

UQAC

UNIVERSITÉ DU QUÉBEC
À CHICOUTIMI

— Projet TNI —

Système d'aide à l'entretien des espaces publics : cas particuliers des parcs ou espaces verts

Semestre d'Hiver 2021

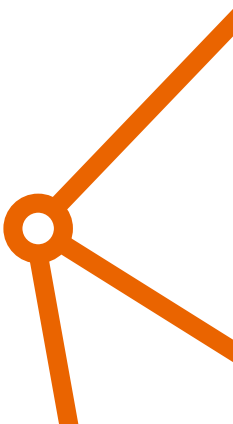
DUQUETTE Steven

POMALEGNI Augustin Primous Prince

Sous la supervision :

Julien Maître

7 May, 2021



Abstract

Les réseaux de neurones artificiels permettent de réaliser plusieurs tâches comme la classification, la régression et bien d'autres encore. Dans ce travail, nous aborderons la mise en place d'un système d'aide à l'entretien qui procédera à la détection des objets qu'il aura appris. Nous partirons de la mise en place d'un dataset pour cette tâche à l'analyse des résultats de notre prototype d'application.

Mots clés: YOLO. Dataset.

Contents

Listes des figures	3
Acronyms	6
1 Introduction	7
1.1 Contexte	7
1.2 Objectifs de l'application	7
1.3 Plan de travail	8
2 Dataset	10
2.1 Choix du Dataset	10
2.2 Mise en place du dataset	11
2.2.1 Les images	11
2.2.2 Les annotations	12
2.3 Présentation du dataset	14
2.3.1 Exemples de fichiers DataSetFond_blanc	15
2.3.2 Exemples de fichiers DataSet_Gazon	17
2.4 Chargement des données dans le projet	21
2.4.1 Mise en place des sous-ensembles	21
2.4.2 Chargement des données	21
3 Choix technologiques et implémentation	22
3.1 Choix technologiques	22
3.1.1 Réseau YOLO	23
3.2 Implémentation et définition des paramètres	24
3.2.1 Réseau YOLO	24

4	Méthodes d'évaluation et résultats des tests	28
4.1	Méthodes d'évaluation	28
4.2	Résultats des tests	29
4.2.1	En phase d'entraînement de validation YOLO : Sur 10 epochs	30
4.2.2	En phase de déploiement : Cas de situation réelle	35
4.3	Conclusion sur les résultats	35
5	Conclusion	37
5.1	Annexe	39
5.1.1	Annexe 1: fichiers joints , documents et code	39
5.1.2	Annexe 2 : Lien vidéos pour voir la détection sur les epochs	39
	References	39

List of Figures

2.1	Exemple d'annotation sur Make Sense : bouteille de coca (fb46.jpg)	13
2.2	Format de téléchargement proposé par Make Sense	13
2.3	Zip de fin du cas exemple	14
2.4	Contenu d'un fichier au format YOLO (.txt)	14
2.5	Constitution du dataSet final	14
2.6	Données de l'image fb0	15
2.7	Données de l'image fb10	16
2.8	Données de l'image fb43	16
2.9	Données de l'image fb55	17
2.10	Données de l'image 2	17
2.11	Données de l'image 66	18
2.12	Données de l'image 257	19
2.13	Données de l'image 2202	19
2.14	Données de l'image 2154	20
3.1	Architecture de YOLOv5	23
3.2	Description imagée des différents modèles	24
3.3	Liste des paramètres de train.py	25
3.4	Liste des paramètres de test.py	26
3.5	Liste des paramètres de detect.py	27
4.1	Interface proposée par l'outil de visualisation Wandb	29
4.2	Décompte des instances d'objets sur l'ensemble du dataset utilisé	30
4.3	Interface de présentation des résultats sur wandb	31
4.4	Matrice de confusion	31
4.5	Images données au modèle (labelisés)	32
4.6	Prédiction avec indice de confiance	33

4.7	La courbe de précision et de rappel	34
4.8	Score F1	34

Acronyms

TNI Traitement Numériques des Images

YOLO You Only Look Once

Chapter 1

Introduction

1.1 Contexte

L'intelligence artificielle est au coeur de nombreuses applications de nos jours. Elle aide à réaliser de nombreuses tâches et peut être utilisée sur des sources de données différentes compte tenu de la nature de son application. Qu'ils s'agissent des données textuelles ou audio qu'on peut utiliser pour mettre en place des RNN performants, ou des CNN qui sont adaptés pour les images et les vidéos, il existe des données de plusieurs natures qui peuvent aider à créer de nouvelles applications dans un monde où la quantité des données accroit de façon exponentielle.

Dans ce sillage, on ne peut que se demander quel plus nous pouvons apporter en nous basant sur les connaissances acquises mais surtout sur les compétences que nous avons développées tout au long de ce module. Pour cela, nous avons pensé à mettre sur pied **un système d'aide à l'entretien des espaces publics** comme les parcs qui vise à seconder les entreprises chargées de la gestion des espaces publics à profiter des merveilles de l'Intelligence Artificielle mais surtout de la force des réseaux YOLO dédiés aux tâches de détection multiples pour fournir des services plus complets.

1.2 Objectifs de l'application

L'application que nous souhaitons mettre en place est, comme nous l'avons dit plus haut, un système d'aide à l'entretien des espaces publics en utilisant le réseau YOLO pour la détection des objets. Dans ce sens, les tâches qui lui seront attribuées sont tributaires de l'environnement dans lequel il sera déployé. Il peut s'agir de l'entretien de villes modernes, qui ne cessent d'être

mis en place par de nombreuses métropoles du monde, de l'entretien d'un parc accueillant les personnes souhaitant profiter de la bonté de la nature ou simplement d'un immeuble dont les propriétaires souhaitent mettre en avant la propreté de leurs biens. Notre application pourrait donc être déployée dans beaucoup de cas d'utilisation. Mais de ce travail, nous nous focaliserons sur le cas d'un parc ou d'un espace vert dédié aux personnes.

Ainsi, compte tenu de la circonscription du cadre d'utilisation de notre application, ces objectifs seront de :

- profiter des installations de surveillances de ces lieux afin de les utiliser comme source de données,
- détecter les éléments qui peuvent considérer comme des intrus dans un espace vert et nuit à la propreté de ce dernier (déchets, feuilles de papier qui traîne, bouteille vide déposée au sol, du plastique) mais aussi des objets que l'on peut facilement perdre ou laisser tomber sans en apercevoir (porte feuille, porte-clé, bracelets ou des goussets de montre)
- Une fois ces éléments détectés, le système devra les envoyer sur un écran de contrôle qui se chargera de notifier la présence des ces éléments au télé-opérateur qui en fonction de l'importance de propreté définit décidera d'envoyer quelqu'un sur les lieux pour le ramasser;
- le système disposera aussi d'une prise en charge automatique dans le cas où on souhaite qu'il se charge de la distribution des ramassages dans un secteur en envoyant une notification à un agent;
- le dernier objectif auquel nous avons pensé est de pouvoir aider à la recherche d'objets perdus par exemple en demandant au système de parcourir les sources de données pour rechercher tel ou tel autre objet perdu par un particulier lors de sa visite;

1.3 Plan de travail

Pour atteindre ces différents objectifs de notre application (qui nous rappelons a été limitée au cadre d'un espace vert disposant de télé surveillance) en proposant un premier prototype, nous partirons de la présentation du Dataset utilisé, des choix technologiques où nous évoquerons le réseau YOLO et l'implémentation que nous mise en place. Nous évoquerons par la suite les résultats

des tests que nous avons menés et enfin nous finirons par la conclusion en évoquant les perspectives de commercialisation et aussi d'amélioration de notre application que nous entrevoyons.

Chapter 2

Dataset

Dans ce chapitre, nous présenterons l'ensemble de dataset qui a été utilisé dans le cadre de la mise en place de notre application mais surtout de sa construction qui a été une tâche assez particulière.

2.1 Choix du Dataset

Pour ce qui est du choix du dataset dans ce travail, nous étions partis sur la base d'utiliser des ensembles de données existants car la plus grande partie du système consistait à une phase d'apprentissage et de détection d'objets car basé sur YOLO. Il était alors question de trouver un ensemble de données d'images qui contiendrait les informations suivantes :

- un ensemble d'images avec les objets que nous souhaitons faire apprendre à notre réseau; la particularité de ces objets étant qu'ils soient des objets qui salissent un environnement (sachets trainant, plastique, bouteille abandonnée, carton, feuille de papier qui traîne) ou des objets que l'on peut perdre en se promenant dans un parc par exemple (bracelet, montre, porte-clé ...);
- ces différents objets présents sur les images du dataset doivent avoir été annotés en utilisant des bounding boxes qui localiseraient leur présence sur l'image pour ensuite sauvegarder ces informations de localisation de l'objet sur l'image. Ces informations de localisation de l'objet X sur l'image 1.jpg peut être :
 1. la classe de l'objet qui peut être un indice de classe ou une chaîne de caractères;
 2. la valeur x du centre de la bounding box qui encadre l'objet;
 3. la valeur y du centre de la bounding box qui encadre l'objet;

4. la largeur de la bounding box qui encadre l'objet;;
5. la hauteur de la bounding box qui encadre l'objet;;

Informations qui peuvent être sauvegardées dans des fichiers au format Pascal VOC(XML ou JSON) ou YOLO (.txt) ou même CSV suivant l'outil qui a été utilisé.

- un fichier regroupant l'ensemble des labels du dataset.

Partant à la recherche du dataset qui nous conviendrait sur les sites comme Kaggle, nous n'avons pas eu gain de cause et avons failli changer d'idées d'application. Mais il s'est avéré que par la suite, on s'est lancé comme défi d'avoir une idée de la charge de travail que peut inclure la mise en place d'un dataset (car habituellement nous récupérons les datasets construits par d'autres personnes) et par la même occasion avoir un dataset qui répondrait à ce que nous recherchions. Et sur ce plan nous avons été servi :).

Allons donc à la découverte de comment nous avons mis en place notre premier dataset.

2.2 Mise en place du dataset

Cette partie pour présenter comment nous avons construit notre dataset.

2.2.1 Les images

Cette étape peut-être divisée en plusieurs sous-étapes comme suit :

Etape 1 : Prise des photos des objets

La première étape étant bien d'avoir des images, nous nous sommes servis de nos téléphones intelligents à défaut d'avoir d'appareils photo. Rappelons qu'avec les smartphones, on peut s'approcher de la qualité photo d'un appareil photo.

Nous avons donc pris des photos des objets que nous souhaitons faire détecter par notre réseau YOLO sous différents angles et sur deux fonds (un fond pelouse pour simuler un parc et un fond blanc). Ce sont donc un ensemble de 14 objets (qu'on peut répartir en deux catégories [objet salissant, objet perdu ou perdable] qui ont été photographiés comme on peut lire leur nom comme suit :

- une bouteille de coca : objet salissant

- du Gel hydroalcoolique : objet salissant
- une montre : objet perdable
- un porte-feuille ou porte-monnaie : objet perdable
- un carton (couleur bleue) : objet salissant
- de petits sachets : objet salissant
- un grand sachet : objet salissant
- un stylo bic : objet salissant ou perdable (si ça a une valeur spéciale);
- des écouteurs : objet perdable mais aussi salissant;
- un sachet de préservatif : objet salissant;
- du plastique : objet salissant;
- un bracelet (de couleur noire) : objet perdable;

Etape 2 : Augmentation des données

Ces photos de base ont ensuite été augmentées en terme de nombre. Il existe pour cela de nombreuses techniques d'augmentation de données (rotation image, changement de luminosité, de contraste, effet de zoom) ou même de simples copies sachant qu'il va falloir au niveau de la création des bounding boxes changer les formes pour ne pas avoir beaucoup de fois la même zone pour un objet.

Etape 3: Resize des images

La taille des images peut affecter la rapidité en phase d'apprentissage. Prévoyant ça, on a redimensionné les images en les faisant passer de 3000*4000 pixels à 500*500 px.

2.2.2 Les annotations

Disposant des images, on peut maintenant passer à la phase d'annotations c'est-à-dire ajouter les bounding boxes et identifier les objets présents en leur associant leur label. Il existe pour cela de nombreux outils dont DarkLabel dont on peut avoir l'exécutable sur son poste, labelMe outil en ligne et aussi le site makesense.ai que nous avons utilisé pour dessiner nos bounding boxes et ceci pour plusieurs raisons :

- la première sa simplicité et son ergonomie;

- sa facilité d'utilisation et les options qu'elle offre dont l'option de se faire aider par une IA qui au besoin mettrait des boxes sur les objets présents sur une image si ces dernières ressemblaient à ceux de sa base d'apprentissage;
- les annotations contenant les infos sur les bounding boxes sont téléchargeables en zip suivant les formats CSV, YOLO(.txt) et VOC (.Xml);
- la dernière raison pour laquelle nous avons choisi (à la place de DarkLabel dont nous avons lu que de bons retours après utilisation en ligne) est liée à la faible puissance de calcul de notre ordinateur qui nous reste lorsqu'on a plusieurs processus de lancés.

Vous pouvez voir sur les figures suivantes l'interface proposée par ce outil en ligne (Veuillez consulter le lien suivant si vous voulez en savoir plus : [Make Sense alpha](#))

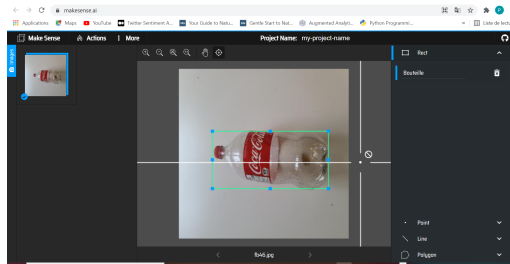


Figure 2.1: Exemple d'annotation sur Make Sense : bouteille de coca (fb46.jpg)

Sur la figure suivante on peut voir les différents formats de téléchargement proposés dont nous avons parlé plus haut.

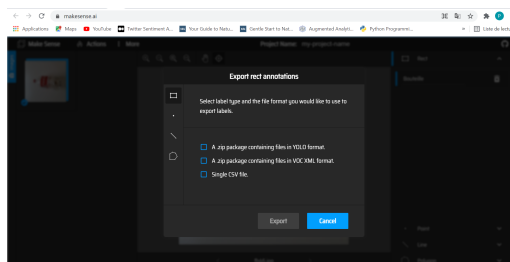


Figure 2.2: Format de téléchargement proposé par Make Sense

La figure suivante est le zip téléchargé suite à notre exemple.

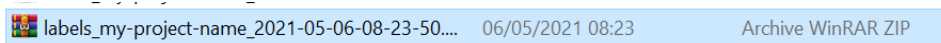


Figure 2.3: Zip de fin du cas exemple

Et pour finir, sur cette figure, on a le contenu du fichier .txt qui porte le nom de l'image exemple et qui contient les informations comme mentionnées à la section 2.1

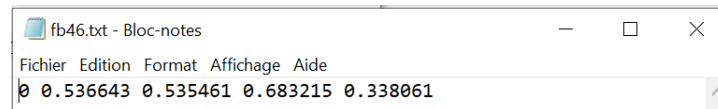


Figure 2.4: Contenu d'un fichier au format YOLO (.txt)

Ainsi, à la suite de ces différentes étapes plus ou moins éprouvantes, nous avons pu définir et construire notre dataset final qui sera présenté dans la section suivante.

2.3 Présentation du dataset

Notre dataset final est un dossier réparti en deux gros sous dossiers : le premier contient les data sur les images prises sur pelouse ou gazon et le second est contient es images sur fond blanc. Il y a aussi un fichier .txt qui contient l'ensemble des 14 labels de la section 2.1.

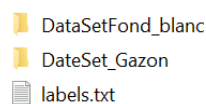


Figure 2.5: Constitution du dataSet final

Chacun des sous-dossiers est réparti comme suit :

DataSetFond_blanc

Il regroupe les images d'objets avec fond blanc. Nous avons donc

- 58 images dans le dossier Images_500_500
- 58 fichiers txt dans le dossier ALL_Labels

DataSet_Gazon

Il regroupe les images prises sur pelouses. Nous avons donc

- 2352 images dans le dossier Images_500_500
- 2352 fichiers txt dans le dossier ALL_Labels

Rappelons que sur ces différentes images nous avons les différents objets de notre label. Nous n'avons pas procédé au décompte suivant chaque label car sur certaines images nous avons mis plusieurs objets.

Nous vous présenterons par la suite les quelques exemples de fichiers de chaque dataset

2.3.1 Exemples de fichiers DataSetFond_blanc

Données de l'image fb0.jpg



(a) L'image

fb0.txt - Bloc-notes				
Fichier	Edition	Format	Affichage	Aide
6	0.471631	0.535461	0.368794	0.446809

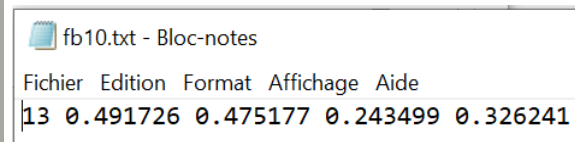
(b) Annotation en txt

Figure 2.6: Données de l'image fb0

Données de l'image fb10.jpg



(a) L'image



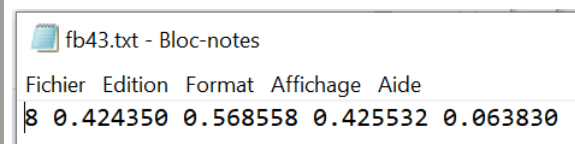
(b) Annotation en txt

Figure 2.7: Données de l'image fb10

Données de l'image fb43.jpg



(a) L'image



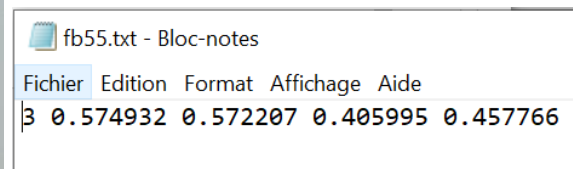
(b) Annotation en txt

Figure 2.8: Données de l'image fb43

Données de l'image fb55.jpg



(a) L'image



(b) Annotation en txt

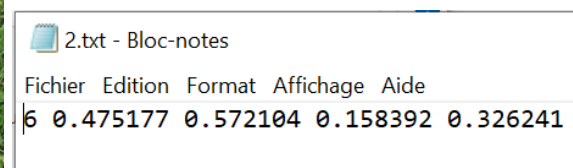
Figure 2.9: Données de l'image fb55

2.3.2 Exemples de fichiers DataSet_Gazon

Données de l'image 2.jpg



(a) L'image



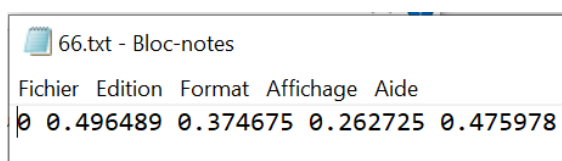
(b) Annotation en txt

Figure 2.10: Données de l'image 2

Données de l'image 66.jpg



(a) L'image



(b) Annotation en txt

Figure 2.11: Données de l'image 66

Données de l'image 257.jpg



Figure 2.12: Données de l'image 257

Données de l'image 2202



Figure 2.13: Données de l'image 2202

Données de l'image 2154.jpg

Sur cet exemple en particulier, on peut noter la présence de plusieurs objets annotés.



(a) L'image

2154.txt - Bloc-notes					
Fichier	Edition	Format	Affichage	Aide	
6	0.509456	0.442080	0.141844	0.170213	
12	0.682033	0.565012	0.108747	0.208038	
5	0.507092	0.672577	0.137116	0.139480	
4	0.639480	0.806147	0.184397	0.208038	

(b) Annotation en txt

Figure 2.14: Données de l'image 2154

On a pu voir tout au long de cette partie comment nous avons pu mettre sur place notre ensemble de données allant de la prise des photos aux annotations des objets. C'est un exercice très éprouvant. Reste à savoir si la manière dont le dataset a été construit et sa taille n'impactera pas de façon négative la qualité des résultats que nous obtiendrons.

Evoquons à présent le chargement des données dans le projet.

2.4 Chargement des données dans le projet

Pour ce qui est du chargement des données, on peut le subdiviser en deux étapes : la mise en place des différents ensembles (train, test et validation) et le chargement des dits ensembles dans le projet.

2.4.1 Mise en place des sous-ensembles

Les différents sous-ensembles sont directement concus ou créés par le script lui-même qui s'occupe de faire les associations entre image et annotation txt correspondant.

2.4.2 Chargement des données

En ce qui concerne le chargement des données, on lui spécifie le chemin des données avant exécution. Ainsi, les données sont chargées au cours de leur appel pendant l'exécution.

Nous avons pu aborder dans ce chapitre, la méthodologie que nous avons utilisée pour construire notre dataset afin qu'il se rapproche plus de la nature des données que nous souhaitons utiliser. Cette phase de construction du dataset était assez intéressante et nous pouvons comprendre tout le travail qui est fait par les auteurs sur chaque dataset que nous récupérons sur les sites qui en fournissent dont fait partie Kaggle. Il serait intéressant pour nous, de partager ce dataset afin d'avoir des retours qui pourront sûrement nous aider à affiner notre méthodologie.

Chapter 3

Choix technologiques et implémentation

Dans ce chapitre où nous nous approchons de plus en plus de la mise en place de notre système(application), nous aborderons et justifierons les choix technologiques que nous avons faits et nous présenterons également l'implémentation faite dans le cadre de ce travail.

3.1 Choix technologiques

Nous avons choisi de travailler sur le premier sujet du projet soit : "La première est de travailler avec des vidéos et la détection d'objets multiples à l'aide d'un algorithme de type YOLO, RetinaNet ou encore DeepSort." Nous avons opté pour YOLO puisque c'est l'algo que nous trouvions intéressant et qui fait partie des plus populaires disposant de plusieurs documentations faites par les différents auteurs suivant la version.

Il est bon de mentionner que nous avons fait des tests d'implémentation en partant de zéro, mais nous avons eu des mauvais résultats. Ce qui nous a poussé à partir d'implémentation complète disponible en ligne notamment sur le site Github qui propose un panoplie d'implémentation de YOLO sous ces différentes versions mais aussi avec plusieurs framework (tensorflow keras de Google, torch Vision de Facebook ...). Nous avons trouvé une implémentation sur Git qui répondait à nos besoins notamment en terme de documentation sur comment l'utiliser. Nous nous sommes donc servi de cette implémentation pour construire le premier prototype de notre application.

Cependant avant d'aller en détails sur le modèle que nous avons utilisé comme base, nous n'avons pas négliger le fait d'en apprendre un peu plus sur le réseau YOLO en partant de la lecture d'un des articles écrit par les concepteurs des réseaux YOLO. L'article porte le titre de **"You Only Look Once: Unified, Real-Time Object Detection"**, et est écrit par **Joseph Redmon, Ross Girshick, Santosh Divvala et Ali Farhadi**. Il est accessible par ce lien [Article YOLO](#).

3.1.1 Réseau YOLO

Comme mentionné ci-dessus, nous sommes partis d'une implémentation existante, disponible sur le dépôt github [suivant](#), créé par **Ultralytics**, bien documenté et qui a été construite en utilisant principalement le framework torch(alternative de tensorflow).C'était donc l'occasion pour nous d'en apprendre plus sur ce outil et d'ajouter ces connaissances à celles reçues en utilisant tensorflow.

Cette implémentation utilise YOLOv5. YOLOv5 est une famille de modèles de détections d'objets à l'échelle composée qui a été entraîné sur l'ensemble de données COCO. COCO est un ensemble de données de détection, de segmentation et de sous-titrage d'objets à grande échelle. On peut définir son architecture(YOLOv5) comme sur la figure suivante et sa présentation complète est disponible sur le lien [Architecture YOLOv5](#)

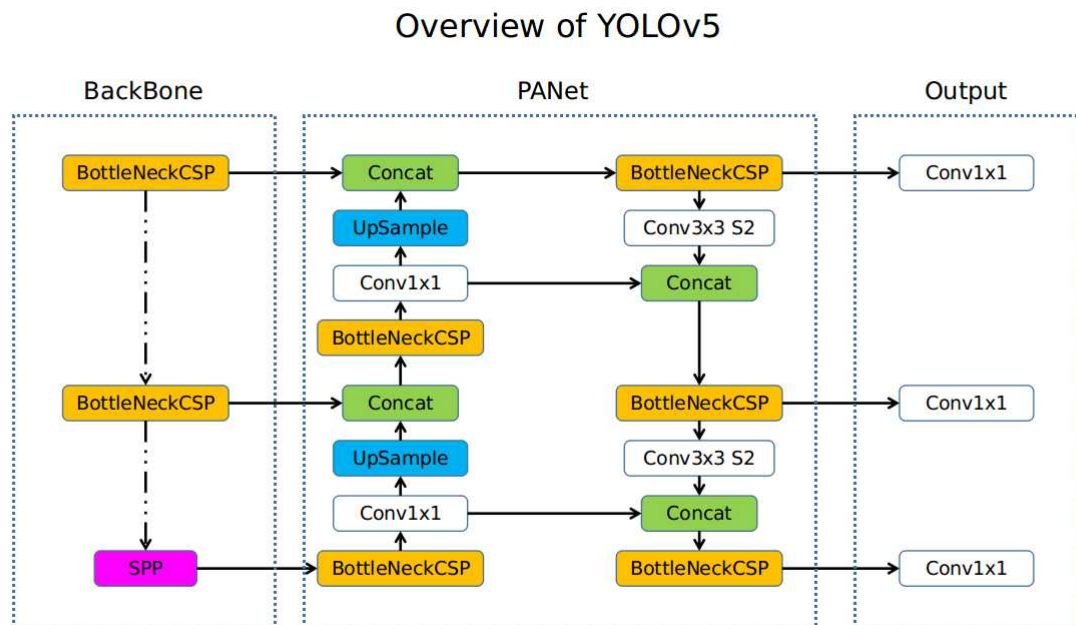


Figure 3.1: Architecture de YOLOv5

L'implémentation proposée a été conçue en plusieurs modèles comme sur la figure suivante, chaque modèle ayant ses caractéristiques.




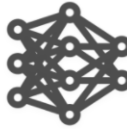
MODEL DESCRIPTION			
			
Small YOLOv5s	Medium YOLOv5m	Large YOLOv5l	XLarge YOLOv5x
14 MB _{FP16} 2.2 ms _{V100} 36.8 mAP _{COCO}	41 MB _{FP16} 2.9 ms _{V100} 44.5 mAP _{COCO}	90 MB _{FP16} 3.8 ms _{V100} 48.1 mAP _{COCO}	168 MB _{FP16} 6.0 ms _{V100} 50.1 mAP _{COCO}

Figure 3.2: Description imagée des différents modèles

Nous avons utilisé le modèle YOLOv5s qui est le modèle le plus simple, mais également le plus rapide. Nous avons fait ce choix, parce que notre objectif est de détecter des objets dans un parc en temps réel. Nous considérons que le modèle proposé doit être assez rapide pour réaliser cette tâche.

3.2 Implémentation et définition des paramètres

3.2.1 Réseau YOLO

Après avoir parcouru les différents répertoires et code de base pour comprendre la conception qui a été faite, nous pouvons séparer la finalité de l'implémentation en trois parties (ou en trois fichiers) : la première étant le `train.py`, la deuxième étant `test.py` et la troisième étant `detect.py`.

Train.py

Voici une capture d'écran de la liste des arguments que le programme prend en paramètres.

```

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default='yolov5s.pt', help='initial weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--data', type=str, default='data/coco128.yaml', help='data.yaml path')
    parser.add_argument('--hyp', type=str, default='data/hyp.scratch.yaml', help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300)
    parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all GPUs')
    parser.add_argument('--img-size', nargs='+', type=int, default=[640, 640], help='[train, test] image sizes')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--notest', action='store_true', help='only test final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
    parser.add_argument('--evolve', action='store_true', help='evolve hyperparameters')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache-images', action='store_true', help='cache images for faster training')
    parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
    parser.add_argument('--adam', action='store_true', help='use torch.optim.Adam() optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not modify')
    parser.add_argument('--workers', type=int, default=8, help='maximum number of dataloader workers')
    parser.add_argument('--project', default='runs/train', help='save to project/name')
    parser.add_argument('--entity', default=None, help='W&B entity')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--quad', action='store_true', help='quad dataloader')
    parser.add_argument('--linear-lr', action='store_true', help='linear LR')
    parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
    parser.add_argument('--upload_dataset', action='store_true', help='Upload dataset as W&B artifact table')
    parser.add_argument('--bbox_interval', type=int, default=-1, help='Set bounding-box image logging interval for W&B')
    parser.add_argument('--save_period', type=int, default=-1, help='Log model after every "save_period" epoch')
    parser.add_argument('--artifact_alias', type=str, default="latest", help='version of dataset artifact to be used')
    opt = parser.parse_args()

```

Figure 3.3: Liste des paramètres de train.py

Les paramètres que nous avons utilisés sont :

1. weights qui dicte ou récupère les poids d'un modèle déjà entraîné (transfer learning), Nous avons chargé les poids de yolov5s.pt qui sont ceux obtenus après entraînement sur dataset de type COCO (pour les raisons mentionnées plutôt);
2. l'argument data en créant un fichier d'extension .yaml dont le programme a besoin pour trouver les données sur lesquelles nous voulions que le modèle s'entraîne et aussi les différents labels du dataset;
3. l'époque qui dicte le nombre d'époques, nous avons fait des tests sur 5, 10 et 15 époques.
4. batchsize: Nous avons également fait des tests sur le batchsize que nous avons fini par laisser à deux. Raison fortement liée à la puissance de calcul du matériel. Nous nous doutons bien que ceci pourrait affecter la qualité de nos résultats.
5. img-size qui dicte la taille en pixel des images qui est de 500*500px pour les images de notre dataset.

Test.py

Ci-dessous une capture d'écran de la liste des arguments que le programme prend en paramètres pour la phase de test.

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(prog='test.py')
    parser.add_argument('--weights', nargs='+', type=str, default='yolov5s.pt', help='model.pt path(s)')
    parser.add_argument('--data', type=str, default='data/coco128.yaml', help='*.data path')
    parser.add_argument('--batch-size', type=int, default=32, help='size of each image batch')
    parser.add_argument('--img-size', type=int, default=640, help='inference size (pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.001, help='object confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.6, help='IOU threshold for NMS')
    parser.add_argument('--task', default='val', help='train, val, test, speed or study')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--single-cls', action='store_true', help='treat as single-class dataset')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--verbose', action='store_true', help='report mAP by class')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-hybrid', action='store_true', help='save label+prediction hybrid results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-json', action='store_true', help='save a cocoapi-compatible JSON results file')
    parser.add_argument('--project', default='runs/test', help='save to project/name')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    opt = parser.parse_args()
    opt.save_json |= opt.data.endswith('coco.yaml')
    opt.data = check_file(opt.data)  # check file
    print(opt)
    check_requirements(exclude=('tensorboard', 'pycocotools', 'thop'))
```

Figure 3.4: Liste des paramètres de test.py

Ici, nous avons seulement changé le paramètre weight en prenant soin de mettre le modèle que nous avons calculé avec le train.py et qui présentait les meilleurs scores pendant l'apprentissage et aussi le même .yaml passé au script train.py.

Detect.py

Voici une capture d'écran de la liste des arguments que le programme prend en paramètre .

```

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default='yolov5s.pt', help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='data/images', help='source') # file/folder, 0 for webcam
    parser.add_argument('--img-size', type=int, default=640, help='inference size (pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.25, help='object confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='IOU threshold for NMS')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='display results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default='runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    opt = parser.parse_args()
    print(opt)
    check_requirements(exclude=('tensorboard', 'pycocotools', 'thop'))

```

Figure 3.5: Liste des paramètres de detect.py

Ici tout comme le test.py nous avons seulement changé le paramètre weights, puis le paramètre source. Pour source, nous avons fait un vidéo pour voir comment le modèle allait réagir.

Nous avons présenté dans ce chapitre les choix, technologiques opérés et les raisons qui nous ont poussé à ces choix. Aussi, nous avons abordé la présentation du réseau que nous avons utilisé comme base de notre travail et son implémentation sans oublier les choix de paramètres que nous avons faits pour nos différents tests.

Chapter 4

Méthodes d'évaluation et résultats des tests

Dans ce chapitre nous mentionnerons les différentes méthodes d'évaluation qui ont été utilisées afin de juger de l'efficacité de l'autre modèle et des résultats des tests que nous avons obtenus au cours des différentes phases.

4.1 Méthodes d'évaluation

En Intelligence Artificielle, il existe différentes méthodes d'évaluation d'un système. Qu'il s'agisse de la matrix de confusion ou plus encore du score F1, ces différentes valeurs permettent de juger les résultats de son modèle et savoir également comment ce dernier évolue. Il y a aussi les courbes d'apprentissage que l'on obtient à la fin des phases de training qui peuvent également servi. Cependant dans le domaine du computer vision et notamment pour la détection d'objet, nous pensons que l'une des valeurs d'évaluation d'un modèle qui est intéressante est le score **mAP** qui se base sur la courbe de précision et de rappel du modèle afin de tirer les conclusions. Etant donné que dans notre applications, nous avons plus de deux classes le mAP est calculé en définissant un seuil IOU qui a été fixé à **0.5** dans notre travail. On peut ajouter à ça l'indice de confiance qui est mis sur les boxes pour juger des perfs du modèle.

En ce qui concerne la visualisation de ces différentes mesures, il est possible de les avoir en les affichant ou en les enregistrant. Cependant il est également possible de le faire grâce à un outil en ligne qui se charge de nous les dessiner au cours des phases de training et de validation du modèle : le site [Weights and Biases](#) qui donne accès à un ensemble d'outils pour développeurs

en machine learning. Il propose une interface très intuitive et le seul pré-requis est d'installer le package wandb sur sa machine au moyen de la commande

```
pip install wandb
```

La suite n'est qu'une succession de simples étapes pour accéder aux résultats de son travail pendant les différentes phases.

On peut voir sur l'image qui suit l'interface proposée par ce site.

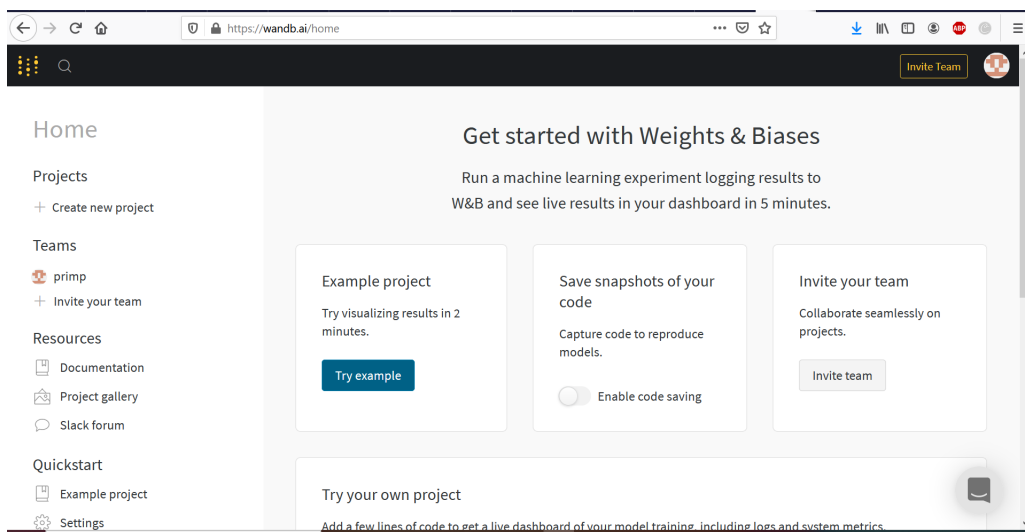


Figure 4.1: Interface proposée par l'outil de visualisation Wandb

Après cette présentation des mesures de perfs qui pourraient nous permettre d'évaluer le modèle, nous passons à la présentation des différents résultats obtenus.

4.2 Résultats des tests

Pour ce qui est des résultats, nous pouvons les subdiviser en deux phases : la phase d'entraînement et de validation et de test et la phase de déploiement pour simuler le cas pratique d'un flux vidéo. Ainsi pour des soucis de clarté de notre travail, nous évoquerons ici que les résultats obtenus avec 10 epochs. Nous mettrons en annexe un dossier "result" contenant l'ensemble des tests par étape de train, de test et de detect pour chaque nombre d'epochs que nous avons testé (5, 10, 15). Le sous-dossier datasetTrain contient les graphiques pour l'entraînement, le sous dossier datasetTest contient les graphiques pour les tests, et le sous dossier datasetDetect contient la vidéo traitée par

notre modèle (il affiche le bounding box et la classe ainsi que son degré de certitude sur la classe attribuée)

Nous pouvons mentionner que les tests effectués ont été faits sur un ordinateur Dell G5 15 ayant 16 Gig de ram, un processeur intel I7 de 8iem Gen et une carte graphique Nvidia Geforce GTX 1060.

Cependant, nous devons également préciser que les images sur fond blanc n'ont pas été utilisées au cours des différentes phases car après les premiers tests nous n'avions pas eu des résultats satisfaisants compte tenu de la petite quantité de ce type de données (58 images). Nous sommes donc restés sur les images sur pelouse avec leurs annotations en txt. Nous pouvons également expliquer ces mauvais résultats par le fait qu'il n'y avait aucune image sur fond blanc dans la phase de training pour avoir plus de caractéristiques des objets.

Ainsi sur l'ensemble des images, nous nous retrouvons avec les objets qui se retrouvent comme suit:

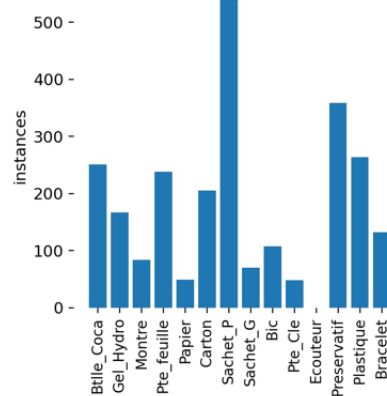


Figure 4.2: Décompte des instances d'objets sur l'ensemble du dataset utilisé

4.2.1 En phase d'entraînement de validation YOLO : Sur 10 epochs

Comme mentionné plus haut, on a une vue globale sur l'ensemble des résultats avec l'outil en ligne **Weights and Biases** comme le montre la figure qui suit .

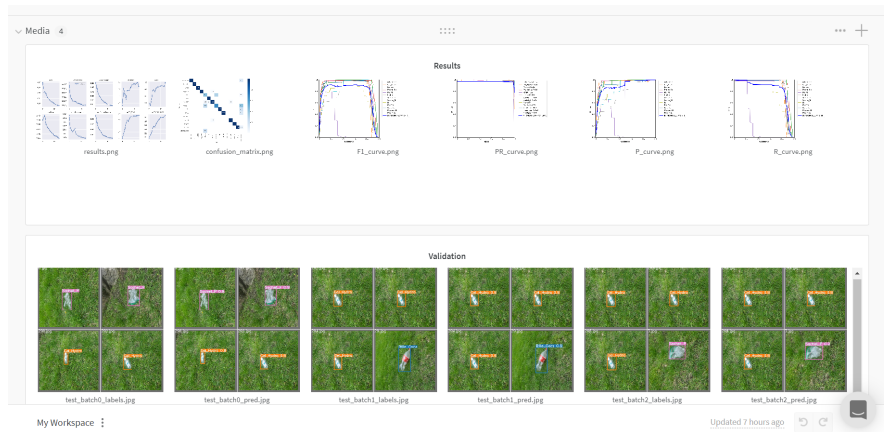


Figure 4.3: Interface de présentation des résultats sur wandb

Nous allons nous intéresser à trois courbes(ou mesures) en particulier.

La matrice de confusion

Un graphique intéressant est le graphique de confusion. C'est le graphique qui montre le nombre d'éléments bien identifiés (le programme compare la prédiction faite au true value en se basant sur le label donné pour cet objet).

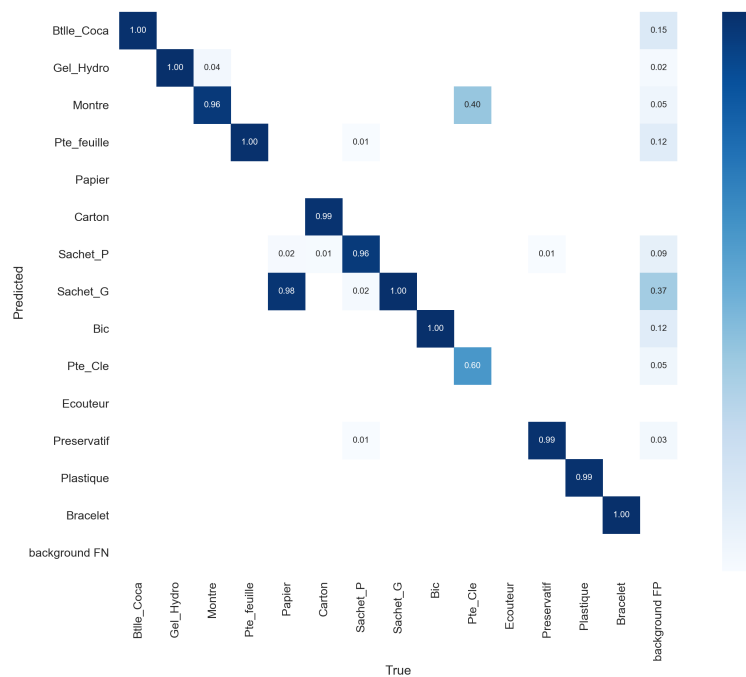


Figure 4.4: Matrice de confusion

Nous pouvons voir sur ce graphique que certains objets ont été plus difficiles pour le modèle à prédire comme par exemple Pte-cle ou encore Papier. Pour le reste, nous pouvons voir que le modèle est assez efficace.

Ci-dessous un exemple d'images données au modèle versus ce qu'il a prédit. Le nombre après la classe de l'objet est son degré de certitude que l'objet actuelle est bel et bien l'objet sur l'image.



Figure 4.5: Images données au modèle (labelisés)



Figure 4.6: Prédiction avec indice de confiance

La précision et le rappel et mAP@0.5

Pour ces mesures, on peut voir sur la figure suivante qu'on obtient une mAP avec un seuil IOU défini à 0.5 qui avoisine le 0.95. Une légère baisse est à remarquer sur les objets de Papier ce qui est peut-être dû à leur faible représentation sur l'ensemble des données.

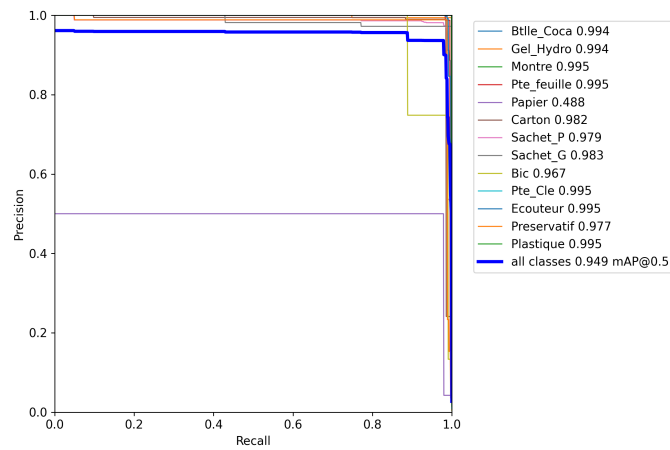


Figure 4.7: La courbe de précision et de rappel

Le score F1 : l'indice de confiance

Pour ce qui est de l'indice de confiance qui permet de définir la certitude avec laquelle on est sûr que l'objet détecté appartient à la classe qui lui est assignée, il oscille entre 0.7 et 0.9 ce que nous avons jugé pas mal pour le nombre d'épochs qui est de 10 dans ce cas.

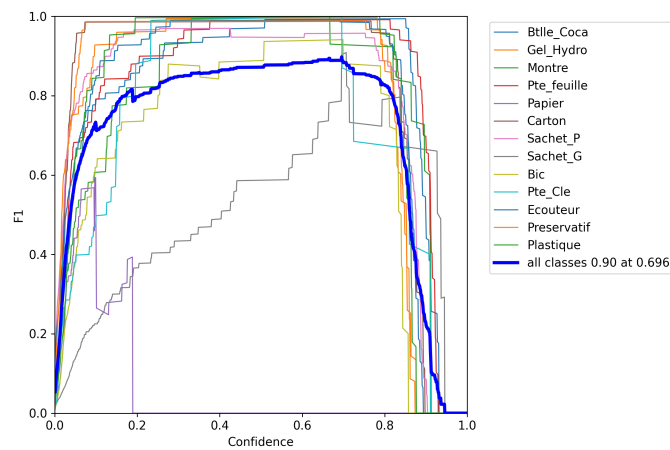


Figure 4.8: Score F1

En général, les différents résultats sur les mesures présentées sont encore meilleurs en phase de test notamment comme vous pouvez le voir dans les fichiers mis en annexe. Il en va que ces résultats soient différents pour les époques 5 et 15 mais nous ne les présenterons pas afin de ne pas allourdir le document. Nous vous invitons néanmoins à les consulter afin d'en avoir une idée et de remarquer l'importance qu'à le nombre d'époques sur tout système basé sur les réseaux de neurones.

4.2.2 En phase de déploiement : Cas de situation réelle

Pour simuler un cas de situation réelle comme l'utilisation de notre modèle sur un flux vidéo venant de caméra, nous avons utilisé une vidéo montée à partir des images de notre dataset. Elle est en annexe de ce document.

Le résultat obtenu après les détections faites en utilisant la sauvegarde du réseau sur 10 epochs est également en annexe de ce document. Mais vous pouvez également y accéder en cliquant sur ce lien [Test Vidéo10 avec annotations](#).

Vous pouvez voir sur la vidéo que les différents objets sont détectés en temps réels même s'il y a une erreur de détection sur le papier qui est labellisé Préservatif vers la fin de la vidéo. Nous avons émis l'hypothèse que cela était dû à la faible représentativité des objets "Papier" dans l'ensemble du dataset.

4.3 Conclusion sur les résultats

Sur l'ensemble des résultats présentés et sur ceux que nous avons mis en annexe et qui sont liés aux epochs 5 et 15, nous pouvons dire que nos meilleurs résultats sont obtenus, lorsque, nous avons entraîné le modèle sur 10 epochs. Ça semble être petit, mais nous n'avions que 2500 images pour 14 classes. A 15 epochs, nous abordions déjà en over-fitting et à 5 epochs, l'indice de confiance était un peu bas.

Nous avons ainsi noté l'importance du nombre d'époques sur les performances d'un système. Aussi, il est très important de disposer d'un ensemble de données bien garni avec dans le meilleur des cas, les différents objets de chaque classe en forte représentativité. C'est deux facteurs permettraient d'augmenter les perfs du système et d'avoir de meilleurs résultats. Il est également possible et nous n'avons vu dans les différents tests menés de modifier les paramètres du réseau dont le taux d'apprentissage afin d'avoir une amélioration en prédiction. Enfin, le fait de disposer d'une machine avec de bonnes caractéristiques et une meilleure puissance de calcul permettrait de

tester sur plusieurs batch size afin d'affiner la correction des poids pendant la descente du gradient en phase d'apprentissage.

Nous sommes néanmoins satisfaits des résultats de détection que nous avons pour une version alpha de notre application et reconnaissons que l'atteinte du système parfait nécessite encore plus de travail sur le dataset et sur le modèle.

Chapter 5

Conclusion

En conclusion, pour la mise en place du premier prototype de notre application, nous sommes passés par plusieurs étapes; de la mise en place du dataset qui a été un exercice très éprouvant mais dont nous avons appris beaucoup de choses à la phase de déploiement et de test sans oublier la phase d'implémentation en se basant sur une conception faite avec Torch et disponible en ligne. En testant par la suite notre modèle sur l'ensemble de données, nous avons eu une diversité de résultats dont les meilleurs que nous avons jugés sont ceux obtenus après 10 epochs d'apprentissage. L'objectif de notre application qui consiste nous le rappelons à faire des détections d'objets salissant les parcs ou les espaces verts et à retrouver des objets perdus a été atteint pour un premier prototype. Cependant, il demeure possible et c'est une perspective d'amélioration de paufiner plus le modèle pour obtenir de meilleurs résultats sur les versions suivantes de notre application.

Il y a donc plusieurs pistes pour amener ce travail encore plus loin. Premièrement, il faudrait agrandir le data set en y ajoutant des exemples d'objets avec plusieurs fond (arrière-plan) : des objets sur du sable, sur de l'asphalte, du gravier, etc qui sont d'autres types d'environnements possibles dans un parc. Deuxièmement, nous pourrions ajouter des classes pour rendre ce model plus complet. Troisièmement, en ajoutant des classes comme les seringues et en utilisant des images en temps réel d'un parc, nous pourrions former notre application pour qu'il soit capable d'aviser lorsqu'il détecte des objets dangereux comme les seringues afin d'éviter de possibles répercussions sur la santé. Le plus serait de partir du modèle pour choisir quel parc ou quelle zone du parc à nettoyer en premier. Ce qui serait un avantage sur nos concurrents qui sont les entreprises de nettoyage et même des mairies.

D'autres perspectives d'adaption sont également possibles comme l'usage de ce même système dans les rues, les marchés et ce même avec l'usage de drone à courte ou moyenne portée et pourquoi pas ajouter dans un futur proche une interface GUI.

Pour ce qui est des perspectives de commercialisation, il serait intéressant de monter un business plan afin de voir la viabilité d'une entreprise spécialisée dans ce genre d'application ou de partir de connaissances ou même d'enseignants qui pourraient nous aider dans la mise en place d'une startup. Bien évidemment, avant toute commercialisation on devrait se rassurer que la licence de l'implémentation de base utilisée permet de s'en servir pour faire de la commercialisation et dans le cas contraire apporter des corrections qu'il faut afin d'y remédier. Cela est d'une importance capitale car étant que futurs ingénieurs, il est primordial de respecter le travail d'un autre : il en va de l'éthique de notre future profession.

5.1 Annexe

5.1.1 Annexe 1: fichiers joints , documents et code

- le script utilisé pour construire le dataset;
- dataset utilisé contenant les images et les annotations;
- le fichier contenant la liste des différents labels du dataset;
- code source de l'implémentation complète du modèle (contenant le répertoire yolov5 et sa configuration pour accéder aux données)
- l'ensemble des résultats pour les différents epochs 5, 10 et 15 en passant du train et test sur les images à la détection sur la vidéo.

5.1.2 Annexe 2 : Lien vidéos pour voir la détection sur les epochs

1. sur 5 epochs : [TestVidéo5 avec annotations](#)
2. sur 10 epochs : [TestVidéo10 avec annotations](#)
3. sur 15 epochs : [TestVidéo15 avec annotations](#)

Les sources : [6], [3], [5], [7], [1], [4], [2]

References

- [1] Hong-Yuan Mark Liao Alexey Bochkovskiy Chien-Yao Wang. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *Article*. Consulté du 25 Mars au 6 Mai 2021. URL: <https://arxiv.org/abs/2004.10934>.
- [2] Joyce Echessa. “Image Processing in Python with Pillow”. In: *Autho*. Consulté du 25 Mars au 6 Mai 2021. URL: <https://auth0.com/blog/image-processing-in-python-with-pillow/>.
- [3] Santosh Divvala et Ali Farhadi Joseph Redmon Ross Girshick. “You Only Look Once: Unified, Real-Time Object Detection”. In: *Site de Stackoverflow*. Consulté du 25 Mars au 6 Mai 2021, p. 10. URL: <https://pjreddie.com/media/files/papers/yolo.pdf>.
- [4] Ayoosh Kathuria. “What’s new in YOLO v3?” In: *Site Towardsdatascience*. Consulté du 25 Mars au 6 Mai 2021. URL: <https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b>.
- [5] “Overview of model structure about YOLOv5”. In: *Foire aux questions*. Consulté du 25 Mars au 6 Mai 2021. URL: <https://github.com/ultralytics/yolov5/issues/280>.
- [6] “StackOverflow pour les erreurs”. In: *Site de Stackoverflow*. Consulté du 25 Mars au 6 Mai 2021. URL: <https://stackoverflow.com/>.
- [7] “Train Custom Data”. In: *Dépôt github*. Consulté du 25 Mars au 6 Mai 2021. URL: <https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data>.