SORBONNE UNIVERSITE

Faculté des Sciences et Ingénerie

Master Informatique - M1

Parcours Données Apprentissage Connaissance DAC

MACHINE LEARNING

Projet - Réseau de neurones : DIY

Rapport

Auteur

Ben Kabongo Buzangu

Encadrants

Nicolas Thome
Nicolas Baskotis

Janvier 2023 - Mai 2023

1 Introduction

Dans ce rapport nous allons présenter notre travail (nos résultats et nos expérimentations) pour le projet **Réseaux de neurones**, sur lequel nous avons travaillé dans le cadre du cours de **Machine Learning**.

Des précieuses indications sur l'implémentation ayant déjà été fournies dans le sujet du projet. Nous allons donc très brièvement présenter la structure du code produit, puis aborder dans la suite les différentes expérimentations que nous avons menées afin de prouver que notre implémentation fonctionne.

2 Implémentation : structure du code

Les différentes briques de base ont été découpées en modules python suivants :

— **module** : contient la classe **Module**.

— **linear** : contient la classe **Linear** pour les modules linéaires des réseaux de neurones.

— **activation** : contient les classes des modules d'activation qui héritent de la classe mère **Activation** : on y retrouve les classes **TanH**, **ReLU**, **Sigmoid**, **LeakyReLU**, **ELU** et **Softmax**.

— **loss** : contient les classes des modules de coût qui héritent de la classe mère **Loss**. On y retrouve les classes pour la mean squared error **MSELoss**, la binary cross entropy loss **BCELoss**, la cross entropy **CELoss** et la hinge loss **HingeLoss**.

— **sequential** : contient la classe **Sequential**.

— **optimizers** : contient la classe **Optim** et la fonction **SGD**.

— **autoencoder** : contient la classe **AutoEncoder**.

— **convolution** : contient toutes les classes pour la convolution. Notamment la classe **Flatten**. Pour la convolution 1D les classes **Conv1D**, **MaxPool1D**, **AvgPool1D**, **ConvTranspose1D** et **DoubleConv1D** qui reconnait des motifs horizontaux et verticaux.

— D'autres modules rassemblent les constantes et des fonctions utilitaires.

L'intégralité des expérimentations effectuées se retrouvent dans les notebooks du dossier **tests**.

3 Formules utilisées

Les modules de réseaux de neurones appliquent différentes fonctions (linéaires, activations, coûts) lors de la passe forward et la dérivée de ces mêmes fonctions lors de la passe backward. Dans cette section, nous allons donner quelques formules et les dérivées correspondantes de ces formules.

3.1 Fonction linéaire

Soit W est la matrice de poids des perceptrons de la couche courante de même dimension que l'entrée x et b le vecteur des biais :

— La formule à la base des modules linéaires est $z = Wx + b$.

— Les dérivées de la sortie par rapport aux paramètres sont : $\frac{\partial z}{\partial W} = x$ $\frac{\partial z}{\partial b} = 1$.

3.2 Fonctions d'actiations

3.2.1 ReLU

$$reLU(x) = \max(0, x) \quad \frac{\partial reLU(x)}{\partial x} = \begin{cases} 0 & \text{si } x \leq 0 \\ 1 & \text{si } x > 0 \end{cases}$$

3.2.2 LeakyReLU

$$leakyReLU(x) = \begin{cases} x & \text{si } x \geq 0 \\ \alpha x & \text{si } x < 0 \end{cases} \quad \frac{\partial leakyReLU(x)}{\partial x} = \begin{cases} 1 & \text{si } x \geq 0 \\ \alpha & \text{si } x < 0 \end{cases}$$

3.2.3 ELU

$$elu(x) = \begin{cases} x & \text{si } x \geq 0 \\ \alpha(e^x - 1) & \text{si } x < 0 \end{cases} \quad \frac{\partial elu(x)}{\partial x} = \begin{cases} 1 & \text{si } x \geq 0 \\ elu(x) + \alpha & \text{si } x < 0 \end{cases}$$

3.2.4 Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \frac{\partial \sigma(x)}{\partial x} = \sigma(x) \cdot (1 - \sigma(x))$$

3.2.5 Softmax

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad \frac{\partial softmax(x_i)}{\partial x_j} = \begin{cases} softmax(x_i) \cdot (1 - softmax(x_i)) & \text{si } i = j \\ -softmax(x_i) \cdot softmax(x_j) & \text{si } i \neq j \end{cases}$$

3.2.6 Tangente hyperbolique

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad \frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x)$$

3.3 Fonctions de coûts

3.3.1 Binary cross entropy

$$BCE(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y_i}) + (1 - y_i) \log(1 - \hat{y_i})] \quad \frac{\partial BCE(y,\hat{y})}{\partial \hat{y_i}} = -\frac{y_i}{\hat{y_i}} + \frac{1-y_i}{1-\hat{y_i}}$$

3.3.2 Cross entropy

$$CE(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij}) \quad \frac{\partial CE(y,\hat{y})}{\partial \hat{y}_{ij}} = -\frac{y_{ij}}{\hat{y}_{ij}}$$

3.3.3 Hinge Loss

$$\text{Hinge Loss}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i\hat{y}_i) \quad \frac{\partial \text{Hinge Loss}(y,\hat{y})}{\partial \hat{y}_i} = \begin{cases} -y_i & \text{si } y_i\hat{y}_i < 1 \\ 0 & \text{sinon} \end{cases}$$

3.3.4 Mean squared error

$$\text{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad \frac{\partial \text{MSE}(y,\hat{y})}{\partial \hat{y}_i} = -2(y_i - \hat{y}_i)$$

4 Expérimentations

4.1 Régression linéaire

— **Régression linéaire** = module linéaire et MSE.

— Fichier de test : **tests/linear_regression.ipynb**

4.1.1 Données aléatoires

lr=**0.1**, epochs=**200**, loss=**MSE**

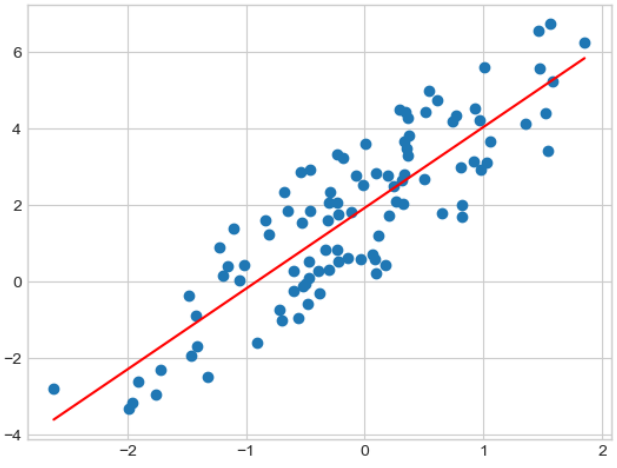

FIGURE 1 – Droite de régression donnée après convergence

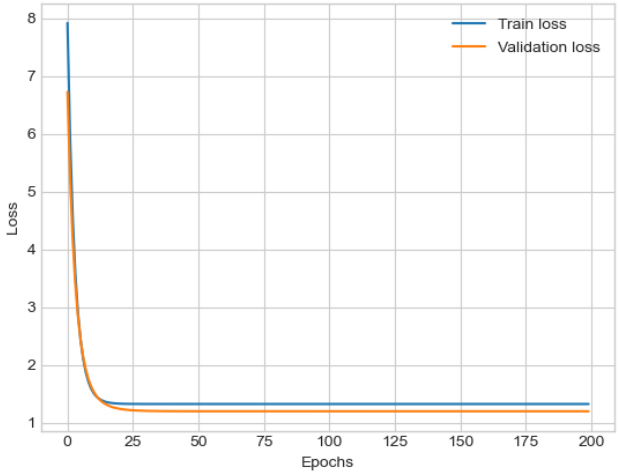

FIGURE 2 – Train & validation loss en fonction des époques

.

4.1.2 Boston et Fetch california housing

Boston data

— Informations sur la valeur des maisons dans les banlieues de Boston.

— Nombre d'exemples : **506**

— Nombre de caractéristiques : **13**

Fetch california housing

— Informations sur les logements dans différentes régions de Californie.

— Nombre d'exemples : **20 640**

— **8** caractéristiques : la latitude, la longitude, le nombre moyen de pièces par habitation, le revenu moyen des habitants de la région, etc.

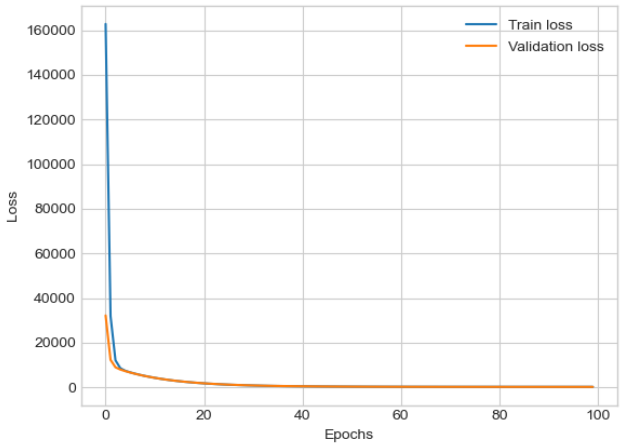

FIGURE 3 – Régression linéaire - **Boston Data** - lr=10^{-6}, epochs=**100**, loss=**MSE**.
En bleu la loss en apprentissage, en orange la loss en validation. Convergence rapide.

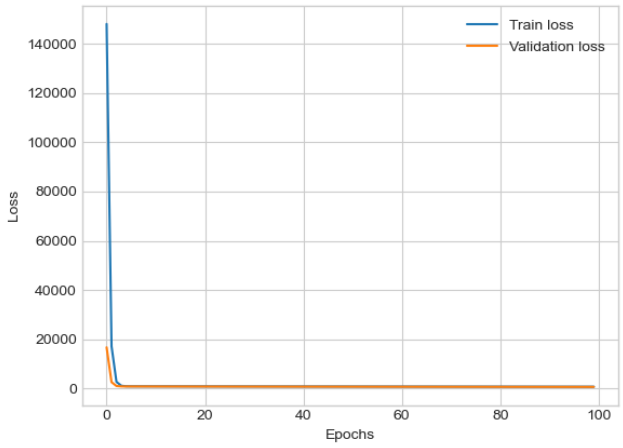

FIGURE 4 – Régression linéaire - **Fetch Data** - lr=10^{-7}, epochs=**100**, loss=**MSE**.
En bleu la loss en apprentissage, en orange la loss en validation. Convergence rapide.

4.2 Linéaire - Tangente Hyperbolique - Sigmoid

— Tests portant sur le module linéaire et les activations **TanH** et **Sigmoid**.

— Fichier de test :

tests/linear_tanh_sigmoid.ipynb

— **Datasets de tests :** données aléatoires.

lr=**0.1**, epochs=**200**, loss=**MSE**

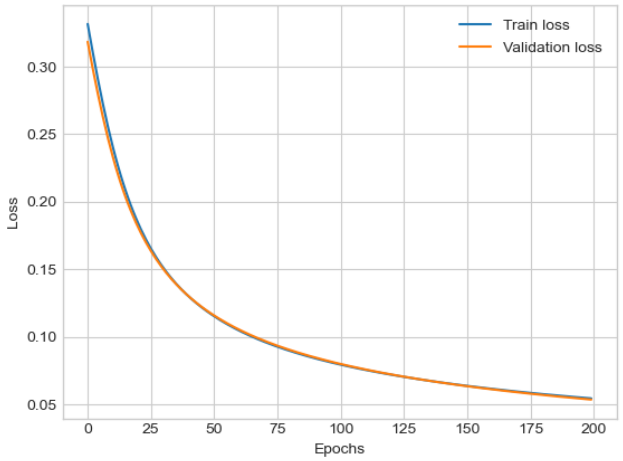

FIGURE 5 – Train & Validation loss

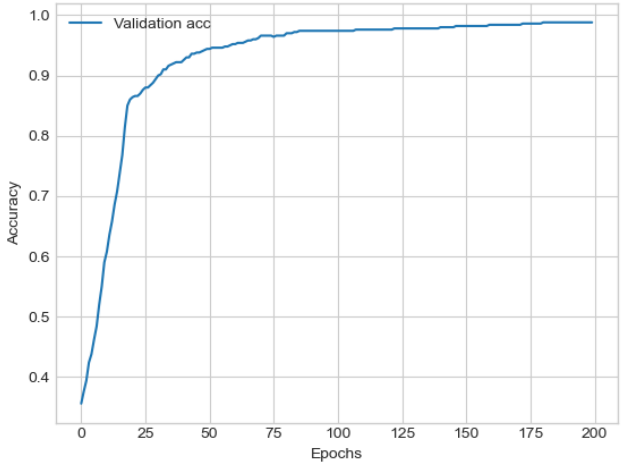

FIGURE 6 – Validation accuracy

4.3 Multi-classes (Iris & Digits)

— Multi-classes sur quelques datasets.

— Fichier de test : **tests/multiclass.ipynb**

4.3.1 Iris

— Nombre d exemples : **150**

— Nombre de features : **4**

— Nombre de classes : **3**

lr=**0.1**, epochs=**1000**, loss=**Cross Entropy Loss**

Linear(4, 10) - ReLU() - Linear(10, 3)

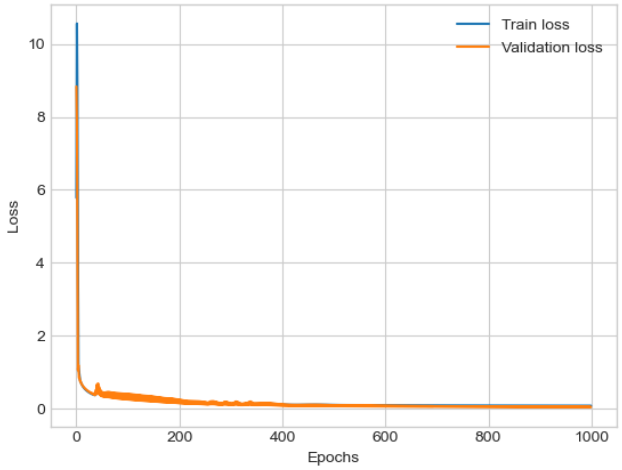

FIGURE 7 – IRIS - Train & Validation loss

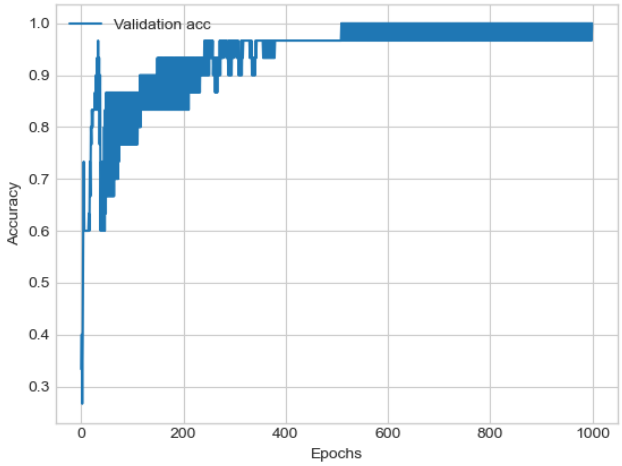

FIGURE 8 – IRIS - Validation accuracy

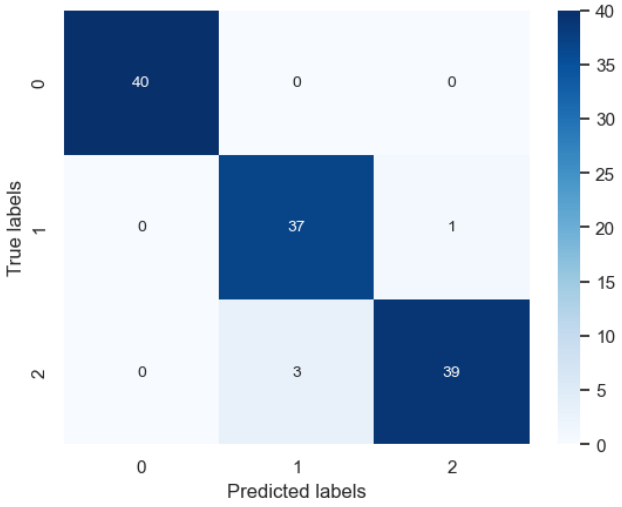

FIGURE 9 – IRIS - Train - Matrice de confusion

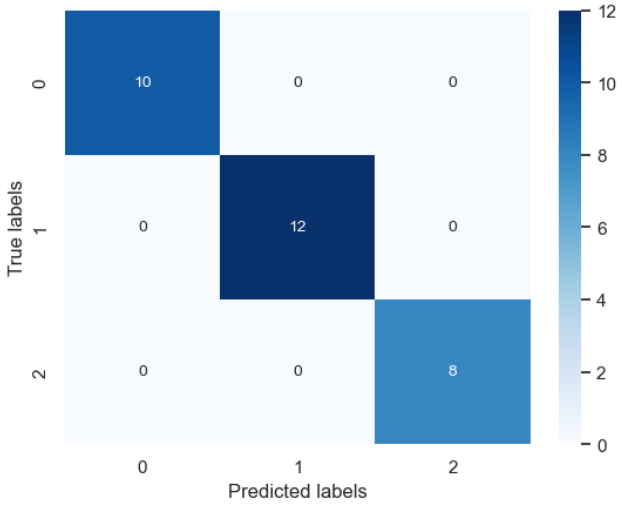

FIGURE 10 – IRIS - Test - Matrice de confusion

4.3.2 Digits

— Dataset d'images 8x8 de chiffres de 0 à 9.

— Nombre d exemples : **1797**

— Nombre de features : **64**

— Nombre de classes : **10**

lr=**0.1**, epochs=**1000**, loss=**Cross Entropy Loss**

Linear(64, 100) - ReLU() - Linear(100, 100) - ReLU() -Linear(100, 10)

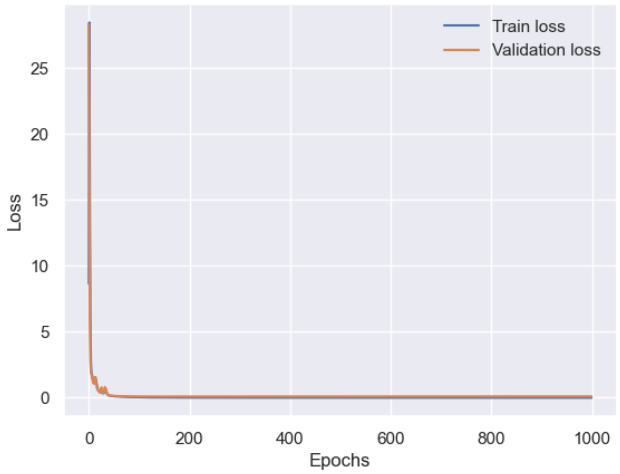

FIGURE 11 – DIGITS - Train & Validation loss

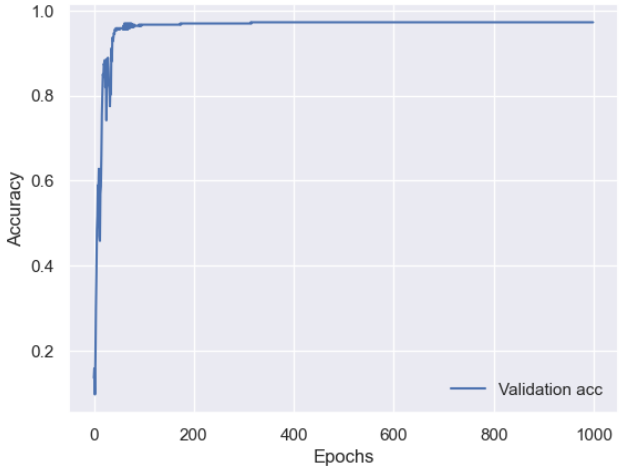

FIGURE 12 – DIGITS - Validation accuracy

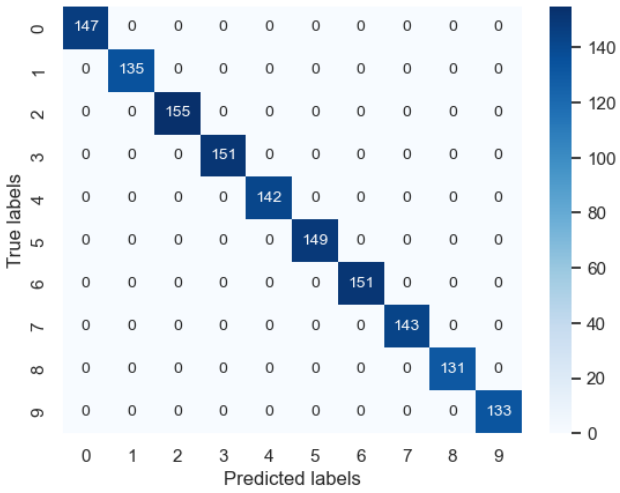

FIGURE 13 – DIGITS - Train - Matrice de confusion

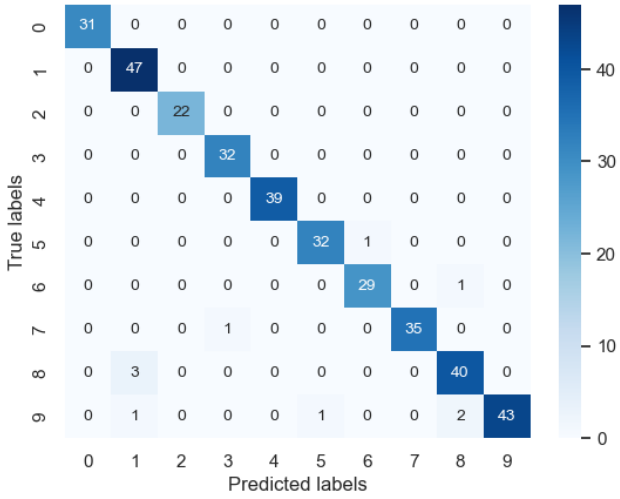

FIGURE 14 – DIGITS - Test - Matrice de confusion

4.4 Multi-classes MNIST

— Multi-classes sur le dataset MNIST.

— Fichier de test : **tests/multiclass_mnist.ipynb**

— Datasets de tests : **MNIST** : données d'images 28x28 de chiffres de 0 à 9.

4.4.1 Linear(784, 100) - TanH() - Linear(100, 100) - TanH() - Linear(100, 10) + Cross Entropy Loss

lr=**0.01**, epochs=**10000**

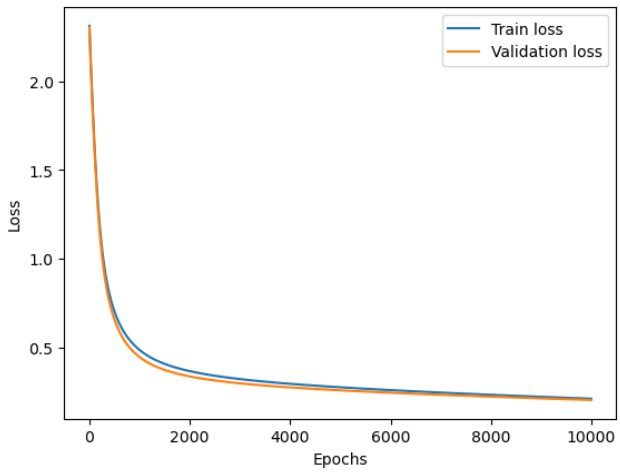

FIGURE 15 – MNIST - Train & validation loss

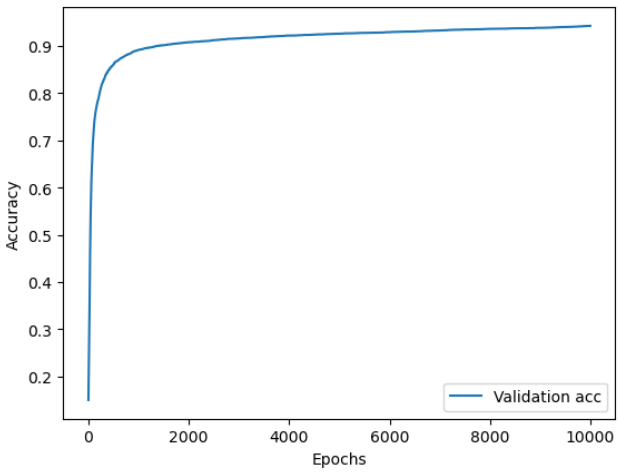

FIGURE 16 – MNIST - Validation accuracy

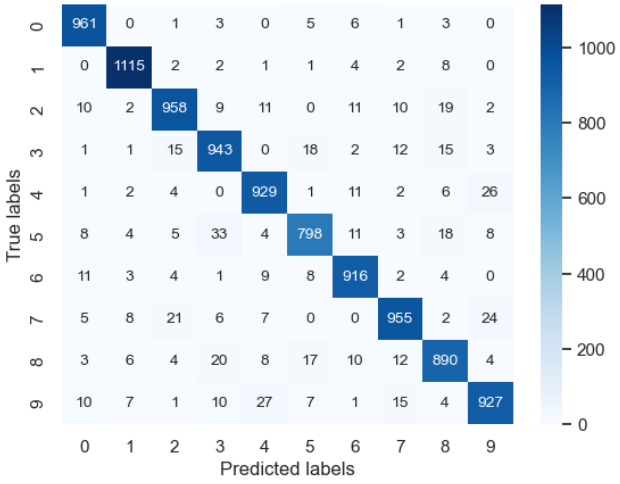

FIGURE 17 – MNIST - Test - Matrice de confusion - **Test accuracy : 93.92%**

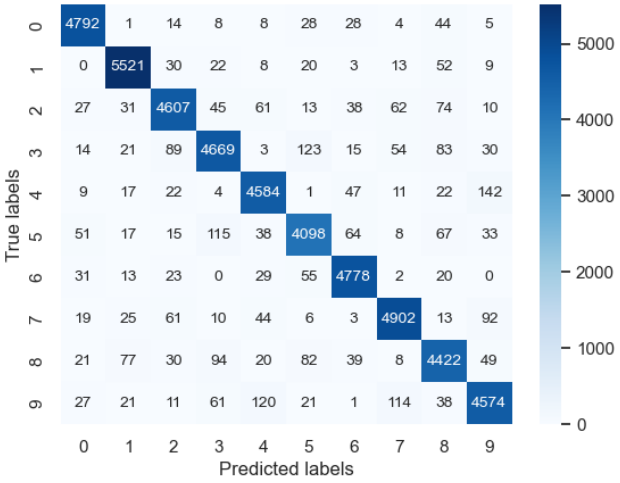

FIGURE 18 – MNIST - Train - Matrice de confusion

4.4.2 Impact de la dimension de la couche cachée

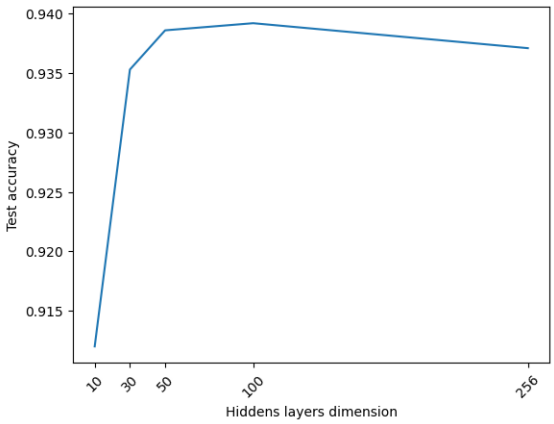

FIGURE 19 – MNIST - Impact de la dimension de la couche cachée - Test accuracy - lr=0.01, epochs=10000

4.5 Multi-classes USPS

— Multi-classes sur le dataset USPS.

— Fichier de test : **tests/multiclass_usps.ipynb**

— Datasets de tests : **USPS** : données d'images 16x16 de chiffres de 0 à 9.

4.5.1 Impact du learning rate

epochs=**5000**, loss=**MSE**

Linear(256, 100) - TanH() - Linear(100, 100) - TanH() - Linear(100, 10) - Softmax()

learning rates : [1e-4, 1e-3, 1e-2, 1e-1, 1]

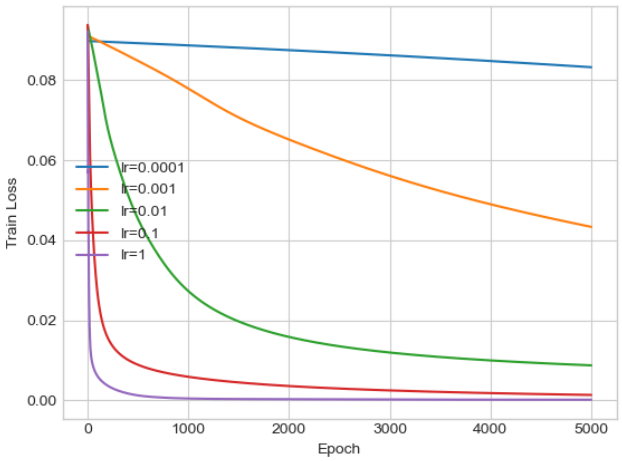

FIGURE 20 – USPS - Impact du learning rate - Train loss

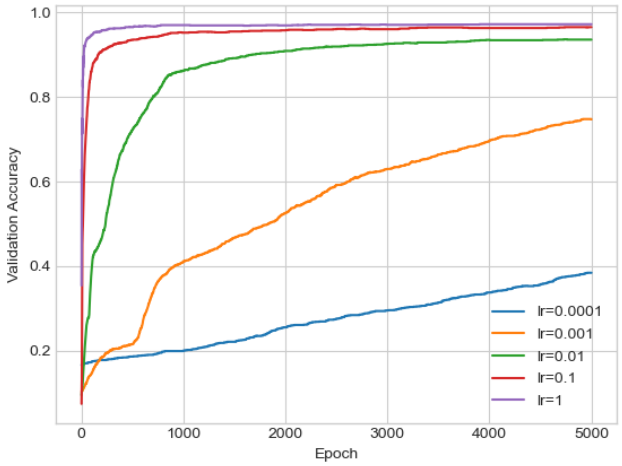

FIGURE 21 – USPS - Impact du learning rate - Validation accuracy

4.5.2 Impact de la dimension de la couche cachée

lr=**0.01**, epochs=**5000**, loss=**CE Loss**

Linear(256, hidden) - TanH() - Linear(hidden, hidden) - TanH() - Linear(hidden, 10)

hiddens : [10, 30, 50, 100, 200]

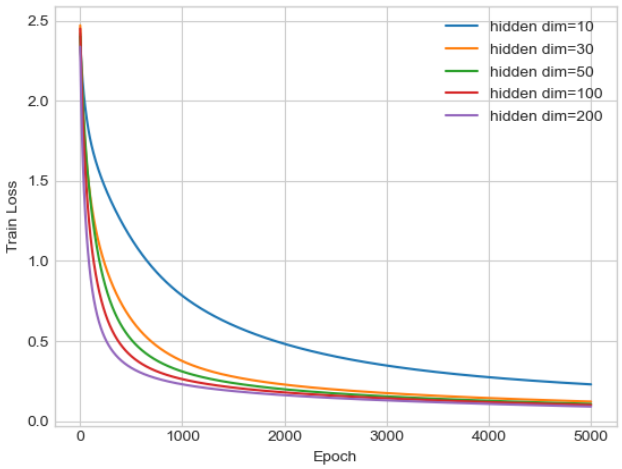

FIGURE 22 – USPS - Impact de la dimension de la couche latente - Train loss

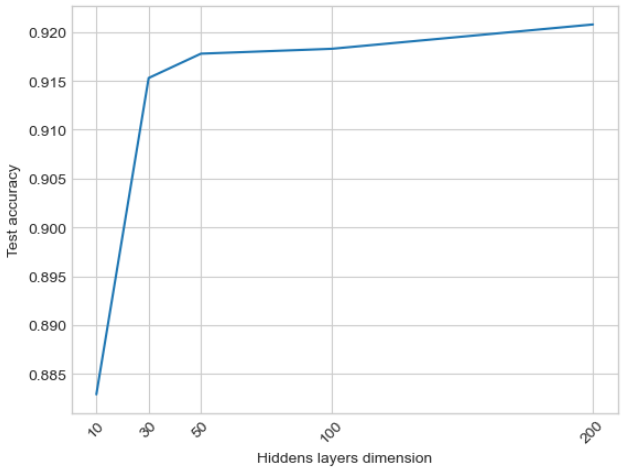

FIGURE 23 – USPS - Impact de la dimension de la couche latente - Test accuracy

4.5.3 Impact des fonctions d'activations

lr=**0.01**, epochs=**5000**, loss=**MSE**

Linear(256, hidden) - Activation1() - Linear(hidden, hidden) - Activation2() - Linear(hidden, 10) - SoftMax()

Activation1 & Activation2 : ReLU, Sigmoid, TanH.

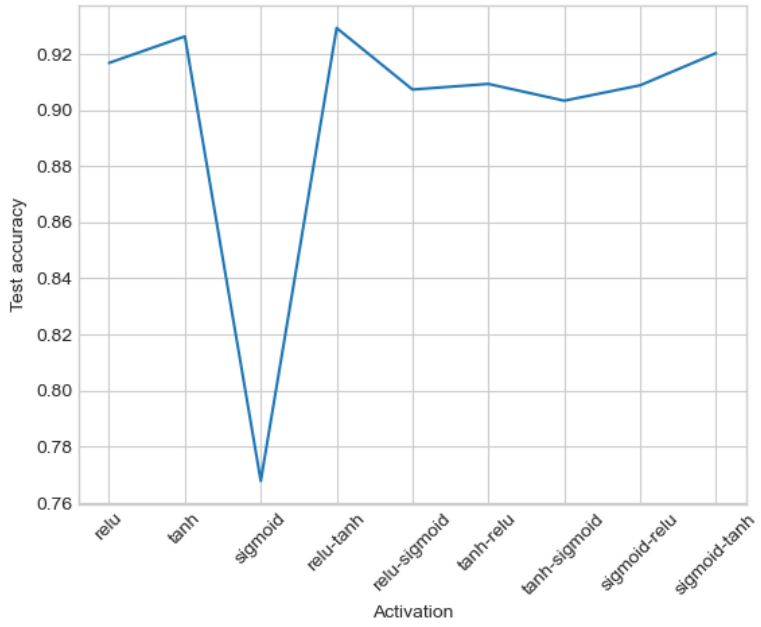

FIGURE 24 – USPS - Impact de la dimension de la couche latente - Test accuracy

4.5.4 Impact des fonctions coûts

lr=**0.01**, epochs=**10000**

(1) **Linear(256, 100) - TanH() - Linear(100, 100) - TanH() - Linear(100, 10) - Softmax() + MSE**

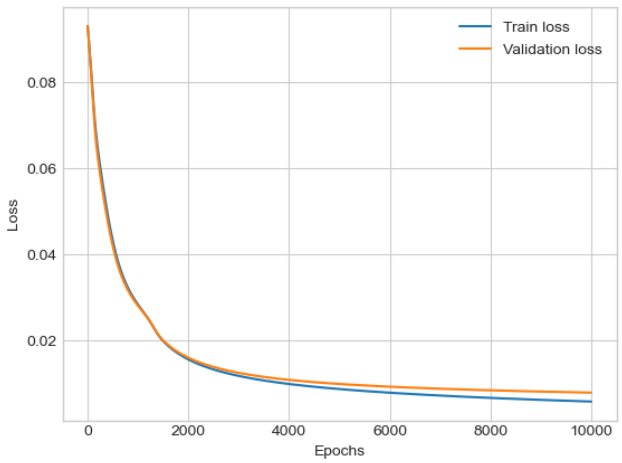

FIGURE 25 – USPS - Train & validation loss - MSE

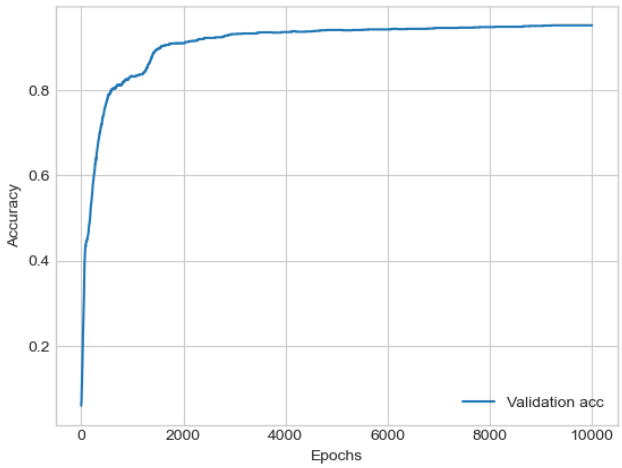

FIGURE 26 – USPS - Validation accuracy - MSE

(2) **Linear(256, 100) - TanH() - Linear(100, 100) - TanH() - Linear(100, 10) - Softmax() + Cross Entropy Loss**

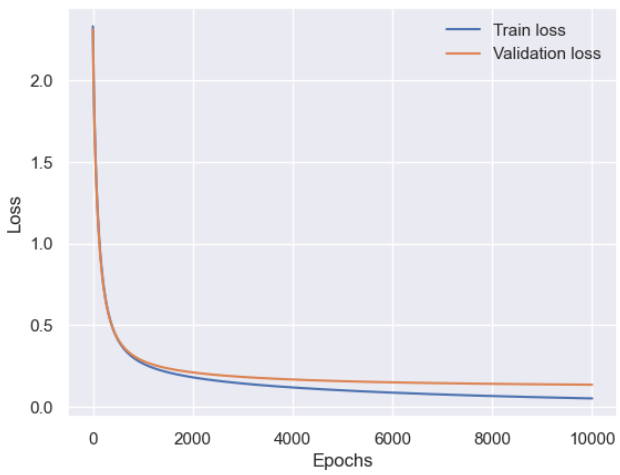

FIGURE 27 – USPS - Train & validation loss - CE Loss

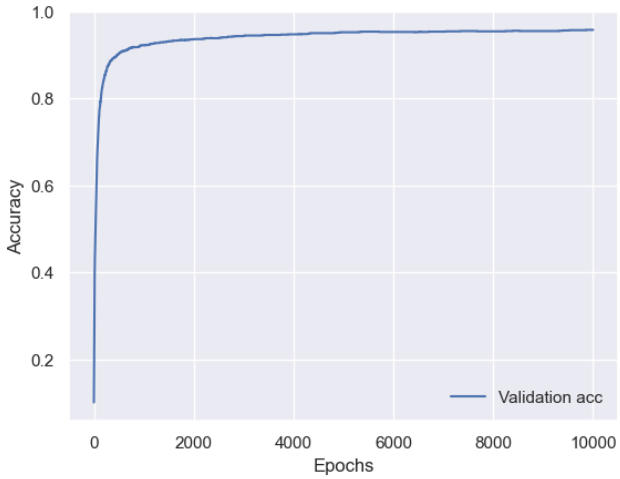

FIGURE 28 – USPS - Validation accuracy - CE Loss

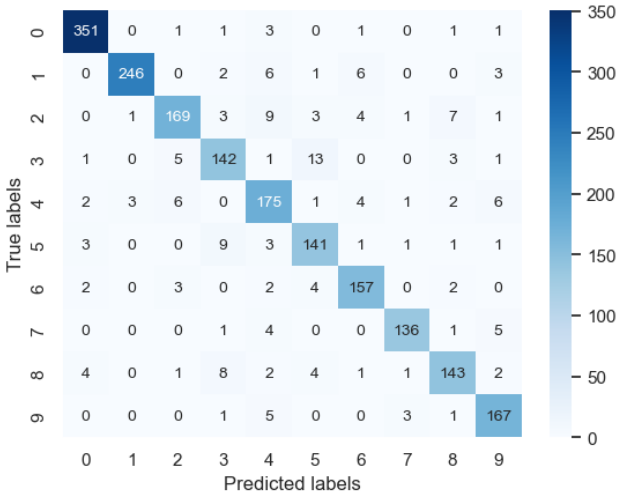

FIGURE 29 – USPS - Matrice de confusion - MSE - **Test accuracy : 91.03%**

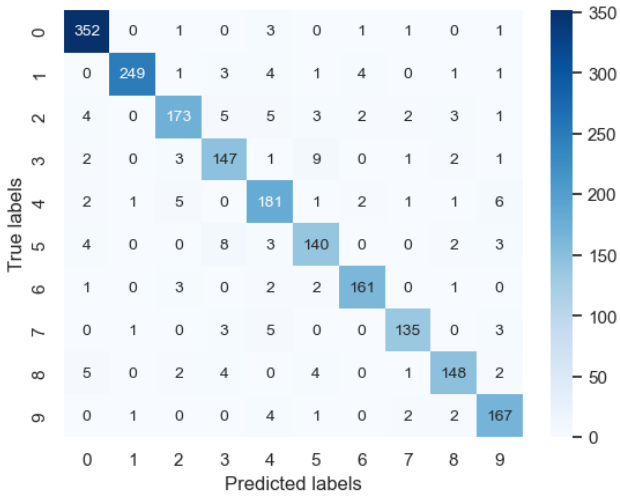

FIGURE 30 – USPS - Matrice de confusion - CE Loss - **Test accuracy : 92.32%**

4.6 Auto-encodeur USPS

— Auto-encodeur sur le dataset USPS.

— Fichier de test : **tests/autoencoder_usps.ipynb**

— Datasets de tests : **USPS** : données d'images 16x16 de chiffres de 0 à 9.

4.6.1 Auto-encodeur (1)

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

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

Taille de l'espace latent : **10**

lr=**0.1**, epochs=**2000**, loss=**MSE**.

Apprentissage de l'auto-encodeur (1) et reconstruction des exemples

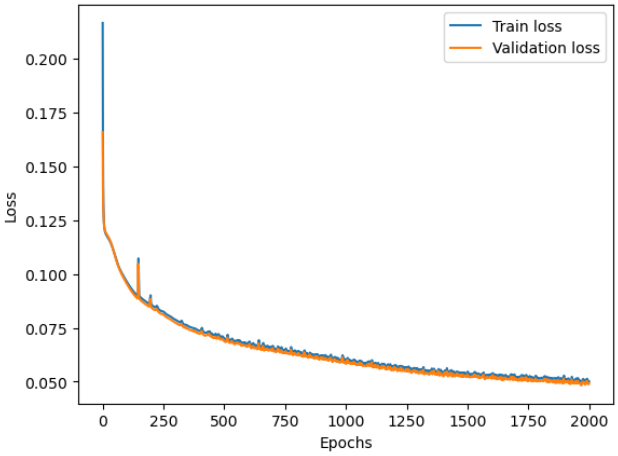

FIGURE 31 – Auto-encodeur (1) - Apprentissage

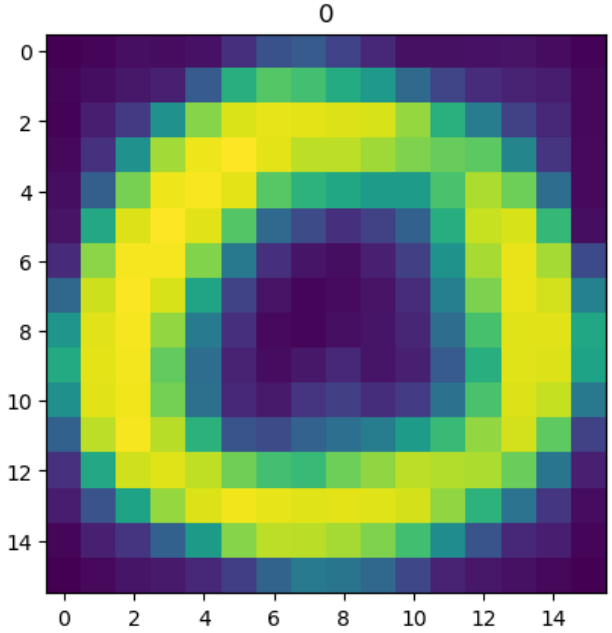

FIGURE 32 – Auto-encodeur (1) - Reconstruction d'un exemple

Etude du clustering induit dans l'espace latent de l'auto-encodeur (1)

Train ARI : 0.5594926165775456

Test ARI : 0.5091815717701222

Visualisation de représentations latente de l'auto-encodeur (1) en 2D

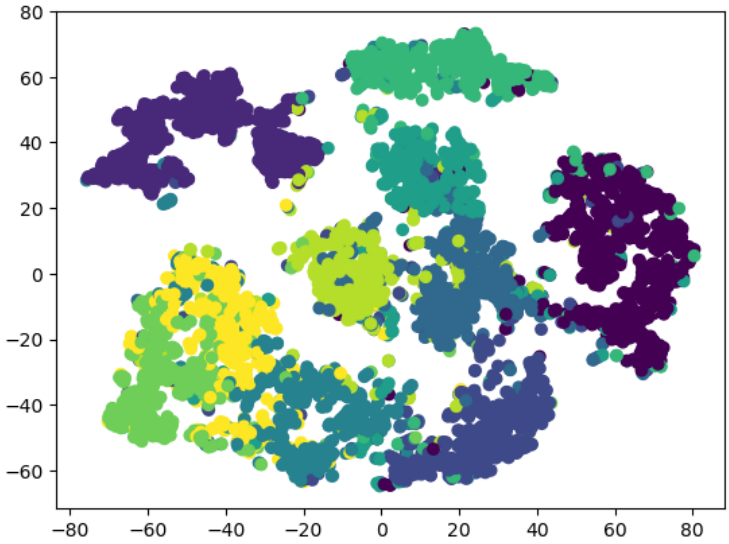

FIGURE 33 – Auto-encodeur (1) - Visualisation de l'espacelatent

Etude des performances en classification avec la représentation latente construite par l'auto-encodeur (1)

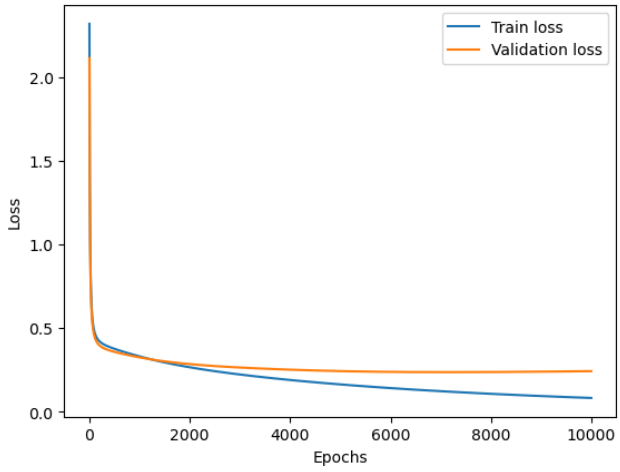

FIGURE 34 – Auto-encodeur (1) - Train & Validation Loss

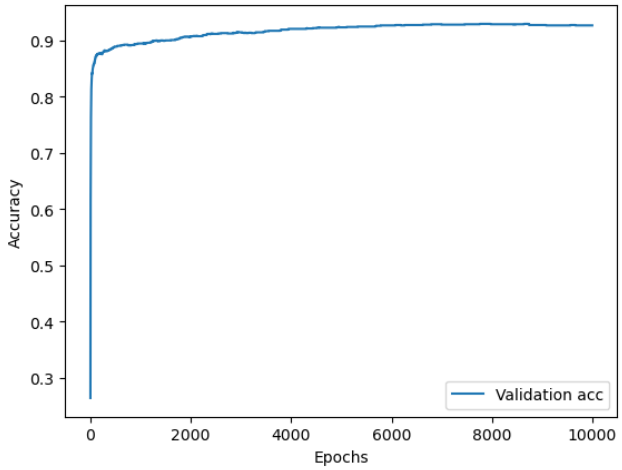

FIGURE 35 – Auto-encodeur (1) - Validation accuracy

4.6.2 Auto-encodeur (2)

Encodeur : **Linear(256, 128) - TanH() - Linear(128, 64) - TanH() - Linear(64, 10) - TanH()**

Décodeur : **Linear(10, 64) - Sigmoid() - Linear(64, 128) - Sigmoid() - Linear(128, 256) - Sigmoid()**

Taille de l'espace latent : **10**

lr=**0.1**, epochs=**2000**, loss=**Binary Cross Entropy**.

Apprentissage de l'auto-encodeur (2) et reconstruction des exemples

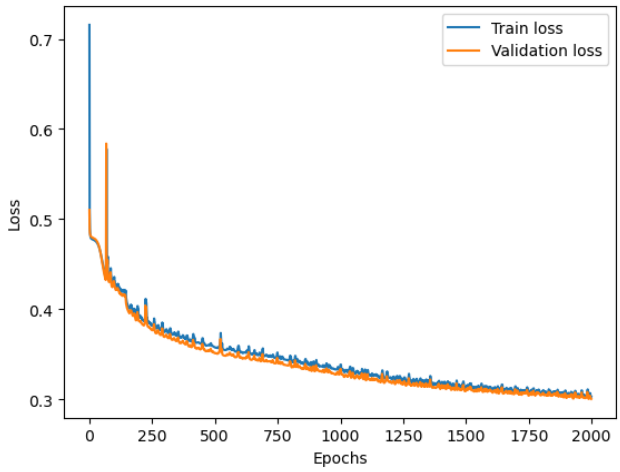

FIGURE 36 – Auto-encodeur (2) - Apprentissage

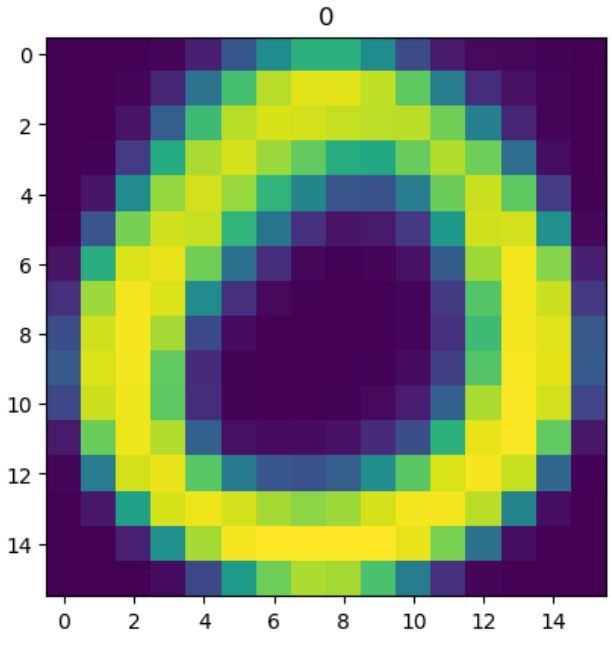

FIGURE 37 – Auto-encodeur (2) - Reconstruction d'un exemple

Etude du clustering induit dans l'espace latent de l'auto-encodeur (2)

Train ARI : 0.4727457855133672

Test ARI : 0.4223961192863275

Visualisation de représentations latente de l'auto-encodeur (2) en 2D

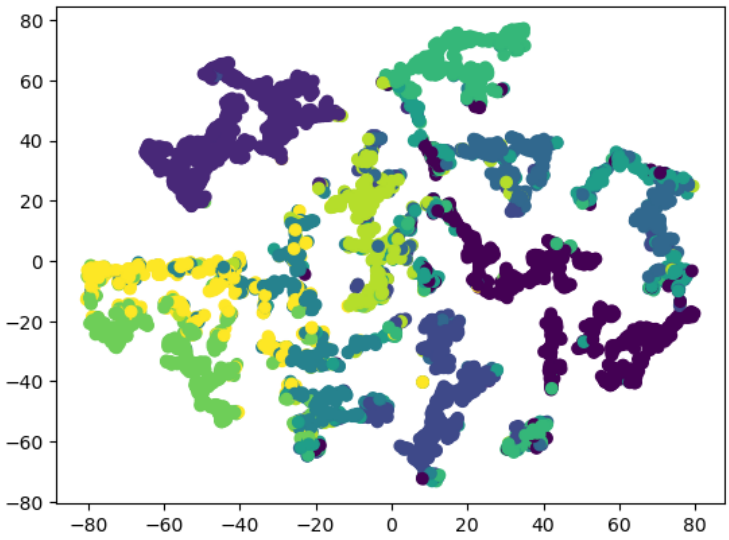

FIGURE 38 – Auto-encodeur (2) - Visualisation de l'espacelatent

Etude des performances en classification avec la représentation latente construite par l'auto-encodeur (2)

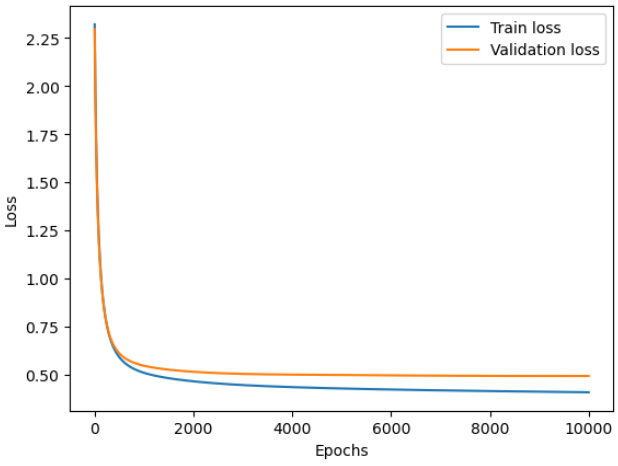

FIGURE 39 – Auto-encodeur (2) - Train & Validation Loss

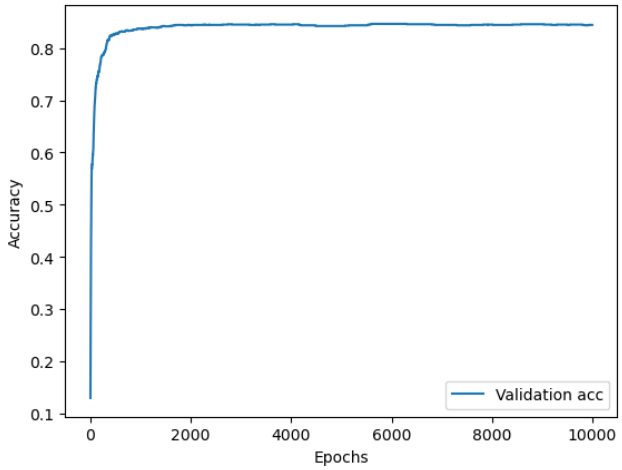

FIGURE 40 – Auto-encodeur (2) - Validation accuracy

4.6.3 Auto-encodeur (3)

Encodeur : **Linear(256,200) - Linear(200, 128) - TanH - Linear(128,100) - TanH - Linear(100, 64) - TanH- Linear(64,10)**

Décodeur : **Linear(10, 64) - Sigmoid() - Linear(64, 100) - Sigmoid() - Linear(100, 128) - Sigmoid() - Linear(128, 200) - Sigmoid() - Linear(200, 256) - Sigmoid()**

Taille de l'espace latent : **10**

lr=**1e-4**, epochs=**5000**, loss=**Binary Cross Entropy**.

Apprentissage de l'auto-encodeur (3) et reconstruction des exemples

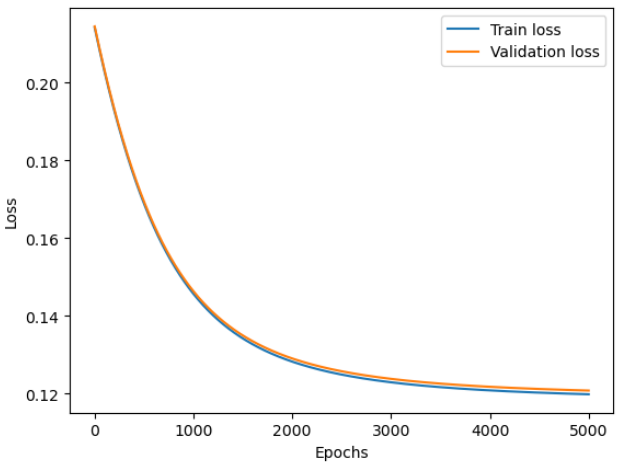

FIGURE 41 – Auto-encodeur (3) - Apprentissage

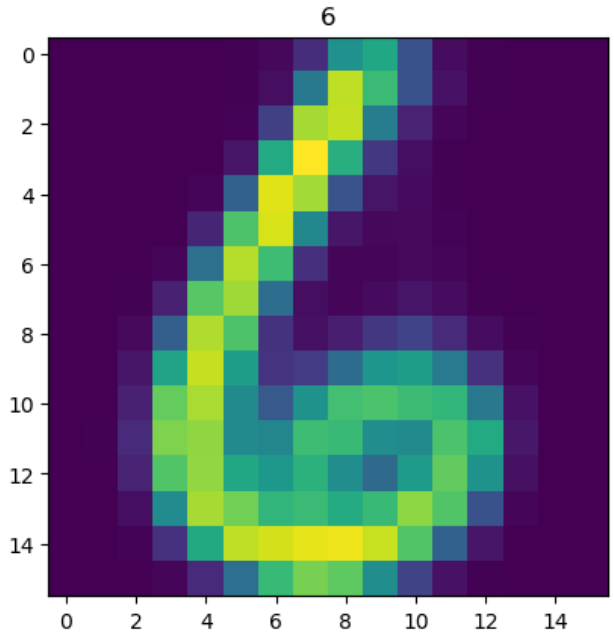

FIGURE 42 – Auto-encodeur (3) - Reconstruction d'un exemple

Etude du clustering induit dans l'espace latent de l'auto-encodeur (3)

Train ARI : 0.5037027117189826

Test ARI : 0.45647350578184137

Visualisation de représentations latente de l'auto-encodeur (3) en 2D

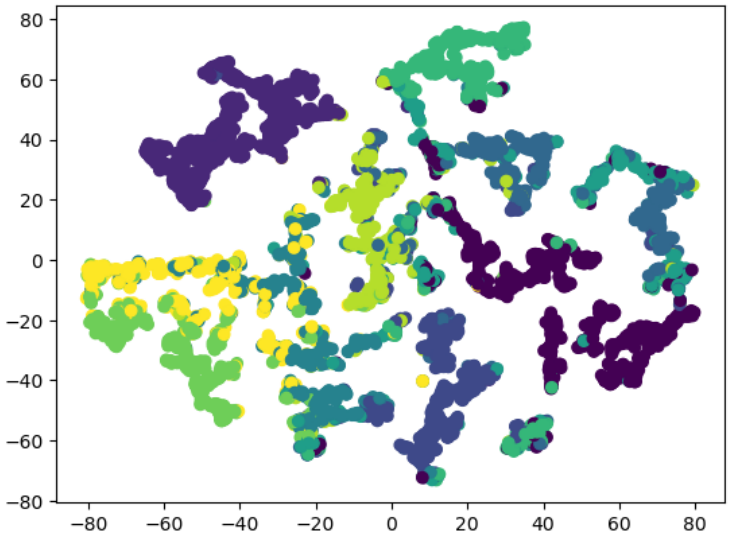

FIGURE 43 – Auto-encodeur (3) - Visualisation de l'espacelatent

Etude des performances en classification avec la représentation latente construite par l'auto-encodeur (3)

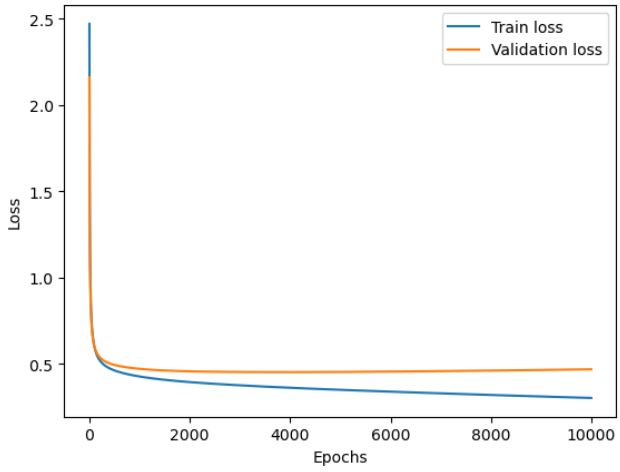

FIGURE 44 – Auto-encodeur (3) - Train & Validation Loss

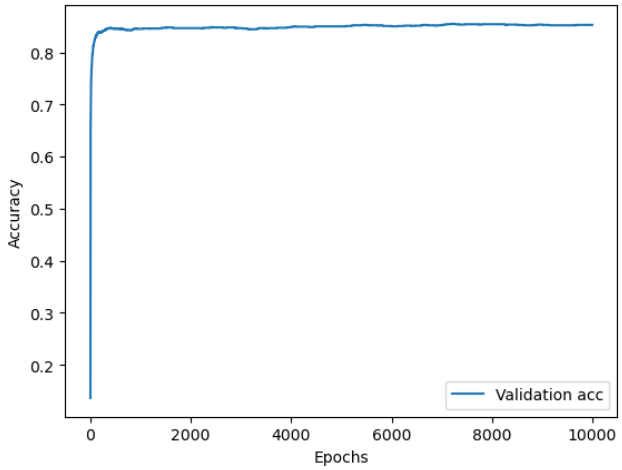

FIGURE 45 – Auto-encodeur (3) - Validation accuracy

4.7 Convolution 1D sur des données de série temporelle

— Convolution 1D.

— Fichier de test : **tests/conv1D.ipynb**

— Datasets de tests : **Air passangers** : contient les données mensuelles sur le nombre de passagers aériens internationaux de 1949 à 1960.

lr=**0.1**, epochs=**1000**, loss=**MSE**

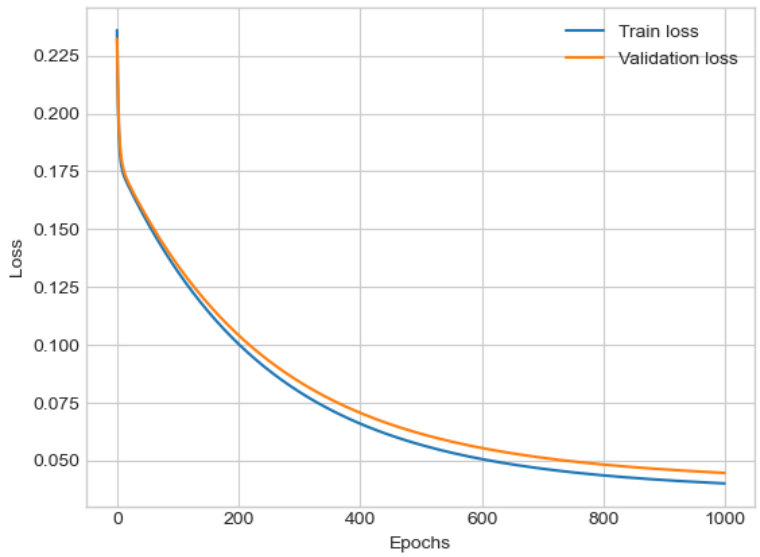

FIGURE 46 – Convolution - Série temporelle - Train & Validation loss

4.8 Convolution 1D DIGITS

— Convolution 1D sur des données chiffrées

— Fichier de test : **tests/conv1D.ipynb**

— Datasets de tests : **Digits**

4.8.1 Convolution 1D

Conv1D(k_size=3, chan_in=1, chan_out=32, stride=1) - MaxPool1D(k_size=2, stride=2) - Flatten() - Linear(992, 100) - ReLU() - Linear(100, 10)

lr=**0.1**, epochs=**1000**, loss=**Cross Entropy**

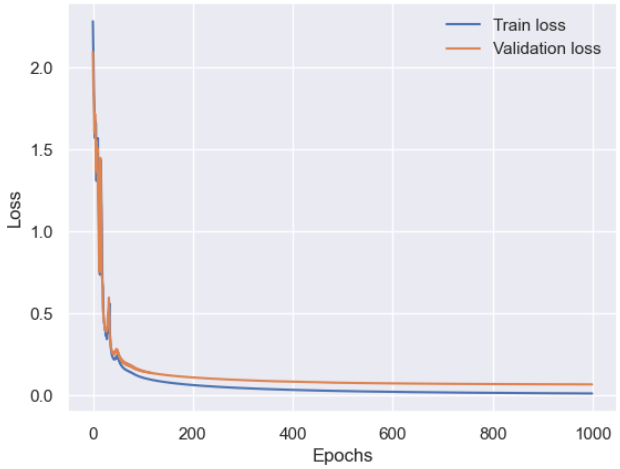

FIGURE 47 – Convolution - DIGITS - Train & Validation Loss

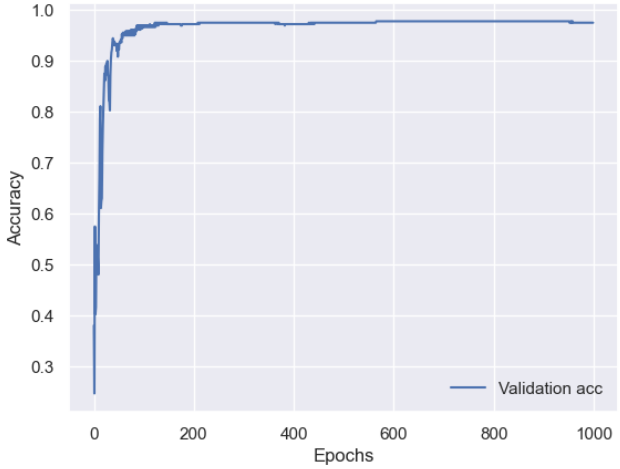

FIGURE 48 – Convolution - DIGITS - Validation accuracy

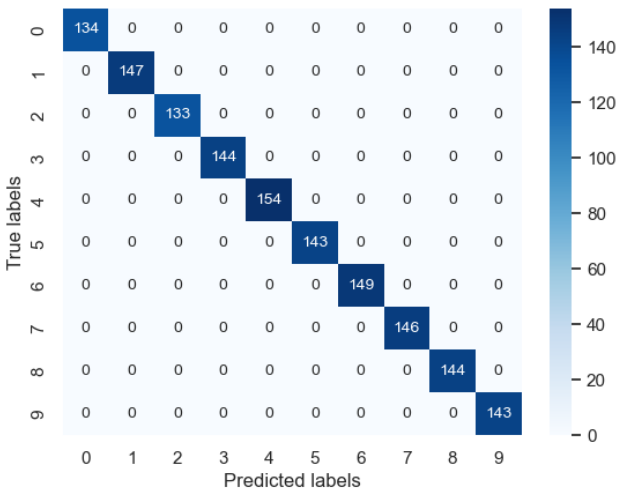

FIGURE 49 – Convolution - DIGITS - Train - Matrice de confusion

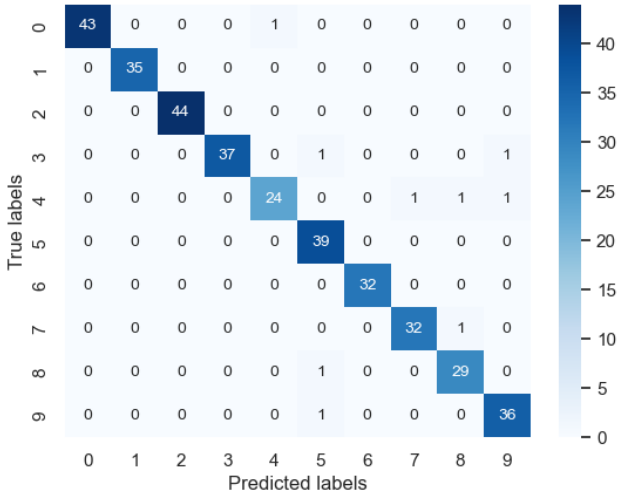

FIGURE 50 – Convolution - DIGITS - Test - Matrice de confusion

4.8.2 Average pooling

Conv1D(k_size=3, chan_in=1, chan_out=32, stride=1) - AvgPool1D(k_size=2, stride=2) - Flatten() - Linear(992, 100) - TanH() - Linear(100, 10)

lr=**0.1**, epochs=**1000**, loss=**Cross Entropy**

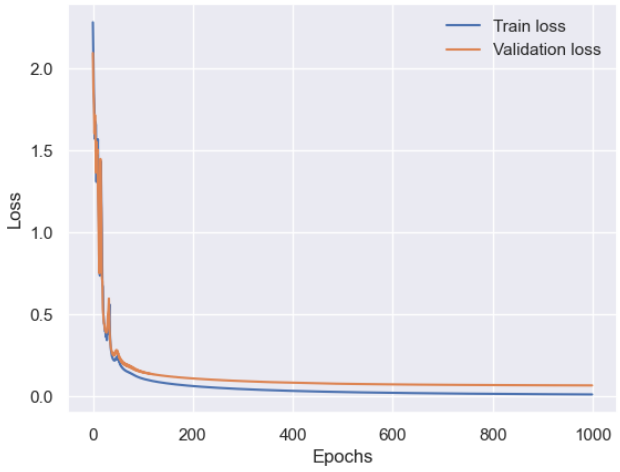

FIGURE 51 – Convolution et average pooling - DIGITS - Train & Validation Loss

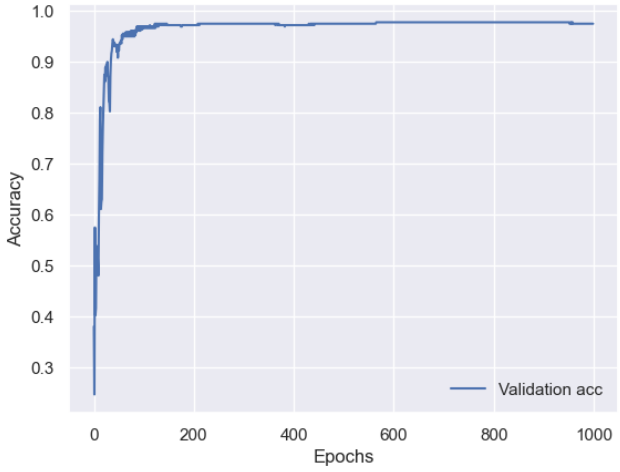

FIGURE 52 – Convolution et average pooling - DIGITS - Validation accuracy

4.8.3 Convolution 1D transposée

ConvTranspose1D(k_size=3, chan_in=1, chan_out=32, stride=1) - MaxPool1D(k_size=2, stride=2) - Flatten() - Linear(992, 100) - ReLU() - Linear(100, 10)

lr=**0.1**, epochs=**1000**, loss=**Cross Entropy**

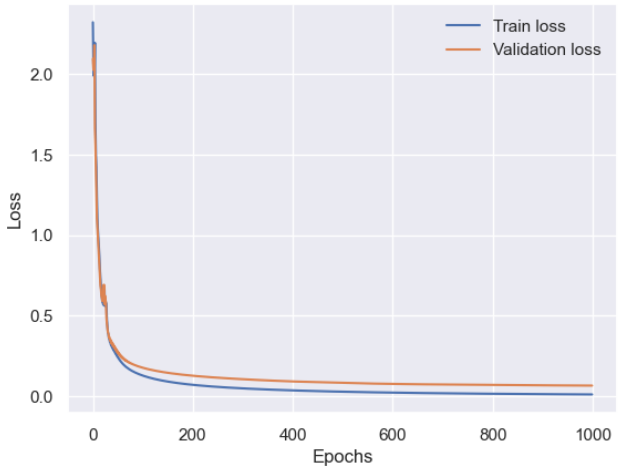

FIGURE 53 – Convolution transpoée - DIGITS - Train & Validation Loss

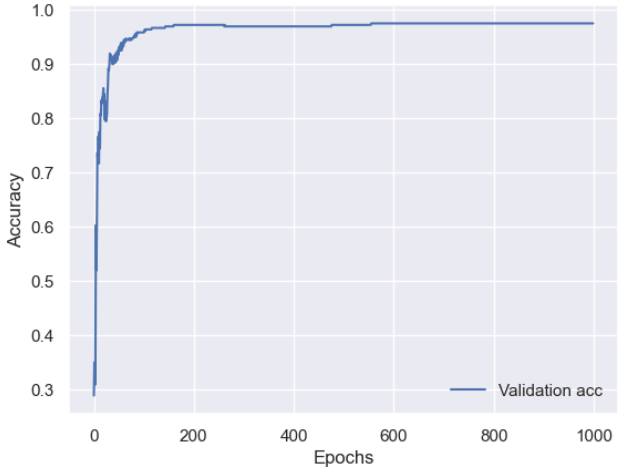

FIGURE 54 – Convolution - DIGITS - Validation accuracy

4.9 Convolution 1D USPS

— Convolution 1D sur des données chiffrées

— Fichier de test : **tests/conv1D_usps.ipynb**

— Datasets de tests : **USPS**

Conv1D(k_size=3, chan_in=1, chan_out=32, stride=1) - MaxPool1D(k_size=2, stride=2) - Flatten() - Linear(4096, 100) - ReLU() - Linear(100, 10)

lr=**0.1**, epochs=**10000**, loss=**Cross Entropy**

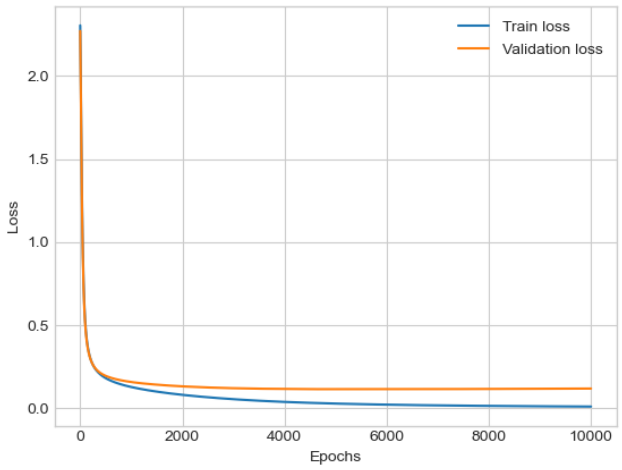

FIGURE 55 – Convolution - USPS - Train & Validation Loss

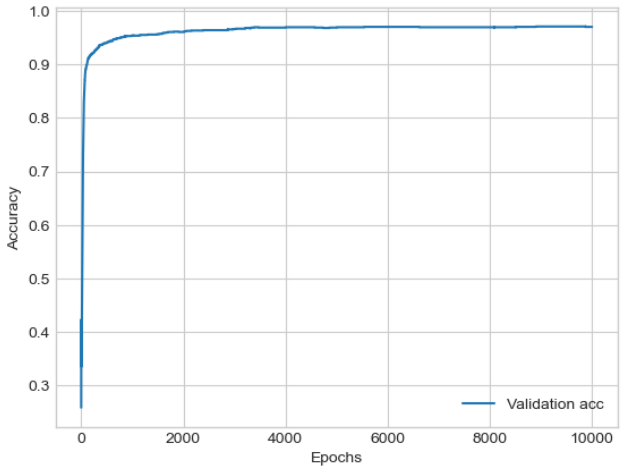

FIGURE 56 – Convolution - USPS - Validation accuracy

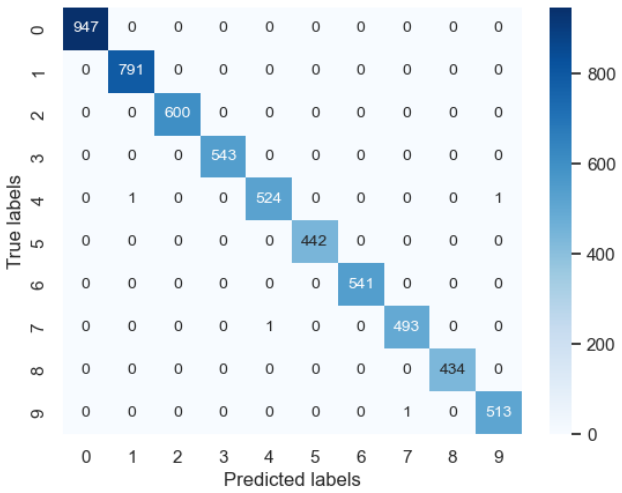

FIGURE 57 – Convolution - USPS - Train - Matrice de confusion - **Train accuracy : 99.99%**

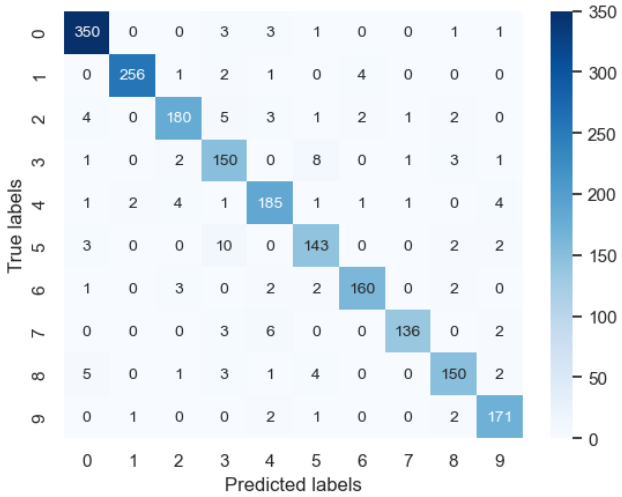

FIGURE 58 – Convolution - USPS - Test - Matrice de confusion - **Test accuracy : 93.72%**