

# Projet détection d'ecocups

SY32 - RAPPORT  
Juin 2022

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Choix du modèle</b>	<b>3</b>
2.1	Modèle . . . . .	3
2.2	Déférence avec le modèle R-CNN présenté en cours . . . . .	4
2.3	CNN utilisé pour la classification de région . . . . .	4
<b>3</b>	<b>Création des données d'entraînement</b>	<b>5</b>
<b>4</b>	<b>Création et entraînement du modèle</b>	<b>7</b>
4.1	Chargement des données d'entraînement et création du modèle . . . . .	7
4.2	Entraînement du modèle et évaluation sur l'ensemble de validation . . . . .	7
<b>5</b>	<b>Visualisation des performances sur les données de validation</b>	<b>10</b>
5.1	Structure de l'algorithme . . . . .	10
5.2	Visualisation des résultats . . . . .	10
<b>6</b>	<b>Résultats sur les données de tests</b>	<b>13</b>
<b>7</b>	<b>Comparaison avec un modèle YOLOv5</b>	<b>14</b>
7.1	Entraînement du modèle YOLOv5 . . . . .	14
7.2	Résultats et comparaison . . . . .	14
7.2.1	Performance de détection . . . . .	14
7.2.2	Performance temporelle . . . . .	15
<b>8</b>	<b>Conclusion et pistes d'améliorations</b>	<b>16</b>

# Chapitre 1

## Introduction

L'objectif de ce projet est d'implémenter un détecteur d'ecocups sur des images. Nous avons à notre disposition un ensemble d'images d'entraînement labellisées et les données de tests sur lesquelles nous devons détecter les ecocups grâce à notre modèle de détection.

Nous avons vu en cours de SY32 plusieurs algorithmes de détection (fenêtres glissantes puis classificateurs tels que les Support Vector Machine ou les Random Forests sur ces fenêtres pour déterminer si la fenêtre contient un ecocup, R-CNN ou encore des modèles de détection plus récents tels que YOLO ou SSD). Pour ce projet, nous allons utiliser un modèle inspiré du R-CNN pour détecter les ecocups.

Dans un premier temps, nous allons expliciter le modèle que nous avons choisi. Nous parlerons ensuite de la création du jeu de données d'entraînement que nous utiliserons pour entraîner notre CNN à classifier une image comme étant une image d'ecocup ou non. Les premiers tests sur un ensemble de validation ainsi que les premiers résultats seront analysés et commentés. Nous parlerons enfin des améliorations que nous avons essayé d'apporter à notre modèle ainsi que des seconds résultats.

# Chapitre 2

## Choix du modèle

### 2.1 Modèle

Nous avons décidé d'implémenter un modèle inspiré du R-CNN (Region - Convolutional Neural Network) pour ce projet. Le modèle complet est le suivant :

Nous prenons en entrée une image de taille variable. Nous appliquons ensuite une *Selective Segmentation* afin de déterminer des régions susceptibles de contenir un objet quelconque. Les régions trouvées sont ensuite envoyées à un CNN dont le but est de classifier la région comme étant une image d'ecocup ou non. Si le CNN classe cette région comme étant un ecocup, nous sauvegardons les coordonnées de la région (ligne, colonne du sommet haut gauche, largeur hauteur de la région) ainsi que le score de détection renvoyé par le CNN (i.e la probabilité que cette région soit un ecocup). Une fois toutes les régions classifiées, nous appliquons la *Non-Max-Suppression* pour ne garder que les régions avec la plus de probabilité de contenir un ecocup. Les sorties de notre modèle sont donc les coordonnées ainsi que le score de détection des régions gardées après application de la *Non-Max-Suppression*. Notre modèle est résumé sur la figure 2.1.

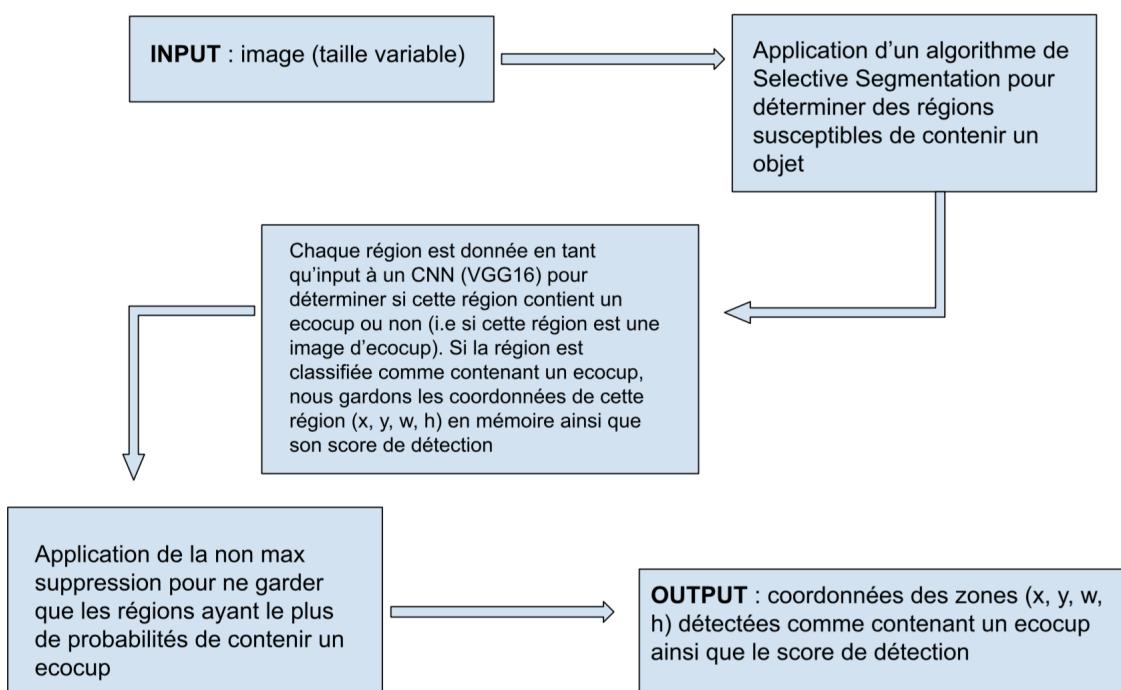


FIGURE 2.1 – Modèle de détection d'ecocup

## 2.2 Différence avec le modèle R-CNN présenté en cours

La différence entre notre modèle et le modèle R-CNN présenté en cours est dans la sortie du réseau de convolution :

Dans le modèle présenté en cours, la sortie du réseau de convolution est un vecteur de 4096 valeurs qui sont ensuite données comme entrée à un *Support Vector Machine* pour classifier si la région contient un ecocup ou pas. Or nous avons choisi comme sortie du réseau de convolution une couche à deux valeurs : une valeur représentant la probabilité que l'image ne soit pas un ecocup et une valeur représentant la probabilité que l'image soit un ecocup. Ce choix est justifié par le fait que notre classification réside en une classification binaire : ecocup ou non ecocup. Nous ne voulons pas avoir plusieurs classes en sortie mais la probabilité que cette région contienne un ecocup ou non.

## 2.3 CNN utilisé pour la classification de région

Nous utilisons le modèle VGG16 présenté en cours comme réseau de convolution. Pour ce projet, nous allons ré-entraîner uniquement la dernière couche *Dense* (ou *Fully Connected*) de ce modèle. Nous modifions également la couche de sortie en la remplaçant par une couche à 2 valeurs. La fonction d'activation que nous utilisons est la même que celle du modèle VGG16 de base, à savoir la fonction softmax, qui permet de sommer à 1 toutes les valeurs de sortie. L'architecture du modèle VGG16 est présentée en figure 2.2.

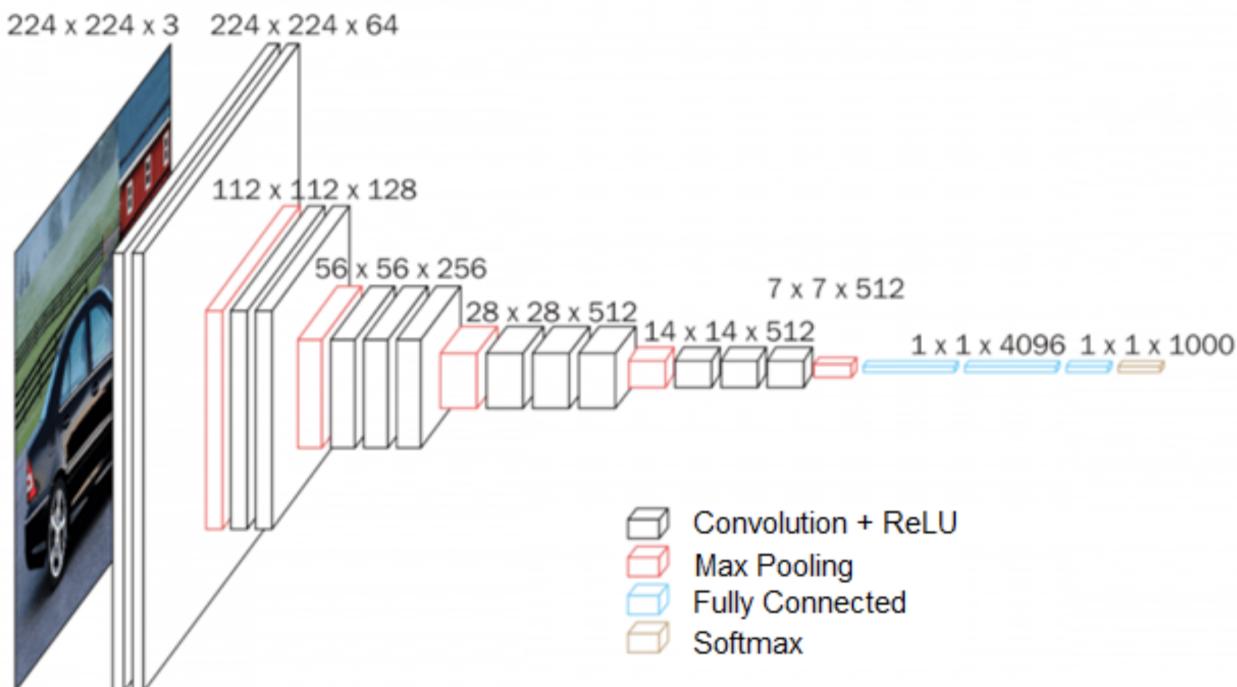


FIGURE 2.2 – Architecture du classifieur VGG16

Il existe d'autres modèles de type VGG comme VGG19 qui comporte 19 couches de convolutions. Cependant l'architecture VGG16 est plus répandue car plus rapide à entraîner aux vue de son nombre de paramètres inférieur au modèle VGG19.

# Chapitre 3

## Création des données d'entraînement

Un jeu de données avec des images comportant des ecocups (images positives, labellisées) et des images sans ecocup (images négatives) à été créé de manière collaborative par l'ensemble des étudiants participant au projet. Ce jeu de données à ensuite été décomposé en un ensemble d'entraînement et un ensemble de test.

Pour créer les régions qui serviront à l'entraînement du réseau convolutionnel, nous utilisons l'algorithme suivant :

*Pour chaque image dans le jeu de données d'entraînement créé précédemment :*  
*création des régions d'intérêt à l'aide de la SelectiveSearchSegmentation de OpenCV*  
*si l'image est une image positive (contenant au moins un ecocup) :*  
    *on charge les labels indiquant l'emplacement des ecocups dans l'image*  
    *pour chaque région trouvée par la SelectiveSearchSegmentation :*  
        *si la valeur de IoU entre la région et les labels chargés est supérieur à 0.7 :*  
            *on ajoute l'image dans un ensemble d'entraînement*  
            *on ajoute le label "1" dans l'ensemble de labels correspondant aux images*  
        *sinon :*  
            *on ajoute l'image dans un ensemble d'entraînement*  
            *on ajoute le label "0" dans l'ensemble de labels correspondant aux images*  
*si l'image est négative (ne contenant pas d'ecocup) :*  
    *on ajoute l'image dans un ensemble d'entraînement*  
    *on ajoute le label "0" dans l'ensemble de labels correspondant aux images*

Pour éviter d'avoir un nombre très déséquilibré de régions négatives, nous utilisons un indicateur d'ajout à l'ensemble d'entraînement. Cet indicateur (0 pour non ajout, 1 pour ajout) est généré selon une loi binomiale de paramètre  $p=0,007$  (ce paramètre a été choisi arbitrairement). De plus, nous avons fait le choix de ne considérer que les régions dont la hauteur et la largeur est au moins égale à 10% de la hauteur et la largeur de l'image. Ce choix est justifié par le manque de ressources et de temps pour générer un ensemble d'entraînement contenant de trop petites régions. Une fois l'ensemble d'entraînement créé ainsi que l'ensemble des labels correspondant, nous enregistrons ces deux ensembles dans deux fichiers distincts afin de pouvoir les réutiliser. L'ensemble de cet algorithme est contenu dans le fichier `create_data.py` dans le dossier `python_code`. Il suffit de lancer ce code python en ligne de commande pour générer les ensembles d'entraînement et l'ensemble des labels correspondant.

Cependant, en visualisant l'intégralité des données d'entraînement ainsi que leur labellisation, nous remarquons que certaines données sont mal labellisées. Hors nous savons que la qualité des données d'entraînement joue un rôle crucial dans les résultats du modèle. Au total,

nous re-labellisons une quarantaine d'images (Figure 3.1).

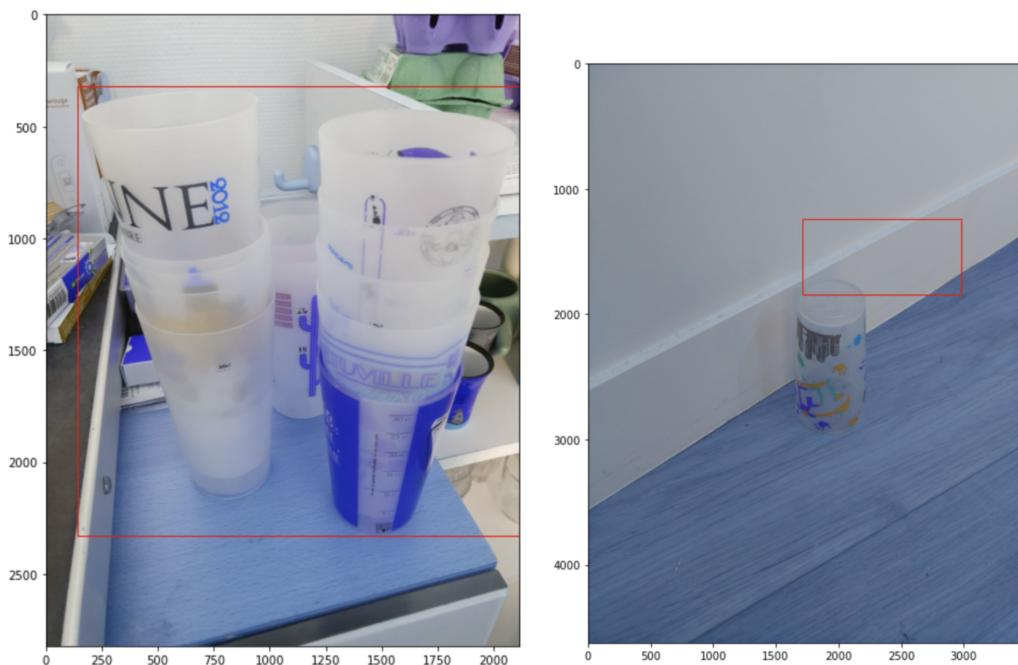


FIGURE 3.1 – Exemple de mauvaise labellisation trouvée dans le jeu de données

Pour enrichir notre jeu de données, nous ajoutons les images créées pour le même projet lors du semestre P21 à nos données d'entraînement. Comme pour le jeu de données de base, nous analysons les labellisations et les ajustons lorsqu'elles sont fausses. De cette manière nous avons enrichi et nettoyé notre jeu de données. NB : Néanmoins nous n'avons pas pu entraîner notre réseau convolutionnel sur cet ensemble de données enrichi et nettoyé par manque de temps étant donné la puissance de nos ordinateurs.

# Chapitre 4

## Création et entraînement du modèle

### 4.1 Chargement des données d’entraînement et création du modèle

Comme énoncé précédemment, notre réseau de convolution utilisé pour classifier si une région contient un ecocup ou non est le modèle VGG16. Nous appliquons la méthode de *Transfert Learning* afin d’adapter ce modèle à notre modèle. Le *Transfert Learning* consiste à utiliser un modèle déjà entraîné (dans notre cas VGG16 entraîné sur *ImageNet*) puis de bloquer les poids sur les premières couches du modèle, ce qui signifie que nous n’allons pas modifier les poids des couches bloquées. Nous allons entraîné notre modèle sur les dernières couches afin d’adapter le modèle entraîné à notre problème. Nous décidons de bloquer toutes les couches de convolution et de modifier uniquement les poids des couches denses (*fully connected* sur le schéma). Nous n’entraînons que ces couches car dans les classes de base de VGG16, nous retrouvons les classes “cup” et “measuring cup”. Comme un ecocup est semblable en apparence à ces deux objets, mais présente quand même certaines distinctions, nous n’estimions pas nécessaire de ré-entraîner les dernières couches de convolution pour l’extraction de formes précises.

Afin d’entraîner notre modèle nous chargeons les données que nous avons créées et sauvegardées dans la section précédente, puis nous les séparons en deux ensembles : un ensemble d’entraînement et un ensemble de validation. Cette séparation nous permet d’estimer les performances de notre réseau de convolution à différentier un ecocup d’un autre objet. L’ensemble d’entraînement est composé de 14109 images (de dimensions 224x224x3) et l’ensemble de validation est composé de 1558 images de mêmes dimensions.

### 4.2 Entraînement du modèle et évaluation sur l’ensemble de validation

Nous entraînons notre modèle en utilisant les 14109 images. Nous décidons d’utiliser *Adam* comme optimiseur ainsi que la *Binary Cross entropy* comme *loss function*. Nous utilisons cette loss function car notre problème est un problème de classification binaire. Le taux d’apprentissage pour notre descente de gradient est de 0,0001 (cette valeur a été choisie de manière arbitraire, nous avons essayé un taux d’apprentissage de 0,001 et les résultats étaient moins bons). Pour l’entraînement de notre modèle nous utilisons 50 *epochs* (ce nombre nous permet d’augmenter les performances de notre modèle, mais il peut aussi entraîner un sur-apprentissage) avec 10 étapes par *epoch*, ainsi qu’un *batch\_size* de 28 (28 images seront utilisées avant d’appliquer l’algorithme de descente de gradient). Nous obtenons après entraînement ces courbes de *loss value* et d’*accuracy score* :

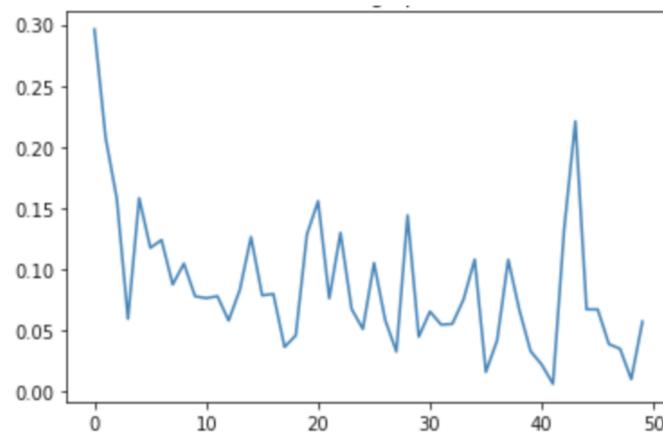


FIGURE 4.1 – Courbe de loss value (en abscisse le nombre d'epochs et en ordonnée la loss value)

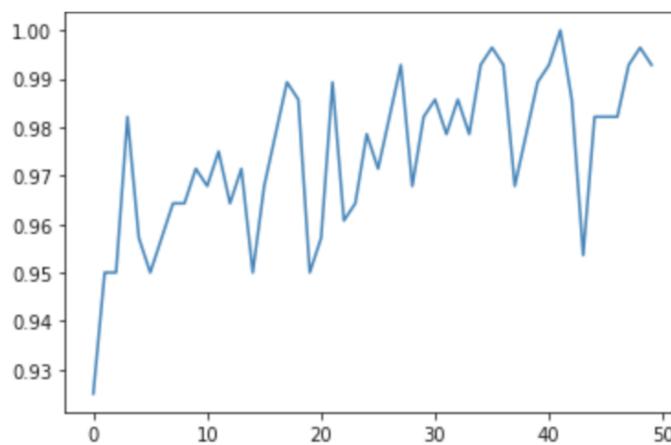


FIGURE 4.2 – Courbe de l'accuracy score (en abscisse le nombre d'epochs et en ordonnée l'accuracy score)

Nous remarquons que malgré un certain bruit, notre *loss value* semble diminuer et notre *accuracy score* semble augmenter au fur et à mesure des itérations d'entraînement du modèle.

Après entraînement, nous décidons de tester les capacités de notre modèle à déterminer si une région est un ecocup ou non sur des nouvelles données en utilisant notre ensemble de validation. Nous obtenons la matrice de confusion ainsi suivante :

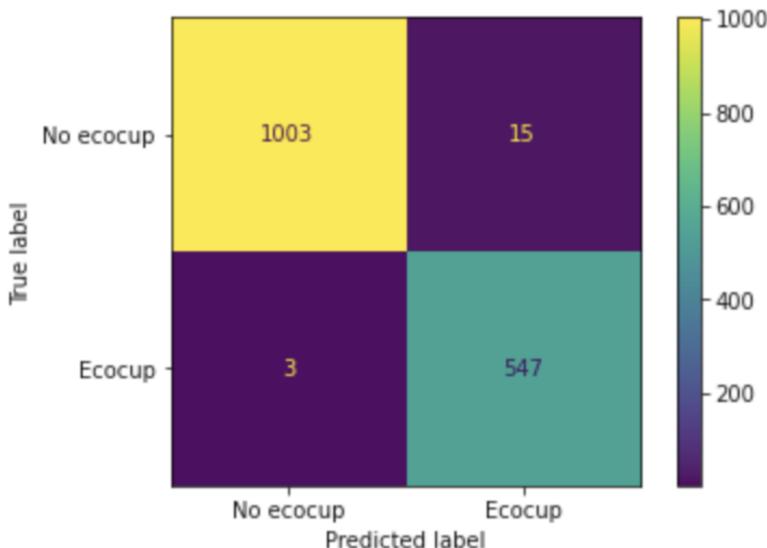


FIGURE 4.3 – Matrice de confusion présentant les classifications de notre modèle sur l'ensemble de validation

Nous remarquons que notre modèle semble être très performant dans la classification d'ecocup ou non. Même si cette matrice de confusion semble montrer que notre modèle est performant, nous n'avons pas regardé en détail toutes les images de notre ensemble d'entraînement. Nous n'avons donc pas d'informations sur la capacité de notre modèle à différencier un verre d'un ecocup par exemple.

Une fois notre modèle entraîné, nous sauvegardons l'ensemble de ses poids afin de pouvoir réutiliser notre modèle pour détecter les ecocups sur d'autres images.

L'ensemble de cette section est contenue dans les fichiers *vgecup.py* (pour la déclaration et la compilation de notre réseau de convolution) et *train\_vgecup.py* (pour l'entraînement du modèle). Le fichier *vgecup.py* ne contient qu'une fonction permettant de récupérer le modèle (si une modification du modèle doit être faite, c'est dans ce fichier qu'il faut intervenir), tandis que le fichier *train\_vgecup.py* pour l'entraînement du modèle peut être lancé en ligne de commande.

# Chapitre 5

## Visualisation des performances sur les données de validation

### 5.1 Structure de l'algorithme

Nous pouvons maintenant tester notre algorithme de détection d'ecocup sur nos premières données de validation. Les résultats de cette partie peuvent être visualisés dans le fichier *visualize\_vgecup\_results.ipynb*.

La structure de notre algorithme, semblable à la structure du R-CNN, est la suivante, pour chaque image :

- Extraction de régions d'intérêts à l'aide de la *SelectiveSearchSegmentation*
- Pour les 3000 régions présentant le plus d'intérêt (c'est à dire ayant le plus de probabilité de contenir un objet quelconque), si les dimensions de cette région sont au moins égales à 10% des dimensions de l'image (pour éviter que des trop petites régions soient considérées), alors nous classifions à l'aide de notre réseau convolutionnel si cette région est un ecocup ou non. Si la région est classifiée comme ecocup, nous sauvegardons ces coordonnées ainsi que le score de détection (la probabilité retournée par notre classifieur).
- Une fois toutes les régions classifiées et les coordonnées des régions classifiées comme ecocup enregistrées, nous appliquons l'algorithme de *Non-Max-Suppression* pour ne garder hypothétiquement qu'une seule région détectée comme ecocup (celle présentant le plus de probabilité d'être un ecocup dont le recouvrement avec une autre région est inférieur à un seuil fixé)
- Une fois la *Non-Max-Suppression* appliquée, nous enregistrons les coordonnées des régions restantes comme les coordonnées des ecocups détectés

Nous avons choisi le nombre de 3000 régions pour éviter de devoir classifier trop de régions. Enfin, nous choisissons comme seuil de recouvrement (la valeur de l'IoU) la valeur de 0,1.

### 5.2 Visualisation des résultats

Nous avons fait tourner notre algorithme de détection sur nos données de validation. Les résultats obtenus confirment notre hypothèse :

Notre classifieur n'est pas aussi performant qu'espéré (lorsqu'on regarde la matrice de confusion). Un exemple est montré sur l'image suivante : notre classifieur classe bien l'ecocup comme un ecocup, mais il classe un objet de forme similaire à un ecocup comme un ecocup également.

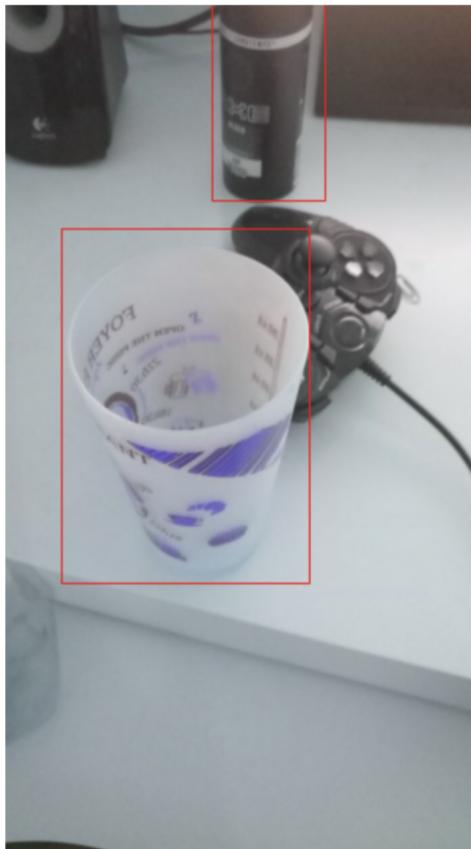


FIGURE 5.1 – Exemple de l'hypothèse de manque de performance de notre classifieur à distinguer un ecocup d'un objet ayant une forme similaire

Nous avons essayé d'implémenter une fonction permettant d'obtenir les métriques d'évaluation du projet (précision, rappel et F1 score) ainsi qu'une matrice de confusion pour visualiser nos résultats sur les données de validation. La matrice de confusion que nous obtenons est la suivante :

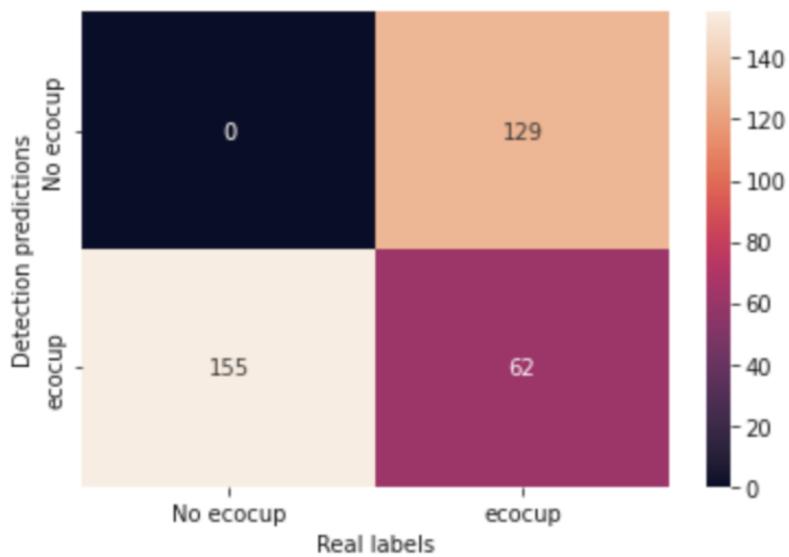


FIGURE 5.2 – Matrice de confusion obtenue sur les données de validation

Nous pouvons ainsi déterminer que la précision de notre modèle sur ces données est de 28,6% et que le rappel est de 32,5%. Ces métriques montrent que notre algorithme n'est pas très

performant dans la détection d'ecocup. Il est possible d'augmenter ces précisions en entraînant plus notre réseau convolutionnel pour la classification, en utilisant des techniques de data augmentation par exemple, ou bien en augmentant notre jeu de données d'entraînement avec d'autres images d'ecocups ou d'objets autres que des ecocups provenant d'une distribution différente (images venant d'internet par exemple). Nous pourrions aussi utiliser notre jeu de données nettoyé et augmenté pour améliorer les performances.

# Chapitre 6

## Résultats sur les données de tests

Nous utilisons notre algorithme sur les données de tests (issues d'une autre distribution et jamais vu par notre algorithme ni pendant l'entraînement ni pendant la validation). Nous obtenons les résultats suivants :

précision ↴	rappel ↴	F1 ↴	auc ↴
44.25	56.62	49.68	39.82
(46.55)	(47.93)	(47.23)	(34.22)

FIGURE 6.1 – Résultats de notre algorithme sur les données de test

La précision de notre modèle est de 44,25%. La précision étant la capacité d'un modèle à minimiser les faux positifs, nous pouvons conclure que si notre modèle détecte un ecocup, il y a 44,25% de chance que cette détection soit réellement un ecocup. Le rappel est une mesure d'indication de la capacité de notre modèle à détecter les positifs (ce qui signifie détecter tous les ecocups), nous remarquons que notre modèle réussi à détecter plus de la moitié de tous les ecocups présents dans les données de test. Enfin, le F1 score (établi comme la moyenne harmonique de la précision et du rappel), donne une indication de notre modèle à bien détecter les positifs (détecter tous les positifs et ne pas détecter de faux positifs). Notre F1 score est de 49,68%, ce qui montre bien que notre modèle de détection d'ecocup n'est pas aussi performant que nous pouvions l'espérer.

Bien que ces résultats ne soient pas excellents, ils sont meilleurs que les résultats que nous avons obtenus sur nos données de validation.

# Chapitre 7

## Comparaison avec un modèle YOLOv5

### 7.1 Entraînement du modèle YOLOv5

Dans le but de comparer les performances de notre modèle nous avons également entraîné un modèle YOLOv5 sur le même jeu de données. Nous savons que ce modèle est très performant, il est très populaire pour réaliser de la détection sur des images. Ce modèle est une amélioration du modèle de base YOLO (You Only Look Once).

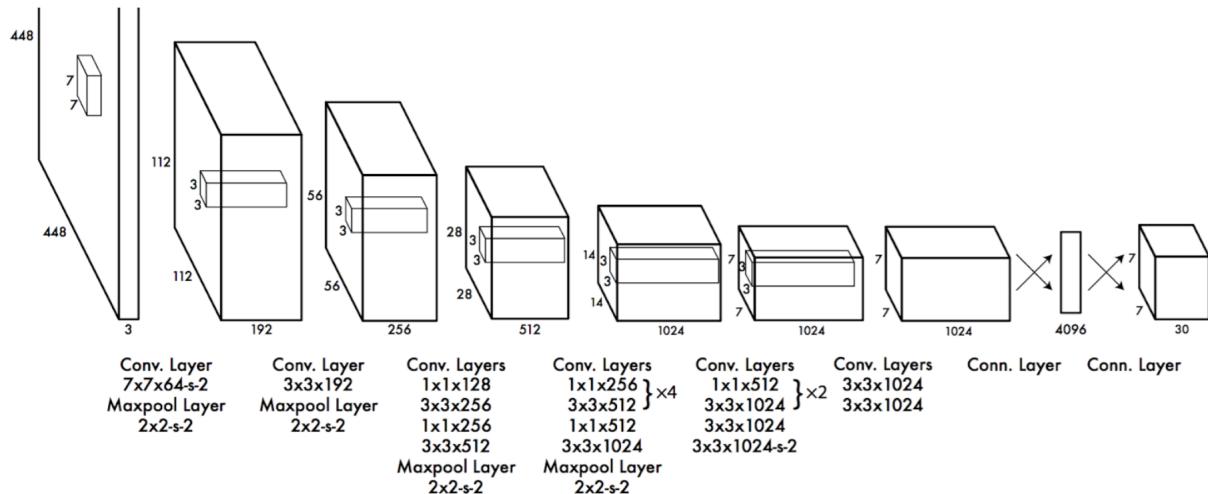


FIGURE 7.1 – Architecture du modèle YOLO

Etant donné que notre étude n'est pas centrée sur le modèle YOLOv5, nous n'allons pas détailler les différences qu'il existe entre YOLOv5 et YOLO dans ce rapport.

Comme pour le VGG16, nous avons réalisé du *transfer learning*. Nous avons utilisé le modèle YOLOv5 entraîné sur le jeu de données COCO et avons ré-entraîné les dernières couches sur nos images d'ecocups. L'entraînement était paramétré avec 100 epochs et un *batch size* de 32 et à duré 32 heures.

### 7.2 Résultats et comparaison

#### 7.2.1 Performance de détection

Nous obtenons de très bonnes mesures de performances pour la détection avec le modèle YOLOv5. D'après nos observations, il est plus difficile pour le modèle de détecter des ecocup

opaques et colorées. Cela est sûrement dû à un manque d'images de ce type d'ecocup dans notre jeu de données d'entraînement.

<b>précision 1l</b>	<b>rappel 1l</b>	<b>F1 1l</b>	<b>auc 1l</b>
<b>75.82 (86.93)</b>	<b>85.29 (78.70)</b>	<b>80.28 (82.61)</b>	<b>79.08 (76.56)</b>

FIGURE 7.2 – Performances obtenues avec YOLOv5

### 7.2.2 Performance temporelle

En moyenne, notre modèle YOLOv5 met 0.2 secondes à traiter une image. Le temps de traitement d'une image par notre modèle principal (Segmentation + CNN) est d'environ 2 minutes. YOLOv5 est donc quasiment 600 fois plus rapide que notre R-CNN. Si nous devions traiter les images en flux continu (détection sur vidéo), nous opterions donc pour un modèle YOLO.

Ces résultats sont cohérents étant donné l'utilisation quasiment systématique de l'utilisation d'une architecture YOLO pour un problème de détection sur image aujourd'hui.

# Chapitre 8

## Conclusion et pistes d'améliorations

Lors de ce projet nous avons pu mettre en œuvre un algorithme de détection sur image étudié en cours de SY32. Nous avons pu découvrir de nouvelles techniques d'apprentissage modernes comme le *transfer learning*. Notre modèle est fonctionnel, cependant les résultats en terme de précision et de vitesse d'exécution ne sont pas aussi bon qu'un modèle YOLOv5. Le but de l'étude étant de mettre en œuvre une méthode quelconque en l'appréhendant et non d'obtenir l'algorithme le plus performant, ces résultats ne sont pas problématiques.

Comme nous l'avons dit précédemment, nous avons été limités par le temps et la puissance de nos ordinateurs. Nous avons cependant des pistes d'améliorations. Dans un premier temps, nous pourrions améliorer les performances de notre modèle en le ré-entraînant sur le jeu de données nettoyé et enrichi. Nous avons aussi remarqué un grand nombre de classification d'objet comme ecocup alors qu'ils n'en étaient pas (Faux-Positifs). Ajouter davantage de régions négatives aux données d'entraînements pourrait résoudre ce problème.

Aussi, nous savons que les performances temporelles de notre R-CNN sont plutôt mauvaises. Nous pourrions alors nous pencher sur les améliorations de ce modèle (Fast R-CNN, Faster R-CNN).