M2 MIAGE IDA – Deep Learning
TD2 : Réseaux de neurones convolutionnels

Objectifs :

- Manipuler un ensemble d'images
- Comprendre le principe de la convolution
- Implémenter un réseau de neurones convolutionnel en python avec Tensorflow

Préambule

Avant toute chose, il va falloir télécharger un ensemble d'image (un dataset). Il existe de nombreux datasets de tous types de données, avec un objectif de régression ou de classification, disponibles avec Tensorflow. Une liste se trouve ici :
https://www.tensorflow.org/datasets/catalog/overview#all_datasets

Pour pouvoir télécharger un dataset directement depuis un script python, il faut installer le package tensorflow_datasets dans votre environnement.

Exercice 1

L'objectif de cet exercice est d'améliorer les performances des réseaux développés dans le dernier exercice du TD précédent sur le dataset CIFAR-10. L'utilisation de couches de convolution va ici permettre de prendre en compte la structuration des données images. Nous allons construire un petit réseau pour montrer l'intérêt de cet opérateur.

1. Charger le dataset CIFAR-10
2. Préparer le dataset pour l'apprentissage : normaliser les images, encoder les sorties pour un réseau de neurones
3. Implémenter et tester l'architecture suivante :
    a. 1 couche Conv2D avec 32 filtres de taille 3x3, padding = same
    b. 1 couche MaxPooling2D de taille 2x2
    c. 1 couche Conv2D avec 64 filtres de taille 3x3, padding = same
    d. 1 couche MaxPooling2D de taille 2x2
    e. 1 couche Dense avec 256 et une activation Relu
    f. 1 couche Dense avec 10 neurones configurés pour la classification
4. Comparer les résultats de l'apprentissage avec les MLP développés dans le TD précédent, en performance et en nombre de paramètres. Evaluer la capacité de ce réseau à généraliser de cette architecture.
5. Modifier l'architecture précédente pour améliorer les résultats obtenus.

Exercice 2

Cet exercice a pour but de manipuler un opérateur de convolution afin de repérer les arêtes sur une image.

1. Télécharger le dataset ‘stanford_dogs’ directement depuis votre notebook :

```
import tensorflow_datasets as tfds
```

La méthode tfds.load permet de télécharger des datasets. Utilisez cette méthode en téléchargeant également les métadonnées (infos), en mode supervisé, et avec un split ‘train, test’.

2. Récupérez le nombre de classes à partir des informations. Combien y at’il de classes ?

3. Visualisez les premières images du dataset à l’aide de la méthode show_examples

4. Récupérez la première image. Cela peut se faire avec next et iter, ou bien avec take et get_single_element. Attention, n’oubliez pas que ces méthodes récupères un tuple (image, label).

5. Convertissez cette image en grayscale (nécessite d’installer tensorflow_io) avec rgb_to_grayscale.
6. Une couche de convolution va être appliquée sur l’image grayscale. En tensorflow, les données sont transmises aux différentes couches des réseaux de neurones par batch. Il faut donc rajouter une dimension devant l’image pour représenter le batch. Dans un premier temps, vérifier les dimensions de l’image (avec np.shape par exemple). Ajoutez une dimension avec expand_dims, puis vérifiez le résultat.

7. Définissez un noyau de convolution, à l’aide d’un array numpy à 2 dimensions. Chacune des dimenions devra contenir 3 éléments de type float32. Définissez les valeurs vous-même, manuellement.

8. Vous allez maintenant appliquer le noyau de convolution sur l’image grayscale, à l’aide de tf.nn.conv2d. Utilisez un stride de 1 et un padding ‘SAME’. Le noyau doit avoir les bonnes dimensions pour être utilisé par conv2d. Il faut le redimensionner à l’aide de tf.reshape. Les dimensions sont [hauteur, largeur, canaux_entrants, canaux_sortant]. Appliquez le filtre et visualisez l’image en mode ‘grayscale’ (plt.imshow(…, cmap = ‘grayscale’))

9. Maintenant que vous savez appliquer un noyau, améliorez votre programme afin qu’il implémente un filtre de Sobel :
https://fr.wikipedia.org/wiki/Filtre_de_Sobel

Exercice 3

Dans cet exercice, nous allons manipuler le dataset d’image afin de le préparer pour un apprentissage.

1. Télécharger le dataset ‘stanford_dogs‘ depuis votre notebook avec les mêmes options que dans l’exercice précédent. Normalement, le dataset est déjà sur votre machine. Cette opération devrait donc être très rapide cette fois.

2. Le dataset est composé d'un jeu d'entraînement et d'un jeu de test. Pour chacun de ces jeux, vous allez appliquer une transformation sur les labels. Les labels sont décrits par défaut comme des nombres entiers, vous allez les convertir avec un encodage 'one_hot'. Pour réaliser cela, vous pouvez vous aider de la fonction map qui peut être appliquée sur un dataset, ainsi que de la fonction de Tensorflow one_hot.

3. Toutes les images n'ont pas la même taille dans le dataset. Vous allez donc appliquer des transformations aux images pour qu'elles aient toutes les mêmes dimensions : 128x128. Ensuite, vous allez normaliser les valeurs des pixels pour qu'elles soient entre 0 et 1 (pour aider l'apprentissage). Pour appliquer ces transformations, vous allez utiliser une couche de preprocessing. Celle-ci se déclare comme un modèle Sequential auquel vous passerez en paramètre une couche Resizing ainsi qu'une couche Rescaling). Pour commencer, appliquer cette procédure sur une seule image.

4. Appliquez les transformations à l'ensemble du dataset à l'aide de la fonction map.

5. Similairement, il est possible de faire de l'augmentation de données à l'aide de couches de preprocessing. Différents types de couches sont disponibles :

- RandomBrightness
- RandomContrast
- RandomFlip
- RandomRotation
- RandomTranslation
- RandomZoom
- …

Expérimentez les différentes transformations en testant leur effet sur une image. Vous pouvez combinez plusieurs transformations dans la même couche de preprocessing.
Nota bene : lorsque vous appliquerez ces transformation à un jeu de donnée, il ne faudra les appliquer que sur l'ensemble d'apprentissage.

Exercice 4

Dans cet exercice, nous allons créer un réseau de neurones convolutionnel et le tester sur un jeu de données.

1. Téléchargez et préparez le dataset 'beans' pour l'entrainement.

2. Implémentez une fonction qui retourne un réseau de neurones convolutionnel. Pour cela, vous allez créer un modèle Sequential vide auquel vous allez ajouter une série de couches Conv2D et MaxPooling2D. Le modèle suivra l'architecture des réseaux VGG : 1 bloc VGG est constitué de 2 couches convolutionnelles (padding : 'same') suivies d'une couche MaxPooling2D. Votre réseau doit contenir 3 blocs avec des nombres de filtres en sortie des couches convolutionnelles différents dans chaque : 32 dans le premier bloc, 64 dans le second et 128 dans le troisième.

Vous terminerez votre réseau avec une couche Flatten() et une couche Dense. Attention à bien paramétrer les couches pour que tout soit compatible avec les données en entrée. N'oubliez pas d'ajouter les fonctions d'activations appropriées.

3. Compilez votre réseau avec compile, et affichez le détail de l'architecture créée. Observez le nombre de paramètres pour chaque couche. La première couche convolutionnelle doit contenir 896 paramètres et la seconde 9248. Pourquoi ces nombres ? Comment calculer le nombre de paramètres d'une couche convolutionnelle ?

4. Entrainez le sur le dataset avec fit pendant 50 epochs. Lors de l'entrainement, utilisez un ensemble de validation avec 10 % des données. Lors de l'appel à fit, récupérez le résultat dans une variable history (history = mon_reseau.fit(…)). Utilisez une métrique pour enregistrer l'accuracy à chaque epoch.

5. Une fois l'entrainement terminé, utlisez le contenu de history pour tracer des courbes d'évolution de la loss et de l'accuracy, sur les ensembles d'entrainement et de validation.

6. Pour finir, testez le réseau sur des données non observées lors de l'entrainement. Le résultat est-il satisfaisant ?

Exercice 5

L'objectif de cet exercice est d'améliorer les performances du réseau.

1. Sans changer l'architecture globale que vous aviez choisi, améliorez les performances de votre réseau en utilisant tous les mécanismes à votre disposition. Vous pouvez par exemple ajouter une couche d'augmentation de données au début de votre modèle, des couches de Dropout, de BatchNormalization.

2. Entrainez à nouveau votre réseau et réglez les differents paramètres pour obtenir un apprentissage satisfaisant, avec des courbes d'entrainement et de validation proches l'une de l'autre.