TP – CNN avec peu de données

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. Reconnaissance avec un CNN

1.1. Chargement et mise en forme des données

On travaille maintenant sur la base de données « caltech101 » (https://docs.ultralytics.com/fr/datasets/classify/caltech101/) composée de 101 classes d'objets parmi lesquels nous avons sélectionné 4 classes pour ce TP. Que remarquez-vous quant à la taille des images ? Au fond des objets ? A l'orientation des objets ?
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))

shape = X_train[0].shape
```

Questions

- Quelle est la taille de X_train ? Comment a-t-on fait ?
- Quelle est la taille de y_train ? de Y_train ? Comment a-t-on, fait pour passer de l'un à l'autre, pourquoi ?

1.2. Apprentissage from scratch d'un modèle convolutionnel

Apprenez à reconnaître les classes en utilisant un réseau convolutionnel (basez-vous sur celui étudié en TP de ROB4). On utilisera dans un premier temps un réseau simple :

```
def CNN(shape):
    inputs = Input(shape)
    x= inputs
    x=Convolution2D(32,(3,3), activation='relu',padding='same')(x)
    x=MaxPooling2D(pool_size=(3, 3), strides=2,padding='same')(x)

    x=Convolution2D(64,(3, 3), activation='relu',padding='same')(x)
    x=MaxPooling2D(pool_size=(3, 3), strides=2,padding='same')(x)

    x=Flatten()(x)
    x=Dense(100, activation='relu')(x)
    outputs=Dense(4, activation='softmax')(x)
```

```
    model = Model(inputs, outputs)
    return model

model = CNN(shape)
model.summary()
```

L'apprentissage sera réalisé avec l'optimiseur Adam.

Questions

- Combien y a-t-il de paramètres à estimer ? Estimer le par le calcul puis faites-le afficher pour vérifier
- Comment est le temps d'apprentissage du réseau ?
- Quel est le comportement du réseau ?

1.3. Amélioration du modèle et des hyperparamètres

Essayez de faire varier le learning rate, le nombre de couches, de filtres, de mettre du dropout, d'augmenter le nombre d'epochs,… de manière à optimiser les résultats.
Poussez l'apprentissage de la meilleure configuration en augmentant le nombre d'epochs. Que se passe-t-il ?

Questions

- Rappeler ce qu'est le dropout. A quoi sert-il ?
- Comment est le temps d'apprentissage du réseau ?
- Quel est le comportement du réseau ?

1.4. Data augmentation

On souhaite réaliser de l'augmentation de données.

```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("vertical"),
])

# Exemple d'augmentation d'une image
plt.figure(figsize=(10, 4))
for i in range(5):
    augmented_img = data_augmentation(tf.expand_dims(X_train[0], 0))
    plt.subplot(1, 5, i+1)
    plt.imshow(augmented_img[0])
    plt.axis("off")
plt.show()

def CNN(shape):
    inputs = Input(shape)
    x= inputs
    x = data_augmentation(x, training=True)
      ….
```

Refaites des apprentissages à partir de la meilleure configuration de la section 1. Essayez plusieurs configurations d'augmentation de données et retenez la meilleure.

Questions

- Rappelez le principe de l'augmentation de données.

- Expliquez quelles sont les méthodes d'augmentation de données qui vous ont paru adéquates.
- Quel est l'effet de l'augmentation de données sur l'apprentissage ?
- Comment sont les résultats?

1.5. Transfer learning

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

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

Questions

- Combien de couches possède ce réseau ?
- Combien y a-t-il de paramètres à apprendre ?
- Que réalise le réseau ?
- Si on découpe ce réseau en une partie « extraction de caractéristiques » et une partie « classification », où s'arrête la première partie ?

On décide d'utiliser ce réseau pour extraire des caractéristiques :

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

1.6. Fine Tuning

Plutôt que d'extraire directement les features entraînées sur une autre base, on construit un seul réseau dont on initialise les poids avec les poids appris sur une autre base. Ainsi, on récupère le réseau VGG (sans les couches entièrement connectées) auquel on concatène le MLP précédent :

```
input_tensor = Input(shape=(224,224,3))
VGG = tf.keras.applications.VGG16(weights='imagenet',
```

```
                    include_top=False,
                    input_tensor=input_tensor)

for layer in VGG.layers:
    layer.trainable = False

#Adding custom Layers
x = VGG.output
predictions = MLP(x) # MLP precedent
# creating the final model
model2 = Model(input_tensor, predictions)
```

Tester le réseau sans relancer l'apprentissage. Qu'obtient-on ?

Relancer l'apprentissage total en dégelant certaines couches du VGG :

Questions

- Combien y a-t-il de paramètres à apprendre ?
- Comment se passe l'apprentissage ?
- Conclusion sur les résultats ?

1.7. Test sur une nouvelle image

On a récupéré l'image « test.jpg » sur internet.

Mettre en place la procédure pour reconnaître cette image.

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

Nous allons travailler sur la base de données MNIST.

```
(X_train, y_train), (X_test, y_test) = load_data()

# 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
```

On bruite ces données (apprentissage et test) et on les visualise de nouveau :

```
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 auto-encodeur afin d'apprendre à débruiter les images. L'entrée sera l'image bruitée et la sortie l'image débruitée.

Partie encodeur :

- une couche convolutionnelle composée de 16 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

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 16 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 ?

2.3. Apprentissage

Réaliser l'apprentissage avec Adam, 20 epochs (batch de 32) et une fonction perte MSE. On vérifiera visuellement le comportement de la fonction perte.
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. 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 = layers.Dense(2, name="mu")(x)
log_var = layers.Dense(2, name="log_var")(x)
z = Sampling()([mu, log_var])
encoder = Model(encoder_inputs, [z, mu, log_var])
```

avec :

```
# Couche personnalisée Sampling avec ajout de la loss KL
class Sampling(layers.Layer):
    def call(self, inputs):
        mu, log_var = inputs
        epsilon = tf.random.normal(shape=tf.shape(mu))
        z = mu + tf.exp(0.5 * log_var) * epsilon

        coeff = 1

        kl_loss = -0.5 * tf.reduce_sum(1 + log_var - tf.square(mu) - tf.exp(log_var), axis=1)
        self.add_loss(coeff*tf.reduce_mean(kl_loss))  # Ajout KL divergence loss au modèle
        return z
```

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 ?
- Que représente KL_loss ?

3.2. 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 16 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.

3.3. VAE

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

3.4. Apprentissage

On réalisera l'apprentissage avec :

```
vae_inputs = Input(shape=(28, 28, 1))
z, mu, log_var = encoder(vae_inputs)
reconstructions = decoder(z)
vae = Model(vae_inputs, reconstructions, name="vae")

vae.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        loss=tf.keras.losses.MeanSquaredError())

history = vae.fit(X_train, X_train, epochs=15, batch_size=128, validation_data=(X_test, X_test))

plt.plot(history.history["loss"], label="train loss")
plt.plot(history.history["val_loss"], label="val loss")
plt.show()
```

Questions

- Comment fonctionne la fonction perte, que représentent ses différents termes ?
- Comparez différentes valeurs de coeff. Conclusion ?

Réaliser l'encodage/décodage des images de test et afficher le résultat pour les 5 premières images avec :

```
for i in range(5):
   plt.subplot(2, 5, i + 1)
```

```
    plt.imshow(X_test[i].reshape(28, 28), cmap="gray")
    plt.axis("off")
    plt.subplot(2, 5, i + 6)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray")
    plt.axis("off")
plt.show()
```

Commenter sur la qualité de la reconstruction.

Faites varier x et y entre -3 et 3 (10 valeurs) et pour chaque couple (x,y), décoder l'image correspondante et afficher là à la position x,y d'une figure (subplot) comme précédemment.

Commenter la qualité de l'espace latent.

Faire varier la variable coeff de la classe Sampling(layers.Layer). Comment agit ce paramètre ? Que lest d'après vous sa valeur optimale ?