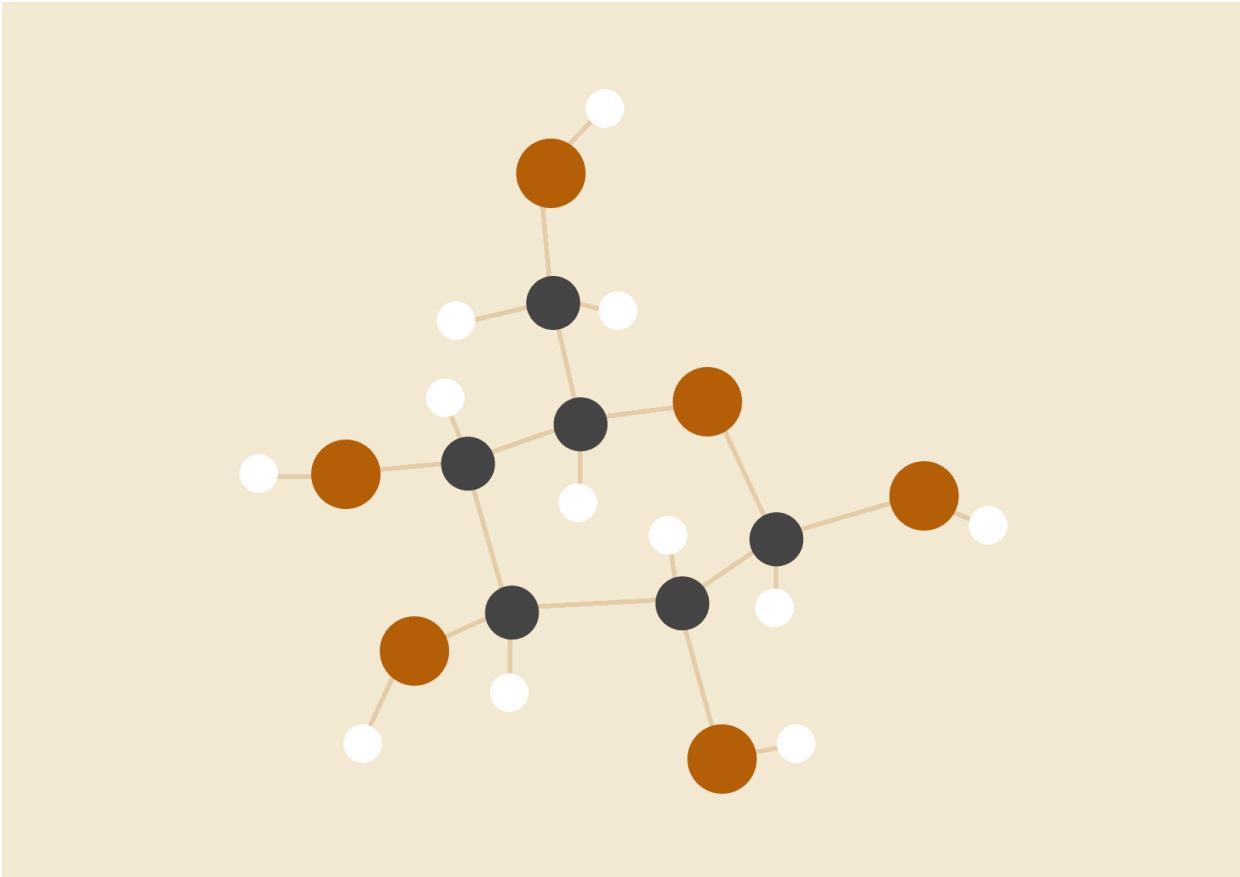


Rapport de recherche



Efkan TUREDI

30.01.2022
OpenClassrooms

INTRODUCTION

Ce papier va s'articuler en 5 parties:

- La première partie sera consacrée à la définition de la problématique réelle que nous souhaitons résoudre.
- La deuxième partie sera consacrée au passage en revue des principaux papiers de recherche déjà présent qui nous permettront de mettre en place notre propre méthode pour résoudre le problème
- La troisième partie sera consacrée à la recherche d'un dataset permettant de faire un apprentissage automatique d'algorithme de machine learning. Cas échéant, nous devrons créer notre propre dataset
- La quatrième partie sera consacrée à la mise en place de l'entraînement de notre modèle d'apprentissage automatique.
- La cinquième et dernière partie sera consacrée aux résultats de notre modèle

PROBLEMATIQUE

La première étape importante de ce projet était de définir une problématique que nous trouvons intéressante à résoudre. Pour cela nous nous sommes posé les questions suivantes:

- Quel sujet nous tient à cœur?
- Quel type d'apprentissage souhaitons-nous mettre en place?
- Cette résolution résout-elle un problème réel?
- Y a-t-il des ressources déjà existantes qui vont nous aider dans la résolution?

Cette liste de questions non-exhaustive a été notre boussole dans la recherche de problématiques que nous souhaitons résoudre.

C'est pourquoi après réflexion, nous avons pensé que résoudre une problématique liée à l'application de gestes barrières dans un contexte sanitaire compliqué semble être une bonne opportunité intéressante. De plus, notre appétit pour les problématiques de computer vision nous a conduit à penser à une résolution de détection du port du masque sur des photos et vidéos qui lui seront soumis.

Notre objectif peut être donc résumé de la façon suivante:

Mettre en place une solution de détections de port de masques sur des photos soumises par l'utilisateur

REVUE DES PAPIERS EXISTANTS

Revues des sources

Nous avons fait nos recherches de solutions existantes en axant les recherches sur la détection d'objets. Nous avons fait nos recherches sur les ressources suivantes:

- paperswithcode.com
- arxiv.org
- Github
- Kaggle

Papers with code: Ce site est une source en accès libre regroupant tout un nombre de papier de recherche à propos d'apprentissage automatique. A ce jour elle constitue une formidable ressource pour apprendre à propos des dernières nouveautés en apprentissage automatique tout en implémentant ces méthodes

arxiv.org: Service de distribution gratuit et une archive en libre accès de 2 millions articles scientifiques dans les domaines de la physique, des mathématiques, de l'informatique, de la biologie quantitative, de la finance quantitative, des statistiques, de l'électrotechnique et de la science des systèmes, et de l'économie.

GitHub: Service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de version Git. Le site assure également un contrôle d'accès et des fonctionnalités destinées à la collaboration comme le suivi des bugs, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet.

Kaggle: Plateforme web organisant des compétitions en science des données. Sur cette plateforme, les entreprises proposent des problèmes en science des données et offrent un prix aux datalogistes obtenant les meilleures performances

Qu'est ce que la détection d'objets?

Tout d'abord, que signifie exactement une détection d'objets en termes de processus de reconnaissance visuelle? La détection d'objets sur une image ou une vidéo implique en fait aussi de la classification d'image et de la localisation d'objets.

Une classification d'image consiste à prendre une image, avec un seul objet, en entrée, et de

renvoyer une classe

Une localisation d'objets consiste à prendre une image avec un ou plusieurs objets et de renvoyer un ou plusieurs conteneurs d'objets.

Une détection d'objets consiste à avoir une image avec plusieurs objets et de renvoyer un ou plusieurs conteneurs d'objets avec un label pour chaque conteneur.

La base de données ImageNet est notamment reconnue pour son concours de détection d'objets.

Revue des méthodes mises en place

Dans cette partie nous allons regardons les familles d'algorithme les plus connues dans ce domaine: R-CNN, Fast R-CNN, Faster R-CNN, et YOLO

R-CNN: Développé dans un papier de 2014: <https://arxiv.org/abs/1311.2524>

La structure de cette méthode repose sur 3 modules qui s'enchaînent:

- Module 1: Proposition de régions. Génère et extrait des catégories indépendantes des régions c-a-d des contours candidats
- Module 2: Extraction de caractéristiques. Extraire des features de chaque région candidat à l'aide d'un CNN
- Module 3: Classification. Classifier les features obtenues dans l'une des classes disponibles à l'aide d'un softmax, logit ou SVM par exemple

L'inconvénient majeur de cette méthode est qu'elle est très lente même avec de grosses ressources.

Fast R-CNN: Développé dans un papier de 2015: <https://arxiv.org/abs/1504.08083>

Il reprend essentiellement la structure de l'algorithme en lui donnant des améliorations pour, comme son nom l'indique, augmenter rapidement sa vitesse d'entraînement. Son innovation principale consiste du fait que cette méthode demande des "Region of Interest" en input avec les images qui lui sont soumises. L'output de ces CNN est ensuite soumise à des couches connectés de 2 manières: un softmax pour classification, et un SVM linéaire pour la réalisation

des contours des objets.

Faster R-CNN: Développé dans un papier de 2016: <https://arxiv.org/abs/1506.01497>

Encore une fois une amélioration du modèle précédent. Il se compose en fait d'un Fast R-CNN précédé d'un module supplémentaire concernant les "Region of interest".

- Module 1: CNN pour proposer des régions intéressantes.
- Module 2: Fast R-CNN pour extraction des features et dessiner des contours

YOLO: Développé par Joseph Redmon en 2015: <https://arxiv.org/abs/1506.02640>

Cette famille de modèles est généralement moins précise que R-CNN, mais leurs vitesses d'exécution rendent par exemple possible la reconnaissance d'objets en temps réel sur une webcam.

Elle consiste à utiliser un seul et unique CNN de bout en bout , qui prend une photo en entrée, dessine les contours des objets, et s'occupe de mettre un label sur chaque conteneur obtenus.

Au fur et à mesure des années, YOLO a connu des améliorations dans le temps de calcul jusqu'à avoir des v4 et v3.

Nous avons donc fait le tour des méthodes qui sont disponibles. A notre échelle, il est préférable de ne pas donner trop d'importance à la performance, mais plutôt au temps de calcul car cela nous permettra d'itérer rapidement, et de développer un modèle en production de manière efficace.

C'est pourquoi, nous avons décidé de privilégier la famille YOLO pour implémenter notre solution.

Maintenant, il faut trouver ou créer un dataset qui pourra nous permettre de mettre en place l'apprentissage souhaité. C'est l'objet de la prochaine partie.

DATASET POUR APPRENTISSAGE AUTOMATIQUE

Description d'un dataset

Nous avons fait des recherches notamment sur Kaggle, pour trouver un dataset correspondant qui pourrait nous aider à résoudre notre problème. Cela serait d'autant plus intéressant car cela nous permettrait d'économiser un gros temps de préparation de dataset, tâche au combien importante pour la suite de notre exercice.

Heureusement, les datasets dans le domaine sont nombreux notamment à cause de la crise sanitaire qui a certainement mis les données / datasets dans le domaine de la santé au cœur de nos vies.

Nous avons sélectionné le dataset suivant se trouvant sur Kaggle:

[Labeled Mask Dataset \(YOLO_darknet\) | Kaggle](#)

Il est également déjà mis au format compatible pour une mise en implémentation avec YOLO, ce qui est aussi facilitateur de notre exercice.

Il possède deux classes de photos: "mask", "without_mask" qui seront bien évidemment les deux classes que nous allons essayer de prédire avec YOLO.

Maintenant que notre dataset est prêt, nous pouvons passer à l'implémentation de notre apprentissage automatique.

Quelques chiffres sur notre dataset

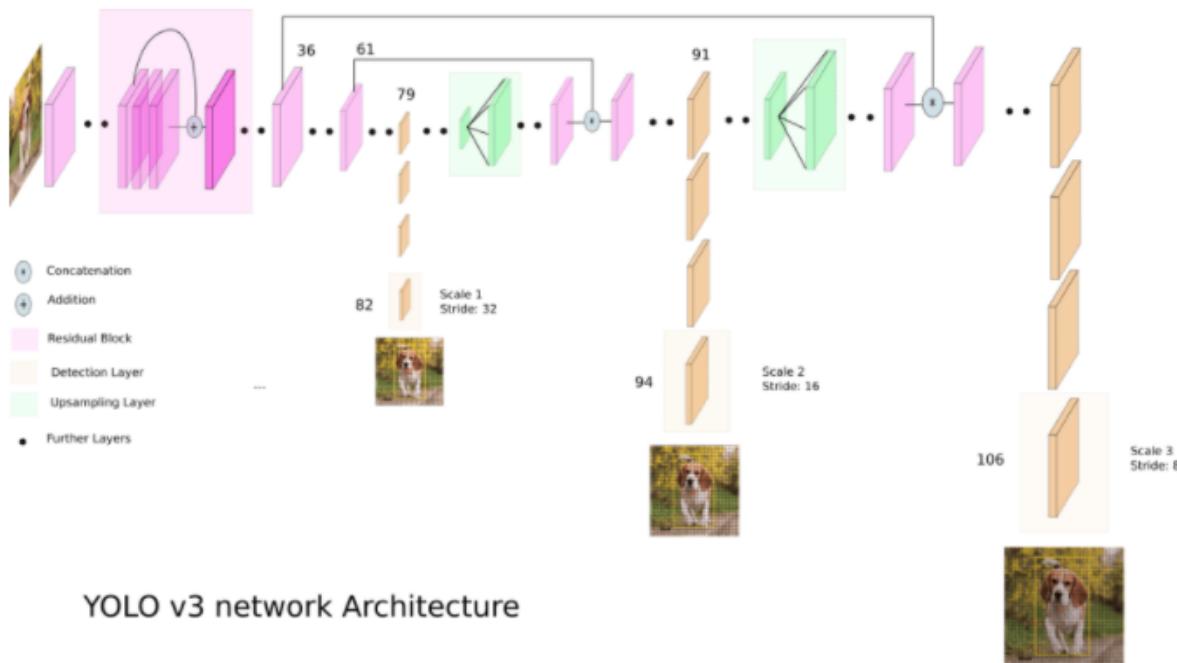
Nous avons un dataset qui est relativement fortement déséquilibré. En effet, le nombre de personnes avec masque est largement au dessus de celui du nombre de personnes sans masque. Notre dataset contient 2568 visages à détecter dans le test set: dont 1991 sont masqués et 577 sans masque.

Concernant le test set, nous avons 577 visages à détecter dans le test set: dont 445 sont masqués et 132 sans masque.

IMPLEMENTATION

Nous n'allons pas utiliser TensorFlow ou PyTorch comme framework pour mettre en place notre apprentissage. L'implémentation la plus courante de YOLO se fait avec Darknet, qui est un framework écrit en C++, et qui utilise notamment les commandes de terminal pour l'utiliser. La difficulté essentielle de ce framework sera donc de trouver les bons paramètres pour pouvoir lancer les algorithmes.

Architecture de YOLOv3



La structure de YOLOv3 s'organise de la façon suivante:

- YOLOv3 passe les images à un CNN
- En sortie, nous avons une liste de conteneurs avec la classe prédictive. Chaque contour est représenté par un vecteur de longueur 6
- Puis nous faisons un Intersection over Union et un Non maximum suppression pour éviter de choisir des contours qui seraient l'un sur l'autre

YOLOv3 utilise une cross entropie binaire pour la fonction perte de la classification pour chaque label, alors que le score de confiance se calcule avec une régression logistique

Architecture de YOLOv4

YOLOv4 se décompose en trois blocs fonctionnels:

- **Backbone**: extrait les features de l'image soumise en entrée. Il fournit en sortie une carte de l'ensemble des features présentes dans l'image.
- **Neck**: identifie les features pertinentes.
- **Dense**: identifie la position des features utiles en traçant des bounding box autour. Pour chaque, il donne la nature de l'objet détecté.

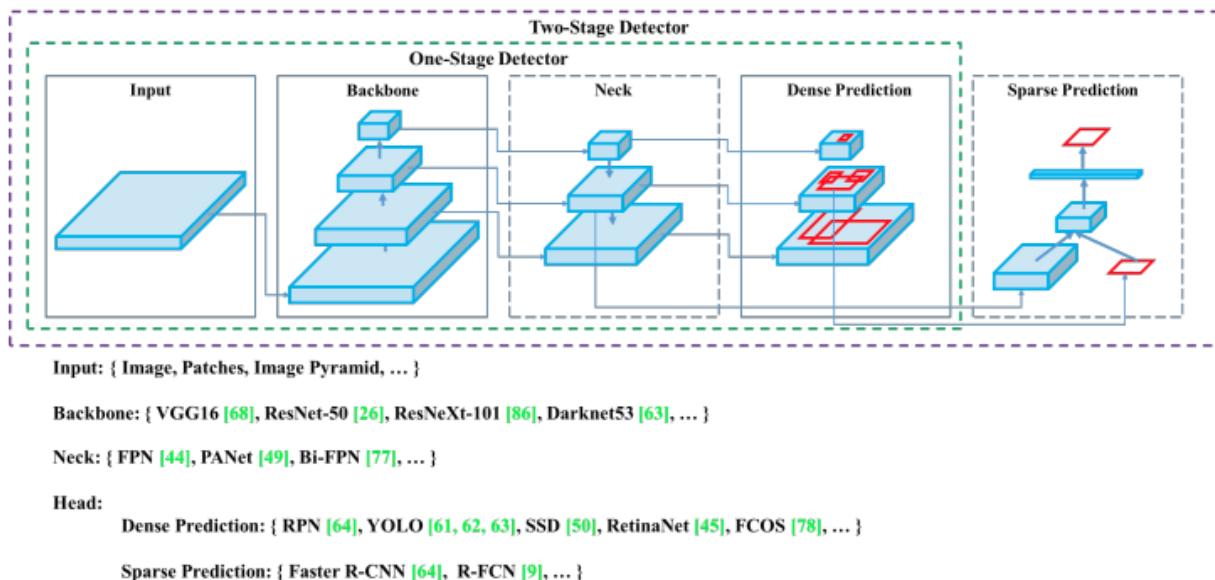
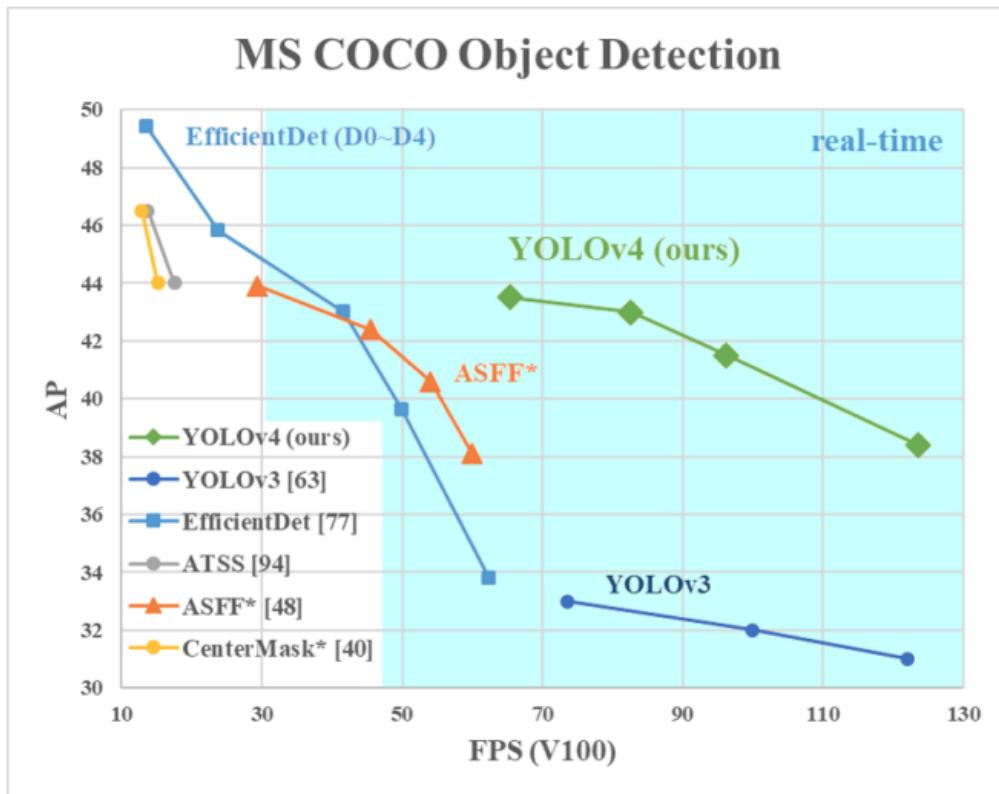


Figure 2: Object detector.

Performances de YOLOv4 / v3

Pour cette partie nous allons faire référence à l'article de publication de YOLOv4 où cet algorithme est comparé à d'autres en termes de précision moyenne en fonction de FPS sur le dataset COCO.



Ainsi une grille de lecture de ce graphique pourrait être la suivante:

- YOLOv4 est moins sensible à une forte baisse de la précision moyenne avec une augmentation du FPS
- Pour des vidéos modernes à 60FPS, YOLOv4 surpassé les autres algorithmes déjà existant

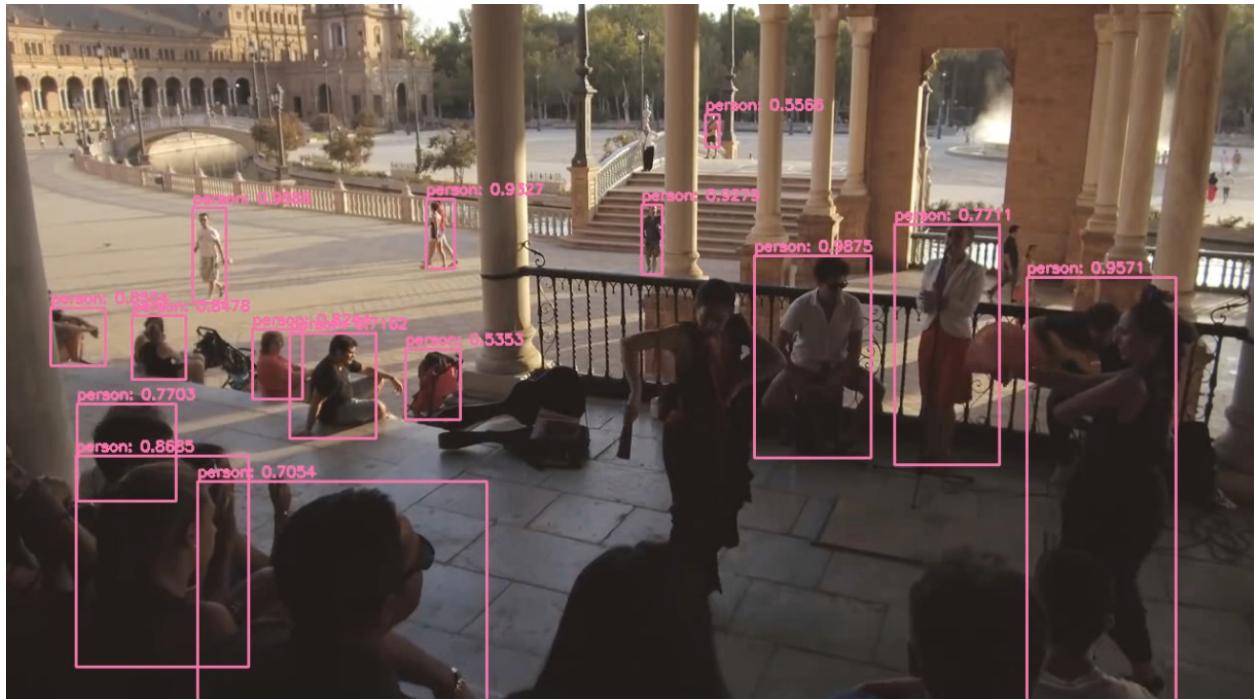
Pour ces deux raisons, les choix de l'algorithme de la famille YOLO semble être la plus judicieuse pour nous.

Utilisation de YOLOv4 avec les poids pré-calculés sur une vidéo aléatoire

Pour se rendre compte de la puissance de cet algorithme et aussi mieux se rendre compte de la manière dont il faudra paramétrer, nous décidons de faire tourner YOLOv4 / v3 sur une vidéo aléatoire téléchargée depuis la plateforme Youtube.

La vidéo retenue est celle-ci: [Travel to Sevilla - Spain - 4K](#)

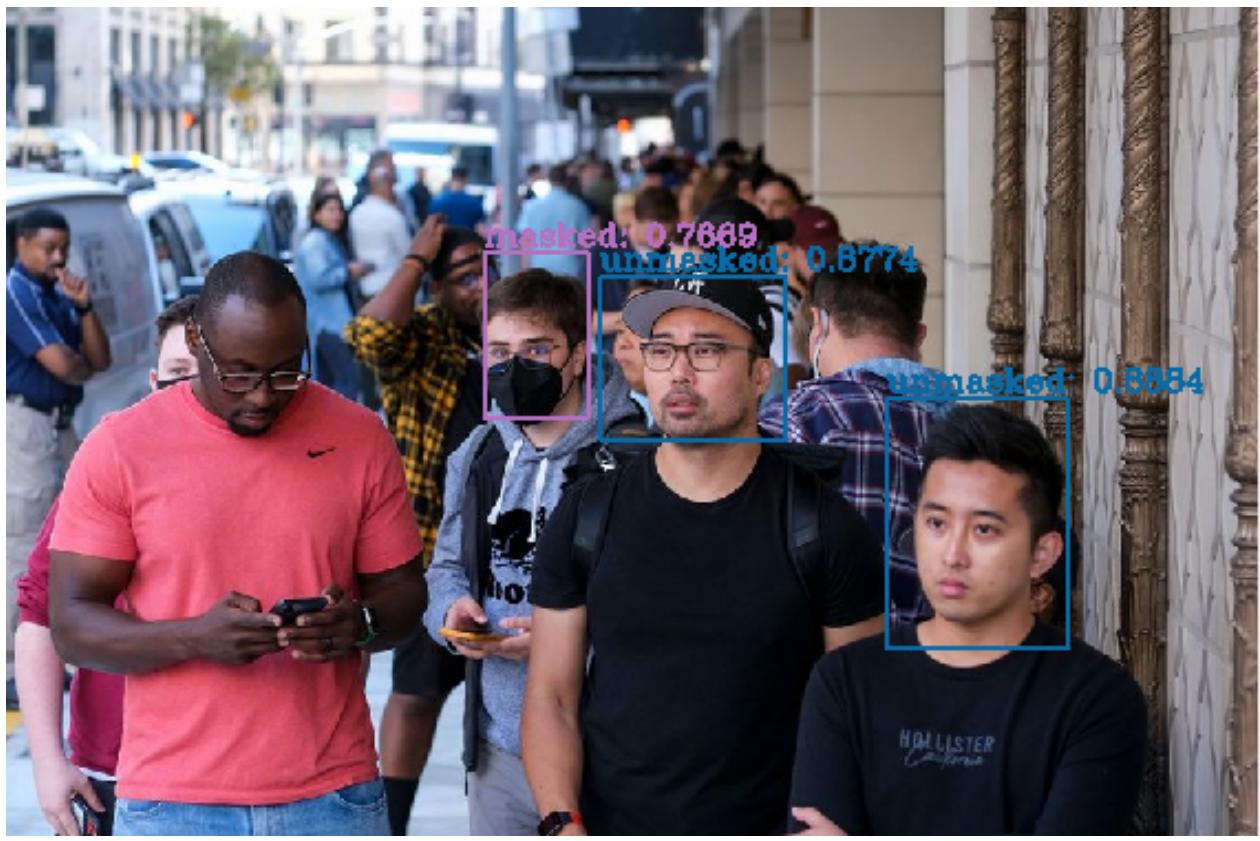
Nous allons comparer la même image pré-YOLO et post-YOLO



Utilisation de YOLO avec notre propre dataset

De la même manière nous allons comparer une image soumis à notre algorithme et évaluer la détection





RESULTATS

Dummy baseline

Le déséquilibre de classes est une bonne opportunité pour prendre le classificateur qui renvoie à la classe la plus fréquente, c'est à dire "unmasked".

A noter que pour rendre les choses simples, nous supposons que l'algorithme est parfait en terme de détection des sujets: notre algorithme détecte parfaitement les sujets de l'image. Son seul travail est de classifier les sujets. Cette simplification permet de facilement faire une implémentation avec scikit-learn et sa classe DummyClassifier

Voici le tableau récapitulatif des résultats de ce classifieur sur notre test set:

	Precision	Recall	Count
Mask	77%	100%	445
Unmasked	0%	0%	132

	Precision	Recall	Count
Accuracy macro avg	39%	50%	577
Accuracy Weighted avg	59%	77%	577

YOLOv4

Concernant les résultats nous avons pour 3 métriques de classification: mAP, précision et recall. Nous sommes capables d'obtenir ces calculs directement avec des commandes simples de darknet

Voici les tableaux récapitulatifs:

	mAP	Precision	Recall
Dataset Mask	89%	84%	89%

YOLOv4	mAP	TP	FP
Masked	84%	387	73
Unmasked	95%	126	27

YOLOv3

	mAP	Precision	Recall
Dataset Mask	96%	95%	90%

YOLOv3	mAP	TP	FP
Masked	96%	400	23
Unmasked	97%	118	6

Conclusion

YOLOv3 fonctionne pour notre dataset. D'autant plus que le temps d'entraînement de YOLOv4 est un critère important à notre niveau, car la capacité d'itération rapide est importante pour nous. Pour développer un prototype nous privilégierons YOLOv3

REFERENCES & BIBLIOGRAPHIE

R-CNN: [\[1311.2524\] Rich feature hierarchies for accurate object detection and semantic segmentation](#)

Fast R-CNN: [\[1504.08083\] Fast R-CNN](#)

Faster R-CNN: [\[1506.01497\] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks](#)

YOLO: [\[1506.02640\] You Only Look Once: Unified, Real-Time Object Detection](#)

YOLOv4: [\[2004.10934\] YOLOv4: Optimal Speed and Accuracy of Object Detection](#)

YOLOv3: [\[1804.02767\] YOLOv3: An Incremental Improvement](#)

Dataset: [Labeled Mask Dataset \(YOLO_darknet\) | Kaggle](#)

Tutoriel YOLO: <https://github.com/theAIGuysCode/YOLOv4-Cloud-Tutorial>

Tutoriel YOLO pour customisation de dataset: [Train YOLO for Object Detection with Custom Data | Udemy](#)

Darknet framework: [Darknet: Open Source Neural Networks in C](#)

Tutoriel - Detection d'objet: [A Gentle Introduction to Object Recognition With Deep Learning](#)