Projet - Réseau de neurones : DIY

Candice MOYET, Manon LEFEVRE

Mai 2022

Table des matières

1 Introduction

Ce projet a pour but de créer une bilbliothèque en python permettant d'implémenter différents types de réseaux de neurones. Pour cela nous avons implémenté différents modules : un module linéaire, plusieurs fonctions d'activations (Tanh, sigmoïde, softmax) et ++++ L'encapsulage des différents modules sera automatisé. Dans ce rapport, nous présentons l'ensemble des expérimentations réalisées sur plusieurs réseaux de neurones : avec seulement un module linéaire, puis en ajoutant des fonctions d'activations, en multi classe et finalement avec un auto-encodeur.

2 Linéaire

Nous commençons par implémenter un module linéaire, nous testons ce réseau très simple avec une génération de données artificielles en deux dimensions à partir d'un mélange de deux gaussiennes avec un bruit de 0,5. Sur la figure 1 est représentée la frontière résultante de l'application de ce réseau de neurone à nos données. Nous pouvons observer que les données sont bien séparées linéairement. En effet, comme indiqué dans la table 1 nous obtenons de très bons scores d'accuracy en entraînement et en test. A chaque itération de la descente de gradient nous calculons le coût avec la fonction des moindres carrés, le coût décrémente correctement et assez rapidement comme nous l'observons sur la figure 2

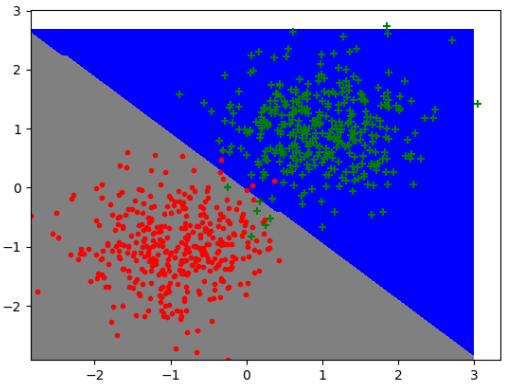


FIGURE 1 – Frontière de séparation des données avec un module linéaire

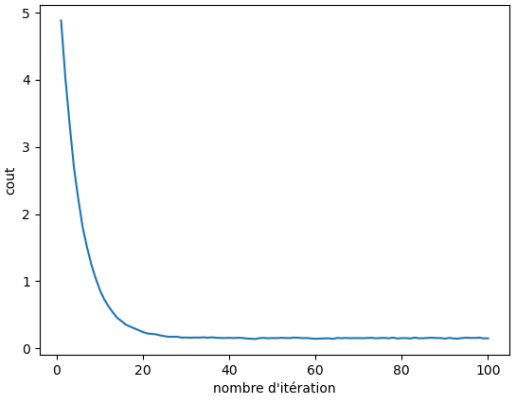


FIGURE 2 – Evolution du coût avec un module linéaire

Train accuracy	Test accuracy
0.991	0.990

TABLE 1 – Scores d'accuracy avec un module linéaire

3 Non Linéaire

Nous poursuivons par l'implémentation de trois fonctions d'activations : tanh, sigmoïde et ReLU. Nous les testons avec un réseau à quatre couches : un module linéaire à trois neurones activé par un tanh puis un second module linéaire suivie d'une sigmoïde.

3.1 Test sur un mélange de quatre gaussiennes

Comme précédemment nous générons des données artificielles, cette fois-ci à partir d'un mélange de quatre gaussiennes et avec un bruit de 0,5. Les frontières obtenues, représentées sur la figure 3 ne sont plus linéaires et les données sont correctement séparées. Le coût est également calculé avec les moindres carrés, sur la figure 4 nous observons la correcte diminution de ce coût. Les scores en entraînement et en test que nous obtenons sont dans la table 2

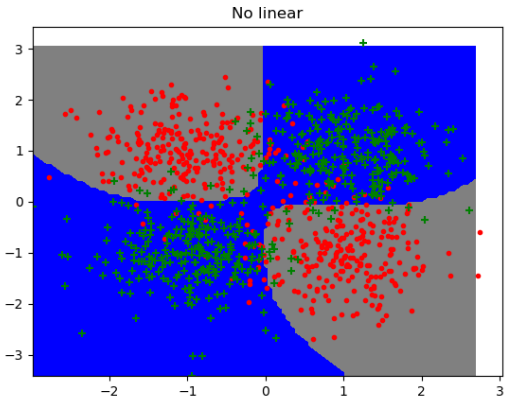


FIGURE 3 – Frontière de séparation des données avec deux modules linéaires et des fonctions d'activations

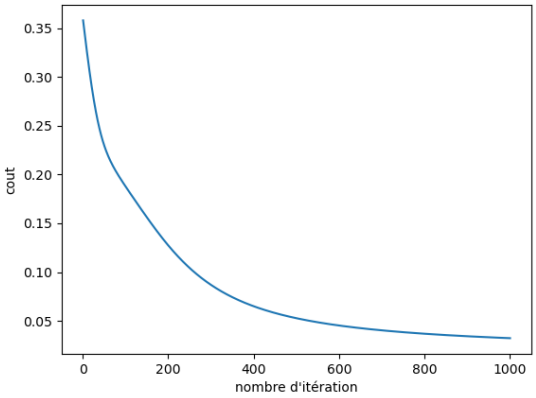


FIGURE 4 – Evolution du coût avec deux modules linéaires et des fonctions d'activations

Train accuracy	Test accuracy
0.985	0.926

TABLE 2 – Scores d'accuracy avec deux modules linéaires et des fonctions d'activations

3.2 Tests sur un échiquier

Nous avons également tenté d'entraîner un réseau pour classifier des données en échiquier à 8 cases. Pour cela, nous avons trois modules linéaires à 128 neurones activés respectivement par une fonction tanH, une fonction sigmoïde et à nouveau une fonction sigmoïde. Les résultats que nous obtenons sont évidemment un peu moins que précédemment (voir fig. 3). En effet, nous observons que le coût décroît moins vite que précédemment (voir fig. 6). La visualisation de la séparation des données représentée sur la figure 5 est assez satisfaisante.

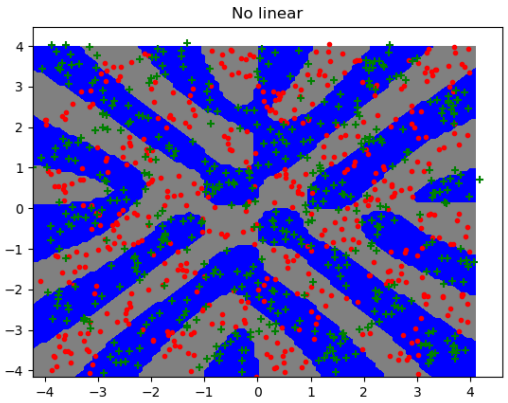


FIGURE 5 – Frontière de séparation des données en échiquier

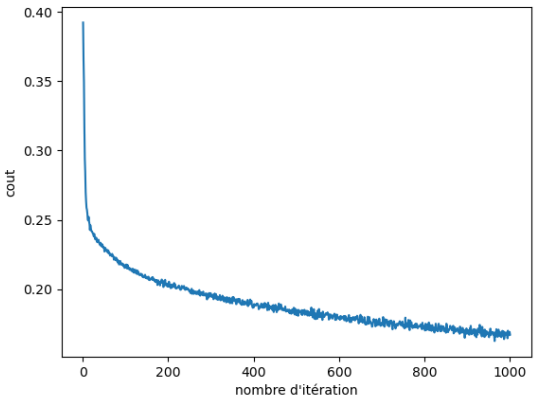


FIGURE 6 – Evolution du coût pour les données en échiquier

Train accuracy	Test accuracy
0.78	0.77

TABLE 3 – Scores d'accuracy pour les données en échiquier

4 Multi classes

Dans cette troisième partie, nous nous intéressons au multi classe. Pour cela nous implémentons la fonction de coût softmax et nous servons d'un one-hot encodeur. Pour tester nos réseaux de neurones sur du multi-classe, nous utilisons les données USPS. Les résultats que nous présentons ont été obtenus par un réseau avec deux modules linéaire chacun activé avec une fonction tanh. Sur la figure 7 est représentée la matrice de confusion calculée à l'issue de ces tests. La diagonale se dessinant bien nous confirme les bons scores d'accuracy que nous obtenons présentés dans la table **??**. Nous ressortons également quelques prédictions aléatoirement pour les comparer à la vérité terrain. Il reste évidemment des erreurs dans la classification, par exemple sur la figure 8c, la prédiction est '0' alors que l'image représente un '5'. Cependant, le cinq est assez arrondi nous comprenons donc pourquoi la classification a échouée.

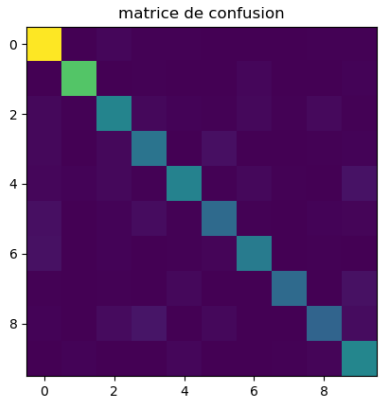
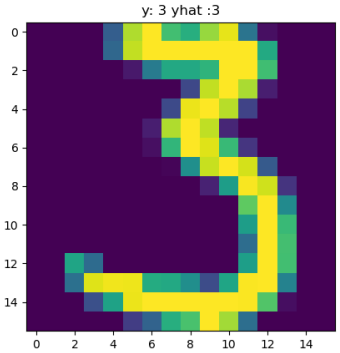


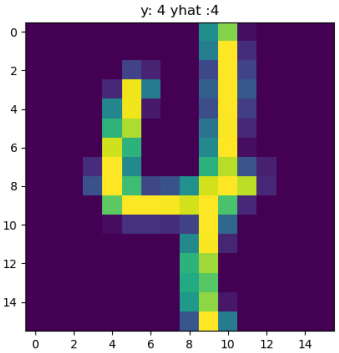
FIGURE 7 – Matrice de confusion de la classification multi-classes

Train accuracy	Test accuracy
0.909	0.838

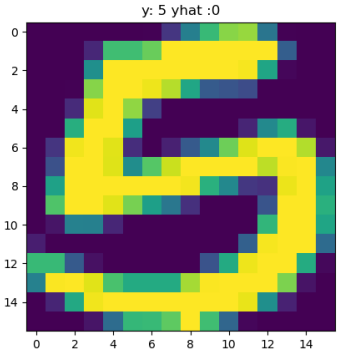
TABLE 4 – Scores d'accuracy de la classification multi-classes



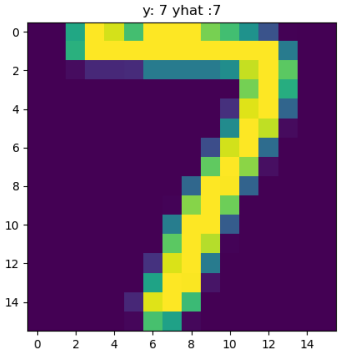
(a) 3 bien classé



(b) 4 bien classé



(c) 5 mal classé en 0



(d) 7 bien classé

FIGURE 8 – Prédictions par le réseau multiclasse comparées à la vérité terrain

5 Auto-encodeurs

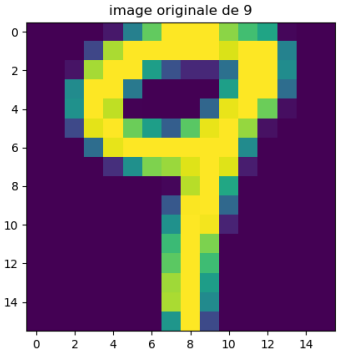
Pour finir nous avons implémenté des auto-encodeurs, dans la suite nous présentons nos expérimentations de compression et reconstruction d'images ainsi que de débruitage faites sur les données USPS.

5.1 Compression

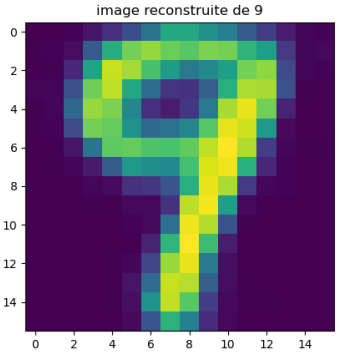
Nous commençons par la compression, dans cette partie, le coût est calculé à chaque époque avec la fonction BCE.

5.1.1 Vers un espace latent de dimension 5

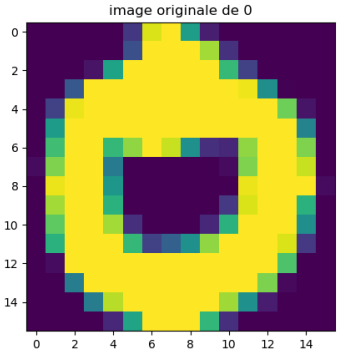
Nous avons réalisé plusieurs expériences différentes ; tout d'abord avec une compression d'une dimensions 50 vers un espace latent de dimension 5. Sur la figure 10, nous pouvons observer les images originales des chiffres de USPS ainsi que celles reconstruites après compression. La reconstruction est généralement satisfaisante, dans le cas du '8' nous observons que l'image reconstruite ressemble plus à un '3', cela est compréhensible puisque les formes d'un '8' et d'un '3' sont assez proches.



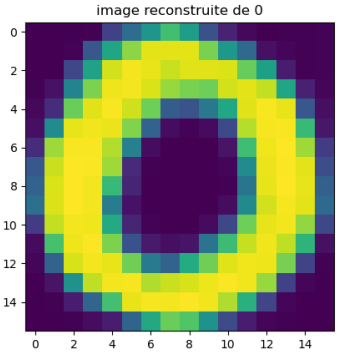
(a) Image originale de 9



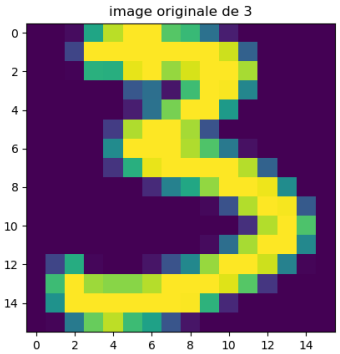
(b) Image reconstruite de 9



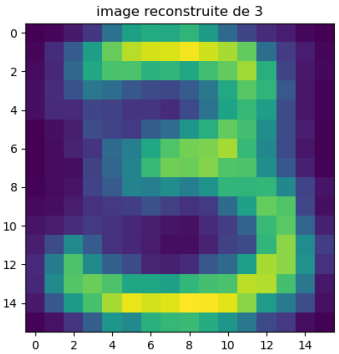
(c) Image originale de 0



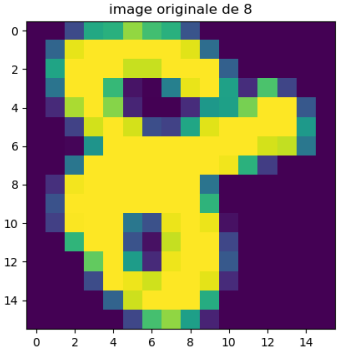
(d) Image reconstruite de 0



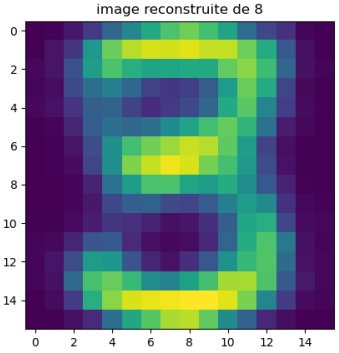
(e) Image originale de 3



(f) Image reconstruite de 3



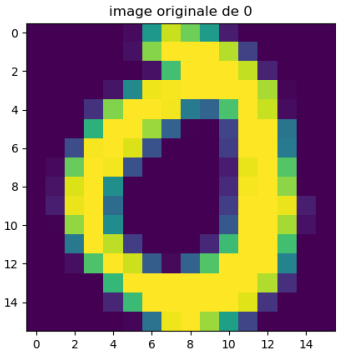
(a) Image originale de 8



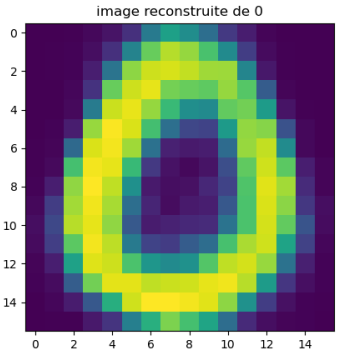
(b) Image reconstruite de 8

FIGURE 10 – Images originales et reconstruites après compression d'une dimension 50 vers un espace latent de dimension 5

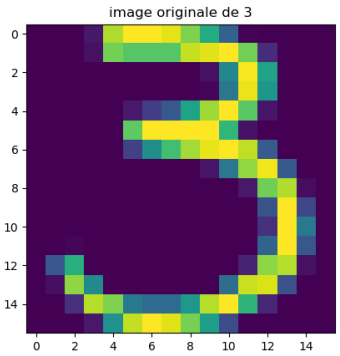
La seconde expérimentation que nous avons menée a été faite depuis une dimension 100 toujours compressé sur un espace latent de dimension 5 (voir fig. 12). Encore une fois, les images reconstruites sont assez satisfaisantes mais pas parfaite, le '2' ici n'est pas bien reconstruit. En revanche, nous ne voyons pas de différence notable entre la reconstruction après une compression depuis une dimension 100 vers un espace latent de dimension 5 et depuis 50 vers 5. Nous voulons donc maintenant tester avec une différente dimension de compression.



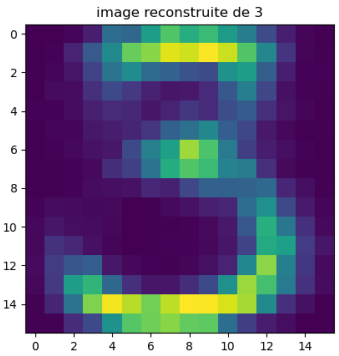
(a) Image originale de 0



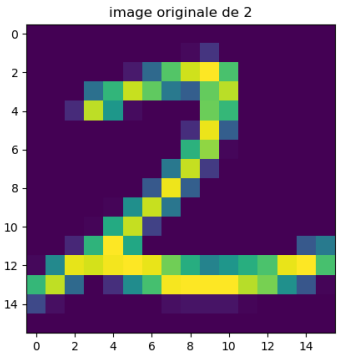
(b) Image reconstruite de 0



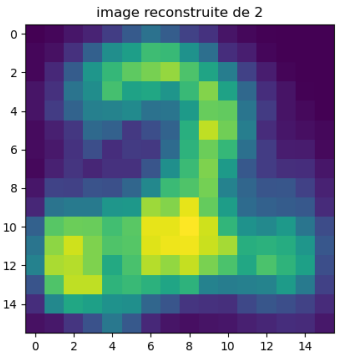
(c) Image originale de 3



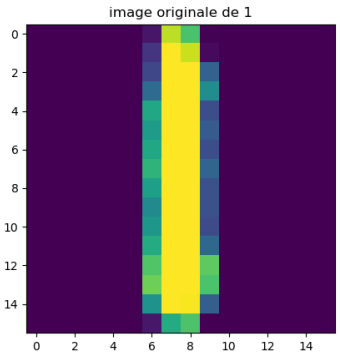
(d) Image reconstruite de 3



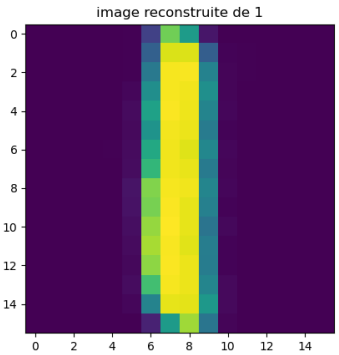
(a) Image originale de 2



(b) Image reconstruite de 2



(c) Image originale de 1

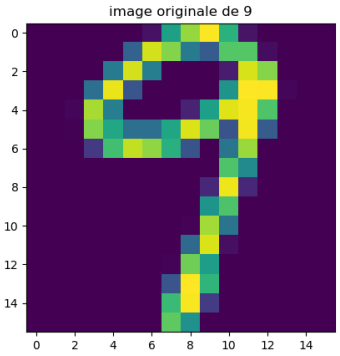


(d) Image reconstruite de 1

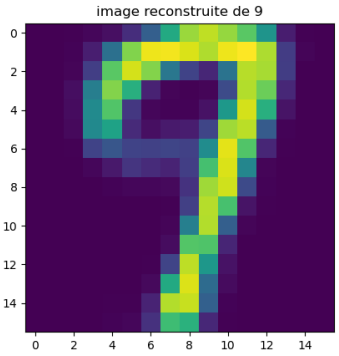
FIGURE 12 – Images originales et reconstruites après compression d'une dimension 100 vers un espace latent de dimension 5

5.1.2 Vers un espace latent de dimension 10

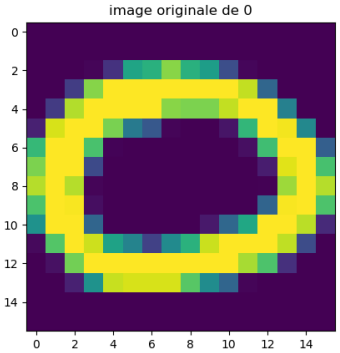
Nous passons maintenant à la compression sur un espace latent de dimension 10 avec, d'une part, une dimension de départ de 50 et d'autre part, de 100 (voir fig. 14 et 16). La reconstruction n'est toujours pas parfaite, par exemple sur la figure 20b, le '9' est mal reconstruit. De plus, nous ne remarquons pas de différence significative dans la qualité de la reconstruction quand on passe vers une dimension 5 ou une dimension 10.



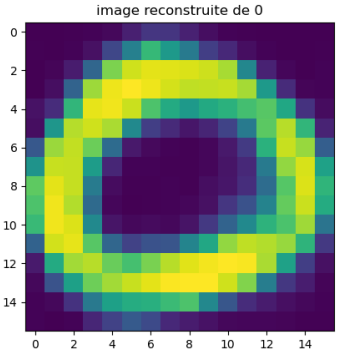
(a) Image originale de 9



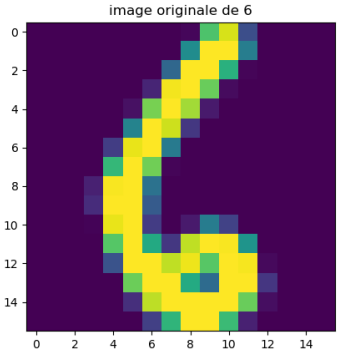
(b) Image reconstruite de 9



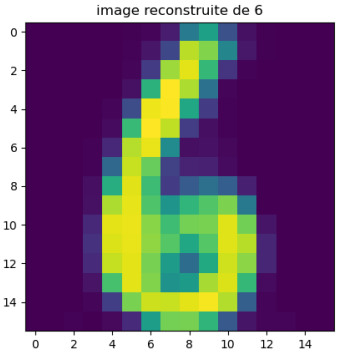
(a) Image originale de 0



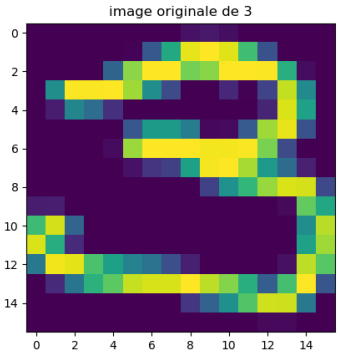
(b) Image reconstruite de 0



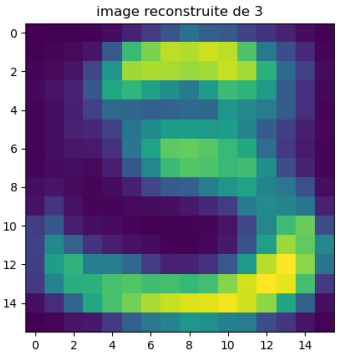
(c) Image originale de 6



(d) Image reconstruite de 6

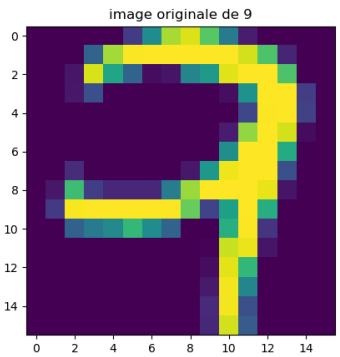


(e) Image originale de 3

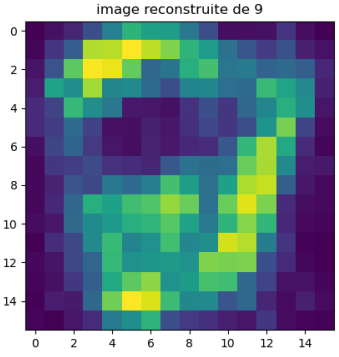


(f) Image reconstruite de 3

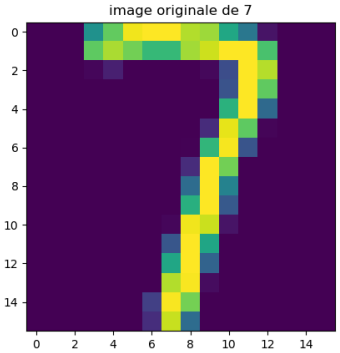
FIGURE 14 – Images originales et reconstruites après compression d'une dimension 50 vers un espace latent de dimension 10



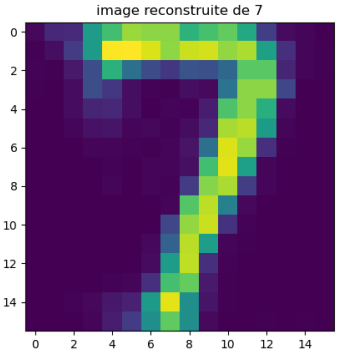
(a) Image originale de 9



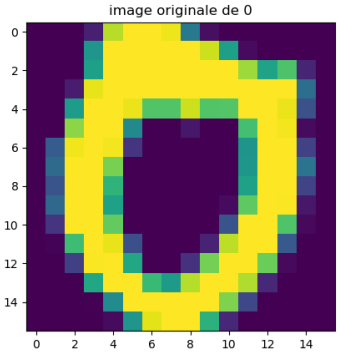
(b) Image reconstruite de 9



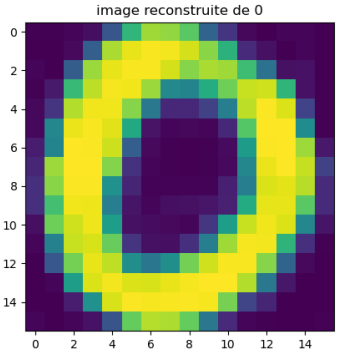
(a) Image originale de 7



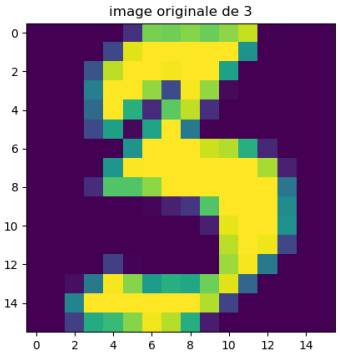
(b) Image reconstruite de 7



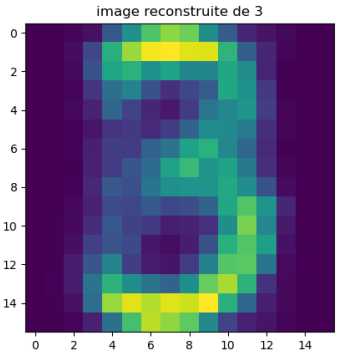
(c) Image originale de 0



(d) Image reconstruite de 0



(e) Image originale de 3

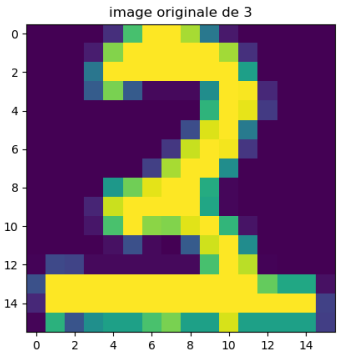


(f) Image reconstruite de 3

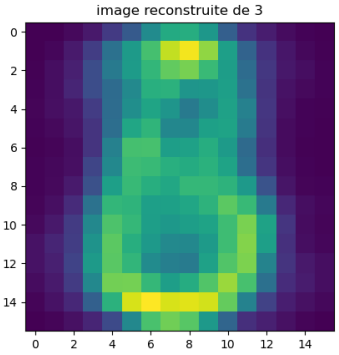
FIGURE 16 – Images originales et reconstruites après compression d'une dimension 100 vers un espace latent de dimension 10

5.1.3 Vers un espace latent de dimension 2

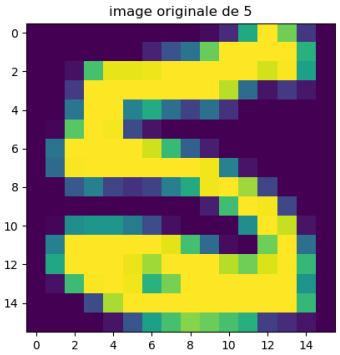
Pour finir avec la compression, nous testons avec une forte compression c'est à dire nous passons d'une dimension 50 vers un espace latent de dimension 2 (voir fig. 18). La reconstruction marche beaucoup moins bien avec une forte compression.



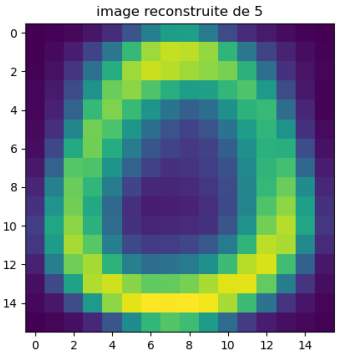
(a) Image originale de 3



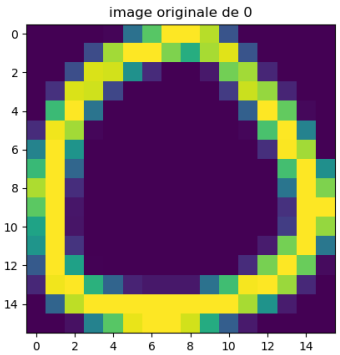
(b) Image reconstruite de 3



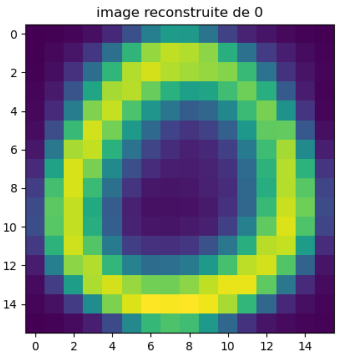
(a) Image originale de 5



(b) Image reconstruite de 5



(c) Image originale de 0

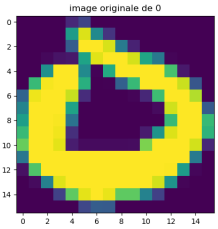


(d) Image reconstruite de 0

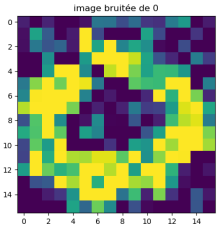
FIGURE 18 – Images originales et reconstruites après forte compression

5.2 Débruitage

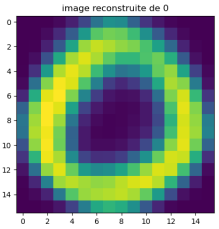
Nous continuons ces expérimentations avec les auto-encodeurs en testant le débruitage. En premier lieu, nous bruitons les données USPS puis nous appliquons notre réseau sur ces images pour les débruiter. Nous commençons avec un bruit de 0.3 et un espace latent de dimension 5 (voir fig 19). Le débruitage fonctionne relativement correctement ; le '4' ressemble plus à un '9' une fois débruité, ce la peut s'expliquer par la forme de ce '4' proche de celle d'un '9'.



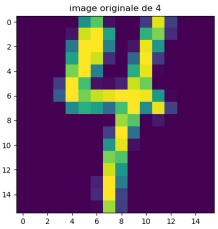
(a) Image originale de 0



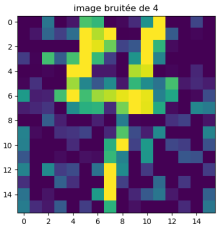
(b) Image bruitée de 0



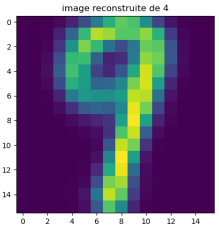
(c) Image débruitée de 0



(d) Image originale de 4



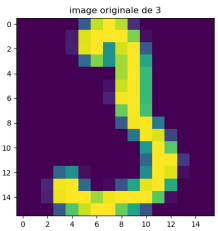
(e) Image bruitée de 4



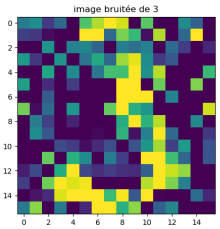
(f) Image débruitée de 4

FIGURE 19 – Images originales, bruitées puis débruitées

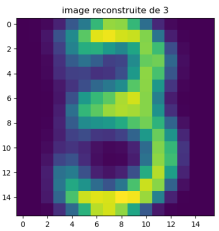
Nous essayons cette fois-ci avec un espace latent plus grand (dimension 10) et plus de bruit (0.5) (voir fig. 20). Nous n'observons pas de différences de performance significatives avec les test précédent. Il est intéressant de noter que le '3' débruité semble corrigé, et correspondre plus à une moyenne des '3'.



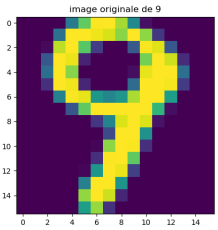
(a) Image originale de 3



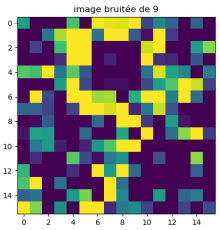
(b) Image bruitée de 3



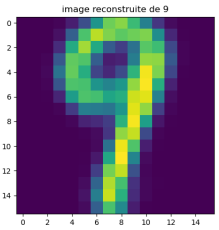
(c) Image débruitée de 3



(d) Image originale de 9



(e) Image bruitée de 9



(f) Image débruitée de 9

FIGURE 20 – Images originales, bruitées puis débruitées

6 Tests supplémentaires

6.1 Classification avec des données imparfaites en entraînement

Pour finir cette série d'expérimentations sur les auto-encodeurs, nous nous intéressons à la capacité d'un réseau de neurones à classifier correctement des données si nous lui donnons en entraînement des données reconstruite après compression. Pour cela nous avons choisis de tester un réseau de neurones à deux couches linéaires chacune activée par une fonction tanH.

Nous avons compressé les données vers un espace latent de 5.

Dans la table 5 sont représentés les résultats du test de ce réseau de neurones avec les données présentées précédemment en entraînement et les données originales en test mis en parallèle avec

le même réseau de neurones entraîné sur les données USPS originales. Nous observons que lorsque nous entraînons le réseau sur les données compressées puis reconstruites, nous avons de bien moins bons scores, en effet nous observons que la diagonale de la matrice de confusion s'efface (voir fig. 21).

Données en train	Test accuracy
originales	0.838
décompressées	0.535

TABLE 5 – Scores d'accuracy pour les données imparfaites en entraînement

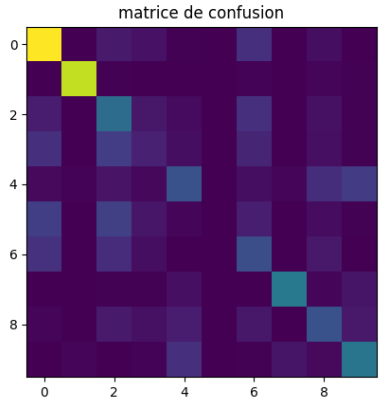


FIGURE 21 – Matrice de confusion avec l'entraînement sur les données décompressées

6.2 Classification avec des données imparfaites en test

Pour conclure ces expérimentations sur les réseaux de neurones, nous testons la classification sur des données reconstruites après compression et bruitées sur un réseau entraîné sur les données originales de USPS. Nous avons compressé les données vers un espace latent de 5. Nous avons bruitées les données avec 0.4. Pour ce test nous utilisons un réseau à deux modules linéaires chacun est activé par une fonction tanH. Nous observons que le test sur les données compressées puis reconstruites marche moins bien, cependant on distingue encore assez bien la diagonale sur la matrice de confusion (voir fig. 22). En revanche, le test sur les données bruitées la classification marche très mal, on le voit d'ailleurs sur la matrice de confusion qui est beaucoup plus homogène (voir fig. 23).

Données en test	Test accuracy
originales	0.838
décompressées	0.684
bruitées	0.481

TABLE 6 – Scores d'accuracy pour les données imparfaites en test

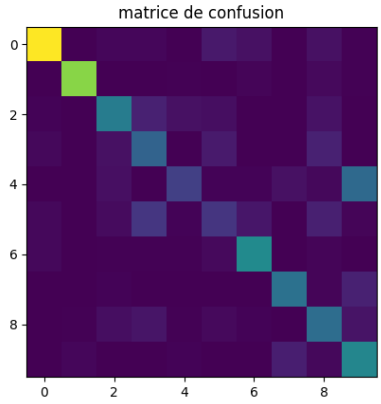


FIGURE 22 – Matrice de confusion pour le test sur les données décompressées

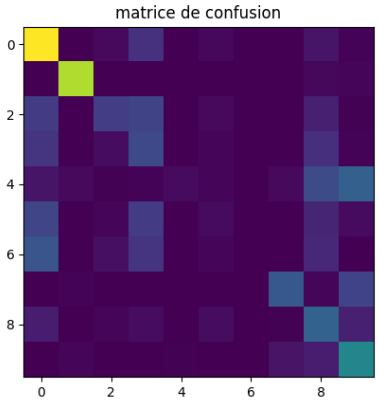


FIGURE 23 – Matrice de confusion pour le test sur les données bruitées