Replay Detection

July 1, 2019

Contents

1 TODO : Problématique

1.1 Replay

1.1.1 Pourquoi les replays

1.1.2 Logo

2 Objectif

Nous voulons comparer l'efficacité des méthodes de vision par ordinateur "classique" aux méthodes par apprentissage profond (deep learning). L'objectif dans cette parti est double. Le premier est de détecter le plus efficacement les logos dans les vidéos afin d'avoir une **base line** à comparer avec l'approche par deep learning. Le second est de mettre en place un système capable d'extraire les frames avec des logo, et ce sans intervention humaine (pas de labeling, pas d'ensemble d'entraînement à construire), afin de pouvoir constituer automatiquement un ensemble de données permettant d'entraîner des modèles de machine learning.

3 Première partir : Détection sans deep learning

4 Détection des replays : état de l'art

4.1 A Robust Replay Detection

Cette approche détecte les replays en trouvant les logos dans la 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. Tous les

frames qui ont obtenu un score supérieur à un certain seuil sont considérés comme des "logo templates".

4.1.1 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 ce frame et les 17 frames précédents. Nous obtenons un score L_i pour chaque frame i dans la vidéo. Tous les frames dont le score est inférieur à un certain seuil sont écartés, les autres vont servir à trouver le logo template.

4.1.2 Recherche du logo template parmis les frames filtrés

Le logo template est le frame qui représente le mieux tous les logos dans la vidéo. Pour déterminer le logo template parmis les frames filtrés, 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, nous allons chercher dans le cluster avec le centre de cluster le plus élevé, le frame m minimisant la distance avec tous les autres frames N du cluster.

4.1.3 Recherche des logo

Une fois que le logo template est déterminé, chaque logo trouvé en 1 va être comparé avec le logo template. La mesure de comparaison est la distance 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.

4.1.4 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).

4.1.5 Conclusion sur cette méthode

Trop dépendant de la luminance, ne parvient pas à détecter les logo peu lumineux. Peu précis (trop de faux positif) Dépend de trop de paramètres (seuil de luminance, ...) affectant les logos détectés.

4.2 Fast Highlight Detection and Scoring for Broadcast Soccer Video Summarization using On-Demand Feature Extraction and Fuzzy Inference

4.2.1 Idée

Entrainer un algorithme CART des histogrammes 3D sur des frames de logos choisi à la main. Utilisé l'algorithme CART sur les histogrammes pour prédire logo/non-logo.

4.2.2 Conclusion

Nous ne voulons pas maintenir un ensemble d'apprentissage. Cette méthode ne convient pas à nos besoins.

4.3 Automatic summarization of soccer highlights using audio-visual descriptors

4.3.1 Idée

S = Détecter tous les shots (plans) dans la vidéo L = Pour chaque shot S_i:

- L_{istart} = La "luminance" des frames au début du shot
- L_{iend} = La "luminance" des frames à la fin du shot
- $L_{template}$ = Trouver le "logo template" dans L
- Pour chaque logo l dans L:
  - Diff l avec $L_{template}$ = conversion grayscale puis somme de la soustraction pixel par pixel
  - Si Diff l avec $L_{template} <$ threshold => l est un logo

4.3.2 Conclusion

Cette méthode est trop semblable à l'approche "Robust Replay Detection" qui ne répond pas à nos besoin, 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.

4.4 MEAN SHIFT BASED VIDEO SEGMENT REPRESENTATION AND APPLICATIONS TO REPLAY DETECTION

4.4.1 Idée

Segmenter la vidéo en frame, puis calculer une représentation compressée de chaque frame. Pour détecter les logo (ou n'importe quoi), il faut d'abord "apprendre" plusieurs formes compressées de logo (sur des vidéos d'apprentissage que nous aurons labelisé à la main), puis il faut simplement calculer une distance entre la forme compressé du shot à définir et les formes compressées apprises.

4.4.2 Conclusion

Cette approche est intéressante, néanmoins l'article n'est pas assez précis, notamment sur la manière dont les images sont compressées. De plus, nous ne voulons par maintenir un ensemble d'apprentissage. Cette méthode ne convient donc pas à nos besoins.

4.5 Real-time field sports scene classification using colour and frequency space decompositions

4.5.1 Idée

Classifie les shots en fonction de la distance (proche, moyen, loin) et de ce qu'il y a dedans (visage, épaule, un seul joueur, plusieurs joueurs, terrain, spectateur).

4.5.2 Conclusion

A l'air d'être une approche solide. Néanmoins, c'est de la classification supervisée, il faut donc un ensemble d'apprentissage. Cette méthode ne convient donc pas à nos besoins.

5 Les approches proposées

5.1 TODO : mettre en avant le fait que l'algo va être mis en prod

Les méthodes "état de l'art" ne donnent pas d'assez bons résultats et ne peuvent pas être mises en production. Nous cherchons donc notre propre

méthode.

Pour détecter les replays, nous faisons les hypothèses que :

- 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 logo de replay dans les vidéo de sport. Dans cette partie, chacune de ces approches n'utilisent que des algorithmes de computer vision classique (flouttage, filtre de Canny, ORB, ...) et des algorithmes de machine learning non-supervisés (K-NN). Ces restrictions s'appliquent pour les raisons suivantes :

- le programme doit être le plus rapide possible (les réseaux de neurones sont en général trop lents, trop exigeant en ressource); d'où le choix d'algorithme plus simple.
- si la solution doit être mise en production, il est préférable de ne pas avoir d'ensemble d'apprentissage à obtenir ou maintenir; d'où le choix d'algorithme non-supervisé uniquement.

5.2 Détection des plans

Les approches que nous proposons itérent 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 logo parmi tous les frames de la vidéo, nous réduisons la recherche à tous les frames qui sont entre deux plans.

5.2.1 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.

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 appliquée sur M. M = UWV, où W est la matrice de valeur singulière.

Les diagonales de la matrice W comportent des poids S ordonnées de manière non croissante. Le premier poid S_1 est le poid maximal. Ces poids représentent l'information contenu dans le vecteur V.

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

Si le rang d'une fenêtre est plus que grand que le rang de la fenêtre avant elle, alors le contenu visuel de la fenêtre est différent de la fenêtre précédente. A 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 parmis les fenêtres environnantes.

1. Résultats obtenus et conclusion Cette méthode pour trouver les plans dans une vidéo est très efficace, et constitue la base 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 vidéo de 120 minutes à 60 fps), il est impensable d'utiliser un algorithme en $O(N^2)$, par exemple en comparant toutes les frames entre elles, 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 potentiels.

6 Première approche : ORB

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

6.1 Extraction des caractéristiques

OpenCV permet d'extraire des features à partir des images (détection des bords des objets dans l'image). A partir de ça, nous pouvons représenter l'image comme un vecteur de feature. Les méthodes d'extraction sont ORB et AKAZE.

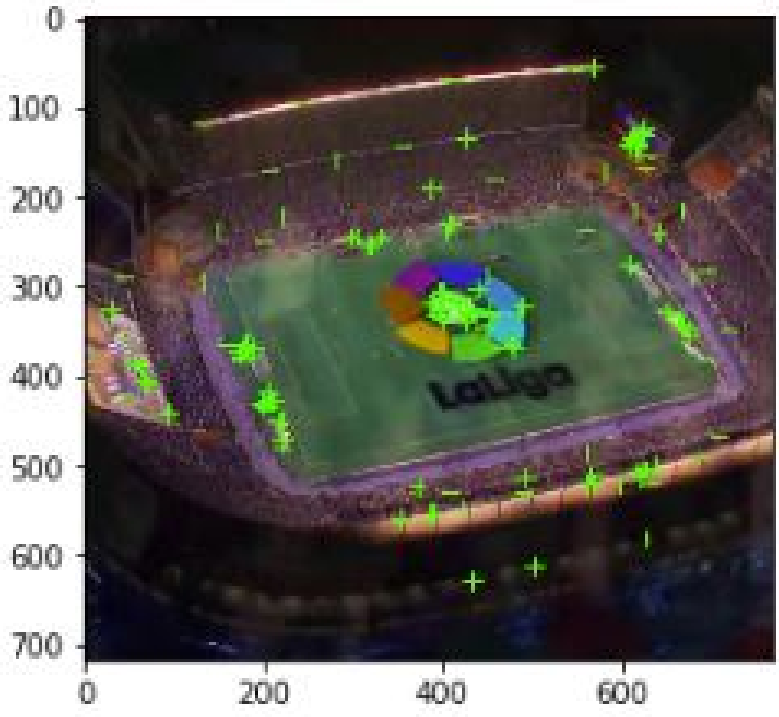
6.2 KMeans

OpenCV implémente aussi l'algorithme KMeans. Celui-ci permet de regrouper les objets similaires en fonction de leur feature. Dans notre cas, il va nous permettre de créer 2 groupes d'images : logo / non logo. L'avantage de KMeans est qu'il est est très rapide et assez efficace dans la plupart des cas. C'est l'un des algorithmes de clusterisation les plus utilisés.

6.3 Expérimentation et résultat:

Ensemble de test : une vidéo de ligue 1, une vidéo de liga, une vidéo de premier league et une vidéo NFL. Dans toutes les expérimentations, la vidéo est découpée en shot (plan). Soit S l'ensemble des shots.

6.4 1 frame par shot



- 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

Résultats :

6.4.1 TODO meilleurs res

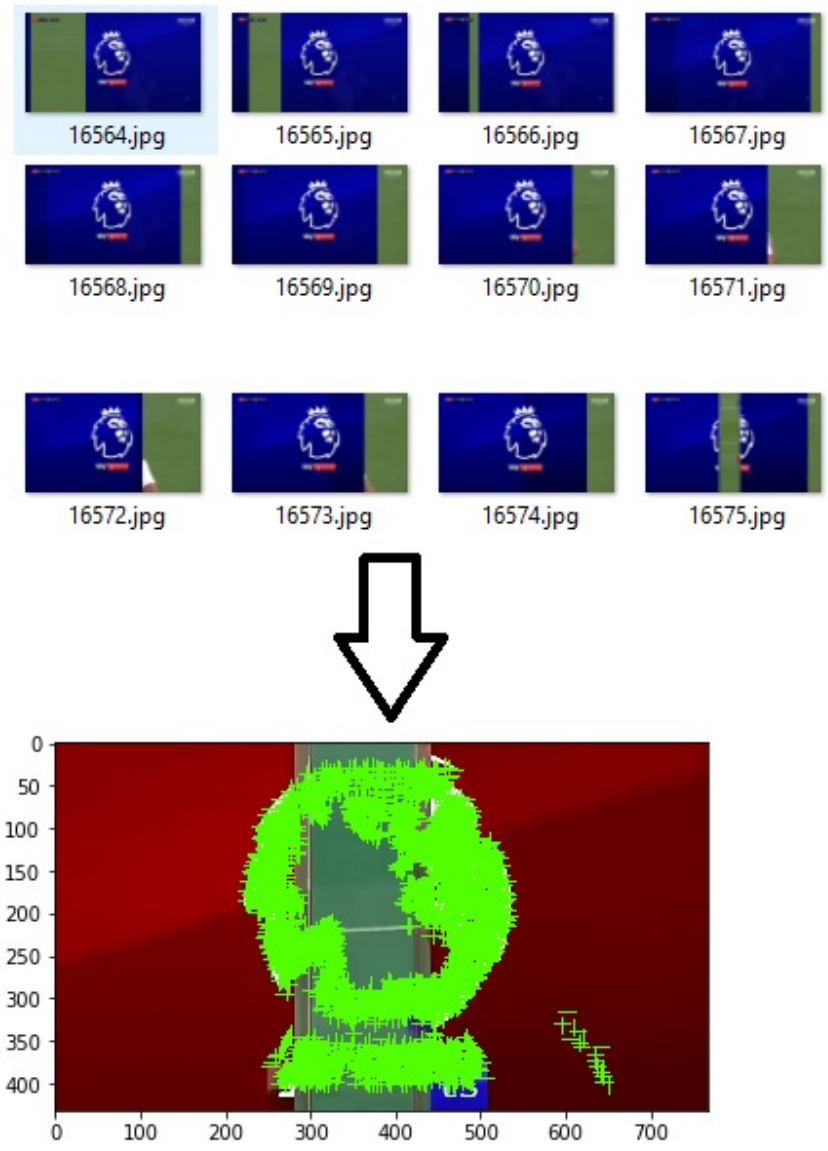
Mauvais sur toutes les vidéos

6.5 W frames par shot:

- Récupérer W frames pour chaque shot
  - nous obtenons |S*W| frame, où W est le nombre de frame
- Pour chaque frame, calculer ses features (orb ou akaze)
  - nous obtenons |S*W| vecteurs
- Utiliser KMeans avec K=2 pour séparer les vecteurs en deux groupes
  - le groupe le plus petit est le groupe des logo

Résultats : Mauvais sur toutes les vidéos

6.6 1 fenêtre de frame par shot:



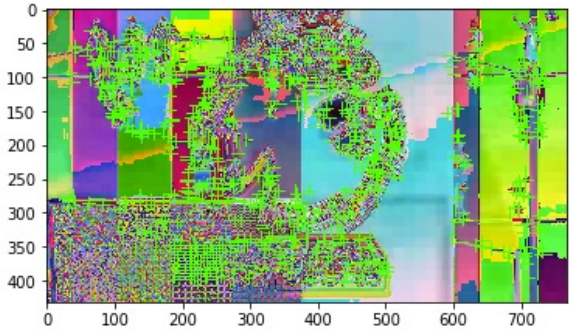
- Récupérer W frames pour chaque shot, les régrouper en une fenêtre
  - nous obtenons |S| fenêtre de dimension W, où W est le nombre de frame
- Pour chaque fenêtre, calculer ses features (orb ou akaze)

  - 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

Résultats: De bons résultats sur la vidéo de PL. Mauvais résultats sur les autres vidéos.

6.7 1 fenêtre de frame par shot et différence des frames dans la fenêtre:



- Récupérer W frames pour chaque shot, les régrouper en une fenêtre
  - nous obtenons |S| fenêtre de dimension W, où W est le nombre de frame
- Pour chaque fenêtre, calculer la matrice M égale à la différence de toute les autres frames dans la fenêtre

- Pour chaque matrice de différence, calculer ses features
  - nous obtenons |S| vecteur s
- Utiliser KMeans avec K=2 pour séparer les vecteurs en deux groupes
  - le groupe le plus petit est le groupe des logo

Résultats : De bons résultats sur la vidéo de PL. Mauvais résultats sur les autres vidéos.

7 Seconde approche : matching de contours

La méthode choisie différe avec les autres sur un point : au lieu de chercher à différencier les frames logo des frames non-logo, 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 $2 < t' < 90$ (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 '.

7.1 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 mosaiques TODO : explain this
- Pour chaque mosaique de plan S_A :
  - Pour chaque mosaique 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 (ceux qui ont au moins K points)
    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

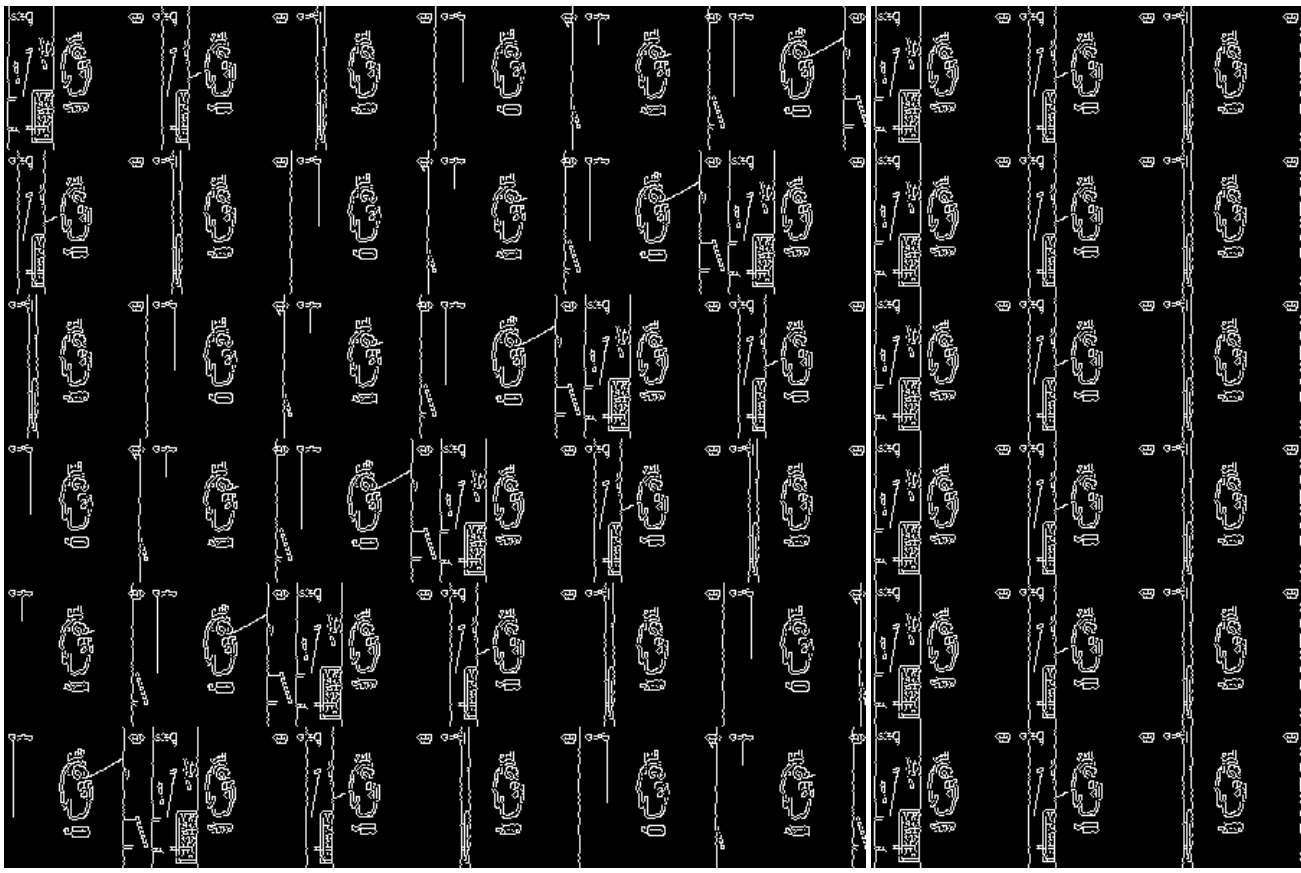
- Trouver les replays grâce aux logos

Pré traitement : Les frames sont resizé puis cropé vers le centre (pour ne pas avoir l'affichage en haut de l'écran etc...), puis un blur est appliqué (bilateralFilter, permet de filtrer certains faux positifs), et enfin on applique Canny Edge Detection.

Le point 3 de l'algorithme sert à filtrer les éventuels faux positifs. 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.

7.2 Mosaique de plan

7.2.1 TODO resize img

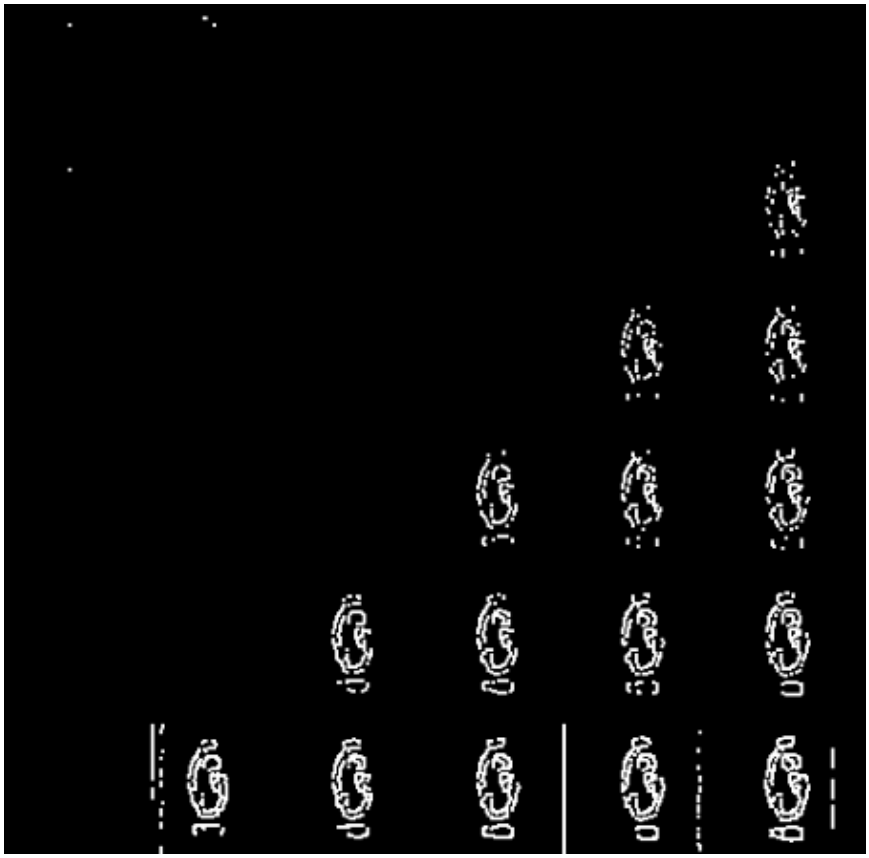


Pour chaque shot deux images au format .png (pas au format jpg, car celui-ci prend trop d'espace disque) sont générées.

Chaque image est de dimension I * I * width * height où I est le nombre de frame dans le shot.

Ces images sont en faites des matrices d'image qui vont permettre de comparer rapidement deux shot. La première matrice a un décalage d'un frame par ligne, la seconde n'a pas de décalage.

Pour comparer deux shot, il suffit d'appliquer un ET binaire entre les matrices des mosaiques, puis de calculer la longueur du conteur dans cette matrice.



7.3 Résultats et limitation

Les résultats sans le filtrage des faux positifs (l'étape 3 de l'algorithme) sont un bon moyen d'évaluer l'efficacité de notre méthode.

7.3.1 TODO : mettre les résultats ici

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 mosaiques.

Les limitations de notre méthode sont les suivantes :

- Dans certaines vidéos, il n'y a pas de logo pour les replays (simple fondu)
- Dans certaines vidéos, les logo de début et fin de replay ne sont pas les mêmes.

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

8 Deuxième partie : Détection avec deep learning

8.1 Etat de l'art

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 consécutives; il y a donc une composante temporelle à notre recherche, et nous pouvons considérer la transition d'un logo comme une action.

8.1.1 Two-Stream Convolutional Networks for Action Recognition in Videos

Cet article est écrit par Karen Simonyan et Andrew Zisserman. 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 sur 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 (). Ces classifieurs sont des réseaux de neurones convolutifs profonds.

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.

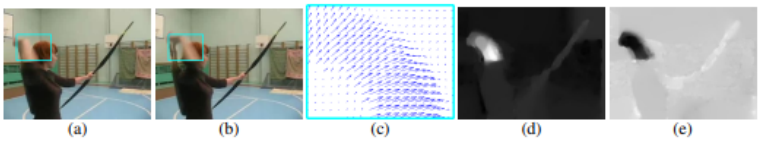


Figure 2: **Optical flow.** (a),(b): a pair of consecutive video frames with the area around a moving hand outlined with a cyan rectangle. (c): a close-up of dense optical flow in the outlined area; (d): horizontal component d^x of the displacement vector field (higher intensity corresponds to positive 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).

2. Classifieur temporel
   Source : Two-Stream Convolutional Networks for Action Recognition in Videos, Figure 2 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". Dans celle-ci ils utilisent la méthode "optical flow" pour détecter le mouvement des objets entre des frames consécutives. Ces mouvements vont être stockés dans des vecteurs et utilisés par le classifieur. Ils définissent aussi un hyperparamètre L qui définit la distance maximum entre deux frames pour laquel il faut calculer l'optical flow. Par exemple, si L=5, alors pour le frame #50, il faudra calculer l'optical flow entre le frame #50 et le frame #51; entre #50 et #52; etc... jusqu'à #50 à #55.

Table 4: **Mean accuracy (over three splits) on UCF-101 and HMDB-5**

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

3. Méthode d'évalutation et résultats obtenus
   Source : Table 4: Mean accuracy (over three splits) on UCF-101 and HMDB-51.

   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 tâche). Les dataset utilisés pour l'entraînement et l'évaluation sont UCF-101 et 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) : $y_t = y_{tspatial} + y_{ttemporal} / 2$
   - 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 en **link table** 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 renforcé dans l'hypothèse qu'il est nécessaire d'étudier une vidéo non pas comme une suite d'image indépendante, mais comme une suite de séquence avec un lien au sein de chaque séquence. Il semblerai que la temporalité a une très grande importance pour l'analyse de vidéos.

8.1.2 Temporal Segment Networks: Towards Good Practices for Deep Action Recognition

ConvNet ne peut pas être appliquée à la reconnaîssance d'action dans les vidéos pour deux raisons :

- Les dépendances temporelles éloignées jouent un rôle important dans la compréhension des actions dans les vidéos.

Or, les réseaux ConvNet classiques se focalisent sur les dépendances proches (les mouvements brefs).

- Entraîner un réseau de type ConvNet requiert un gros volume de données pour obtenir des performances optimales.

Or, obtenir des ensembles de données annotées et de qualité est une tâche compliquée; le sur apprentissage est un problème important à considérer.

8.2 Scrapping

L'approche par matching de contours convient tout à fait à la tâche de scrapping. En effet, elle est :

- rapide : moins de **X** minutes sur une machine **machine de référence EC2 (ou autre)** pour une vidéo de 90 minutes à 60 fps
- précise : seulement **X** % de faux positifs sur **Y** logos scrappés

8.2.1 Architecture du scrapper

- requête HTTP avec une ID youtube => logo uploadé sur GCP

- image docker (avec le serveur à l'écoute des requêtes) déployée sur le cloud

Cette architecture est scalaire; ceci nous a permi de scrapper plusieurs vidéos en parallèle et d'obtenir un dataset conséquent.

8.3 Datasets

- Dataset non logo
- Dataset logo
- Dataset logo séparé en fonction du logo (ligue 1, premier league, ...)

8.4 Détection des frames logo

8.4.1 Propre modèle

8.4.2 VGG net

8.4.3 Transfert Learning

8.4.4 Comparaison résultat

8.5 Détection des séquences de frames logo

9 Appendice

Clustering : Histogramme : Frame : Shot : plan FPS : frame per second / image par seconde Cropper

10 Source

Two-Stream Convolutional Networks for Action Recognition in Videos, Karen Simonyan Andrew Zisserman, 2014