Du repérage sémantique robuste d'actions vers leur détection dans les vidéos

Guillaume Vaudaux-Ruth

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

École doctorale N° 391
Sciences Mécaniques, Acoustique, Électronique et Robotique de Paris

Présentée par

Guillaume VAUDAUX-RUTH

le 8 décembre 2021

Pour obtenir le grade de docteur délivré par Sorbonne Université

Sujet de la thèse :

Du repérage sémantique robuste d'actions vers leur détection dans les vidéos

Devant le jury composé de :

Stéphane DONCIEUX	Professeur, Sorbonne Université	Président
Jean-Philippe VANDEBORRE	Professeur, Université de Lille	Rapporteur
Quoc Cuong PHAM	Directeur de recherche, CEA-LIST	Rapporteur
Alexandre BOULCH	Ingénieur de recherche, Valeo.ai	Examinateur
Catherine ACHARD	Professeure, Sorbonne Université	Directrice de thèse
Adrien CHAN-HON-TONG	Ingénieur de recherche, ONERA	Encadrant

Remerciements

Je souhaite tout d'abord remercier Jean-Philippe Vandeborre et Quoc Cuong Pham d'avoir accepté de rapporter ce manuscrit de thèse. Je remercie également Stéphane Doncieux, et Alexandre Boulch d'avoir accepté de faire partie de mon jury de thèse en tant qu'examinateurs.

Je remercie Catherine Achard, ma directrice de thèse, de m'avoir accompagné et supporté pendant ces trois années. Merci pour la disponibilité et le soutien sans faille malgré mes lubies et les contraintes que nous avons connues.

Un immense merci à Adrien Chan-Hon-Tong, mon encadrant à l'ONERA, qui m'a escorté durant toute la thèse. Merci de m'avoir épaulé dans tous mes combats, du plus ambitieux au plus farfelu. Merci aussi pour ton état d'esprit, ton humilité et plus globalement ta personnalité, qui permettent l'épanouissement des personnes que tu encadres.

Merci aux hasards de la vie et à Stéphane de m'avoir propulsé, presque malgré moi, dans le monde de la recherche et vers cette thèse de dernière minute. Merci également aux membres du DTIS que j'ai côtoyés à l'ONERA, et qui ont partagé mon quotidien durant mon stage de fin d'études, et ces trois années de thèse. Merci aux membres permanents de toutes ces époques : Guy, Fred, Fabrice J., Elise, Aurélien, Alex B., Philippe, Martial, Alex E., Anthelme, Alain, Fabrice S., Gilles, Pierre F., Flora, Julien, Pauline, Baptiste. Je remercie plus particulièrement Patrick d'avoir partagé ma passion pour le sport et Benjamin P. d'avoir sublimé ma vie de judoka.

Merci à tous les doctorants et stagiaires d'avoir égayé, chacun à leur manière, mon quotidien et plus généralement celui de notre étage : Maxime C., Maxime F., Guillaume H., Javiera, Gaston, Maxime B., Louis, Pierre G., Alexis, Rémy, Soufiane, Rodrigo, Rodolphe, Laurane, William, Benjamin B., Marius, Quentin, Thomas DM., Nathan, Philip, Simon, Kevin, Pol, Adrien LC., Antoine, Thomas, Nina, Silya, Dina, Celena, Aymeric, Thibaud.

Et comme ce qui compte ce n'est pas (seulement) l'arrivée, c'est la quête, je souhaiterais d'autant plus remercier les protagonistes qui y ont participé et qui y

participeront encore longtemps : à Nina, à ma famille, à tous les copains (ils ne sont pas nombreux et leur particularité réside dans le fait qu'ils se reconnaîtront aisément sans les citer) de Marseille, de Dijon, du judo, de l'athlétisme, et ceux « d'enfance ». Merci pour votre soutien et tous les « à-côtés » qui font que ces trois dernières années sont finalement passées beaucoup trop vite.

Résumé

La compréhension de vidéos nécessite une caractérisation à la fois spatiale et temporelle de leur contenu. Ainsi, face au succès des méthodes d'apprentissage statistique par réseaux de neurones pour l'analyse automatique d'images, ces méthodes ont rapidement été étendues au contexte spatio-temporel. La dimension temporelle introduit de nouvelles problématiques algorithmiques, que ce soit pour la caractérisation des vidéos ou pour l'extraction d'informations sémantiques.

Dans cette thèse, nous nous intéressons à la détection temporelle d'actions qui vise, non seulement à reconnaître les actions présentes dans une vidéo, mais aussi à en connaître les limites temporelles. Plus précisément, nous étudions l'impact que peut avoir la recherche d'une localisation temporelle fine sur la robustesse de l'extraction sémantique.

Ainsi, dans la première partie de ce manuscrit, nous abordons le problème du repérage d'actions, qui vise une extraction sémantique robuste sans contrainte forte de localisation temporelle. Pour cela, nous introduisons un algorithme basé sur une technique d'apprentissage par renforcement, permettant de réduire la sensibilité aux annotations lors de l'apprentissage et ainsi d'améliorer la qualité de l'extraction sémantique.

Dans une seconde partie, nous étendons cette méthode de manière à obtenir également des indications sur la localisation temporelle des actions. Nous introduisons aussi la notion de confiance temporelle qui permet de focaliser le processus d'extraction de contenu d'intérêt sur les zones ayant une forte probabilité de contenir une action.

Enfin, nous proposons d'étudier les limites de ces approches et proposons des pistes pour arriver à une localisation temporelle précise des actions.

Mots-clés : apprentissage profond, traitement vidéo, repérage d'actions, détection d'actions, apprentissage par renforcement

Abstract

Video understanding requires both spatial and temporal characterization of their content. Thus, given the success of statistical learning methods using neural networks for automatic image analysis, these methods were quickly extended to the spatio-temporal context. The temporal dimension introduces new algorithmic issues, whether for video characterization or for semantic information extraction.

In this thesis, we are interested in temporal action detection, which aims not only at recognizing the actions present in a video, but also at knowing their temporal limits. More precisely, we study the impact that the search for a fine temporal localization can have on the robustness of semantic extraction.

Thus, in the first part of this manuscript, we address the problem of action spotting, which aims at robust semantic extraction without strong temporal localization constraints. For this purpose, we introduce an algorithm based on a reinforcement learning technique, allowing to reduce the sensitivity to annotations during the learning process and thus to improve the quality of the semantic extraction.

In a second part, we extend this method in order to also obtain indications on the temporal location of actions. We also introduce the notion of self-assessment which allows us to focus the process of extracting content of interest on areas with a high probability of containing an action.

Finally, we propose to study the limitations of these approaches and suggest ways to achieve a precise temporal localization of actions.

Keywords : deep learning, video processing, action spotting, action detection, reinforcement learning

Sommaire

Table des figures

Liste des tableaux

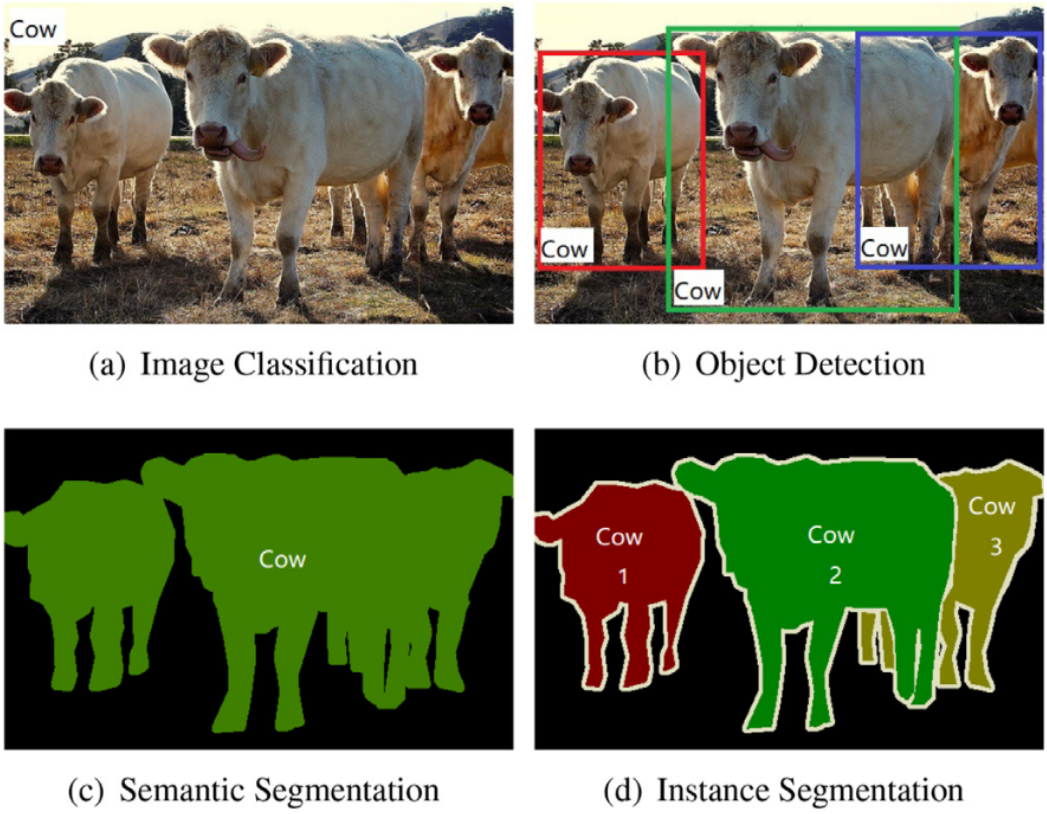
Chapitre 1

Introduction

Sommaire

1.1 Contexte

La vision par ordinateur est un domaine de l'ingénierie informatique né de la volonté d'automatiser la compréhension visuelle du monde. Elle cherche ainsi à imiter le processus de vision humaine pour acquérir, traiter et comprendre des images. La compréhension de l'information visuelle a alors de nombreux domaines applicatifs comme les télécommunications (S. Ma et al., 2021), la sécurité (Sreenu et al., 2019), ou alors la robotique autonome (Grigorescu et al., 2020). Pour atteindre la compréhension visuelle complète dont est capable un être humain, une des thématiques sous-jacentes à la vision par ordinateur est la reconnaissance visuelle (Mogadala et al., 2019), dont l'objectif est de décrire automatiquement le contenu d'une image.



(a) Image Classification (b) Object Detection

(c) Semantic Segmentation (d) Instance Segmentation

FIGURE 2 – **Exemple de description automatique d'une image** suivant différents niveaux d'analyse. Crédit image : X. WU et al., 2020.

Pour aborder le problème de reconnaissance visuelle, les méthodes développées par la communauté lors de la dernière décennie utilisent presque exclusivement des techniques d'apprentissage statistique profond, appelé communément *Deep Learning* (DL). Le succès de l'apprentissage profond vient de sa capacité à automatiser l'extraction de caractéristiques visuelles, de manière conjointe à la tâche réalisée. En effet, avant l'ascension de l'apprentissage profond, la majorité des méthodes requéraient une extraction préalable des caractéristiques (MINGQIANG et al., 2008), réalisée selon des méthodologies issues du traitement du signal.

Le premier succès de l'apprentissage profond pour une tâche de vision par ordinateur a été la consécration d'un algorithme utilisant un réseau convolutif (LECUN et al., 1995) lors d'une compétition de classification d'images à ILSVRC (KRIZHEVSKY et al., 2012). Depuis, l'apprentissage profond a été étendu à des tâches d'analyse d'images nécessitant une compréhension sémantique plus fine. On y trouve notamment la détection d'objets (X. WU et al., 2020) qui requiert à la fois localisation et classification, ou la segmentation sémantique (HAFIZ et al., 2020) qui cherche à classer chacun des pixels (voir Figure 2).

La réussite des méthodes d'apprentissage profond sur des problèmes de com-

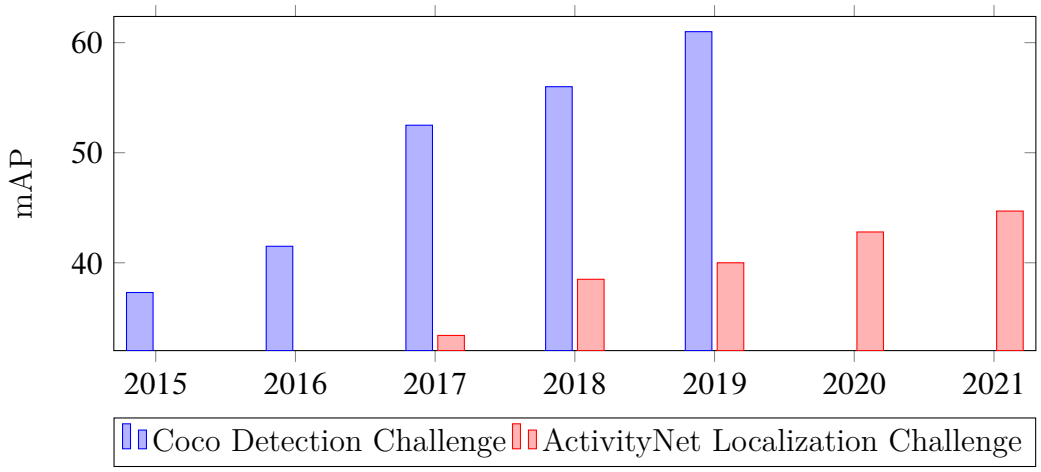


FIGURE 3 – Evolution temporelle de la précision des modèles vainqueurs du challenge *Coco Detection Challenge* (T.-Y. LIN et al., 2014), de détection d'objets, et du challenge *ActivityNet Localization Challenge* (CABA HEILBRON et al., 2015), de détection d'actions dans des vidéos.

préhension d'images laisse alors présager une réussite similaire pour leur extension à des problématiques de compréhension vidéo. Mais, de manière structurelle, on passe d'une analyse de données en 2 dimensions (2 dimensions spatiales) à une analyse de données en 3 dimensions (2 dimensions spatiales et 1 dimension temporelle). L'augmentation du nombre de dimensions des données augmente alors non seulement la complexité de leur traitement mais aussi la complexité des analyses sémantiques possibles. Comme le montre la Figure 3, l'évolution de la performance des modèles d'analyse vidéo est moins rapide que celle de l'analyse d'images. Comme l'expriment DIBA et al., 2019, l'analyse sémantique de vidéos englobe la reconnaissance de multiples aspects sémantiques comprenant : une scène ou un environnement, des objets, des actions, des événements, des attributs et des concepts. Mais les analyses actuelles se limitent généralement à un seul aspect, comme la compréhension d'actions (HUTCHINSON et al., 2020).

La compréhension d'actions, une des thématiques les plus étudiées en traitement vidéo, trouve un nombre considérable d'applications dans la vie quotidienne. En effet, l'automatisation des applications nécessitant soit des analyses vidéo en temps réel, soit un traitement vidéo très volumineux, permet de diminuer la pénibilité de l'intervention humaine. On trouve alors des applications telles que la vidéo surveillance (SREENU et al., 2019), l'indexation et la caractérisation de contenus *web* (PODLESNAYA et al., 2016), la robotique (REZAZADEGAN et al., 2017), les véhicules autonomes (YURTSEVER et al., 2020), le sous-titrage vidéo (H. XU et al., 2019), ou encore le monitorage de personnes en situation de dépendance (BUZZELLI et al., 2020).

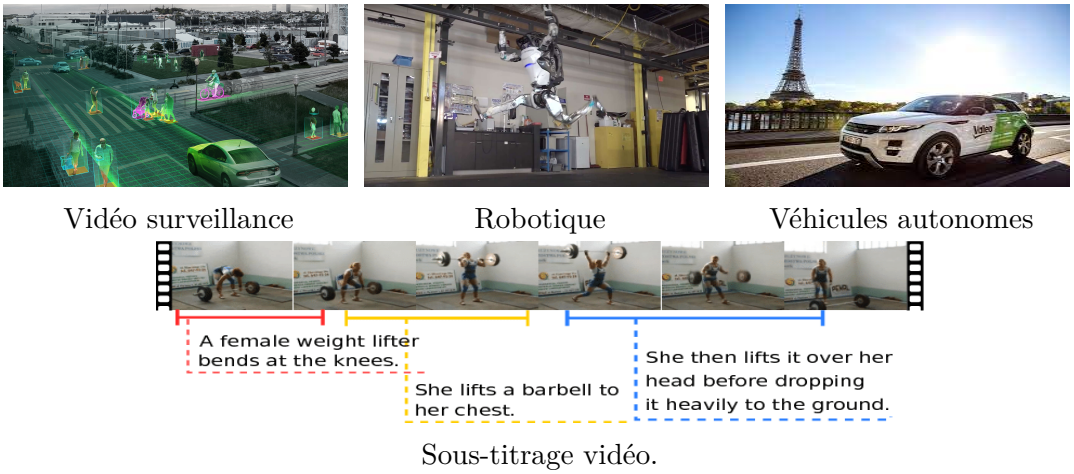


FIGURE 4 – **Applications de la compréhension d'actions dans des vidéos**.

1.2 Compréhension d'actions

La compréhension d'actions regroupe ainsi l'ensemble des problématiques qui s'intéressent à l'activité humaine dans des vidéos (YURTSEVER et al., 2020). Comme en analyse d'images, on y retrouve des thématiques de classification, des thématiques de recherche spatiale et/ou temporelle et des problèmes mêlant recherche et classification.

Dans la suite, la composante de recherche spatiale (GU et al., 2018) sera omise. Comme le montre alors la Figure 5, les problématiques centrales sont la reconnaissance d'actions, le repérage, la localisation temporelle, et la détection.

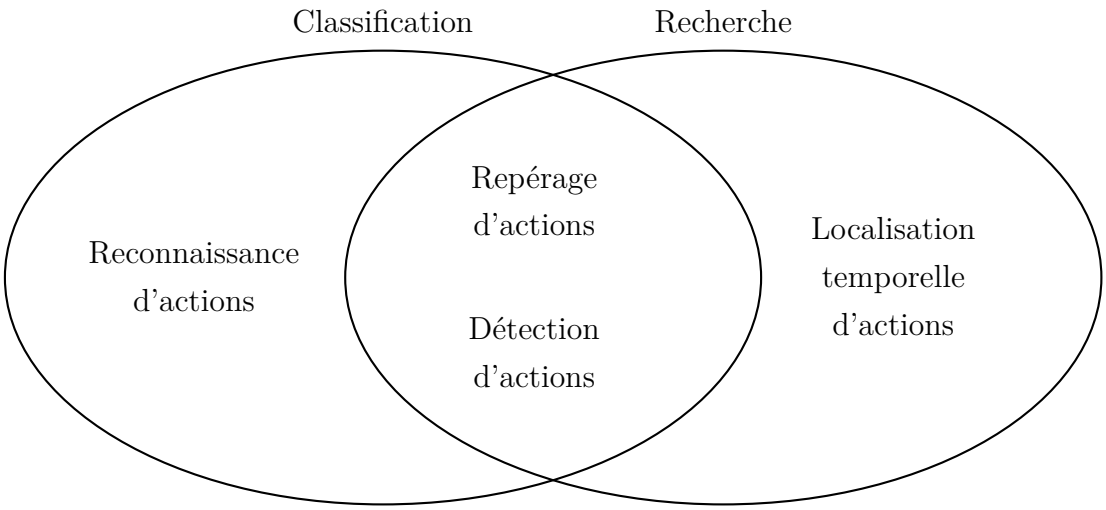


FIGURE 5 – Taxonomie des thématiques de compréhension d'actions dans des vidéos (YURTSEVER et al., 2020).

1.2.1 Reconnaissance d'actions

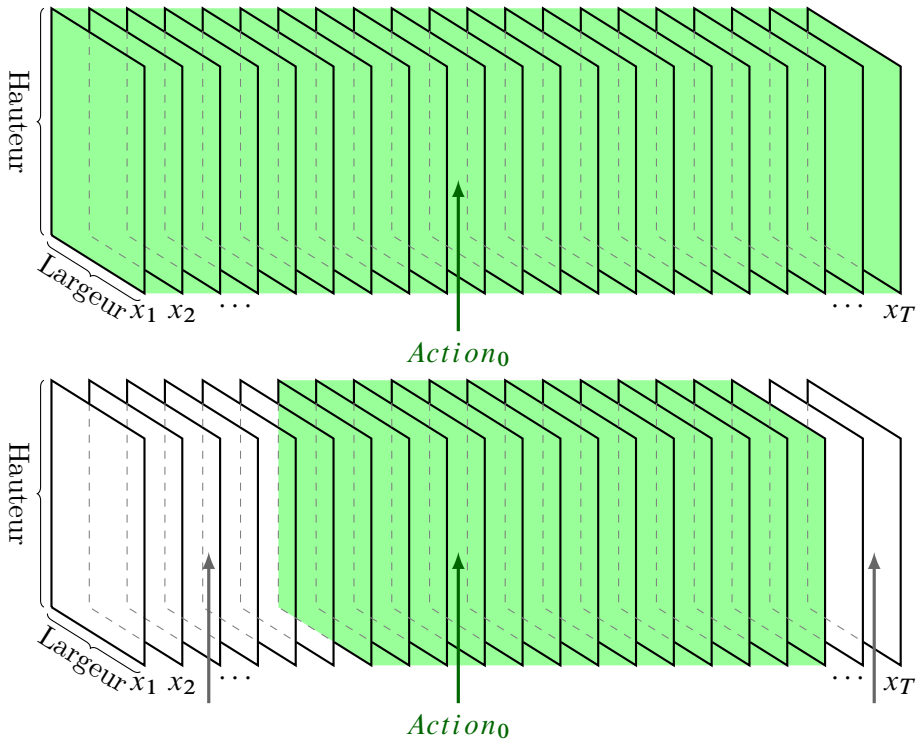


FIGURE 6 – **Problème de reconnaissance d'actions** : en haut pour une vidéo *bornée* et en bas pour une vidéo *non bornée*. La vidéo est représentée comme un empilement de T images.

La reconnaissance d'actions, ou *Action Recognition (AR)* dans la littérature, est le processus de classification d'une entrée vidéo en une classe d'action unique reflétant de manière globale l'activité présente dans la vidéo. Si l'action s'étend sur toute la durée de la vidéo, on qualifie la vidéo de *bornée* et le problème est connu sous le nom de *reconnaissance d'actions bornées*. *A contrario*, si l'action n'est pas présente sur toute la durée de la vidéo, mais seulement sur une portion, on qualifie la vidéo de *non bornée* et le problème est alors connu sous le nom de *reconnaissances d'actions non bornées*. De cette manière, une vidéo considérée comme *bornée* contient uniquement l'action, tandis qu'une vidéo *non bornée* contient également des informations non pertinentes pour déterminer la classe d'action, et qui sont alors considérées comme du bruit(voir Figure 6). Dans les deux cas, la reconnaissance d'actions est réalisée sans aucune extraction de localisation temporelle de l'action. La reconnaissance d'actions dans des vidéos *non-bornées* est donc une tâche plus complexe que celle de vidéos bornées puisque l'information pertinente peut alors être très locale et non saillante.

1.2.2 Repérage d'actions

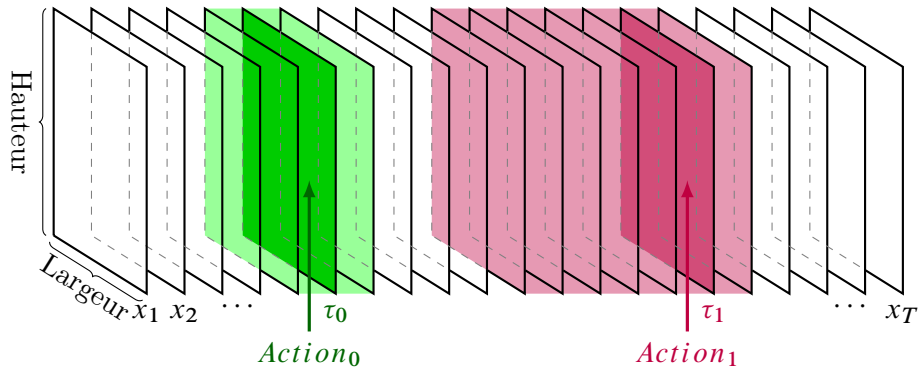


FIGURE 7 – **Problème de repérage d'actions**. La vidéo est représentée comme un empilement de T images. Ici, les occurrences d'actions sont repérées grâce aux marqueurs temporels des images foncées τ_i.

Le repérage d'actions, ou *Action Spotting (AS)* dans la littérature, consiste à trouver toute occurrence temporelle d'une action dans une vidéo *non-bornée*, en observant le moins d'images possible. Une occurrence temporelle se définissant simplement comme une image, ou son marqueur temporel, appartenant à une action (voir Figure 7). Le repérage d'actions permet donc d'extraire des indications sur la présence d'actions dans des vidéos et leur localisation temporelle approximative.

1.2.3 Proposition temporelle d'actions

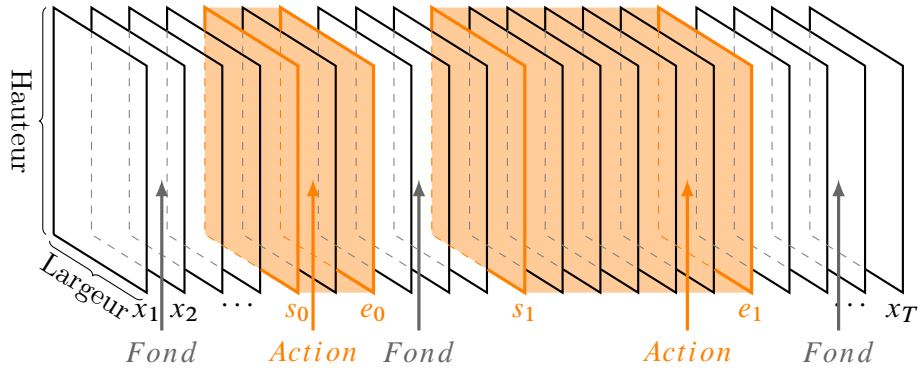


FIGURE 8 – **Problème de proposition temporelle d'actions**. La vidéo est représentée comme un empilement de T images et les segments temporels d'actions sont repérés grâce aux empilements d'images de couleur orange, délimités par les bords $[s_i, e_i]$.

La tâche de proposition temporelle d'actions, ou *Temporal Action Proposal* (TAP) dans la littérature, est le processus d'échantillonnage de segments tempo-

rels d'actions (portions de vidéo) à partir d'une vidéo *non-bornée*. Ce processus consiste donc à proposer des paires de marqueurs temporels début/fin et de les classer de manière binaire en tant qu'action ou classe de fond (voir Figure 8). Les propositions permettent donc de localiser les actions sans chercher la classe d'action précise.

1.2.4 Détection temporelle d'actions

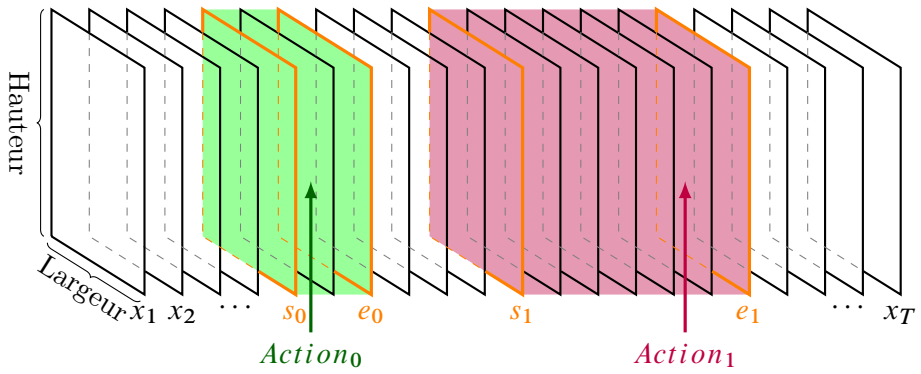


FIGURE 9 – **Problème de détection temporelle d'actions**. La vidéo est représentée comme un empilement de T images et les segments temporels d'actions sont repérées grâce aux empilements d'images de couleur, spécifique à leur classe d'action, et délimités par les bords $[s_i, e_i]$.

La détection temporelle d'actions, ou *Temporal Action Localization/Detection (TAL/D)* dans la littérature, est le processus double de proposition temporelle d'actions et de classification des propositions. Comme défini précédemment, à partir d'une vidéo non-bornée, des segments temporels sont échantillonnés, puis pour chaque segment vidéo, une classe d'action est déterminée (voir Figure 9). Une même vidéo peut contenir différentes classes d'action. De cette manière, les actions sont localisées et leur contenu est caractérisé précisément.

1.3 Contraste entre recherche d'une localisation précise et sémantique

Parmi les thématiques de compréhension d'actions, la place de la localisation des actions dans les vidéos a plus ou moins d'importance. Alors que la tâche de repérage d'actions cherche une localisation temporelle approximative, les tâches de localisation et de détection cherchent à localiser les limites temporelles des actions, de manière relative à un critère de finesse de localisation, la *temporal*

Intersection over Union (tIou). Ce critère, qui sera défini précisément plus tard, correspond à une valeur de recouvrement entre la localisation temporelle prédite et la localisation réelle de l'action. Ainsi, plus la tIoU voulue est proche de 1 et plus les limites temporelles prédites doivent être précises. A l'inverse, plus elle est proche de 0 et plus les limites prédites peuvent être approximatives. Dans la suite, le niveau de tIoU voulu pour la tâche de détection sera précisé sous la forme *détection@tIoU*. Ainsi, plus la localisation temporelle des actions prend d'importance et plus la performance des modèles de l'état de l'art décroît (voir Figure 10).

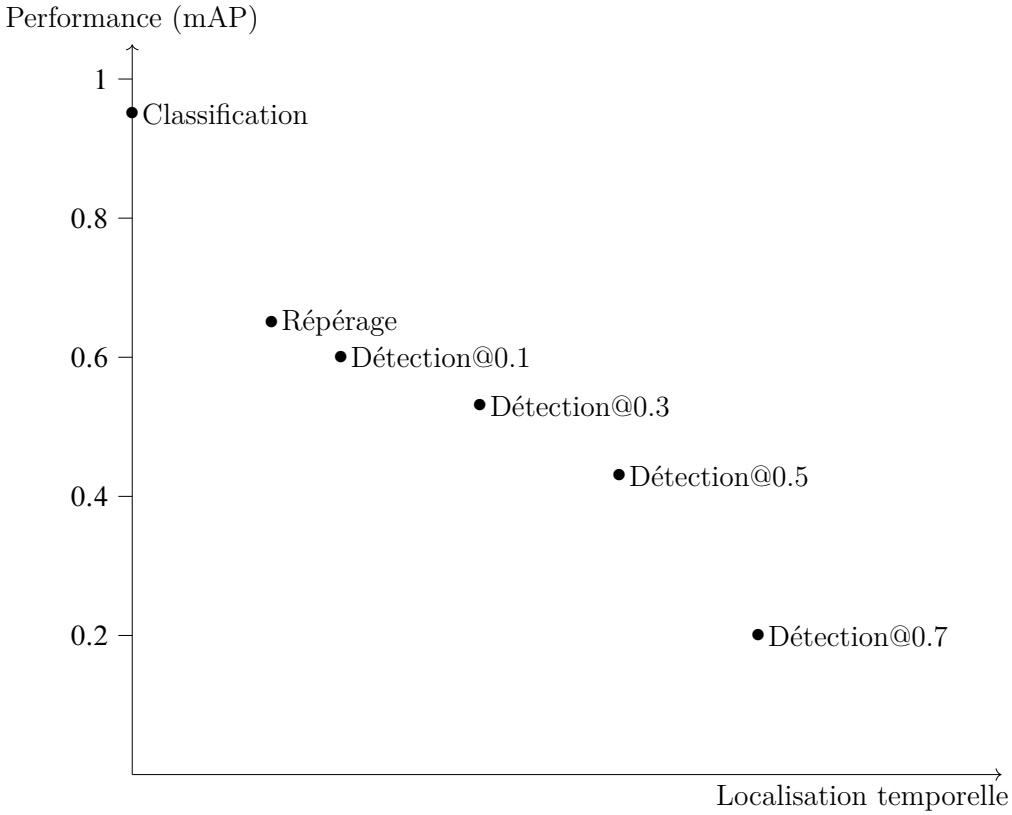


FIGURE 10 – Performances suivant la mAP, métrique de l'état de l'art, issues de (CHAO et al., 2018 ; VAUDAUX-RUTH et al., 2021a), pour les différentes tâches de compréhension d'action sur le dataset THUMOS14 (IDREES et al., 2017), en fonction de l'importance de la recherche temporelle au sein de la tâche.

La classification d'actions, qui n'inclut pas de composante de localisation temporelle est la tâche la plus maîtrisée dans l'état de l'art (Y. ZHU et al., 2020). On trouve ensuite le repérage d'actions (VAUDAUX-RUTH et al., 2021a), et pour finir la détection d'actions (XIA et al., 2020). Les performances sur cette dernière tâche décroissent lorsque le niveau de précision de localisation des limites temporelles des action augmente.

Mais ces performances sont relatives aux métriques utilisées par l'état de l'art. En effet, comme il a été indiqué, si l'on se concentre sur la tâche de détection d'actions, les métriques de l'état de l'art sont relatives à la tIoU, critère de précision de localisation des bords. En vidéo, la recherche précise des bords d'action est souvent mise à mal par plusieurs facteurs. Par exemple, lorsqu'une action cherchée est courte, une variation d'une image dans l'extraction du segment temporel peut facilement l'empêcher de remplir le critère de tIoU, alors que son positionnement et l'extraction sémantique sont correctes. Aussi, on constate lors de l'annotation de bases de données de détection d'actions, que les bords des actions proposés par les annotateurs ont une variance très importante (Gu et al., 2018), la définition d'une même action pouvant être très différente entre deux annotateurs. Cet effet est aussi présent pour les problématiques images mais beaucoup plus marqué en vidéo. Si l'on cherche à détecter un lancer de javelot, doit-il comporter la course d'élan ou simplement le lancer ? Un changement de plan est il une rupture dans la continuité d'une action ? C'est autant de questions qui introduisent du bruit lors de l'annotation et donc des problèmes pour la confection d'algorithmes précis, au sens des métriques de l'état de l'art. Mais cette perte de précision sur la localisation temporelle n'est pas forcément accompagnée d'une perte de précision sémantique (reconnaissance).

Ainsi, le repérage d'actions, ou une détection à faible tIoU (recherche de bords peu précis), amène à une extraction sémantique robuste au détriment de la qualité de la localisation temporelle. Il est à noter que ceci est suffisant pour certaines applications comme le monitorage de personnes en situation de dépendance, où des statistiques sont réalisées pour décrire l'activité journalière d'une personne.

Dans ce manuscrit, nous étudions l'impact que peut avoir la recherche d'une localisation temporelle fine sur la robustesse de l'extraction sémantique.

1.4 Plan du manuscrit

Ce manuscrit est axé autour de six chapitres.

Après le premier chapitre introductif, le chapitre 2 propose un état de l'art à la fois technique avec les principales méthodes d'apprentissage et méthodologique avec les méthodes récentes de reconnaissance d'actions, de repérage d'actions, de proposition temporelle d'actions et de détection d'actions. Il se termine par une présentation des bases de données utilisées durant cette thèse.

Le chapitre 3 présente les travaux réalisés sur le repérage d'actions qui ont pour objectif d'atteindre une extraction sémantique robuste. Nous introduisons

une nouvelle mesure permettant de quantifier la qualité sémantique des extractions réalisées par les algorithmes de repérage d'actions. Ensuite, nous présentons ActionSpotter, une nouvelle méthode de repérage d'actions qui sera évaluée sur des bases de données de l'état de l'art. Lors du chapitre 4, la méthode de repérage sera étendue à la détection d'actions arrivant ainsi à une nouvelle méthode nommée SALAD, basée sur la notion de confiance temporelle. Après une étude approfondie des limites des méthodes précédentes, le chapitre 5 propose des pistes pour affiner la localisation temporelle des actions. Plus précisément, trois approches seront présentées pour combiner détection sémantique robuste et précision sur la localisation temporelle des actions. Lors du chapitre 5, nous résumons nos contributions, ouvrons une discussion plus générale et présentons des perspectives à ce travail.

1.5 Contributions

Plusieurs contributions ont été introduites durant cette thèse.

— **Une mesure d'évaluation permettant de quantifier la qualité des algorithmes de repérage d'actions**. Le repérage d'actions est une thématique nouvelle et importante puisqu'il permet de connaitre l'ordre séquentiel de toutes les actions présentes dans une vidéo. Ainsi, il est très facile de faire des statistiques à partir du repérage, par exemple pour contrôler l'évolution du comportement des actions à long terme. Aucune mesure n'existait à ce jour pour comparer les rares méthodes de l'état de l'art. Nous avons ainsi proposé une telle mesure et mis le script à disposition pour les chercheurs du domaine.

— **Un algorithme robuste de repérage d'actions**. Nous avons proposé une méthode de repérage d'actions s'appuyant sur une méthode d'apprentissage par renforcement, capable de s'ajuster aux spécificités de la tâche, tout en simplifiant l'annotation des vidéos requise par les méthodes de l'état de l'art. La méthode proposée, nommée *ActionSpotter*, a été évaluée sur des bases de données de l'état de l'art. Ceci nous a permis de montrer la grande robustesse d'*ActionSpotter* et plus généralement l'intérêt des techniques de repérage d'actions, pour l'extraction de contenu sémantique dans les vidéos.

— **Un algorithme de détection d'actions**. Notre proposition consiste à étendre la méthode de repérage robuste précédente, en introduisant une nouvelle méthodologie algorithmique de détection d'actions appelée *SA-*

LAD, fondée sur la notion de confiance temporelle. L'introduction de cette notion dans le processus d'apprentissage permet ainsi de pouvoir coupler la régression des propositions temporelles d'action et leur tri. Ce couplage conduit alors à des résultats à l'état de l'art sur les *benchmarks* du domaine, pour des seuils faibles de tIoU.

— **Un raffinement des détections**. Après avoir étudié les limites des approches précédentes pour les fortes tIoU, nous proposons des pistes pour arriver à une localisation temporelle fine des actions. Pour cela, nous introduisons 3 améliorations du modèle *SALAD* permettant d'affiner les détections. Nous proposons tout d'abord une technique d'hybridation permettant d'affiner *a posteriori* les bords des prédictions. Nous proposons ensuite d'enrichir temporellement l'extraction de caractéristiques en réalisant une pyramide de *pooling* vidéo. Nous proposons finalement d'utiliser une méthode de *curriculum learning* permettant d'augmenter progressivement la complexité de la tâche de détection à l'apprentissage. Cette modification d'apprentissage met en lumière les limites d'un apprentissage multi-seuils de tIoU.

1.6 Publications

Les travaux réalisés ont donné lieu à des publications scientifiques en conférences internationales avec comité de lecture :

— **G. Vaudaux-Ruth**, A. Chan-Hon-Tong, C. Achard. "Action Spotter : Deep Reinforcement Learning Framework for Temporal Action Spotting in Videos". 25^{th} *International Conference on Pattern Recognition* (ICPR 2020). Publiée.

— **G. Vaudaux-Ruth**, A. Chan-Hon-Tong, C. Achard. "SALAD : Self-Assessment Learning for Action Detection". *Winter Conference on Applications of Computer Vision* (WACV 2021). Publiée.

Ces travaux de thèse ont aussi conduit à une amorce de transfert technologique/création d'entreprise soutenue par le consortium Blast [1] [2] [3]. Ce projet de maturation est par ailleurs en cours d'étude à la SATT Lutech et la SATT Paris-Saclay.

1. https ://www.blast-accelerator.com
2. https ://www.onera.fr/fr/actualites/blast-embarquement-immediat-pour-4-chercheurs-onera
3. https ://www.industrie-techno.com/article/newspace-aviation-decarbonee-et-defense-du-futur-le-programme-d-acceleration-blast-presente-sa-premiere-cohorte-de-deeptechs.65869

Chapitre 2

Etat de l'art général

Sommaire

Ce chapitre est consacré à l'état de l'art général et introduit d'abord les structures de réseaux de neurones utiles pour le traitement vidéo et les spécificités de l'apprentissage par renforcement. Puis, sont détaillés les modèles de l'état de l'art et l'implication des réseaux de neurones pour les différentes tâches de compréhension d'actions introduites dans le Chapitre 1. La présentation des bases de données utilisées dans ce manuscrit clos alors ce chapitre.

2.1 Aperçu de l'apprentissage profond

Nous introduisons dans cette section les notions algorithmiques qui concernent l'apprentissage profond, outil d'apprentissage statistique utilisé dans ce manuscrit pour aborder l'ensemble des problématiques de compréhension d'actions dans des vidéos.

2.1.1 Apprentissage par réseaux de neurones

Le champ de l'apprentissage automatique, ou *Machine Learning* (ML) (Friedman et al., 2001 ; Turing, 2009), étudie les algorithmes capables d'améliorer leurs performances, pour une tâche donnée, à l'aide de données, d'exemples, ou d'expériences. La valeur des algorithmes d'apprentissage automatique est ainsi représentée par leur capacité à répondre à un problème spécifique à partir d'un processus d'apprentissage, et non après avoir été conçu à l'aide de règles explicites pour cette tâche. Ces algorithmes sont généralement catégorisés selon le mode d'apprentissage qu'ils emploient. Les modes d'apprentissage les plus connus sont l'apprentissage supervisé (A. Singh et al., 2016), le non-supervisé (Barlow, 1989), le semi-supervisé (X. J. Zhu, 2005) et l'apprentissage par renforcement (Sutton et al., 2018).

L'apprentissage profond, ou *Deep Learning* (DL) (LeCun et al., 2015) regroupe les méthodes d'apprentissage automatique qui comportent des architectures composées de réseaux de neurones artificiels (*Artificial Neural networks* (ANN)) (Yegnanarayana, 2009). De multiples architectures ont été proposées depuis leur avènement en vision par ordinateur en 2012 (Krizhevsky et al., 2012). Aujourd'hui, les réseaux de neurones se présentent comme l'outil quasi exclusif de l'état de l'art, pour la quasi totalité des tâches de vision par ordinateur réalisables par méthodes d'apprentissage, y compris pour les tâches relatives à la compréhension d'actions. Les architectures à réseaux de neurones sont généralement construites grâce à des réseaux linéaires (Pal et al., 1992), des réseaux

convolutifs (LeCun et al., 1995), mais aussi des réseaux de neurones récurrents (Medsker et al., 2001), qui ont chacun leurs spécificités.

2.1.2 Réseaux de neurones usuels pour le traitement d'images

Perceptron multicouche. Un perceptron multicouche, ou *MultiLayer Perceptron* (MLP) dans la littérature, est un modèle de réseaux de neurones, construit de plusieurs couches du perceptron, introduit par Rosenblatt, 1958. Ce modèle, inspiré des neurosciences, est constitué d'une brique de base, le neurone artificiel.

Cette brique, aussi appelée perceptron monocouche dans le cas du MLP, est une fonction qui, à partir d'un vecteur d'entrée $\mathbf{x} \in \mathbb{R}^n$, réalise une prédiction $\mathbf{y} \in \mathbb{R}$. Cette fonction est généralement composée d'une multiplication matricielle de paramètres $\mathbf{w}$, d'un biais b et d'une fonction d'activation non linéaire σ (voir Figure 11) :

$$f(\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x} + b) \tag{2.1}$$

$\mathbf{w} \in \mathbb{R}^n$ et $b \in \mathbb{R}$ étant appelés les poids du réseau, qui seront optimisés durant l'apprentissage.

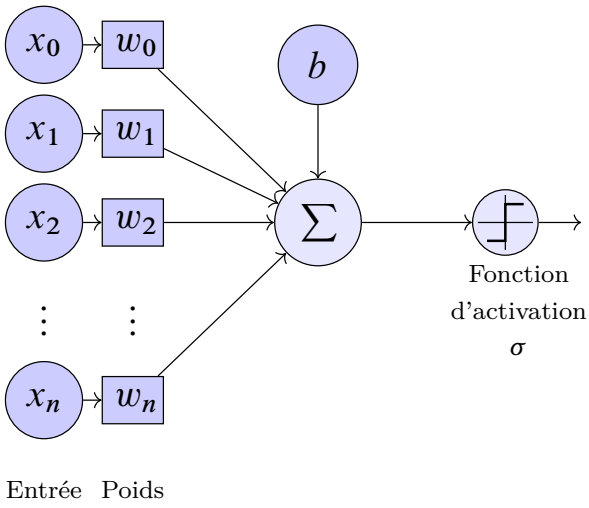


Figure 11 – **Neurone artificiel**, pour une entrée de taille n avec biais.

Un ensemble de m neurones parallèles peut être utilisé, ainsi que différentes fonctions d'activations, afin d'augmenter le pouvoir séparateur du modèle lors de son utilisation. Pour ce faire, il suffit d'adapter les opérations et les dimensions de l'équation précédente afin d'obtenir un équivalent de m neurones (voir Figure 12) :

$$f(\mathbf{x}) = \sigma(\mathbf{W}^\intercal \mathbf{x} + \mathbf{B}) \tag{2.2}$$

ou $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{B} \in \mathbb{R}^m$, $\mathbf{W} \in \mathbb{R}^{n \times m}$.

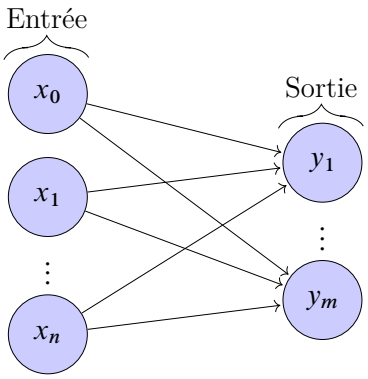


FIGURE 12 – **Graphe d'un perceptron monocouche** composé de m neurones, pour une entrée de taille n. Les opérations réalisées par les neurones sont ici représentées par les flèches.

Le perceptron multicouche est alors défini comme une superposition de multiples perceptrons monocouches. Un MLP composé de k perceptrons monocouches $f_i(\mathbf{x})$, $i \in \{1, ..., k\}$ se définit donc comme :

$$f(\mathbf{x}) = f_k(f_{k-1}(...f_1(\mathbf{x}))) \tag{2.3}$$

Le nombre k détermine la profondeur du réseau (voir Figure 13), et donne ainsi le sens au terme « profond » de l'expression « apprentissage profond ». Les $k-1$ premières fonction f sont appelées les couches cachées du réseau.

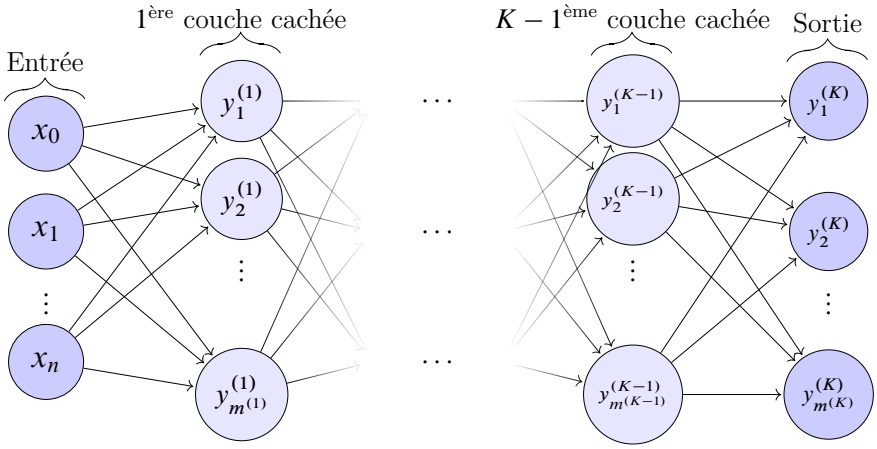


FIGURE 13 – **Graphe d'un perceptron multicouche** composé de K-couches. La $k-1^{\text{ème}}$ couche cachée contient $m^{(k-1)}$ neurones.

Réseaux de neurones convolutifs 2D. De par son architecture, le perceptron multicouche considère les éléments de l'entrée $\mathbf{x}$ de manière équivalente et ignore donc la position de l'information dans le vecteur d'entrée. Aussi, pour chaque couche cachée, chaque neurone est entièrement connecté aux neurones de la couche précédente, ce qui nécessite de limiter la taille (nombre de neurones et profondeurs) des MLP lorsqu'on travaille avec des données de grande dimension, comme les données de vision par ordinateur. Ainsi, les réseaux convolutifs (LeCun et al., 1995) ont été introduits pour disposer de réseaux plus efficaces, qui réduisent le nombre de connections entre les couches cachées et utilisent activement la structure des données.

Les *Convolutional Neural Networks* (CNN), dans la littérature, introduisent alors des connections locales, des poids partagés et des fonctions de mutualisation de l'information dites de *pooling* (Scherer et al., 2010). Les CNN sont les architectures d'apprentissage profond ayant eu l'impact le plus important dans le domaine de la vision par ordinateur.

Étant donnée une image $\mathbf{I} \in \mathbb{R}^{C \times W \times H}$, où W et H sont les dimensions de l'image et C le nombre de canaux de couleurs, une convolution en deux dimensions peut être définie de la manière suivante :

$$\mathbf{I}_{out} = \mathbf{B} + \sum_{k=1}^{C} \mathbf{K}[k] * \mathbf{I}[k] \tag{2.4}$$

où $*$ est l'opérateur de corrélation croisée 2D, $\mathbf{B} \in \mathbb{R}$ un biais, et $\mathbf{K} \in \mathbb{R}^{c \times p \times q}$, appelé le noyau de convolution. K et $\mathbf{B}$ sont les poids optimisables du réseau.

Plus précisément, si $\mathbf{K}$ est un noyau de convolution 2D, et $\mathbf{I}$ une image, le produit de convolution $\hat{\mathbf{I}}$ de $\mathbf{I}$ par $\mathbf{K}$ est donné par :

$$\hat{\mathbf{I}}_{w,h} = (\mathbf{K} * \mathbf{I})_{w,h} \tag{2.5}$$

$$= \sum_{\substack{\frac{-p}{2} \leq \alpha \leq \frac{p}{2} \\ \frac{-q}{2} \leq \beta \leq \frac{q}{2}}} \mathbf{K}_{\alpha,\beta} \mathbf{I}_{w+\alpha,h+\beta} \tag{2.6}$$

Cette formulation ne prend pas en compte les effets de bords qui sont réglés informatiquement par le *padding*.

De manière similaire à l'Equation 2.2 relative aux perceptrons, m convolutions peuvent être réalisées en parallèle :

$$\mathbf{I}_{out}[m_i] = \mathbf{B} + \sum_{k=1}^{C} \mathbf{K}[m_i, k] * \mathbf{I}[k] \tag{2.7}$$

où $\mathbf{K} \in \mathbb{R}^{m \times c \times p \times q}$

On appelle alors couche de convolution la composition d'une convolution, d'une fonction d'activation. Elle est parfois suivie d'une opération de *pooling* (voir Figure 14).

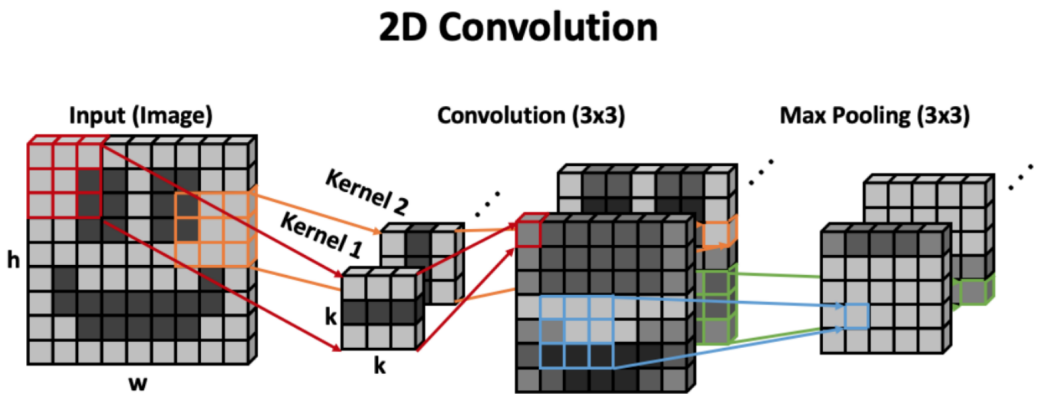


FIGURE 14 – **Exemple de convolution 2D** à canal unique, ayant deux noyaux de taille $(3, 3)$ et suivie d'un *max-pooling* de taille 3×3. Crédit image : HUTCHINSON et al., 2020.

L'opération de *pooling* est une opération de mutualisation de l'information qui permet d'augmenter la robustesse du réseau à différentes transformations spatiales, mais surtout de réduire la taille des caractéristiques en sortie des couches cachées, d'une manière non linéaire. Elle permet alors de réduire le coup de calcul global du réseau de convolutions, tout en conservant les informations calculées

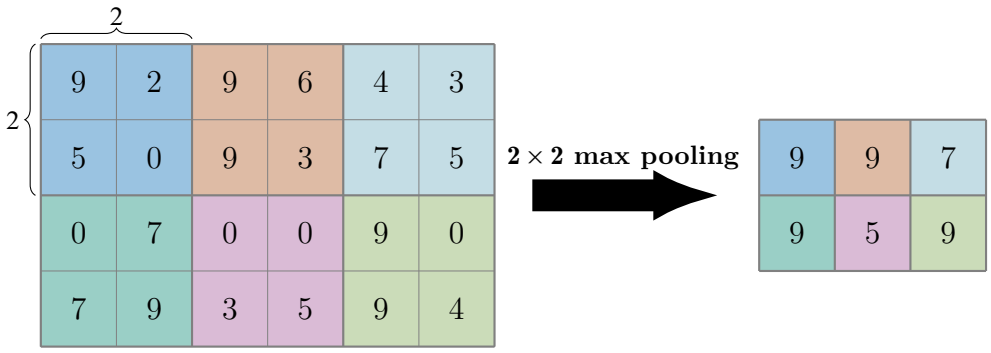


FIGURE 15 – **Opération de *max-pooling*** avec une zone de *pooling* de taille 2×2 et un *stride* 2. La résolution de l'image est dégradée d'un facteur 2.

les plus importantes. Une opération de *pooling* classiquement utilisée est le *max-pooling* (Murray et al., 2014), illustré Figure 15, et défini en 2D comme suit :

$$\text{Pool}(\mathbf{I})_{w,h} = \max_{\substack{s_0 \cdot w \leq i < s_0 \cdot w + p \\ s_1 \cdot h \leq j < s_1 \cdot h + q}} \mathbf{I}(i, j) \tag{2.8}$$

où $[s_0, s_1]$ est appelé le *stride* de l'opération et p et q définissent la taille de la zone de *pooling*.

Finalement un réseau de neurones convolutif est classiquement composé de plusieurs couches de convolution, qui vont extraire de l'information visuelle, et se termine par un perceptron multicouche, qui va traiter l'information extraite de manière spécifique à la tâche réalisée (voir Figure 16). Augmenter le nombre de couches de convolution d'un réseau permet notamment d'augmenter le champ récepteur du réseau, c'est-à-dire le nombre de pixels de l'image d'entrée dont un pixel de la sortie dépend.

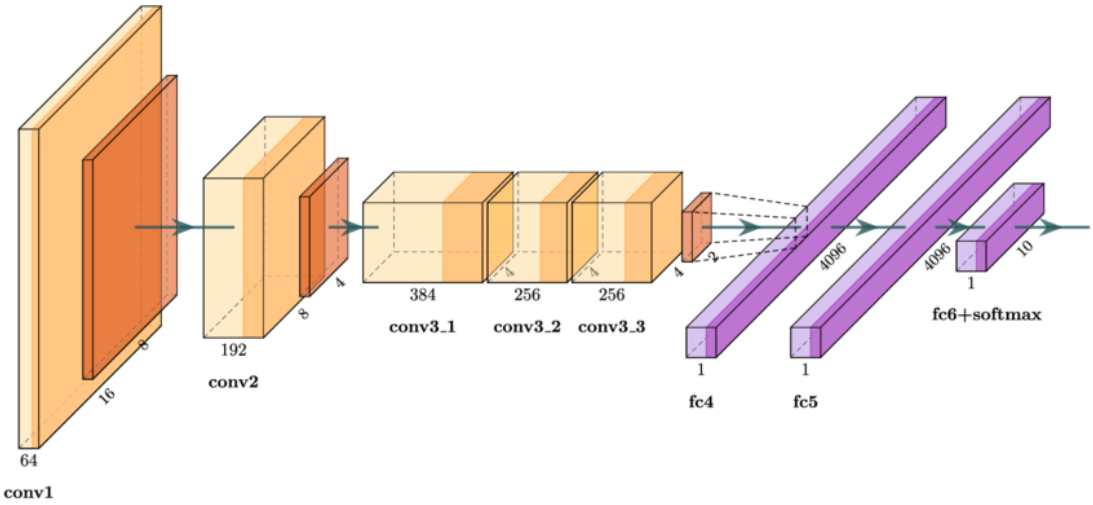


Figure 16 – **Schéma du CNN Alexnet** (Krizhevsky et al., 2012) utilisé pour la classification d'images. Les couches de convolution sont de couleur orangée, et le perceptron multicouche est de couleur violette. Crédit image : Strisciuglio et al., 2020

2.1.3 Réseaux de neurones usuels pour le traitement vidéo

Les réseaux convolutifs en 2 dimensions, introduits dans la section précédente, sont particulièrement adaptés pour le traitement de données comme les images. Une vidéo pouvant être considérée comme un empilement d'images, on peut, de manière structurelle, la considérer comme un volume d'images en 3 dimensions.

Aussi, la dimension temporelle supplémentaire introduit un caractère séquentiel et donc une structuration des données à traiter comme telle. Pour cela, la communauté utilise principalement des réseaux récurrents ou des réseaux convolutifs en 3D.

Réseaux récurrents. Un réseau récurrent (Y. Ma et al., 2019), ou *Recurrent Neural Network* (RNN) dans la littérature, contient, par contraste à un perceptron multicouche considéré comme un réseau à propagation avant, au moins un cycle dans sa structure. Cela permet d'introduire un comportement temporel dynamique et est donc utilisé pour des données présentant une séquence temporelle. En définissant $\mathbf{x}_t$ comme le vecteur d'entrée à l'instant t (dans notre cas l'image de la vidéo à l'instant t) et $\mathbf{h}_t$ comme l'état caché, qui représente la séquence des données jusqu'à l'instant t, un réseau récurrent peut alors être représenté comme dans la Figure 17.

Plusieurs architectures de réseaux récurrents existent. Les plus populaires sont le Long Short-Term Memory (LSTM) (Hochreiter et al., 1997) et une variante possédant une architecture allégée, le Gated Recurrent Unit (GRU) (Chung et al., 2014).

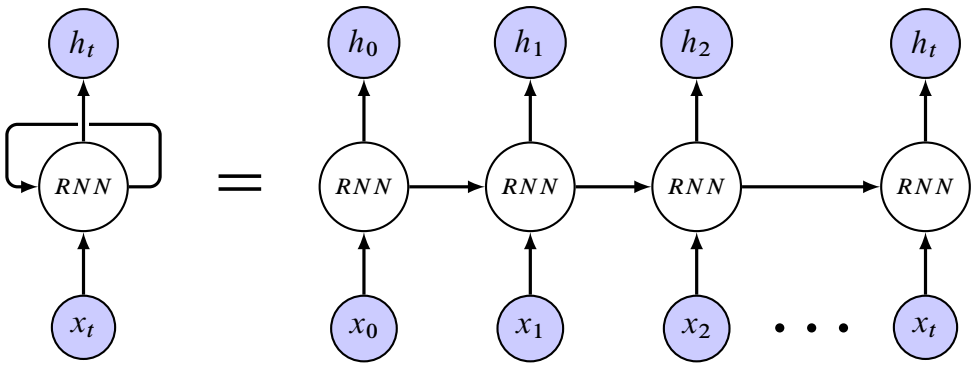


Figure 17 – **Représentation d'un réseau récurrent**. Les données $\mathbf{x}$ sont séquentiellement fournies au réseau qui en donne une représentation $\mathbf{h}$. Les poids du réseau RNN restent fixes durant toute la séquence.

Long Short-Term Memory. Le comportement temporel dynamique du LSTM (Hochreiter et al., 1997) est géré par une structure mémoire, appelée cellule, qui transporte l'information utile à travers la séquence.

A l'instant t, en définissant $\boldsymbol{c}_t$ comme la cellule, le graphe d'un LSTM est visible en Figure 18.

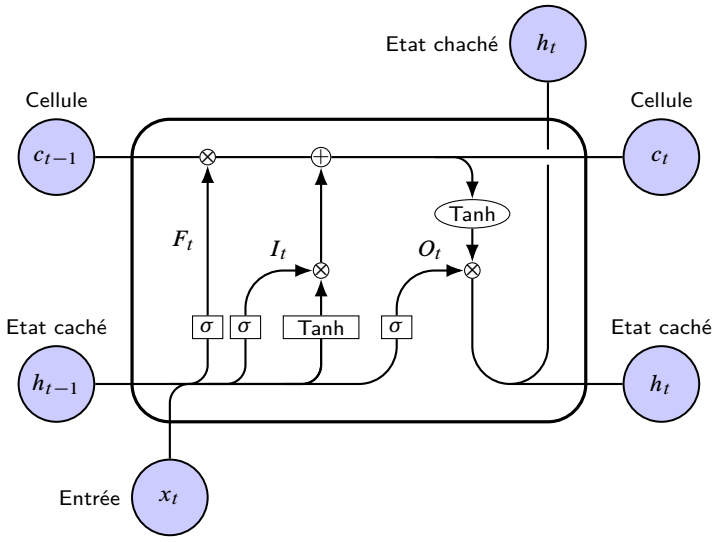


FIGURE 18 – **Graphe des opérations d'un LSTM** à l'instant t. Les poids ne sont pas représentés.

En définissant $\odot$ comme le produit matriciel de Hadamard (produit terme à terme), et $\{\mathbf{W}_k, \mathbf{U}_k, \mathbf{b}_k\}_{k\in[F,I,O,c]}$, comme les poids des différentes parties du réseau, la dynamique du LSTM à l'instant t est la suivante :

$$\mathbf{F}_t = \sigma(\mathbf{W}_F \mathbf{x}_t + \mathbf{U}_F \mathbf{h}_{t-1} + \mathbf{b}_F) \tag{2.9}$$

$$\mathbf{I}_t = \sigma(\mathbf{W}_I \mathbf{x}_t + \mathbf{U}_I \mathbf{h}_{t-1} + \mathbf{b}_I) \tag{2.10}$$

$$\mathbf{O}_t = \sigma(\mathbf{W}_O \mathbf{x}_t + \mathbf{U}_O \mathbf{h}_{t-1} + \mathbf{b}_O) \tag{2.11}$$

$$\mathbf{c}_t = \mathbf{F}_t \odot \mathbf{c}_{t-1} + \mathbf{I}_t \odot \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \tag{2.12}$$

$$\mathbf{h}_t = \mathbf{O}_t \odot \tanh(\mathbf{c}_t) \tag{2.13}$$

Le LSTM contient trois portes qui vont venir réguler l'information au passage dans le réseau. $\boldsymbol{F}_t$ est la porte d'oubli, $\boldsymbol{I}_t$ est la porte d'entrée et $\mathbf{O}_t$ est la porte de sortie. Ces trois portes vont donc réguler les informations transmises des données d'entrée $\mathbf{c}_{t-1}$, $\mathbf{h}_{t-1}$, $\mathbf{x}_t$ vers les données de sortie $\mathbf{c}_t$ et $\mathbf{h}_t$.

Gated Recurrent Unit. Le GRU (CHUNG et al., 2014) est une variante simplifiée du LSTM, nécessitant moins de paramètres et définissant directement un état des données sans la nécessité d'une cellule. Le GRU ne comporte alors que deux portes. Pour cette structure, les portes d'entrée et d'oubli sont fusionnées en une porte d'actualisation tandis que la porte de réinitialisation se substitue à la porte de sortie. $\mathbf{Z}_t$ est donc la porte d'oubli et $\mathbf{R}_t$ la porte de réinitialisation. En définissant $\{\mathbf{W}_k, \mathbf{U}_k, \mathbf{b}_k\}_{k\in[Z,R,h]}$ comme les poids des différentes parties du réseau, le graphe d'un GRU est alors présenté en Figure 19.

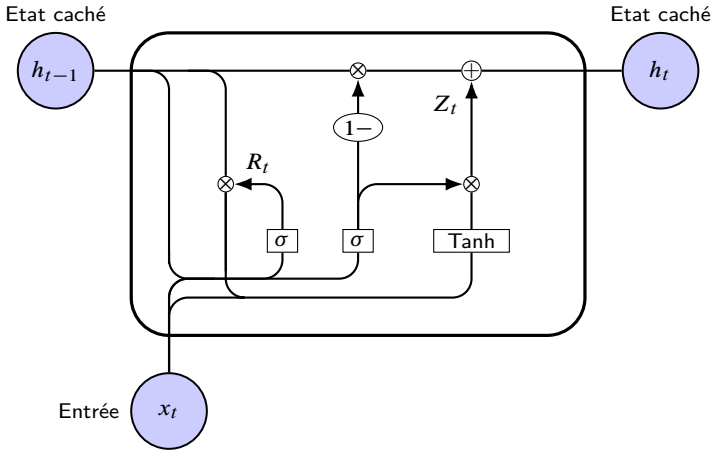


FIGURE 19 – **Graphe des opérations d'un GRU** à l'instant t. Les poids ne sont pas représentés

La dynamique du GRU à l'instant t est la suivante :

$$\mathbf{Z}_t = \sigma(\mathbf{W}_Z \mathbf{x}_t + \mathbf{U}_Z \mathbf{h}_{t-1} + \mathbf{b}_Z) \tag{2.14}$$

$$\mathbf{R}_t = \sigma(\mathbf{W}_R \mathbf{x}_t + \mathbf{U}_R \mathbf{h}_{t-1} + \mathbf{b}_R) \tag{2.15}$$

$$\mathbf{h}_t = \mathbf{Z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{Z}_t) \odot \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h(\mathbf{R}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \tag{2.16}$$

Réseau récurrent bidirectionnel. Un réseau récurrent bidirectionnel (SCHUSTER et al., 1997) est une extension d'un réseau récurrent simple. Il consiste à traiter les données de manière séquentielle dans les deux sens de traitement possible et d'ensuite aggréger les états de sortie (voir Figure 20). Ils permettent d'obtenir de l'information à la fois des états passés et des états futurs et finalement d'augmenter l'information du réseau récurrent. Ils sont donc adaptés au traitement de données vidéos hors ligne.

Réseaux de convolution en 3D. Pour traiter des données en 3 dimensions comme les vidéos, une extension des convolutions 2D a été aussi introduite. Etant donnée une video $\mathbf{V} \in \mathbb{R}^{C \times T \times W \times H}$, ou W et H sont les dimensions des images, C le nombre de canaux de couleur et T le nombre d'images de la vidéo, une convolution en trois dimensions de m noyaux peut être définie de la manière suivante :

$$\mathbf{V}_{out}[m_i] = \mathbf{B}[m_i] + \sum_{k=1}^{C} \mathbf{K}[m_i, k] * \mathbf{V}[k] \tag{2.17}$$

où $*$ est l'opérateur de corrélation croisée 3D, $\mathbf{B} \in \mathbb{R}^m$ un biais, et $\mathbf{K} \in \mathbb{R}^{m \times c \times r \times p \times q}$

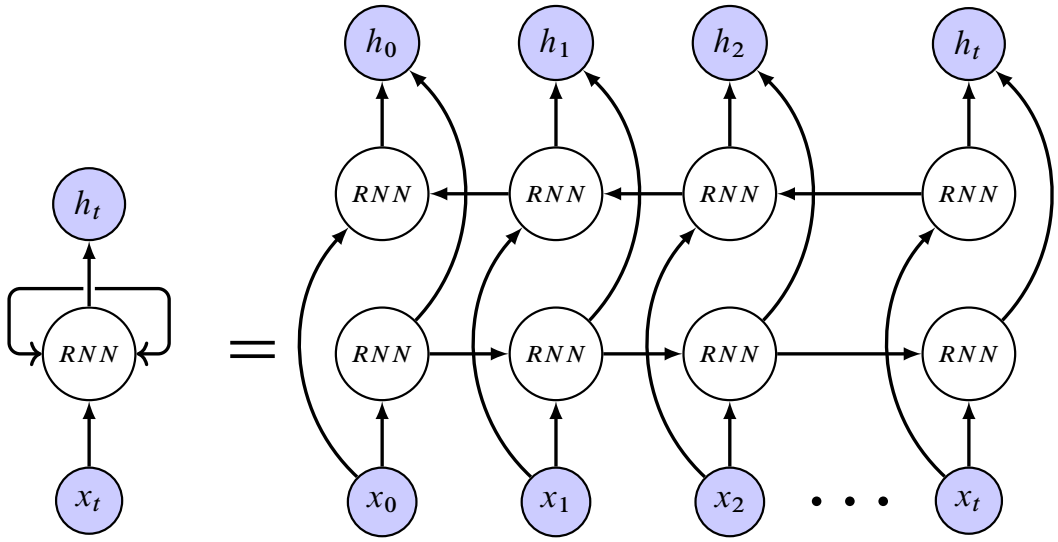


FIGURE 20 – **Représentation d'un réseau récurrent bidirectionnel**. Les données $\mathbf{x}$ sont séquentiellement fournies au réseau qui en donne une représentation $\mathbf{h}$ aggrégée du passé et de l'avenir. Les poids du réseau RNN restent fixes durant toute la séquence.

qui est appelé le noyau de convolution. K et $\mathbf{B}$ contiennent les poids optimisables du réseau.

Plus précisément, le produit de convolution $\hat{\mathbf{V}}$ de $\mathbf{V}$ par $\mathbf{K}$ est donné par :

$$\hat{\mathbf{V}}_{t,w,h} = (\mathbf{K} * \mathbf{I})_{t,w,h} \tag{2.18}$$

$$= \sum_{\substack{\frac{-r}{2} \leq \tau \leq \frac{r}{2} \\ \frac{-p}{2} \leq \alpha \leq \frac{p}{2} \\ \frac{-q}{2} \leq \beta \leq \frac{q}{2}}} \mathbf{K}_{\tau,\alpha,\beta} \mathbf{I}_{t+\tau,w+\alpha,h+\beta} \tag{2.19}$$

L'extension de la convolution 2D à la convolution 3D permet alors également d'étendre la couche de convolution en 3D (voir Figure 21) mais surtout d'étendre l'ensemble des CNN construits pour le traitement d'images à des structures capables de traiter des données vidéo (CARREIRA et al., 2017 ; HARA et al., 2017 ; JI et al., 2012 ; TRAN et al., 2015 ; Xiaolong WANG et al., 2018). Mais l'extension à des CNN permettant de traiter des vidéos implique des architectures très lourdes en termes de calculs et forcément des champs récepteurs temporels réduits. FEICHTENHOFER, 2020 estime que l'extension vidéo (Xiaolong WANG et al., 2018) du Resnet (K. HE et al., 2016) réalise environ 27 fois plus d'opérations. Pour faire face à ces enjeux, de nombreux réseaux convolutifs hybrides ont vu le jour (CHEN et al., 2018 ; J. LI et al., 2020 ; J. LIN et al., 2019 ; QIU et al., 2017 ; TRAN et al., 2019 ; L. WANG et al., 2018a).

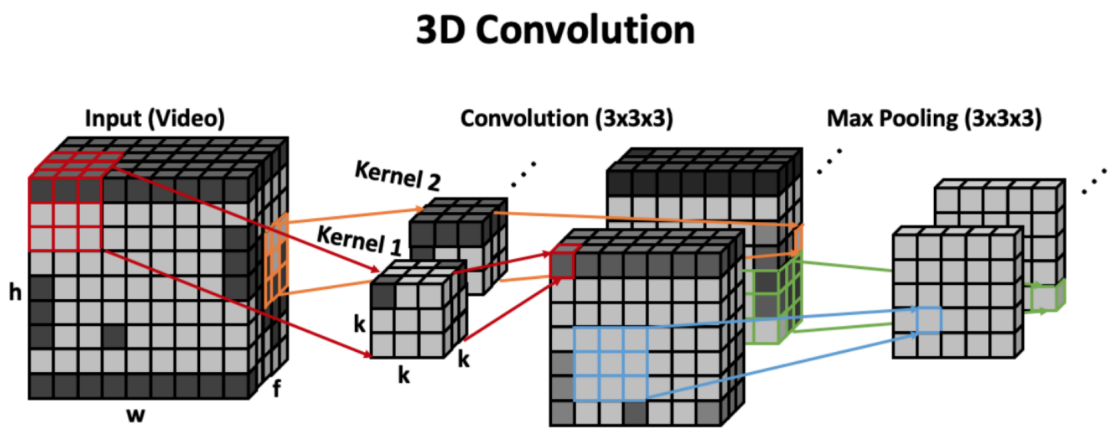


FIGURE 21 – **Exemple de convolution 3D** à canal unique, ayant deux noyaux de taille $(3, 3, 3)$ et suivie d'un *max-pooling* de taille 3×3. Crédit image : HUTCHINSON et al., 2020.

2.2 Apprentissage par renforcement

L'apprentissage par renforcement, ou *Reinforcement Learning (RL)* dans la littérature, est une sous catégorie de l'apprentissage automatique, qui traite les tâches nécessitant une prise de décision séquentielle. L'apprentissage par renforcement a donc pour objectif d'optimiser la séquence de décisions à prendre, appelée séquence d'actions. Cela est rendu possible grâce à un processus d'essais/erreurs au sein d'un environnement, réalisé par le preneur de décision, appelé agent. Lors de ce processus, l'agent reçoit une récompense quantitative au cours de la séquence, qui va lui permettre d'acquérir de l'expérience et de modifier son comportement de manière incrémentale. Un des principaux défis lors d'un processus d'apprentissage par renforcement est de trouver le meilleur compromis entre l'exploration, qui consiste à obtenir davantage d'observations sur son environnement, et l'exploitation, qui consiste à maximiser les aptitudes de l'agent compte tenu des connaissances actuelles. Le processus de communication entre un agent et son environnement, lors de l'apprentissage par renforcement, est schématisé Figure 22.

Interaction Agent-Environnement. De manière formelle, dans une problématique d'apprentissage par renforcement, un agent interagit avec son environnement de la manière suivante : A chaque pas de temps t, l'agent choisit une action à réaliser $a_t \in \mathbf{A}$. Ce choix est suivi de 3 conséquences :

— L'agent obtient une récompense $r_t \in \mathbb{R}$
— L'environnement passe d'un état $s_t \in \mathbf{S}$ à un état $s_{t+1} \in \mathbf{S}$
— Une observation ω_{t+1} du nouvel état de l'environnement est retournée à l'agent

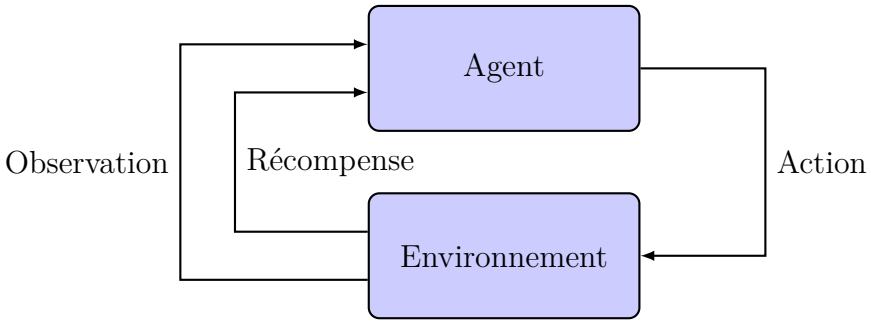


FIGURE 22 – **Cycle de communication entre un agent et son environnement lors d'un apprentissage par renforcement**. Compte tenu de ses connaissances, l'agent va réaliser une action, qui va modifier l'environnement. L'agent reçoit une récompense pour son action et une observation du nouvel état de l'environnement, qui va lui permettre de prendre une nouvelle décision d'action.

Processus de décision Markovien. Dans le cas où $\forall t, \omega_t = s_t$, on dit que l'environnement est entièrement observable et le problème de renforcement est alors défini comme un processus décisionnel de Markov, ou *Markov Decision Process (MDP)* dans la littérature. Dans ce cas, un MDP est un 5-tuple $(\mathbf{S}, \mathbf{A}, \mathbf{T}, r, \gamma)$ où :

— $\mathbf{S}$ est l'espace d'états
— $\mathbf{A}$ est l'espace d'actions
— $T : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to [0, 1]$ est une probabilité appelée fonction de transition
— $r : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to \mathbb{R}$ est la fonction de récompense
— $\gamma \in]0, 1]$ est le facteur de dévaluation

Politique. Grâce aux interactions avec l'environnement, l'agent va pouvoir construire une fonction π, appelée politique , qui définit la manière dont l'agent va choisir les actions. La politique peut être soit déterministe, soit stochastique :

— π est déterministe, et on a $\pi(s) : \mathbf{S} \to \mathbf{A}$. L'action choisie dépend alors de l'état dans lequel se trouve l'environnement.
— π est stochastique et on a $\pi(s, a) : \mathbf{S} \times \mathbf{A} \to [0, 1]$, la probabilité que l'agent choisisse l'action a en observant l'état s.

Trajectoire. Étant donnée un état initial s_0 et une politique π, la suite des couples (état, action) qui en résulte forme une trajectoire. Typiquement la trajectoire $\tau = (s_0, a_0, s_1, a_1, ...)$ avec $a_i = \pi(s_i)$ et $s_{i+1} = T(s_i, a_i)$ dans le cas déterministe. Dans le cas général, la probabilité d'une trajectoire τ sera :

$$\pi(\tau) = \prod_i T(s_i, a_i, si + 1)\pi(s_i, a_i) \tag{2.20}$$

Gain. L'objectif étant de trouver une politique optimale, on définit le gain comme la récompense à long terme pour une trajectoire τ, qui va permettre de quantifier l'apprentissage de la politique :

$$r(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \tag{2.21}$$

La valeur de γ aura pour effet de pondérer le rapport entre les actions présentes et celles à venir. De plus, si $\gamma < 1$, alors G est forcément défini.

A l'instant t, maximiser le gain revient à trouver la politique $\pi(s, a) \in \Pi$ qui maximise $V^\pi(s) : \mathbf{S} \to \mathbb{R}$ (appelée la V-valeur) et qui est définie de la manière suivante :

$$V^\pi(s) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi] \tag{2.22}$$

et ainsi, la V-valeur optimale est V^* telle que :

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s) \tag{2.23}$$

On peut aussi définir la Q-valeur, $Q^\pi(s, a) : \mathbf{S} \times \mathbf{A} \to \mathbb{R}$:

$$Q^\pi(s, a) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi] \tag{2.24}$$

et la fonction d'avantage $A^\pi(s, a) : \mathbf{S} \times \mathbf{A} \to \mathbb{R}$:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{2.25}$$

Algorithmes d'apprentissage. Dans un algorithme d'apprentissage par renforcement, un agent peut contenir :

— un modèle du MDP
— une fonction de représentation de la V-valeur
— une fonction de représentation de la politique $\pi(s)$ ou $\pi(s, a)$

Si l'algorithme se réfère à la première composante, on dit qu'il est *basé modèle*, alors que s'il dépend des deux autres, on dit qu'il *n'est pas basé modèle*.

Ainsi, pour apprendre une fonction de représentation de la V-valeur, de la politique, ou du modèle, l'utilisation de fonctions d'estimations définies par des réseaux de neurones présente l'avantage d'être adaptée aux données en grandes dimensions, ce qui est généralement le cas des environnements explorés. Dans les

algorithmes d'apprentissage par renforcement profond de l'état de l'art, seront donc utilisés des MLP, des CNN, des réseaux récurrents, ...

Dans le cas d'une politique modélisée par un réseau neuronal, on notera la politique π_θ, avec θ les poids du réseau et $\pi_\theta(\tau)$ la probabilité de la trajectoire τ étant donnés les poids θ. L'objectif lors de l'apprentissage par renforcement est alors d'optimiser les poids du réseau de manière à maximiser l'espérance du gain (pour simplifier les équations, on se place dans le cas $\gamma = 1$) :

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\tau\sim\pi_\theta(\tau)} \sum_t r(s_t, a_t) \tag{2.26}$$

En posant $r(\tau) = \sum_t r(s_t, a_t)$, on a :

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)} \sum_t r(s_t, a_t) \tag{2.27}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta(\tau)} r(\tau) \tag{2.28}$$

$$= \int \pi_\theta(\tau) r(\tau) d\tau \tag{2.29}$$

Et alors :

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, J(\theta) \tag{2.30}$$

$$= \underset{\theta}{\operatorname{argmax}} \int \pi_\theta(\tau) r(\tau) d\tau \tag{2.31}$$

La particularité de chaque algorithme d'apprentissage par renforcement profond sera la manière de venir optimiser les poids θ afin de converger vers θ^*.

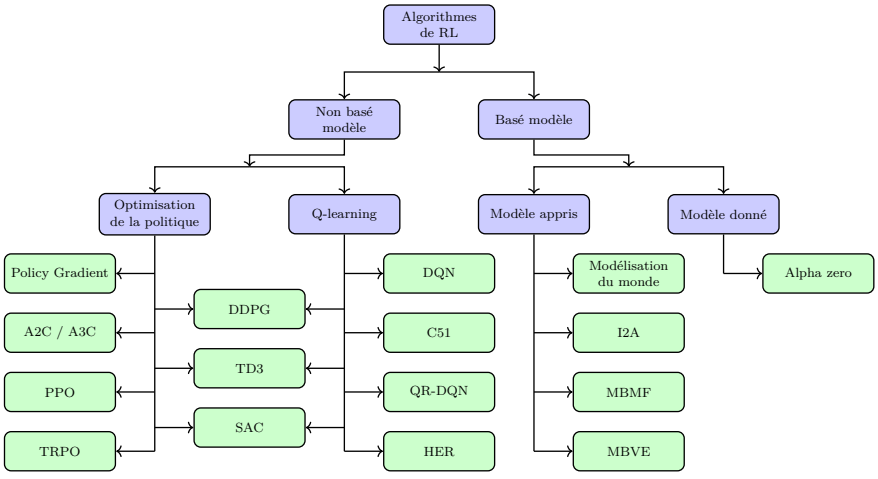


FIGURE 23 – **Taxonomie non exhaustive des algorithmes d'apprentissage par renforcement** (OPENAI, 2020).

On peut retrouver une taxonomie non exhaustive des algorithmes de l'état de l'art en RL dans la Figure 23.

L'apprentissage d'un algorithme par renforcement est un processus cyclique en 3 étapes (voir Figure 24) :

— **Génération d'un échantillon de trajectoires.** L'échantillonnage est réalisé en utilisant la politique actuelle, de manière déterministe ou stochastique.

— **Estimation du gain / Ajustement du modèle.** Les trajectoires sont évaluées. Suivant la taxonomie du modèle, cela passe par le calcul du gain, une estimation de la V-valeur ou du modèle.

— **Amélioration de la politique.** La politique est ajustée soit en propageant directement le gradient dans la politique, en ajustant la fonction de V-valeur ou le modèle. C'est à ce niveau que la stratégie d'optimisation des poids θ intervient.

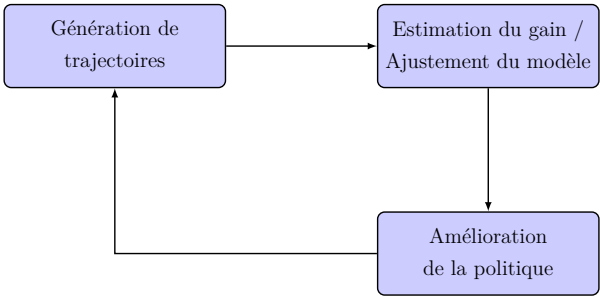


FIGURE 24 – **Processus d'apprentissage par renforcement d'une politique**. La politique est progressivement améliorée par l'acquisition d'expérience.

2.2.1 Optimisation stochastique de la politique

Les méthodes *policy gradient* cherchent à optimiser directement la politique en calculant son gradient.

En effet, sachant que :

$$J(\theta) = \int \pi_\theta(\tau) r(\tau) d\tau \tag{2.32}$$

et qu'on dispose de l'identité suivante :

$$\nabla_X f(X) = f(X) \frac{\nabla_X f(X)}{f(X)} = f(X) \nabla_X \log f(X) \tag{2.33}$$

alors

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau \tag{2.34}$$

$$= \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau \tag{2.35}$$

$$= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \tag{2.36}$$

et comme

$$\log \pi_\theta(\tau) = \sum_{t=1}^{T} \log \pi_\theta(s_t, a_t) + \sum_{t=1}^{T} \log T(s_t, a_t, s_{t+1}) \tag{2.37}$$

alors :

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t))(\sum_{t=1}^{T} r(s_t, a_t))] \tag{2.38}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} [(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t))(\sum_{t=1}^{T} r(s_t, a_t))] \tag{2.39}$$

De cette approximation découle l'algorithme REINFORCE (Sutton et al., 2018), qui travaille par échantillonnage de Monte-Carlo et permet d'optimiser directement la politique par descente de gradient (voir Figure 25) :

1. Echantillonne $\{\tau^i\}$ de $\pi_\theta(s_t, a_t)$
2. $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} [(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t))(\sum_{t=1}^{T} r(s_t, a_t))]$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

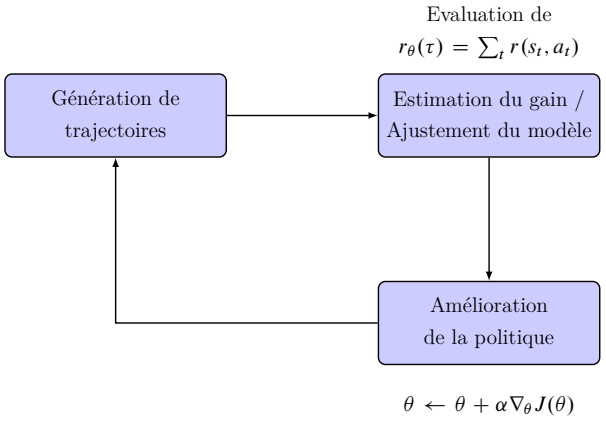


Figure 25 – **Processus d'apprentissage par renforcement par une méthode de type *policy-gradient*.**

2.2.2 Value based

Par opposition aux méthodes de type *policy gradient*, les méthodes basées sur les fonctions de valeur (*value-based*) cherchent à déterminer une politique optimale par modélisation de la fonction de valeur. Ici, l'objectif est de déterminer la valeur de chaque action, pour un état donné, afin de choisir la meilleure.

Les *Deep Q-networks* (Mnih et al., 2013) sont les algorithmes basés valeurs les plus populaires en apprentissage profond. Leur principe est d'apprendre la fonction de Q-valeur (voir Figure 26) à l'aide d'un réseau de neurones. Sachant que le nombre de couples état/action est généralement très élevé, cette manière d'apprendre la Q-valeur permet de s'affranchir de la construction d'un tableau très complexe les regroupant.

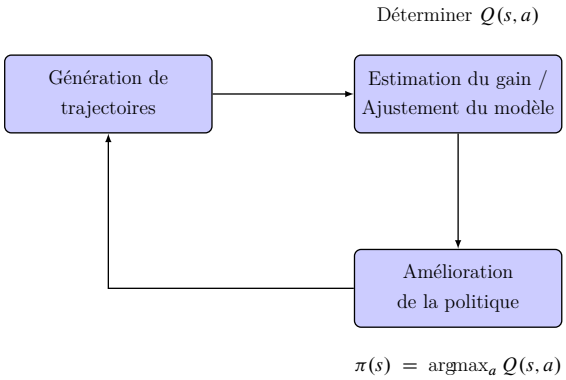


FIGURE 26 – **Processus d'apprentissage par renforcement par une méthode basée sur une fonction de valeur**.

2.2.3 Acteur-Critique

Les méthodes Acteur-Critique (Konda et al., 2000) combinent les bénéfices des méthodes *value-based* et des méthodes de *policy gradient*. Le principe est alors d'estimer d'une part une fonction de valeur et d'autre part la politique (voir Figure 27. Comme discuté précédemment, ces estimations sont réalisées avec des réseaux de neurones (Mnih et al., 2016). Les méthodes Acteur-Critique nécessitent alors deux modèles qui peuvent partager ou non des paramètres :

— Le Critique qui estime la fonction de valeur.

— L'Acteur qui estime la politique.

Les méthodes les plus populaires sont celles basées sur la fonction avantage (voir Eq. 2.25) telles que les méthodes A2C (Sutton et al., 2018), A3C (Mnih et al., 2016) ou GAE (Schulman et al., 2015).

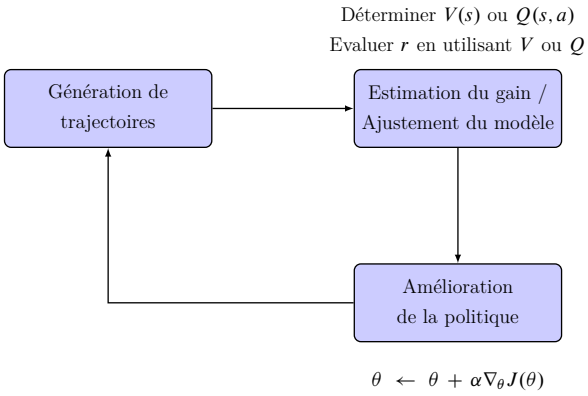


FIGURE 27 – **Processus d'apprentissage par renforcement par une méthode Acteur-Critique**.

2.3 Reconnaissance d'actions et extraction de caractéristiques

Pour les problématiques images, l'apprentissage profond a progressivement remplacé les architectures réalisant une extraction de caractéristiques réalisée de manière experte puis un traitement à l'aide d'algorithmes d'apprentissage classiques. Parfois appelées *shallow learning*, ces méthodes se sont vues concurrencées par des architectures dites profondes à base de CNN, capables d'apprendre directement des relations non linéaires très complexes dans des données structurées (voir Figure 28). De cette manière les algorithmes de l'état de l'art en traitement d'images sont quasi exclusivement appris de bout en bout, c'est-à-dire que l'ensemble de la chaîne entre l'entrée et la sortie est optimisé de manière conjointe.

Pour les problématiques vidéo, l'apprentissage profond a lui aussi complètement révolutionné l'extraction de caractéristiques visuelles. Mais, à cause du volume des données à traiter en vidéo, les algorithmes construits pour des tâches très complexes de compréhension d'actions sont encore bien souvent construits en deux temps (ALWASSEL et al., 2020 ; C.-Y. WU et al., 2019), à l'image du *shallow learning* : extraction de caractéristiques visuelles à l'aide d'un algorithme de l'état de l'art pré-entraîné, puis traitement des caractéristiques à l'aide d'un algorithme spécifique à la tâche. Pour extraire les caractéristiques visuelles des vidéos, la quasi totalité des algorithmes utilise des caractéristiques extraites grâce à des architectures CNN construites pour la classification de vidéo. Une sur-couche algorithmique vient ensuite utiliser ces caractéristiques dans une optique spécifique à la tâche abordée.

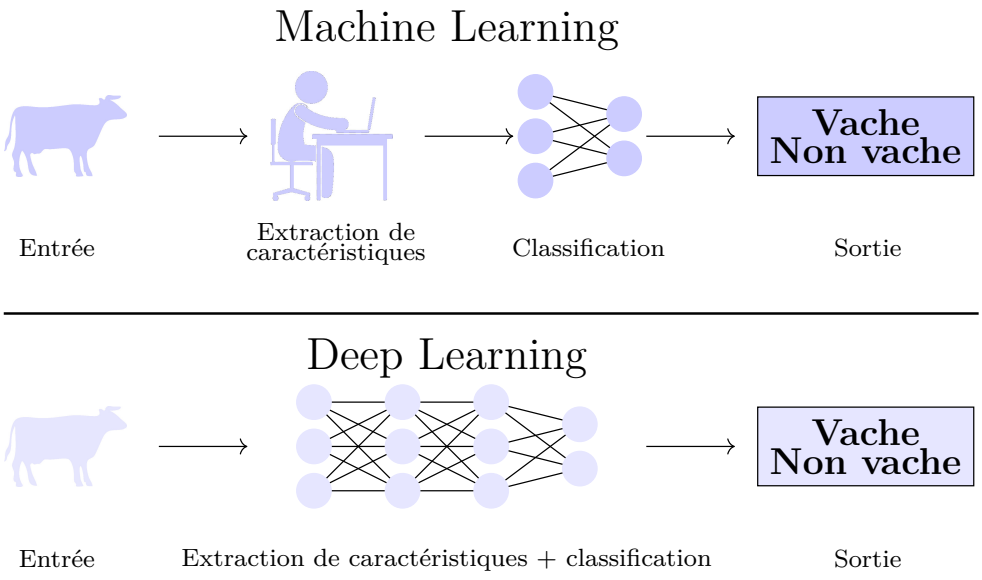


FIGURE 28 – **Evolution de la vision par ordinateur**. Le *machine learning* a progressivement été remplacé par l'apprentissage profond pour les problématiques image.

2.3.1 Flot optique

Plusieurs représentations expertes du mouvement trouvent aussi leur place dans les modèles profonds de l'état de l'art basés pour venir enrichir l'information vidéo extraite. C'est notamment le cas de l'estimation dense du champ de mouvement apparent, le flot optique (*Optical Flow (OF)* dans la littérature). Dans un contexte d'analyse vidéo, l'estimation dense du flot optique correspond à suivre chacun des pixels d'une image en examinant un voisinage spatial dans des images adjacentes pour voir si le pixel a bougé. L'estimation du flot optique fournit donc un vecteur de déplacement par pixel, en deux dimensions (voir Figure 29). Plusieurs méthodes d'estimation du flot optique existent : la méthode Lucas-Kanade (LUCAS et al., 1981), la méthode Horn-Schunck (HORN et al., 1981), l'approche TV-L1 (ZACH et al., 2007), la méthode Farneback (FARNEBÄCK, 2003), et d'autres. Le flot optique entre deux images successives s'est révélé bénéfique pour la reconnaissance d'actions (SEVILLA-LARA et al., 2018), et a depuis été plus largement utilisé pour la compréhension des actions.

2.3.2 Architectures de reconnaissance d'actions

Comme indiqué précédemment, les modèles de reconnaissance d'action ont une double utilité. Ils permettent non seulement de réaliser la tâche de classification de vidéos mais aussi de réaliser la pré-extraction des caractéristiques vidéo, pour

FIGURE 29 – **Estimation du flot optique entre deux instants consécutifs**. Les objets mobiles comportent des vecteurs de déplacement rouges. Images issues du jeu de données nuScenes (CAESAR et al., 2020).

les thématiques de compréhension d'actions incluant de la recherche temporelle.

Ces architectures sont regroupées en plusieurs familles (HUTCHINSON et al., 2020). On y trouve les architectures simple flux, les architectures double flux, les architectures à segmentation temporelle et pour finir les architectures à deux étages. Dans l'ensemble de ces modèles, si les vidéos sont traitées image par image, on parle d'échantillonnage mono-image. Si elles sont traitées par blocs d'images, on parle alors d'échantillonnage multi-image.

2.3.2.1 Architectures à simple flux

La première famille est constituée des architectures à simple flux (voir Figure 30). Ces architectures vont traiter les images vidéos soit individuellement, soit par blocs d'images. Dans le premier cas, l'architecture du réseau est composée de CNN 2D (D. HE et al., 2019 ; JIANG et al., 2019 ; KARPATHY et al., 2014) tandis que dans l'autre, on peut trouver des CNN 2D (J. LIN et al., 2019), 3D (HARA et al., 2018 ; JI et al., 2012 ; TRAN et al., 2015) ou hybrides (CHEN et al., 2018 ; J. LI et al., 2020 ; J. LIN et al., 2019 ; QIU et al., 2017 ; TRAN et al., 2019 ; L. WANG et al., 2018a). En sortie du CNN, la classe d'action est directement prédite. Les méthodes à simple flux peuvent induire des problèmes de résolution temporelle. En effet, ces méthodes ont le champ récepteur de l'architecture CNN utilisée. Ainsi, pour avoir une résolution temporelle importante, il faut avoir un

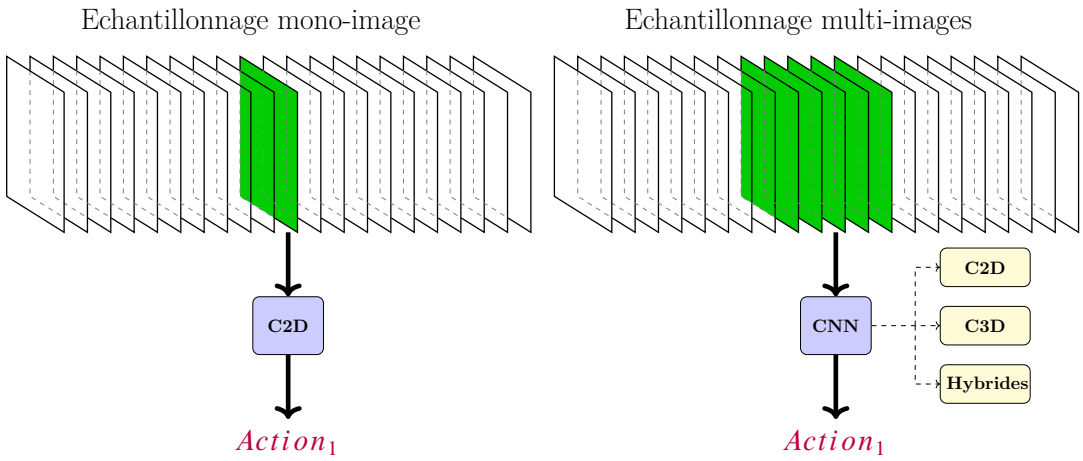


FIGURE 30 – **Exemples d'architectures de reconnaissance d'actions à simple flux**. L'échantillonnage peut être réalisé en mono-image ou multi-images. Différents modèles de CNN sont utilisés.

CNN très profond ou à gros noyaux, ce qui n'est pas forcément possible à cause du volume des données vidéo.

2.3.2.2 Architectures à double flux

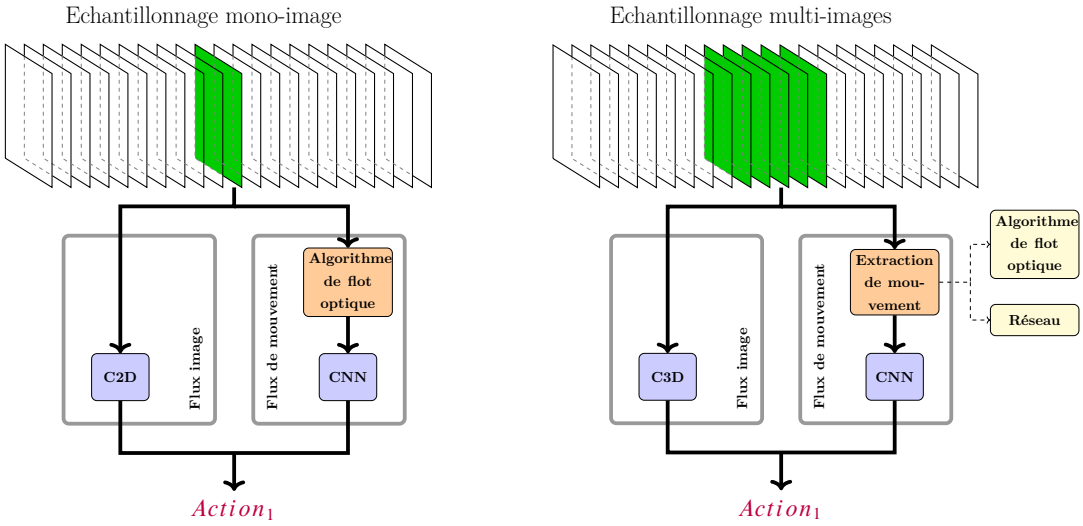


FIGURE 31 – **Exemples d'architectures de reconnaissance d'actions à double flux**. Deux flux sont traités en parallèle : un flux image et un flux de mouvement. L'échantillonnage pour le flux image peut être réalisé en mono-image ou multi-images. Le flux de mouvement utilise 2 images.

Les architectures à double flux (voir Figure 31) ont été introduites pour apporter des éléments de réponse à la problématique de la résolution temporelle. Ces architectures traitent à la fois un flux d'images et un flux contenant des ca-

ractéristiques de mouvement (Carreira et al., 2017 ; Simonyan et al., 2014), de type flot optique (Zach et al., 2007). L'apprentissage sur chacun des flux est généralement réalisé indépendamment avant que soient concaténés les résultats pour prédire la classe d'action. Cependant le calcul du flot optique doit être réalisé en amont et les méthodes mathématiques classiques sont très coûteuses en temps de calcul. Ainsi, plusieurs modèles récents calculent une représentation du mouvement grâce à un réseau profond spécifique (Crasto et al., 2019 ; Stroud et al., 2020).

2.3.2.3 Architectures à segmentation temporelle

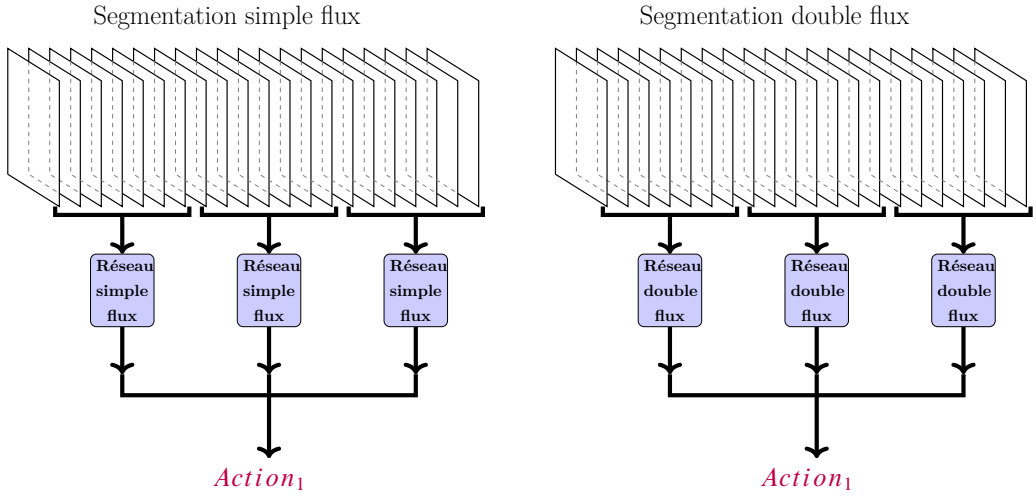


Figure 32 – **Exemples d'architectures de reconnaissance d'actions à segmentation temporelle**. La vidéo est segmentée, puis une architecture de reconnaissance à simple ou double flux est utilisée pour chaque segment. Les résultats des segments sont ensuite fusionnés.

On constate que les méthodes à simple ou double flux sont particulièrement bien adaptées à la reconnaissance d'action dans des vidéos *bornées*. Dans ce cas, les vidéos ne contiennent donc que l'actions, sans classe de fond pouvant introduire un bruit dans le flux vidéo. Pour des vidéos *non bornées*, les architectures à segmentation temporelle (voir Figure 32) sont des méthodes plus adaptées (L. Wang et al., 2016 ; 2018, b). Ces architectures divisent préalablement la vidéo en plusieurs segments avant d'appliquer à chaque portion une méthode à simple ou double flux. Une prédiction au niveau vidéo est ensuite réalisée en fusionnant les prédictions des segments (Feichtenhofer et al., 2019 ; K. Liu et al., 2018 ; Zhou et al., 2018 ; Zolfaghari et al., 2018).

2.3.2.4 Architectures à deux étages

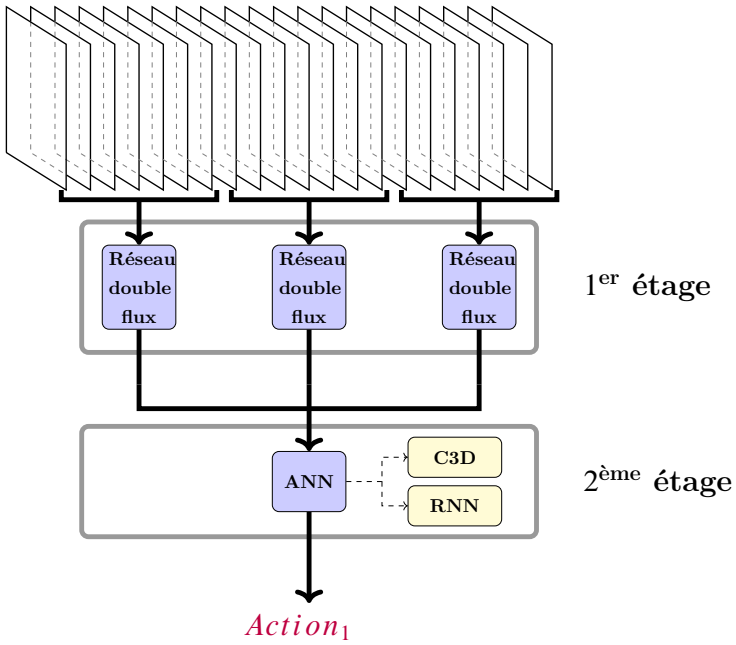


FIGURE 33 – **Exemple d'architecture de reconnaissance d'actions à deux étages**. Une architecture de reconnaissance à segmentation temporelle est suivie de la classification des caractéristiques extraites à l'aide d'un second réseau de neurone. Différents réseaux de neurones (ANN) peuvent être utilisés.

Pour finir, parmi les méthodes de compréhension d'actions, on trouve celles à deux étages. Ces méthodes utilisent un apprentissage en deux étapes (voir Figure 33). Lors de la première étape est réalisée une extraction de caractéristiques à l'aide de méthodes à segmentation temporelle, puis une seconde architecture est entraînée sur les caractéristiques des segments pour réaliser la classification. On retrouve notamment des méthodes utilisant des réseaux récurrents (DONAHUE et al., 2015 ; JOEFRIE et al., 2019 ; Xianyuan WANG et al., 2019 ; YUE-HEI NG et al., 2015) ou des sur-couches de CNN (FEICHTENHOFER et al., 2016).

2.4 Repérage d'actions

Le repérage d'actions est une problématique récente, introduite par ALWASSEL et al., 2018 et encore peu étudiée. Les auteurs introduisent des séquences de recherche d'annotateurs humains réalisant la tâche de repérage d'actions ainsi qu'une architecture RNN (voir Figure 34) qui va venir chercher à imiter ce comportement humain. Ces travaux sont à ce jours les seuls à se consacrer à la tâche de repérage d'actions.

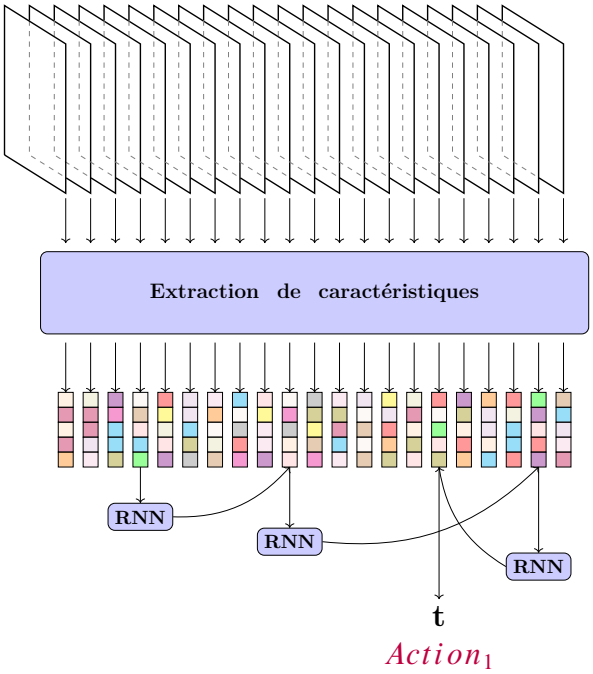


FIGURE 34 – **Exemple d'architecture de repérage d'actions**. Un réseau récurrent imite une méthode de recherche humaine des actions en se déplaçant dans la vidéo.

2.5 Proposition temporelle d'actions

Parmi les méthodes de proposition temporelle d'actions, on dénombre 3 familles d'algorithmes : les architectures descendantes, ascendantes et hybrides. Les méthodes de proposition temporelle d'actions utilisent presque exclusivement des caractéristiques pré-extraites venant d'algorithmes de l'état de l'art en compréhension d'action, plutôt que de traiter les images vidéos directement. Cette pré-extraction sera schématisée dans la suite par la notation « Extraction de caractéristiques ».

2.5.1 Architectures descendantes

Les architectures descendantes, ou *top-down* dans la littérature, prédisent des segments temporels en se basant sur des méthodes d'ancres (voir Figure 35). Les ancres peuvent êtres explicites (GAO et al., 2017b ; SHOU et al., 2016), c'est-à-dire que la portion de vidéo définie par l'ancre est extraite de la vidéo pour être caractérisée, ou bien implicite (BUCH et al., 2017 ; ESCORCIA et al., 2016) en utilisant le caractère séquentiel de réseaux de neurones récurrents. Leur particularité est donc d'extraire directement les segments de vidéos pour les caractériser.

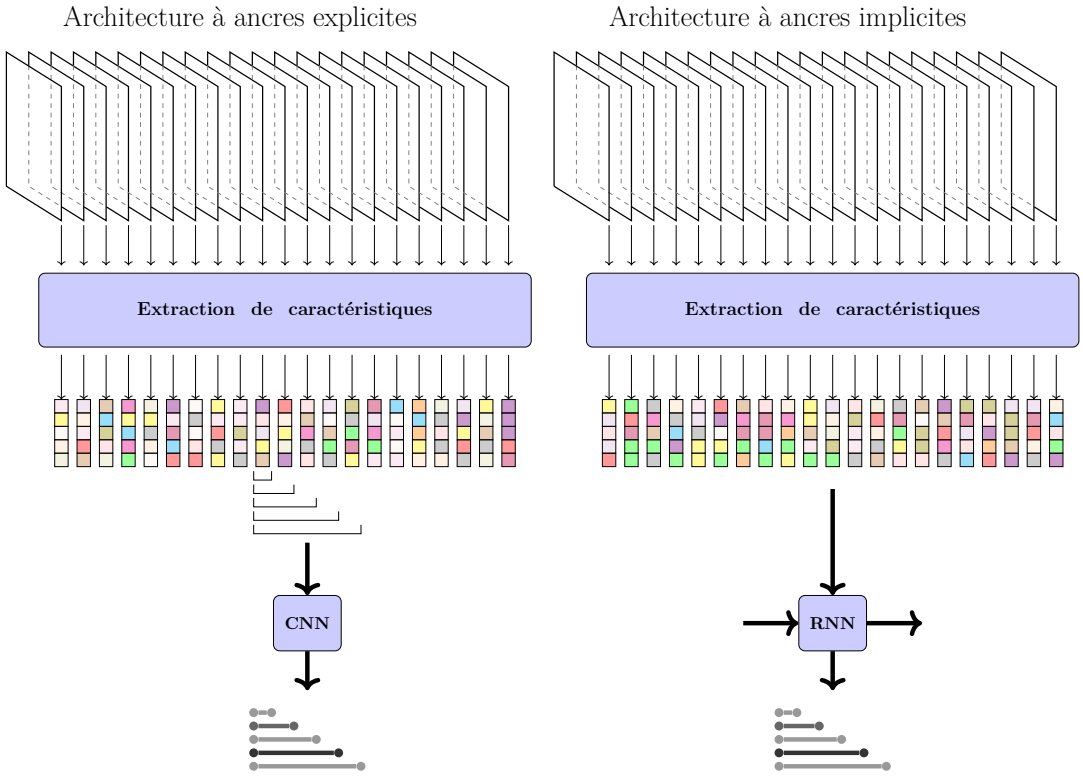


FIGURE 35 – **Exemples d'architectures descendantes de propositions temporelles d'actions**. La localisation temporelle des segments est déterminée à l'aide d'ancres explicites ou implicites.

2.5.2 Architectures ascendantes

Par opposition aux méthodes descendantes, les architectures ascendantes (voir Figure 36), ou *bottom-up* dans la littérature, vont commencer par caractériser chacune des images de la vidéo, pour ensuite utiliser ces prédictions de niveau image pour prédire des segments temporels (CHAO et al., 2018 ; T. LIN et al., 2018 ; 2019 ; H. XU et al., 2017 ; ZHAO et al., 2017). Pour chaque image, une probabilité d'appartenance aux trois classes suivantes est prédite : *action*, *début d'action*, ou *fin d'action*. Les classes *début* et *fin* d'actions sont généralement utilisées pour créer les segments et celle d'*action* est ensuite utilisée pour les caractériser.

2.5.3 Architectures hybrides

Les méthodes hybrides vont utiliser conjointement une architecture descendante et un ascendante (voir Figure 37). Pour cela, des méthodes à ancres réalisent des prédictions de niveau segment dont les scores sont ensuite affinés grâce à des prédictions réalisées au niveau image. On trouve notamment GAO et al.,

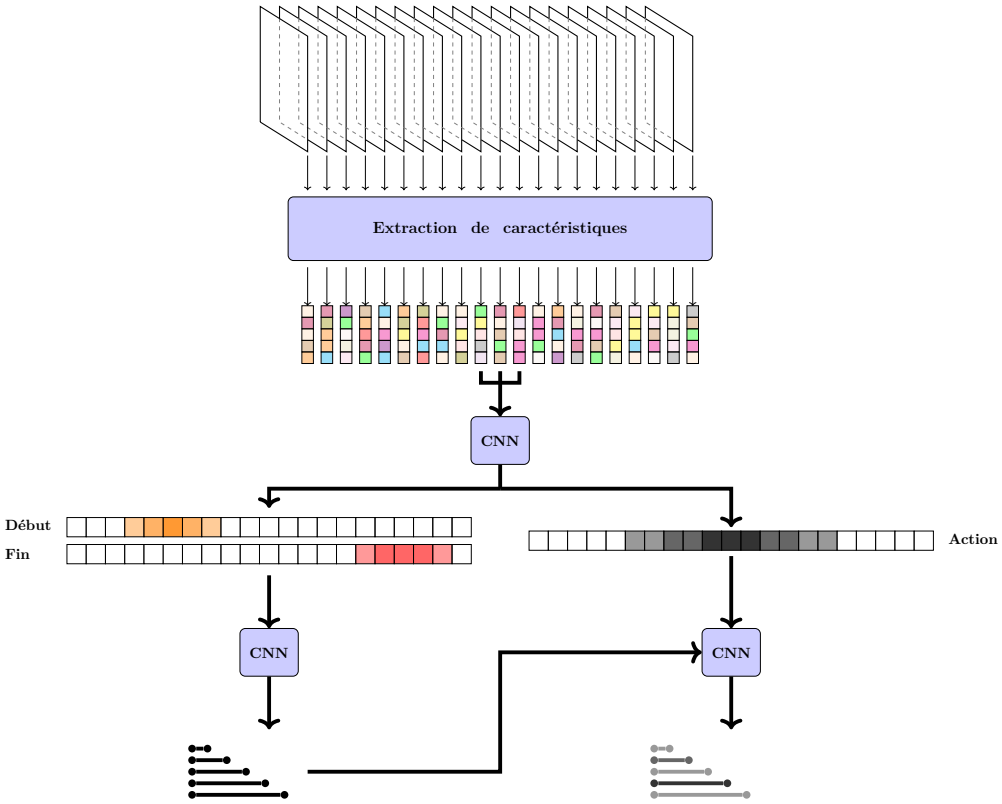


FIGURE 36 – **Exemple d'architecture ascendante de proposition temporelles d'actions**. Le modèle prédit pour chaque image si elle appartient au début, à la fin, ou au milieu d'une action. Ici, les débuts/fins probables sont appairés, puis la probabilité d'action sert à donner une probabilité globale au segment.

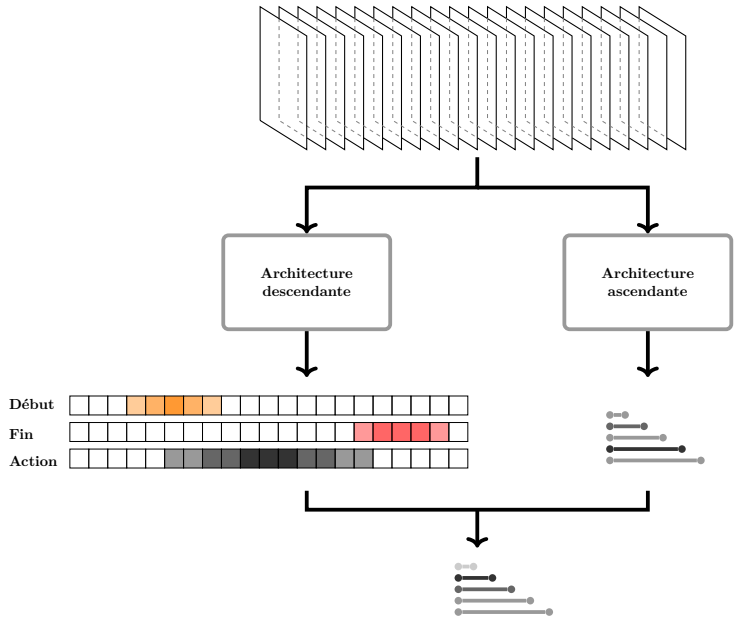


FIGURE 37 – **Exemple d'architecture hybride de proposition temporelle d'actions**. Les résultats d'une architecture descendante permettent d'améliorer les prédictions d'une architecture ascendante.

2018 ; L. Li et al., 2019 ; Y. Liu et al., 2019 ; Shou et al., 2017.

2.6 Détection d'actions

Les méthodes de détection d'actions sont souvent présentées comme des extensions des méthodes de proposition temporelle d'actions. En effet, la seule différence réside dans la caractérisation des segments temporels prédits. Contrairement à la prédiction de segments temporels, la détection va attribuer une classe d'action a chacun des segments. C'est pourquoi on trouve des méthodes à un ou deux étages.

2.6.1 Architectures à deux étages

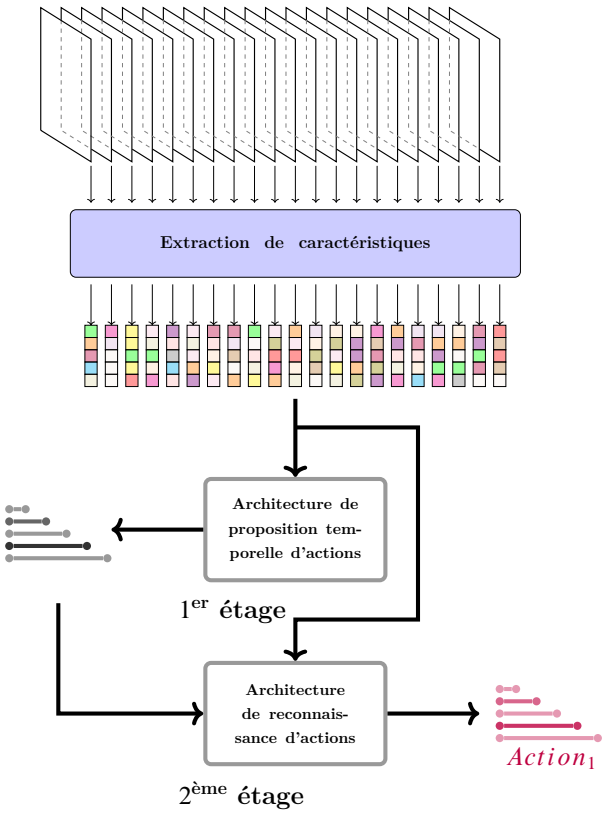


Figure 38 – **Exemple d'architectures de détection d'actions à deux étages**. Les prédictions d'une architecture de proposition temporelle d'actions sont caractérisées par une architecture de reconnaissance d'actions.

Les méthodes en deux étapes vont d'abord réaliser des propositions de segments pour ensuite les classer (voir Figure 38). Les deux tâches sont donc effectuées distinctement et successivement. Généralement elles sont réalisées par des

méthodes de l'état de l'art de prédiction de segments temporels puis de reconnaissance d'actions. Ces méthodes sont principalement utilisées dans les challenges de détection d'actions.

2.6.2 Architectures à un étage

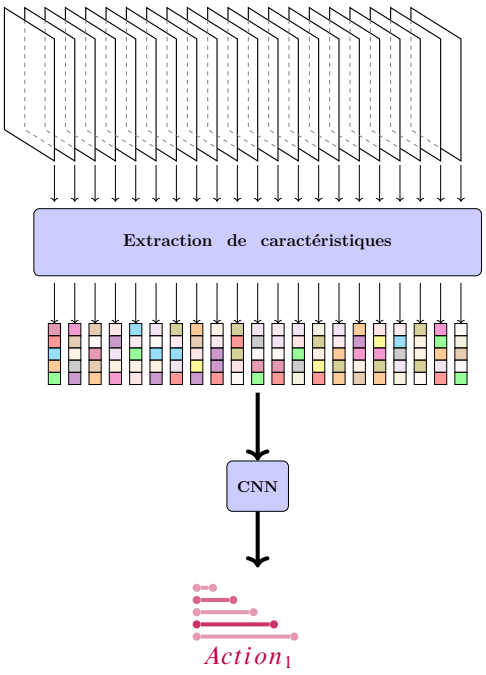


FIGURE 39 – **Exemple d'architecture de détection d'actions à un étage**. Les tâches de proposition temporelle d'actions et de classification sont réalisées conjointement.

Les méthodes en une étape (voir Figure 39) vont réaliser conjointement les étapes de prédiction de segment et de classification (BUCH et al., 2019 ; HUANG et al., 2019 ; T. LIN et al., 2017 ; LONG et al., 2019). Elles permettent notamment de tirer profit de l'apprentissage conjoint des deux tâches.

2.7 Bases de données

Depuis le début de l'utilisation de l'apprentissage machine pour les thématiques de compréhension d'action, plusieurs dizaines de bases de données vidéos annotées ont été partagées par la communauté (ÖZYER et al., 2021). Ici sont détaillées les bases de données utilisées dans ces travaux, à savoir les bases de données THUMOS14 (IDREES et al., 2017) et ActivityNet (CABA HEILBRON et al., 2015), utilisées par la communauté pour la tâche de détection d'action et

la base de données Kinetics (Kay et al., 2017) utilisée pour la reconnaissance d'actions mais principalement pour l'entraînement des réseaux utilisés pour l'extraction de caractéristiques.

2.7.1 Kinetics

La base de données Kinetics (Kay et al., 2017) est depuis son introduction la base de référence pour la reconnaissance d'actions. Chacune des vidéos de la base provient de Youtube, dure environ 10 secondes, et est annotée d'un seule classe d'actions. Les actions contenues dans les vidéos peuvent être des interactions entre humains, ou des interactions entre humains et objets. La base se décline en 3 versions : Kinetics 400, Kinetics 600 et Kinetics 700. Leur distinction se fait par le nombre de classes d'actions qu'elles contiennent. Ainsi, la version Kinetics 400 continent plus de 300k vidéos pour 400 classes d'actions (voir Figure 40). La version Kinetics 600 est une extension et contient près de 500k vidéos pour 600 classes d'actions. Pour finir, Kinetics 700 est une extension de la version 600 et contient environ 650k vidéos, pour 700 classes d'action.

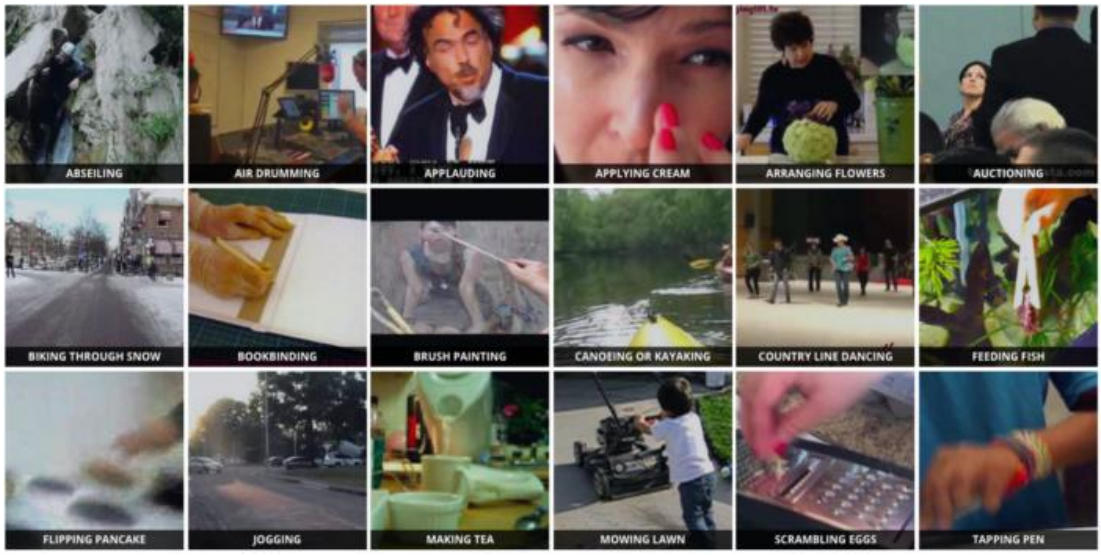


FIGURE 40 – **Exemple de classes d'actions du jeu de données Kinetics**.

Kinetics 700 est une des bases de données publiques contenant le plus de vidéos mais aussi celle ayant la plus grande diversité de classes d'actions. Cela la rend très utilisée pour le pré-entraînement de réseaux avant *finetuning* sur des jeux de données plus petits (de manière analogue à ImageNet (Deng et al., 2009) pour l'image), ou pour l'entraînement de réseaux à des fins d'extractions de caractéristiques.

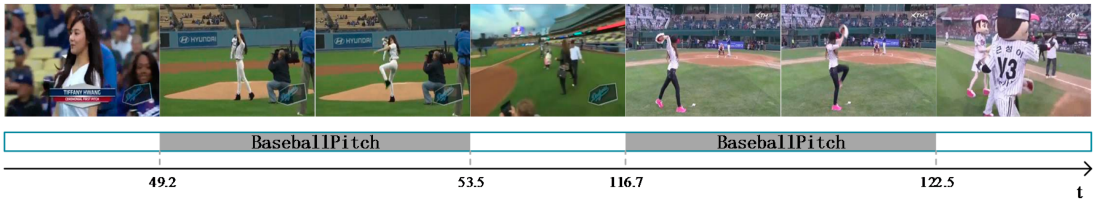


FIGURE 41 – **Exemple de vidéo annotée issue du jeu de données THUMOS14**.

2.7.2 THUMOS14

THUMOS14 (IDREES et al., 2017) est une des bases de données de l'état de l'art pour les problématiques de proposition temporelle d'actions et de détection. La base contient 410 vidéos non-bornées issues de Youtube, d'une moyenne de 3 minutes, et annotées pour 20 classes d'actions sportives. Chaque vidéo (voir Figure 41) contient en moyenne 16 actions qui peuvent être de nature différente. Traditionnellement, 200 vidéos sont utilisées pour la phase d'entraînement et 210 pour la phase de validation des algorithmes d'apprentissage.

2.7.3 ActivityNet

ActivityNet (CABA HEILBRON et al., 2015) est un autre jeu de données de vidéos non bornées provenant de Youtube, utilisé pour les thématiques de proposition temporelle d'actions et de détections. ActivityNet a été décliné en deux versions : la version 1.2 et la version 1.3. La version 1.2 comporte 9682 vidéos contenant 100 classes d'actions et divisées en 3 sous ensembles de 4819 vidéos pour l'entraînement, 2 383 pour la validation et 2 480 pour le test d'algorithmes d'apprentissage. La version 1.3 est une extension de la précédente et contient 19 994 vidéos pour 200 classes d'actions. Le jeu de données est également divisé en 3 sous ensembles de 10 024 vidéos pour l'apprentissage, 4 926 pour la validation et 5 044 pour le test d'algorithmes d'apprentissage. Chaque vidéo contient en moyenne 1,5 actions d'une classe d'action unique (voir Figure 42).

FIGURE 42 – **Exemple de vidéo annotée issue du jeu de données ActivityNet**.

2.8 Positionnement de la thèse

Dans ce chapitre, nous avons commencé par introduire les outils d'apprentissage profond et certaines spécificités de l'apprentissage par renforcement nécessaires à la compréhension de l'état de l'art des thématiques de compréhension vidéo et permettant de traiter les spécificités des signaux vidéos.

Dans un second temps, nous avons présenté les méthodes d'apprentissage profond qui constituent aujourd'hui l'état de l'art de la compréhension d'action dans des vidéos. Une taxonomie générale des méthodes de reconnaissance d'actions a été exposée puisque ces méthodes permettent d'extraire des caractéristiques vidéos utiles aux autres tâches de compréhension d'actions et notamment le repérage, la localisation temporelle et la détection.

Il se trouve que les architectures de localisation temporelle ascendantes donnent aujourd'hui les meilleures performances en détection. Cependant on peut s'interroger sur le fait que ces performances soient biaisées par une métrique favorisant la localisation au détriment de la sémantique. Le positionnement de cette thèse a consisté à se demander si l'utilisation d'un repérage sémantique précis permettant par ailleurs d'avoir une information faiblement localisée ne permettrait pas de meilleures performances finales. Dans cette optique, les contributions suivantes ont été apportées :

— Une nouvelle architecture de repérage d'actions, utilisant une méthode d'apprentissage par renforcement, et permettant une meilleure prise en compte du problème abordé ;
— Une nouvelle architecture descendante de détection d'actions basée sur un *a priori* de repérage.
— Des architectures hybrides de détection d'actions qui cherchent à équilibrer des approches ascendantes et descendantes.

Chapitre 3

Repérage d'actions

Sommaire

3.1 Introduction

L'extraction sémantique la plus robuste de l'état de l'art est à ce jour le repérage d'actions, ou *Action Spotting* (AS) dans la littérature. Elle est aussi la tâche de compréhension d'actions la plus récente, introduite par Alwassel et al., 2018.

Cette tache consiste à trouver toute occurrence temporelle d'action dans une vidéo, en observant le moins d'images possible. Cette méthode diffère de la détection d'action de deux façons. Premièrement, le repérage d'actions nécessite de

trouver uniquement une seule image dans le segment temporel de l'instance d'action (appelée *spot frame*), plutôt que des marqueurs de début et de fin. Deuxièmement, le repérage d'actions s'intéresse aussi à l'efficacité du processus de recherche et donc à minimiser le nombre d'images visualisées.

Ainsi, le repérage d'actions peut correspondre à une localisation très approximative des actions, au sens de la tâche de détection. De fait, disposer de segments temporels issus d'une détection d'actions parfaite, permet de manière évidente de réaliser un repérage parfait. Par contre, extraire des *spot frames* qui répondent parfaitement à la tâche de repérage, ne permet que de disposer d'une information partielle sur la localisation des actions, puisqu'aucune information sur les bords des actions n'est disponible. C'est d'ailleurs pour utiliser le repérage comme une information *a priori* pour réaliser la tâche de détection qu'Alwassel et al., 2018 ont introduit la tâche de repérage d'actions.

Pour autant ce n'est pas une approche purement théorique. Par exemple, le processus de repérage d'actions est une approche rapide permettant de compter et caractériser les actions dans des vidéos sans contrainte de précision temporelle stricte.

Une remarque importante est que contrairement à la tâche de sélection d'images clées (*key frames*), qui vise à extraire des images saillantes et réaliser un résumé visuel des vidéos, le repérage d'actions réalise un résumé sémantique des actions dans la vidéo. Ansi, les *spot frames* résultantes ne sont pas censés être des *key frames*.

Plus formellement, le but est de naviguer dans une vidéo et d'y sélectionner des *spot frames* résumant la sémantique de l'activité humaine dans les vidéos (voir Figure 48). Il est alors intéressant d'optimiser à la fois la qualité des *spot frames* et la proportion d'images non visualisées, appelée *skip ratio*.

Soit $\mathbf{V} = \boldsymbol{x}_1, ..., \boldsymbol{x}_T$ avec T images (ou T tronçons de vidéo), une vidéo où $\boldsymbol{x}_t$ dénote le vecteur de caractéristiques de la t-ième image (ou du t-ième tronçon). Cette vidéo est associée à une vérité terrain $\mathbf{Q}$ composée d'un ensemble de K actions d'intérêt temporellement ordonnées tel que $\mathbf{Q} = \{[s_k, e_k], c_k\}_{k=1}^{K}$ où s_k, e_k et c_k sont respectivement l'instant initial, l'instant final et la classe d'action du segment de vérité terrain k.

Alors la tâche de repérage d'actions consiste à produire un ensemble de K *spot frames* associées à leur classe et leur probabilité :

$$\mathcal{V} = \{(\tau_1, l_1, \alpha_1), ..., (\tau_K, l_K, \alpha_K)\} \tag{3.1}$$

tels que $\forall k \in \{1, ..., K\}$, $\tau_k \in [s_k, e_k]$, $\alpha_k = c_k$ et l_k représente la probabilité

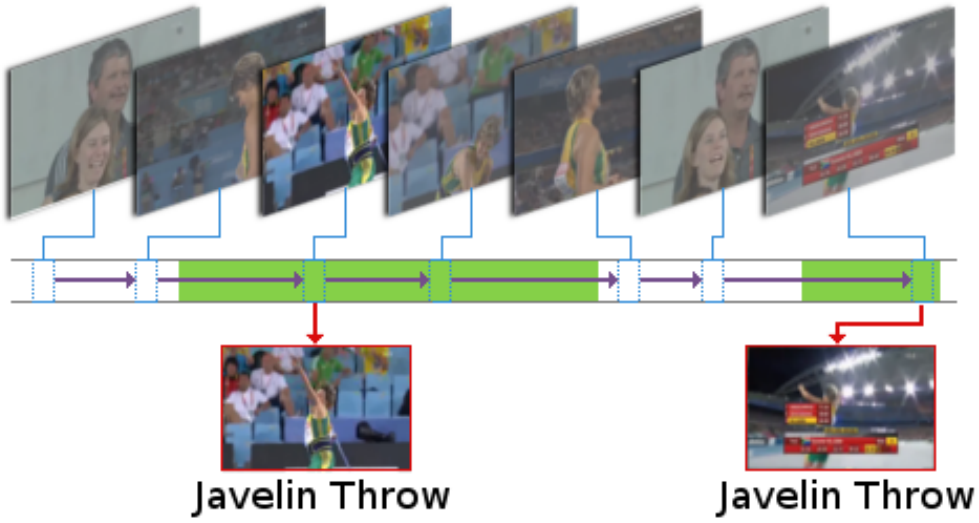


FIGURE 43 – **Vue d'ensemble de l'environnement d'extraction de *spot frames***. Le modèle extrait une *spot frame* par occurrence d'action (en vert sur la figure) en parcourant la vidéo de manière éparse.

que l'image τ_k soit une *spot frame*. L'objectif est donc d'extraire une et une seule image par action reflétant sa sémantique.

3.2 Métrique

Dans la section précédente a été défini le problème de repérage d'actions, introduit par ALWASSEL et al., 2018. Il a été précisé que le repérage peut correspondre à une tâche de détection avec localisation approximative des actions. Cependant, comme indiqué, *Alwassel et al.* utilisent le repérage comme un *a priori* pour une réaliser une détection. Ainsi la valeur du résumé sémantique extrait par les *spot frames* n'a jamais été quantifiée dans la littérature.

Or, de par les applications possibles, il est pertinent de pouvoir quantifier la qualité des *spot frames* extraites.

Une des contributions de ce manuscrit est d'introduire une manière d'évaluer la qualité de l'extraction sémantique de *spot frames*, prenant en compte non seulement la position des *spot frames*, mais aussi leur nombre et leur classe d'action. En effet, pour un segment d'action donné, la sélection de toute image appartenant à cet intervalle temporel en tant que *spot frame* est correcte. Par conséquent, deux *spot frames* extraites peuvent être différentes mais tout aussi valables. La prédiction est considérée comme parfaite si et seulement s'il y a exactement une *spot frame* extraite par segment de vérité terrain, avec une classification correcte (per-

mettant par exemple un comptage parfait). Le problème est donc asymétrique : alors que la vérité terrain est composée d'un ensemble de segments temporels, la prédiction est un ensemble de *spot frames*.

Ainsi, afin de proposer une métrique non biaisée reflétant la qualité des *spot frames* extraites, elle a été construite à partir de la *mean Average Precision* (mAP), une métrique utilisée par l'état de l'art en détection d'objets, mais aussi en localisation/détection temporelle d'actions. Cette métrique est par exemple utilisée pour les challenges de vision par ordinateur Pascal (Everingham et al., 2015), THUMOS14 (Idrees et al., 2017), ou encore AVA (Gu et al., 2018). Ainsi, nous l'avons adaptée au repérage d'actions et nous avons proposé un nouveau script d'évaluation disponible publiquement pour garantir une évaluation équitable (voir Algorithme 1).

Pour calculer cette métrique, pour chaque classe d'action, les *spot frames* prédites sont triées par ordre décroissant de probabilité. Ensuite, l'intersection entre les horodatages des *spot frames* et les segments de vérité terrain est calculée de manière itérative. Une *spot frame* est alors considérée comme un repérage correct si et seulement si son horodatage intersecte un segment de vérité terrain, a la bonne classe d'action et est la première à correspondre avec le segment de vérité terrain. Une *spot frame* qui ne correspond à aucun segment de vérité terrain, ou qui n'est pas la première à correspondre, est une fausse alarme. Enfin, un segment de vérité terrain qui ne correspond à aucune *spot frame* correspond à une détection manquée. Ensuite, pour chaque classe d'action, les valeurs de précision et de rappel sont calculées pour chaque position dans la liste triée de *spot frames*, dans le but de tracer la courbe de précision/rappel. Enfin, l'aire sous la courbe couverte par ces points, après lissage, moyennée sur toutes les classes d'activité, est retournée comme la qualité du spotting (mAP).

3.3 Repérage et détection à faible tIoU

L'indice de Jaccard (Jaccard, 1901) est une mesure de similarité entre deux ensembles, aussi connu sous le nom d'*Intersection over Union* (IoU). L'IoU est connue et utilisée notamment pour la mesure de similarité entre deux surfaces dans les problèmes de détection d'objets. Pour mesurer la similarité entre deux segments temporels, l'indice de Jaccard est aussi utilisé et prend le nom de *temporal Intersection over Union* (tIoU). La tIoU constitue alors une mesure de similarité pour deux ensembles en 1 dimension (voir Figure 44). La tIoU entre un segment prédit $[\hat{s}, \hat{e}]$ et un segment de la vérité terrain $[s, e]$ se calcule de la

Algorithme 1 : Calcul de la mAP de repérage

Entrées : $\mathcal{V} = \{\tau_n, l_n, \alpha_n\}_{n=1}^{N}$ l'ensemble des *spot frames* extraites.
$\{[s_n, e_n]\}_{n=1}^{N}$ les segments de vérité terrain de la vidéo.
μ le seuil de tIoU.
C le nombre de classes d'actions de la vérité terrain.

$\text{AP} = \{0\}_{i=1}^{c}$
/* Initialisation de l'AP pour chaque classe d'action */
pour $c = 1, ..., C$ **faire**
  Calculer σ_c
  /* La liste des positions n, triées par ordre décroissant de l_n, telles que $\alpha_n = c$ et $l_{\sigma(n+1)} \geq l_{\sigma(n)}$ */
  $\text{Tp} = \{0\}_{i=1}^{\#\sigma_c}$, $\text{Fp} = \{1\}_{i=1}^{\#\sigma_c}$
  /* Initialisation des vecteurs de vrais positifs et faux positifs */
  $K_c = \#\{\mathbf{Q}_c = \{[s_k, e_k] \in \mathbf{Q}, c_k = c\}\}$
  /* Initialisation du blocage des segments temporels de vérité terrain */
  $\text{lock} = \{\text{False}\}_{i=1}^{K_c}$
  pour $n = 1, ..., N$ **faire**
    pour $k = 1, ..., K_c$ **faire**
      si $s_k \geq \tau(\sigma_c(n)) \geq e_k$ **et** $\neg lock[k]$ **alors**
        $\text{Tp}[\sigma_c(n)] = 1$
        $\text{Fp}[\sigma_c(n)] = 0$
        $\text{lock}[k] = \text{True}$
      sinon
      fin
    fin
    /* Calculer la liste de sommes cumulées des vrais positifs et des faux positifs */
    $\text{Tp}_{\text{cumsum}}[n] = \text{somme}(\text{Tp}[: n+1])$
    $\text{Fp}_{\text{cumsum}}[n] = \text{somme}(\text{Fp}[: n+1])$
    /* Calcul de la liste cumulée du rappel et de la liste cumulée de la précision */
    $\text{Rappel}_{\text{cumsum}}[n] = \frac{\text{Tp}_{\text{cumsum}}[n]}{K_c}$
    $\text{Precision}_{\text{cumsum}}[n] = \frac{\text{Tp}_{\text{cumsum}}[n]}{\text{Tp}_{\text{cumsum}}[n] + \text{Fp}_{\text{cumsum}}[n]}$
  fin
  /* Calcul de l'AP (aire sous la courbe précision/rappel après lissage) */
  $\text{AP}[c] = \text{AveragePrecision}(\text{Rappel}_{\text{cumsum}}, \text{Precision}_{\text{cumsum}})$
fin
$\text{mAP} = \frac{\sum_{c=1}^{C} \text{AP}[c]}{C}$

Sorties : mAP

manière suivante :

$$tIoU([\hat{s},\hat{e}],[s,e]) = \frac{|[\hat{s},\hat{e}] \cap [s,e]|}{|[\hat{s},\hat{e}] \cup [s,e]|} \tag{3.2}$$

$$= \frac{|[\hat{s},\hat{e}] \cap [s,e]|}{|[\hat{s},\hat{e}]| + |[s,e]| - |[\hat{s},\hat{e}] \cap [s,e]|} \tag{3.3}$$

$$= \frac{\min(e,\hat{e}) - \max(s,\hat{s})}{\max(e,\hat{e}) - \min(s,\hat{s})} \tag{3.4}$$

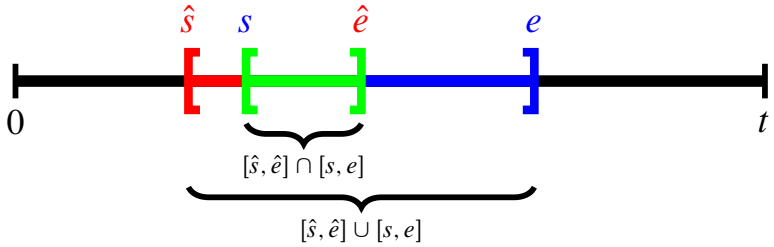


FIGURE 44 – Illustration de la tIoU.

En détection, et notamment en détection d'actions dans des vidéos, la mAP est calculée relativement à un niveau de tIoU. Ce niveau correspond à un seuil de tIoU que doit dépasser une prédiction avec un segment temporel de la vérité terrain, pour être considéré comme une bonne détection, autrement dit un vrai positif. Ainsi, plus ce seuil est élevé et plus les bords des prédictions doivent être précis.

Or, nous avons introduit, qu'en première approximation, le repérage d'actions peut être associé à une détection à faible tIoU, sachant que les deux tâches correspondent à une localisation approximative. Lorsqu'on utilise les algorithmes originellement construits pour réaliser la tâche de détection à des fins de repérage d'actions (voir Section 4.3 pour la mise en oeuvre), on s'attend donc à trouver des niveaux de performance similaires entre les deux tâches. Or, il apparaît que la mAP de repérage (voir adaptation des résultats de détection en résultats de repérage en Section 3.5.2) est inférieure à la mAP de détection de plus de 15% pour l'utilisation d'un seuil de tIoU de 0.1, considéré comme seuil d'une détection approximative (voir Tableau 3.1).

Il est donc naturel de se questionner sur les raisons de cette différence et de se demander si les performances des algorithmes de détection lorsqu'ils sont utilisés pour le repérage constituent une limite haute qu'il n'est pas possible de dépasser, ou s'il faut développer des algorithmes spécifiques à la problématique de repérage. Pour répondre à cette interrogation, nous introduisons, dans la section suivante, un nouvel algorithme de repérage d'action, nommé ActionSpotter.

THUMOS'14		
Approche	Détection mAP@0.1	Repérage mAP
M-CNN (Shou et al., 2016)	47.7	41.2
TURN (Gao et al., 2017b)	54.0	44.8
R-C3D (H. Xu et al., 2017)	54.5	52.2
CBR (Gao et al., 2017a)	60.1	50.1

Table 3.1 – **Comparaison du repérage et de la détection à faible tIoU sur le jeu de validation THUMOS14.** Seconde colonne : résultat en mAP pour les détecteurs de l'état de l'art. Dernière colonne : résultats en mAP pour la tâche de repérage.

3.4 ActionSpotter : Extracteur de *spot frames* sémantiques basé sur un Acteur-Critique

Comme introduit, les *spot frames* extraites ne sont pas censées être des images clés, et ne sont pas annotées dans la vérité terrain (uniquement composée des segments temporels).

Afin d'effectuer un repérage d'actions entièrement supervisé, Alwassel et al., 2018 a introduit des annotations coûteuses sur le comportement des annotateurs humains lors du repérage d'actions. Mais nous pensons que le recours à l'apprentissage supervisé n'est pas évident, car le problème est asymétrique, et proposons l'utilisation de l'apprentissage par renforcement pour produire des *spot frames* à partir de segments temporels uniquement, sans annotations supplémentaires. Cet algorithme, appelé ActionSpotter, peut traiter des vidéos contenant plusieurs classes d'actions avec de multiples occurrences.

3.4.1 Construction

Nous avons conçu une architecture appelée ActionSpotter contenant trois réseaux qui travaillent parallèlement pour à la fois parcourir en ligne les images de la vidéo, et extraire des *spot frames* reflétant l'activité humaine. L'état global de notre pipeline à l'instant n est donné par trois éléments : une image courante τ_n, une mémoire h_n et un ensemble de *spot frames*/probabilités/classes $\mathcal{V}_n$. Il est important de noter que les images sont utilisées en suivant le flux vidéo et ne peuvent être utilisées qu'une seule fois.

Mémoire : Au pas de temps n, l'image (ou le tronçon vidéo) $\boldsymbol{x}_{\tau_n}$ est transmise à un réseau d'extraction de caractéristiques $\boldsymbol{BB}$, basé sur des CNN. $\boldsymbol{BB}$ extrait un vecteur de caractéristiques $\boldsymbol{f}_n$ qui contient des informations spatiales

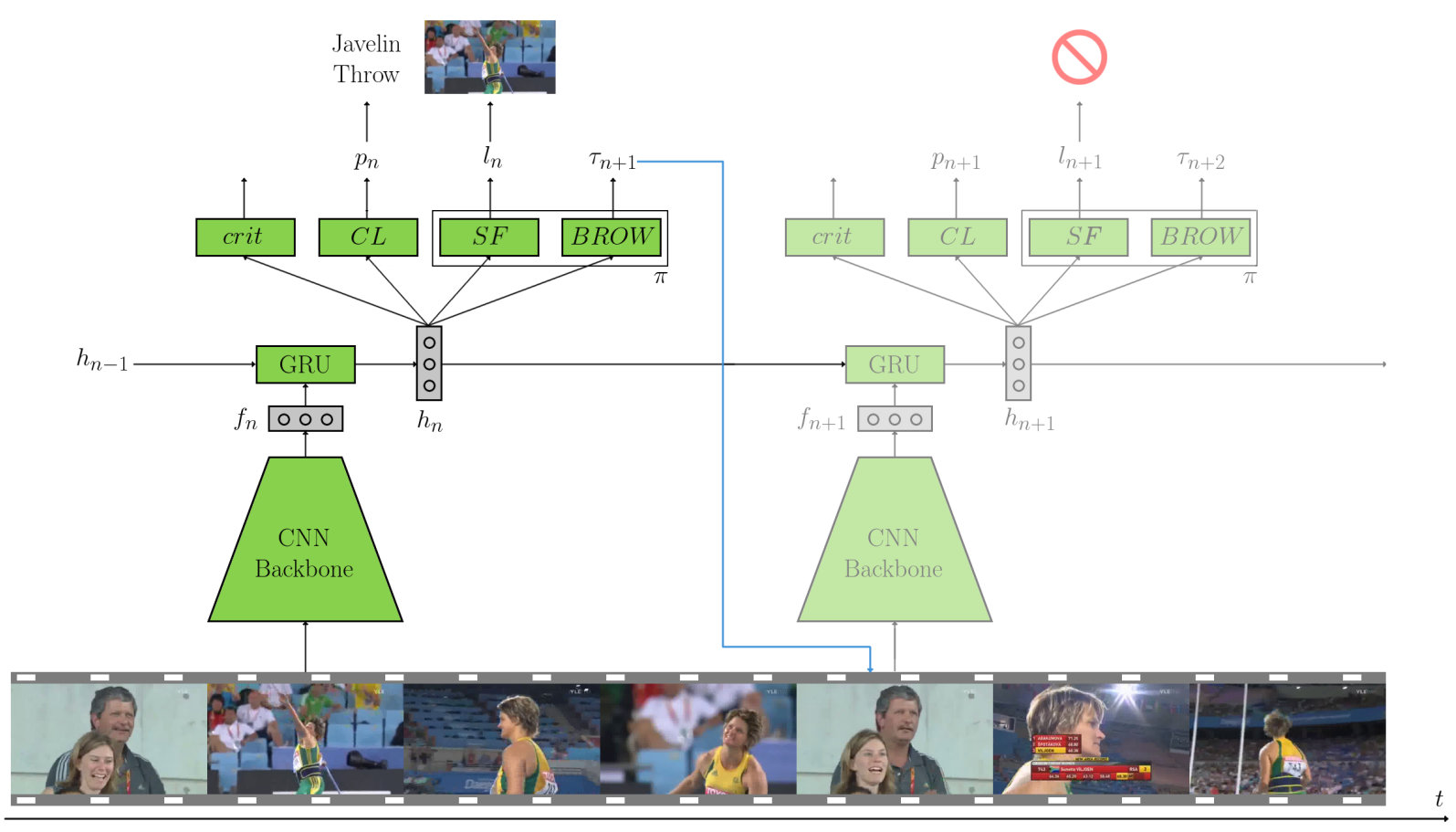


FIGURE 45 – **Description générale de notre méthode.** Dans un premier temps, l'extracteur CNN encode l'image (ou un bloc d'images successives) en un vecteur de caractéristiques qui est ensuite transmis à un GRU. Le vecteur d'état caché résultant est ensuite traité individuellement par (SF), (CL), (BROW) et ($crit$). Le réseau (SF) prend la décision de transformer l'image actuelle en une *spot frame* ou de la passer. Le réseau (CL) prédit la classe d'action liée à la *spot frame* et le réseau (BROW) indique la position de la prochaine image à analyser dans la vidéo. Le réseau (crit) est uniquement utilisé pour assurer une meilleure convergence de l'ensemble de l'architecture lors de l'apprentissage par renforcement.

$\boldsymbol{f}_n = \boldsymbol{BB}(\boldsymbol{x}_{\tau_n})$. Tout réseau d'extraction de caractéristiques de l'état de l'art peut être utilisé pour $\boldsymbol{BB}$ (voir Section.2.3.2). Ensuite, un réseau récurrent (GRU) (Chung et al., 2014) est utilisé pour coder l'information temporelle. Il prend comme entrées le vecteur de caractéristiques $\boldsymbol{f}_n$ et l'état caché précédent $\boldsymbol{h}_{n-1}$ et produit l'état caché courant $\boldsymbol{h}_n$, vu comme un vecteur latent qui contient la mémoire des images vues précédemment : $\boldsymbol{h}_n = \mathrm{GRU}(\boldsymbol{f}_n, \boldsymbol{h}_{n-1})$.

Réseau de classification : Ensuite, un réseau de classification $\boldsymbol{CL}$ traite l'état caché $\boldsymbol{h}_n$ et produit une distribution de probabilités sur les classes d'action $p_n = \boldsymbol{CL}(\boldsymbol{h}_n) \in \mathbb{R}^C$. La classe prédite est alors telle que $\alpha_n = \arg\max_c(p_{n,c})$.

Agent de sélection de *spot frames* : La mémoire $\boldsymbol{h}_n$ est également transmise à l'agent sélecteur de *spot frames* $\boldsymbol{SF}$ qui produit la probabilité pour l'image courante d'être une *spot frame* ou non : $l_n = \boldsymbol{SF}(\boldsymbol{h}_n)$. Formellement, la sortie de notre algorithme est mise à jour comme suit : $\mathcal{V}_{n+1} = \mathcal{V}_n \cup \{\tau_n, l_n, \alpha_n\}$. Lors de l'utilisation en phase de production, on peut choisir un seuil de détection σ ne gardant que les *spot frames* dont la probabilité est supérieure à σ. Cela conduit à un point de fonctionnement décrit par sa précision et son rappel. Il est important de rappeler que la mAP proposée est totalement invariante par rapport à σ.

Agent de navigation : En parallèle, cette mémoire $\boldsymbol{h}_n$ est également transmise à l'agent de navigation $\boldsymbol{BROW}$ qui décide de la prochaine image à visualiser, c'est-à-dire $\tau_{n+1} = \tau_n + \boldsymbol{BROW}(\boldsymbol{h}_n)$.

Ratio d'images visualisées : Il est important de noter que $\tau_n \neq n$ car l'agent de navigation ne parcourt pas toutes les images : soit N tel que $\tau_N = T$ (c'est-à-dire le nombre d'étapes pour traiter la vidéo), alors, le ratio d'images visualisées est défini comme $\frac{N}{T}$.

Dynamique globale : Alors, à l'étape n, ActionSpotter ($\boldsymbol{AS}$), a la dynamique suivante :

$$\boldsymbol{AS}(\boldsymbol{x}_{\tau_n}, \boldsymbol{h}_{n-1}) : \begin{cases} \boldsymbol{f}_n = \boldsymbol{BB}(\boldsymbol{x}_{\tau_n}) \\ \boldsymbol{h}_n = \mathrm{GRU}(\boldsymbol{f}_n, \boldsymbol{h}_{n-1}) \\ l_n = \boldsymbol{SF}(\boldsymbol{h}_n) \\ p_n = \boldsymbol{CL}(\boldsymbol{h}_n) \\ \alpha_n = \arg\max_c (p_{n,c}) \\ \mathcal{V}_{n+1} = \mathcal{V}_n \cup \{(\tau_n, l_n, \alpha_n)\} \\ \tau_{n+1} = \tau_n + \boldsymbol{BROW}(\boldsymbol{h}_n) \end{cases} \tag{3.5}$$

La figure 45 donne une illustration de l'ensemble de l'architecture.

3.4.2 Apprentissage et fonction de perte

Sachant que nous utilisons un modèle pré-entraîné pour BB (voir section 4.3), l'objectif de l'apprentissage est d'optimiser les paramètres du GRU, de $BROW$, SF et CL de sorte qu'en traitant la vidéo V, la politique $\pi = BROW + SF$ fournisse un ensemble précis de *spot frames* $\mathcal{V}$, tout en sautant un grand nombre d'images (c'est-à-dire avec $BROW(h_n)$) aussi élevé que possible. Il est important de noter que le navigateur $BROW$ et le réseau de sélection de *spot frames* SF ne sont pas appris de manière supervisée, ce qui élimine le besoin d'annotations spécifiques. Au lieu de cela, un algorithme d'apprentissage par renforcement de l'état de l'art est utilisé, avec une fonction de récompense spécifique, quantifiant la pertinence de chaque étape de la politique π (à la fois pour la précision et le taux de navigation).

Fonction de récompense. Comme nous utilisons la mAP pour évaluer la performance du modèle, nous choisissons d'introduire une récompense directement liée à cette métrique : la politique globale doit maximiser la mAP finale de la vidéo traitée (plus un terme d'entropie $\rho\mathcal{H}(\pi(n))$ qui sera expliqué plus tard). De plus, nous pouvons facilement trouver un compromis entre l'efficacité de l'exploration de la vidéo et la précision de la sélection des *spot frames* en pondérant la mAP finale par γ^N où $\gamma \in (0, 1]$ est le facteur de dévaluation (voir Section.2.2) et N le nombre d'images explorées.

Selon la théorie de mise en forme des récompenses (*reward shaping*) (Laud, 2004), cette récompense finale peut être hybridée avec une mise en forme basée sur le potentiel pour aider à la convergence de l'apprentissage par renforcement, sans changer la politique optimale. De manière directe, l'utilisation de la mAP après chaque étape est un signal de récompense ayant une forme intéressante. Ainsi, notre récompense ponctuelle $r_{\pi,n}$ à l'étape n est juste la différence de mAP entre l'étape n et $n-1$ sous la politique π (plus le terme d'entropie) :

$$r_{\pi,n} = \gamma \text{mAP}(\mathcal{V}_n) - \text{mAP}(\mathcal{V}_{n-1}) + \rho\mathcal{H}(\pi(n)) \tag{3.6}$$

Alors, $R_{\pi,n}$ la récompense cumulée actualisée est :

$$R_{\pi,n} = \sum_{k=1}^{n} \gamma^k r_{\pi,k} \tag{3.7}$$

En omettant le terme d'entropie, on peut directement vérifier que :

$$\begin{aligned} R_{\pi,N} &= \sum_{k=1}^{N} \gamma^k r_{\pi,k} = \sum_{k=1}^{N} \gamma^k \gamma \text{mAP}(\mathcal{V}_k) - \gamma^k \text{mAP}(\mathcal{V}_{k-1}) \\ &= \gamma^{N+1} \text{mAP}(\mathcal{V}_N) - \text{mAP}(\mathcal{V}_0) \end{aligned}$$

exactement comme désiré, comme $\text{mAP}(\mathcal{V}_0) = 0$. L'invariance de la mise en forme est vraie même avec l'entropie, comme le montre (Laud, 2004).

Optimisation Acteur-Critique. La méthode Policy-Gradient (voir Section.2.2.1) est basée sur la récompense totale attendue, et nécessite donc une longue séquence d'actions pour mettre à jour la politique. Les bonnes et mauvaises actions sont ensuite moyennées, ce qui peut introduire des problèmes de convergence. L'approche Acteur-Critique (Haarnoja et al., 2018) est connue pour être un moyen d'éviter ce problème en évaluant chaque action indépendamment. Elle utilise deux modèles nommés acteur et critique.

Comme introduit en Section.2.2.3 l'acteur est directement entraîné à trouver la politique qui maximise la récompense cumulée :

$$J_{actor} = \mathbb{E}[R_{\pi,N}] \tag{3.8}$$

Le critique mesure la qualité de la politique (*value-based*) et produit une estimation de la fonction de valeur qui est la récompense escomptée actualisée $crit(h_n) \approx \mathbb{E}[R_{\pi,N} - R_{\pi,n}|h_n]$. La fonction de perte liée au critique est définie comme :

$$\mathcal{L}_{critique} = \frac{1}{2} \| crit(h_n) - \mathbb{E}[R_{\pi,N} - R_{\pi,n}|h_n] \|_2 \tag{3.9}$$

En apprentissage par renforcement, il est crucial d'équilibrer l'exploration et l'exploitation. En suivant les travaux de Haarnoja et al., 2018, il est possible d'intégrer cet équilibre directement dans la récompense, en ajoutant une pénalité d'entropie qui oblige l'acteur à explorer uniformément les états à récompenses égales. Ainsi, une pénalité $\rho\mathcal{H}(\pi(n))$ est ajoutée dans l'Equation 3.6. ρ est le paramètre de température qui équilibre l'exploration et l'exploitation et est automatiquement ajusté selon Haarnoja et al., 2018. $\mathcal{H}()$ est la fonction d'entropie. Dénotons que sans *shaping* ou température, le renforcement a du mal à converger.

Dans notre algorithme, l'acteur est la combinaison de l'agent de navigation $BROW$ et de l'agent sélecteur de *spot frames* SF. Ainsi, $\mathcal{H}(\pi(n))$ est l'entropie appliquée à la distribution $\pi(n)$ choisissant de mettre à jour l'état actuel ou non *i.e.* $BROW(h_n)$ et $SF(h_n)$.

D'autre part, le réseau de classification CL est entraîné sur chacune des images de la vidéo, de manière supervisée, en utilisant l'entropie croisée (CE). Ainsi,

$$L_{cls} = CE(p_n, a_{\tau_n}) \tag{3.10}$$

avec a_{τ_n} la classe de l'image τ_n.

Fonction de perte finale : En combinant les fonctions de pertes précédentes, notre objectif final est de minimiser :

$$\mathcal{L}_{global} = \mathcal{L}_{cls} + \lambda_1 \mathcal{L}_{critic} - \lambda_2 J_{actor} \tag{3.11}$$

Comme l'objectif J_{actor} est non différentiable, nous utilisons la méthode REINFORCE (Sutton et al., 2018) (voir Section 2.2.1) pour dériver l'espérance du gradient :

$$\nabla J_{actor} = \nabla \mathbb{E}\left[\sum_{n=1}^{N} \log(\pi(n))(R_{\pi,n} - \mathbb{E}[R_{\pi,n}|h_n])\right] \tag{3.12}$$

Nous pouvons ensuite approximer cette équation en utilisant l'échantillonnage de Monte-Carlo et enfin utiliser la descente de gradient stochastique pour minimiser notre objectif final.

3.5 Expériences

Étant les premiers à aborder le repérage d'actions comme une tâche spécifique, il n'est pas facile de positionner notre algorithme par rapport à l'état de l'art. Nous proposons donc deux expériences pour mettre en évidence les capacités de cet algorithme. Tout d'abord, nous montrons que ActionSpotter produit de meilleurs ensembles de *spot frames* que les détecteurs de l'état de l'art. Ensuite, nous comparons ActionSpotter sur plusieurs bases de données de référence pour souligner que le repérage d'actions est une tâche difficile et que l'apprentissage par renforcement est une manière pertinente de traiter ce problème asymétrique où la sortie idéale n'est pas définie dans la vérité terrain. Les jeux de données sont THUMOS14 et ActivityNet v1.2, dont les spécificités sont détaillées en Section 2.7.2 et 2.7.3. Enfin, quelques expériences supplémentaires révèlent que ActionSpotter est capable d'équilibrer la précision et le ratio d'images visualisées, simplement en modifiant le facteur de dévaluation associé à l'apprentissage par renforcement.

THUMOS'14						
Approche	Détection mAP@					Repérage mAP
	0.1	0.2	0.3	0.4	0.5	
Glimpses (Yeung et al., 2016)	48.9	44.0	36.0	26.4	17.1	-
SMS (Z. Yuan et al., 2017)	51.0	45.2	36.5	27.8	17.8	-
M-CNN (Shou et al., 2016)	47.7	43.5	36.3	28.7	19.0	41.2
CDC (Shou et al., 2017)	-	-	41.3	30.7	24.7	31.5
TURN (Gao et al., 2017b)	54.0	50.9	44.1	34.9	25.6	44.8
R-C3D (H. Xu et al., 2017)	54.5	51.5	44.8	35.6	28.9	52.2
SSN (Zhao et al., 2017)	*66.0*	*59.4*	51.9	41.0	29.8	-
Alwassel et al., 2018	-	-	51.8	42.4	30.8	-
CBR (Gao et al., 2017a)	60.1	56.7	50.1	41.3	31.0	50.1
BSN + UNet (T. Lin et al., 2018)	-	-	53.5	45.0	36.9	-
Chao et al., 2018	59.8	57.1	53.2	48.5	42.5	-
D-SSAD (Huang et al., 2019)	-	-	*60.2*	*54.1*	*44.2*	*59.7*
Ours (caractéristiques TSN)	-	-	-	-	-	62.4
Ours (caractéristiques I3D)	-	-	-	-	-	**65.6**

Table 3.2 – **Résultats sur le jeu de validation de la base THUMOS14.** Deuxième colonne : résultats de l'état de l'art suivant la métrique de détection, calculée pour différents seuils de tIoU classés de 0.1 à 0.5. Dernière colonne : résultats en mAP pour la tâche de repérage.

3.5.1 Détails de mise en oeuvre

Comme nous l'avons mentionné précédemment, n'importe quel type d'extracteur de caractéristiques peut être utilisé pour encoder localement les vidéos. Afin de disposer des mêmes extracteurs que les détecteurs d'action de état de l'art, nous nous appuyons sur des extracteurs de caractéristiques à double flux TSN (Simonyan et al., 2014) et I3D (Carreira et al., 2017) classiques (voir Section 2.3.2.2). Ce paramétrage permet une bonne reproductibilité car ces caractéristiques sont fournies par Paul et al., 2018 et T. Lin et al., 2018. Ces techniques fonctionnent à la fois sur les images RVB et sur le flot optique pour capturer les informations sur l'apparence et le mouvement. La technique TSN opère sur des images vidéo individuelles tandis que les caractéristiques I3D sont extraites de tranches vidéo de 16 images sans chevauchement. Pour cette deuxième technique, comme les caractéristiques représentent des tranches de vidéos non superposées, l'agent de navigation BROW ne traite pas les images individuelles mais les tranches d'images. Cela ne modifie pas le taux de saut car cela revient à considérer une tranche sur K au lieu d'une image sur K.

Pour la mémoire, nous utilisons un GRU à une couche avec respectivement

2 048 et 400 neurones cachés pour THUMOS14 et ActivityNet (respectivement la taille des caractéristiques). BROW, SF, CL et $crit$ ont 3 couches linéaires et une fonction d'activation ReLu. Les deux premières couches ont le même nombre de neurones cachés que la couche GRU et la dernière a la taille de la sortie du réseau.

Dans notre configuration, BROW peut choisir de passer à l'image suivante, de sauter une image ou de sauter trois images. Au moment de l'apprentissage, afin d'approcher l'équation 3.12, les actions effectuées par BROW et SF sont échantillonnées à partir d'une distribution catégorielle paramétrée par les logits respectifs. Au moment du test, l'action exécutée est déterministe et est celle qui présente la plus forte probabilité pour BROW (et pour CL). SF produit directement une probabilité.

Nous utilisons PyTorch pour l'implémentation et Adam (Kingma et al., 2014) pour l'optimisation avec un *learning rate* initial de 10^{-4} et une taille des échantillons d'apprentissage de 32 vidéos. La convergence est beaucoup plus rapide lorsque CL et SF sont entraînés seuls pendant plusieurs itérations avant de débuter le processus d'apprentissage par renforcement complet avec $\lambda_1 = \lambda_2 = 1$.

3.5.2 Comparaison entre ActionSpotter et les détecteurs d'actions

ActivityNet v1.2					
Approche	Detection mAP@				Repérage mAP
	0.5	0.75	0.95	Moy.	
W-TALC (Paul et al., 2018)	37.0	14.6	-	18.0	-
SSN-SW (Zhao et al., 2017)	-	-	-	18.1	-
3C-Net (Narayan et al., 2019)	37.2	23.7	9.2	21.7	-
FPTADC (J. He et al., 2019)	37.6	21.8	2.4	21.9	-
SSN-TAG (Zhao et al., 2017)	39.2	25.3	5.4	25.9	55.4
BSN (T. Lin et al., 2018)	46.5	30.0	8.0	30.0	49.6
BMN (T. Lin et al., 2019)	50.1	34.8	8.3	33.85	55.3
Ours (caractéristiques TSN)	-	-	-	-	58.1
Ours (caractéristiques I3D)	-	-	-	-	**60.2**

Table 3.3 – **Résultats sur le jeu de validation de la base ActivityNetv1.2.** La colonne Moy. indique la mAP moyenne pour les seuils de tIoU : 0.5 :0.05 :0.95.

Nous évaluons les performances d'ActionSpotter sur les jeux de données **THU-**

MOS14 et **ActivityNet** en utilisant la métrique de repérage proposée. Les résultats sont présentés dans les Tableaux 3.2 et 3.3. Ces tableaux présentent également les résultats obtenus par les détecteurs les plus performants sur la tâche de repérage. Nous utilisons les résultats publiés ou les codes disponibles pour obtenir les résultats de détection, il n'y a donc aucun problème de réimplémentation. Les résultats de détection sont redessinés en résultats de repérage en extrayant les centres des segments prédits.

Ces expériences montrent que ActionSpotter surpasse de manière significative les détecteurs de l'état de l'art sur la tâche de spotting : la mAP du dernier détecteur d'action D-SSAD (Huang et al., 2019) est améliorée de 59% à 65% sur la base de données THUMOS14.

On peut souligner que FrameGlimpses (Yeung et al., 2016) offre des performances très faibles mais présente un taux de sauts de 98%. Actuellement, les performances d'ActionSpotter diminuent également de 62,4% à 50,9% lorsqu'on utilise l'extracteur de caractéristiques TSN et qu'on augmente le taux de sauts à 98%. Cela montre qu'il est difficile d'effectuer un repérage précis (et encore plus une détection) avec seulement 2% des images, ce qui n'est pas le cas pour la classification des actions (Z. Wu et al., 2019).

Comme il a été constaté dans la Section 3.3, les performances de repérage ne sont pas similaires aux performances de détection avec une faible valeur de seuil de *temporal Intersection over Union* (tIoU). En fait, cela est en partie dû à une différence dans le mécanisme d'appariement des segments pendant le calcul du critère d'évaluation. Dans le cas de la détection, les segments prédits vont être associés aux segments de la vérité terrain en fonction de la meilleure tIoU, alors que dans le cas du repérage, les *spot frames* sont associées aux segments de vérité terrain en fonction de leurs scores (ce qui est logique puisque la tIoU n'existe plus). Ainsi, pour la tâche de détection, il est préférable de prédire des segments avec une bonne localisation et une confiance aléatoire que de produire un segment par action avec une bonne confiance mais une localisation grossière. A l'inverse, seul le score de confiance compte pour le repérage. Cette différence entre les processus d'appariement induit certains changements dans le classement des prédictions entre la détection et le repérage : alors que CDC (Shou et al., 2017) est plus efficace en détection que M-CNN (Shou et al., 2016), c'est le contraire pour le repérage.

Ainsi, les résultats de repérage obtenus avec les détecteurs ne peuvent pas être facilement comparés à ceux des algorithmes de détection car leur objectif principal n'est pas le même. Cependant, l'écart de performance important entre

notre méthode et les méthodes de détection montre qu'il ne suffit pas de post-traiter les résultats des détecteurs pour avoir un repérage optimal, et que des algorithmes spécifiques tels que ActionSpotter sont nécessaires.

3.5.3 Etude d'ablation et comparaisons

Les expériences précédentes montrent que le repérage d'actions nécessite une approche spécifique. De plus, le repérage d'actions est un problème asymétrique puisque n'importe quelle image appartenant à un segment de vérité terrain peut être utilisée comme *spot frame* et, par conséquent, les *spot frames* optimales ne sont pas définies par ces segments.

Sur cette base, l'apprentissage par renforcement semble convenir pour s'adapter à cette spécificité et nous montrons : (*i*) des résultats comparatifs entre l'algorithme de repérage appris de manière supervisée et par renforcement, (*ii*) une version ablatée d'ActionSpotter. Les résultats sont présentés dans le tableau 3.4.

Méthode	mAP (%)
Segmentation naïve	32
Segmentation multitâche	43
ActionSpotter supervisé	52
ActionSpotter sans mémoire	45
ActionSpotter (mémoire + renforcement)	**65**

TABLE 3.4 – **Comparaison entre ActionSpotter et d'autres algorithmes de repérage sur le jeu d'évaluation de THUMOS14.**

La segmentation naïve est une segmentation sémantique simple basée sur une architecture CNN (caractéristiques extraites I3D + 2 couches de convolution) où la vérité terrain est construite comme suit : le centre de chaque segment d'action est étiqueté par sa classe d'action et toutes les autres images sont étiquetées comme images de fond. Cette base de référence orientée vers le repérage n'atteint que 32% de mAP.

Un processus d'apprentissage à deux tâches est ensuite utilisé pour une segmentation multitâche : la première tâche apprend le centre des segments et la seconde prédit la classe d'action. Cette deuxième méthode de référence traite un problème plus équilibré que le précédent, aidant à la stabilisation du gradient et conduit à 43% de la mAP.

Ensuite, nous entraînons l'architecture ActionSpotter à réaliser la tâche de segmentation supervisée des centres d'action (au lieu de l'apprentissage par ren-

	γ				
	1.0	0.99	0.98	0.96	0.95
mAP (%)	**65.6**	64.3	63.4	61.4	61.3
Ratio de sauts (%)	23	29	**53**	51	51

TABLE 3.5 – **Performance sur le jeu de validation de THUMOS14 relative au facteur de dévaluation γ.**

forcement). Dans ce contexte, les performances atteignent 52,3% mAP, ce qui est supérieur aux autres méthodes de référence mais bien inférieur à ActionSpotter.

Enfin, nous considérons ActionSpotter (avec apprentissage par renforcement) en supprimant sa mémoire. La mAP chute à 45% : comme prévu, sans mémoire, la décision du réseau est seulement basée sur l'image courante et, dans ce cas, l'apprentissage par renforcement est relativement équivalent à l'apprentissage supervisé.

Ces résultats soulignent le fait qu'il n'est pas facile de superviser le repérage d'actions. En fait, il est préférable de laisser le réseau choisir la *spot frame* la plus facile pour lui plutôt que de l'imposer. C'est exactement ce que fait le renforcement puisque la récompense est uniquement basée sur la sortie finale.

La figure 46 montre des résultats qualitatifs sur cinq vidéos : dans la plupart des cas, les *spot frames* (troisième ligne en rouge) ne sont pas extraites au milieu des actions (première ligne en vert). ActionSpotter s'adapte à la difficulté des vidéos traitées, comme le montre le fait que le navigateur explore moins d'images (deuxième rangée en bleu) dans les zones sans actions.

En outre, ActionSpotter permet d'optimiser directement le score de mAP en utilisant une technique de *shaping* de l'état de l'art. En effet, cette technique de *shaping* est un composante importante car la performance chute à 47% sans elle. ActionSpotter est donc basé sur 3 idées clés : l'apprentissage de bout en bout, l'apprentissage par renforcement pour lutter contre l'asymétrie et le shapping pour permettre une meilleure convergence.

3.5.4 Compromis entre précision et taux de saut

ActionSpotter gère l'absence de vérité terrain pour les *spot frames* et s'attaque au problème de repérage en optimisant la mAP. De plus, il permet d'équilibrer la mAP et le taux de saut du navigateur. Nous présentons dans le Tableau 3.5 la mAP et le taux de saut pour différentes valeurs de γ afin de mettre en évidence l'impact du facteur d'actualisation sur l'apprentissage de la politique.

Comme prévu, plus le facteur de dévaluation γ est faible, plus le taux de saut est élevé (les problèmes de convergence apparaissent rapidement lorsque le γ diminue). Mais, comme il existe un compromis entre la vitesse et la précision, la précision diminue également. Néanmoins, lorsque le taux de saut passe de 23% à 53%, la mAP ne diminue que de 2%. Pour $\gamma = 0.98$, notre algorithme est toujours meilleur que le meilleur détecteur utilisé pour le repérage en utilisant seulement 47% des images. Actuellement, avec le paramétrage de l'espace d'actions présenté qui met beaucoup l'accent sur la mAP, il est difficile de descendre en dessous d'un ratio de saut de 53%. Mais, en ajoutant plus d'actions, il est possible de sauter plus d'images même si la mAP diminue de manière significative. Par exemple, une mAP de 50,9% est obtenue en utilisant seulement 2% des images (même taux de saut que Yeung et al., 2016). Mais il n'y a aucun intérêt à sauter autant d'images si cela conduit à un niveau de performance aussi faible. Comme mentionné précédemment, il semble difficile d'effectuer une détection ou un repérage précis avec 2% des images, contrairement à la classification (Z. Wu et al., 2019) qui ne suppose qu'une seule action.

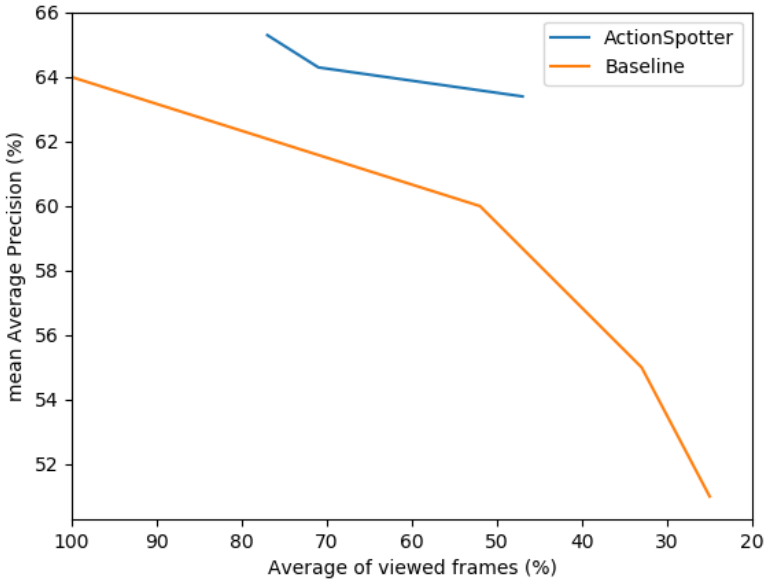


FIGURE 47 – **Bénéfice de la navigation par renforcement** : mAP relative au pourcentage d'images visualisées pour l'algorithme ActionSpotter et un sous-échantillonnage uniforme.

Ainsi, alors que le fait de sauter de nombreuses images dégrade les performances, le fait de sauter quelques images les améliore. En effet, en supprimant le réseau de navigation et en traitant toutes les images vidéo, la mAP passe de

65,6% à 64,0%. L'utilisation de l'agent de navigation entraîne une amélioration de 1,6% de la mAP tout en utilisant 23% d'images en moins.

Il est intéressant de noter que nous avons également entraîné notre pipeline sans le navigateur mais avec un sous-échantillonnage uniforme. Plus précisément, nous sous-échantillonnons des vidéos avec des taux d'échantillonnage différents et observons la mAP. La figure 47 montre qu'ActionSpotter produit de meilleurs résultats quel que soit le taux de saut.

3.6 Conclusion du chapitre

Nous proposons un algorithme appelé ActionSpotter capable de s'attaquer à la tâche de repérage d'actions : collecter une et une seule image par instance d'action. Notre algorithme est basé sur un algorithme par renforcement de l'état de l'art et est capable de réaliser un repérage d'actions dans un contexte de flux vidéo en ligne (les images ne sont pas stockées) tout en sautant certaines images vidéo.

Pour évaluer l'algorithme proposé, nous introduisons une métrique qui quantifie la précision du repérage d'actions. Elle est basée sur la mean average precision (mAP) classiquement utilisée en détection.

Le principal résultat de ce chapitre est que l'adaptation des détecteurs d'action de l'état de l'art pour le repérage d'actions est moins efficace que l'apprentissage de la tâche de repérage de bout en bout : ActionSpotter atteint 65% de mAP (tout en sautant 23% des images vidéo), alors que les détecteurs de l'état de l'art n'atteignent que 59% de mAP (avec une exploration dense).

En effet, contrairement aux détecteurs d'action qui doivent gérer des limites temporelles ambiguës, ActionSpotter se concentre sur l'extraction d'une image par instance grâce à l'apprentissage par renforcement. Ces résultats montrent donc qu'une limite haute de performance en repérage n'est pas atteinte avec les détecteurs de l'état de l'art et que des approches spécifiques permettent d'extraire de meilleures informations sémantiques des vidéos.

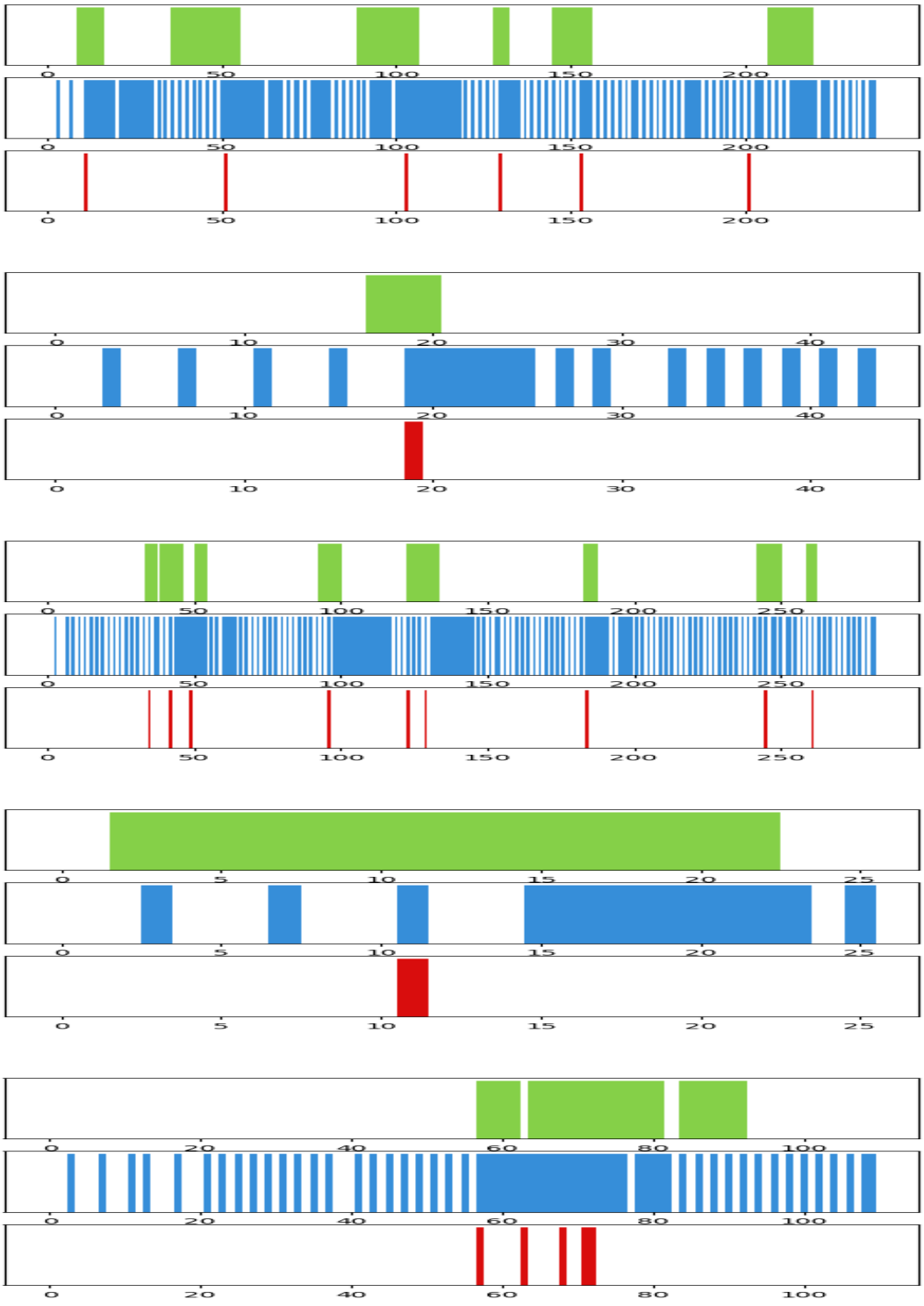


FIGURE 46 – **Exemple qualitatif de résultats d'ActionSpotter.** La figure montre les sorties d'ActionSpotter sur des vidéos du jeu de validation de THUMOS14 : les colonnes représentent les images successives et les 3 lignes représentent respectivement les segments temporels de la vérité terrain contenant les actions, les images explorées et les *spot frames* selectionnées.

Chapitre 4

Détection d'actions

Sommaire

4.1 Introduction

La tâche de repérage d'actions permet de réaliser une première analyse sémantique robuste des actions présentes dans les vidéos en extrayant une et une seule *spot frame* par action. Ainsi, avec une certaine précision, l'algorithme ActionSpotter est capable de situer, caractériser et compter ces actions, en introduisant une méthode basée sur la notion de confiance temporelle. Mais lorsque le besoin de connaître précisément le moment où l'action débute et celui où elle se termine est présent, les algorithmes de repérage ne sont plus adaptés. Malgré tout, et

comme l'a montré Alwassel et al., 2018, le repérage d'actions permet d'extraire un *a priori* sur la localisation temporelle des régions d'intérêt pour la détection d'action. En ce sens, si une méthode de repérage est capable d'extraire des *spot frames* pertinentes, il est peut-être possible de localiser et d'extraire les débuts et les fins de chaque action en analysant les images de leur voisinage temporel.

Cependant, pour avoir la meilleure détection possible, c'est-à-dire localiser précisément les extrémités temporelles des actions, il est nécessaire que les *spot frames* utilisées soient appropriées pour la localisation des débuts/fins. Il y a peu de chances que les *spot frames* extraites lors du repérage soient optimales pour la détection d'actions. De cette manière, la tâche de détection d'actions devient une tâche double. En effet, il est nécessaire de non seulement trouver les *spot frames* ayant la probabilité la plus forte de contenir un voisinage contenant assez d'information pour trouver le début et la fin de l'action mais aussi d'être capable localiser ou régresser le début et la fin à partir de cet endroit.

4.1.1 Détection d'actions

Comme introduit en Section 1.2.4, la tâche de détection temporelle d'actions consiste à localiser dans le temps les débuts et fins d'actions ainsi qu'à les caractériser. Soit $\mathbf{V} = x_1, ..., x_T$ avec T images (ou T portions de vidéos) une vidéo où x_t désigne le vecteur de caractéristiques de la t-ième image (ou t-ième portion). Cette vidéo est associée à un ensemble de segments de vérité terrain $\mathbf{G} = \{[s_n, e_n], c_n\}_{n=1}^{N}$, où s_n, e_n et c_n sont respectivement l'instant de début, l'instant de fin et la classe du segment de vérité terrain n.

La tâche de détection d'action consiste à prédire un ensemble de K propositions $\mathbf{P} = \{[s_k, e_k], c_k\}_{k=1}^{K}$ à partir de $\mathbf{V}$ correspondant aussi fidèlement que possible à $\mathbf{G}$.

Aussi, on rappelle que la tâche de repérage d'actions consiste à produire un ensemble de K *spot frames* avec une probabilité et une classe :

$$\mathcal{V} = \{(\tau_1, l_1, \alpha_1), ..., (\tau_K, l_K, \alpha_K)\} \tag{4.1}$$

tels que $K = N$ et $\forall k \in \{1, ..., K\},\ \tau_k \in [s_k, e_k]$ et $\alpha_k = c_k$.

Ainsi l'extraction de *spot frames* $\mathcal{V}$ peut être vue comme un *a priori* pour la

détection puisque $\forall k \in [1, ..., K]$, il existe $\epsilon_k = (\epsilon_{k,start}, \epsilon_{k,end})$ tels que :

$$\begin{cases} s_k = \tau_k - \epsilon_{k,start} \\ e_k = \tau_k + \epsilon_{k,end} \end{cases} \tag{4.2}$$

et la tâche de détection peut alors être décomposée en une tâche de repérage des images τ_k suivie d'une tâche de régression de ϵ_k. Et dans ce cas, la tâche de détection d'action consiste à prédire un ensemble de K propositions $\mathbf{P} = \{[\tau_k - \epsilon_{k,start}, \tau_k + \epsilon_{k,end}], l_k, c_k\}_{k=1}^{K}$ à partir de $\mathbf{V}$ correspondant aussi fidèlement que possible à $\mathbf{G}$.

4.1.2 Régression naïve par auto-évaluation

Lorsqu'on effectue une régression, on ne dispose que d'une seule sortie : la valeur régressée. Il est alors impossible de savoir quel degré de confiance on peut accorder à cette valeur. Ce n'est pas le cas en classification où le réseau profond produit une distribution des scores sur les classes. De cette distribution, on peut extraire à la fois le *argmax* (la classe prédite) et la marge entre le maximum et le second maximum, classiquement utilisée comme confiance.

Pour la régression, une solution consiste à attribuer un score *a posteriori* aux boites prédites. Alternativement, nous proposons d'estimer une confiance de régression en utilisant un réseau à deux têtes, l'une qui effectue une régression classique et l'autre qui estime si nous pouvons être confiants ou non dans cette valeur régressée. Par conséquent, nous exprimons le problème de confiance sous la forme d'une classification binaire. Par exemple, pour une entrée x associée à une vérité terrain $z(x)$, un tel réseau devrait produire $\hat{z}(x)$ (régression) et $\hat{p}(x)$ (auto-évaluation). Si κ est la tolérance autorisée sur la valeur régressée, alors une perte pourrait être :

$$l = ||z(x) - \hat{z}(x)||_2^2 + \alpha \begin{bmatrix} y(x)\log(\hat{p}(x)) + \\ (1 - y(x))\log(1 - \hat{p}(x)) \end{bmatrix} \tag{4.3}$$

avec $y(x) = 1$ si $||z(x) - \hat{z}(x)||_2^2 < \kappa$ et 0 sinon

i.e. un terme de régression et un terme d'entropie croisée.

Tout d'abord, en ajoutant une telle tête, le système sera en mesure de produire un score permettant de quantifier la confiance en la prédiction, ce qui est d'ailleurs un premier niveau d'exigence pour la certification des systèmes afin de pouvoir les utiliser dans la vie réelle. Ensuite, grâce à la régularisation multi-

tâches, cette deuxième tête pourrait aider à améliorer la régression (l'utilisation d'un tel couplage peut être trouvé dans Choi et al., 2018, mais le bénéfice de l'auto-évaluation pour la tâche sous-jacente n'est pas le but de leur travail).

4.1.3 Détection d'actions par auto-évaluation

Comme présenté dans la Section 4.1.1, le problème de détection peut être abordé comme un repérage d'actions suivi d'un problème de régression. Sauf qu'il a aussi été vu en introduction de la section Section 4.1, qu'il n'est pas évident d'affirmer que les *spot frames* extraites dans le cadre du repérage soient le meilleur point d'ancrage *a priori* pour réaliser la régression. Ainsi, pour choisir les meilleurs points d'ancrage, nous proposons d'apprendre simultanément une tâche de régression et une auto-évaluation de cette tâche (voir Section 4.1.2).

Notre contribution est d'adapter la tête d'auto-évaluation, introduite pour une régression naïve, à la détection d'actions. Le bénéfice possible est potentiellement encore plus grand puisque cette tête peut être utilisée pour repérer des zones temporelles où la tâche de régression pourra être efficace et donc d'apporter une information permettant d'améliorer la capacité du détecteur à réaliser la tâche de repérage dans l'optique de la tâche de détection d'actions. Ceci permet alors de considérer le score d'auto-évaluation comme un *a priori* attentionnel. Aussi, il est plus facile d'encoder la spécificité de la métrique de détection d'action dans l'objectif d'auto-évaluation plutôt que dans la régression elle-même, ce qui peut donc conduire à de bien meilleures performances. Cette idée est illustrée dans la Figure 48, où l'auto-évaluation aide à l'extraction des caractéristiques, à la régression des segments et à la sélection des segments pertinents.

Un tel apprentissage conjoint est particulièrement pertinent pour la détection d'action car il permet de prendre naturellement en compte certaines de ses spécificités comme, par exemple, l'importance de ne prédire qu'une seule détection par instance d'action (éviter la double détection). Ainsi, cet apprentissage optimise à la fois la confiance individuelle de chaque segment prédit et leur capacité mutuelle à fournir un résultat global de détection correct. Plus formellement, l'apprentissage simultané de la tâche de régression et de l'auto-évaluation de cette tâche permet de coder la métrique complexe de la *mean Average Precision* (mAP) qui n'est pas directement prise en compte par une régression simple, basée sur la temporal Intersection over Union (tIoU). Enfin, il sera montré en Section 4.2 que cette confiance peut également être utilisée pour empêcher certains points d'ancrage (certaines images) de participer au processus d'apprentissage. Ce contrôle de la participation à l'apprentissage, qu'on appelle élagage, est inspiré des récents tra-

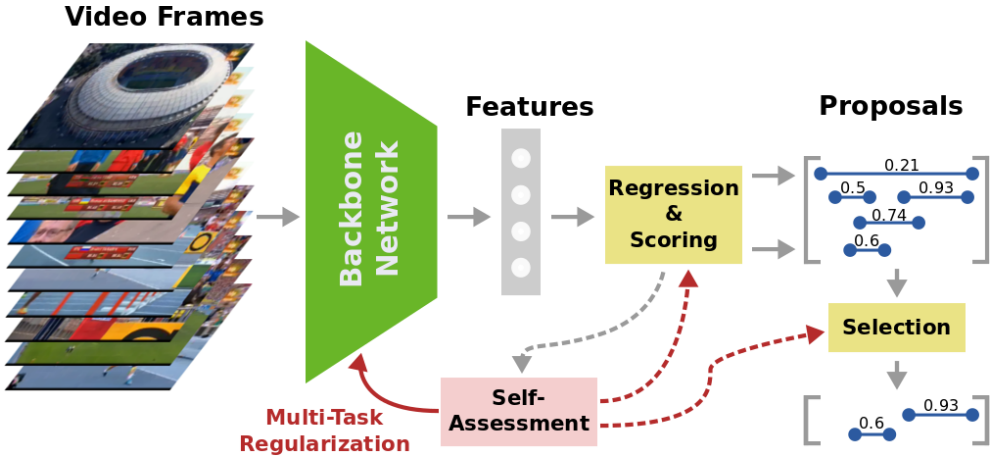


FIGURE 48 – **Apprentissage conjoint du score d'auto-evaluation et de la localisation des actions.** Durant l'apprentissage, l'auto-évaluation (*self-assessment*) permet d'améliorer la régression, la prédiction du score et la sélection de segments temporels, ainsi que l'extraction de caractéristiques par un effet de régularisation multi-tâches.

vaux de *Deep Reinforcement Learning* (W. WU et al., 2019 ; Z. WU et al., 2019 ; YEUNG et al., 2016), et est naturellement intégré à la méthode proposée puisque le score d'auto-évaluation peut être interprété comme une valeur attentionnelle.

4.2 Détecteur SALAD

4.2.1 Architecture

L'architecture globale de notre méthode, appelée SALAD (pour *Self-Assessment Learning for Action Detection*), est présentée Figure 49. Chaque image (ou portion de vidéo), au temps t, est d'abord caractérisée par un vecteur de caractéristiques en utilisant un extracteur de l'état de l'art (voir Section 2.3). La série temporelle des vecteurs de caractéristiques est ensuite traitée par un réseau récurrent bidirectionnel, un *Bidirectionnal Gated Recurrent Unit* (voir Section 2.1.3), qui produit, pour chaque temps t, deux vecteurs latents. Le premier peut être considéré comme une représentation de la vidéo avant le pas de temps t (point d'ancrage t) tandis que le second est une représentation de la vidéo après ce pas de temps. Les deux vecteurs latents sont ensuite partagés par 3 modules entièrement connectés qui produisent respectivement un segment régressé $[\hat{s}_t, \hat{e}_t]$, un score d'auto-évaluation de ce segment $\hat{p}_t$ et sa classe d'action $\hat{c}_t$.

Mais l'objet principal de ces travaux n'est pas cette architecture mais l'apprentissage conjoint de la régression des segments et de l'auto-évaluation, au lieu de

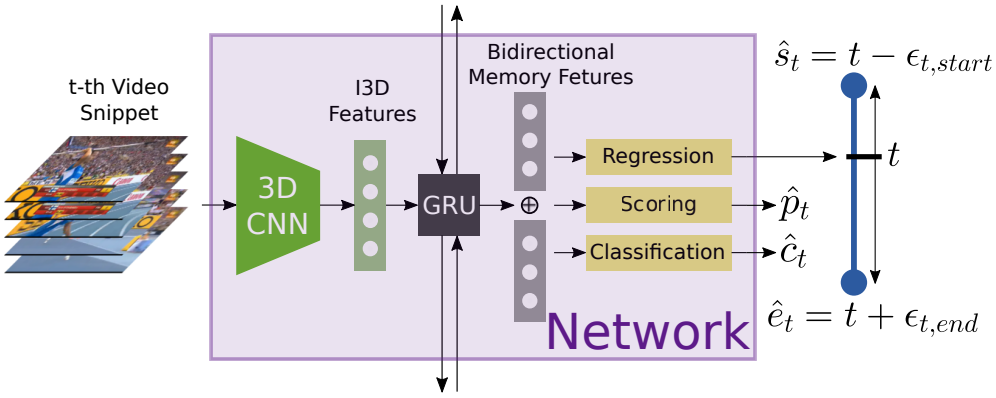


FIGURE 49 – **Architecture du réseau SALAD.** Chaque image (ou portion de vidéo) t est d'abord représentée par un vecteur de caractéristiques extrait grâce à un réseau de l'état de l'art. Un GRU bidirectionnel est ensuite utilisé pour produire une mémoire des images précédentes et une mémoire des images suivantes. Les deux mémoires sont utilisées par trois modules qui estiment une proposition temporelle $[\hat{s}_t, \hat{e}_t]$ incluant le temps t, un score de confiance $\hat{p}_t$ et une distribution de probabilités $\hat{c}_t = \{\hat{c}_t^i\}_{i=1}^C$ sur les classes d'action.

s'appuyer sur un module externe. L'utilisation de l'auto-évaluation permet d'élaguer les points d'ancrage pendant l'apprentissage et d'améliorer le repérage de ces points d'ancrage, en utilisant un mécanisme attentionnel, et une régularisation multi-tâches, comme illustré Figure 50.

4.2.2 Mécanisme de détection d'actions par auto-évaluation

Une des problèmes qui rend la détection des actions complexe est qu'un nombre inconnu de segments doit être régressé. De plus, nous devons gérer la précision des segments détectés mais aussi l'absence de détection, et la double détection. Le critère classique utilisé pour mesurer la précision des segments détectés, introduit en Section 3.3 est la *temporal Intersection over Union* (tIoU) entre le segment prédit $[\hat{s}, \hat{e}]$ et la vérité terrain $[s, e]$. On rappelle qu'elle se définit comme suit :

$$tIoU([\hat{s}, \hat{e}], [s, e]) = \frac{\min(e, \hat{e}) - \max(s, \hat{s})}{\max(e, \hat{e}) - \min(s, \hat{s})} \tag{4.4}$$

Aussi, les têtes de régression classiques prennent en compte le critère de tIoU en l'estimant localement mais sans traiter les relations entre segments, ce qui peut induire, par exemple, des doubles détections. A l'inverse, comme dans YUN et al., 2020, l'auto-évaluation pourrait facilement prendre en compte le comportement global. Typiquement, elle peut considérer si un segment régressé correspond à un segment de vérité terrain avec une tIoU μ minimum et s'il est le meilleur parmi

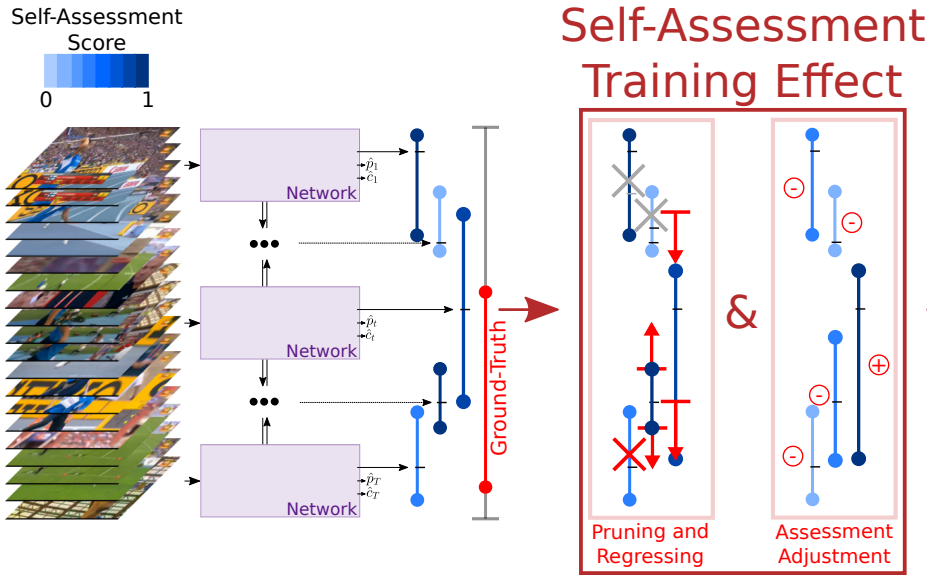


FIGURE 50 – **Illustration de l'apprentissage par auto-évaluation**. Les segments temporels régressés et leur score d'auto-évaluation sont utilisés, conjointement à la vérité terrain, pour calculer la fonction de perte. Durant ce calcul, certains segments sont élagués (marqués par une croix sur la figure) tandis que les autres continuent le processus (segments sans croix). Ces derniers sont catégorisés comme sûr, ou non, suivant la valeur de tIoU et le score d'auto-évaluation. Avec cette fonction de perte, la régression des bords des segments non élagués peut leur permettre de se rapprocher très précisément des bords de la vérité terrain et le score d'auto-évaluation des meilleurs segments est optimisé pour croître, tandis que ceux des autres sont optimisés pour décroître, comme le montre la partie droite de la figure avec les signes $\oplus$ et $\ominus$.

tous les autres segments pour ce segment de vérité terrain en particulier.

Comme indiqué, un autre intérêt de la tête d'auto-évaluation est d'accéder au score pendant l'apprentissage, qui va rendre possible la stratégie d'élagage. Cette stratégie d'élagage est cohérente avec des travaux récents basés sur l'apprentissage par renforcement, qui montrent que l'utilisation de toutes les images n'est pas optimale pour la détection d'actions (VAUDAUX-RUTH et al., 2021a ; W. WU et al., 2019 ; Z. WU et al., 2019).

Fonction de perte d'auto-évaluation. Le principe algorithmique général du modèle SALAD est d'utiliser des mémoires de la vidéo, relatives à des points d'ancrage temporels, afin de réaliser une régression des bords des actions, de manière relative à ces mêmes points d'ancrage. Le score d'auto-évaluation permet alors de piloter l'apprentissage de cette régression, en donnant de l'importance ou non à certains points d'ancrage.

Pour cela, à chaque itération, le réseau prédit pour chaque point d'ancrage t, un segment temporel régressé $[\hat{s}_t, \hat{e}_t]$ contenant l'image t et un score d'auto-

évaluation $\hat{p}_t$. Oublions d'abord la prédiction des classes d'action, nécessaires pour caractériser les segments temporels et pour compléter le processus de détection, qui seront gérées séparément.

Ensuite, des variables binaires $(\alpha_{t,n})_{t,n\in[1,T]\times[1,N]} \in \{0,1\}$ et $(y_t)_{t\in[1,T]} \in \{0,1\}$ sont introduites : $\alpha_{t,n}$ est fixé à 1 si le point d'ancrage t participe au processus de régression du segment $[s_n, e_n]$, et, y_t est fixé à 1 si le segment t est considéré précis. Dans ce cas, l'optimisation doit alors chercher à augmenter son score d'auto-évaluation $\hat{p}_t$. Dans le cas contraire, le statut de ces variables est nul. Ces variables sont attribuées selon l'Algorithme 2 (une explication plus littérale est présentée après), et sont ensuite utilisées pour calculer la fonction de perte des têtes de régression et d'auto-évaluation en utilisant :

$$\begin{aligned} L_{r,sa} = &\sum_{t=1}^{T} \left(y_t \log(\hat{p}_t) + (1-y_t)\log(1-\hat{p}_t)\right) \\ &-\lambda_1 \sum_{t=1}^{T}\sum_{n=1}^{N} \left(\alpha_{t,n} tIoU([\hat{s}_t, \hat{e}_t], [s_n, e_n])\right) \end{aligned} \tag{4.5}$$

où λ_1 est un paramètre de pondération.

Par analogie avec l'équation 4.3, on retrouve le terme d'entropie croisée et le terme de régression (ici la tIoU et non une norme L_2). Le seuil κ peut alors être ici un seuil de tIoU.

Apprentissage par auto-évaluation. Plus littéralement, nous commençons par trier les points d'ancrage par score d'auto-évaluation, de manière décroissante : σ est une permutation telle que $\forall u \leq v \in [1,T],\ \hat{p}_{\sigma(u)} \geq \hat{p}_{\sigma(v)}$. Un point d'ancrage $\sigma(t)$ en dehors de tous les segments de vérité terrain n'est jamais utilisé pour la régression et devrait conduire à une faible auto-évaluation $y_{\sigma(t)} = 0$, $(\alpha_{\sigma(t),n})_n = \mathbf{0}$. Inversement, considérons un point d'ancrage $\sigma(t)$ à l'intérieur d'un segment de vérité terrain $[s_n, e_n]$. Deux cas apparaissent. Si ce segment de vérité terrain a déjà été apparié avec un autre segment régressé ($\beta_n = 1$ dans l'algorithme 2), alors cela signifie que l'auto-évaluation $\hat{p}_{\sigma(t)}$ est inférieure à celle qui a été appariée avec $[s_n, e_n]$ (la boucle sur les points d'ancrage se fait par ordre décroissant de $\hat{p}$). Ainsi, nous n'utilisons pas ce point d'ancrage $\sigma(t)$ pour la régression (il est élagué) et nous fixons $y_{\sigma(t)} = 0$ afin de diminuer $\hat{p}_{\sigma(t)}$. Dans l'autre cas, le point d'ancrage est considéré comme compétitif ($(\alpha_{\sigma(t),n})_n = \mathbf{1}$) et participe à la fonction de perte de la régression. De plus, si la tIoU entre le segment régressé ($[\hat{s}_{\sigma(t)}, \hat{e}_{\sigma(t)}]$ et la vérité terrain $[s_n, e_n]$) est supérieur à μ, alors le point d'ancrage $\sigma(t)$ est considéré comme le meilleure pour prédire le segment de vérité terrain ($\beta_n = 1$) et $y_{\sigma(t)} = 1$ afin d'essayer d'augmenter $\hat{p}_{\sigma(t)}$.

Il s'agit donc d'un processus dynamique où, au début, tous les points d'ancrage

Algorithme 2 : Calul de la perte d'auto-évaluation

Entrées : $\{[\hat{s}_t, \hat{e}_t], \hat{p}_t\}_{t=1}^{T}$ les segments régressés à chaque point d'ancrage t et leurs scores d'auto-évaluation.
$\{[s_n, e_n]\}_{n=1}^{N}$ les segments de vérité terrain de la vidéo.
μ le seuil de tIoU.

Calculer $\sigma(t)$
/* la liste des temps t, triés par ordre décroissant de $\hat{p}_t$ */
/* $p_{\sigma(t+1)} \geq p_{\sigma(t)}$ */
Initialiser $\alpha = \mathbf{0}$, $\beta = \mathbf{0}$, $y = \mathbf{0}$
pour $t = 1, ..., T$ **faire**
  pour $n = 1, ..., N$ **faire**
    si $\beta_n = 0$ **alors**
      si $e_n \leq \sigma(t) \leq s_n$ **alors**
        $\alpha_{\sigma(t),n} \leftarrow 1$
        $\rho \leftarrow tIoU([\hat{s}_{\sigma(t)}, \hat{e}_{\sigma(t)}], [s_n, e_n])$ // Eq.5.1

        si $\rho > \mu$ **alors**
          $\beta_n \leftarrow 1$
          $y_{\sigma(t)} \leftarrow 1$
        sinon
        fin
      sinon
      fin
    sinon
    fin
  fin
fin
Calculer la valeur de la fonction de perte d'auto-évaluation L // en utilisant l'Eq.4.5

Sorties : Perte L

participent à la régression des bords des segments d'action. Puis, progressivement, certains points d'ancrage à faible potentiel sont élagués afin de concentrer la régression sur des prédictions pertinentes et de les améliorer. De plus, l'auto-évaluation évolue également : alors que la confiance dans les points d'ancrage non optimaux est encouragée à diminuer, la confiance dans les meilleurs points d'ancrages (qui correspondent à la vérité terrain avec une tIoU supérieure à μ et qui ont la plus grande confiance dans le segment correspondant) est incitée à augmenter.

Ce processus est illustré Figure 50 où cinq points d'ancrage produisent un segment régressé autour d'eux $[\hat{s}_t, \hat{e}_t]$, une valeur d'auto-évaluation $\hat{p}_t$ et une classe $\hat{c}_t$. La valeur d'auto-évaluation est représentée par un bleu plus ou moins foncé, selon l'échelle indiquée dans le coin supérieur gauche de la figure. Les propositions ayant une $tIoU$ nulle avec le segment de vérité terrain sont élaguées à l'aide d'une croix grise, tandis que celles élaguées par le processus d'auto-évaluation sont élaguées à l'aide d'une croix rouge.

Classification. Le modèle SALAD fait partie des méthodes de détection à deux étages (voir Section 2.6.1). La classification est effectuée en parallèle du processus de régression des segments. Nous avons choisi d'effectuer une classification au niveau des points d'ancrage. Ainsi, à chaque pas de temps, le réseau utilise la mémoire du point d'ancrage pour produire $\hat{c}_t$ une distribution de probabilité sur les classes d'action, y compris une classe de contexte. Nous maximisons le rappel des classes d'action sur les vidéos en utilisant la fonction de perte :

$$L_{cls} = \sum_{t=1}^{T} w_t \left(c_t \log(\hat{c}_t) + (1 - c_t) \log(1 - \hat{c}_t) \right) \tag{4.6}$$

où w_t est fixé à 0 si c_t est un fond, et 1 sinon.

Fonction de perte de SALAD. La fonction de perte globale utilisée pour l'apprentissage est la somme de la perte de régression/auto-évaluation et de la perte de classification (avec un paramètre de pondération λ_2) :

$$L = L_{r,sa} + \lambda_2 L_{cls} \tag{4.7}$$

Il est important de noter qu'il existe une perte de classification lors de l'apprentissage, mais que nous n'utilisons pas la valeur de confiance de la classification (voir l'étude sur l'ablation dans la Section 4.3.4). En effet, nous constatons que la classification est bonne, mais mal calibrée. Il n'y a donc aucun intérêt à fusionner la confiance de la classification avec notre score d'auto-évaluation de la régression.

Module	Réseau	Taille Entrée	Taille Sortie	Activation
Mémoire	GRU	2048	2048	Identité
Régression	Linéaire	2048	2048	ReLu
	Linéaire	2048	1024	ReLu
	Linéaire	1024	1024	ReLu
	Linéaire	1024	2	Sigmoide
Auto-Evaluation	Linéaire	2048	2048	ReLu
	Linéaire	2048	1024	ReLu
	Linéaire	1024	1024	ReLu
	Linéaire	1024	1	Sigmoide
Classification	Linéaire	2048	2048	ReLu
	Linéaire	2048	1024	ReLu
	Linéaire	1024	classes + 1	Softmax

TABLE 4.1 – Détails de l'architecture du réseau.

4.3 Expériences

Dans cette section, nous discutons d'abord des détails de notre implémentation. Ensuite, SALAD est comparé aux approches de l'état de l'art. Enfin, nous examinons la contribution de chaque composant de notre apprentissage par auto-évaluation à la tâche de détection d'actions à travers des études ablatives et des discussions.

4.3.1 Détails d'implémentation

Métrique de détection. La pratique courante en détections d'actions consiste à utiliser la mean average precision (mAP) à différents seuils de tIoU pour évaluer la qualité d'un ensemble de détections. Pour suivre les travaux précédents, les seuils de tIoU $\{0,1, 0,2, 0,3, 0,4, 0,5\}$ et $\{0,5, 0,75, 0,95\}$ sont respectivement utilisés pour THUMOS14 et ActivityNet1.3.

Caractéristiques. Pour les deux jeux de données, nous utilisons le même réseau d'extraction de caractéristiques, de manière préalable. Comme dans les travaux les plus récents (CHAO et al., 2018 ; ZENG et al., 2019), nous utilisons les caractéristiques d'une architecture à doubles flux (voir Section 2.3.2.2, extraites par le réseau I3D (CARREIRA et al., 2017), pré-entraîné sur Kinetics. Nous utilisons les caractéristiques pré-extraites fournies par PAUL et al., 2018. Les vidéos sont préalablement échantillonnées à 25 images par seconde, et l'algorithme de flux optique TV-L1 (PÉREZ et al., 2013) est appliqué. À partir de là, les caractéristiques sont extraites de tranches vidéo de 16 images sans chevauchements

pour produire 2 vecteurs de caractéristiques de taille 1024 (un vecteur pour le flux RVB et un vecteur pour le flux de mouvement).

Construction du réseau. L'architecture du réseau, présentée Figure 49, a été conçue avec les paramètres du Tableau 4.1. Les caractéristiques des images RVB et de flot optique sont calculées ensemble, elles sont ensuite fusionnées dans un vecteur de caractéristiques de 2048. Ainsi, nous conservons cette taille pour chaque vecteur latent GRU. Les trois têtes de réseau, pour la régression, la prédiction du score d'auto-évaluation et la classification, ont respectivement 4, 4 et 3 couches entièrement connectées dont le nombre de neurones est donné dans le Tableau 4.1. La régression des bords est effectuée par rapport à la position t, comme présenté Figure 49. Ainsi, pour une implémentation simplifiée, t est normalisé et la tête de régression produit une version normalisée de $[\epsilon_{t,start}, \epsilon_{t,end}]$. On a donc $[\epsilon_{t,start}, \epsilon_{t,end}] \in [0,1]^2$ et $[\hat{s}_t, \hat{e}_t] = [T \times (t - \epsilon_{t,start}), T \times (t + \epsilon_{t,end})]$.

Apprentissage et inférence. L'implémentation a été réalisée en utilisant Pytorch 1.0, Python 3.7 et CUDA 10.0. L'optimisation est effectuée en utilisant Adam, avec un *learning rate* initial de 10^{-4}. Pour le jeu de données THUMOS14, nous fixons la taille des échantillons de vidéo lors de l'apprentissage à 4 et pour ActivityNet1.3, nous la fixons à 16. Les deux jeux de données sont entraînés pendant 100 époques. Notons que la convergence est meilleure lorsqu'un pré-entraînement de la tête de classification est effectué avant l'entraînement complet, en utilisant $\lambda_1 = 1$, $\lambda_2 = 0.1$ et $\mu = 0.5$. Lors de l'inférence, nous utilisons toutes les propositions produites par le réseau, ainsi que le soft-NMS pour calculer les mAP sur THUMOS14 (une par pas de temps) alors que nous utilisons les 20 propositions ayant le score d'auto-évaluation le plus élevé pour ActivityNet1.3, puisque le nombre de vérités terrain par vidéo est plus faible.

4.3.2 Comparaison avec les résultats de l'état de l'art

THUMOS14. Le Tableau 4.2 compare notre modèle aux détecteurs de l'état de l'art sur le jeu de données THUMOS14. La méthode proposée obtient la mAP la plus élevée pour tous les seuils, ce qui montre que le processus d'auto-évaluation est capable bien détecter. En particulier, notre méthode surpasse de plus de 7% la précédente meilleure performance rapportée pour une tIoU@0.1 et améliore la mAP à tIoU@0.5 de 42,2% à 44,6%. Nous combinons également notre méthode avec P-GCN (Zeng et al., 2019), la méthode de post-traitement à l'état de l'art, qui vient exploiter les relations de dépendances entre les segmentes, à l'aide d'un *Graph Neural Network* (Kipf et al., 2016). Cette combinaison améliore légèrement nos résultats à chaque tIoU et surpasse toutes les méthodes de l'état de

l'art à tIoU $\leq 0,5$. Ces résultats montrent également que notre apprentissage par auto-évaluation ne nécessite pas autant de post-traitement que les autres méthodes dont les scores se détériorent sans lui.

tIoU	0.1	0.2	0.3	0.4	0.5
Oneata et al., 2014	36.6	33.6	27.0	20.8	14.4
L. Wang et al., 2014	18.2	17.0	14.0	11.7	8.3
Heilbron et al., 2016	-	-	-	-	13.5
J. Yuan et al., 2016	39.7	35.7	30.0	23.2	15.2
Shou et al., 2016	47.7	43.5	36.3	28.7	19.0
Yeung et al., 2016	48.9	44.0	36.0	26.4	17.1
Richard et al., 2016	51.4	42.6	33.6	26.1	18.8
DAPs (Escorcia et al., 2016)	-	-	-	-	13.9
SST (Buch et al., 2017)	-	-	37.8	-	23.0
CDC (Shou et al., 2017)	-	-	40.1	29.4	23.3
Z. Yuan et al., 2017	51.0	45.2	36.5	27.8	17.8
SS-TAD (Buch et al., 2019)	-	-	45.7	-	29.2
CBR (Gao et al., 2017a)	60.1	56.7	50.1	41.3	31.0
Hou et al., 2017	51.3	-	43.7	-	22.0
TCN (Dai et al., 2017)	-	-	-	33.3	25.6
TURN-TAP (Gao et al., 2017b)	54.0	50.9	44.1	34.9	25.6
R-C3D (H. Xu et al., 2017)	54.5	51.5	44.8	35.6	28.9
SSN (Zhao et al., 2017)	66.0	59.4	51.9	41.0	29.8
BSN (T. Lin et al., 2018)	-	-	53.5	45.0	36.9
BMN (T. Lin et al., 2019)	-	-	56.0	47.4	38.8
Chao et al., 2018	59.8	57.1	53.2	48.5	42.8
G-TAD (M. Xu et al., 2020)	-	-	54.5	47.6	40.2
SALAD	**73.3**	**70.7**	**65.7**	**57.0**	**44.6**
BSN + PGCN (Zeng et al., 2019)	69.5	67.8	63.6	57.8	49.1
G-TAD + PGCN (Zeng et al., 2019)	-	-	66.4	60.4	**51.6**
SALAD + PGCN	**75.2**	**73.4**	**69.4**	**61.6**	49.8

Table 4.2 – **Résultats de détection sur le jeu de test de THUMOS14**, mesurés en mAP (%) pour différents seuils de tIoU. SALAD surpasse significativement toutes les autres méthodes pour toutes les tIoU et est même légèrement amélioré par une combinaison P-GCN.

ActivityNet1.3. Le Tableau 4.3 présente les résultats de l'état de l'art sur le jeu de données ActivityNet1.3. Notre algorithme surpasse de 1,4 % la meilleure performance précédente pour une tIoU@0.5. À une tIoU plus élevée, certaines méthodes de l'état de l'art sont plus efficaces que la nôtre. P-GCN (Zeng et al., 2019) ne proposant pas d'implémentation pour ce jeu de données, cette méthode n'est pas utilisée ici.

tIoU	0.5	0.75	0.95	Average
G. Singh et al., 2016	34.47	-	-	-
SCC (Heilbron et al., 2017)	40.00	17.90	4.70	21.70
CDC (Shou et al., 2017)	45.30	26.00	0.20	23.80
R-C3D (H. Xu et al., 2017)	26.80	-	-	-
SSN (Zhao et al., 2017)	39.12	23.48	5.49	23.98
BSN (T. Lin et al., 2018)	46.45	29.96	8.02	30.03
Chao et al., 2018	38.23	18.30	1.30	20.22
P-GCN (Zeng et al., 2019)	48.26	33.16	3.27	31.11
BMN (T. Lin et al., 2019)	50.07	**34.78**	8.29	33.85
G-TAD (M. Xu et al., 2020)	50.36	34.60	**9.02**	**34.09**
SALAD	**51.72**	31.21	3.33	31.02

Table 4.3 – **Résultats de détection sur le jeu de validation d'ActivityNet1.3**, mesurés en (%) pour différents seuils de tIoU et en mAP moyenne. SALAD obtient la meilleur performance pour une tIoU@0.5.

Cependant, tous les algorithmes de l'état de l'art sont peu performants pour des valeurs élevées de tIoU, en particulier pour 0,95 où le meilleur algorithme n'atteint que 9% de mAP. Ainsi, nous considérons que toutes les méthodes actuelles ne sont pas suffisamment matures pour gérer la détection d'actions à des tIoU élevées.

Il n'est pas surprenant que l'auto-évaluation ne soit pas utile sur un problème aussi ambigu : plus un problème est ambigu, moins il est possible d'avoir une auto-évaluation cohérente.

Maintenant, même sur ActivityNet, nous améliorons la mAP à tIoU ≤ 0.5. Bien que ces résultats ne soient pas conventionnellement rapportés, nous comparons SALAD à des tIoU inférieures avec BMN (T. Lin et al., 2019), qui est la meilleure méthode open-source disponible au moment de la rédaction (*JJBOY/BMN-Boundary-Matching-Network*). Les résultats de SALAD, présentés dans le Tableau 4.4, surpassent clairement ceux de BMN, pour toutes les tIoU ≤ 0.5.

On peut supposer que lorsque la détection est réalisée pour des tIoU ≤ 0.5, le problème devient moins ambigu et l'algorithme SALAD est alors capable de surpasser les méthodes de l'état de l'art sur les jeux de données THUMOS14 et ActivityNet1.3.

tIoU	0.1	0.2	0.3	0.4	0.5
BMN (T. Lin et al., 2019)	70.91	64.46	58.79	54.14	50.07
SALAD	**77.68**	**70.66**	**64.06**	**57.45**	**51.72**

Table 4.4 – **Résultats de détection sur le jeu de validation d'ActivityNet1.3**, mesurés en (%) pour des seuils de tIoU plus faibles que 0.5. SALAD obtient des résults significativement meilleurs que BMN à faible tIoU.

4.3.3 Études ablatives / Discussions

L'objectif initial de SALAD est d'apprendre conjointement la régression et d'attribuer un score aux segments par auto-évaluation. Un tel apprentissage améliore les performances comme nous l'avons montré précédemment (en plus de donner une fonction d'auto-évaluation au réseau). Un tel apprentissage conduit, en particulier, à une amélioration de la qualité de l'utilisation des caractéristiques et donc à une augmentation des performances comme montré précédemment.

Dans cette section, nous décomposons la fonction de perte proposée afin de comprendre quelles parties sont importantes pour l'amélioration des résultats et de mieux comprendre ce qui conduit à cette amélioration significative. Nous continuons à ne pas utiliser les points d'ancrage, lors de d'entraînement, en dehors des segments de vérité terrain (comme dans la Section 4.1.3).

Elagage. Nous étudions d'abord l'influence de l'élagage introduit Equation 4.5 avec les paramètres $\alpha_{t,n}$.

Précisément, nous comparons cinq méthodes. (*i*) Pas d'élagage ($(\alpha_{t,n})_{t,n} = \mathbf{1}$). (*ii*) Élagage aléatoire où les valeurs de $(\alpha_{t,n})$ sont fixées aléatoirement à 0 ou 1. (*iii*) élagage du top 1 de tIoU, où seule la proposition avec la meilleure tIoU est régressée. (*iv*) Élagage gelé qui consiste à extraire les valeurs finales $(\alpha_{t,n})$ de SALAD et à réapprendre l'algorithme depuis le début avec ces valeurs gelées. (*v*) SALAD.

Les résultats, présentés Tableau 4.5, montrent clairement que l'élagage est un point important de l'algorithme, puisque trop d'élagage (top 1) conduit à de très mauvais résultats et aucun élagage ou un élagage aléatoire à de mauvais résultats. Ceci est cohérent avec les travaux précédents comme ceux de Z. Wu et al., 2019 ou de W. Wu et al., 2019 qui montrent également que l'élagage est important.

Plus important encore, en utilisant, à l'apprentissage, l'élagage figé de notre meilleur modèle, les performances sont moins bonnes. Ainsi, la régression seule ne permet pas d'obtenir des résultats similaires car il semble important d'adapter l'élagage au comportement de l'algorithme. Ainsi, même si l'élagage est un élément clé, il n'est pas le seul, ce qui met en évidence la pertinence de l'apprentissage par auto-évaluation.

mAP@tIoU	0.1	0.2	0.3	0.4	0.5
Sans élagage	66.2	63.0	57.0	46.9	32.0
Top 1 tIoU	55.7	53.4	48.3	38.9	27.4
Aléatoire	65.8	62.6	56.6	46.2	32.9
Gelé	63.2	57.5	45.4	45.4	31.5
SALAD (élagage)	**73.3**	**70.7**	**65.7**	**57.0**	**44.6**

TABLE 4.5 – Comparaison entre notre entrainement de SALAD et différentes méthodes d'élagage et stratégies de régression pour THUMOS14, mesurés en mAP(%)

Apprentissage du score d'auto-évaluation. Nous étudions ensuite l'influence de l'auto-évaluation (paramètre y_t dans l'Equation 4.5) et comparons différentes idées. La première consiste à ignorer la tIoU et à fixer $y_t = 1$ pour le point d'ancrage avec le plus haut score $\hat{p}_t$ à l'intérieur de chaque segment de vérité terrain. La deuxième idée est que tout segment dont le niveau de score d'auto-évaluation $\hat{p}_t$ est supérieur à 0.5 est considéré comme sûr. La dernière idée impose une condition sur la tIoU, qui doit être supérieure au seuil μ (sans considérer $\hat{p}_t$). Ainsi, les deux dernières méthodes ne prennent pas en compte les segments voisins lors de l'attribution du score d'auto-évaluation.

mAP@tIoU	0.1	0.2	0.3	0.4	0.5
$y_t = 1 \Leftrightarrow t = \sigma(0)$	59.4	56.7	51.3	42.0	30.6
$y_t = 1 \Leftrightarrow \hat{p}_t > 0.5$	66.4	62.7	53.8	41.2	26.7
$y_t = 1 \Leftrightarrow tIoU_t > \mu$	65.5	62.3	53.7	40.9	28.0
SALAD	**73.3**	**70.7**	**65.7**	**57.0**	**44.6**

TABLE 4.6 – Comparaison entre SALAD et d'autres stratégies d'auto-évaluation sur THUMOS14, mesuré en mAP(%).

Les résultats, présentés dans le Tableau 4.6, montrent que les trois affectations alternatives de y_t conduisent à une chute spectaculaire des performances par rapport à SALAD. Ils mettent en évidence que la pertinence de l'auto-évaluation

provient de l'introduction de spécificités de détection d'action au cours du processus d'apprentissage, comme par exemple, autoriser un seul segment prédit par segment de vérité terrain. Ainsi, pour un segment de vérité terrain donné, la régression, par exemple, du segment ayant le meilleur score d'auto-évaluation, dégrade les résultats. De même, la régression des segments avec un score élevé, sans prendre en compte les autres segments, ou la régression des seuls segments avec une bonne tIoU, ne sont pas optimales.

Les études précédentes sur l'élagage et l'apprentissage par auto-évaluation établissent clairement que le succès de SALAD provient de sa capacité à élaguer les points d'ancrage pendant l'apprentissage et du processus d'auto-évaluation qui permet d'injecter des *a priori* sur la métrique mAP qui peuvent difficilement être injectées dans la fonction de perte locale classique.

4.3.4 Confiance en la classification

Dans toutes les expériences présentées précédemment, les scores de la classification n'ont pas été utilisés (la tête de classification produit une distribution de probabilité sur les classes à partir de laquelle un niveau de confiance naïf peut être obtenu). Cependant, on peut se demander s'il est pertinent de fusionner les deux confiances. Notons $p_r = \hat{p}_t$ et $p_c = \max_i \hat{c}_t^i$ le score de régression et celui de classification.

Ainsi, nous présentons dans le Tableau 4.7 une comparaison avec différentes stratégies de fusion : la moyenne arithmétique des deux scores $\frac{p_r+p_c}{2}$, leur produit géométrique $\sqrt{p_r\,p_c}$ et le produit du score d'auto-évaluation de la régression avec le score de classification normalisé $p_r \times (1 - \exp(-\zeta p_c))$. L'idée de cette dernière méthode est de diminuer le score d'auto-évaluation en cas d'ambiguïté importante dans la classification. Enfin, SALAD utilise uniquement le score d'auto-évaluation de la régression : $p_r + 0 \times p_c$.

mAP@tIoU	0.1	0.2	0.3	0.4	0.5
Moyenne arithmétique	63.5	61.7	58.2	51.1	41.1
Moyenne géométrique	65.7	63.8	60.0	52.7	42.4
Produit normalisé	67.2	65.2	64.5	53.8	43.2
SALAD	**73.3**	**70.7**	**65.7**	**57.0**	**44.6**

TABLE 4.7 – Comparaison entre SALAD et des méthodes de fusion des scores de classification et d'auto-évaluation sur THUMOS14, mesurée en mAP(%).

Les résultats montrent clairement qu'il n'est pas pertinent de fusionner le score

de notre auto-évaluation avec celui de la classification, ce qui explique pourquoi SALAD ne le fait pas.

4.3.5 Résultats qualitatifs

Dans la Figure 51, nous présentons quelques résultats de localisation sur le jeu de données THUMOS14. Bien sûr, l'affichage de quelques échantillons fournit des informations limitées sur le comportement global de notre algorithme. Cependant, un point très important que nous voulons souligner est que les points d'ancrage utilisés pour régresser les segments varient le long des segments de vérité terrain. Il est donc très important de sélectionner les meilleurs points d'ancrage pendant l'apprentissage, comme proposé dans SALAD.

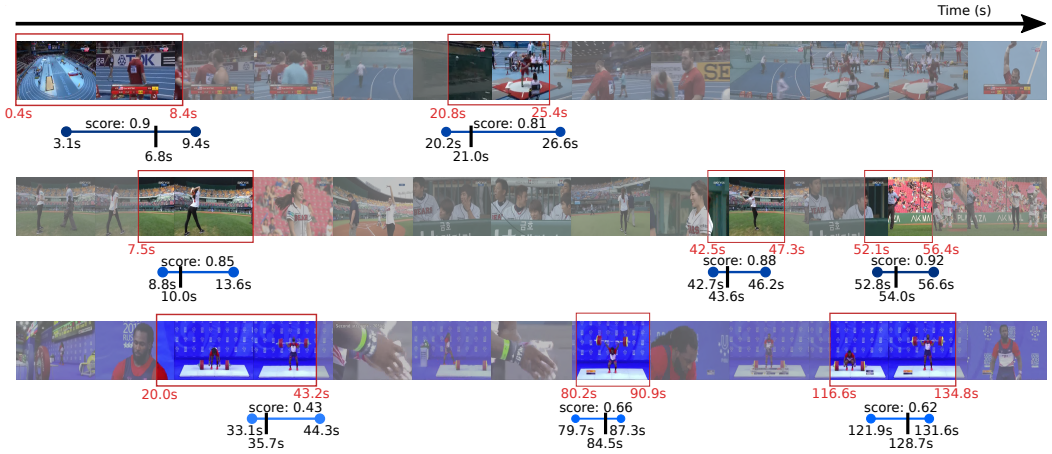


FIGURE 51 – **Résultats qualitatifs.** Nous montrons des résultats qualitatifs de localisation sur le jeu de test de THUMOS14. Les segments de vérité terrain sont les boîtes rouges. Les prédictions réalisées par SALAD sont les segments en dégradés de bleu et les points d'ancrage des prédictions sont les pas de temps en noir.

4.4 Conclusion du chapitre

Dans ce chapitre, nous proposons un nouvel algorithme de détection d'action nommé SALAD qui surpasse les performances de l'état de l'art sur les jeux de données THUMOS14 et ActivityNet1.3.

Ce gain de performance est obtenu en ajoutant un processus d'auto-évaluation directement dans l'apprentissage du réseau. En effet, cette auto-évaluation permet d'élaguer certains points d'ancrage et d'améliorer l'utilisation des caractéristiques en utilisant un mécanisme attentionnel et une régularisation multi-tâches.

De plus, cette auto-évaluation permet de capturer toute la spécificité de la métrique de détection d'action dans la fonction de perte, contrairement aux fonctions de pertes de régression qui ne mesurent que la performance locale. Ainsi, même si la recherche de robustesse, de calibration ou d'explicabilité des algorithmes introduit souvent des contraintes à l'apprentissage dégradant les performances, nous utilisons l'une d'entre elles pendant le processus d'apprentissage comme un moyen d'améliorer le processus de détection.

Chapitre 5

Affinement des détections

Sommaire

5.1 Introduction

5.1.1 Détection d'actions à hautes tIoU

Au moins deux cas de figure peuvent nécessiter une détection à forte tIoU. Le premier est la détection fine des actions, par exemple dans des contextes de robotique autonome. Le second est la détection d'actions de longue durée,

ou occupant une part importante de la durée de la vidéo. En effet, de manière structurelle, dans ce cas, détecter ces actions avec une tIoU de 0.5 n'apporte que peu d'information.

On rappelle que la *mean Average Precision* (mAP), métrique d'évaluation des problèmes de détection, que ce soit dans un contexte d'analyse d'images ou d'analyse de vidéos dépend de l'*Intersection over Union* (tIoU), mesure de similarité entre deux régions introduite en Section 3.3. Dans le cas particulier de la détection d'action dans des vidéos, la tIoU est une mesure de similarité en 1 dimension. On rappelle le calcul de la tIoU entre un segment prédit $[\hat{s}, \hat{e}]$ et un segment de la vérité terrain $[s, e]$:

$$tIoU([\hat{s}, \hat{e}], [s, e]) = \frac{|[\hat{s}, \hat{e}] \cap [s, e]|}{|[\hat{s}, \hat{e}] \cup [s, e]|} \tag{5.1}$$

L'inverse de la valeur de tIoU peut donc être considéré comme une valeur d'incertitude sur l'intersection entre la proposition d'action et la vérité terrain, et donc comme une valeur d'incertitude sur la précision du positionnement de cette proposition. Plus la tIoU entre une proposition et la vérité terrain se rapproche de 1 et plus la proposition a été positionnée précisément.

Logiquement, plus la tIoU seuil recherchée lors de la conception d'un algorithme de détection d'actions est importante, et plus il est compliqué d'atteindre une mAP élevée. La Figure 52 montre la corrélation négative entre la recherche d'une tIoU élevée et la mAP, pour SALAD (Vaudaux-Ruth et al., 2021b) et BMN (T. Lin et al., 2019).

5.1.2 SALAD pour la détection à hautes tIoU

Le modèle SALAD est une méthode de détection d'actions basée sur une architecture descendante. Cette méthode de détection a montré de bonnes dispositions pour la détection d'action sur les datasets de l'état de l'art THUMOS14 et ActivityNet, en améliorant la mAP@0.5 respectivement de 42.8% à 44.6% et de 50.4% à 51.7%. Malgré tout, lorsqu'une analyse précise est résultats de détection à forte tIoU est réalisée, il apparaît que SALAD ne conserve pas le même avantage sur l'état de l'art.

En effet, lorsqu'on compare la courbe mAP/tIoU du modèle SALAD à celle de BMN, on constate que les courbes se croisent pour une tIoU de 0.5. Avant, c'est le modèle SALAD qui est le plus performant, ensuite c'est le modèle BMN. Cela nous a amenés à chercher à comprendre pourquoi le modèle de régression SALAD est proportionnellement moins performant à hautes tIoU qu'à faibles tIoU, et s'il

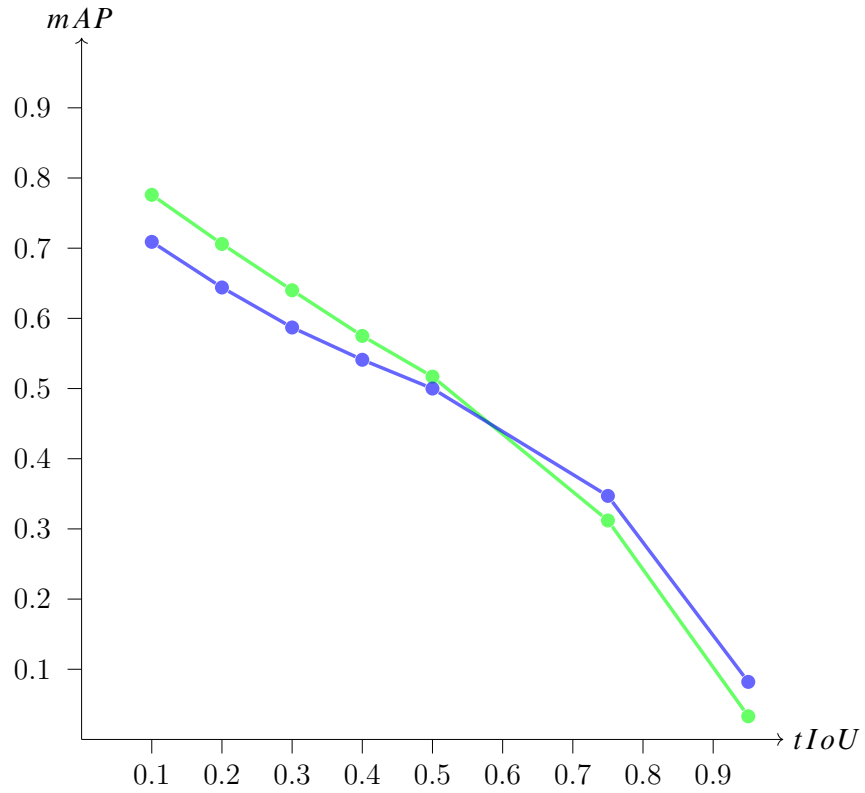


FIGURE 52 – Décroissance de la mAP en fonction du seuil de tIoU utilisé. La méthode SALAD est représentée en vert et la méthode BMN en bleu.

était possible d'améliorer ces résultats.

Dans ce chapitre, 3 pistes pour améliorer les performances à hautes tIoU sont présentées.

5.2 Hybridation du modèle SALAD

5.2.1 Retour sur les méthodes ascendantes

Les architectures ascendantes utilisent des informations locales (images ou courtes portions de vidéo) pour localement attribuer des scores locaux d'activité ou de frontières (début/fin d'action). Ces scores locaux sont ensuite combinés et utilisés afin de réaliser des propositions de segments temporels. Parmi ces méthodes on trouve notamment BSN (T. LIN et al., 2018) et BMN (T. LIN et al., 2019), deux méthodes plus précises que SALAD à hautes tIoU.

L'objectif est alors d'hybrider (voir Section 2.5.3) l'architecture SALAD en la combinant avec une approche ascendante, comme les pré-citées, afin d'extraire de l'information plus localement et de voir si ce complément d'information début/fin/action permet d'améliorer la précision de SALAD à haute tIoU.

5.2.2 Raffinement local par méthode ascendante

On sait que les méthodes locales BSN et BMN, utilisant à la fois une probabilité d'action mais aussi des probabilités de début et fin d'actions, ont révélé d'importantes dispositions pour proposer des appariements de début/fin et par suite des propositions précises à haute tIoU.

Là ou les méthodes hybrides utilisent principalement le score d'activité des méthodes ascendantes pour affiner le score des détections, nous allons ici utiliser les scores de début/fin d'action afin de venir affiner les bords des propositions sans modifier le score.

Une manière d'améliorer les propositions provenant de la méthode de régression SALAD est donc de venir ajuster les bords en cherchant l'argument maximum d'un score local de caractérisation début/fin dans un voisinage des bords prédits par le modèle SALAD (voir Figure 53).

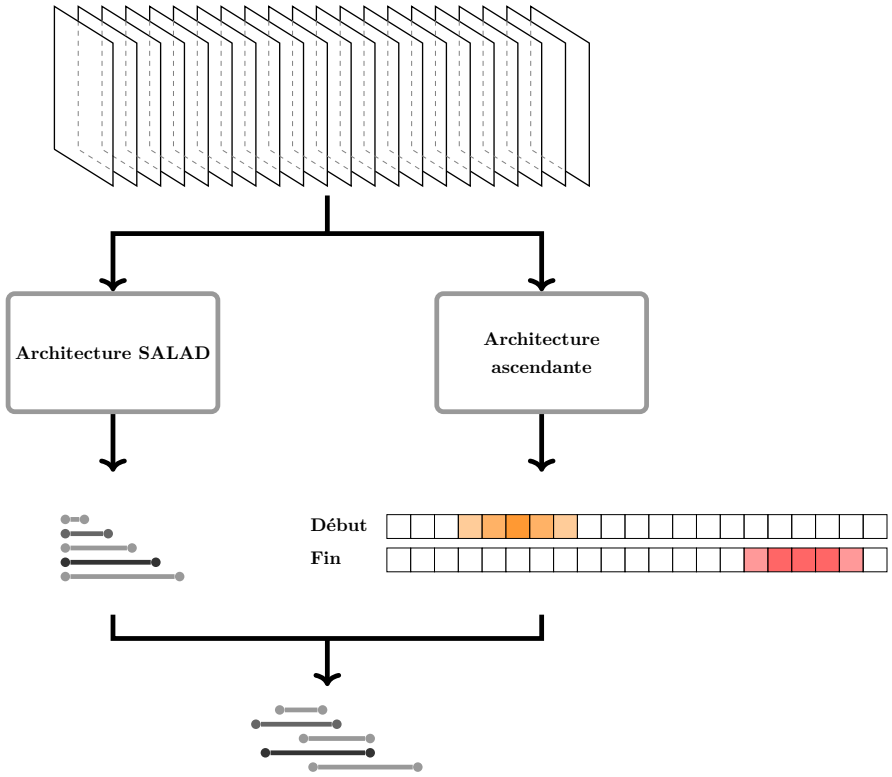


FIGURE 53 – **Hybridation du modèle SALAD.** Des prédictions locales de début/fin d'action réalisées par une architecture ascendante viennent affiner les bords des prédictions du modèle SALAD.

5.2.3 Architecture

Une architecture neuronale indépendante de celle de SALAD va venir générer des probabilités locales d'appartenance aux bords des actions. Soit $\mathbf{V} = \boldsymbol{x}_1, ..., \boldsymbol{x}_T$

avec T images (ou T portions de vidéo) une vidéo où $\boldsymbol{x}_t$ désigne le vecteur de caractéristiques de la t-ième image (ou t-ième portion de vidéo) à partir d'un réseau d'extraction de caractéristiques de l'état de l'art. Alors on définit $\mathbf{S} = \{\hat{p}_t^s\}_{t=1}^T$, $\mathbf{E} = \{\hat{p}_t^e\}_{t=1}^T$respectivement la séquence de probabilités locales d'être un début d'action et une fin d'action. Les probabilités précédentes sont prédites par deux réseaux respectifs f^s et f^e composés d'une structure mémoire de type GRU commune suivie d'une structure linéaire respective (voir Figure 54), tels que $\hat{p}_t^s = f^s(\boldsymbol{x}_t)$ et $\hat{p}_t^e = f^e(\boldsymbol{x}_t)$. Alors l'objectif est, pour chaque segment $[\hat{s}_t, \hat{e}_t]$ prédit par l'algorithme SALAD, de venir affiner $\hat{s}_t$ et $\hat{e}_t$ en tirant parti des scores $\mathbf{S}$ et $\mathbf{E}$, dans un voisinage de j images noté $v^j(\boldsymbol{x}_t) = \{\boldsymbol{x}_{t+i}\}_{i=-j}^j$.

5.2.4 Fonction de raffinement

Indépendamment du voisinage qui sera utilisé pour le raffinement des propositions, et en suivant la méthode de T. Lin et al., 2018, la manière de venir caractériser les images de la vidéo en tant que début ou fin est construite de manière régionale. Pour chaque segment $[s_n, e_n]$ de la vérité terrain, on pose $d_n = e_n - s_n$ la durée de l'action, $r_n^s = [s_n - \frac{d_n}{5}, s_n + \frac{d_n}{5}]$ et $r_n^e = [e_n - \frac{d_n}{5}, e_n + \frac{d_n}{5}]$ les régions de début de fin. Pour chaque $t \in [0, ..., T]$, on va définir g_t^s et g_t^e la vérité terrain d'être respectivement une image de début et une image de fin. g_t^s et g_t^e sont définis comme le score maximal d'intersection (IoP) entre les régions r_n^i et $[t - \delta, t + \delta]$, l'intervalle temporel entre deux images successives de la vidéo.

La fonction de perte des modules de prédiction des probabilités locales d'être un début ou une fin d'action est alors la suivante :

$$L = L_{start} + L_{end} \tag{5.2}$$

Les deux composantes sont définies comme la somme d'une régression logistique binaire, et sont définies ainsi :

$$L_i = \frac{1}{T} \sum_{t=1}^{T} (b_t . \log(\hat{p}_t^i) + (1 - b_t) . \log(1 - \hat{p}_t^i)), \tag{5.3}$$

où $b_t = \text{signe}(g_t^i - \text{th}_{IoP})$ est une valeur binaire basée sur un seuil d'intersection, fixé à 0.5 par la suite.

Ensuite, pour chaque prédiction SALAD $[\hat{s}_t, \hat{e}_t]$, $t \in [0, ..., T]$ si une image dans le voisinage des bords a une probabilité plus importante d'être un début (respectivement une fin), au sens du modèle local, alors l'intervalle est ajusté pour commencer (respectivement terminer) sur cette image. Le raffinement est

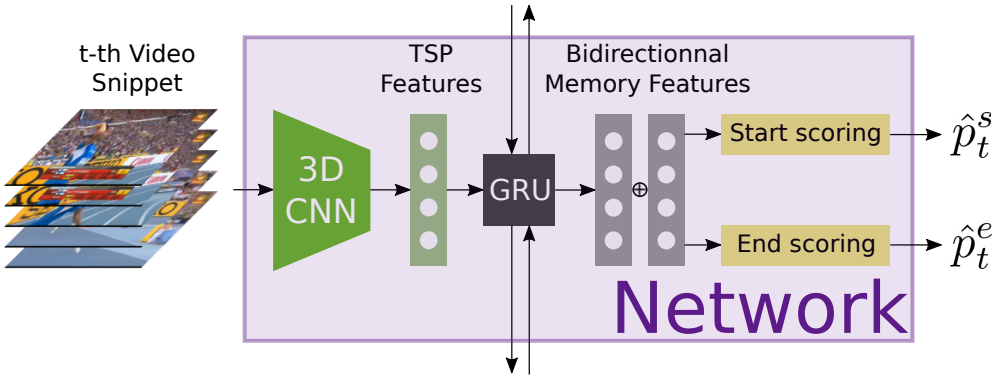


FIGURE 54 – **Architecture de la branche ascendante du modèle hybride.** Chaque image (ou portion de vidéo) t est d'abord représentée par un vecteur de caractéristiques extrait grâce à un réseau de l'état de l'art. Un GRU bidirectionnel est ensuite utilisé pour produire une mémoire des images précédentes et une mémoire des images suivantes. Les deux mémoires sont ensuite utilisées par deux modules qui estiment s'il est probable que l'image soit un début ($\hat{p}_t^s$) ou une fin ($\hat{p}_t^e$). Ces scores vont permettre de venir affiner les segments prédits par la branche SALAD.

alors le suivant :

$$\hat{s'}_t = \underset{t' \in v^j(\hat{s}_t)}{\operatorname{argmax}} \hat{p}_{t'}^s \tag{5.4}$$

$$\hat{e'}_t = \underset{t' \in v^j(\hat{e}_t)}{\operatorname{argmax}} \hat{p}_{t'}^e \tag{5.5}$$

5.2.5 Expériences

5.2.5.1 Détails d'implémentation

Caractéristiques. Le jeu de données utilisé est ActivityNet1.3. Pour les méthodes de raffinement proposées, on utilise non pas les images vidéos directement mais des caractéristiques pré-extraites. Par contre, les caractéristiques utilisées sont différentes de celles utilisées dans le Chapitre 4 pour l'évaluation du modèle SALAD. Alors que les caractéristiques utilisées pour l'évaluation du modèle SALAD étaient les caractéristiques I3D (CARREIRA et al., 2017), les caractéristiques utilisées ici sont les caractéristiques TSP (ALWASSEL et al., 2020). Ces caractéristiques ont l'avantage d'être moins volumineuses, et donc de réduire les temps de calcul, pour des résultats au moins équivalents. L'apport du raffinement sur le modèle SALAD est donc comparé en utilisant ces caractéristiques, mais la même analyse aurait pu être réalisée avec n'importe quelle autre extraction.

Construction du réseau. Comme ici, seule la localisation des actions nous intéresse, et non pas la classification des actions, nous simulons une classification

	Module	Réseau	Taille Entrée	Taille Sortie	Activation
Branche SALAD	Mémoire	GRU	512	512	Identité
	Régression	Linéaire	1024	1024	ReLu
		Linéaire	1024	512	ReLu
		Linéaire	512	512	ReLu
		Linéaire	512	2	Sigmoide
	Auto-Evaluation	Linéaire	1024	1024	ReLu
		Linéaire	1024	512	ReLu
		Linéaire	512	512	ReLu
		Linéaire	512	1	Sigmoide
Branche de raffinement	Mémoire	GRU	512	512	Identité
	Classification début	Linéaire	1024	1024	ReLu
		Linéaire	1024	512	ReLu
		Linéaire	512	512	ReLu
		Linéaire	512	1	Sigmoide
	Classification fin	Linéaire	1024	1024	ReLu
		Linéaire	1024	512	ReLu
		Linéaire	512	512	ReLu
		Linéaire	512	1	Sigmoide

TABLE 5.1 – Détails de l'architecture du réseau hybridé.

parfaite pour l'ensemble des expériences. Pour le jeu de données ActivityNet1.3, cela est possible car chaque vidéo ne comporte qu'une seule classe d'actions. Il suffit donc d'utiliser une classification parfaite des vidéos, appelée oracle.

L'architecture du réseau est constituée de deux branches parallèles (cf Figure 53). La première branche est exactement l'architecture du modèle SALAD, sauf que les caractéristiques passent d'une taille de 2048 à 512 (voir Section 4.3.1). La branche de raffinement a une structure similaire. Elle est composée elle aussi d'un réseau récurrent bidirectionnel produisant deux mémoires à chaque instant t de la vidéo, qui seront utilisées pour déterminer localement si les images peuvent être considérées comme des débuts ou comme des fins, grâce à deux têtes linéaires comportant très exactement 4 couches. Les caractéristiques utilisées ayant une taille différente, le détail de l'ensemble de l'architecture est donné dans le Tableau 5.1.

Apprentissage et inférence. L'implémentation a été réalisée en utilisant Pytorch 1.8, Python 3.8 et CUDA 11.2. Les caractéristiques vidéos sont échantillonnées pour avoir une dimension temporelle de taille 100 maximum. L'optimisation est effectuée en utilisant Adam, avec un *learning rate* initial de 10^{-4}. La taille d'échantillons utilisée est de 16. La branche SALAD est optimisée d'abord

mAP@tIoU	0.5	0.55	0.6	0.65	0.70	0.75	0.8	0.85	0.9	0.95
SALAD	53.0	49.8	46.3	42.4	38.2	33.3	27.4	20.8	13.4	5.5
Hybride	**54.6**	**51.1**	**47.7**	**44.5**	**40.3**	**36.0**	**30.4**	**23.9**	**18.0**	**8.5**

TABLE 5.2 – **Résultats de détection sur le jeu de validation d'ActivityNet1.3**, mesurés en (%). L'hybridation du modèle SALAD permet d'améliorer la mAP à toutes les tIoU lors d'un raffinement des bords dans un voisinage de 4 images.

de manière individuelle. Ensuite les poids de la branche SALAD sont fixés, puis la branche de raffinement est optimisée. Lors de l'inférence, une soft-NMS est utilisée sur l'ensemble des propositions, après raffinement. Pour le calcul de la mAP, sont ensuite extraites par vidéo les 10 propositions ayant le score d'auto-évaluation le plus élevé.

5.2.5.2 Comparaison avec le modèle SALAD

Le Tableau 5.2 présente la mAP du modèle SALAD avec les caractéristiques TSP et une classification parfaite, ainsi que le modèle hybridé avec un entrainement conjoint des deux branches et un raffinement local des propositions sur un voisinage de 4 images. La Figure 55 présente l'amélioration relative apportée par cette hybridation, à différentes tIoU. On constate que l'hybridation apporte 55% d'amélioration pour une IoU de 0.95, 27% à 0.9 ou encore 15% à 0.85. L'hybridation entraînant le raffinement permet donc une amélioration relative significative du modèle SALAD à forte tIoU.

5.2.5.3 Etudes / Discussions

Le raffinement réalisé grâce à l'hybridation du modèle SALAD dépend de la taille du voisinage. Dans les Tableaux 5.3, 5.4, 5.5, 5.6 on trouve une comparaison de scores pour un voisinage de respectivement 1, 2, 3 et 4 images.

Pour chaque voisinage, on compare le modèle SALAD avec le modèle SALAD hybridé, mais aussi un raffinement optimal, appelé oracle, et un raffinement réalisé avec les scores de début/fin issus du modèle BMN (T. LIN et al., 2019). Pour chaque segment prédit par le modèle SALAD, si la vérité terrain contient un bord dans le voisinage de raffinement, le raffinement par oracle ramène les bords sur la vérité terrain. Le raffinement avec les scores BMN est réalisé de la même manière que SALAD hybridé.

On constate que plus la taille du voisinage augmente et plus la capacité de raffinement augmente (modèle oracle). L'hybridation permet aussi de suivre la

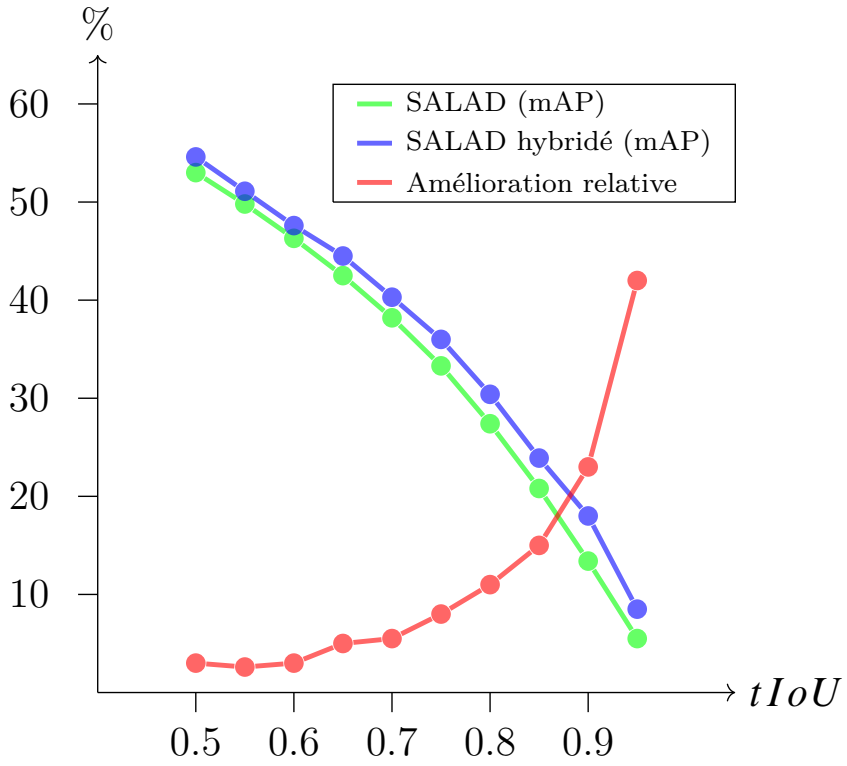


FIGURE 55 – Apport de l'hybridation du modèle SALAD à différentes tIoU.

tendance de l'oracle. Pour chaque voisinage, le modèle SALAD hybridé permet d'extraire des scores de début/fin mieux adaptés au raffinement du modèle SALAD que ceux du modèle BMN, qui améliorent significativement la mAP du modèle SALAD, à toutes les tIoU.

Le meilleur raffinement est atteint pour le modèle hybridé avec un voisinage de 4 images et un apprentissage couplé (cf Tableau 5.6). Pour l'apprentissage couplé, la branche de raffinement utilise la cellule mémoire (GRU bidirectionnel) de la branche SALAD et l'apprentissage de la branche de raffinement vient peaufiner l'optimisation des poids de la branche SALAD. L'apport d'un apprentissage multi-tâche permet donc d'améliorer l'extraction de contenu et donc d'améliorer conjointement les prédictions SALAD et le raffinement.

Mais, si le raffinement optimal était atteint, il devrait tendre vers celui du modèle oracle. La mAP à tIoU de 0.95 devrait donc être améliorée de près de 10%, alors qu'elle n'est améliorée que de 3%. Les caractéristiques locales ne sont finalement pas suffisantes, ce qui implique que les bords des actions n'ont pas une saillance locale évidente. De cette manière, il est seulement possible de conclure que les méthodes ascendantes ont une structure qui apporte de l'information au modèle SALAD et qui permet d'améliorer les propositions à haute IoU.

mAP @ tIoU pour un voisinage de 1 image	0.5	0.75	0.95
SALAD	53.02	33.35	5.45
Oracle	53.86	34.65	7.03
Raffinement BMN	53.17	33.5	5.75
SALAD hybridé	53.52	34.55	6.68

TABLE 5.3 – **Comparaison entre les différentes méthodes de raffinement local dans un voisinage d'1 image.**

mAP @ tIoU pour un voisinage de 2 images	0.5	0.75	0.95
SALAD	53.02	33.35	5.45
Oracle	54.63	36.09	9.03
Raffinement BMN	53.03	33.54	6.15
SALAD hybridé	54.09	35.03	7.06

TABLE 5.4 – **Comparaison entre les différentes méthodes de raffinement local dans un voisinage de 2 images.**

mAP @ tIoU pour un voisinage de 3 images	0.5	0.75	0.95
SALAD	53.02	33.35	5.45
Oracle	55.27	37.43	11.36
Raffinement BMN	53.15	33.48	6.4
SALAD hybridé	54.21	35.12	7.8

TABLE 5.5 – **Comparaison entre les différentes méthodes de raffinement local dans un voisinage de 3 images.**

mAP @ tIoU pour un voisinage de 4 images	0.5	0.75	0.95
SALAD	53.02	33.35	5.45
Oracle	56.8	40.43	14.43
Raffinement BMN	53.15	33.77	6.73
SALAD hybridé	54.21	35.61	8.36
SALAD hybridé, apprentissage couplé	**54.6**	**36.0**	**8.5**

TABLE 5.6 – **Comparaison entre les différentes méthodes de raffinement local dans un voisinage de 4 images.**

5.3 Pooling Pyramidal pour SALAD

Il a été montré dans la partie précédente que si, pour avoir des bords précis, il suffisait de les caractériser localement, alors le raffinement par hybridation du modèle SALAD devrait être presque aussi précis que le raffinement par oracle. Mais le raffinement par hybridation n'équivaut qu'à 30% de l'oracle. Il est donc nécessaire d'augmenter encore l'information temporelle en enrichissant temporellement les caractéristiques.

5.3.1 Raffinement temporel par méthode pyramidale

Structurellement, SALAD est construit pour analyser des séquences temporelles grâce à un réseau récurrent de type GRU. Il est connu que les réseaux récurrents peuvent souffrir d'une mémoire à long terme appauvrie. La problématique peut d'autant plus se matérialiser dans un contexte d'analyse vidéo, là ou une séquence d'une seconde comporte déjà entre 25 et 30 images, contenant plusieurs milliers de caractéristiques.

Venir mettre en correspondance le début et la fin d'une action à l'aide d'un réseau récurrent peut donc être compliqué. D'autant plus lorsqu'il s'agit de régresser la localisation relativement à un point d'ancrage comme dans la méthode SALAD. Être précis à haute IoU peut donc être structurellement compliqué.

Ainsi, bien qu'un réseau récurrent ait théoriquement un champ récepteur plus ou moins infini, le champ récepteur empirique ne l'est pas. Pour s'adapter à la spécificité du GRU, une manière d'augmenter sa résolution temporelle est de l'utiliser en réalisant une analyse temporelle pyramidale de la vidéo. L'idée ici est d'enrichir les informations temporelles en réalisant différents *pooling* temporels le long de la vidéo. Ensuite des mémoires temporelles à différentes résolutions sont créées à l'aide du GRU. Ces différentes mémoires sont ensuite utilisées conjointement pour réaliser les prédictions d'actions.

Dans le modèle SALAD, une prédiction par image est réalisée. Dans le modèle pyramidal (noté PP pour pyramide de *pooling* dans la suite), différentes résolutions temporelles sont obtenues grâce à différents *pooling* (cf Figure 56).

A l'aide d'un échantillonnage des différents niveaux de *pooling*, il est possible de moduler la résolution temporelle des caractéristiques de la vidéo en faisant varier le nombre de niveaux de la pyramide. Ainsi, plus le nombre de niveaux de la pyramide est élevé, plus les caractéristiques vidéos sont enrichies d'information temporelle à basse résolution. Cela permet d'augmenter leur portée temporelle et d'envisager des prédictions de segments temporels plus précises.

5.3.1.1 Pyramide de *pooling*

Avec $\mathbf{V} = \boldsymbol{x}_1, ..., \boldsymbol{x}_T$ avec T images (ou T portions de vidéo) une vidéo où $\boldsymbol{x}_t$ désigne le vecteur de caractéristiques de taille Γ de la t-ième image (ou t-ième portion de vidéo) à partir d'un réseau d'extraction de caractéristiques de l'état de l'art, le *pooling* réalisé pour enrichir temporellement ces caractéristiques est un *average pooling* adaptatif 1D sur chaque caractéristique, défini comme une fonction de *pooling* (cf Eq.2.8 pour le *max-pooling*) de *stride* $s = \left\lfloor \frac{T}{K} \right\rfloor$ et de noyau de taille $p = T - (K-1) \times s$. Ainsi, cette méthode de *pooling*, adapte ses paramètres pour fournir une sortie de taille égale à K.

Nous définissons ensuite une pyramide à L niveaux (voir Figure 56) par une succession d'*average pooling* avec des *strides* et des noyaux de plus en plus élevés, permettant ainsi de diminuer progressivement la résolution temporelle :

$$\text{Pyramide}_L(\boldsymbol{V}) = \{\text{AdapativeAvgPool}_{K_l}(\boldsymbol{V})\}_{l=1}^{L} \tag{5.6}$$

avec $0 \leq l < L$ et $K_l = K.2^l$ est la taille de la sortie du niveau l de la pyramide.

Pour chaque niveau de la pyramide, un échantillonnage régulier est effectué de manière à retrouver une dimension temporelle égale à K pour tous les niveaux de la pyramide. A noter que grâce à l'*adaptative average pooling*, le signal original n'est pas nécessairement un des niveaux de la pyramide. Les caractéristiques des différents niveaux sont ensuite concaténées de sorte que $\boldsymbol{X}_k^{pool} = \boldsymbol{x}_{k,1}^{pool}, ..., \boldsymbol{x}_{k,L}^{pool}$. En résulte, $\boldsymbol{V}^{pool} = \boldsymbol{X}_1^{pool}, ..., \boldsymbol{X}_K^{pool}$, l'image de la vidéo $\boldsymbol{V}$ par la fonction pyramidale de *pooling*. Chaque pas de temps de la nouvelle résolution (K) est ainsi caractérisé par un vecteur $\boldsymbol{X}_k^{pool}$ de dimension $L * \Gamma$.

Finalement, l'algorithme SALAD utilise les K vecteurs de $\boldsymbol{V}^{pool}$ pour faire les prédictions.

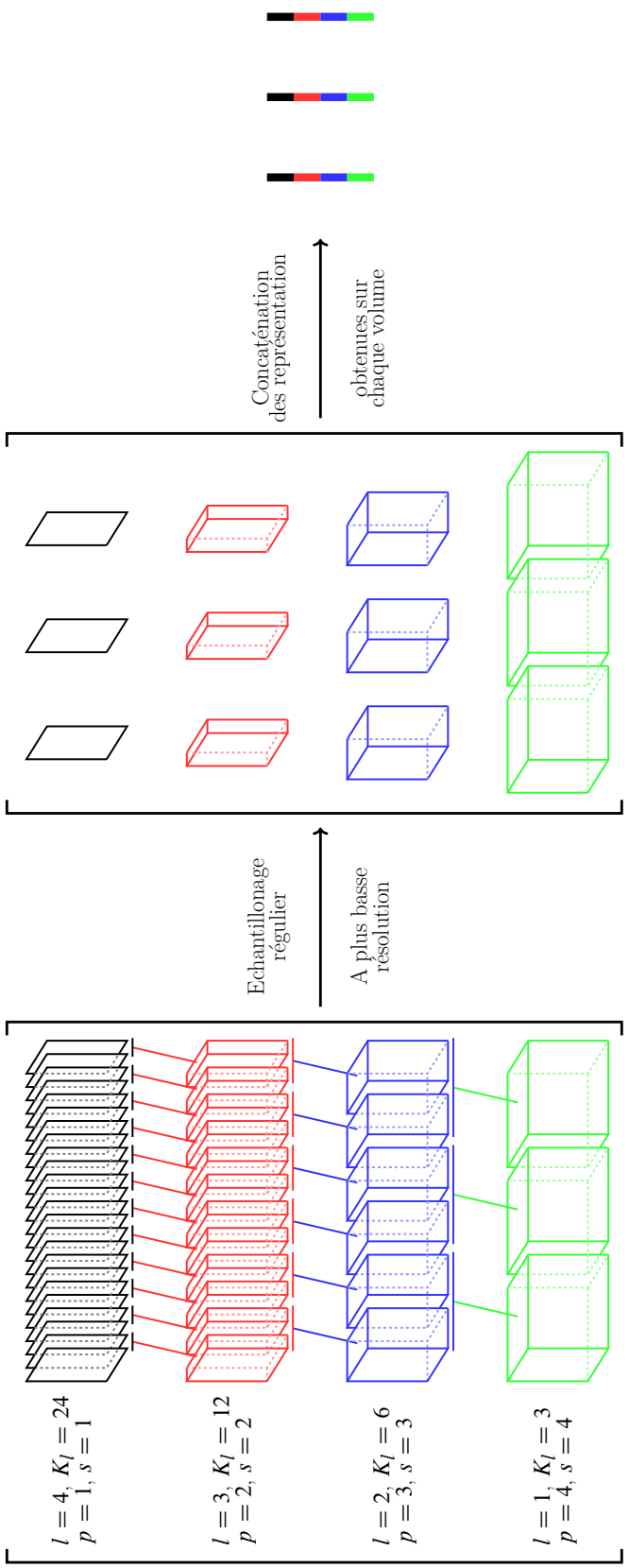


FIGURE 56 – **Pyramide de *pooling* sur les caractéristiques extraites d'une vidéo, avec $L = 4$ et $K = 3$.** Les listes des tailles des noyaux et des *strides* des 4 niveaux de *pooling* sont respectivement $p = [1, 2, 4, 8]$ et $s = [1, 2, 4, 8]$. Un échantillonnage régulier de 3 cartes de *pooling* est réalisé pour chaque niveau de la pyramide. Une concaténation des 4 niveaux en résulte.

mAP@tIoU	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
SALAD	53.0	49.8	46.3	42.4	38.2	33.3	27.4	20.8	13.4	5.5
SALAD PP	**58.2**	**54.9**	**51.5**	**48.1**	**44.3**	**39.1**	**33.1**	**26.2**	**17.7**	**8.8**

TABLE 5.7 – **Résultats de détection sur le jeu de validation d'ActivityNet1.3**, mesurés en (%). Les caractéristiques, issues de la pyramide de *pooling*, et utilisées par SALAD permettent d'améliorer la mAP à toutes les tIoU.

5.3.2 Implémentation et Expériences

5.3.2.1 Détails d'implémentation.

La pyramide de *pooling* est réalisée à partir des mêmes caractéristiques pré-extraites que dans la Section 5.2.5.1. Les détails d'optimisation et le détail de l'architecture du réseau SALAD utilisé sont aussi celles de cette section. Avant toute conception pyramidale, les vidéos sont échantillonnées pour contenir au maximum 100 pas de temps (images ou portions de vidéos).

Au moment de l'évaluation du modèle, quel que soit le nombre de propositions extraites (hyperparamètre K) de la pyramide, la mAP est calculée avec les 20 propositions avec le score le plus élevé par vidéo, sur le jeu de données ActivityNet1.3, comme pour l'algorithme SALAD original (voir Section 4.3.1.

5.3.2.2 Comparaison avec le modèle SALAD

Le Tableau 5.7 présente la mAP du modèle SALAD avec les caractéristiques TSP et une classification parfaite, ainsi que l'utilisation de SALAD sur une pyramide de *pooling* de 20 propositions et 3 niveaux. La Figure 57 présente l'amélioration relative apportée par l'enrichissement temporel des caractéristiques, à différentes tIoU. On constate que cet enrichissement apporte 60% d'amélioration pour une IoU de 0.95, 32% à 0.9 ou encore 26% à 0.85. A faibles tIoU l'apport est aussi significatif puisqu'à la tIoU de 0.5 l'augmentation de la mAP est déjà de 10%. Ainsi, comme le raffinement des propositions réalisé dans la Section précédente, l'enrichissement temporel améliore fortement le modèle SALAD à toutes tIoU, fortes ou faibles.

5.3.2.3 Etudes / Discussions

La pyramide de *pooling* réalisée à partir des caractéristiques dépend du nombre de propositions voulues pour le modèle SALAD, ainsi que du nombre de niveaux de la pyramide. Les scores pour différents nombres de propositions et différents niveaux de pyramides, reflètent un enrichissement temporel des caractéristiques

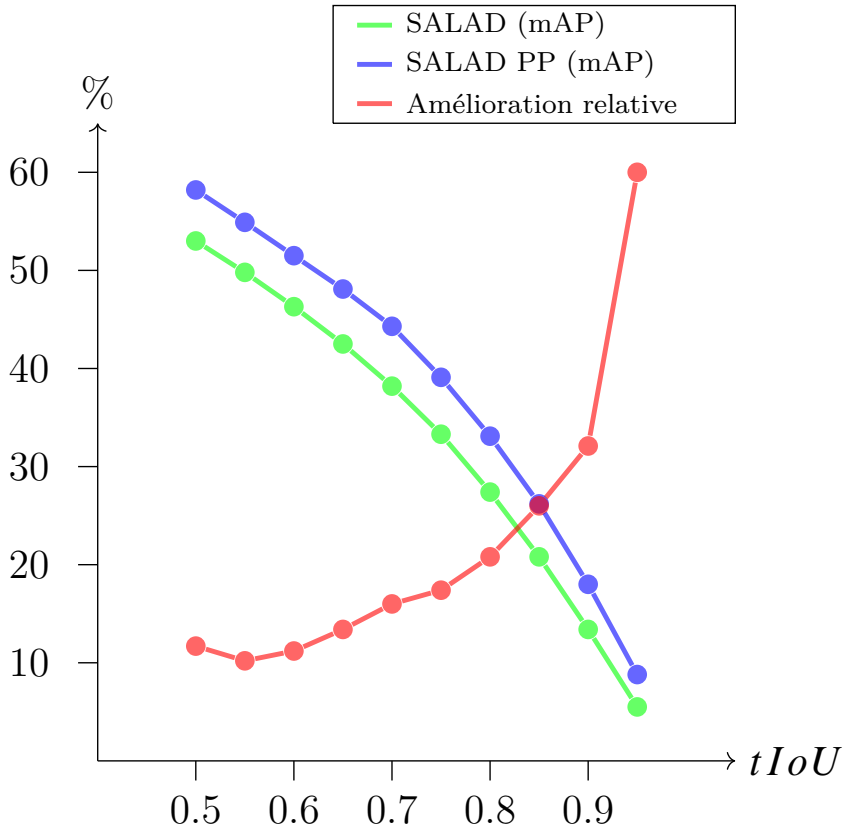


FIGURE 57 – Apport de la pyramide de *pooling* pour le modèle SALAD à différentes tIoU.

de différents niveaux d'échelles et sont visibles dans les Tableaux 5.8, 5.9, 5.10.

A un nombre de propositions K fixé, on compare SALAD appris et évalué en faisant varier le nombre de niveaux de la pyramide.

On constate qu'augmenter le nombre de niveaux permet d'enrichir temporellement les caractéristiques, et ainsi d'améliorer la mAP. Le meilleur modèle pyramidale ayant 3 niveaux et un nombre de propositions de 20. Mais aucune tendance concrète ne ressort des variations de ces hyperparamètres, puisque l'augmentation du nombre de niveaux améliore la mAP jusqu'à ce que le nombre de niveaux atteigne un seuil qui voit décroître ensuite la mAP. C'est la même tendance pour l'augmentation du nombre de propositions, puisque cette même tendance s'observe à hautes tIoU.

Ainsi, l'amélioration de la qualité temporelle des caractéristiques permet d'améliorer la prédiction de segments temporels. Mais elle ne permet toujours pas d'atteindre le niveaux des meilleurs modèles de l'état de l'art à hautes tIoU. La prédiction de segments à hautes tIoU ne dépend donc pas seulement de la qualité des caractéristiques.

mAP @ tIoU pour 10 propositions	0.5	0.75	0.95
10 propositions + 2 niveaux	56.00	37.82	8.61
10 propositions + 3 niveaux	56.40	38.13	7.32
10 propositions + 4 niveaux	56.18	38.69	7.77
10 propositions + 5 niveaux	56.87	38.39	8.33

TABLE 5.8 – **10 propositions avec différents niveaux pyramidaux reflétant différentes résolutions.**

mAP @ tIoU pour 20 propositions	0.5	0.75	0.95
20 propositions + 2 niveaux	57.91	39.55	7.69
20 propositions + 3 niveaux	**58.18**	**39.07**	**8.80**
20 propositions + 4 niveaux	57.77	38.47	4.69
20 propositions + 5 niveaux	57.64	38.44	5.94

TABLE 5.9 – **20 propositions avec différents niveaux pyramidaux reflétant différentes résolutions.**

mAP @ tIoU pour 30 propositions	0.5	0.75	0.95
30 propositions + 2 niveaux	58.23	38.43	6.64
30 propositions + 3 niveaux	58.12	37.8	6.64
30 propositions + 4 niveaux	57.89	37.53	7.45
30 propositions + 5 niveaux	57.34	38.26	4.89

TABLE 5.10 – **30 propositions avec différents niveaux pyramidaux reflétant différentes résolutions.**

5.4 Modification de l'optimisation du modèle SALAD

L'optimisation du modèle SALAD est construite autour d'un hyperparamètre μ, considéré comme la valeur minimum de tIoU qu'un segment prédit doit avoir avec un segment de vérité terrain afin de déclencher le processus d'auto-évaluation. Or dans ce modèle, cette tIoU cible μ est commune à toutes les prédictions, et à toutes les vidéos. De cette manière, la modélisation considère que toutes les prédictions doivent être traitées de la même manière, jusqu'à obtenir cette valeur seuil d'IoU. Or, il est facile de constater que toutes les prédictions ne se valent pas. En effet, sachant que la mAP n'est pas maximale pour un objectif de tIoU minimal (0.1 pour le dataset THUMOS14 et 0.5 pour le dataset ActivityNet), toutes les prédictions à réaliser n'ont pas une difficulté équivalente. Aussi, cette disparité est croissante avec la tIoU cible. Avoir un hyperparamêtre μ fixe pour l'ensemble du processus d'optimisation est peut être contradictoire avec l'objectif d'avoir un modèle optimal pour toutes les tIoU, et encore plus pour les fortes tIoU. Si μ est trop faible, le processus complet d'optimisation sera réalisé sur des prédictions optimales à faible tIoU mais dont les caractéristiques n'auront pas forcément assez d'information pour réaliser des prédictions aussi précises à haute tIoU. Inversement, si μ est un seuil trop élevé, l'objectif de tIoU à atteindre pourra être top important pour une partie des prédictions réalisées et aucune caractéristique ne se détachera durant le processus d'optimisation, le rendant sous optimale.

5.4.1 Curriculum Learning

On constate naturellement que le modèle ne sera pas performant pour les tIoU supérieures à la tIoU cible μ. Aussi, plus la tIoU cible est élevée, plus la mAP moyenne est élevée, mais plus la mAP à faible tIoU diminue. Le fait d'augmenter μ augmente alors la complexité de la tâche à réaliser et l'optimisation à tendance à "sacrifier" une partie de sa mAP à faible tIoU pour augmenter celle à forte tIoU.

Le *curriculum learning* repose sur l'idée d'acquisition progressive du savoir, à l'image de l'être humain. L'idée de base est donc d'apprendre les aspects les plus faciles d'une tâche avant de progressivement augmenter sa complexité. Dans le cas concret de la détection d'actions et plus précisément de l'amélioration du modèle SALAD pour les hautes tIoU, le *curriculum learning* se matérialise comme l'augmentation progressive de la tIoU cible.

Pour cela, on modifie la définition du *status* des variables $(\alpha_{t,n})_{t,n\in[1,T]\times[1,N]}$

Algorithme 3 : Calul de la perte d'auto-évaluation mono-tIoU

Entrées : $\{[\hat{s}_t, \hat{e}_t], \hat{p}_t\}_{t=1}^{T}$ les segments régressés à chaque point d'ancrage t et leurs scores d'auto-évaluation
$\{[s_n, e_n]\}_{n=1}^{N}$ les segments de vérité terrain de la vidéo
μ le seuil de tIoU

Calculer $\sigma(t)$
/* la liste des temps t, triés par ordre décroissant de $\hat{p}_t$ */
/* $p_{\sigma(t+1)} \geq p_{\sigma(t)}$ */
Initialiser $\alpha = \mathbf{0}$, $\beta = \mathbf{0}$, $y = \mathbf{0}$
pour $t = 1, ..., T$ **faire**
  pour $n = 1, ..., N$ **faire**
    si $\beta_n = 0$ **alors**
      si $e_n \leq \sigma(t) \leq s_n$ **alors**
        $\alpha_{\sigma(t),n} \leftarrow 1$
        $\rho \leftarrow tIoU([\hat{s}_{\sigma(t)}, \hat{e}_{\sigma(t)}], [s_n, e_n])$ // Eq.5.1

        si $\rho > \mu$ **alors**
          $\beta_n \leftarrow 1$
          $y_{\sigma(t)} \leftarrow 1$
        sinon
          $\alpha_{\sigma(t),n} \leftarrow 1$
        fin
      sinon
      fin
    sinon
    fin
  fin
fin
Calculer la valeur de la fonction de perte d'auto-évaluation L // en utilisant l'Eq.4.5

Sorties : Perte L

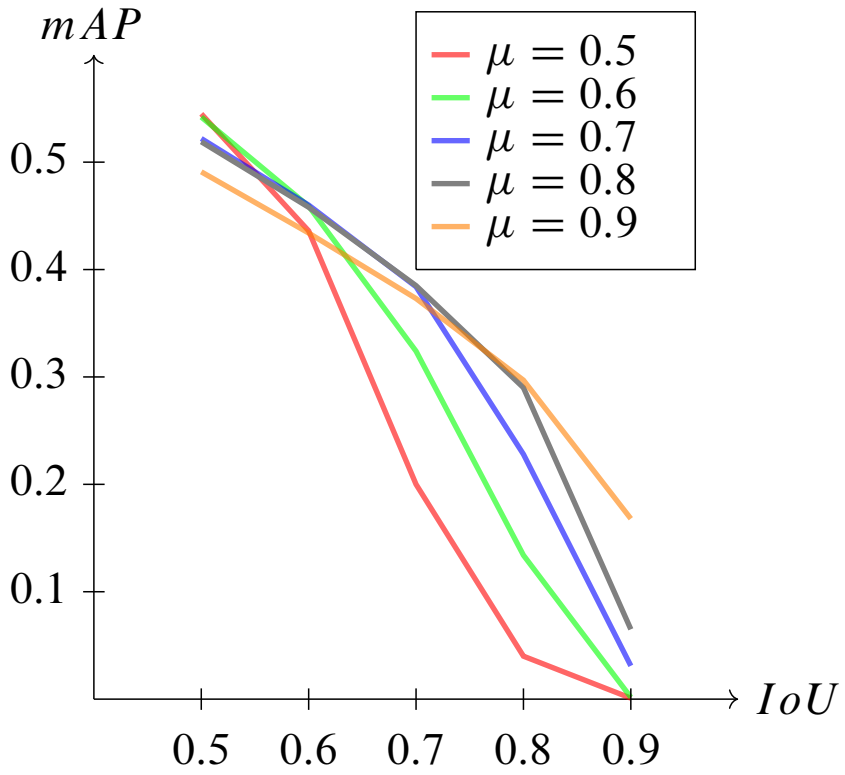


FIGURE 58 – Evolution de la mAP en fonction de l'hyperparamètre μ.

afin de stopper la régression des segments lorsque la tIoU cible est atteinte. La modification réalisée est matérialisée en couleur dans l'Algorithme 3 et définit le modèle SALAD mono-tIou. Ainsi définis, la régression des intervalles s'arrête lorsque pour une vérité terrain fixée, l'intervalle ayant le score d'auto-évaluation le plus élevé, dépasse la tIoU cible. Cette modification va alors permettre de figer l'apprentissage lorsque les intervalles sont correctement prédits à seuil de tIoU fixe. De cette manière il sera possible de faire varier cette tIoU cible et de constater l'effet de cette variation. L'évolution de la mAP en fonction μ est visible sur la Figure 58.

Le *curriculum learning* a déjà été utilisé avec succès pour un grand nombre de tâches (SOVIANY et al., 2021). La difficulté de la méthode, qui la rend limitée pour certaines tâches, est de trouver une manière de caractériser et introduire davantage de difficulté dans l'apprentissage.

Dans notre cas, la fonction de difficulté est explicitement définie par la tIoU cible, et l'augmentation progressive de cette fonction pourrait donc permettre d'améliorer la détection globale et notamment à hautes tIoU sans dégrader les performances à faibles tIoU.

Ainsi, on fixe une liste croissante de I seuils de difficultés $M = (\mu_i)_{i=0}^{I}$ et

on définit θ_{μ_i} les poids optimisés du modèle SALAD avec un seuil de difficulté μ_i. L'optimisation par *curriculum learning* consiste alors à optimiser le modèle SALAD en itérant sur les différents niveaux de difficulté $\mu_i \in M$ en conservant les poids optimisés pour le niveau de difficulté précédent. On a alors l'Algorithme 4 :

Algorithme 4 : *Curriculum learning* pour l'optimisation de SALAD

```
M = (μ_i)_{i=0}^{I} ;
On cherche θ_{μ_0} les poids de SALAD initialisés aléatoirement et
  qui maximisent mAP@θ_{μ_0};
pour i ∈ [1, ..., I] faire
    On cherche θ_{μ_i} les poids de SALAD initialisés en θ_{i-1} et qui
     maximisent mAP@θ_{μ_i};
fin
return θ_I;
```

5.4.2 Implémentation et expériences

5.4.2.1 Détails d'implémentation

Le *curriculum learning* est évalué sur la base de données ActivityNet1.3. Le détail de l'architecture neuronale et les détails d'optimisation sont les mêmes que dans la Section 5.2.5.1. Par contre, la liste de seuils de difficultés est ici $M = [0.5, 0.6, 0.7, 0.8, 0.9]$, qui font partie des niveaux de tIou utilisés pour le calcul de la mAP.

5.4.2.2 Comparaison avec le modèle sans *curriculum learning*

Dans le Tableau 5.11, on retrouve les résultats de l'apprentissage par *curriculum learning* comparés avec le même apprentissage sans *curriculum learning*, c'est à dire sans pré-apprentissage seuil de tIoU μ inférieur et SALAD. On constate que l'apprentissage par *curriculum learning* n'est pas plus efficace que l'apprentissage mono-tIoU puisque la mAP évaluée pour une tIoU égale à μ est systématiquement inférieure lors d'un pré-apprentissage à niveau μ inférieur.

Par contre l'apprentissage en mono-tIoU permet d'améliorer systématiquement la mAP pour le niveau de tIoU égal au seuil μ. Il est alors légitime de se demander si les méthodes de détection de l'état de l'art cherchant à maximiser le score de mAP, en moyenne, sur différents seuils de tIoU ont réellement un sens. En effet, on constate que lorsqu'on essaye d'obtenir un modèle performant à faibles tIoU, cela introduit des problèmes à fortes tIoU. Et inversement, lorsqu'on cherche la performance à hautes tIoU, les modèles se dégradent à faibles tIoU.

mAP @ tIoU	0.5	0.6	0.7	0.8	0.9
SALAD	53.0	46.3	38.2	27.4	13.4
$\mu = 0.5$	54.5	43.6	20.0	4.1	0.1
Curriculum	54.3	35.7	9.2	1.9	0.2
$\mu = 0.6$	54.2	45.9	32.4	13.4	0.2
Curriculum	53.6	45.2	23.4	4.3	0.5
$\mu = 0.7$	51.9	46.4	38.4	22.8	3.1
Curriculum	51.7	45.3	36.6	16.1	1.8
$\mu = 0.8$	51.9	45.8	38.5	29.1	6.5
Curriculum	48.4	42.2	36.1	27.1	7.8
$\mu = 0.9$	49.1	43.4	37.7	29.7	16.8
Curriculum	45.5	40.0	34.6	27.9	17.0

TABLE 5.11 – **Optimisation par *curriculum learning* pour différents seuil d'auto-évaluation.**

5.4.2.3 Discussions

Nous avons mis en lumière que la méthode de *curriculum learning* présentée n'était pas adaptée pour améliorer les modèles mono-tIoU. Par contre, l'apprentissage mono-tIoU a permis de disposer de modèles plus performants pour un seuil donné, remettant en question l'importance de réaliser des détecteurs multi-tIoU.

Mais, quand on se penche précisément sur la vérité terrain des différents jeux de données de l'état de l'art, et plus particulièrement sur le jeu de données ActivityNet, on constate que les bords des actions contenus dans la vérité terrain sont très bruités. La sélection de bords temporels apparaît beaucoup plus ambiguë que la sélection de boîtes englobantes en image, comme en détection d'objets. Chercher à détecter des actions dans des vidéos à fortes tIoU n'est donc pas forcément un problème bien posé.

5.5 Conclusion

Dans ce chapitre, nous avons montré que le raffinement des propositions issues du modèles SALAD pouvait être réalisé par une hybridation du modèle à l'aide d'une architecture ascendante. L'utilisation de caractéristiques très locales permet alors d'obtenir des bords plus fins. Mais, l'amélioration effectivement apportée est très loin de l'amélioration théorique possible, puisque le gain en mAP ne représente que 30% du gain optimal. Cela ne permet donc pas de conclure que les bords des actions sont assez saillants pour que l'utilisation de caractéristiques

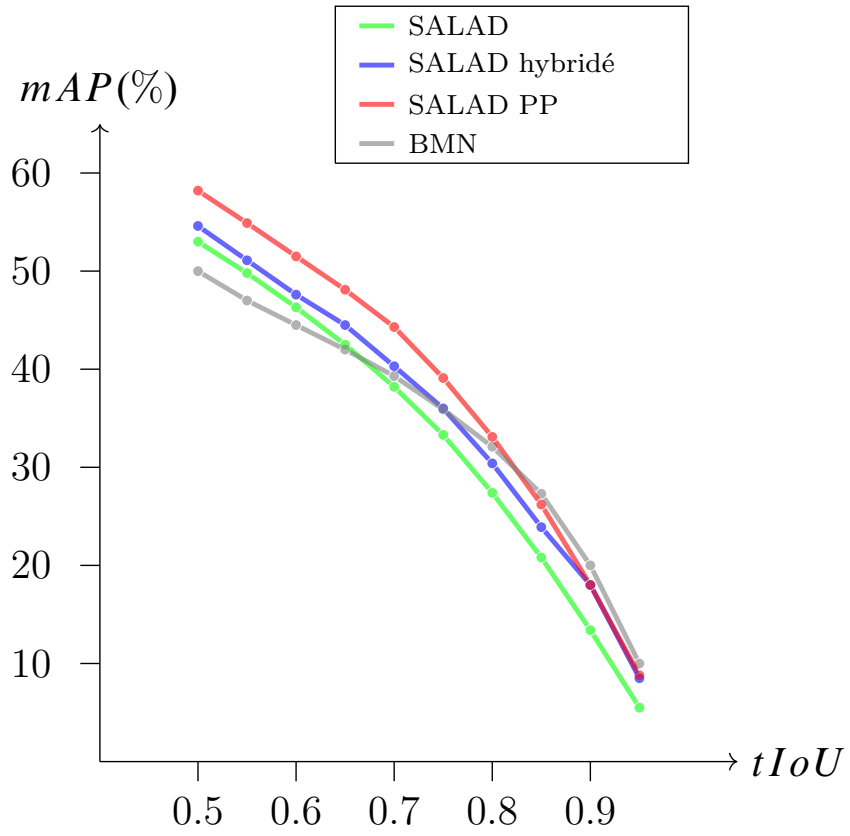


FIGURE 59 – **Comparaison des méthodes de raffinement proposées avec l'état de l'art**.

locales soit suffisante pour réaliser avec une grande précision la tâche de détection à fortes tIoU.

Ensuite, nous avons essayé d'enrichir temporellement les caractéristiques en réalisant une pyramide de *pooling*. Mais, même si l'apport a été significatif, il ne permet pas d'améliorer assez le modèle à hautes tIoU pour concurrencer les meilleurs modèles de l'état de l'art. La prédiction de segments précis à fortes tIoU ne dépend donc pas seulement de la qualité des caractéristiques.

Ainsi, même si ces travaux sur les caractéristiques ont apporté des améliorations, il n'en est pas ressorti de méthode générale permettant de réaliser un détecteur fiable à hautes tIoU (voir Figure 59).

Aussi, nous avons montré que l'apprentissage et l'utilisation de modèles mono-tIoU était plus performant que l'apprentissage multi-tIoU, remettant en question leur pertinence. D'autant plus que la recherche de hautes tIoU sur les jeux de données de l'état de l'art apparait comme un modèle artificiel au vu du bruit dans la définition des bords.

Finalement, pour chercher à réaliser des détecteurs précis à hautes tIoU, peut être faudrait il concevoir des modèles à apprentissage *end-to-end* sans décimation

temporelle. Mais face au volume des données vidéos, la puissance de calcul est encore limitante pour ce genre d'approches.

Chapitre 6

Conclusion et perspectives

Sommaire

Dans ce manuscrit nous avons abordé la thématique de la compréhension d'actions dans des vidéos en étudiant l'impact que peut avoir la localisation temporelle fine sur la robustesse de l'extraction sémantique.

Pour permettre une extraction sémantique robuste sans contrainte forte de localisation temporelle, nous avons présenté dans le chapitre 3 un algorithme permettant de réduire la sensibilité aux annotations lors de l'apprentissage et ainsi d'améliorer la qualité de l'extraction sémantique.

Afin d'obtenir également des indications sur la localisation temporelle des actions nous avons étendu cette méthode dans le chapitre 4.

Enfin, pour arriver à une localisation temporelle précise des actions, nous avons proposé d'étudier les limites de ces approches et proposé des pistes d'amélioration dans le chapitre 5.

Nous présentons maintenant un résumé de nos contributions et de leurs résultats et concluons ce manuscrit avec des perspectives pour d'éventuels travaux futurs.

6.1 Résumé des contributions

6.1.1 Repérage d'actions

L'objectif du chapitre a été de réaliser une extraction sémantique robuste. Nous avons proposé un script d'évaluation permettant de quantifier la qualité sémantique des extractions réalisées par les algorithmes de repérage d'actions. Ensuite, nous avons introduit *ActionSpotter*, une méthode de repérage d'actions s'appuyant sur une méthode d'apprentissage par renforcement, capable de s'ajuster aux spécificités de la tâche, tout en simplifiant l'étiquetage des vidéos jusque là nécessaire. Nous avons validé empiriquement l'intérêt d'*ActionSpotter*, et plus précisément des techniques de repérage d'actions, pour l'extraction de contenu sémantique dans les vidéos.

6.1.2 Détection d'actions

Afin de tirer profit de la robustesse du repérage d'*ActionSpotter*, nous avons introduit dans ce chapitre la notion de confiance temporelle afin d'étendre cette méthode de repérage à la détection d'actions, en couplant la régression de propositions temporelles d'actions et leur tri. A travers un ensemble d'expériences, nous avons montré que ce couplage, appelé *SALAD*, permet de réaliser une détection supérieure à l'état de l'art pour des seuils faibles de tIoU.

6.1.3 Affinement des détections

Nous avons proposé d'étudier dans ce chapitre les limites des approches *ActionSpotter* et *SALAD* et avons proposé des pistes afin d'arriver à une localisation temporelle fine des actions. Pour cela, nous avons réalisé 3 améliorations du modèle *SALAD* permettant d'affiner les détections. Nous avons proposé une technique d'hybridation permettant d'affiner *a posteriori* les bords des prédictions en utilisant des caractéristiques spatio-temporelles locales qui a eu l'effet escompté sans parvenir à dépasser l'état de l'art. Nous avons ensuite proposé d'enrichir temporellement l'extraction de caractéristiques en réalisant une pyramide de *pooling* vidéo. Augmenter la capacité temporelle du modèle s'est aussi révélé prometteur. Nous avons finalement utilisé une méthode de *curriculum learning* permettant d'augmenter progressivement la complexité de la tâche de détection à l'apprentissage. Cette modification d'apprentissage n'a pas permis d'améliorer la capacité du détecteur à hautes tIoU mais a mis en lumière les limites d'un apprentissage multi-seuils de tIoU.

6.1.4 Publications

Ces travaux ont donné lieu à deux publications en conférences internationales (ICPR et WACV) ainsi qu'à un programme de maturation en vu d'un transfert technologique (BLAST, SATT).

6.2 Perspectives

Un des problèmes majeurs du traitement vidéo est le volume des données vidéo. Typiquement, le *streaming* vidéo représentait 61% du trafic internet mondial en 2019 [1]. Un autre exemple étant que le million d'images d'Imagenet représente un volume de données comparable aux 200 vidéos du jeu de données THUMOS14. De ce fait, la communauté a tendance à s'adapter à cette contrainte de volume, plutôt que de développer des méthodes qui seraient plus cohérentes.

L'exemple typique est le sous-échantillonnage des vidéos à 100 images sur le jeu de données ActivityNet qui est un non sens d'un point de vue des données. Les méthodes développées dans cette thèse gagneraient très probablement à travailler plus proche de la donnée d'origine. Mais il faudrait pour cela une puissance de calcul qui ne semble pas être à la disposition de laboratoires académiques. Notamment, les problèmes de localisation rencontrés à forte tIoU sont peut être principalement dûs à ces adaptations pour gérer de volumétrie des vidéos.

Un autre grand problème pour traiter le problème de détection d'actions à hautes tIoU est l'ambiguïté des débuts/fins d'actions, en particulier pour les actions longues et reflétant la vie courante. En observant les vidéos d'ActivityNet, on peut observer beaucoup de bruit dans les annotations. Par exemple, pour la classe "sauter", l'action contient parfois une phase de course, parfois non. Certains jeux de données proposent de moyenner les annotations de différents annotateurs au vu de ce problème (*crowdsourced annotations*). Cela montre que le début et la fin de l'action ne sont pas saillants dans le signal. Ce qui encore une fois remet en cause le concept même de détection à haute tIoU.

1. Source : Sandvine | The Global Internet Phenomena Report

Bibliographie

Alwassel, Humam et al. (2018). « Action search : Spotting actions in videos and its application to temporal action localization ». In : *Proceedings of the European Conference on Computer Vision (ECCV)*, p. 251–266.

Alwassel, Humam et al. (2020). « Tsp : Temporally-sensitive pretraining of video encoders for localization tasks ». In : *arXiv preprint arXiv :2011.11479*.

Barlow, Horace B (1989). « Unsupervised learning ». In : *Neural computation* 1.3, p. 295–311.

Buch, Shyamal et al. (2017). « Sst : Single-stream temporal action proposals ». In : *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, p. 2911–2920.

Buch, Shyamal et al. (2019). « End-to-end, single-stream temporal action detection in untrimmed videos ». In : *Procedings of the British Machine Vision Conference 2017*. British Machine Vision Association.

Buzzelli, Marco et al. (2020). « A vision-based system for monitoring elderly people at home ». In : *Applied Sciences* 10.1, p. 374.

Caba Heilbron, Fabian et al. (2015). « Activitynet : A large-scale video benchmark for human activity understanding ». In : *Proceedings of the ieee conference on computer vision and pattern recognition*, p. 961–970.

Caesar, Holger et al. (2020). « nuscenes : A multimodal dataset for autonomous driving ». In : *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, p. 11621–11631.

Carreira, Joao et al. (2017). « Quo vadis, action recognition ? a new model and the kinetics dataset ». In : *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 6299–6308.

Chao, Yu-Wei et al. (2018). « Rethinking the faster r-cnn architecture for temporal action localization ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 1130–1139.

Chen, Yunpeng et al. (2018). « Multi-fiber networks for video recognition ». In : *Proceedings of the european conference on computer vision (ECCV)*, p. 352–367.

Choi, Sungil et al. (2018). « Learning descriptor, confidence, and depth estimation in multi-view stereo ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, p. 276–282.

Chung, Junyoung et al. (2014). « Empirical evaluation of gated recurrent neural networks on sequence modeling ». In : *arXiv preprint arXiv :1412.3555*.

Crasto, Nieves et al. (2019). « Mars : Motion-augmented rgb stream for action recognition ». In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 7882–7891.

Dai, Xiyang et al. (2017). « Temporal context network for activity localization in videos ». In : *Proceedings of the IEEE International Conference on Computer Vision*, p. 5793–5802.

Deng, Jia et al. (2009). « Imagenet : A large-scale hierarchical image database ». In : *2009 IEEE conference on computer vision and pattern recognition*. Ieee, p. 248–255.

Diba, Ali et al. (2019). « Holistic large scale video understanding ». In : *arXiv preprint arXiv :1904.11451* 38, p. 39.

Donahue, Jeffrey et al. (2015). « Long-term recurrent convolutional networks for visual recognition and description ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 2625–2634.

Escorcia, Victor et al. (2016). « Daps : Deep action proposals for action understanding ». In : *European Conference on Computer Vision*. Springer, p. 768–784.

Everingham, Mark et al. (2015). « The pascal visual object classes challenge : A retrospective ». In : *International journal of computer vision* 111.1, p. 98–136.

Farnebäck, Gunnar (2003). « Two-frame motion estimation based on polynomial expansion ». In : *Scandinavian conference on Image analysis*. Springer, p. 363–370.

Feichtenhofer, Christoph (2020). « X3d : Expanding architectures for efficient video recognition ». In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 203–213.

Feichtenhofer, Christoph et al. (2016). « Convolutional two-stream network fusion for video action recognition ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 1933–1941.

Feichtenhofer, Christoph et al. (2019). « Slowfast networks for video recognition ». In : *Proceedings of the IEEE/CVF international conference on computer vision*, p. 6202–6211.

Friedman, Jerome et al. (2001). *The elements of statistical learning*. T. 1. 10. Springer series in statistics New York.

Gao, Jiyang et al. (2017a). « Cascaded boundary regression for temporal action detection ». In : *arXiv preprint arXiv :1705.01180*.

Gao, Jiyang et al. (2017b). « Turn tap : Temporal unit regression network for temporal action proposals ». In : *Proceedings of the IEEE international conference on computer vision*, p. 3628–3636.

Gao, Jiyang et al. (2018). « Ctap : Complementary temporal action proposal generation ». In : *Proceedings of the European conference on computer vision (ECCV)*, p. 68–83.

Grigorescu, Sorin et al. (2020). « A survey of deep learning techniques for autonomous driving ». In : *Journal of Field Robotics* 37.3, p. 362–386.

Gu, Chunhui et al. (2018). « Ava : A video dataset of spatio-temporally localized atomic visual actions ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 6047–6056.

Haarnoja, Tuomas et al. (2018). « Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor ». In : *International conference on machine learning*. PMLR, p. 1861–1870.

Hafiz, Abdul Mueed et al. (2020). « A survey on instance segmentation : state of the art ». In : *International journal of multimedia information retrieval*, p. 1–19.

Hara, Kensho et al. (2017). « Learning spatio-temporal features with 3d residual networks for action recognition ». In : *Proceedings of the IEEE International Conference on Computer Vision Workshops*, p. 3154–3160.

— (2018). « Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet ? » In : *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, p. 6546–6555.

He, Dongliang et al. (2019). « Stnet : Local and global spatial-temporal modeling for action recognition ». In : *Proceedings of the AAAI Conference on Artificial Intelligence*. T. 33. 01, p. 8401–8408.

He, Jiang et al. (2019). « Bi-direction feature pyramid temporal action detection network ». In : *Asian Conference on Pattern Recognition*. Springer, p. 889–901.

He, Kaiming et al. (2016). « Deep residual learning for image recognition ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 770–778.

Heilbron, Fabian Caba et al. (2016). « Fast temporal activity proposals for efficient detection of human actions in untrimmed videos ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 1914–1923.

Heilbron, Fabian Caba et al. (2017). « Scc : Semantic context cascade for efficient action detection ». In : *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, p. 3175–3184.

Hochreiter, Sepp et al. (1997). « Long short-term memory ». In : *Neural computation* 9.8, p. 1735–1780.

Horn, Berthold KP et al. (1981). « Determining optical flow ». In : *Artificial intelligence* 17.1-3, p. 185–203.

Hou, Rui et al. (2017). « Real-Time Temporal Action Localization in Untrimmed Videos by Sub-Action Discovery. » In : *BMVC*. T. 2, p. 7.

Huang, Yupan et al. (2019). « Decoupling localization and classification in single shot temporal action detection ». In : *2019 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, p. 1288–1293.

Hutchinson, Matthew et al. (2020). « Video Action Understanding : A Tutorial ». In : *arXiv preprint arXiv :2010.06647*.

Idrees, Haroon et al. (2017). « The THUMOS challenge on action recognition for videos "in the wild" ». In : *Computer Vision and Image Understanding* 155, p. 1–23.

Jaccard, Paul (1901). « Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines ». In : *Bull Soc Vaudoise Sci Nat* 37, p. 241–272.

Ji, Shuiwang et al. (2012). « 3D convolutional neural networks for human action recognition ». In : *IEEE transactions on pattern analysis and machine intelligence* 35.1, p. 221–231.

Jiang, Boyuan et al. (2019). « Stm : Spatiotemporal and motion encoding for action recognition ». In : *Proceedings of the IEEE/CVF International Conference on Computer Vision*, p. 2000–2009.

Joefrie, Yuri Yudhaswana et al. (2019). « Action recognition by composite deep learning architecture I3D-DenseLSTM ». In : *2019 International Conference of Advanced Informatics : Concepts, Theory and Applications (ICAICTA)*. IEEE, p. 1–6.

Karpathy, Andrej et al. (2014). « Large-scale video classification with convolutional neural networks ». In : *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, p. 1725–1732.

Kay, Will et al. (2017). « The kinetics human action video dataset ». In : *arXiv preprint arXiv :1705.06950*.

Kingma, Diederik P et al. (2014). « Adam : A method for stochastic optimization ». In : *arXiv preprint arXiv :1412.6980*.

Kipf, Thomas N et al. (2016). « Semi-supervised classification with graph convolutional networks ». In : *arXiv preprint arXiv :1609.02907*.

Konda, Vijay R et al. (2000). « Actor-critic algorithms ». In : *Advances in neural information processing systems*, p. 1008–1014.

Krizhevsky, Alex et al. (2012). « Imagenet classification with deep convolutional neural networks ». In : *Advances in neural information processing systems* 25, p. 1097–1105.

Laud, Adam Daniel (2004). *Theory and application of reward shaping in reinforcement learning*. University of Illinois at Urbana-Champaign.

LeCun, Yann et al. (1995). « Convolutional networks for images, speech, and time series ». In : *The handbook of brain theory and neural networks* 3361.10, p. 1995.

LeCun, Yann et al. (2015). « Deep learning ». In : *nature* 521.7553, p. 436–444.

Li, Jun et al. (2020). « Spatio-temporal deformable 3d convnets with attention for action recognition ». In : *Pattern Recognition* 98, p. 107037.

Li, Luxuan et al. (2019). « Deep point-wise prediction for action temporal proposal ». In : *International conference on neural information processing*. Springer, p. 475–487.

Lin, Ji et al. (2019). « Tsm : Temporal shift module for efficient video understanding ». In : *Proceedings of the IEEE/CVF International Conference on Computer Vision*, p. 7083–7093.

Lin, Tianwei et al. (2017). « Single shot temporal action detection ». In : *Proceedings of the 25th ACM international conference on Multimedia*, p. 988–996.

Lin, Tianwei et al. (2018). « Bsn : Boundary sensitive network for temporal action proposal generation ». In : *Proceedings of the European Conference on Computer Vision (ECCV)*, p. 3–19.

Lin, Tianwei et al. (2019). « Bmn : Boundary-matching network for temporal action proposal generation ». In : *Proceedings of the IEEE/CVF International Conference on Computer Vision*, p. 3889–3898.

Lin, Tsung-Yi et al. (2014). « Microsoft coco : Common objects in context ». In : *European conference on computer vision*. Springer, p. 740–755.

Liu, Kun et al. (2018). « T-C3D : Temporal convolutional 3D network for real-time action recognition ». In : *Proceedings of the AAAI conference on artificial intelligence*. T. 32. 1.

Liu, Yuan et al. (2019). « Multi-granularity generator for temporal action proposal ». In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 3604–3613.

Long, Fuchen et al. (2019). « Gaussian temporal awareness networks for action localization ». In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 344–353.

Lucas, Bruce D et al. (1981). « An iterative image registration technique with an application to stereo vision ». In : Vancouver, British Columbia.

Ma, Shugao et al. (2021). « Pixel Codec Avatars ». In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 64–73.

Ma, Ying et al. (2019). « A taxonomy for neural memory networks ». In : *IEEE transactions on neural networks and learning systems* 31.6, p. 1780–1793.

Medsker, Larry R et al. (2001). « Recurrent neural networks ». In : *Design and Applications* 5, p. 64–67.

Mingqiang, Yang et al. (2008). « A survey of shape feature extraction techniques ». In : *Pattern recognition* 15.7, p. 43–90.

Mnih, Volodymyr et al. (2013). « Playing atari with deep reinforcement learning ». In : *arXiv preprint arXiv :1312.5602*.

Mnih, Volodymyr et al. (2016). « Asynchronous methods for deep reinforcement learning ». In : *International conference on machine learning*. PMLR, p. 1928–1937.

Mogadala, Aditya et al. (2019). « Trends in integration of vision and language research : A survey of tasks, datasets, and methods ». In : *arXiv preprint arXiv :1907.09358*.

Murray, Naila et al. (2014). « Generalized max pooling ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 2473–2480.

Narayan, Sanath et al. (2019). « 3c-net : Category count and center loss for weakly-supervised action localization ». In : *Proceedings of the IEEE/CVF International Conference on Computer Vision*, p. 8679–8687.

Oneata, Dan et al. (2014). « The lear submission at thumos 2014 ». In :

OpenAI (2020). *MS Windows NT Kernel Description*. URL : https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html (visité le 05/09/2021).

Özyer, Tansel et al. (2021). « Human action recognition approaches with video datasets—A survey ». In : *Knowledge-Based Systems* 222, p. 106995.

Pal, Sankar K et al. (1992). « Multilayer perceptron, fuzzy sets, classifiaction ». In :

Paul, Sujoy et al. (2018). « W-talc : Weakly-supervised temporal activity localization and classification ». In : *Proceedings of the European Conference on Computer Vision (ECCV)*, p. 563–579.

Pérez, Javier Sánchez et al. (2013). « TV-L1 optical flow estimation ». In : *Image Processing On Line* 2013, p. 137–150.

Podlesnaya, Anna et al. (2016). « Deep learning based semantic video indexing and retrieval ». In : *Proceedings of SAI intelligent systems conference*. Springer, p. 359–372.

Qiu, Zhaofan et al. (2017). « Learning spatio-temporal representation with pseudo-3d residual networks ». In : *proceedings of the IEEE International Conference on Computer Vision*, p. 5533–5541.

Rezazadegan, Fahimeh et al. (2017). « Action recognition : From static datasets to moving robots ». In : *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, p. 3185–3191.

Richard, Alexander et al. (2016). « Temporal action detection using a statistical language model ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 3131–3140.

Rosenblatt, Frank (1958). « The perceptron : a probabilistic model for information storage and organization in the brain. » In : *Psychological review* 65.6, p. 386.

Scherer, Dominik et al. (2010). « Evaluation of pooling operations in convolutional architectures for object recognition ». In : *International conference on artificial neural networks*. Springer, p. 92–101.

Schulman, John et al. (2015). « High-dimensional continuous control using generalized advantage estimation ». In : *arXiv preprint arXiv :1506.02438*.

Schuster, Mike et al. (1997). « Bidirectional recurrent neural networks ». In : *IEEE transactions on Signal Processing* 45.11, p. 2673–2681.

Sevilla-Lara, Laura et al. (2018). « On the integration of optical flow and action recognition ». In : *German Conference on Pattern Recognition*. Springer, p. 281–297.

Shou, Zheng et al. (2016). « Temporal action localization in untrimmed videos via multi-stage cnns ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 1049–1058.

Shou, Zheng et al. (2017). « Cdc : Convolutional-de-convolutional networks for precise temporal action localization in untrimmed videos ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 5734–5743.

Simonyan, Karen et al. (2014). « Two-stream convolutional networks for action recognition in videos ». In : *arXiv preprint arXiv :1406.2199*.

Singh, Amanpreet et al. (2016). « A review of supervised machine learning algorithms ». In : *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. Ieee, p. 1310–1315.

Singh, Gurkirt et al. (2016). « Untrimmed video classification for activity detection : submission to activitynet challenge ». In : *arXiv preprint arXiv :1607.01979*.

Soviany, Petru et al. (2021). « Curriculum learning : A survey ». In : *arXiv preprint arXiv :2101.10382*.

Sreenu, G et al. (2019). « Intelligent video surveillance : a review through deep learning techniques for crowd analysis ». In : *Journal of Big Data* 6.1, p. 1–27.

Strisciuglio, Nicola et al. (2020). « Enhanced robustness of convolutional networks with a push–pull inhibition layer ». In : *Neural Computing and Applications* 32.24, p. 17957–17971.

Stroud, Jonathan et al. (2020). « D3d : Distilled 3d networks for video action recognition ». In : *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, p. 625–634.

Sutton, Richard S et al. (2018). *Reinforcement learning : An introduction*. MIT press.

Tran, Du et al. (2015). « Learning spatiotemporal features with 3d convolutional networks ». In : *Proceedings of the IEEE international conference on computer vision*, p. 4489–4497.

Tran, Du et al. (2019). « Video classification with channel-separated convolutional networks ». In : *Proceedings of the IEEE/CVF International Conference on Computer Vision*, p. 5552–5561.

Turing, Alan M (2009). « Computing machinery and intelligence ». In : *Parsing the turing test*. Springer, p. 23–65.

Vaudaux-Ruth, Guillaume et al. (2021a). « ActionSpotter : Deep Reinforcement Learning Framework for Temporal Action Spotting in Videos ». In : *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, p. 631–638.

— (2021b). « SALAD : Self-Assessment Learning for Action Detection ». In : *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, p. 1269–1278.

Wang, Limin et al. (2014). « Action recognition and detection by combining motion and appearance features ». In : *THUMOS14 Action Recognition Challenge* 1.2, p. 2.

Wang, Limin et al. (2016). « Temporal segment networks : Towards good practices for deep action recognition ». In : *European conference on computer vision*. Springer, p. 20–36.

Wang, Limin et al. (2018a). « Appearance-and-relation networks for video classification ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 1430–1439.

Wang, Limin et al. (2018b). « Temporal segment networks for action recognition in videos ». In : *IEEE transactions on pattern analysis and machine intelligence* 41.11, p. 2740–2755.

Wang, Xianyuan et al. (2019). « I3d-lstm : A new model for human action recognition ». In : *IOP Conference Series : Materials Science and Engineering*. T. 569. 3. IOP Publishing, p. 032035.

Wang, Xiaolong et al. (2018). « Non-local neural networks ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 7794–7803.

Wu, Chao-Yuan et al. (2019). « Long-term feature banks for detailed video understanding ». In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 284–293.

Wu, Wenhao et al. (2019). « Multi-agent reinforcement learning based frame sampling for effective untrimmed video recognition ». In : *Proceedings of the IEEE/CVF International Conference on Computer Vision*, p. 6222–6231.

Wu, Xiongwei et al. (2020). « Recent advances in deep learning for object detection ». In : *Neurocomputing* 396, p. 39–64.

Wu, Zuxuan et al. (2019). « Adaframe : Adaptive frame selection for fast video recognition ». In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 1278–1287.

Xia, Huifen et al. (2020). « A Survey on Temporal Action Localization ». In : *IEEE Access* 8, p. 70477–70487.

Xu, Huijuan et al. (2017). « R-c3d : Region convolutional 3d network for temporal activity detection ». In : *Proceedings of the IEEE international conference on computer vision*, p. 5783–5792.

Xu, Huijuan et al. (2019). « Joint event detection and description in continuous video streams ». In : *2019 IEEE winter conference on applications of computer vision (WACV)*. IEEE, p. 396–405.

Xu, Mengmeng et al. (2020). « G-tad : Sub-graph localization for temporal action detection ». In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, p. 10156–10165.

Yegnanarayana, Bayya (2009). *Artificial neural networks*. PHI Learning Pvt. Ltd.

Yeung, Serena et al. (2016). « End-to-end learning of action detection from frame glimpses in videos ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 2678–2687.

Yuan, Jun et al. (2016). « Temporal action localization with pyramid of score distribution features ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 3093–3102.

Yuan, Zehuan et al. (2017). « Temporal action localization by structured maximal sums ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 3684–3692.

Yue-Hei Ng, Joe et al. (2015). « Beyond short snippets : Deep networks for video classification ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 4694–4702.

Yun, Sukmin et al. (2020). « Regularizing class-wise predictions via self-knowledge distillation ». In : *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, p. 13876–13885.

Yurtsever, Ekim et al. (2020). « A survey of autonomous driving : Common practices and emerging technologies ». In : *IEEE access* 8, p. 58443–58469.

Zach, Christopher et al. (2007). « A duality based approach for realtime tv-l 1 optical flow ». In : *Joint pattern recognition symposium*. Springer, p. 214–223.

Zeng, Runhao et al. (2019). « Graph convolutional networks for temporal action localization ». In : *Proceedings of the IEEE/CVF International Conference on Computer Vision*, p. 7094–7103.

Zhao, Yue et al. (2017). « Temporal action detection with structured segment networks ». In : *Proceedings of the IEEE International Conference on Computer Vision*, p. 2914–2923.

Zhou, Bolei et al. (2018). « Temporal relational reasoning in videos ». In : *Proceedings of the European Conference on Computer Vision (ECCV)*, p. 803–818.

Zhu, Xiaojin Jerry (2005). « Semi-supervised learning literature survey ». In :

Zhu, Yi et al. (2020). « A comprehensive study of deep video action recognition ». In : *arXiv preprint arXiv :2012.06567*.

Zolfaghari, Mohammadreza et al. (2018). « Eco : Efficient convolutional network for online video understanding ». In : *Proceedings of the European conference on computer vision (ECCV)*, p. 695–712.