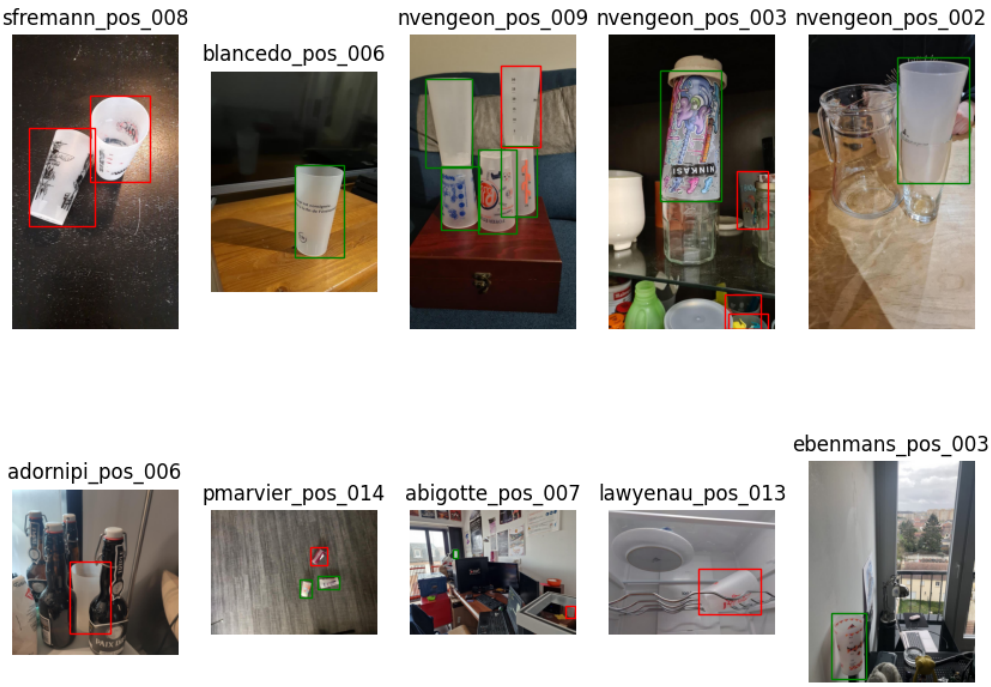
Sy32 P23
Projet ecocup
16/04/2023

Adrien Bigotte, Constant Roussel

April 30, 2023



Contents

1 Présentation du projet

Le projet ecocup est un projet de reconnaissance objet à titre didactique. Il s'agit de détecter dans une image, grâce à un apprentissage automatique préalable, la présence et la localisation d'une ou plusieurs ecocup. Le choix a été fait d'appréhender ce problème de reconnaissance objet, en plusieurs problèmes de classification.

1.1 Étapes du projet

On a découpé le projet en deux. D'abord un apprentissage sur des images positives c'est-à-dire sur des images dans lequel il y a une ecocup avec un ratio entre 1.5 et 2 et sur les images négatives générées aléatoirement. La seconde étape est le test sur les images complètes qui seront subdivisées par une fenêtre glissante en petites images pour avoir des images avec un englobement parfait des ecocup. Ensuite on récupère les faux positifs issus de ce test et on les ajoute comme image négative à notre ensemble d'apprentissage et on refait un apprentissage. On refait ce "boosting" un nombre certain de fois jusqu'à avoir un nombre faible et satisfaisant de faux positif

2 Premier apprentissage

2.1 Génération des données d'apprentissages

Pour générer les données d'apprentissages pour le classifieur binaire il a fallu utiliser les labels des données "train/pos", récupérer les coordonnées des boîtes englobantes, et faire un découpage de l'image à la taille et la position des ecocup. Pour cela on joint les noms des images avec celui des labels. On récupère ainsi la taille, la position, du découpage qui doit être effectué. Également on en profite pour ne se focaliser que sur les exemples faciles et sur les formait de boîte englobante ayant un ratio entre 1.5 et 2.
Pour faire un apprentissage efficace il faut que les images soient toutes de la même taille, or le ratio étant diffèrent pour chaque image il a fallu ajouter une marge autour des labels avant le découpage pour laisser une marge pour pouvoir les redimensionner sans couper une partie de l'ecocup. Ainsi on peut voir dans le fichier operation_sur_images/resize.py et la fonction crop_and_save_images() l'implémentation de ce découpage. Ensuite on redécoupe toutes les images pour avoir le même ratio de 1.5 avec la fonction crop_to_ratio(). Finalement avec la fonction operation_sur_images/resize_images() on standardise la taille des images en 160*240.
Pour les images négatives il suffit de faire une découpe aléatoire dans les images négative avec un ratio de 1.5 et avec une taille de 160*240.
Enfin pour augmenter notre jeu de données on fait un "flip" de toutes les images grâce à la fonction flip_images()

2.2 Formats des vecteurs descripteurs

Chaque image d'entrée possède une taille de 160*240. Pour réduire le nombre de données dans le vecteur descripteur, nous avons choisi l'algorithme HOG. Nous avions aussi testé la détection de contours, mais le score était moins bon en validation croisée.

Précisément, nous utilisons un algorithme HOG qui utilise des cellules carrées de 16 pixels de côté, avec une cellule par bloc, et 8 orientations.

2.3 Comparaison des classifieurs

Pour le classifieur, nous nous somme concentré que sur le classifieur AdaBoost qui utilise comme estimateur l'arbre de décision. Il nous fallait alors choisir le bon nombre d'estimateurs, adapté à notre problème.
Pour ce faire, nous avons fait une validation croisée pour chaque nombre d'estimateurs den 1 à 100. On obtient alors la courbe suivante.

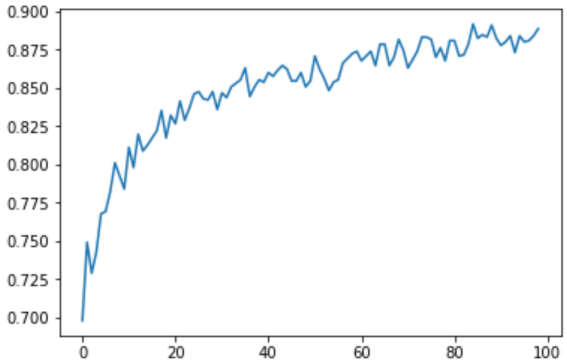


Figure 1: Courbe de précision selon le nombre d'estimateurs

On peut constater une convergence vers environ 90% à partir de 100 estimateurs. C'est donc le nombre d'estimateurs que nous choisissons.
Il est intéressant de noter que notre classifieur, une fois entraîné, est capable de correctement classifier toutes ses données d'entraînement, mais nous n'observons pas de décroissance notable dans la précision de celui-ci. Cela signifie que nous n'avons pas assez d'entrées pour avoir de l'overfitting, mais aussi que le choix du type de classifieur n'est pas si important (tous les classifieurs qui peuvent parfaitement classifier leurs données d'entraînement arriveraient au même résultat) ; c'est le vecteur descripteur qui importe le plus.

En anticipation de l'augmentation du nombre de données pendant la seconde phase d'entrainement, nous avons mis 400 estimateurs.

3 Deuxième apprentissage

3.1 Fenêtre glissantes

Pour implémenter une fenêtre glissante il a fallu transformer toutes les images en entrée en plusieurs petites images sans passer à côté des ecocups. La difficulté ici est de trouver où s'arrêter dans la fréquence du découpage. Pour écrire cette fonction il a été décider de ne pas utiliser de fenêtres à dimensions variables mais plutôt d'utiliser plusieurs tailles d'une même image pour faire varier cette fenêtre. Le redimensionnement est ainsi implémenté dans operation_sur_images/fenetre_glissantes.py /resize_images(). Pour ce qui est de la découpe des images par fenêtres glissantes, il se fait incrémentalement avec un chevauchement qu'on peut définir au vu de la qualité de l'engobage des ecocup.
Pour retrouver les coordonnées du découpage dans l'image originale, on transmet le facteur d'échelle utilisé dans le nom de l'image lors du redimensionnement, puis lors du découpage par les fenêtres glissantes on divise les coordonnées par ce facteur d'échelle pour avoir la taille (h,l) et les coordonnées (i,j).

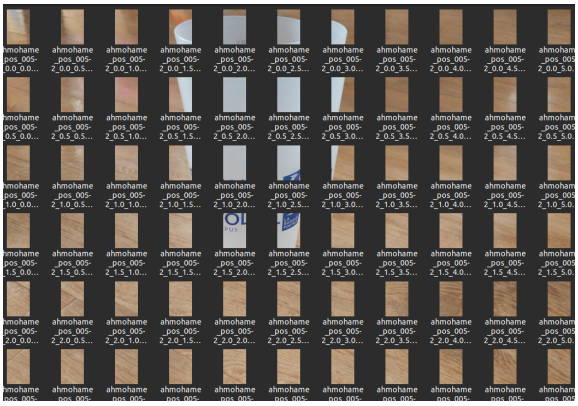


Figure 2: fenêtre glissantes trop petites

3.2 Ensemble d'apprentissage

Pour le second apprentissage, nous utilisons les fenêtres glissantes sur l'ensemble d'entraînement positif. Par essais erreur et par observation des images en sortie il on a fini par trouver que cinq redimensionnements des images étaient suffisants pour englober les ecocup (0.5, 0.25, 0.75, 1, 1.5) avec un recouvrement de 0.5. Le choix de ces facteurs d'échelles s'est fait par observation des images de sortie des fenêtres glissantes. En effet il y a des images qui sont prises de très loin, d'autres de très près ce qui rend difficile l'englobement.

3.3 Processus d'apprentissage

Pour le second apprentissage, nous entraînons un classifieur sur les données d'entraînement. Nous faisons une prédiction pour chacune des images générées par les fenêtres glissantes, et nous calculons le taux de faux positifs (une prédiction est considérée comme étant faux positif si elle a été prédite positive par le classifieur mais qu'il n'y a **strictement** aucune écocup dans l'image, donc une intersection nulle entre les rectangles). Ensuite, nous ajoutons 10% des faux positifs aux données d'apprentissage négatifs, et nous répétons cette opération jusqu'à obtenir moins de 10% de faux positifs.
Pour choisir ces faux positifs, nous trions les prédictions par ordre décroissant de score de prédiction (ce qui nous donne la courbe suivante)

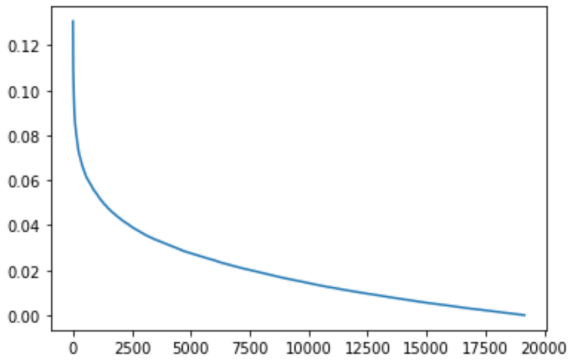


Figure 3: Score des prédictions dans l'ordre décroissant

Nous isolons ensuite les faux positifs et nous prenons le premier dixième de cet ensemble. Cela permet d'isoler les prédictions où le classifieur s'est le plus trompé (là où il était le plus sûr de lui, mais a fait une mauvaise prédiction)

4 Prédiction des données de test

4.1 Génération images de test

Pour le data-set de test, au vu du faible nombre d'images et de l'exigence de on a décidé d'augmenter le nombre de fenêtres glissantes par six redimensionnements (0.5, 0.25, 0.75, 1, 1.5,2) et un chevauchement de 0.75.

4.2 Ajustement du seuil de score

Pour réaliser des prédictions les plus justes possibles, nous avons besoin d'établir le score de détection à partir duquel on prend en compte le rectangle. Pour cela, nous avons réalisé un petit script qui permet de visualiser les fenêtres classifiées positives lors du second entrainement, ainsi que leur score.

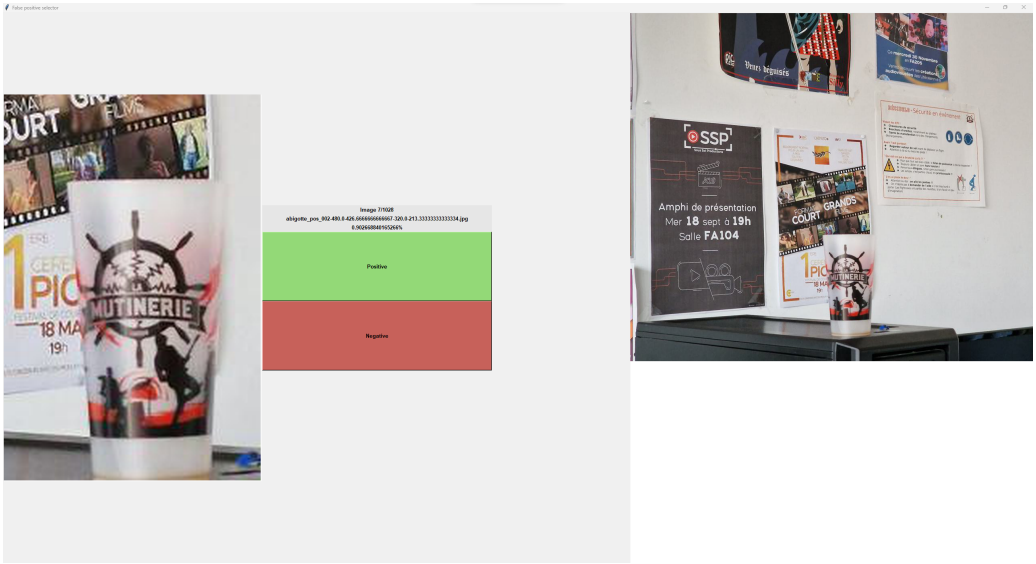


Figure 4: Programme de visualisation des positifs

À l'origine, ce programme permettait d'effectuer le second entraînement manuellement avant d'automatiser le processus (quand on clique sur "Negative", il ajoute l'image à la base d'entrainement).
Par la suite, il nous a permis d'observer le score de détection pour chacune des images. Cela nous a permi d'estimer un premier seuil à environ 0.005. Nous avons également testé avec des seuils de 0.008 et 0.0065, et ce dernier seuil donne le meilleur score F1.

4.3 Évaluation, précision/rappel

La soumission de nos prédictions sur robotics.gi.utc/SY32/projet/ nous donne la courbe de précision/rappel ci-dessous. On peut voir que notre implementation a eu tendance à classer un grand nombre d'échantillons négatifs comme positifs. Cependant la courbe augmente de nouveau progressivement qui est caractéristique du classifieur Adaboost.

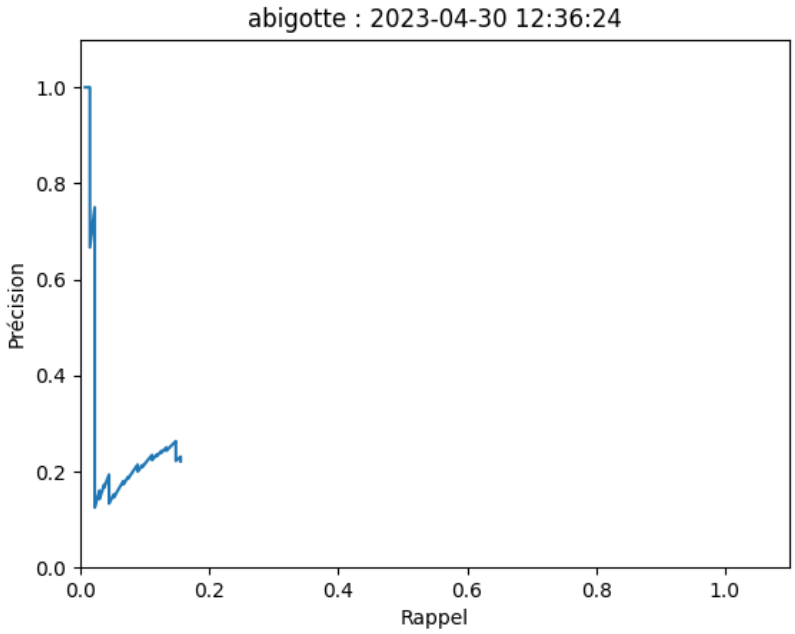


Figure 5: Courbe précision/rappel avec seuil 0.008

4.4 Perceptives d'amélioration

Pour améliorer notre classification il faudrait en outre plus de données, car la base d'apprentissage est faible. Malgré un flip des images cela reste pas suffisant. Également ont un ajustement plus minutieux des paramètres pourrait aider mais de nombreux tests sont à faire, et ceux-ci sont très longs. En particulier, il serait intéressant de calibrer plus finement le nombre d'estimateurs, le seuil de score de détection et le pourcentage de faux négatifs ajoutés au dataset.