Exercice

Réseaux de convolution

Pour installer tensorflow et keras essayez :

<https://www.tensorflow.org/install/pip?lang=python3>

1. Intallez keras !pip install keras dans la console
2. Importez les données grâce à la commande suivante :

|  |
| --- |
| # Plot ad hoc mnist instances from keras.datasets import mnist # load (downloaded if needed) the MNIST dataset (X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data() |

1. Visualisez les quatres premières images du dataset grâce à seaborn
2. Créez un objet num\_pixels égal au nombre de pixels dans une image
3. Utilisez l’attribut np.reshape pour créer des dataset avec une image par ligne et une colonne par pixel
4. Normalisez les valeurs des pixels en divisant tout par 255
5. Transformez la variable cible en 10 variables cibles binaires
6. Importez les packages suivants

|  |
| --- |
| from keras import Sequential from keras.layers import Dense |

1. Définissez une fonction appelée baseline\_model qui ne prend aucun argument
   1. Définissez un objet model égal à Sequential()
   2. Utilisez l’attribut .add pour ajouter à model une couche de neurones de type Dense avec units = num\_pixels, input\_dim = num\_pixels, kernel\_initializer = ‘normal’ et activation = ‘relu’
   3. Ajoutez une deuxième couche de neurones Dense avec units égal au nombre de classes kernel\_initializer = ‘normal’ activation = ‘softmax’
   4. Compilez le modèle à l’aide de l’attribut .compile avec loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy']
   5. Return model
2. Définissez une instance de baseline\_model
3. Estimez le modèle à partir des données train
4. Utilisez l’attribut .evaluate pour évaluer les performances du modèle sur le train et le test
5. Importez les pakages suivants

|  |
| --- |
| import numpy from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense from keras.layers import Dropout from keras.layers import Flatten from keras.layers.convolutional import Conv2D from keras.layers.convolutional import MaxPooling2D from keras.utils import np\_utils from keras import backend as K K.set\_image\_dim\_ordering('th') |

1. Définissez une fonction baseline\_model sans arguments
   1. Créez un objet model = Sequential()
   2. Ajoutez une couche grâce à la fonction .add et et la couche doit être de type Conv2D avec les arguments 32, (5,5), input\_shape = (1,28,28) et activation='relu'
   3. Ajoutez une couche de MaxPooling2D avec pool\_size=(2, 2)
   4. Puis une couche Dropout(0.2)
   5. Puis une couche Flatten()
   6. Une couche Dense(128, activation='relu')
   7. Une couche Dense(num\_classes, activation='softmax')
   8. Compilez le modèle à l’aide de l.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])
   9. return model
2. Définissez une instance de baseline\_model
3. Entraînez le modèle sur le train avec epochs=10, batch\_size=200, verbose=2
4. Évaluez le modèle
5. Créez maintenant un modèle plus complexe avec les spécification suivantes :

|  |
| --- |
| def larger\_model():  # create model  model = Sequential()  model.add(Conv2D(30, (5, 5), input\_shape=(1, 28, 28), activation='relu'))  model.add(MaxPooling2D(pool\_size=(2, 2)))  model.add(Conv2D(15, (3, 3), activation='relu'))  model.add(MaxPooling2D(pool\_size=(2, 2)))  model.add(Dropout(0.2))  model.add(Flatten())  model.add(Dense(128, activation='relu'))  model.add(Dense(50, activation='relu'))  model.add(Dense(num\_classes, activation='softmax'))  # Compile model  model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  return model |

1. Définissez une instance de ce modèle, entraînez le et évaluez le

# Exercice supplémentaire

1. Reprenez les données du challenge 2 et tentez d’améliorer vos performances à l’aide d’un réseau de convolution

|  |
| --- |
| from scipy.ndimage import imread import os import numpy as np '''train walk''' os.chdir("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/train/walk") listfiles = os.listdir() walk\_train = [imread(a) for a in listfiles]  walk\_train\_BW\_2D = [np.mean(a[:,:,:-1],axis = 2) for a in walk\_train] walk\_train\_BW\_2D\_array = np.array(walk\_train\_BW\_2D) shape = walk\_train\_BW\_2D\_array.shape walk\_train\_BW\_2D\_array = np.reshape(walk\_train\_BW\_2D\_array,(shape[0],1,shape[1],shape[2]))  y\_train\_walk = np.zeros(shape = walk\_train\_BW\_2D\_array.shape[0])  '''train run''' os.chdir("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/train/run") listfiles = os.listdir() run\_train = [imread(a) for a in listfiles]  run\_train\_BW\_2D = [np.mean(a[:,:,:-1],axis = 2) for a in run\_train] run\_train\_BW\_2D\_array = np.array(run\_train\_BW\_2D) shape = run\_train\_BW\_2D\_array.shape run\_train\_BW\_2D\_array = np.reshape(run\_train\_BW\_2D\_array,(shape[0],1,shape[1],shape[2]))  y\_train\_run = np.zeros(shape = run\_train\_BW\_2D\_array.shape[0]) + 1  '''test walk''' os.chdir("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/test/walk") listfiles = os.listdir() walk\_test = [imread(a) for a in listfiles]  walk\_test\_BW\_2D = [np.mean(a[:,:,:-1],axis = 2) for a in walk\_test] walk\_test\_BW\_2D\_array = np.array(walk\_test\_BW\_2D) shape = walk\_test\_BW\_2D\_array.shape walk\_test\_BW\_2D\_array = np.reshape(walk\_test\_BW\_2D\_array,(shape[0],1,shape[1],shape[2]))  y\_test\_walk = np.zeros(shape = walk\_test\_BW\_2D\_array.shape[0])  '''test run''' os.chdir("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/test/run") listfiles = os.listdir() run\_test = [imread(a) for a in listfiles]  run\_test\_BW\_2D = [np.mean(a[:,:,:-1],axis = 2) for a in run\_test] run\_test\_BW\_2D\_array = np.array(run\_test\_BW\_2D) shape = run\_test\_BW\_2D\_array.shape run\_test\_BW\_2D\_array = np.reshape(run\_test\_BW\_2D\_array,(shape[0],1,shape[1],shape[2]))  y\_test\_run = np.zeros(shape = run\_test\_BW\_2D\_array.shape[0]) + 1  ''' separate x et y ''' X\_train = np.concatenate((walk\_train\_BW\_2D\_array,run\_train\_BW\_2D\_array)) X\_test = np.concatenate((walk\_test\_BW\_2D\_array,run\_test\_BW\_2D\_array)) y\_train = np.concatenate((y\_train\_walk,y\_train\_run)) y\_test = np.concatenate((y\_test\_walk,y\_test\_run)) import keras as k y\_train = k.utils.to\_categorical(y\_train) y\_test = k.utils.to\_categorical(y\_test)  '''OBJECTIF MEILLEUR SCORE POSSIBLE DE PREDICTION RUN WALK SUR LES DONNEES DE TEST''' import numpy from keras.models import Sequential from keras.layers import Dense from keras.layers import Dropout from keras.layers import Flatten from keras.layers.convolutional import Conv2D from keras.layers.convolutional import MaxPooling2D from keras.utils import np\_utils from keras import backend as K K.set\_image\_dim\_ordering('th')  num\_classes = y\_test.shape[1] |

|  |
| --- |
| def first\_model():  # create model  model = Sequential()  model.add(Conv2D(32, (5, 5), input\_shape=(1, 224, 224), activation='relu'))  model.add(MaxPooling2D(pool\_size=(4, 4)))  model.add(Dropout(0.2))  model.add(Flatten())  model.add(Dense(16, activation='relu'))  model.add(Dense(num\_classes, activation='softmax'))  # Compile model  model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  return model |

|  |
| --- |
| # build the model model = first\_model() # Fit the model model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=10, batch\_size=None, verbose=2) # Final evaluation of the model scores = model.evaluate(X\_test, y\_test, verbose=0) print("CNN Error: %.2f%%" % (100-scores[1]\*100)) |

|  |
| --- |
| import numpy as np # linear algebra import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv) import keras from keras.models import \* from keras.layers import \* from keras.losses import \* from keras.callbacks import \* from keras.applications.vgg16 import VGG16 from keras.applications.inception\_v3 import InceptionV3 import skimage from skimage.color import rgb2gray import sklearn from sklearn.metrics import \* from imageio import imread import seaborn as sns import matplotlib.pyplot as plt import glob  test = pd.DataFrame() test['file'] = glob.glob("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/test/run/\*")+glob.glob("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/test/walk/\*") test['label'] = [ 1 for \_ in glob.glob("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/test/run/\*")]+[0 for \_ in glob.glob("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/test/walk/\*")]  train = pd.DataFrame() train['file'] = glob.glob("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/train/run/\*")+glob.glob("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/train/walk/\*") train['label'] = [ 1 for \_ in glob.glob("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/train/run/\*")]+[0 for \_ in glob.glob("C:/Users/charl/OneDrive/Documents/jedha/full\_time\_exo/S6/challenge/train/walk/\*")] train = train.sample(frac=1).reset\_index(drop=True) train.head()    def make\_model():  transfer\_model = InceptionV3(include\_top=False, weights='imagenet', input\_tensor=None, input\_shape=(None,None,3), pooling='avg', classes=1000)  model = Sequential()  model.add(InputLayer((None,None,3)))  model.add(transfer\_model)  model.add(Dropout(0.5))  model.add(Dense(1,activation='sigmoid'))  return transfer\_model, model  transfer\_model, model = make\_model() model.summary()  def image\_gen(files, labels, batch\_size=10, randomized=False, random\_seed=1):  rng = np.random.RandomState(random\_seed)  img\_batch = []  label\_batch = []  while True:  indices = np.arange(len(files))  if randomized:  rng.shuffle(indices)  for index in indices:  img = imread(files[index])[:,:,0:3]/255  label = labels[index]  img\_batch.append(img)  label\_batch.append(label)  if len(img\_batch) == batch\_size:  yield np.array(img\_batch), np.array(label\_batch)  img\_batch = []  label\_batch = []    if len(img\_batch) > 0:  yield np.array(img\_batch), np.array(label\_batch)  img\_batch = []  label\_batch = []  transfer\_model.trainable=False model.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['acc']) batch\_size=100 epochs=50 history = model.fit\_generator(image\_gen(train['file'], train['label'], batch\_size=batch\_size, randomized=True, random\_seed=1),  steps\_per\_epoch=int(np.ceil(len(train)/batch\_size)),  epochs=epochs,  validation\_data=image\_gen(test['file'], test['label'], batch\_size=batch\_size, randomized=True),  validation\_steps=int(np.ceil(len(test)/batch\_size)),  callbacks=[EarlyStopping(monitor='val\_loss', patience=3, verbose=0), ModelCheckpoint(filepath='./weights.hdf5', monitor='val\_loss', verbose=0, save\_best\_only=True)],  verbose=2,  ) model.load\_weights('weights.hdf5') |

# Exercice supplémentaire 2

1. Si vous avez encore du temps intéressez vous au problème kaggle suivant qui consiste à classer des oeuvres d’art par catégorie : [kagge\_art\_data](https://www.kaggle.com/thedownhill/art-images-drawings-painting-sculpture-engraving)