
UE 19IA-C Applications de l'IA

Analyse de Vidéos

Boris Mansencal

Année 2021-2022

Albane Arthuis

Paul Lorgue

Emma Navarro

SOMMAIRE

Sommaire	2
Introduction	3
Données	3
Architecture	4
Classifieur d'images	6
Récupération des données	6
Modèle de classification	6
Tracking	8
Incremental learning	10
Conclusion	10

INTRODUCTION

Ce projet consiste en la construction d'un modèle capable d'aider à la détection d'un objet d'intérêt sur une vidéo. L'application directe permettrait à des personnes amputées des membres supérieurs de diriger des prothèses grâce à des neuroprothèses. Si la direction du regard exprime l'intention et permet de savoir quel objet l'utilisateur veut saisir, alors le modèle doit être capable de suivre cet objet et son emplacement. Pour répondre à cette problématique nous avons dans un premier temps entraîné un modèle capable de reconnaître, de classer plusieurs objets d'une cuisine (un bol, du riz, etc). Dans un second temps nous avons procédé au suivi de cet objet appelé "tracking", c'est-à-dire qu'à partir de la position d'un objet à un temps donné le modèle doit prédire où se trouve l'objet d'intérêt au temps d'après. Ici nous entraînons notre modèle sur des vidéos, les instants sont alors des *frames*.

DONNÉES

Le *dataset* utilisé pour ce projet est le *dataset Grasping in the Wild*¹. Il s'agit d'un *dataset* de vidéos ayant lieu dans des cuisines dans lesquelles une personne munie de *eye tracker* attrape un objet. Le point de vue des vidéos correspond bien à notre projet visant à simuler un point de vue de vision humaine. Les objets attrapés peuvent appartenir à seize classes différentes : *Bowl, Can of Coca Cola, Frying Pan, Glass, Jam, Lid, Milk Bottle, Mug, Oil Bottle, Plate, Rice, Sauce Pan, Sponge, Sugar, Vinegar Bottle, Wash Liquid*. Ainsi, le *dataset* permet d'entraîner un classifieur capable de reconnaître quel est l'élément attrapé par la personne dans telle ou telle vidéo. Pour notre projet, nous utilisons *Grasping in the Wild light* : cette version ne contient que cinq classes : *Bowl, Sugar, Rice, Milk Bottle* et *Can of Coca Cola*.

Les données fournies permettent également de savoir où est réellement localisé l'objet d'intérêt sur chaque *frame* et de connaître le numéro de la première *frame* où il apparaît. De fait, nous avons trois dossiers comprenant : les vidéos, les documents

¹ <https://www.labri.fr/projet/AIV/graspinginthewild.php>

textes indiquant la position des objets(*frame* et repère orthonormé sur celle-ci) et la découpe des boîtes définies dans ce document permettant de produire un grand nombre d'instances utiles à l'entraînement du classifieur.

Par exemple, pour la vidéo d'entraînement SugarPlace1Subject1, le document texte de la Figure 1.a indique que le paquet de sucre apparaît à la *frame* n°39 et qu'il est localisé dans une box caractérisée par l'ordonnée et l'abscisse de son point supérieur gauche, ainsi que par sa largeur et sa hauteur. Ici le sucre a un point supérieur gauche de coordonnées (996, 346), une largeur de 136 et une hauteur de 184. Ces coordonnées permettent de récupérer une boîte englobante qui encadre l'objet d'intérêt permettant de récupérer une image d'entraînement telle que celle sur la Figure 1.b.

35	0				
36	0				
37	0				
38	0				
39	1	996	346	136	184
40	1	986	346	136	184
41	1	972	344	136	184
42	1	966	350	136	184
43	1	950	354	136	184
44	1	936	356	136	184
45	1	922	356	136	184

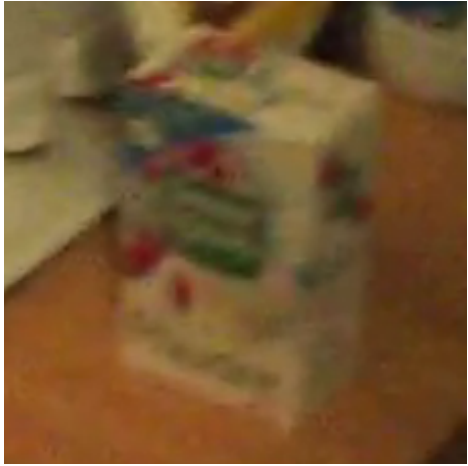


Figure 1.a : Document texte correspondant à la vidéo SugarPlace1Subject1

Figure 1.b : Image correspondant aux données de la boîte englobante fournies par le document texte

ARCHITECTURE

Notre code est organisé tel que :

- dossier data : contient les types de données explicités dans la partie précédente
 - dossier DB

- dossier test : images des objets d'intérêts servant au test du modèle
 - dossier Bowl
 - fichier BowlPlace1Subject4_2_bboxes_12_0.png
 - ...
 - dossier CanOfCocaCola
 - dossier MilkBottle
 - dossier Rice
 - dossier Sugar
- dossier train : images des objets d'intérêts servant à l'entraînement du modèle
 - dossier Bowl
 - dossier CanOfCocaCola
 - dossier MilkBottle
 - dossier Rice
 - dossier Sugar
- dossier validation : images des objets d'intérêts servant au test du modèle, dossier construit par nos soins à partir des images de train
 - dossier Bowl
 - dossier CanOfCocaCola
 - dossier MilkBottle
 - dossier Rice
 - dossier Sugar
- dossier GT
 - fichier BowlPlace1Subject1_2_bboxes.txt
 - ...
- dossier VIDEOS
 - fichier BowlPlace1Subject1.mp4
 - ...
- dossier models : là où sont enregistrés les modèles entraînés et récupérés

- vgg16_a_9.h5
- ...
- dossier utils : ce dossier contient des fonctions développées afin de parvenir à nos fins dans les notebooks
 - dossier helpers : les fichiers de ce dossier contiennent les fonctions et constantes que nous avons rédigé et qui sont utiles aux notebooks
 - fichier constants.py
 - fichier data.py
 - fichier plot.py
 - fichier video.py
 - fichier extractPatches.py
 - fichier make_database.py
 - fichier viewBBoxes.py
- fichier classification.ipynb
- fichier incremental_learning.ipynb
- fichier tracking.ipynb

Les dossiers data et models ne sont pas partagés sur le Git. C'est à chaque utilisateur de les compléter en récupérant les données à l'adresse :

https://dept-info.labri.fr/~mansenca/GITW_light/ . Attention toutefois, le dossier data>DB>validation est créé dans le notebook classification.ipynb. D'autres modèles peuvent également être produits à partir de ce notebook.

CLASSIFIEUR D'IMAGES

Le but d'un classifieur est de reconnaître l'élément présent sur une image dans un nombre limités de labels. Pour ce faire, le modèle prend en entrée une image et prédit une distribution de probabilité associant à chaque classe la probabilité que l'image la représente. Le pourcentage le plus élevé de ce vecteur ainsi obtenu détermine le label attribué à l'image. Si celui-ci correspond à l'objet présent réellement alors la prédiction du modèle est bonne, sinon elle est fausse.

Notre classifieur est entraîné dans le notebook `classification.ipynb`. Nous suivrons ici les étapes de ce notebook et procéderons successivement à leur explication si cela est nécessaire.

RÉCUPÉRATION DES DONNÉES

Cette partie permet la création d'un jeu de validation à partir du *dataset* *Grasping in the Wild* téléchargé au préalable. Ces cases ne doivent donc être exécutées qu'à une seule reprise.

Cette étape permet également de charger les données dans des variables et de les visualiser.

MODÈLE DE CLASSIFICATION

C'est ici que sont entraînés les différents classifieurs pour le projet. Etant donné qu'un CNN demande énormément de ressources à entraîner et ne devrait pas atteindre des scores très élevés, nous avons priorisé le *fine-tuning* d'un modèle préentraîné. C'est-à-dire que nous avons repris un modèle de classification d'images entraîné sur un autre *dataset* que *GITW* que nous avons modifié puis réentraîné. Nous avons gardé et figé les poids des couches de convolutions (pour l'extraction des features) auxquelles nous avons ajouté de nouvelles couches de neurones (pour la classification). Nous avons repris un modèle VGG16 préentraîné sur ImageNet², un *dataset* général entraîné sur plus de 1000 classes. Nous utilisons l'architecture suivante pour la classification des features en sortie des couches de convolution:

- une couche Flatten,
- une couche Dense (512) avec fonction d'activation ReLu
- une couche Dropout (0,5)
- une couche Dense(256) avec fonction d'activation ReLu
- une couche Dense(5) avec fonction d'activation Softmax

Nous avons 5 neurones sur la dernière couche car nous avons 5 classes.

² <https://www.image-net.org/index.php>

Après 11 époques notre modèle a cessé de s'améliorer de façon significative :

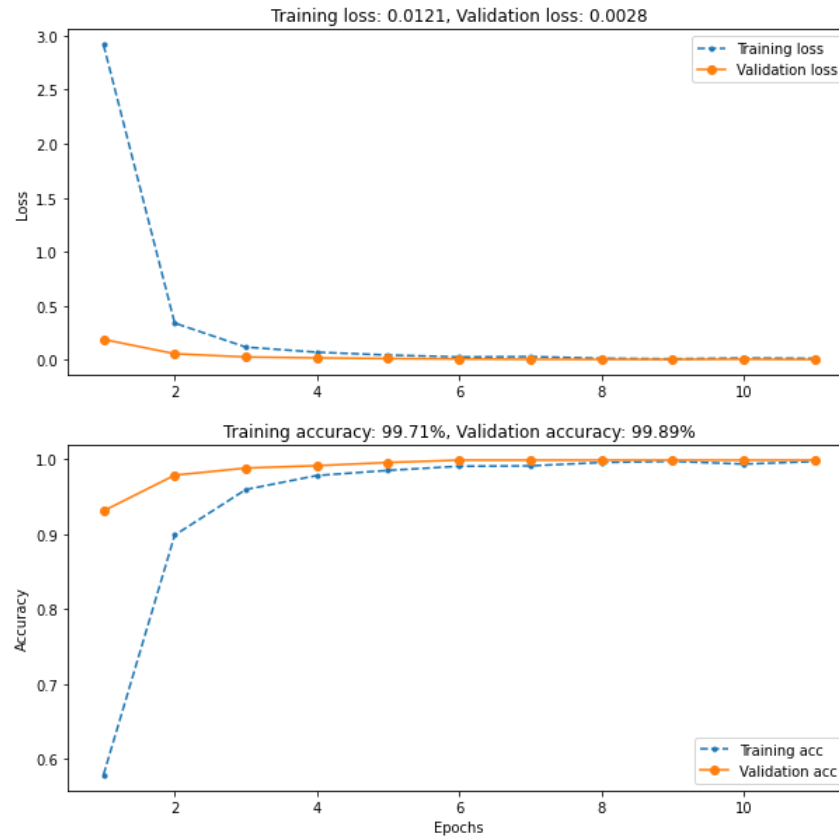


Figure 2 : Entraînement du classifieur à partir de VGG16 après *fine-tuning*

Nous obtenons de très bons résultats : la précision (*accuracy*) atteint 99,71% sur le jeu d'entraînement et 99,89% sur le jeu de validation. De plus, nous n'observons pas de sur-apprentissage, en effet les performances ne sont pas meilleures sur le jeu d'entraînement que sur le jeu de validation.

Pour confirmer ces performances nous testons notre modèle sur le jeu de données de test. Le *f1_score* (macro average) moyen est de 90%, ce qui reste très honorable. La matrice de confusion observable en Figure 3 nous permet de constater que si le modèle est performant il semble meilleur sur la prédiction de certaines classes : s'il parvient 98% du temps à prédire qu'un paquet de riz est bien du riz, il avance souvent qu'un paquet de sucre est un paquet de riz 13,5% du temps. Néanmoins, le modèle reste exploitable pour la suite de nos opérations.

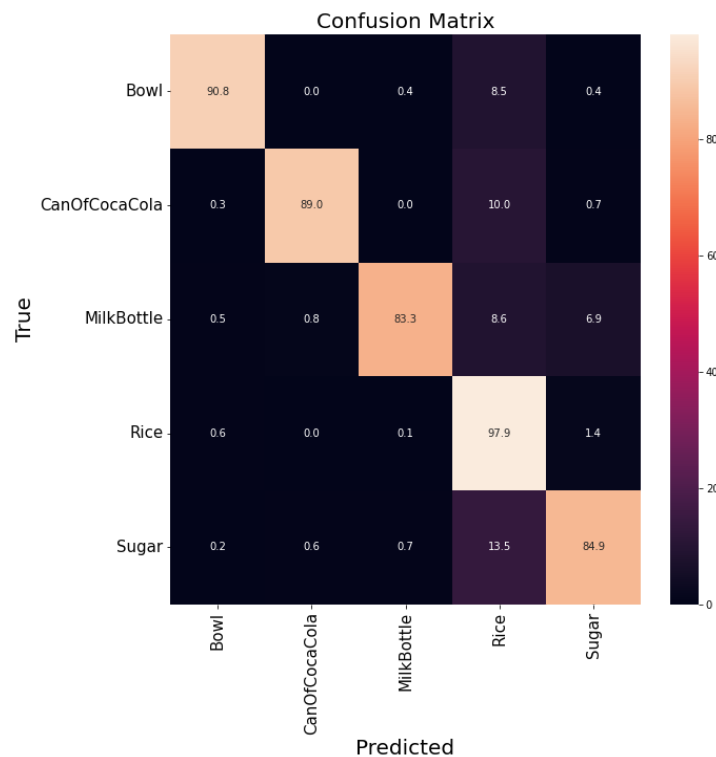


Figure 3 : Matrice de confusion de performancs du modèle sur le jeu de données de test

TRACKING

Le but du tracking est de détecter le positionnement de l'objet d'intérêt à partir d'une première détection correcte. Pour ce faire, il faut connaître pour chaque *frame* quel objet est détecté et quel est le positionnement de sa boîte englobante.

Les données citées précédemment nous permettent de récupérer la première *frame* où l'objet d'intérêt apparaît et de visualiser la boîte qui l'englobe tel que :

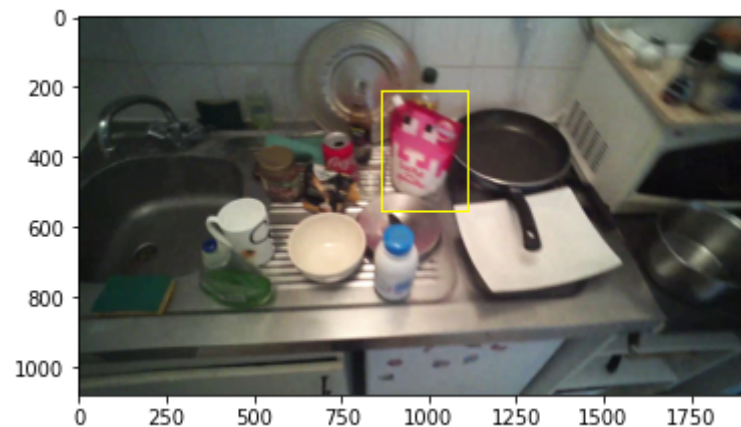


Figure 4 : Visualisation de la boîte englobante pour la première *frame* où l'objet d'intérêt apparaît

Pour prédire la boîte englobante suivante, nous récupérons la *frame* suivante sur laquelle nous générons de potentielles boîtes créées. Ces boîtes sont créées en modifiant la boîte englobante réelle de la *frame* précédente : changement de hauteur, de largeur, déplacement, etc.

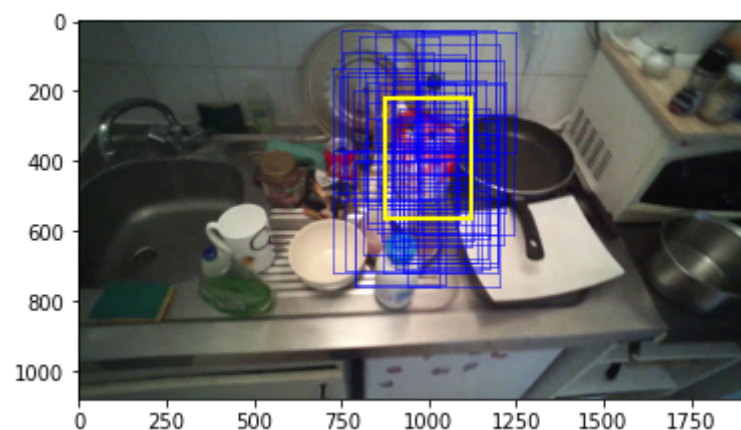


Figure 5 : Visualisation des propositions de boîtes par rapport à la boîte précédente

La fonction générant ces nouvelles boîtes dans *utils/helpers* nous permet de définir à quel point les propositions de boîtes auront une taille différente et une position différente de la boîte d'origine. Ces paramètres influent légèrement sur l'efficacité du suivi et ils devraient être différents pour chaque vidéo pour un *tracking* optimal. En effet, si l'objet bouge vite il faut générer des boîtes plus éloignées de celle d'origine sinon on peut potentiellement perdre le suivi. Par ailleurs, il faut limiter le nombre de

propositions pour réduire au maximum le temps de prédiction. Cependant un nombre trop réduit de boîtes empêche de générer une boîte qui englobe correctement l'objet. Il y a donc un compromis à faire entre justesse et rapidité.

Parmi les boîtes proposées est sélectionnée celle qui selon le classifieur représente le mieux l'objet suivi. Pour ce faire, nous effectuons des prédictions à l'aide du classifieur entraîné précédemment, la "meilleure" boîte est alors la boîte dont la probabilité est la plus élevée pour l'objet sujet de la vidéo. Il est alors possible de prédire le déplacement pour toute la vidéo en répétant cette opération pour chaque *frame*.

Le calcul de l'IOU (*Intersection over Union*), nous permet d'évaluer notre *tracking* par rapport aux boîtes englobantes réelles. La multiplicité des boîtes générables semble indiquer que l'IOU ne sera jamais parfait. Toutefois, c'est une bonne mesure pour évaluer les performances de notre tracking.

INCREMENTAL LEARNING

Afin de parfaire notre tracking, il est possible d'utiliser des techniques d'*incremental learning*. A partir des prédictions de boîtes englobantes sur les *frames* de nos vidéos, nous pouvons créer de nouvelles données que l'on souhaite utiliser pour affiner notre classifieur.

En effet, dans le notebook intitulé `incremental_learning.ipynb`, nous avons réutilisé le code du notebook `tracking.ipynb`, que nous avons itéré sur l'ensemble des vidéos dont nous disposons. Le but est, pour une vidéo donnée, de regarder pour chaque *frame* si la boîte prédite par le classifieur est proche ou non de la réelle boîte englobante. Pour évaluer cette notion de proximité, nous avons calculé l'IOU (*Intersection Over Union*) entre la boîte réelle et la boîte prédite. Si l'IOU obtenue est supérieure à 0.5, elle est considérée comme une bonne prédiction. Nous ajoutons alors l'image correspondant à la boîte prédite dans un tableau qui est utilisé comme entrée du classifieur. Nous ajoutons également à un tableau de labels le vecteur représentant

la classe correspondante. Cependant, si l'IOU obtenue est inférieure à 0.5, nous ajoutons également l'image correspondant à la boîte prédite au tableau d'images, mais cette fois-ci nous ajoutons au tableau de labels un vecteur nul, ce qui correspond à une image dans laquelle il n'y a pas d'objet. Une fois toutes les *frames* d'une vidéo testées, nous entraînons à nouveau les dernières couches de notre classifieur à partir des images sauvegardées durant le traitement de la vidéo et leur label correspondant.

Le code reprenant l'idée énoncée ci-dessus a bel et bien été écrit dans le notebook `incremental_learning.ipynb`, mais n'ayant pas de gpu, il n'a pas pu être entièrement exécuté car demandait un grand temps d'exécution. En effet, le traitement de deux vidéos a pris 25 minutes et il y a une cinquantaine de vidéos mises à notre disposition. Nous n'avons donc pas pu comparer les performances de tracking entre un classifieur restant "statique" et un classifieur entraîné avec de nouvelles données arrivant à la volée.

CONCLUSION

Ainsi pour ce projet nous avons réussi à entraîner un classifieur pour distinguer différents types d'objets. Il fut intéressant d'utiliser celui-ci pour procéder à un *tracking* sans recourir à des techniques de détection sémantique d'objet comme YOLO ou RCNN. Cependant, nous nous sommes rendu compte que certains des modèles qui nous ont été proposés étaient plus efficaces que le classifieur entraîné, et nous avons donc utilisé ce modèle par la suite, afin d'obtenir un *tracking* plus précis. Nous avons pu visualiser les performances de notre *tracker* en affichant pour chaque *frame* d'une vidéo les boîtes aléatoires générées, la boîte prédite par le classifieur ainsi que la boîte réelle, ce qui nous a permis d'avoir un ordre d'idée de sa performance. Malgré cela, il nous a été difficile d'obtenir des mesures de performances exactes car il aurait fallu utiliser ce *tracker* sur l'ensemble des vidéos une première fois sans mise à jour du classifieur et relever sa performance totale (une moyenne d'IOU par exemple), puis une autre fois avec mise à jour du classifieur, afin de pouvoir comparer les deux performances.

Néanmoins les moyens dont nous disposons ne nous ont pas permis d'arriver à cette étape, qui aurait été intéressante à analyser.

Nous sommes satisfaits de notre *tracker*, même si nous aurions pu l'améliorer de différentes manières. Tout d'abord, nous avons utilisé les mêmes paramètres de génération de boîtes aléatoires pour chaque vidéo, alors qu'utiliser des paramètres adaptés à la vitesse ou au mouvement de chaque vidéo par exemple aurait certainement mené à de meilleurs résultats. De plus, dans certaines vidéos, après première détection de l'objet, il arrive qu'au cours de la vidéo, l'objet soit caché par une main humaine qui souhaite l'attraper par exemple. Dans ce cas-là, l'objet n'apparaissant plus dans la vidéo, nous n'avons aucune information sur sa localisation. Cette situation peut facilement se prolonger sur une dizaine de frames, ce qui peut nous faire perdre la trace de l'objet. Il faudrait donc trouver une solution pour mieux gérer ces situations. Ces deux problèmes sont, pour nous, les principaux axes d'améliorations que nous aurions abordés, afin d'obtenir de meilleures performances.