# TP7 – CNN

Ce TP sera réalisé sous tensorFlow2 et keras. La documentation pourra être trouvée sous https://www.tensorflow.org/api_docs/python/tf/keras/

## I. Chargement et mise en forme des données

Etudier et exécuter le programme suivant.

```
# Import libraries and modules
# Import libraries and modules
import numpy as np
import time
np.random.seed(123)  # for reproducibility

from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

import tensorflow as tf
from tensorflow.keras import Sequential, Model, Input
from tensorflow.keras.layers import Conv2DTranspose, Reshape, Dense, Activation,
Flatten, Dropout, Convolution2D, MaxPooling2D, Input
from utilitaire import affiche
#####################################################
# I - Load pre-shuffled MNIST data train and test sets
#####################################################
(from tensorflow.keras.datasets.mnist import load_data
from matplotlib import pyplot
# load dataset
(X_train, y_train), (X_test, y_test) = load_data()
X_train, pipo, y_train, pipo = train_test_split(X_train, y_train, test_size=0.9)
X_test, pipo, y_test, pipo = train_test_split(X_test, y_test, test_size=0.9)

for i in range(200):
  plt.subplot(10,20,i+1)
  plt.imshow(X_train[i,:].reshape([28,28]), cmap='gray')
  plt.axis('off')
plt.show()

# Preprocess input data
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

# Preprocess class labels
Y_train = tf.keras.utils.to_categorical(y_train, 10)
```

```
Y_test = tf.keras.utils.to_categorical(y_test, 10)
```

**Questions**

- Que réalise-t-il ? Justifier chaque ligne.
- Combien y a-t-il d'images dans la base de test ? Dans la base d'apprentissage ? Quelle est la taille des images ? Combien y a-t-il de classes ?
- En quoi consiste le pré-traitement des données d'entrées ? Pourquoi le réalise-t-on ?
- A quoi sert la fonction tf.keras.utils.to_categorical ? Quelle est la taille de y_train ? de Y_train ? Commenter.

# II. Régression logistique

## II.1. définition du réseau

On souhaite classifier les données d'entrée en 10 classes et pour cela, on estime la sortie comme une somme pondérée des valeurs entrées. On estime les poids par apprentissage d'un réseau de neurones à une couche :

```
inputs = Input(shape=(28,28,1))
x = inputs
x=Flatten()(x)
outputs=Dense(10, activation='softmax')(x)
model = Model(inputs, outputs)
model.summary()
```

**Questions**

- Pourquoi utilise-t-on la fonction d'activation softmax ?
- Combien y a-t-il de paramètres à apprendre (Trouver les par le calcul puis vérifier) ?
- A quoi sert la commande Flatten ?

## II.2. Apprentissage

On utilise l'optimiseur SGD (Stochastic gradient descent optimizer) avec ses paramètres par défaut :

```
lr= ??
batch_size=256
epochs=10

sgd1= tf.keras.optimizers.SGD(learning_rate=lr)

model.compile(loss='categorical_crossentropy', optimizer=sgd1, metrics=['accuracy'])
tps1 = time.clock()
history =model.fit(X_train, Y_train, batch_size=batch_size, epochs=epochs,
verbose=1,validation_data=(X_test, Y_test))
tps2 = time.clock()

from utilitaire import affiche
affiche(history) #donnee en annexe
print('lr=',lr,'batch_size=',batch_size, 'epochs=',epochs)
print('Temps d apprentissage',tps2 - tps1)
```

**Questions**

- Que représente lr, batch et epochs ?
- Pourquoi utilise-t-on 'categorical_crossentropy' comme fonction perte ?
- Commenter ligne à ligne le programme. Quelle fonction perte est utilisée ?
- Que réalise la fonction affiche(history) donnée en annexe? Est-ce que l'apprentissage se passe bien ?

### II.3. Evaluation du modèle

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1]*100)

y_pred = model.predict(X_test)
y_pred = y_pred.argmax(axis=-1)
print('Confusion Matrix')
print(confusion_matrix(y_test, y_pred))
```

**Questions**

- Que renvoie model.predict et pourquoi utilise-t-on y_pred.argmax ?
- Refaire l'apprentissage en réglant au mieux les valeurs de lr, batch et epochs. Quel est le meilleur taux de reconnaissance obtenu ?

## III. MLP

Reprendre toutes les questions précédentes en ajoutant une couche cachée à 256 neurones et la fonction d'activation adéquate.

**Questions**

- Combien y a-t-il de paramètres à estimer (trouver les par le calcul puis vérifier) ?
- Changer les valeurs du pas d'apprentissage, du momentum, du batch_size pour optimiser les performances du réseau (les valeurs par défaut sont lr=0.01, momentum=0.0).
- Que se passe-t-il si le pas d'apprentissage est trop grand ? Trop petit ? Comment agit le momentum ?
- Ajouter une couche de dropout. Comment agit-elle ?
- Ajouter une seconde couche cachée. Que se passe-t-il ?
- Conclusion.

## IV. CNN

Garder les paramètres d'apprentissage optimaux et définir un réseau en ajoutant avant les couches du MLP :

- une couche convolutionnelle composée de 32 filtres 3x3 suivie de la fonction d'activation RELU
- une couche convolutionnelle composée de 64 filtres 3x3 suivie de la fonction d'activation RELU

**Questions**

- Combien y a-t-il de paramètres à estimer (trouver les par le calcul puis vérifier)? Comment est la convergence du réseau ? Est-elle plus rapide, plus lente ? Comment sont les résultats ?

- Pour être plus robuste aux translations, ajouter un max-pooling de taille 2 après la dernière couche convolutionnelle.
- Pour améliorer la généralisation, ajouter une ou plusieurs couches de dropout et reprendre les mêmes questions.
- Essayer différentes architectures du réseau pour optimiser les résultats.
- Sur ces images, il n'y a pas grand intérêt à utiliser un CNN plutôt qu'un MLP. Pourquoi ? Quelle va être la force des CNN ?

## V. ANNEXE

```
def affiche(history):
    # summarize history for accuracy
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
```