Lenguajes de Programación: Trabajo Práctico IV

M. Sc. Saúl Calderón Ramírez
Instituto Tecnológico de Costa Rica,
Escuela de Ingeniería en Computación,
PAttern Recognition and MAchine Learning Group (PARMA-Group)

23 de octubre de 2016

El proyecto trata sobre el desarrollo de un sistema automático de segmentación temporal, lo que se refiere a la detección de cortes en videos digitales.

Fecha de entrega: 14 de noviembre.

1. Introducción y motivación

La masificación del internet, consecuencia del desarrollo de tecnologías de transimisión y compresión han llevado al uso generalizado y la disponibilidad de video digital. Aplicaciones tales como bibliotecas digitales, aprendizaje a distancia, vídeo bajo demanda, vídeo digital difusión, la televisión interactiva y la información multimedia sistemas generan utilizan grandes colecciones de videos digitales. Esto ha creado la necesidad de herramientas que pueden e indizar , buscar, explorar, y analizar este tipo de datos.

Ejemplo de estos sistemas de análisis automático de vídeo, es el sistema ACE, desarrollado actualmente en el PRIS-Lab, en la Universidad de Costa Rica. Este sistema tiene por objetivo analizar de manera automatica videos de futbol extraídos de transmisiones televisivas. La segmentación temporal en videos digitales se refiere a la distinción de cuadros de corte o transición entre dos secuencias de video distintas. La primer etapa del sistema ACE para analizar el video, vendría dada entonces por la segmentación temporal del video, para así definir las distintas secuencias en el video, y posteriormente clasificar las escenas según su importancia (descartando acercamientos al público y anuncios por ejemplo) y realizar el análisis del movimiento, tácticas y estrategias de los jugadores. En el presente proyecto se propone desarrollar el algoritmo de segmentación temporal, basado en el algoritmo propuesto en [2, 1, 3]. La Figura 1 muestra el proceso general de un algoritmo de segmentacion temporal.

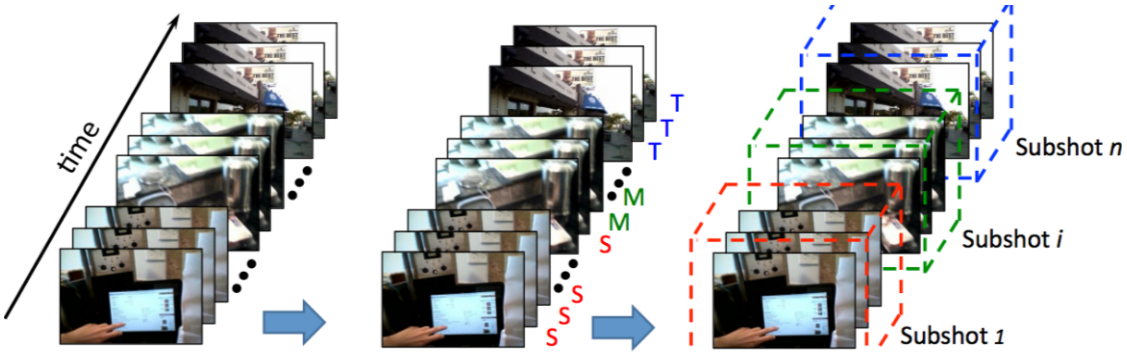

Figura 1: Diagrama general del proceso de segmentación temporal.

2. Algoritmo de segmentación temporal

Un cambio o corte en un video digital, se puede definir como un cambio abrupto o gradual entre dos cuadros consecutivos de un video: U_{t-1} y U_t. El algoritmo a implementar utiliza como descriptor el histograma $h_t(z)$ para cada cuadro o imagen U_t. Un histograma se define como un arreglo que contabiliza la cantidad de apariciones de la escala de gris z (para una imagen en escala de grises $0 \leq z \leq M$, con $M = 255$), normalizado a la cantidad de pixels N en la imagen.

Básicamente, el algoritmo por cada par de cuadros U_{t-1} y U_t calcula los histogramas h_{t-1} y h_t y cuantifica la disimilitud entre ambos histogramas, con el funcional $d\left(h_{t-1}, h_t\right)$. La medida de disimilitud o distancia puede venir dada por la distancia euclidiana, de Mahalanobis o de Bathacharyya (esta última es la que se usará en el presente trabajo). La disimilitud entre cuadros consecutivos se representa entonces en el arreglo g, y tiene un aspecto como el mostrado en la Figura 2, donde se puede observar la presencia de «picos» altos para cada par de cuadros, lo que denota la posible ocurrencia de un corte. A partir entonces del arreglo g se define un umbral óptimo, usando estadística básica, para clasificar un cuadro como de corte o no corte.

A continuación se presenta

1. Abra el vídeo de prueba provisto, e itere por cada uno de los cuadros (de dimensiones $A \times B$), leyendo la imagen RGB correspondiente, y transformándola al espacio de color HSV, con la función $rgb2hsv$ en MATLAB, o similar en la librería *OpenCV* de *Python*. Normalice de 0 a 255 la capa H.

2. Calcule los histogramas normalizados (normalizados respecto a la cantidad total de pixeles muestreados $A \times B$, demodo que $0 \leq h(z) \leq 1$ y $1 = \sum_{z=1}^{M-1} h(z)$) de cada uno de los cuadros usando únicamente la capa H (con 256 cubetas), implementando una función para ello.

3. Implemente la función de distancia entre dos histogramas $d_B(h_1, h_2)$, que

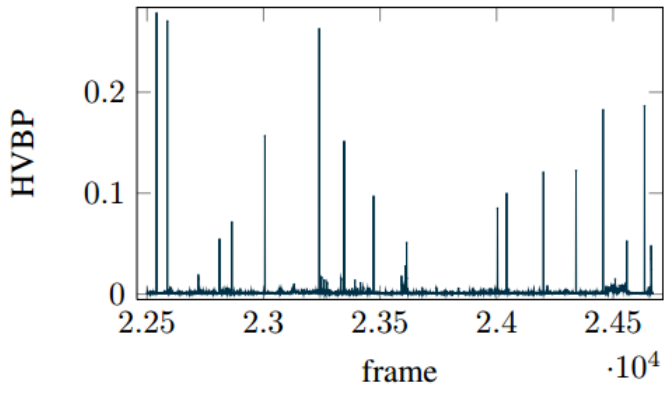

Figura 2: Disimilitud entre cuadros consecutivos usando la distancia de Bhattacharyya modificada en [2].

retorne la disimilitud entre dos histogramas normalizados usando la **distancia de Bhattacharyya**, la cual se define de la siguiente manera:

$$d_B(h_1, h_2) = \sqrt{1 - \alpha\beta},$$

donde el coeficiente de Bhattacharyya $\beta = \sum_z \sqrt{h_1(z)\, h_2(z)}$ y el coeficiente de normalización

$$\alpha = \frac{1}{\sqrt{\overline{h}_1\, \overline{h}_2\, M^2}},$$

donde $\overline{h}_i$ se refiere al valor medio del histograma. Se debe cumplir que y que

4. Calcule el arreglo de disimilitud entre cuadros consecutivos $g(t) = d_B\left(h_{t-1}, h_t\right)$.
5. A partir del arreglo g calcule su media g_μ y desviación estándar g_σ, y suponiendo que las distancias entre cuadros consecutivos siguen una distribución normal, clasifique cada cuadro U_t usando la propiedad de los 3 sigmas:

$$\begin{array}{c} u(t) = 1 \text{ si } g(t) \geq |g_\mu \pm g_\sigma| \\ u(t) = 0 \text{ sino} \end{array},$$

donde si $u(t) = 1$ entonces $U_t \in C$, donde C corresponde al conjunto de cuadros que son cortes.

3. Implementación

Para la implementación del proyecto, debe usar el lenguaje *Python*, en conjunto con las librerías *numpy y tensorflow*. Además puede usar la librería OpenCV, para manipular videos.

3.1. Tensor flow

Para instalar tensorflow debe seguir la documentación disponible en `https://www.tensorflow.org/versions/r0.11/get_started/os_setup.html#download-and-setup`. Tensorflow es una librería desarrollada por Google, que tiene por objetivo facilitar el desarrollo de algoritmos para plataformas de computación paralela. A continuación se presentan los conceptos básicos implementados en la librería (tomado de `https://www.tensorflow.org/versions/r0.11/get_started/basic_usage.html#basic-usage`).

Las ideas clave en tensor flow son las siguientes:

- Tensor flow representa las computaciones de un algoritmo como un grafo. Las dependencias de cómputos son entonces representadas por los enlaces entre los nodos.
- Los datos se representan con tensores los cuales «fluyen» a través de los aristas del grafo.
- Un grafo definido se ejecuta en una sesión, sobre todos los dispositivos habilitados para el cómputo paralelo.

Como se mencionó, el concepto básico de *tensorflow* es representar los cómputos de un algoritmo como un grafo. Los nodos en el grafo son llamados **ops** (abreviatura para operaciones), los cuales pueden ser construídos por el usuario, con base a *ops* provistos por la librería. Una operación toma como entrada uno o más **tensores** y resulta en un **tensor.** Un tensor consiste en un arreglo multidimensional utilizado para almacenar los datos necesarios para los cómputos.

Un grafo se ejecuta en una Sesión, la cual ejecuta las **ops** en los distintos **dispositivos** (Devices), los cuales pueden ser CPU's, GPU's o tarjetas de uso específico.

De esta manera, las interdependencias entre cómputos son claramente definidas de antemano por el programador, lo cual facilita la ejecución en paralelo. Además la implementación cada **op** provista por la librería es paralelizada.

A continuación se presenta un ejemplo sencillo, donde se define un grafo de dos tensores procesados por una **op** de multiplicación.

Primero, se definen los tensores a multiplicar:

```
vector1 = tensorflow.constant([[3., 3.]])

vector2 = tensorflow.constant([[2., 2.]])
```

Posteriormente se crea la operación **matmul** que multiplicará ambos tensores y retornará un tensor:

```
producto = tensorflow.matmul(vector1, vector2)
```

El grafo construído se ilustra en la Figura 3.

Para ejecutar el grafo, se crea una *sesión* la cual asignará en los dispositivos habilitados los computos definidos en el grafo. Para ello, se crea primero la sesión:

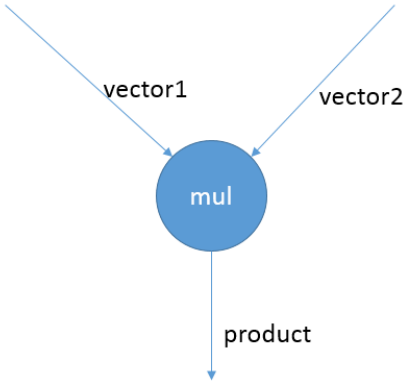

Figura 3: Grafo de computos del ejemplo.

```
sesion = tensorflow.Session()
```

Luego, a tal sesión, se le envía el tensor *producto*, con lo cual se le indica que obtenga tal tensor, lo cuál implica el computo del grafo definido para obtener tal tensor. Para ello se ejecuta el método *run* del objeto *sesion*.

```
resultado = sesion.run(producto)
print(resultado)
sesion.close()
```

Observe que es necesario cerrar la sesión al terminar el cómputo. La implementación de TensorFlow traduce el grafo a operaciones distribuidas entre las unidades de procesamiento disponibles. No es necesario asignar explicitamente un dispositivo a una operación específica. También es posible crear una sesión en *bloque* de ejecución haciendo:

```
with tensorflow.Session() as sesion:
        resultado = sesion.run([producto])
        print(resultado)
```

Para representar estados en un grafo, es posible crear variables. En el siguiente ejemplo, se crea una variable *contador* para representar el estado del grafo:

```
estado = tensorflow.Variable(0, name = "contador")
```

Posteriormente se crea un nodo que incrementa en uno el valor de la variable que representa el estado:

```
uno = tensorflow.constant(1)
nuevoValor = tensorflow.add(estado, one)
```

```
actualizacionEstado = tensorflow.assign(estado, nuevoValor)
```

Antes de ejecutar operaciones basadas en variables, es necesario fijar la operación de inicialización:

```
inicializacionOp = tensorflow.initialize_all_variables()
```

Finalmente se crea y ejecuta la sesión, en la cual se incrementa tres veces el valor del contador:

```
with tensorflow.Session() as sesion:
        sesion.run(inicializacionOp)
        print(sesion.run(estado))
        for _ in range(3):
                sess.run(actualizacionEstado)
                print(sess.run(estado))
```

Para el presente proyecto en particular, se recomienda utilizar la función *histogram_fixed_width(ndarray, value_range, nbins)* la cual toma el arreglo *ndarray* para calcularle el histograma según el rango de valores (cubetas) y la cantidad de cubetas especificada. Se adjunta al enunciado un corto ejemplo del uso de tal función, en conjunto con los arreglos de *numpy*.

4. Requerimientos del programa

1. El programa debe tomar la dirección de un archivo de video, o una carpeta con imágenes a la cual se le especifique un prefijo del nombre de cada imagen, y un sufijo variable de 1 hasta K, para el procesado de un vídeo de K cuadros.

    a) Como salida, el programa debe retornar un archivo en formato *xml* o *yaml* que especifique los cuadros donde el algoritmo declaró un corte (40 %).

2. En la documentación deben realizarse al menos 3 pruebas con videos o series de imágenes, donde en cada prueba debe existir al menos un corte, y debe mostrarse la gráfica de la función $g(t)$ y $u(t)$, y comentar la exactitud de los cortes detectados, según lo verificado visualmente (20 %).

3. La documentación debe incluir un diagrama de clases, el cual muestre el uso de al menos un patrón de diseño (como mínimo debe usarse el patrón MVC) (20 %).

4. La implementación debe usar *TensorFlow* en todas sus etapas (20 %).

Referencias

[1] F. S. Canales, "Ace-football, analysing football from tv broadcasting," *24th German Soccer Conference DVS – Fusball in Lehre und Forshung*, 2013.

[2] F. Siles, "Temporal segmentation of association football from tv broadcasting," in *Intelligent Engineering Systems (INES), 2013 IEEE 17th International Conference on*. IEEE, 2013, pp. 39–43.

[3] F. Siles Canales, "Temporal segmentation of association football from tv broadcasting," in *IEEE 17th International Conference on Intelligent Engineering Systems – INES 2013*. San Jose, Costa Rica: IEEE, 2013.