Rebecca Leygonie Nemanja Kostadinovic Meriem Elghafraou

Projet Docker : Déploiement d'un modèle de reconnaissance automatique de chiffre

Notre projet consiste à déployer un modèle de réseau de neurones à convolution (CNN) grâce aux outils de conteneurisation.

Les technologies utilisées pour la création du modèle de machine learning: Python3 et libraires de traitement de données (numpy,pandas..), Sklearn pour la normalisation des données et Tensorflow/Keras pour l'apprentissage du modèle(CNN).
Le front-end est réalisé à l'aide d'un serveur Flask (python), d'un script html et javascript.
La partie déploiement est réalisée avec les technologies Docker et Vagrant.

L'architecture du projet :

- Vagrantfile: création d'un environnement virtuel (box) ubuntu/bionic64, redirection des proxy nécessaires pour le conteneur ui_container et lancement du docker-compose.yml

- docker-compose.yml : création du volume partagé (/projet_digit/shared/) et construction des conteneurs à partir de leur Dockerfile

- 2 conteneurs :

  algo_container :

  1) traitement des données : script Python cnn_mnist.py qui récupère les données train.csv et test.csv et les normalise

  2) création du modèle (CNN) et son apprentissage (cnn_mnist.py)

  3) l'export du modèle sous forme de fichier *model.json*, ses poids dans le fichier *model.h5* et sa sauvegarde dans le répertoire partagé (/projet_digit/shared/)

  ui_container :

  1) création d'un serveur Python de type Flask avec une interface en HTML et JavaScript

  2) récupération du modèle crée précédemment (disponible dans le répertoire partagé (/projet_digit/shared/)

  3) utilisation du modèle pour prédire le chiffre dessiné par l'utilisateur

- bonus : récupération des images de nos conteneurs depuis docker-hub

Utilisation de notre programme :

1) Décompresser le répertoire :
TAR_Kostadinovic_Nemanja_Leygonie_Rebecca_Elghafraou_Meriem.zip

2) Se placer dans le répertoire : **reconnaissance_chiffre**

3) exécuter la commande : **vagrant up**
prérequis : avoir une installation fonctionnelle de Vagrant et de son plugin vagrant-docker-compose

```
C:\Users\G20980\Desktop\Nemanja\reconnaissance_chiffre>vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: This machine used to live in C:/Users/G20980/Desktop/Nemanja/projet_docker_vagrant but it's now at C:/Users
/G20980/Desktop/Nemanja/reconnaissance_chiffre.
==> default: Depending on your current provider you may need to change the name of
==> default: the machine to run it as a different machine.
==> default: Checking if box 'ubuntu/bionic64' version '20201211.1.0' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 5000 (guest) => 5000 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
```

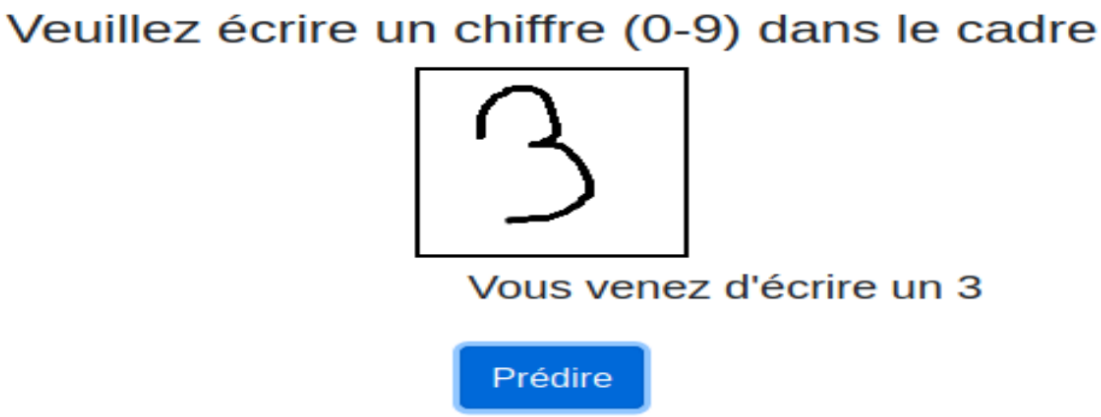
4) Attendre la fin de la création de l'environnement et de l'exécution des conteneurs :

```
    default: Removing intermediate container 480d4a92cf33
    default:  ---> 6c50c11506b5
    default: Step 13/14 : EXPOSE 5000
    default:  ---> Running in 30b40c48eeb7
    default: Removing intermediate container 30b40c48eeb7
    default:  ---> 5338c5b766e2
    default: Step 14/14 : CMD ["python3","-m","flask","run","--host=0.0.0.0","--port=5000"]
    default:  ---> Running in 7ffd4d90010c
    default: Removing intermediate container 7ffd4d90010c
    default:  ---> bceb2f7953f8
    default: Successfully built bceb2f7953f8
    default: Successfully tagged vagrant_ui_container:latest
==> default: Image for service ui_container was built because it did not already exist. To rebuild this image you must use `docker-c
ompose build` or `docker-compose up --build`.
==> default: Creating vagrant_algo_container_1 ...
Creating vagrant_algo_container_1 ... done
==> default: Creating vagrant_ui_container_1   ...
Creating vagrant_ui_container_1   ... done
```

5) sur la machine hôte se rendre à l'adresse : **http://127.0.0.1:5000**

6) dessiner à l'aide de sa souris un chiffre dans le carré et appuyer sur le bouton **Prédire**

La prédiction du modèle est retournée :



7) entrer la valeur dessiné (facultatif) et appuyer sur le bouton **Submit** afin d'effacer le dessin

Précisions sur le projet:

- Pour d'éviter une attente trop longue de algo_container (apprentissage en 20 epochs d'une durée de 2 heures) nous mettons à disposition les fichiers *model.json* et *model.h5* au conteneur ui_container afin de ne pas attendre la fin du premier.
- les données sont issues de : https://www.kaggle.com/oddrationale/mnist-in-csv, il y a 60.000 données d'apprentissage et 10.000 données de test.

L'algorithme d'apprentissage automatique utilisé :

La classification des images consiste à prendre une image en entrée et à en sortir une classe ou une probabilité de classes qui décrit le mieux l'image. Dans le CNN (Convolutionnal Neural Network), nous prenons une image en entrée, en regardant l'importance de ses différents aspects/caractéristiques, nous sommes capables de différencier les images les unes des autres.

Le prétraitement requis dans CNN est beaucoup moins important que dans d'autres algorithmes de classification. Les ordinateurs ne peuvent pas voir les choses comme nous le faisons, car l'image des ordinateurs n'est rien d'autre qu'une matrice de pixels. Nous devons donc transformer l'image en cette matrice et normaliser les valeurs des pixels.

Un CNN comporte généralement plusieurs couches : une couche convolutionnelle, une couche de pooling et une couche entièrement connectée.

Notre modèle est créé à l'aide des librairies python Tensorflow et Keras et a la structure suivante:

- une couche de convolution avec 32 filtres
- une couche de Dropout avec la probabilité=0.5
- une couche de batch normalisation
- une couche de Max Pooling
- une couche de convolution avec 64 filtres
- une couche de Dropout avec la probabilité=0.5
- une couche de batch normalisation
- une couche de Max Pooling
- une couche flatten
- une couche entièrement connectée de 256 neurones (nombre de pixel) avec la fonction d'activation ReLu
- une couche entièrement connectée de 10 neurones (un pour la prédiction de chaque chiffre) avec la fonction d'activation softmax

L'optimisation du gradient se fait grâce au modèle Adam et la fonction coût et l'entropie croisée.

L'objectif principal de la convolution est d'extraire de l'entrée des caractéristiques telles que les bords, les couleurs, les coins. Plus nous nous enfonçons dans le réseau, plus celui-ci commence à identifier des caractéristiques plus complexes. Cette couche réalise un produit matriciel entre deux matrices, où une matrice (appelée filtre/noyau) est l'ensemble des paramètres pouvant être appris, et l'autre matrice est la partie restreinte de l'image.
A la fin du processus de convolution, nous avons une matrice caractéristique qui a des paramètres (dimensions) moins importants que l'image réelle ainsi que des caractéristiques plus complexes que l'image réelle. Le réseau va ensuite faire passer cette image à des couches de pooling. Cette couche a pour seul but de diminuer la puissance de calcul nécessaire au traitement des données. Elle se fait en diminuant encore plus les dimensions de la matrice présentée. Dans cette couche, nous essayons d'extraire les caractéristiques dominantes de l'image.
Ces deux couches permettent de mettre en évidence certaines caractéristiques dans une image et réduire les dimensions de l'image de manière drastique.

À partir de la couche entièrement connectée commence le processus de classification. L'image entrée est convertie en une forme appropriée pour notre architecture multi-couches entièrement connectée, nous allons aplatir l'image en un vecteur à une colonne. La sortie aplatie est transmise à un réseau neuronal à rétropropagation du gradient qui est appliqué à chaque itération lors de l'apprentissage. Une fois l'apprentissage finit, le modèle peut distinguer les éléments dominants dans les images et les classer en différentes catégories ici (0,1,2..9).

Bonus : récupération des images depuis docker-hub

Les images ont été téléchargé sur docker-hub avec le DockerID : meriemtest dans le repértoire myrepository.

Le fichier docker-compose inclus dans le répertoire bonus permet avec la commande de lancement (**docker-compose up**) de télécharger, crée, lancer les images et se rendre à l'adresse : 0.0.0.0 :5000 pour tester le programme.