LISV Laboratoire d'ingénierie des systèmes de Versailles
UVSQ université PARIS-SACLAY

TP - Keras CNN

Halim Djerroud

révision 0.1

Objectifs

Ce TP a pour but de familiariser les participants avec :
— La construction, l'entraînement et l'évaluation de réseaux convolutifs (CNN) avec Keras.
— L'utilisation de couches convolutionnelles, de pooling et de fully connected.
— La visualisation des performances et des résultats des modèles.
— L'application des CNN pour la classification d'images avec le dataset MNIST.

Prérequis

— Connaissances de base en Python et TensorFlow/Keras.
— Familiarité avec les notions fondamentales des réseaux de neurones convolutifs.
— Installation des bibliothèques `TensorFlow/Keras`, `NumPy`, et `Matplotlib`.

1 Partie 1 : Chargement et exploration des données MNIST

1. Chargement des données

Utilisez `Keras` pour charger le dataset MNIST.

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Charger les données
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalisation des données
x_train = x_train / 255.0
x_test = x_test / 255.0

# Reshape des données pour les CNN
x_train = x_train.reshape((-1, 28, 28, 1))
x_test = x_test.reshape((-1, 28, 28, 1))

# Encodage One-Hot pour les labels
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

2. Exploration des données

Visualisez quelques images du dataset avec leurs labels.

```
import matplotlib.pyplot as plt

# Afficher quelques images
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(x_train[i].reshape(28, 28), cmap='gray')
    plt.title(f"Label: {y_train[i].argmax()}")
    plt.axis('off')
plt.show()
```

2 Partie 2 : Construction d'un réseau convolutif (CNN)

Objectif

Construisez un modèle CNN avec les caractéristiques suivantes :

- Une couche convolutionnelle avec 32 filtres (taille `3x3`) et activation `relu`.
- Une couche de max pooling (taille `2x2`).
- Une deuxième couche convolutionnelle avec 64 filtres (taille `3x3`) et activation `relu`.
- Une couche de flatten pour convertir les cartes de caractéristiques en vecteurs.
- Une couche fully connected avec 128 neurones et activation `relu`.
- Une couche de sortie avec 10 neurones (une par classe) et activation `softmax`.

Code du modèle

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Définir le modèle CNN
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# Compilation du modèle
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

3 Partie 3 : Entraînement et évaluation du modèle

1. Entraînement du modèle

Entraînez le modèle sur les données d'entraînement.

```
# Entraînement
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

2. Évaluation sur les données de test

```
# Évaluation
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")
```

3. Visualisation des performances

Tracez les courbes de perte et de précision.

```
# Courbe de précision
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Précision')
plt.legend()
plt.show()

# Courbe de perte
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Perte')
plt.legend()
plt.show()
```

4 Partie 4 : Optimisation et régularisation

Objectif

Améliorez le modèle en ajoutant des techniques de régularisation telles que :
— Dropout
— Régularisation L2

Exemple avec Dropout

```
from tensorflow.keras.layers import Dropout

# Modèle avec Dropout
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Évaluation après régularisation

Répétez les étapes d'entraînement et d'évaluation pour observer les effets de la régularisation sur les performances du modèle.

5 Exercice : Reconnaissance d'images sur CIFAR-10

Étape 1 : Chargement et préparation des données

Chargez le jeu de données CIFAR-10 et préparez-le pour être utilisé avec un modèle CNN. Effectuez les étapes suivantes :

- Normalisez les images pour que leurs valeurs soient comprises entre 0 et 1.
- Encodez les labels sous forme one-hot.
- Divisez les données en ensembles d'entraînement et de test.

Étape 2 : Construction du modèle

Construisez un modèle CNN ayant les caractéristiques suivantes :

- Deux couches convolutionnelles avec activation `relu` et des filtres de taille `3x3`.
- Deux couches de max pooling avec une taille de fenêtre `2x2`.
- Une couche de flatten pour transformer les cartes de caractéristiques en vecteurs.
- Une couche fully connected avec 128 neurones et activation `relu`.
- Une couche de sortie avec 10 neurones et activation `softmax`.

Étape 3 : Compilation et entraînement

Compilez le modèle avec l'optimiseur `adam` et la fonction de perte `categorical_crossentropy`. Entraînez le modèle sur les données d'entraînement pour un total de 15 époques, avec une validation sur 20% des données d'entraînement.

Étape 4 : Évaluation et analyse

Évaluez les performances du modèle sur le jeu de test. Affichez les métriques suivantes :

- La perte (loss) sur les données de test.
- La précision (accuracy) sur les données de test.

Étape 5 : Visualisation des résultats

- Tracez les courbes de perte et de précision pour les ensembles d'entraînement et de validation.
- Affichez quelques prédictions sur les images de test en indiquant les classes prédites et les classes réelles.

Solution

Étape 1 : Chargement des données

Utilisez `Keras` pour charger le dataset CIFAR-10.

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

# Charger les données
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalisation des données
x_train = x_train / 255.0
x_test = x_test / 255.0
```

```
# Encodage One-Hot pour les labels
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

Étape 2 : Construction du modèle CNN

Créez un modèle CNN avec les caractéristiques suivantes :

- Une première couche convolutionnelle avec 32 filtres (taille `3x3`), activation `relu`, et une couche de max pooling (`2x2`).
- Une deuxième couche convolutionnelle avec 64 filtres (taille `3x3`), activation `relu`, et une couche de max pooling (`2x2`).
- Une couche de flatten pour aplatir les caractéristiques.
- Une couche fully connected avec 128 neurones et activation `relu`.
- Une couche de sortie avec 10 neurones (une par classe) et activation `softmax`.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Définir le modèle CNN
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# Compilation du modèle
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Étape 3 : Entraînement du modèle

Entraînez le modèle sur les données d'entraînement.

```
# Entraînement
history = model.fit(x_train, y_train,
                    epochs=15,
                    batch_size=32,
                    validation_split=0.2)
```

Étape 4 : Évaluation et Visualisation des résultats

Évaluez les performances du modèle et tracez les courbes de perte et de précision.

```
# Évaluation
loss, accuracy = model.evaluate(x_test, y_test)
```

```
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")

# Courbes de précision et de perte
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Précision')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Perte')
plt.legend()
plt.show()
```

Étape 5 : Visualisation des activations

Visualisez les activations des couches internes pour une image d'entrée.

```
from tensorflow.keras.models import Model
import numpy as np

# Sélection d'une image d'entrée
image = x_test[0:1]

# Création d'un modèle pour accéder aux activations
layer_outputs = [layer.output for layer in model.layers if 'conv' in layer.name]
activation_model = Model(inputs=model.input, outputs=layer_outputs)
activations = activation_model.predict(image)

# Visualisation des activations
import matplotlib.pyplot as plt

def plot_activations(activations):
    for i, activation in enumerate(activations):
        num_filters = activation.shape[-1]
        size = activation.shape[1]
        plt.figure(figsize=(15, 15))
        for j in range(min(num_filters, 16)):
            plt.subplot(4, 4, j + 1)
            plt.imshow(activation[0, :, :, j], cmap='viridis')
            plt.axis('off')
        plt.suptitle(f"Activations de la couche {i + 1}")
        plt.show()

plot_activations(activations)
```

6 Exercice : Détection de formes géométriques simples

Étape 1 : Génération du jeu de données

Créez un jeu de données artificiel contenant des images binaires de cercles et de carrés. Chaque image doit avoir :

— Une taille fixe, par exemple `64x64 pixels`.

— Une seule forme géométrique aléatoirement placée (soit un cercle, soit un carré).

Préparez également les labels associés aux images (`0` pour cercle, `1` pour carré).

Étape 2 : Construction du modèle

Construisez un modèle CNN avec les caractéristiques suivantes :

— Deux couches convolutionnelles avec activation `relu` et des filtres de taille `3x3`.

— Deux couches de max pooling avec une taille de fenêtre `2x2`.

— Une couche de flatten pour transformer les cartes de caractéristiques en vecteurs.

— Une couche fully connected avec 64 neurones et activation `relu`.

— Une couche de sortie avec 2 neurones et activation `softmax`.

Étape 3 : Compilation et entraînement

Compilez le modèle avec l'optimiseur `adam` et la fonction de perte `categorical_crossentropy`. Entraînez le modèle sur les données générées pour un total de `10 époques`, avec une validation sur 20% des données.

Étape 4 : Évaluation et visualisation

Évaluez les performances du modèle sur un ensemble de test. Visualisez les résultats suivants :

— La perte (loss) et la précision (accuracy) sur les données de test.

— Quelques exemples d'images de test avec leurs classes prédites et réelles.

Étape 5 : Analyse des résultats

Discutez des résultats obtenus. Proposez des améliorations possibles, telles que l'utilisation de régularisation (Dropout, L2) ou la modification de l'architecture du modèle.

Solution

Étape 1 : Génération d'un dataset artificiel

Utilisez `NumPy` pour créer un dataset d'images binaires contenant des formes géométriques simples (cercles et carrés).

```
import numpy as np
import matplotlib.pyplot as plt

def generate_dataset(num_samples=1000, image_size=64):
    X = np.zeros((num_samples, image_size, image_size, 1))
    y = np.zeros((num_samples, 2))  # Deux classes : cercle et carré

    for i in range(num_samples):
        label = np.random.randint(0, 2)  # 0 pour cercle, 1 pour carré
```

```
        y[i, label] = 1

        # Dessiner la forme
        center = np.random.randint(10, image_size - 10, 2)
        size = np.random.randint(5, 15)

        if label == 0:  # Cercle
            for x in range(image_size):
                for y_pixel in range(image_size):
                    if np.sqrt((x - center[0])**2 + (y_pixel - center[1])**2) <= size:
                        X[i, x, y_pixel, 0] = 1
        else:  # Carré
            X[i, center[0]:center[0]+size, center[1]:center[1]+size, 0] = 1

    return X, y

# Générer les données
X, y = generate_dataset()

# Visualiser quelques exemples
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(X[i].squeeze(), cmap='gray')
    plt.title('Cercle' if y[i, 0] == 1 else 'Carré')
    plt.axis('off')
plt.show()
```

Étape 2 : Construction du modèle CNN

Construisez un modèle CNN pour classifier les images en deux catégories : cercles et carrés.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Définir le modèle CNN
model = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=(64, 64, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(2, activation='softmax')
])

# Compilation
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Étape 3 : Entraînement du modèle

Divisez les données en ensembles d'entraînement et de test, puis entraînez le modèle.

```
from sklearn.model_selection import train_test_split

# Diviser les données
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Entraîner le modèle
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test)
```

Étape 4 : Évaluation et visualisation

Évaluez les performances du modèle et visualisez les prédictions sur des exemples de test.

```
# Évaluation
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")

# Visualiser des prédictions
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(X_test[i].squeeze(), cmap='gray')
    pred = model.predict(X_test[i:i+1]).argmax()
    plt.title('Prédit: Cercle' if pred == 0 else 'Prédit: Carré')
    plt.axis('off')
plt.show()
```