Visión por Computador
Grado en Ingeniería Informática
Universidad de Granada
Curso 2017-2018

Trabajo 3.
Indexación y Recuperación de imágenes.

Alberto Armijo
armijoalb@correo.ugr.es

20 de diciembre de 2017

Índice

Índice de figuras

1. Emparejamiento de descriptores.

Para este primer ejercicio, utilizaremos las funciones de la práctica anterior **correspondencias2NN()** para calcular las correspondencias entre dos imágenes, también se utilizará la función **calculateDescriptors()**; para esta función se ha añadido un parámetro llamado mask con el cuál especificamos a la función de OpenCV **detectAndCompute()** la zona de la imagen para la cuál queremos calcular los descriptores.

Para la creación de la máscara, se ha creado una función, llamada **createMak()**, la cuál toma como parámetros la imagen para la que se crea la máscara y las coordenadas de los puntos con los que se creará la máscara. Para crear la máscara, utilizaremos una matriz inicializada a 0, después, crearemos un polígono con los puntos que le hemos pasado a la función (utilizaremos la función **cv2.approxPollyPD**). Tras esto, introduciremos este polígono dentro de la máscara con la función **cv2.fillConvexPoly()**, hay que especificar que el color del polígono debe ser blanco, (255,255,255) si utilizamos RGB. Por último, tenemos que transformar esta máscara a blanco y negro (se puede hacer con la función **cv2.cvtColor()**).

Para tener diferentes máscaras para los diferentes ejemplos, se han creado las siguientes máscaras:

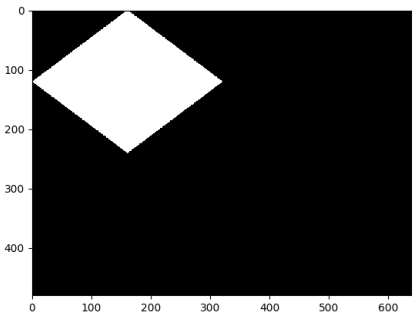

Figura 1.1: Máscara 1

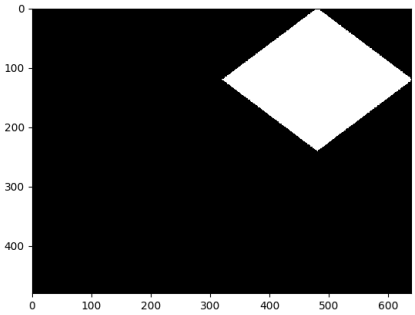

Figura 1.2: Máscara 2

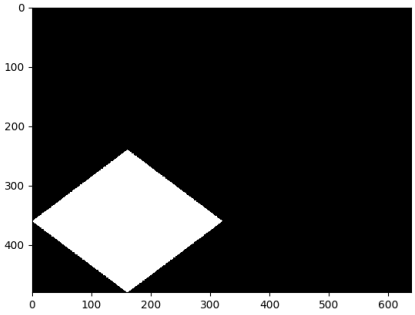

Figura 1.3: Máscara 3

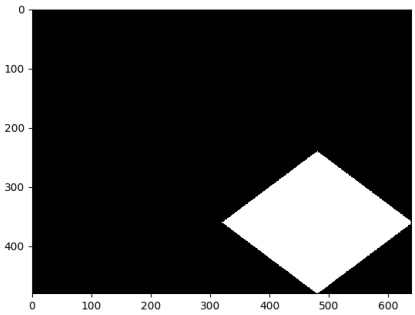

Figura 1.4: Máscara 4

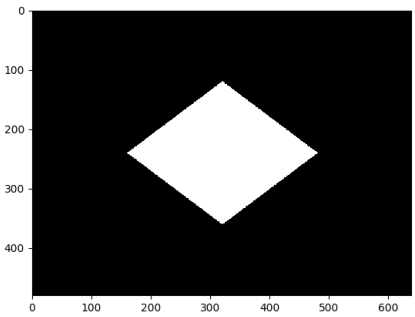

Figura 1.5: Máscara 5

Ahora, utilizaremos la función **calculateCorrespondenciasWithMask()**, esta función calcula las correspondencias entre las dos imágenes que se le pasan como parámetros, pero a la primera sólo se calculan los descriptores que se encuentren dentro de la máscara. Una vez tiene los descriptores calculados, se muestran las correspondencias entre las imágenes.

Para este ejercicio se han incluido dos ejemplos, formados por las parejas (128.png,130.png) y (229.png,232.png). Los resultados obtenidos son los siguientes.

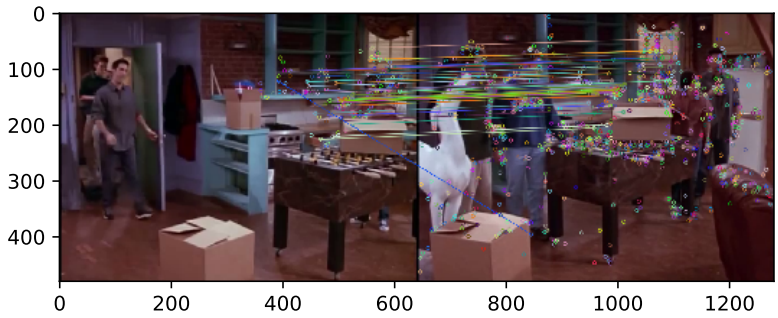

Figura 1.6: Correspondencias 128.png y 130.png

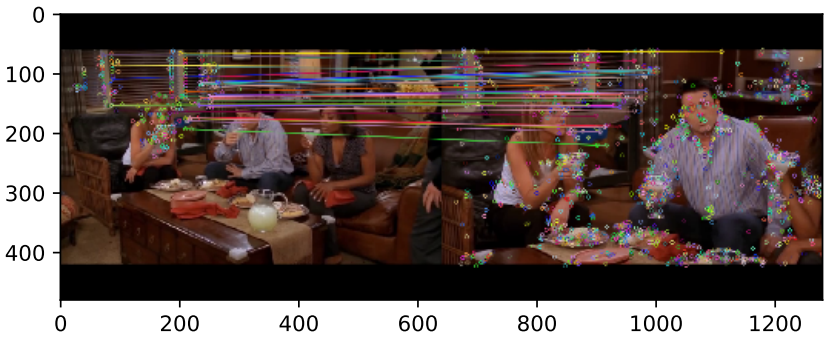

Figura 1.7: Correspondencias 229.png y 232.png

Como se puede ver en la imagen de las correspondencias, la mayoría de de las correspondencias son correctas, pero existen algunos errores. Esto puede ser un problema ya que para la recuperación de imágenes necesitaremos obtener características similares en ambas imágenes, y si utilizamos los descriptores como característica de la imágenes, posiblemente se obtengamos imágenes que no tienen relación con la original pero que tiene descriptores parecidos a la original.

2. Visualización del vocabulario.

Para el ejercicio 2, debemos de calcular los 20 parches más cercanos a una palabra, para ello, debemos calcular que descriptores pertenecen a la palabra y una vez que hemos obtenido los descriptores, calcularemos la distancia que hay entre ellos y la palabra visual.

Para calcular los descriptores que pertenecen a la palabra visual (cluster), se ha creado la función **obtainDescriptorsPerWord**. Esta función tiene como parámetros el índice de la palabra, los descriptores y las etiquetas; lo que debemos hacer es mirar si el índice de la palabra es igual a la etiqueta correspondiente al descriptor.

Ahora pasaremos a calcular la distancia entre la palabra visual y los descriptores, para ello se ha creado la función **calculateNearestDescriptors**. Esta función toma como parámetros los descriptores y la palabra. Para calcular la distancia entre los descriptores

y la palabra, utilizaremos la distancia euclídea; para calcularla, utilizaremos la función **cv2.norm()** especificando su parámetro *code=cv2.NORM_L2*. Una vez hemos calculado todas las distancias, ordenaremos el vector de distancias y nos quedaremos con las 20 primeras.

Además de hacer esto, dentro de esta función se calcula la varianza que hay dentro del cluster, con esto obtenemos un medida de dispersión de los clusters (palabras visuales) que después nos servirán para saber que clusters tienen un conjunto de parches más uniformes y cuáles no. Esto nos servirá a la hora de elegir ejemplos más adelante. Para calcular la varianza, utilizaremos los 10 primeros descriptores y la función **np.var()**.

Por último, calcularemos los parches correspondientes, esto se hace en la función **calculateNearestPatches()** que toma como parámetros los parches y los descriptores obtenidos de la función anterior. Lo único que se hace en esta función es devolver los parches correspondientes a los índices de los descriptores que hemos calculado.

Todo este proceso viene incluido dentro de la función **calculateVarianceAndBestPatchesWord()** esta toma como parámetros el índice de la palabra visual, el vocabulario, los descriptores, las etiquetas de los descriptores y los parches. También está la función **calculateVarianceAndBestPatches()**; esta realiza la misma función que la anterior pero para todas las palabras del vocabulario; esta se ha utilizado unicamente para obtener todas la varianzas y obtener los ejemplos (en el código está comentada).

Para elegir ejemplos, se han tomado dos palabras que tienen poca dispersión, estas son la 931 y la 899; como tercer ejemplo se ha elegido la palabra que más dispersión tiene, la 4054. Los resultados obtenidos son los siguientes:

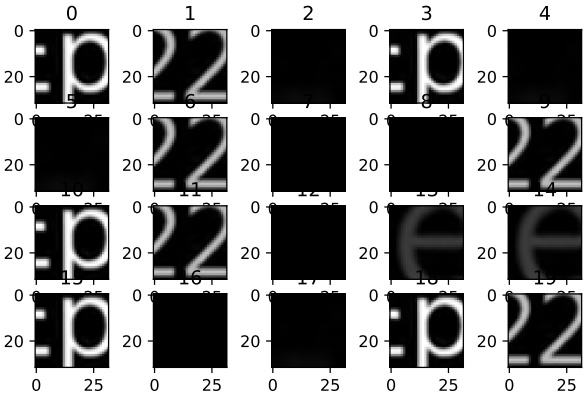

Figura 2.1: Parches ejemplo 1

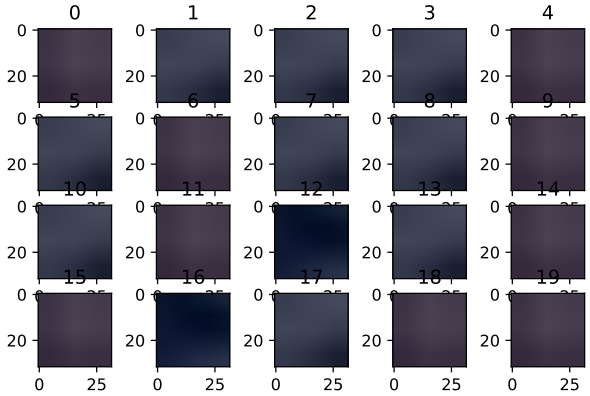

Figura 2.2: Parches ejemplo 2

Como se puede ver en el primer ejemplo todos los parches corresponden con la zona que representa a imágenes que contiene la palabra episodio 22x.

Para el segundo ejemplo podemos ver que tampoco hay mucha diferencia entre los parches, cambia el color y la intensidad de este, pero todos son más claros en el centro de la imagen.

En estos dos ejemplos se cumple lo que se menciona en teoría, y es que los parches que pertenecen a una palabra deben ser muy parecidos entre ellos (si se han obtenido buenos clusters).

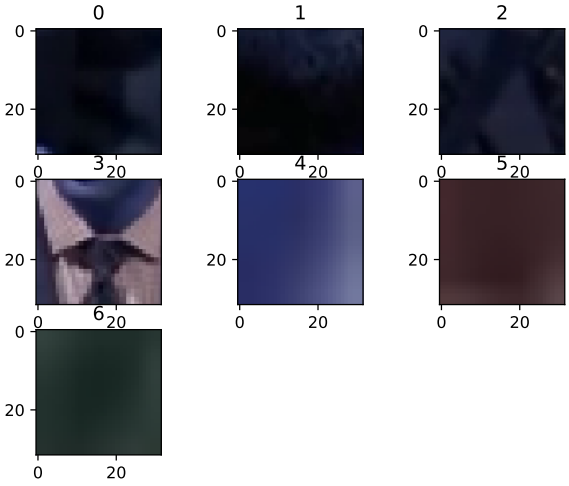

Figura 2.3: Parches ejemplo 3.

En este tercer ejemplo, podemos ver que no hay correspondencia entre ninguna de la imágenes, y que ni siquiera hay 20 parches. Esto se debe a que el algoritmo de k-medias necesita hacer 5000 clusters por defecto y puede ocurrir que aquellas imágenes que no pueda clasificar en otros clusters la mete en un cluster aunque no se parezca, como es el caso.

3. Recuperación de imágenes.

Para el tercer ejercicio, tendremos que crear un índice invertido y calcular la bolsa de palabras de cada imagen para poder después obtener imágenes semejantes a una que se pasa como parámetro.

Lo primero que debemos hacer para crear el índice invertido y el modelo de bolsa de palabras es calcular los votos para cada imagen que hay en el directorio */imágenes*, esto se hace en la función **calculateVotes()**. Para cada una de las imágenes, debemos calcular sus descriptores y normalizarlos, tras esto multiplicaremos el vocabulario de 5000 palabras por el vector de descriptores normalizado. Una vez hemos hecho esto, podremos calcular los votos que cada descriptor de la imagen, para ello recorreremos las columnas de la matriz anteriormente creada y por cada columna nos quedaremos con el índice (fila e índice de la palabra) del elemento con valor más alto. Por último, solo nos queda quedarnos con los votos que nos interesa, esto puede hacerse quedándose con las k palabras más votadas, o bien escogiendo todas las palabras que tengan un número de votos diferente de 0.

Una vez hemos obtenido las palabras más votadas en la imagen como la bolsa de palabras de la imagen (histograma de votos de la imagen), meteremos el nombre de la imagen en

todos los índice del fichero invertido en los cuales tiene votos; por otro lado, guardaremos el histograma de la imagen junto con el nombre de la imagen en una estructura que guarda el modelo de palabras de todas las imágenes que se han usado. Este proceso se realiza en la función **createInvertedIndex()**

Ahora tenemos que crear la segunda parte de este ejercicio, la cuál trata de obtener las imágenes semejantes a una imagen pasada.

Para esto, se ha creado la función **calcualteSimilarImages()**, la cual recibe como parámetros la imagen-pregunta, el índice invertido, el modelo de bolsa de palabras y el vocabulario. Lo primero que debemos hacer es calcular las imágenes que se pueden comparar con la imagen-preguntan, esto se realiza en la función **obtainImagesFromQueryImage()**. Dentro de esa función, se calculan las palabras por las que vota la imagen-pregunta, una vez hecho esto, se extrae por cada palabra votada las imágenes en las que aparece del fichero invertido y devolvemos el conjunto de imágenes. Tras esto obtendremos el histograma de estas imágenes.

Lo siguiente que debemos hacer el calcular qué imágenes son más semejantes a la imagen-pregunta, para ello, calcularemos el producto escalar normalizado entre la bolsa de palabra de la imagen-pregunta y cada una de las bolsas de palabras de las imágenes que hemos calculado antes. Por último, ordenamos las imágenes según el valor de la semejanza obtenido de mayor a menor y devolvemos las 5 mejores.

Los resultados obtenidos son los siguientes:

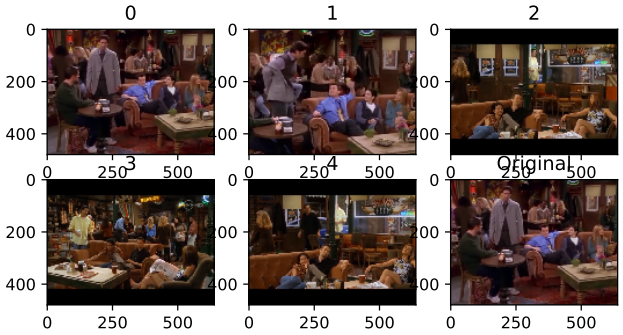

Figura 3.1: Imágenes similares primer ejemplo.

Para este ejemplo podemos ver que todas las imágenes relacionadas están en la cafetería sentados en el sofá, por lo que se obtienen buenos resultados.

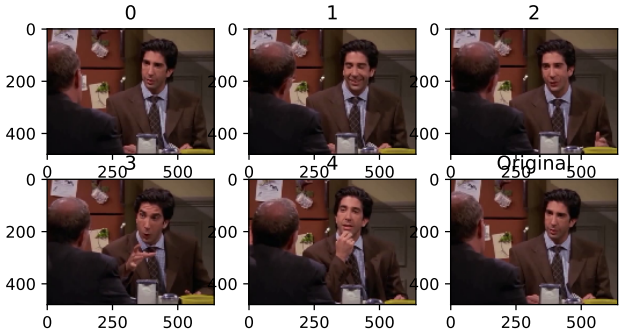

Figura 3.2: Imágenes similares segundo ejemplo.

Para este segundo ejemplo los resultados también son satisfactorio ya que obtiene imágenes de la misma escena, lo único que cambia es la posición y la cara del personaje.

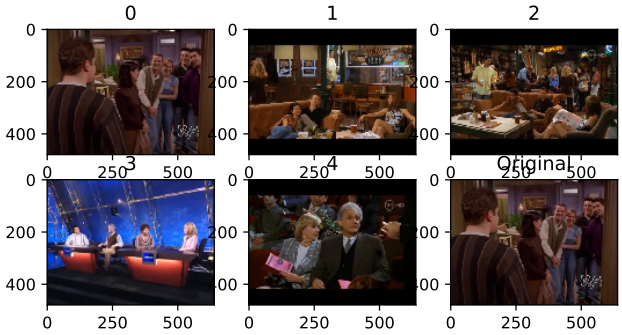

Figura 3.3: Imágenes similares tercer ejemplo.

Para este tercer ejemplo, podemos ver que el modelo de bolsa de palabras falla, dado que en las imágenes 2 y 3 al menos aparecen varios de los personajes que aparecen en la imagen original. Sin embargo, las últimas dos imágenes no tienen ninguna relación con la original. Esto a podido ocurrir debido a que para esta imagen no haya suficientes

ejemplos parecidos dentro del conjunto de entrenamiento y por lo tanto no haya votado por palabras adecuadamente.