# Chapitre 8
# Simuler les attributs de l'intelligence

## 1. Identifier visuellement les objets

### 1.1 Le problème

Même de façon purement statique, qu'un artifact se déplace ou pas, l'identification des objets présents dans son contexte immédiat peut s'avérer très utile.

Comme pour un humain, cette identification peut se révéler incertaine ou imprécise. Après une phase de reconnaissance d'un objet, purement basée sur son aspect visuel et la détermination d'une probabilité associée, on a parfois besoin d'acquérir d'autres informations sur cet objet.

Au repérage global de la position de l'objet correspondant à une image captée dans une direction donnée, on voudra alors appliquer une deuxième passe de reconnaissance pour préciser divers attributs de l'objet qui pourraient par exemple être sa taille, sa position dans un espace 3D, la direction de son éventuel mouvement, sa texture…

On tentera de conforter la détermination des objets ou de leurs attributs en croisant diverses informations et en tenant compte de ce que l'on connaît du contexte de la prise de vue, de la direction de cette prise de vue et des évènements précédents.

Quand on dispose de plusieurs sources d'informations complémentaires, un système de reconnaissance visuelle par ailleurs peu précis peut suffire pour des applications robotiques opérationnelles.

Le travail en deux passes permet de converger vers une reconnaissance plausible. Il est par exemple peu probable de croiser un lion dans sa cuisine ! Ce qui permet d'éliciter plus rapidement l'option *chat* lors d'une reconnaissance visuelle qui donnerait une certaine probabilité à *lion* et une autre probabilité raisonnable à *chat*. Bien entendu, cette appréciation serait à reconsidérer si la direction de la prise de vue était dirigée vers la télévision du salon !

Que cela soit pour reconnaître visuellement des objets, ou pour d'autres prises de décision, il est souvent utile de faire appel à plusieurs technologies **en parallèle** ou les unes après les autres (**en série**).

Rappelons que l'on peut confier les décisions finales d'agrégation de l'opinion des différents algorithmes ou agents à des systèmes experts flous **FBRS** (*Fuzzy rules based System*) ou intégrer certains résultats via les **réseaux ANFIS** (*Adaptive network based fuzzy inference system*) que nous avons évoqués dans le premier chapitre.

## 1.2 Éléments de l'état de l'art

### 1.2.1 Les images labellisées

Pour pouvoir entraîner les modèles, divers contributeurs ont collectionné de grands ensembles d'images. À ces images ont été ajoutés des *labels*, c'est-à-dire la mention de leur contenu.

Parfois, plusieurs contenus sont labellisés sur une seule image et un opérateur a marqué l'endroit où se trouve chacun des objets identifiés. Ce marquage se fait classiquement via la mention d'une **box**, c'est-à-dire d'un rectangle qui entoure l'objet dans l'image et dont on récupère les coordonnées pendant l'entraînement.

On trouve également des images où le contour de l'objet a été déterminé de façon plus précise (comme avec un **lasso**), ce qui est beaucoup plus puissant, mais plus difficile à exploiter.

Le problème de la détermination du contenu d'une image pour la faire labelliser (décrire) par un opérateur humain est en soi un problème plus compliqué qu'il n'y paraît, en effet la manière d'exprimer ce qu'elle voit est très variable d'une personne à l'autre. Pour décrire la présence d'un animal, on peut trouver des mentions aussi diverses que : chat, siamois…

Par ailleurs, on est fréquemment amené à déterminer les parties d'un objet (yeux, queue…) au sein d'un autre objet, ou des objets au sein d'une scène (chiens et chats jouant avec une balle), ou tout ensemble (la queue du chat de gauche) et même ajouter des attributs (un chat qui dort).

On comprend que deux façons de labelliser apparaissent : l'une se référant à une taxonomie bien identifiée (animal -> félin -> chat), l'autre en langage naturel ("un chat qui chasse en faisant semblant de dormir devant un trou de souris").

La première est la plus courante et nécessite peu de compétences en traitement du langage naturel. Elle permet de produire des bases de référence d'objet très fiables.

La deuxième permettra d'introduire une intelligence plus fine, contextualisée et le cas échéant émotionnelle dans l'apprentissage de la machine. Il existe d'ailleurs des datasets labellisant les sentiments évoqués dans des expressions humaines (joie, tristesse…).

Pour ce qui est des formes labellisées au lasso, la forme est parfois suggérée à l'opérateur par un traitement préliminaire sur l'image, qui identifie des contours ou des zones que l'on demandera à l'opérateur de labelliser.

Les packages comme **OpenCV**, que nous avons utilisé plus haut via l'API **Tensorlayers** au chapitre Améliorations du modèle, permettent en effet d'effectuer des traitements automatiques visant à extraire des zones significatives de l'image : la texture de la peau d'un chat est différente de la texture d'une voiture… donc apparaissent deux contours différents dans une image comportant un chat et une voiture que l'opérateur devra ensuite désigner comme tels.

Dans le même ordre d'idée, les zones colorées, de contrastes différents peuvent être proposées à des labellisations différentes.

Pour appréhender le cas des *box*, nous vous proposons d'étudier la façon dont a été construit l'*Open Images Dataset*, aujourd'hui dans sa version V5, au travers de l'article *The Open Images Dataset V4 : Unified image classification, object detection, and visual relationship detection at scale* (Kuznetsova et al. 2018).

La version actuelle étendue de ce dataset contient plus de **15 millions d'éléments tagués sur 600 catégories** et plus de 36 millions de labels au niveau image, étiqueter sur près de plus de 19 000 catégories. De plus, on y trouve plus de 390 000 annotations montrant des **relations entre labels de plus de 300 natures différentes**.

Voici quelques points de repère pour vous aider à naviguer dans ces données :

– Une catégorie possède un identifiant et un nom : /m/01nq_x , *Wild cat*
– Certaines catégories, pas toutes, peuvent déterminer une *box* comme :
  – /m/01yrx, *Cat*
  – /m/01lrl, *Carnivore*
  – /m/04rky, *Mammal*
– On trouve régulièrement la présence d'un héritage entre les catégories :

```
"LabelName": "/m/04rky",
"Subcategory": [
    {
        "LabelName": "/m/01lrl",
        "Subcategory": [

            {
                "LabelName": "/m/01yrx"
```

Il existe non seulement de nombreux datasets comportant des images pour nous aider, mais aussi de nombreux schémas de réseaux éprouvés.

### 1.2.2 Les réseaux neuronaux phares de la reconnaissance d'objets dans des images

Les principaux objectifs assignés à ces réseaux sont :

– la détection d'objets (un ou n objets sont présents, le cas échéant au sein d'une box) ;
– la segmentation sémantique (chaque pixel est affecté à une forme qui à terme possédera un label) ;
– le sous-titrage d'images (exemple : "un chat siamois qui guette une souris blanche" : **captioning**).

Il est intéressant de **tester les différents modèles classiques** sur des cas précis, de préférence dans leur ordre d'apparition, afin de bien assimiler comment les différentes améliorations ont été acquises.

Dans le *fork* suivant https://github.com/laudehenri/ImageModels du *GitHub* de Dan Dixey, on trouve la définition rédigée en syntaxe Keras de certains modèles de référence ayant trait aux traitements d'images.

Il est aisé d'intégrer ces définitions dans votre code et d'expérimenter ces modèles dans vos contextes. Nous vous conseillons de les tester dans l'ordre de leur création :

1. AlexNet (2012, conçu pour 2 GPU) et son clone CaffeNet (une GPU) ;

2. ZFNet (2013) ;

3. VGG19 (2014, VGG16 possède souvent de meilleures performances) ;

4. GoogLeNet (aussi nommé Inception v1, 2014, qui utilise la convolution 1 X 1).

Par ailleurs, le code de ResNet (v1 en 2015 et versions suivantes) est parfaitement documenté dans le tutoriel de base de Keras (https://keras.io/examples/cifar10_resnet/). Il vous faudra le parcourir attentivement.

Une implémentation très instructive via Keras de DenseNet (Huang, Liu, and Weinberger 2016), où une couche reçoit les *feature maps* des couches précédentes, se trouve dans le *fork* suivant https://github.com/laudehenri/DenseNet des travaux de Somshubra Majumdar.

Enfin, le *fork* (https://github.com/laudehenri/keras-efficientnets) implémente la très intéressante idée de rationaliser l'évolution conjointe des dimensions d'un réseau quand on le met à l'échelle, c'est-à-dire le *scaling* tel qu'il est décrit dans un article très instructif (Tan and Le 2019).

Pour faire le point, une revue de ces réseaux, très didactique, est disponible ici : (Jing and Tian 2019).

### 1.2.3 Un exemple d'implémentation

Le package **imageai** permet d'accéder à divers modèles déjà entraînés et sauvés en *.h5*, y compris à vos propres modèles, cela dans de bonnes conditions de performance et avec une interface homogène entre les modèles.

C'est un atout important en termes de gain de temps de conception, mais aussi dans le cas où l'on veut facilement effectuer une identification s'appuyant sur plusieurs modèles à la fois (la technique consistant à compulser plusieurs prédictions dans un modèle qui en fera la synthèse est nommée **ensemble** dans le jargon des *data scientist*).

On peut accéder au package à cette adresse :
https://github.com/OlafenwaMoses/ImageAI/

ImageAI ne nécessite pas de nombreuses dépendances "exotiques" et fonctionne en utilisant TensorFlow comme *backend*. La ligne de commande pour l'installer est la suivante :

```
pip3 install https:// github.com/OlafenwaMoses/ImageAI/releases/download/2.0.3
/imageai-2.0.3-py3-none-any.whl
```

Dans le petit exemple suivant, **imageai** est utilisé avec un modèle très peu puissant, mais très compact et donc rapide.

```
from imageai.Prediction import ImagePrediction
import os
home_path = os.getcwd()


p = ImagePrediction()

# utilisation de Squeezenet, le setModel permet de définit le type de modèle
p.setModelTypeAsSqueezeNet()
p.setModelPath(os.path.join(home_path, "squeezenet_weights_tf_dim_ordering_tf
_kernels.h5"))
```