7

Utilisation du deep learning

Dans ce chapitre, nous allons présenter quelques utilisations des modèles d'apprentissage profond ou *deep learning*. Le deep learning est une méthode qui a surpassé les méthodes traditionnelles de programmation pour la résolution de problèmes informatiques.

La classification d'images consiste à attribuer un nom (plus de 1000 noms possibles) à une image en fonction de son contenu. Par exemple, avant l'année 2012, les algorithmes de classifications d'images avaient un taux d'erreurs de l'ordre de 30%. En 2012, lors du ImageNet Large Scale Visual Recognition Challenge 2012, un nouvel algorithme utilisant l'apprentissage profond obtenait un taux d'erreurs de 15%, soit presque deux fois mieux que les autres. L'apprentissage profond va ensuite envahir tous les domaines, la reconnaissance de la parole, la physique, la chimie, le jeu, la conduite autonome, la traduction automatique, etc.

Attention › *Dans ce chapitre, le mot "classe" est utilisé dans le cadre de la classification surpervisée. On utilisera l'expression "objet de type" pour parler d'une classe au sens de la Programmation Orientée Objet.*

Les exemples présentés dans ce chapitre relèvent du domaine le plus ancien, le traitement des images, dix ans seulement. Ce domaine est aussi le plus documenté et le plus simple à mettre en œuvre.

APPRENTISSAGE PROFOND (OU DEEP LEARNING)

Le deep learning permet de résoudre un problème complexe en utilisant des données d'entraînement. Le processus appliqué aux données pour obtenir le résultat souhaité est décrit dans une architecture. Cette architecture contient un ensemble de paramètres et d'opérations ordonnées décrit dans un graphe. Le nombre de paramètres définissant l'architecture est souvent de l'ordre du million de valeurs. L'étape permettant de calculer le résultat en utilisant l'architecture est nommée l'*inférence*.

L'architecture peut contenir des filtres de convolutions, des extractions d'extremums, des opérations linéaires ou non linéaires, des réseaux de neurones. Cette liste est non exhaustive car l'imagination des spécialistes du deep learning est sans limite. Le fichier contenant l'architecture décrit l'ensemble d'opérations ordonnées et l'ensemble

des valeurs des paramètres permettant l'étape d'inférence. Ces valeurs définissent par exemple les valeurs des filtres de convolutions, les limites des régions où les extremums sont à extraire, les constantes des opérations non linéaires ou bien les poids des nœuds des réseaux de neurones.

Ces valeurs sont ajustées après une phase d'optimisation à partir d'un jeu de données d'entraînement. L'étape permettant de trouver ces valeurs est nommée l'*apprentissage*.

De nombreux modèles sont disponibles sur le Web. Ils sont issus de travaux de recherche. Les auteurs des publications donnent très souvent les liens vers leurs modèles, architecture et poids. Une recherche sur Google à partir du mot clé "deep learning" et du domaine qui vous intéresse (comme "leaf", "gender", "cats", "face", "audio") permet bien souvent de trouver ces modèles.

Dans ce chapitre, le modèle est considéré comme une boîte noire ; on fournit les entrées (une ou plusieurs images) à la boîte noire et on récupère les sorties.

Les deux principales librairies d'apprentissage profond accessibles en langage Python sont TensorFlow et PyTorch. Dans ce chapitre, nous n'utiliserons que le module TensorFlow pour illustrer l'efficacité du deep learning sur les problèmes de traitement des images.

INSTALLATION NÉCESSAIRE

Dans ce chapitre, nous aurons besoin des modules complémentaires suivants :

- `tensorflow`
- `pillow`
- `numpy`
- `opencv-python`

Attention *› Les modèles du deep learning utilisent beaucoup de ressources mémoire et CPU ou GPU. Le module Tensorflow n'est pas disponible sur Raspberry.*

Codes sources dans le dossier `chapitre_dl` 

7.1. Téléchargement des architectures et des poids

Avant d'exécuter les exemples de programme de ce chapitre, vous devez télécharger les données associées aux différents modèles (les boîtes noires) que nous allons mettre en œuvre et les sauvegarder sur votre disque dur dans le répertoire `/tmp`.

- Les premières données à télécharger sont disponibles à l'adresse : `https://storage.googleapis.com/tfhub-modules/tensorflow/faster_rcnn/resnet101_v1_1024x1024/1.tar.gz`

  Après avoir téléchargé ces données, il faut créer un dossier nommé `faster_rcnn_resnet101` dans le dossier `/tmp` et copier les données extraites (le dossier `variables` et le fichier `saved_model.pb`) dans le dossier `faster_rcnn_resnet101`. Il faut aussi copier le fichier `coco_90.json` dans le dossier `/tmp`.

- Les secondes données à télécharger sont disponibles à l'adresse : `https://storage.googleapis.com/tfhub-modules/captain-pool/esrgan-tf2/1.tar.gz`

  Après avoir téléchargé ces données, il faut créer un dossier nommé `esrgan-tf2` dans le dossier `/tmp` et copier les données extraites (le dossier `variables` et le fichier `saved_model.pb`) dans le dossier `esrgan-tf2`.

- Les troisièmes et dernières données à télécharger sont disponibles à l'adresse : `https://storage.googleapis.com/tfhub-modules/google/magenta/arbitrary-image-stylization-v1-256/2.tar.gz`.

  Après avoir téléchargé ces données, il faut créer un dossier nommé `neural_style` dans le dossier `/tmp` et copier les données extraites (les dossiers `assets` et `variables` et le fichier `saved_model.pb`) dans le dossier `neural_style`.

7.2. Détecter des objets dans une image

Exemple 7.1 : *tensorflow_ex1.py*

Afficher le listing suivant dans un navigateur

Dans cet exemple, nous utiliserons le modèle (ou notre boîte noire) `Faster R-CNN` pour nommer et localiser des objets, des personnes ou des animaux dans une image ; cet ensemble constitue les classes identifiables par le modèle. L'ensemble de ces classes sont répertoriées dans le fichier `coco_90.json`.

On importe les modules `json` ❶, `numpy` ❷, `PIL.Image` ❸, `PIL.ImageDraw` ❹ et enfin `tensorflow` ❺. Attention, `tensorflow` occupe en mémoire plus de 150 Mo, son importation peut donc être longue en temps.

En ❻, on appelle la fonction `load_model` avec en paramètre le chemin complet du dossier contenant le modèle (pour notre cas `/tmp/faster_rcnn_resnet101`). La valeur retournée est un objet de type Model. En ❼, on charge l'image dans laquelle on va identifier et localiser des classes en appelant `open`. Cette image est un montage des images `kite.jpg`, `giraffe.jpg` et de l'image `4545.jpg` (de la base `VOC 2007`).

En ❽, on utilise un gestionnaire d'exception pour ouvrir le fichier JSON contenant le nom des classes en appelant la fonction `open` ❾ dans le gestionnaire de contexte `with`. Le contenu est lu en appelant la méthode `read` ❿ donnant comme résultat un type `str`. En ⓫, on appelle la fonction `loads` avec en argument les données brutes de type `str` pour initialiser la variable de type `dict` nommée `nom_classe` avec le nom des classes.

Si une exception de type OSError se produit lors de l'ouverture ou de la lecture du fichier, on met la variable `nom_classe` à la valeur `None` et on écrit un message d'erreur entre ⓬ et ⓭.

Le résultat de l'appel à `open` ❼ est du type Image et doit être converti en tableau NumPy afin d'être utilisé comme entrée du modèle `faster_rcnn_resnet101` pour TensorFlow. La conversion est faite en appelant la fonction `asarray` de NumPy ⓮.

En ⓯, on recherche le nom et la localisation des classes contenues dans l'image. En langage d'apprentissage profond, cela s'appelle l'inférence. L'inférence se fait en appelant l'opérateur `()` après la variable contenant notre modèle et en plaçant l'image a étudier dans une liste. L'opérateur `()` est équivalent à un appel à la méthode `__call__` de type Model.

Le résultat de l'appel à la méthode `__call__` ⓯ est une variable de type `dict` nommée `resultat`. On va tracer les résultats sur l'image en utilisant le module Pillow (❸ et ❹). En ⓰, on convertit l'image lue par Pillow en image du type RGBA. On appelle la méthode `convert` pour convertir l'image lue en image en donnant la valeur `'RGBA'` en argument. Le résultat de l'appel à la méthode `convert` est du type Image. Les résultats de l'identification vont être tracés dans un objet de type ImageDraw. Dans cet objet, on va tracer l'image en appelant la fonction `Draw` ⓱ avec en argument l'image (résultat de l'appel à la méthode `convert`).