FutbolPY: Detectar transiciones de cámaras en partidos de fútbol

May 20, 2019

Abstract

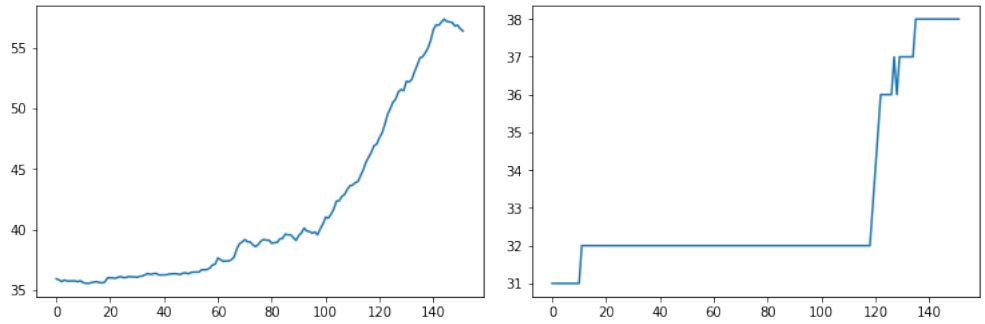
El presente trabajo se centra en la implementación de expresiones matemáticas y algoritmos para la detección de cambios de cámara, como así también el análisis de las propiedades y los estadísticos de los frames que componen un video.

1 Introducción

El concepto a tratar consiste en determinar en que momento se produce la transición de una cámara a otra en las trasmisiones de un partido de futbol por medio de software y a partir del mismo reconstruir a cada uno de los segmentos como videos.

2 Primeros pasos

Como una primera aproximación intuitiva, para detectar la transiciones suaves en un video, surgiría la posibilidad de analizar como si fuese la continuidad de una función f(x) a la media y la mediana que obtenemos a partir de cada uno de los frames en HSV y HSL de un videos. Entonces tenemos, analizando únicamente el canal H su media y mediana correspondiente:



En los gráficos anteriores la variable x representa el número de frame, y la variable y representa la media de ese frame en el canal H. Podríamos interpretar

que si analizarnos los gráficos como una función matemática, los sectores en donde claramente se observa discontinuidad indicaría un evidente cambio de cámara. Sin embargo, basta con hacer una simple una comprobación visual de los frames, de que estos saltos no representan ni la más mínima posibilidad de una transición. De esta forma podemos comprender, que necesitamos de una métrica que sea mucho más descriptiva [1], y en este aspecto la distancia de Bhattacharyya nos permite resolver este problema [2].

3 Distancia de Bhattacharyya

Es utilizado para medir la similitud entre dos distribuciones [3] en en este caso los histogramas de las imágenes (H1, H2), $\bar{H}1$ y $\bar{H}2$sus medias correspondientes:

$$d = \sqrt{1 - \frac{1}{\sqrt{\bar{H}1\bar{H}2N^2}} \sum \sqrt{H1(I)H2(I)}}$$

Obteniendo valores que oscilan entre 0 a 1, siendo el valor 0 que representa la distancia más cercana, es decir son iguales las imágenes.

Para agregarlo a la implementación correspondiente se ha procedido a codificarlo.

```
def distancia (a,b,n=1):
    sumatoria=np.sum(np.sqrt(a*b))
    medio= 1/np.sqrt(np.average(a) * np.average(b)*n*n)
    return math.sqrt((1-(medio*sumatoria)))
```

Para comprobar que el código es correcto se realizaron algunas pruebas simulando los valores del canal H [4] y con dos frames de un video [5].

4 Desarrollo e Implementación

El desarrollo del software se realizo en el entorno Jupyter Notebook usando el lenguaje de programación Python en la versión 3. La solución se encuentra disponible en su repositorio en GitHub [6] y se encuentra dividido en módulos que se detallarán a continuación.

4.1 Módulo de Entrada

Módulo que permite la recepción de opciones y argumentos desde la linea de comando o terminal, esto es a fin de facilitar el uso de FutbolPy como una herramienta.

4.2 Módulo Captura de Frames

Permite obtener todos los frames no dañados, guardarlos a cada uno como una imagen (esto es sumamente útil para comprobar), y obtener la cantidad de frames (teniendo en cuenta que OpenCV no provee una forma efectiva de obtener la cantidad de frames útiles/dañados).

Mediante una rutina optimizada de código, obtenemos los frames que posteriormente los utilizaremos para determinar cuándo se producen las transiciones en un video. También se encarga de enumerar los frames, esto es necesario para las posteriores funcionalidades.

```
def capturarFrames(filename,guardar):
    if guardar:
        path=os.path.dirname(os.path.abspath(filename))
        os.mkdir(path+'/img-'+filename)
    video= cv2.VideoCapture(filename)
    i=0
    success, frame = video.read()
    while success:
        i+=1
        if guardar:
            nombre = 'img-'+filename+'/'+str(i)+'.jpg'
            cv2.imwrite(nombre, frame)
        success, frame = video.read()
    return i
```

4.3 Módulo Captura de Distancias

Es aquí en qué aplicación de los conceptos matemáticos anteriormente desarrollados, los que nos darán la pauta que nos permitirá distinguir y una primera idea de la composición del video con respecto a distintas cámaras. El módulo realiza lo siguiente:

- Recorrer al video considerando a todos los frames.
- Realizar histograma normalizado en base al canal Hue del frame
- Comparar la distancia entre el frame anterior con respecto al frame actual.
- Guardar los valores de las distancia en un arreglo.

```
def obtenerDistancias(file, comparaciones):
    video= cv2.VideoCapture(file)
    valores = []
    hbins=np.arange(0,179) # rango para canal H
    # Frame 1
    i=0
    success, frame = video.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    h1,s1,v1 = cv2.split(hsv);
    # HIST1
```

```
    hist1 , bins = np.histogram(h1, bins=hbins, density=True)

    while success:
        i+=1
        success, frame2 = video.read()
        if i <= comparaciones:
            hsv2= cv2.cvtColor(frame2, cv2.COLOR_BGR2HSV)
            h2,s2,v2 = cv2.split(hsv2)
            # HISTN
            hist2, bins = np.histogram(h2, bins=hbins, density=True)
            valor=distancia(hist1,hist2, len(hbins)-1)
            valores.append(valor)
            hist1=hist2
    return(valores)
```

Como una subutilidad para este módulo se realizo un código para exportar los datos encontrados hacia un texto plano, con el fin de brindar una salida para herramientas externas de análisis de datos.

```
def exportData(file, myarray):
    myfile=file+".txt"
    file = open(myfile,"w")
    for el in myarray:
        file.write(str(el)+"\n")
    file.close()
```

El filtrado deberá permitir seleccionar aquellos valores que sean superiores a un valor de criterio, es decir realizar un corte sobre un umbral sobre las distancias obtenidas.

El primer paso es establecer un criterio para la aceptación/rechazo de las distancias obtenidas para ello se utiliza el criterio establecido en [7].

$$td = \frac{\sum d[i]}{N} + \sqrt{\frac{\sum (d[i] - \bar{d})^2}{N-1}}$$

$$tc = 2td$$

Siendo td un valor representativo de las instancias de movimiento graduales de una imagen y tc un valor representativo de un corte.

```
def criterio(dArray,n,precision=1):
    qua= np.power(dArray - np.average(dArray),2)
    parte1=np.sum(dArray)/n
    parte2=np.sqrt(np.sum(qua)/(n-1))
    return 2* (parte1 + parte2)*precision
```

Se debe tener en cuenta que se ha agregado un valor de precisión que permita brindar una forma para aceptar/rechazar máximos manualmente. No se debe confundir al mismo con el valor de efectividad.

Al implementar el filtrado se establecerá que:
- los valores superiores al criterio serán considerados
- los valores inferiores serán filtrados (se les asigna el valor 0).

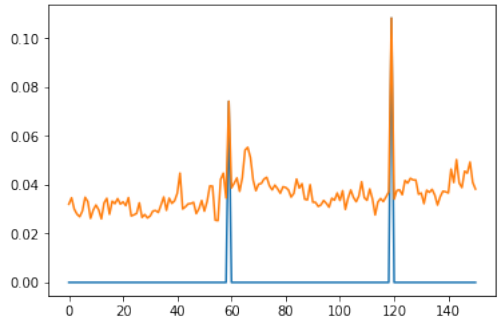
```
valorCriterio=criterio(arrayDistancia,178,0.95)
print("criterio:", valorCriterio)
filtro = (np.where(arrayDistancia>valorCriterio,arrayDistancia,0))

plt.plot(filtro)
plt.plot(arrayDistancia)
plt.show()
```

En base a lo realizado para el video "360.mp4" obtenemos un valor criterio de 0.07385399045684472 para un valor de precisión del criterio del 95%. A continuación se mostrará los valores resultantes de transición de cámaras.

# transición	# comparación de frames	valor de distancia
1	60	0.07419038254152195
2	120	0.10832220838731185

Si superponemos las gráficas de las distancia con respecto al filtro podemos ver su utilidad.



Con lo que podemos determinar que existen dos transiciones:
- comparación #60: correspondiente al frame 60-61
- comparación #120: correspondiente al frame 120-121

4.4 Módulo Reconstrucción de Vistas

Una vez finalizado el análisis nos queda reconstruir cada vista, considerando a una vista como el conjunto de frames representativos a lo capturado por cada cámara. Este módulo se de encargará de ello, lo único necesario que se le debe indicar al módulo es el frame incial y el frame final perteneciente al video.

```
def reconstruir(filename, inicio, fin):
    video= cv2.VideoCapture(filename)

    fps=video.get(cv2.CAP_PROP_FPS)
    size = (int(video.get(cv2.CAP_PROP_FRAME_WIDTH)),
```

```
        int(video.get(cv2.CAP_PROP_FRAME_HEIGHT)))
    codec =cv2.VideoWriter_fourcc(*'XVID')

    nuevoNombre,extension= filename.split(".")
    nuevoNombre+="-"+str(fin)+'.avi'
    nuevoVideo= cv2.VideoWriter(nuevoNombre,codec,fps,size)
    i=0
    success, frame = video.read()
    while success:
        if i>=inicio and i<=fin:
            nuevoVideo.write(frame)
        i+=1
        success, frame = video.read()
```

Para producir los nuevos videos se recorrerá el video original

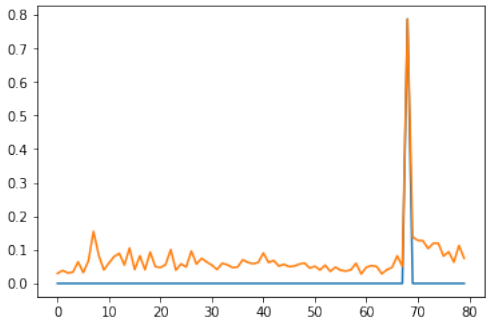
```
indice=0
anterior=0
for el in filtro:
    indice+=1
    if el>0:
        print("transición: ",indice,el)
        reconstruir(file, anterior, indice)
        anterior=indice
    if indice==cantFrames-1:
        reconstruir(file, anterior, cantFrames)
```

5 Pruebas

Para probar que el resultado obtenido no es un hecho aislado, procedemos a utilizar la misma implementación con otros videos.

Alonso

Correspondiente al segundo gol realizado por el 'Pichi' Alonso en el partido de Barcelona FC vs IFK Goteborg por la Copa de Europa 1985-1986 (https://youtu.be/A1Ef19us99s), en un segmento de video de duración 00:00:08 considerando 81 frames. Se encuentra un cambio de cámara en la comparación #69 (frames 69-70).



# transición	# comparación de frames	valor de distancia
1	69	0.7872924316856856


67.jpg

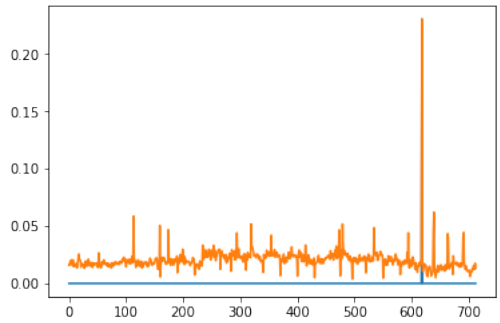

68.jpg


69.jpg


70.jpg

Contra

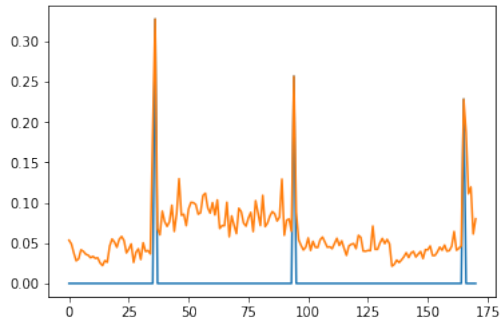
Video del partido amistoso Francia-Italia (duración: 00:00:23, frames: 714) del 01/06/18 (https://www.youtube.com/watch?v=Ajk6yyxX-gU) correspondiente a una jugada ofensiva que termina en penal. Se encuentra una transición de cámara en el frame 619-620.



# transición	# comparación de frames	valor de distancia
1	619	0.23031293079522488

Diego

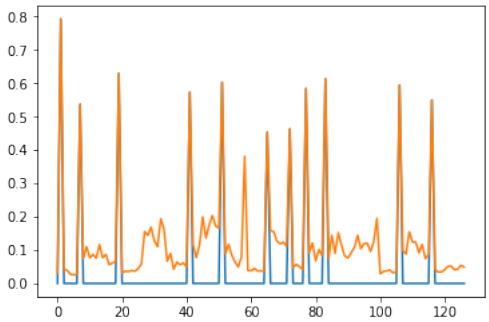
Segmento de video (duración: 00:00:37, frames: 172) correspondiente al Gol del Siglo (https://youtu.be/jOz2uGMTA2w), en el partido Argentina-Inglaterra en la Copa del Mundo México 1986.



# transición	# comparación de frames	valor de distancia
1	37	0.3277215659245849
2	95	0.25680995606178847
3	166	0.22836472847999076

Tafirol

Comercial sobre un analgésico en el que se hace representación de un partido de fútbol y en el que ocurren diversas transiciones bruscas de cámara. Se eligio el video para ampliar el alcance de la implementación y no solo limitarla a cuestiones de partidos oficiales.



# transición	# comparación de frames	valor de distancia
1	2	0.7925642234685557
2	8	0.5366641179534776
3	20	0.6295866567736305
4	42	0.5730945611974225
5	52	0.6030668861090255
6	66	0.45315613197150806
7	73	0.4631639027431631
8	78	0.5841412393214311
9	84	0.6131960378464988
10	107	0.5948208696254212
11	117	0.5496904945741801

6 Conclusiones

La implementación al realizar las graficas nos brinda valores pico que indican una alta probabilidad de que allí haya ocurrido una transición, incluso haciendo una simple control visual se puede constatar esto. El criterio empleado no tiene un 100% de efectividad por lo cual se ha brindado en la herramienta la posibilidad de ajustarlo a fin de aceptar/rechazar probables valores de transición.

7 Futuras líneas de investigación

Para el futuro se propone:

- Ampliar el alcance sobre videos de diversos tipos (deportivos o comerciales)
- Automatizar más las pruebas
- Brindar más mecanismos para mejorar el criterio de aceptación/rechazo
- Brindar una interfaz interactiva para mostrar resultados
- Brindar más características para establecer la solución como un editor de videos.

8 Referencias

Links y Bibliografía:

[1] Pruebas con medidas

[https://github.com/jrodrigopuca/futbolpy/blob/master/video-frame.ipynb]

[2] Temporal Segmentation of Association Football from TV Broadcasting, Fig 10, Siles, Junio 2013

[3] Bhattacharyya, A (1943) "On a measure of divergence between two statistical populations defined by their probability distributions", Bulletin of the Calcutta Mathematical Society, 35: 99–109 MR 0010358

[4] Pruebas con las distancias usando dos arreglos

[https://github.com/jrodrigopuca/futbolpy/blob/master/distancia.ipynb]

[5] Obtener la distancia de dos frames del video (frames: 1,101)

[https://github.com/jrodrigopuca/futbolpy/blob/master/video-distancia.ipynb]

[6] Repositorio de la solución

[https://github.com/jrodrigopuca/futbolpy]

[7] Temporal Segmentation of Association Football from TV Broadcasting, Fig 11, Siles, Junio 2013