

Ce projet est réalisé par Sarah BEHANZIN et Amine BELGACEM dans le cadre de notre cours de Deep Learning.

## Librairies nécessaire

Entrée [134]:

```
1 #Librairies de base
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import pandas as pd
5 import seaborn as sns
6
7 #Librairies Scikit Learn
8 from sklearn.model_selection import train_test_split, KFold
9
10
11 #Librairies keras
12 import keras
13 from tensorflow.keras.datasets import mnist
14 from tensorflow.keras.datasets.mnist import load_data
15 from keras.datasets import mnist
16 from keras.layers import Input, Conv2D, MaxPooling2D, BatchNormalization
17 from keras.layers import UpSampling2D, Dropout, Dense, Flatten
18 from keras.models import Sequential, Model
19 from keras.callbacks import TensorBoard
20 from tensorflow.keras.utils import plot_model
21 from tensorflow.keras.utils import to_categorical
```

## Introduction

Dans le cadre de ce projet, nous devons réaliser un algorithme de Deep Learning nous permettant de pouvoir reconnaître et déchiffrer les numéros écrit sur les images issus de la base de données MNIST.

## Importation des données

On commence par importer les données.

On a récupéré les fichiers test et train de MNIST via la librairie keras. On peut récupérer la base de données via la fonction `load_data()`

Entrée [185]:

```
1 #Chargement de la base de données
2 (train_X, train_y), (test_X, test_y) = mnist.load_data()
```

# Organisation des données

Notre base de données est déjà découpée en plusieurs parties :

- `train_y` va contenir la colonne contenant tous les labels de la base de données d'apprentissage.
- `train_X` va contenir tout le reste, c'est à dire les pixels des images de la base de données d'apprentissage.
- `test_X` va contenir la colonne contenant tous les labels de la base de données de test.
- `test_y` va contenir tout le reste, c'est à dire les pixels des images de la base de données de test.

On regarde la répartition de notre base de donnée `train_X` et `test_X`

Entrée [186]:

1	<code>train_X.shape</code>
---	----------------------------

Out[186]:

(60000, 28, 28)

Entrée [187]:

1	<code>test_X.shape</code>
---	---------------------------

Out[187]:

(10000, 28, 28)

On visualise si nous avons une bonne répartition des labels dans notre jeu de donnée.

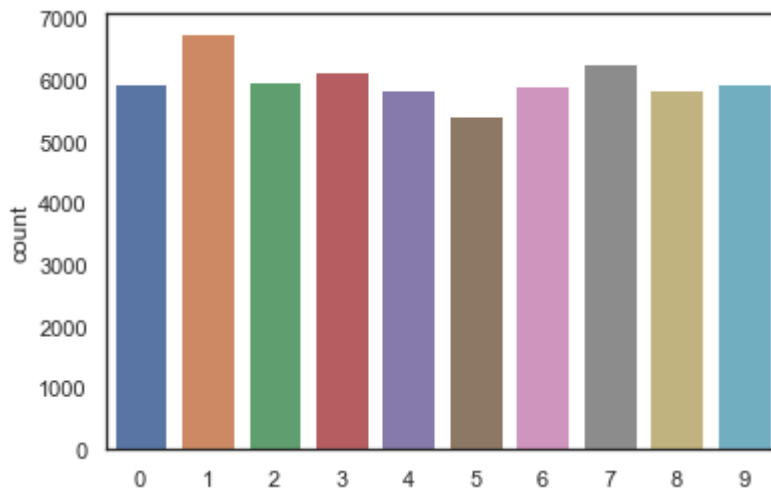
Ce qui est primordiale, car si on a une répartition vraiment déséquilibrée des labels dans notre base d'apprentissage et de test, nous ne pourrions pas bien faire apprendre à notre modèle et donc cela nous donnera une mauvaise prédiction.

Entrée [188]:

```
1 #visualisation de la repartition des labels de la base d'apprentissage
2 sns.set(style='white', context='notebook', palette='deep')
3 ax = sns.countplot(train_y)
```

C:\Users\Belgacem\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

warnings.warn(

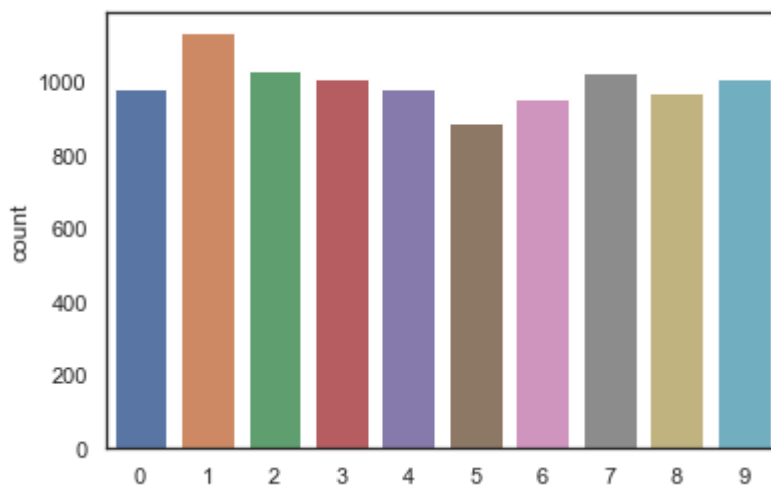


Entrée [189]:

```
1 #visualisation de la repartition des labels de la base de test
2 sns.set(style='white', context='notebook', palette='deep')
3 ax = sns.countplot(test_y)
```

C:\Users\Belgacem\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

warnings.warn(



Pour pouvoir augmenter la précision de notre réseau de neuronne, on doit tout d'abord mettre nos données sous la bonne forme. On a 784 colonnes et on doit les mettre sous forme de matrice 28x28 correspondant à une image de 28x28 pixels.

Entrée [190]:

```
1 #Mise en forme des données
2 train_X = train_X.reshape((train_X.shape[0], 28, 28, 1))
3 test_X = test_X.reshape((test_X.shape[0], 28, 28, 1))
```

Entrée [191]:

```
1 train_X.shape
```

Out[191]:

```
(60000, 28, 28, 1)
```

Entrée [192]:

```
1 test_X.shape
```

Out[192]:

```
(10000, 28, 28, 1)
```

On normalise maintenant nos données pour avoir les valeurs de nos pixels dans le code RVB, entre 0 et 255.

Entrée [193]:

```
1 train_X = train_X.astype('float32') / 255
2 test_X = test_X.astype('float32') / 255
```

On transforme nos labels en variable catégorielles grâce à la fonction `to_categorical()`. Cette fonction va convertir notre chiffre correspondant au label en un vecteur binaire de taille 10 (car nous avons des chiffres allant de 0 à 9).

Cela va nous permettre d'optimiser notre réseau de neurones.

Entrée [194]:

```
1 train_y = to_categorical(train_y)
2 test_y = to_categorical(test_y)
```

## Création du modèle

On va maintenant créer le modèle.

On commence par initialier le modèle :

Le modèle que nous utilisons est le `Sequential()`. Il permet de regrouper et rassembler plusieurs couches.

Entrée [195]:

```
1 model_ = Sequential()
```

On définit maintenant le modèle.

Notre modèle de réseau de neurone convolutif aura, pour commencer, une seule couche de taille (3,3) avec 32 filtres.

Avec la fonction MaxPooling2D(), cela va nous retourner la couche de regroupement maximum. Cette fonction va réaliser un échantillonnage de format 2x2 sur toute notre image et va retourner la valeur maximale du pixel compris dans l'entrée qu'il échantillonne.

Flatten() va nous permettre d'aplatir les filtres pour fournir des caractéristiques à notre classificateur.

La dernière ligne représente la couche de sortie. Comme nous avons une tâche de classification de 10 classes, on doit avoir une couche avec 10 sorties pour pouvoir prédire les images par rapport à leurs labels.

Entrée [196]:

```
1 #implémentation des différentes couches
2 model_.add(Conv2D(32, (3, 3),
3                   activation='relu',
4                   kernel_initializer='he_uniform',
5                   input_shape=(28, 28, 1)))
6
7 model_.add(MaxPooling2D((2, 2)))
8 model_.add(Flatten())
9
10 model_.add(Dense(100,
11                  activation='relu',
12                  kernel_initializer='he_uniform'))
13
14 model_.add(Dense(10, activation='softmax'))
```

On utilise l'optimiseur Adam et la fonction de perte d'entropie croisée catégorielle.

Nous prenons comme métrique d'analyse la précision.

Entrée [197]:

```
1 #Compilation du modèle
2 model_.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Nous pouvons commencer à entraîner le modèle.

Entrée [199]:

```
1 #On entraine le modèle
2 model_.fit(x=train_X,
3           y=train_y,
4           epochs=10,
5           batch_size=32,
6           validation_data=(test_X, test_y),
7           verbose=0)
```

Out[199]:

<keras.callbacks.History at 0x145d42b5e80>

Entrée [200]:

```
1 #Visualisation de la précision et de la perte de notre modèle
2 loss_model, acc_model = model_.evaluate(x=test_X,y=test_y)
3 print('Notre modèle à une précision de ',
4       '%.3f' % (acc_model * 100.0) ,
5       '% sur l\'ensemble de test ')

```

313/313 [=====] - 2s 6ms/step - loss: 0.0626 - accu  
racy: 0.9876  
Notre modèle à une précision de 98.760 % sur l'ensemble de test

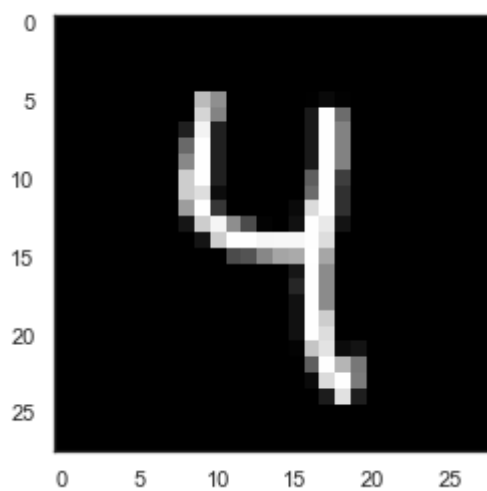
Les lignes de codes ci-dessous vont nous permettre de faire des prédictions et les vérifier visuellement.

On crée une variable `image_index` qui représente un chiffre aléatoire entre 1 et 10 000.

Entrée [201]:

```
1 #Verification du résultats attendus et du résultats obtenues
2 from random import *
3
4 liste = []
5 for i in range(10000):
6     liste.append(randint(0, 1000) )
7
8 shuffle(liste)
9 image_index = choice(liste)
10
11 plt.imshow(test_X[image_index].reshape(28, 28),cmap='gist_gray')
12 pred = model_.predict(test_X[image_index].reshape(1, 28, 28, 1))
13 print('Le nombre est',pred.argmax())
```

Le nombre est 4



## Modèle avec 3-fold validation croisée

On va maintenant réaliser une validation croisée avec 3-Fold

Pour faciliter et rendre le code plus clair, on crée une fonction qui va retourner notre modèle.

Entrée [202]:

```
1 #Création de la fonction du modèle
2 def fonction_model():
3     model = Sequential()
4     model.add(Conv2D(32, (3, 3),
5                       activation='relu',
6                       kernel_initializer='he_uniform',
7                       input_shape=(28, 28, 1)))
8
9     model.add(MaxPooling2D((2,2)))
10
11    model.add(Flatten())
12
13    model.add(Dense(100,
14                    activation='relu',
15                    kernel_initializer='he_uniform'))
16
17    model.add(Dense(10,
18                    activation='softmax'))
19
20    model.compile(optimizer='adam',
21                  loss='categorical_crossentropy',
22                  metrics=['accuracy'])
23    return model
```

Nous allons réaliser une opération de Kfold avec 3 splits.

Entrée [203]:

```
1 #Initialisation de la méthode des K-Folds
2 kfold = KFold(3, shuffle=True, random_state=1)
```

Nous allons maintenant créer une boucle qui va permettre de faire apprendre notre modèle sur les 3 divisions que fait notre Kfold.

Sur chaque itération de la boucle, cela va diviser les bases de données d'apprentissages et de tests de manière différentes pour permettre un meilleur apprentissage du modèle.

Le chargement de cette cellule prends un certain temps.



Entrée [204]:

```
1 #Entrainement du modèle avec La méthode des K-Folds
2 hist_f= list()
3 acc_f = list()
4 loss_f = list()
5 for train_i, test_i in kfold.split(train_X):
6     k_trainX, k_trainY, k_testX, k_testY = train_X[train_i],
7                                           train_y[train_i],
8                                           train_X[test_i],
9                                           train_y[test_i]
10
11     model = fonction_model()
12
13     hist = model.fit(k_trainX,
14                     k_trainY,
15                     epochs=10,
16                     batch_size=32,
17                     validation_data=(k_testX, k_testY),
18                     verbose=0)
19
20     loss,acc = model.evaluate(k_testX,
21                              k_testY,
22                              verbose=0)
23
24     acc_f.append('%0.3f' % (acc * 100.0))
25     loss_f.append('%0.3f' % (loss * 100.0))
26     hist_f.append(hist)
27
```

Entrée [205]:

```
1 #Précision du modèle sur Les différentes divisions de La base de données
2 acc_f
```

Out[205]:

['98.395', '98.290', '98.640']

Entrée [206]:

```
1 #Perte du modèle sur Les différentes divisions de La base de données
2 loss_f
```

Out[206]:

['7.380', '7.969', '6.263']

Entrée [290]:

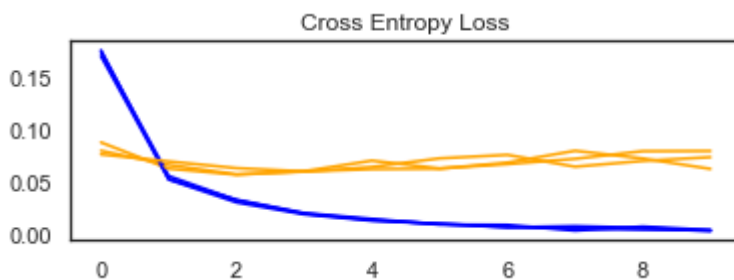
```

1 #On affiche Le graphique correspondant à l'évolution de la perte du modèle lors de l'ap
2
3 for i in range(len(hist_f)):
4     #1er graphique
5     plt.subplot(2, 1, 1)
6     plt.title('Cross Entropy Loss')
7     plt.plot(hist_f[i].history['loss'], color='blue', label='train')
8     plt.plot(hist_f[i].history['val_loss'], color='orange', label='test')
9 plt.show()

```

<ipython-input-290-dece0c34af24>:6: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

```
plt.subplot(2, 1, 1)
```



Entrée [291]:

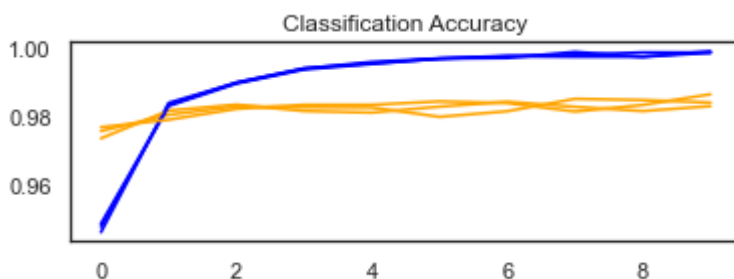
```

1 #On affiche Le graphique correspondant à l'évolution de la précision du modèle lors de
2
3 for i in range(len(hist_f)):
4     #2ème graphique
5     plt.subplot(2, 1, 2)
6     plt.title('Classification Accuracy')
7     plt.plot(hist_f[i].history['accuracy'], color='blue', label='train')
8     plt.plot(hist_f[i].history['val_accuracy'], color='orange', label='test')
9 plt.show()

```

<ipython-input-291-9e828b1d411f>:3: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.

```
plt.subplot(2, 1, 2)
```



Nous allons ré utiliser les lignes de codes nous permettant de prédire et vérifier visuellement la prédiction et les utiliser avec notre modèle avec les 3-Folds.

Nous allons comparer les prédiction des deux modèles en utilisant le même indice d'image.

Entrée [289]:

```
1 from random import *
2
3 liste = []
4 for i in range(10000):
5     liste.append(randint(0, 1000) )
6
7 shuffle(liste)
8 image_index_2 = choice(liste)
9
10 plt.imshow(test_X[image_index_2].reshape(28, 28),cmap='gist_gray')
11 pred_Kfold = model.predict(test_X[image_index_2].reshape(1, 28, 28, 1))
12 print('Le nombre prédit par le modèle avec les Kfold est',pred_Kfold.argmax())
13 pred = model_.predict(test_X[image_index_2].reshape(1, 28, 28, 1))
14 print('Le nombre prédit par le modèle normal est',pred.argmax())
```

Le nombre prédit par le modèle avec les Kfold est 3

Le nombre prédit par le modèle normal est 3

