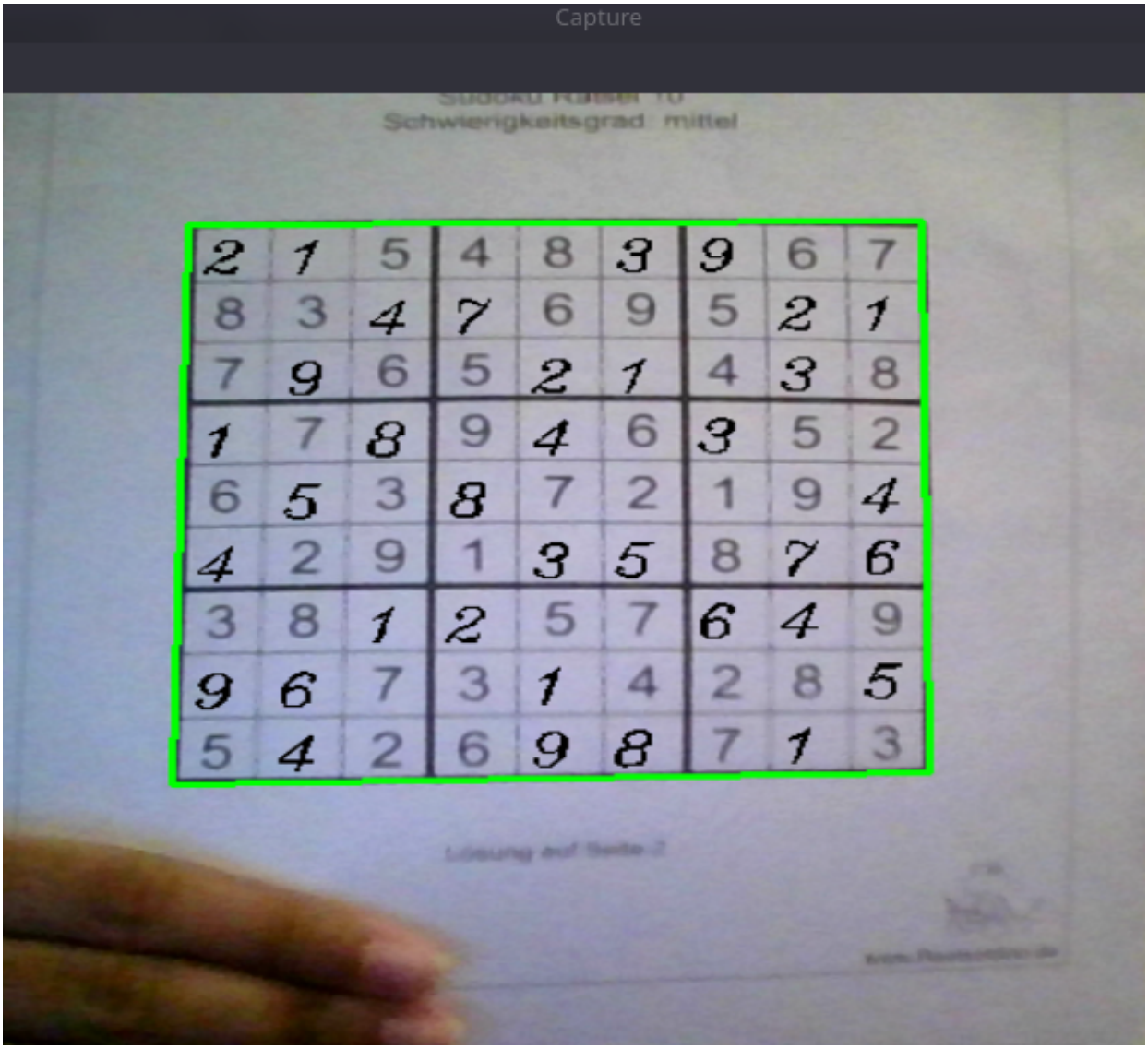
Compte Rendu - Projet n°2 : Deep Learning et Sudoku

Thomas FLORENT



Résultat final !

1. Mise en place de la Raspberry Pi

Ce qui a été fait

La configuration de la machine virtuelle sous VirtualBox ne causa pas le moindre soucis. J'ai basé la mienne sur une image ISO de Raspberry Pi OS (anciennement Raspbian, le nom a été officiellement changé en 2020 pour la sortie de la Pi 4) récupérée directement depuis le site de la fondation Raspberry.

L'objectif d'obtenir une installation fonctionnelle de tensorflow sur la machine pour y faire tourner le projet, cependant, se révéla rapidement beaucoup plus ardu qu'initialement prévu.

En effet, après l'avoir installé via pip, Tensorflow remplissait le fil d'erreurs dès son importation, et ce peu-importe le script. Ci dessous, un exemple en tentant de lancer l'entrainement du modèle que vous nous aviez fourni en début de projet.

```
pi@raspberry:/media/sf_Sudopi/src $ python3 train.py -d dataset
/home/pi/.local/lib/python3.7/site-packages/skimage/io/manage_plugins.py:23: UserWarning: Your ins
talled pillow version is < 7.1.0. Several security issues (CVE-2020-11538, CVE-2020-10379, CVE-202
0-10994, CVE-2020-10177) have been fixed in pillow 7.1.0 or higher. We recommend to upgrade this l
ibrary.
  from .collection import imread_collection_wrapper
Traceback (most recent call last):
  File "train.py", line 21, in <module>
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
  File "/home/pi/.local/lib/python3.7/site-packages/tensorflow/__init__.py", line 23, in <module>
    from tensorflow.python import *
  File "/home/pi/.local/lib/python3.7/site-packages/tensorflow/python/__init__.py", line 49, in <m
odule>
    from tensorflow.python import pywrap_tensorflow
  File "/home/pi/.local/lib/python3.7/site-packages/tensorflow/python/pywrap_tensorflow.py", line
28, in <module>
    _pywrap_tensorflow = swig_import_helper()
  File "/home/pi/.local/lib/python3.7/site-packages/tensorflow/python/pywrap_tensorflow.py", line
24, in swig_import_helper
    _mod = imp.load_module('_pywrap_tensorflow', fp, pathname, description)
  File "/usr/lib/python3.7/imp.py", line 242, in load_module
    return load_dynamic(name, filename, file)
  File "/usr/lib/python3.7/imp.py", line 342, in load_dynamic
    return _load(spec)
ImportError: /home/pi/.local/lib/python3.7/site-packages/tensorflow/python/_pywrap_tensorflow.so:
cannot open shared object file: No such file or directory
```

Bien que l'erreur semble mentionner un problème en lien avec PyWrap, après vérification et revérification, ce dernier était bel et bien installé et à jour. Rechercher

cette erreur ne mena à rien d'autre qu'un thread github encore ouvert et non résolu.

En poursuivant les recherches dans les packages que je découvris la source du problème : la version de tensorflow fournie par le repository pip de la PI était terriblement obsolète.

```
tensorflow in ./.local/lib/python3.7/site-packages (0.11.0)
```

Le tensorflow fourni est en 0.11.0, version dâtée *d'au moins* quelques années.

Un peu plus tard, je vois qu'il existe une documentation officielle pour l'installation de tensorflow sur raspberry, en compilant directement le code source original. Cependant, cette installation elle-même requiert la configuration de la plateforme de développement Docker

À nouveau, on peut trouver une documentation pour installer Docker sur Debian (et Raspbian), mais l'article précise que les utilisateurs de Raspbian se voient dans l'obligation d'installer la plateforme via un script spécifique.

> **Raspbian users cannot use this method!**
>
> For Raspbian, installing using the repository is not yet supported. You must instead use the convenience script.

Et visiblement, ce script semble vouloir bouder lui aussi.

```
pi@raspberry:~ $ curl -fsSL https://get.docker.com -o get-docker.sh
pi@raspberry:~ $ sudo sh get-docker.sh
# Executing docker install script, commit: 7cae5f8b0decc17d6571f9f52eb840fbc13b2737
+ sh -c apt-get update -qq >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-https ca-certificates curl >/dev/null
+ sh -c curl -fsSL "https://download.docker.com/linux/debian/gpg" | apt-key add -qq - >/dev/null
Warning: apt-key output should not be parsed (stdout is not a terminal)
+ sh -c echo "deb [arch=i386] https://download.docker.com/linux/debian buster stable" > /etc/apt/sources.list.d/docker.list
+ sh -c apt-get update -qq >/dev/null
+ [ -n  ]
+ sh -c apt-get install -y -qq --no-install-recommends docker-ce >/dev/null
E: Package 'docker-ce:amd64' has no installation candidate
pi@raspberry:~ $
```

(Mal ?)heureusement, l'apparition de cette erreur semble assez courante :

- Certains utilisateurs pensent que cela provient d'une incompatibilité d'architecture.
- D'autres, sur un thread plus âgé, suggèrent de revenir à une version précédente.

Ce qu'il reste à faire

- Compléter l'installation de Docker, puis Tensorflow, pour commencer à adapter le programme à la Raspberry.

2. Création du réseau neural

Ce qui a été fait

Le modèle initial fût construit autour du dataset MNIST, une base de données de chiffres manuscrits déjà intégrée à Keras.

J'ai peiné à trouver la configuration nécessaire pour réellement entraîner un modèle correct. Notamment, au début, mon CNN confondait régulièrement les 5 - 9, 6 - 8, et les 2 - 7. Il arrivait aussi que le grain de l'image le pousse à 'détecter' des chiffres dans les cases vierges, même après un flou gaussien sur la capture. En lançant le programme avec ce modèle, il y a en général, à chaque frame, une demi douzaine de caractères erronés empêchant la résolution du sudoku.

Après avoir essayé différentes configurations de couches neuronales, d'epochs, de seuils et de filtres sans trop de résultat, puis en cherchant des datasets qui pourraient potentiellement être plus adaptés, je suis tombé sur un modèle utilisant un set de chiffres extraits manuellement de fichiers de polices de caractères. Le modèle se révéla être beaucoup plus efficace (je suppose non seulement à cause des caractères non-manuscrits, mais aussi parce qu'il était sans doute bien mieux programmé que le mien...). Très peu d'erreurs sont commises, et sur un format standard, les grilles sont en général analyées puis résolues relativement vite.

Je pense donc essayer, par la suite, de me renseigner sur la manière à procéder pour générer un modèle similaire, qui semble plus prometteur pour cette utilisation.

Principaux documents utilisés

- https://keras.io/examples/vision/mnist_convnet/
  Exemple de réseau convoluté simple rédigé par l'auteur de Keras, et utilisant

MNIST.

- https://faroit.com/keras-docs/1.0.0/getting-started/sequential-model-guide/
  Guide sur les modèles séquentiels, documentation officielle de Keras.
- https://www.tensorflow.org/api_docs/python/tf/keras/Model
  Classe Model, documentation officielle de Tensorflow.
- https://docs.opencv.org/4.5.2/d4/d13/tutorial_py_filtering.html
  Guide d'adoucissement d'image, documentation officielle d'OpenCV.

Difficultés rencontrées

- J'ai toujours un peu de mal à différencier l'entièreté des différents types de couches de neurones proposés par Keras, ou à comprendre en profondeur la plupart des mécanismes mathématiques derrière les réseaux convolutés (les convolutions en elles-même, notamment...). Il faudra encore que je continue à étudier leur fonctionnement avant d'être réellement à l'aise avec le sujet, mais l'abondance de ressources et de documentation disponible en ligne aide énormément !
- Trouver un dataset convenable. Je suis étonné, par exemple, qu'il n'existe pas vraiment de set de chiffres exclusivement dactylographiés (ou alors j'ai très mal cherché !).

Ce qu'il reste à faire

- Développer un autre CNN, prenant comme dataset des chiffres dactylographiés, en extrayant par exemple ceux de fichiers de polices de caractère.

3. Résolution et affichage de la grille

Ce qui a été fait

De très nombreux algorithmes de résolution pour grilles de sudokus existent, et trouver un programme adapté n'était pas vraiment un problème. Il fallait, surtout,

trouver un moyen d'afficher cette solution à l'utilisateur.

Projeter directement la solution sur la grille, comme demandé, était une tâche rendue relativement complexe par l'absence d'un dit projecteur. À la place, j'ai préféré opter pour une superposition des chiffres sur l'image de la grille sur la capture elle-même.

Cela passa notamment par l'utilisation de la librairie OpenCV pour la panoplie d'outils de traitement d'image qu'elle propose, et, dans une moindre mesure, Numpy pour la manipulation de points.

La caméra observe en temps réel jusqu'à trouver une grille. Elle en analyse puis résout le contenu (si possible) avant d'afficher le résultat sous forme "d'overlay" au dessus des cases.

Principaux documents utilisés

- https://docs.opencv.org/4.5.3/d4/d73/tutorial_py_contours_begin.html
  Détection et utilisation des contours, documentation officielle d'OpenCV.
- https://docs.opencv.org/4.5.2/d4/d13/tutorial_py_filtering.html
  Adoucissement d'image, documentation officielle d'OpenCV.
- https://docs.opencv.org/4.5.2/da/d6e/tutorial_py_geometric_transformations.html

  Transformation géométrique d'image, documentation officielle d'OpenCV.
- Autres objets d'étude, de nombreux algorithmes utilisant, de manière similaire, les réseaux neuronaux pour résoudre des sudokus :
  - https://github.com/Kyubyong/sudoku
  - https://github.com/1nfinityLoop/Sudoku-Solver-AI
  - https://github.com/neeru1207/AI_Sudoku

Difficultés rencontrées

- J'ai trouvé cette partie particulièrement intensive au niveau du code. Beaucoup de notions complexes sont abordées et combinées, comme la transformation de perspective, et sont assez difficiles à appréhender initialement.

4. Conclusion

Axes d'amélioration

- Il devrait être possible de pouvoir modifier aisément les dimensions cible des grilles (exemple, prendre en compte des grilles "expert" à 18×18 au lieu de 9×9), ou, encore mieux, que la caméra et l'algo se chargent eux-même de déterminer l'intégralité des dimensions, qu'il s'agisse du format de la grille, du nombre de cases....