Rapport TP Reconnaissance Images

Composition du groupe

Le groupe est composé de CORRAL Jonathan, DAUDEL Adrien et GARAYT Thomas

Objectif initiaux

Nos objectifs initiaux étaient d'implémenter Tensor Flow au sein d'une application afin de distinguer différents panneaux de signalisation et d'être en mesure de lire les caractères au sein de ceux-ci, comme par exemple les panneaux de limitations de vitesses.

Peu de temps après, nous avons remarqués que l'utilisation de Tensor Flow dans le délai imparti était très compliqué. A partir de ce moment là nous avons donc recherché d'autres technologies permettant l'application de réseaux de neurones.

Notre choix s'est finalement porté sur **Keras** et **OpenCV**

Nous nous concentrerons dans un premier temps sur la détection d'objet (un seul par détection), basé sur une image couleur.

Présentation de l'application finale

Notre application de reconnaissance d'image permet de **classifier un objet** selon une banque d'images données.

Notre modèle est créé dynamiquement par rapport à l'ensemble des images fournies dans le dossier **dataset** en éxécutant la commande ***python train_network.py***

Pour ensuite tester une commande, il suffira de lancer la commande suivante :
python test_network.py --image [imageFile]

Technologies utilisées

L'application est programmée en **Python3** et utilise **Keras** pour l'implémentation des réseaux de neuronnes (https://keras.io/), Sklearn

Modélisation

Afin de personnaliser le modèle du réseau neuronal, il suffit d'ajouter des images dans le dossier **dataset** et de relancer la création du modèle.
les images doivents se situer dans un sous-répertoire correspondant au nom de la catégorie associée (cat, dog, human, ...)
Au plus il y a d'images par catégorie, au plus le modèle devrait être précis.

La création du modèle devrait générer un fichier `dataset.model` correspondant au modèle entrainé, ainsi qu'un fichier **labels.json** correspondant aux catégories qui seront reconnues. Ces deux fichiers sont nécessaire pour tester la reconnaissance d'une image.

Structure de dataset

```
dataset/
   label1/
        image1.jpg
        image2.jpg
        …
   label2/
        image1.jpg
        image2.jpg
        …
```

Note : Il est nécessaire de fournir un minimum de 10 images par catégories ou de modifier la valeur du batch_size (variable **BS**) dans le fichier **train_network.py**

Afin de tester l'application, nous avons d'abord créé un petit modèle avec quelques images de fruits (pommes et poires), téléchargeable ici :
https://drive.google.com/open?id=13FeZpWXDwjqu7j5Zv2wqSvgAHwA0kJPo

Nous avons ensuite téléchargé une banque de plus de 100 catégories proposées par CalTech. Les donnés peuvent être téléchargées sur :
http://www.vision.caltech.edu/Image_Datasets/Caltech101/101_ObjectCategories.tar.gz
ou https://drive.google.com/open?id=1RSTK3wjfGwC7dx7OtJgEfz7p8x0IodId

Il suffit ensuite d'extraire le contenu de l'archive dans le dossier `dataset` comme précisé précédemment et de relancer la création du modèle.
Cette étape étant relativement longue, nous avons mis à disposition le modèle déjà entraîné dans l'archive de l'application.

Liste des commandes

Ajouter des modules complémentaires avec pip (dépendances)
Il suffit d'installer PIP : **https://pip.pypa.io/en/stable/installing/#do-i-need-to-install-pip**
Puis de lancer la commande : `sudo pip3 install keras imutils opencv-python sklearn numpy`

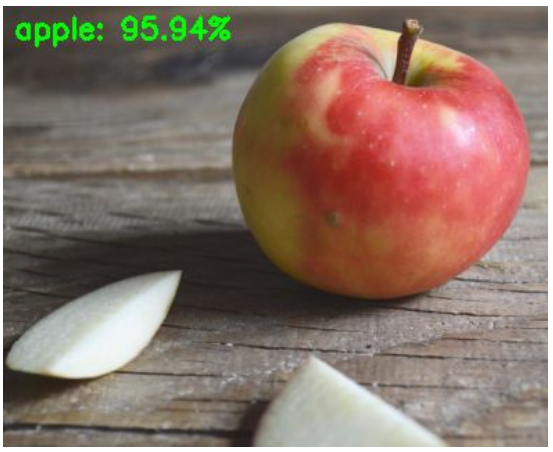
Création du modèle
```
python train_network.py
```

Test d'une image
```
python test_network.py --image [imageFile]
```

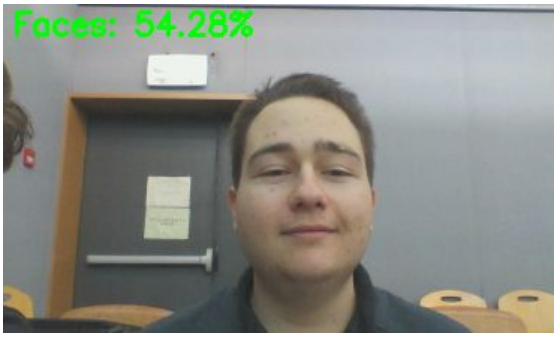
Exemple
```
python test_network.py --image  images/cat.jpg
```

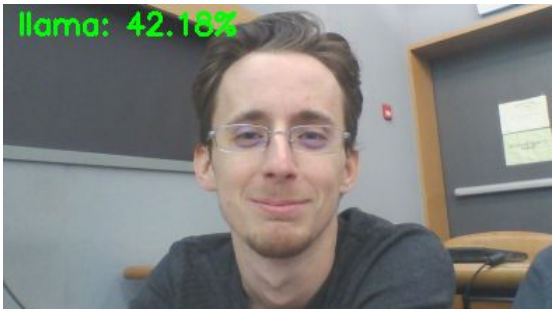
Résultats

Lors de nos premiers tests avec une petite banque d'images (fruits) nous arrivons à obtenir des résultats satisfaisant avec une assez bonne distinction entre une pomme et une poire.



Nous pensions donc qu'avec une banque de données conséquente (CalTech, ~8000 images réparties dans 100 catégories), nous obtiendrons des résultats très satisfaisant. Malheureusement ce n'est pas le cas, nos visages n'ont pas toujours été reconnus correctement.





Nous constatons donc qu'au plus il y a de catégories, au plus le système peut se tromper.

Optimisations

Afin d'essayer d'améliorer la précision de la détection, nous avons fixé le nombre de cycles d'entrainement à 100.
Nous avons aussi modifié le batch_size (nombre d'images utilisées lors de chaque sous passage) afin qu'il soit calculé automatiquement selon le nombre d'images dans les catégories.
Enfin, si le nombre de catégories est supérieur à deux, nous basculons la méthode **loss** du mode **binary_crossentropy** au mode **categorical_crossentropy**
CF. https://keras.io/losses/ et https://keras.io/optimizers/

Résultats après optimisations

Cette fois ⅔ images sont reconnues comme des visages. On s'aperçoit que si l'image n'est pas optimale (cadrage, luminosité, netteté, nombre d'objets), le système à plus de difficultés à classifier correctement.