RAPPORT

TP SYSTEMES INTELLIGENTS AVANCES

Pierre-Nicolas CHASSAGNE IT- S9ILC1

Nicolas FORGERON IT-S9ILC1

Sommaire

I. Conception du dataset 4

II. Choix de l'algorithme ou modèle 4

1. Choix de l'algorithme 4

2. Utilisation de roboflow 5

3. Présentation du dépôt Github 6

4. Présentation des entrainements 8

III. Mise en œuvre et évaluation 15

1. Evaluation sur la dataset de validation avec les meilleurs poids 15

2. Inférence et affichage en temps réel 16

Table des illustrations

Figure 1 : Etat de l'art des notebooks utilisant le dataset de validation 4
Figure 2 : Export du dataset au bon format et avec une taille fixe 5
Figure 3 : Résultat de l'entrainement 1 8
Figure 4 : Calcul du mAP de l'entrainement 1 8
Figure 5 : Résultat de l'entrainement 2 9
Figure 6 : Calcul du mAP de l'entrainement 2 9
Figure 7 : Résultat de l'entrainement 3 10
Figure 8 : Calcul du mAP de l'entrainement 3 10
Figure 9 : Résultat de l'entrainement 4 11
Figure 10 : Calcul du mAP de l'entrainement 4 11
Figure 11 : Résultat de l'entrainement 5 12
Figure 12 : Calcul du mAP de l'entrainement 5 12
Figure 13 : Récapitulatif des entrainements 13
Figure 14 : Récapitulatif des entrainements 14
Figure 15 : Calcul du mAP avec cartucho 15
Figure 16 : Nombre de faux positifs par rapport au nombre de vrais positifs 15
Figure 17 : Détection en temps réel de la classe without_mask 16
Figure 18 : Détection en temps réel de la classe with_mask 16
Figure 19 : Détection en temps réel de la classe with_incorrect_mask 17
Figure 20 : Détection en temps réel de la classe with_incorrect_mask 17

I. Conception du dataset

Présentation du dataset

Le dataset d'entrainement est composé de 3435 images annotées au format Pascal/VOC à l'aide de bounding boxes.

3 classes d'objets sont présentes :

- Classe without_mask : visage sans masque,
- Classe with_mask : visage avec masque,
- Classe with_incorrect_mask : visage avec masque mal porté.

Le dataset de validation utilisé pour tester la fiabilité de l'algorithme est :

https://www.kaggle.com/andrewmvd/face-mask-detection

Il est composé de 853 images annotées au format Pascal/VOC et comportant les mêmes classes.

II. Choix de l'algorithme ou modèle

1. Choix de l'algorithme

Pour choisir notre algorithme, nous avons recherché les différentes implémentations de détection de masque utilisant ce dataset directement depuis kaggle. Parmi les résultats les plus votés, on retrouve beaucoup de réseaux two-stage avec de bons résultats. Cependant, ceux-ci ne font pas d'inférence en temps réel. Notre but étant de détecter des masques sur une vidéo, nous avons opté pour un détecteur one-stage : YOLOv5.

Celui-ci apparait en premier dans les réseaux en one-stage. De plus, il possède les commentaires et les modifications les plus récentes.

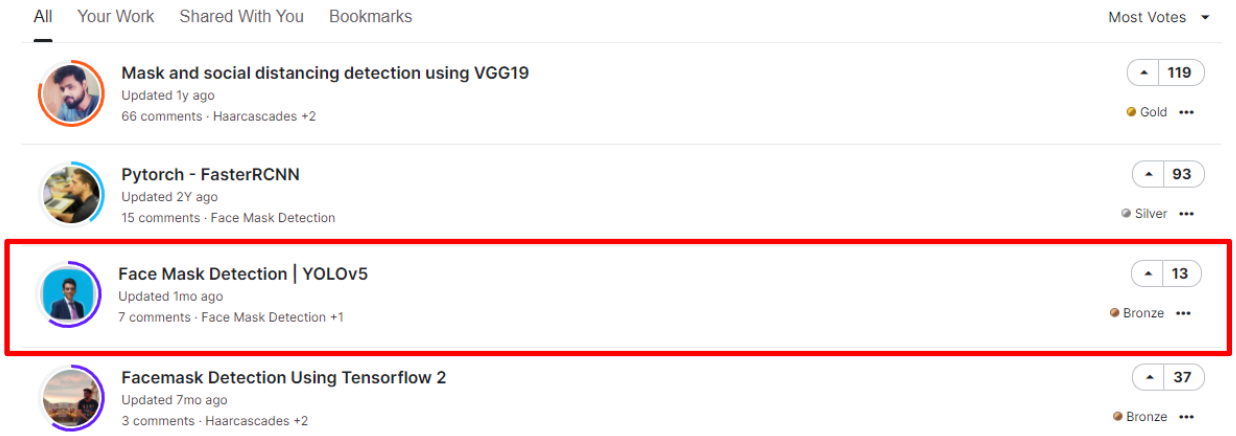

Figure 1 : Etat de l'art des notebooks utilisant le dataset de validation

Ce notebook ainsi que l'exemple de détection fourni sur le Github de YOLOv5 nous a servi de base pour mettre en place la détection de masque.

2. Utilisation de roboflow

Le format des annotations de YOLOv5 est différent du format XML Pascal/VOC. On doit convertir toutes les annotations. Pour cela, nous avons utilisé l'application Web Roboflow. Celle-ci permet de gérer toutes les étapes de l'entrainement d'un modèle. Il permet notamment de convertir en masses des annotations, de découper un dataset en dataset d'entrainement, de validation et de test, d'ajouter de la data augmentation avec une interface graphique.

Chaque image des datatasets d'entrainement et de validation ont été resize en 416 x 416.

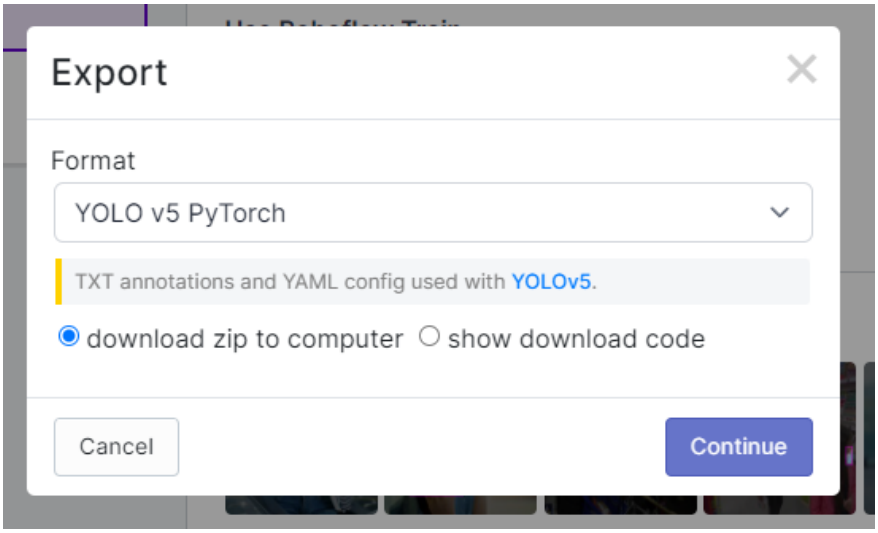

Figure 2 : Export du dataset au bon format et avec une taille fixe

3. Présentation du dépôt Github

   Le lien du dépôt Github est :

   https://github.com/pc260533/MasqueDetection

   Il contient les éléments suivant :

   - Le dossier yolov5 :

     Ce dossier contient le clone du repository :

     https://github.com/ultralytics/yolov5

     On utilise les fichiers train.py pour lancer un entrainement, detect.py pour faire des inférences et val.py pour calculer la valeur du mAP.
   - Le dossier mAP :

     Ce dossier contient le clone du repository :

     https://github.com/Cartucho/mAP

     On utilise le fichier main.py pour calculer la valeur du mAP avec les annotations dataset de validation et les annotations obtenues après prédiction.
   - Le dossier datasets :

     Ce dossier contient les images et les annotations des datasets d'entrainement et de validation.
   - Le dossier results :

     Ce dossier contient trois sous-dossiers.

     Le sous-dossier detect contient les images et les annotations des prédictions du dataset de validation.

     Le sous-dossier train contient les traces de chaque entrainement.

     Le sous-dossier map contient le calcul du mAP obtenue avec val.py et le fichier des meilleurs poids.
   - Le fichier best_20.pt :

     Ce fichier contient les meilleurs poids obtenus à l'entrainement. On utilise ce fichier pour les prédictions.
   - Le fichier yoloNewToYoloOld.py :

     Ce fichier permet de convertir les annotations de la prédiction d'un format YOLO à un autre.
   - Le fichier affichageDetectionDatasetValidation.py :

     Ce fichier permet d'afficher les 10 premières images avec les boudings boxes prédites du dataset de validation.

- Le fichier requirements.txt :

  Ce fichier contient toutes les dépendances à installer dans un environnement virtuel.
- Le fichier streams.txt :

  Ce fichier contient

De plus, le fichier README.md du dépôt décrit la procédure pour :

- Lancer un entrainement avec le dataset d'entrainement,
- Evaluer la solution avec le dataset de validation,
- Faire des inférences et afficher les résultats de la détection en temps réel.

4. Présentation des entrainements

   Plusieurs entrainements ont été réalisés pour obtenir les meilleurs poids.

   Pour évaluer chaque entrainement, on calcule le mAP avec le fichier val.py de YOLOv5. On utilisera la méthode de calcul du mAP de https://github.com/Cartucho/mAP sur le meilleur. Le détail de chaque entrainement est accessible dans le dossier trainResult.

   Entrainement 1 avec les paramètres :

   - Nombre d'epochs : 5,
   - Taille de batch : 16,
   - Optimizer : SGD.

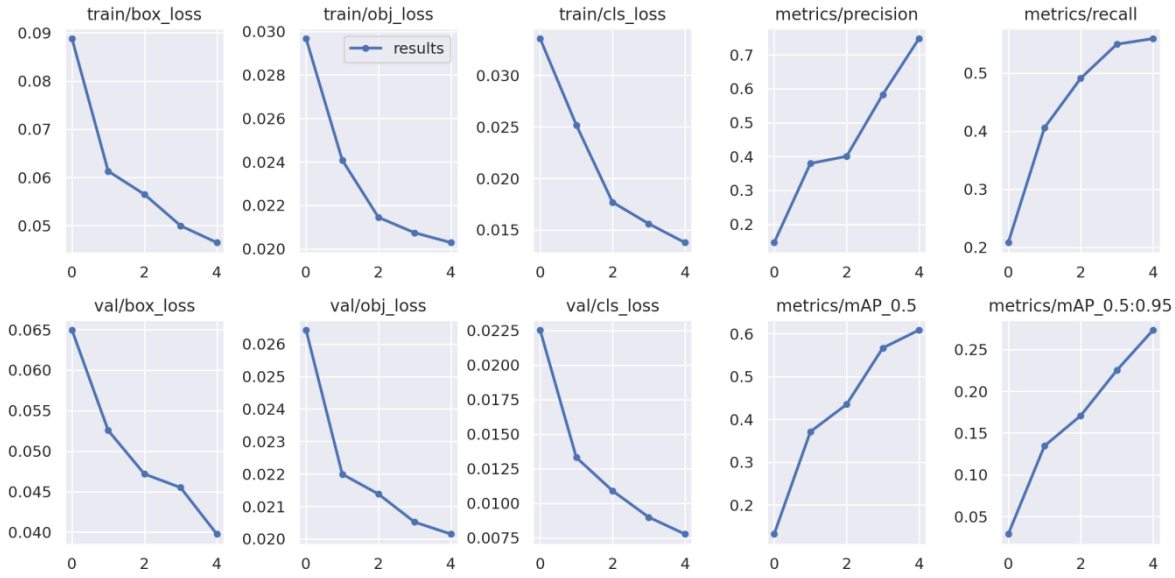

Figure 3 : Résultat de l'entrainement 1

```
val: data=../datasets/data.yaml, weights=['../best.pt'], batch_size=32, imgsz=224, conf_thres=0.001, iou_thres=0.6, task=val, device=, w
orkers=8, single_cls=False, augment=False, verbose=False, save_txt=False, save_hybrid=False, save_conf=False, save_json=False, project=.
./mapDatasetValidation, name=exp, exist_ok=True, half=False, dnn=False
YOLOv5  2022-1-19 torch 1.10.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
val: Scanning '..\datasets\valid\labels.cache' images and labels... 848 found, 0 missing, 0 empty, 0 corrupt: 100%|| 848/848 [00:00<?,
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|          | 27/27 [00:18<00:00,  1.47it/s]
                 all        848       4021       0.71      0.424      0.479      0.201
 with_incorrect_mask        848        121      0.538      0.132      0.186     0.0868
           with_mask        848       3184       0.83      0.709      0.755      0.345
        without_mask        848        716      0.763       0.43      0.497      0.172
Speed: 0.2ms pre-process, 19.2ms inference, 0.7ms NMS per image at shape (32, 3, 224, 224)
Results saved to ..\mapDatasetValidation\exp
Appuyez sur une touche pour continuer...
```

Figure 4 : Calcul du mAP de l'entrainement 1

Entrainement 2 avec les paramètres :

- Nombre d'epochs : 5,
- Taille de batch : 16,
- Optimizer : Adam.

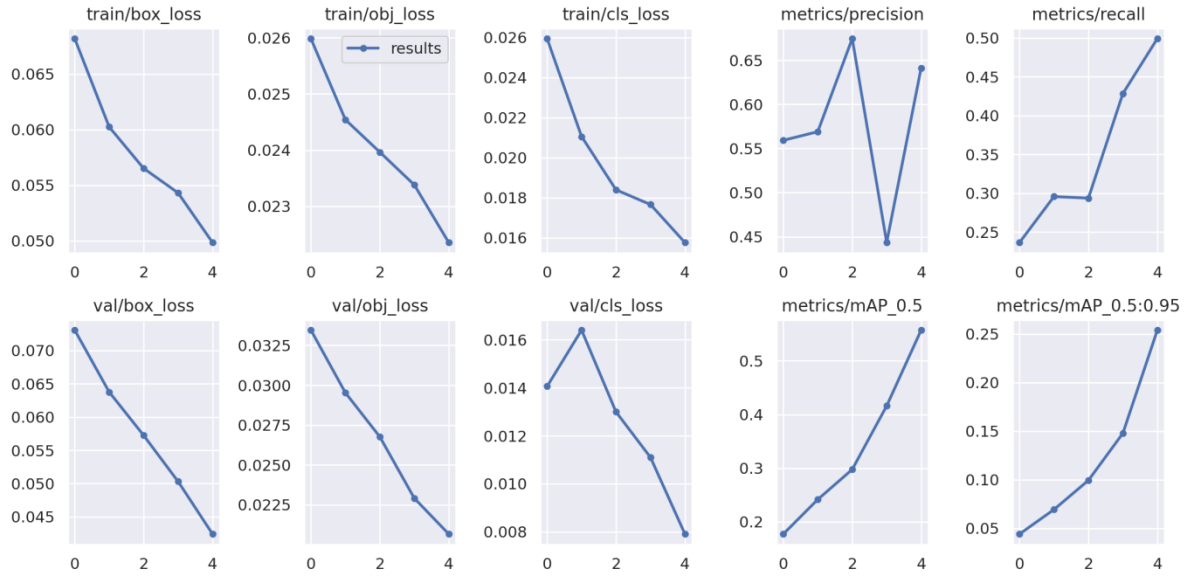

Figure 5 : Résultat de l'entrainement 2

```
val: data=../datasets/data.yaml, weights=['../best.pt'], batch_size=32, imgsz=224, conf_thres=0.001, iou_thres=0.6, task=val, device=, w
orkers=8, single_cls=False, augment=False, verbose=False, save_txt=False, save_hybrid=False, save_conf=False, save_json=False, project=.
./mapDatasetValidation, name=exp, exist_ok=True, half=False, dnn=False
YOLOv5  2022-1-19 torch 1.10.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
val: Scanning '..\datasets\valid\labels.cache' images and labels... 848 found, 0 missing, 0 empty, 0 corrupt: 100%|| 848/848 [00:00<?,
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|| 27/27 [00:18<00:00,  1.50it/s]
                 all        848       4021      0.435      0.489      0.443      0.191
 with_incorrect_mask        848        121      0.296      0.248      0.144     0.0637
           with_mask        848       3184      0.564      0.748      0.742      0.348
        without_mask        848        716      0.446      0.472      0.443       0.16
Speed: 0.2ms pre-process, 18.9ms inference, 0.7ms NMS per image at shape (32, 3, 224, 224)
Results saved to ..\mapDatasetValidation\exp
Appuyez sur une touche pour continuer...
```

Figure 6 : Calcul du mAP de l'entrainement 2

Entrainement 3 avec les paramètres :

- Nombre d'epochs : 5,
- Taille de batch : 16,
- Optimizer : AdamW.

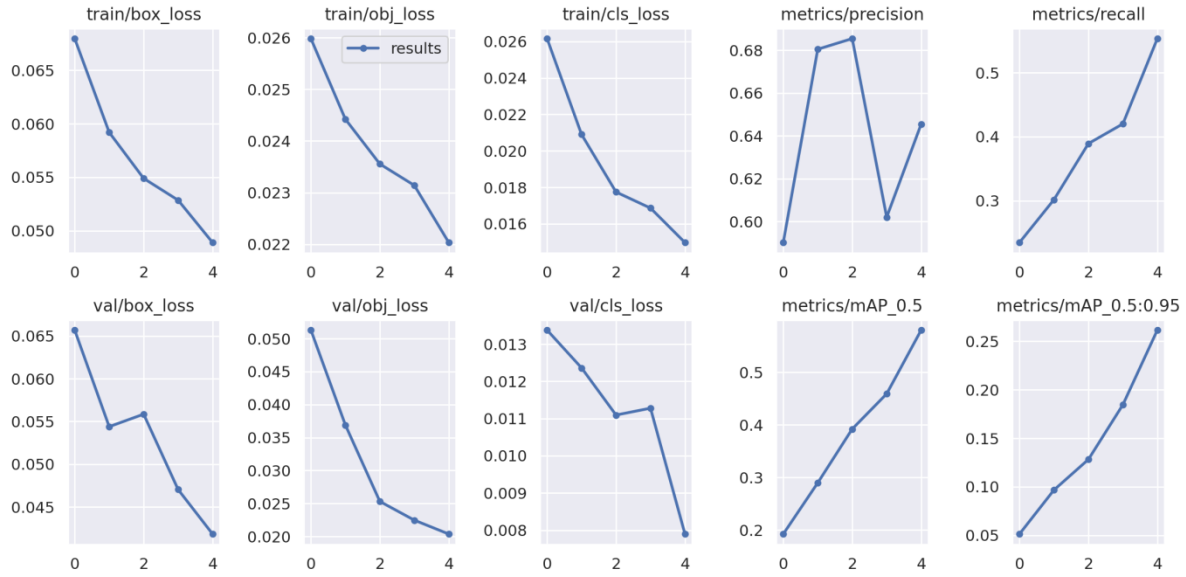

Figure 7 : Résultat de l'entrainement 3

```
val: data=../datasets/data.yaml, weights=['../best.pt'], batch_size=32, imgsz=224, conf_thres=0.001, iou_thres=0.6, task=val, device=, w
orkers=8, single_cls=False, augment=False, verbose=False, save_txt=False, save_hybrid=False, save_conf=False, save_json=False, project=.
./mapDatasetValidation, name=exp, exist_ok=True, half=False, dnn=False
YOLOv5  2022-1-19 torch 1.10.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
val: Scanning '..\datasets\valid\labels.cache' images and labels... 848 found, 0 missing, 0 empty, 0 corrupt: 100%|| 848/848 [00:00<?,
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|| 27/27 [00:18<00:00,  1.47it/s]
                 all        848       4021      0.721      0.415      0.466      0.209
 with_incorrect_mask        848        121      0.679      0.124      0.203      0.109
           with_mask        848       3184      0.841       0.68      0.734      0.339
        without_mask        848        716      0.642      0.441       0.46      0.181
Speed: 0.2ms pre-process, 19.4ms inference, 0.6ms NMS per image at shape (32, 3, 224, 224)
Results saved to ..\mapDatasetValidation\exp
Appuyez sur une touche pour continuer...
```

Figure 8 : Calcul du mAP de l'entrainement 3

On remarque que l'optimizer à peu d'influence. On utilise donc SGD pour les prochains entrainements.

Entrainement 4 avec les paramètres :

- Nombre d'epochs : 5,
- Taille de batch : 128,
- Optimizer : SGD.

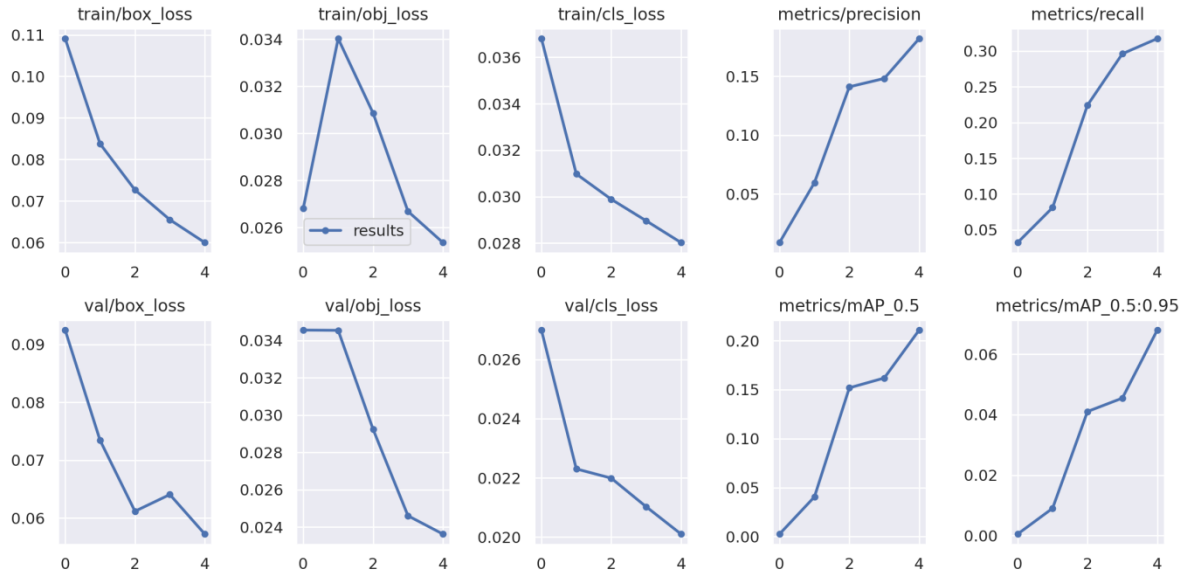

Figure 9 : Résultat de l'entrainement 4

```
val: data=../datasets/data.yaml, weights=['../best.pt'], batch_size=32, imgsz=224, conf_thres=0.001, iou_thres=0.6, task=val, device=, w
orkers=8, single_cls=False, augment=False, verbose=False, save_txt=False, save_hybrid=False, save_conf=False, save_json=False, project=.
./mapDatasetValidation, name=exp, exist_ok=True, half=False, dnn=False
YOLOv5  2022-1-19 torch 1.10.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
val: Scanning '..\datasets\valid\labels.cache' images and labels... 848 found, 0 missing, 0 empty, 0 corrupt: 100%|| 848/848 [00:00<?,
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|| 27/27 [00:20<00:00,  1.29it/s]
                 all        848       4021      0.155       0.21      0.141     0.0405
 with_incorrect_mask        848        121     0.0188    0.00826     0.0153    0.00474
           with_mask        848       3184       0.36      0.516      0.367      0.106
        without_mask        848        716     0.0867      0.107      0.041     0.0108
Speed: 0.2ms pre-process, 19.5ms inference, 3.2ms NMS per image at shape (32, 3, 224, 224)
Results saved to ..\mapDatasetValidation\exp
Appuyez sur une touche pour continuer...
```

Figure 10 : Calcul du mAP de l'entrainement 4

Avec une taille de batch plus grande, la valeur du mAP augmente moins vite. On décide donc pour le dernier entrainement de garder une taille de batch à 16 et d'augmenter le nombre d'epochs.

Entrainement 5 avec les paramètres :

- Nombre d'epochs : 50,
- Taille de batch : 16,
- Optimizer : SGD.

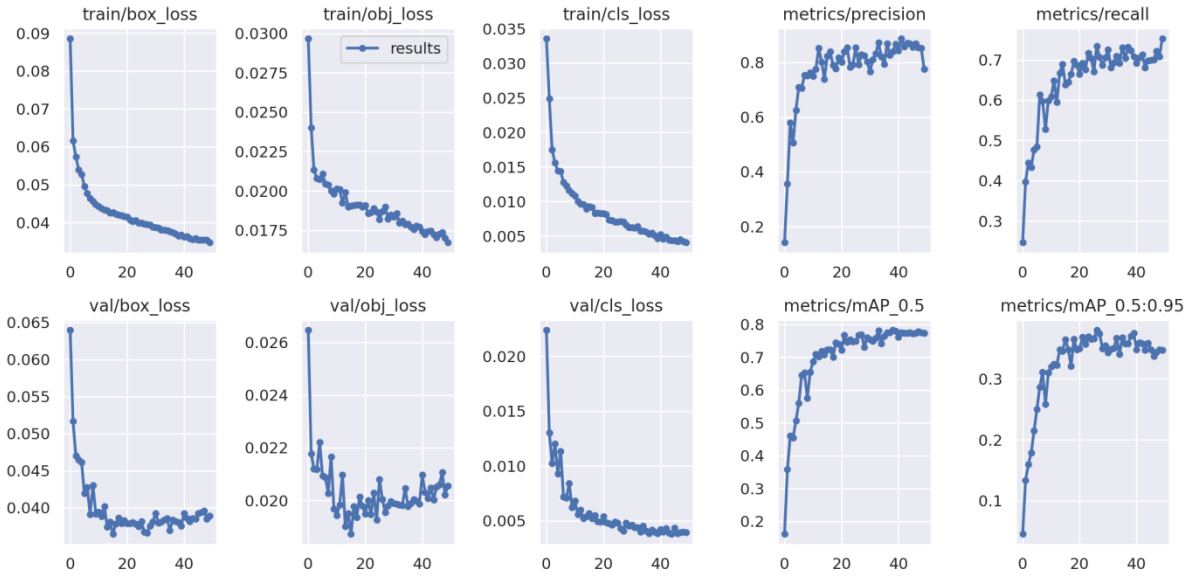

Figure 11 : Résultat de l'entrainement 5

```
val: data=../datasets/data.yaml, weights=['../best_50.pt'], batch_size=32, imgsz=224, conf_thres=0.001, iou_thres=0.6, task=val, device=
, workers=8, single_cls=False, augment=False, verbose=False, save_txt=False, save_hybrid=False, save_conf=False, save_json=False, projec
t=../result/map, name=exp, exist_ok=True, half=False, dnn=False
YOLOv5  2022-1-19 torch 1.10.1+cpu CPU

Fusing layers...
Model Summary: 213 layers, 7018216 parameters, 0 gradients, 15.8 GFLOPs
val: Scanning '..\datasets\valid\labels.cache' images and labels... 848 found, 0 missing, 0 empty, 0 corrupt: 100%|| 848/848 [00:00<?,
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|| 27/27 [00:17<00:00,  1.51it/s]
                 all        848       4021      0.792      0.569      0.642      0.312
 with_incorrect_mask        848        121       0.64      0.471      0.516      0.266
           with_mask        848       3184      0.893      0.745      0.805      0.399
        without_mask        848        716      0.844       0.49      0.605      0.271
Speed: 0.2ms pre-process, 19.0ms inference, 0.5ms NMS per image at shape (32, 3, 224, 224)
Results saved to ..\result\map\exp
Appuyez sur une touche pour continuer...
```

Figure 12 : Calcul du mAP de l'entrainement 5

Les deux images suivantes représentent le récapitulatif des entrainements dans TensorBoard :

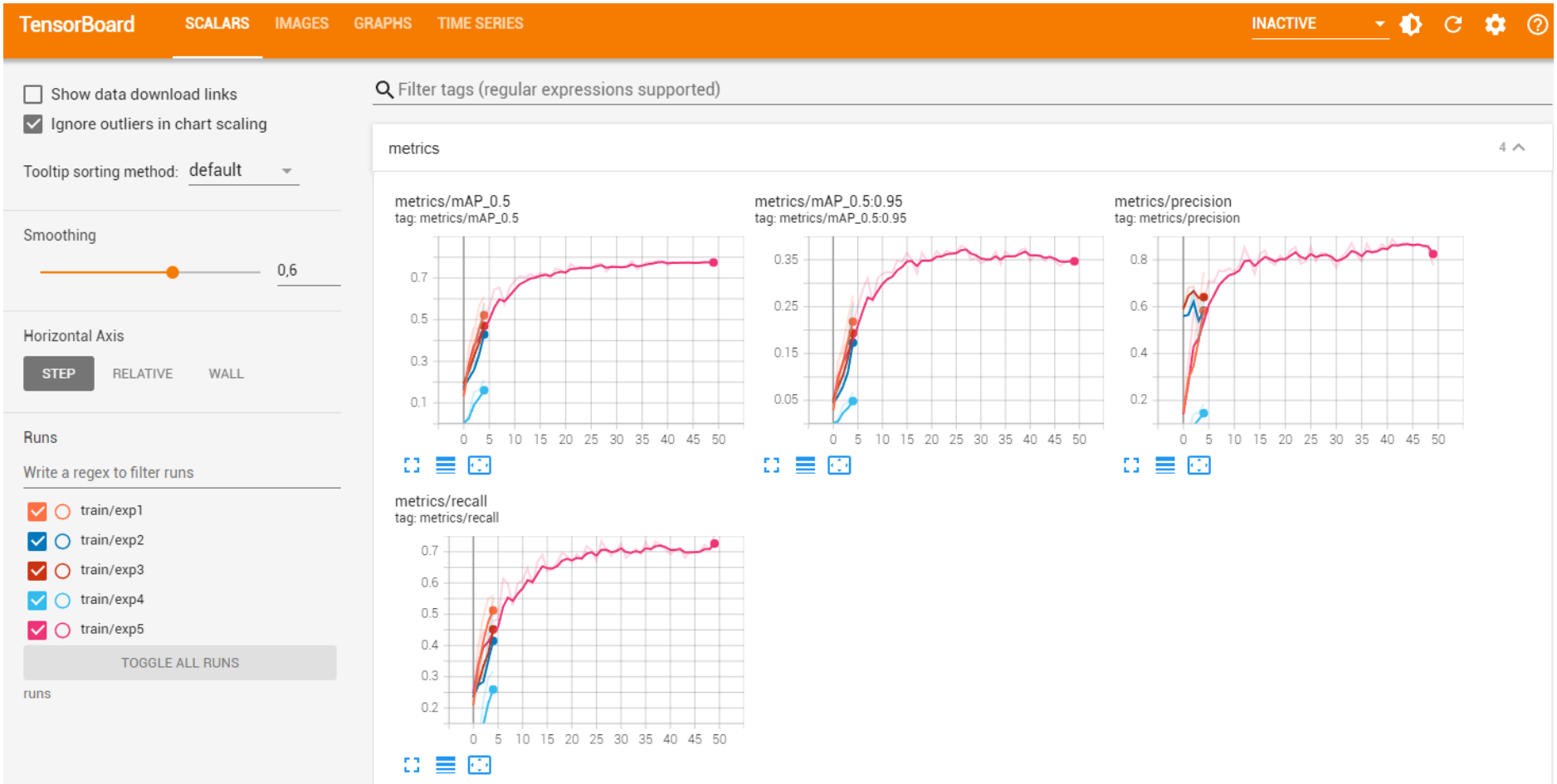

Figure 13 : Récapitulatif des entrainements

Figure 14 : Récapitulatif des entrainements

III. Mise en œuvre et évaluation

1. Evaluation sur la dataset de validation avec les meilleurs poids

On évalue la solution sur le dataset de validation pour fournir les informations de rapidité, de performance de détection. Pour cela, on utilise les fonctions utilisées pour le projet VOC disponible dans le lien suivant :

https://github.com/Cartucho/mAP

On obtient la valeur de mAP suivante :

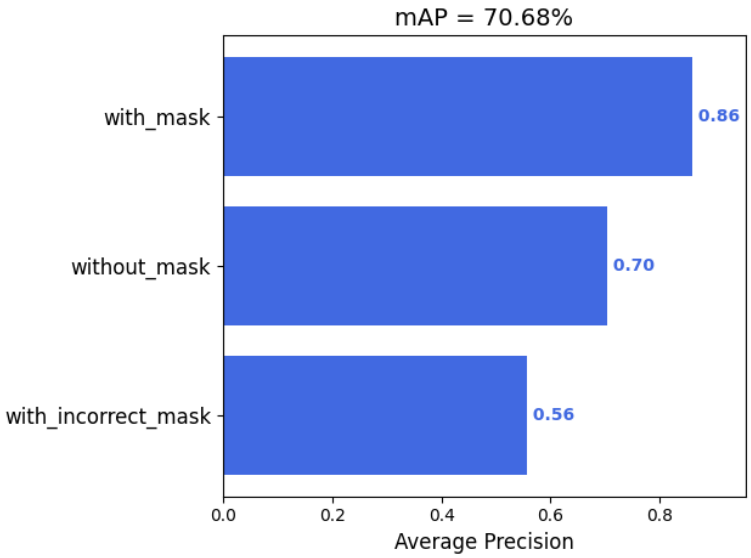

Figure 15 : Calcul du mAP avec cartucho

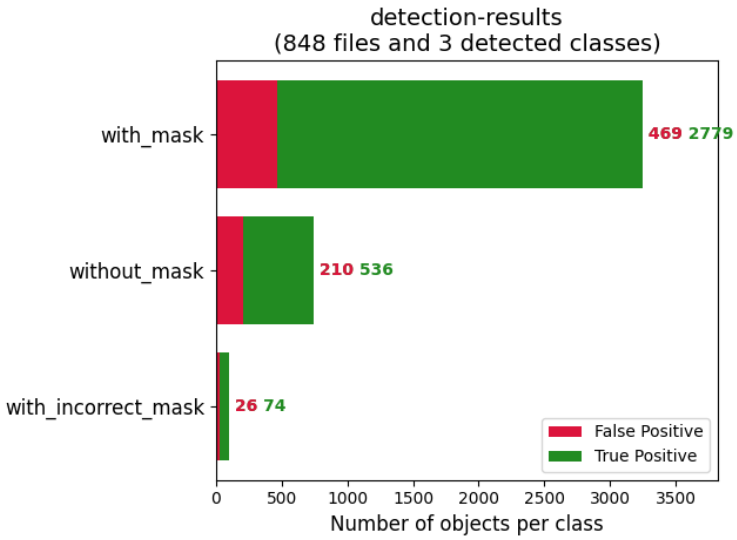

Figure 16 : Nombre de faux positifs par rapport au nombre de vrais positifs

2. Inférence et affichage en temps réel

   En temps réel, on détecte bien les visages avec un masque, sans un masque et avec un masque mal porté. Le temps d'inférence moyen est compris entre 0,01 et 0,02. Le nombre de FPS est donc compris entre 50 et 100.

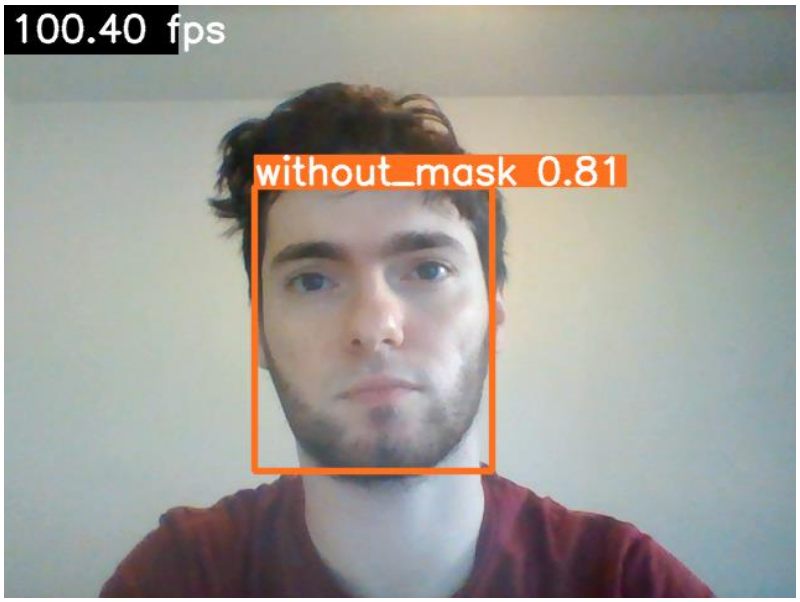

Figure 17 : Détection en temps réel de la classe without_mask

Figure 18 : Détection en temps réel de la classe with_mask

Figure 19 : Détection en temps réel de la classe with_incorrect_mask

Figure 20 : Détection en temps réel de la classe with_incorrect_mask