AVANTIX

Avancement Simulateur IQ

AVANTIX

YOLOv8 fonction de coût

GBB & ProbIoU (1)

Source : Gaussian Bounding Boxes and Probabilistic Intersection-over-Union for Object Detection (2021)

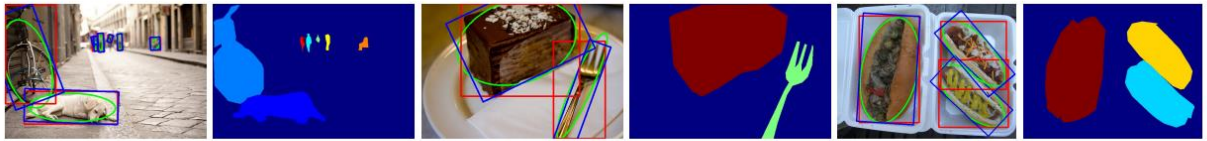


Figure 1: Different annotation types for images in the COCO dataset: HBBs (red), OBBs (blue) and GBBs (green), as well as the GT segmentation masks.

xywhr → xyabc où a,b et c sont les paramètres de l'ellipse inscrite dans l'OBB

Définition GBB avec a>0, b>0 et ab-c² >0: $$\Sigma = \begin{bmatrix} a & c \\ c & b \end{bmatrix} = R_\theta \begin{bmatrix} a' & 0 \\ 0 & b' \end{bmatrix} R_\theta^T = \begin{bmatrix} a' \cos^2\theta + b' \sin^2\theta & \frac{1}{2}(a' - b') \sin 2\theta \\ \frac{1}{2}(a' - b') \sin 2\theta & a' \sin^2\theta + b' \cos^2\theta \end{bmatrix}$$

Passage OBB → GBB :
Covariance d'une région de l'image $$\Sigma = \frac{1}{N} \int_{\boldsymbol{x} \in \Omega} (\boldsymbol{x} - \boldsymbol{\mu}) (\boldsymbol{x} - \boldsymbol{\mu})^T$$

Implique a' = $w^2/12$, b' = $h^2/12$

GBB & ProbIoU (2)

Fonction de coût : Hellinger Distance

$$H_D(p,q) = \sqrt{1 - B_c(p,q)}$$

$$B_C(p,q) = \int_{\mathbb{R}^2} \sqrt{p(\boldsymbol{x})q(\boldsymbol{x})}d\boldsymbol{x},$$

Pseudo code :

a_gt, b_gt, r_gt = obb2gbb(whr_gt)
a_pred, b_pred, r_pred = obb2gbb(whr_pred)

bc = exp(
    - t1(a_gt, b_gt, r_gt, a_pred, b_pred, r_pred, xy_gt, xy_pred) -
    t2(a_gt, b_gt, r_gt, a_pred, b_pred, r_pred, xy_gt, xy_pred) -
    0,5*ln(t3(a_gt, b_gt, r_gt, a_pred, b_pred, r_pred, xy_gt, xy_pred))
)
hd = sqrt(1 – bc)

ProbIoU = 1 - hd

GBB & ProbIoU (3)

Extrait du papier :

1. $\mathcal{L}_1(\boldsymbol{p}, \boldsymbol{q}) = H_D(\boldsymbol{p}, \boldsymbol{q}) = 1 - \text{ProbIoU}(\boldsymbol{p}, \boldsymbol{q}) \in [0, 1]$,
2. $\mathcal{L}_2(\boldsymbol{p}, \boldsymbol{q}) = B_D(\boldsymbol{p}, \boldsymbol{q}) = -\ln\left(1 - \mathcal{L}_1^2(\boldsymbol{p}, \boldsymbol{q})\right) \in [0, \infty]$,

However, $\mathcal{L}_1$ might produce values very close to one when the GBBs are far from each other, which might lead to very small gradients and lead to slow convergence. On the other hand, $\mathcal{L}_2$ does not present this limitation, but it is not geometrically related to an IoU measure. Hence, we suggest to start training with $\mathcal{L}_2$ and then switch to $\mathcal{L}_1$.

Dans le code d'ultralytics seul L1 est implémentée.

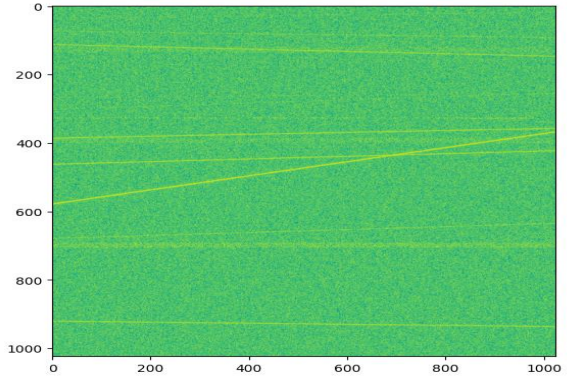
TODO : Tester avec L2 ? Non car utilisation de Pixels-IoU

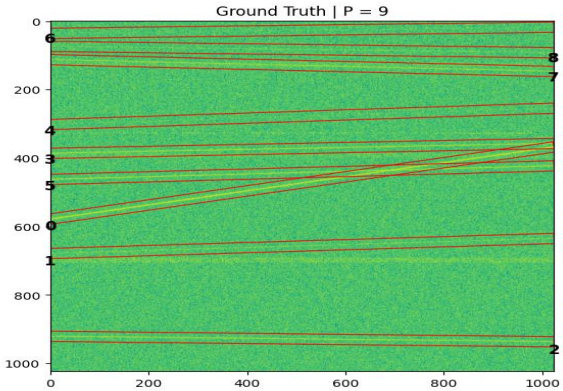
GBB & ProbIoU (4)

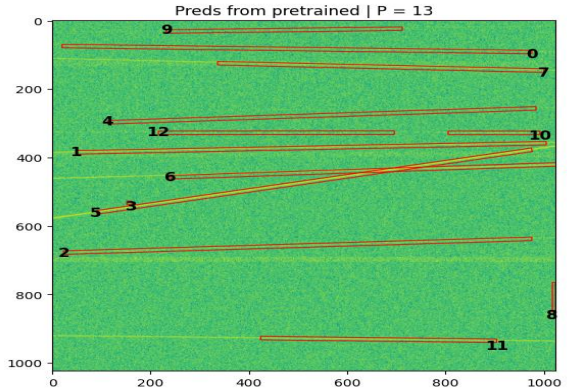
Extrait du papier :

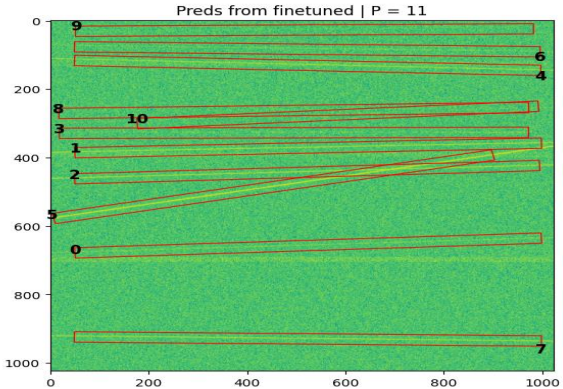
Another issue concerns very thin elongated objects, for which a or b might be very small (consider the axis-aligned GBB for simplicity). In these cases, the Bhatacharrya Distance might produce very large gradients for a or b when the compared GBBs are not aligned (see the first term in Eq. (16) and its derivative w.r.t. a and b), which might cause instabilities in the training step and compromise convergence for this kind of objects.

Test en diminuant le rapport d'aspect des CWs







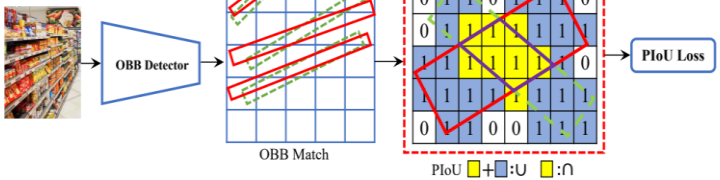


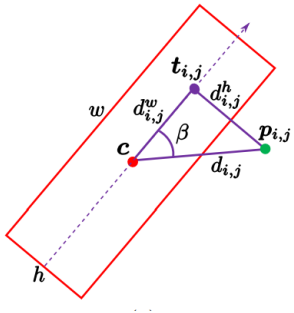
Cela corrige même les NaN !

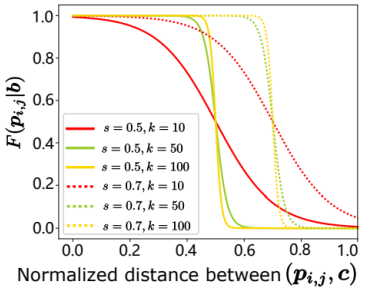
Pixels-IoU (1)

Source : [PIoU Loss: Towards Accurate Oriented Object Detection in Complex Environments (2020)]

To overcome the limitations of existing OBB-based approaches, we encourage the community to adopt more robust OBB detectors in a shift from conventional aerial imagery to more complex domains. We collected a new benchmark dataset, *Retail50K*, to reflect the challenges of detecting oriented targets with high aspect ratios, heavy occlusions, and complex backgrounds. Experiments show that the proposed frameworks with PIoU loss not only have promising performances on aerial images, but they can also effectively handle new challenges in Retail50K.







Pixels-IoU (2)

$$\delta(\boldsymbol{p}_{i,j}|\boldsymbol{b}) = \begin{cases} 1, & d^w_{i,j} \le \frac{w}{2}, d^h_{i,j} \le \frac{h}{2} \\ 0, & otherwise \end{cases}$$

$$d_{ij} = d(i,j) = \sqrt{(c_x - i)^2 + (c_y - j)^2}$$
$$d^w_{ij} = |d_{ij} \cos\beta|$$
$$d^h_{ij} = |d_{ij} \sin\beta|$$
$$\beta = \begin{cases} \theta + \arccos\frac{c_x - i}{d_{ij}}, & c_y - j \ge 0 \\ \theta - \arccos\frac{c_x - i}{d_{ij}}, & c_y - j < 0 \end{cases}$$

$$S_{\boldsymbol{b}\cap\boldsymbol{b}'} = \sum_{\boldsymbol{p}_{i,j}\in B_{\boldsymbol{b},\boldsymbol{b}'}} \delta(\boldsymbol{p}_{i,j}|\boldsymbol{b})\delta(\boldsymbol{p}_{i,j}|\boldsymbol{b}')$$

$$S_{\boldsymbol{b}\cap\boldsymbol{b}'} \approx \sum_{\boldsymbol{p}_{i,j}\in B_{\boldsymbol{b},\boldsymbol{b}'}} F(\boldsymbol{p}_{i,j}|\boldsymbol{b})F(\boldsymbol{p}_{i,j}|\boldsymbol{b}')$$

Approximation différentiable

$$F(\boldsymbol{p}_{i,j}|\boldsymbol{b}) = K(d^w_{i,j}, w)K(d^h_{i,j}, h)$$

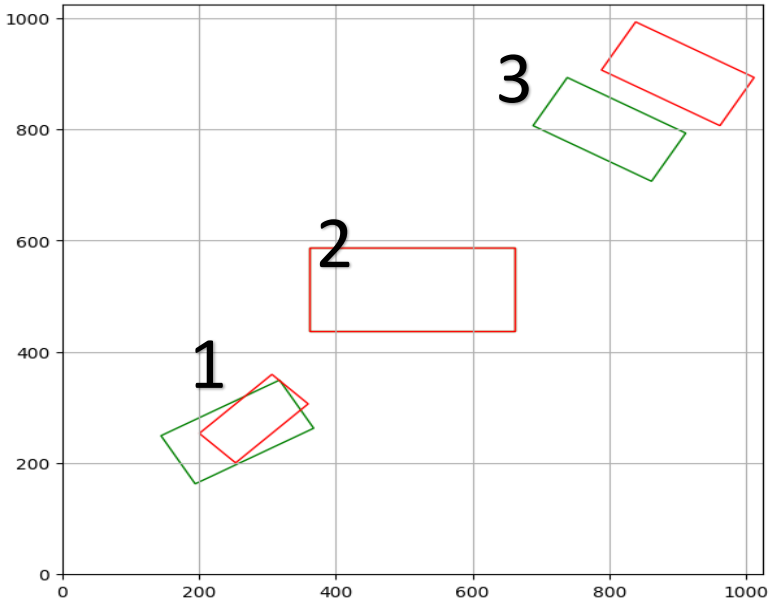
$$K(d,s) = 1 - \frac{1}{1 + e^{-k(d-s)}}$$

$$S_{\boldsymbol{b}\cup\boldsymbol{b}'} = w \times h + w' \times h' - S_{\boldsymbol{b}\cap\boldsymbol{b}'}$$

$$PIoU(\boldsymbol{b},\boldsymbol{b}') = \frac{S_{\boldsymbol{b}\cap\boldsymbol{b}'}}{S_{\boldsymbol{b}\cup\boldsymbol{b}'}}$$

Pixels-IoU (3)

Implémentation de la loss function et debug sur image ci-dessous




```python
import time

import torch
import numpy as np

from ultralytics.utils.metrics import pixelsiou, probiou

obbs_gt = torch.tensor(
    np.array(
        [   # x    y     w   h      r
            [256, 256, 200, 100, 30 / 180 * np.pi],  # partially overlaps with pred[0]
            [512, 512, 300, 150, 0],                 # fully overlaps with pred[1]
            [800, 800, 100, 200, 60 / 180 * np.pi],  # overlaps with none
        ]
    )
).to('cuda:1')

obbs_pred = torch.tensor(
    np.array(
        [   # x    y     w   h      r
            [280, 280, 150,  75, 45 / 180 * np.pi],  # partially overlaps with gt[0]
            [512, 512, 300, 150, 0],                 # perfect overlap with gt[1]
            [900, 900, 100, 200, 60 / 180 * np.pi],  # overlaps with none
        ]
    )
).to('cuda:1')


### ProbIoU ###
start = time.time()
probiou = probiou(obbs_gt, obbs_pred).cpu().numpy()
print(f"ProbIoU = {probiou.reshape(3,)}")
print(f"TIme elapsed = {time.time() - start}")


### PixelsIoU ###
start = time.time()
pixelsiou = pixelsiou(obbs_gt, obbs_pred).cpu().numpy()
print(f"PixelsIoU = {pixelsiou}")
print(f"TIme elapsed = {time.time() - start}")
```

✓ 0.0s

```
ProbIoU = [    0.66611     0.99955    0.029374]
TIme elapsed = 0.0035181045532226562
PixelsIoU = [    0.36192     0.99006     0.11169]
TIme elapsed = 0.008699417114257812
```

AVANTIX

Autres travaux

Autres travaux

- BT 202 : 10k spectro avec fs=1GSps et nperseg=512 et apprentissage en 1024x1024
- BT 203 : 3k spectro avec fs=1GSps et nperseg=512 avec uniquement des CW+FMCW
- Correction bug dans le calcul des spectrogrammes

AVANTIX

TODO

TODO

- YOLO : Intégrer PixelsIoU comme loss
- BT300 : Contient tous les fs (4MSps → 4GSps) et nperseg <= 1024 afin d'entrainer un modèle en 1024x1024
- Iq2pdw :
  - Découpler inférence et fusion des bbox (inférence par batch)
  - Refact

AVANTIX

MERCI