
Visión por Computador

Práctica 3: Trabajo de implementación

Francisco Solano López Rodríguez

1. Emparejamiento de descriptores.

- Mirar las imágenes en `imagenesIR.rar` y elegir parejas de imágenes 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 ello 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 (ayuda: use el concepto de máscara con el parámetro `mask`).
- Pinte las correspondencias encontradas sobre las imágenes.
- Jugar con distintas parejas de imágenes, valorar las correspondencias correctas obtenidas y extraer conclusiones respecto a la utilidad de esta aproximación de recuperación de regiones/objetos de interés a partir de descriptores de una región.

Para este ejercicio he reutilizado las funciones de la práctica anterior para obtener los descriptores y hallar las correspondencias.

```
# Obtener keypoints y descriptores
def get_keyPoints_and_descriptors(img, mask=None):
    sift = cv.xfeatures2d.SIFT_create()

    return sift.detectAndCompute(img, mask)

# Obtener correspondencias
def getMatchesKNN(desc_1, desc_2, k_=2):
    bf = cv.BFMatcher()
    matches = bf.knnMatch(desc_1, desc_2, k=k_)

    good = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            good.append([m])

    return good
```

Para obtener las correspondencias entre dos imágenes, he implementado la función `draw_matches_mask` a la cual se le pasa como parámetros las dos imágenes (`img1, img2`) y además los puntos (`points`) que definen los vértices de la región de la que calcularemos los puntos SIFT en la primera imagen. Con estos vértices construiremos una máscara que definirá la región deseada, para ello he utilizado la función de OpenCV `fillConvexPoly`, la cual define un polígono convexo cuyos vértices se corresponden con los datos en el parámetro `points`.

Tras obtener la máscara se obtienen los descriptores de la región definida por dicha máscara en la imagen 1. Después obtenemos los descriptores de la segunda imagen y calculamos las correspondencias. Tras esto dibujamos ambas imágenes junto con las correspondencias obtenidas.

```
# Obtener las correspondencias entre una región de la primera imagen y la
segunda imagen
def draw_matches_mask(img1, img2, points):
    np.set_printoptions(threshold=np.nan)
    # Máscara
    mask = np.zeros(shape=(np.shape(img1)[0], np.shape(img1)[1]))
    print(mask)
    cv2.fillConvexPoly(mask, np.array(points, dtype=np.int32), color=1)
    #print(mask)
```

```

# Descriptores
kp1, desc1 = get_keyPoints_and_descriptors(img1, mask=np.array(mask,
    dtype=np.uint8))
kp2,desc2 = get_keyPoints_and_descriptors(img2)

# Correspondencias
matches = getMatchesKNN(desc1, desc2)

# Dibujamos correspondencias
img_matches_2 = cv.drawMatchesKnn(img1, kp1, img2, kp2, matches1to2=
    matches,outImg=None,flags=2)
display_image(img_matches_2)

```

```

# prueba 1
img1 = cv2.imread("imagenes/64.png")
img2 = cv2.imread("imagenes/65.png")
points = [(470,125), (620, 125), (620, 330), (470, 330)]
draw_matches_mask(img1, img2,points)

```



```

# prueba 2
img3 = cv2.imread("imagenes/57.png")
img4 = cv2.imread("imagenes/58.png")
points_2 = [(10,90), (120,90), (120,220), (10,220)]
draw_matches_mask(img3, img4, points_2)

```



```

# prueba 3
img5 = cv2.imread("imagenes/54.png")
img6 = cv2.imread("imagenes/55.png")
points_3 = [(40,90), (250,90), (250,240), (40,240)]
draw_matches_mask(img5, img6, points_3)

```



Como podemos observar en todos los casos los resultados obtenidos han sido bastante buenos, incluso a pesar de haber cambios de escala y punto de vista en la prueba 1, e incluso oclusión como ocurre con la cabeza del personaje al tapar las tazas en la prueba 3. Esto es gracias a que los descriptores SIFT son robustos frente al tipo de deformaciones como los cambios de vista, escalados, giros, etc. Luego en este tipo de casos esta técnica funciona bien.

2. Recuperación de imágenes

- Implementar un modelo de índice invertido + bolsa de palabras para las imágenes dadas en `imagenesIR.rar` usando el vocabulario dado en `kmeanscenters2000.pkl`.
- 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 qué conclusiones obtiene de este experimento.

Para este ejercicio he creado 3 funciones:

- `get_histogram(img_name, dictionary, dictionary_norm)`
- `get_inverted_file_index(dictionary_name)`
- `retrieval_images(image_name, inverted_file, dictionary_name, histograms)`

A continuación explico la lógica de cada función.

```
def get_histogram(img_name, dictionary, dictionary_norm):
    # Leer imagen
    img = cv2.imread("imagenes/" + img_name)
    # Obtener descriptores
    kp, desc = get_keyPoints_and_descriptors(img)
    desc_norm = np.apply_along_axis(np.linalg.norm, 1, desc)

    # Calcular similaridad y normalizar
    similarities = np.dot(dictionary, desc.T)
    similarities_norm = np.divide(similarities, desc_norm * dictionary_norm[:, None])

    histogram = Counter(np.argmax(similarities_norm, axis=0))

    return histogram
```

Con esta función obtenemos un histograma calculado a partir de los descriptores de la imagen y el diccionario. Para ello usamos la fórmula de similaridad vista en las transparencias de clase: $\frac{\langle d_j, q \rangle}{\|d_j\| \|q\|}$, la cual da un valor entre 0 y 1 (en realidad entre -1 y 1 pero como los vectores de los que disponemos no contienen números negativos el valor obtenido siempre será positivo, esta fórmula en realidad es la fórmula del coseno). Para ello hemos multiplicado la matriz del diccionario por la matriz de los descriptores y después se ha dividido por el producto de las normas del diccionario y el descriptor. Por último se ha obtenido el histograma obteniendo los índices de los valores máximos y después usando `Counter` para obtener una lista de parejas de índices y número de apariciones de estos índices. Así nuestro histograma estará compuesto por los índices de las palabras del diccionario y el número de repeticiones de estos índices.

Con la siguiente función obtenemos el fichero de índices invertidos.

```
def get_inverted_file_index(dictionary_name):

    accuracy, labels, dictionary = loadDictionary(dictionary_name)
    dictionary_norm = np.apply_along_axis(np.linalg.norm, 1, dictionary)
    histograms = []

    inverted_file = collections.defaultdict(list)

    for img in range(441):
        img_name = str(img) + ".png"
        histogram = get_histogram(img_name, dictionary, dictionary_norm)
        histograms.append(histogram)

        for i in histogram:
            inverted_file[i].append(img)

    return dict(inverted_file), histograms
```

Primero se ha usado la función aportada para la práctica `loadDictionary`, con la que cargamos el diccionario, después iteramos todas las imágenes y por cada imagen obtenemos su histograma, usando la función anteriormente definida. Después recorremos el histograma para obtener los índices de las palabras encontradas en la imagen, los cuales van a ser utilizados como claves en nuestro diccionario. El valor que añadimos en esta clave del diccionario es la imagen asociada al histograma que estamos recorriendo. Así el resultado final será un diccionario cuyas claves serán los índices de las palabras y el valor asociado será un vector de imágenes en las cuales se encontrada esta palabra.

La siguiente función es usada para la recuperación de las imágenes, para ello pasamos por parámetros la imagen de la cual queremos obtener las imágenes con mayor similaridad a ella, el fichero de índices invertidos, el diccionario y los histogramas con los que se realizará la comparación.

```
def retrieval_images(image_name, inverted_file, dictionary_name, histograms):
    :

    histogram = histograms[int(os.path.splitext(image_name)[0])]
    histogram_image = [histogram[k] for k in range(2000)]

    images = []

    for i in histogram:
        for img in inverted_file[i]:
            images.append(img)

    # Obtenemos las imágenes junto al número de apariciones
```

```

images_names, _ = np.unique(images, return_counts=True)

for i in images_names:
    histogram_images_i = [histograms[int(i)][k] for k in range(2000)]

    similarity = np.dot(histogram_images_i, histogram_image)
    n1 = np.linalg.norm(histogram_images_i)
    n2 = np.linalg.norm(histogram_image)
    similarity = similarity/(n1*n2)

    values.append(similarity)

values = np.array(values)
index_sort = np.argsort(-values)

images_names_sorted = images_names[index_sort]

display_image(cv2.imread("imagenes/" + image_name))

for i in images_names_sorted[0:5]:
    display_image(cv2.imread("imagenes/" + str(i) + ".png"))

```

Lo primero que hacemos es obtener el histograma de la imagen pasada, esta vez no hay que calcularlo ya que lo tenemos en el vector de histogramas histograms. Despu s obtenemos todas la im genes que contengan palabras que contiene nuestra imagen. Luego recorremos las im genes obtenidas y obtenemos la similaridad haciendo el producto escalar de los histogramas de estas im genes con nuestra imagen y dividiendo por el producto de las normas de los histogramas. Por  ltimo nos quedamos con las 5 im genes que tengan una mayor valor de similaridad (cuanto mayor sea el valor mayor similaridad habr  y el valor m ximo que puede obtenerse es 1 que se obtiene cuando los histogramas son id nticos) y mostramos dichas im genes.

```

print("Obteniendo indice invertido...")
inverted_file, histograms = get_inverted_file_index(dictionary_name)
print("Obtenido con éxito.")

# imagen-pregunta 1
retrieval_images("353.png", inverted_file, histograms)
# imagen-pregunta 2
retrieval_images("115.png", inverted_file, histograms)
# imagen-pregunta 3
retrieval_images("4.png", inverted_file, histograms)

```

Imagen-pregunta 1:





Imagen-pregunta 2:



Imagen-pregunta 3:





Como podemos ver en las pruebas realizadas el modelo a funcionado razonablemente bien, en especial en la prueba 1, en la que el personaje aparece solo con un fondo morado y azul en algunos casos y parece que es la más fácil para la recuperación de imágenes.

En la segunda prueba parece que también funciona bien, sin embargo podemos ver una imagen intrusa. Efectivamente la penúltima imagen parece no encajar con las demás, el problema en este caso es que esta imagen tenía algo en común con las demás, la persiana y debido a ella parece que ha tomado esta imagen como similar. El problema en este caso puede deberse a que el diccionario de palabras con el que contábamos no tiene demasiada riqueza de palabras.

Por último la tercera prueba en la que aparecen Mónica y Chandler vemos que los resultados no han sido muy buenos, ya que a primera vista no parece que las imágenes sean muy similares, sin embargo si que han tenido casi todas algo en común y es que en ellas aparecen estos dos personajes con la cabeza girada hacia nuestra izquierda. De nuevo vemos una imagen intrusa, que vuelve a ser la penúltima, en esta seguramente habrá encontrado como palabras en común el sofá marrón y alguna otra palabra. De nuevo pienso que el fallo puede deberse a las pocas palabras de las que dispone nuestro diccionario.