TP4 – CNN avec peu de données – Auto-encodeur

Ce TP sera réalisé sous tensorFlow2 et keras. La documentation pourra être trouvée sous https://www.tensorflow.org/api_docs/python/tf/keras/
Il se fera sous google collab. Pour accélérer les calculs, modifier les paramètres par défaut :
Execution-> modifier le type d'execution ->GPU

1. Transfert learning

1.1. Chargement et mise en forme des données

On travaille maintenant sur la base de données « caltech101 » composée de 101 classes d'objets. Parcourir cette base avec l'explorateur de fichiers et choisir 4 classes d'objets à reconnaitre. On fera attention à ne pas choisir de classes avec un grand nombre d'image de manière à limiter les temps de calcul (entre 100 et 150 images au total). Que remarquez-vous quant à la taille des images ? Que remarquez-vous quant au fond des objets ? Quant à l'orientation des objets ?
Ne conserver que les 4 répertoires choisis et les compresser dans Data.zip en conservant la structure de répertoire
Uploader le fichier Data.zip et décompresser le avec :

```
!unzip Data.zip
```

Chargez les images en utilisant le code :

```
import numpy as np
import matplotlib.pyplot as plt
import os
import random
import time
np.random.seed(123)  # for reproducibility
from sklearn.metrics import confusion_matrix
from utilitaire import affiche
import tensorflow as tf
from tensorflow.keras import Sequential, Model, Input
from tensorflow.keras.layers import Dense,  Flatten, Dropout, Convolution2D,
MaxPooling2D

# Chargement et mise en forme des données
DATA_PATH = 'Data'
categories=["accordion", "anchor", "barrel", "binocular"]
for i in range(len(categories)):
    categories[i]=DATA_PATH + '/'+categories[i]

data = []
for c, category in enumerate(categories):
    images = [os.path.join(dp, f) for dp, dn, filenames in os.walk(category)
            for f in filenames if os.path.splitext(f)[1].lower() in ['.jpg', '.png', '.jpeg']]
    for img_path in images:
            img = tf.keras.preprocessing.image.load_img(img_path, target_size=(224, 224))
            x=np.array(img)
```

```
            x = np.expand_dims(x, axis=0)
            data.append({'x': np.array(x[0]), 'y':c})

num_classes = len(categories)
random.shuffle(data)

#create train / val / test split
train_split = 0.7
idx_train = int(train_split * len(data))
train = data[:idx_train]
test = data[idx_train:]

#separate data and labels
X_train, y_train = np.array([t['x'] for t in train]), [t['y'] for t in train]
X_test, y_test = np.array([t['x'] for t in test]), [t['y'] for t in test]

#normalize data
X_train = X_train.astype("float32") / 255.0
X_test = X_test.astype("float32") / 255.0

#convert labels to one-hot vectors
Y_train = tf.keras.utils.to_categorical(y_train, num_classes)
Y_test = tf.keras.utils.to_categorical(y_test, num_classes)

print('finished loading',len(data),'images from',num_classes,'categories')
print('train / test split:',  len(X_train), len(X_test))
print('training data shape: ', X_train.shape)
print('training label shape: ', len(y_train))
```

Questions

- Quelle est la taille de X_train ? Comment a-t-on fait ?
- Quelle est la taille de y_train ? de Y_train ?

1.2. Apprentissage from scratch d'un modèle convolutionnel

Apprenez à reconnaitre les classes en utilisant un réseau convolutionnel (basé vous sur celui étudié lors du dernier TP. On ajoutera après chaque couche convolutionnelle un maxpooling (2,2) avec un stride de 2 de manière à diminuer le nombre de paramètres du réseau.

Questions

- Combien y a-t-il de paramètres à estimer ?
- Comment est le temps d'apprentissage du réseau ?
- Quel est le comportement du réseau ?
- Essayer d'ajouter des couches pour diminuer le nombre de paramètres à estimer.
- Conclusion sur l'apprentissage.

1.3. Transfert learning

Afin d'éviter les problèmes précédents, on va utiliser le réseau VGG16 appris sur imagenet (4,197,122 d'images, 21841 classes). Pour cela, taper :

```
input_tensor = Input(shape=(224,224,3))
model = tf.keras.applications.VGG16(weights='imagenet',
                    include_top=True,
                    input_tensor=input_tensor)
model.summary()
```

On va extraire toutes les couches de ce réseau, sauf les dernières couches entièrement connectées. Et on va utiliser ce réseau pour extraire les features des images :

```
input_tensor = Input(shape=(224,224,3))
model = tf.keras.applications.VGG16(weights='imagenet',
                    include_top=False,
                    input_tensor=input_tensor)model.summary()
#extraction des features
feature_train = model.predict(X_train)
feature_test = model.predict(X_test)
```

Construire un réseau MLP à une couche cachée qui prend en entrée ces caractéristiques et prédit la classe des images.

Questions

- Quelle est la taille des caractéristiques extraites ?
- Combien y a-t-il de paramètres à apprendre ?
- Comment se passe l'apprentissage ?
- Conclusion sur les résultats ?

2. Auto-encodeur

On va maintenant s'intéresser à l'apprentissage non supervisé en mettant en place, dans un premier temps, un auto-encodeur pour débruiter les images.

2.1. Chargement et mise en forme des données

Pour cela, nous allons retravailler sur la base de données MNIST et reprendre toute la partie chargement et mise en forme des données du TP III, section I.
On bruitera ensuite les données d'apprentissage et de test et on les visualisera :

```
X_train_noise = X_train + 0.2 * np.random.normal(loc=0.0, scale=1.0, size=X_train.shape)
X_test_noise = X_test + 0.4 * np.random.normal(loc=0.0, scale=1.0, size=X_test.shape)

X_train_noise = np.clip(X_train_noise, 0.0, 1.0)
X_test_noise = np.clip(X_test_noise, 0.0, 1.0)

# Display the train data and a version of it with added noise
for i in range(5):
  plt.subplot(2,5,i+1)
  plt.imshow(X_train[i,:].reshape([28,28]), cmap='gray')
  plt.axis('off')
  plt.subplot(2,5,i+6)
```

```
  plt.imshow(X_train_noise[i,:].reshape([28,28]), cmap='gray')
  plt.axis('off')
plt.show()
```

Questions

- 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 ?
- A quoi sert la fonction np.clip ?

2.2. Définition du réseau

On va maintenant définir le réseau avec la partie encodeur :

- une couche convolutionnelle composée de 32 filtres 3x3 suivie de la fonction d'activation RELU
- un max pooling de taille 3x3 avec un stride de 2
- une couche convolutionnelle composée de 32 filtres 3x3 suivie de la fonction d'activation RELU
- un max pooling de taille 3x3 avec un stride de 2

Et la partie décodeur :

- une couche de convolution transposée composée de 32 filtres 3x3, un stride de 2 suivie de la fonction d'activation RELU
- une couche de convolution transposée composée de 32 filtres 3x3, un stride de 2 suivie de la fonction d'activation RELU
- une couche de convolution dont on déterminera :
    - le nombre de filtres permettant de reconstruire une image niveau de gris de taille 28x28
    - la fonction d'activation permettant de reconstruire l'image

Questions

- Quelle est la taille du tenseur au niveau du backbone ?
- A quoi sert la fonction np.clip ?

2.3. Apprentissage

Réaliser l'apprentissage avec Adam, 50 epochs (battch de 32) et une fonction perte MSE :

```
opt = keras.optimizers.Adam(learning_rate=0.0005)
autoencoder.compile(loss="mse", optimizer=opt)
```

Réaliser le débruitage des données de test et afficher les données bruitées et débruitées.

Questions

- Quelle fonction perte utiliser ? Pourquoi ?

3. Variational Auto-encodeur

Dans cette dernière partie, nous nous intéressons à la génération de données avec un auto-encodeur variationnel et allons travailler sur les données de MNIST

3.1. Chargement et mise en forme des données

Reprendre toute la partie chargement et mise en forme des données du TP III, section I.

3.2. Définition du modèle

3.2.1. Encodeur

On va définir un réseau propre pour réaliser l'encodage. Il prendra en entrée des images 28x28x1 et renverra un vecteur de taille 2 et sera composé de :

- une couche convolutionnelle composée de 32 filtres 3x3 suivie de la fonction d'activation RELU
- un max pooling de taille 3x3 avec un stride de 2
- une couche convolutionnelle composée de 32 filtres 3x3 suivie de la fonction d'activation RELU
- un max pooling de taille 3x3 avec un stride de 2
- une mise à plat des données
- une couche dense de taille 16
- mu = Dense(2)(x)
- log_variance = Dense(2)(x)
- encoder_output = tf.keras.layers.Lambda(sampling)([mu, log_variance])

avec :

```
def sampling(mu_log_variance):
    mu, log_variance = mu_log_variance
    epsilon =keras.backend.random_normal(shape=keras.backend.shape(mu), mean=0.0, stddev=1.0)
    random_sample = mu +keras.backend.exp(log_variance/2) * epsilon
    return random_sample
```

Définir le modèle encoder qui prend en entrée l'image et renvoie encoder_output puis utiliser encoder.summary() pour déterminer le nombre de paramètres à apprendre dans ce réseau

Questions

- Que représentent les variables mu et log_variance, quelles sont leurs dimensions ?
- Que représente encoder_output ? Comment est-il calculé ?

3.2.2. Décodeur

Le décodeur va reprendre les opérations inverses de l'encodeur. Il prend en entrée un vecteur de taille 2 puis réalise :

- Une couche dense avec 7x7x32 neurones
- Une mise en forme des données sous forme de tenseur (7,7,32) avec la fonction reshape
- une couche de convolution transposée composée de 32 filtres 3x3, un stride de 2 suivie de la fonction d'activation RELU
- une couche de convolution transposée composée de 32 filtres 3x3, un stride de 2 suivie de la fonction d'activation RELU
- une couche de convolution dont on déterminera :
    - le nombre de filtres permettant de reconstruire une image niveau de gris de taille 28x28
    - la fonction d'activation permettant de reconstruire l'image

Définir le modèle decoder qui prend en entrée le vecteur de taille 2 et renvoie une image 28x28 puis utiliser decoder.summary() pour déterminer le nombre de paramètres à apprendre dans ce réseau.

3.2.3. VAE

Définir le VAE qui prend en entrée l'image de taille 28x28, l'encode avec l'encodeur et la décode avec le décodeur

3.2.4. Apprentissage

On réalisera l'apprentissage comme pour l'auto-encodeur :

```
vae.compile(optimizer=keras.optimizers.Adam(learning_rate= 0.0005), loss=loss_func(encoder_mu, encoder_log_variance))

vae.fit(X_train, X_train, epochs=50, batch_size=32, shuffle=True, validation_data=(X_test, X_test))
```

Mais en utilisant une fonction perte spécifique au VAE :

```
def loss_func(encoder_mu, encoder_log_variance):
    def vae_reconstruction_loss(y_true, y_predict):
        reconstruction_loss_factor = 1000
        reconstruction_loss  =keras.backend.mean(tf.keras.backend.square(y_true-y_predict), axis=[1, 2, 3])
        return reconstruction_loss_factor * reconstruction_loss

    def vae_kl_loss(encoder_mu, encoder_log_variance):
        kl_loss = -0.5 * keras.backend.sum(1.0 + encoder_log_variance - keras.backend.square(encoder_mu) - keras.backend.exp(encoder_log_variance), axis=1)
        return kl_loss

    def vae_loss(y_true, y_predict):
        reconstruction_loss = vae_reconstruction_loss(y_true, y_predict)
        kl_loss = vae_kl_loss(y_true, y_predict)

        loss = reconstruction_loss + kl_loss
        return loss

    return vae_loss
```

Questions

- Comment fonctionne la fonction perte, que représentent ses différents termes ?

Réaliser l'encodage/décodage des images de test et afficher le résultat pour quelques images. Choisissez deux nombres entre 0 et 1 et générer l'image correspondante