Universidad de Granada

Ingeniería Informática

Computación y Sistemas Inteligentes

Practica 3

Autor: José Antonio Ruiz Millán
Asignatura: Visión por Computador
20 de diciembre de 2018

1. **Emparejamiento de descriptores [4 puntos]**

   - **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.**

     Para este proceso se ha hecho uso de la función *extractRegion()* y *clickAndDraw()* dada por el profesor, obteniendo los vértices del polígono seleccionado dentro de la imagen. El proceso que realiza esta función se puede comprobar directamente en el fichero adjunto a la práctica.

     En mi caso, para los siguientes pasos, he seleccionado las parejas de imágenes [95,116], [130,128], [234,248].

   - **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).**

     El primer paso para poder realizar esto, es la creación de la máscara. Para ello, he definido la siguiente función:

```
def crearMascara(img,vertices):
    #Creamos una matriz vacia
    m = np.zeros(img.shape,dtype=np.uint8)

    #Creamos un array con los puntos seleccionados de la imagen
    approCurve = cv2.approxPolyDP(np.array(vertices,dtype=np.int64),1.0,True)

    #Ahora pasamos estos puntos dentro de la matriz creada al inicio.
    cv2.fillConvexPoly(m, approCurve,color=(255,255,255))

    #Pasamos el color de la mascara a grises
    m = cv2.cvtColor(m,cv2.COLOR_BGR2GRAY)

    #Devolvemos la mascara
    return m
```

     Como podemos ver, se ha hecho uso de las funciones *approxPolyDP()* y *fillConvexPoly()* que nos permiten en el primer caso, aproximar un polígono dados unos vertices, y en el segundo caso, dado un polígono, insertarlo en este caso en una matriz dandole valores a cada uno de las celdas a las que pertenece el polígono.

     Por otra parte, una vez explicado esto, el funcionamiento de este método es simple, creamos la matriz vacía con todos los valores como 0, aproximamos el polígono con los vertices obtenidos con las funciones comentadas anteriormente y insertamos ese polígono en la matriz poniendo su interior a los valores máximos (255). Por último se pasa la máscara a gris y se devuelve.

     Como **segundo paso** tenemos que calcular los puntos SIFT en esta seccion de la imagen usando la máscara y calcular las correspondencias con la segunda imagen, por lo que he definido la siguiente funcion:

```
def calcularCorrespondencias(img1,img2,masc):
    # Calculamos los descriptores.
    kp1, desc1 = calcularDescriptores(img1, masc)
    kp2,desc2 = calcularDescriptores(img2)

    # Calculamos las correpondencias.
    corr  = BFL2NN(desc1,desc2)

    # Dibujamos las correspondencias.
    matches = cv2.drawMatchesKnn(img1=img1,keypoints1=kp1,img2=img2,keypoints2=kp2,
     matches1to2=corr,outImg=None,flags=2)

    return matches

```

Donde *calcularDescriptores()* se define como:

```
def calcularDescriptores(img,masc=None):
    #Pasamos la imagen a gris
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    #Calculamos los descriptores usando SIFT
    sift  = cv2.xfeatures2d.SIFT_create()
    kp,desc = sift.detectAndCompute(gray,masc)

    #Devolvemos tanto los keypoints como los descriptores
    return [kp,desc]

```

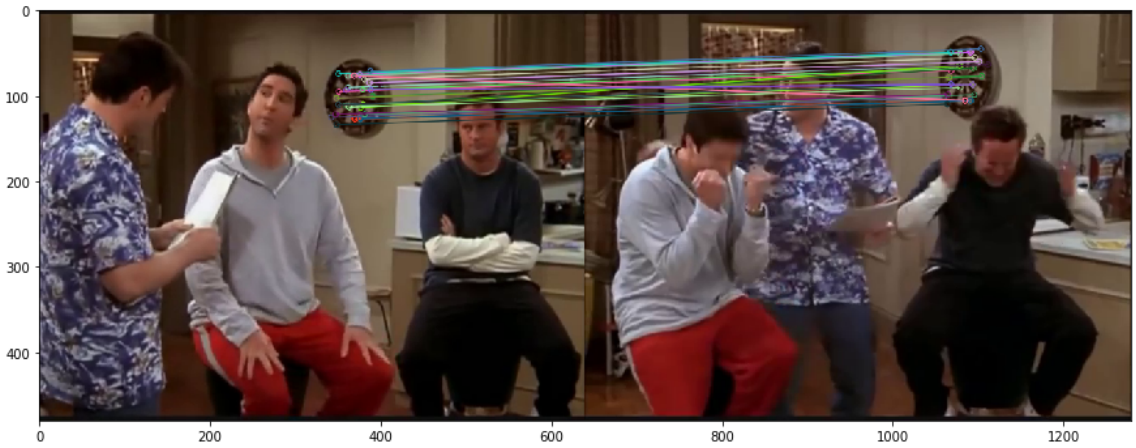
Ésta última funcion es de la practica anterior por lo que simplemete comentar que lo que hace es calcular los keypoints SIFT y los descriptores para devolverlos.

Por otro lado, la función *calcularCorrespondencias()*, como podemos ver, una vez tiene los keypoints y lo descriptores ded las dos imágenes (la primera calculada con la máscara creada anteriormente), crea las correspondencias utilizando la función *BFL2NN()* de la practica anterior. Después, creamos la imagen para poder dibujar los matches y finalmente en el main se imprime esa imagen.

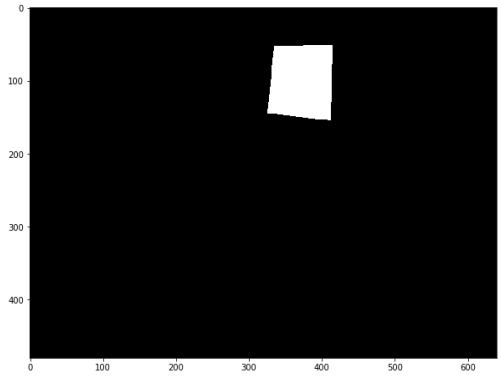
- **Pinte las correspondencias encontrados sobre las imágenes.**

  Como he comentado al principio he escogido 3 parejas de imágenes para este ejercicio, obteniendo:

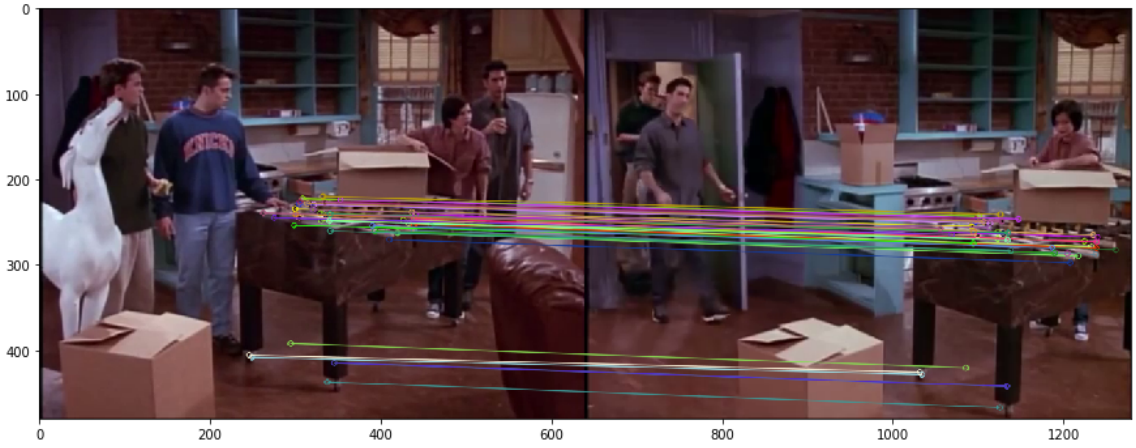
Pareja [95,116]



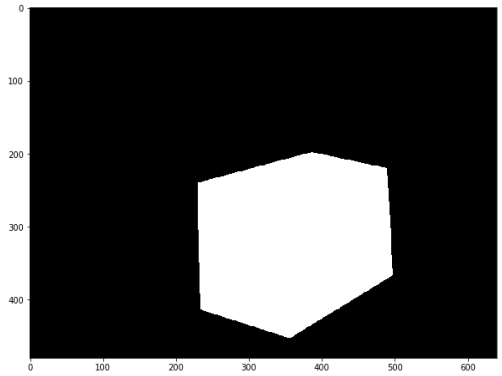
Donde la máscara seleccionada ha sido:



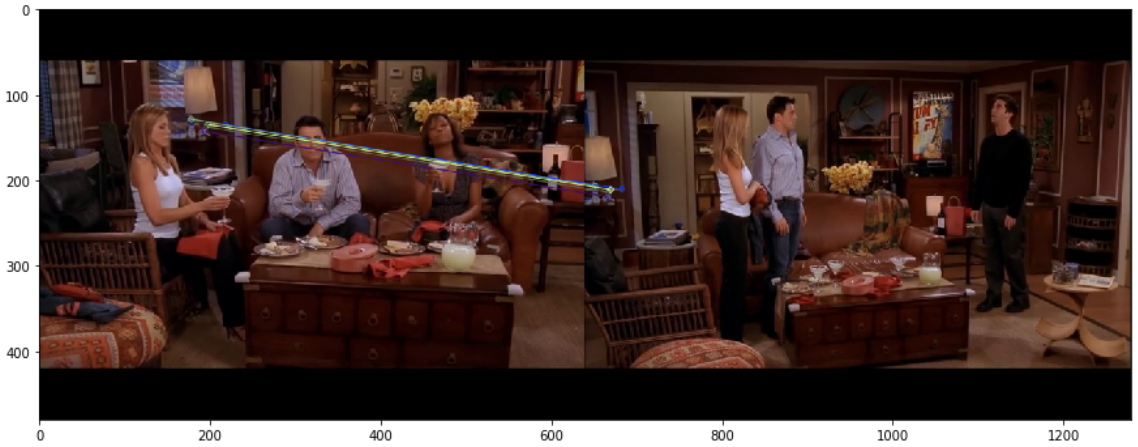
Pareja [130,128]



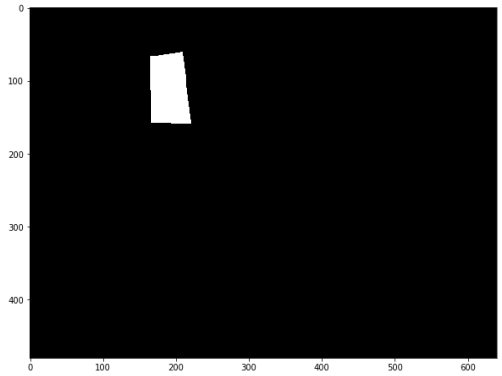
Donde la máscara seleccionada ha sido:



Pareja [234,248]



Donde la máscara seleccionada ha sido:



- **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.**

  Después de realizar varias pruebas como se puede ver en el apartado anterior más otras más hechas por entender los resultados, he podido comprobar que este proceso funciona correctamente cuando tenemos imágenes que mantienen el mismo ángulo de la cámara, es decir, cuando la imagen se mueve horizantalmente o verticalmente como es el caso del primer y segundo ejemplo, se obtienen unos resultudos con bastantes matches entre ellos. Sin embargo como podemos apreciar en la tercera imagen tenemos un minúsculo

cambio de ángulo y obtenemos muy pocos matches buenos. Cuando el cambio de ángulo es aún más significativo, no conseguía sacar ni un match que se considere "bueno".

Como conclusión diría que este proceso funciona correctamente en escenas que mantienen el ángulo igual o que se ha modificado minísculamente ya que en esos casos funciona bastante bien pero cuando el ańgulo cambia bastante, no es capaz de obtener unos buenos resultados.

2. **Recuperación de imágenes [4 puntos]**

   - **Implementar un modelo de índice invertido + bolsa de palabras para las imágenes dadas en imagenesIR.rar usando el vocabulario dado en kmeans-centers2000.pkl.**

     Para crear el modelo de índice invertido + bolsa de palabras he creado el siguiente método:

```
def crearIndicesInvertidos(diccionario):
    #Cargamos los nombre de todas las imagenes
    nombres = [img for img in listdir('./') if isfile(img) and '.pkl' not in img]
    #Creamos un diccionario que contendra el modelo indice invertido
    indicesInvertidos = dict([index,[]] for index in range(diccionario.shape[0]))
    #Creamos la bolsa de palabras conjunta
    bolsa = dict()

    #Para cada imagen
    for fich in nombres:
        print("calculando votos para:", fich)
        #Leelemos la imagen
        img = leeimagen(fich,1)
        #Calculamos los votos de esta imagen con el diccionario de entrada
        votos, bolsaImg = calcularVotos(img,diccionario)

        #Alamacenamos en la bolsa de palabras final el resultado obtenido
        #para la imagen en la que estamos.
        bolsa[fich] = bolsaImg

        #Recorremos los votos y anadimos para cada uno de los indices,
        #la propia imagen.
        for index,vot in votos.items():
            indicesInvertidos[index].append(fich)

    #Devolvemos el modelo de indice invertido mas la bolsa de palabras
    return [indicesInvertidos, bolsa]

```

     Para explicar el funcionamiento de este método lo haré paso por paso para que sea más fácil de entender.

     - En primer paso cargo todas los nombres de las imágenes que tenemos en el directorio.
     - Inicializo las variables que voy a necesitar que en este caso serán el propio modelo y la bolsa de palabras.

- Recorro todas las imagenes y para cada una de ellas, la cargo, calculo los votos (posteriormente se verá el funcionamiento de éste) y guardo el resultado tanto en la bolsa de palabras global como en los indices invertidos guardando el nombre de la imagen en las palabras que ha votado.
- Una vez hecho esto con todas las imágenes, ya tendríamos el modelo creado, por lo que devuelvo el modelo y la bolsa de palabras conjunta de todas las imágenes.

Para el corrector funcionamiento de este método, se tiene que implemetar la función *calcularVotos()* tal que:

```
1  def calcularVotos(img,diccionario):
       #Calculamos os decriptores de la imagen.
3      kp, desc = calcularDescriptores(img)

5      # Normalizamos los descriptores de la imagen.
       desc = cv2.normalize(src=desc,dst=None,norm_type=cv2.NORM_L2)
7      dicc = cv2.normalize(src=diccionario,dst=None,norm_type=cv2.NORM_L2)

9      # calculamos de semejanza entre las palabras y los  descriptores.
       votos = np.dot(dicc,desc.T)
11


13     # Calculamos las palabras que han sido ms votadas para la imagen.
       palabrasImg = np.zeros(shape=votos.shape[0], dtype=np.int)
15     for col in range(votos.shape[1]):
           # Obtenemos la matriz.
17         desc_column = votos[:, col]
           # Obtenemos el indice del mejor valor de semejanza
19         min_index = np.argmax(desc_column)
           # Aumentamos en uno el numero de votos.
21         palabrasImg[min_index] += 1


23
       # Guardamos palabras y numero de votos.
25     palabras = [[index, palabrasImg[index]] for index in range(len(palabrasImg)) if palabrasImg[
       index] > 0]
       palabras = dict(palabras)
27


29     return [palabras,palabrasImg]
```

Este método permite dada una imagen y el diccionario (vocabulario), obtener para cada una de las palabras que tenemos, los votos que ésta imagen hace. Para ello:

- Calculo los descriptores de la imagen.
- Normalizo los descriptores como nos aconsejan al comienzo del guión de la práctica.
- Calculo la semejanza entre el descriptor de la imagen con el diccionario haciendo el producto matricial entre ellos.
- Ahora recorro el resultado, comprobando la fuerza con la que vota a cada una de las palabras y se le suma 1 a la palabra con la que más fuerza halla votado.

- Creo un diccionario con el resultado que contendrá para cada una de las palabras cuantos votos ha realizado.
- Lo devuelvo.

Con esto, ya tenemos el modelo de índice invertido + bolsa de palabras creado para poder seguir con los siguientes apartados.

- **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.**

  Para este apartado lo que he hecho ha sido escoger un conjunto de 5 imágenes, 4 de ellas relacionadas (misma escena) y otra no relacionada. Obtengo la similitud entre ellas y muestro el resultado, comprobando que cuando las escenas son parecidas obtenemos un valor cercano a 1 (255) y cuando la escena no tiene nada que ver obtenemos valores cercanos a 0. Las imágenes seleccionadas para este experimento son [106,107,108,109,25].

  Para ello, he creado el siguiente método:

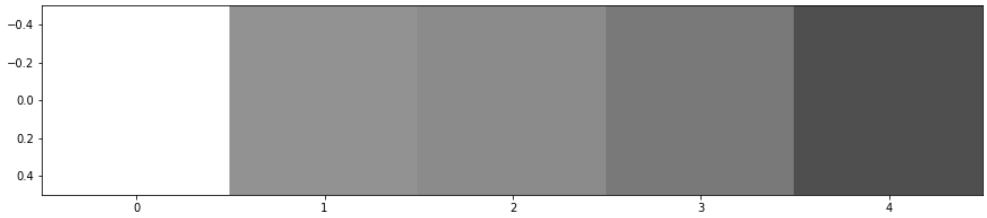
```
def mismaEscena(imgs,m_invertido,diccionario,bolsaImagenes):
    bolsa = dict([fich ,bolsaImagenes[fich]] for fich in imgs )

    bolsaImg1 = bolsa.get(imgs[0])

    sim = [ (( np.dot(bolsaImg1,bimg.T) ) / (norm(bolsaImg1)*norm(bimg)))*255 for name,bimg in
     bolsa.items()]

    sim = np.array(sim,dtype=np.uint8).reshape((1,5))

    return sim

```

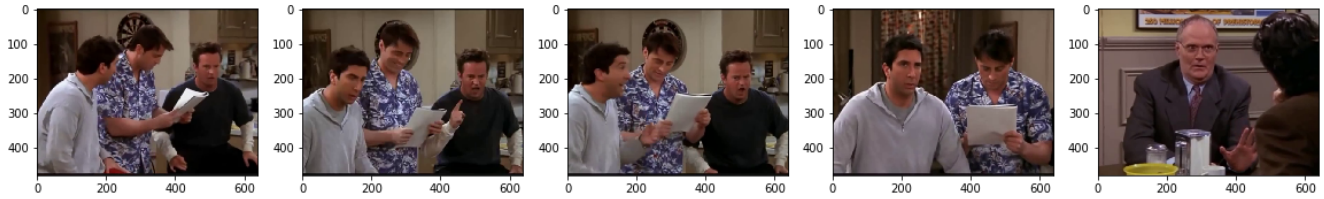
  Como vemos, esta función lo que hace es en primer lugar alamacenar las bolsas de palabras de cada una de las imágenes, después nos quedamos con la bolsa de palabras de la imagen que vamos a utilizar como base que en mi caso es la primera que mandamos y despues calculamos la distancia entre ésta imagen con el resto de imágenes. La similitud entre ellas se calcula como:

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

  En mi caso, multiplico el resultado por 255 porque la simetría se mide entre 0 y 1 y a mi me intersa entre 0 y 255 para poder pintarla.

  Finalmente, sabiendo que tenemos 5 imágenes de las cuales las 4 primeras pertenecen a la misma escena y la última no, obtengo lo siguiente:





Podemos apreciar que claramente el primer valor es blanco completamente ya que es la propia imagen, pero las otras 3 imágenes tenemos un valor grisaceo lo cual nos dice que la escena es parecida a ésta. Sin embargo, la útima es muy oscura, lo que nos dice que esta imagen pertenece a otra escena.

- **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.**

  Para este proceso he decidido utilizar las imágenes [425,104,62]. Siendo las dos primeras para comprobar el buen funcionamiento de este proceso y la última para comprobar que éste método puede fallar.

  Para poder calcular las imágenes similares a una imagen dada, he creado el siguiente método:

```
def calcularImagenesSimilares(img,indicesInvertidos,diccionario, bolsa,nmax=5):
    # Calculamos las imagenes que tienen los mismas palabras que la imagen.
    imgs = imagenesMismasPalabras(diccionario,img,indicesInvertidos)

    # calculamos la bolsa de palabras de las imagenes que tienen
    # palabras parecidas con las palabras de la imagen.
    bolsaImgs = dict([fich,bolsa[fich]] for fich in imgs)
    # Calculamos la bolsa de palabras de nuestra imagen.
    vot, bolsaImg = calcularVotos(img,diccionario)


    # Para cada imagen de bolsaImgs calculamos la semejanza con nuestra imagen.

    similar = [ [fich, ( np.dot(bolsaImg,bImg.T) ) / (norm(bolsaImg)*norm(bImg))]
                                for fich,bImg in bolsaImgs.items()]

    similar = sorted(similar, key=lambda par: par[1], reverse=True)[:nmax+1]

    similar = dict(similar)

    return similar

```

  Esta función es la encargada de dada una imagen, devolver las 5 imágenes con mayor similitud a la imagen que le pasamos. Para ello:

  - Lo primero que hago es obtener las imágenes que contienen alguna de las palabras que tiene la imagen que queremos comparar (el método se ve después).

- Despues obtengo la bolsa de palabras de cada una de estas imágenes para poder calcular la similitud.
- Se recorren todas las imágenes obtenidas anteriormente y se calcula la similitud con ésta utilizando la fórmula comentada anteriormente.
- Se ordena el resultado y se cojen las 5 imágenes (en este caso) más similares a la nuestra.
- Creo un diccionario con esta información y lo devuelvo.

Para el correcto funcionamiento, necesitamos comentar cómo está implementada la función *imagenesMismasPalabras()*:

```
1  def imagenesMismasPalabras(diccionario,img, indicesInvertidos):
       # Calculamos los votos de la imgenes.
3      votos, bolsaImg = calcularVotos(img,diccionario)

5      # Obtenemos las imagenes que tambien tienen esas palabras.
       imagenes = set()
7      for palabra, vot in votos.items():
           for nombre in indicesInvertidos[palabra]:
9              imagenes.add(nombre)

11     return imagenes
```

Simplemente lo que realizo en este método es en primer lugar calcular los votos de la imagen, para seguidamente comprobar para cada una de las palabras que ha votado esta imágen, las imágenes que tienen también esa palabra. Añado éstas imágenes y devuelvo el resultado.

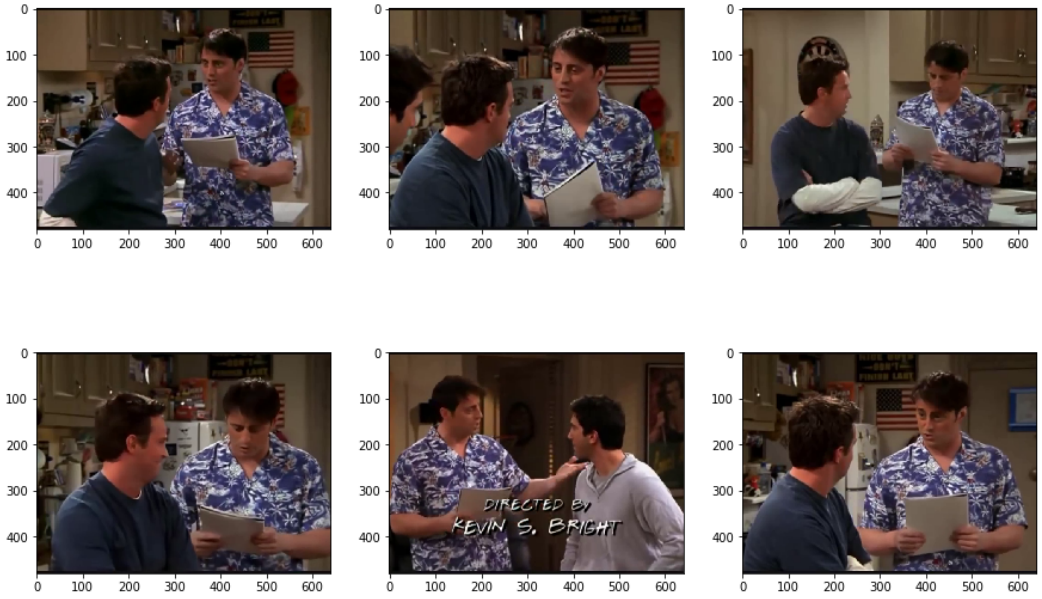
- **Explicar qué conclusiones obtiene de este experimento.**

  En los resultados que vamos a ver a continuación, **la primera imágen es la imágen seleccionada y el resto serían las imágenes más cercanas a ella**.

  Como he comentado al principio las imágenes-preguntas escogidas son:

Imágen 425:

Imágen 104:



Imágen 62:

Como conclusión, vemos como en los dos primero casos obtenemos un resultado muy satisfactorio, donde teniendo como imagen de partida la primera imagen de las 6 que se muestran, las otra 5 son muy parecidas, representado la misma escena.

Sin embargo, la tercera imágen no consigue ninguna imagen parecida. Esto se debe a que éste método no tiene ninguna forma de rechazar, es decir, si yo quiero obtener las 5 imágenes más semejantes a otra, las vamos a obtener aunque no existan. En esta base de datos de imágenes no tenemos imágenes del estilo de la imágen base y no sabe calcular entonces imágenes similares a ésta. Cuando se calcula la similitud entre la imagen base y el resto, vamos a obtener siempre valores y el algoritmo siempre cojerá la que más similitud tenga, que no quiere decir que realmente la tenga, si no que de todas las que tenemos, es la que más se parece.

A la vista de un ojo humano vemos claramente que no son parecidas ninguna de ellas, pero el algoritmo no sabe diferenciar ésto ya que únicamente se fija en el grado de similitud calculado con una fórmula y ese valor siempre es calculado.

3. **Visualización del vocabulario [3 puntos]**

   - **Usando las imágenes dadas en imagenesIR.rar se han extraido 600 regiones de cada imagen de forma directa y se han re-escalado en parches de 24x24 píxeles. A partir de ellas se ha construido un vocabulario de 2.000 palabras usando k-means. Los ficheros con los datos son descriptorsAndpatches2000.pkl (descriptores de las regiones y los parches extraídos) y kmeanscenters2000.pkl (vocabulario extraído).**
   - **Elegir al menos dos palabras visuales diferentes y visualizar las regiones imagen de los 10 parches más cercanos de cada palabra visual, de forma que se muestre el contenido visual que codifican (mejor en niveles de gris).**

     He decidido seleccionar 3 palabras visuales [4,18,21] para este apartado.

Para ete apartado he tenido que definir la siguiente función que será la encargada de calcular dado un parche, los parches más "parecidos" a éste:

```
def calcularMejoresParches(i_palabra,vocabulario,descriptores,etiquetas,parches):
    # calculamos los parches de la palabra.
    descriptores_sub = descriptoresPalabra(i_palabra, descriptores, etiquetas)
    mejores_desc = descriptoresCercanos(descriptores_sub, vocabulario[i_palabra])
    mejores = dict([[ID,parches[ID]] for ID in mejores_desc.keys()])

    # devolvemos los parches
    return mejores

```

En esta función lo que hago es en primer lugar calcular los descriptores relacionados con la palabra que nosotros queremos comprobar. En segundo lugar, de éstos descriptores obtenidos, se seleccionan los 10 mejores, y por último obtengo los parches asociados a los mejores descriptores obtenidos.

Para profundizar en este método necesitamos definir los métodos que utiliza, por lo tanto:

```
def descriptoresPalabra(i_palabra,descriptores,etiquetas):
    # Obtenemos los indices de los descriptores.
    i_desc = [index for index in range(len(etiquetas)) if etiquetas[index] == i_palabra]

    # Guardamos el subconjunto en un diccionario.
    descriptoresPalabra = dict([index, descriptores[index]] for index in i_desc)

    # Devolvemos el subconjunto.
    return descriptoresPalabra

```

Este es el encargado de dado un indice de una palabra concreta, obtenemos los descriptores relacionados con esta palabra. Para ello, en primer luegar obtenemos los indices de los descriptores relacionados con la palabra y seguidamente creamos un diccionario con el indice del descriptor y la información del mismo para devolverlo.

Por último:

```
def descriptoresCercanos(descriptores,palabra):

    # Calculamos las distancias entre la palabra y el descriptor.
    distancias = [[index,cv2.norm(src1=palabra,src2=desc, normType=cv2.NORM_L2)] for index,
    desc in descriptores.items()]

    # Ordenamos por menor distancia y nos quedamos con los 10 primeros.
    distancias = sorted(distancias, key=lambda d: d[1])[:10]

    distancias = dict(distancias)

    # Devolvemos los 10 descriptores mas cercanos.
    return distancias

```

Este es el encargado de obtener los 10 mejores descriptores y devolverlos. Para ello calculamos la distancia entre la palabra que queremos obtener y el cada uno de los descriptores. Despúes ordenamos el resultado y cojemos los 10 primeros. Se devuelve como diccionario y listo.

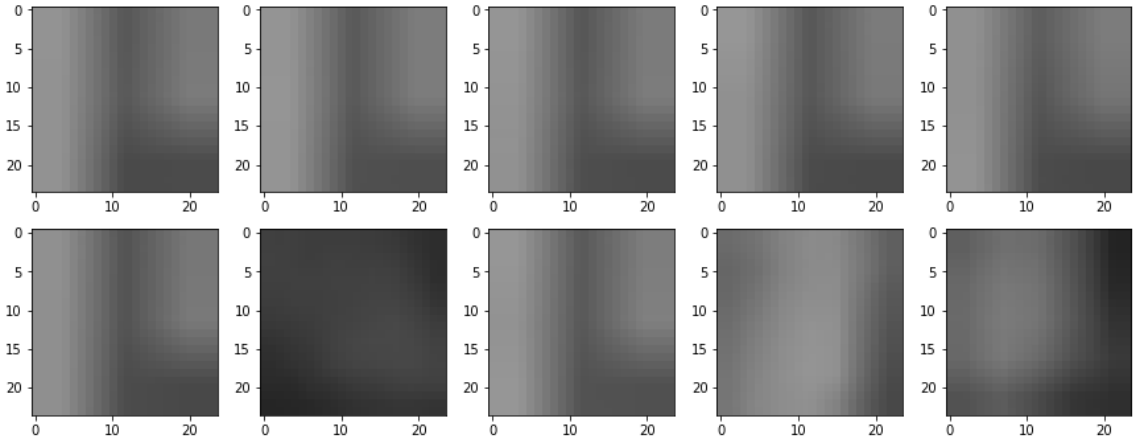
Finalmente, despues de todo el proceso, obtenemos los 10 parches más cercanos.

- **Explicar si lo que se ha obtenido es realmente lo esperado en términos de cercanía visual de los parches.**

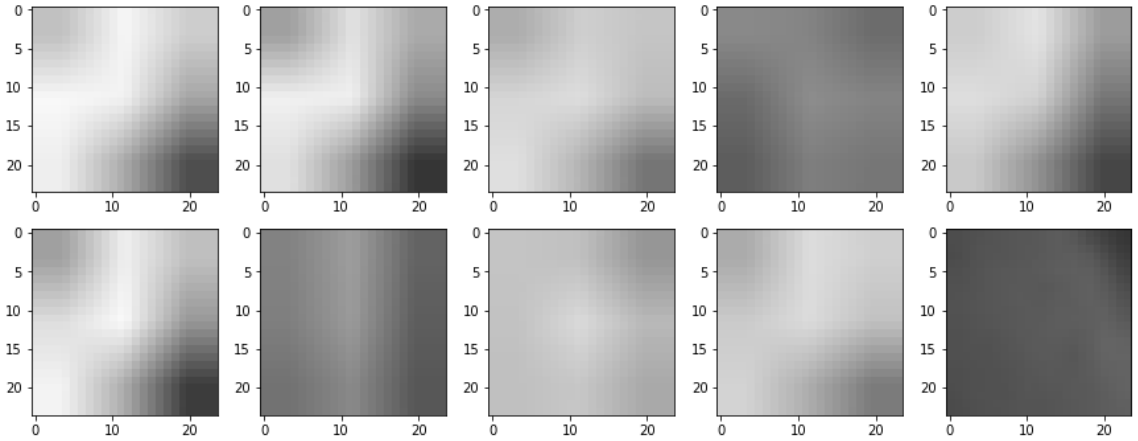
  Como en el caso anterior, **la primera imagen del resultado que vemos es el parche escogido**.

  Para cada uno de los casos se ha obtenido:

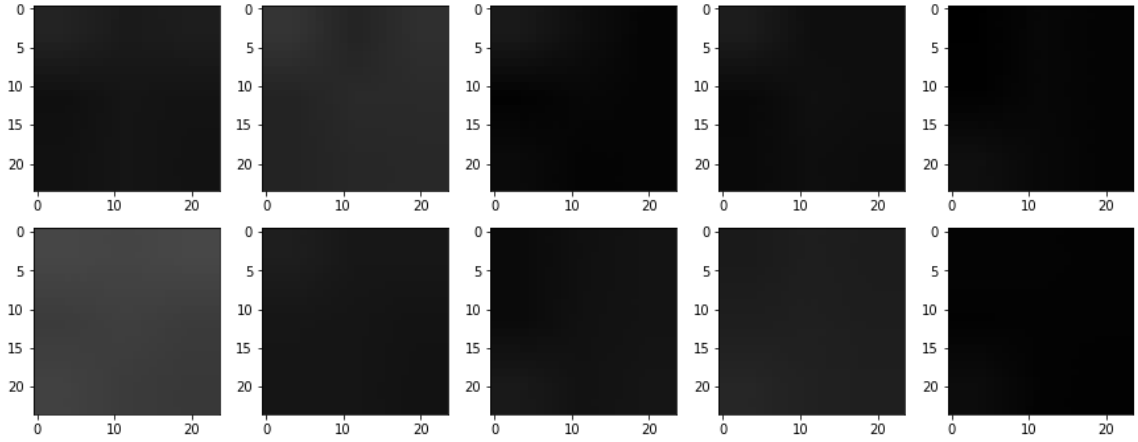
  Caso 4:



  Caso 18:



Caso 21:



Como conlusión, vemos como los parches en un procentaje bastante elevado son prácticamente idénticos aunque haya alguno que se diferencie un poco poco del parche principal (el primero que se muestra en cada caso) ya que en el primer caso podemos apreciar algo parecido a una esquina y los parches semenjantes son prácticamente idénticos a éste aunque alguno cambie en intensidad pero sigue mostrando lo mismo.

En el segundo caso vemos como tenemos una especie de línea blanca que cruza el parche y podemos comprobar también que el resto de parches cumple con esa característica.

En el tercer caso tenemos una imagen bastante fuerte con un tono negro y efectivameten vemos como el resto de imágenes tabmién cumplen con ese patron.

Por lo que podemos decir que los parches obtenidos respecto a un parche específico están altamente relacionados y es capaz de obtener parches similares al que comparamos.

Referencias

[1] OpenCV Reference, `https://docs.opencv.org/3.4.3/`, Accedido el 20 de diciembre de 2018.