RAPPORT

Projet de l'option IA 2020-2021

Thomas GENTILHOMME

COMPUTER VISION

Suivi automatique d'un individu dans un environnement multi-caméras

Encadrant : Laurent CERVONI

Référent : Olivier FERCOQ

TABLE DES MATIERES

Mots clés : *Detection, Tracking, Tracking-by-Detection, MOT, Re-Identitication, Segmentation, RCNN, FPN, mask, SORT, DeepSORT, Kalman filters, Hungarian algorithm, Nearest Neighbors, Cosine distance.*

Lien GitHub : https://github.com/Thomas-Gentilhomme/AI-Project

1. INTRODUCTION

1.1. Objectifs

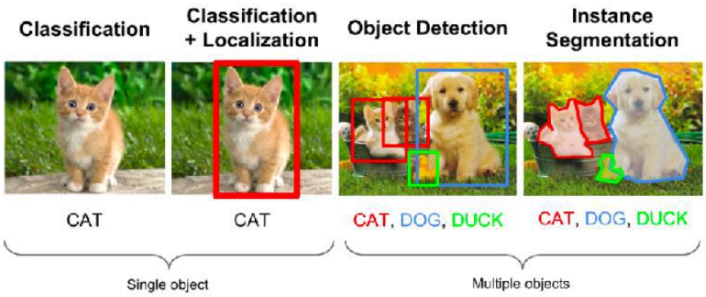
La détection automatique et le tracking d'individus dans un environnement multi-caméras est un enjeu majeur de la Computer Vison et de l'IA en général par les multiples applications que l'on peut en tirer, des nombreux défis techniques que cela représente et des problèmes éthiques que cela peut poser.

Que cela soit pour des applications dans le domaine de la surveillance, de la sécurité ou du simple marketing, en repérant un enfant perdu dans une gare ou créant un parcours utilisateur au sein d'un centre commercial, on rencontre les mêmes problématiques : améliorer la qualité de la détection d'un individu au sein d'une image, reconnaitre ce même individu sur une autre image de la même caméra ou d'une autre, résoudre les problèmes d'occlusions et de superpositions entre différentes caméras, conserver une certaine efficacité même lors de scènes bondés, être assez performant pour réaliser un tracking en direct.

L'objectif de ce projet est d'implémenter un système de ***tracking-by-detection***, c'est-à-dire d'implémenter un algorithme de détection de personnes puis de reconnaître ces mêmes personnes dans d'autres images d'un environnement multi-caméras.

1.2. Généralités

Il faut tout d'abord distinguer deux choses bien différentes : la **détection** de multiples objets au sein d'une vidéo (leur classe (« individus » en l'occurrence) et leur position) et le **tracking** de ces mêmes objets en conservant en mémoire certaines de leurs apparitions précédentes. La détection (ou la segmentation) est à différentier des simples classifications : dans le cas de la détection ou de la segmentation, nous dessinons une boite englobante ou un masque l'objet classifié, et nous faisons cela pour les différents objets d'intérêts au sein d'une même image :



Il faut également différencier deux approches permettant de traiter cette problématique, dont les différences sont subtiles mais fondamentales :

- Le **tracking multi-caméras multi-cibles** consistant à suivre plusieurs individus au travers de vidéos prises depuis plusieurs caméras distinctes.
  Avec cette méthode, on détermine la **position de chaque individu dans chaque image de chaque vidéo**. Il en ressort les **trajectoires de chaque individu** : cette méthode est particulièrement adaptée à la détection d'anomalies (trajectoires étranges), au tracking de sportifs ou à l'analyse du comportement d'une foule de personnes. Le principal problème de cette méthode étant les périodes d'occlusions lorsque les individus se situent hors-champs, le fait que l'on ne connaisse pas le nombre d'individus en avance et la quantité gigantesque de volume de données à traiter.

- La **Re-Identification** de personnes consistant à produire un classement de similarité des individus précédemment enregistrés par rapport à une nouvelle image d'un individu.
  Considérant la capture d'un personne (une boite englobante appelée la query), cette méthode recherche dans une base de données une liste d'autres captures d'individus, prises par d'autres caméras (ou bien la même) à d'autres moments, et les classe par similarité décroissante par rapport à la query. Le but étant que chaque capture décrivant une même personne que celle de la query soit très bien classée.

Ces deux méthodes peuvent être complémentaires, c'est ce que nous verrons dans la dernière partie de ce rapport. En effet, le tracking multi-caméras multi-cibles peut s'appuyer sur un système de Re-Identification lorsque le tracking est préalablement performé sur chaque vidéo indépendamment les unes des autres. Dans ce cas, il ne reste simplement qu'à retrouver, pour le suivi d'un individu donné d'une vidéo donné, les suivis du mêmes individus dans les autres vidéos. Ce qui est intéressant avec cette méthode qui découple la partie tracking et la partie multi-caméras est qu'elle permet d'effectuer successivement une étude de tracking-by-detection sur une vidéo unique puis une étude dans un environnement multi-caméras.

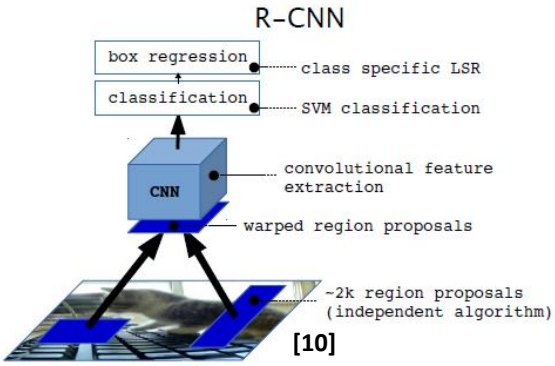
2. ALGORITHMES DE DETECTION

Avec l'essor du Deep Learning en 2012, de multiples algorithmes ont été développé afin de d'améliorer les tâches de *détection* (ou *segmentation*) d'individus au sein d'une image. Parmi les plus connus, on peut citer les *R-CNN* **[1]** et ses améliorations, *OpenPose*, *SSD*, *SPP*, *YOLO* ou le très récent *DETR* basé sur une architecture Transformer.

J'ai décidé de m'intéresser à la méthode **Mask R-CNN** pour ses performances par rapport à la plupart des autres méthodes citées, pour la richesse de la documentation la concernant, pour sa capacité à produire aussi bien des boites englobantes que des masques et pour l'intérêt personnel que j'ai trouvé à étudier en détail son architecture. Je décris ci-après l'architecture du *R-CNN* (*Regions with CNN Features*) proposée par Ross Girshick en 2014 et qui combine réseaux convolutionnels et algorithmes de proposition de régions, ainsi que ses principales améliorations, jusqu'à la plus récente proposée par le même auteur en 2018 au sein du FAIR (Facebook AI Research) : le *Mask R-CNN*.

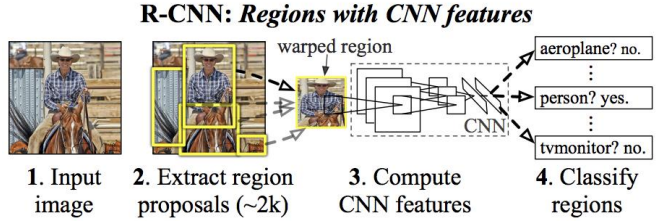
2.1. R-CNN

Cette architecture consiste à proposer 2000 régions d'intérêts sur une image. Ces régions d'intérêts sont ensuite remodelées sous une forme fixe pour pouvoir passer à travers un CNN afin d'extraire des features qui sont ensuite classifiées à l'aide d'un SVM, proposant ainsi une **classe** pour l'objet de la région et une **bounding box** pour celui-ci en prédisant 4 valeurs de correction permettant d'ajuster la région proposée étudiée.



[10]

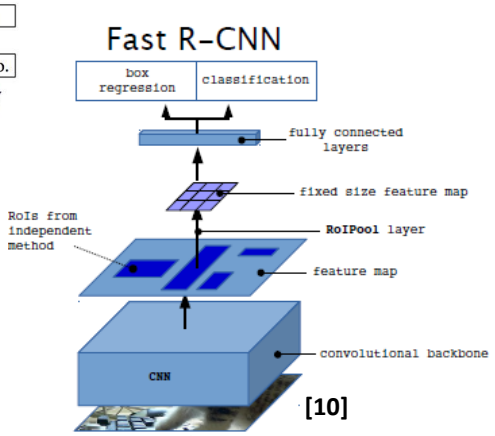
Le problème majeure de cette méthode est que l'algorithme proposant les 2000 régions d'intérêts (*Algorithme de recherche sélective* **[2]***)* est très long et n'apprend rien.



[5]

2.2. Fast R-CNN

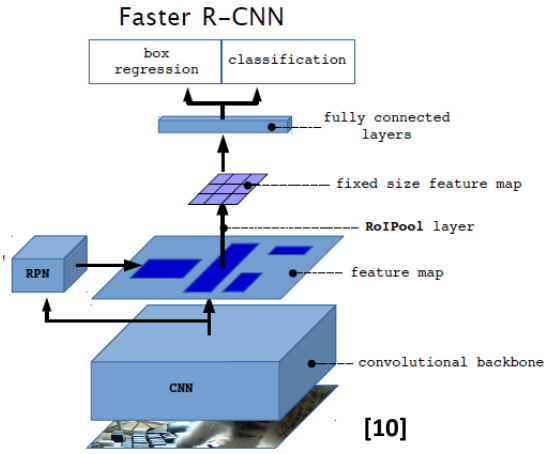
Une première amélioration de cet algorithme a été proposée en 2015 par le même auteur et s'intitule **Fast R-CNN. [3]**



[10]

L'approche est presque identique, à la différence que cette fois les régions proposées le sont après un passage dans un CNN pour obtenir une feature map. C'est au niveau de cette feature map que l'algorithme de recherche sélective génère ses propositions de régions. Cette approche améliore nettement la durée d'entrainement et de test de la méthode R-CNN mais celle-ci demeure très élevée, et rend notamment impossible la détection en temps réel. Le détail de l'étage de détection est expliqué dans la partie suivante.

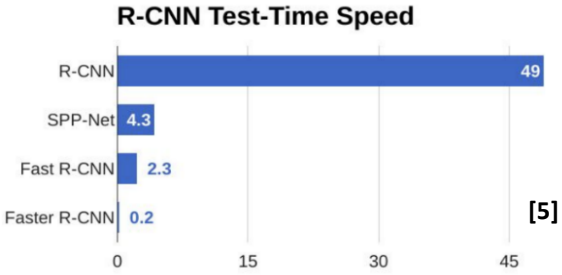
2.3. Faster R-CNN



[10]

Nous nous appuyons donc sur une autre évolution de cet algorithme appelée **Faster R-CNN**, proposée en 2015, consistant à apprendre les régions d'intérêts grâce à un CNN plutôt que d'utiliser un algorithme de recherche sélective. **[4]**

Ici, l'image est donnée en entrée d'un réseau convolutif afin de produire une feature map. Cette feature map, au lieu de passer au travers d'un algorithme de recherche sélective comme dans le cas de la méthode Fast R-CNN, est passée en entrée d'un autre réseau convolutif afin de produire un certain nombre de propositions de région. Ces propositions de régions sont ensuite remodelées à l'aide d'une RoI Pooling Layer (**Region of Interest Pooling**) puis utilisées pour classifier l'image ainsi que prédire les valeurs de corrections (*offset values*) des bounding boxes proposées.



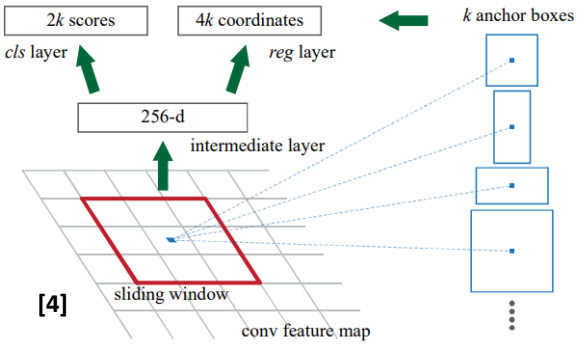
[5]

L'algorithme Faster RCNN est bien plus rapide que ces prédécesseurs RCNN et Fast RCNN, il est d'ailleurs **suffisamment rapide pour pouvoir s'approcher de la détection d'objet en temps réel (environ $10\ fps$).**

2.3.1. Region Proposal Network (RPN)

A la différence de R-CNN et Fast R-CNN, l'étage de proposition de région n'est plus une *recherche sélective* est un *réseau de neurones convolutifs* dont la sortie a une influence sur le réseau de détection : la proposition de régions et l'étage de détection ne sont plus décorrélés. En effet, la *recherche sélective* disposait de deux problèmes : le premier étant sa lenteur (cet étage concentrait une proportion très importante du temps de calcul ($\approx 86\%$)) et le second étant qu'il ne participait aucunement à l'apprentissage de l'algorithme, pouvant alors générer de mauvais candidats de régions.

D'abord, l'image d'entrée passe à travers des couches de convolutions successives afin d'extraire une feature map. Ensuite, le *RPN* utilise une fenêtre glissante de taille $3x3$ pour parcourir cette feature map. A chaque localisation de cette fenêtre, au maximum k propositions de régions sont générées. Pour cela, k boites de références (*anchor boxes*), centrées sur la fenêtre correspondante, sont utilisées (généralement $k = 9$ avec 3 échelles différentes : 128, 256 et 512 ; 3 ratios différents : $1{:}1, 1{:}2, 2{:}1$). Une couche de classification (**cls**) génère $2k$ scores indiquant s'il y a un objet ou non pour les k *anchor boxes* (*object class vs. background*) et une couche de régression (**reg**) génère $4k$ coordonnées pour les k *anchor boxes*. Avec une feature map de taille WH, il y a en tout WHk anchor boxes.

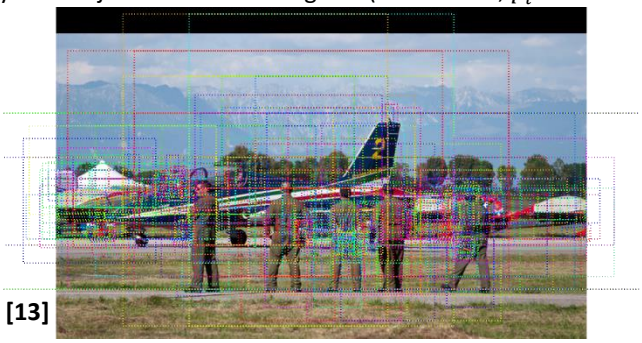


[4]

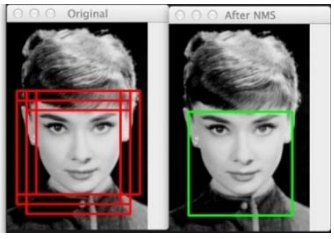
La fonction de *loss* associé au RPN est :

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)$$

La fonction de *loss* totale additionne la ***loss* de la couche de classification entre 2 classes** (il y a un objet *vs* il n'y a pas d'objet) à la ***loss* de la couche de régression** *seulement* lorsqu'il y a un objet dans la bounding box (dans ce cas, $p_i^* = 1$). p_i est la probabilité prédite que l'*anchor* i soit un objet et p_i^* vaut 1 si le l'*anchor* est labelisé positif et 0 s'il est labelisé négatif. Un *anchor* étant labelisé positif s'il chevauche suffisamment une vraie boite englobante d'objet de l'image d'entrée (*i.e.* il a un ratio IoU (Intersection Over Union) supérieur à 0.7 pour n'importe quelle véritable boite) et négatif si ce ratio est inférieur à 0.3. Les autres *anchors* ne sont pas considérés pour l'entrainement. t_i est le vecteur de coordonnées de l'*anchor* i et t_i^* les coordonnées de la boite réelle associée. L_{cls} est la *Cross-Entropy Binaire* et L_{reg} la loss *Smooth L1*.

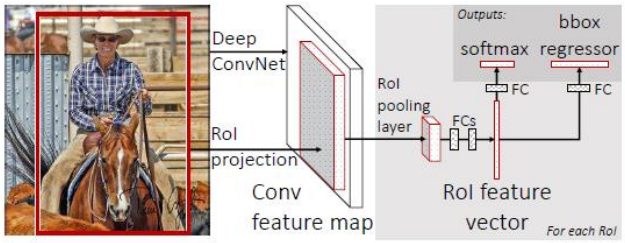

[13]

Ainsi, le réseau de proposition de régions ne fait pas que proposer des régions, mais réalise aussi un « pre-check » afin de savoir *quelles* sont les régions contenant effectivement un objet et ces régions vont ensuite passer à travers le réseau de détection afin de connaître la classe de l'objet en question et retourner la bounding box correspondante.



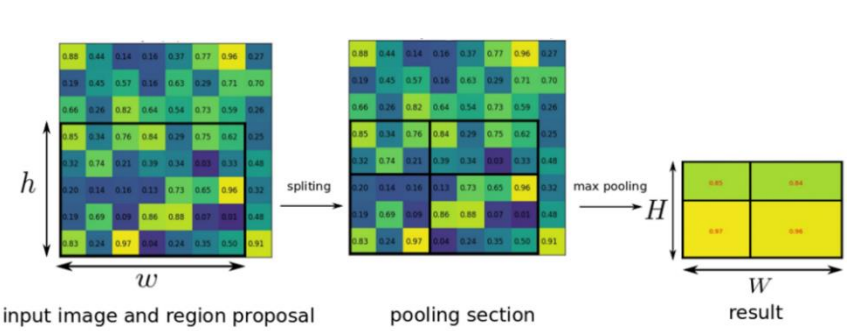
NB : comme énormément de régions vont se superposer dû au fait que certains *anchor* sont dénotent les mêmes objets, la méthode **NMS** (*Non-Maximum Suppression*) **[7]** permet de réduire considérablement le nombre de régions proposées d'environ 6000 à N (généralement $N \approx 300$) en regroupant les régions voisines.

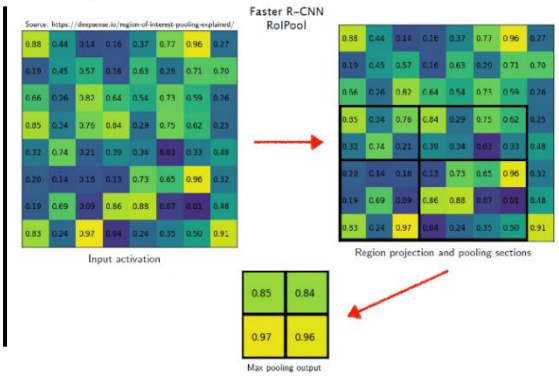
2.3.2. Detection Network



Ce niveau est similaire à celui d'un *Fast R-CNN*. **Pour chaque région d'intérêt**, on passe d'abord dans une *RoI Pooling Layer* puis à travers un réseau convolutionnels ainsi que deux branches comportant chacune une *Fully-connected Layer* : une pour la **classification de l'objet** (*softmax*) et une pour la **génération de la bounding box** (*bounding box regressor*). La RoI Pooling Layer est un cas particulier d'une couche **SSP** (*Spacial Pyramid Pooling*) de *SSPNet* avec l'utilisation d'**une seule pyramide. [8]**

Chaque région proposée issue de l'étape précédente produit la région d'intérêt utilisée dans la couche de RoI pooling. Si dans la feature map initiale nous disposons d'une région proposée de taille $h \times w$ et que l'on souhaite avoir en sortie de la couche de pooling une région de taille $H \times W$, alors la surface de chaque zone de pooling est égale à $h/H \times w/W$. Comme pour la classique *MaxPooling Layer*, **on ne conserve que la valeur maximum dans une fenêtre de pooling.**





Une *multi-task loss* est alors utilisée pour apprendre aussi bien la classe de l'objet que la position et taille de la bounding box associée :

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v)$$

Avec :

$$L_{cls}(p, u) = -\log p_u \text{ : } \textit{loss} \text{ pour la vraie classe } u.$$

$$L_{loc}(t_u, v) = \sum_{i \in [x,y,w,h]} smooth_{L_1}(t_i^u - v_i) \text{ : } \textit{loss} \text{ pour la bounding box.}$$

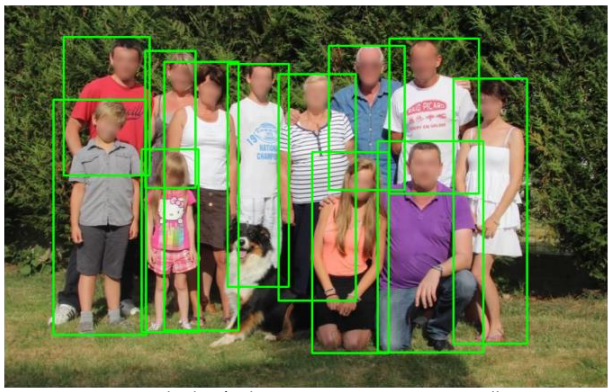
$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & if\ |x| < 1 \\ |x| - 0.5 & otherwise \end{cases}$$

$[u \geq 1]$: égale à 1 quand $u \geq 1$ ($u = 0$ correspondant à la classe « *background* »).

p: Predicted Class ; u: GT Class ; t^u: Predicted Bounding Box for class u ; v: GT Bounding Box.

Un autre aspect très intéressant de la méthode *Faster R-CNN* est le partage des couches de convolutions entre l'étage *RPN* et l'étage de *détection* pour obtenir un réseau unifié. Pour cela, la méthode *Alternating training* est appliquée. Il y a quatre étapes : le RPN est d'abord entrainé seul. Il est initialisé sur un modèle pré-entrainé sur *ImageNet* et *fine-tuné* sur la tache de proposition de région. Ensuite, l'étage de *détection* est entrainé séparément en utilisant comme *inputs* les régions proposées par le *RPN* à l'étape précédente. Ce réseau est également initialisé avec le modèle pré-entrainé sur *ImageNet*. A ce moment-là, les deux étages de partage rien. La troisième étape consiste à utiliser le réseau de *détection* pour initialiser l'entrainement du *RPN*, sauf que les couches de convolutions communes aux deux réseaux sont fixées et seules les couches uniques au *RPN* sont *fine-tunées*. Finalement, toujours en fixant les couches de convolutions communes, les couches uniques de l'étage de *détection* sont également *fine-tunées.* Ainsi, les deux réseaux partagent les mêmes couches de convolutions et forment un réseau unifié.

NB : pourquoi utiliser un *softmax* plutôt qu'un *SVM* dans la branche de classification ? Les résultats présentés dans la littérature sont sensiblement meilleurs pour un *softmax*. De plus, le *softmax* n'a pas besoin de conserver en mémoire des données intermédiaires, à l'inverse d'un *SVM* qui va garder en mémoire les vecteurs (éventuellement plusieurs *giga* de données).

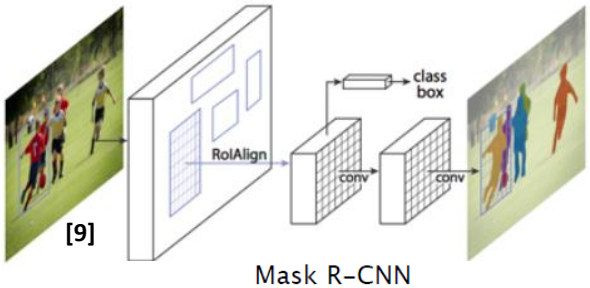


Exemple de résultat sur une image personnelle.

2.4. Mask R-CNN

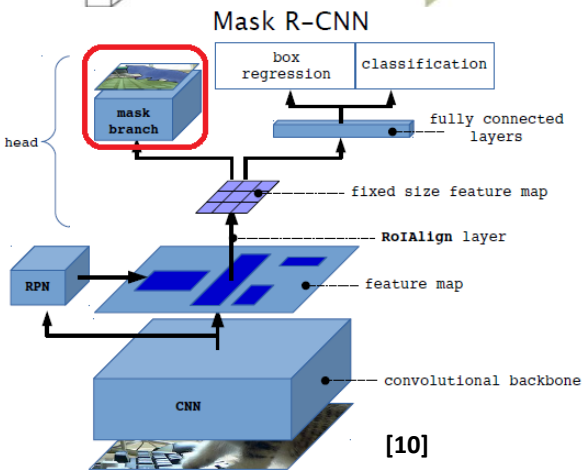
L'algorithme qui sera finalement utilisé dans ce projet est le **Mask R-CNN**, il s'agit d'une amélioration de *Faster R-CNN* proposé par le même auteur*, R. Girshick*, et développé au sein du *FAIR*. Ses performances ont dépassé celles des gagnants du challenge COCO2016 et le gagnant de COCO2017 est basé sur une architecture *Mask R-CNN*. **[9][10][12]**

Pour le moment, nous n'avons vu que de la détection d'objets (classer de multiples objets sur une image tout en donnant leur bounding box). Mask R-CNN permet d'aller plus loin en faisant de l'**Instance Segmentation**, c'est-à-dire que nous seulement nous pourront réaliser de la détection d'objet avec boite englobante, mais nous pourrons également délimiter chaque objet d'intérêt distinct dans l'image avec un masque.



[9]

Nous nous appuyons donc principalement sur l'architecture du *Faster R-CNN* (voir image ci-contre), la différence majeure étant l'ajout, à la fin du réseau, d'une **autre branche de sortie** (la *mask branch*), qui génère le masque pour la segmentation d'instance



[10]

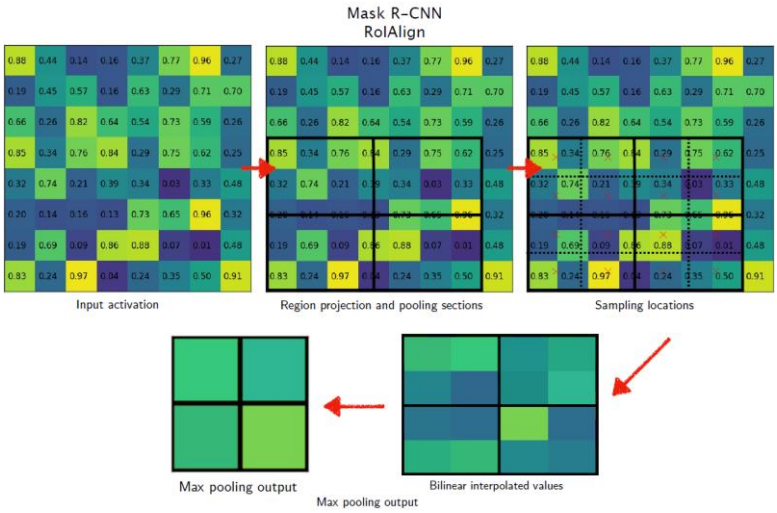
Nous disposons donc toujours d'une architecture en deux étages, avec un premier étage (le *RPN*) qui génère les régions candidates dans la feature map et un second étage qui, pour chaque région candidate, lui applique une couche de *RoI Align* puis détermine sa classe, sa boite englobante ainsi que son masque.

Nous retrouvons donc logiquement le concept de la *multi-task loss* vue précédemment, avec en plus un L_{mask} :

$$Loss = L_{cls} + L_{loc} + L_{mask}$$

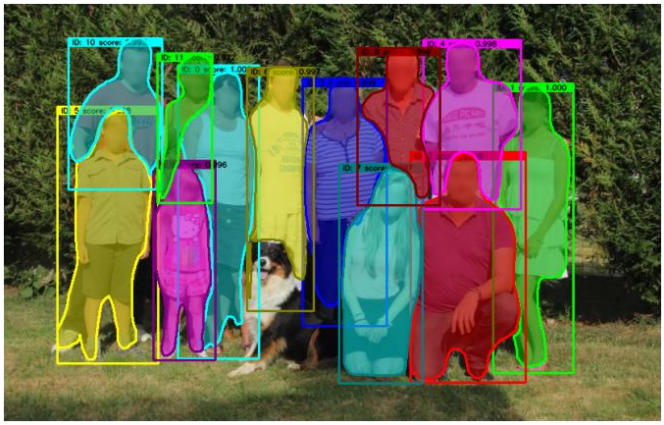
L_{mask} est la *loss* du masque binaire. Cette branche a Km^2 sortie pour chaque région passant par la *RoI Align layer*, qui sont K masques de résolution $m \times m$ (K étant le nombre de classes). Cette *loss* utilise la *Average Binary Cross-Entropy* : une fonction *sigmoid* est appliqué sur *chaque* pixel, seulement le k*-ième* mask contribue à l'actualisation de la *loss*, k étant la vraie classe de l'objet. On utilise une fonction *sigmoid* et la *binary cross-entropy* (et non une *softmax* avec une *cross-entropy*) puisque les tâches de masque et de prédiction de classe sont ici complètement séparées.

Le *Mask R-CNN* dispose d'une autre variante par rapport au *Faster R-CNN*. On n'utilise plus ici une couche de *RoI Pooling* mais une couche de *RoI Align*. La différence entre les deux est, qu'au lieu d'arrondir la taille des fenêtres de pooling pour avoir des tailles entières, on réalise des fenêtres qui sont toutes de même tailles (voir image ci-dessous). En se basant sur les surfaces se superposant, on utilise l'interpolation bilinéaire pour obtenir une feature map intermédiaire, puis on effectue comme précédemment un *MaxPooling* sur cette feature map intermédiaire.



NB **:** le *Mask R-CNN* peut également être amélioré pour faire de la Human Pose Estimation. Le *mAP* diminue légèrement en ajoutant cette fonctionnalité)

Exemples de Human Pose Estimation avec MaskRCNN issus du papier de l'auteur.



Exemple de résultat sur une image personnelle.

3. METHODES DE TRACKING

Afin de réaliser un tracking d'individus, de multiples algorithmes existent, certains très anciens comme l'algorithme du Mean-Shift et la transformée de Hough. D'autres sont plus récents comme les réseaux siamois, l'algorithme SORT ou son amélioration Deep SORT.

3.1. Mean-Shift et la transformée de Hough

L'algorithme du **Mean Shift** est un algorithme efficace pour traquer des objets dont **l'apparence est définie par des histogrammes**. Il s'agit d'un algorithme **itératif** permettant de traquer les modes d'une distribution. Il déplace un point *x* vers le maximum local le plus proche. L'idée est que, par une modélisation différentiel des mesures de similarité entre distributions, on va pouvoir trouver de façon itérative la position la plus probable de l'objet en calculant de façon marginale l'augmentation ou la diminution de cette distance entre la position estimée (position initiale du point) et la position finale du point pour trouver celui qui va maximiser cette distance.

L'algorithme de la **transformée de Hough** consiste à calculer cette transformée sur toutes les images de la séquence pour en déduire la nouvelle position du point central d'un objet tracké sur chacune d'entre elles avec la valeur maximale de cette transformée. Pour cela, l'algorithme calcule l'orientation locale de la fenêtre englobante (*i.e.* l'argument du gradient des pixels) et conserve uniquement les pixels dont l'orientation est significative en définissant un **seuil sur le module** du gradient : cela revient donc à conserver uniquement les « **pixels de bords** » de l'objet d'intérêt.

Une *R-Table* est ensuite créée, c'est-à-dire un modèle de forme de l'objet que l'on souhaite traquer, indexé sur l'orientation. Il s'agit d'un tableau dont chaque ligne correspond à une orientation de bords (qui est perpendiculaire à l'orientation du gradient des pixels le constituant) et dont chaque entrée représente la localisation d'un pixel de bord par rapport à un point central arbitrairement choisi.

Une opération similaire est effectuée sur l'ensemble des frames de la vidéo : on récupère les pixels de bords et l'orientation associée. On pourra alors calculée la nouvelle position du point central de l'objet tracké en déterminant le transformée de Hough entre la R-Table de l'objet et la frame.

Les avantages de ces algorithmes sont principalement d'être **online** et l'**inutilité de maintenir l'utilisation d'une détection sur chacune des frames**. Un détection initiale suffit, ensuite le principe même de ces algorithmes est d'utiliser les caractéristiques propres de l'image pour maintenir une fenêtre englobante de frame en frame. Il s'agit d'un avantage conséquent notamment en gain de temps de calcul, celui-ci étant principalement consommé par la partie détection comme indiqué plus loin dans la partie implémentation de ce rapport

J'ai pu implémenter et tester ces algorithmes mais je ne les ai pas gardés face à leurs nombreux inconvénients :

- Il est difficile pour ces algorithmes de différencier deux individus, surtout si ceux-ci ont des couleurs similaires et sont situés proches l'un de l'autre.
- Ces méthodes ne sont pas du tout adaptées aux occlusion, ainsi qu'aux cas où un individu sort de l'image, ou bien arrive après le début de la vidéo. Également, la taille de la fenêtre englobante demeure fixe au fil des frames.

Ces contraintes m'ont poussé à abandonner ces méthodes pour me tourner vers des algorithmes plus récents.

3.2. Réseaux Siamois

Un modèle permettant de mettre en place la Re-Identification est le **Siamese Network** (= Réseau Siamois). Ce réseau prend en entrée deux bounding box différentes (contenant donc chacun un individu) et détermine une distance de similarité entre ces deux détections. C'est cette distance qui permet de classer, pour chaque bounding box, la liste des autres bounding box les plus similaires, et donc de reconnaitre d'autres individus. L'apprentissage d'un tel réseau est effectué nottement grâce à la **loss triple**. Cette *loss* est calculée sur un triplet d'image A, P, N où A est une *query,* P un exemple positif et N un exemple négatif : $l(A, P, N) = \max(d(A, P) - d(A, N) + \alpha,\ 0)$. Un dataset intéressant pour entrainer ce réseau de Re-ID est **Market-1501.**

Je n'ai pas choisi cette méthode car elle n'était pas adaptée à notre problème pour son principal inconvénient : elle est **offline**. En effet, il s'agit de re-identification : c'est-à-dire que le réseau doit d'ores et déjà avoir parcouru la vidéo complète avant de réidentifier chaque individu. Cette méthode requiert également de connaitre à l'avance le nombre d'individus à traquer pour pouvoir clustériser les résultats en se basant sur les mesures de similarité en sortie. Cependant, cette méthode sera utilisée dans le cas multi-caméras détaillé dans la dernière partie de ce rapport.

3.3. Deep Sort: Simple Online and Realtime Tracking with a deep association metric

La méthode que j'ai finalement conservée est celle du **DeepSORT [15][16]**, une amélioration de **SORT [14]**, car elle présente de nombreux avantages :

- Cette méthode est **online.** Elle permet donc de traquer des individus de frames en frames en live (sous réserve d'un temps d'exécution suffisamment faible).
- Elle correspond à l'idée **intuitive** que je me faisais d'un algorithme de tracking issus d'un étage de détection : l'utilisation de features issus des fenêtres englobantes d'une frame pour identifier des individus en calculant la distance minimale aux features des fenêtres englobantes des frames précédentes.
- Cette méthode est particulièrement efficace pour distinguer les individus les uns des autres et pour prendre en considération plusieurs paramètres tels que la **mémoire de l'algorithme** (à partir de quand un individu qui n'est plus détecté peut être oublié) et l'**IoU matching** (si la distance entre deux individus est faible mais que leur fenêtre respective ne se superposent que peu (voire pas du tout), ne pas prendre en compte ce match : cela revient à dire qu'il est impossible d'avoir des individus se téléportant ou se déplaçant extrêmement vite).
- Cette méthode est à la fois la *plus rapide* et la *plus efficace* d'après la littérature. Cela sera confirmée par l'étude détaillée plus loin.

L'algorithme considère le problème de tracking comme un **problème d'association données** : le but étant de correctement associer entre elles les différentes détections d'une vidéo. Le tracking d'un individu se fait dans un espace à 8 dimensions $(u, v, \gamma, h, \dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$ où (u, v) représentent le centre de la boite englobante, γ le rapport de forme de la boite, h sa hauteur et $(\dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$ leur vélocité respective. Ces données permettent de prédire une boite englobante cible par l'utilisation de filtres de Kalman : c'est cette boite prédite qui sera comparée aux boites détectées dans l'algorithme d'association. Un filtre de Kalman est une opération consistant à estimer l'état d'un système dynamique à partir d'observation partielles et bruitées : dans notre cas il s'agit de prédire la future boite englobante d'un objet tracké. **[17]**

Dans la suite, nous qualifierons de « ***track*** » un objet représentant un individu sur un ensemble de frames.

3.3.1. Usage d'un descripteur d'apparence

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

L'un des avantages majeurs de cette technique est qu'elle concentre la majeur partie de la complexité computationnel dans un **pré-entrainement offline, totalement indépendant des objets traqués**.

En effet, l'une des composantes de l'algorithme d'association Tracks/Détections détaillé plus loin est l'utilisation de **descripteurs d'apparence riches** des boites englobantes. Ces descripteurs sont obtenus par l'utilisation d'un *CNN pré-entrainé* sur un dataset de Re-Identification de piétons : ***MARS*** (*Motion Analysis and Re-identification Set*), une amélioration de *Market1501* contenant 1,1 million d'images de 1,261 individus différents. Le *CNN* utilisé est un *ResNet* composé de deux couches de convolutions et de six couches résiduelles : la feature map finale est calculée en utilisant une couche *fully-connected* et une *batch+L2 normalization*. Une fois entrainé, le réseau permet d'obtenir une représentation riche des boites d'entrées dans un format adapté à l'usage de la distance Cosinus détaillé plus loin. En effet, lors de son entrainement ce réseau minimise une *loss* triple en minimisant la distance cosinus entre les mêmes individus et en la maximisant pour les individus différents.

La qualité de ces features dépend donc très fortement de la qualité de la détection en amont.

3.3.2. La suppression et la création de tracks

Quand un individu entre ou sort de l'image, son track associé doit être soit créé soit supprimé.

Pour chaque track $\boldsymbol{k}$, un compteur $\boldsymbol{a_k}$ est maintenu pour savoir le nombre de frames écoulé depuis la dernière association réussie. Par association réussie, on entend « la dernière fois qu'une boite englobante détectée a été affectée au track ». Ce compteur est incrémenté à chaque prédiction du filtre de Kalman et réinitialisé à 0 à chaque association track-détection réussie. A chaque fois que la valeur d'un compteur dépasse une valeur maximale $\boldsymbol{A_{max}}$ (un **âge maximal**), le track associé est supprimé **: l'individu tracké est oublié, on considère qu'il a définitivement quitté l'image**.

A chaque nouvelle détection, de **nouveaux tracks sont initialisés** pour les détections qui n'ont pas pu être associées avec un des tracks déjà existant et sont considérés comme **provisoires** : c'est-à-dire qu'ils seront considérés comme tracks **confirmés** seulement s'ils sont associés avec succès à une détection lors des $\boldsymbol{n_{init}}$ premières frames de leur exitance, sinon ils sont supprimés.

3.3.3. Résoudre le problème d'association

Le but est ici d'associer correctement les états prédis par le filtre de Kalman pour chaque track et les détections de la nouvelle frame. DeepSORT résout ce problème d'assignation par l'utilisation de l'**algorithme Hongrois. [18]** Cette méthode permet de résoudre des problèmes d'affection en temps polynomial. Elle permet de **trouver le meilleur couplage dans un graphe biparti** (états prédits des tracks et détections dans notre cas) dont les arrêtes sont valorisées en minimisant (ou maximisant) le coût de chaque association.

Pour effectuer ces associations, il faut donc préalablement mesurer le coûts de chacune d'entre elles. Pour cela, on utilise **deux mesures de distances** entre le $\boldsymbol{i}$**-ième track** et la $\boldsymbol{j}$**-ième boite englobante** :

- **Distance de Mahalanobis carrée :**

$$d^{(1)}(i,j) = \left(d_j - y_i\right)^T S_i^{-1}\left(d_j - y_i\right)$$

- **Plus petite distance cosinus :**

$$d^{(2)}(i,j) = \min\left\{1 - r_j^T r_k^{(i)} \middle| r_k^{(i)} \in \mathcal{R}_i\right\}$$

Avec :

- $(\boldsymbol{y_i}, \boldsymbol{S_i})$ la projection de la distribution du i-ième track dans l'espace de mesure ;
- $\boldsymbol{d_j}$ la boite englobante de la j-ième détection ;
- $\boldsymbol{r_j}$ $(\|r_j\| = 1)$ un descripteur de $\boldsymbol{d_j}$ calculée à partie d'un *CNN* pré-entrainé ;
- $\boldsymbol{\mathcal{R}_i} = \left\{\boldsymbol{r_k^{(i)}}\right\}_{\boldsymbol{k=1}}^{\boldsymbol{100}}$ une galerie des 100 derniers descripteurs associés au $\boldsymbol{i}$**-ième** track ;

De plus, ces deux distances doivent être en-dessous d'une valeur minimale pour être conservées (*i.e.* on rejette les associations trop différentes car probablement fausses). Pour cela, une variable binaire par distance est créée pour indiquer si l'association est acceptée ou non :

$$b_{i,j}^{(1)} = \mathbb{1}\left[d^{(1)}(i,j) \leq t^{(1)}\right] \quad ; \quad b_{i,j}^{(2)} = \mathbb{1}\left[d^{(2)}(i,j) \leq t^{(2)}\right]$$

$t^{(1)} = 9.4877$ et $t^{(2)}$ est un paramètre du système, que l'on nommera ***MaxCosineDistance.*** La valeur de $t^{(1)}$ garantie à 95% l'exclusion des associations improbables.

Ces deux mesures permettent de mesurer deux aspects différents du problème. D'un côté, la **distance de Mahalanobis** donne de l'**information sur la position possible des objets en se basant leur mouvement** (ce qui est efficace pour les prédictions à court-terme), d'un autre côté, la **distance Cosinus** prend en compte l'**information comprises directement dans les descripteurs** (ce qui efficace pour reconnaitre un individu après une longue occlusion là où son déplacement est moins important). Ces deux mesures, ainsi que leur variables binaires associées, sont combinées :

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j) \quad ; \quad b_{i,j} = b_{i,j}^{(1)} b_{i,j}^{(2)}$$

L'association de ces mesures et de l'algorithme hongrois est détaillée dans ce que les auteurs de *DeepSORT* appellent l'algorithme de **Matching Cascade.** En effet, une fois les matrices de coût $\boldsymbol{C} = [\boldsymbol{c_{i,j}}]$ et d'acceptation/refus $\boldsymbol{B} = [\boldsymbol{b_{i,j}}]$ calculées, pour chaque âge $\boldsymbol{n}$ entre l'instant $\boldsymbol{-1}$ et l'instant $\boldsymbol{-A_{max}}$ une matrice d'association $[\boldsymbol{x_{i,j}}]$ est calculée entre $\boldsymbol{C}$**,** les **tracks** non-associés depuis $\boldsymbol{n}$ et les détections non-encore associées. Les listes des associations réussies $\boldsymbol{\mathcal{M}}$ et des associations rejetées $\boldsymbol{U}$ (en utilisant $\boldsymbol{B}$) sont alors mises à jour. Les associations ne sont donc pas résolues toutes ensembles, mais en *cascade*, c'est-à-dire en commençant par les tracks les plus récemment mis à jour (et donc en les privilégiant), plus probables que certains tracks dont l'âge est plus élevé.

Listing 1 Matching Cascade

Input: Track indices $\mathcal{T} = \{1, \ldots, N\}$, Detection indices $\mathcal{D} = \{1, \ldots, M\}$, Maximum age $A_{\max}$
Compute cost matrix $\boldsymbol{C} = [c_{i,j}]$ using Eq. 5
Compute gate matrix $\boldsymbol{B} = [b_{i,j}]$ using Eq. 6
Initialize set of matches $\mathcal{M} \leftarrow \emptyset$
Initialize set of unmatched detections $\mathcal{U} \leftarrow \mathcal{D}$
for $n \in \{1, \ldots, A_{\max}\}$ **do**
  Select tracks by age $\mathcal{T}_n \leftarrow \{i \in \mathcal{T} \mid a_i = n\}$
  $[x_{i,j}] \leftarrow$ min_cost_matching$(\boldsymbol{C}, \mathcal{T}_n, \mathcal{U})$
  $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i,j) \mid b_{i,j} \cdot x_{i,j} > 0\}$
  $\mathcal{U} \leftarrow \mathcal{U} \setminus \{j \mid \sum_i b_{i,j} \cdot x_{i,j} > 0\}$
end for
return $\mathcal{M}, \mathcal{U}$

Un dernier algorithme est finalement appliqué sur les détections $\boldsymbol{U}$ et les tracks restants non-associés d'âge $\boldsymbol{n} = \mathbf{1}$: le **IoU matching**, c'est-à-dire l'association **« Intersection over Union » [19]**. IoU est une métrique permettant de mesurer la qualité d'une boite englobante issue d'une détection par rapport à une vérité terrain. Pour la mesurer, il suffit de faire l'opération suivante entre les deux boites :

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

En mettant un **seuil sur cette valeur d'IoU**, on « rattrape » en quelques sorte les erreurs de la Matching Cascade en associant des tracks et détections dont l'écart spatial entre la boite englobante prédite par le filtre de Kalman et la boite englobante détectée est inférieur à un seuil. Ainsi, on améliore la robustesse de l'algorithme d'association.

4. IMPLEMENTATION

4.1. Généralités

L'apport technique de ce projet repose dans la construction d'un système complet de *tracking-by-detection* d'individus. C'est-à-dire à joindre ensemble les deux architectures de détection et de tracking couvert par ce rapport dans le but d'opérer un suivi sur un ensemble de vidéos, d'en observer les résultats afin d'affiner les paramètres et conclure quant à la qualité de ces algorithmes et finalement d'étudier le cas particulier du tracking multi-caméras.

4.2. Sources

Afin de réaliser ce projet, je me suis appuyé sur deux implémentations très populaire et libre de droit, que j'ai adapté par endroit au problème sous-jacent :

- L'implémentation de la méthode **Mask-RCNN** en **Python3, Keras** et **TensorFlow** par **Matterport**. **[13]**

Il s'agit de l'implémentation **la plus populaire et la plus utilisée** dans la quasi-intégralité des projets de détection avec *Mask R-CNN*, même plus utilisé que le code d'origine des chercheurs du *FAIR* à l'origine de cette méthode. L'avantage de cette implémentation dans ce projet est qu'elle a été pré-entrainée sur **MS COCO** (*Common Objects in Context*) : un dataset populaire pour entrainer et tester une architecture de détection. En effet, l'une des 80 catégories de ce dataset est justement les individus : on peut donc directement utiliser l'architecture et les poids associés pour détecter des individus.

En sortie, nous avons un dictionnaire par image contenant les clés :

- $\boldsymbol{rois}$: listes des coordonnées des boites englobantes des objets détectés au format $[y_1, x_1, y_2, x_2]$.
- $\boldsymbol{class\_ids}$: liste des **ids** au format *MS COCO* des détections. L'ID **« 1 »** correspondant aux personnes.
- $\boldsymbol{scores}$ **:** liste des scores de confiance entre 0 et 1 dans la classe associée.
- $\boldsymbol{masks}$ **:** liste de masques binaires pour chaque objets détectés dans les boites associées.

- L'implémentation de la méthode **DeepSORT** en **Python** par **Nicolaï Wojke**, l'un des auteurs de *SORT* et *DeepSORT.* **[16]**

Il s'agit de l'implémentation direct de *DeepSORT* par ses auteurs qui ont rendu leur code *opensource*. L'avantage est identique à celui-ci de *Mask R-CNN* : le *CNN* utilisé comme descripteur d'apparence est également pré-entrainé, sur le dataset *MARS* comme indiqué plus tôt, et les poids sont également disponibles.

4.3. Construction d'une architecture globale

Mon système de *tracking-by-detection* est constitué de plusieurs étages :

La création d'une classe **Individuals** permettant de sauvegarder à chaque frame les caractéristiques de tous les individus détectés dans celle-ci : à savoir les coordonnées de leur boites englobantes détectées, leur scores, leur masques, leur identifiant unique fournit par *DeepSORT* et la couleurs utilisé pour chacun d'entre eux.

Une fonction ***tracking*** prenant en argument un objet *Detector*, un objet *Tracker*, un nom *VideoFile* (chemin vers la vidéo à analyser), un nom *OutputVideo* (chemin vers l'endroit où sera enregistré la vidéo avec les détections et le tracking). Cette fonction lit la vidéo frame par frame, applique le détecteur sur chacune d'entre elles puis le tracke. Finalement, elle appelle une fonction ***draw_with_tracking*** que je détaillerai puis loin pour dessiner les boites englobantes ainsi que les masques. La fonction *tracking* renvoie les temps d'exécution des trois étapes principales : à savoir la détection, le tracking et le dessin sur les frames, ainsi que le temps d'exécution global par frame.

Le détecteur utilisé est un objet de la classe **MaskRCNN**. Cette classe créait une instance de *Mask R-CNN* de l'implémentation de *Matterport* avec les poids entrainés sur *MS COCO*. Cette classe est dotée d'une fonction ***predict*** – celle appelée dans la fonction ***tracking*** – qui exécute la détection à l'aide de la fonction ***detect*** du modèle créé, applique ensuite la fonction ***update_results*** puis enregistre les résultats de la détection dans un objet **Individuals** qui est alors retourné par la fonction.

La fonction ***update_results*** prend en entrée le dictionnaire en sortie de la détection et retourne ce même dictionnaire, soulagé des objets qui ne sont pas des personnes et des détections dont le score est inférieur à un seuil réglable, $\mathbf{0.9}$ par exemple.

Le traqueur utilisé est un objet de la classe **DeepSORT**. Cette classe créait une instance de l'encoder (*i.e.* le descripteur d'apparence) et de l'objet **Tracker** de l'implémentation de *Nicolaï Wojke*. Cette classe est dotée d'une fonction ***perform_track*** – celle appelée dans la fonction **tracking** – qui exécute le tracking. Pour cela, elle prend en entrée la frame sur laquelle on travaille ainsi que l'objet **Individuals** retourné par le détecteur. Cette fonction commence par appelée une autre fonction, ***convert_roi_shape*** qui permet de mettre les coordonnées des boites englobantes détectées au format d'entrée utilisé par *DeepSORT*, à savoir $(x_1, y_1, x_2 - x_1, y_2 - y_1)$ où (x_1, y_1) sont les coordonnées du coin en haut à gauche de la boite. Ensuite, on calcule les features d'apparence de chaque sous-images de la frame (l'image contenu dans chaque boites détectées) en utilisant l'encoder. On créait alors une liste qui contient chaque détection sous la forme d'objet **Detection** : il s'agit d'un objet contenant, pour un individu détecté, les coordonnées de sa boite englobante, son score de confiance et son descripteur d'apparence dans la frame en cours d'étude. Ensuite, on applique l'algorithme de *Non-Max-Suppression* (NMS) **[7]** pour supprimer les éventuelles détections redondantes se superposant.

Le tracking en lui-même est appliqué par l'appel successifs à deux fonctions de l'objet Tracker : ***predict*** et ***update***.

La fonction ***predict*** applique des filtres de Kalman sur chaque track connu pour prédire l'état du track à la frame actuelle. Elle incrémente aussi l'âge de la dernière association de chaque track de $\mathbf{1}$.

La fonction ***update*** prend en entrée la liste des objets **Detection** et réalise les associations détaillées dans la partie sur l'algorithme *DeepSORT*. Pour cela, elle commence par réaliser la **matching cascade** entre toutes les détections et tous les tracks sur **max_age** itérations : la fonction sépare le set de tracks en deux sets `*confirmed*` et `*unconfirmed*`, où `*unconfirmed*` sont les tracks dans l'état soit `*tentative*` soit `*deleted*`. La fonction calcule alors la matrice de coût entre les tracks et les détections et la fonction ***linear_assignment*** de ***sklearn*** réalise le matching avec l'algorithme hongrois. Le *IoU* matching est ensuite appliqué sur les tracks non-confirmés et les confirmés non-associés lors de la cascade. Finalement, la fonction met à jour la liste des tracks en mettant à jour la moyenne et la variance des filtres de Kalman associés, en ajoutant le descripteur des détections associées aux descripteurs des détections précédentes des mêmes tracks et en mettant à jour les états des tracks (de `*tentative*` à `*confirmed*` si besoin). La fonction met dans l'état `*deleted*` les tracks `*tentative*` non associés sur cette frame et les tracks dont la dernière association est plus ancienne que ***max_age***. La fonction finit par initialiser de nouveaux tracks `*tentative*` pour les détections non associées.

J'ai légèrement modifié les classes **Track** et **Tracker** pour associer à chaque track une couleur qui sera utilisée pour le dessin des boites. Après avoir appliqué ***predict*** et ***update*** la fonction ***perform_track*** met à jour l'objet **Individuals** reçu en entrée en associant à chaque détection l'**ID** et le **couleur** du track auquel elle a été associée. Finalement, la fonction retourne l'objet **Individuals** mis à jour.

Enfin, les fonctions ***draw_without_tracking*** et ***draw_with_tracking*** dessinent sur chaque frame :

- Un *cadre* autour de chaque individu détecté (+ le *cadre prédit* dans le cas avec tracking) ;
- Un *caption* avec son score et son ID, pour pouvoir suivre le tracking le cas échéant ;
- Un *mask* détourant l'individu détecté et un contour du mask est appliqué ;

Tout cela est fait dans une même couleur, associé à l'identifiant du track, afin de suivre au mieux le tracking.

Tous le code implémenté a été associé d'une documentation complète dans un **docstring**. Toutes ces fonctions et classes sont implémentés dans le Notebook associé à ce rapport : seules les fonctions ***update_results***, ***draw_without_tracking, draw_with_tracking, convert_roi_shape, candidates, crop_box*** et ***display_crops*** ont été implémentées dans un fichier à part ***utils.py***. Ces trois dernières fonctions seront expliquées dans la dernière partie de ce rapport.

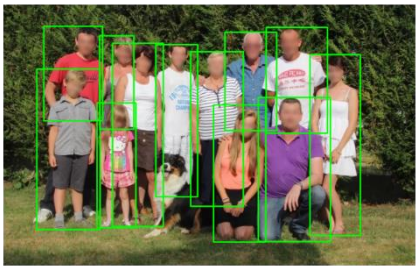
4.4. Méthodologie d'étude et analyse qualitative sur des documents tests

4.4.1. Documents utilisés

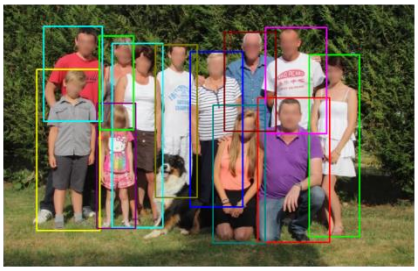
Tout au long de l'implémentation, j'ai mené des tests en utilisant **trois** documents personnels différents.

- Une **photo** de famille pour faire des tests de détection et dessiner les boites, captions, masques et contours de masques. Cette photo est intéressante car il y a 12 individus différents, ils se superposent par endroit et il y a un autre « objet » (un chien), donc non-humain.

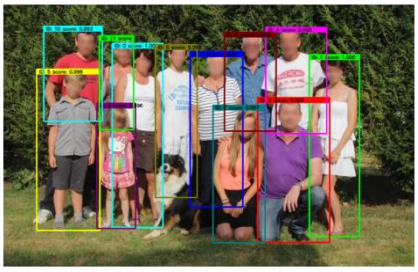
Image d'origine



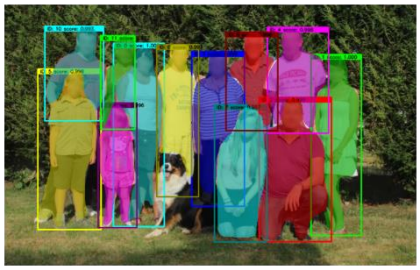
Avec boites englobantes



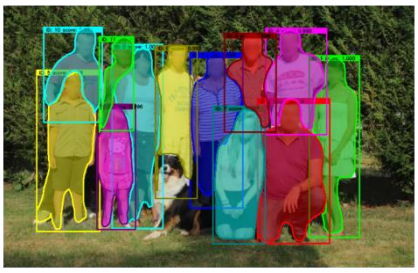
Avec boites en couleurs différentes



Avec captions



Avec masques



Avec contours de masques

Cette image m'a permis dans un premier temps de tester l'implémentation utilisée de *Mask R-CNN* puis d'écrire et vérifier mes autres fonctions en utilisant la méthode du **test unitaire**. C'est-à-dire que cette simple photo m'a permis de tester chaque ligne de code, de voir ses effets et de corriger les éventuels problèmes. Plus concrètement, j'ai d'abord pu écrire et tester ma fonction ***update_result()*** et vérifier que l'objet en sortie ne contenait que des individus, et autant d'individus que de personnes sur l'image. J'ai ensuite pu écrire les fonctions de dessin qui utile des méthodes *OpenCV* pour dessiner des boites englobantes, pour écrire des captions (avec background pour être plus visible), des masques en transparence et enfin un contour aux masques. C'est cette dernière fonctionnalité qui m'a donnée le plus de difficultés.

- Une courte **vidéo** *video_test1.mp4* de moi-même. *Durée : 14s. Résolution : 1920x1080. Fréquence d'images : 15 FPS.* Elle est particulièrement utile car je suis le seul individu dessus. Il y a deux occlusions.

- Une courte **vidéo** *video_test2.mp4* de quelques piétons prises depuis ma fenêtre. *Durée : 20s. Résolution : 1920x1080. Fréquence d'images : 30 FPS.* Il y a plusieurs individus dessus, très petits et quelques d'occlusions. Elle est particulièrement intéressante pour analyser les limites de notre système.

Ces deux vidéos m'ont permis dans un premier temps de faire les mêmes tests qu'avec la photo, mais en parcourant une vidéo frame par frame, puis en sauvegardant les images de sorties dans une autre vidéo afin de pouvoir visualiser les résultats.

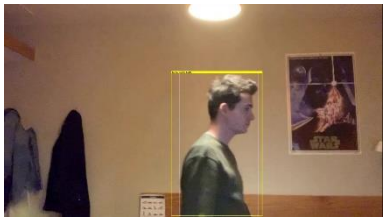
Dans un second temps, ces deux vidéos m'ont permis de faire d'autres tests vidéo sur mes fonctions pour le tracking, et notamment la méthode ***perform_track()*** de ma classe DeepSORT. Pour rappel, il s'agit de la fonction récupérant, pour chaque frame, l'objet Individuals renvoyé par le détecteur et effectue le tracking pour mettre à jour l'objet Individuals avec les identifiants des individus détectés.

4.4.2. Analyse de la qualité de la détection et du tracking

Ces courtes vidéos m'ont permis de faire quelques tests sur les performances des différents étages du système de tracking. Tout d'abord, on observe **visuellement** un résultat très correct. Les détections sont justes et il y a très **peu d'erreurs**. Lorsque c'est le cas, c'est sur des **objets inanimés ayant une forme humaine** tels qu'un manteau suspendu ou un poster avec des individus comme le montre la seconde image ci-dessous.


Lorsque de tels objets sont présents (c'est le cas sur *video_test1.mp4* notamment), il m'a suffi d'**augmenter le seuil de détection à 0.97** pour corriger ce problème. Je peux alors visualiser l'efficacité du tracking même avec un unique individu. En effet, **malgré les quelques défauts de détections restants, l'ID de l'individu sur la photo reste à 1, et ce malgré une courte période d'occlusion en début de vidéo** (*i.e.* sortie du champ de l'image). Cependant, le track est perdu sans raison apparente à la 7ème seconde la vidéo : l'ID passe de **1** à **9**. Lors d'une seconde période d'occlusion en fin de vidéo, plus longue cette fois-ci. L'ID passe alors à de **9** à **12** pour l'individu. C'est très intéressant car cela également nous donne une information sur la qualité de la détection : en effet il n'y a eu que 9 détections « autres » que celle de l'individu sur l'ensemble des frames (12 IDs différents moins le 1, le 9 et le 12) et 2 mauvaises associations.

En effectuant un autre test, en augmentant à **0.98** le **seuil**, de 30 frames à **60** frames la **mémoire du tracking** et de 5 à **10** le nombre de détections minimales consécutives $\boldsymbol{n_{init}}$ pour créer un track on obtient un résultat parfait. En effet, la nouvelle valeur de $\boldsymbol{n_{init}}$ permet d'éviter de créer un track seulement pour quelques mauvaises détections consécutives dû au manteau ou au poster.


*Malgré une mauvaise détection sur l'image 1 (rectangle **blanc** en bas à gauche), aucun track n'est créé. On constate bien sur les dernières frames (image 3) que le track de l'unique individu est toujours maintenu à l'ID **1**.*

Evidemment, ces paramètres sont **propres à cette vidéo** qui correspond à **une situation particulière**. Les **paramètres** (seuil de détection, durée de mémoire, nombre de détections consécutives) sont donc **à ajuster en fonction de la situation** (nombre d'individus, tailles de ceux-ci, objets pouvant être mal classifiés par le détecteur, nombre et durée des périodes d'occlusion, etc.).

Pour illustrer cela, des tests similaires ont été effectués sur l'autre vidéo test.

Cette vidéo a particulièrement été choisie afin d'illustrer les limites de ces algorithmes. On constate, même en diminuant le seuil de détection et en augmentant la mémoire du tracking, que celui-ci, bien que néanmoins correct, n'est pas optimal. Ce qui est mis en avant avec cette vidéo, c'est surtout **l'influence de l'étage de détection.** En effet, sur ces images la dizaine d'individus présents sont **très petits** et se **superposent**. Cela perturbe le détecteur, alors incapable de détecter certains individus sur l'ensemble des frames, où bien mélangeant quelques individus très proches l'un de l'autre dans une seule fenêtre de track.

Le système devient trop instable lorsque les détections sont trop petites et les occlusions nombreuses. En changeant les paramètres avec un seuil de 0.6, un max_{age} de 60 et un n_{init} de 10 on obtient un résultat correct, mais loin d'être acceptable.

4.4.3. Analyse du temps d'exécution

On se rend compte que le temps d'exécution global est moyen. Par moyen, je veux dire qu'il est raisonnablement rapide, mais loin de l'être assez pour effectuer un tracking en temps réel. Analysons quelle partie de l'algorithme consomme le plus de temps de calcul en observant les temps d'exécution sur *Google Colaboratory* :

	Vidéo de test 1	*Vidéo de test 2*
Part du temps de *détection* moyen	*80.01%*	*44.36%*
Part du temps de *tracking* moyen	*1.52%*	*2.46%*
Part du temps de *dessin* moyen	*12.10%*	*49.36%*
Fréquence d'exécution moyenne	**1,86 images par seconde**	**0,96 images par seconde**

NB : la somme ne fait pas 100% puisqu'il y a d'autres opérations mineures non prises en compte ici, telle que la lecture des frames ou l'écriture de celles-ci.

Comme on peut le voir, **le tracking est très performant** sur ce critère et ne représente qu'une part très faible du temps de calcul total, même lorsque le nombre de tracks augmente comme avec la seconde vidéo. La plupart du temps de calcul est donc partagé entre le tracking et la fonction de dessin. On remarque que la fonction de dessin, sur laquelle nous pouvons agir, représente une part non-négligeable du temps de calcul. De plus, cette part augmente considérablement à mesure que le nombre d'individus à représenter augmente.

Pour rappel, la fonction ***draw_with_tracking()*** effectue une copie de l'image d'entrée et **quatre** opérations par détection sur celle-ci : dessin de deux *boites englobantes*, écriture d'une *caption* dans une autre boite, dessin d'un *masque transparent*, dessin d'un *contour* à ce masque. Les temps pris par chacune de ces opérations dans le cas d'*une image* avec 12 *détections* sont indiqués dans le tableau ci-dessous et montrent que la quasi-intégralité ($\approx 97\%$) du temps de calcul provient du dessin des masques et de leur contours.

	Image test
Temps copie	*1.00%*
Somme temps boites	*0.09%*
Somme temps captions	*0.09%*
Somme temps masques	***47.02%***
Somme temps contours	***51.52%***

Nous pouvons en conclure que l'on ne peut pas à la fois être performant sur les temps de calculs et générer un tracking « ***esthétique*** ». *Mask-RCNN* est donc une méthode efficace pour obtenir des masques. Cependant, si ceux-ci ne sont pas utilisés par l'étage de tracking (c'est notre cas ici : le tracking utilise uniquement les coordonnées des boites englobantes et des features qui y sont issus), ils n'ont alors qu'un intérêt esthétique et ne font que ralentir le tracking. Deux solutions sont alors possibles :

- Enregistrer toutes les caractéristiques de chaque individu sur chaque frame d'une vidéo et faire le dessin en offline. Cette solution n'est pas efficace car elle demande beaucoup de mémoire.
- Abandonner l'affichage des masques si le but est uniquement le tracking le plus rapide possible.

Dans la suite de cette étude, nous nous placerons dans le second cas. Nous mettrons donc les paramètres **show_masks** et **show_masks_contour** de ***show_with_tracking()*** sur **False**. Nous obtenons alors les résultats suivants :

	Avec masques		***Sans masques***	
	Vidéo test 1	***Vidéo test 2***	***Vidéo test 1***	***Vidéo test 2***
Détection	*80.01%*	*44.36%*	*87.96%*	*83.83%*
Tracking	*1.52%*	*2.46%*	*1.77%*	*4.67%*
Dessin	*12.10%*	*49.36%*	*1.38%*	*0.68%*
TOTAL	*93.63%*	*96.18%*	*91.11%*	*89.18%*
Fréquence d'exécution	**1,86 fps**	**0,96 fps**	**2,01 fps**	**1,78 fps**

NB : la somme ne fait pas 100% puisqu'il y a d'autres opérations mineures non prises en compte ici, telle que la lecture des frames ou l'écriture de celles-ci.

Nous remarquons que le temps d'exécution a presque été divisé par deux lors du traitement de la seconde vidéo. Ce gain de temps est donc important lorsqu'il y a plusieurs détections, mais il ne suffit pas à faire du live tracking. L'essentiel ($\approx 85\%$) du temps de calcul est désormais concentré dans la détection.

4.5. Optimisation et résultats sur un dataset de tracking d'individus

Afin d'optimiser l'algorithme, divers tests sont effectués sur le dataset « *EPFL : Muti-camera pedestrians videos* » **[20]** et plus particulièrement sur les vidéos issues du sous-set *Laboratory Sequences* avec quatre caméras et six individus différents : *6p-c0.avi, 6p-c1.avi, 6p-c2.avi, 6p-c1.avi*. Etant donné qu'aucune vérité terrain n'est fourni, les résultats présentés ici seront principalement qualitatifs. Chaque fichier vidéo étudié dure *1min58*, la résolution est de *360x288* pixels et la fréquence de *25* FPS.

Sur la figure ci-dessous quelques exemples de détections. Celles-ci sont globalement très bonnes malgré de nombreuses occlusions, on constate tout de même certaines (rares) non-détections comme sur la troisième image où un individu est non détecté (au fond) et un autre détecté mais sans track confirmé (rectangle blanc).

*Les rectangles blancs sont les **détections**, c'est-à-dire les boites englobantes prédites par Mask R-CNN. Les rectangles colorés sont les boites englobantes des **tracks**, c'est-à-dire celles prédites par le filtre de Kalman du track associé.*

Le tableau suivant montre les temps d'exécution pour chaque vidéo, effectué sur **Google Colaboratory :**

	VIDEO 1	VIDEO 2	VIDEO 3	VIDEO 4	**TOTAL**
Détection	94,21%	93,97%	94,79%	94,89%	$\mathbf{94,44\%}$
Tracking	4,36%	4,48%	3,63%	3,56%	$\mathbf{4,01\%}$
Dessin	0,15%	0,16%	0,15%	0,15%	$\mathbf{0,15\%}$
Fréquence	2,92 FPS	2,98 FPS	3,02 FPS	3,03 FPS	$\mathbf{2,99\ FPS}$

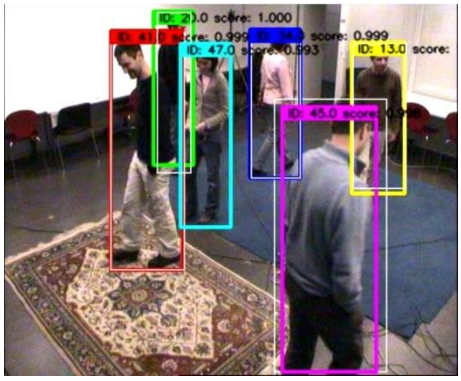
Le temps d'exécution total pour les 4 vidéos est de **65 minutes et 58 secondes**. L'expérience a été réalisée avec les paramètres suivants sur *Google Colaboratory* :

Paramètres	Valeurs	Commentaires
max_{age}	50	*Une mémoire de **2** secondes*
n_{init}	5	*Eviter de créer des tracks pour quelques mauvaises détections consécutives*
$max_{cosine\_distance}$	0.2	
$max_{iou\_distance}$	0.7	
$threshold_{detection}$	0.95	*Compromis entre la qualité de la détection et les erreurs de détections*

On remarque encore une fois la performance de *DeepSORT*, que cela soit en temps de calcul (environ 3% du temps d'exécution total) ou en qualité du tracking. Bien que les personnes soient relativement similaires (même taille sur l'image, arrière-plan et vêtement similaire pour plusieurs d'entre eux), elles sont assez bien suivies et le nombre de switch d'ID reste faible.

*On voit bien sur ces images espacées d'une quinzaine de secondes – entre lesquelles un nouvel individu est apparu – que les identifiants **2, 7** et **8** ont été conservées.*

Cependant, lorsque l'on compare des instants éloignés de la séquence, on se rend compte d'un important accroissement du nombre d'ID. Cela est dû aux nombreuses et récurrentes occultations du mouvement circulaires effectués par les différents individus. Ainsi, lorsqu'un track est perdu à la suite d'une occultation trop longue, un nouvel identifiant lui est associé. Insistons bien sur la différence entre deux problèmes : celui des ***switches***, c'est-à-dire l'incapacité du traqueur à différencier les individus au cours du temps (ce problème, bien que présent, est relativement faible ici) et celui des ***occlusions***, c'est-à-dire la multiplication des tracks pour un même individus à la suite de trop longues disparitions.



Il y a plusieurs façons d'influencer l'algorithme pour diminuer au mieux ces problèmes.

- **Accroitre $max_{cosine\_distance}$** : cela permet de limiter les non-associations de track et détections dénotant pourtant un même individu en abaissant notre exigence de similarité. En effet, les vecteurs issus du descripteur d'apparence d'un même individu lorsque celui-ci est occulté et ne l'est pas sont sensiblement différents. Il y a un compromis à trouver puisqu'évidemment une augmentation de $max_{cosine\_distance}$ s'accompagnera nécessairement d'un accroissement du nombre de switches d'identifiants.
- **Accroitre n_{init}, max_{age} et $max_{iou\_distance}$** *:* cela peut permettre d'augmenter la probabilité de retrouver un individu suite à une occultation sans pour autant créer de nouveaux tracks. Augmenter $max_{iou\_distance}$ permet également d'augmenter cette probabilité en évitant qu'une bonne association track-détection post-occultation ne soit pas trouvée à cause de ce critère. Il y a toujours un compromis à trouver.

En utilisant les valeurs ci-dessous, on obtient un résultat plus correct. On constate même que le nombre de switches s'est raréfié du fait que les tracks retrouvent plus souvent leur vrai individu suite à une occultation.

Paramètres	Valeurs	Commentaires
max_{age}	100	*Une mémoire de **4** secondes*
n_{init}	10	*Eviter de créer des tracks pour quelques mauvaises détections consécutives*
$max_{cosine\_distance}$	0.4	
$max_{iou\_distance}$	1	
$threshold_{detection}$	0.95	*Compromis entre la qualité de la détection et les erreurs de détections*

4.6. Cas du tracking multi-vidéos

Il y a deux cas à distinguer :

- Le cas où les caméras se superposent, c'est-à-dire filment la même scène mais d'un point de vue différent.
- Le cas où les caméras filment des endroits différents, où les mêmes individus peuvent passer.

Dans le cadre de ce projet, je me place dans le premier cas et en particulier dans le cas particulier du dataset sur lequel nous avons jusqu'à présent travaillé, c'est-à-dire un environnement de 4 caméras filmant 6 personnes dans un même endroit depuis un point de vue différent. Les méthodes mises en place pouvant cependant être utilisées dans les deux cas de figure, ce qui n'est pas le cas de toutes les méthodes.

4.6.1. Test « naïf »

La première chose que j'ai fait fut d'effectuer une méthode « naïve ». C'est-à-dire que je prenais pour hypothèse que mon tracking était parfait, et que les tracks de fin de vidéo étaient les mêmes que ceux de début de vidéo. Ainsi, j'ai tenté d'effectuer un « tracking bout à bout ». Pour cela, je n'ai utilisé qu'un seul traqueur que j'ai appliqué consécutivement aux quatre vidéos sans réinitialisation des paramètres (dont les tracks), avec une mémoire max_{age} très élevé (1000, correspondant à 40 secondes) pour être certain de récupérer les individus entre la fin d'une vidéo et leur première apparition dans la suivante.

Cette méthode donne évidement de très mauvais résultats et n'est pas à recommander.

4.6.2. Re-Identification et *N*-plus-proche-voisins à partir des features de DeepSORT

J'ai décidé d'utiliser une méthode de Re-Identification afin de proposer pour chaque identifiant nous intéressant (que nous appellerons ***query*** dans la suite) la liste de N IDs les plus similaires dans chacune des autres vidéos.

Pour cela j'ai décidé d'utiliser directement les features des détections générées par DeepSORT. En effet, puisque ces features sont encodés avec un descripteur d'apparence entrainé sur une tâche de Re-Identification, ils sont d'ores et déjà parfaitement adaptés à notre problème ! De plus, ils sont déjà calculés, il n'y a qu'à les récupérer.

J'ai donc ajouté un attribut à la classe **Track** de DeepSORT, *self.last_feature*, qui va me permettre, à chaque frame, de récupérer les features des détections associés aux tracks mis à jour. Afin de ne pas récupérer de track inutile, je ne récupère uniquement les features lorsque l'état du track est confirmé, c'est-à-dire qu'il a été correctement associé à une détection $\boldsymbol{n_{init}}$ fois de suite.

Une nouvelle classe **MultiTrack** a donc été créée, très similaire à la classe **Track** précédente mais cette fois-ci on crée également un dictionnaire ***features*** dont les clés sont les **ID** de tous les tracks ayant atteint l'état `*confirmed*` et les valeurs la **liste des features** de toutes les détections associées. Une fois ce dictionnaire récupéré, on répète cette opération pour chaque caméra. On diminue le nombre d'éléments de ces dictionnaires en retirant tout identifiant contenant moins de $3 * frame\ rate$ éléments : autrement dit, en retirant tous les tracks ayant existé moins de $3\ secondes$ afin de ne prendre en compte seulement les tracks significatifs. Le **vecteur moyen** des features pour chaque identifiant est alors calculé.

On dispose alors d'un vecteur représentatif de chaque track de chaque vidéo. Chaque vecteur étant identifié par un couple *(identifiant, caméra)* unique. L'idée est donc de calculer une distance entre chacun de ces identifiants. Ainsi, lorsque l'on cherchera les IDs candidats de ceux dénotant le même individu que celui qui nous intéresse, on choisira ceux dont la distance est la plus faible. Comme pour DeepSORT, j'ai choisi d'utiliser la distance cosinus puisqu'il s'agit notamment de la distance sur laquelle l'encoder a été entrainé.

Ainsi, chaque vecteur moyen d'une vidéo est comparé à tous les vecteurs moyens issus des autres vidéos à l'aide de la distance cosinus. Les listes alors obtenus sont triées et on peut retourner, pour chaque *query*, une liste des N identifiants les plus similaires pour une caméra donnée en appelant la fonction ***candidates()***. Cette fonction prend en argument la matrice des distances, deux dictionnaires pour récupérer le couple (cam, ID) d'un track à partir de son indice dans la matrice et inversement, le couple query (cam, ID), le nombre N de candidats à retourner, la caméra dans laquelle on recherche les candidats. Cette fonction a été implémentée dans le fichier *functions.py.* Le code est entièrement adapté pour pouvoir être testé sur d'autres datasets multi-caméras.

On observe alors, par exemples, les résultats suivants :

```
        ##### QUERY #####

Camera:0 ; ID:3

        ##### CANDIDATES #####

ID candidates for cam 1: 52 15 30 50 31
ID candidates for cam 2: 43 4 30 46 11
ID candidates for cam 3: 9 3 1 6 15
```

$QUERY$: $caméra\ 0$; $ID\ 3$.

$Caméra\ 1$; $IDs\ 52, 15, 20, 50, 21.$

$Caméra\ 3$; $IDs\ 9, 3, 1, 6, 15.$

Les images ci-dessous sont des captures d'écrans des identifiants retournés pour les caméras 1 et 3 lorsque l'on cherche des candidats pour la *query* $(cam = 0\ ; ID = 3)$. La vidéo 2 n'est pas considérée ici car son point de vue cache le haut des boites englobantes, on ne peut donc pas visuellement observer les ID. On observe un résultat très correct pour la vidéo 1 et un peu moins bon pour la vidéo 3.

Cependant, il y a un problème non pris en compte ici : les switches ! En effet, un identifiant peu dénoter plusieurs individus différents. En effectuant une capture d'écran à un moment précis de la vidéo, je choisis arbitrairement quel individu associé à l'identifiant. Afin d'associer de façon plus rigoureuse une personne à un identifiant, je détermine l'*individu moyen* d'un track et je conserve uniquement l'individu qui en est le plus proche.

Concrètement, j'ai ajouté un nouvel attribut à la classe **Track** : *self.last_crop.* Il s'agit d'une image : la frame associée au track qui a été rognée pour ne contenir que le contenu de la fenêtre englobante associée grâce à la fonction ***crop_box()*** également implémentée dans le fichier *functions.py.* La classe **MultiTrack** est également modifiée en créant un dictionnaire ***crops*** dont les clés sont les **ID** e tous les tracks ayant atteint l'état `*confirmed*` et les valeurs la **liste des images rognées** de toutes les détections associées. On ne conserve que les images ayant une longueur et une largeur supérieures à la moitié de la plus grande longueur et de la plus grande largeur des autres images de l'ID associé. Ces images sont utilisées pour calculer une *image moyenne* par ID. Finalement, on ne conserve pour chaque ID que l'image rognée la plus proche (en calculant une différence) de sa moyenne associée.

On dispose alors d'une image d'un individu unique pour chaque track de chaque vidéo. L'idée est donc de calculer une distance entre chacun de ces identifiants. Maintenant, il faut afficher cette image lorsque l'ID correspondant est renvoyé par la fonction *candidates()***.**

Cela est fait grâce à la fonction ***display_crops()*** qui prend en argument la liste des images rognées correspondant aux candidats puis les affiche.

On observe alors, par exemples, les résultats suivants :

$QUERY$: $caméra\ 0$; $ID\ 3$.

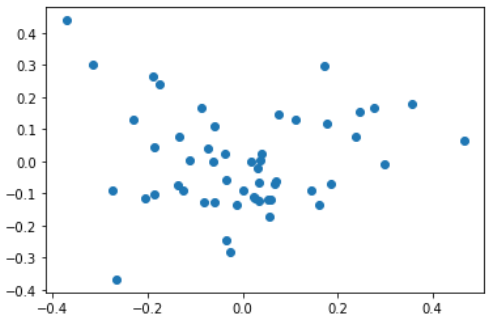
$Caméra\ 1$; $IDs\ 52, 15, 30, 50, 31$.

$Caméra\ 2$; $IDs\ 43, 4, 30, 46, 11$.

$Caméra\ 3$; $IDs\ 9, 3, 1, 6, 15$.

On remarque en moyenne des résultats corrects avec, entre la *query* et les 5 plus proches ID de la caméra 1, 4 bonnes associations. On en observe seulement 3 pour la caméra 2 et seulement 2 pour la caméra 3. Ces résultats ne sont pas renversant mais constitue une approche intéressante du tracking multi-caméras. Notons que la re-identification sur l'ID le plus ressemblant (la première image) est toujours correcte.

Finalement, une manière d'observer la performance du tracking et de la Re-Identification multi-caméras est de visualiser chaque feature moyenne associée à un ID dans un espace de dimension 2 grâce à l'algorithme du ***PCA*** (*Principal Component Analysis*). On devrait théoriquement distinguer 6 clusters étant donné que l'on a 6 individus distincts sur les 4 vidéos. Comme nous pouvons le voir sur la figure ci-contre, ce n'est pas le cas. On peut observer quelques groupement très localisés de points mais pas de clusters globaux. Cela est probablement dû au fait que les individus soient assez similaires et que l'on a finalement une représentation assez bruitées puisqu'il s'agit de moyennes de toutes les features avec des individus se tournant et retournant, se superposant parfois.



5. CONCLUSION

Le tracking d'individus est un sujet vaste et un domaine particulièrement populaire depuis l'essor du Deep Learning. Les opportunités données par les récentes évolutions des capacités de calculs et des algorithmes sont nombreuses, avec notamment la possibilité nouvelle d'un tracking en direct dans un environnement multi-caméras avec des résultats de qualité. Cependant, comme nous l'avons montré, certains obstacles sont encore à surmonter comme la vitesse de détection, la capacité à gérer les occlusions ainsi qu'à différencier des individus très similaires. De nouvelles innovations émergent chaque année, comme les récents *Recurrent R-CNN* ou l'application de réseau *transformers* comme avec le *DETR* de Facebook. L'amélioration permanente des performances de ces algorithmes, que ce soit en qualité de détection et de suivi ou en temps d'exécution, ne doit pas faire oublier les nombreux enjeux éthiques soulevés par ce genre de technologie et doit nous pousser à constamment prendre du recul pour mieux en appréhender les usages. Après tout, « un grand pouvoir implique de grandes responsabilités ».

Concernant le projet lui-même, il m'a permis de découvrir le domaine de la Computer Vision, de développer une compréhension fine des algorithmes étudiés et d'effectuer mes propres analyses afin d'en évaluer la qualité. Ma méthodologie a constamment évolué au cours du projet. Celui-ci étant très libre, il y a eu beaucoup d'égarement au début, que ce soit sur les méthodes à implémenter, sur l'implémentation elle-même (complètes ou non) ou la simple compréhension de ce domaine de recherche qui m'était alors totalement inconnu. J'ai pu affiner mes recherches et ma compréhension pour finalement arriver à une structure cohérente par rapport au problème posé : un tracking par détection à partir d'implémentations de Mask R-CNN et DeepSORT, tout en étudiant l'aspect multi-caméras.

6. BIBLIOGRAPHIE

[1] RCNN - https://arxiv.org/pdf/1311.2524.pdf, *Ross Girshick et Al. – UC Berkeley*

[2] Selective Search Algorithm for Object Recognition – https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf, *J.R.R. Uijlings et Al. – University of Trento, University of Amsterdam.*

[3] Fast RCNN - https://arxiv.org/pdf/1504.08083.pdf, *Ross Girshick – Microsoft Research*

[4] Faster RCNN - https://arxiv.org/pdf/1506.01497.pdf, *Shaoquing Ren et Al. – Microsoft Research*

[5] R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms – https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, *Rohith Gandhi – Towards Data Science*

[6] Region of Interest Pooling - https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af, *Sambasivarao. K*

[7] NMS - https://github.com/opencv/opencv/blob/master/modules/dnn/src/nms.inl.hpp - *OpenCV*

[8] SPP - https://arxiv.org/pdf/1406.4729.pdf - *Shaoquing Ren et Al.*

[9] Mask RCNN - https://arxiv.org/pdf/1703.06870.pdf, *Ross Girshick et Al. – Facebook AI Research*

[10] Mask RCNN slides - https://www.slideshare.net/windmdk/mask-rcnn, *Ross Girshick et Al. – Facebook AI Research*

[11] Detection&Segmentation - http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf, *Fei-Fei Li et Al. – Stanford University*

[12] maskrcnn-benchmark - https://github.com/facebookresearch/maskrcnn-benchmark/, *Facebook Research*

[13] MaskRCNN for Object Detection & Segmentation - https://github.com/matterport/Mask_RCNN, *Waleed Abdulla – Matterport*

[14] Simple Online and Realtime Tracking - https://arxiv.org/pdf/1602.00763.pdf, *Alex Bewley et Al.*

[15] Simple Online and Realtime Tracking with a deep association metric - https://arxiv.org/pdf/1703.07402.pdf, *Nicolai Wojke, Alex Bewley et Al.*

[16] DeepSORT implementation - https://github.com/nwojke/deep_sort, *Nicolai Wojke*

[17] Filtre de Kalman - https://fr.wikipedia.org/wiki/Filtre_de_Kalman, *Wikipédia*

[18] Algorithme hongrois - https://fr.wikipedia.org/wiki/Algorithme_hongrois, *Wikipédia*

[19] IoU matching - https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[20] EPFL dataset - https://www.epfl.ch/labs/cvlab/research/research-surv/research-body-surv-index-php/ https://arxiv.org/pdf/1812.00442.pdf