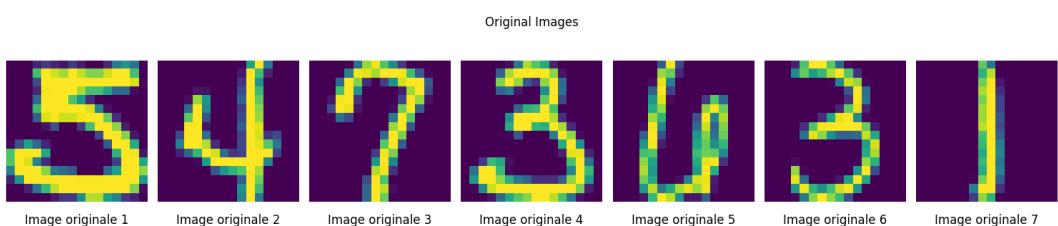


FIGURE 33 – Exemples de données

Nous avons obtenu les prédictions correspondantes pour chacune de ces données : 5, 4, 7, 3, 6, 3, 1.

Les performances de notre modèle sont évaluées à l'aide de la matrice de confusion et du rapport de classification, présentés ci-dessous :

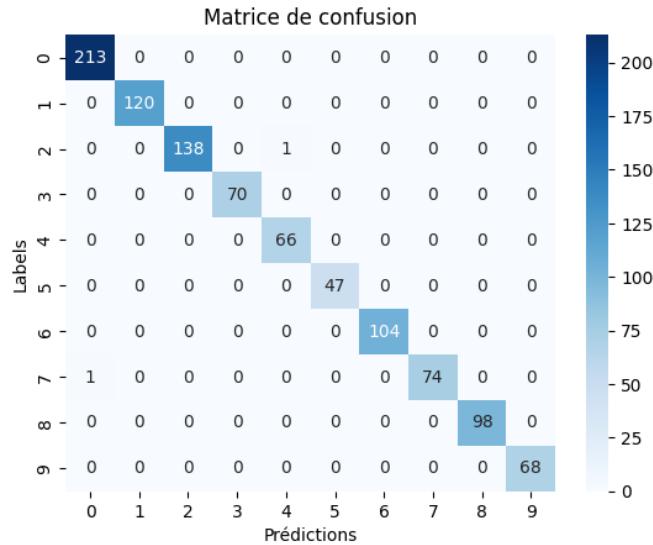


FIGURE 34 – Matrice de confusion du classification

Class	Precision	Recall	F1-score	Support
0	0.995327	1.0	0.997658	213
1	1.0	1.0	1.0	120
2	1.0	0.992806	0.99639	139
3	1.0	1.0	1.0	70
4	0.985075	1.0	0.992481	66
5	1.0	1.0	1.0	47
6	1.0	1.0	1.0	104
7	1.0	0.986667	0.993289	75
8	1.0	1.0	1.0	98
9	1.0	1.0	1.0	68
<b>Accuracy</b>	0.998000			1000

TABLE 6 – Résultats de classification

Les résultats indiquent des performances exceptionnelles du modèle, avec des scores de précision, de rappel et de F1 élevés pour toutes les classes. L'accuracy globale de 99,8% confirme la fiabilité et l'efficacité du modèle dans la classification des données.

## VIII Conclusion

Ce projet nous a permis de plonger profondément dans le fonctionnement des réseaux de neurones en implémentant chaque composant essentiel, des couches linéaires aux fonctions d'activation en passant par les fonctions de perte. En expérimentant avec différentes architectures et hyperparamètres, nous avons pu comprendre l'impact de ces choix sur les performances du modèle.

En résumé, cette expérience nous a offert une compréhension pratique et concrète du processus d'apprentissage automatique par les réseaux de neurones. En plus de renforcer nos connaissances théoriques, nous avons développé des compétences en programmation et en résolution de problèmes.