

Amélioration du noyau des systèmes siamois pour le suivi d'objets

Paulin Brissonneau

CentraleSupélec

paulin.brissonneau@student-cs.fr

Abstract

Les avancées récentes en suivi d'objet dans les vidéos tirent leurs profits de métriques de similitude approximées par des réseaux de neurones profonds. On s'intéresse en particulier au siamFC [1], méthode à base de réseaux siamois qui utilise des extracteurs de caractéristiques pour comparer, à chaque instant de la vidéo, l'image de la cible à suivre et l'image de recherche. Dans sa forme originale, le siamFC utilise une architecture de type AlexNet à cinq couches ainsi que une seule et unique image de la cible fournie avant le début du suivi qu'il cherche ensuite dans chaque image de la séquence vidéo. Dans ce rapport, nous montrons que l'architecture de référence du siamFC peut-être légèrement améliorée, puis nous montrons qu'il est aussi possible d'améliorer les performances du suiveur d'objet en prenant en compte, dans les bonnes proportions, un historique glissant des apparences de l'objet à suivre.

Introduction

Le suivi d'objets est un domaine de la vision numérique qui consiste à suivre un ou plusieurs objets présents sur une vidéo. Pendant longtemps, l'état de l'art était tenu par des algorithmes purement statistiques, qui associaient des probabilités de correspondance entre les formes présentes sur deux images à la suite dans une vidéo. Depuis quelques années, le domaine du suivi d'objets a été drastiquement influencé par le développement de l'apprentissage profond. En quelques années, les algorithmes basés sur des métriques profondes ont détrôné tous les algorithmes classiques à l'état de l'art. L'utilisation de réseaux profonds permet une comparaison plus fine des différents objets et résiste mieux aux occlusions de longue durée grâce à une comparaison des objets à un haut niveau d'abstraction.

L'introduction des réseaux profonds en suivi d'objets prend plusieurs formes. Les méthodes basées sur les détections (*two stages trackers*) infèrent le suivi en deux étapes distinctes : d'abord une détection des objets présents sur l'image, puis une association des différents objets d'une image à la suivante. Cette façon de faire donne des résultats remarquables pour un temps de calcul des inférences court. Les algorithmes SORT [2] puis sa variante à métrique profonde deepSORT [34] sont des représentants de cette famille.

Au contraire, les méthodes unifiées font toutes ces opérations

en une seule étape. De nombreux travaux ont tenté d'unifier ces deux étages avec différentes méthodes. Parmi les solutions unifiées, celles basées sur les réseaux siamois se sont emparées de l'état de l'art et montrent des performances très prometteuses [1, 36, 33].

Or, contrairement aux méthodes à détection, les méthodes à réseaux siamois dans leurs formes simples se contentent de chercher l'image initiale de la cible dans chacune des images de la vidéo. Ceci ne permet pas d'utiliser l'information relative à l'historique de l'apparence de la cible dans la séquence vidéo, comme le font structurellement les méthodes à détection. Cette absence d'historique a des avantages, notamment le fait de pouvoir suivre une cible sur des fragments de vidéos voire des images indépendantes. Cela apporte aussi des inconvénients, comme une moins bonne résistance aux changements d'apparence, et un risque accru de confusion entre plusieurs instances semblables d'un même objet dans une vidéo.

C'est pour cette raison qu'il est intéressant d'étudier l'intérêt d'un système de stockage en mémoire des images de la cible tout au long de la séquence vidéo. Dans ce rapport, nous introduisons l'aspect de mémorisation en utilisant un *noyau glissant*, c'est-à-dire en gardant un historique des cartes de caractéristiques profondes extraites de la cible à chaque instant.

Nous étudions les comportements des systèmes siamois en général et ne cherchons pas à atteindre les performances de l'état de l'art. C'est pour cette raison que nous utiliserons l'architecture simple du siamFC [1], et non des architectures plus récentes et plus poussées qui présentent de meilleurs résultats. Grâce à une architecture de référence sobre, nous serons plus à même d'étudier en détail l'effet de nos modifications et d'interpréter nos résultats sans autres effets parasites que pourraient apporter des architectures plus complexes.

État de l'art

Réseaux Siamois

Les réseaux siamois sont une ancienne idée qui a fait son apparition en 1994 grâce à Bromley et al. [3]. Le principe des réseaux siamois est d'encoder de manière non-linéaire deux entrées différentes avec un même réseau extracteur de caractéristiques. L'objectif est que les cartes de caractéristiques obtenues soient porteuses de suffisamment d'information sémantique pour pouvoir comparer les données initialement données en entrée. À titre d'exemple, pour comparer deux

visages et dire s'ils appartiennent à la même personne, on encode chacun des deux visages par un seul et même réseaux, la comparaison des cartes de caractéristiques nous donnera une idée de la similarité des deux visages, non pas d'un point de vue photographique, mais d'un point de vue sémantique.

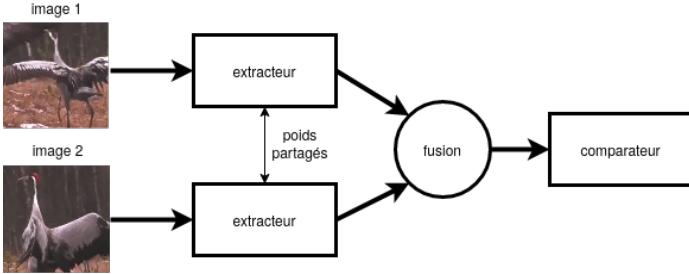


FIGURE 1. Structure générale des réseaux siamois.

Les réseaux siamois sont donc constitués de principalement deux étages, plus ou moins séparés selon les applications. La structure globale d'un réseau siamois est explicité sur la figure 1. Le premier étage est l'extracteur de caractéristiques qui sera le plus souvent, lorsque les données sont structurées et compositionnelles, un réseau convolutionnel. Le deuxième étage permet la comparaison des caractéristiques extraites et peut prendre plusieurs formes, allant de la simple distance vectorielle au réseau de neurones. Il y a parfois un étage intermédiaire permettant de fusionner les cartes de caractéristiques. Il peut s'agir par exemple d'une simple soustraction termes à termes ou d'une corrélation croisée (*cross-correlation*).

Les réseaux siamois ont été utilisés pour un très grand nombre de tâches, à savoir le *hashing* [23], le *clustering* [22] et surtout l'apprentissage sur peu d'exemples (*few-shot learning*) [16]. De par leur structure, ils sont aussi particulièrement adaptés à la comparaison de deux données, comme par exemple la vérification faciale [5], la localisation [30], la rénovation d'images [11], et bien sûr le suivi d'objets [1, 36, 33, 29].

Suivi d'objets

a. Approches classiques

Les premières approches pour le suivi *online* étaient basées sur des filtres de Kalman [18]. C'est un filtre récursif bayésien dont l'objectif est de calculer une distribution de probabilités pour la position de l'objet d'une image à la suivante dans la vidéo. Le filtre couple les informations venant de l'observation (supposée altérée par un bruit gaussien), et celles venant d'un modèle dynamique de l'objet qui prend typiquement la forme d'un modèle de vitesse constante pour chaque objet de l'image. Une autre méthode classique est le filtre à particules, qui approxime la distribution de probabilité de présence par la densité d'un groupe de particules [6].

Ces méthodes sont adaptées pour le suivi d'un seul objet sur une image (problème **SOT** pour *Single Object Tracking*), cependant pour suivre plusieurs objets il faut ajouter

une méthode d'association de **chaque objet à chaque objet** de l'image suivante. Le problème de suivi simultané de plusieurs objets est appelé **MOT** (pour *Multiple Object Tracking*). Les deux méthodes historiques pour le problème MOT sont **MHT** (*Multi-Hypothesis Tracker*) et **JPDA** (*Joint Probabilistic Data Association*).

Les méthodes **MHT** [4, 19] et **JPDA** [7, 27] calculent et stockent en mémoire toutes les associations possibles entre tous les objets de l'image et ceux de l'image suivante. Chaque probabilité d'association est raffinée en se basant sur les images suivantes. L'attente des images suivantes permet d'appréhender efficacement les occlusions puisque l'on peut attendre que l'objet réapparaisse. Cela apporte aussi son lot d'inconvénients, notamment, le suivi *online* est impossible et le coût calculatoire d'un tel algorithme est considérable.

Bien que ces méthodes en elles-même soient dépassées, elles sont importantes à mentionner car, comme on va le voir, certaines méthodes plus modernes leur font écho et les problèmes de coûts calculatoires dont elles souffrent sont, eux, toujours d'actualité.

b. Approches modernes

L'utilisation des réseaux de neurones pour le suivi d'objet apporte beaucoup d'avantages. Notamment cela permet de s'appuyer sur des métriques de similarité plus pertinentes entre les différents objets. Alors que les méthodes classiques se contentaient d'utiliser des modèles de mouvement et d'apparence superficiels et simplistes, les métriques profondes peuvent reconnaître des objets malgré des occlusions très longues, même dans le cas où l'apparence de l'objet a changé (un objet dont l'angle de vue a changé par exemple).

Une première façon de mettre à profit les réseaux de neurones pour le suivi a été de développer des méthodes à deux étages : d'abord une détection des objets présents sur l'image, puis une association des différents objets d'une image à la suivante. C'est ce qu'ont proposé Bewley et al. [2] avec l'algorithme **SORT**. Il repose sur l'idée très simple d'utiliser les détections d'un détecteur classique (par exemple YOLO [25] ou RCNN et ses variantes [10, 9, 26]) pour ensuite appliquer un filtre de Kalman qui se chargera d'associer les différents objets d'une *frame* à la suivante. Il s'inspire donc directement des méthodes classiques du paragraphe précédent.

Le filtre de Kalman impose intrinsèquement une hypothèse de vitesse constante pour chacun des objets à détecter, on comprend donc que cette méthode est limitée lorsque les objets détectés font des mouvements brusques comme c'est le cas quand la caméra bouge. Pour remédier à ce problème, Wojke et al. [34] ont proposé de coupler le **SORT** à une métrique profonde spécialement conçue pour comparer sémantiquement deux objets. Cette petite modification permet de raffiner la méthode d'association du **SORT** et d'associer des objets plus éloignés dans le temps et dans l'espace. **DeepSORT** donne des résultats remarquablement proches de l'état de l'art pour un coût calculatoire plus faible que toutes les autres méthodes.

Cependant les méthodes à deux étages ont une limitation intrinsèque : l'indépendance des deux étages provoque une redite des calculs d'extraction de caractéristiques. Les premières couches du premier étage ont un rôle très proche des couches

du deuxième étage. Les travaux de recherche se sont donc concentrés sur des méthodes unifiées, selon lesquelles un seul réseau doit pouvoir extraire un maximum d'information et doit pouvoir résoudre le problème de détection et de suivi en une seule inférence.

Étant donnée la nature récursive du problème, les études se sont concentrées sur l'entraînement de **réseaux récursifs** (RNN). Gan et al. [8] ont proposé d'utiliser un RNN pour prédire directement la position de l'objet traqué dans l'image. Cette méthode a été ensuite améliorée par Ebrahimi-Kahou et al. [17] par l'ajout de mécanisme d'attention pour donner le **RATM** (*Recurrent Attentive Tracking Model*).

En faisant le constat que ces méthodes à RNN étaient prometteuses sans pour autant montrer des résultats à l'état de l'art, de nombreuses équipes ont proposé de remplacer les réseaux récurrents par des réseaux siamois¹. Il n'est pas possible d'entraîner à partir de zéro un réseau pour chaque vidéo, il faut donc s'appuyer sur un apprentissage *offline* pour ensuite simplifier l'apprentissage *online* qui se fera sur la vidéo elle-même. C'est ce qu'ont proposé Wang et al. avec SO-DLT [32] et Nam et al. avec MDNet [24]. Cependant cela oblige à continuer l'apprentissage même pendant le suivi *online*, ce qui est coûteux en calcul et ne permet pas d'atteindre du temps-réel. D'autres travaux, comme Wang et al. avec le FCNT [31] ou encore Held et al. [13], ont tenté de complètement se passer d'apprentissage *online*, en se basant complètement sur les caractéristiques profondes extraites grâce à un réseau pré-entraîné complètement en *offline*. Ces méthodes ne sont pas complètement convolutionnelles, ce qui implique des réseaux toujours trop lourds pour des applications en temps réel en plus de nécessiter une augmentation importante des données pour forcer l'invariance à la translation. Bertinetto et al. [1] ont proposé **siamFC**, qui est un réseau siamois complètement convolutionnel (*Fully Convolutional Siamese Network*). Le **siamFC** est le premier réseau à proposer une inférence (*online*) en temps-réel tout en assurant des résultats remarquables. Cette méthode est le point de départ de nombreuses améliorations qui ont chacune leur tour donné des résultats à l'état de l'art en SOT (*Single Object Tracking*). En particulier **siamMask** (2018) [33] propose de coupler le suivi à une tâche de segmentation, **siamFc++** (2020) [36] entraîne une seule architecture unifiée à quatre tâches différentes : la classification, l'estimation du prochain état de l'objet, un suivi sans *prior*, et une estimation globale de la qualité du suivi. Les résultats du **siamFC++** sont aujourd'hui inégalés. Notons que les méthodes à réseaux siamois convolutifs donnent aussi des performances remarquables en MOT (*Multiple Objet Tracking*), puisque le **track-RCNN** (2020) [29], qui est un réseau siamois aidé d'un extracteur de RoI (*Region of Interest*) est actuellement l'algorithme le plus performant en MOT. Tous ces algorithmes infèrent (*online*) le suivi d'objets plus rapidement que les standards du temps réels, au prix d'un apprentissage *offline* très coûteux en terme de calculs.

¹. Notons que les réseaux siamois peuvent être assimilés à des réseaux récurrents manipulant des séquences de deux images.

Approche

Méthode de suivi

La méthode que l'on utilise est celle du **siamFC** proposée par L.Bertinetto et al. [1]. À chaque début de suivi, on calcule la carte de caractéristiques de l'image initiale de la cible à suivre, grâce à l'extracteur entraîné *offline*. **Dans la méthode de référence, cette image est fixe pendant toute la durée du suivi, on l'appelle le noyau (*kernel*) du système siamois.**

Pendant le suivi, à chaque étape, on découpe l'image de la vidéo (dans laquelle on cherche l'objet) autour de la position de la cible sur l'image précédente. En d'autre termes, on limite la recherche à une petite zone autour de la dernière position estimée de l'objet. Cela est équivalent modéliser la vitesse de la cible comme étant limitée, c'est une hypothèse classique pour les suiveurs. Dans les faits, on utilise plusieurs tailles différentes pour les zones de recherche, dans ce rapport, on utilise trois tailles différentes dépendantes de la taille de l'image initiale de la cible .

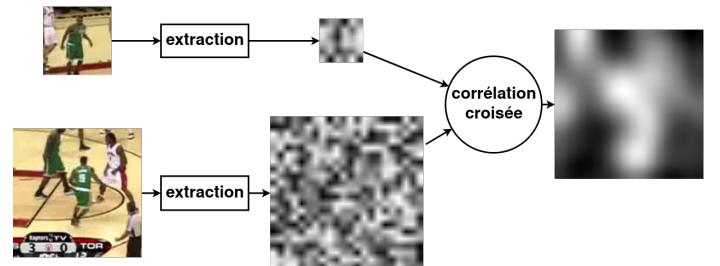


FIGURE 2. Structure du réseau siamois de l'approche du rapport. Sur la branche du haut, l'extracteur analyse l'image de initiale de la cible à suivre, en extrait les caractéristiques pour créer le *noyau*. Sur la branche du bas, l'extracteur analyse l'image de recherche. La tête du réseau à droite est une corrélation croisée qui génère les cartes de corrélations. Sur le schéma, la résolution de la carte de corrélation est augmentée.

Ensuite, on extrait la carte de caractérisitiques de l'image de recherche par le même réseau extracteur que pour le noyau. Une carte de corrélation est ensuite calculée par corrélation croisée entre le noyau du système et la carte de caractérisitiques de recherche. La figure 2 montre la structure générale pour le calcul des cartes de corrélations.

La position du pixel de plus haute intensité sur la carte de corrélation devient l'estimation du centre de la position de l'objet. Seule la position de l'objet est estimée et la taille des *bounding boxes* ne change que par la prise en compte de la taille la zone de recherche. L'algorithme 1 résume chacune des grandes étapes de la méthode de suivi. Cette méthode constitue une référence et nous en proposons une amélioration dans la partie *Amélioration du noyau*.

Ensembles de données

La série des bases de données **VOT** [20] font office de références dans le domaine du SOT (*Single Object Tracking*). Il s'agit d'un challenge proposé tous les ans sur un nouveau dataset créé pour l'occasion. Les banques **OTB-50** et **OTB-100** [35] font aussi office de norme. Cependant, ces banques de données n'ont pas pour vocation à servir à l'entraînement des

Algorithme 1 : Algorithme de suivi d'objets.

\mathcal{K} est le noyau. \mathcal{R} est l'image de recherche. \mathcal{F} est la carte de caractérisitiques. $crop_{\mathcal{I}}$ est la fonction qui sélectionne la zone de recherche. La valeur d'intérêt est c , le centre de l'objet prédit pour chaque étape.

```

initialiser le centre  $c$ ;
 $\mathcal{K} = extraction(cible)$ ;
pour image  $\mathcal{I}$  dans vidéo faire
     $\mathcal{R} = crop_{\mathcal{I}}(\mathcal{I}, c)$ ;
     $\mathcal{F} = extraction(\mathcal{R})$ ;
     $\mathcal{C} = correlation(\mathcal{K}, \mathcal{F})$ ;
     $c = argmax(\mathcal{C})$ ;
fin

```

modèles, mais à la comparaison des performances. Nous utiliserons la banque de données **GOT-10k** [14] pour l'entraînement ainsi que pour la validation, et nous utiliserons OTB-100 pour comparer les performances de nos modèles.

Métriques

Il existe différentes métriques pour mesurer les performances des modèles en suivi d'objets et elles sont différentes pour les tâches SOT et MOT. Nous baserons notre étude sur la dualité entre précision et taux de succès pour l'évaluation.

Le taux de succès (*succes rate*) consiste à calculer la proportion de temps pendant lequel l'intersection de la prédition et de la vérité étiquetée dépasse un certain seuil d'erreur admissible selon l'indice de Jaccard (*Intersection Over Union*). Un graphique donnera l'évolution de ce ratio en fonction du seuil d'erreur, celui-ci variant de 0 à 1. Notons que cette métrique mesure une performance proche de l'espérance du chevauchement moyen (*expected average overlap*), où le chevauchement est l'indice de Jaccard. C'est une autre métrique classique en VOT.

Cependant cette métrique ne permet pas d'avoir une bonne idée du comportement du suiveur, en effet chaque modèle prédit à la fois une région (*bounding box*) et le centre de l'objet, c'est pourquoi une métrique basée seulement sur la région proposée de permet pas de mesurer la précision du suiveur. La précision (*precision*) est mesurée par le pourcentage d'images pour lesquelles le centre de l'objet prédit et le vrai centre sont espacés d'une distance inférieure à un seuil donné. Là encore, un graphique donnera la précision en fonction de ce seuil.

Comme un graphique ne peut pas donner une méthode directe de comparaison des suiveurs, nous utiliserons l'aire sous la courbe de succès (*AUC*) pour classer les modèles.

Méthode d'apprentissage de l'extracteur

La seule partie apprenante du système est l'extracteur. Pour son apprentissage, nous construisons des paires aléatoires d'images du même objet séparées par un nombre aléatoire de *frames* dans la vidéo. Pour des raisons de cohérence de l'apparence des couples d'entraînement, les images ne peuvent pas être séparées de plus de 100 *frames*. De chaque couple, on extrait les cartes de caractérisitiques fournies par le modèle ex-

tracteur apprenant. La tête calcule la corrélation croisée de ces cartes. La carte de corrélation obtenue est alors comparée aux données étiquetée : les coordonnées réelles de l'objet cible dans l'image de recherche.

On ne peut pas directement construire une fonction de coût basée sur la comparaison de la carte corrélation (données matricielles) et les coordonnées de la cible étiquetées vectorielles (*bounding boxes*) puisqu'elle ne sont pas de même nature : on doit transformer le problème en un problème de classification.

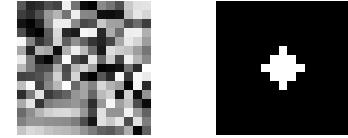


FIGURE 3. **Visualisation des entrées de la fonction de coût.** À gauche, une carte de corrélation prédictive. À droite, l'objectif auquel la comparer.

Pour cela on construit (à partir des coordonnées de la cible étiquetées) une carte de même dimension que la carte de corrélation, et telle qu'on associe aux pixels proches de la position de la cible (*connue*) une valeur $y_p = 1$, et aux pixels éloignés une valeur $y_p = 0$. La distance entre la position de la cible et les pixels négatifs est un hyper-paramètre du suiveur r_{neg} . Un exemple de carte ainsi construite est montré figure 3. Le problème revient à un problème de classification classique sur chacun des pixels. On utilise comme fonction de coût L une entropie croisée binaire sur chaque pixel de chaque image d'un batch :

$$l_p = y_p \cdot \log \sigma(x_p) + (1 - y_p) \cdot \log(1 - \sigma(x_p))$$

$$L = \sum_{\mathcal{I} \in batch} \sum_{p \in \mathcal{I}} l_p$$

Cependant, chaque carte de labels a un nombre très inégal d'instances positives et négatives puisque seule une petite zone de chaque image correspond effectivement à la présence de la cible. Un tel comportement peut avoir des effets néfastes déstabilisateurs sur l'apprentissage². Pour prendre cela en considération, on utilise un coût équilibré (*Class-Balanced Loss*) :

$$L = \sum_{\mathcal{I} \in batch} \sum_{p \in \mathcal{I}} w_p l_p$$

où

$$w_p = \frac{1}{\sum_{\mathcal{I} \in batch} p_{\{+/-\}}}$$

où $p_{\{+/-\}}$ est égal au nombre d'instances positives (*resp. négatives*) dans l'image si y_n est positive (*resp. négative*).

À l'exception des modifications de valeurs dans le cadre des expérimentations (qui seront mentionnées), on utilise les hyper-paramètres d'apprentissage proposés dans le papier original du

2. Le modèle aurait tendance à prédire qu'aucun des pixels ne correspond à la cible. Comme les pixels tels que $y_p = 0$ sont très majoritaires, la fonction de coût admettrait un minimum local pour la prédiction *facile* $\forall p \quad x_p = 0$, ce qui rendrait la convergence plus difficile.

siamFC [1]. Il s'agit en particulier : d'une taille de **mini-batch de 8 couples** de cartes de corrélations, une **régulation L_2 de 5e-4**, un **SGD de moment 0.9 et de taux d'apprentissage exponentiellement décroissant de 1e-3 à 1e-5**. Les calculs sont effectués sur GPU GeForce RTX3070, pour des temps de calcul de l'ordre de 10h pour 100 passes. Le nombre de passes pour un apprentissage est déterminé par *early stopping* et dépend des expérimentations.

Architecture de l'extracteur

L'architecture que nous utilisons en tant que référence est celle proposée par L.Bertinetto [1]. Dans cette architecture, chaque réseau extracteur de caractéristiques est construit sur la base des extracteurs AlexNet [21], auxquels on ajoute des couches de *Batch Normalization*. L'architecture de l'extracteur est proposée sur la figure 4. La tête de l'architecture est une corrélation croisée, et elle n'admet aucun paramètre apprenable.

Un des buts de ce travail est d'étudier la pertinence de cette architecture de référence. On se concentre en particulier sur l'utilisation des *Batch Normalization* et sur le nombre optimal de couches dans le réseau. Pour l'étude des *Batch Normalization* notre approche expérimentale sera de comparer l'apprentissage puis les performances du modèle avec et sans les *Batch Normalization*, tout en conservant l'architecture de référence en ce qui concerne les autres composants du réseau. Pour l'étude du nombre optimal de couches, on teste des architectures en ajoutant successivement des couches. Les différentes architectures testées sont présentées en annexe B.

Chacune des expérimentations se fait en optimisant les autres hyper-paramètres par recherche manuelle (la régulation L_2 , le moment, et le taux d'apprentissage) sur l'ensemble de validation. Chaque expérience est ré-itérée cinq fois, et les performances à comparer sont mesurées sur l'ensemble de test puisqu'il ne s'agit pas d'un réglage d'hyper-paramètre, et que l'on souhaite prendre en compte la meilleure estimation possible de l'erreur de généralisation.

Amélioration du noyau

La méthode de référence compare, à chaque instant, l'image de l'objet que l'on cherche à l'image dans laquelle on cherche l'objet. Or, cette méthode n'est pas optimale dans plusieurs cas, notamment lorsque que l'objet change d'apparence ou lorsque plusieurs objets sont proches et se ressemblent beaucoup. Ces situations seront discutées dans la partie *Étude des cas d'échecs*. Dans ce rapport, on propose d'étudier une amélioration qui consiste à ne pas seulement chercher l'image de la cible telle qu'elle a été donnée lors de l'initialisation de la séquence, mais de chercher aussi la cible telle qu'elle a été détectée durant la séquence.

Pour cela, on introduit un deuxième noyau, que l'on appellera *noyau glissant* (noté $\mathcal{K}_{glissant}$). Il s'agit d'un historique des apparences de l'objet-cible dans les dernières images analysées. La mise à jour de ce noyau est régie par un nouvel hyper-paramètre γ . En d'autres termes, le noyau glissant est équivalent à une moyenne mobile exponentielle des dernières cartes de caractéristiques associées à la cible. La mise à jour

sera de la forme :

$$\mathcal{K}_{glissant+} = \gamma(\mathcal{K}_{\mathcal{I}} - \mathcal{K}_{glissant})$$

où $\mathcal{K}_{\mathcal{I}}$ est la carte de caractéristiques calculée à partir de la dernière détection de l'objet.

Comme dans la méthode de référence, la position de l'objet est estimée par l'argument maximal des cartes de corrélations du noyaux et de l'image de recherche. Dans cette nouvelle méthode, il y a deux noyaux, donc deux cartes de corrélation, et donc deux vecteurs de coordonnées pour l'estimation de la position de l'objet (notés c_{ref} et $c_{glissant}$). L'estimation retenue pour le suivi sera une moyenne c pondérée par β des deux vecteurs :

$$c = \beta c_{ref} + (1 - \beta)c_{glissant}$$

Cette nouvelle méthode est résumée par l'algorithme 2. Dans la partie expérimentale, nous comparerons les résultats obtenus par la méthode de référence et la méthode modifiée. De la même manière que dans le paragraphe précédent, chacune des expérimentations se fait en optimisant les autres hyper-paramètres (la régulation L_2 , le moment, le taux d'apprentissage, et maintenant β ainsi que γ) par recherche manuelle sur l'ensemble de validation.

Algorithme 2 : Amélioration de l'algorithme 1, avec les mêmes notations. La fonction $crop_{\mathcal{K}}$ effectue la même action que $crop_{\mathcal{I}}$ mais sur le noyau au lieu des images.

Soient γ et β hyper-paramètres.

initialiser le centre c ;

$\mathcal{K} = extraction(cible)$;

$\mathcal{K}_{glissant} = \mathcal{K}$;

pour image \mathcal{I} dans vidéo faire

$\mathcal{R} = crop_{\mathcal{I}}(\mathcal{I}, c)$;

$\mathcal{F} = extraction(\mathcal{R})$;

$\mathcal{K}_{\mathcal{I}} = crop_{\mathcal{K}}(\mathcal{F})$;

$c_{ref} = correlation(\mathcal{K}, \mathcal{F})$;

$c_{glissant} = correlation(\mathcal{K}_{glissant}, \mathcal{F})$;

$c_{ref} = argmax(\mathcal{C}_{ref})$;

$c_{glissant} = argmax(\mathcal{C}_{glissant})$;

$c = \beta c_{ref} + (1 - \beta)c_{glissant}$;

$\mathcal{K}_{glissant+} = \gamma(\mathcal{K}_{\mathcal{I}} - \mathcal{K}_{glissant})$;

fin

Expérimentations

Résultats de référence

Notre implémentation de l'approche de référence (architecture du siamFC [1] et algorithme 1) fournit des résultats qui correspondent à l'état de l'art de 2016, date de publication du siamFC. Compte tenu des métriques expliquées dans la partie *Métriques*, les courbes de précision et de taux de succès sont disponibles sur la figure 8, il s'agit des courbes notées *avec BN*. L'aire sous la courbe de précision (*AUC*) est de **0.741**, et le score

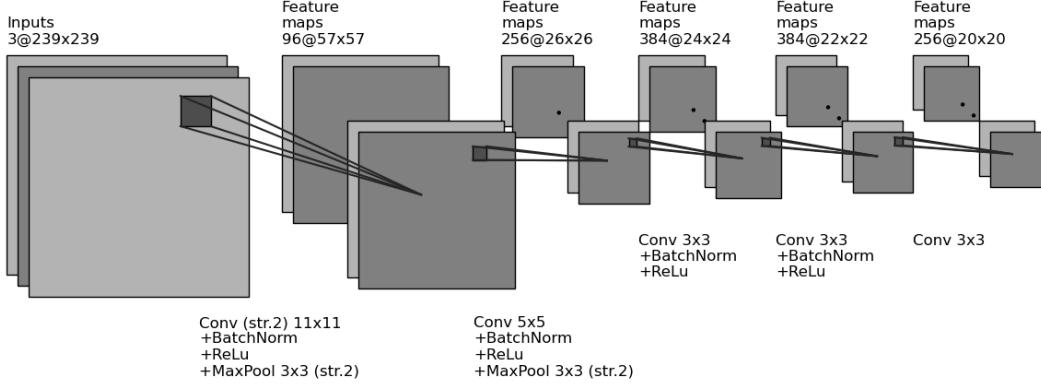


FIGURE 4. Structure de référence de l'extracteur de caractéristiques qui est utilisé dans chacune des branches du modèle siamois. La tête du réseau qui effectue la corrélation n'est pas représentée sur le schéma.

de succès pour un seuil de 20 est de **0.553**. Des exemples d'instances de suivis sur les séquences de test sont donnés en annexe A. Ces résultats cohérents donnent une bonne référence à laquelle comparer les études sur les architectures et sur le noyau glissant. Pour bien comprendre le comportement du suiveur, la partie suivante propose une étude des cas d'échecs typiques du suiveur.

Étude des cas d'échecs

Une étude des séquences donnant les moins bons résultats met en avant quatre cas d'échec classiques : *a. Objets similaires proches*, *b. Obstructions longues*, *c. Changements d'aspect de la cible*, et *d. Flash*.

a. Objets similaires proches

De nombreux échecs sont dûs à une confusion du suiveur entre deux objets très similaires dans la vidéo. Ceci s'explique par le fait que le suiveur ne se base que sur la similarité de l'objet suivi avec l'image de la cible fournie au début de la vidéo. La cohérence du suivi n'est assurée que par la proximité de l'objet d'une image à la suivante qui est imposée par la petite taille de l'image de recherche centrée autour de la dernière position de l'objet. Or si par défaillance le suiveur change de cible, alors il ne pourra pas retrouver la cible originale qui sera trop loin. Cela mène aussi à des situations où le suiveur se bloque sur des objets en arrière-plan sans pouvoir rattrapper la vraie cible.

b. Obstructions longues

Comme pour beaucoup de méthodes de suivi, la robustesse aux obstructions longues est un problème majeur. C'est un problème beaucoup plus complexe dans le cas des tâches MOT *Multi Object Tracking* que pour les tâches SOT *Single Object Tracking*. Pour le siamFC, en cas d'obstructions longues et en l'absence d'information sémantique sur les cartes de caractéristiques celles-ci auront tendance à être naturellement constantes sur toute l'image. Les cartes de corrélations ne seront donc pas porteuses d'informations, et par défaut la zone de détection restera là où elle était avant l'obstruction. Une telle situation est montrée sur la figure 5. Le ré-accrochage de la cible



FIGURE 5. Exemple d'obstruction longue (dataset de test OTB). En haut, les images de recherche dans lesquelles on doit trouver la cible, ici l'oiseau. Elles sont ordonnées temporellement mais non consécutives. En bas, les cartes de corrélations associées. On observe que pendant l'obstruction la carte de corrélation n'est pas porteuse d'information : la valeur de la corrélation est constante sur toute l'image. Le suiveur aura tendance à rester dans la même position, c'est-à-dire à toujours prédire l'objet au centre de l'image de recherche.

dépend donc du mouvement de l'objet dans la vidéos *pendant l'obstruction*. Si l'objet reste dans une zone proche de la zone dans laquelle le suiveur l'a perdu, ce dernier pourra retrouver la cible à suivre. Notons que l'objet doit rester fixe dans le repère de l'image, ce qui signifie que tout mouvement de la caméra pendant l'obstruction pourra faire perdre de vue l'objet au suiveur.

c. Changements d'aspect de la cible

Un cas fréquent d'échec survient quand l'objet change d'apparence. En particulier, dans la plupart des vidéos *naturelles*, les objets ne sont pas vus du même angle pendant toute la vidéo. Or, avec la méthode de référence, l'objet que l'on cherche doit être similaire sémantiquement à l'objet tel qu'il est donné à l'initialisation de la séquence. Malgré le haut niveau d'abstraction auquel s'effectue la comparaison, le modèle peine à reconnaître certains objets (comme une personne de dos). Un exemple extrême provenant de la base de données de validation est disponible sur la figure 6. Ce problème de changement d'aspect de la cible est l'une des raisons pour lesquelles nous avons proposé la mémorisation des noyaux, qui permet de garder en mémoire les toutes dernières apparences de l'objet.

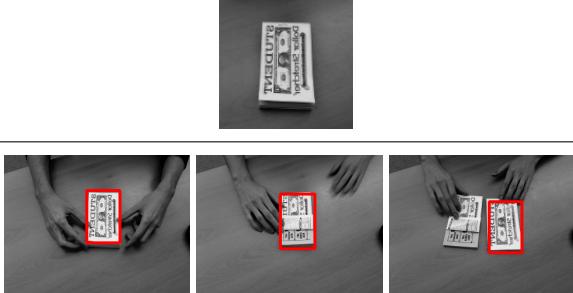


FIGURE 6. Exemple d'une séquence qui comporte un changement d'aspect. En haut, l'image initiale de la cible à suivre. En bas, des images de la séquences ordonnée temporellement et non consécutives. Dans la vidéo, un manipulateur plie le billet en deux, l'objectif du suiveur est de suivre le billet après son pliage. Après le pliage, le billet à suivre ressemble moins au noyau que les autres billets, c'est donc une séquence très difficile pour un suiveur, en particulier quand celui-ci n'a pas d'historique des apparences de la cible.

d. Flash

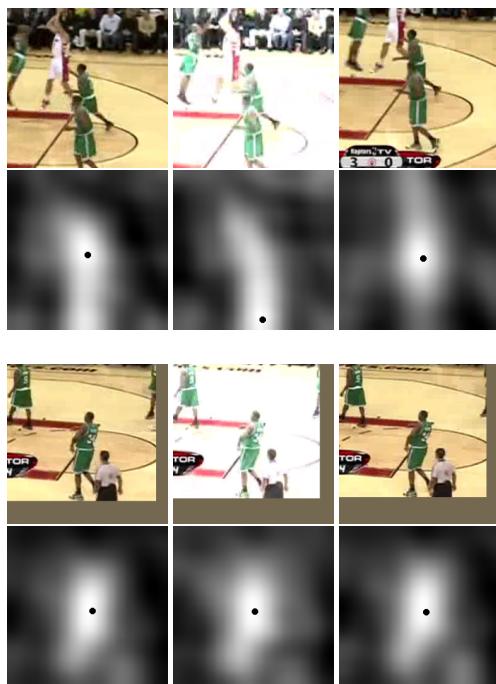


FIGURE 7. Effet d'un flash sur les cartes de corrélations. Sur les lignes supérieures, il s'agit de deux séquences dont les images sont consécutives. Sur les lignes inférieures, leurs cartes de corrélations associées dont le point noir indique le maximum. Dans les deux cas, le flash modifie de manière non uniforme les cartes de corrélations (elles sont *blanchies*). Dans la séquence du haut, le flash fait perdre la cible au suiveur, mais l'image du bas montre que ce n'est pas toujours le cas.

Certaines vidéos comportent des flash (d'appareil photo pour la plupart). Un bon suiveur doit pouvoir être robuste aux flash car c'est une situation courante. La figure 7 montre l'effet d'un flash sur l'une des images de la vidéo. Dans l'espace des images, un flash va blanchir toute l'image, rendant difficile à distinguer les objets. Dans l'espace des cartes de corrélations, on observe que les flash ont tendance augmenter la valeur de corrélation un peu partout sur la carte, mais de manière non uniforme. Ceci provoque des irrégularités dans les cartes de

corrélation, et donc des changements de cibles du suiveur dans les cas de malchance.

Architecture

a. Effet des normalisations par batch

Les résultats de l'expérimentation consistant à étudier l'impact des normalisations par batch montrent que celles ci permettent d'augmenter significativement la qualité du suiveur, autant en précision qu'en taux de succès. Les résultats sont donnés sur la figure 8.

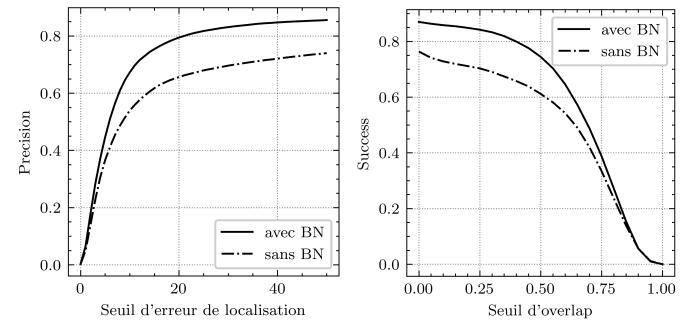


FIGURE 8. Effet des normalisations par batch sur les performances. Performance du suiveur équipé d'un extracteur à 5 couches de type AlexNet (modèle de référence) avec et sans normalisations par batch (BN).

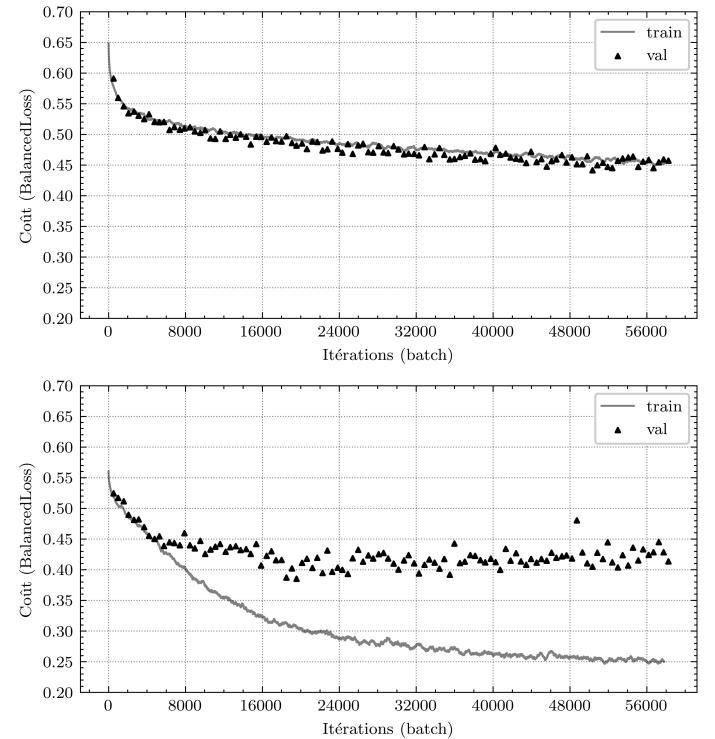


FIGURE 9. Comparaison du comportement des modèles au cours de l'apprentissage en fonction de la présence des normalisations par batch. L'étiquette *train* correspond au coût en entraînement, et *val* est le coût en validation. En haut, l'apprentissage sans normalisation par batch. En bas, l'apprentissage avec les normalisations. L'entraînement est accéléré par les normalisations et la performance de généralisation est améliorée.

On peut aussi noter qu'en plus d'améliorer la performance sur les données de test, les normalisations par batch accélèrent fortement la convergence lors de l'apprentissage comme on le voit sur la figure 9 qui montre l'évolution de l'apprentissage sur les premières itérations. Cela s'explique par la réduction de *covariance shift* décrite par S.Ioffe et al. [15] même si le sujet est encore discuté [28]. La suppression des *Batch Normalization* diminue légèrement le temps d'inférence du modèle, cependant cela ne suffit pas à justifier une telle perte de performance. **Finalement, l'architecture ne peut pas être améliorée en supprimant les normalisations par batch.**

b. Nombre de couches

Les résultats de l'expérimentation consistant à étudier l'impact du nombre de couches comme définie à la section précédente montrent que la performance en fonction du nombre de couche passe par un maximum pour 4 couches. Les trois indicateurs retenus (taux de succès, score de succès, et précision), admettent leur maximum pour la même configuration. Ces résultats sont présentés sur la figure 10. Il semble qu'il soit légèrement plus efficace d'utiliser le réseau à 4 couches que l'on a défini plutôt que le réseau à 5 couches qui constituait le modèle de référence. Il serait intéressant de pousser l'étude en vérifiant si cet effet se produit sur d'autres banques de vidéos, et en mettant en relation cet effet avec la *difficulté* des séquences.

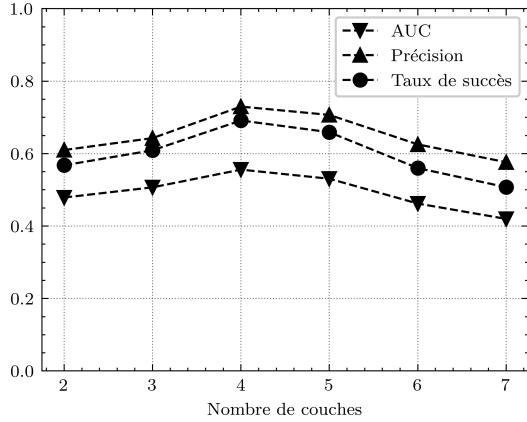


FIGURE 10. Performances du suiveur en fonction du nombre de couches de l'extracteur. Les significations des métriques sont données dans la partie *Métriques*. En particulier, *Précision* est la précision pour un seuil de 20, *AUC* est l'aire sous la courbe de succès, et le *Taux de succès* est défini pour un seuil de 0.5. Les détails des architectures utilisées sont données en annexe B. On observe des performances maximales pour l'architecture à 4 couches, puis 5 couches.

Les mauvaises performances des modèles qui comportent peu de couches s'expliquent par une complexité trop faible, qui implique une difficulté à modéliser correctement la tâche. En effet la comparaison des images se fait à un niveau sémantique, il est donc nécessaire de laisser le réseau monter en niveau d'abstraction avant de comparer les cartes de caractérisitiques.

De la même manière, les performances du modèle se déterioreront si on lui ajoute trop de couches. Cela s'explique par un sur-apprentissage sur les données d'entraînement. De plus l'ajout de couches augmente beaucoup le temps d'inférence du modèle. Notons enfin que nous avons expérimenté l'utilisation

d'un extracteur de ResNet51 [12] (modèle entier sans la tête de classification), ce qui a conduit, sans surprise, à un fort sur-apprentissage et des performances drastiquement diminuées en validation.

Amélioration du noyau

a. Analyse quantitative

Les résultats de l'amélioration du noyau grâce au noyau glissant montrent que les performances sont un peu améliorées par ce nouveau noyau. Comme le montre la figure 11, ces améliorations sont cependant à relativiser. L'aire sous la courbe de succès passe de 0.553 avec le modèle de référence à 0.555 avec le noyau glissant.

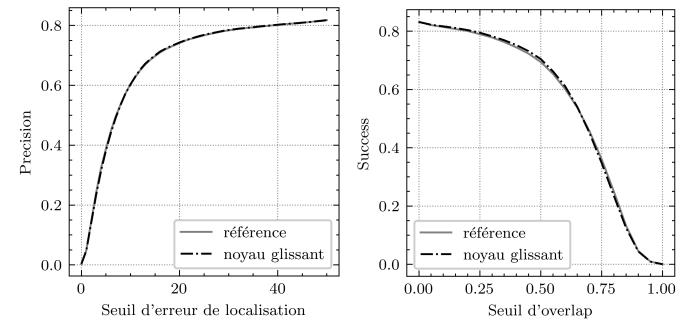


FIGURE 11. Comparaison des performances de la méthode de référence et du noyau glissant. Les performances globales sont proches, mais le taux de succès est amélioré par le noyau glissant.

Il est intéressant de noter que ces évolutions de performances ne sont pas équivalentes pour toutes les séquences de test. En particulier, la figure 12 montre des exemples de séquence où l'évolution de la performance est particulièrement bonne (*Girl2* et *Walking2* de OTB), et des exemples où le noyau glissant baisse significativement la performance (*David3* et *ClifBar* de OTB). De telles différences poussent à étudier l'effet de la modification du noyau d'un point de vue qualitatif.

b. Analyse qualitative

L'ajout du deuxième noyau défini dans *Amélioration du noyau* apporte plusieurs avantages qui se vérifient en explorant les résultats de l'expérimentation. Pour visualiser ceci, nous avons marqué sur les images de sortie à la fois les prédictions faites par le suiveur de référence, les prédictions faites par le nouveau noyau, et enfin les prédictions faites par la moyenne pondérée des deux (la méthode finale). Quelques exemples sont donnés par la figure 13, où ces trois prédictions sont respectivement en rouge, en vert, et en bleu.

On y observe que le fait de garder en mémoire les images précédentes de la cible permet bien de gagner en robustesse vis à vis des changements d'apparence des objets. Par exemple, sur l'image (a) les danseurs tournent sur eux-mêmes et la cible est le danseur. On observe que le changement de pose du danseur pousse le modèle de référence (en rouge) à prédire l'objet au niveau de la danseuse, alors que le noyau glissant (en vert) permet de rester sur le danseur.

De manière générale le noyau glissant assure une certaine inertie. Ceci est intuitif puisqu'il force le modèle à se concentrer

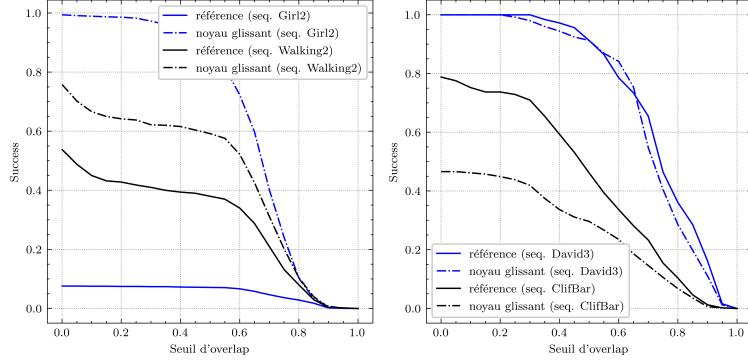


FIGURE 12. Comparaison des performances sur certaines séquences notables. À gauche, les séquences *Girl2* et *Walking2* de OTB, pour lesquelles l'amélioration est notable. À droite, les séquences *David3* et *ClifBar* de OTB, pour lesquelles l'effet est négatif.

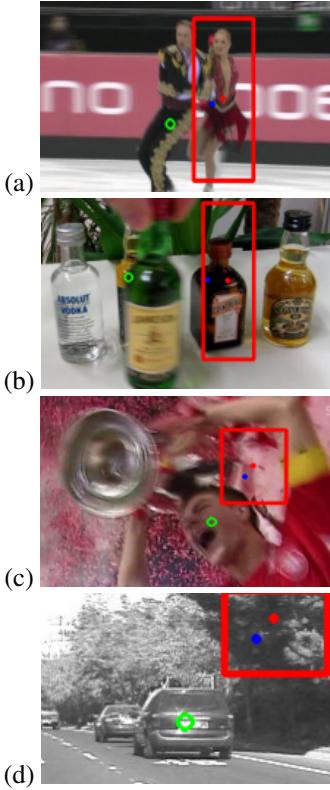


FIGURE 13. Effet qualitatif du noyau glissant sur le suivi. Les points rouges sont les prédictions de référence. Les cercles verts sont les prédictions du noyau glissant. Les points bleus sont les moyennes pondérées des deux, pour $\beta = 0.8$.

sur les formes qu'il a déjà vu. Cet effet est très bénéfique pour le suivi puisque le modèle de référence a tendance à confondre des objets, que ce soit à cause d'une ressemblance, d'une obstruction, ou d'un flash (voir *Étude des cas d'échecs*). L'effet d'inertie va empêcher la prédiction d'accrocher un autre objet qui n'est pas la cible, ce qui va laisser plus de chance au suiveur de revenir à la cible lors de la prochaine prédiction. C'est ce phénomène que montrent les images (b) à (d) de la figure 13.

Ces remarques sont à fois des avantages et des inconvénients selon la situation. En effet, l'effet d'inertie peut limiter les performances du suiveur lorsque l'objet à suivre admet une vitesse relative très importante par rapport au repère de l'image (objet rapide ou instabilité de la caméra). De plus, bien que l'inertie limite le décrochage à la cible, elle limite aussi le

réaccrochage. En d'autres termes, le suiveur sera plus robuste et fera moins d'erreur, mais dès lors que le suiveur décroche, les conséquences seront plus graves.

c. Travail futur

Considérant les remarques du paragraphe précédent, on comprend qu'il est nécessaire d'ajuster l'importance du noyau glissant en fonction de la situation. Plus concrètement, il s'agira d'indexer le comportement du suiveur sur ses valeurs d'état interne, permettant de caractériser la situation. Ensuite, on peut adapter le comportement du suiveur à travers des hyperparamètres (comme typiquement β qui va permettre de plus ou moins prendre en compte l'effet du noyau glissant). Nous avons tenté cette amélioration en adaptant β ainsi que γ en fonction de :

- la distance de la prédiction de référence à la prédiction précédente
- la distance de la prédiction du noyau glissant à la prédiction précédente
- la valeur maximale de la carte de corrélation de référence
- la valeur maximale de la carte de corrélation du noyau glissant

Le but étant de, pour ainsi dire, donner de l'importance au noyau glissant seulement lorsque celui-ci donne les signes d'une prédiction pertinente.

Nous avons implémenté cette méthode avec des règles définies à la main après observation qualitative des cas d'échecs. **Cette méthode n'a pas été concluante pour l'instant.** Cependant, cela ouvre la voie à une possibilité prometteuse : il est possible de modéliser la fonction qui donne la qualité de chacune des deux prédictions (de référence, et du noyau glissant) par un réseau de neurones qui prend en entrée les valeurs d'état du suiveur, ce qui permettrait **d'apprendre à donner de l'importance au noyau glissant**. Un tel réseau peut-être simple, et il pourrait même s'agir d'une simple combinaison linéaire des indicateurs s'il s'avère que l'expérience montre que c'est suffisant. Un point important est que la définition du noyau glissant ne contient que des dépendances simples et surtout différentiables. En conséquence, **il est possible de connaître l'erreur de prédiction en sortie de cette fonction**, ce qui permettrait d'entrainer efficacement ce réseau par rétropropagation. Ce travail demande du temps mais se présente comme la prochaine étape après ce rapport.

Conclusion

Nous avons montré que la méthode de référence du siamFC, qui se base sur AlexNet, peut être légèrement améliorée à la fois en diminuant le nombre de couches et à la fois en ajoutant un noyau glissant. Ce noyau glissant a pour rôle de garder un historique des images de la cible, ce qui augmente la robustesse du suiveur. Nous avons montré que ce noyau glissant n'est utile que dans certaines situations. Par conséquence une prochaine amélioration consiste à prédire l'importance du noyau glissant à partir des états internes du suiveur. La structure générale de la méthode permet de calculer l'erreur sur cette estimation, ce qui permettrait un apprentissage efficace de l'importance du noyau, par rétropropagation.

Références

- [1] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr. Fully-convolutional siamese networks for object tracking, 2016.
- [2] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. Simple online and realtime tracking. *2016 IEEE International Conference on Image Processing (ICIP)*, Sep 2016.
- [3] J. Bromley, J. Bentz, L. Bottou, I. Guyon, Y. Lecun, C. Moore, E. Sackinger, and R. Shah. Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7 :25, 08 1993.
- [4] N. Chenuard, I. Bloch, and J.-C. Olivo-Marin. Multiple Hypothesis Tracking for Cluttered Biological Image Sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11) :2736–2750, Nov. 2013.
- [5] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, page 539–546, USA, 2005. IEEE Computer Society.
- [6] A. Doucet, S. Godsill, and C. Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10 :197 – 208, 2000.
- [7] T. Fortmann, Y. bar shalom, and M. Scheffe. Sonar tracking of multiple targets using joint probabilistic data association. *ieee journal of oceanic engineering, oe-8*, 173-184. *Oceanic Engineering, IEEE Journal of*, 8 :173 – 184, 08 1983.
- [8] Q. Gan, Q. Guo, Z. Zhang, and K. Cho. First step toward model-free, anonymous object tracking with recurrent neural networks, 2015.
- [9] R. Girshick. Fast r-cnn, 2015.
- [10] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.
- [11] A. Gordo, J. Almazan, J. Revaud, and D. Larlus. End-to-end learning of deep visual representations for image retrieval, 2017.
- [12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.
- [13] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 fps with deep regression networks, 2016.
- [14] L. Huang, X. Zhao, and K. Huang. Got-10k : A large high-diversity benchmark for generic object tracking in the wild, 10 2018.
- [15] S. Ioffe and C. Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift, 2015.
- [16] S. Jadon and A. Srinivasan. Improving siamese networks for one-shot learning using kernel-based activation functions. 03 2019.
- [17] S. E. Kahou, V. Michalski, and R. Memisevic. Ratm : Recurrent attentive tracking model, 2016.
- [18] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1) :35–45, 03 1960.
- [19] C. Kim, F. Li, A. Ciptadi, and J. M. Rehg. Multiple hypothesis tracking revisited. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4696–4704, 2015.
- [20] M. Kristan, A. Leonardis, J. Matas, and M. Felsberg. Vot2018 : The visual object tracking challenge. 2018.
- [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.
- [22] M. T. Law, R. Urtasun, and R. S. Zemel. Deep spectral clustering learning. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1985–1994, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.
- [23] J. Masci, M. M. Bronstein, A. M. Bronstein, and J. Schmidhuber. Multimodal similarity-preserving hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4) :824–830, 2014.
- [24] H. Nam and B. Han. Learning multi-domain convolutional neural networks for visual tracking, 2016.
- [25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once : Unified, real-time object detection, 2016.
- [26] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn : Towards real-time object detection with region proposal networks, 2016.
- [27] S. H. Rezatofighi, A. Milan, Z. Zhang, Q. Shi, A. Dick, and I. Reid. Joint probabilistic data association revisited. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3047–3055, 2015.
- [28] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization ?, 2019.
- [29] B. Shuai, A. G. Berneshawi, D. Modolo, and J. Tighe. Multi-object tracking with siamese track-renn, 2020.
- [30] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks, 2015.
- [31] L. Wang, W. Ouyang, X. Wang, and H. Lu. Visual tracking with fully convolutional networks. pages 3119–3127, 12 2015.
- [32] N. Wang, S. Li, A. Gupta, and D.-Y. Yeung. Transferring rich feature hierarchies for robust visual tracking, 2015.
- [33] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. S. Torr. Fast online object tracking and segmentation : A unifying approach, 2019.
- [34] N. Wojke, A. Bewley, and D. Paulus. Simple online and realtime tracking with a deep association metric, 2017.
- [35] Y. Wu, J. Lim, and M. Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9) :1834–1848, 2015.
- [36] Y. Xu, Z. Wang, Z. Li, Y. Yuan, and G. Yu. Siamfc++ : Towards robust and accurate visual tracking with target estimation guidelines, 2020.

Appendices

A. Quelques exemples de prédictions

Exemples corrects

OTB - Bolt



OTB - Car



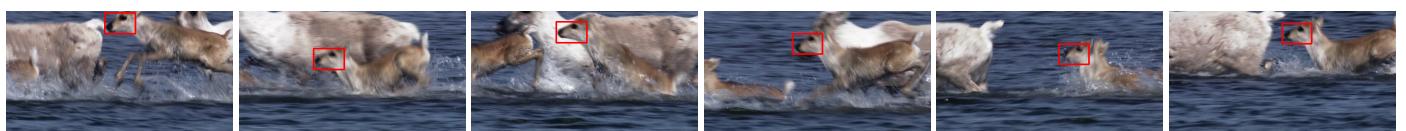
OTB - Dancer



OTB - DragonBall



OTB - Deer



OTB - Football



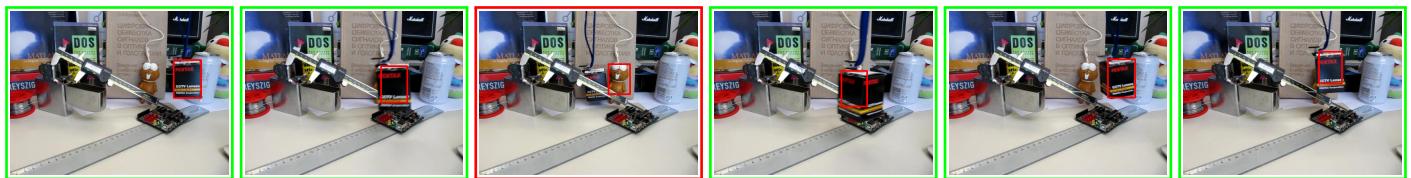
Exemples d'échecs

Les prédictions en vert sont correctes, les prédictions en rouge ne le sont pas.

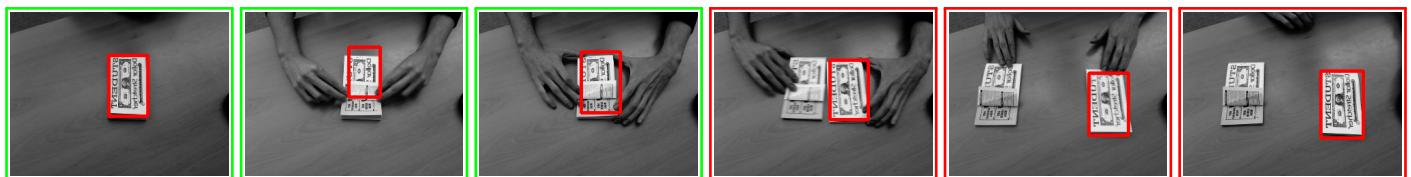
OTB - Biker



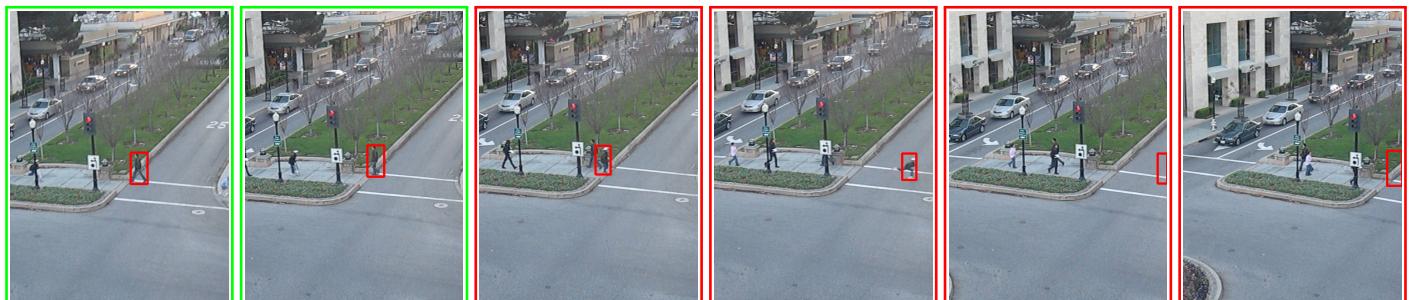
OTB - Box



OTB - Coupon



OTB - Human3



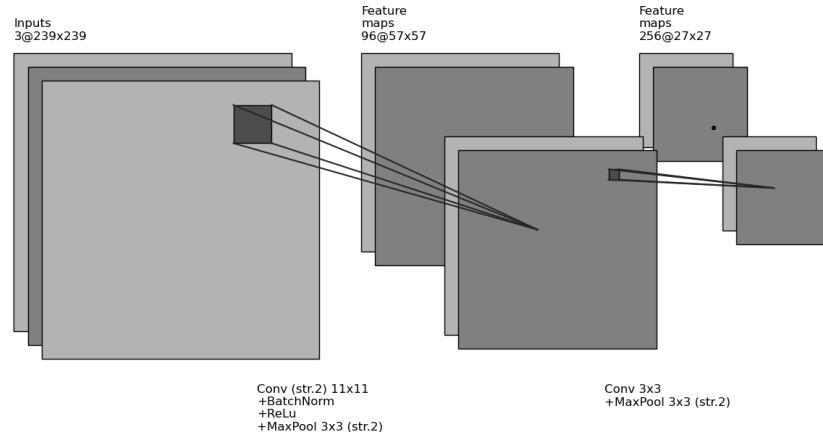
OTB - Skiing



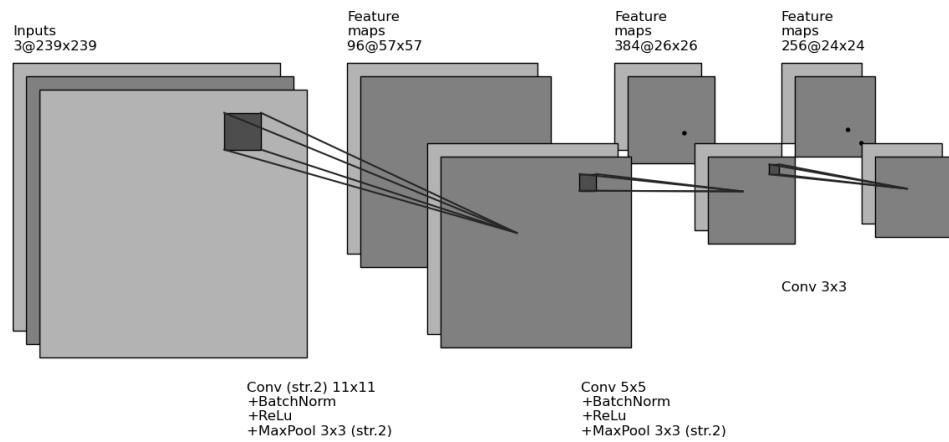
B. Les différentes architectures testées

Les matrices d'entrée (images à 3 canaux) ne sont pas représentées à l'échelle pour des raisons de lisibilité.

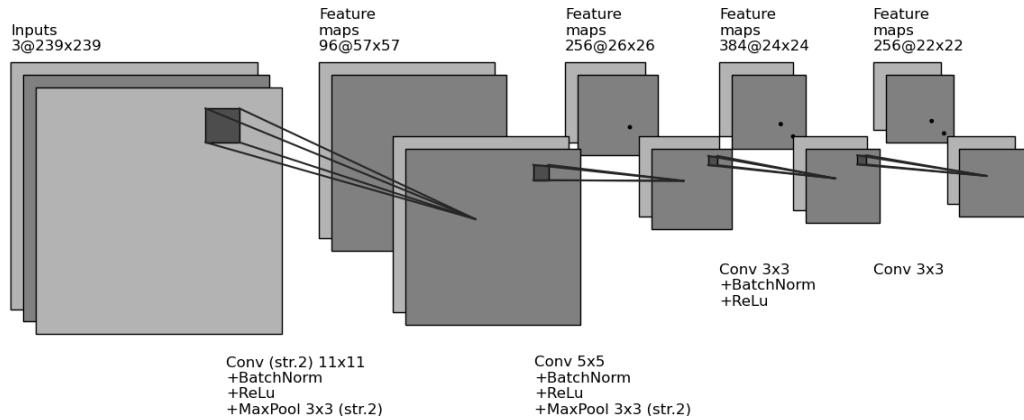
Architecture à 2 couches.



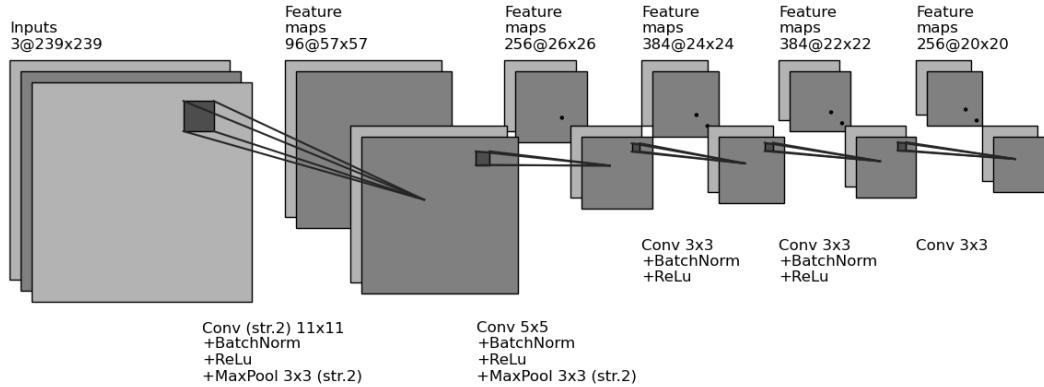
Architecture à 3 couches.



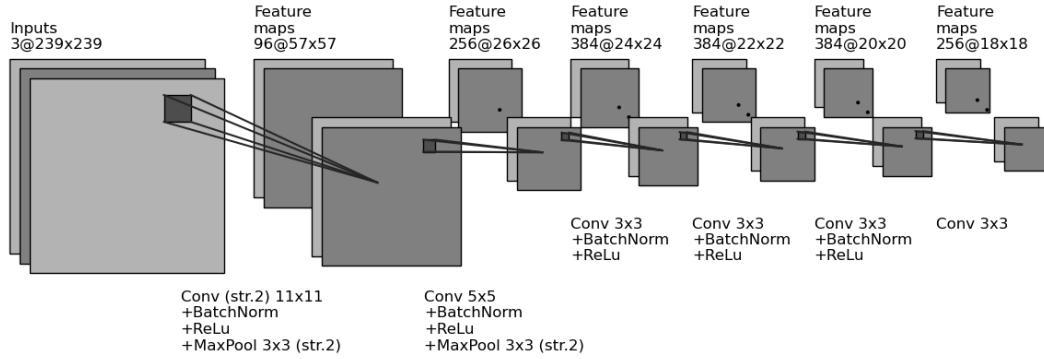
Architecture à 4 couches.



Architecture à 5 couches (modèle de référence [1]).



Architecture à 6 couches.



Architecture à 7 couches.

