

Projet COTS

Participation à un concours Kaggle

Efkan TUREDI

Introduction

- Notre objectif est de réaliser un algorithme qui permet d'aider à la reconnaissance des "COTS", une race d'animal sous marin qui menace les étoiles de mer
- Nous allons donc utiliser des méthodes de computer vision notamment des algorithmes YOLO
- Nous allons faire un tour succinct des méthodes disponibles en computer vision, et nous allons largement nous appuyer sur des kernels Kaggle
- Lien vers le concours Kaggle : <https://www.kaggle.com/c/tensorflow-great-barrier-reef/overview>

Préambule

Qu'est ce que la détection d'objets ?

- Consiste à avoir une image avec plusieurs objets et de renvoyer un ou plusieurs objets avec un label pour chaque conteneur
 - Implique l'utilisation de technique de classification d'image et de localisation d'objets
- Une base de donnée très connue dans ce domaine est la base de données ImageNet qui est très souvent utilisé pour des concours dans cette exercice

Problématique et métrique

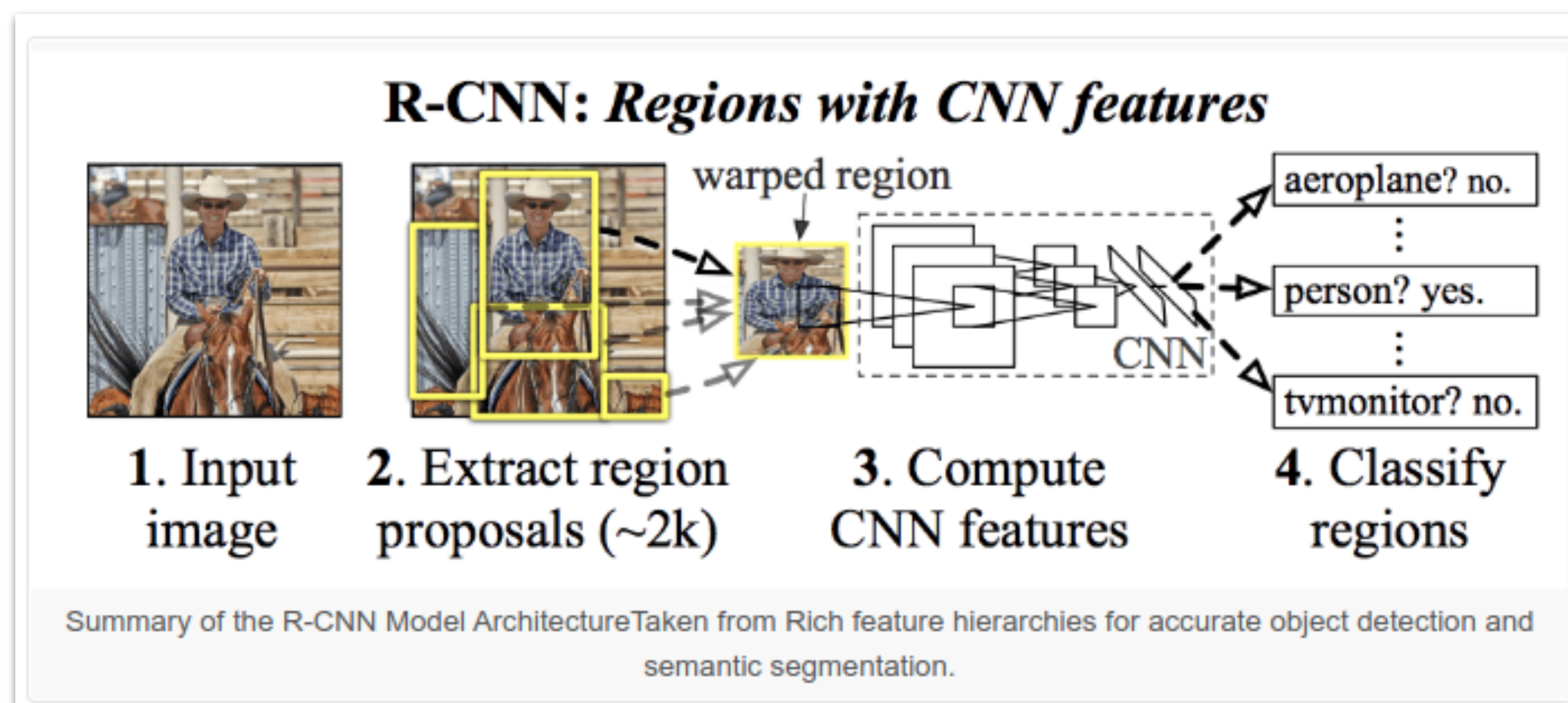
- Nous souhaitons développer un système de machine learning qui permet de détecter les proliférations de COTS
 - Notre base de données préparées contient des photos de 3 différentes vidéos. Nous ferons notre entraînement avec ce de dataset
 - L'exercice nous impose de choisir le score F2 comme mesure de performance. Donc voici la formule générale:
- Le score F2 donne plus de poids au recall qu'à la précision
 - Il est acceptable d'avoir un peu plus de faux positifs, plutôt que de rater des COTS

$$F_{\beta} = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

Passage en revue des méthodes

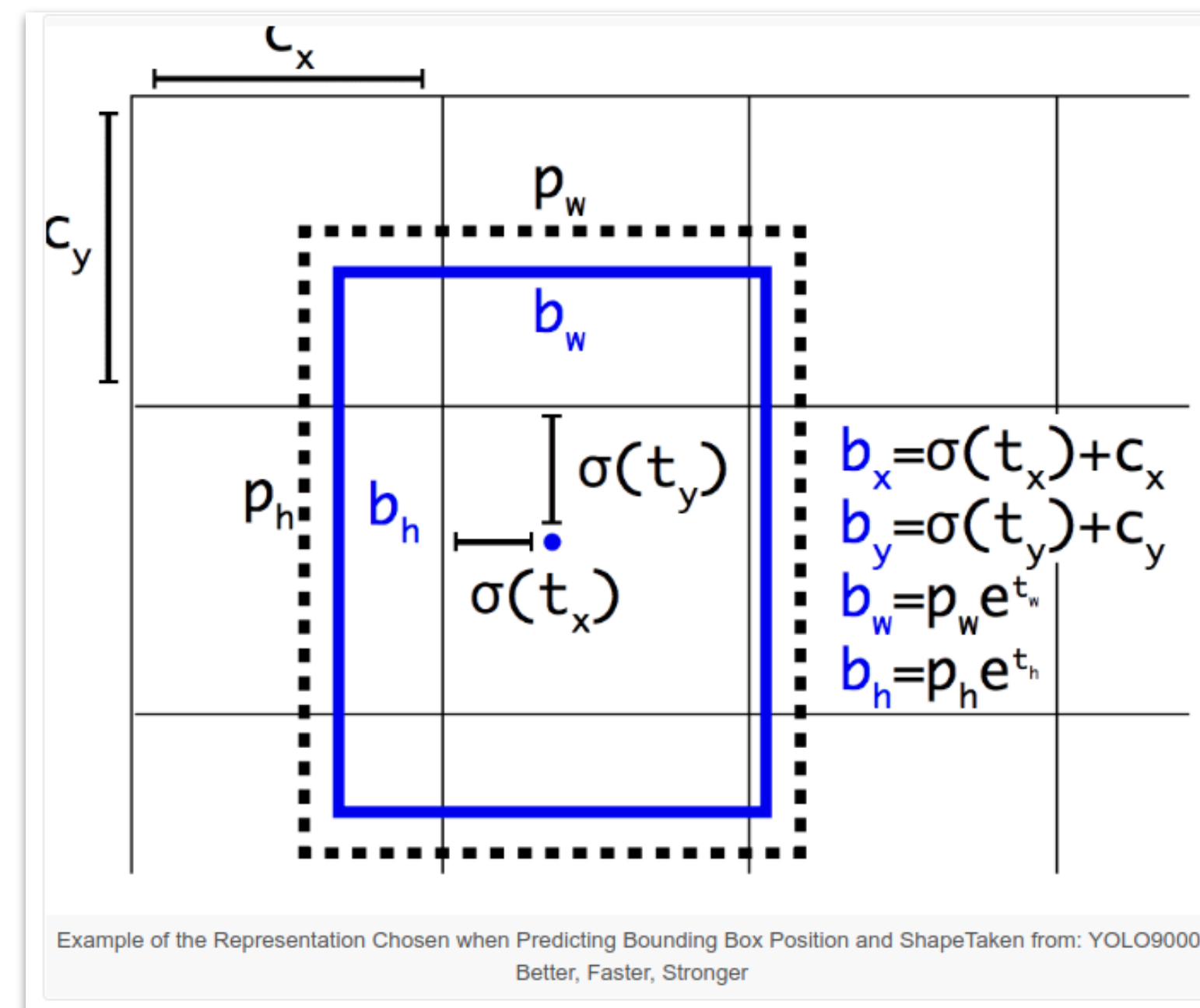
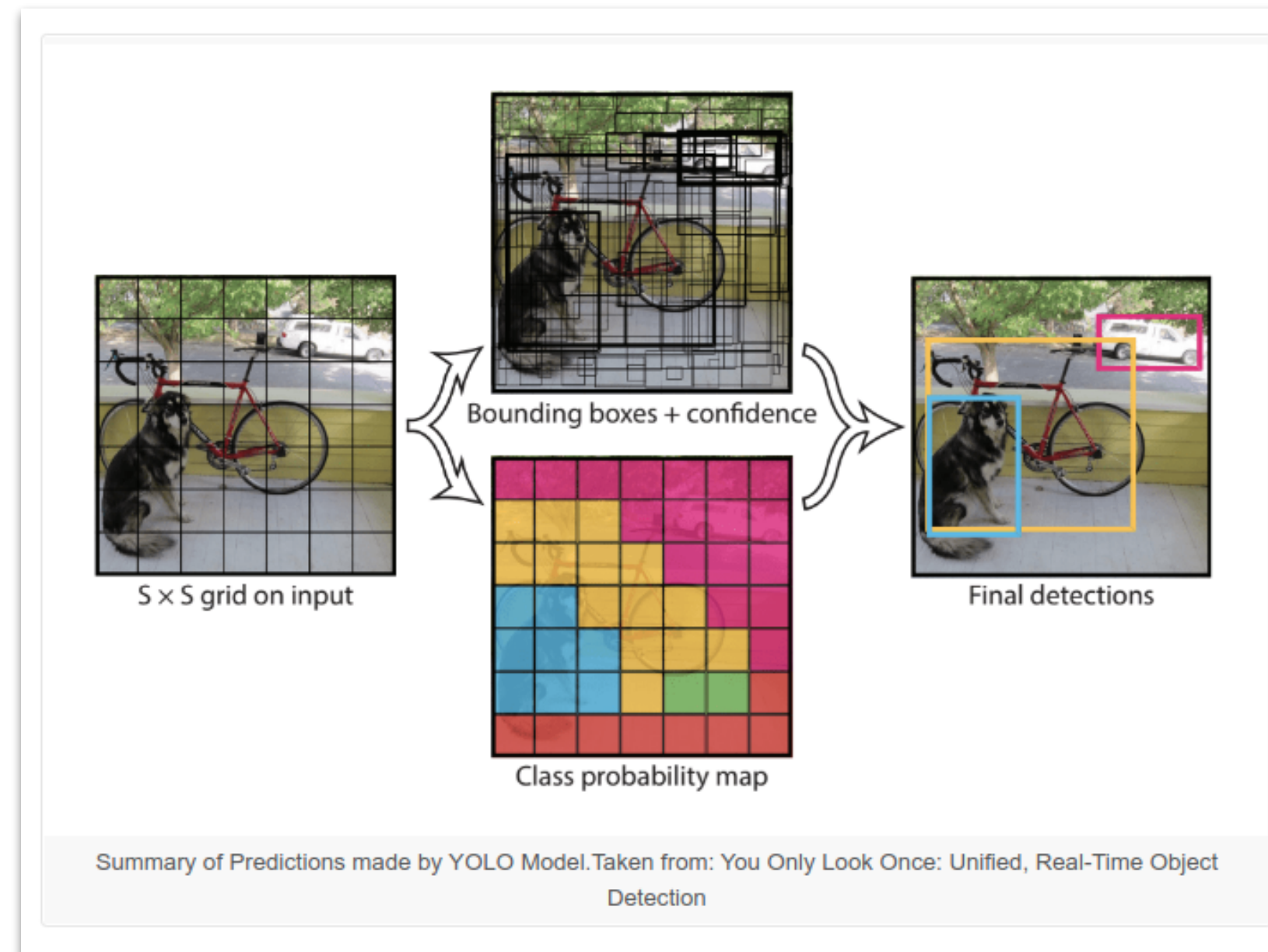
La famille des R-CNN

- Cette famille repose sur les modules suivants:
 - Module 1: Proposition des régions. Génération des contours candidats
 - Module 2: Extractions des caractéristiques avec l'aide d'une CNN
 - Module 3: Classification avec un softmax, Logit ou SVM
- L'inconvénient majeur de cette méthode est sa lenteur, même avec de grosses ressources de calculs



- Des variantes ont été développées par la suite: Fast R-CNN et Faster R-CNN
- Les variantes reposent sur les mêmes principes mais ajoutent des modules supplémentaires

La famille YOLO



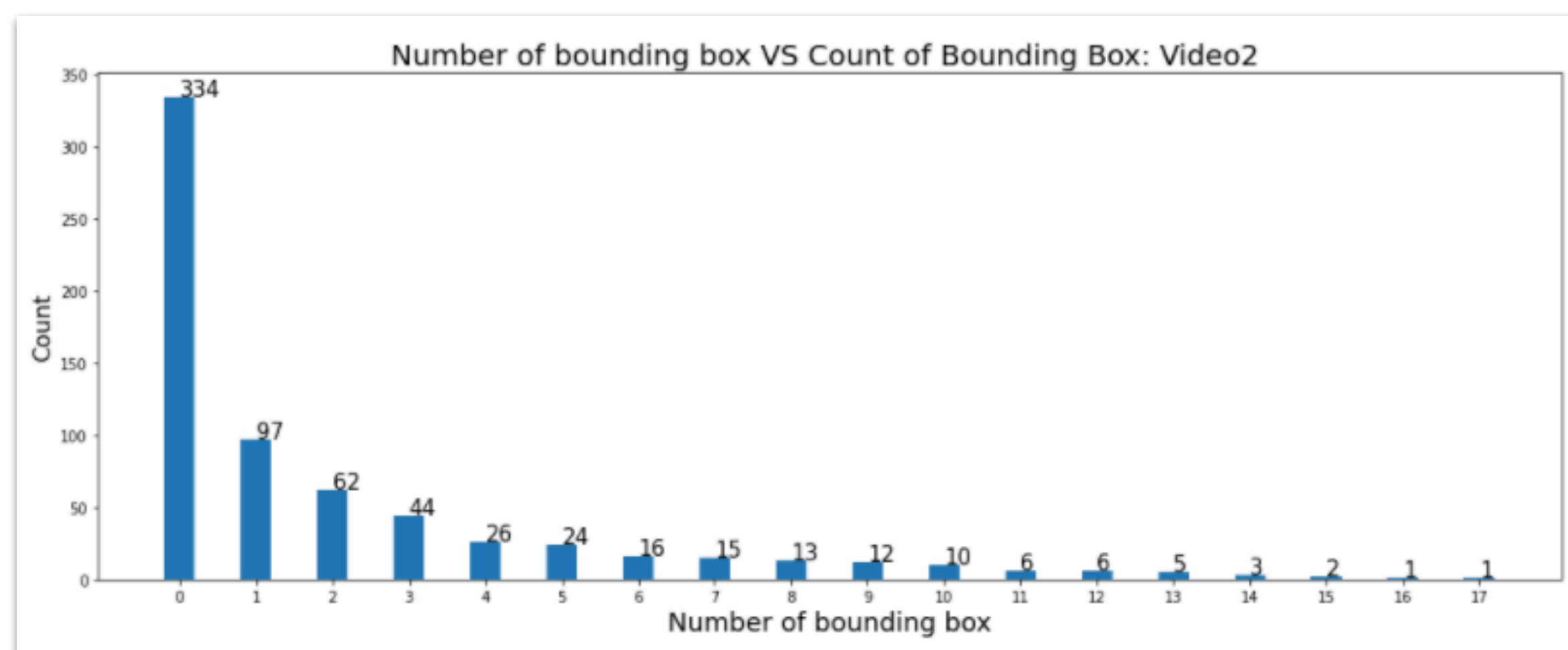
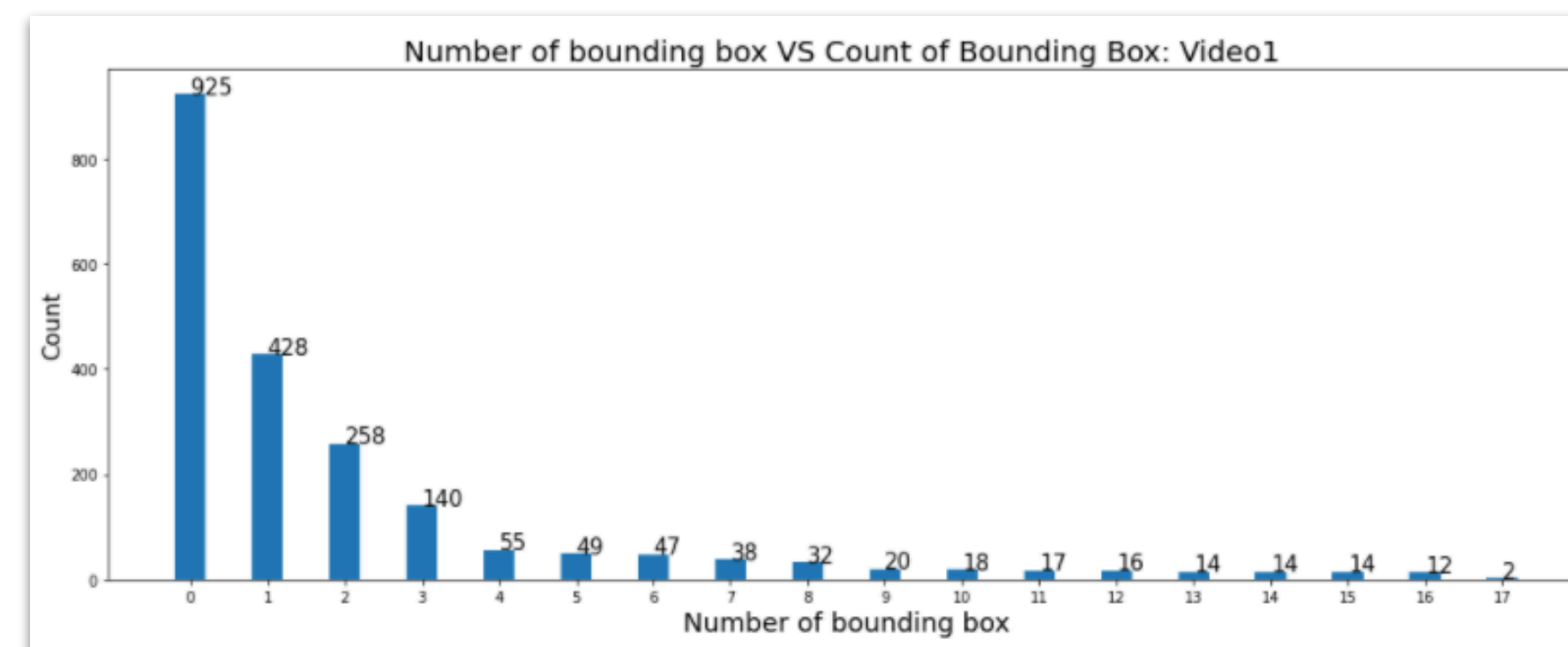
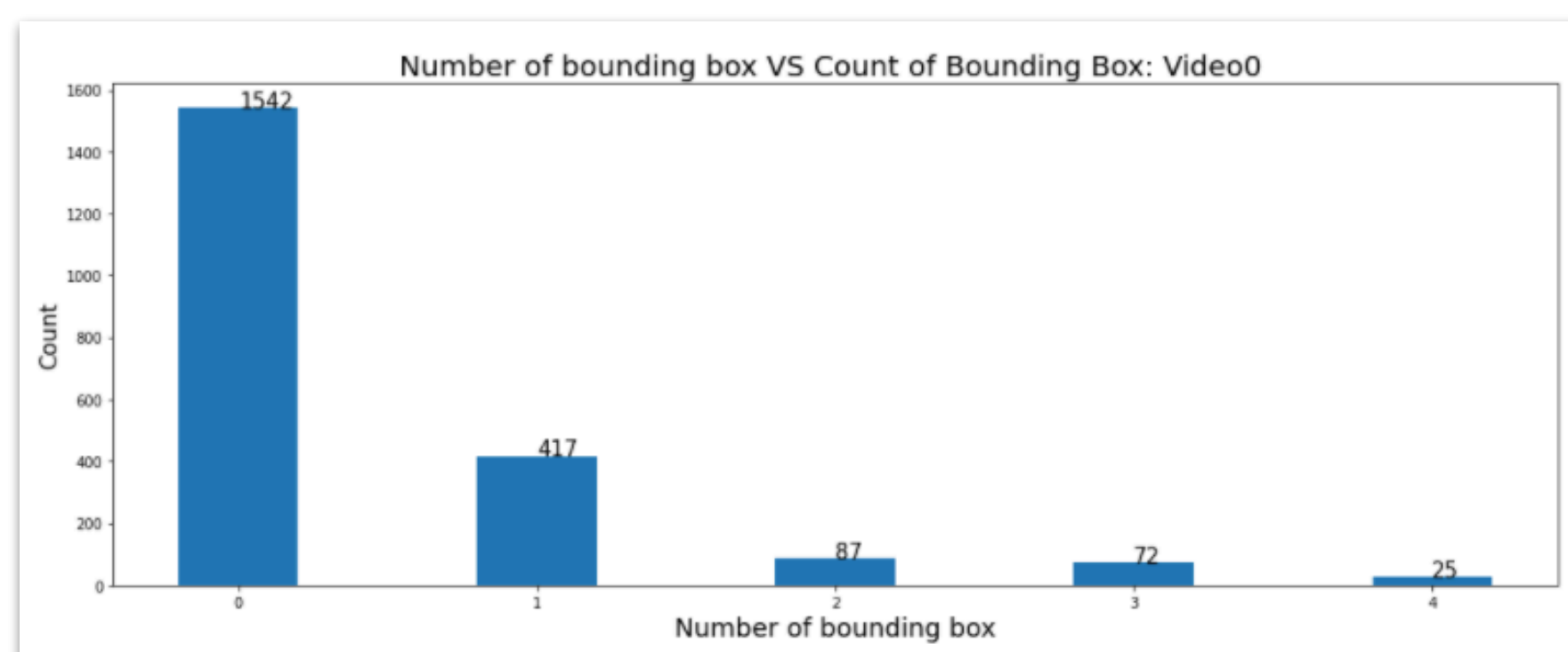
- La famille YOLO est née de la nécessité d'avoir des modèles plus rapides
- Il y a désormais 5 variantes du modèle initial
- YOLO a permis des avancées majeures en termes d'utilisation du computer vision

Les frameworks YOLO

- Nous décidons d'utiliser la dernière version de YOLO développée par Ultralytics, qui nous permet de l'utiliser rapidement avec PyTorch: <https://docs.ultralytics.com/>
 - Une autre possibilité aurait été d'utiliser le framework Darknet pour l'entraînement. Celui ci écrit en C++, est très rapide et relativement simple d'utilisation
 - La très grosse majorité de la communauté Kaggle utilise la framework d'Ultralytics, nous décidons de faire ce choix aussi
- Il a plus de 23k images dans notre dataset, mais seul 5k d'entre eux contiennent des COTS. Une bonne idée est de nous concentrer sur cet échantillon réduit.
 - Les vidéos ne sont pas équilibrées en terms de COTS. La vidéo 0 a très peu d'images en contenant

Dataset

Détails sur les vidéos



- Il a plus de 23k images dans notre dataset, mais seul 5k d'entre eux contiennent des COTS. Une bonne idée est de nous concentrer sur cet échantillon réduit.
- Les vidéos ne sont pas équilibrées en terms de COTS. La vidéo 0 a très peu d'images en contenant

Format CoCo vs YOLO

- Il existe déjà des fonctions permettant de faire la conversion entre les deux formats, ce qui facilite notre tâche

```
In [11]: # check https://github.com/awsaf49/bbox for source code of following utility functions
from bbox.utils import coco2yolo, coco2voc, voc2yolo
from bbox.utils import draw_bboxes, load_image
from bbox.utils import clip_bbox, str2annot, annot2str

def get_bbox(anns):
    bboxes = [list(annot.values()) for annot in anns]
    return bboxes

def get_imgsize(row):
    row['width'], row['height'] = image_size.get(row['image_path'])
    return row

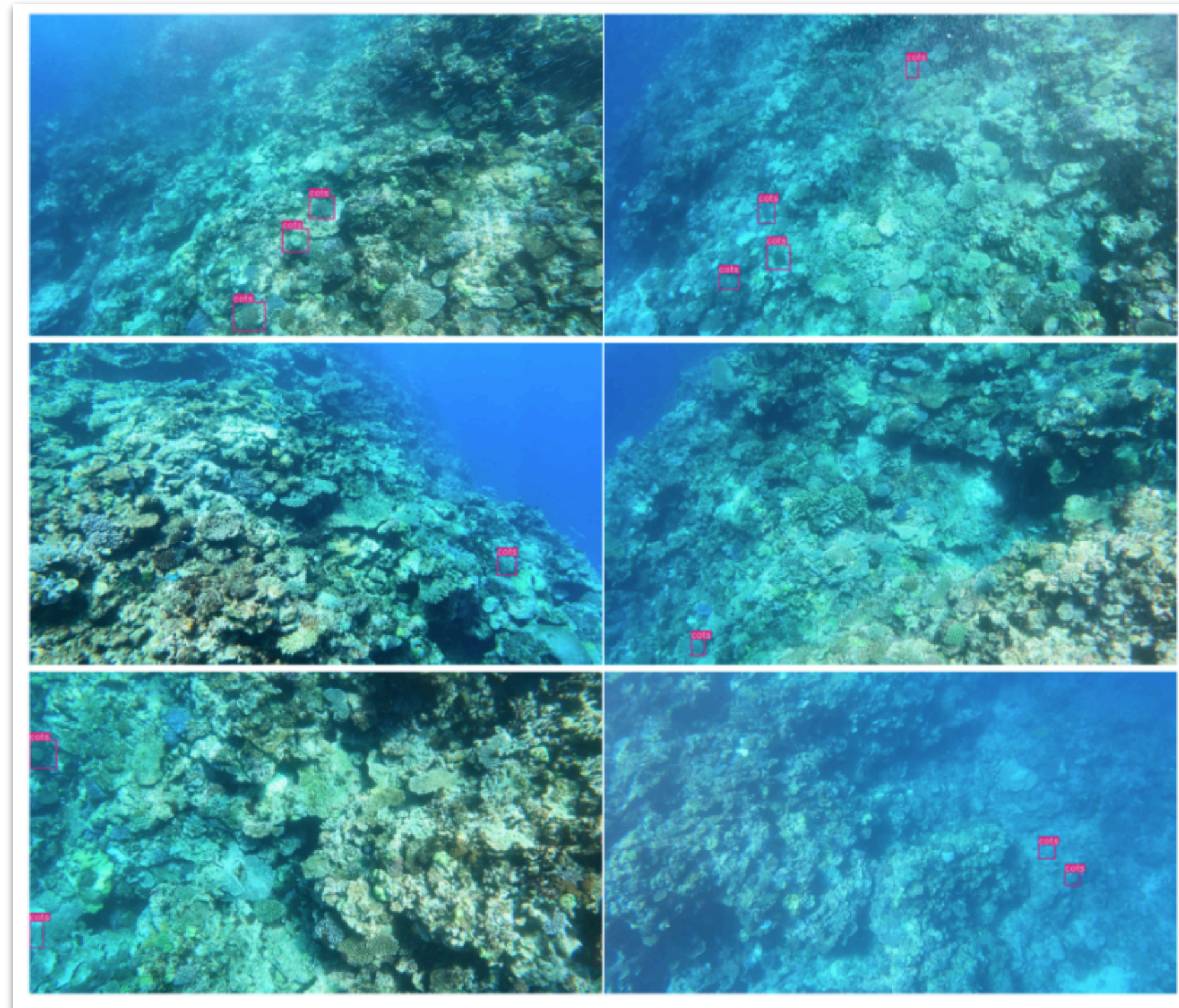
np.random.seed(32)
colors = [(np.random.randint(255), np.random.randint(255), np.random.randint(255))\
          for idx in range(1)]
```

- Notre dataset initial est au format Coco, qui correspond au format très connu d'un dataset public
- Les algorithmes YOLO nécessitent d'avoir le dataset dans un format spécifique aussi

Implémentation

Création des border box

- Une fois la conversion au format YOLO réalisée nous pouvons regarder la position relative de nos border box sur les images d'entraînement



- Les borders boxes sont des conteneurs qui identifient la nature et la position d'une détection
- Ils ont une place importante dans notre entraînement car nous allons regarder la superposition de border box

L'entraînement

```
In [24]: %cd /kaggle/working
!rm -r /kaggle/working/yolov5
# !git clone https://github.com/ultralytics/yolov5 # clone
!cp -r /kaggle/input/yolov5-lib-ds /kaggle/working/yolov5
%cd yolov5
%pip install -qr requirements.txt # install

from yolov5 import utils
display = utils.notebook_init() # check
```

YOLOv5 🚀 2022-1-18 torch 1.9.1 CUDA:0 (Tesla P100-PCIE-16GB, 16281MiB)

Setup complete ✅ (2 CPUs, 15.6 GB RAM, 3201.8/4030.7 GB disk)

🚀 Training

```
In [25]: !python train.py --img {DIM}\
--batch {BATCH}\
--epochs {EPOCHS}\
--optimizer {OPTIMIZER}\
--data /kaggle/working/gbr.yaml\
--hyp /kaggle/working/hyp.yaml\
--weights {MODEL}.pt\
--project {PROJECT} --name {NAME}\
--exist-ok
```

```
%%writefile /kaggle/working/hyp.yaml
lr0: 0.001 # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.1 # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937 # SGD momentum/Adam beta1
weight_decay: 0.0005 # optimizer weight decay 5e-4
warmup_epochs: 3.0 # warmup epochs (fractions ok)
warmup_momentum: 0.8 # warmup initial momentum
warmup_bias_lr: 0.1 # warmup initial bias lr
box: 0.05 # box loss gain
cls: 0.5 # cls loss gain
cls_pw: 1.0 # cls BCELoss positive_weight
obj: 1.0 # obj loss gain (scale with pixels)
obj_pw: 1.0 # obj BCELoss positive_weight
iou_t: 0.20 # IoU training threshold
anchor_t: 4.0 # anchor-multiple threshold
# anchors: 3 # anchors per output layer (0 to ignore)
fl_gamma: 0.0 # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015 # image HSV-Hue augmentation (fraction)
hsv_s: 0.7 # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4 # image HSV-Value augmentation (fraction)
degrees: 0.0 # image rotation (+/- deg)
translate: 0.10 # image translation (+/- fraction)
scale: 0.5 # image scale (+/- gain)
shear: 0.0 # image shear (+/- deg)
perspective: 0.0 # image perspective (+/- fraction), range 0-0.001
flipud: 0.5 # image flip up-down (probability)
fliplr: 0.5 # image flip left-right (probability)
mosaic: 0.5 # image mosaic (probability)
mixup: 0.5 # image mixup (probability)
copy_paste: 0.0 # segment copy-paste (probability)
```















- Nous installons les packages nécessaires pour utiliser le framework
- A noter que nous nous concentrons uniquement sur les paramètres les plus connues pour l'entraînement de nos modèles: IoU, Learning Rate, etc...

Kernel d'inférence

- Nous faisons le choix de faire un kernel d'entraînement et un kernel d'inférence
- Plusieurs motivations pour ce choix:
 - Réduire le temps de calculs par Kernel
 - Permet de séparer le travail sur les paramètres d'inférence
 - Permet de partager chaque solution de manière modulaire avec la communauté Kaggle

Résultats

Les résultats finaux

Submission and Description	Status	Private Score	Public Score	Use for Final Score
Great-Barrier-Reef: YOLOv5 [infer]  (version 7/7) 14 hours ago by Efkan Turedi Notebook Great-Barrier-Reef: YOLOv5 [infer]  Version 7	Succeeded 	0.562	0.571	<input type="checkbox"/>
Great-Barrier-Reef: YOLOv5 [infer]  (version 6/7) a day ago by Efkan Turedi Notebook Great-Barrier-Reef: YOLOv5 [infer]  Version 6	Succeeded 	0.577	0.500	<input type="checkbox"/>
Great-Barrier-Reef: YOLOv5 [infer]  (version 5/7) a day ago by Efkan Turedi Notebook Great-Barrier-Reef: YOLOv5 [infer]  Version 5	Succeeded 	0.582	0.592	<input type="checkbox"/>
Great-Barrier-Reef: YOLOv5 [infer]  (version 4/7) 2 days ago by Efkan Turedi Notebook Great-Barrier-Reef: YOLOv5 [infer]  Version 4	Succeeded 	0.313	0.477	<input type="checkbox"/>
Great-Barrier-Reef: YOLOv5 [infer]  (version 3/7) 17 days ago by Efkan Turedi Init test : YOLOv5 [infer]  Version 3	Succeeded	0.588	0.627	<input type="checkbox"/>



- Notre première soumissions utilise les poids d'un autre utilisateur ayant entraîné sur une plateforme autre que Kaggle avec une RAM plus élevé
- Notons l'importance du matériel permettant d'avoir un algo mieux entraîné


Annexes

Liens Kernel

- Kernel 1: <https://www.kaggle.com/awsaf49/great-barrier-reef-yolov5-train>
- Kernel 2: <https://www.kaggle.com/awsaf49/great-barrier-reef-yolov5-infer>
- Kernel 3: <https://www.kaggle.com/soumya9977/learning-to-sea-underwater-img-enhancement-eda>

Participation communauté


Great-Barrier-Reef: YOLOv5 [infer] 






1

Comment 17 days ago on notebook

Hello, practical question: What is the length of the submission time? Been waiting for 1h30 now, and its still "scoring"

Thanks a lot !




Great-Barrier-Reef: YOLOv5 [infer] 




1

Comment 17 days ago on notebook

For those interested in the answer, it's roughly 4h !



Great-Barrier-Reef: YOLOv5 [train] 





1

Comment a day ago on notebook

Wonderful job on this ! Thanks a lot for sharing job with us




Great-Barrier-Reef: YOLOv5 [train] 



586

Updated 1mo ago

103 comments · TensorFlow - Help Protect the Great Barrier Reef +1



Gold

...