Trabajo 3

Indexación y Recuperación de Imágenes

22 de Diciembre de 2017

Gema Correa Fernández 75572158T

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

Ejercicio 1

Emparejamiento de descriptores [3 puntos]: Leer parejas de imágenes de imagenesIR.rar que tengan partes de escena comunes (p.e. (128,130), (143,145), (229,232), etc). 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 ello solo hay que fijar los vértices de un polígono que contenga a la región (ver explicación más abajo). 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 (ayuda: use el concepto de máscara del parámetro "mask"). Pinte las correspondencias encontrados 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.

Lo primero que haremos será escoger parejas de imágenes para extraer regiones comunes entre tales parejas.

Figura 1: Pareja 1

Figura 2: Pareja 2

Figura 3: Pareja 3

Figura 4: Pareja 4

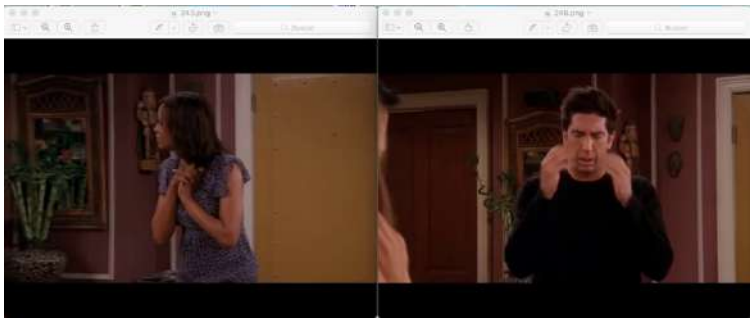


Figura 5: Pareja 5

Usaremos las anteriores imágenes en blanco y negro:

```
# Leemos las parejas de imágenes

pareja1_1 = leer_imagen(filename='imagenesIR/15.png', flag_color=False)
pareja1_2 = leer_imagen(filename='imagenesIR/16.png', flag_color=False)

pareja2_1 = leer_imagen(filename='imagenesIR/337.png', flag_color=False)
pareja2_2 = leer_imagen(filename='imagenesIR/338.png', flag_color=False)

pareja3_1 = leer_imagen(filename='imagenesIR/390.png', flag_color=False)
pareja3_2 = leer_imagen(filename='imagenesIR/391.png', flag_color=False)

pareja4_1 = leer_imagen(filename='imagenesIR/44.png', flag_color=False)
pareja4_2 = leer_imagen(filename='imagenesIR/49.png', flag_color=False)

pareja5_1 = leer_imagen(filename='imagenesIR/242.png', flag_color=False)
pareja5_2 = leer_imagen(filename='imagenesIR/246.png', flag_color=False)
```

Extraemos de la primera imagen (*imagen izquierda*) de la pareja, la región que queramos corresponder con la segunda imagen (*imagen derecha*) de la pareja. Para extraer los puntos que delimitarán nuestra región hacemos uso de la función proporcionada por el profesor `extractRegion()`. Con la cual, cogeremos los vértices de nuestro polígono:

```
# Selecciono una región de cada pareja de imágenes
# y extraigo los puntos que me interesan
extractRegion(pareja1_1)
extractRegion(pareja2_1)
extractRegion(pareja3_1)
extractRegion(pareja4_1)
extractRegion(pareja5_1)
```

Cogeré regiones de distintas formas, para comprobar el funcionamiento del ejercicio. Para la pareja 1, cogeré de la imagen 15 la siguiente región, correspondida con los siguientes puntos:

```
pareja1_puntos = np.array([[161, 211],[227, 204],[235, 258],
                           [227, 266],[227, 300],[174, 306],[159, 212]])
```

Figura 6: Región (Primera Pareja)

Para la pareja 2, cogeré de la imagen 337 la siguiente región, correspondida con los siguientes puntos:

```
pareja2_puntos = np.array([[208, 387],[347, 394],[341, 445],
                           [206, 442],[200, 391]])
```

Figura 7: Región (Segunda Pareja)

Para la pareja 3, cogeré de la imagen 390 la siguiente región, correspondida con los siguientes puntos:

```
pareja3_puntos = np.array([[113, 155],[140, 158],[149, 196],
                           [145, 205],[117, 201],[112, 155]])
```

Figura 8: Región (Tercera Pareja)

```
pareja4_puntos = np.array([[156, 180],[80, 250],[93, 421],
                           [95, 466],[159, 469],[123, 294],
                           [127, 233],[154, 180]])
```

Para la pareja 4, cogeré de la imagen 44 la siguiente región, correspondida con los siguientes puntos:

Figura 9: Región (Cuarta Pareja)

Para la pareja 5, cogeré de la imagen 242 la siguiente región, correspondida con los siguientes puntos (*la región que escojo es la cabeza del muñeco de la pared*):

```
pareja5_puntos = np.array([[226, 108],[242, 77],[264, 90],
                           [256, 128],[232, 156],[224, 113]])
```

Figura 10: Región (Quinta Pareja)

Una vez, que tenemos nuestra región seleccionada y guardada, comenzamos a implementar la función para este ejercicio.

`emparejamientoDescriptores`: con esta se función extraeremos los puntos SIFT contenidos en la región seleccionada de la primera imagen y calcularemos las correspondencias con todos los puntos SIFT de la segunda imagen. Para ello, pintaremos las correspondencias encontradas sobre las imágenes. Recibiremos por parámetro de entrada:

- La *imagen1* de la que extraigo la región.
- La *imagen2,* la cual tiene presente la región extraída de la imagen1.
- Los *puntos* que serán los vértices del polígono que contienen la región.

Primero inicializamos una máscara con las dimensiones de la imagen, ya que como máximo la región que podemos hacer es del tamaño de la imagen. Una vez inicializada nuestra máscara, tendremos que crearnos la máscara para ello tendremos que hacer uso de la función de OpenCV `fillConvexPoly` [1], que dibuja un polígono convexo lleno y se utiliza para ocultar todos los píxeles que están fuera del polígono, en nuestro caso de la región seleccionada. Con ello haremos que nuestra máscara tenga el tamaño de la región seleccionada. A continuación, extraeremos los puntos SIFT [2] contenidos en la región seleccionada de la primera imagen. Deberemos pasar la máscara para calcular sus puntos SIFT:

```
keypoints_imagen1, descriptors_imagen1 = sift.detectAndCompute
                                         (image=imagen1, mask=mask)
```

Extraemos también todos los puntos SIFT de la segunda imagen:

```
keypoints_imagen2, descriptors_imagen2 = sift.detectAndCompute
                                          (image=imagen2, mask=None)
```

Una vez obtenidos los puntos SIFT, obtendremos las correspondencias entre la región de la imagen1 y entre todos los puntos SIFT de la segunda imagen. Para ello, usamos un objeto de tipo `BFMatcher` [3] el cual obtendrá las correspondencias mediante fuerza bruta. Le pasamos a su constructor para que use la normal L2 y que use *cross-check* (validación cruzada). Con esto obtenemos un vector de correspondencias, el cual nos identifica los keypoints que hay relacionados entre ambas imágenes. Por último, ordenamos por mejor distancia y visualizamos las correspondencias que queramos entre ellas.

```
def emparejamientoDescriptores(imagen1, imagen2, puntos):

    # Inicializamos una máscara con las dimensiones de la imagen
    mask = np.zeros((imagen1.shape[0], imagen1.shape[1]),
                    dtype=np.uint8)

    # Crear la máscara que define el polígono de puntos
    # (región que hemos seleccionado)
    cv2.fillConvexPoly(mask, puntos, 1)

    # Usamos el descriptor SIFT
    sift = cv2.xfeatures2d.SIFT_create()

    # Extraigo los puntos SIFT contenidos en la región seleccionada
    # de la primera imagen (le paso la mascara)
    keypoints_imagen1, descriptors_imagen1 =
                       sift.detectAndCompute(image=imagen1, mask=mask)

    # Extraigo todos los puntos SIFT de la segunda imagen
    keypoints_imagen2, descriptors_imagen2 =
                       sift.detectAndCompute(image=imagen2, mask=None)

    # Calculo las correspondencias con todos los puntos SIFT de
    # la segunda imagen
    correspondecias = cv2.BFMatcher(normType=cv2.NORM_L2,
                                    crossCheck=True)
    matches = correspondecias.match(descriptors_imagen1,
                                    descriptors_imagen2)

    # Ordenamos las correspondencias por mejor distancia
    matches = sorted(matches, key=lambda x: x.distance)

    # Cogemos un numero de correspondecias extraidas para apreciar
    # la calidad de los resultados
    numero_correspondencias = 10

    # Pinto las correspondencias encontrados sobre las imágenes.
    resultado = cv2.drawMatches(img1=imagen1,
                        keypoints1=keypoints_imagen1, img2=imagen2,
                        keypoints2=keypoints_imagen2,
                        matches1to2=matches[:numero_correspondencias],
                        outImg=imagen2.copy(), flags=0)

    representar_imagenes([resultado],['Emparejamiento de Descriptores'])
```

Pareja 1

Figura 11: 10 correspondencias en la pareja 1

Visualizamos 10 correspondencias entre ellas, y como se ve tales correspondencias son adecuadas. ¿Pero qué pasaría si aumentáramos las correspondencias a 50?

Figura 12: 50 correspondencias en la pareja 1

Al aumentar las correspondencias, vemos que hay válidas y no válidas. Ya que tenemos puntos SIFT de una región pequeña, que queremos corresponder con todos los puntos SIFT de una imagen de 640 × 480.

Pareja 2

Figura 13: 10 correspondencias en la pareja 2

Visualizamos 10 correspondencias entre ellas, y como se aprecia, las letras coinciden bastante bien. Por lo que si reducimos las correspondencias, sabremos que obtendríamos un buen resultado.

Figura 14: 3 correspondencias en la pareja 2

Pero ahora probemos con 100 correspondencias. Debemos saber qué aunque aumentemos el número de correspondencias entre las imágenes, los puntos SIFT que hay en la región de la imagen 1, son limitados, por lo que no podremos pintar más correspondencias.

Figura 15: 100 correspondencias en la pareja 2

Como se aprecia, al aumentar las correspondencias entre ellas, se obtiene un peor resultado.

Pareja 3

Figura 16: 15 correspondencias en la pareja 3

Ahora hemos probado entre una región en un tamaño, y su mismo objeto en una escala mayor en la segunda imagen. Como en la región de la imagen tenemos menos puntos SIFT que en la segunda imagen, es probable que no obtengamos un resultado de calidad. Ya que disponemos de menos puntos SIFT en la primera imagen que en la misma región de la segunda imagen. Entonces, ¿que pasaría si reducimos las correspondencias?

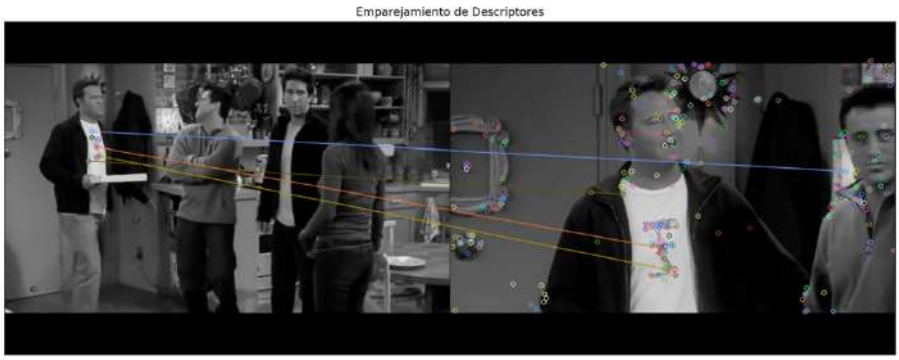


Figura 17: 3 correspondencias en la pareja 3

Con 3 correspondencias obtenemos la anterior salida. Pero seguimos en las mismas, estamos comparando zonas con distinta escala, ya que la misma región en la primera imagen tiene muchísimos menos puntos SIFT de los que aparecen en la segunda imagen. Cómo se ve hay dos *matches* válidas, por eso vamos a reducir el número de correspondencias a 2.

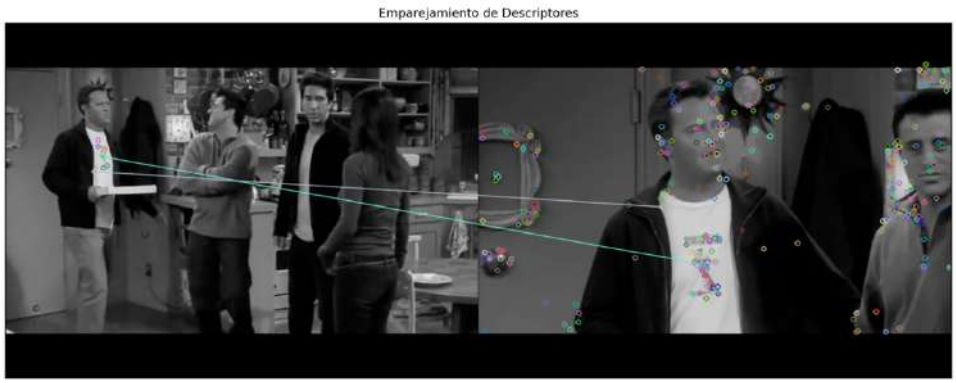


Figura 18: 2 correspondencias en la pareja 3

Como se ve en la salida anterior, aunque no estén bien los *matches*, ambos indican una zona blanca de la misma camiseta.

Pareja 4

Figura 19: 10 correspondencias en la pareja 4

En esta pareja, da igual el número de correspondencias que pongamos, ya que solo existen dos puntos SIFT para la región seleccionada. Además, vamos a comparar los puntos SIFT puestos en una dirección con otros puestos en otra orientación, es decir, estamos comparando un brazo con otra imagen que tiene el brazo en otra dirección. Además, como se ve en la primera imagen, la manga del jersey no tiene arrugas sin embargo, en la segunda sí tiene arrugas. Sin lugar a duda, la orientación es un factor a tener muy en cuenta a la hora de calcular correspondencias entre imágenes ya que con objetos de distinta orientación obtenemos unos resultados de muy mala calidad.

Pareja 5

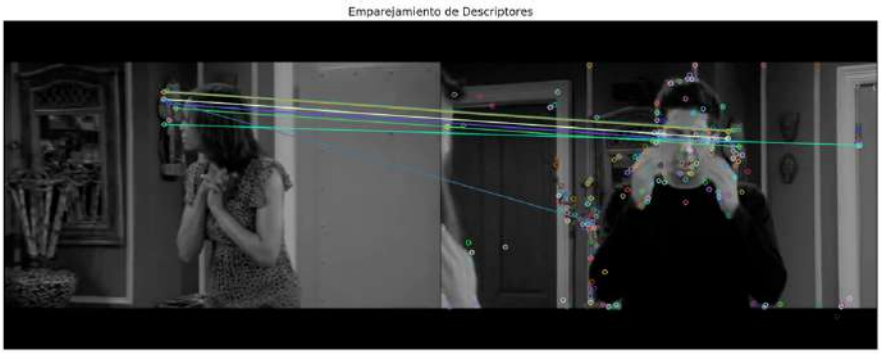


Figura 20: 10 correspondencias en la pareja 5

Como en la anterior pareja de imágenes, la primera imagen dispone de unos puntos limitados de la región, solo vamos a poder hacer *matching* con esos puntos SIFT. En este ejemplo, queremos ver la diferencia de comprobar un objeto que tiene más espacio ocupado en la imagen primera, que en la segunda imagen. Además, el objeto está detrás del personaje en todo momento. Vamos a disminuir el *matching* entre ambos.

Figura 21: 2 correspondencias en la pareja 5

Al disminuir, como hemos ido explicando a lo largo de la práctica, obtenemos un resultado de mayor calidad. Incluso, podríamos sacar de en clave, que las zonas que son más oscuras de una imagen, obtendríamos muy pocos puntos SIFT. Vamos a hacer el *matching* entre 7 correspondencias, se ve en la siguiente imagen que se obtiene una buena salida.

Figura 21: 7 correspondencias en la pareja 5

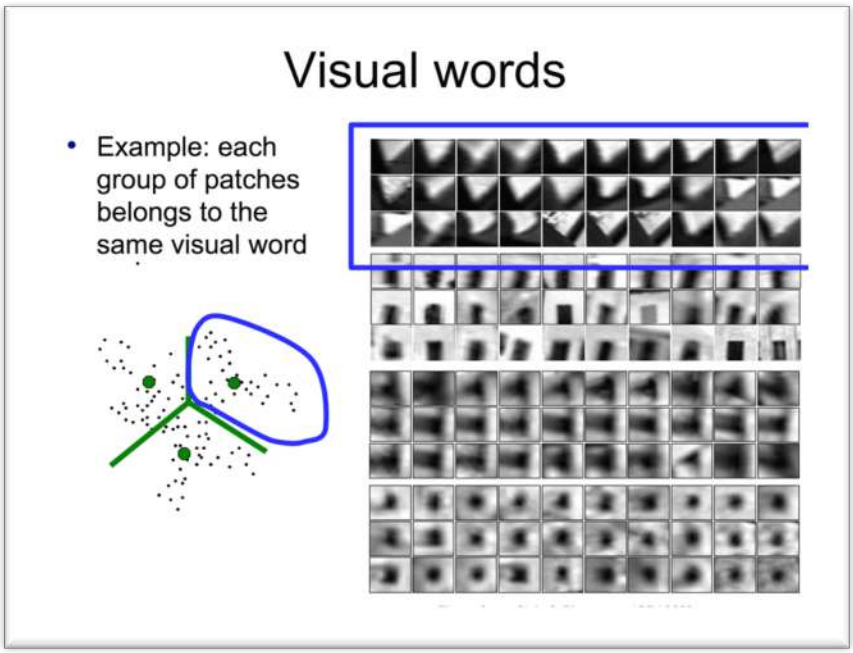
Como conclusión final, podríamos decir que cuántas menos correspondencias pongamos entre las imágenes, mejor calidad del resultado se obtendrá. Y pasará lo mismo a la inversa, que al aumentar las correspondencias se obtendrá un peor resultado. Además, la escala de un objeto no es un factor tan influyente a la hora de visualizar bien tales correspondencias, porque en general, la salida que se obtendrá será adecuada. En este caso en que la escala sea un factor, sí que será aconsejable usar un menor matching ya que los puntos SIFT que extraigamos de una región más pequeña serán menos que los que extraigamos del mismo objeto en una escala mayor. Sin embargo, la orientación es un factor a tener muy en cuenta a la hora de calcular las correspondencias entre imágenes ya que con objetos de distinta orientación obtenemos unos resultados de muy mala calidad.

Indistintamente podría haber realizado el ejercicio con las imágenes a color.

Ejercicio 2

Visualización del vocabulario [3 puntos]: Usando las imágenes dadas en imagenesIR.rar se han extraido 600 regiones de cada imagen y se ha construido un vocabulario de 5.000 palabras usando k-means. Se han extraido 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.

Para entender mejor lo que pide este ejercicio, hacemos uso del temario de teoría [4]:



La idea general de este ejercicio, consiste en sacar los clusters, escoger los de menor varianza, ya que serán más representativos y para cada cluster que escogemos, pintar los parches asociados a los mejores descriptores, es decir, lo que están más cerca al centroide del cluster. Para realizar este ejercicio, he hecho uso de la siguiente función.

`visualizacionVocabulario`: el objetivo de esta función es que partir de dos palabras visuales diferentes podamos visualizar las regiones imagen de los 20 parches más cercanos de cada palabra visual, de forma que se muestre el contenido visual que codifican. Lo primero que haremos será cargar el diccionario de 5000 palabras "**`kmeanscenters5000.pkl`**" con la función proporcionada por el profesor `loadDictionary()`, con ello obtendremos la exactitud, las etiquetas y el vocabulario:

```
# Cargamos el diccionario
accuracy, labels, dictionary = loadDictionary("kmeanscenters5000.pkl")
```

Luego cargamos los descriptores y sus parches extraídos de las imágenes.

```
# Cargamos los descriptores y sus parches extraidos de las imágenes
descriptors, patches = loadAux("descriptorsAndpatches.pkl", True)
```

Una vez que tenemos los descriptores, deberemos normalizarlos, para ello calculamos la norma a mano. A continuación, agrupamos los descriptores en función del centroide al que estén asociados, para ello hacemos uso de las etiquetas (`labels`), ya que las etiquetas y los descriptores se corresponden 1-1. Por ejemplo, la primera etiqueta representa el primer descriptor y el centroide al que está asociado. El contenido del *label* es un número comprendido entre el 0 y el 4999.

Una vez que ya tenemos un *multiarray* como (label, descriptor). Calculamos la distancia del descriptor a su centroide asociado, para ello hago la resta entre el descriptor en el que estoy con su respectiva etiqueta y nos guardamos su distancia. A tal distancia le hago la normalización. Volvemos a asociar la distancia del descriptor a su centroide asociado, teniendo (centroide, distancia). Ordenamos de menor a mayor los descriptores en función de la distancia obtenida anteriormente y nos quedamos con los 20 primeros, que serán los que más cerca se encuentren. Una vez que tengo los 20 descriptores más cercanos para cada centroide, calculo la varianza entre los mejores descriptores. Para ello hago uso de la función de NUMPY `np.var` [5], que calcula la varianza de un conjunto de un conjunto de elementos.

Ordenamos las varianza de menor a mayor valor, ya que nos interesan los de menor valor. Ya que una varianza elevada significa que los datos están más dispersos, mientras que un valor de varianza bajo indica que los valores están por lo general más próximos a la media, por lo tanto, tendríamos menos dispersión. Los *clusters* que están en el medio son los más interesantes.

Por último, elegimos al menos dos palabras visuales diferentes y visualizamos las regiones imagen de los 20 parches más cercanos de cada palabra visual, de forma que se muestre el contenido visual que codifican. Con eso lo que se nos quiere decir es que cojamos los dos centroides en los que hemos tenido una menor varianza en sus descriptores e imprima los 20 parches asociados. Con ello visualizamos las regiones imagen de los 20 parches más cercanos a cada palabra.

```
def visualizacionVocabulario ():

    # Cargamos el diccionario
    accuracy,labels,dictionary =loadDictionary("kmeanscenters5000.pkl")

    # Convierto los labels en una lista
    for x in labels:
        for y in x:
            labels_lista.append(y)

    # Cargamos los descriptores y sus parches extraidos de las imágenes
    descriptors, patches = loadAux("descriptorsAndpatches.pkl", True)
```

```
# Como los descriptores no están normalizados debemos normalizarlos
for i in range(0,descriptors.shape[0]):
    # Aplicamos la norma: elevar al cuadrado cada elemento
    # del descriptor, sumarlos y aplicarles la raíz cuadrada
    norm = np.sqrt(np.sum(descriptors[i]*descriptors[i]))
    # Normalizar el descriptor
    descriptors[i] = descriptors[i]/norm

# Agrupamos los descriptores en función del centroide al que estén
# asociados, a partir de las etiquetas (labels)
# Las etiquetas y los descriptores se corresponden 1-1, la primera
# etiqueta representa el primer descriptor
# y el centroide al que esta asociado (contenido del label es un
# numero del 0 al 4999)
for i,j in zip(labels_lista, descriptors):
    descriptors_centroide_aux = i,j
    descriptors_centroide.append(descriptors_centroide_aux)

# Calculo la distancia del descriptor a su centroide asociado, para
# ello hago la resta entre el descriptor en el
# que estoy con su respectiva etiqueta (guardo la distancia)
for i in range(0, len(descriptors):
    # np.linalg.norm(descriptor_en_cuestion - centroide_asociado)
    distancia_descriptor_labels_aux = np.linalg.norm(descriptors[i]
                                                     - labels[i])
    distancia_descriptor_labels.append(
                                    distancia_descriptor_labels_aux)

# Asocio la distancia del descriptor a su centroide asociado
for i,j in zip(labels_lista, distancia_descriptor_labels):
    descriptors_labels_aux = i,j
    descriptors_labels.append(descriptors_labels_aux)

# Ordenar estos descriptores en función de su distancia al
# centroide asociado (de menor a maypr)
descriptors_labels = sorted(descriptors_labels,
                            key=lambda a_entry: a_entry[1])

# Me quedo con los 20 más cercanos, que serán los que más cerca
# se encuentren
descriptors_mas_cercanos = descriptors_labels[:20]

# Una vez que tengo los 20 descriptores mas cercanos para cada
# centroide, calculo la varianza entre los descriptores
# Cojo las mejores varianzas y de esos centroides imprimes los
# parches asociados
# Una varianza elevada significa que los datos están más dispersos,
# mientras que un valor de varianza bajo indica
# que los valores están por lo general más próximos a la media
for i in descriptors_mas_cercanos:
    for j in descriptors_centroide:
        if i[0] == j[0]:
            varianza_descriptors.append(j)

# Asocio la varianza del descriptor con su centroide asociado
for i in range(len(varianza_descriptors)):
    var_centroide_aux = varianza_descriptors[i][0]
    var_centroide.append(var_centroide_aux)
    var_descriptors_aux=np.mean(np.var(varianza_descriptors[i][1]))
    var_descriptors.append(var_descriptors_aux)
```

```
    for i,j in zip(var_centroide, var_descriptors):
        varianza_final_aux = i,j
        varianza_final.append(varianza_final_aux)

    # Ordenamos las varianza de menor a valor valor
    # (cogemos las mejores varianzas)
    varianza_final = sorted(varianza_final,
                            key=lambda a_entry: a_entry[1])

    '''
    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.
    '''
    # Coger los dos centroides en los que he tenido una menor varianza
    # en sus descriptores

    # Elijo el primer centroide e imprimo los parches asociados
    centroide1 = varianza_final[0][0]

    # Para el centroide 1 obtengo los indices de los descriptores e
    # imprimo los 20 parches asociados a ellos
    for i in varianza_descriptors:
        if i[0] == centroide1: descriptors_centroide1 = i[1]

    # Cogemos los 20 mejores
    descriptors_centroide1 = np.argsort(descriptors_centroide1)[:20]

    for i in descriptors_centroide1:
        descriptors_centroide1_mejores.append(i)

    # Visualizar las regiones imagen de los 20 parches más cercanos
    # a cada palabra
    for i in descriptors_centroide1_mejores:
        imagenes.append(patches[i])
        titulos.append("")

    representar_imagenes(imagenes,titulos)

    # Elijo un centroide e imprimo los parches asociados
    centroide2 = varianza_final[1][0]

    # Para el centroide 1 obtengo los indices de los descriptores e
    # imprimo los 20 parches asociados a ellos
    for i in varianza_descriptors:
        if i[0] == centroide2: descriptors_centroide2 = i[1]

    # Cogemos los 20 mejores
    descriptors_centroide2 = np.argsort(descriptors_centroide2)[:20]

    for i in descriptors_centroide2:
        descriptors_centroide2_mejores.append(i)

    # Visualizar las regiones imagen de los 20 parches más cercanos a
    # cada palabra
    for i in descriptors_centroide2_mejores:
        imagenes.append(patches[i])
        titulos.append("")

    representar_imagenes(imagenes,titulos)
```

Si escogemos el centroide donde hemos obtenido una mejor varianza:

Viendo la imagen anterior, nos podemos hacer una idea que lo que cae en ese *cluster*. Para comparar los parches entre sí, debemos comprobar que el color sea parecido, pero no es un requisito necesario, en sí, o tan importante como otros. Será muy importante para ver la calidad, basarnos en la iluminación o en las formas que conseguimos apreciar. Como se puede ver, el 90% de los parches tienen tendencia a una iluminación en medio de la región en dirección vertical.

Si escogemos el centroide donde hemos obtenido la segunda mejor varianza:

Viendo la imagen anterior, nos podemos hacer una idea de lo que cae en ese *cluster*. Como se ve en los parches, se sique una tendencia de iluminación en las esquinas inferiores.

Ejercicio 3

Recuperación de imágenes [3 puntos]: Implementar un modelo de índice invertido + bolsa de palabras para las imágenes dadas en imágenesIR.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 de imágenes de la base de datos. Elegir dos imágenes-pregunta en las 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.

La idea general de este ejercicio consiste en sacar para imagen sus descriptores y ver a que palabra se asemejan más estos descriptores. Con eso calculamos las semejanzas de todas las imágenes y obtenemos los histogramas de cada imagen con los votos a las palabras del diccionario.

Nota: *Por ejemplo, una imagen de un edificio contiene ventanas. Estos elementos visuales se llamarán palabras visuales. El conjunto de palabras visuales constituye el diccionario.*

Una vez hecho esto, tendremos para cada imagen las palabras que aparecen y cuántas veces aparecen, así construimos los histogramas. Luego construimos el fichero invertido y pasamos de tener para cada imagen una lista de palabras, a tener para cada palabra una lista de imágenes. Pues ahora con una imagen nueva, repetimos el proceso, y calculamos la semejanza, obteniendo las palabras que salen en esa imagen. Y con esas palabras, nos vamos al diccionario y sacamos todas las imágenes que las contienen. Se hace un recuento y nos quedamos con las palabras que más veces se repiten. A los histogramas de la bolsa de palabras, les calculamos la semejanza con el de la imagen concreta. Y así sacamos que imágenes se parecen más a la que hemos elegido como imagen consulta. A continuación, explicaré el proceso más detalladamente, para ello he implementado la siguiente función:

`recuperacionImagenes`: el objetivo de esta función es que a partir de una imagen pregunta, obtengamos las imágenes más semejantes de nuestra base datos. Para ello recibiremos por parámetro de entrada:

- La *imagen_test* que será la imagen pregunta.

Lo primero que haremos será cargar el diccionario de 5000 palabras "**`kmeanscenters5000.pkl`**" con la función proporcionada por el profesor `loadDictionary()`, con ello obtendremos la exactitud, las etiquetas y el vocabulario:

```
# Cargamos el diccionario
accuracy, labels, dictionary = loadDictionary("kmeanscenters5000.pkl")
```

Debemos normalizar el diccionario, ya que para calcular las semejanzas haremos uso de la fórmula [6]:

$$sim(d_j, q) = \frac{\langle d_j, q \rangle}{\|d_j\| \|q\|}$$

donde q es la palabra y d_j es el descriptor.

Una vez normalizado el diccionario, nos creamos un bucle donde recorreremos las 441 imágenes que tenemos en **imagenesIR.rar**. Extraemos de cada imagen sus descriptores SIFT y calculamos la normalización a tales descriptores. Por lo tanto, ya tengo dos matrices, una matriz será la de palabras/vocabulario (`dictionary`) y la otra matriz será la de los descriptores, en el que cada descriptor va por fila.

A continuación, calculamos el numerador de la fórmula de la semejanza con la operación de Numpy `np.dot` [7], el cual calcula el producto de dos matrices. Pero deberemos tener cuidado, ya que hay que transponer los descriptores para hacer la multiplicación ya que la dimensión del diccionario es $(5000, 128)$ y la del es descriptor $(\text{n\_descriptores}, 128)$.

```
semejanza_denominador = np.dot(dictionary, descriptors_images.T)
```

Con todo esto, ya podemos calcular nuestra semejanza:

```
# Calculo la fórmula de la semejanza ({dj,q}) / (||dj|| ||q||)
semejanza_imagenes = np.divide(semejanza_denominador,
                               norma_diccionario[:,None])
```

Una vez calculada mi matriz de semejanzas, busco el máximo para cada columna, con esto relaciono las imágenes y, calculo el histograma para cada palabra y veo sus votos con la función de Numpy `np.bincount` [8], que cuenta el número de ocurrencias de cada valor en el conjunto de entradas. Con esto, ya puedo construir mi fichero invertido, que consiste en sacar de cada histograma a que imagen va. Pero para saber si esa palabra ha votado, debemos coger aquellas palabras que tengan voto distinto a cero.

Pues bien, ya tenemos para cada palabra asociado su lista de imágenes. Ahora paso a realizar el mismo proceso que antes, pero para la imagen-pregunta. Es decir, extraigo los descriptores SIFT de la *imagen_pregunta*, y los normalizo. Aplico la misma fórmula de semejanza que antes y una vez calculada mi matriz de semejanzas, busco el máximo para cada columna. Calculo el histograma para cada palabra y veo sus votos, cojo en el histograma los votos que tienen un valor distinto de 0 y cojo los índices de las imágenes de mi histograma test.

Una vez hecho esto, vamos a seleccionar las cinco imágenes más semejantes a la *imagen_pregunta*, para ello me voy al *fichero_invertido* y recupero las imágenes correspondiente a las palabras del histograma de la imagen pregunta. Obtengo las imágenes y me quedo con las imágenes más repetidas. Luego para cada imagen de la lista de imágenes comparo el histograma de la bolsa de palabras (*histogramas_imagenes*) con el histograma de mi imagen pregunta (*histograma_imagen_test*). Y comparo las semejanzas entre ellas, y me quedo con las que tengan un valor mayor. Por último, ya solo me queda representar las imágenes.

Veamos un pequeño ejemplo de cómo obtendríamos las imágenes más semejantes a una de consulta: Partimos de nuestro histograma consulta, por ejemplo: [0:3, 1:4, 7:1] donde [0:3] significa [palabra, contador(veces que se repite)]. Nos vamos al fichero invertido y recuperamos las imágenes fichero_invertido[0], fichero_invertido[1] y fichero_invertido[7]. Obtenemos una lista de imágenes por ejemplo [0,1,2]. Para cada imagen de nuestra lista, compararmos histogramas_fichero_invertido[lista_imagenes] con el histograma_consulta. Si esa comparación supera un umbral, mostramos las 5 mejores imágenes.

```
def recuperacionImagenes(image_test):

    # Cargamos el diccionario (el diccionario ya está normalizado)
    accuracy,labels,dictionary = loadDictionary("kmeanscenters5000.pkl")

    # Normalizamos el diccionario como pone en la fórmula de la página
    # 17 del tema "13.instances_recog.pdf"
    for i in range (0, dictionary.shape[0]):
        norma_diccionario.append(np.sqrt(np.sum(
                                dictionary[i]*dictionary[i])))

    # Nuestra norma_diccionario es una lista con 5000 elementos,
    # lo convierto en vector
    norma_diccionario = np.asarray(norma_diccionario, dtype=np.float32)

    # Recorremos las 441 imágenes que tenemos en "imagenesIR.rar"
    for i in range(0,441):

        # Leemos todas las imagenes
        name = "imagenesIR/" + str(i) + ".png"
        image = cv2.imread(name)

        # Extraemos los descriptores SIFT de todas las imágenes
        sift = cv2.xfeatures2d.SIFT_create()
        keypoints_images, descriptors_images =
                        sift.detectAndCompute(image=image, mask=None)

        # Normalizamos los descriptores
        for i in descriptors:
            i = np.linalg.norm(i)

        # Hacemos el denominador de la fórmula de la semejanza {dj,q}
        # (pagina 17 en "13.instances_recog.pdf")
        # Obtengo una matriz donde cada fila es una palabra y cada
        # columna es un descriptor (5000 x descriptores)
        # Transpongo los descriptores para hacer la multiplicacion
        # dicionario: (5000, 128); descriptor: (n_descriptores, 128)
        semejanza_denominador = np.dot(dictionary, descriptors_images.T)
        # Calculo la fórmula de la semejanza ({dj,q}) / (||dj|| ||q||)
```

```
        semejanza_imagenes = np.divide(semejanza_denominador,
                                       norma_diccionario[:,None])

        # Una vez calculada mi matriz de semejanzas, busco el máximo
        # para cada columna (relacionar)
        relaciones_imagenes = np.argmax(semejanza_imagenes, axis=0)

        # Calculo el histograma para cada palabra y veo sus votos con
        # "np.bincount"
        histograma = np.bincount(relaciones_imagenes, weights=None,
                                 minlength=5000)
        histograma_imagenes.append(histograma)

    # Convierto en array "histograma_imagenes"
    histograma_imagenes = np.array(histograma_imagenes)

    # Construyo el fichero invertido (sacar de cada histograma a que
    # imagen va). Por ejemplo, la imagen 0 tiene (palabras columnas)
    # Cogemos de cada palabra, las imágenes que tienen voto distinto a 0
    for i in range (0,5000):
        palabras = np.where(histograma_imagenes[:,i] != 0)[0]
        fichero_invertido.append(palabras)

    # Ya tengo para cada palabra asociado su lista de imágenes
    # (fichero_invertido)
    # Ahora paso a realizar el mismo proceso que antes, pero para la
    # imagen-pregunta leer la imagen test, la de consulta

    # Extraemos los descriptores SIFT de la imagen_pregunta
    sift = cv2.xfeatures2d.SIFT_create()
    keypoints_image_test, descriptors_image_test =
                      sift.detectAndCompute(image=image_test, mask=None)

    # Normalizamos los descriptores
    for i in descriptors_image_test:
        i = np.linalg.norm(i)

    # Hacemos el denominador de la fórmula de la semejanza {dj,q}
    # Obtengo una matriz donde cada fila es una palabra y cada columna
    # es un descriptor (5000 x descriptores)
    # Transpongo los descriptores para hacer la multiplicacion
    # dicionario: (5000, 128); descriptor: (n_descriptores, 128)
    semejanza_denominador_test = np.dot(dictionary,
                                        descriptors_image_test.T)

    # Calculo la fórmula de la semejanza ({dj,q}) / (||dj|| ||q||)
    semejanza_image_test = np.divide(semejanza_denominador_test,
                                     norma_diccionario[:,None])

    # Una vez calculada mi matriz de semejanzas, busco el máximo
    # para cada columna (relacionar)
    relacion_image_test = np.argmax(semejanza_image_test, axis=0)

    # Calculo el histograma para cada palabra y veo sus votos
    histograma_test = np.bincount(relacion_image_test, weights=None,
                                  minlength=5000)

    # Cojo en el histograma los votos que tienen un valor distinto de 0
    for i in range(0,5000):
        palabras_image_test.append(np.where(histograma_test[i] != 0)[0])
    # Cojo los índices de las imágenes de mi histograma test
```

```
    for i in histograma_test:
        if i != 0: indices_image_test.append(cont)
        cont = cont+1

    # Una vez esto, vamos a seleccionar las cinco imágenes más
    # semejantes de cada una de las imágenes-pregunta. Tengo creado mi
    # histograma_test, así que voy al fichero_invertido y recupero las
    # imágenes correspondiente a las palabras de mi histograma
    for i in indices_image_test:
        recuperar_imagenes.append(fichero_invertido[i])

    # Obtengo la lista de imágenes
    recuperar_imagenes = np.concatenate(recuperar_imagenes)

    # Nos quedamos con las imágenes más repetidas
    # y cogemos las 50 primeras
    imagenes_final = imagenes_repetidas[:50]

    for i in imagen_final:
        comparar_histogramas = np.dot(histograma_imagenes[i],
                                      histograma_test)
        if comparar_histogramas > umbral: mostrar_imagenes.append(i)

    # Me quedo con las 5 imagenes semejantes
    imagenes_finales = mostrar_imagenes[:5]

    # Vamos a visualizar las imágenes
    # Metemos la imagen TEST
    lista_imagenes.append(image_test)
    lista_titulos.append("Imagen Pregunta")

    for i in imagenes_finales:
        nombre = "imagenesIR/" + str(i) + ".png"
        img = cv2.imread(nombre,1)
        lista_imagenes.append(img)
        lista_titulos.append("")

    # Representamos las imagenes
    representar_imagenes(lista_imagenes,lista_titulos)
```

Antes de ponerme a ejecutar la función, pensemos en que imágenes nos pueden dar mejores resultados. De primeras supondremos que serán aquellas imágenes que tengan cosas representativas, como una camisa hawaiana. Ya que los gradientes buscan en las derivadas donde hay cambio. Sin embargo, suponemos que encontraremos peores resultados cuando la imagen sea más uniforme, es decir, cuando por ejemplo una persona lleve una camisa del mismo color que el sofá, ya que será más fácil encontrar regiones que tienen cambios.

Imagen 86

```
image_test = cv2.imread("imagenesIR/86.png",1)
```

Para la imagen anterior, comprobamos que hemos obtenido una muy buena salida. Y como es normal, la imagen que más se asemeja es ella misma. Hemos obtenido un buen resultado escogiendo una imagen donde se nota que hay cambios en las regiones, vamos a usar otra camisa en la que también haya cambios en las regiones.

Imagen 333

```
image_test = cv2.imread("imagenesIR/333.png",1)
```

Para la imagen anterior, coincidimos con la conclusión a la que hemos llegado anteriormente. Ahora vamos a escoger otro tipo de imagen, por ejemplo, veamos si coge bien que haya texto escrito en un fondo negro.

Imagen 202

```
image_test = cv2.imread("imagenesIR/202.png",1)
```

Como se ve el resultado es óptimo, ya que en nuestra BD, habrá pocas fotos que tengan esas características específicas. Acabamos de comprobar que los resultados obtenidos con el diccionario de 5000, son de alta calidad. Entonces, nos preguntamos, ¿habrá alguna imagen donde el modelo pueda fallar? Sin llegar a concretar un ejemplo, podemos suponer que fallará cuando estemos cogiendo una imagen donde no hay otra igual en la BD. Comprobémoslo.

Imagen 36

Imagen 62

¿Qué ha pasado en las imágenes? Pues qué no hay ninguna imagen parecida en nuestra BD, ya que no se puede buscar algo parecida, para una imagen que es única. Pero ahora nos preguntamos, ¿y si yo no sé qué imágenes hay en la BD? Pues muy sencillo, por ejemplo, en la imagen 36, la de la puerta, esa imagen dispone de pocas palabras visuales (solo tiene una puerta), así que seguramente muchos de sus puntos SIFT se correspondan con la misma palabra.

Una vez que hemos hecho, esta comparación, vamos a coger la imagen 86 ya que obteníamos un muy buen resultado con ella y probamos otro diccionario (Usaremos la misma imagen y el mismo umbral). Probaremos un diccionario con 1000 palabras y otro con 500 palabras.

Imagen 86

Kmeanscenter5000

Kmeanscenter1000

Kmeanscenter500

Como se aprecia, conforme el vocabulario baja, las imágenes que se asemejan con la imagen consulta pierden parecido. Esto es así, ya que para reconocer muchas imágenes u objetos el vocabulario debe ser grande. Por eso con el diccionario de 5000 obtenemos tan ben resultado.

Pero por ejemplo, cojamos la imagen 202, la cual, se obtiene un buen resultado con el diccionario de 5000. En esta imagen solo tenemos que distinguir que hay texto escrito en blanco en un fondo negro.

Imagen 202

Kmeanscenter5000

Kmeanscenter500

Obtenemos el mismo resultado, para ambos diccionario, simplemente cambia el orden de aparición. Pero siempre aparece como primera imagen semejante la misma.

Bonus

Ejercicio 1

Recuperación de regiones [2 puntos]: Usando la misma función del apartado.1 seleccione una región rectangular que le parezca relevante de una imagen del conjunto imagenesIR.zip y calcule su modelo de bolsa de palabras ponderando las palabras con los pesos calculados por el criterio tf-idf.

Apartado a

Use esa imagen para extraer las 20 imágenes más cercanas a ella en la base de datos usando el punto anterior y verifique que en un porcentaje razonable de ellas aparece la región seleccionada aunque en otros contextos. Use un esquema de ventana deslizante sobre las 20 imágenes seleccionadas usando el mismo tamaño de la región seleccionada.

Apartado b

Recorra las 20 imágenes usando un modelo de ventana deslizante y calcule en cada posición de la ventana un modelo de bolsa de palabras. Mida la distancia de los modelos de bolsa de palabras (ventana y región seleccionada) y guarde las coordenadas de la región y su distancia asociada. Muestre en orden las 20 regiones más cercanas a la región seleccionada. Que conclusiones extrae de sus resultados.

Ejercicio 2

Consistencia espacial [2 puntos]. Implementar la verificación espacial afín a través del método RANSAC al problema del bonus anterior. Probar con ejemplos la mejora que supone frente al no uso de verificación espacial. (Leer el paper de Sivic and Zisserman http://www.robots.ox.ac.uk/~vgg/publications/papers/sivic09a.pdf para entender bien la técnica)

Ejercicio 3

Creación de un vocabulario [2 punto]: Calcular desde todas las imagenesIR.rar los ficheros dados en el punto-2 usando los mismos parámetros. Aplicar con el nuevo diccionario lo pedido con el punto-2

Referencias

[1] Función de OpenCV `fillConvexPoly`
https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html

[2] Función de OpenCV `SIFT.detectAndCompute`
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html

[3] Feature Matching
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html

[4] Diapositiva 9, Tema 13 "Object Instance Recognition"

[5] Función `numpy.var`
https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.var.html

[6] Diapositiva 17, Tema 13 Tema 13 "Object Instance Recognition"

[7] Función `numpy.dot`
https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.dot.html

[8] Función `numpy.bincount`
https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.bincount.html