

# Mémoire

Billal Boudjoghra

September 8, 2019

## Contents

<b>1</b>	<b>TODO plus détaillé la partie détection de contours</b>	<b>4</b>
<b>2</b>	<b>Sportagraph</b>	<b>4</b>
<b>3</b>	<b>Introduction</b>	<b>6</b>
<b>4</b>	<b>Analyse d'image : état de l'art concernant la détection de replay</b>	<b>8</b>
4.1	A Robust Replay Detection . . . . .	8
4.1.1	Calcul du score de la luminance et filtrage des frames .	8
4.1.2	Recherche du logo template parmi les frames filtrées .	8
4.1.3	Recherche des logo . . . . .	9
4.1.4	Recherche des replays . . . . .	9
4.1.5	Résultats des auteurs . . . . .	9
4.2	Mean Shift Based video Segment Representation And Applications To Replay Detection . . . . .	9
4.2.1	Algorithme . . . . .	10
4.2.2	Résultats . . . . .	10
<b>5</b>	<b>Apprentissage automatique : les bases théoriques</b>	<b>12</b>
5.1	Réseaux de neurones récurrents (RNN) . . . . .	12
5.2	LSTM . . . . .	12
5.3	CNN . . . . .	13
5.3.1	Intéraction parcimonieuse . . . . .	14
5.3.2	Partage de paramètres . . . . .	15
5.3.3	Représentations équivariantes . . . . .	15
5.3.4	Pooling . . . . .	16

<b>6</b>	<b>Apprentissage profond : état de l'art pour la reconnaissance d'action dans les vidéos</b>	<b>18</b>
6.1	Two-Stream Convolutional Networks for Action Recognition in Videos . . . . .	18
6.1.1	Classifieur spatial . . . . .	18
6.1.2	Classifieur temporel . . . . .	19
6.1.3	Méthode d'évaluation et résultats obtenus . . . . .	19
6.2	Learning Spatiotemporal Features with 3D Convolutional Networks . . . . .	21
6.2.1	C3D: Convolution et pooling 3D . . . . .	21
6.2.2	Architecture et entraînement du réseau . . . . .	22
6.2.3	Résultat pour la tâche de reconnaissance d'action . . .	23
6.2.4	Conclusion . . . . .	24
6.3	Beyond Short Snippets: Deep Networks for Video Classification	24
6.3.1	Approche . . . . .	24
6.3.2	Architecture . . . . .	25
6.3.3	Utilisation du flux optique . . . . .	26
6.3.4	Résultat obtenu . . . . .	26
<b>7</b>	<b>Les approches proposées</b>	<b>28</b>
7.1	Détection des plans . . . . .	29
7.1.1	Online, Simultaneous Shot Boundary Detection And Key Frame Extraction For Sports Videos Using Rank Tracing . . . . .	29
7.2	Analyse d'image - Première approche : ORB . . . . .	30
7.2.1	Première expérimentation : 1 frame par shot . . . . .	31
7.2.2	Deuxième expérimentation : 1 fenêtre de frame par shot:	32
7.3	Analyse d'image - Seconde approche : matching de contours .	34
7.3.1	Détection de contours . . . . .	34
7.3.2	Logos et contours . . . . .	35
7.3.3	Mosaique de plan . . . . .	35
7.3.4	Algorithme . . . . .	37
7.4	Apprentissage profond - Première approche : convolution 3D	38
7.5	Flux optique . . . . .	38
7.6	Apprentissage profond - Deuxième approche : convolution 2D sur des images de flux optique . . . . .	39
7.7	Apprentissage profond - Troisième approche : convolution 3D sur des images de flux optique . . . . .	39

<b>8</b>	<b>Collecte des données &amp; entraînement</b>	<b>39</b>
8.1	Datasets . . . . .	39
8.2	Collecte des données . . . . .	40
8.3	Obtention des images de flux optiques . . . . .	40
8.3.1	Image de flux optique : 2L canaux . . . . .	41
8.3.2	Image de flux optique : somme des fluxs . . . . .	41
8.4	Architecture du scrapper . . . . .	41
8.5	Données collectées . . . . .	44
8.6	Entraînement et architectures des modèles . . . . .	44
8.7	Difficultés rencontrées . . . . .	44
8.7.1	Adaptation des données pour le réseau . . . . .	44
8.7.2	Avoir une machine assez puissante pour la convolution 3D . . . . .	44
8.7.3	La gestion de l'ensemble de données . . . . .	45
<b>9</b>	<b>Résultats obtenus</b>	<b>46</b>
9.1	Procédure d'évaluation . . . . .	46
9.2	Résultats pour ORB . . . . .	46
9.3	Résultats pour la détection par luminance . . . . .	47
9.4	Résultats pour l'approche par matching de contours . . . . .	48
<b>10</b>	<b>Appendice</b>	<b>50</b>
<b>11</b>	<b>Table des figures</b>	<b>51</b>

# 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 nous 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 permis 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 cet étude, nous allons nous intéresser 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 double.

Dans un premier temps, nous mettons en place un système de détection de replays en utilisant uniquement les méthodes "classiques" tels que ORB ([2]) ou la détection de contours.

Ensuite, nous comparons plusieurs approches d'apprentissage automatique pour la tâche de détection dans les vidéos.

Enfin, nous comparons l'efficacité des méthodes de vision par ordinateur "classiques" aux méthodes par apprentissage profond (deep learning).

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

L'un des objectifs de notre recherche est de mettre en place un modèle de détection de replays n'utilisant que des techniques ne faisant pas intervenir d'apprentissage automatique.

Ce modèle servira de base de comparaison avec le modèle par apprentissage automatique que nous proposerons par la suite.

Afin d'avoir le meilleur modèle, nous avons comparé plusieurs approches.

La détection de replays est un domaine de recherche à part entière et les articles sont abondants ([1, 3, 4, 5] ...).

#### 3.1 A Robust Replay Detection

Cette approche [6] 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.

##### 3.1.1 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.

##### 3.1.2 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 celui-ci est celui qui a le centre de 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 1.

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

Figure 1: Formule de la distance entre deux frames dans le cluster

### 3.1.3 Recherche des logos

Une fois que le logo template est déterminé, chaque logo trouvé en précédemment va être comparé avec le logo template.

La mesure de comparaison est la distance (figure 1) 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.

### 3.1.4 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).

### 3.1.5 Résultats des auteurs

## 3.2 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 ([7]), puis de se servir de cette base de données pour trouver les logo au début à la fin de replays.

### 3.2.1 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 .

### 3.2.2 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 2: Performance sur la tâche de détection de replay :

Les résultats obtenus par les auteurs sont présentés dans la Figure 2. 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

## 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  $[e_1, e_2, \dots, e_n]$  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  $[s_1, s_2, \dots, s_n]$ .

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.

### 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

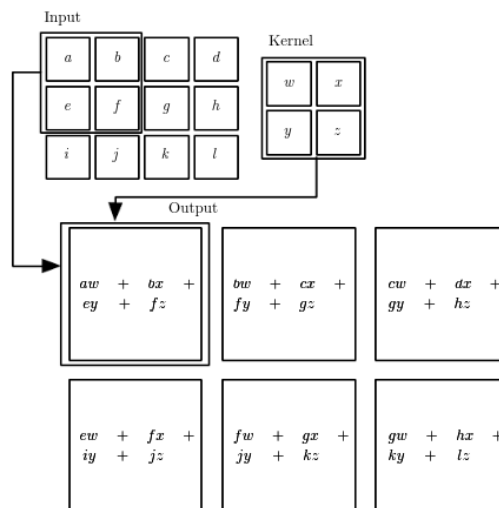


Figure 3: Opération de convolution

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'**interaction 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 fonction de **pooling**.

#### 4.3.1 Interaction parcimonieuse

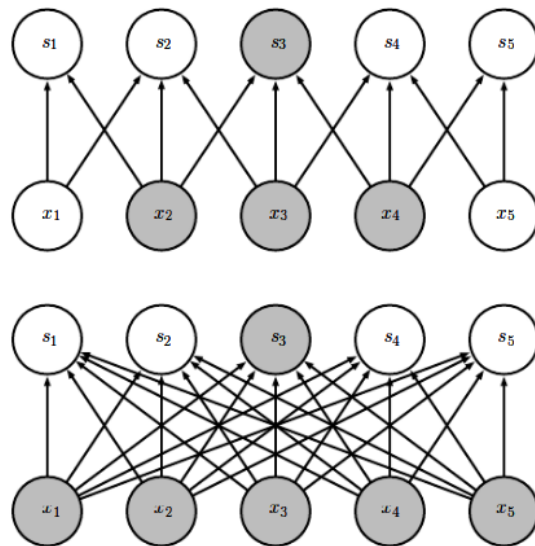


Figure 4: Interaction parcimonieuse (en haut), interaction non parcimonieuse (en bas)

À la différence des réseaux classiques où toutes les sorties interagissent avec toutes les entrées, les réseaux à convolution ont des **interactions parcimonieuses**.

C'est à dire que la taille du noyau (donc de l'interaction 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'interaction parcimonieuse car il faudra une connection 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'interaction parcimonieuse permet aux CNN de stocker moins de paramètres que les autres types de réseau.

Par conséquent, ils ont donc 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 signifie que le déplacement des pixels n'a pas d'influence sur le réseau.

#### 4.3.4 Pooling

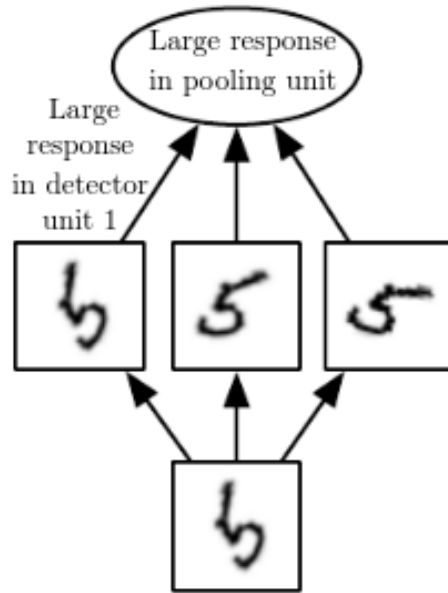


Figure 5: Pooling & invariance

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 [8]. 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

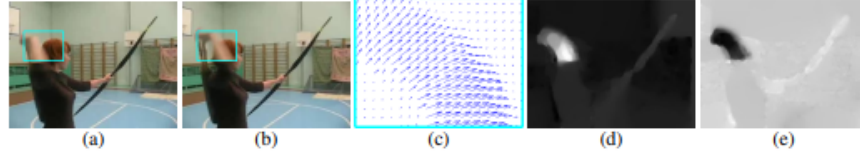
Ce réseau a une architecture de classifieur d'image classique.

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 6: 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 la méthode "optical flow" pour détecter le mouvement des objets entre des frames consécutifs.

Ils définissent aussi un hyperparamètre  $L$  qui définit la distance maximum entre deux frames pour lequel 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  $t+5$ .

Ainsi, des images représentant le flux optique entre les différents frames de la vidéo vont être générées.

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

### 5.1.3 Méthode d'évaluation et résultats obtenus

Le classifieur spatial est pré-entraîné avec ImageNet, tandis que le temporel est entraîné de zéro (car il n'y a pas de réseau déjà entraîné pour cette tâche). Les dataset utilisés pour l'entraînement et l'évaluation sont UCF-101 et

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]	<b>87.9%</b>	61.1%
IDT with stacked Fisher encoding [21] (based on Deep Fisher Net [23])	-	<b>66.8%</b>
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)	<b>88.0%</b>	<b>59.4%</b>

Figure 7: Résultats obtenus par l’approche Two-stream model

HMDB-51, contenant à eux deux près de 20000 vidéos annotées.

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

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

Les résultats (figure 7) 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% (HMDB-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 renforcé dans l’hypothèse qu’il est nécessaire d’étudier une vidéo non pas comme une suite d’images indépendantes, mais comme

une suite de séquence avec un lien entre chaque élément de la séquence. La temporalité a une très grande importance pour l'analyse de vidéos.

## 5.2 Learning Spatiotemporal Features with 3D Convolutional Networks

Dans cet article [9], 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)

### 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.

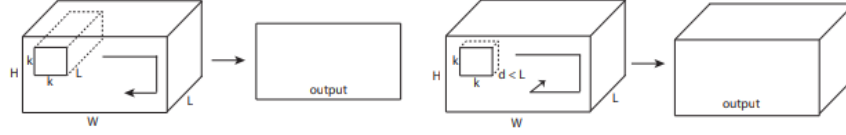


Figure 8: Convolution 2D sur une séquence d'images (gauche), convolution 3D sur une séquence d'images (droite)

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.

### 5.2.2 Architecture et entraînement du réseau

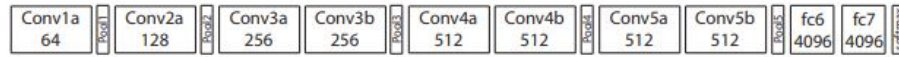


Figure 9: Architecture C3D

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 passés 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

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
<b>C3D</b> (1 net) + linear SVM	82.3
<b>C3D</b> (3 nets) + linear SVM	<b>85.2</b>
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
<b>C3D</b> (3 nets) + iDT + linear SVM	<b>90.4</b>

Figure 10: Résultats pour l'approche Learning Spatiotemporal Features with 3D Convolutional Networks (C3D) comparés à d'autres approches état de l'art

Ces résultats (figure 10) 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 sur plusieurs tâches d'analyse de vidéos.

De plus, cette approche est très élégante car elle fonctionne sans ajout artificiel de caractéristiques (comme le flux optique), c'est le réseau qui va se charger d'apprendre ce dont il a besoin pour apprendre la temporalité.

### 5.3 Beyond Short Snippets: Deep Networks for Video Classification

Dans cet article [10], 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.

Les 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 n'est pas possible avec la puissance de calcul actuelle.

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 [8]) est calculé entre les frames adjacents.

Ainsi, l'information temporelle et l'information spatiale 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 vidéo comme des séquences d'images et va apprendre à prédire la classe de la vidéo en fonction de ces séquences.

### 5.3.2 Architecture

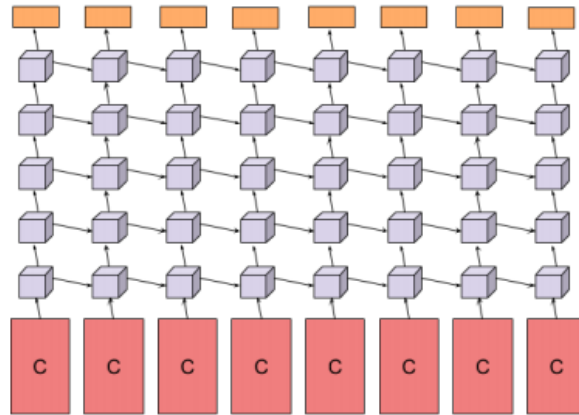


Figure 11: Architecture du LSTM

Comme pour [8], un réseau à convolution va être utilisé 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 11.

La sortie du CNN est traitée 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 [8], les images de flux vont être pré-calculées et servir lors de l'entraînement du CNN.

### 5.3.4 Résultat obtenu

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)	<b>88.2</b>
LSTM with 30 Frame Unroll (Optical Flow + Image Frames)	<b>88.6</b>

Figure 12: Résultat obtenu pour l'approche combinant CNN et LSTM (LSTM with 30 Frame Unroll)

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 [8]).

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 [8] 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.

## 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)
- un replay a un logo de fin (II)
- les logos de début et de fin sont les mêmes (III)
- 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)

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 [9])

- 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

Les approches que nous proposons itèrent sur tous les frames de la vidéo, à la recherche des logo pouvant se trouver au début et à la fin des replays.

Si nous faisons l'hypothèse qu'un replay entraînera toujours un changement de plan, alors au lieu de rechercher les logos parmi tous les frames de la vidéo, nous pouvons réduire la recherche à tous les frames qui sont entre deux plans.

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 [11].

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  $M$ .  $M = U W V$ , 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 poids 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  $s/S_1 > \text{threshold}$ . 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 inférieure à la fenêtre précédente, alors le contenu visuel se stabilise. S'il est de 1, alors c'est stable.

Le début d'un frame est celui qui maximise le rang parmi les fenêtres environnantes.

Cette méthode pour trouver les plans dans une vidéo est très efficace, et nous servira tout au long de 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).

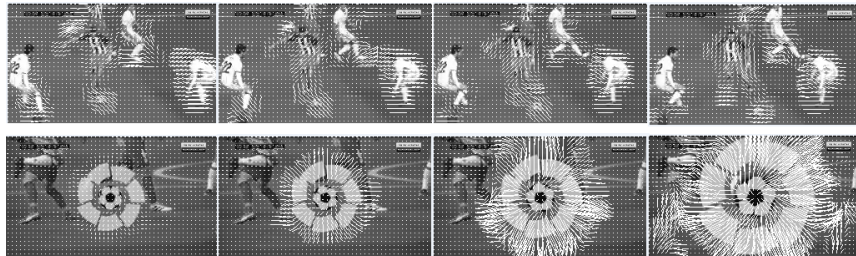
Il est impensable d'utiliser un algorithme en  $O(N^2)$ , par exemple en comparant tous les frames entre eux, avec un  $N$  aussi grand.

Après avoir segmenter la vidéo en plan, nous obtenons un  $N'$  au 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 champs 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 abscisses

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 [12], nous allons traquer le déplacement des objets grâce au flux optique.

Il existe deux types d'algorithmes de calcul du flux optique :

- sparse/creux : seulement les déplacements de certains points d'intérêt vont être traqués (méthode de Lucas-Kanade)
- dense : le déplacement de tous les points dans l'image vont être traqués (algorithme de Gunner Farneback)

Le calcul du flux optique dense est plus coûteux 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.

Il existe plusieurs manières de représenter 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 vectoriser l'image de tel sorte qu'un algorithme comme K-Means obtienne de bons résultats, ces caractéristiques ne sont pas assez discriminantes.

De plus dans notre cas, nous souhaitons regrouper non pas des frames (images) mais des plans (séquence d'images).

OpenCV implémente l'algorithme KMeans. Celui-ci 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 très rapide et assez efficace dans la plupart des cas.

C'est l'un des algorithmes de clusterisation les plus utilisés.

Pour fonctionner, KMeans a besoin que les objets que l'on souhaite séparer en groupe soit 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.

Dans cette partie, nous utilisons l'algorithme ORB pour extraire des caractéristiques à partir pour chaque plan dans la vidéo.

### 6.3.1 Première expérimentation : 1 frame par shot

Ici, nous ne récupérons les caractéristiques que d'une seule image par plan.

L'algorithme est le suivant :

- Récupérer le frame à la fin de chaque shot
  - 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



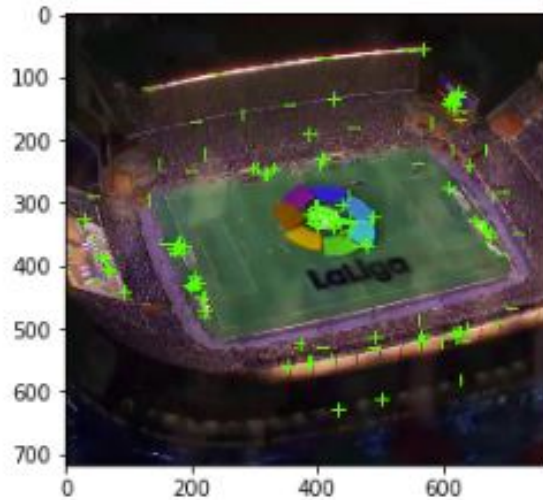


Figure 13: Caractéristiques extraites par ORB (en vert), pour une seule image par plan

### 6.3.2 Deuxième expérimentation : 1 fenêtre de frame par shot:

L'approche précédente ne récupère qu'une seule image par plan.

Or, les logos sont présents sur plusieurs frames consécutifs.

Dans cette approche, nous récupérerons plusieurs images consécutives pour chaque plan, et nous extrayons les caractéristiques pour chaque frame (voir figure 14).

L'algorithme est le suivant :

- Récupérer  $W$  frames pour chaque shot, les regrouper en une fenêtre
  - 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 logo

## 6.4 Matching de contours

Dans cette approche, nous allons chercher les frames qui ont des formes en commun dans la vidéo.

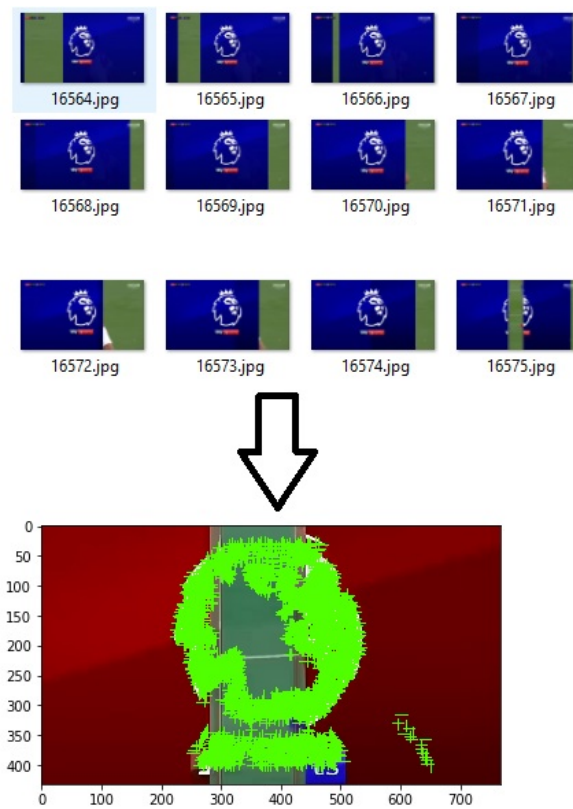


Figure 14: Caractéristiques extraites par ORB (en vert) pour un plan (aggrégat des caractéristiques de chaque image dans le plan)

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 ([11]). Cette 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 plan.

La méthode de [11] va nous permettre réduire la zone de recherche des replays aux frames qui font la transition entre deux plans.

L'idée est de chercher, pour chaque transition de plan, si 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



Figure 15: Détection de contours par filtre de Canny

Avec des algorithmes comme le filtre de Canny il est possible de détecter les contours des objets dans une image. Cette 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. Cette 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.

#### 6.4.2 Logos et contours

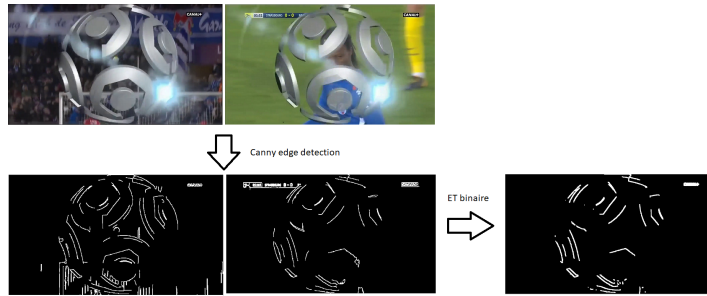


Figure 16: Comparaison des contours entre deux frames logos

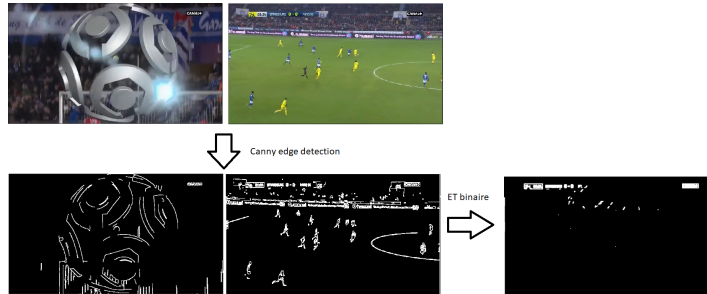


Figure 17: Comparaison des contours entre deux frames non-logo

Les logos sont des séquences d'images incrustés dans la vidéo. En général, chaque séquence d'image est la même au pixel près que les autres. Nécessairement, les contours détectés par filtre de Canny seront les mêmes.

#### 6.4.3 Mosaique 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.

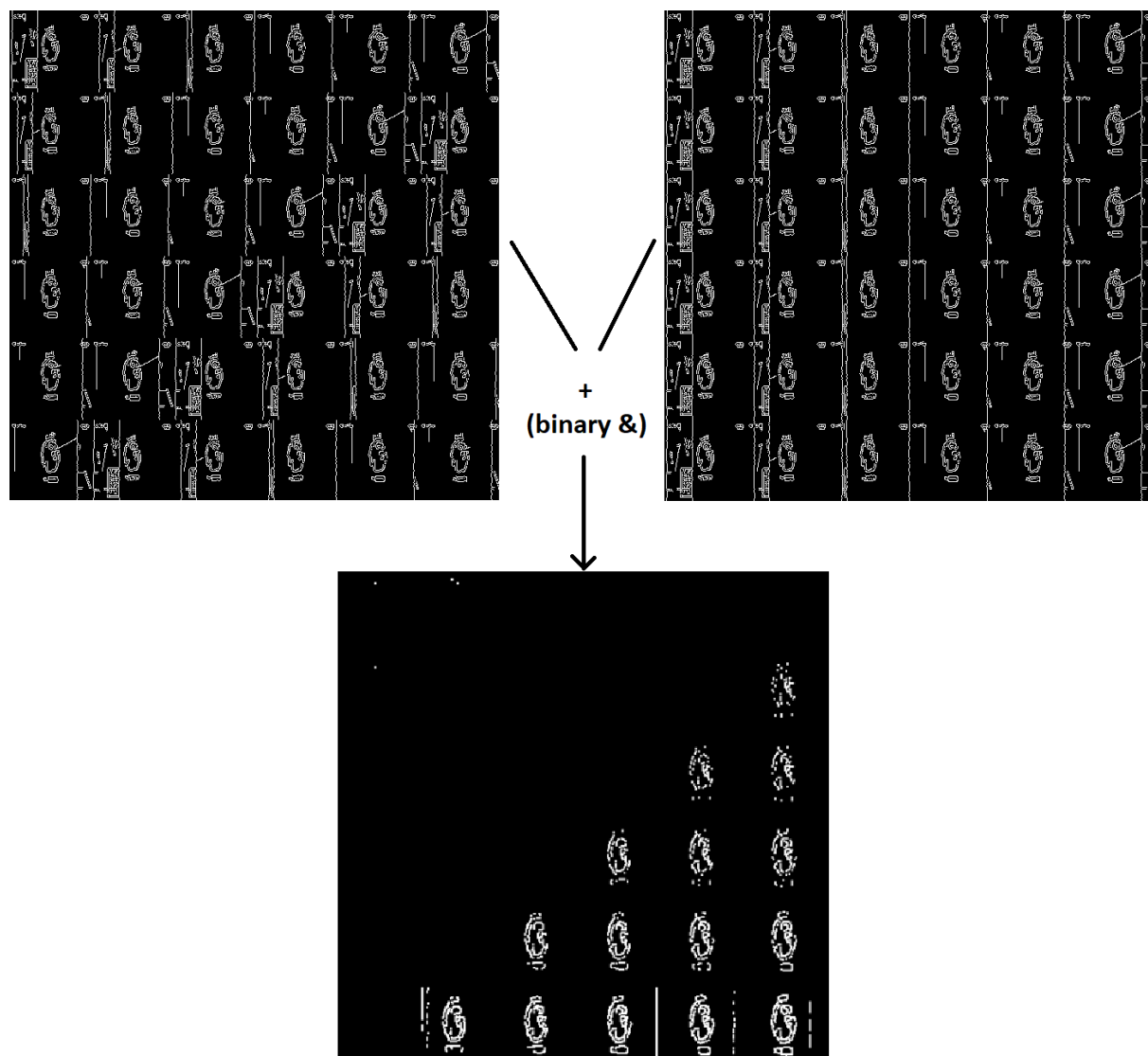


Figure 18: ET binaire (en bas) appliqué à une mosaïque de plan à l'instant t (à gauche) et à une mosaïque de plan à l'instant t' (à droite)

C'est pourquoi, pour chaque transition de plan, deux images sont générées.

Chaque image est de dimension  $I^2 * width * height$  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 18), 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).

#### 6.4.4 Algorithme

- Pré traitement sur les shots
  1. Redimensionner
  2. Cropper
  3. Supprimer le background (s'étendre la dessus)
  4. Détecter le contour (Canny Edge Detection)
  5. 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 du contour de Contour\_commun (cv2.findContours)
    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 logo  $L'$  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 filtrer lors du pré-traitement sur les plans, notamment en rajoutant du blur ou en supprimant le background, 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 [9]. 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 [11]) seront traitées.

Lors de l'entraînement, le réseau va apprendre à partir de séquences d'images labellisées (logo/non-logo).

Comme pour [9], 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 ([11]).

# 7 Collecte des données & entraînement

## 7.1 Datasets

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

- Dataset non logo

- Dataset logo

Chaque élément du dataset contient :

- une séquence de 20 images représentant la séquence (pour un classifieur d'image à convolution 3D)
- une séquence 19 images représentant (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 cinquantaine 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 qui la suite; ainsi, nous obtenons une matrice de dimension, 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

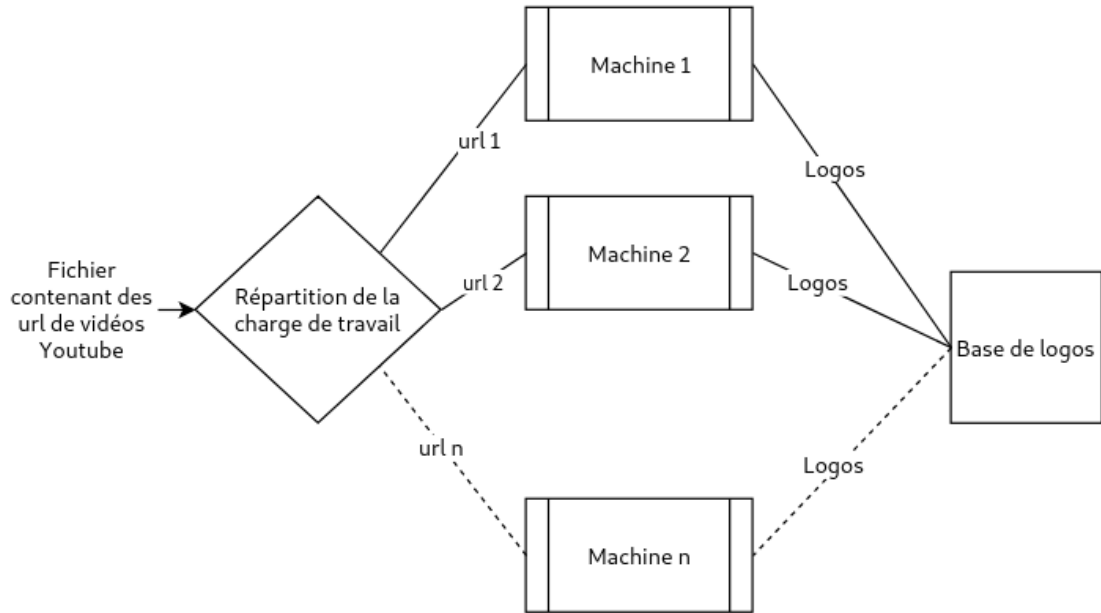


Figure 19: Architecture proposé pour le scrapper de logos

Nous voulons une architecture de scrapping qui soit distribuée.

En effet, le traitement d'une vidéo prend en moyenne trente minutes, 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 upload 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.