Machine Learning

Réseau de neurones

BOUCHOUCHI Nour et BORCHANI Sofia

Mars-Mai 2023

Table des matières

Introduction **4**

I. Mon premier est... linéaire ! **5**
1. Expérimentations 5
   a. Regression 5
   b. Regression bruité 6
   c. Classification 6
   d. Classification bruitée 7
2. Récapitulatif des résultats 7

II. Mon second est ... non-linéaire ! **8**
1. Expérimentations 8
   a. Classification 8
   b. Variation du nombre de neurones 9
2. Récapitulatif des résultats 10

III. Mon troisième est un encapsulage **11**
1. Expérimentations 11
   a. Cas hors-ligne (batch) 11
   b. Cas en-ligne (stochastique) 12
   c. Hybride : mini-batch 12
2. Récapitulatif 13

IV. Mon quatrième est multi-classe **14**
1. Expérimentations 14
   a. Tests sur les données de chiffres manuscrits 14
   b. Ajout de bruit 15
2. Récapitulatif 16

V. Mon cinquième se compresse **17**
1. Expérimentations 18
   a. Visualisation des images reconstruites après une forte compression 18
   b. Classification des images reconstruites 19
   c. Visualisation des représentations obtenues dans un espace 2D 19
   d. Performances en débruitage 21

2. Récapitulatif des résultats obtenus en classification 26

VI. Mon sixième se convole et mon tout s'améliore **26**
1. Expérimentations 27
   a. Max Pooling 27
   b. Average Pooling 28
2. Récapitulatif des résultats 28

Introduction

Le but de ce projet est d'implémenter un réseau de neurones en s'inspirant des anciennes versions de PyTorch en Lua, ainsi que des implémentations analogues qui permettent d'obtenir des réseaux génériques très modulaires. Chaque couche du réseau est considérée comme un module et le réseau est composé d'un ensemble de modules. Il convient de souligner que les fonctions d'activation sont également considérées comme des modules.

Dans la première partie de notre projet, nous avons réalisé une régression linéaire dans le but de modéliser un réseau de neurones à une seule couche linéaire.

Dans la deuxième partie, nous avons élaboré un réseau de neurones non linéaire en incorporant deux fonctions d'activation, à savoir la fonction tangente et la fonction sigmoïde.

La troisième partie a été consacrée à l'optimisation des procédures de descente de gradient en automatisant le processus de forward et backward du réseau de neurones.

La quatrième partie a porté sur la classification multi-classe, où chaque dimension correspond à la probabilité d'appartenance à une classe donnée.

La cinquième partie a traité de l'auto-encodage, une technique d'apprentissage non supervisé permettant d'apprendre un encodage efficace des données afin de réduire leurs dimensions.

Enfin, dans la sixième partie, nous avons abordé les réseaux de neurones convolutifs et leur utilisation pour extraire des caractéristiques d'images en appliquant des filtres.

Au cours de ce rapport, nous allons détailler les méthodes et les résultats obtenus pour chaque partie du projet.

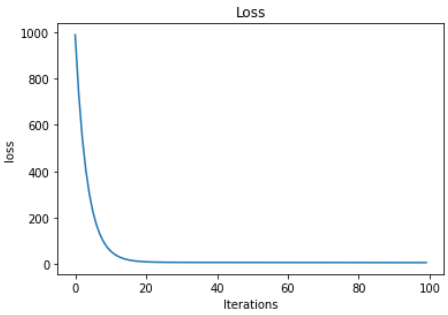
I. Mon premier est... linéaire !

La première partie de ce projet consiste à implémenter deux classes nécessaires pour réaliser une régression linéaire. Tout d'abord, une classe MSELoss pour calculer la fonction de coût Mean Squared Error (MSE) à minimiser. Ensuite, une classe Linear pour représenter un réseau à une couche linéaire.

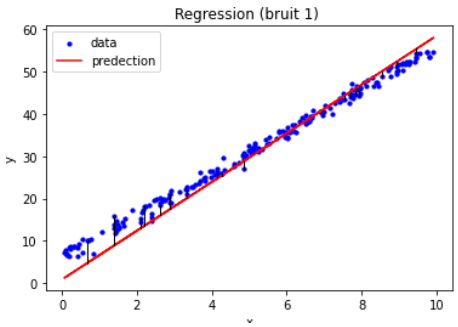
1. Expérimentations

a. Regression

Dans le cadre de nos expérimentations, nous avons généré des points aléatoires selon une loi affine avec une perturbation gaussienne, puis utilisé notre modèle de régression linéaire pour tracer une droite correspondant à la fonction $y = ax + b$. Les coefficients a et b ont été optimisés par le modèle pour établir une corrélation entre les points et la droite de régression linéaire.



(a) Courbe loss de la régression linéaire.



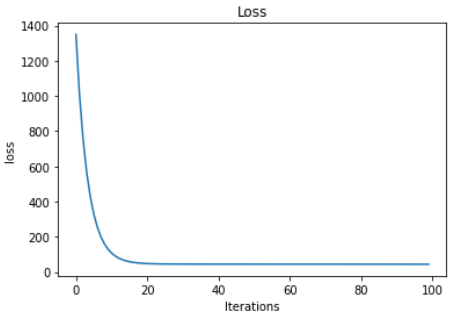
(b) Regression linéaire de données faiblement bruitées.

FIGURE 1 – Regression linéaire sur des données faiblement bruitées.

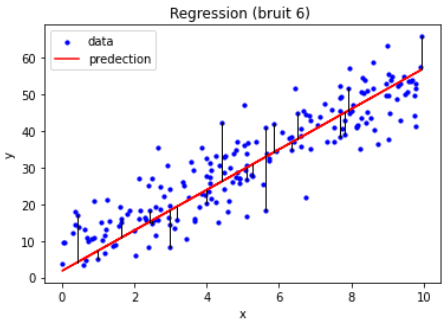
Lors de l'analyse de la première figure, nous pouvons observer que la fonction de perte présente une décroissance rapide, témoignant ainsi de la capacité du réseau à optimiser les poids et biais en un temps limité. Celui-ci a convergé en seulement 20 itérations.

b. Regression bruité

Nous avons égelement évalué la capacité de notre modèle à effectuer des prédictions précises en présence de données bruitées.



(a) Courbe loss de la régression linéaire sur données bruitées.



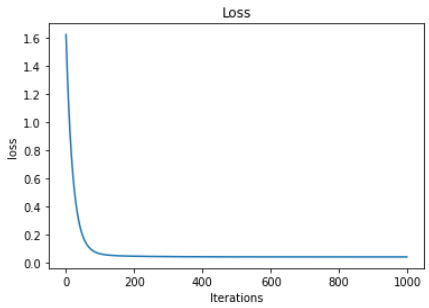
(b) Régression linéaire sur des données bruitées.

FIGURE 2 – Régression linéaire sur des données bruitées (6).

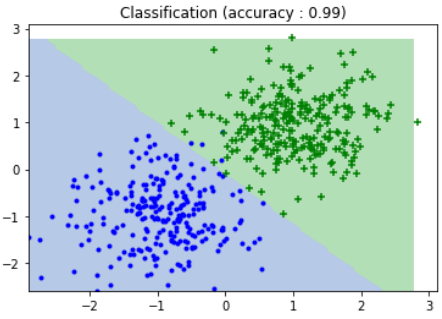
Malgré la présence de perturbations gaussiennes dans les données, notre modèle s'est révélé performant et robuste. En effet le réseau coverge toujours en seulement 20 itérations. On observe sur la figure 2b quelques exemples de MSE qui ont été représentées.

c. Classification

Dans une seconde série de tests, nous avons soumis notre modèle à une tâche de classification pour tracer une frontière linéaire qui optimise la séparation de deux classes de points distinctes.



(a) Courbe loss pour une classification sans bruit.



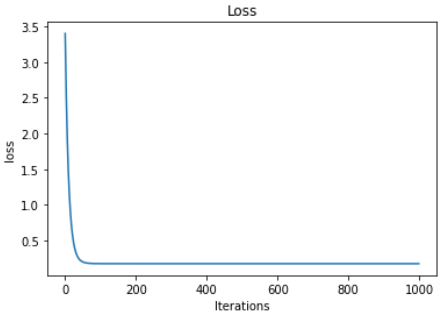
(b) Classification sans bruit.

FIGURE 3 – Classification dans le cas linéaire et sans bruits dans les données.

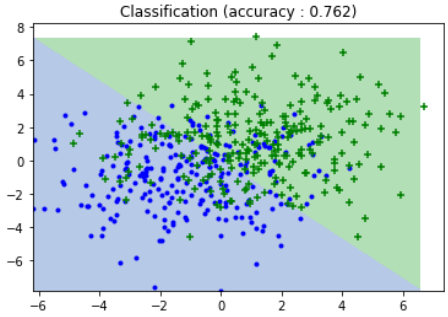
Bien que nous ayons effectué 1000 itérations lors de l'entraînement du réseau, celui-ci a convergé en moins de 100 itérations, démontrant ainsi sa grande efficacité puisqu'il classe en outre les données avec une précision de 99%.

d. Classification bruitée

L'ajout de bruit a réduit les performances de notre modèle, comme en témoigne sa précision qui tombe alors à seulement 76%. La frontière obtenue est néanmoins optimale [figure 4b].



(a) Courbe loss pour une classification linéaire bruitée.



(b) Classification linéaire de données bruitées.

FIGURE 4 – Classification linéaire de données bruitées.

2. Récapitulatif des résultats

	Taux de bonne classification
Sans bruit	0,99%
Avec bruit	0,702%

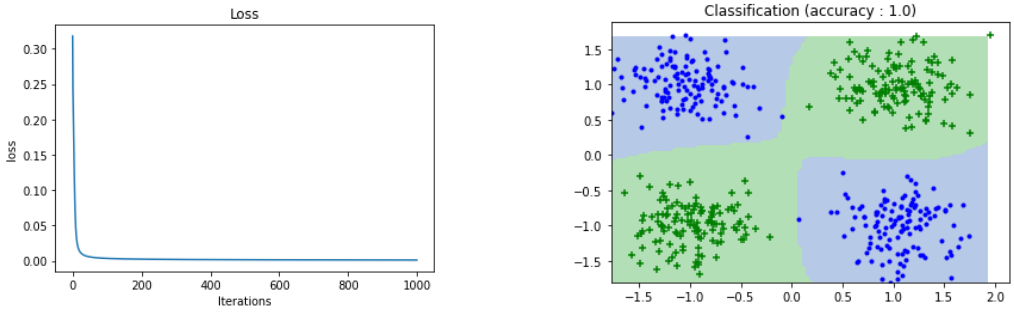
II. Mon second est ... non-linéaire !

Notre objectif pour la deuxième partie est de construire un réseau de neurones avec deux couches linéaires, utilisant une activation tangente entre les deux couches et une activation sigmoïde à la sortie. Nous utilisons les classes TanH et Sigmoide pour implémenter ces fonctions d'activation respectives. Ce modèle cherche à minimiser la MSE.

1. Expérimentations

a. Classification

Nous avons testé un modèle à 5 neurones sur les données d'un problème de classification binaire sans bruit.



(a) Courbe loss.

(b) Classification de données non linéaires.

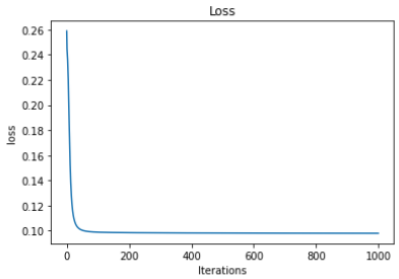
FIGURE 5 – Classification de données non linéaires sans bruit.

Le taux de bonne classification obtenu dans cette configuration est de 100% [Figure 5b] avec un covergence de la fonction de perte en moins de 50 itérations [Figure 5a].

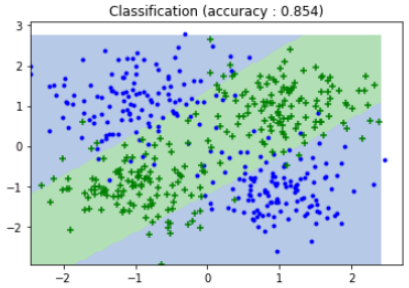
b. Variation du nombre de neurones

Nous avons également évalué l'aptitude de notre modèle à effectuer des classifications en présence de données bruitées. Pour ce faire, nous avons procédé à une variation du nombre de neurones afin d'observer l'effet de cette variable sur les performances.

- Réseau à 2 neurones



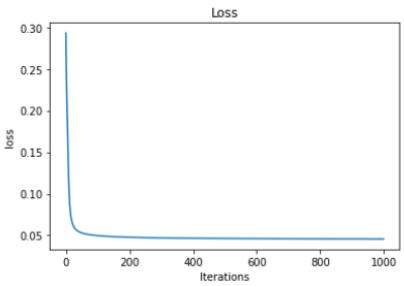
(a) Courbe loss 2 neurones.



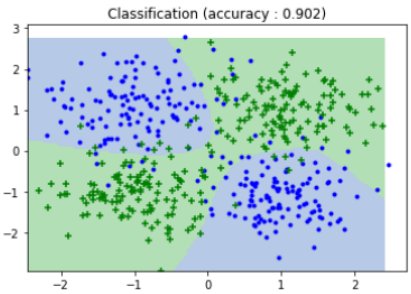
(b) Classification 2 neurones.

FIGURE 6 – Classification pour un réseau à 2 neurones.

- Réseau à 8 neurones



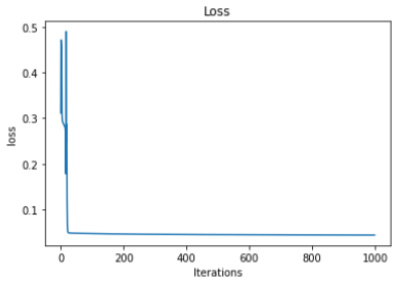
(a) Courbe loss 8 neurones.



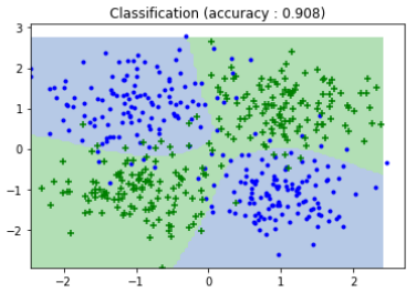
(b) Classification 8 neurones.

FIGURE 7 – Classification pour un réseau à 8 neurones.

- Réseau à 16 neurones



(a) Courbe loss pour un réseau à 16 neurones.



(b) Classification 16 neurones.

FIGURE 8 – Classification pour un réseau à 16 neurones.

On observe une hausse du taux de bonnes classifications lorsque le nombre de neurones augmente. En effet, ces taux sont de 85,4%, 90,2% et 90,8% pour des réseaux de neurones à 2, 8 et 16 neurones respectivement. Ainsi, l'augmentation du nombre de neurones améliore significativement les performances de notre modèle.

2. Récapitulatif des résultats

	Nombre de neurones	**Taux de bonne classification**
Sans bruit	5	100%
Avec bruit	2	85,4%
	8	90,2%
	16	90,8%

III. Mon troisième est un encapsulage

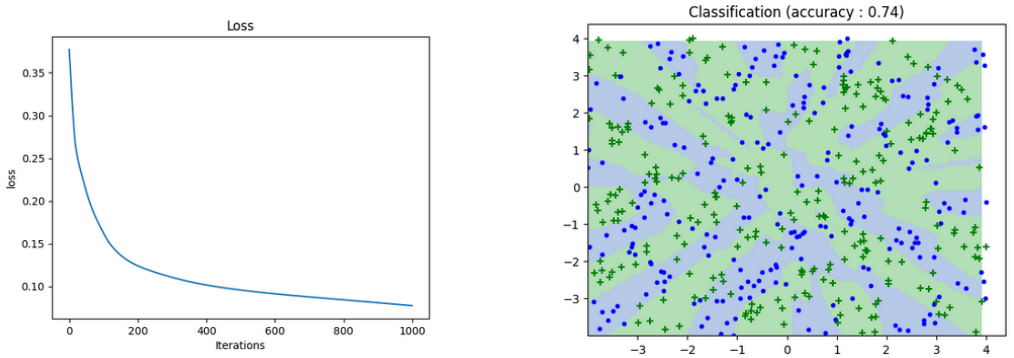
Nous avons constaté la répétition des opérations de chaînage entre modules lors de la descente de gradient. Nous avons donc mis en place une classe Sequentiel qui permet d'ajouter des modules en série et d'automatiser les procédures de forward et backward, quelle que soit la taille du réseau.

De même, nous avons créé une classe Optim qui permet de condenser une itération de gradient et de spécifier la taille des batchs lors de l'apprentissage.

1. Expérimentations

a. Cas hors-ligne (batch)

À chaque époque de correction des paramètres du modèle, nous itérons sur l'ensemble de la base de données d'exemples.

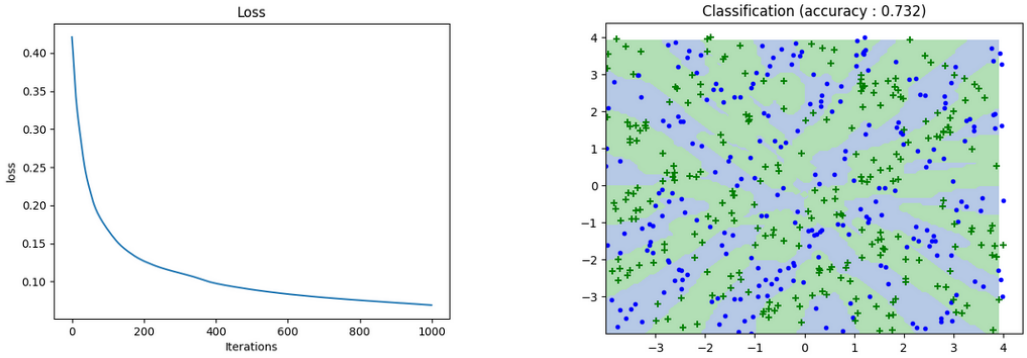


(a) Courbe loss. (b) Classification dans la cas batch.

FIGURE 9 – Classification dans le cas d'une descente de gradient en batch.

b. Cas en-ligne (stochastique)

Lors de chaque itération, un exemple est sélectionné au hasard dans la base de données pour effectuer la mise à jour des paramètres.



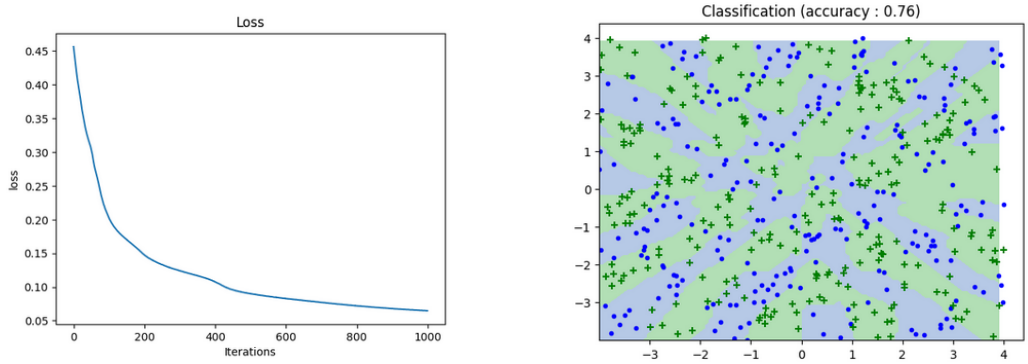
(a) Courbe loss (stochastique) (b) Classification (stochastique).

FIGURE 10 – Classification dans le cas d'une descente de gradient stochastique.

c. Hybride : mini-batch

À chaque époque, des petits sous-ensembles d'exemples sont tirés au hasard, la correction se fait selon le gradient calculé sur ces exemples.

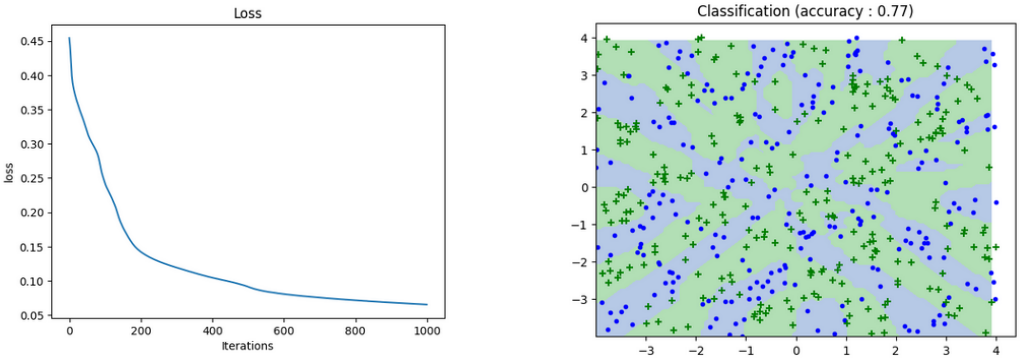
- Taille du batch = 100



(a) Courbe loss (mini-batch 100). (b) Classification (mini-batch 100).

FIGURE 11 – Classification dans le cas d'une descente de gradient avec mini-batch de taille 100.

- Taille du batch = 10



(a) Courbe loss (mini-batch 10). (b) Classification (mini-batch 10).

FIGURE 12 – Classification dans le cadre d'une descente de gradient avec mini-batch de taille 10.

2. Récapitulatif

	Taux de bonne classification
Batch	74%
Stochastique	73,2%
Mini-batch 100	76%
Mini-batch 10	77%

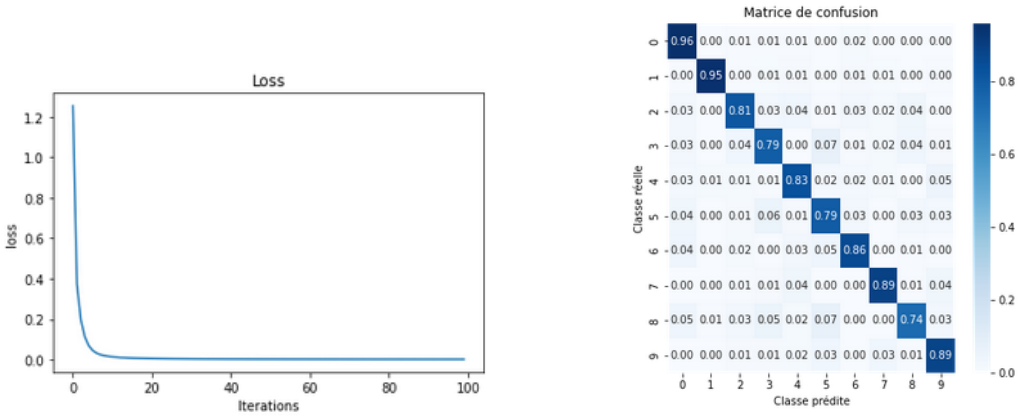
IV. Mon quatrième est multi-classe

En classification multi-classe, chaque dimension correspond à la probabilité d'appartenance à une classe donnée. La prédiction de la classe se fait alors en sélectionnant simplement la dimension ayant la probabilité la plus élevée.

Nous utilisons un Softmax passé au logarithme suivi d'un coût cross-entropique, cette combinaison permet d'obtenir une mesure de l'erreur plus précise. En effet la MSE est souvent inadaptée pour les tâches de classification car elle a tendance à minimiser la moyenne des erreurs, plutôt que de maximiser la probabilité d'affectation à la classe correcte.

1. Expérimentations

a. Tests sur les données de chiffres manuscrits



(a) Courbe loss. (b) Matrice de confusion.

FIGURE 13 – Classification sur des chiffres manuscrits à partir d'un réseau multiclasse.

Pour les données de chiffres manuscrits nous obtenons un taux de bonnes classifications de 86,6% avec une convergence de la fonction de perte en moins de 10 itérations.

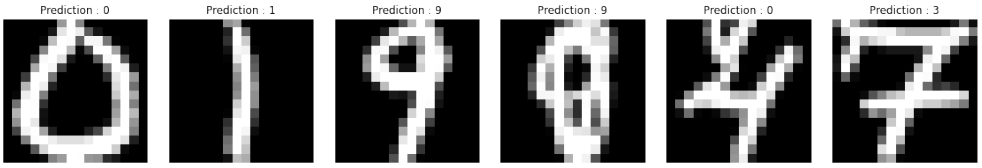


FIGURE 14 – Prédiction, correctes ou non, sur des chiffres manuscrits. Les prédictions sont : 0, 1, 9, 9, 0, 3.

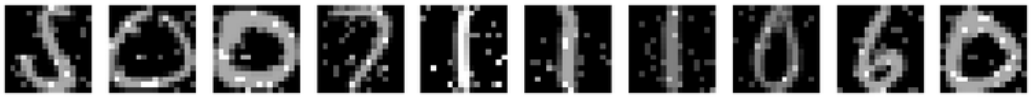
Le modèle présente une performance globalement satisfaisante sur la classification des données, cependant il éprouve des difficultés à identifier certains chiffres mal manuscrits ou présentant des particularités ainsi que le montre la figure 14 pour laquelle un 4 est par exemple classifié comme un 0.

b. Ajout de bruit

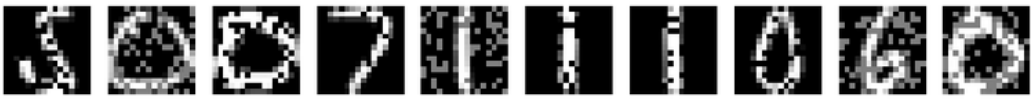
- Données originales : 86,6%

- Bruit gaussien : 85,4%

- Bruit de poisson : 80,1%



- Bruit sel et poivre : 69,1%



Nous constatons que l'inclusion de bruit gaussien n'affecte que légèrement les performances du modèle, avec une diminution de l'ordre de 1%. En revanche, l'ajout de bruit de Poisson s'avère plus perturbateur, entraînant une

baisse des performances de plus de 6 points. Enfin, l'inclusion de bruit sel et poivre a un impact considérable sur les performances, entraînant une chute importante à 69,1%.

2. Récapitulatif

	Taux de bonne classification
Sans bruit	86,6%
Bruit gaussien (0,5)	85,4%
Bruit de poisson (0,5)	80,1%
Bruit sel et poivre (0,25)	69,1%

V. Mon cinquième se compresse

L'objectif d'un auto-encodeur est d'apprendre un encodage efficace des données afin d'en réduire les dimensions. Il doit ensuite être capable de décoder la représentation latente obtenue afin d'obtenir l'image d'origine.

Les applications permises par un auto-encodeur sont diverses : débruitage d'images, reconstruction de parties cachées, clustering, visualisation. Dans cette partie, nous nous sommes penchées sur les tâches de visualisation, de classification des données reconstruites, de visualisation des représentation latente obtenues dans un espace 2D et de performance de débruitage. Pour cela, nous nous sommes à nouveau basées sur la base de données de chiffres USPS.

L'architecture de notre réseau est la suivante :

Encodeur :
- Linear(256,100)
- TanH
- Linear(100,10)
- TanH

Décodeur :
- Linear(10,100)
- TanH
- Linear(100,256)
- Sigmoid

1. Expérimentations

a. Visualisation des images reconstruites après une forte compression

Nous avons tout d'abord cherché à visualiser les images obtenues après une très forte compression. On observe sur la figure 16 que, si certaines images reconstruitent peremttent clairement de reconnaître le chiffre original, d'autres sont assez floues et la modification de leur forme provoque une confusion entre différents chiffres (par exemple entre le 4 et le 9 dans la première image).

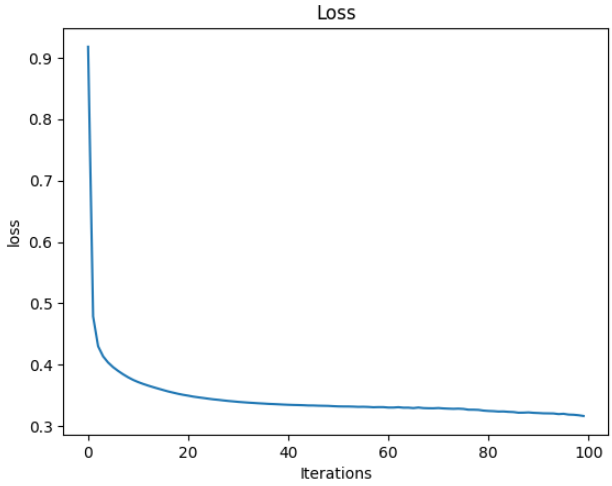
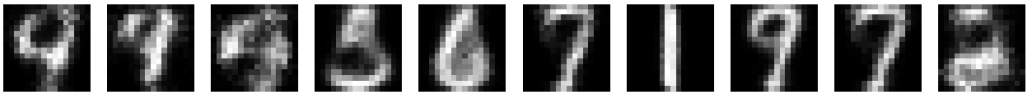


FIGURE 15 – Courbe de la loss (auto-encodeur).

(a) Images USPS avant encodage.



(b) Images USPS reconstruites.

FIGURE 16 – Images USPS avant et après encodage-décodage.

b. Classification des images reconstruites

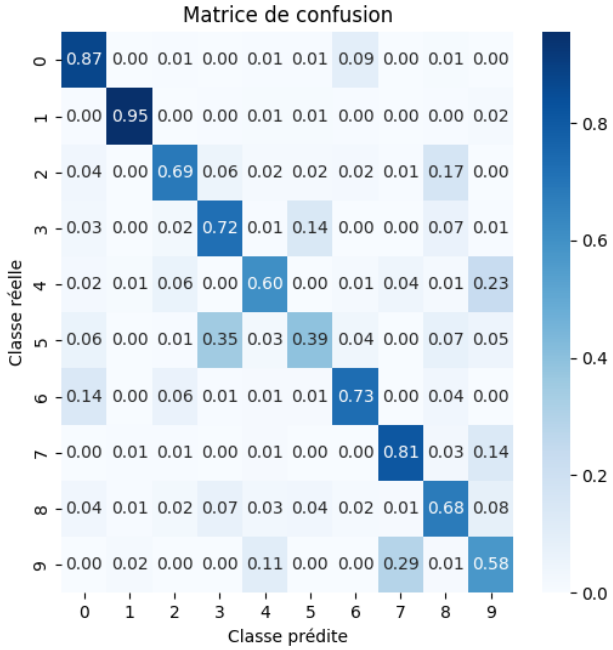


FIGURE 17 – Matrice de confusion des performances de classification sur images reconstruites.

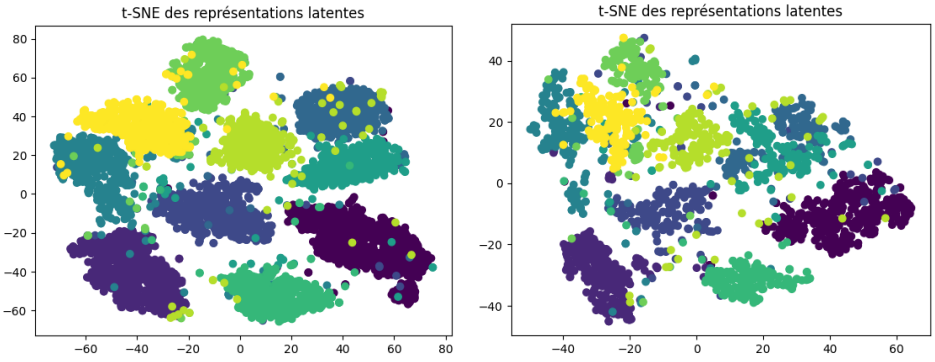
Le score de bonne classification obtenu est de 72,94%.

On note que certains chiffres sont mieux reconstruits que d'autres et donc mieux reconnus. Par exemple, le chiffre 2 est correctement prédit dans 95% des cas tandis que le chiffre 5 ne l'est que dans 39% des cas car souvent confondu avec un 3 (dans 35% des cas) probablement du fait de leur similarité de structure. C'est également le cas des chiffres 9 et 7 ou encore 4 et 9. Ce dernier cas est d'ailleurs illustré dans la figure 16 où le premier chiffre est un 4 mais ressort comme un 9 après encodage et décodage.

c. Visualisation des représentations obtenues dans un espace 2D

T-SNE (t-Distributed Stochastic Neighbor Embedding) est un algorithme de réduction de dimensionnalité non linéaire utilisé pour visualiser des données à haute dimension. Son principe est de représenté des données dans un espace de faible dimension en conservant les distances entre les points. Cela permet donc de visualiser les données : les données initialement similaires

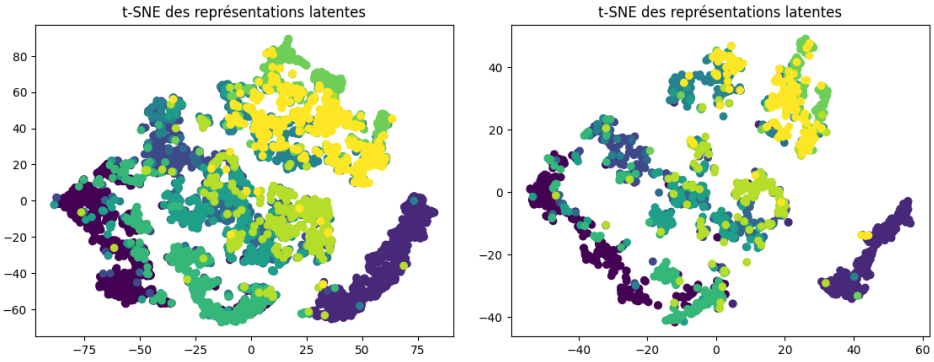
seront proches dans le nouvel espace et inversement les données différentes seront éloignées.



(a) T-SNE sur les données d'entraînement initiales.

(b) T-SNE sur les données de test initiales.

FIGURE 18 – T-SNE sur les données initiales.



(a) T-SNE sur les données d'entraînement après encodage.

(b) T-SNE sur les données de test après encodage.

FIGURE 19 – T-SNE en 2 dimensions sur les images après encodage.

Les résultats obtenus avec le T-SNE confirment les résultats précédents, à savoir que la compression des données fait perdre de l'information discriminante pour la tâche de classification. En effet, alors que les 10 classes (correspondantes aux 10 chiffres de 0 à 9) sont bien distinguées sur les données initiales [figure 18], on note que certaines classes se superposent après encodage [figure 19].

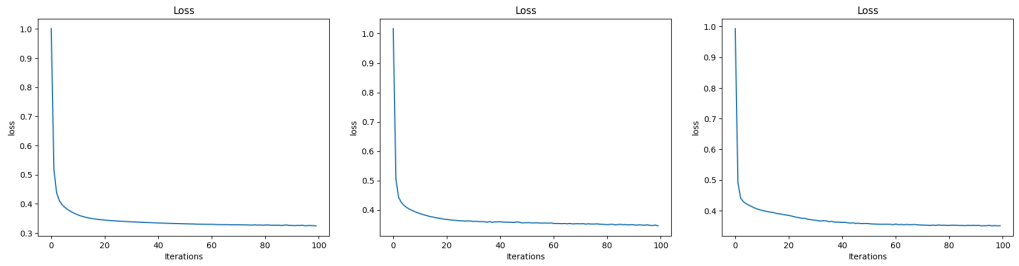
d. Performances en débruitage

Nous avons étudié l'effet de trois types de bruit sur le débruitages des images ainsi que le taux de bonne classification des images reconstruites :

— bruit gaussien (0.75)
— bruit de poisson (0.5)
— bruit sel et poivre (0.2)

Pour chaque type de bruit nous avons étudié les capacités de classification non seulement dans le cas où l'apprentissage avait été effectué sur les données brutes, mais aussi dans le cas où celui-ci avait été effectué sur les données bruitées.

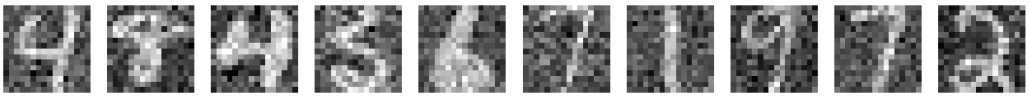
On obtient alors les courbes loss suivantes pour les apprentissages sur données bruitées :



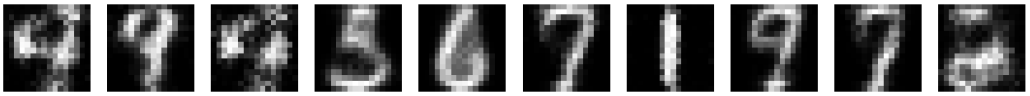
(a) Bruit gaussien. (b) Bruit de poisson. (c) Bruit sel et poivre.

FIGURE 20 – Courbe de la loss pour l'apprentissage sur des données bruitées.

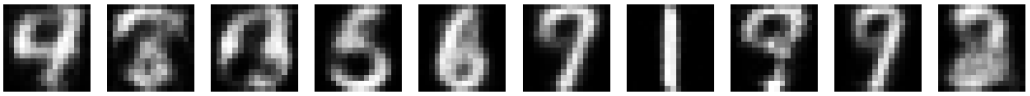
Bruit gaussien (0.75) :



(a) Images bruitées avec un bruit gaussien.

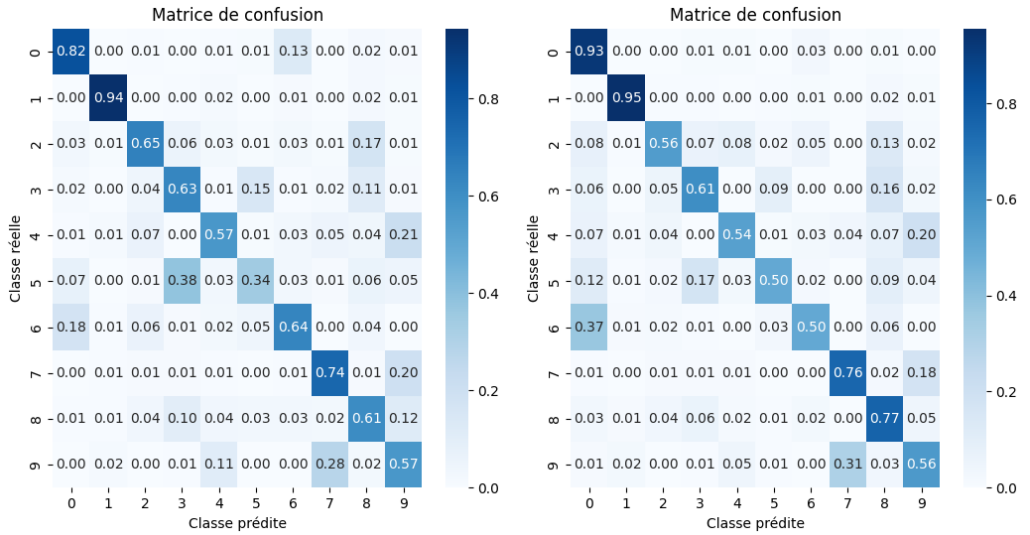


(b) Images reconstruite sans apprentissage sur images bruitées.



(c) Images reconstruite avec apprentissage sur images bruitées.

FIGURE 21 – Image USPS avec un bruit gaussien (0.75).



(a) Matrice de confusion sans apprentissage sur images bruitées.

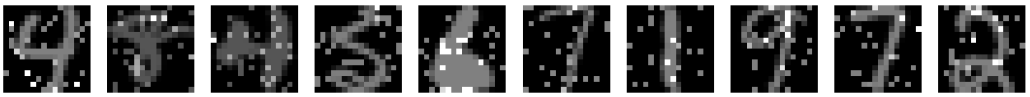
(b) Matrice de confusion avec apprentissage sur images bruitées.

FIGURE 22 – Matrice de confusion sur des données avec un bruit gaussien (0.75).

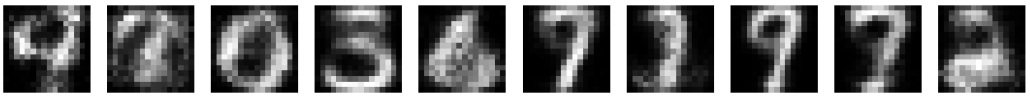
Visuellement, les images reconstruites sont plus nettes [figure 21], en par-

ticulier lorsque l'encodage et le décodage ont été appris sur des images bruitées. Néanmoins la classification n'est pas forcément meilleure. Par exemple, la troisième image de la ficgure 21, qui était clairement un 4 initialement, est difficilement reconnaissable après compression et reconstruction.
On note ici un taux de bonnes classifications de 76,38% sur les données bruitées. Lorsque l'on effectue un encodage puis décodage on obtient alors un taux de bonnes classifications de 68,06% si l'on n'apprend pas à encoder et décoder sur des données bruitées, et de 70,05% dans le cas contraire.

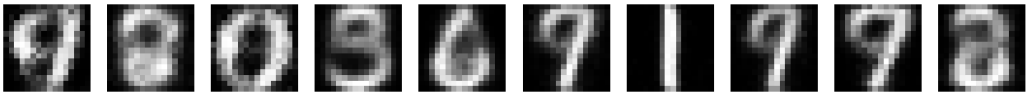
Bruit de poisson (0.5) :



(a) Images bruitées avec un bruit de poisson.

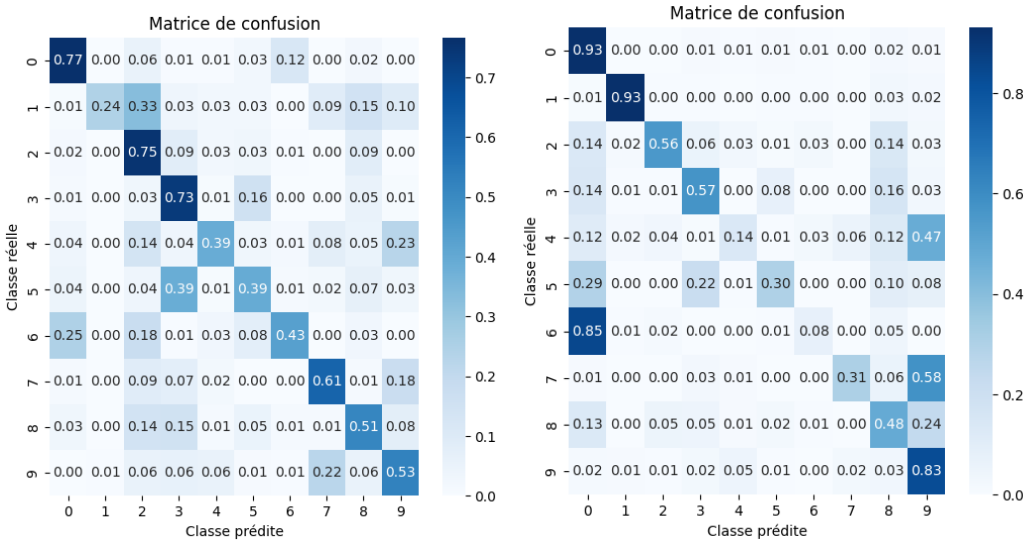


(b) Images reconstruite sans apprentissage sur images bruitées.



(c) Images reconstruite avec apprentissage sur images bruitées.

FIGURE 23 – Image USPS avec un bruit de poisson (0.5).



(a) Matrice de confusion sans apprentissage sur images bruitées.

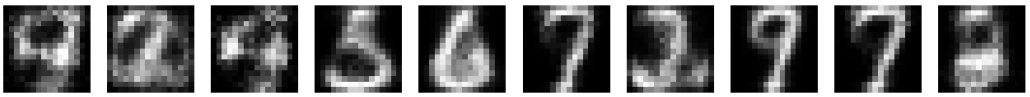
(b) Matrice de confusion avec apprentissage sur images bruitées.

FIGURE 24 – Matrice de confusion sur des données avec un bruit de poisson (0.5).

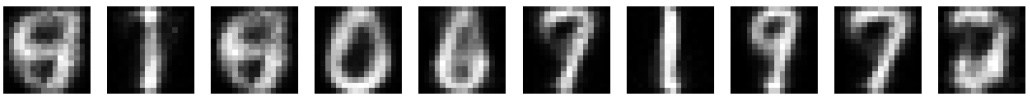
Cette fois encore les images reconstruites ressemblent plus aux données originales [figure 23], en particulier lorsque l'encodage et le décodage ont été appris sur des images bruitées. En effet, cela a permis déviter certaines erreurs, par exemple, la septième image de la figure 23 est correctement reconstruite comme un 1 lorsque l'apprentissage est fait sur des images bruitées alors que dans le cas contraire l'image ressort comme une sorte de 3. Néanmoins, les images ainsi obtenues sont très floues ce qui rend certaines images difficiles à déchiffrer.
Le taux de bonnes classifications est alors de 66,12% sur les données bruitées. Lorsque l'on effectue un encodage puis décodage on obtient un taux de bonnes classifications de 54,41% si l'on n'apprend pas à encoder et décoder sur des données bruitées. Ce chiffre augmente pour atteindre 57% dans le cas où l'apprentissage est réalisé sur des données bruitées.

Bruit sel et poivre (0.2) :

(a) Images bruitées avec un bruit sel et poivre.

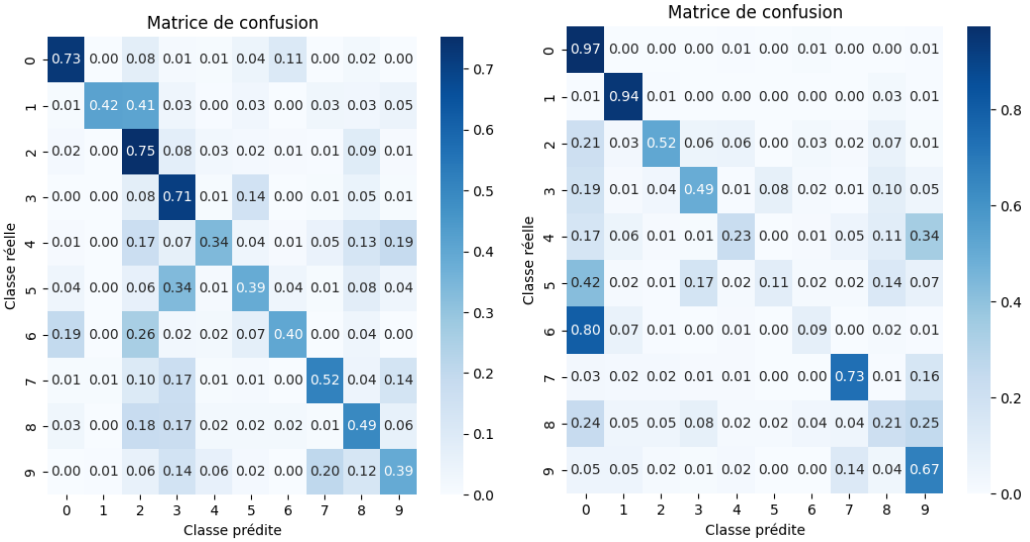


(b) Images reconstruite sans apprentissage sur images bruitées.



(c) Images reconstruite avec apprentissage sur images bruitées.

FIGURE 25 – Image USPS avec un bruit sel et poivre (0.2).



(a) Matrice de confusion sans apprentissage sur images bruitées.

(b) Matrice de confusion avec apprentissage sur images bruitées.

FIGURE 26 – Matrice de confusion sur des données avec un bruit sel et poivre (0.2).

Dans le cas de l'ajout d'un bruit sel et poivre les images deviennent

difficiles à déchiffrer. A l'instar du bruit de poisson l'apprentissage sur des données bruitées permet de corriger certaines erreurs notamment pour la prédiction du 1 (figure 25).
Cette fois 65,67% des chiffres sont correctement identifiés à partir des images bruitées. Ce chiffre tombe à 52,96% dans le cas d'images reconstruites avec encodeur ayant été entraîné sur des images non bruitées. Le taux de bonnes classification augmente à 55,95% lorsque l'entraînement a été effectué sur des images bruitées.

2. Récapitulatif des résultats obtenus en classification

		Apprentissage	**Taux de bonne classification**
Sans bruit	images reconstruites	-	72,94%
Bruit gaussien	images avant encodage	-	76,38%
	images reconstruites	sans bruit	68,06%
		avec bruit	70,05%
Bruit de poisson	images avant encodage	-	66,12%
	images reconstruites	sans bruit	54,41%
		avec bruit	57%
Bruit sel et poivre	images avant encodage	-	65,67%
	images reconstruites	sans bruit	52,95%
		avec bruit	55,95%

VI. Mon sixième se convole et mon tout s'améliore

La couche de convolution extrait des caractéristiques d'une image en appliquant des filtres et génère des feature maps qui sont utilisées comme entrée pour les couches suivantes du réseau de neurones convolutif. Nous avons implémenté les modules suivant :

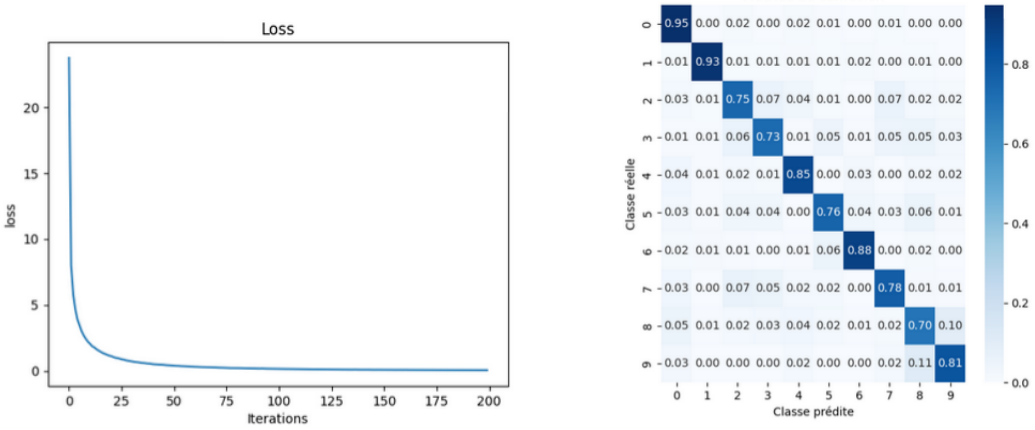
- Conv1D : applique des filtres pour extraire des caractéristiques de signaux 1D.
- MaxPool1D : réduit la taille de ces caractéristiques en préservant les valeurs les plus importantes.
- AvgPool1D : réduit également la taille des caractéristiques en prenant la moyenne des valeurs.

- Flatten : transforme les caractéristiques en un vecteur unidimensionnel.
- ReLU : fonction d'activation ajoutant de la non-linéarité aux sorties des couches.

1. Expérimentations

a. Max Pooling

Nous avons conçu un réseau de neurones de type Conv1D(3,1,32) → MaxPool1D(2,2) → Flatten() → Linear(4064,100) → ReLU() → Linear(100,10). Nous l'avons testé sur des données de chiffres manuscrits (USPS).



(a) Loss (MaxPool)1D.

(b) Matrice de confusion (MaxPool)1D.

FIGURE 27 – Loss et matrice de confusion pour un réseau de neurones convolutifs avec MaxPool1D.

Nous obtenons un taux de bonnes classifications de 83.36 %.

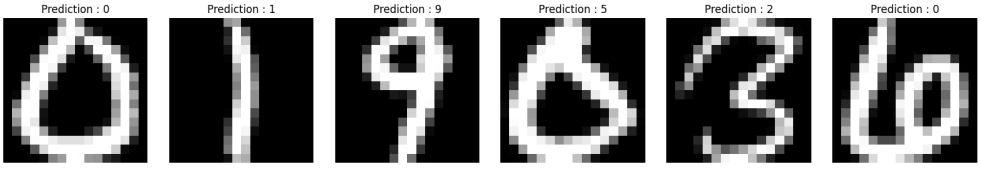
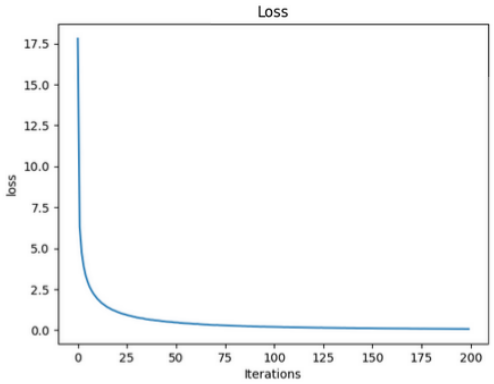


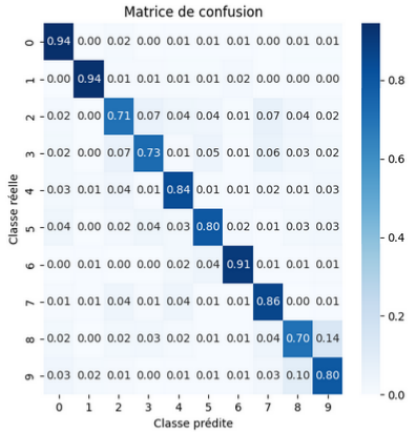
FIGURE 28 – Prédictions, correctes ou non, sur différents chiffres obtenus avec un réseau de neurones convolutif utilisant MaxPool1D. Les prédictions effectuées sont dans l'ordre : 0, 1, 9, 2, 2, 4.

b. Average Pooling

De même avons conçu un réseau de neurones de type Conv1D(3,1,32) → AvgPool1D(2,2) → Flatten() → Linear(4064,100) → ReLU() → Linear(100,10).



(a) Loss (AvgPool1D).



(b) Matrice de confusion (AvgPool1D).

FIGURE 29 – Loss et matrice de confusion pour un réseau de neurones convolutifs avec AvgPool1D.

Nous obtenons un taux de bonnes classifications très légèrement supérieur : 83.81 %.

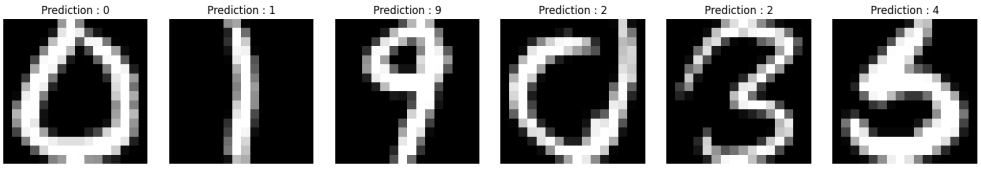


FIGURE 30 – Prédictions (correctes ou non) sur différents chiffres obtenus avec un réseau de neurones convolutif utilisant AvgPool1D.

2. Récapitulatif des résultats

	Taux de bonnes classifications
Maximum Pooling	83,36%
Average Pooling	83,81%

Table des figures

1 Regression linéaire sur des données faiblement bruitées. 5
2 Régression linéaire sur des données bruitées (6). 6
3 Classification dans le cas linéaire et sans bruits dans les données. 6
4 Classification linéaire de données bruitées. 7
5 Classification de données non linéaires sans bruit. 8
6 Classification pour un réseau à 2 neurones. 9
7 Classification pour un réseau à 8 neurones. 9
8 Classification pour un réseau à 16 neurones. 10
9 Classification dans le cas d'une descente de gradient en batch. 11
10 Classification dans le cas d'une descente de gradient stochastique. 12
11 Classification dans le cas d'une descente de gradient avec mini-batch de taille 100. 12
12 Classification dans le cadre d'une descente de gradient avec mini-batch de taille 10. 13
13 Classification sur des chiffres manuscrits à partir d'un réseau multiclasse. 14
14 Prédiction, correctes ou non, sur des chiffres manuscrits. Les prédictions sont : 0, 1, 9, 9, 0, 3. 15
15 Courbe de la loss (auto-encodeur). 18
16 Images USPS avant et après encodage-décodage. 18
17 Matrice de confusion des performances de classification sur images reconstruites. 19
18 T-SNE sur les données initiales. 20
19 T-SNE en 2 dimensions sur les images après encodage. 20
20 Courbe de la loss pour l'apprentissage sur des données bruitées. 21
21 Image USPS avec un bruit gaussien (0.75). 22
22 Matrice de confusion sur des données avec un bruit gaussien (0.75). 22
23 Image USPS avec un bruit de poisson (0.5). 23
24 Matrice de confusion sur des données avec un bruit de poisson (0.5). 24
25 Image USPS avec un bruit sel et poivre (0.2). 25
26 Matrice de confusion sur des données avec un bruit sel et poivre (0.2). 25
27 Loss et matrice de confusion pour un réseau de neurones convolutifs avec MaxPool1D. 27

28 Prédictions, correctes ou non, sur différents chiffres obtenus avec un réseau de neurones convolutif utilisant MaxPool1D. Les prédictions effectuées sont dans l'ordre : 0, 1, 9, 2, 2, 4. . . 27
29 Loss et matrice de confusion pour un réseau de neurones convolutifs avec AvgPool1D. 28
30 Prédictions (correctes ou non) sur différents chiffres obtenus avec un réseau de neurones convolutif utilisant AvgPool1D. . . 28