Rapport de projet

Thomas Hautier

Reconnaissance de panneaux routiers

Objectif

Le but du projet était de pouvoir identifier des panneaux routiers comme peuvent le faire certaines voitures. Pour cela j'ai eu deux approches différentes, la première par traitement de l'image, la deuxième par un réseau de neurones convolutif.

Méthodes

- Par "traitement du signal" :

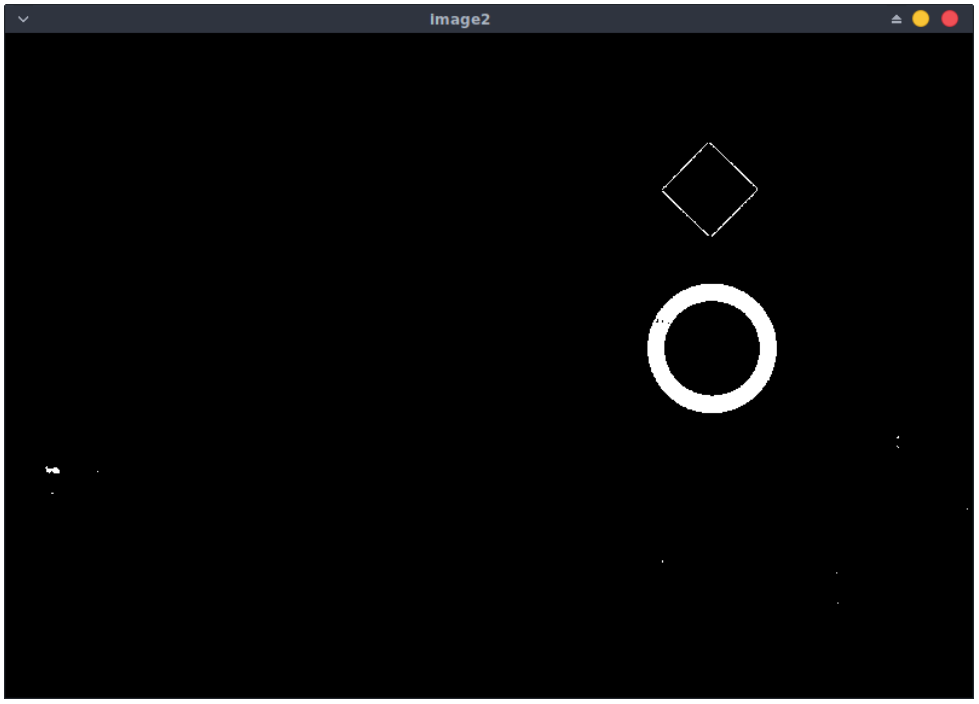
La première méthode qui me semblait la plus simple à mettre en place était d'appliquer différents filtres sur l'image, puis à l'aide de différents algorithmes et méthodes reconnaître la forme et le type de panneau.

La bibliothèque OpenCV est principalement utilisée.

Par exemple prenons cette image.

En passant du système de couleur RGB à HSV *(Hue Saturation Value)* on peut appliquer un filtre permettant d'extraire une certaine couleur uniquement (ici le rouge).

On obtient alors ceci :



On applique ensuite l'algorithme CHT*(Circle Hough Transform)*[1] qui permet d'identifier des cercles dans une image.

On obtient

Il faudrait alors appliquer un algorithme de OCR afin de lire l'intérieur du panneau.

[1] https://en.wikipedia.org/wiki/Circle_Hough_Transform

Cependant cette méthode n'étant pas viable et s'orientant davantage dans du traitement de signal et non dans de la fouille de données, j'ai décidé de changer de méthode.

Même si cette méthode est simple à appliquer elle pose quelques problèmes :
- On peut difficilement appliquer cette méthode sur des panneaux plus compliqués (par exemple tous ceux ayant des formes géométriques à l'intérieur)
- Supporte mal les transformations géométriques (rotations, translations etc …)

- Par CNN (réseau de neurones convolutif)

J'ai donc décidé d'utiliser un réseau de neurones pour identifier les panneaux. Pour les données, j'ai utilisé le Dataset *BelgiumTS*[2]. Celui-ci est composé de deux ensembles, le premier d'entraînement (4700 images) et le deuxième de test (2645 images) pour 62 classes.
A chaque dossier correspond une classe, et les images sont au format .ppm.

Pour les outils, j'ai d'abord voulu utiliser *Tensorflow*, mais je l'ai trouvé un peu trop complexe, j'ai alors décidé d'utiliser Keras[3] à la place, une bibliothèque basée sur *Tensorflow* proposant un niveau d'abstraction un peu plus haut niveau, pour manipuler les images j'ai utilisé skimage, et pour tout ce qui est mathématique Numpy.

Après avoir étudié différents modèles et articles sur internet, je me suis basé principalement sur celui-ci[4] qui quasi-équivalent à la plupart des réseaux que j'ai trouvée pour ce genre problème.

Le réseau proposé est :

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 64, 64)	896
conv2d_2 (Conv2D)	(None, 32, 62, 62)	9248
max_pooling2d_1 (MaxPooling2	(None, 32, 31, 31)	0
dropout_1 (Dropout)	(None, 32, 31, 31)	0
conv2d_3 (Conv2D)	(None, 64, 31, 31)	18496
conv2d_4 (Conv2D)	(None, 64, 29, 29)	36928
max_pooling2d_2 (MaxPooling2	(None, 64, 14, 14)	0

[2] http://btsd.ethz.ch/shareddata/
[3] http://keras.io
[4] https://chsasank.github.io/keras-tutorial.html

dropout_2 (Dropout)	(None, 64, 14, 14)	0
conv2d_5 (Conv2D)	(None, 128, 14, 14)	73856
conv2d_6 (Conv2D)	(None, 128, 12, 12)	147584
max_pooling2d_3 (MaxPooling2	(None, 128, 6, 6)	0
dropout_3 (Dropout)	(None, 128, 6, 6)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_1 (Dense)	(None, 512)	2359808
dropout_4 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 62)	31806

Total params: 2,678,622
Trainable params: 2,678,622
Non-trainable params: 0

Pour expérimenter le modèle je l'ai d'abord appliqué à 4 classes.
Chaque classe est identifié par un identifiant.

- *Réseau de neurones convolutif "allégé"*

Les 4 classes utilisées



'25' '21' '19' '61'

Les images sont de tailles différentes, on doit les redimensionner (on choisit 48x48). Dans certains cas le ratio de l'image ne sera pas respecté, mais les déformations induites ne seront pas un problème, car elles permettent d'apporter plus de variations dans les images et les perspectives.

Après les avoir chargées on crée une matrice qui associe à chaque image son label. Pour ça on crée une matrice à laquelle on associe à chaque colonne un label, et à une ligne, une entrée.

Par exemple :

```
M = [ [0, 0, 0, 1],
      [0, 0, 1, 0]]
```

Signifie que pour M[0] appartient à la classe 3, et M[1] à la classe 2.

Le réseau de neurones prend donc un tableau d'image et cette matrice d'association image-label en entrée.

Ces 4 classes représentent 538 images, on en utilise 10% comme valeur de test pendant l'apprentissage. L'apprentissage se fait en 3 epochs, assez pour avoir des résultats satisfaisants tout en ayant un temps de calcul rapide.

On a environ ~90% de bons résultats.

- *Réseau de neurones convolutif "complet"*

Plutôt satisfait de ces résultats (le modèle a également été testé sur des données hors ensemble de test pour vérifier qu'il n'y avait pas de surapprentissage), j'ai essayé de faire des prédictions pour l'ensemble des classes, et les résultats n'ont pas été concluants (de l'ordre de 3-4% de bonnes prédictions).

Tout d'abord vu l'augmentation de la tailles des entrées et l'augmentation du nombre de epoch, le temps de calcul est naturellement plus long, ce qui rend difficile le debuggage et le "tuning" des différents paramètres. Pour donner une estimation, un epoch demande environ 10 minutes, et le nombre de epoch est passé de 3 à entre 10 et 20 (de plus je n'ai pas pu utiliser l'accélération GPU de mon ordinateur).

En étudiant les différentes estimations, j'ai pu constater que la plupart des erreurs d'estimation venaient de panneaux se ressemblant, le modèle semble alors avoir du mal à faire la distinction entre deux panneaux similaires.
Par exemple avec ce type de variante :

Le réseau ne donne pas assez d'importance à l'intérieur du panneau, et se concentre sur la forme et la couleur.

Je pense que ce comportement peut s'expliquer par le manque de données d'entraînement, car en moyenne on a (4700 / 62) = 75 images pour une classe, ce qui est relativement peu sachant que s'agissant d'une moyenne, certaines classes ont beaucoup moins de 75 images.
Un moyen d'atténuer ce problème serait de générer des variantes, par exemple avec des rotations, changer l'exposition, le contraste etc … Malheureusement je n'ai pas eu le temps de tester cela.

J'ai également pensé à changer de dataset, notamment à utiliser celui-ci http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset contenant 50 000 images pour 40 classes. Cependant le temps de calcul devenait trop important pour que je puisse vraiment l'expérimenter.

Bibliographie

Voici les articles dont je me suis servis :

https://hal.archives-ouvertes.fr/hal-00823162/file/Qualita2013_TT_AC.pdf

https://pdfs.semanticscholar.org/776f/a7abda970e0421b42b42cd2f0d043f777a0f.pdf

https://medium.com/@waleedka/traffic-sign-recognition-with-tensorflow-629dffc391a6

http://bartlab.org/Dr.%20Jackrit%27s%20Papers/ney/1.TRAFFIC_SIGN_Lorsakul_ISR.pdf

https://campushippo.com/lessons/build-a-tensorflow-traffic-sign-classifier-at-95-accuracy-214727f24

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html