AVANTIX

Avancement Simulateur IQ

Spectrogramme : Scipy (CPU) vs PyTorch (GPU)

```
nperseg = 128
noverlap = nperseg//2
nfft = 512
IQ = 10 M
```

Scipy	PyTorch
3,47 s	8,62 ms

Simulateur

FO RADAR impulsionnel :

- DI < 10µs
- No mod, p mod et chirp

FO RADAR LPI :

- DI > 10 µs
- CW
- FMCW
- Chirp : [0.00001, 100] MHz/µs
- PMOD:
  - Random biphasique → 7 < n_moments < 70 | angle ∈ $\{180°, k * 45°, k * 30°\}$
  - Barker biphasique → n_moments ∈ $\{2; 3; 4; 5; 7; 11; 13\}$ | angle ∈ $\{180°, k * 45°, k * 30°\}$
  - Frank, P1, P2, P3, P4 → n_moments ∈ $\{3^2; 15^2\}$
  - ~~T1, T2, T3, T4~~ → non inclus car variation en angle assez proche des précédents

Barker

Table 5.1: Nine Barker Codes with Corresponding PSL and ISL

Code Length	Code Elements	PSL (dB)	ISL (dB)
2	−+, +−	−6.0	−3.0
3	++−	−9.5	−6.5
4	+++−	−12.0	−6.0
4	+++−	−12.0	−6.0
5	+++−+	−14.0	−8.0
7	+++−−+−	−16.9	−9.1
11	+++−−−+−−+−	−20.8	−10.8
13	+++++−−++−+−+	−22.3	−11.5

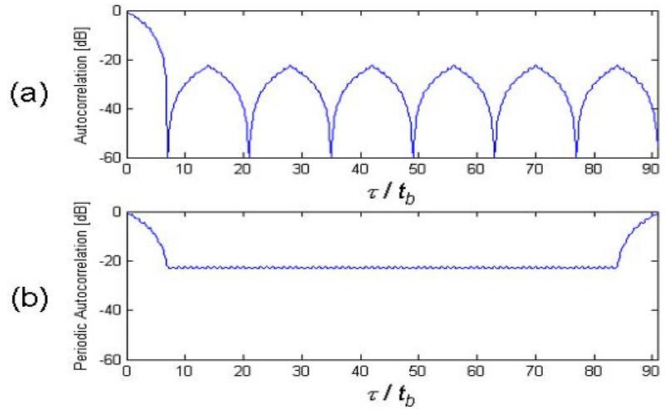


Figure 5.2: (a) ACF and (b) PACF for the $N_c = 13$-bit Barker binary PSK signal (PSL = −22 dB).

$$angle \in \{180°, k * 30°, k * 45°\}$$

Frank

$$\phi_{i,j} = \frac{2\pi}{M}(i-1)\ (j-1)$$

$$i = 1, 2, \ldots, M, \text{ and } j = 1, 2, \ldots, M.$$

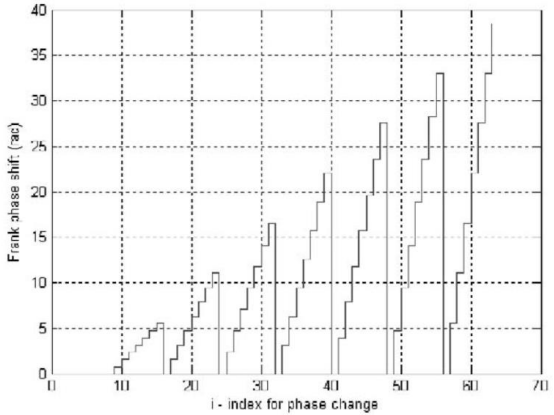
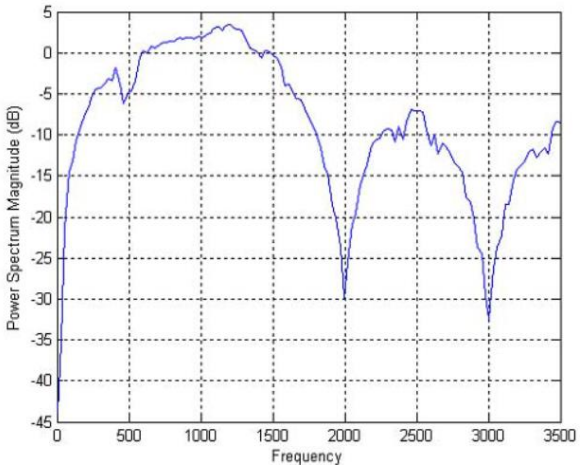


Figure 3.1: Frank phase modulation for $M = 8$ ($N_c = 64$).



Frank

$$\phi_{i,j} = \frac{2\pi}{M}(i-1)\ (j-1) \qquad 3 \le M \le 15$$

$$i = 1, 2, \ldots, M, \text{ and } j = 1, 2, \ldots, M.$$

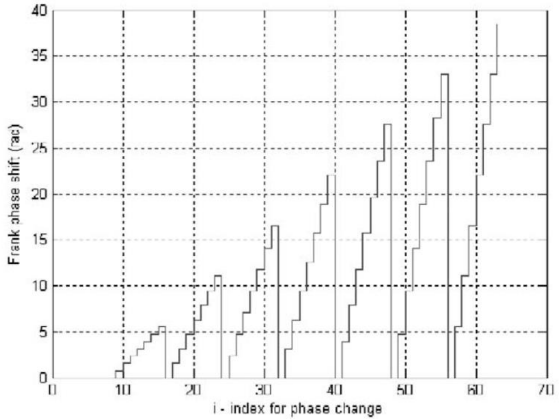


Figure 3.1: Frank phase modulation for $M = 8$ ($N_c = 64$).

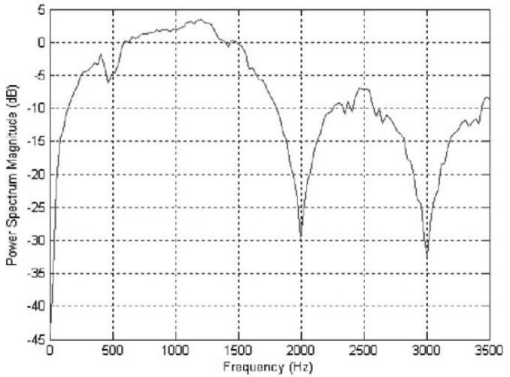
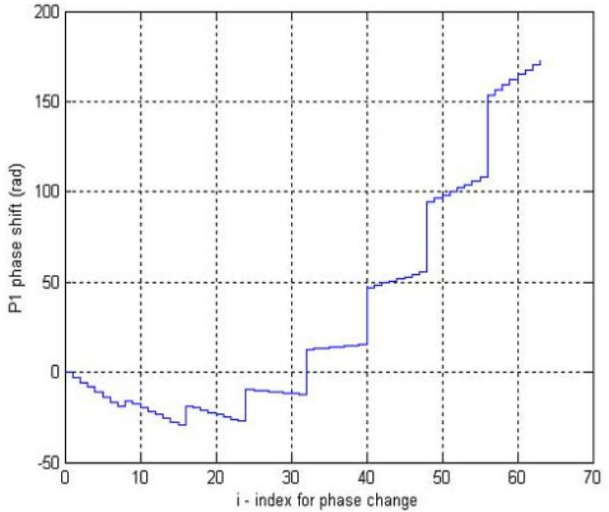


Figure 3.2: Power spectral density for Frank phase modulation for $M = 8$ ($N_c = 64$) with $f_c = 1$ kHz, $f_s = 7$ kHz, and $cpp = 1$.

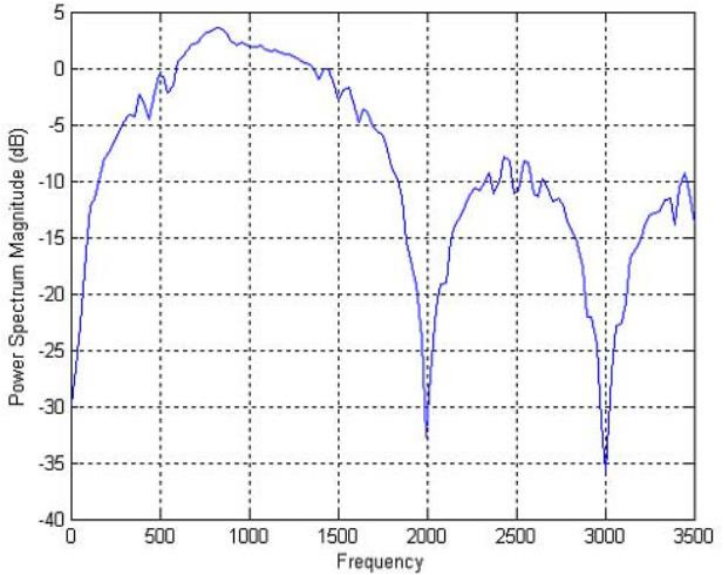
P1

$$\phi_{i,j} = \frac{-\pi}{M}[M-(2j-1)][(j-1)M+(i-1)]$$

$$i = 1, 2, \ldots, M, \text{ and } j = 1, 2, \ldots, M, \qquad 3 \leq M \leq 15$$



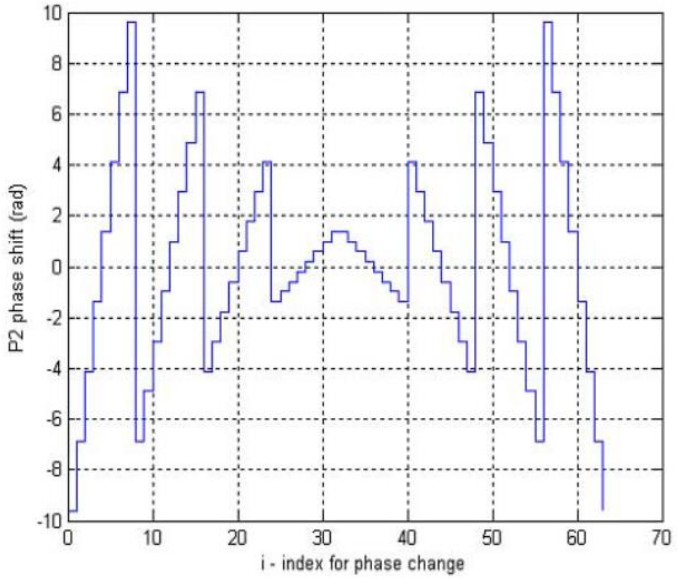
(a)

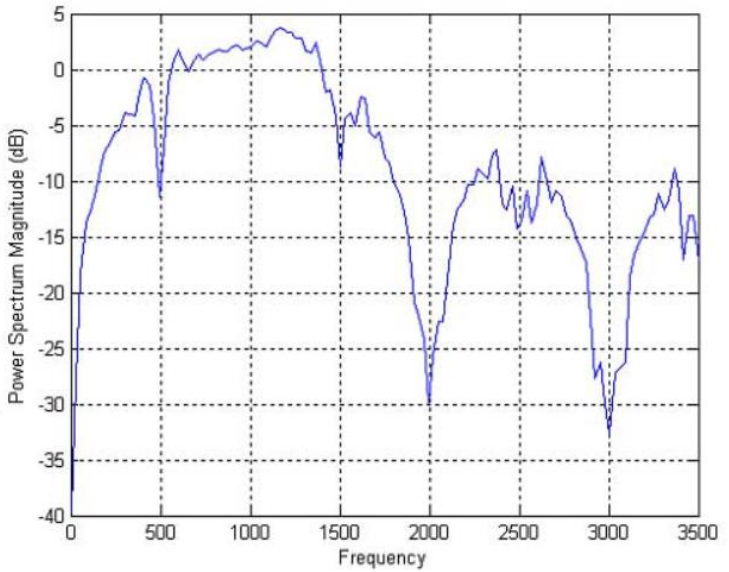


P2

$$\phi_{i,j} = \frac{-\pi}{2M}[2i - 1 - M][2j - 1 - M]$$

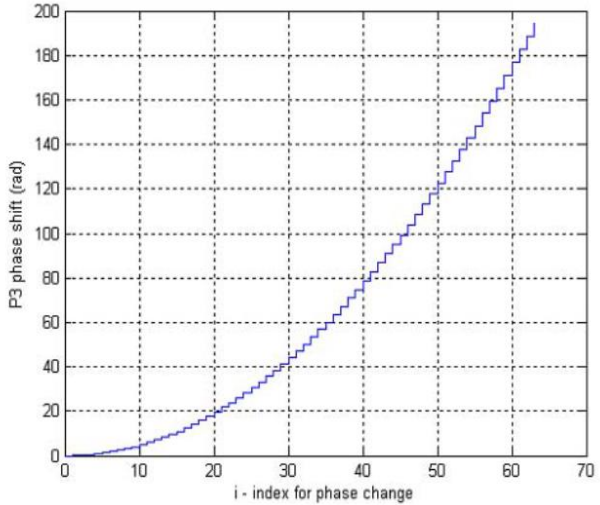
$$i = 1, 2, \ldots, M, \text{ and } j = 1, 2, \ldots, M, \qquad 3 \leq M \leq 15$$

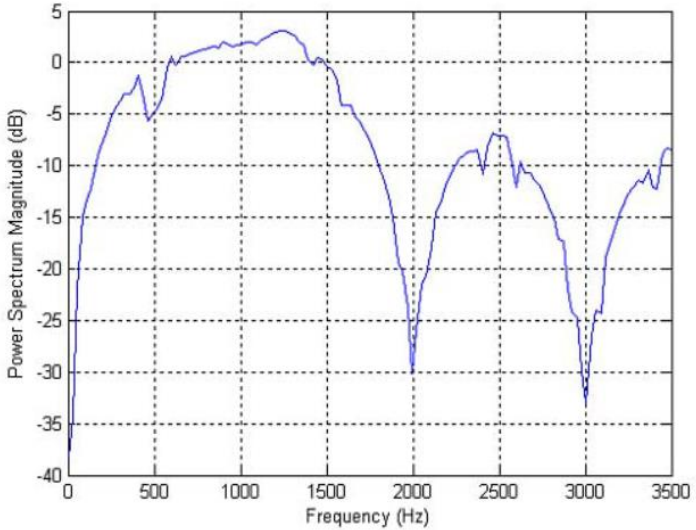




P3

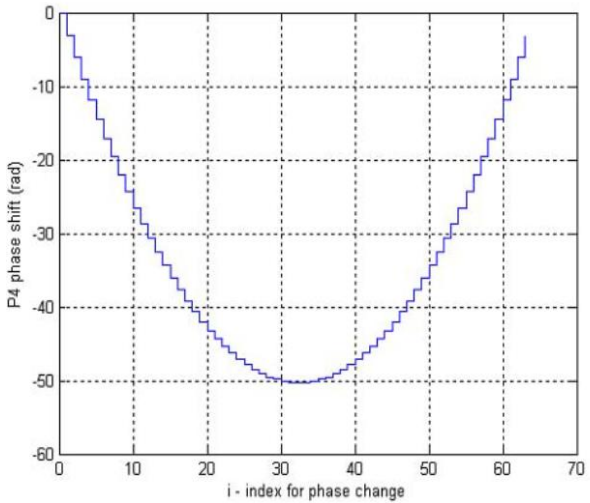
$$\phi_i = \frac{\pi}{N_c}(i-1)^2 \qquad 3^2 \leq N_c \leq 15^2$$

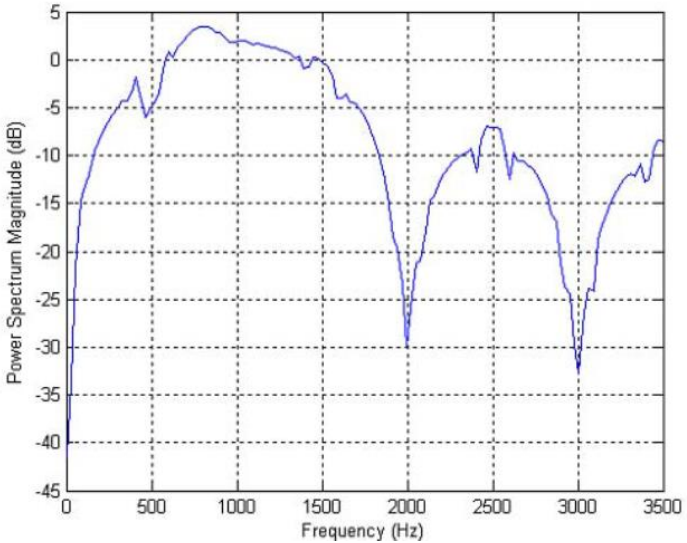




P4

$$\phi_i = \frac{\pi(i-1)^2}{N_c} - \pi(i-1) \qquad 3^2 \le N_c \le 15^2$$





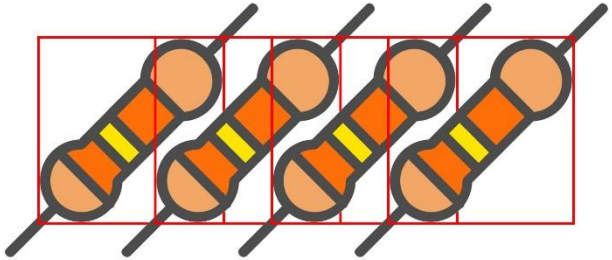
BT1.0.1

Paramètres :

nperseg = [128, 256, 512]
noverlap = nperseg / 2
nfft = 512

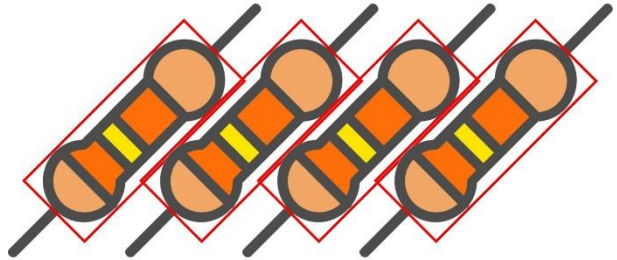
fc_rec = 5 GHz
fs_rec = [1, 5, 25, 100, 500, 1e3, 4e3] MS/s
dt_rec = 513*noverlap/fs_rec
sigma_noise = 1mV
sig_types = [‘radar’] → Interférences pourront être ajoutées plus tard
~~snr_db = 3 à 40dB~~ → Update FVE
~~Snr_db_eff > 8dB~~ → Update FVE

YOLOv8 : Oriented Bounding Box (OBB)



Yolo classic xywh

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8



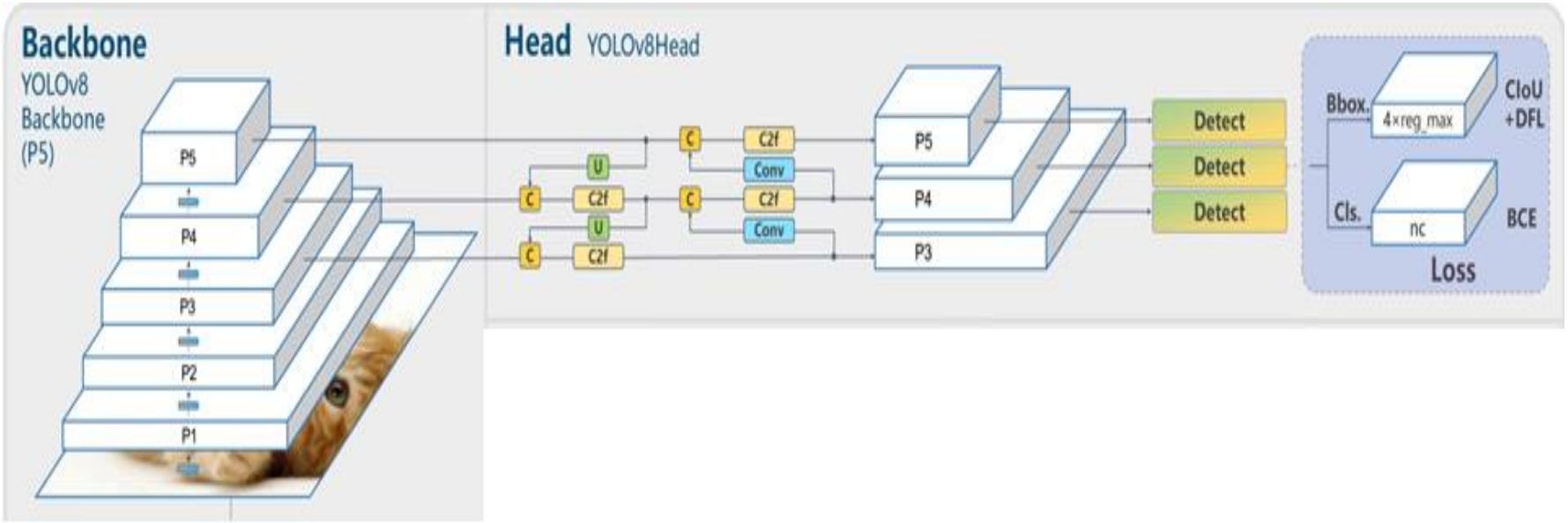
Yolo OBB xyxyxyxy

Model	size (pixels)	mAP^{test} 50	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n-obb	1024	78.0	204.77	3.57	3.1	23.3
YOLOv8s-obb	1024	79.5	424.88	4.07	11.4	76.3
YOLOv8m-obb	1024	80.5	763.48	7.61	26.4	208.6
YOLOv8l-obb	1024	80.7	1278.42	11.83	44.5	433.8
YOLOv8x-obb	1024	81.36	1759.10	13.23	69.5	676.7

Inférence : accélération

- Triton : Facilite l'écriture de code GPU sans nécessiter de connaissances approfondies en CUDA, tout en offrant des performances comparables à celles obtenues par un expert en CUDA
- TorchScript : Sérialise un programme Python et le fait tourner en standalone en C++
- TensorRT : Optimise le réseau de neurones et exploite les optimisations spécifiques des GPU
- Pruning : Suppression des poids inférieurs à un certain seuil
- Profile : Vérifier où sont les bottlenecks et chercher des solutions de contournements

YOLOv8 : architecture



[Unofficial paper: Real-Time Flying Object Detection with YOLOv8]

[Schéma architecture YOLOv8]

YOLOv8 : Fonction de coût

Complete IoU

Binary Cross Entropy

Distribution Focal Loss

L2 reg

$$\mathcal{L}(\theta) = \frac{\lambda_{box}}{N_{pos}}\mathcal{L}_{box}(\theta) + \frac{\lambda_{cls}}{N_{pos}}\mathcal{L}_{cls}(\theta) + \frac{\lambda_{dfl}}{N_{pos}}\mathcal{L}_{dfl}(\theta) + \phi\|\theta\|_2^2 \quad (1)$$

$$V^t = \beta V^{t-1} + \nabla_\theta \mathcal{L}(\theta^{t-1}) \quad (2)$$

$$\theta^t = \theta^{t-1} - \eta V^t \quad (3)$$

$CIoU$: Générer les bbox autour des objets
BCE: Trouve la classe de bbox
DFL: Concentre les efforts sur les bbox/classes les plus difficiles à prédire
$\phi||\theta||^2$: Force le modèle à avoir des poids faibles

θ: Poids du modèle
N_{pos}: Nombre de positifs
λ: Pondération de pénalisation

YOLOv8 : Limitations

Attention aux choix des classes. Le modèle aura du mal si on lui demande de distinguer 2 classes qui ont beaucoup de similarités.
La similarité se juge sur le rapport d'aspect et la texture.

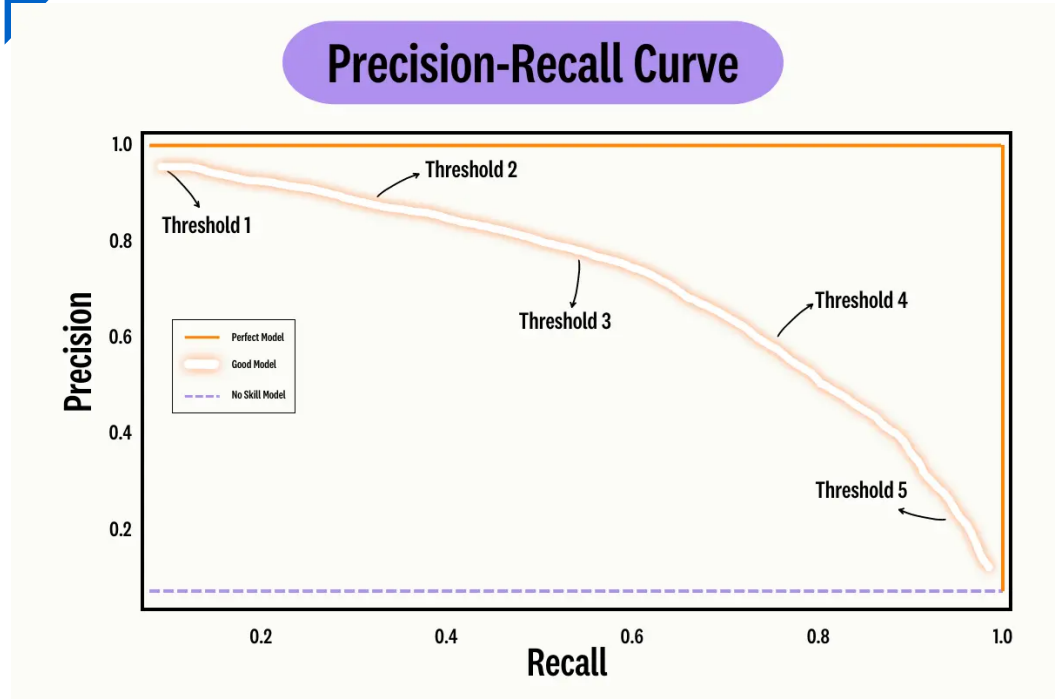
Continuer à fusionner pmod et no_mod ? Vérifier qu'avec les nouveaux PMOD
Quelles classes (a minima) les méthodes post-détection de caractérisation des pulses ont-ils besoin ?

HP tuning : Méthodologie

- 3 classes : NoMod_ PMod, CW, FMod_FMCW
- Image size : 512x512x1 → **TODO debug channel=1**
- Métrique : mAP@0.5:0.05:0.95

- Modèle le plus petit
- HP tuning sur : **TODO**
- 10 epochs
- 1000 itérations

AP



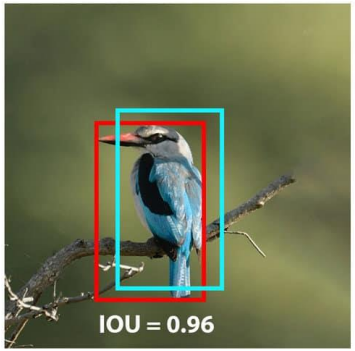
$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

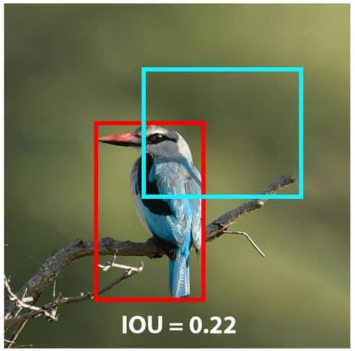
$$Threshold \in [0, 1]$$

AP@IoU_Thresh

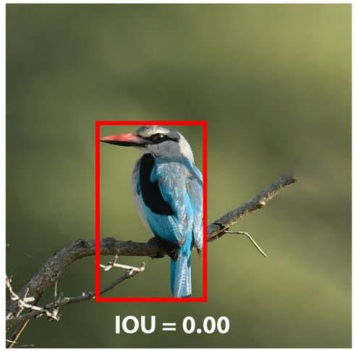
Une AP curve se construit en ayant prédéfini TP, FP et FN.
Pour cela on choisit un seuil sur l'Intersection Over Union.
Dans l'usage, on fait varier ce seuil entre 0.5 et 0.95 par pas de 0.05



True Positive



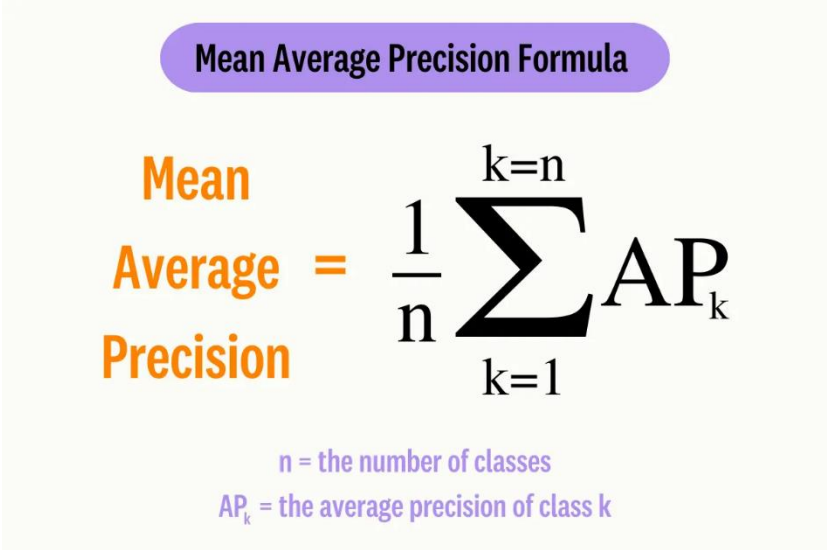
False Positive



False Negative

mAP@0.5:0.05:0.95

On passe du AP@0.5:0.05:0.95 au mAP@0.5:0.05:0.95 simplement en moyennant sur toutes les classes



AVANTIX

MERCI