SY32, Printemps 2022 (P22)
Compte-Rendu du projet 2

Quoc Hung TRAN - Benjamin Vivat Groupe 5

8 mai 2022

Table des matières

1 Introduction

L'objectif de ce projet est d'implémenter un modèle de détection d'objets. L'ensemble d'apprentissage et l'ensemble de test sont divisés en deux phases. La première phase consiste à collecter les données qui vont servir durant la phase d'apprentissage. La deuxième phase du projet sera l'annotation des images de la base d'entraînement en utilisant l'outil **labelImg**.

L'ensemble des images de test et d'apprentissage sont disponibles sur le dépôt Gitlab, dans le dossier **test** et **train**.

Pour évaluer les performances de notre détecteur d'ecocup sur l'ensemble de test, nous l'avons ensuite soumis sur le site suivant : évaluation des résultats

Les résultats de nos détections serons mis dans un fichier csv dans lequel chaque ligne correspond à une boîte de détection décrite par six valeurs : img, i, j, h, l, s, avec :

img	nom de l'image
(i, j)	coordonnées (ligne, colonne) du coin supérieur gauche de la boîte
(h, l)	taille (hauteur, largeur) de la boîte
s	score de détection

L'objectif est de coder puis d'analyser l'efficacité de différentes méthodes de détection d'objets.

2 Détection par fenêtre glissante

À l'origine, nous avions voulu développer un détecteur d'écocups à partir de ce que nous avions vu en TD, à savoir en entraînant un classifieur, filtrant les détections qui se chevauchent, et implémentant un algorithme de fenêtres glissantes. Cependant, les performances admises nous ont portés à nous diriger sur une approche par deep learning, qu'on abordera dans une seconde partie du rapport.

2.1 Génération des images

Avant toute chose, il a fallu qu'on génère des sous-images positives et négatives, à partir des images et annotations fournies. De dimensions fixes, ce sont ces images qui entraîneront notre classifier.

Pour pouvoir faire ça, on devait donc choisir des dimensions pour nos boîtes ; l'idée que nous avons retenue ici a été de combiner toutes les annotations CSV dans un seul fichier, pour pouvoir ensuite le lire et trouver le ratio hauteur/largeur (à plus ou moins 0.05) qui était le plus utilisé pour encadrer les écocups, à savoir 1.8.

Cependant, pour une raison que nous avons pas pu identifier, la fonction *hog* de *skimage* nous renvoyait une erreur, que nous n'avions pas avec des fenêtres carrées. Nous avons donc finalement utilisé une taille de fenêtre de 16*16 pixels.

Pour les images positives, nous avons donc généré 1065 images (721 du dossier *train* et 344 de *train_p21*). Concernant les images négatives, nous avons décidé de générer aléatoirement 10 images de dimensions désirées pour chaque image négative qui nous était fournie, pour un total de 4240 images négatives.

2.2 Entraînement du classifieur

Maintenant que nous avons les images, il nous faut entraîner un classifieur pour qu'il puisse reconnaître les écocups. Pour des besoins de tests, on a réparti nos images de la manière suivante :

— Les images du dossier *train* pour entraîner notre classifieur,
— Les images du dossier *train_p21* pour tester l'efficacité de celui-ci.

Ainsi, pour chaque image, on va créer une liste de *features*, en se basant sur leur HOG (Histogramme des Gradients Orientés), ce qui améliore grandement l'efficacité de notre classifieur, que ce soit en temps ou en résultats obtenus. De plus, l'utilisation d'HOG nous affranchit de plusieurs facteurs, tels que les couleurs des écocups, rendant notre classifieur plus robuste à certaines différences imprévues que l'on pourrait rencontrer sur l'ensemble de test.

Plusieurs classifieurs ont ainsi été testés (adaBoost, random forest, k-neighbors, ...) mais celui qui nous a donné les meilleurs résultats a été SVC, avec une erreur se situant autour de 5%.

2.3 Implémentation de la fenêtre glissante

Notre classifieur étant opérationnel, on s'est ensuite attaqué au problème du parcours de l'image. Pour y répondre, on a implémenté un algorithme de fenêtre glissante.

Ainsi, pour un pas donné (ici 5 pixels), on parcourt l'image avec une fenêtre de 16*16, qu'on découpe, en demandant pour chacune d'entre elles à notre classifieur si il y détecte un écocup.

Une fois toute l'image parcourue, on la downscale d'un certain facteur, de façon à pouvoir détecter un écocup peu importe sa taille, sans avoir à augmenter notre taille de fenêtre. On répète cette opération en augmentant notre facteur de réduction de 0.05 en 0.05, jusqu'à arriver en dessous de 5% de la taille originale de l'image.

2.4 Filtrage des résultats

Le résultat des étapes précédentes nous donne toutes les boîtes de détection dans lesquelles notre classifieur pense avoir trouvé un écocup, avec un score de confiance associé.

La dernière étape, des plus importantes, consiste à trier les résultats obtenus, pour éviter au maximum les détections multiples d'un même écocup. Pour réaliser ce tri on va, pour chaque image, comparer toutes les combinaisons de 2 détections, et, pour chacune de ces combinaisons pour lesquelles l'IoU (Intersection Over Union) est supérieure à 0.5, ne garder que celle au score le plus élevé.

2.5 Résultats

Les résultats de cette méthode ne furent pas concluant. En effet, en plus d'être excessivement longue, on a pu s'apercevoir que le nombre de faux positifs était très important.

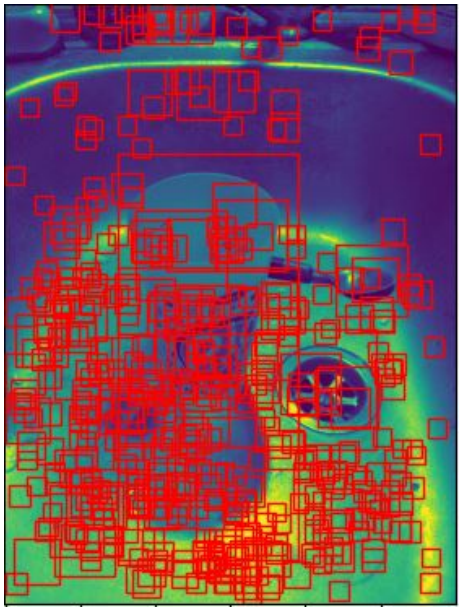

FIGURE 1 – Exemple de détection via fenêtre glissante

Sur une simple image avec un seul écocup bien en évidence, par exemple, on obtenait un taux de précision de 0.002 et un taux f1 de 0.004. Le rappel, lui, était de 100%, assez logiquement au vu du nombre de boîtes présentes.

2.6 Améliorations

Au vu du résultat, nous n'avons pas poussé cette méthode plus loin. Cependant, nous avons pensé à plusieurs pistes d'amélioration :

- Effectuer une passe de notre algorithme de détection à fenêtre glissante sur toutes les images négatives, et nourrir notre classifieur avec tous les faux positifs trouvés,
- Augmenter significativement la taille de nos ensembles de données, positives comme négatives,
- Réussir à utiliser un ratio de fenêtre proche de celui de la majorité des vérités,
- Explorer d'autres classifieurs, et d'autres caractéristiques que HOG.

3 Mask Region-based Convolutional Neural Network

Les résultats de notre algorithme précédent étant peu concluants, nous avons décidé de nous intéresser à des techniques plus efficaces en s'appuyant sur du deep learning ; ce sera l'objet de la suite de ce rapport.

Mask R-CNN est un réseau de neurones convolutif (CNN) et à la pointe de la technologie en termes de segmentation d'images. Cette variante d'un Deep Neural Network détecte les objets dans une image et génère un masque de segmentation de haute qualité pour chaque instance.

Dans cette partie, on va donner un aperçu simple et de haut niveau de Mask R-CNN. Ensuite, nous aborderons les concepts de base nécessaires pour comprendre ce qu'est Mask R-CNN et l'implémentation sur le jeu des données.

3.1 Réseau de neurones convolutifs (CNN)

3.1.1 Introduction

Un réseau neuronal convolutif (CNN) est un type de réseau neuronal artificiel utilisé dans la reconnaissance et le traitement d'images qui est optimisé pour traiter les données de pixels. Par conséquent, les réseaux de neurones convolutifs sont les éléments de base fondamentaux de la tâche de vision par ordinateur de segmentation d'images (segmentation CNN). L'architecture de réseau neuronal convolutif se compose de trois couches principales :

Couche convolutive (convolution layer) : cette couche permet d'abstraire l'image d'entrée sous forme de carte de caractéristiques via l'utilisation de filtres et de noyaux.

Couche de regroupement (Pooling layer) : cette couche aide à sous-échantillonner les cartes d'entités en résumant la présence d'entités dans des parcelles de la carte d'entités.

Couche entièrement connectée (Fully connected layer) : Les couches entièrement connectées connectent chaque neurone d'une couche à chaque neurone d'une autre couche.

La combinaison des couches d'un CNN permet au réseau de neurones conçu d'apprendre à identifier et à reconnaître l'objet d'intérêt dans une image. Les réseaux de neurones convolutifs simples sont conçus pour la classification d'images et la détection d'objets avec un seul objet dans l'image.

3.1.2 La fonctionnalité du CNN

À l'image d'entrée, nous appliquons différents filtres ou détecteurs d'entités pour produire des cartes d'entités. Les filtres ou les détecteurs de caractéristiques sont spatialement petits par rapport à l'image d'entrée. Ces filtres s'étendent sur toute la profondeur de l'image d'entrée.

Plusieurs convolutions sont effectuées en parallèle en appliquant la fonction non linéaire ReLU à la couche convolutive.

Le détecteur de caractéristiques multiples identifie différentes choses comme la détection des bords, différentes formes, courbures ou différentes couleurs, etc.

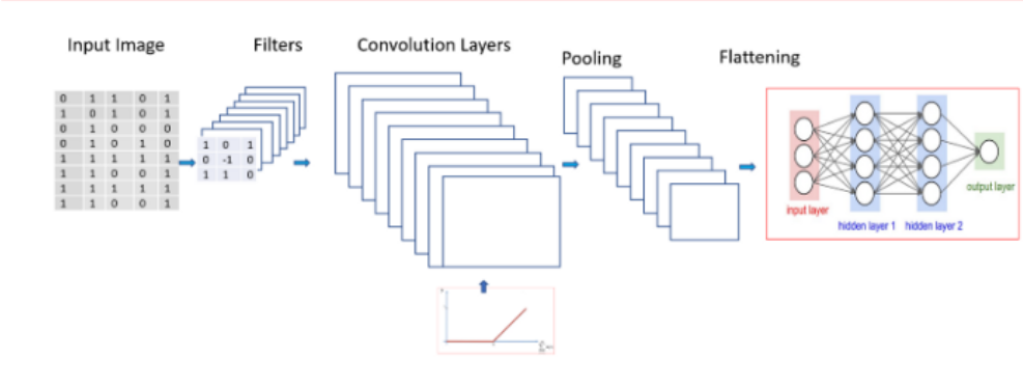

FIGURE 2 – Concept de l'architecture CNN

Nous appliquons le Pooling à la couche convolutive. Nous pouvons appliquer Min Pooling, Max Pooling ou Average Pooling. La fonction de mise en commun maximale offre de meilleures performances par rapport à la mise en commun minimale ou moyenne.

La mise en commun aide à **l'invariance translationnelle**. L'invariance translationnelle signifie que lorsque nous modifions légèrement l'entrée, les sorties regroupées ne changent pas. L'invariance de l'image implique que même lorsqu'une image est tournée, dimensionnée différemment ou vue sous un autre éclairage, un objet sera reconnu comme le même objet.

Dans l'étape suivante, nous aplatissons la couche regroupée pour l'entrer dans un réseau de neurones entièrement connecté (FC). Cependant, CNN est principalement utilisé pour la détection d'un seul objet par image en utilisant des boîtes englobantes, mais ne fonctionne pas bien lorsque plusieurs objets se trouvent dans le champ visuel en raison d'interférences.

Dans une situation plus complexe avec plusieurs objets dans une image, une architecture CNN simple n'est pas optimale. Pour ces situations, Mask R-CNN est une architecture de pointe, basée sur R-CNN (également appelé RCNN).

3.2 R-CNN

Tout d'arbord, on va comprendre le fonctionnement du R-CNN. R-CNN ou RCNN, signifie Region-Based Convolutional Neural Network, c'est un type de modèle d'apprentissage automatique qui est utilisé pour les tâches de vision par ordinateur, en particulier pour la détection d'objets.

3.2.1 Recherche sélective

R-CNN part du principe qu'un seul objet d'intérêt dominera dans une région donnée. R-CNN utilise un des algorithmes de proposition de région (recherche sélective). Ces méthodes prennent une image en entrée et retournent toutes les boîtes correspondantes aux zones qui sont les plus susceptibles d'être des objets. Ces propositions de région peuvent être bruyantes, se chevaucher et ne pas contenir parfaitement l'objet, mais parmi ces propositions de région, il y aura une proposition qui sera très proche de l'objet réel dans l'image. Nous pouvons ensuite classer ces propositions à l'aide du modèle de reconnaissance d'objets. Les propositions de région avec les scores de probabilité élevés sont les emplacements de l'objet. Des algorithmes de proposition de région identifient des objets potentiels dans une image à l'aide de la segmentation. Dans la segmentation, nous regroupons les régions adjacentes qui sont similaires les unes aux autres en fonction de certains critères tels que la couleur, la texture, etc.

Le principe de l'algorithme de **Recherche sélective** :
- Utilise le regroupement ascendant des régions d'image pour générer une hiérarchie de petites à grandes régions
- L'objectif est de générer un petit ensemble d'emplacements d'objets de haute qualité
- Combine le meilleur des intuitions de la segmentation et de la recherche exhaustive.
- La segmentation d'image exploite la structure de l'image pour générer des emplacements d'objets
- La recherche exhaustive vise à capturer tous les emplacements d'objets possibles.

3.2.2 Propositions de régions

Les propositions de région sont un ensemble de détections candidates disponibles pour le détecteur. CNN exécute les fenêtres glissantes sur toute l'image, mais R-CNN ne sélectionne que quelques fenêtres. R-CNN utilise 2000 régions pour une image.

Comme nous avons mentionné dans la section dessus, les propositions de région utilisent un algorithme appelé algorithme de segmentation qui utilise la recherche sélective.

La détection d'objets dans R-CNN :

— Générer des propositions de régions indépendantes des catégories à l'aide de la recherche sélective pour extraire environ 2 000 propositions de régions déformer chaque proposition.

— Les propositions de régions déformées sont transmises à un vaste réseau neuronal convolutif. CNN agit comme un extracteur de caractéristiques qui extrait un vecteur de caractéristiques de longueur fixe de chaque région. Après avoir traversé le CNN, R-CNN extrait un vecteur de caractéristiques à 4096 dimensions pour chaque proposition de région.

— Appliquer SVM (Support Vector Machine) aux fonctionnalités extraites de CNN, pour classifier la présence de l'objet dans la région. Le régresseur est utilisé pour prédire les quatre valeurs de la boîte englobante.

3.2.3 Non-maximum Suppression

À toutes les régions notées d'une image, une Non-maximum suppression e est appliquée. La suppression Non-Max rejette une région si elle présente un chevauchement d'intersection sur union (IoU) avec une région sélectionnée à score plus élevé supérieur à un seuil appris. IoU calcule l'intersection sur l'union des deux boîtes englobantes, la boîte englobante de la vérité et celle pour la boîte prédite par algorithme

Lorsque IoU est égal à 1, cela implique que les cadres de délimitation prédits et ceux de la vérité se chevauchent parfaitement.

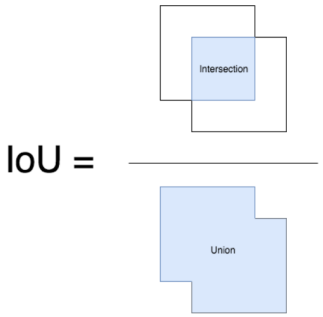

FIGURE 3 – Intersection over Union

3.2.4 L'architecture de R-CNN

Pour comprendre ce qu'est RCNN, nous examinerons son architecture. L'image suivante illustre le concept de CNN régional (R-CNN). Cette approche utilise des cadres de délimitation dans les régions d'objets, qui évaluent ensuite les réseaux convolutifs indépendamment sur toutes les régions d'intérêt (ROI) pour classer plusieurs régions d'image dans la classe proposée.

L'architecture RCNN a été conçue pour résoudre les tâches de détection d'images. De plus, l'architecture R-CNN constitue la base de Mask R-CNN et elle a été améliorée pour devenir ce que nous appelons Faster R-CNN.

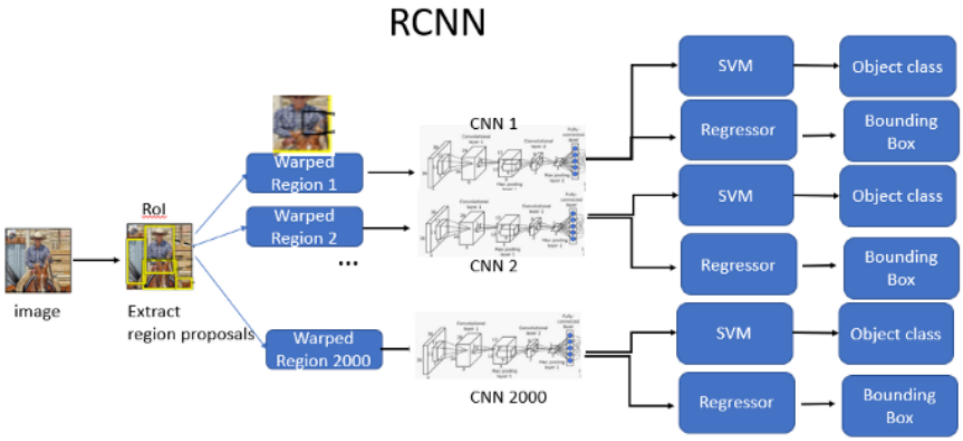

FIGURE 4 – Conception de R-CNN– Region-based Convolutional Networks

3.3 Faster R-CNN

3.3.1 Fast R-CNN

Fast R-CNN fonctionne en extrayant les fonctionnalités à l'aide de RoIPool (Region of Interest Pooling) de chaque boîte candidate et effectue une classification et une régression de la boîte englobante. Il prend l'image et un ensemble de propositions d'objets en entrée. Contrairement à R-CNN, Fast R-CNN utilise un seul Deep ConvNet pour extraire une seule fois les fonctionnalités de l'image entière.

RoIPool est une opération permettant d'extraire une petite carte de caractéristiques de chaque RoI en détection.

On crée également un ensemble de ROI (région d'intérêt) pour l'image en utilisant la recherche sélective. La couche de région d'intérêt (RoI) extrait un **vecteur de caractéristiques** de longueur fixe de la carte de caractéristiques pour chaque proposition d'objet pour la détection d'objet. La couche RoI est un cas particulier de la couche de regroupement de pyramides spatiales avec un seul niveau de pyramide.

Les couches entièrement connectées (FC) nécessitent une entrée de taille fixe. Par conséquent, nous utilisons la couche **ROI Pooling** pour déformer les correctifs des cartes de caractéristiques pour la détection d'objets à une taille fixe.

La couche de pooling RoI est ensuite introduite dans le FC pour la classification ainsi que la localisation, elle utilise le pooling maximum. Il convertit les entités à l'intérieur de toute région d'intérêt valide en une petite carte d'entités.

La couche entièrement connectée se ramifie en deux couches de sortie sœurs

— Une avec des estimations de probabilité softmax sur K classes d'objets (dans ce projet K = 1 équivalent classe "**ecocup**") plus une classe fourre-tout "d'arrière-plan"

— Une autre couche avec un régresseur pour produire quatre nombres à valeur réelle pour une position de boîte englobante raffinée pour chacune des classes d'objets K.

Les principales différences entre R-CNN et Fast R-CNN sont les suivantes :

— Fast R-CNN utilise un seul Deep ConvNet pour les extractions de fonctionnalités. Un seul ConvNet profond accélère considérablement le traitement de l'image contrairement à R-

CNN qui utilise 2000 ConvNets pour chaque région de l'image.

— Fast R-CNN utilise softmax pour la classification des objets au lieu de SVM utilisé dans R-CNN. Softmax surpasse légèrement SVM pour la classification des objets

— Fast R-CNN utilise la perte multi-tâches pour réaliser un entraînement de bout en bout de Deep ConvNets qui augmente la précision de détection.

3.3.2 Implémentation du Faster R-CNN

Le Faster R-CNN est composé de deux étapes :

- La première étape est le réseau profond entièrement convolutif qui propose des régions appelées un réseau de proposition de région (RPN). Le module RPN apporte le concept d'"attention" au réseau unifié.

- La deuxième étape est le détecteur Fast R-CNN qui extrait les caractéristiques à l'aide de RoIPool de chaque boîte candidate et effectue la classification et la régression de la boîte englobante.

La principale différence entre Fast et Faster RCNN est que Fast R-CNN utilise une recherche sélective pour générer des régions d'intérêt, tandis que Faster R-CNN utilise un «réseau de proposition de région» (RPN).

3.4 Mask RCNN

Le masque R-CNN a été construit à partir de Faster R-CNN. Alors que Faster R-CNN a 2 sorties pour chaque objet candidat, une étiquette de classe et un décalage de boîte englobante, Mask R-CNN ajoute une troisième branche qui produit le masque d'objet. La sortie de masque supplémentaire est distincte des sorties de classe et de boîte, nécessitant l'extraction d'une disposition spatiale beaucoup plus fine d'un objet.

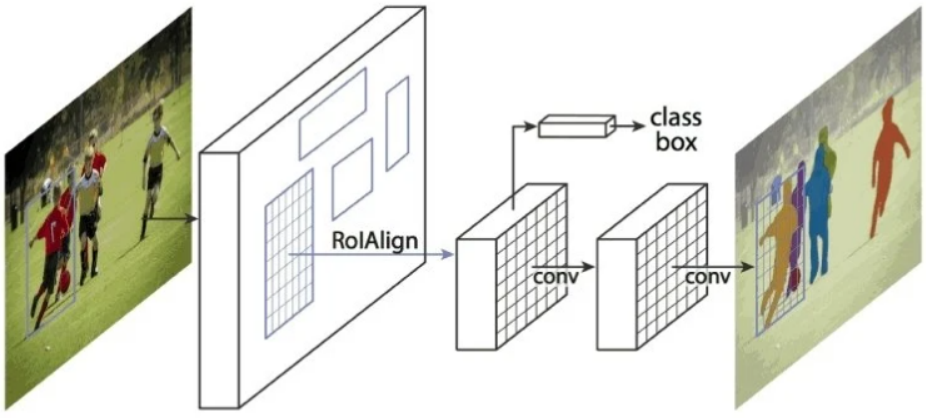

Figure 5 – Conception de Mask R-CNN

3.5 Implémentation

Tout d'abord, on va préparer un jeu de données pour la détection d'objets. nous utiliserons le jeu de données **ecocup**. L'ensemble de données est composé de 884 photographies contenant des

ecocups et de fichiers d'annotation XML (et CSV) qui fournissent informations relatives à la détection des **ecocups** présents dans chaque image, telles que les coordonées de la boîte englobante, ou encore sa taille.

On va développer un objet **Ecocup Dataset** pouvant être utilisé par la bibliothèque Mask R CNN, puis tester l'objet de l'ensemble de données pour confirmer que nous chargeons correctement les images et les annotations. On crée la fonction **extract_boxes()** pour extraire les boîtes englobantes depuis les annotation des fichiers XML.

La bibliothèque mask-rcnn nécessite que les ensembles de données d'entraînement, de validation et de test soient gérés par un objet **mrcnn.utils.Dataset**.

Cela signifie qu'une nouvelle classe doit être définie qui étend la classe **mrcnn.utils.Dataset** et définit une fonction pour charger l'ensemble de données, avec n'importe quel nom comme **load_dataset()**, et remplacer deux fonctions, une pour charger un masque appelé **load_mask ()** et une pour charger une référence d'image (chemin ou URL) appelée **image_reference()**. Un masque est un tableau bidimensionnel ayant les mêmes dimensions que la photographie avec toutes les valeurs nulles là où l'objet n'est pas et toutes les valeurs à 1 là où l'objet est dans la photographie. Nous créons aussi une nouvelle classe appelée **Ecocup Dataset** qui sera utilisée comme suit :

```
train_set = EcoCupDataset()
train_set.load_dataset(dataset_dirs, is_train=True)
train_set.prepare()
```

Ensuite, La fonction de chargement personnalisé, par ex. **load_dataset()** est responsable à la fois de la définition des classes et de la définition des images dans l'ensemble de données. Nous pouvons implémenter une fonction load_dataset() qui prend le chemin vers le répertoire du jeu de données et charge toutes les images du jeu de données.

3.6 Train Mask Modèle R-CNN pour la détection de ecocup

Un modèle Mask R-CNN peut être ajusté à partir de zéro, bien que, comme d'autres applications de vision par ordinateur, le temps puisse être économisé et les performances peuvent être améliorées en utilisant l'apprentissage par transfert.

On a utilisé le pré-formation du modèle Mask R-CNN sur l'ensemble de données de détection d'objets MS COCO peut être utilisé comme point de départ, puis adapté à l'ensemble de données spécifique, dans ce cas, l'ensemble de données **ecocup**, Et puis tester sur l'ensemble du test, On a le résultat suivant :

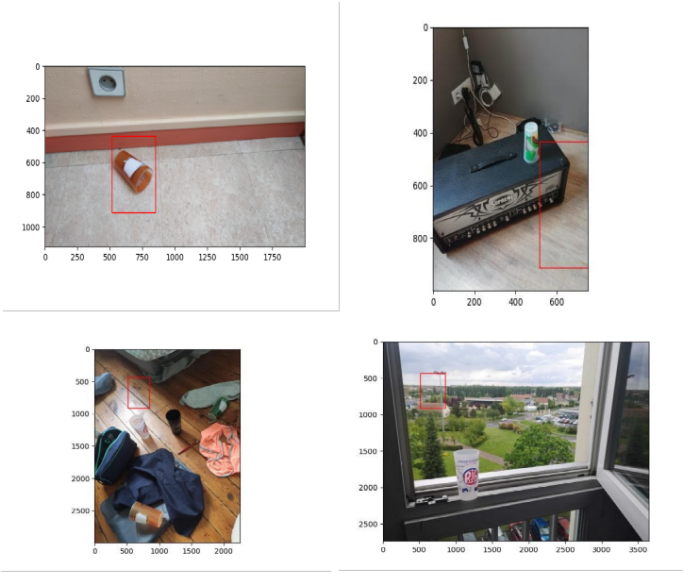

FIGURE 6 – Visualisation des résultats sur l'ensemble de test

L'algorithme trouve beaucoup de False Positive qui montre l'algo qui n'est pas parfait mais cela nous aide beaucoup à comprendre la conception de Convolution Neural Network dans la détection d'objets.

4 YOLO

Comme discuté précédemment, Fast-RCNN, Faster-RCNN, Mask R-CNN décomposent le problème de détection d'objet en deux étapes :

1. Détection de régions d'objets possibles.
2. Classer l'image dans ces régions en classes d'objets.

Au lieu de sélectionner des parties intéressantes d'une image, ils prédisent des classes et des cadres de délimitation pour l'image entière en une seule exécution de l'algorithme. Les deux exemples les plus connus de ce groupe sont les algorithmes de la famille YOLO (You Only Look Once) que nous allons mentionner par la suite.

YOLO est d'une simplicité rafraîchissante : voir la figure suivant.

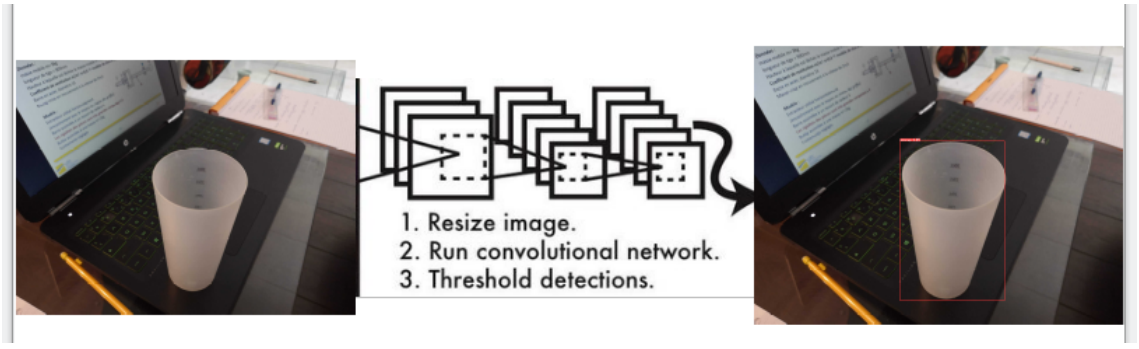

FIGURE 7 – Le système de détection YOLO

Un seul réseau convolutif prédit simultanément plusieurs boîtes englobantes et probabilités de classe pour ces boîtes.

YOLO s'entraîne sur des images complètes et optimise directement les performances de détection. Ce modèle unifié présente plusieurs avantages par rapport aux méthodes traditionnelles de détection d'objets

4.1 La fonctionalité de YOLO

Les composants de la détection d'objets sont dans un seul réseau neuronal. Le réseau utilise les caractéristiques de l'image entière pour prédire chaque boîte englobante. Il prédit également toutes les boîtes englobantes dans toutes les classes pour une image simultanément. Cela signifie que notre réseau raisonne globalement sur l'image complète et tous les objets de l'image.

Le système divise l'image d'entrée en une grille **S X S**. Si le centre d'un objet tombe dans une cellule de la grille, cette cellule de la grille est responsable de la détection de cet objet. Chaque cellule de la grille prédit B boîtes englobantes et les scores de confiance pour ces boîtes. Ces scores de confiance reflètent la confiance du modèle dans le fait que la boîte contient un objet et également la précision avec laquelle il pense que la boîte prédit. Formellement, nous définissons la confiance comme $Pr(Objet) \times IOU_{pred}^{vrai}$.

Chaque cellule de la grille prédit également les probabilités de classe conditionnelle C, $Pr(Class_i|Objet)$ dans ce projet cela est équivalent à $Pr(0|Objet)$, les probabilités sont conditionnées sur la cellule de grille contenant un ecocup. Nous ne prédisons qu'un seul ensemble de probabilités de classe par cellule de grille, quel que soit le nombre de cases B. Au moment du test, on multiplie les probabilités de classe conditionnelles et les prédictions de confiance des cases individuelles, $Pr(Class_i|Objet) \times Pr(Object) \times IOU_{pred}^{vrai} = Pr(Class_i) \times IOU_{pred}^{vrai}$ qui nous donne des scores de confiance spécifiques à la classe pour chaque case. Ces scores encodent à la fois la probabilité que cette classe apparaisse dans la boîte et la façon dont la boîte prédite correspond à l'objet.

Si aucun objet n'existe dans cette cellule, les scores de confiance doivent être nuls. Sinon, nous voulons que le score de confiance soit égal à l'intersection sur l'union (IOU) entre la boîte prédite et la boîte vérité.

Chaque boîte englobante se compose de 5 prédictions : b_x, b_y, b_w, b_h et le score de détection. Les coordonnées (b_x, b_y) représentent le centre de la boîte par rapport aux limites de la cellule de la grille. La largeur et la hauteur sont prédites par rapport à l'ensemble de l'image.

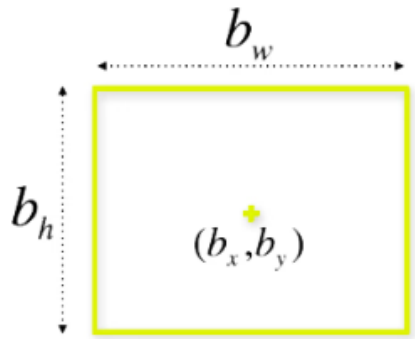

FIGURE 8 – Coordonnées d'une boîte englobante

YOLO renvoi aussi un score d'objectivité lors de la détection pour chaque cadre. Ce score indique à quel point le modèle est certain que l'objet souhaité est présent dans cette boîte englobante.

Comme mentionné dessus, L'algorithme NMS va donc sélectionner le cadre avec le score d'objectivité le plus élevé, ensuite il retirera tous les autres cadres qui ont un chevauchement important avec le cadre choisi.

4.2 Implémentation

Voici une chronologie présentant le développement de YOLO au cours des dernières années :

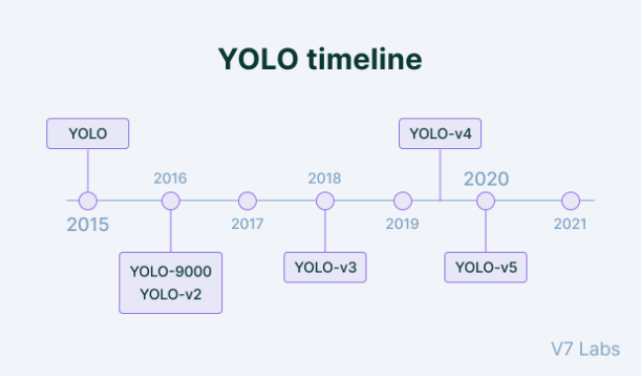

FIGURE 9 – YOLO Time Line

Dans ce sujet, nous avons choisi d'utiliser la cinquième version de YOLO. La cinquième itération de l'algorithme de détection d'objets le plus populaire a été publiée peu de temps après YOLOv4, mais cette fois par Glenn Jocher.

Tout d'abord, YOLO v5 attend des annotations pour chaque image sous la forme d'un fichier .txt où chaque ligne du fichier texte décrit une boîte englobante. On a implémenté la méthode **process_data** qui convertit des annotation depuis csv au format txt. Par exemble, il y a 1 objets au total (1 ecocup) alors chaque ligne représente un de ces objets. Les spécifications pour chaque ligne sont les suivantes :

```
0 0.571429 0.484127 0.075397 0.195767
```

FIGURE 10 – Coordonnées des boîtes englobantes

Une ligne par objet, chaque ligne est de classe x_cente,r y_center, largeur, hauteur format. Les coordonnées de la boîte doivent être normalisées par les dimensions de l'image (c'est-à-dire avoir des valeurs comprises entre 0 et 1. Les numéros de classe sont indexés à zéro.

Ensuite, nous partitionnons l'ensemble de données dans un ensemble d'entraînement et de validation, qui contiennent respectivement 80% et 10% des données. Après ça, nous formons le réseau. Nous utilisons divers drapeaux pour définir des options concernant l'entraînement du réseau. Ici, Nous définissons l'emplacement de train, val, le nombre de classes (1) et les noms des classes ("ecocup"). Étant donné que l'ensemble de données est petit et que nous n'avons pas beaucoup d'objets par image, nous commençons avec le plus petit des modèles pré-formés **yolo5s** pour garder les choses simples et éviter l'overfiting. Nous gardons une taille de lot de 8, une taille d'image de 1024 et nous nous entraînons pour 100 époques.

```
os.system("python3 yolov5/train.py --img 1024 --batch 8 --epochs 100 --data ecocup.
    yaml --weights yolov5s.pt --cache")
```

On a essayé à modifier les hyper-paramètres pour trouver le meilleur modèle. Le plus important est la taille du lot (batch_size), qui affecte certains indicateurs tels que le temps de formation global, le temps de formation par époque, la qualité du modèle, etc. Habituellement, nous choisissons la taille du lot comme une puissance de deux, comprise entre 16 et 512. Mais généralement, la taille de 16 est une règle empirique et un bon choix dans ce cas là.

4.3 Résultat

best.pt contient les poids du modèle le plus performant enregistré pendant l'entraînement.

Voici quelques visualisations des résultats de l'exécution du modèle sur l'ensemble de test.

FIGURE 11 – Visualisations des résultats sur l'ensemble des test

Pour évaluer le modèle, on utilise une suite de mesures de précision qui dépendent du seuil d'IoU et des performances du modèle. Les mesures de précision sont les suivantes :

— Précision— La précision est le rapport entre le nombre de vrais positifs et le nombre total de prédictions positives.

— Rappel—Rappel est le rapport entre le nombre de vrais positifs et le nombre total d'objets à détecter.

— Score F1 : le score F1 est une moyenne pondérée de la précision et du rappel. Les valeurs vont de 0 à 1, où 1 signifie la précision la plus élevée.

— Area Under Curve (AUC) représente le degré ou la mesure de séparabilité. Un modèle avec une AUC plus élevée est plus efficace pour prédire les vrais positifs et les vrais négatifs.

En vérifiant certains des hyper-paramètres, le première résultat est obtenu depuis le jeu de données principal avec une taille d'image d'entrée de 1028, une taille du lot de 16, et un nombre d'époques d'entraînement de 500.

Précision	Rappel	F1	AUC
74.10 (84.89)	75.74 (69.82)	74.91 (76.62)	72.39 (68.90)

On peut considérer que la précision et le rappel sont encore faible. Le modèle **Yolov5s** est un modèle complexe, donc de la sur-détection se produit également lorsque le modèle essaie de faire des prédictions sur des données très bruitées, ce qui est dû à un modèle trop complexe ayant trop de paramètres. Ainsi, pour cette raison, le modèle surajusté est inexact car la tendance ne reflète pas la réalité présente dans les données. Dans le cas des réseaux de neurones, l'augmentation des données signifie simplement augmenter la taille des données, ce qui augmente le nombre d'images présentes dans l'ensemble de données. Lorsque le réseau essaie d'apprendre à partir d'un petit ensemble de données, il aura tendance à avoir un plus grand contrôle sur l'ensemble de données et s'assurera de satisfaire exactement tous les points de données. Ainsi, le réseau essaie de mémoriser

chaque point de données et ne parvient pas à capturer la tendance générale à partir de l'ensemble de données de formation. En conséquence, on a décidé d'ajouter les données du semestre P21 pour varier l'ensemble de données ; on observe une amélioration au niveau de notre résultat.

Précision	Rappel	F1	AUC
78.47 (88.89)	83.09 (75.74)	80.71 (81.79)	79.35 (74.29)

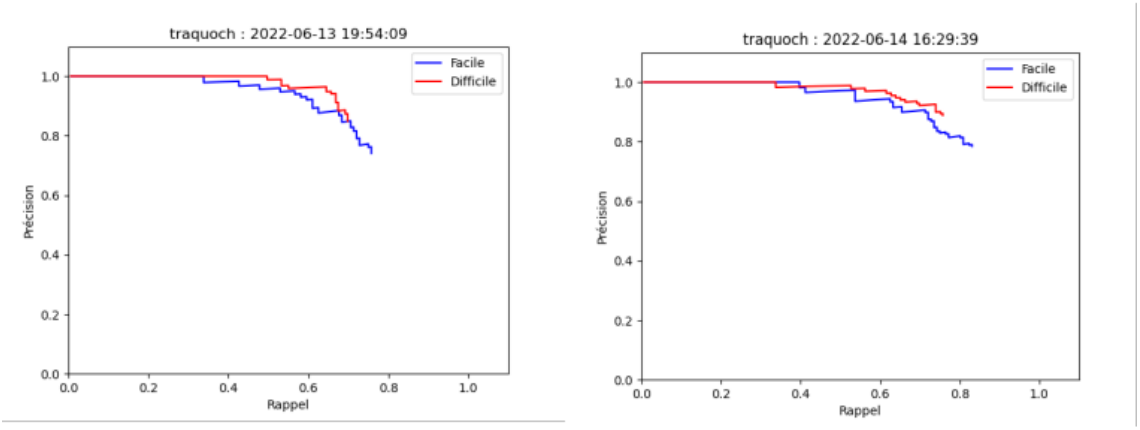

FIGURE 12 – Courbe précision-rappel des résultats sur l'ensemble de test

4.4 Pour aller plus loin

Il existe des versions en développement YOLOv5x et YOLOv5x6, qui donnent de meilleurs résultats dans presque tous les cas, mais qui nécessitent plus de paramètres et donc plus de mémoire pour l'entraînement, et qui prennent beaucoup plus de temps. De plus, comme mentionné ci-dessus, comme le modèle de pré-formation est complexe et conséquent, il a une plus grande capacité de généralisation, et une plus grande capacité d'extraction extensive des caractéristiques. L'augmentation des données d'image est peut-être la technique la plus connue d'augmentation des données. Elle regroupe les techniques utilisées pour augmenter artificiellement la taille d'un groupe de données d'apprentissage en créant des versions modifiées d'images à partir des images d'apprentissage disponibles.

Nous pouvons alors améliorer efficacement le processus d'apprentissage puisqu'on aurait alors plus d'échantillons pour l'entraînement du réseau de neurones. Les techniques d'augmentation peuvent créer des variations d'images qui peuvent améliorer la capacité des modèles d'entraînement pour généraliser ce qu'ils ont appris à de nouvelles images, ce qui améliore fortement la performance du modèle.

5 Conclusion

Ce projet de détecteur d'écocups a été très intéressant, puisqu'il nous aura permis de nous approprier plus concrètement tout ce qu'on a pu voir en cours, que ce soit au niveau du machine learning avec la détection à fenêtre glissante, ou au niveau du deep learning avec YOLO et R-CNN.

On pourra regretter le manque de performance de notre premier algorithme, même si nous ne l'avons pas poussé à son maximum ; cependant, cela a pu nous démontrer la grande séparation qu'il y a entre ces deux méthodes, tant au niveau de la difficulté d'implémentation que de la qualité des résultats observés.

Finalement, bien que très intéressant, ce projet nous a demandé beaucoup de temps (notamment pour faire les deux techniques), et on aurait apprécié pouvoir y passer davantage de temps, pour explorer plus en détail l'optimisation des nombreux paramètres existants et arriver à un résultat encore meilleur.

6 Bibliographie

— Adrian Rosebrock (10 novembre 2014), *Histogram of Oriented Gradients and Object Detection*, https://pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/
— Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick, *Mask R-CNN*, https://arxiv.org/pdf/1703.06870.pdf
— Adrian Rosebrock (12 novembre 2018), *YOLO object detection with OpenCV*,://pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/
— Le cours, bien évidemment :-)