TP N°4 : Réseau de neurones convolutif

L'objectif de ce TP est de :

- Apprendre à construire et à entraîner un modèle d'un réseau de neurones convolutif (en anglais : *Convolutional Neural Networks*, CNN) pour la classification d'images.
- Évaluer les performances d'un réseau de neurones en utilisant des métriques telles que la précision, la prédiction et la perte.
- Visualiser les cartes de caractéristiques extraites à partir d'une image donnée.

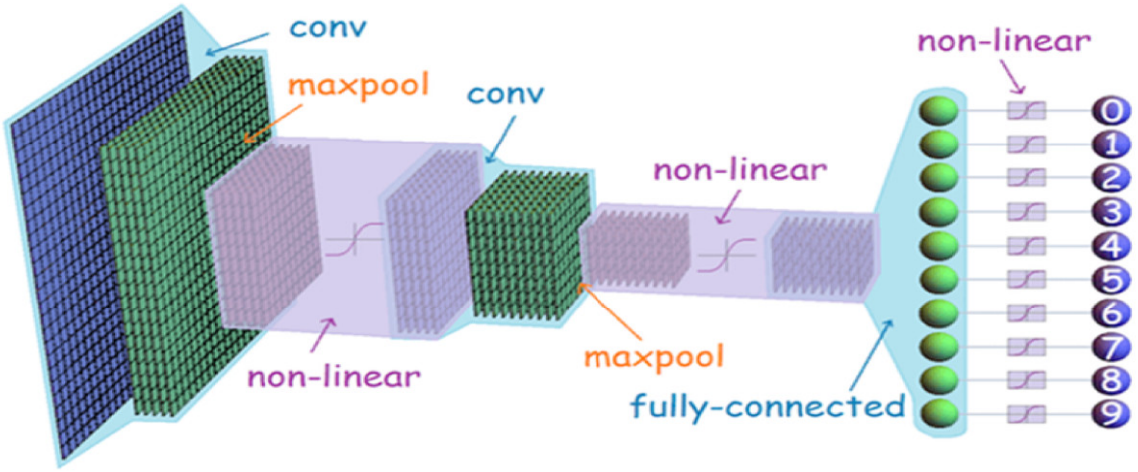
Matériel :

- TP4data.mat - Jeu de données 2.
- model.mat - Modèle CNN pré-entraîné.
- Matlab® 2017 ou version ultérieure.
- Biblothéque Keras.

1 Partie théorique

Un réseau de neurones convolutif (CNN) est composé d'une ou plusieurs couches convolutives (souvent avec une étape de sous-échantillonnage) suivies d'une ou plusieurs couches entièrement connectées, comme dans un réseau de neurones multicouches standard. L'architecture d'un CNN est conçue pour tirer parti de la structure 2D d'une image d'entrée (ou d'une autre entrée $2D$ telle qu'un signal vocal). Un autre avantage des réseaux CNN est qu'ils sont plus faciles à former et qu'ils ont beaucoup moins de paramètres que les réseaux entièrement connectés avec le même nombre d'unités cachées.

Figure 1 Modèle de réseau de neurones convolutif.



Pour définir la structure d'un CNN, plusieurs paramètres doivent être pris en compte, tels que

- N est le nombre total de paramètres d'une couche de convolution donnée
- K est la taille du noyau (kernel)
- C_{in} est le nombre de canaux d'entrée (input channels)
- C_{out} est le nombre de canaux de sortie (output channels)
- P est la taille du rembourrage (padding)
- S est le pas (stride)
- O est la taille de la sortie d'une couche donnée
- I est la taille de l'entrée d'une couche donnée
- Ps est la taille de la fenêtre de pooling (pool size)
- F est le nombre de neurones dans la couche entièrement connectée ("Fully Connected", FC).

En fonction des paramètres ci-dessus

Q.1 : Donner la taille de la sortie d'une couche de convolution donnée O

Q.2 : Donner la taille de la sortie d'une couche de pooling

Q.3 : Donner le nombre total de paramètres d'une couche de convolution N

Q.4 : Donner le nombre total de paramètres d'une couche FC connectée à une couche de convolution

Q.5 : Donner le nombre total de paramètres d'une couche FC connectée à une couche de FC

2 Partie pratique

Dans le TP précédent, nous avons implémenté la propagation avant pour les réseaux neuronaux et l'avons utilisée pour prédire les chiffres manuscrits à l'aide des poids prédéfinis. Dans le même contexte, ce TP vous permettra également d'implémenter et d'entraîner un réseau de neurones convolutionnel (CNN) sur la base de données MNIST. Nous allons d'abord préparer les données en les divisant en ensembles d'entraînement, et de test, puis nous allons définir la structure du CNN en utilisant différentes couches. Ensuite, nous allons compiler le modèle en utilisant la fonction de Matlab "trainNetwork" avec des options d'entraînement spécifiées. Enfin, nous évaluerons la performance du modèle sur l'ensemble de test en utilisant la fonction "classify" et en calculant la précision.

2.1 Jeu de données

Nous allons travailler sur la base de données MNIST qui contient 60000 images en niveaux de grille de résolution 28×28, représentant les 10 chiffres de 0 à 9, ainsi qu'un jeu de test de 10000 images. Tout d'abord, chargeons ce jeu de données. Le code suivant charge l'ensemble de données et trace un échantillon

```
clear all; close all; clc
load TP4data.mat
ch = randperm(60000,20);
for j = 1:20
subplot(4,5,j);
imshow(XTrain(:,:,:,ch(j)));
```

```
title(sprintf('\n %c \n', yTrain(ch(j))));
end
```

2.2 Implémenter et entraîner un CNN

Dans cette tâche, nous allons construire une architecture CNN de base en utilisant la boîte à outils d'apprentissage profond de Matlab. Pour ce faire, nous allons définir les couches du CNN à l'aide de la fonction "layerGraph". Commençons par une architecture simple, telle que "Layers", qui consiste en une série de couches convolutives, de couches de pooling et de couches entièrement connectées.

```
layers = [
imageInputLayer([28 28 1])

convolution2dLayer(5,8)
batchNormalizationLayer
reluLayer

maxPooling2dLayer(2,'Stride',2)

convolution2dLayer(3,16,'Padding','same')
batchNormalizationLayer
reluLayer

maxPooling2dLayer(2,'Stride',2)

convolution2dLayer(3,32,'Padding','same')
batchNormalizationLayer
reluLayer

fullyConnectedLayer(10)
softmaxLayer
classificationLayer];

options = trainingOptions('sgdm', ...
'MaxEpochs',5);
rng('default') %Pour la reproductibilite
```

Q.1 : Combien de couches possède ce modèle CNN ?

Q.2 : Combien de neurones y a-t-il dans la dernière couche du modèle ?

Q.3 : Combien de neurones y a-t-il dans la première couche entièrement connectée ?

Q.4 : Combien de filtres de convolution sont utilisés dans la première couche de convolution ?

Q.5 : Quelle est la taille du pool utilisé dans la première couche de pooling ?

Q.6 : De quelle manière les sorties de la dernière couche de convolution sont-elles connectées à la couche entièrement connectée ?

Maintenant, vous allez lancer l'entraînement du modèle en utilisant les données d'entraînement. Vous pouvez utiliser la fonction "trainNetwork" de Matlab pour entraîner le modèle. Vous allez visualiser l'évolution de la fonction de perte et de la précision sur les ensembles d'entraînement et de validation pour suivre la progression de l'apprentissage.

Ensuite, nous allons lancer l'apprentissage des paramètres.

```
model = trainNetwork(XTrain,yTrain,layers,options);
%save('model.mat','model'); Sauvegarder le modele
```

Q.7 : Quelle est la fonction de perte utilisée pour l'entraînement du modèle ?

Q.8 : A quoi correspond l'erreur au début de la première époque ? Comment s'interprète-t-elle ?

Q.9 : Quel est l'optimiseur utilisé pour entraîner le modèle ?

R [**Note d'implémentation** :] Il est important de sauvegarder le modèle d'un CNN afin de pouvoir l'utiliser ultérieurement. En Matlab, la fonction save() permet de sauvegarder le modèle dans un fichier MAT, tandis que la fonction load() permet de charger un modèle existant depuis un fichier.

2.3 Évaluer la précision sur l'ensemble de données de test

Maintenant que nous avons entraîné le modèle, nous pouvons lui demander de faire des prédictions sur un ensemble de données de test qu'il n'a jamais vu auparavant. Dans cet exemple, la matrice XTest comprend notre ensemble de données de test. Pour évaluer la précision, nous pouvons vérifier si les prédictions du modèle correspondent aux labels provenant de la matrice yTest.

```
Pred = classify(model,XTest);
fprintf('\nPrecision du modele: %f\n', mean(double(Pred == yTest)) *
    100);
```

Q.1 : Donner la valeur de la précision ?

Q.2 : Vérifier votre prédiction en visualisant la sortie pour un chiffre aleatoire de XTest

```
rp = randperm(10000,20);
for j = 1:20
subplot(4,5,j);
imshow(XTest(:,:,:,rp(j)));
title(char(Pred(rp(j))))
end
```

2.4 Visualisation des couches cachées

La visualisation de carte de caractéristiques est un outil essentiel pour comprendre comment les réseaux de neurones convolutifs traitent les images. Elle permet d'identifier les zones de l'image importantes pour la classification, les motifs appris par le réseau, et de diagnostiquer les erreurs de classification et améliorer les performances du modèle. Pour visualiser ces cartes, il est courant d'afficher la sortie de la couche de convolution avant l'activation pour une image donnée. Cette

sortie de couche contient des informations sur les motifs de l'image détectés par les différents filtres de la couche de convolution. Le code suivant affiche les quatre premiers filtres d'une couche donnée

```
NCouche=....;% Numero de couche ( model.Layers)
clear filters  image Img_flt
m=size(XTest,4);
rp = randperm(m,1);
image=XTest(:,:,:,rp);
filters = model.Layers(NCouche).Weights;% Extraire les poids de la
   nieme couche de convolution
figure;% Afficher les quatres premiers filtres
subplot(2,2,1);
imshow(filters(:,:,1,1),[]);
title('Filtre 1');
subplot(2,2,2);
imshow(filters(:,:,1,2),[]);
title('Filtre 2');
subplot(2,2,3);
imshow(filters(:,:,1,3),[]);
title('Filtre 3');
subplot(2,2,4);
imshow(filters(:,:,1,4),[]);
title('Filtre 4');
```

Q.1 : Afficher les filtres de premiere couche de convolution, et la derniere couche de convolution.

Afin d'approfondir sur le fonctionnement de couche de convolution nous disposons ce code

```
Img_flt = zeros(size(image,1), size(image,2), size(filters,4));
for i = 1:size(filters,4)
Img_flt(:,:,i) = conv2(image, filters(:,:,1,i), 'same');
end
figure;
montage(mat2gray(Img_flt))
```

Q.2 : Quel est le résultat de cette opération de convolution 2D entre une image et un filtre spécifique ?

Q.3 : Comment la carte de caractéristiques résultante est-elle calculée à partir de cette opération de convolution ?

Q.4 : Que signifie la chaîne de caractères "same" dans cette fonction ?

Passons maintenant à la tâche de visualisation de carte de caractéristiques. Cette tâche consiste à afficher les activations des neurones de la couche de convolution pour une image d'entrée donnée. L'intérêt de cette tâche est de mieux comprendre comment le réseau de neurones convolutif traite les images et de diagnostiquer les erreurs de classification. Le code suivant affiche les cartes de caractéristiques d'une couche donnée

```
clc
clear featureMaps image
m=size(XTest,4);
```

```
rp = randperm(m,1);
image=XTest(:,:,:,rp);
% Extraire les cartes de caracteristiques de la couche de convolution
Couche = '....'; % Nom de couche ( model.Layers)
featureMaps = activations(model, image, Couche);
% Afficher les cartes de caract\U{e9}ristiques
figure;
montage(featureMaps);
title(sprintf('Cartes de caracteristiques de la couche %s', Couche));
```

Q.5 : Afficher l'activation de premiere couche de convolution, et la derniere couche de convolution.

Q.6 : Comment les cartes de caractéristiques évoluent-elles au fur et à mesure que l'on progresse dans les différentes couches du modèle ?

3 HomeWork

Soit le modèle CNN (net) qui possède les spécifications suivantes :

- 2 couches de convolution avec 32 filtres de taille 3×3 pour la première couche et 64 filtres de taille 3×3 pour la seconde couche
- Une couche de pooling max de taille 2×2 après chaque couche de convolution
- Un padding de valeur "same" .
- Un stride de valeur 1 pour les convolutions et de valeur 2 pour le pooling

Dans cette partie, nous essayons de tester votre ce modèle sur une image quelconque. Pour ce faire il faut :

Q.1 : Charger l'image que vous souhaitez tester et la prétraiter en fonction des prétraitements effectués sur les images d'entraînement. Par exemple, redimensionner l'image à la taille requise pour le modèle et normaliser ses valeurs de pixel.

Q.2 : Passer l'image prétraitée dans le modèle chargé à l'aide d'une fonction de prédiction.

Q.3 : Afficher l'image d'origine avec la classe prédite à l'aide de la fonction imshow et en utilisant le nom de la classe correspondant à l'indice trouvé précédemment.

Q.4 : Afficher les 4 filtres de premiere couche de convolution, et la derniere couche de convolution.

Q.5 : Afficher l'activation de premiere couche de convolution, et la derniere couche de convolution.

4 Tâches optionnelles

Dans cette section, nous avons inclus quelques codes Python™ (facultatifs). Ils vous aideront à approfondir votre compréhension du TP, et si vous êtes en mesure de le faire, nous vous encourageons à les compléter également.

4.1 Chargement des bibliothèque

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import keras
from keras.models import␣
↪Sequential, Model, load_model
from keras.layers import␣
↪Activation, Dense, Dropout
from keras.layers import Conv2D,␣
↪MaxPooling2D, AveragePooling2D,␣
↪Flatten, MaxPool2D
from keras.utils import␣
↪to_categorical, plot_model
from keras.datasets import mnist
from keras.layers import␣
↪BatchNormalization
from keras.optimizers import SGD
from keras import backend as K
```

4.2 Jeu de données

[ ]:
```
# charger le jeu de données MNIST
(XTrain, yTrain), (XTest, yTest) =␣
↪mnist.load_data()
# déterminer le nombre d'étiquettes
num_labels = len(np.unique(yTrain))
yTest_labels=yTest
# convertir en vecteur
yTrain = to_categorical(yTrain)
yTest = to_categorical(yTest)
# dimensions de l'image d'entrée
image_size = XTrain.shape[1]
# redimensionner et normaliser
XTrain = np.reshape(XTrain,[-1,␣
↪image_size, image_size, 1])
XTest = np.reshape(XTest,[-1,␣
↪image_size, image_size, 1])
XTrain = XTrain.astype('float32') /␣
↪255
XTest = XTest.astype('float32') /␣
↪255

# paramètres du réseau
input_shape = (image_size,␣
↪image_size, 1)
dropout = 0.2
```

4.3 Afficher les données

[ ]:
```
m = XTest.shape[0]
indices = np.random.
↪randint(0,m,1500)
fig = plt.figure(figsize = (9,6))

for i in range (36):
img = XTest[indices[i]].
↪reshape(28,28,order='F')
img2 = yTest_labels[indices[i]]
fig.add_subplot(6,6,i+1)
plt.title(img2)
plt.imshow(img,cmap='gray_r')
plt.axis("off")
plt.subplots_adjust(hspace = 0.5)
```

4.4 Implémenter et entraîner un CNN

[ ]:
```
model = Sequential()

model.add(Conv2D(filters=8,
kernel_size=(5,5),
input_shape=input_shape))
convout1 = Activation('relu')
model.add(convout1)

model.add(BatchNormalization())

convout2 =␣
↪MaxPooling2D(pool_size=(2,␣
↪2),strides=(2, 2))
model.add(convout2)
model.add(BatchNormalization())

model.add(Conv2D(filters=16,
kernel_size=(3, 3),
activation='relu',
input_shape=input_shape))


model.add(BatchNormalization())

model.
↪add(MaxPooling2D(pool_size=(2,␣
↪2),strides=(2, 2)))
#model.add(BatchNormalization())

model.add(Conv2D(filters=32,
kernel_size=(3, 3),
activation='relu',
```

```
input_shape=input_shape))

model.add(BatchNormalization())

model.add(Flatten())
        # dropout added as␣
↪regularizer
model.add(Dropout(dropout))
# output layer is 10-dim one-hot␣
↪vector

model.add(Dense(num_labels))
#model.add(BatchNormalization())
model.add(Activation('softmax'))
```

[ ]:
```
model.summary()
model.layers
```

[ ]:
```
model.
↪compile(loss='categorical_crossentropy
optimizer='sgd',
metrics=['accuracy'])
# train the network
history =model.fit(XTrain, yTrain,␣
↪epochs=5)

loss, acc = model.evaluate(XTest,␣
↪yTest)
```

[ ]:
```
print("\nPrécision du modéle: %.
↪1f%%" % (100.0 * acc))
```

[ ]:
```
print(history.history.keys())
```

[ ]:
```
YPredict = np.argmax(model.
↪predict(XTest),axis=1)
```

[ ]:
```
m = XTest.shape[0]
indices = np.random.
↪randint(0,m,10000)
fig = plt.figure(figsize = (9,6))
for i in range (36):
img = XTest[indices[i]].
↪reshape(28,28,order='F')
img2 = YPredict[indices[i]]
fig.add_subplot(6,6,i+1)
plt.title(YPredict[indices[i]])
plt.imshow(img,cmap='gray_r')
plt.axis("off")
plt.subplots_adjust(hspace = 0.5)
```

4.5 Visualisation des couches cachées

4.5.1 La carte des caractéristiques

[ ]:
```
layer_outputs = [layer.output for␣
↪layer in model.layers]
activation_model = Model(inputs =␣
↪model.input, outputs =␣
↪layer_outputs)
#Image 5 dans notre cas
activations = activation_model.
↪predict(XTest[4].reshape(-1, 28,␣
↪28, 1))
```

[ ]:
```
layer_names = []
for layer in model.layers[:12]:
layer_names.append(layer.name) #␣
↪Names of the layers, so you can␣
↪have them as part of your plot

images_per_row = 8

for layer_name, layer_activation in␣
↪zip(layer_names, activations): #␣
↪Afficher les cartes des␣
↪caractéristiques
n_features = layer_activation.
↪shape[-1] # Nombre d'éléments dans␣
↪la carte des éléments
size = layer_activation.shape[1] #␣
↪La carte des caractéristiques a␣
↪les dimensions(1, size, size,␣
↪n_features).
n_cols = n_features //␣
↪images_per_row # Mosaïque des␣
↪canaux d'activation dans cette␣
↪matrice
display_grid = np.zeros((size *␣
↪n_cols, images_per_row * size))
for col in range(n_cols): # Les␣
↪filtres sont regroupés dans une␣
↪grande grille horizontale
for row in range(images_per_row):
channel_image = layer_activation[0,
:, :,
col * images_per_row + row]
channel_image -= channel_image.
↪mean() # Post-traitement de la␣
↪caractéristique pour la rendre␣
↪visuellement acceptable
```

```
channel_image /= channel_image.std()
channel_image *= 64
channel_image += 128
channel_image = np.
↪clip(channel_image, 0, 255).
↪astype('uint8')
display_grid[col * size : (col + 1)␣
↪* size, # Afficher la grille
row * size : (row + 1) * size] =␣
↪channel_image
scale = 1. / size
plt.figure(figsize=(scale *␣
↪display_grid.shape[1],
scale * display_grid.shape[0]))
plt.title(layer_name)
plt.grid(False)
plt.imshow(display_grid,␣
↪aspect='auto', cmap='viridis')
```

4.5.2 Visualiser les fitres

[ ]:
```
filters , bias = model.layers[5].
↪get_weights()
```

[ ]:
```
f_min, f_max = filters.min(),␣
↪filters.max()
filters = (filters - f_min) /␣
↪(f_max - f_min)
```

[ ]:
```
n_filters =6
ix=1
fig = plt.figure(figsize=(20,15))
for i in range(n_filters):
# get the filters
f = filters[:,:,:,i]
for j in range(1):
# subplot for 6 filters and 3␣
↪channels
plt.subplot(n_filters,3,ix)
plt.imshow(f[:,:,j] ,cmap='gray')
ix+=1
#plot the filters
plt.show()
```

[ ]:
```
model.save("model.h5")
```