Visión por Computador

Trabajo 3

Curso 2017/2018

Francisco Javier Caracuel Beltrán

caracuel@correo.ugr.es

4º - Grado en Ingeniería Informática – CCIA – ETSIIT – UGR

Índice

0. Consideraciones previas:

Para el desarrollo de este trabajo se han utilizado las siguientes funciones del trabajo1 y trabajo2 de esta asignatura:

- *set_c_map(imgs, cmap)*: asigna un esquema de color a todas las imágenes que se reciben.
- *show_images(imgs, names, cols, title, gray, cvt_color)*: muestra una lista de imágenes.
- *get_keypoints_descriptors(img, own, mask)*: calcula los keypoints y descriptores de una imagen.
- *get_matches_knn(img1, img2, k, ratio, n, flag, get_data, improve, mask1, mask2)*: obtiene los puntos que coinciden en dos imágenes utilizando la técnica "Lowe-Average-2NN".

1. **Emparejamiento de descriptores**: Leer parejas de imágenes de imagenesIR.rar que tengan partes de escena comunes. Haciendo uso de una máscara binaria o de las funciones extractRegion() y clickAndDraw(), seleccionar una región en la primera imagen que esté presente en la segunda imagen. Para ellos solo hay que fijar los vértices de un polígono que contenga a la región. Extraiga los puntos SIFT contenidos en la región seleccionada de la primera imagen y calcule las correspondencias con todos los puntos SIFT de la segunda imagen. Pinte las correspondencias encontradas sobre las imágenes. Jugar con distintas parejas de imágenes y decir que conclusiones se extraen de los resultados obtenidos con respecto a la utilidad de esta aproximación en la recuperación de imágenes a través de descriptores.

Para el desarrollo de este punto se ha creado la función *match_imgs(img1, img2)*. Esta función recibe dos imágenes e inicia una ventana en la que se puede seleccionar el área sobre la que se quieren obtener las correspondencias para emparejar esa área con la segunda imagen.

El objetivo de este ejercicio es obtener las correspondencias entre un área de la primera imagen y una segunda imagen completa. En definitiva, es el mismo proceso que se realizó en el trabajo2 con dos imágenes completas. En ese caso, se utilizó la función *get_matches_knn()*, por lo que se vuelve a utilizar en esta ocasión.

La diferencia entre el ejercicio del trabajo2 y éste es que se deben realizar las correspondencias solo entre un área de la primera imagen. Para realizarlo, se debe indicar a la función que calcula los keypoints y descriptores una máscara que contenga dicha área.

Para seleccionar el área se ha utilizado la función *extractRegion(img)* ofrecida en el fichero con las funciones auxiliares. Una vez que se tienen las coordenadas de la selección, se debe crear una matriz del tamaño de la imagen con todos los valores a 0 para indicar que son zonas que no interesan y con valores a 1 en la zona que contenga delimitado el polígono que se forma con dichas coordenadas (la máscara debe coincidir con el tamaño de la imagen sobre la que se quieren obtener los puntos SIFT). Para hacer este proceso se hace uso de la función de *OpenCV pointPolygonTest()*. Esta función recibe las coordenadas seleccionadas y devuelve un valor positivo si el punto que se comprueba está dentro del contorno enviado (puntos seleccionados), 0 si es el propio contorno y negativo si se encuentra fuera de él. Teniendo esta información solo es necesario recorrer cada punto de la matriz (inicialmente se encuentra completamente con valores a 0), comprobando con la función de *OpenCV* si ese punto pertenece al polígono para asignarle el valor 1.

Cuando se llega a este punto, se tiene una máscara que permite delimitar la zona seleccionada en el primer paso, pudiéndose indicar en el momento de obtener los keypoints y descriptores.

- Ejemplo 1:

Selección del área:

Máscara para obtener los puntos SIFT:

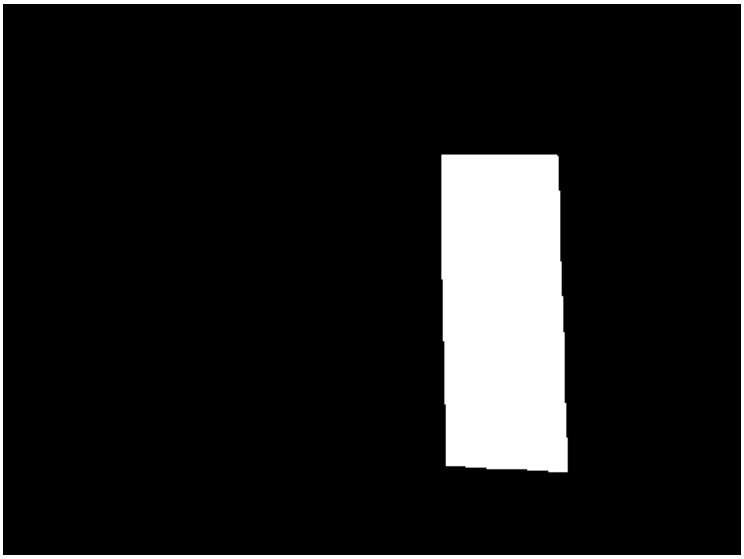
Matches:

La técnica utilizada no tiene una gran calidad, pero permite mostrar como la mayor parte de puntos SIFT encuentran su correspondiente en la segunda imagen. Se han podido encontrar las correspondencias gracias a que es una zona bien delimitada y característica que se mantiene en la segunda imagen.

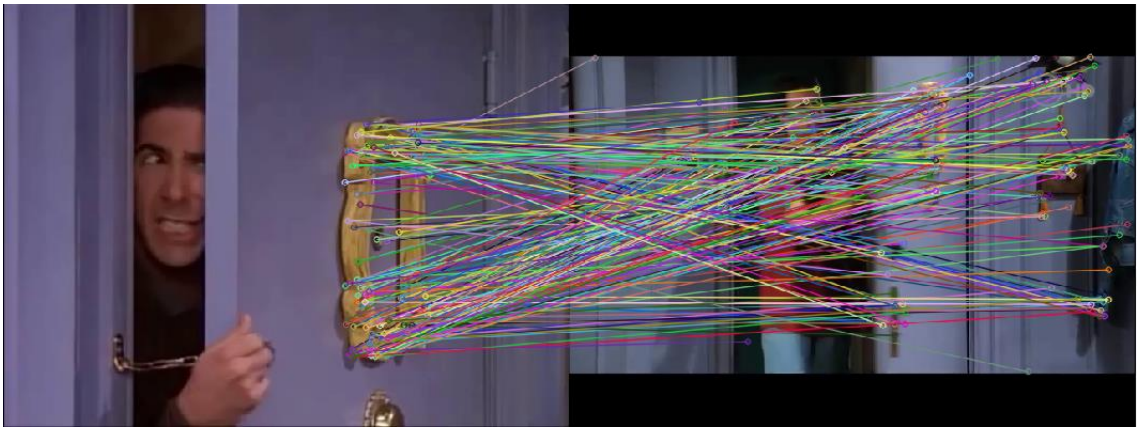
- Ejemplo 2:

Selección del área:

Máscara para obtener los puntos SIFT:



Matches:

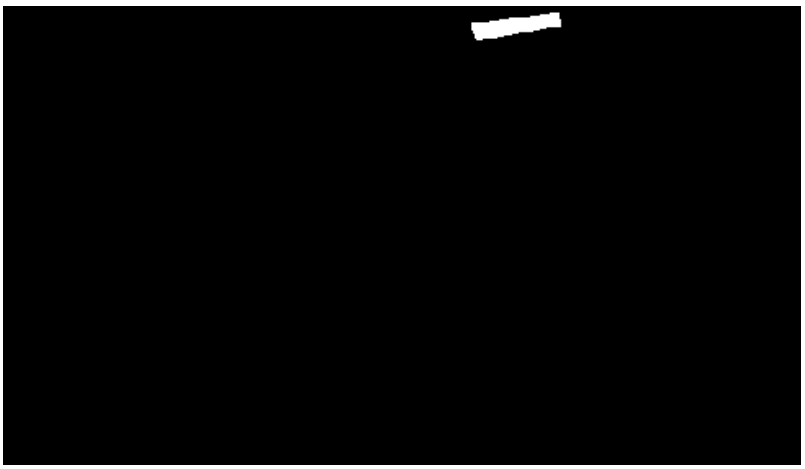


En este caso, la segunda imagen es muy regular, predominando el mismo color. Al realizar las correspondencias se han detectado los puntos en todas aquellas zonas que no forman parte de esa regularidad, lo que hace que estas correspondencias no se ajusten de forma correcta.

- **Ejemplo 3:**

Selección del área:

Máscara para obtener los puntos SIFT:



Matches:

La segunda imagen es muy heterogénea y la zona seleccionada es muy característica en ambas, por lo que la mayor parte de puntos sí han podido encontrar correctamente sus correspondencias entre ellas.

2. **Visualización del vocabulario**: Usando las imágenes dadas en imagenesIR.rar se han extraído 600 regiones de cada imagen y se ha construido un vocabulario de 5.000 palabras usando k-means. Se han extraído de forma directa las regiones imágenes asociadas y se han re-escalado a 24x24 píxeles. Los ficheros con los datos son descriptors.pkl, vocabulary.pkl y patches.pkl. Leer los ficheros usando loadAux() y loadDictionary(). Elegir al menos dos palabras visuales diferentes y visualizar las regiones imagen de los 20 parches más cercanos de cada palabra visual, de forma que se muestre el contenido visual que codifican. Explicar lo que se ha obtenido.

En este ejercicio se tienen varios archivos:

- *descriptors_and_patches.pkl*: contiene un listado con 193.041 descriptores y otro listado con 193.041 parches. Los parches son trozos de imágenes sacadas de imagenesIR.rar y los descriptores son los descriptores correspondientes a cada parche.
- *kmeanscenterX*.pkl: es un fichero que contiene la precisión, etiquetas y el vocabulario. Los valores que interesan son las palabras del vocabulario, que pueden ser 500, 1.000 o 5.000 (dependiendo del fichero utilizado). El vocabulario contiene los centroides, que son los descriptores de las zonas más representativas de cada clúster, por lo que se tendrán 500, 1.000 o 5.000 clústeres, uno por cada palabra o centroide. El otro valor importante son las etiquetas. Las etiquetas no identifican el nombre de cada clúster, sino que identifican el clúster al que pertenece cada descriptor/parche del fichero *descriptors_and_patches.pkl*. Un clúster se identifica por la posición que ocupa en la lista que se obtiene del fichero *kmeanscenterX.pkl*, por lo que si se tiene un descriptor cuya etiqueta es el número 40, el clúster correspondiente es el 40 y su centroide está en la posición 40 del vocabulario.

Explicado qué es cada elemento que se va a utilizar, lo que pide el ejercicio es elegir dos palabras del vocabulario y mostrar los 20 mejores parches correspondientes a esas palabras. Si se seleccionase la palabra 60, se deberían obtener los 20 mejores parches que pertenecen al clúster 60.

Se pide que se elijan dos palabras, por lo que se puede elegir una que sea buena y otra que sea mala. En este caso, se van a mostrar dos palabras buenas y dos palabras malas, pero el proceso se explica para una sola palabra, repitiéndose tantas veces como palabras se quieran mostrar.

Para desarrollar este ejercicio hay que tener en cuenta dos aspectos: cómo calcular los 20 mejores parches de cada clúster y como saber lo bueno o malo que es cada clúster.

- Calcular los 20 mejores parches de cada clúster:
  Por un lado, se tiene el descriptor que corresponde al centroide de cada clúster y, por otro, se tiene una lista con 193.041 descriptores pertenecientes a cada parche. Se deben agrupar todos los descriptores en clústeres. Para hacer esto se ha utilizado un diccionario, siendo la clave de cada elemento del diccionario, la etiqueta del clúster.
  Si se recorren todos los descriptores, colocándolos en su lugar correspondiente del diccionario, ya se tienen agrupados. El siguiente paso es calcular lo bueno o malo que es cada parche. Para calcularlo se utiliza el descriptor del centroide, obteniendo la distancia euclidea de éste con el descriptor del parche. Cuando se hayan calculado todas las distancias, se pueden ordenar de menor a mayor distancia y se tendrán ordenados de mejor a peor los parches del clúster, pudiendo devolver los 20 mejores cuando se requiera. Este proceso se repite por cada clúster.

En este punto ya se tienen todos los parches agrupados en sus respectivos clústeres y ordenados de mejor a peor. Ahora se debe saber lo bueno o malo que es cada clúster.

- Calcular la calidad de cada clúster:
  Esto se produce debido a la dispersión que existe en cada clúster como consecuencia de la diferencia entre los parches que lo componen. Para calcular lo bueno o malo que es, se calcula la varianza que existe en cada clúster, permitiendo ordenarlos por este valor.

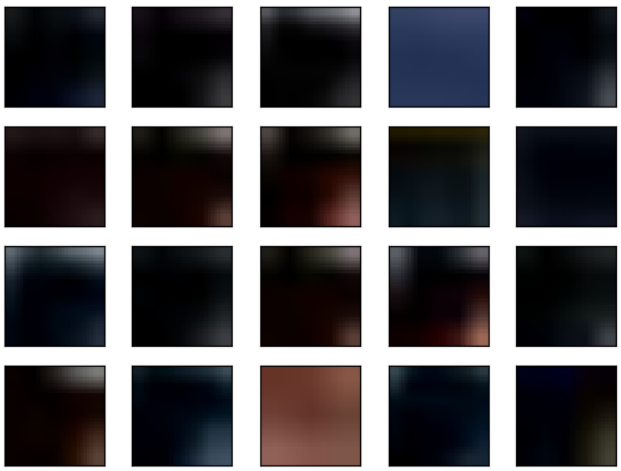
Ya se tienen todos los datos necesarios, por lo que se puede elegir una palabra buena o mala dependiendo de la posición que ocupe su clúster en la lista ordenada.

Durante la implementación se ha aprovechado para realizar los pasos expuestos anteriormente de manera agrupada, con el fin de aumentar la eficiencia del proceso.

- Palabra 555, clúster 555:

Se ha encontrado un clúster en el que los 20 mejores parches comparten forma geométrica y, aunque los parches se componen de dos colores, esta forma geométrica se puede apreciar en todos ellos.

- Palabra 43, clúster 43:



En este clúster predomina una zona negra, con colores en forma de lo que podría ser luces naranjas y blancas. La mayor parte de los parches comparten este patrón de colores y zonas en los que aparecen.

- Palabra 61, clúster 61:

En este clúster los parches no tienen mucha relación entre sí, mostrándose multitud de colores diferentes y formas no definidas entre unos y otros.

- Palabra 687, clúster 687:

Como en el clúster anterior, se compone de parches de varios colores y formas con poca relación entre ellos. Se puede apreciar incluso una letra en uno de los parches.

3. **Recuperación de imágenes**: Implementar un modelo de índice invertido + bolsa de palabras para las imágenes dadas en imagenerIR.rar usando el vocabulario calculado en el punto anterior. Verificar que el modelo construido para cada imagen permite recuperar imágenes de la misma escena cuando la comparamos al resto e imágenes de la base de datos. Elegir dos imágenes-pregunta en las que se ponga de manifiesto que el modelo usado es realmente muy efectivo para extraer sus semejantes y elegir otra imagen-pregunta en la que se muestre que el modelo puede realmente fallar. Para ello muestre las cinco imágenes más semejantes de cada una de las imágenes-pregunta seleccionadas usando como medida de distancia el producto escalar normalizado de sus vectores de bolsa de palabras. Explicar los resultados.

En este ejercicio se deben obtener una serie de imágenes similares a una seleccionada. Para desarrollarlo se han creado las siguientes funciones:

- *load_imgs(path)*: guarda en una lista todas las imágenes que se encuentren en una ruta dada.
- *get_same_imgs(img, imgs, dictionary, n)*: recibiendo una imagen, devuelve las *n* imágenes más parecidas.
- *get_histogram(dictionary, descriptors)*: calcula el histograma de una imagen dado un vocabulario y unos descriptores.
- *compare_histograms(hist1, hist2)*: compara dos histogramas y devuelve el valor correspondiente entre ambos siguiendo la fórmula vista en clase (y que se expone a continuación).

La operación más importante de este ejercicio es calcular un histograma. Calcular el histograma de cada imagen permite comparar histogramas entre sí para saber si coinciden o no y, de este modo, saber los que más relación tienen con la imagen-pregunta.

El primer paso es cargar en memoria todas las imágenes de la base de datos. Para esto se ha creado la función *load_imgs()* que devuelve una lista con todas las imágenes del directorio donde se encuentran.

El siguiente paso es cargar el fichero que contiene todas las palabras del vocabulario. Cuando ya se tiene el vocabulario, se calculan los descriptores de la imagen sobre la que se quiere obtener las imágenes más parecidas y se obtiene su histograma. El histograma es necesario para compararlo con todos los histogramas de todas las imágenes, por lo que se calcula en este punto y no es necesario volver a hacerlo.

Para calcular el histograma se ha creado la función *get_histogram()*, que recibe el vocabulario y los descriptores de la imagen-pregunta. El histograma tendrá el tamaño del número de palabras (500, 1.000, 5.000) y se calcula utilizando la función de *scipy cluster.vq.vq()*. Esta función devuelve dos vectores que tienen tantos elementos cada uno como descriptores tiene la imagen. El primer vector contiene el índice de la mejor palabra que le corresponde al descriptor que ocupa esa misma posición. El segundo vector contiene la distancia entre el descriptor y la palabra del diccionario. El vector que interesa es el que contiene el índice de la mejor palabra.

Cuando ya se sabe la palabra que corresponde a cada descriptor, se rellena el histograma sumando 1 en la posición a la que pertenece cada palabra.

El siguiente paso es calcular los histogramas de todas las imágenes de la base de datos.

Cuando ya se tienen todos los histogramas se comparan con el histograma de la imagen-pregunta y se ordenan de mejor a peor.

Para comparar dos histogramas se utiliza la siguiente fórmula vista en clase:

$$sim(d_j, q) = \frac{< d_j, q >}{||d_j||\,||q||}$$

Finalmente, el último paso es devolver las *n* mejores imágenes que se han obtenido.

- Imagen-pregunta (266.png):

Imágenes parecidas:





La imagen-pregunta pertenece a una escena muy característica con zonas muy oscuras, por lo que cuando se han calculado los histogramas, se han encontrado correspondencias con las palabras que forman las zonas oscuras de la escena, mostrando las 5 imágenes que corresponden con la imagen-pregunta con un alto grado de parecido.

- Imagen-pregunta (364.png):

Imágenes parecidas:





En esta ocasión, las imágenes encontradas también concuerdan perfectamente con la imagen-pregunta. El motivo se debe, probablemente, a que el elemento más característico de la escena es la camisa de cuadros, por lo que, al calcular los histogramas, la correspondencia de las palabras con zonas de la camisa de cuadros es amplia.

- Imagen-pregunta (62.png):

Imágenes parecidas:





En esta ocasión la imagen de la ciudad es la única que se encuentra en la base de datos con esas características y no ha podido encontrar ninguna que se le pueda parecer. Como datos a destacar, las 5 imágenes elegidas tampoco tienen relación entre sí, así como también se puede ver que las elegidas tienen el marco negro en las zonas superior e inferior como tiene la imagen-pregunta.