Utilisation des techniques CNN en Python

Objectifs du TP

Familiarisation avec les architectures des réseaux de neurones convolutifs et Ieur utilisation en Python pour la classification d'image en utilisant la bibliothèque *keras*.

Les étapes de TP

Etape 01

1. Téléchargement du code et de dataset ; et installation des bibliothèques Keras et TensorFlow :

```
#============================
# Step 1: Installer les paquets
#============================
# On désinstalle Keras au cas où une ancienne version causerait des conflits
!pip uninstall keras -y
# On installe TensorFlow et Keras pour utiliser l'API Keras intégrée à TensorFlow
!pip install tensorflow
!pip install keras

#================================
# Step 2: Import libraries
#================================
import os  # Gestion des chemins et fichiers
from tensorflow.keras.preprocessing.image import ImageDataGenerator  # Pour
charger et augmenter les images
from tensorflow.keras.models import Sequential  # Pour construire un modèle
séquentiel de CNN
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Activation  # Couches CNN
from tensorflow.keras import backend as K  # Pour vérifier le format des images
(channels_first / channels_last)
import matplotlib.pyplot as plt  # Pour tracer les courbes d'apprentissage


#================================
# Step 3: Charger dataset
#================================
# Permet à l'utilisateur d'uploader des fichiers depuis mon ordinateur vers Colab
from google.colab import files
uploaded = files.upload()
```

2. Analyse du code et insertion de commentaires expliquant le rôle de chaque section principale indiquée dans le code :

```
#================================
# Step 4: Section paramétres
#================================
# Taille des images d'entrée
img_width, img_height = 224, 224

# Chemins vers les dossiers de données pour entraînement et validation
train_data_dir = '/content/images/train'
validation_data_dir = '/content/images/test'

# Paramètres d'entraînement
nb_train_samples = 400  # Nombre total d'images d'entraînement
nb_validation_samples = 100  # Nombre total d'images de validation
epochs = 15  # Nombre d'itérations complètes sur le dataset
batch_size = 16  # Nombre d'images traitées simultanément
num_filters = 8  # Nombre de filtres pour la première couche convolutive
filter_size = 3  # Taille du filtre 3x3

# Vérification du format des images selon la configuration Keras
if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)  # Format : (canaux, hauteur, largeur)
else:
    input_shape = (img_width, img_height, 3)  # Format : (hauteur, largeur, canaux)

print("Input shape:", input_shape)  # Affiche la forme attendue des images pour le
CNN

#================================
# Step 5: Section transformation données
#================================
# Générateur d'images pour l'entraînement avec augmentation pour éviter le
surapprentissage
train_datagen = ImageDataGenerator(
    rescale=1./255,         # Normalisation des pixels entre 0 et 1
    shear_range=0.2,        # Application de transformations de cisaillement
    zoom_range=0.2,         # Application de zoom aléatoire
    horizontal_flip=True    # Retour horizontal aléatoire des images
)

# Générateur pour la validation, uniquement normalisation (pas d'augmentation)
test_datagen = ImageDataGenerator(rescale=1./255)

# Chargement des images d'entraînement depuis le dossier et application des
transformations
train_generator = train_datagen.flow_from_directory(
    '/content/v_data/train',
    target_size=(img_width, img_height),
```

```
    batch_size=batch_size,
    class_mode='categorical'  # Classification multi-classes
)

# Chargement des images de validation
validation_generator = test_datagen.flow_from_directory(
    '/content/v_data/test',
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
)

#================================
# Step 6: Section architecture du CNN
#================================
# Définition d'un modèle séquentiel (couches empilées)
model = Sequential()

# Couche convolutive 2D : extraction de caractéristiques
model.add(Conv2D(num_filters, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))  # Fonction d'activation ReLU pour non-linéarité

# Couche de pooling : réduction de la taille des cartes de caractéristiques
model.add(MaxPooling2D(pool_size=(2, 2)))

# Aplatir les cartes de caractéristiques pour les envoyer aux couches denses
model.add(Flatten())

# Couche dense avec 24 neurones et activation ReLU
model.add(Dense(24))
model.add(Activation('relu'))

# Couche de sortie avec 2 neurones (2 classes) et softmax pour probabilité
model.add(Dense(2))
model.add(Activation('softmax'))

# Affichage du résumé du modèle : couches, formes de sortie, nombre de paramètres
print(model.summary())

#================================
# Step 7: Section entrainement du modéle et évaluation
#================================
# Compilation du modèle
# loss = categorical_crossentropy car classification multi-classes
# optimizer = RMSprop, metrics = accuracy pour suivre la précision
model.compile(
    loss='categorical_crossentropy',
```

```
    optimizer='rmsprop',
    metrics=['accuracy']
)

# Entraînement du modèle avec les générateurs d'images
history = model.fit(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size
)

#================================
# Step 8: Section visualisation
#================================
# Tracé des courbes de précision (accuracy) sur les données d'entraînement et de
validation
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='validation')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()
# Tracé des courbes de perte (loss) sur les données d'entraînement et de
validation
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='validation')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()
```

Etape 02

1. Changement de la section de code :

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
```

Par:

```
train_datagen = ImageDataGenerator(
    rescale=1./255
)
```

2. **Ce qui change dans le code**

Avant:

- Utilisation de l'augmentation d'images : cisaillement (shear), zoom (zoom_range), retournement horizontal (horizontal_flip).
- Chaque image d'entraînement est vue sous plusieurs variations aléatoires.
- Effet : le modèle devient plus robuste, apprend à généraliser sur des images légèrement différentes de celles du training set.

Après (modifié):

- Plus d'augmentation, seulement normalisation des pixels (valeurs entre 0 et 1).
- Le modèle voit uniquement les images originales du dataset.
- Effet : le modèle peut atteindre une précision plus élevée sur le training set, mais moins performant sur le test **set** ou de nouvelles images → risque de surapprentissage.

Conclusion:

Supprimer l'augmentation d'images rend le modèle moins robuste et réduit sa capacité de généralisation, même si la précision sur les données d'entraînement peut sembler meilleure.

Etape 03

Nouvelle architecture CNN

- **Plusieurs couches convolutionnelles** pour extraire des caractéristiques plus complexes.
- **Pooling** après chaque convolution pour réduire la taille des cartes de caractéristiques.
- **Dropout** pour réduire le surapprentissage.
- **Dense + softmax** à la fin pour la classification multi-classes.

1. **le code Python/kera**

```
#============================
# Step 1: Installer les paquets
#============================
!pip uninstall keras -y
!pip install tensorflow
!pip install keras
```

```
#================================
# Step 2: Import libraries
#================================
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Activation, Dropout
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt


#================================
# Step 3: Charger dataset
#================================
from google.colab import files
uploaded = files.upload()

#================================
# Step 4: Section paramétres
#================================
img_width, img_height = 224, 224
train_data_dir = '/content/images/train'
validation_data_dir = '/content/images/test'

# Paramètres d'entraînement
nb_train_samples = 400
nb_validation_samples = 100
epochs = 15
batch_size = 16
num_filters = 8
filter_size = 3

# Format des images selon Keras
if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)

print("Input shape:", input_shape)

#================================
# Step 5: Section transformation données
#================================
# Générateur d'images pour l'entraînement avec augmentation pour éviter le
surapprentissage
train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

# Générateur pour la validation, uniquement normalisation (pas d'augmentation)
test_datagen = ImageDataGenerator(rescale=1./255)

# Chargement des images d'entraînement depuis le dossier et application des
transformations
train_generator = train_datagen.flow_from_directory(
    '/content/v_data/train',
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
)

# Chargement des images de validation
validation_generator = test_datagen.flow_from_directory(
    '/content/v_data/test',
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
)
#================================
# Step 6: Section architecture du CNN amélioré
#================================
# Définition du modèle CNN amélioré
model = Sequential()

# --- 1ère couche convolutive ---
model.add(Conv2D(32, (3, 3), padding='same', input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# --- 2ème couche convolutive ---
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# --- 3ème couche convolutive ---
model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# --- Dropout pour réduire le surapprentissage ---
```

```
model.add(Dropout(0.5))

# --- Couche de flattening ---
model.add(Flatten())

# --- Couche dense intermédiaire ---
model.add(Dense(128))
model.add(Activation('relu'))

# --- Dropout supplémentaire ---
model.add(Dropout(0.5))

# --- Couche de sortie ---
model.add(Dense(2))  # 2 classes
model.add(Activation('softmax'))

# Résumé du modèle
print(model.summary())

#=================================
# Step 7: Section entrainement du modéle et évaluation
#=================================
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

history = model.fit(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size
)


#=================================
# Step 8: Section visualisation
#=================================
# Courbes d'accuracy
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='validation')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
```

```
plt.show()

# Courbes de loss
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='validation')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()
```

2. **Les performances en classification de la nouvelle CNN:**

⇨ Précision sur le training set
- ✓ Les valeurs de accuracy sur l'entraînement commencent à 73 % (Epoch 1) et atteignent 96–97 % à la fin (Epoch 12–15).
- ✓ Cela montre que le modèle apprend très bien les images du dataset d'entraînement.

⇨ Précision sur le validation set
- ✓ La val_accuracy varie entre 82 % et 92,7 % au cours des epochs.
- ✓ La meilleure val_accuracy est 92,7 % (Epoch 14).
- ✓ Certaines epochs montrent une baisse de performance (ex. Epoch 12 → 82,29 %), ce qui peut indiquer un début de surapprentissage.

⇨ Perte (loss) et val_loss
- ✓ loss diminue globalement sur l'entraînement (0,603 → 0,109), ce qui est bon.
- ✓ val_loss est plus irrégulier et parfois plus élevé que la loss d'entraînement, ce qui confirme que le modèle s'adapte mieux au training qu'aux nouvelles images.

⇨ Conclusion sur les performances
- ✓ La CNN améliorée a une très bonne capacité à apprendre le dataset d'entraînement (jusqu'à 96 % d'accuracy).
- ✓ La performance sur les données de validation est bonne mais un peu instable, oscillant autour de 85–93 %.
- ✓ Cela suggère que la CNN est efficace pour la classification, mais qu'un peu de régularisation supplémentaire (dropout, data augmentation) pourrait encore améliorer sa robustesse sur de nouvelles images.

Etape 04

CNN travaille uniquement sur la luminance (niveaux de gris)

1. **Conversion des images en niveaux de gris**

Dans le code *Keras* où on utilises *ImageDataGenerator* et *flow_from_directory*, on dois spécifier *color_mode='grayscale'*

```
#================================
# Step 5: Section transformation données
#================================
```

```
# Générateur pour l'entraînement
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    '/content/v_data/train',
    target_size=(img_width, img_height),
    color_mode='grayscale',           # 🔥 GRAYSCALE
    batch_size=batch_size,
    class_mode='categorical'
)

validation_generator = test_datagen.flow_from_directory(
    '/content/v_data/test',
    target_size=(img_width, img_height),
    color_mode='grayscale',           # 🔥 GRAYSCALE
    batch_size=batch_size,
    class_mode='categorical'
)
```

⇨ Cela assure que toutes les images chargées ont 1 canal au lieu de 3.

2. Modification de l'input du modèle CNN

Dans le modèle Sequential, la première couche Conv2D doit avoir input_shape= (hauteur,largeur,1) au lieu de (hauteur, largeur, 3) :

```
num_classes = train_generator.num_classes
model = Sequential()
model.add(Conv2D(32, (3,3), input_shape=(224,224,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2,2)))
model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2,2)))
model.add(Conv2D(128, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2,2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

3. Ajustements éventuels

- Data augmentation : elle reste utile pour éviter le surapprentissage.
- Batch normalization : si tu veux améliorer la convergence sur des images monochromes.
- Resizing / rescaling : inchangé (rescale=1./255 pour normaliser les pixels de 0 à 1).

4. Conclusion

En mode *grayscale*, l'entrée du CNN passe de (224,224,3) à (224,224,1), ce qui réduit la complexité du modèle et force le réseau à apprendre uniquement à partir de la luminance.