Université Paris 8

Mémoire

Détection de replays dans les vidéos

Billal Boudjoghra

Tuteur à l'université:
M. Larbi Boubchir
Tuteur d'alternance:
M. Régis Jean-Gilles

Table des Matières

1 Sportagraph

Sportagraph est une start-up filiale de Everspeed, fondée en 2013 et lancée en 2016 après trois années de recherche et développement (R&D).
Son produit est un DAM (Digital Asset Manager) spécialisé dans les événements sportifs.

Le DAM permet aux utilisateurs de stocker et de gérer plusieurs types de fichiers (principalement des images, mais les vidéos et d'autres types sont aussi supportés).
L'équipe travaillant sur ce projet est composée de :

- Alex Macris : Chief Experience Officer
- Edouard Binchet : Chief Strategy and Operation Officer
- Régis Jean-Gilles : Lead Server Developer (mon tuteur d'alternance)
- Benoit Hozjan : SaaS operation manager
- Francois Jacquier : Lead Developer Front
- Khedidja Almherabet : Operation manager
- Mate Gyomrei : Développeur front-end
- Daniel Denke : Développeur iOS
- Zoltan Tarsoly : Test Automation Engineer
- Youva Ammaouche : Assistant chef de projet

En tant que développeur back end, je suis chargé de rajouter de nouvelles fonctionnalités au DAM.
L'une des missions qui m'a été affectée est la détection des replays dans les vidéos de sport.
Cette fonctionnalité permettra de fournir à nos clients un résumé automatique des vidéos qu'ils publient sur notre plateforme.

2 Introduction

Au cours des dernières années, les techniques d'apprentissage automatique ont joué un rôle de plus en plus important dans les systèmes de reconnaissance automatique.
Les avancements dans le domaine de l'apprentissage profond et l'accès à un volume massif de données ont permi de mettre en place des solutions de plus en plus performantes.
En particulier, l'adoption globale des téléphones intelligents et de leur caméra intégrée a accru de manière exponentielle le nombre de vidéos disponibles sur Internet, à tel point qu'il est devenu impossible pour l'humain d'ingérer manuellement le contenu de toutes les vidéos disponibles sur la toile.
Dans cette étude, nous nous intéressons à la tâche de "video summarization" ou, en français, de récapitulation de vidéo.
Plus particulièrement, nous focalisons notre recherche sur les vidéos de football.

Dans ces vidéos, notre objectif va être de parvenir à identifier les replays; car à partir de ces replays, nous serons capables de mettre en avant les moments importants du match.

Qu'est-ce qu'un replay ? Un replay est la retransmission d'une action qui s'est déjà passée au cours d'une vidéo.
Les replays sont intéressants car ils sont un indicateur d'un moment fort dans une vidéo.
En effet, c'est l'équipe technique chargée du montage de la vidéo qui décide ou non de créer un replay pour une action. Un replay est une annotation humaine sur une vidéo.
Typiquement, un replay sera incrusté dans la vidéo après une action importante comme, par exemple, un but ou un pénalty.

Caractéristique des replays Les replays sont introduits et se terminent par un logo ([1]).
Ces logos ont en général une apparence qui se démarque facilement des autres images dans la vidéo.
Les replays ne sont pas à confondre avec les ralentis.
Les ralentis sont un type particulier de replays où l'action est montrée de nouveau en *slow-motion*, mais tous les replays ne sont pas des ralentis.
C'est pourquoi, la vitesse de déplacement des objets dans l'image (pour détecter l'effet de *slow-motion*) n'est pas un bon critère pour la détection de

replay.

L'objectif de notre recherche est de créer un système robuste de détection de replays dans les vidéos de football.
Dans un premier temps, nous dressons un inventaire de l'état de l'art concernant la détection de replays, puis nous présentons des méthodes par apprentissage profond pour la reconnaissance d'action dans les vidéos.
Une fois les bases théoriques posées, nous proposons quatre approches :

- la première utilise l'algorithme ORB [2] pour extraire des caractéristiques à partir des images de la vidéo et K-Means à partir de ces caractéristiques

- dans la seconde, nous utilisons la détection de contours pour trouver les replays

- dans la troisième, nous nous servons d'un réseau à convolution 3D pour classifier des séquences d'images représentant des morceaux de la vidéo

- et enfin, dans la dernière approche, nous présentons un réseau à convolution 3D qui va apprendre à classifier à partir de séquences d'images de flux optique

Pour conclure cette étude, nous comparons les résultats obtenus pour déterminer quelle méthode répond le mieux à notre besoin.

3 Analyse d'image : état de l'art concernant la détection de replay

La détection de replays est un domaine de recherche à part entière et les articles sont abondants ([1, 3, 4, 5] ...).
Dans cette partie, nous présentons quelques approches que nous trouvons intéressantes.

3.1 Automatic Detection Of Replay Segments In Broadcast Sports Programs By Detection Of Logos In Scene Transitions

Figure 1: Exemple de transition logo

Dans cet article ([1]), Pan propose de détecter les replays dans les vidéos de sport en trouvant les frames contenant des logos.
L'auteur fait l'hypothèse que dans les vidéos de sport, les replays sont toujours compris entre deux animations spéciales qui font la transition entre le replay et le temps réel.
Ces transitions contiennent en général le logo de la chaîne émettant la vidéo ou de l'organisme en charge de l'événement sportif. Un montage spécifique est associé au logo.
Un exemple d'une de ces transitions est donné dans la figure 1. Nous voyons que le montage du logo est incrusté dans la vidéo et se déroule sur plusieurs frames; de plus, sa forme est facilement reconnaissable par rapport aux autres objets.
Une autre hypothèse formulée par l'auteur est que ces transitions sont les mêmes tout au long de la vidéo.
À partir de ces deux hypothèses, les auteurs proposent un système permettant de détecter les replays. L'idée est de trouver dans la vidéo un *logo template* (pour ça, il propose d'utiliser un détecteur de *slow-motion* [6]), puis de trouver tous les logos dans la vidéo grâce au *logo template* et à une mesure de comparaison entre une frame et le *logo template*, puis finalement

d'identifier les replays en regroupant les logos deux à deux.

Nous trouvons cette approche intéressante car elle donne un critère puissant permettant de détecter les replays : ceux-ci commencent et se terminent par une transition de logo.

3.2 A Robust Replay Detection

Cette approche [7] s'inspire de la méthode introduite par Pan et al. ([1]), et détecte les replays en trouvant les logos dans les vidéo.
Les logos sont trouvés grâce à la luminance. Nous savons qu'un logo est présent pendant 0.8 secondes soit 18 frames pour une vidéo de 24 FPS.
Toutes les frames qui ont obtenues un score supérieur à un certain seuil sont considérés comme des candidats pour être le *logo template*.
Le *logo template* est le logo qui représente le mieux les logos dans la vidéo et c'est celui-ci qui va servir de référence pour trouver tous les logos.

3.2.1 Calcul du score de la luminance et filtrage des frames

L'idée est de parcourir toute la vidéo et de calculer pour chaque frame la différence de luminance qu'il y a entre cette frame et les 17 frames précédentes.
Nous obtenons un score L_i pour chaque frame i dans la vidéo.
Toutes les frames dont le score est inférieur à un certain seuil sont écartés, les autres vont servir à trouver le logo template.

3.2.2 Recherche du logo template parmi les frames filtrées

Le logo template est le frame qui représente le mieux tous les logos dans la vidéo.
Pour déterminer le logo template parmi les frames filtrées, l'algorithme K-means est utilisé pour séparer cet ensemble en deux ($K = 2$) en fonction de la luminance moyenne des frames.
Pour trouver le logo template, il faut d'abord trouver le cluster qui contient les logos. Les auteurs estiment que c'est le cluster avec le centre det cluster le plus élevé.
Une fois le cluster de logos déterminé, le *logo template* est la frame m qui minimise la distance avec toutes les autres frames du cluster. La formule qui

permet de calculer la distance est donnée en 2.

$$d_{mn} = \sqrt{\sum_{i=0}^{\text{binsize}} \left[\frac{H_m(i)}{\sum_{j=0}^{\text{binsize}} H_m(j)} - \frac{H_n(i)}{\sum_{j=0}^{\text{binsize}} H_n(j)} \right]^2} \quad (4)$$

Figure 2: Formule de la distance entre deux frames dans le cluster

3.2.3 Recherche des logo

Une fois que le logo template est déterminé, chaque logo trouvé précédemment va être comparé avec le logo template.
La mesure de comparaison est la distance (figure 2) qu'il y a entre le frame et le template dans le cluster.
Tous les frames qui ont une distance inférieure à un certain threshold sont considérés comme des logos.

3.2.4 Recherche des replays

Une fois que les logos sont détectés, nous pouvons trouver les replays en cherchant les paires de logos éloignés de moins de 80 seconde (durée maximum d'un replay).

3.2.5 Résultats obtenus et conclusion sur cette méthode

	Video					
Video width / Video height / Luminance threshold	Football : Premier League (150k frames)	Football: Ligue 1 (80k frames)	Football: Ligue 1 (150k frames)	Football: Liga (300k frames)	Tennis: Australia Open (400k frames)	American football: NFL (*) (200k frames)
100,100, 50000	>100,5/48,150	45,0/16,70	>100, 0/34, 200	>100,0/36,150	>100, 12/32, 280	**X**
100,100, 75000	>100,12/48,150	21,0/16,70	58,0/34,200	43, 0/36, 150	**>100, 12/32**, 280	**X**
100,100, 100000	52, 32/48, 150	7, 0/16, 70	16, 0/34, 200	23, 12/36, 150	53, 15/32, 280	**X**
100,100, 125000	12, 48/48, 150	**4, 0/16, 70**	**5, 0/34, 200**	13, 36/36, 150	33, 32/32, 280	**X**
100,100, 150000	5, 48/48, 150	2, 2/16, 70	2, 3/34, 200	4, 36/36, 150	23, 32/32, 280	**X**

Score : False Positive (FP), False Negative (FN), Time (rounded)
(*) Didn't work because in NFL games, the logo at the and at the beginning of a replay do not match

Après avoir implémenté cette méthode, nous avons constaté que celle-ci n'est pas efficace et ne fonctionne pas du tout sur notre ensemble de test.

Cette approche est trop dépendante de la luminance et elle ne parvient pas à détecter les logos peu lumineux.
De plus, les bases mathématiques (notamment la manière de choisir le cluster et la mesure de distance) sont un peu douteuses.
Enfin, celle-ci dépend trop du paramètre "seuil de luminance" affectant les logos détectés.
Le seuil de luminance fourni par les auteurs ne produit pas de bons résultats sur toutes les vidéos.
Nous n'avons pas réussi à trouver une valeur pour le seuil de luminance qui obtienne universellement de bons résultats.
Un seuil à 100000 détecte les logos de Ligue 1 mais pas les logos de Liga.
Un seuil de 75000 détecte les logos de Liga et de Ligue 1, mais laisse passer trop de faux positifs.
Les logos de Premier League quant à eux ne sont pas tous détectés avec un seuil à 50000, alors que ce seuil accepte un grand nombre de faux positifs.

Pour conclure, cette approche n'est pas celle qui va nous permettre de mettre en place un système de détection de replays robuste et efficace. Néanmoins, cet article nous a renforcé dans notre idée qu'une solution basée sur l'analyse d'image peut obtenir de bons résultats.

3.3 Mean Shift Based video Segment Representation And Applications To Replay Detection

Dans cet article [3], les auteurs présentent une méthode permettant de détecter les replays.
L'idée est d'apprendre une base de représentation compressée de logos avec une méthode comme le spectral hashing ([8]), puis de se servir de cette base de données pour trouver les logos au début à la fin de replays.

3.3.1 Algorithme

```
L = []
R = []
Segmenter la vidéo en frame
Pour chaque frame f
  Calculer la représentation r_f de f
  Pour chaque représentation r dans la base de représentation:
```

```
    Si distance(r_f, r) < Seuil:
      Ajouter f à L
Pour chaque logo l dans L:
  Trouver le logo l' lui correspondant
  Ajouter (l, l') à R
```

La représentation des images est un hash obtenu par un algorithme de hashing d'image (spectral hashing).
La distance utilisée pour comparer le hash des images est la distance de Wasserstein .

3.3.2 Résultats

Match	Total	Correct	False Alarm	Recall	Precision
GER-KOR (25/06/02)	67	65	4	97.0%	94.2%
GER-BRA (30/06/02)	33	30	5	90.9%	85.7%
SEN-TUR (22/06/02)	48	46	4	95.8%	92.0%
SEN-FRA (31/05/02)	54	52	6	96.3%	89.7 %

Figure 3: Performance sur la tâche de détection de replay

Les résultats obtenus par les auteurs sont présentés dans la Figure 3.
Ceux-ci sont bons, mais l'ensemble de test n'est pas assez représentatif (seulement quatres vidéos).
Les avantages des représentations hashées pour les images sont les suivants :

- un hash est compacte (peu d'espace nécessaire pour les stocker)
- comparer des hash est rapide (comparer deux frames)
- chercher un hash dans une table de hachage est rapide (chercher un frame dans une base de données)

Pour ces raisons, cette approche est tout à fait adaptée à la reconnaissance de logo.

3.4 Automatic summarization of soccer highlights using audio-visual descriptors

Cette approche [9] ressemble à l'approche par luminance ([7]), mais introduit néanmoins une différence importante : un pré-traitement sur la vidéo pour en extraire les plans.

3.4.1 Algorithme

L'algorithme est le suivant :

```
S = Détecter tous les shots (plans) dans la vidéo
L = Pour chaque shot S_i:
1. L_i_start = La "luminance" des frames au début du shot
2. L_i_end = La "luminance" des frames à la fin du shot
3. L_template = Trouver le "logo template" dans L
4. Pour chaque logo l dans L:
   1. Diff l avec L_template = conversion grayscale puis somme de la soustraction pixel
   2. Si Diff l avec L_template < threshold => l est un logo
```

3.4.2 Apport de l'article

Cette méthode est trop semblable à l'approche "Robust Replay Detection" qui ne répond pas à nos besoins, cette approche ne fonctionnera pas dans notre cas (la luminance n'est pas un critère assez discriminant pour la reconnaissance de logo).
Néanmoins, l'idée de découper la vidéo en "shot" (en plan) est intéressante et nous nous en servons par la suite.

4 Apprentissage automatique : les bases théoriques

Dans notre recherche, nous allons aborder plusieurs types de réseaux d'apprentissage automatique.
Nous allons présenter dans cette partie les principes fondamentaux à la bonne compréhension de ces derniers.

4.1 Réseaux de neurones récurrents (RNN)

Les RNN (Recurrent Neural Networks), ou réseaux de neurones récurrents (RNR) en français, sont capables de répéter leur couche cachée, en utilisant comme entrée la sortie de toutes les couches précédentes et de générer une sortie pour chaque couche.
Cela va leur permettre de prendre en entrée des séquences et de retourner des séquences.
En effet, pour une entrée $[e_1, e_2, \ldots, e_n]$ et un initialiseur s_0, le RNN va répéter n fois sa couche cachée, de telle sorte à générer une sortie s_1 associée à la couche 1 et à l'entrée (e_1, s_0); puis il va générer une sortie s_2 associée à la couche 2 et à l'entrée (e_2, s_1), etc ...
Pour finir, nous aurons en sortie la séquence $[s_1, s_2, \ldots, s_n]$.

Par exemple, appliqués à la génération de phrase, les RNN vont être capables de générer (mot par mot, ou n-gram par n-gram) des séquences de phrases de longueur arbitraire.

Pour apprendre un modèle, le RNN va avoir besoin d'un ensemble d'entraînement qui met en avant les propriétés qui nous intéressent dans le modèle.

La nature récursive de ces réseaux les rend particulièrement adaptés aux tâches de traitement du langage naturel ou pour traiter la temporalité (ce qui nous intéresse car la temporalité est ce qui différencie l'analyse d'images à l'analyse vidéo).

4.2 LSTM

Les LSTM (Long Short Term Memory) sont un type de RNN **à portes (gated RNN)**.
Ces portes vont permettre de stocker l'information apprise par le réseau à

travers le temps.
À la différence des RNN classiques, les LSTM sont capables d'oublier de l'information grâce à leur **leaky unit** afin d'éviter une explosion ou une disparition du gradient.
Par exemple, si nous voulons entraîner un LSTM pour qu'il puisse reconnaître une action courte dans une vidéo, ce dernier n'a pas besoin d'enregistrer toutes l'information acquise depuis le premier frame, il lui suffit de connaître un voisinage de quelques frames.
La puissance de ces réseaux à portes est que c'est le réseau qui va apprendre à décider quand vider son état interne.
Concrètement, cela va leur permettre de pouvoir capturer des dépendances à long terme de manière bien plus efficace que les RNN classiques.

4.3 CNN

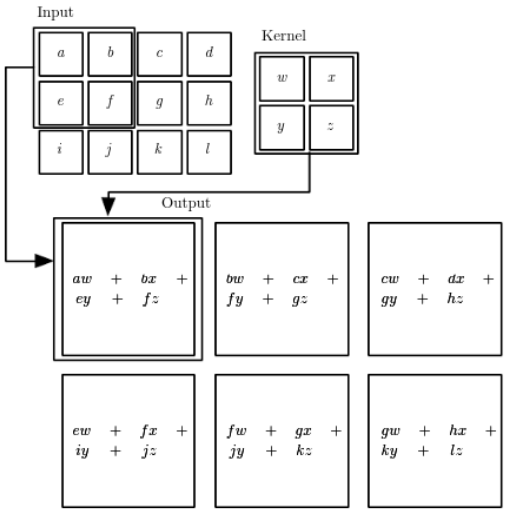


Figure 4: Opération de convolution

Les CNN (Convolutional Neural Networks), ou réseaux de neurones convolutifs (RNC) en français, sont un type de réseau de neurones qui utilisent la convolution au lieu de la multiplication matricielle dans au moins une de leurs couches.
La convolution est une opération qui prend en argument l'entrée (typiquement un vecteur représentant une donnée) et un **noyau** (les paramètres qui

vont être appris par le CNN) et renvoie une **carte de caractéristiques** (feature map).
Le noyau est une matrice qui va parcourir l'entrée et appliquer l'opération de convolution.
Pour parcourir l'entrée, celle-ci va être divisée en plusieurs matrices carrées de même taille que le noyau (par exemple 2x2 ou 6x6) en ajoutant si nécessaire du *padding* et du *striding*.
La fonction de convolution a trois caractéristiques importantes : l'**intéraction parcimonieuse** ("sparse interaction"), le **partage de paramètres** et les **représentations équivariantes**.
La couche de convolution est généralement composée de la fonction de convolution suivie d'une fonction d'activation non linéaire (par exemple, ReLU ou tanh) et d'une d'une fonction de **pooling**.

4.3.1 Intéraction parcimonieuse

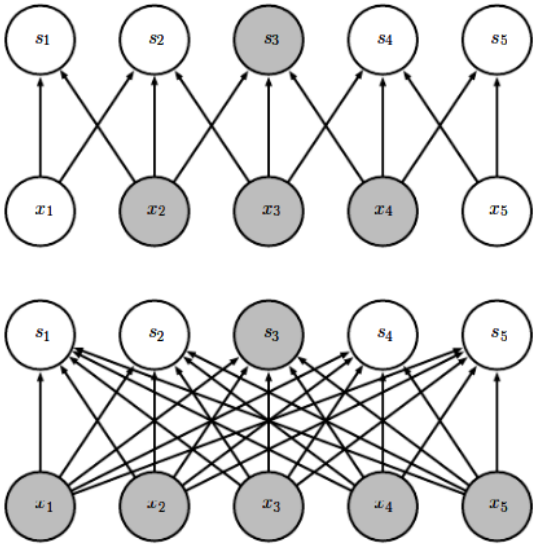


Figure 5: Intéraction parcimonieuse (en haut), intéraction non parcimonieuse (en bas)

À la différence des réseaux classiques où toutes les sorties intéragissent avec toutes les entrées, les réseaux à convolution ont des **intéractions parci-**

monieuses.
C'est-à-dire que la taille du noyau (donc de l'intéraction avec l'entrée), est plus petite que la taille de l'entrée.
Une image a une dimension en entrée de $c*l*w$ où c est le nombre de canaux de l'image (un seul pour une image en noir et blanc, trois pour une image en couleur), l la largeur en pixel de l'image et w la longueur en pixel de l'image.
Une petite image couleur de dimension 100*100 aura 100*100*3 paramètres en entrée, ce qui provoque une explosion combinatoire avec les réseaux classiques qui n'ont pas d'intéraction parcimonieuse car il faudra une connexion entre chaque paramètre d'entrée et une entrée du réseau.
Un réseau de convolution, quant à lui, aura un noyau d'une dizaine ou d'une centaine de pixel qui parcourt l'image à la recherche de caractéristiques significatives comme des contours.
Cela signifie que l'intéraction parcimonieuse permet aux CNN de stocker moins de paramètres que les autres types de réseau.
Par conséquent, ils ont donc ont besoin de moins de mémoire (pour la même tâche) et ont une meilleure efficacité statistique.
C'est l'une des raisons faisant que les réseaux à convolution sont très efficaces pour le traitement d'image.

4.3.2 Partage de paramètres

Dans un réseau classique, un poids (un paramètre) est associé à chaque paramètre d'entrée et ne sert qu'une fois.
Tandis que dans un réseau convolutif, le noyau utilisé par une couche de convolution est le même sur toutes les matrices représentant l'entrée.
Grâce à ce **partage des paramètres**, il n'y a que les poids du noyau à apprendre au lieu d'un poids pour chaque neurone d'entrée.
De plus, la taille du noyau est en général largement inférieure à celle de la couche d'entrée.

4.3.3 Représentations équivariantes

Une fonction est **équivariante** si, quand l'entrée change, la sortie change de la même manière.
En terme mathématique, cela signifie que si $y = f(x)$ alors $g(y) = g(f(x))$.
Les réseaux convolutifs sont équivariants à la translation.
Dans le cas de l'image, cela veut dire que le déplacement des pixels n'a pas

d'influence sur le réseau.

4.3.4 Pooling

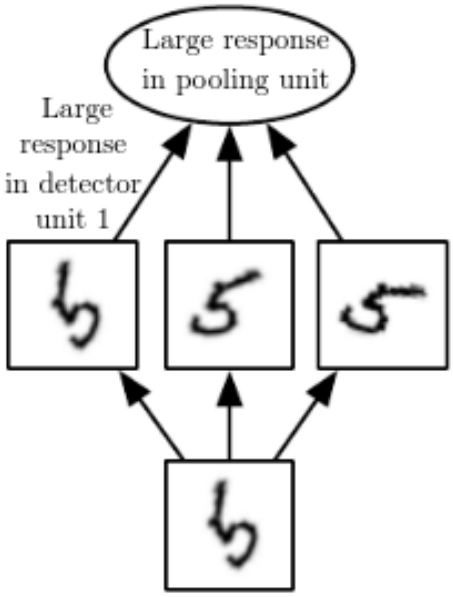


Figure 6: Pooling & invariance

La fonction de pooling va modifier la sortie de la couche de convolution.
Pour chaque valeur dans la carte des caractéristiques à la sortie de la convolution (après la fonction d'activation), la fonction de pooling va remplacer celle-ci en fonction de la valeur des cases voisines dans la carte.
Une fonction de pooling usuelle est max pooling, qui va renvoyer la plus grande valeur dans un voisinage rectangulaire.
L'utilité de la fonction de pooling est de rendre la représentation apprise par la couche de convolution **invariante** à de petites modifications sur l'entrée. Par exemple, dans le cas de la reconnaissance d'image, le réseau ne va pas chercher dans l'image en entrée les informations au pixel près.
Si le réseau a appris à détecter les visages, il n'a pas besoin de retrouver l'emplacement des yeux au pixel près, une position approximative de ceux-ci lui suffira.
Une autre utilité du pooling est de réduire la taille de la sortie de la couche

de convolution.
Nous pouvons voir le pooling comme un résumé de la carte des caractéristiques obtenue par convolution.

5 Apprentissage profond : état de l'art pour la reconnaissance d'action dans les vidéos

Nous nous intéressons à l'état de l'art concernant la détection d'action dans les vidéos.
En effet, la transition d'un logo s'effectue sur plusieurs frames consécutifs; il y a donc une composante temporelle à notre recherche, et nous pouvons considérer la transition d'un logo comme une action.

5.1 Two-Stream Convolutional Networks for Action Recognition in Videos

Cet article est écrit par Karen Simonyan et Andrew Zisserman [10]. Dans celui-ci, ils proposent de séparer la tâche de reconnaissance d'action dans les vidéos en deux parties : une composante spatiale et une composante temporelle.

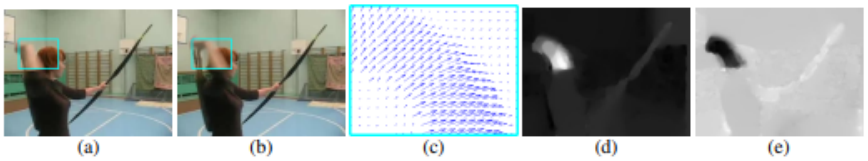
La composante spatiale contient l'information concernant les objets dans la vidéo; tandis que la composante temporelle l'information sur les déplacements de ces objets et de la caméra.
A partir de ces observations, les auteurs proposent d'entraîner un classifieur spatial (Spatial stream ConvNet) et un classifieur temporel (Temporal stream ConvNet).

Ces classifieurs sont des réseaux de neurones convolutifs profonds.

5.1.1 Classifieur spatial

Ce réseau a une architecture de classifieur d'image classique.
Il va permettre de donner un indice fort pour la prédiction, car certaines actions sont très liées à certains objets.
De plus, la recherche dans le domaine de la classification est un domaine à part entière; toutes les avancées dans le domaine augmenteront l'efficacité de ce classifieur.
Il n'est pas nécessaire d'apprendre ce réseau "from scratch" (de zéro), les approches par *transfer learning* sont efficaces.

5.1.2 Classifieur temporel



Optical flow. (a),(b): a pair of consecutive video frames with the area around a moving hand outlined with a cyan rectangle. (c): a close-up of dense optical flow in the outlined area; (d): horizontal component d^x of the displacement vector field (higher intensity corresponds to positive values, lower intensity to negative values). (e): vertical component d^y. Note how (d) and (e) highlight the moving hand and bow. The input to a ConvNet contains multiple flows (Sect. 3.1).

Figure 7: Flux optique

L'innovation de l'article vient de l'introduction du classifieur temporel.

L'idée est de détecter le mouvement des objets dans la vidéo, car un mouvement est la représentation d'un objet dans le temps.
Les auteurs appellent leur approche "optical flow stacking" (empilement de flux optique).

Dans celle-ci, ils utilisent le flux optique pour représenter le mouvement des objets entre des frames consécutifs.

Ils définissent aussi un hyperparamètre L qui définit la distance maximum entre deux frames pour laquelle il faut calculer le flux optique.
Par exemple, si L=5, alors pour le frame t, il faudra calculer le flux entre le frame t et le frame t+1; entre t+1 et t+2; etc... jusqu'à t+4 et t+5.
Le flux optique entre deux frames est une matrice de dimension 2, il peut donc être sauvegardé sous forme d'image.
Ainsi, des images représentant le flux optique entre les différents frames de la vidéo vont être générées.

Chacune de ces images servira d'entrée au CNN (classifieur temporel).

5.1.3 Méthode d'évalutation et résultats obtenus

Le classifieur spatial est pré-entrainé avec ImageNet, tandis que le temporel est entraîné de zéro (car il n'y a pas de réseau déjà entraîné pour cette

Table 4: **Mean accuracy (over three splits) on UCF-101 and HMDB-51.**

Method	UCF-101	HMDB-51
Improved dense trajectories (IDT) [26, 27]	85.9%	57.2%
IDT with higher-dimensional encodings [20]	**87.9%**	61.1%
IDT with stacked Fisher encoding [21] (based on Deep Fisher Net [23])	-	**66.8%**
Spatio-temporal HMAX network [11, 16]	-	22.8%
"Slow fusion" spatio-temporal ConvNet [14]	65.4%	-
Spatial stream ConvNet	73.0%	40.5%
Temporal stream ConvNet	83.7%	54.6%
Two-stream model (fusion by averaging)	86.9%	58.0%
Two-stream model (fusion by SVM)	**88.0%**	**59.4%**

Figure 8: Résultats obtenus par l'approche Two-stream model

tâche).
Les ensemble de données utilisés pour l'entraînement et l'évaluation sont UCF-101 et HMDB-51, contenant à eux deux près de 20000 vidéos annotées. Avant l'entraînement, les images de flux optiques sont pré-calculées afin de n'avoir à générer ces images qu'une seule fois.

Note Pour calculer la classe d'un frame à l'instant t, les auteurs proposent deux méthodes :

- fusion par la moyenne (by averaging) : $y_t = y_{tspatial} + y_{ttemporal} \ / \ 2$
- fusion par SVM (by SVM) : un SVM multiclasse linéaire est entrainé pour prédire la classe à partir du softmax des scores L2-normalisés.

Les résultats (figure 8) montrent l'efficacité de leur méthode par rapport aux autres approches état de l'art.

Nous pouvons voir que leur approche two-stream avec fusion SVM est la plus efficace sur le dataset UCF-101, et qu'elle a aussi de bons résultats sur HMDB-51.

Ce qui est le plus intéressant dans cet article, c'est l'amélioration qu'apporte l'ajout de la composante temporelle.
En effet, le classifieur d'image simple (spatial) n'a que 73.0% (UCF-101) et 40.5% (HMBD-51), tandis que le classifieur qui prend en compte l'image et la temporalité (two-stream model) atteint **88.0%** et 59.4%; ce qui est une

nette amélioration.

Cet article nous a renforcés dans l'hypothèse qu'il est nécessaire d'étudier une vidéo non pas comme une suite d'images indépendantes, mais comme une suite de séquences avec un lien entre chaque élément de la séquence. La temporalité a une très grande importance pour l'analyse de vidéos, et le flux optique est une méthode efficace pour modéliser le déplacement des objets entre deux instants.

5.2 Learning Spatiotemporal Features with 3D Convolutional Networks

Dans cet article [11], les auteurs proposent une approche pour apprendre les caractéristiques spatio-temporelles dans les vidéos grâce à un réseau de neurones à convolution.

Ils font l'hypothèse qu'un réseau avec une couche de convolution 3D qui prend en entrée une séquence d'images est capable d'apprendre efficacement les mouvements des objets dans les vidéos.

L'objectif est d'apprendre des caractéristiques qui soient :

- génériques : c'est-à-dire la capacité à représenter différents types de vidéos
- compactes : afin de pouvoir stocker un grand nombre de ces caractéristiques
- efficace (computationnellement): pour traiter les vidéos en temps réel
- simples : afin de fonctionner même avec les modèles simples (comme un classifieur linéaire)

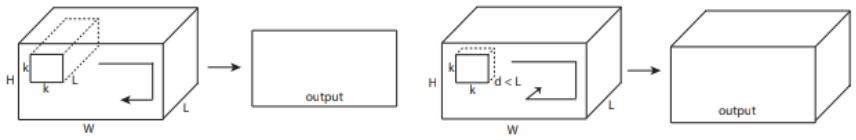


Figure 9: Convolution 2D sur une séquence d'images (gauche), convolution 3D sur une séquence d'images (droite)

5.2.1 C3D: Convolution et pooling 3D

Les auteurs appellent leur approche C3D (3D ConvNet).
Comparé aux réseaux à convolution 2D, C3D est capable de modéliser plus efficacement l'information spatio-temporelle grâce à la convolution et au pooling sur trois dimensions.
La convolution 2D appliquée à une image produira en sortie une image, la convolution 2D appliquée à une suite d'images produira aussi une image.
C'est pourquoi les réseaux à convolution 2D perdent l'information temporelle après l'opération de convolution. La convolution 3D permet, elle, de préserver cette information dans sa dimension additionnelle.

5.2.2 Architecture et entraînement du réseau

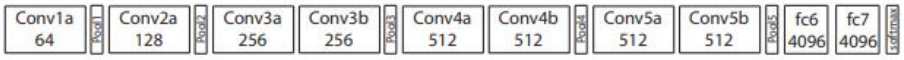


Figure 10: Architecture C3D

L'entrée de ce réseau est de dimension c * l * h * w où c est le nombre de canaux des images (3 pour la couleur, 1 pour les images en noir et blanc), l le nombre d'images dans les séquences, h la longueur et w la largeur en pixel des images.

L'architecture conseillée par les auteurs est 8 couches de convolution et 5 couches de pooling, ainsi que 2 couches complètement connectées et la fonction softmax pour la couche de sortie.
Le kernel recommandé par les auteurs est 3 * 3 * 3 avec un pas (stride) de 1 * 1 * 1 pour toutes les couches de convolution.
Toutes les couches de pooling sont max pooling avec une taille de kernel 2 *

2 * 2 (sauf pour la première qui est 1 * 2 * 2) avec un stride 2 * 2 * 2 (sauf pour la première qui a un stride de 1 * 2 * 2).
Pour finir avec l'architecture, les deux couches complètement connectées ont 4096 sorties.

Ce réseau va être entraîné de zéro par descente du gradient à partir de séquences d'images annotées.
Le taux d'apprentissage est de 0.003 et est divisé par 10 toutes les 4 epoch.
L'entraînement s'arrête après 16 epoch.

Après l'entraînement, le réseau peut être utilisé comme un extracteur de caractéristiques pour des tâches d'analyse vidéo.
Pour se faire, la vidéo va être découpée en des clips de 16 frames (avec 8 frames de chevauchement entre deux clips consécutifs).
Ensuite, chacun de ces clips va être passé au réseau et l'avant dernière couche complètement connectée (fc6) va contenir les caractéristiques du clip.

Qu'est-ce que ce réseau apprend ? Ce réseau apprend à se focaliser sur l'image des premiers frames, et à traquer leur déplacement dans les frames suivants.

5.2.3 Résultat pour la tâche de reconnaissance d'action

Ces résultats (figure 11) ont été obtenus par les auteurs pour la tâche de reconnaissance d'action sur le corpus de vidéo UCF101.
Nous voyons que l'approche par réseau à convolution 3D est la plus efficace.

5.2.4 Conclusion

Dans cet article, les auteurs ont adressé le problème de la temporalité dans les vidéos.
Ils ont montré qu'un réseau à convolution 3D est capable de modéliser l'information temporelle et spatiale simultanément, et donc d'obtenir de meilleurs résultats que les réseaux à convolution 2D pour l'analyse de vidéos.
De plus, cette approche est élégante car elle fonctionne sans ajout artificiel d'information (comme le flux optique), c'est le réseau qui va se charger d'apprendre ce dont il a besoin pour apprendre la temporalité.

Method	Accuracy (%)
Imagenet + linear SVM	68.8
iDT w/ BoW + linear SVM	76.2
Deep networks [18]	65.4
Spatial stream network [36]	72.6
LRCN [6]	71.1
LSTM composite model [39]	75.8
C3D (1 net) + linear SVM	82.3
C3D (3 nets) + linear SVM	**85.2**
iDT w/ Fisher vector [31]	87.9
Temporal stream network [36]	83.7
Two-stream networks [36]	88.0
LRCN [6]	82.9
LSTM composite model [39]	84.3
Conv. pooling on long clips [29]	88.2
LSTM on long clips [29]	88.6
Multi-skip feature stacking [25]	89.1
C3D (3 nets) + iDT + linear SVM	**90.4**

Figure 11: Résultats pour l'approche Learning Spatiotemporal Features with 3D Convolutional Networks (C3D) comparés à d'autres approches état de l'art

5.3 Beyond Short Snippets: Deep Networks for Video Classification

Dans cet article [12], les auteurs proposent d'utiliser une architecture hybride à base de CNN et de RNN (LSTM) pour l'analyse vidéo.
Leur objectif est d'apprendre des dépendances à long terme dans les vidéos, d'où l'utilisation d'un LSTM.
Les CNN sont des réseaux particulièrement efficaces pour analyser les frames des vidéos, c'est le CNN qui va se charger de la composante spatiale de la vidéo.
Le LSTM va servir à apprendre la composante temporelle.

5.3.1 Approche

L'objectif des auteurs est d'apprendre des dépendances à long terme dans les vidéos.
Les réseaux à convolution sont très efficaces pour l'analyse d'image; mais leur coût computationnel est très élevé.
C'est pourquoi, il n'est pas possible de se servir d'un CNN pour apprendre les dépendances à long terme; en effet, il faudrait que le réseau prenne en

entrée toute la vidéo (ou bien une grande partie), ce qui va provoquer une explosion combinatoire des paramètres à apprendre par le réseau.
Les auteurs font l'hypothèse que tous les frames dans la vidéo ne sont pas utiles, et qu'il est judicieux de ne garder qu'un sous-ensemble des frames de la vidéo; ils proposent donc de ne traiter qu'un frame par seconde. L'intérêt de ne garder qu'un sous ensemble des images de la vidéo et qu'il va donc être possible de traiter la vidéo sur une plage temporelle plus large pour le même coût en calcul.
Néanmoins, ne regarder qu'une seule image par seconde dans la vidéo ne préserve pas le déplacement des pixels entre l'instant t et l'instant t + 1 (en seconde) et donc l'information du mouvement des objets.
Pour palier à ce problème, le flux optique (de la même manière que [10]) est calculé entre les frames adjacents.
Ainsi, l'information temporelle et l'information spatialle sont préservées, tout en ne traitant qu'une seule image par seconde, ce qui réduit beaucoup le coût de calcul.
Pour apprendre les dépendances qu'il y a entre les frames, un LSTM est utilisé; celui-ci va traiter les vidéos comme des séquences d'images et va apprendre à prédire la classe de la vidéo en fonction de ces séquences.

5.3.2 Architecture

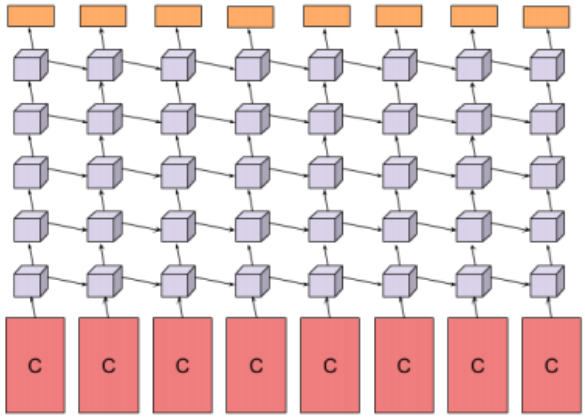


Figure 12: Architecture du LSTM

Comme pour [10], un réseau à convolution va être utiliser pour extraire les

caractéristiques visuelles de la vidéo.
Les architectures utilisées pour ce réseau sont GoogLeNet et AlexNet.

L'architecture proposée pour le LSTM est présentée en 12.
La sortie du CNN est processée par propagation avant à travers cinq couches de LSTM.
La couche de sortie du LSTM est munie de la fonction softmax et prédit une classe à chaque étape.
Les paramètres du réseau à convolution et de la couche de sortie du LSTM sont partagés pour toutes les étapes.

5.3.3 Utilisation du flux optique

Le flux optique encode l'information des déplacements des objets dans la vidéo.
Comme pour l'approche [10], les images de flux vont être pré-calculées et servir lors de l'entraînement du CNN.

5.3.4 Résultat obtenu

Les auteurs évaluent leur approche sur le dataset Sports-1M et UCF-101 sur la tâche de classification de vidéos.
Nous pouvons voir que leur approche obtient les meilleurs résultats (88.6% contre 88.0% pour [10]).
Ces résultats sont intéressants car à la différence des autres approches, celle-ci est capable de prédire une classe pour des morceaux de vidéos pouvant aller jusqu'à deux minutes (contre quelques secondes pour les autres).
De plus, les meilleurs résultats sont là aussi obtenus en utilisant le flux optique, confirmant l'hypothèse faite par [10] que ce dernier est nécessaire pour le traitement des vidéos.

Dans notre cas, cette approche n'est pas la plus adaptée. En effet, nous souhaitons reconnaître les logos dans les vidéos, or un logo ne dure pas plus longtemps que quelques secondes et les dépendances à long-terme que le LSTM va apprendre ne nous intéressent pas.
Néanmoins, une variante de celle-ci où le LSTM reçoit en entrée une séquence de frames consécutifs (et pas une séquence formée d'un frame par seconde) pourrait avoir de bons résultats pour la tâche de détection de logos.

Method	3-fold Accuracy (%)
Improved Dense Trajectories (IDTF)s [23]	87.9
Slow Fusion CNN [14]	65.4
Single Frame CNN Model (Images) [19]	73.0
Single Frame CNN Model (Optical Flow) [19]	73.9
Two-Stream CNN (Optical Flow + Image Frames, Averaging) [19]	86.9
Two-Stream CNN (Optical Flow + Image Frames, SVM Fusion) [19]	88.0
Our Single Frame Model	73.3
Conv Pooling of Image Frames + Optical Flow (30 Frames)	87.6
Conv Pooling of Image Frames + Optical Flow (120 Frames)	**88.2**
LSTM with 30 Frame Unroll (Optical Flow + Image Frames)	**88.6**

Figure 13: Résultat obtenu pour l'approche combinant CNN et LSTM (LSTM with 30 Frame Unroll)

6 Les approches proposées

L'objectif de notre recherche est de détecter les replays dans les vidéos de sport.

Pour détecter les replays, nous faisons les hypothèses suivantes :

- un replay a un logo de début (I) [1]
- un replay a un logo de fin (II) [1]
- les logos de début et de fin sont les mêmes (III) [1]
- les logos ont une forme facilement reconnaissable qui se distingue des autres images dans la vidéo (IV)
- un replay dure entre 2 et 90 secondes (V)

En faisant ces hypothèses, la tâche de détection de replays devient une tâche de détection de logos.

Nous proposons plusieurs approches permettant de détecter les logos de replay dans les vidéos de sport.
En premier lieu, nous proposons deux approches n'utilisant que des algorithmes d'analyse d'images classiques (flouttage, filtre de Canny, ORB, ...) :

- la première se sert de l'algorithme ORB ([2]) et de l'algorithme K-Means
- la seconde utilise la détection de contours pour trouver les images avec des contours similaires dans la vidéo

Ensuite, nous présentons deux approches par apprentissage profond :

- la première utilise un réseau à convolution 3D sur une séquence d'images (similaire à [11])

- la seconde utilise un réseau à convolution 3D sur des séquences d'images représentant le flux optique des objets dans la vidéo

6.1 Détection des plans

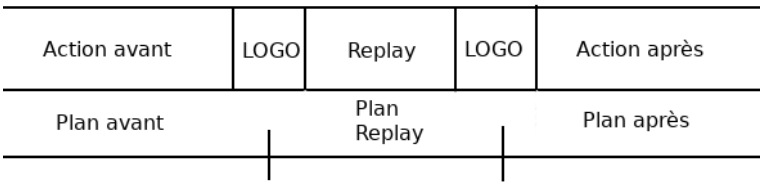


Figure 14: Changement de plan introduit par un replay

Les approches que nous proposons recherche dans la vidéo les logos pouvant se trouver au début et à la fin des replays.
Or, un replay introduit un changement avant et après lui (car l'action rejouée par le replay n'est pas sur le même plan que l'action avant et après lui).
Si nous faisons l'hypothèse qu'un replay entraînera toujours un changement de plan et que les logos sont au début et à la fin du replays, alors au lieu de rechercher les logos parmi tous les frames de la vidéo, nous pouvons réduire la recherche seulement aux frames qui sont entre deux plans.
En effet, le logo au début du replay introduit un changement de plan entre l'action avant le replay et le replay, et le logo à la fin du replay introduit un changement de plan entre l'action après le replay et le replay. La figure 14 illustre cette hypothèse.

C'est pourquoi nous allons chercher une méthode permettant de détecter les changements de plan dans les vidéos.

6.1.1 Online, Simultaneous Shot Boundary Detection And Key Frame Extraction For Sports Videos Using Rank Tracing

Cette méthode est proposée par W. Abd-Almageed en 2008 [13].

Chaque frame est converti en HSV et les histogrammes H, S et V sont calculés.
Un vecteur est formé pour chaque frame à partir de ces histogrammes.
Ensuite, une matrice M de dimension N * L, représentant une fenêtre de N frames va être formée à partir de ces vecteurs, où L est la taille des histogrammes et N la taille de la fenêtre.

L'algorithme SVD (singular value decomposition) va être appliqué sur M. $M = UWV$, où W est la matrice de valeurs singulières.

Les diagonales de la matrice W comportent des poids S ordonnés de manière décroissante.
Le premier poids S_1 est le poid maximal. Ces poids représentent l'information contenue dans le vecteur V.

Nous allons assigner un rang à la matrice M, ce rang va être égal au nombre d'éléments s dans S tel que s/S1 $>$ threshold. Le rang va être calculé pour chaque fenêtre de frames dans la vidéo.

Si le rang d'une fenêtre est plus grand que le rang de la fenêtre avant elle, alors le contenu visuel de la fenêtre est différent de la fenêtre précédente.
À l'inverse, si le rang est inférieur à la fenêtre précédente, alors le contenu visuel se stabilise. S'il est de 1, alors c'est stable.

Le début d'un nouveau plan est indiqué par la fenêtre qui maximise le rang parmi les fenêtres environnantes.

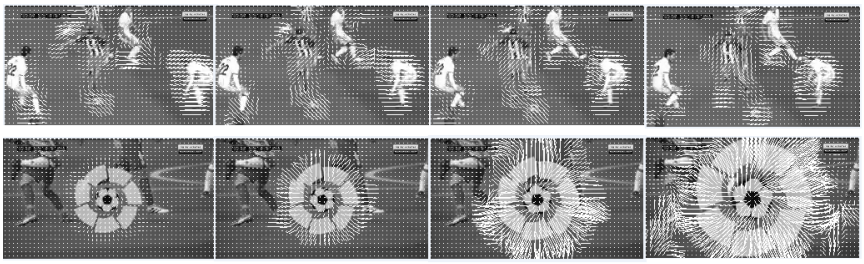
Cette méthode pour trouver les plans dans une vidéo est très efficace, et nous servira dans la suite de notre recherche.

En effet, avant de segmenter la vidéo en plan, nous comparions N frames , où N peut être aussi grand que 400000 (pour des vidéos de 120 minutes à 60 fps).
Il est impensable d'utiliser un algorithme en $O(N^2)$, par exemple en comparant tous les frames entre eux, avec un N aussi grand.

Après avoir segmenté la vidéo en plan, nous obtenons un N' aux alentours de 2000 pour une vidéo de 120 minutes à 60 fps.
Nous pouvons donc nous permettre d'utiliser des algorithmes plus complexes que sans la segmentation en plan.

De plus, la segmentation en plan réduit le champ de recherche des frames logo, et donc le nombre de faux positifs.

6.2 Flux optique



Le flux optique décrit le déplacement des objets entre deux frames consécutifs, ce mouvement peut être causé par un déplacement de la caméra ou de l'objet lui-même.
Le flux optique est une matrice X * Y * D, où X et Y sont l'axe des absisses et des ordonnées (resp.) dans l'image, et D une droite décrivant le déplacement du pixel à la position (x,y) entre le frame t et le frame t + 1.

De la même manière que pour [14], nous allons traquer le déplacement des objets grâce au flux optique.
Il existe deux types d'algorithme de calcul du flux optique :

- sparse/creux : seulement le déplacement de certains points d'intérêt va être traqué (méthode de Lucas-Kanade)

- dense : le déplacement de tous les points dans l'image va être traqué (algorithme de Gunner Farneback)

Le calcul du flux optique dense est plus couteux mais à l'avantage de ne pas nécessiter de déterminer les points d'intérêt dans l'image avant de calculer le flux optique.

6.3 ORB et K-means

Dans cette approche, nous cherchons à reconnaître les logos dans les vidéos. Pour ce faire, nous optons pour une approche de clustering.
L'idée est de séparer la vidéo en deux groupes : un groupe pour les frames logo, et un autre groupe pour les frames non-logo.

L'algorithme KMeans permet de regrouper les objets similaires en fonction de leurs caractéristiques.
Dans notre cas, il va nous permettre de créer deux groupes d'images : logo / non logo.
L'avantage de KMeans est qu'il est est rapide et efficace dans la plupart des cas.

Pour fonctionner, KMeans a besoin que les objets que l'on souhaite séparer en groupe soient sous forme de vecteurs et que ces vecteurs soient discriminants. C'est-à-dire qu'un frame logo ne doit pas être proche dans l'espace d'un frame non-logo.

Il existe plusieurs manières de vectoriser une image, par exemple :

- une matrice w * l * c où c est le nombre de canal (3 pour une image couleur, 1 pour une image en noir et blanc), w et l la largeur et la longueur (resp.) de l'image en pixel
- un histogramme des couleurs RGB
- un histogramme HSV

Néanmoins, aucune de ces méthodes ne permet de représenter l'image de tel sorte qu'un algorithme comme K-Means obtienne de bons résultats, ces caractéristiques ne sont pas assez discriminantes.

Dans cette partie, nous utilisons l'algorithme ORB pour extraire des caractéristiques pour chaque transition de plan dans la vidéo. Cet algorithme calcule un vecteur de dimension 1 à partir d'une image; chaque élément i du vecteur est à un point (donc à deux coordonnées x_i et y_i) correspondant à

un point d'intérêt dans l'image.

6.3.1 Première expérimentation : une seule image par transition de plan

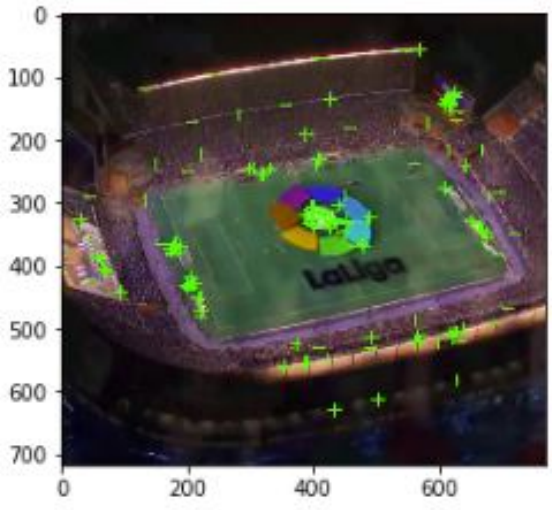


Figure 15: Caractéristiques extraites par ORB (en vert), pour une seule image par plan

Ici, nous ne récupérons les caractéristiques que d'une seule image par transition de plan. L'image pour laquelle nous extrayons les caractéristiques est la dernière frame de la séquence, car c'est la dernière image faisant la transition entre deux plans (et dans le cas des logos, c'est dans celle-ci que sa forme visuelle est la plus nette).

L'algorithme est le suivant :

```
- S = les transition de plans détectées
- Récupérer le frame à la fin de chaque transition de plan dans S
  - nous obtenons |S| frame
- Pour chaque frame, calculer ses features (orb ou akaze)
  - Nous obtenons |S| vecteurs
- Utiliser KMeans avec K=2 pour séparer les vecteurs en deux groupes
  - le groupe le plus petit est le groupe des logo
```

6.3.2 Deuxième expérimentation : une séquence d'images par transition de plan:

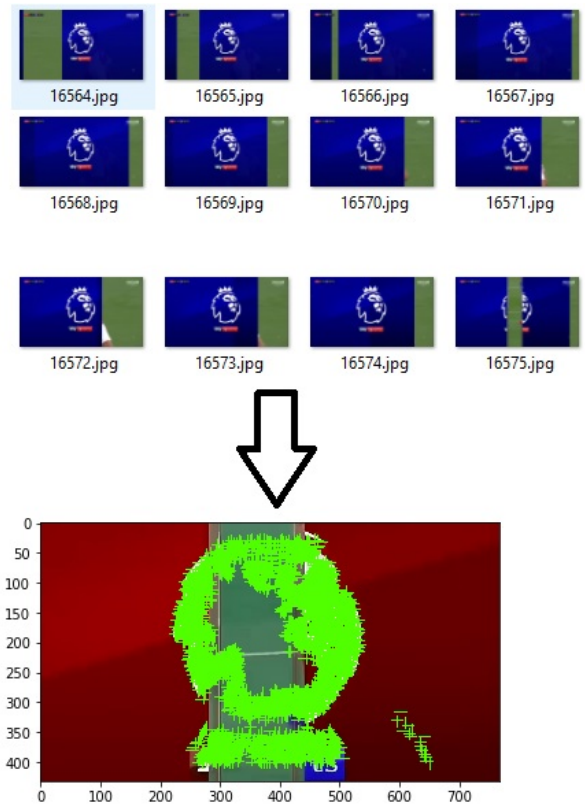


Figure 16: Caractéristiques extraites par ORB (en vert) pour un séquence logo (agrégat des caractéristiques de chaque image dans le plan)

L'approche précédente ne récupère qu'une seule image par plan.
Or, le visuel des logos est présent sur plusieurs frames consécutifs.
Dans cette approche, nous récupérons plusieurs images consécutives pour chaque plan, et nous extrayons les caractéristiques pour chaque frame (voir figure 16).

L'algorithme est le suivant :

```
- S = les transition de plans détectées
```

```
- Récupérer W frames pour chaque transition de plan dans S
  - nous obtenons |S| fenêtres de dimension W, où W est le nombre de frame
- Pour chaque fenêtre, calculer les features de chacun de ses frames (ORB)
  - Nous obtenons un vecteur de dimension S*W
- Utiliser KMeans avec K=2 pour séparer les vecteurs en deux groupes
  - le groupe le plus petit est le groupe des logos
```

6.4 Matching de contours

Dans cette approche, nous allons chercher les frames qui ont des formes en commun dans la vidéo.
En effet, d'après l'hypothèse III, il est fort probable que si un frame à l'instant t a beaucoup de formes en commun avec un frame à l'instant t', avec $2 < t' - t < 90$ (hypothèse V), alors il y a un logo à l'instant t et un logo à l'instant t', et un replay entre t et t'.

Dans cette méthode, nous utilisons là aussi le découpage en plan ([13]).
Cet algorithme est particulièrement efficace dans notre cas, car les replays provoquent un changement de plan, les logos seront donc toujours à la transition entre deux plans. Cette méthode va nous permettre réduire la zone de recherche des logos aux frames qui font la transition entre deux plans.

L'idée est de chercher, pour chaque transition de plan, s'il existe une autre transition de plan dans son voisinage tel qu'ils ont des contours en commun dans plusieurs de leurs frames.

6.4.1 Détection de contours

Avec des algorithmes comme le filtre de Canny il est possible de détecter les contours des objets dans une image. Cet algorithme applique une opération de convolution sur les pixels de l'image. La matrice résultant de l'application de cette convolution est l'image décrivant les contours dans l'image source. Cet algorithme est sensible au bruit dans l'image, c'est pourquoi il est nécessaire d'opérer un lissage (par exemple un filtrage Gaussien) avant de l'appliquer.
L'intérêt de la détection de contours est de réduire l'information à traiter dans une image.

Figure 17: Détection de contours par filtre de Canny

6.4.2 Logos et contours

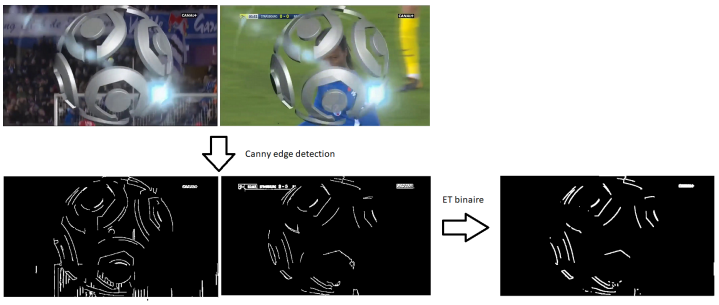


Figure 18: Comparaison des contours entre deux frames logos

Les logos sont des séquences d'images incrustées dans la vidéo. En général, chaque séquence d'image logo est la même au pixel près que les autres séquences logos. Nécessairement, les contours détectés par filtre de Canny seront les mêmes. Pour savoir si deux images ont les mêmes contours, il suffit d'appliquer un ET binaire entre leur contour respectif, et de calculer la longueur des segments dans l'image résultant de ce ET binaire.
La longueur des segments nous donne un score, et si ce score est supérieur à un certain seuil, alors nous considérons que les deux images sont des logos.

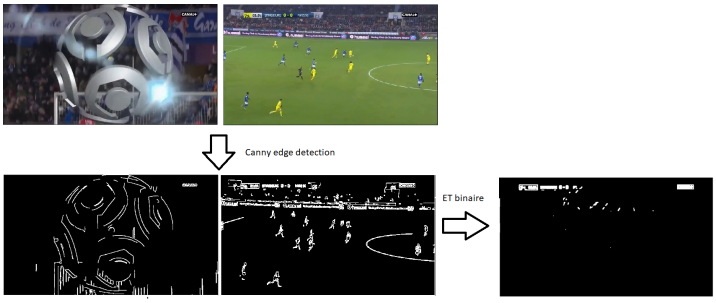


Figure 19: Comparaison des contours entre une frame logo et une frame non logo

6.4.3 Mosaïque de plan

Pour des raisons d'optimisation, il n'est pas viable de comparer chaque frame de chaque transition de plan avec chaque frame des transitions voisines.
C'est pourquoi, pour chaque transition de plan, deux images sont générées. Chaque image est de dimension (width * I) * (height*I) où I est le nombre de frames dans la transition. Chaque "case" dans l'image va correspondre aux contours d'une frame dans la transition de plan.
La première matrice a un décalage d'un frame par ligne, la seconde n'a pas de décalage.
Pour comparer deux shot (figure 20), il suffit d'appliquer un ET binaire entre les matrices des mosaïques, puis de calculer la longueur des contours dans cette matrice (les images sont des matrices).
L'idée derrière la mosaïque de plan est d'avoir les contours pour toutes les images de la transition de plan dans une même image.

6.4.4 Algorithme

```
- Pré traitement sur les images
  1. Redimensionner chaque image
  2. Cropper chaque image (afin de supprimer certains faux positifs)
  3. Détecter le contour (par filtre de Canny)
- Génération des mosaïques
- Pour chaque mosaïque de plan S_A :
  - Pour chaque mosaïque de plan S_B après S_A :
    1. Contour_commun = C_A & C_B
    2. Contours_diff = Détection des contours de Contour_commun
```

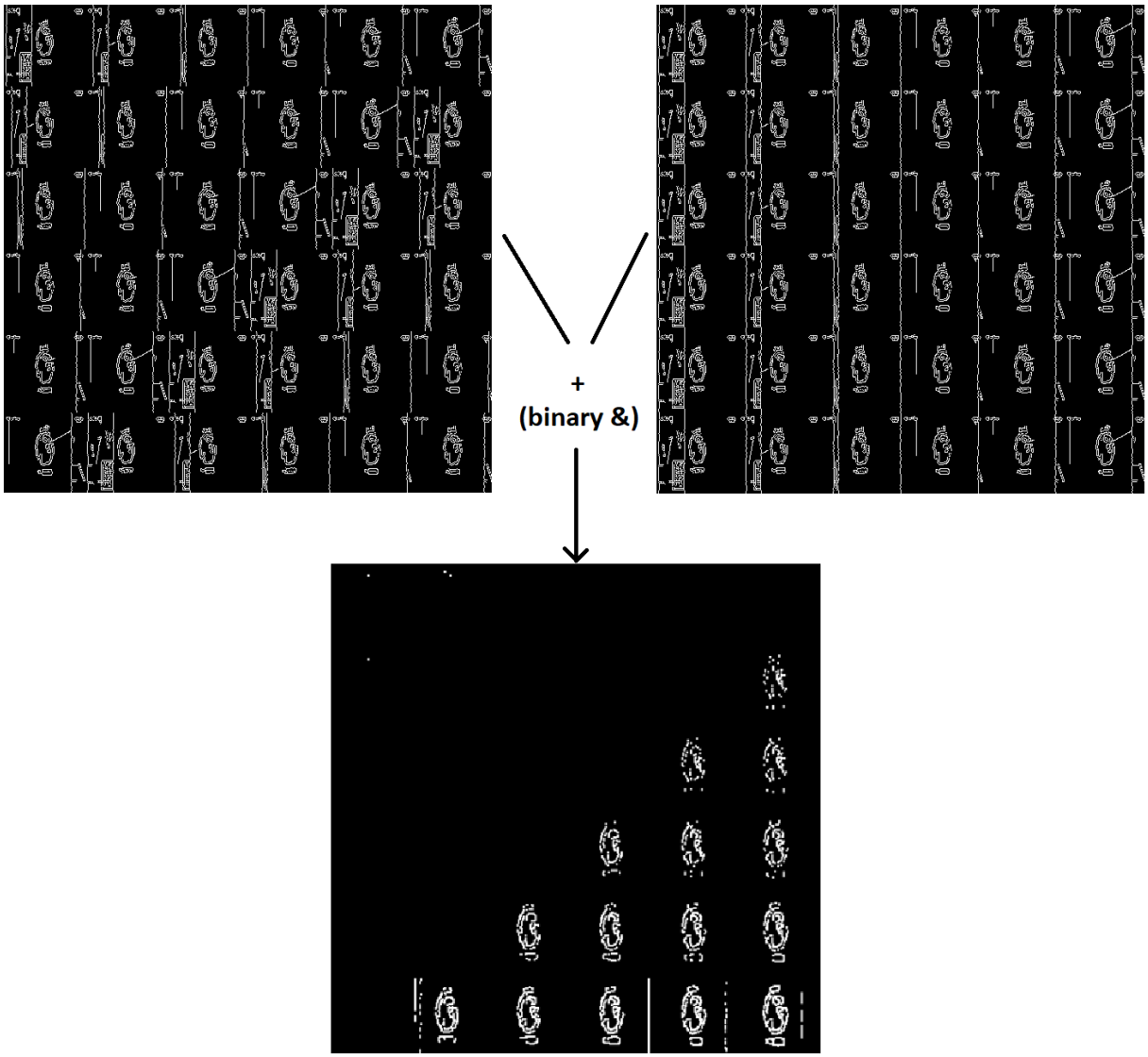


Figure 20: ET binaire (en bas) appliqué à une mosaïque de plan à l'instant t (à gauche) et à une mosaïque de plan à l'instant t' (à droite)

```
    3. Résultat = Ne garder que les contours qui sont assez longs (|contours| > K)
    4. Si Résultat > Seuil : alors S_A et S_B sont des logos potentiels
- Pour chaque logo potentiel LP :
  1. Le comparer avec les autres logo L' (même procédure qu'en 2)
  2. Si au moins 2 logos L'  match, alors LP est un logo
- Trouver les replays grâce aux logos
```

Notre algorithme est sensible au plan fixe et aux images avec beaucoup de bruits (ces images ont beaucoup de contours détectés par l'algorithme de détection de contours).
Beaucoup de ces faux-positifs peuvent être filtrés lors du pré-traitement sur les plans, notamment en rajoutant du blur, néanmoins, nous ne sommes pas parvenus à filtrer 100% des faux-positifs.

6.5 Convolution 3D sur des séquences d'images

Dans cette approche, nous allons implémenter une méthode similaire à [11]. L'idée va être d'entraîner un réseau à convolution avec une couche de convolution 3D.
Cette couche va prendre en entrée une séquence d'images et va prédire une classe logo ou non-logo.
Pour réduire la zone de recherche, seulement les séquences d'images entre deux plans (détectées avec [13]) seront traitées.
Lors de l'entraînement, le réseau va apprendre à partir de séquences d'images labellisées (logo/non-logo).
Comme pour [11], nous pensons que la convolution 3D va permettre au réseau d'apprendre l'apparence des logos mais aussi leur animation dans la vidéo.

6.6 Convolution 3D sur des séquences d'images de flux optique

Dans cette approche, nous allons entraîner un classifieur de séquences d'images de flux optique.
Nous pensons que, comme pour l'approche par convolution 3D sur les séquences d'images logos/non-logos, la convolution 3D va permettre d'apprendre le déplacement des objets dans le temps. La troisième dimension du réseau est la temporalité, et les deux autres sont la composante spatiale modélisée par les images de flux optiques.
Lors de l'entraînement, le réseau va apprendre à partir de séquences d'images

de flux optique labellisées (logo/non-logo).
Là aussi, nous n'allons traiter que les frames qui sont entre deux plans ([13]).

7 Collecte des données & entraînement

7.1 Ensemble de données

L'objectif de la collecte de données est d'obtenir les ensembles de données suivants :

- Dataset non logo
- Dataset logo

Chaque élément de l'ensemble de données contient :

- une séquence de 20 images labellisée logo ou non logo (pour un classifieur d'image à convolution 3D)
- une séquence de 19 images de flux optique labellisée logo ou non logo (pour un classifieur de flux optique à convolution 3D)

7.2 Collecte des données

L'approche par matching de contours convient tout à fait pour former notre ensemble de données de logo.

En effet, elle est :

- rapide : une vidéo au format 100x100 de 200000 frames va être traitée en moins de cinq minutes, et en moyenne une cinquentaine de logos (séquence d'images pendant laquelle un logo apparaît) sont extraits par vidéo
- précise : il est possible de modifier les paramètres pour que l'algorithme ne renvoie presque pas de faux-positifs ($>1\%$)

7.3 Obtention des images de flux optiques

Pour obtenir les images de flux optiques, nous allons utiliser notre algorithme de détection de logo par matching de contours afin d'obtenir les séquences d'images comprises entre deux plans.
L'algorithme va récupérer des séquences d'images logos/non-logos. Pour obtenir une séquence d'images de flux optique à partir de ces séquences, il suffit de calculer le flux optique entre une frame et la frame suivante; ainsi, nous obtenons une matrice de dimension 2, et donc une image.
Dans notre cas, les séquences d'images sont de taille 20, donc les séquences d'images de flux optique sont de taille 19.

7.4 Architecture du scrapper

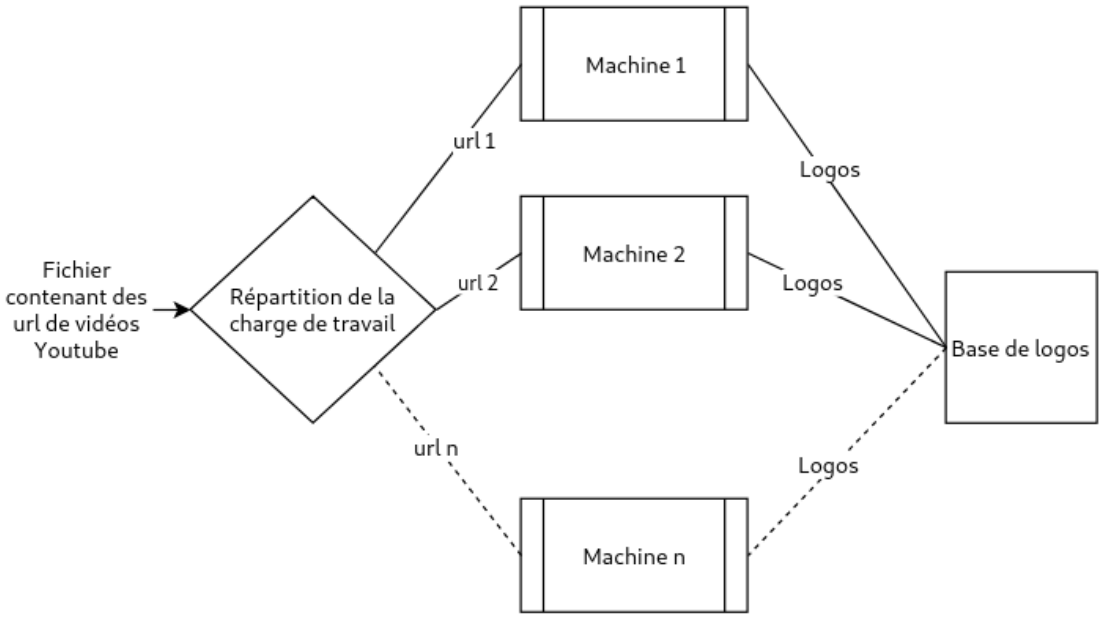


Figure 21: Architecture du scrapper de logos

Nous voulons une architecture de *scrapping* qui soit distribuée. Cette architecture est présentée dans la figure 21.
En effet, le traitement d'une vidéo prend en moyenne trente minutes (10 minutes pour trouver les replays, et 20 minutes pour l'import des fichiers etc..),

il est donc nécessaire de pouvoir traiter plusieurs vidéos en même temps.
Pour mettre en place cette architecture, nous avons créé une image *Docker* englobant le programme de détection de logos par matching de contours et qui importe les frames des séquences logo ainsi que leur flux optique sur la solution de stockage de GCP (storage).
L'image est stockée sur le Container Registry de GCP, et va être lancée dans un container.

7.4.1 Kubernetes

Managed pods

Revision	Name	Status	Restarts	Created on
1	replay-detector-69c6754b45-9k4wt	Running	0	Aug 26, 2019, 2:49:41 PM
1	replay-detector-69c6754b45-nchd8	Running	0	Aug 26, 2019, 3:02:42 PM
1	replay-detector-69c6754b45-p6fnf	Running	0	Aug 26, 2019, 3:02:42 PM
1	replay-detector-69c6754b45-2xcjl	Running	0	Aug 26, 2019, 3:02:42 PM
1	replay-detector-69c6754b45-sbwrr	Running	0	Aug 26, 2019, 3:02:57 PM

Figure 22: Cluster Kubernetes sur GCP

Un *cluster* Kubernetes 22 avec un *load balancer* va se charger d'ordonner les containers et de repartir la charge de travail entre les différents "pods" (unité exécutant un conteneur Docker).
Ainsi, il va être possible de traiter plusieurs vidéos en même temps car la charge de travail ne sera plus répartie sur une seule machine, mais sur plusieurs. Kubernetes est un outil efficace de mise à l'échelle.

7.5 Données collectées

Nous avons parsé plusieurs vidéos de matchs de football. Au total, 2611 séquences de logos et 2611 séquences non-logo ont été obtenues.
En tout, plus de 200000 images sont générées (8 go sur disque) :

- 2611 séquences de 20 images non-logos
- 2611 séquences de 20 images logos

- 2611 séquences de 19 images de flux optique non-logos
- 2611 séquences de 19 images de flux optique logos

7.6 Entraînement et architectures des modèles

L'entraînement va se faire en utilisant les instances de machines virtuelles préconfigurées pour l'apprentissage profond de Google Cloud. Le framework utilisé est Keras.
Pour entraîner les réseaux à convolution 3D (classifieur de séquences d'images et classifieur de séquences d'images de flux optique), nous allons reprendre l'architecture proposée par [11].

Le ratio ensemble d'entraînement/ensemble de validation est de 0,8.

7.6.1 Entraînement du classifieur de séquences d'images

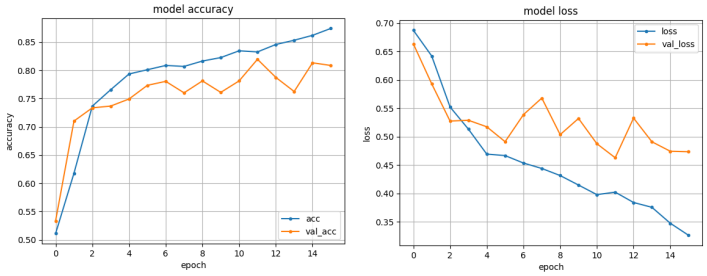


Figure 23: Entraînement du classifieur de séquence d'images

Nous voyons que le minima de la fonction de perte (val_{loss}) est atteint après 11 epoch (0,46), la précision (val_{acc}) sur l'ensemble de test après 11 epoch est de 82%.

7.6.2 Entraînement du classifieur de séquences d'images de flux optique

Le minima de la fonction de perte (val_{loss}) est atteint après 7 epoch (0,50), pour une précision (val_{acc}) de 77% sur l'ensemble de test.

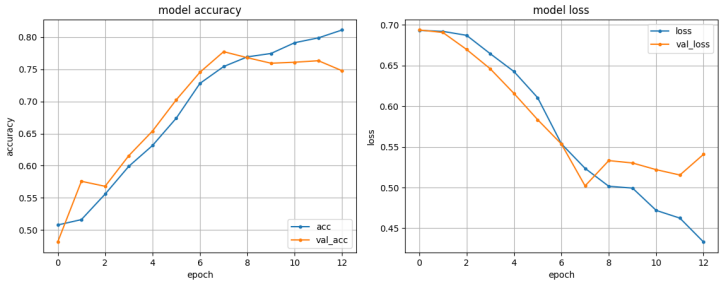


Figure 24: Entraînement du classifieur de séquence d'images de flux optique

7.7 Difficultés rencontrées

7.7.1 Avoir une machine assez puissante pour la convolution 3D

L'entraînement n'est pas possible sur une machine non dotée d'une carte graphique puissante. Pour ça, nous avons du utilisé une machine virtuelle fournie par Google. L'inconvénient de cette solution est le prix. En effet, celle-ci coûte près de 300€ par mois; ce qui ne nous a permis d'essayer autant de modèle qui nous aurions aimé, car l'entraînement est long et donc coûteux.

7.7.2 La gestion de l'ensemble de données

Pour 2611 exemples de séquences logo et 2611 exemples de séquences non-logo, la taille de l'ensemble de données atteint presque les 10go pour plus de 200000 images. Pour archiver ces images, nous avons opté pour la structure arborescente suivante : tag/type-image/id-séquence/images-séquences; où tag peut être logo ou non logo, type-image peut être flux optique ou normal (séquence de frame dans la vidéo), id-séquence un identifiant unique associé à chaque séquence et images-séquences les images de la séquence.
Manipuler autant de données est un problème complexe à bien des niveaux : l'importation/exportation des fichiers est longue (surtout le déplacement de ceux-ci entre la machine locale et la machine à distance), réorganiser la structure des dossiers est compliqué en cas d'erreur, les données erronnées (par exemple si le dossier représentant une séquence est vide suite à une défaillance du scrapper) sont difficilement détectables, l'extraction de l'archive contenant le dataset prend du temps, etc... À titre d'exemple, même sur la

machine très puissante fournie par Google, l'extraction de l'archive contenant toutes les images de notre ensemble de données prend plus d'une heure.

7.7.3 Adaptation des données pour le réseau

Les données sont des dossiers contenant des séquences d'images logo/non-logo.
Pour pouvoir donner en entrée ces données aux réseaux, nous avons besoin de transformer les séquences d'images en vecteur de dimension 4 (nombres d'images, largeur de chaque image, longueur de chaque image et nombre de canaux de chaque image) et de faire en sorte de pouvoir donner ces vecteurs en entrée du réseau par batch. La solution a ce problème a été de créer un *generator* personnalisé qui se charge d'aller lire les séquences dans le bon dossier et de ne charger en mémoire que le nombre nécessaire de séquences (taille du batch).

8 Résultats obtenus

Dans cette partie, nous allons évaluer chaque approche sur le même ensemble d'évalutation.

8.1 Procédure d'évaluation

L'ensemble d'évaluation est le suivant : une vidéo de Ligue 1 de 150k frames, une vidéo de Ligue 1 de 80k frames, une vidéo de Liga et enfin une vidéo de Premier League. Trois logos différents (Liga, Ligue 1 et Premier League) vont servir à évaluer nos approches.
Dans toutes les expérimentations, la vidéo est découpée en plans et nous récupérons les transitions de plan. L'évaluation porte sur le nombre de logos correctement détectés dans la vidéo.

8.2 Résultats pour ORB

Feature extracted by ORB on :	Football:PL 150k frames 48 logos	Football:Ligue 1 80k frames 16 logos	Football:Ligue 1 150k frames 34 logos	Football:Liga 300k frames 36 logos	Tennis:AusOpen 400k frames 32 logos
1 frame / shot	29.73% 68.75%	14.94% 40.63%	30.86% 73.53%	19.19% 52.78%	11.36% 46.88%
1 frame window / shot	55.10% 84.38%	31.51% 71.88%	34.67% 76.47%	34.62% 75.00%	14.29% 53.13%
Process time	2700 s	1600 s	2700 s	4000 s	5300 s

Score : Precision (left), Time (middle, in italic), Recall (right)

Figure 25: Résultats obtenus pour le clustering avec les caractéristiques extraites par ORB

Les résultats sont présentés dans la figure 25. Cette approche à tendance à considérer trop de séquences comme des séquences logos (faible précision sur toutes les vidéos).
Le gros inconvénient de cette méthode est le caractère arbitraire de la sélection du groupe de logo après le clustering.
En effet, nous faisons l'hypothèse qu'il y aura toujours plus de non-logos que de logos, ce qui n'est pas toujours vrai.
De plus, l'algorithme d'extraction de caractéristiques ne marche pas toujours très bien, ce qui fausse évidemment le clustering par la suite.

Néanmoins, il est intéressant de remarquer que les résultats avec la fenêtre de frame sont toujours meilleurs que les résultats avec un seul frame.

8.3 Résultats pour l'approche par matching de contours

Minimum score / Video width / Video height / Window size / Minimum logo segment length / Number of frame before removed / Number of frame after removed / Crop ratio	Football : Premier League 150k frames 48 logos		Football: Ligue 1 80k frames 16 logos		Football: Ligue 1 150k frames 34 logos		Football: Liga 300k frames 36 logos		Tennis: Australia Open 400k frames 32 logos	
With delete BG + dilate contour	*200 seconds*		*100 seconds*		*150 s*		*300 s*		*200 s*	
5000,100,100,5,11,2,2, 0.64	100.00%	100.00%	93.75%	93.75%	100.00%	100.00%	65.63%	65.63%	85.71%	88.24%
5000,100,100,5,11,0,4, 0.64	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	71.88%	71.88%	94.12%	94.12%
Without delete BG + Gauss contour / mosaic 10000,100,100,10,85,0,0, 0.81 (gaussian blur mosaic + contourDiff)	*340 s*		*160 s*		*150 s*		*290 s*		*350s*	
	97.96%	100.00%	50.00%	56.25%	100.00%	100.00%	62.50%	62.50%	94.12%	94.12%
With delete BG + gauss contour **Gauss contour / mosaic + group shot by seconds (rounded) + deleteBG = 0.8 sec** 10000,100,100,10,10,1,1, 0.81	*300 s*		*180 s*		*150 s*		*290 s*		*300 s*	
	95.92%	97.92%	55.56%	62.50%	97.14%	100.00%	62.86%	68.75%	97.06%	97.06%
With delete BG + gauss mosaic +dilate (2) contourDiff + deleteBG = 0.8 sec 10000,100,100,10,20,1,1, 0.81	*300 s*		*280 s*		*330 s*		*320 s*		*600 s*	
	97.96%	100.00%	100.00%	100.00%	91.89%	100.00%	75.00%	75.00%	97.06%	97.06%
With delete BG + gauss mosaic +dilate (2) contourDiff + deleteBG = 0.8 sec + group shot by half-second + saveWindowSize relative to fps	*430 s*		*260 s*		*640 s*		*260 s*		*1620 s*	
X*1000,100,100,X,20, 1,1, 0.81	97.96%	100.00%	100.00%	100.00%	91.89%	100.00%	75.00%	75.00%	91.89%	100.00%
X*1500,100,100,X,10, 1,1, 0.81	93.75%	93.75%	100.00%	100.00%	85.29%	86.11%	68.75%	68.75%	100.00%	100.00%

Score : Precision (left), Time (middle, in italic), Recall (right)

Figure 26: Résultats obtenus

Les résultats obtenus sont présentés en 26. Chaque ligne correspond à une configuration de paramètres différente, les paramètres sont :

- With delete BG : lors de la création de la mosaïque de plan, nous faisons pour chaque frame de la mosaïque la soustraction entre ce frame et les 3 frames au début du plan. Nous faisons ceci afin de supprimer les pixels fixes dans le plan qui sont probablement du bruit, comme par exemple, les lignes du terrain, les cages du gardien ou le logo de la chaine. En effet, les frames avec des pixels fixes vont avoir beaucoup de contours en commun et risque donc d'être détectées comme logo sans l'être.

- dilate contour : les pixels des contours sont dilatés (de 2 pixels)

- gauss contour/mosaic : un filtre gaussien est appliqué sur la matrice résultant du & binaire entre deux mosaïques de contours de plan.

- saveWindowSize relative to fps : la taille des mosaïques est relative au nombre d'images par seconde dans la vidéo

Après avoir fait plusieurs essais avec des paramètres différents à chaque fois, nous ne sommes pas parvenus à trouver une combinaison telle qu'il n'y ait aucun faux positif et aucun faux négatif.

Concernant le temps d'exécution, celui-ci est relié presque entièrement à la taille de la vidéo donnée en entrée, ainsi qu'à la taille des mosaïques. Il reste néanmoins faible, et cette approche peut être utilisée en temps réel.

L'inconvénient de cette méthode est qu'elle n'est pas capable de détecter les replays dans les cas suivants :

- s'il n'y a pas de logo pour les replays (simple fondu)

- si les logos au début et à la fin des replays ne sont pas les mêmes

- s'il y a des logos au début des replays, mais pas de logo à la fin des replays (un simple fondu remplace le logo).

8.4 Résultats pour l'approche par apprentissage profond

Model	Football:PL 150k frames		Football:Ligue 1 80k frames		Football:Ligue 1 150k frames		Football:Liga 300k frames	
Convolution3D	65.6%	40.1%	20,00%	20.7%	33.3%	14.6%	4.3%	6.1%
Convolution3D on optical flow images	17.1%	36.2%	8.3%	31.0%	48.1%	11.3%	88.8%	13.3%

Score : Precision (left), Recall (right)

Figure 27: Résultats obtenus par les réseaux

Les résultats obtenus par les réseaux sont présentés en 27. Ils sont moins bons que ce que nous espérions.
Le réseau à convolution 3D sur des séquences d'images obtient des résultats satisfaisant sur la vidéo de Premier League (PL) avec 65% de précision et 40% de rappel; mais de mauvais résultats sur les autres vidéos.
Le réseau à convolution 3D sur des séquences d'images de flux optique obtient de mauvais résultats sur toutes les vidéos; le résultat obtenu pour la vidéo de Liga est néanmoins intéressant. En effet, avec une précision de 88%, il semblerait que le réseau ait commencé à apprendre à reconnaître les séquences non-logo.

Les possibles raisons pouvant expliquer ces mauvais résultats sont les suivantes :

- les données d'entraînement ne sont pas assez représentatives (I) : l'ensemble d'entraînement est constitué de 50% de séquences logos et de 50% de séquences non logos alors que dans les faits, seulement 5% en moyenne des séquences dans les matchs de football sont des séquences logos. Nous avons procédé ainsi pour avoir un ensemble de données plus petit que si nous avions gardé toutes les séquences non-logos (le dataset aurait été près de 5 fois plus volumineux)

- les données d'entraînement ne sont pas assez représentatives (II) : la majorité des séquences logos (~90%) que nous avons dans l'ensemble d'entraînement sont des logos de Liga. Ceci est du au fait que la plupart des vidéos que nous avons pu récupérer sont des vidéos de Liga.

- les données d'entraînement ne sont pas d'assez bonne qualité : certaines vidéos sont de mauvaise qualité et même dans certains cas les logos sont floutés

- le scrapper n'est pas parfait : l'annotation logo/non-logo des séquences est automatique. Il est probable qu'une quantité non négligeable de séquences soient incorrectement annotées.

- l'ensemble de données est trop petit : nous entraînons les réseaux "de zéro", il est probable que 2000 séquences logos et 2000 séquences non-

logos ne suffisent pas. Nous avons fait le choix de garder l'ensemble d'entraînement à 4000 exemples car un dataset plus gros aurait accru le temps d'entraînement (et donc le coût de la machine distante).

9 Conclusion

L'approche la plus efficace pour la détection de replays est l'approche par détection de contours. En effet, celle-ci obtient les meilleurs résultats sur toutes les vidéos d'évaluation. De plus, elle est la plus rapide et n'est pas coûteuse en terme de puissance de calcul : les calculs se font sur le processeur (et non pas sur carte graphique comme les approches par apprentissage profond) et il faut seulement une vingtaine de minutes pour analyser 200000 frames sur un processeur Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz. Il est donc possible d'utiliser cette méthode en temps réel.
Néanmoins, l'approche par détection de contours présente des inconvénients. Le premier est qu'elle est trop sensible aux bruits dans les vidéos. Le second est la trop forte dépendance aux hyper-paramètres (taille en pixel des images, nombre de frames dans la mosaïque, paramètres du floutage gaussien, ...). Le dernier est que cette approche ne parviendra jamais à détecter certaines séquences logos, comme par exemple les logos qui ne sont pas les mêmes au début d'un replay et à la fin d'un replay.
Ces inconvénients s'expliquent par le fait que cette méthode n'est pas *adaptative* car elle n'apprend pas ce qu'est une séquence logo.
Même si les résultats obtenus par les réseaux ne sont pas excellents, ils ne sont pas affectés par les trois inconvénients cités plus haut.
Les résultats obtenus pour la vidéo de Premier League nous laissent penser que le réseau *a commencé* à apprendre la forme des séquences non logos dans les vidéos.
Il serait intéressant de reprendre cette étude, avec cette fois-ci un ensemble de données à la fois plus conséquent et plus propre (avec de l'annotation humaine).
Une autre piste à étudier est le *transfer learning*. Dans notre cas, nous pouvons utiliser les poids appris par les auteurs du réseau C3D sur le dataset UCF101 [11]. Il serait aussi intéressant d'étudier d'autres modèles comme celui mélant LSTM et CNN [12].
Pour conclure sur cette recherche, nous dirons que l'analyse vidéo n'est pas chose aisée. Entraîner un classifieur pour cette tâche demande des machines très performantes ainsi qu'un ensemble de données très riches.
Il est important de préciser que la complexité croit de manière exponentielle avec la taille de l'ensemble de données. Par exemple, changer la convention de nommage des exemples d'entraînement n'est pas le même problème si le dataset fait 500mo ou 10go. Une grande partie de notre travail a consisté à ré-organiser l'ensemble de données et à retrouver des exemples défaillants parmi la grande quantité d'images.

10 Glossaire

- Batch : un groupe d'échantillon d'entraînement; permet de traiter l'ensemble d'apprentissage plus vite qu'en ne traitant qu'un seul échantillon à la fois.

- Clustering : procédé permettant de regrouper des éléments

- Histogramme : représentation d'une image en fonction de ses canaux de couleurs (rouge, vert, bleu)

- Frame : une image à l'instant t d'une vidéo

- Shot : un plan

- FPS : frame per second / image par seconde

- Cropper : sélectionner une partie continue des pixels l'image

- RNN : Recurrent Neural Networks, ou réseaux de neurones récurrents (RNR) en français

- CNN : Convolutional Neural Network, réseaux de neurones convolutifs (RNC) en français

- LSTM : Long Short Term Memory

- Cluster : un groupe; en apprentissage automatique, signifie un groupe d'objet partageant des caractèristiques similaires; en infrastructure réseau, signifie un ensemble de machines

- Scrapping : technique d'extraction de contenu de sites Web; dans notre cas, nous récupérons des vidéos sur Youtube

- Transfer learning: procédé d'apprentissage automatique consistant à réutiliser les poids appris par un réseau dans un autre réseau

- Slow motion : nous parlons de slow-motion quand les objets se déplacent moins vite que la normale dans une vidéo

- Docker : solution permettant de lancer une application dans son propre environnement (image) à partir d'une suite d'instruction (Dockerfile)

- GCP : Google Cloud Platform

- Load balancer : technique permettant de répartir la charge de travail parmi un ensemble de machines

- Framework : ensemble de librairies facilitant la tâche du programmeur

- Dataset : anglicisme fréquemment utilisé pour parler d'un ensemble de données

11 Table des figures

Press. Chapitre 9. Figure 9.1

6 Goodfellow, I., Bengio, Y., & Courville, A., Deep Learning (2016), : MIT Press. Chapitre 9. Figure 9.9

9 Tran, D., Bourdev, L., Fergus, R., Torresani, L., & Paluri, M., Learning spatiotemporal features with 3d convolutional networks, 2015 IEEE International Conference on Computer Vision (ICCV), (), (2015). `http://dx.doi.org/10.1109/iccv.2015.510`. Figure 1

13 Ng, J. Y., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., & Toderici, G., Beyond short snippets: deep networks for video classification, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (), (2015). `http://dx.doi.org/10.1109/cvpr.2015.7299101`. Table 7

References

[1] Hao Pan, Baoxin Li, and Sezan. Automatic detection of replay segments in broadcast sports programs by detection of logos in scene transitions. *IIEEE International Conference on Acoustics Speech and Signal Processing*, 2002.

[2] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. *2011 International Conference on Computer Vision*, Nov 2011.

[3] Ling-Yu Duan, Min Xu, Qi Tian, and Chang-Sheng Xu. Mean shift based video segment representation and applications to replay detection. *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2004.

[4] Wen-Sheng Chu, Yale Song, and Alejandro Jaimes. Video co-summarization: Video summarization by visual co-occurrence. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015.

[5] Ali Javed, Aun Irtaza, Yasmeen Khaliq, Hafiz Malik, and Muhammad Tariq Mahmood. Replay and key-events detection for sports video

summarization using confined elliptical local ternary patterns and extreme learning machine. *Applied Intelligence*, 49(8):2899–2917, Feb 2019.

[6] H. Pan, P. van Beek, and M.I. Sezan. Detection of slow-motion replay segments in sports video for highlights generation. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, page nil, - nil.

[7] Wei Xu and Yang Yi. A robust replay detection algorithm for soccer video. *IEEE Signal Processing Letters*, 18(9):509–512, 2011.

[8] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1753–1760. Curran Associates, Inc., 2009.

[9] A Raventós, R Quijada, Luis Torres, and Francesc Tarrés. Automatic summarization of soccer highlights using audio-visual descriptors. *SpringerPlus*, 4(1), Jun 2015.

[10] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199, 2014.

[11] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015.

[12] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015.

[13] Wael Abd-Almageed. Online, simultaneous shot boundary detection and key frame extraction for sports videos using rank tracing. *2008 15th IEEE International Conference on Image Processing*, 2008.

[14] Heng Wang, Alexander Klaser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *CVPR 2011*, page nil, 6 2011.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.