Rapport

Younes BOKHARI, Alexandre ROBERT et Arthur VILLARD

Dans le cadre de ce projet SMART, nous sommes parvenus à développer deux pipelines d'entraînement et d'inférence, pour entrainer une IA à détecter un ensemble de 10 produits.

Expérimentations

Nous avons mené différentes expérimentations afin d'obtenir le meilleur modèle possible. Nous commençons avec les HP par défaut de ultralytics, avons appris via des recherches que AdamW était l'optimizer le plus adapté, et utilisons le cache disque pour augmenter la vitesse d'entrainement car nos GPU sont assez faibles.

1.

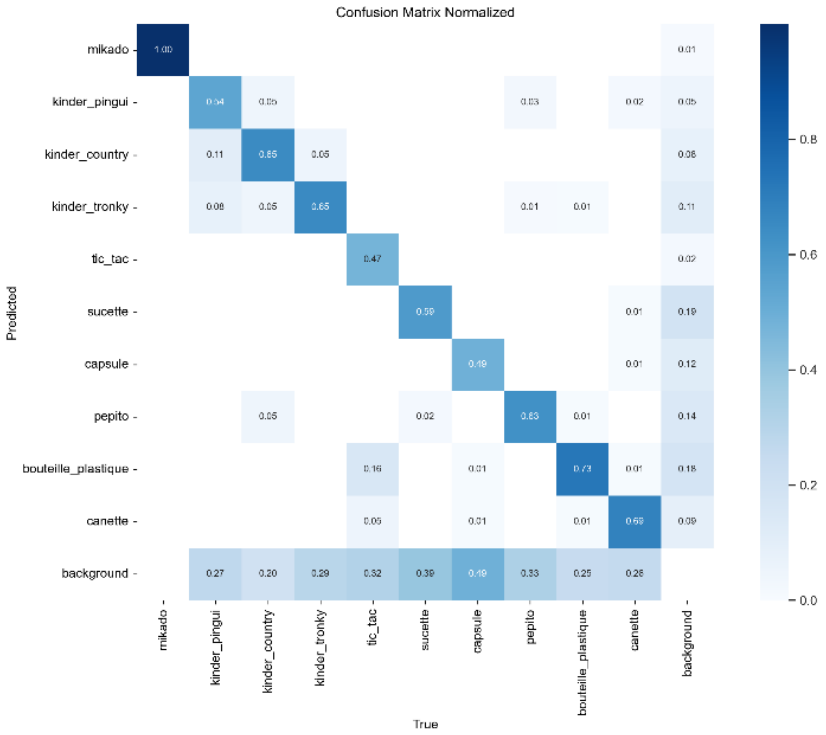
En premier lieu, nous avons mené une expérience visant à observer la totalité du processus d'entrainement. Nous avons pris les hyper paramètres par défaut d'ultralytics, qui fixe notamment la learning rate à 0.01, à l'exception du nombre d'epochs fixé à 200 et de la patience à 20. Nous avons choisi cela car 100 nous semblait peu. Le modèle utilisé pour cette expérience était yolo11n. Cette expérience s'est arrêtée à l'epoch 109 pour une fitness de 0.52.

2.

A partir de l'expérience précédente, nous avons conduit une série d'expériences en modifiant des parametres d'optimisation principalement, comme batch size par exemple, ou meme la taille du modèle utilisé. Nous avons constaté que cela ne changeait pas tellement le résultat mais avons tout de même réussi à obtenir un modèle avec 0.65 de fitness.

1. Modèle n, 115 epochs patience 10

Beaucoup de background. Nous utiliserons un autre modèle pour le prochain pour comparer.

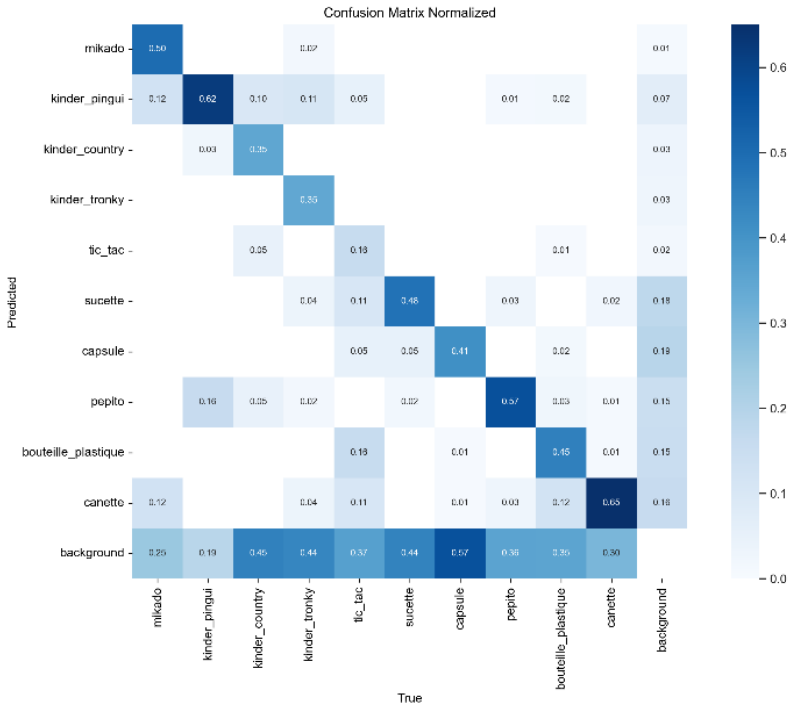


2. Modèle m, 115 epochs

Arrêt manuel au bout de 5 epochs, 20 minutes donc trop long, retour au modèle n.

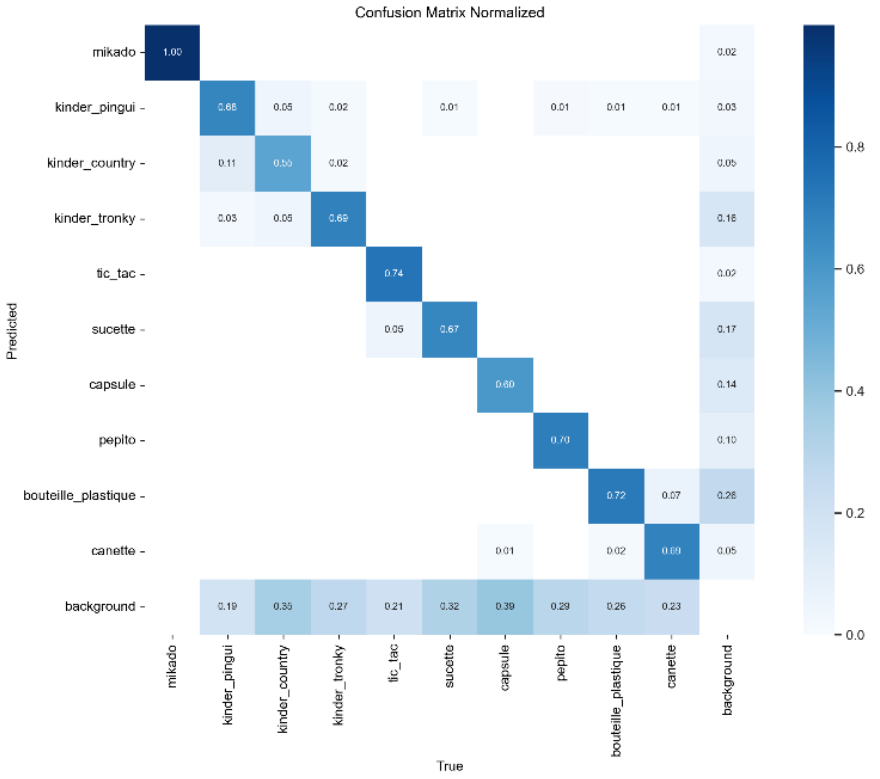
3. Modèle n, 50 epochs mais avec optimisation batch size et workers

Beaucoup de background encore, essai modèle S pour le suivant.

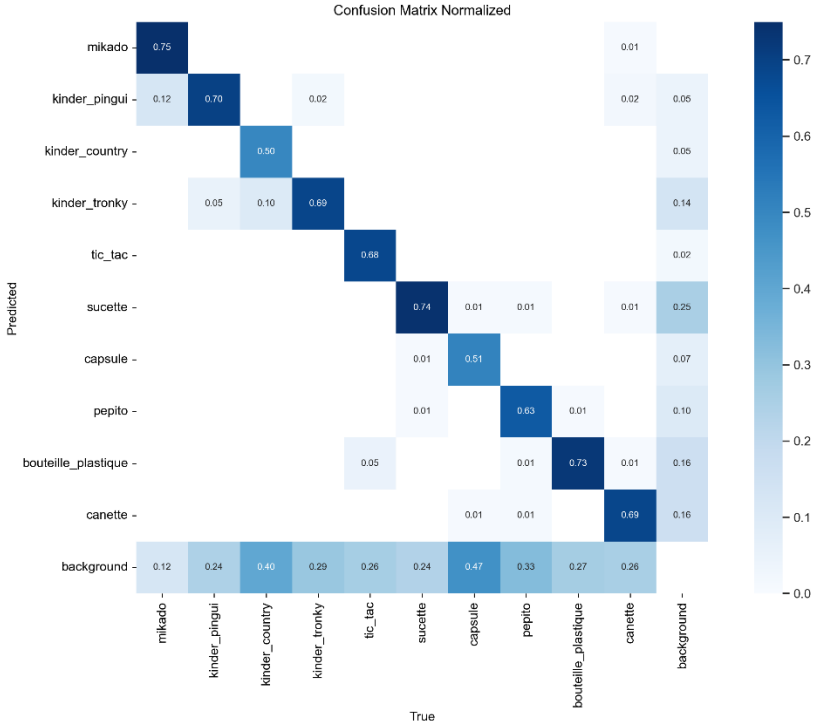


4. Modèle s, 50 epochs

Résultats meilleurs, essai avec beaucoup d'epochs pour le suivant.



5. Modèle s, 400 epochs patience 20



Arrêt au bout de 160 epochs car la patience est atteinte. Les résultats semblent bien meilleurs. C’est notre meilleur résultat.

Malheureusement, ce modèle ne s’est pas enregistré sur picsellia car la configuration et le code ne fonctionnait pas certaines fois en fonction des machines utilisées.

3.

Ensuite, nous avons tenté de modifier la learning rate de 0.01 par défaut à 0.001, ce qui a abouti à peu près aux mêmes résultats pour le même nombre d'epochs (200). A partir de cela, j'ai décidé de lancer un entrainement à 500 epochs, avec une patience de 50, une learning rate de 0.001 et voir le résultat. Cependant, à cause d'un bug de variable d'environnement, l'experiment a été écrasée. Nous avons tout de même obtenu un bon modèle.

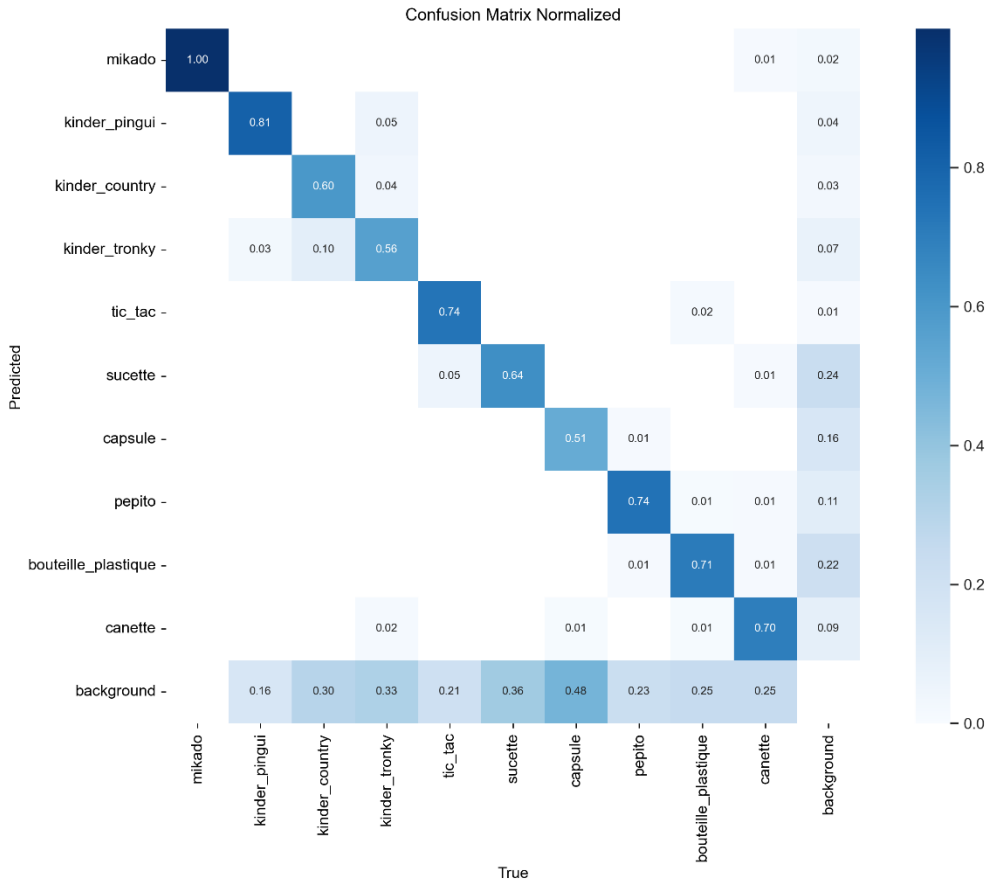
4.

Pour l'experiment finale, nous avons décidé de partir du modèle obtenu à la précédente, afin de voir si nous pouvons affiner le modèle. Pour cela, nous avons également modifié totalement les HP :

- Modèle de base : "Test2-2025-02-09-16-33-56"
- epochs: 500
- patience: 50
- lr0: 0.0001
- cos_lr: True
- flipud: 0.4
- degrees: 180.0
- perspective: 0.0004
- shear: 180.0
- hsv_h: 0.05

Nous avons augmentés certains paramètres d'augmentation dans l'espoir de renforcer la robustesse du modèle. Nous avons utilisé une learning rate bien plus basse car nous avons appris après des recherches que c'est un fonctionnement adapté à l'optimizer AdamW, couplé à l'utilisation de cosine learning rate et des augmentations dont nous avons parlé précédemment.

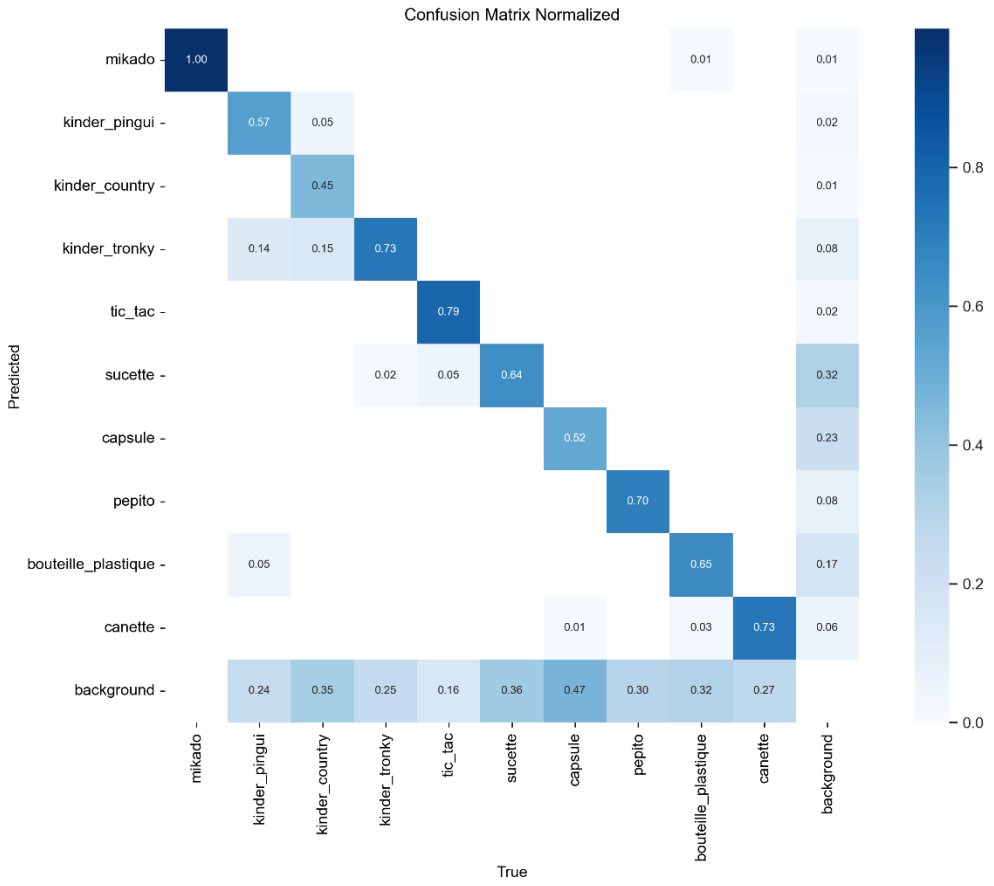
Résultat : La best-fitness a baissé à 0.36 mais nous obtenons la matrice de confusion ci-jointe :



Cela semble être un bon résultat mais nous ne sommes pas capables de dire si le deuxième entrainement était rentable ou non.

5.

À la suite de cette expérimentation, nous avons donc décidé d'effectuer un dernier en partant du résultat obtenu à la 4ème. Nous utilisons une learning rate à 0.001 et des augmentations légèrement réduites pour 300 epochs et une patience à 30. On obtient un fitness de 0.55 et la matrice suivante :



Nous ne constatons pas de réelle amélioration entre celle-ci et la précédente. Ce n'est pas une bonne stratégie.

Au final, les meilleurs résultats sont ceux obtenus en laissant les hyper paramètres par défaut d'ultralytics donc nous ne parvenons pas à conclure sur les changements à effectuer pour obtenir de meilleurs résultats.

Pistes d’amélioration

Pour tenter d'améliorer notre modèle, nous pourrions effectuer une série d'entrainement en partant tout d'abord d'un modèle plus grand comme L par exemple, puis utiliser la fonction de tuning pour obtenir les meilleurs hyper paramètres. Cela nous permettrait ensuite de relancer des entrainements successifs sur la même base pour affiner ses performances.

Nous pourrions également tenter d'utiliser des gpu plus efficaces via le cloud, puis lancer des entrainements sur un modele XL et faire les mêmes tests que nous avons fait en local.

Remarques

Nos entrainements ont été énormément limités par des erreurs fréquentes liées au sdk de Picsellia, les incompréhensions de son fonctionnement et le manque de documentation claire. Nous avons perdu beaucoup de données d'entrainement à cause de ça.

Nous n'avons volontairement pas configuré de pre-commit car nous ne livrons uniquement du code qui compile et s'execute sans souci grâce notamment aux extensions de lint et formatting du code (extension VSCode en temps réel), et nous considérons que la taille du projet et le nombre de collaborateurs ne justifie pas son utlisation.

Nous ne polluons pas non plus notre code avec des docstring car le code structuré, les variables nommées et les types renseignés sont suffisant pour comprendre le code à première vue. (cf. cours de M. Edouard MANGEL)

Liens

Vous trouverez ici les liens vers les deux modèles que nous considérons les meilleurs. Nous ne pouvons pas en choisir un seul car les deux sont doués dans certaines détections respectivement.

Modèle 1

Modèle 2