P7- POC
Détection d'objets et Segmentation d'images

Courtillon Richard | OpenClassroom Projet 7 | le 31/10/2021

Introduction

Le but de ce projet est de présenter un rapport sur le projet P7 du parcours 'Machine Learning' d'OpenClassroom, à savoir un P.O.C. (Proof of Concept) sur un sujet identifié.

Le sujet choisi est le suivant : la détection d'objets et la segmentation d'images. Il est en quelque sorte la suite logique du P6 sur la classification d'images, et provient directement d'un besoin métier réel.

Le besoin métier :

Une entreprise de traitement de déchets papier située en Ile de France a besoin, avant recyclage, d'identifier, pour chaque remplissage de bacs, quel est la part de papier de la part d'autres déchets.

Il s'agira donc d'établir un traitement automatique d'images source, qui devra produire en sortie un pourcentage identifié de déchets papier par rapport aux autres déchets présents.

Le projet va donc être décomposé en plusieurs étapes. Dans ce rapport, nous ne traiterons que la partie POC, c'est-à-dire une première analyse avec des premiers résultats, sans traiter de la totalité du projet, qui nécessiterait plus de temps.

1. Identifier un ou plusieurs algorithmes appropriés, et en sélectionner un au final
2. Récupérer un modèle déjà pré entraîné et écrire ou modifier un code existant pour l'adapter à notre besoin
3. Premiers tests et résultats
4. Récupérer un dataset différent annoté sur le traitement de déchets, et entraîner le modèle pour voir si cela améliore les résultats, ou créer notre propre dataset d'entraînement, à annoter.

Sommaire

I. Présentation des derniers algorithmes de computer vision

Rappel du besoin : il s 'agit d'aller un peu plus loin que la simple détection d'objets dans une image. Le besoin est ici de séparer les objets de classe papier des autres classes d'objets, et d'estimer en pourcentage leur proportion respective.

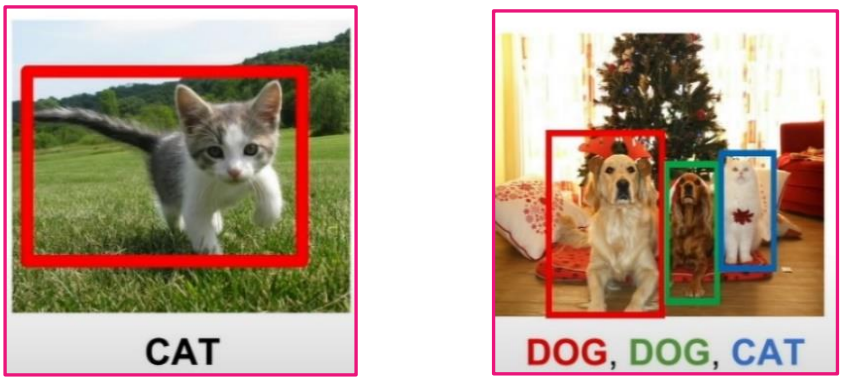
I.1 Quelles approches ?

On distingue plusieurs approches dans la 'computer vision' :

- **Image classification :**
  - On a une image en entrée dont il faut prédire en sortie la classe d'appartenance

Exemple de classifieur de race de chiens.

- **Simple object detection :**
  - On identifie les objets appartenant à une des classes de départ par un rectangle (localisation + taille), que l'on appellera bbox, associés à la classe prédite.
  - Le nombre d'objets distincts à prédire est à priori inconnu et dépend de l'image :



Object detection, un 'objet' / plusieurs 'objets'
(Source ici)

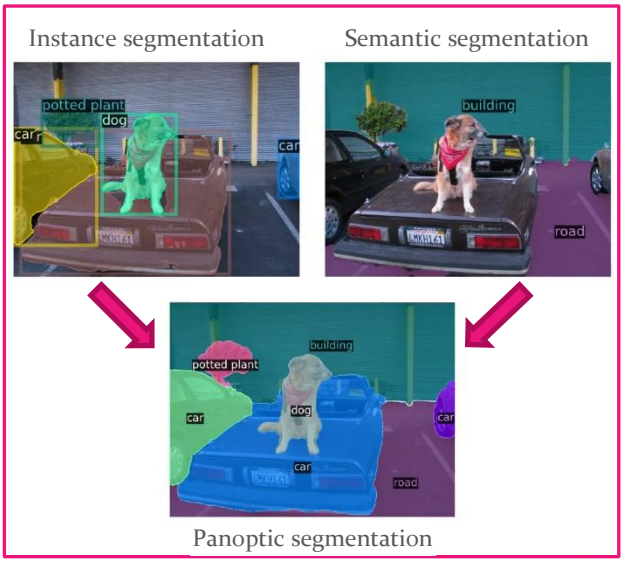
- **Semantic segmentation :**
    - Ici, on ne parle pas d'identification d'objets au sens propre, mais d'identification de pixels associés chacun à une classe spécifique. Il n'y a pas de notion de localisation d'objets, juste une classification d'appartenance de chaque pixel.

Semantic segmentation (source ici)

- **Instance segmentation :**
    - Ici, on associe des données de localisation et de classification d'objets avec une vue pixel par pixel. On aura donc en sortie, par image, une liste d'objets, de bbox associées (taille + position), et un masque représentant en noir et blanc l'objet en question : chaque pixel de la bbox est donc associé à la catégorie identifiée ou pas.
    - On peut en déduire aisément, comme dans le cas précédent, le nombre de pixels de l'objet identifié.

Instance segmentation (source ici)

- **Panoptic segmentation :**
    - C'est une fusion de l'instance segmentation et de la semantic segmentation.
    - Chaque pixel doit faire l'objet d'une attribution à une classe



Panoptic segmentation (source ici)

I.2 Quelles techniques ?

I.2.1. YOLO & YOLO R

Ce sont des algorithmes pointus de détection d'objets **en temps réel**.

YOLO V1 to V5 : 'You only Look Once'

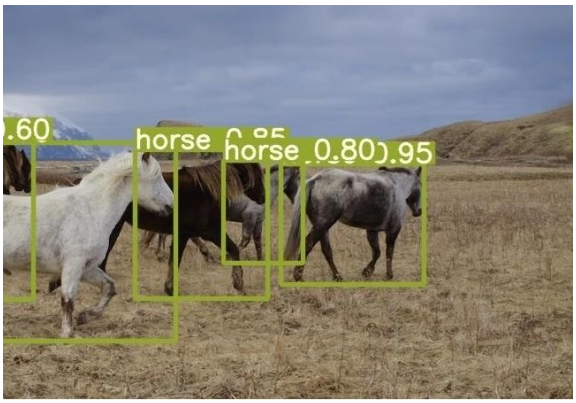
YOLO est un algorithme populaire de détection d'objets, qui a fait l'objet de plusieurs évolutions ces 5 dernières années :

- V1 introduite en 2016., V2 en 2017, et V3 en 2018, par le même auteur issu de l'université de Washington *Joseph Redmon*. Lien vers les papiers officiels des différentes versions :
    - Version 1
      'You Only Look Once: Unified, Real-Time Object Detection' (2016)
    - Version 2
      'YOLO9000: Better, Faster, Stronger' (2017)
    - Version 3
      'YOLOv3: An Incremental Improvement' (2018)

- YOLO V4 (Février 2020)
    - Auteur : Alexey Bochkovskiy et al.
    - Source vers le papier officiel : 'YOLOv4: Optimal Speed and Accuracy of Object Detection' (Avril 2020.)

- YOLO V5 : Juin 2020
    - Auteur : Glenn Jocher

YOLO-R: "You Only Learn One Representation"

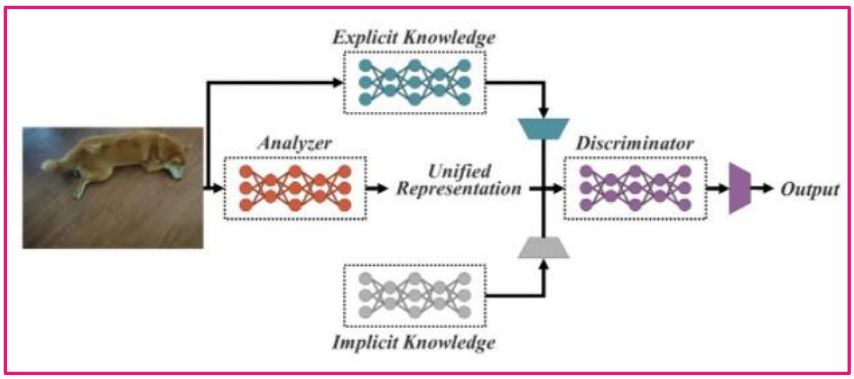


Exemple d'output du YOLO-R (source ici)

- Papier de recherche : [2105.04206] You Only Learn One Representation: Unified Network for Multiple Tasks (arxiv.org)
- Code source : https://github.com/WongKinYiu/yolor
- Sorti en mai 2021

C'est le plus récent des algorithmes de détection d'objets **en temps réel**, créé par des chercheurs au sein de cet institut à Tai Wan : *'Taiwanese Institute of Information Science, Academia Sinica, and Elan Microelectronics Corporation of Taiwan'.*

YOLO R s'inspire des algorithmes précédents YOLO, mais va combiner une approche explicite et une approche implicite, essayant ainsi de copier le fonctionnement humain, dont les décisions sont faites de décisions conscientes et inconscientes.



Représentation schématique du YOLO-R (source ici)

Ses résultats sont comparables en termes de précision (AP : Average Precision) avec le YOLO V4, mais il est beaucoup plus rapide que ses prédécesseurs. Source : le 'Readme.md' du *github*.

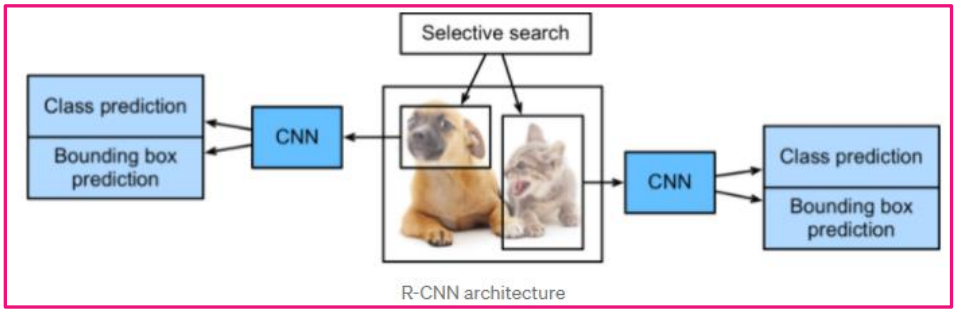
I.2.2. Les algorithmes R-CNN

La famille des algorithmes R-CNN (Region-based Convolutional Neural Networks) part du premier modèle de détection d'objets, le R-CNN, qui a ensuite été amélioré au fil du temps.

- R-CNN -> Fast R-CNN -> Faster R-CNN
- Ils permettent de faire de la **détection d'objets** : B-box + classification

Le R-CNN

- Papier officiel : https://arxiv.org/pdf/1311.2524.pdf



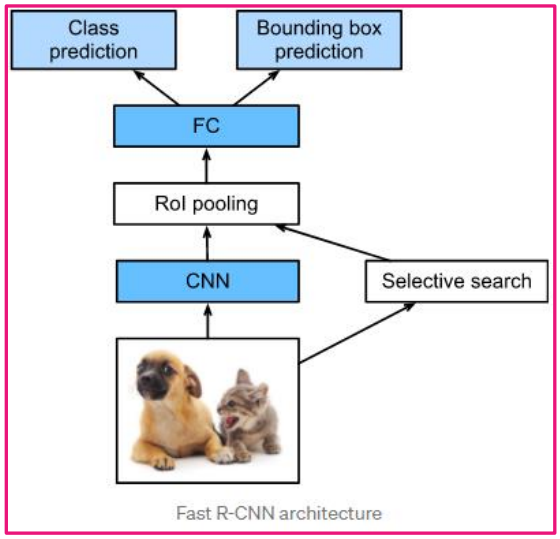
R-CNN (source ici)

Le principe est le suivant : Le R-CNN effectue une sélection de régions (selective search), dont chaque output, appelée ROI (Region of Interest), fera l'objet via réseau CNN :

- d'une prédiction de classe
- via régression d'une estimation de la taille et position associée.

Première amélioration : Le Fast R-CNN

- Papier officiel : https://arxiv.org/abs/1504.08083



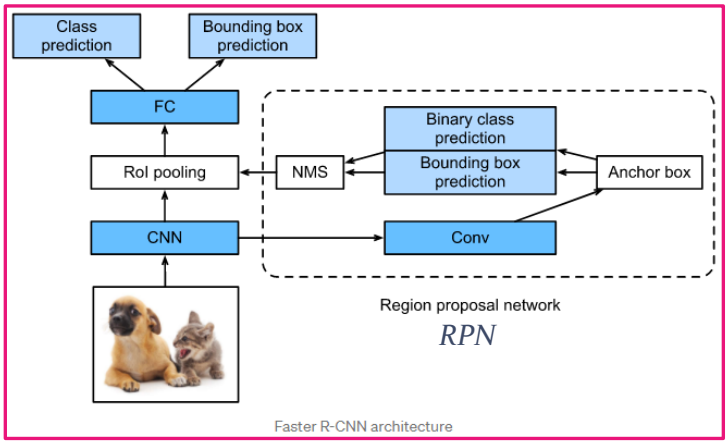
Fast R-CNN (source ici)

Le Fast R-CNN, comme son nom l'indique, vise à réduire les temps de traitement liés principalement au traitement séparé de chaque Région identifiée. Dans le Fast R-CNN, l'image dans son ensemble est envoyée dans un CNN et dans un algorithme de selective search pour identification des ROI et une première classification basique.

La couche ajoutée de 'ROI pooling' sert à uniformiser la taille des ROI en sortie du premier CNN avant d'entrer dans la seconde couche FC pour régression (prédiction des bbox) et classification.

Deuxième amélioration : Le Faster R-CNN

- Papier officiel : https://arxiv.org/abs/1506.01497

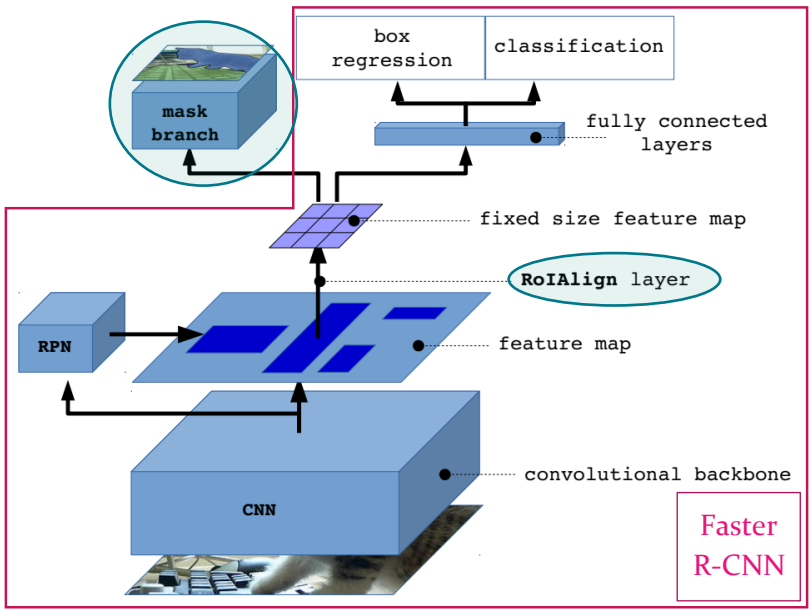


Faster R-CNN (source ici)

L'algorithme de recherche de régions 'selective search' est ici remplacé par un algorithme dénommé RPN (Region Proposal Network), plus rapide, plus sélectif et donc plus efficace que son prédécesseur.

Dernière extension : Le Mask R-CNN

- Papier de recherche officiel : [1703.06870] Mask R-CNN (arxiv.org)
- Auteurs : Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick



Mask R-CNN (source ici)

Le Mask R-CNN est un Faster R-CNN, auquel on rajoute **une brique supplémentaire de prédiction de masque**.

Il permet de faire de la 'instance segmentation' en fournissant en sortie, en plus des outputs du Faster R-CNN, **le masque de chaque objet détecté** :

- Chaque pixel de la ROI est classifié en 1 ou 0.
- Il s'agit donc d'une classification binaire pixel par pixel, pour chaque ROI
- Chaque pixel est passé à travers une couche FC avec une activation de type sigmoide (classification binaire) et une minimisation de la fonction de perte (binary cross entropy loss function)

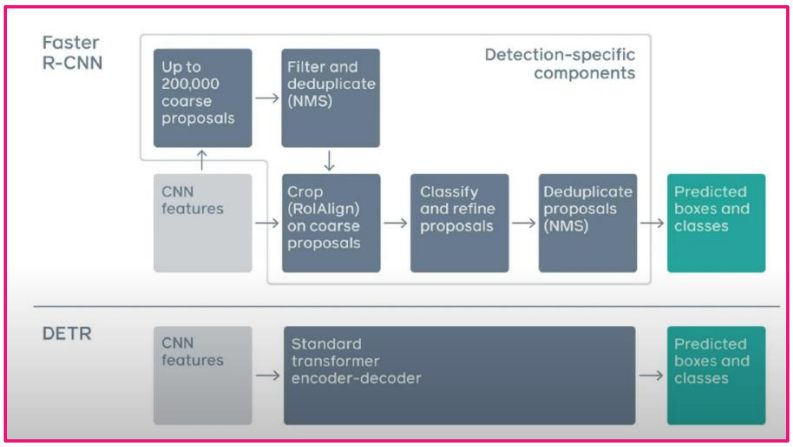
Autre différence par rapport au Fast RCNN : La brique 'ROI pooling' est remplacée par une brique 'ROI align', plus précise dans la réduction de dimension qu'elle effectue, ce qui permet à la couche 'mask' d'être plus performante.

En récupérant ce masque, nous pourrons en calculer le nombre de pixels et donc la proportion, but initial du projet.

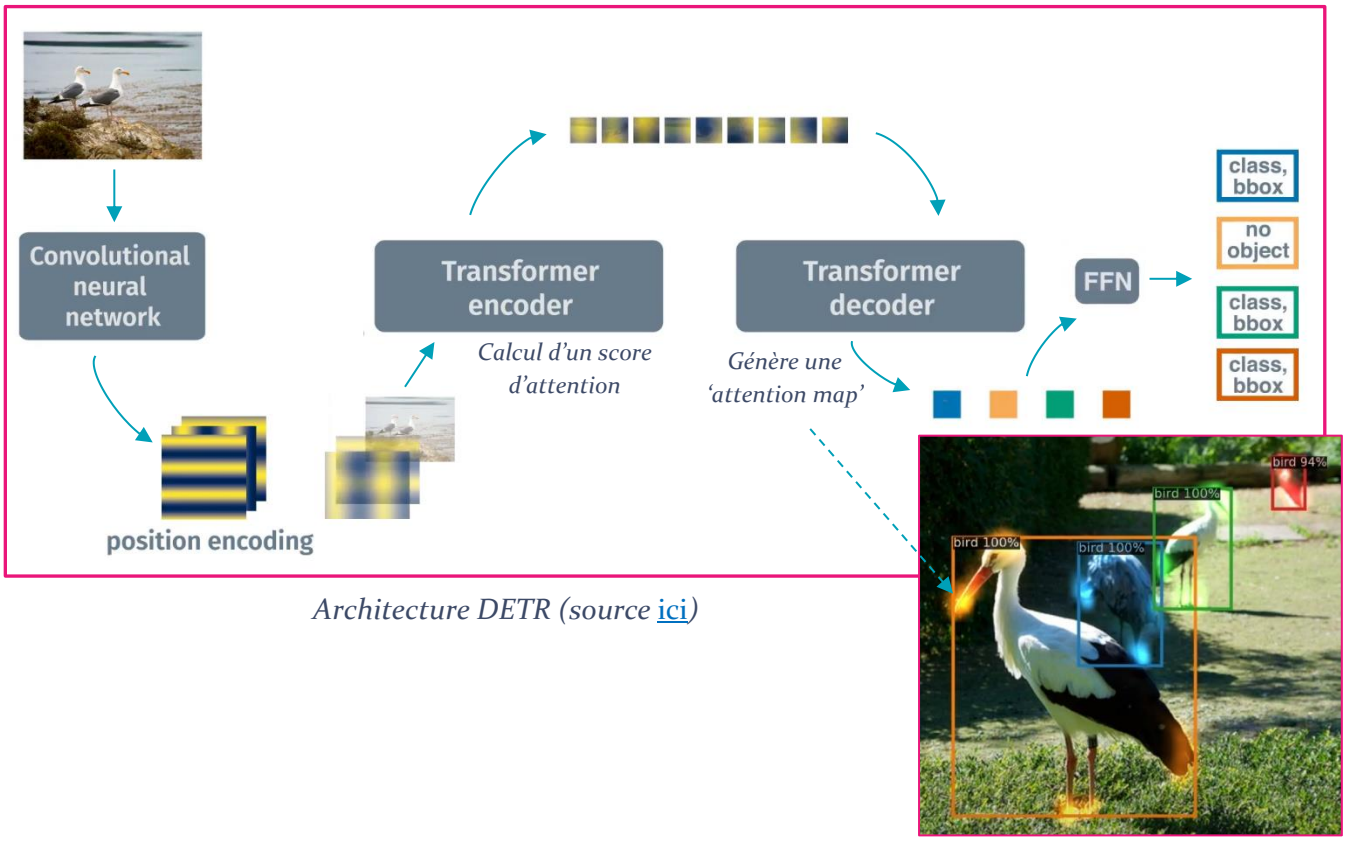
I.2.3. L'algorithme DETR de Facebook AI

DETR = **DE**tection **TR**ansformer: "End-to-End Object Detection With Transformers"

- Algorithme développé par Facebook AI.
  - Code disponible ici sous Github
  - Présentation ici : DETR: End-to-End Object Detection With Transformers (alcinos.github.io)
- Algorithme qui effectue de la détection d'objets et de la panoptic segmentation.
  - Il utilise une prédiction d'ensemble (pas de pipeline différente pour la bbox et la classification)
  - Plus efficace sur des objets larges que les autres algorithmes, et moins sur les petits objets
  - Utilise un transformer (encodeur + décodeur)



Comparaison Faster R-CNN / DETR (source ici)



Architecture DETR (source ici)

II. Sélection de l'algorithme et premiers tests

Pour notre besoin, l'algorithme retenu au final est le Mask R-CNN, qui permet de faire de l'instance segmentation et de récupérer l'élément masque afin de calculer les pixels de chaque masque, puis de chaque classe.

Une application pratique du **Mask R-CNN** a été mise à disposition par l'entreprise américaine **Matterport** via la mise à disposition d'un code de démonstration, basé sur un modèle déjà pré entraîné sur un dataset **COCO** avec des classes prédéfinies.

C'est ce code qui est réutilisé et adapté à notre besoin :

- Source du Github de Matterport et readme.md :
  - matterport/Mask_RCNN: Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow (github.com)
- Modèle pré-entraîné : https://github.com/matterport/Mask_RCNN/releases/download/v2.0/mask_rcnn_coco.h5

II.1 Premiers tests et résultats

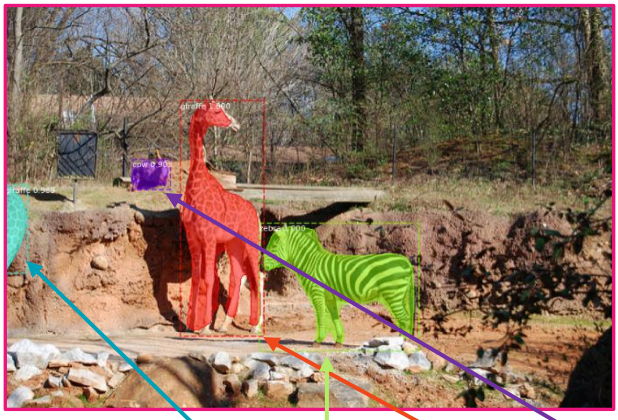
Les premiers tests ont été effectués en utilisant le modèle pré entraîné sans modification des couches supérieures, et sans modification des classes préexistantes. Le module 'model.py' a été partiellement modifié sur certaines lignes pour le rendre compatible avec les dernières versions de Tensorflow et Tensorflow.Keras.

Quelques fonctions ont ensuite été écrites pour extraire les différents masques des différentes catégories et calculer le ratio de pixels de chaque catégorie p.r. à la totalité des catégories listées (hors background).

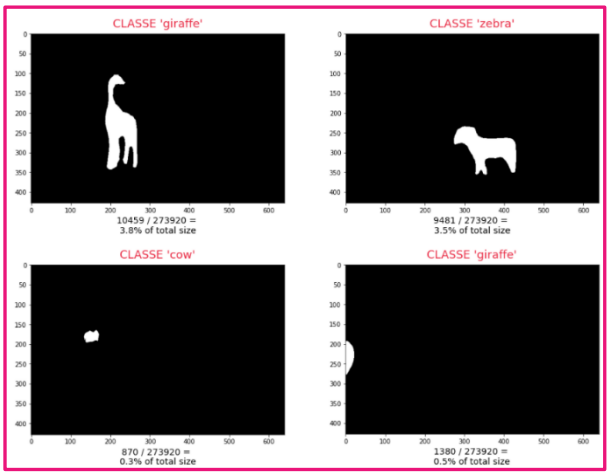
Test 1 :

Image en Input

Visualisation des objets détectés



```
classes: ['giraffe', 'zebra', 'giraffe', 'cow']
```



Visualisation des masques des différents objets détectés

class	px	ratio (%)
cow	870	3.92
giraffe	11839	53.35
zebra	9481	42.73

Résultats : proportion de la surface occupée (px) de chaque classes en %

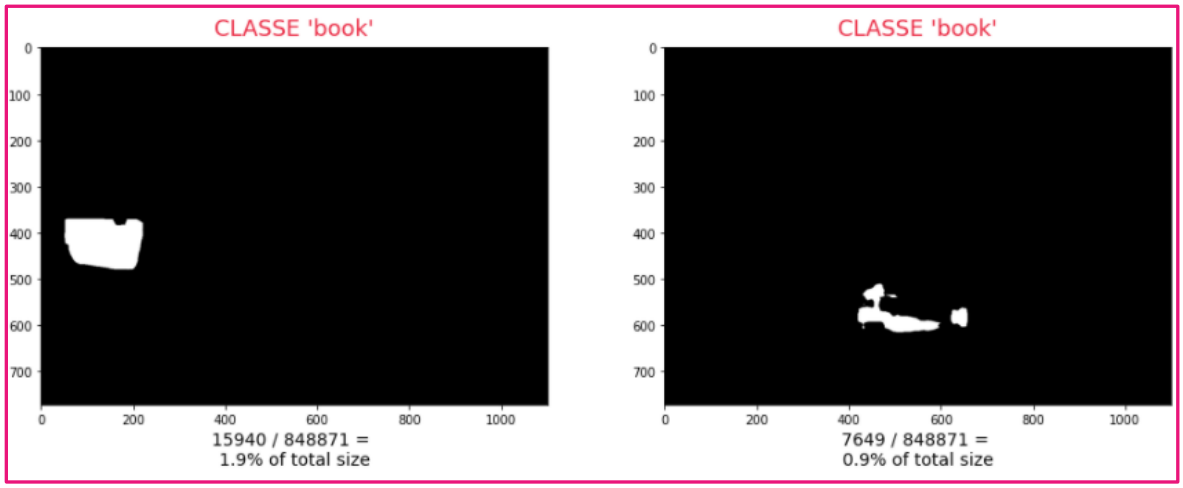
Test 2 :

Les résultats sont par contre décevants quand on l'applique à des déchets papier, étant donné que le modèle ne connaît pas la classe papiers. Il reconnaît ce qui s'en approche, et de manière très parcellaire, à savoir la catégorie 'livre / book' :

Image en Input

Visualisation des objets détectés

Visualisation des masques des différents objets détectés



class	px	ratio (%)
book	23589	100.0

Résultats : proportion de la surface occupée (px) de chaque classes en %

II.2 Pour la suite

Les étapes suivantes sont des pistes d'implémentation et d'amélioration qui n'ont pas été réalisées dans le cadre de ce POC.

Pour répondre à notre besoin précis d'identification de déchets papier p.r. à d'autres déchets (carton, verre, plastique), il va être nécessaire de :

- **1.** Récupérer une banque d'images existantes et annotée spécialisée dans les déchets, et contenant déjà une classe papier
- **Ou**
- **2.** Créer sa propre banque d'images de déchets la plus proche de la réalité (photos prises sur le terrain), annotés précisément

L'annotation d'une image comprend les données suivantes :

- Largeur, hauteur et format de l'image
- Pour chaque objet de l'image :
    - Sa classe
    - Son masque en pixel en format 'polygone' (une liste de points identifiés par leurs coordonnées [x1, y1, x2, y2, x3, y3, ...])
    - La position exacte de sa bbox, identifiée par : position x et y du coin supérieur gauche, largeur et hauteur en px

On commencera par **l'option 1.**, à savoir récupérer une banque d'images de déchets existante, en l'occurence TACO :

- Github : pedropro/TACO: 🧢 Trash Annotations in Context Dataset Toolkit (github.com)
- Papier official : https://arxiv.org/abs/2003.06975

Le but sera ensuite d'entraîner le modèle sur ce jeu de données et de faire des tests sur des images de déchets papier/ carton etc.

Si les résultats ne sont pas satisfaisants, on pourra passer à **l'option 2**, beaucoup plus consommatrice de temps, à savoir constituer un ensemble d'images s'approchant au plus près des photos qui seront prises dans l'entreprise de recyclage de déchets. Il faudra dans ce cas annoter chaque image, ce qui est très consommateur de temps, avant de procéder à l'entraînement du modèle.

Conclusion

Il existe, au sein de la computer vision, de multiples approches : détection d'objets, segmentation sémantique, segmentation d'instances, panoptique, etc.

Il existe de multiples algorithmes récents effectuant de la détection d'objets et pour certains de la segmentation d'images, les plus récents étant **YOLO-R**, **Mask R-CNN**, et **DETR**.

On a choisi dans le cadre de ce POC d'adapter un besoin réel d'une entreprise de recyclage de déchets, à savoir identifier avant dépôt la part de papier dans les déchets reçus. Pour cela, il est nécessaire d'effectuer de la segmentation d'instances, et Mask R-CNN a été retenu, en partie parce qu'il est facilement adaptable via un modèle déjà entraîné et un code qu'on peut réutiliser.

Ceci dit, il reste encore à réentraîner le modèle pour l'adapter à notre besoin et notamment aux classes spécifiques de déchets (bouteille, plastique, papier, cartons, etc.). Pour cela, on peut partir de banques d'images déjà existantes et annotées telles que TACO, ou, si cela ne permet pas d'obtenir des résultats convaincants, de créer notre propre banque d'images à annoter, ce qui est beaucoup plus fastidieux, avant entraînement du modèle.