

ROZIER Loïc, BUCAILLE Nathan, POLI Florian, DAVESNE Adrien

Groupe B

Rapport : Détection de panneaux de signalisation



Contents

1 Constitution de notre base de données	2
1.1 Description du sujet	2
1.2 Acquisition et annotation des données	2
1.2.1 Acquisition	2
1.2.2 Annotation	3
1.3 Pronostics	3
1.4 Script de chargement des données	3
1.5 Exemples d'images de la BD	3
2 Modèle et entraînement	6
2.1 Modèle choisi	6
2.2 Création du script et utilisation	6
2.3 CUDA	7
2.4 Spécification Technique	7
3 Résultats	7
3.1 Analyse qualitative	7
3.1.1 Panneaux de chantier	7
3.1.2 Direction	8
3.1.3 Feux de signalisation	8
3.1.4 Bonnes détections	9
3.2 Analyse quantitative	10
4 Conclusion	14

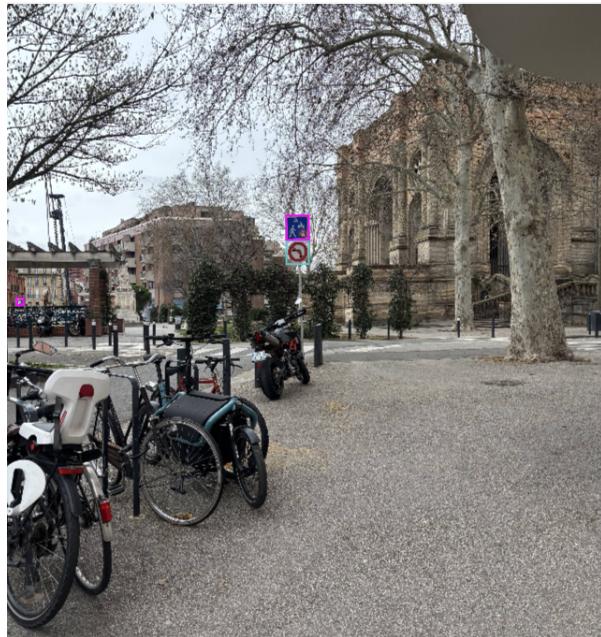
1 Constitution de notre base de données

Lien vers notre base de données :

<https://github.com/flo22208/PAP-DetectionPanneauSignalisation.git>

1.1 Description du sujet

Nous avons décidé de choisir le sujet de la détection de panneaux de signalisation car c'est une action que tout conducteur fait au quotidien. En effet, détecter les panneaux en conduisant est primordial pour ne pas faire d'erreur. Malgré tout, des erreurs arrivent car nous sommes humains, comme pour la personne que nous avons vue tourner à gauche à cette intersection par manque de vigilance.



Cette thématique nous a donc paru appropriée à étudier. Cette technologie permet d'assister le conducteur, il faut donc obtenir un taux de détection des panneaux très élevé pour ne faire aucune erreur, car sinon cela mettrait en danger le conducteur.

1.2 Acquisition et annotation des données

1.2.1 Acquisition

Notre projet se porte sur la détection de panneaux de signalisation, il était donc nécessaire d'obtenir des photos de rues où il y a des panneaux. Nous avons utilisé deux méthodes pour acquérir les données. La première méthode a été de sortir dans Toulouse pour prendre des photos dans la rue. Cette façon de procéder permet d'avoir les images que l'on désire, si l'on veut tel ou tel panneau à tel endroit sur l'image. Par contre, pour annoter ensuite ces images, il a fallu convertir les formats dans un type autorisé par cvat car elles étaient initialement en .heic. La seconde méthode pour obtenir des images a été de faire des captures d'écran sur Google Maps. Cette méthode a permis d'obtenir des images avec des panneaux différents de ceux que l'on peut voir en ville, cela a permis d'avoir une base de données avec des panneaux plus diversifiés.

1.2.2 Annotation

Pour l'annotation des données, nous avons d'abord voulu produire une application avec Python qui permettait d'annoter rapidement chaque image, seulement les annotations étaient finalement plus lentes qu'avec un logiciel externe. L'application produite a été finalisée et mise dans le git du projet. Cependant, nous avons décidé d'utiliser la plateforme cvat. Celle-ci permet de créer des tâches où l'on peut importer les photos puis on peut simplement encadrer les objets désirés tout en choisissant le label qui va avec. Nous avons créé les 7 labels suivants :



Figure 1: Labels

Le format .txt à la sortie de l'annotation ressemble au texte ci-dessous pour l'image donnée précédemment. La première colonne correspond aux numéros associés aux labels que nous avons définis précédemment. Les 4 autres colonnes correspondent aux coordonnées des boîtes englobantes.

```
1 0.483745 0.395875 0.037271 0.028908
3 0.022211 0.457441 0.011447 0.008846
3 0.483468 0.364891 0.037290 0.030310
```

1.3 Pronostics

Nous pensons que notre problème est simple dans le sens où il est déjà utilisé par des entreprises pour rendre autonome la conduite. Nous pensons qu'en termes de précision, nous aurons de bons résultats. Cependant, il faut également que la réponse soit plus rapide qu'un humain, sinon le programme est inutile. Il faudra aussi prendre des panneaux à des distances variables pour les données de test et de validation afin de les comparer au temps de réaction d'un humain. Cependant, la base de données étant limitée, nous ne sommes pas sûrs que le programme va converger vers une solution satisfaisante. De toute façon, le résultat ne sera pas satisfaisant car il ne sera pas au niveau nécessaire pour cette technologie, la difficulté résidant dans le besoin de résultats rapides et sans erreurs afin de ne pas mettre en danger les conducteurs.

1.4 Script de chargement des données

Nous avons créé le script de chargement des données dans Google Colab, il permet d'importer les données du git, d'entraîner le modèle et de lancer les scripts basiques de YOLO pour la vérification. Quelques tests ont été faits sur très peu de données pour vérifier le bon fonctionnement des scripts.

Voici le lien du Google Colab : [lien Google Colab](#)

1.5 Exemples d'images de la BD

Voici quelques exemples d'images de notre base de données :



Figure 2: Exemple 1



Figure 3: Exemple 2



Figure 4: Exemple 3



Figure 5: Exemple 4

2 Modèle et entraînement

2.1 Modèle choisi

Pour mettre en place notre projet, nous avons fait le choix d'utiliser la dernière version de YOLO, YOLOv11, qui nous a semblé la plus adaptée. YOLO (You Only Look Once) est un modèle spécialement conçu pour la détection rapide et efficace d'objets dans une image ou une vidéo, ce qui correspondait parfaitement à notre besoin d'un système réactif, notamment pour éviter des accidents potentiels dans un contexte de détection en temps réel.

YOLOv11 propose plusieurs tailles de modèles, allant de nano à X, permettant d'adapter la complexité en fonction des ressources disponibles et des exigences en termes de performance.

Nous avons testé deux versions :

- **YOLOv11n** (2,6 millions de paramètres), très léger et optimisé pour les systèmes embarqués
- **YOLOv11m** (20,1 millions de paramètres), offrant de meilleures performances mais avec un coût de calcul plus élevé

Logiquement, le plus grand nombre de paramètres dans YOLOv11m lui permet de produire de meilleurs résultats à nombre d'epochs égal. Cependant, ce gain se fait au prix d'un temps de calcul accru, ce qui compromet la capacité de traitement en temps réel. Dans notre cas, la reconnaissance devait se faire sans délai perceptible, c'est pourquoi nous avons retenu YOLOv11n, plus rapide et suffisamment performant selon nos tests.

Par ailleurs, YOLOv11 propose un grand nombre d'hyperparamètres pouvant être ajustés : taux d'apprentissage, stratégie d'augmentation de données, taille des ancrès, régularisation, etc. Toutefois, nous avons opté pour conserver les valeurs par défaut. En effet, les performances obtenues étaient déjà satisfaisantes, et les principales limitations observées provenaient davantage de la qualité de notre base de données que du modèle lui-même.

2.2 Crédit du script et utilisation

Nous avons fait un script sur Google Colab afin d'entraîner le modèle sur les données. Cependant, avec ce script, nous ne pouvions faire tourner l'entraînement qu'à une centaine d'epochs à cause d'une limite temporelle et nous remarquions qu'il n'y avait pas de surapprentissage. Nous avons donc choisi de créer un autre script sur nos ordinateurs personnels, ce qui nous a permis de faire tourner le programme jusqu'à 2000 epochs. Les scripts ont des noms explicites, mais les voici :

- Avant de lancer un quelconque script, il est important d'ajuster les chemins d'accès à la base de données dans le fichier **dataset.yaml**.
- **Train.py** permet de lancer l'apprentissage, les paramètres modifiables sont le modèle utilisé : `model = YOLO("yolo11n.yaml")` qui peut être changé pour une autre taille de YOLO, le nombre d'epochs ainsi que `device` qui peut être '`'cuda'`' ou '`'cpu'`'.
- **Test.py** permet de sauvegarder et d'afficher le résultat fourni par l'IA pour une image donnée. Le paramètre modifiable est donc le chemin d'accès vers l'image d'entrée donné dans `im1 = Image.open('chemin d'accès')`.
- **Result_Test.py** permet d'évaluer l'IA sur la base de données de test. Là encore, il faut ajuster le paramètre `device` entre '`'cuda'`' et '`'cpu'`' selon si `cuda` est disponible.

2.3 CUDA

Le choix de passer sur nos machines a été fait car nous avions accès à un GPU Nvidia récent (RTX série 40) compatible avec la technologie CUDA de Nvidia. CUDA nous a permis d'obtenir un gain de rapidité lors des entraînements de notre modèle. Nous sommes par exemple passés de 1h40 pour 150 epochs sur notre Google Colab initial à 25min pour 500 epochs sur l'ordinateur avec CUDA.

2.4 Spécification Technique

Le modèle a été entraîné en utilisant la librairie Ultralytics version 8.3.130 qui permet l'utilisation de YOLOv11, le tout sous Python 3.13 avec PyTorch 2.7.0. L'entraînement a été réalisé via un script Python exécuté avec CUDA 12.8 pour l'accélération GPU.

Attention, PyTorch 2.7.0 n'est pas compatible avec la dernière version de CUDA, la 12.9.

3 Résultats

3.1 Analyse qualitative

3.1.1 Panneaux de chantier

Les panneaux de travaux sont étonnamment bien détectés. Ils sont détectés en passant outre leur couleur. On peut le voir avec l'image en dessous que les panneaux triangulaires sont quand même bien détectés. Seul le panneau rectangulaire, qui ne ressemble à aucun autre, n'est pas du tout détecté.



Figure 6: Panneaux de chantier

3.1.2 Direction



Figure 7: Erreur panneaux de direction

Les panneaux de directions de cette image ne sont pas détectés. Le fait que ces panneaux ne soient pas détectés peut être dû au fait que ces panneaux de direction sont souvent très nombreux et les uns au-dessus des autres, donc dans notre base de données, les boîtes se chevauchent.

3.1.3 Feux de signalisation

Pour les feux de signalisation, nous n'avons pas convenu au préalable quelle partie nous devions sélectionner du feu pour l'apprentissage. Ainsi, nous avons des feux qui ne sont pris qu'avec la partie d'affichage du feu, d'autres le sont avec le poteau de maintien du feu et certains sont pris de derrière. Ainsi, les résultats peuvent sembler peu satisfaisants. Ils peuvent donc être confondus avec le paysage.

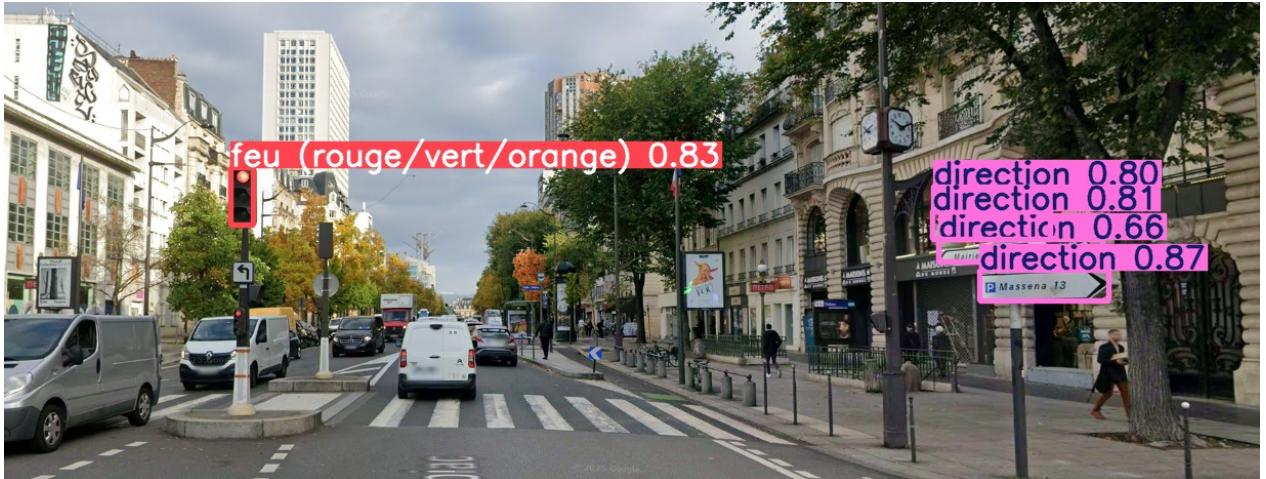


Figure 8: Erreur feux de signalisation

Par exemple sur cette image, on peut observer que le feu n'a pas été bien détecté. Notre modèle a bien détecté le feu principal, mais il n'a pas détecté le petit qui est sur le poteau. S'il suivait exactement ce que

l'un d'entre nous avait détecté pour faire la base de données, il aurait dû détecter deux feux ou bien un grand qui prend le poteau avec.

3.1.4 Bonnes détections

Sur les autres types de panneaux, la détection semble plutôt bonne, comme on peut le voir sur les exemples ci-dessous.



Figure 9: Exemple de bonne détection



Figure 10: Autre exemple de bonne détection

Sur l'exemple ci-dessus, des panneaux sont présents sur une pancarte. Ceci aurait pu causer une détection de panneaux, mais ce n'est pas le cas, ce qui est une bonne chose. L'IA les a bien considérés comme différents des panneaux de signalisation.

3.2 Analyse quantitative

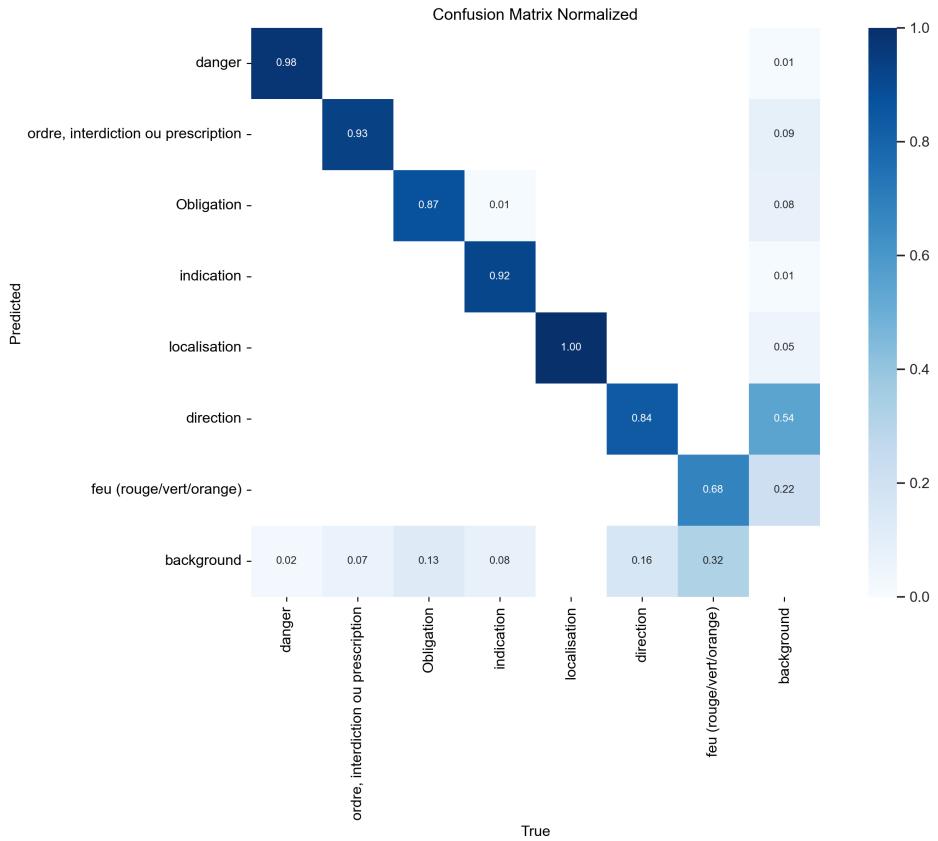


Figure 11: Matrice de confusion normalisée avec 2000 epochs Yolo nano

Nous pouvons observer que les résultats obtenus après 2000 epochs sur Nano sont satisfaisants. Bien que des erreurs de détection persistent, les performances globales sont nettement meilleures par rapport à ce que nous avons pu obtenir en testant avec moins d'epochs. Par exemple, la détection des panneaux de localisation est quasiment parfaite. En revanche, des erreurs importantes subsistent dans la détection des feux de circulation : de nombreux feux ne sont pas détectés dans l'arrière-plan.

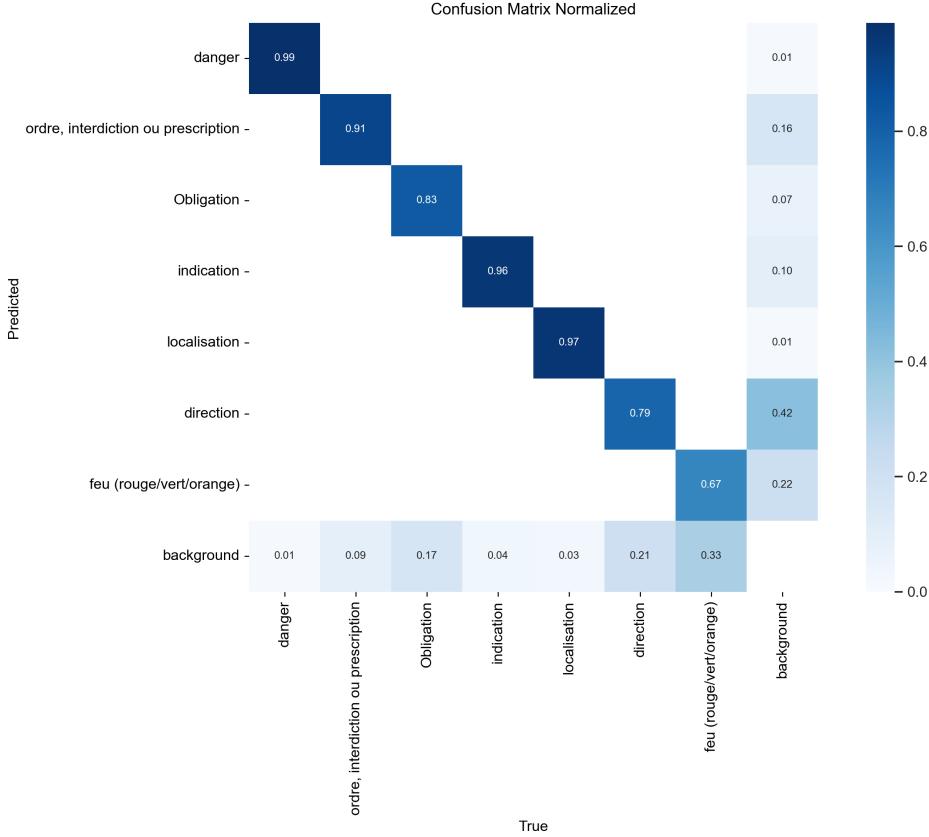


Figure 12: Matrice de confusion normalisée avec 500 epochs Yolo medium

Nous avons également testé Yolo medium avec 500 epochs et on peut observer que les résultats sont très similaires à l'étude précédente sur Yolo nano avec 2000 epochs. Pour éviter de faire autant d'epochs, il serait intéressant d'utiliser Yolo medium. Par contre, il est plus long à entraîner et met plus de temps à détecter.

Modèle	Nb objets	Type panneau	Distance (m)	Vitesse (km/h)	Temps calcul (ms)	Réaction humain (ms)	Réaction à 50m (ms)
YOLOn	1	indication	20	50	9	750	2109
YOLOn	1	feu	10	50	9	750	2109
YOLOn	3	direction, indication	10	30	8.7	750	2109
YOLOm	1	feu	10	50	24	750	2124
YOLOm	1	indication	20	50	28.5	750	2124
YOLOm	3	direction, indication	10	30	23	750	2124

Table 1: Comparaison du temps de traitement des modèles YOLO et du temps de réaction humain

Ce tableau a été effectué avec Google Colab et l'utilisation des deux modèles de YOLO (nano et medium). Le but était de comparer le temps de réaction de notre IA avec celui d'un humain.

Tout d'abord nous pouvons juste comparer le temps de réaction de YOLOm et YOLON, on remarque que la version medium est 3 fois plus longue, d'où l'utilisation du modèle nano lors des tests.

Nous voulons maintenant comparer notre modèle avec la réaction d'un humain. On remarque que le modèle est beaucoup plus rapide pour réagir, environ 10 ms alors qu'un humain mettrait 750 ms. Cependant, un humain peut repérer un panneau à une distance de 50m. Ce schéma illustre notre point.

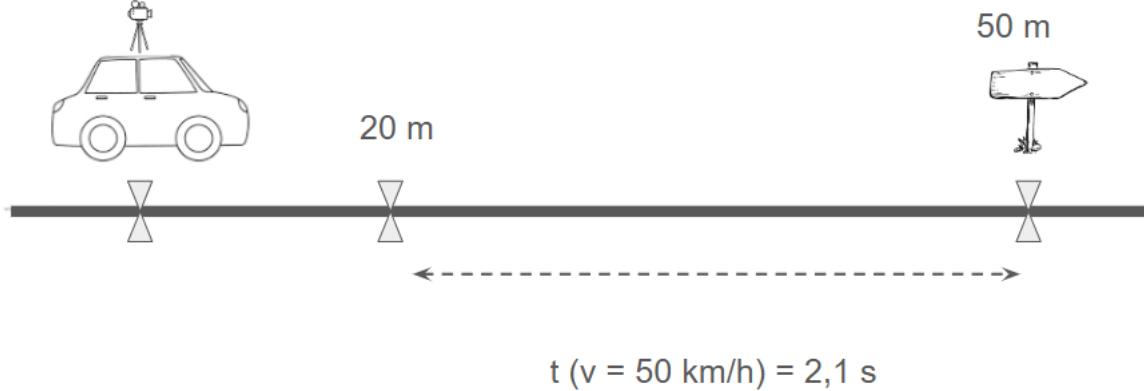


Figure 13: Dessin explication vitesse

Ainsi pour une vitesse de 50km/h le temps pour que le panneau atteigne 20m est de 2,1s. Ainsi, l'humain est forcément beaucoup plus rapide à réagir. On peut cependant penser qu'une voiture autonome n'a pas besoin de réagir à 50m mais qu'à 20m cela risque de créer des accidents. On peut alors penser à d'autres solutions comme un zoom sur l'image lorsqu'il y a une suspicion d'objet. Cela rendrait le programme plus lent, mais avec des caméras performantes, on peut avoir des voitures qui repèrent des panneaux plus loin qu'un humain.

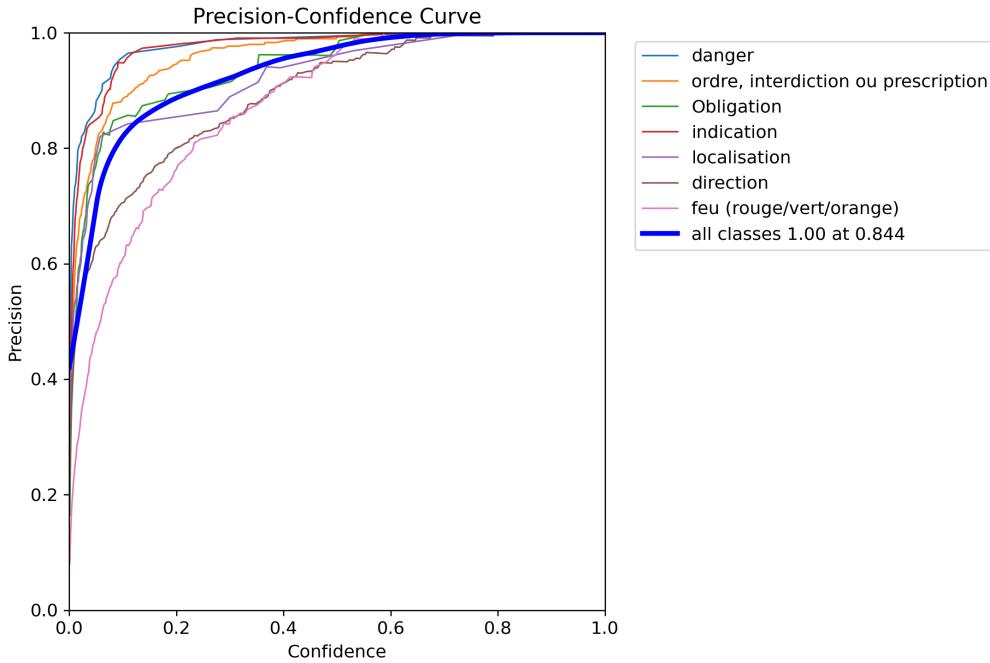


Figure 14: P curve avec 2000 epochs Yolo nano

Pour Yolo nano avec 2000 epochs, les courbes de précision semblent bonnes. Elles se trouvent au-dessus d'une précision de 80% dès 20% de confiance. Cela démontre que notre programme est précis, lorsqu'il détecte

un panneau d'une certaine classe, c'est presque toujours le cas. Malheureusement, ce n'est pas le cas pour les directions et les feux pour des raisons diverses.

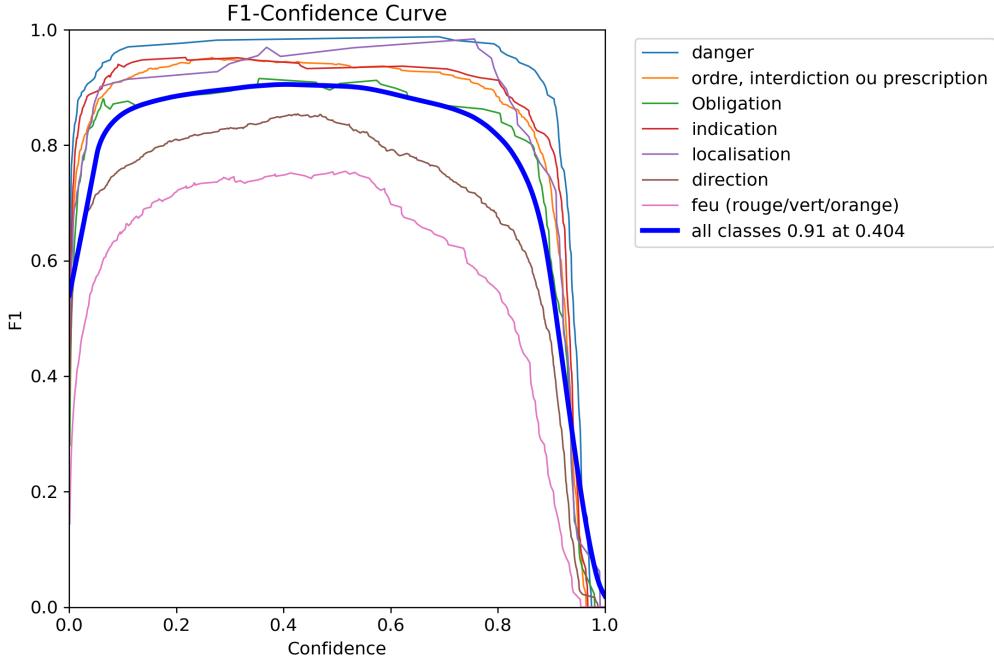


Figure 15: F1 curve avec 2000 epochs Yolo nano

Pour Yolo nano avec 2000 epochs, les courbes de F1 score sont plus ou moins plates et hautes dans la plupart des classes ainsi que pour la moyenne des classes. Cela montre que notre modèle a une haute précision ainsi qu'un bon rappel. En contraste, on observe que pour les directions et surtout les feux, cette courbe nous montre des résultats bien moins probants.

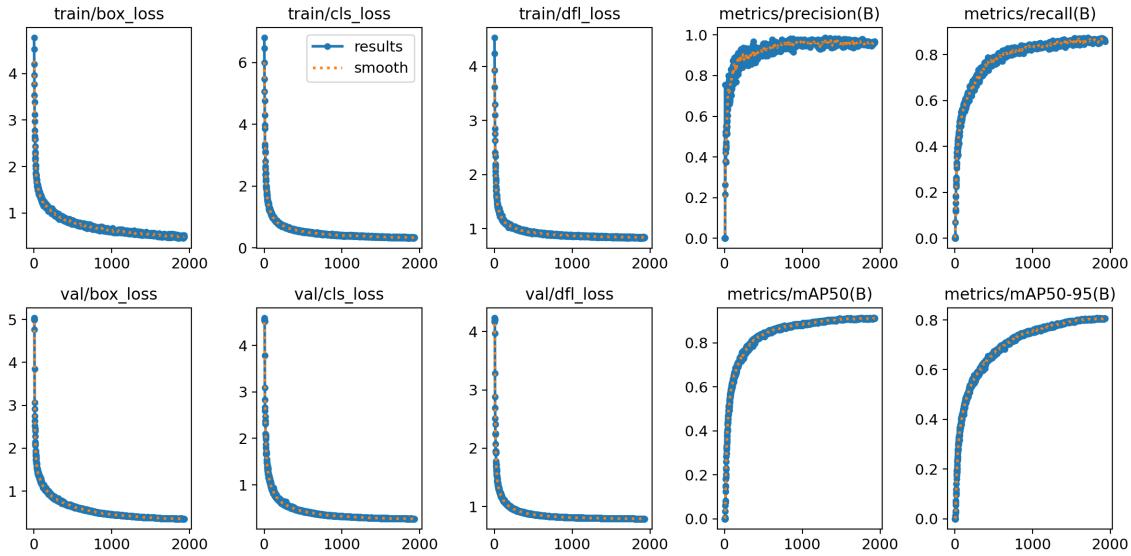


Figure 16: Résultats avec 2000 epochs Yolo nano

Après 2000 epochs, le réseau ne semble pas surapprendre, ce qui est une bonne chose. Le réseau semble être de mieux en mieux entraîné, mais l'amélioration n'est pas significative. Selon les usages, il peut quand même être intéressant de continuer à augmenter le nombre d'epochs.

4 Conclusion

Notre projet de détection de panneau n'est pas assez performant pour être viable dans la vie de tous les jours. Chaque erreur de détection pourrait mettre en danger un automobiliste. En effet, si le modèle fait des erreurs, ce sont des vies qui sont en danger.

Une piste d'amélioration évidente serait de refaire notre base de données pour que l'on sélectionne les feux de circulation de la même manière. Le problème de détection des feux de circulation n'étant pas dû à notre modèle mais bien à la base de données. Un autre élément que l'on pourrait changer dans la base de données pour améliorer la détection serait de rajouter une classe pour prendre en compte les panneaux de chantier.

Finalement, même si notre IA n'est pas assez performante pour faire partie d'une voiture autonome, avec quelques ajustements, elle pourrait être suffisamment performante pour donner des indications au conducteur comme la vitesse du dernier panneau (pour cela, il faudrait identifier les panneaux par vitesse) ou bien donner des informations à des logiciels comme Google Maps.