TP7 - CNN

Ce TP sera réalisé sous tensorFlow et Keras. La documentation pourra être trouvée sous https://keras.io/

I. Chargement et mise en forme des données

Etudier et exécuter le programme suivant.

```
# Import libraries and modules
import time
import numpy as np
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
from tensorflow.keras.optimizers import SGD
from keras.utils import np_utils
from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Flatten, Dropout, Convolution2D,
MaxPooling2D, Input

#####################################################
# I - Load pre-shuffled MNIST data train and test sets
#####################################################
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
#Ne conserve que 10% de la base
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([32,32,3]), cmap='gray')
  plt.axis('off')
plt.show()

# Preprocess input data
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

# Preprocess class labels
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_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 np_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 :

```
# Define model architecture
inputs = Input(shape=(32,32,3))
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 ?
- Que représente 10 sur la ligne outputs=Dense(10, activation='softmax')(x) ? Aurait-on pu utiliser une autre valeur ?
- A quoi sert la commande Flatten ?
- Combien y a-t-il de paramètres à apprendre (Trouver les par le calcul puis vérifier) ?

II.2. Apprentissage

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

```
lr=0.1
batch_size=256
epochs=10

sgd1=SGD(lr=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=0,validation_data=(X_test, Y_test))
tps2 = time.clock()

affiche(history)
print('lr=',lr,'batch_size=',batch_size, 'epochs=',epochs)
print('Temps d apprentissage',tps2 - tps1)
```

La fonction affiche() est donnée en annexe

Questions

- Que représente lr, batch et epochs ?
- Pourquoi utilise-t-on 'categorical_crossentropy' comme fonction perte ?
- Commenter ligne à ligne le programme.
- Que réalise la fonction affiche(history) ? Est-ce que l'apprentissage se passe bien ? Le nombre d'epochs est-il suffisant ?

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 128 neurones avec 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 et du nombre d'epochs pour optimiser les performances du réseau.
- Que se passe-t-il si le pas d'apprentissage est trop grand ? Trop petit ?
- Ajouter une couche de droupout, avec un paramètre de 0.5, entre les deux couches denses. Que représente 0.5 ? Comment sont les résultats ? Optimiser de nouveau le nombre d'épcohs.
- 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. Comment sont les résultats ?

- 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.

V. Annexe

```
def affiche(history):
    # summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    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()
```