
UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

FACULTAD DE INGENIERÍA

INGENIERÍA EN SISTEMAS INTELIGENTES

VISIÓN COMPUTACIONAL

SEMESTRE 2021-2022/II

RODRÍGUEZ GONZÁLEZ FERNANDO DE JESÚS

SEGUNDO EXAMEN PARCIAL

DR. PUENTE MONTEJANO CESAR AUGUSTO

08 DE ABRIL DE 2022

## PLANTEAMIENTO DEL PROBLEMA

Implementar un identificador de objetos utilizando descriptores presentes en OpenCV.

- Escoger al menos 3 fotos/imágenes/objetos favoritos que tengan en casa. De preferencia debe ser colorido y/o con textura.
- Utilizar la biblioteca OpenCV para implementar un programa que mediante una señal de video o que, sea capaz de identificar los objetos elegidos en el paso anterior, utilizando funciones de descriptores que ya contiene dicha biblioteca. Este programa deberá usar los siguientes descriptores:
  - ORB
  - SIFT
  - HOG
- Comparar el desempeño de los 3 descriptores al identificar el objeto.

## DESCRIPCION DE LA SOLUCIÓN

El primer descriptor con el que trabajamos es ORB. Lo que nos indica es que ORB es una fusión del detector de punto clave FAST y el descriptor BRIE con algunas modificaciones para mejorar el rendimiento. Usamos FAST para encontrar los puntos clave, luego se aplica la esquina de Harris para encontrar los mejores N puntos entre ellos.

Pasando a la implementación hecha para este trabajo, tenemos primero la declaración del descriptor mediante la función ORB_CREATE proporcionada por la librería de OpenCV, en esta añadimos un parámetro de 1000 para los nfeatures lo que nos ayudara a tener una mayor cantidad de características. Después de esta función, tenemos el código correspondiente para ayudarnos a hacer la lectura de las imágenes con las que obtendremos los clasificadores. Aquí nos ayudamos con la librería de OS para entrar a la carpeta donde tenemos guardadas las imágenes. Mediante un ciclo FOR recorremos la lista de imágenes obtenidas para ir haciendo el imread y con un parámetro de 0 pasamos la imagen a escala de grises. Añadimos la imagen al arreglo de imágenes y el nombre del archivo lo añadimos a un arreglo en donde almacenamos el nombre de los objetos.

```
orb = cv2.ORB_create(nfeatures=1000)
path = 'imagenes_src'
nombres_imagenes = os.listdir(path)
lista_imagenes = []
nombresObjetos = []
for imagen in nombres_imagenes:
    img = cv2.imread(f'{path}/{imagen}',0)
    lista_imagenes.append(img)
    nombresObjetos.append(os.path.splitext(imagen)[0])
desList = descriptores(lista_imagenes)
```

Como podemos ver en la ultima línea, tenemos la llamada a la función “descriptores” a la cual le pasamos el arreglo de imágenes, veamos qué es lo que hace esa función:

```
def descriptores(images):
    desList=[]
    for i in images:
        kp,des = orb.detectAndCompute(i,None)
        desList.append(des)
    return desList
```

Tenemos la declaración de un arreglo llamado desList en donde guardaremos los descriptores para cada una de las imágenes. Después con un ciclo for, recorremos la lista

de imágenes que recibimos como parámetros y dentro de ese ciclo, en la primera línea tenemos lo que es la llamada a la función “detectAndCompute” que sirve para la detección de keypoints y de procesar los descriptores. Como parámetro le mandamos la imagen a analizar y el parámetro “None” que indica que no estaremos usando una máscara como entrada. La función nos regresa dos cosas, los keypoints (almacenados en kp) y los descriptores (des), este último se almacenará en el arreglo de descriptores y esta lista se retornará al proceso principal.

```
def encuentraClase(img,desList,thres=15):
    kp2,des2 = orb.detectAndCompute(img,None)
    bf = cv2.BFMatcher()
    matchList= []
    res = -1
    for d in desList:
        matches = bf.knnMatch(d,des2,k=2)
        good = []
        for m,n in matches:
            if m.distance < 0.75*n.distance:
                good.append([m])
        matchList.append(len(good))
    if len(matchList)!=0:
        if(max(matchList)>thres):
            res = matchList.index(max(matchList))
    return res
```

Para esta función, lo primero que hacemos es, con la imagen recibida, la cual en esta función es la que vamos a clasificar, aplicamos la función de detectAndCompute para obtener los keypoints de esa imagen. Ahora, abrimos un Brute-Force Matcher con la función de BFMatcher. Y ahora con un ciclo for recorremos todos los elementos de la lista de descriptores y dentro de este, aplicamos la función de “knnMatch” la cual encuentra los k mejores coincidencias para cada descriptor, aquí, debemos mandarle como parámetro, el descriptor a comparar, los descriptores de la imagen a comparar y un k=2 lo que significa que el conteo de mejores coincidencias encontradas para cada descriptor sea 2.

Regresando al flujo principal, validamos que se haya obtenido una clase, si fue así, colocamos un texto en la copia creada de la imagen, este texto será el nombre del objeto detectado y así, mostramos la imagen. En caso contrario, el texto insertado será “N/A” lo que indica que no se detectó ningún objeto y con esto cerramos la ejecución de este descriptor.

Pasando al descriptor SIFT (Scale Invariant Feature Transform) el cual puede realizar la detección de características independientemente de las propiedades. Esto se logra mediante la transformación de los datos de la imagen en coordenadas de escala invariable. Realizar la implementación para SIFT fue realmente muy parecida al descriptor ORB. El proceso de abrir las imágenes, las funciones diseñadas y las comparaciones para hacer las validaciones. Los únicos cambios que tenemos son, en la declaración del SIFT en donde en vez de hacerlo con ORB, implementamos la función de xfeatures2d.SIFT_create y ahora, con el SIFT declarado es desde donde se hará la llamada a la función de detectAndCompute.

```
sift = cv2.xfeatures2d.SIFT_create(nfeatures=1000)
```

En lo respecta al descriptor HOG hubo mayores complicaciones en cuanto a su implementación. Si bien, parte del código es el mismo, lectura de imágenes de entrenamiento, lectura de imágenes a clasificar, ciclos for entre otras funcionalidades, la parte central del código cambian, por ejemplo, para declarar este descriptor tenemos lo siguiente:

- winSize – Tamaño de la ventana de detección.
- cellSize – Tamaño de celda
- blockSize – Tamaño de bloques en pixeles.
- blockStride – Paso de bloque. Múltiplo de cellSize
- nbins – Numero de bins utilizados.

```
winSize = (60,60)
blockSize = (10,10)
blockStride = (5,5)
cellSize = (10,10)
nbins = 9
##No modificar
derivAperture = 1
winSigma = -1.
histogramNormType = 0
L2HysThreshold = 0.2
gammaCorrection = 1
nlevels = 64
##
hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,
cellSize,nbins,derivAperture,winSigma,histogramNormType,
L2HysThreshold,gammaCorrection,nlevels)
```

Algunos parámetros se quedaron como default (nbins) con los demás valores se estuvieron cambiando para ver como cambiaban los resultados.

Los restantes parámetros aquí mostrados son tomados del paper HOG original.

Después, se genera el HOGDescriptor pasando como parámetros los antes declarados. Al momento de hacer el procesamiento se hace la llamada a la función “compute” a la cual le pasamos la imagen a procesar y parámetros de winStride, padding y locations.

```
des = hog.compute(i,winStride,padding,locations)
```

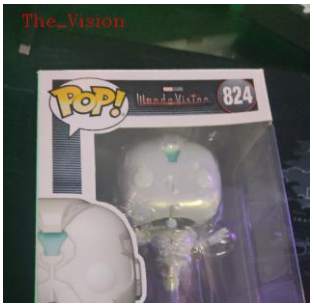
- winStride – Se refiere al paso de ventana y debe ser un multiplo de blockStride
- padding – Tamaño del padding
- locations – Vector de punto

Los valores seleccionados para estos son:

```
winStride = (8,8)
padding = (8,8)
locations = ((10,20),)
```

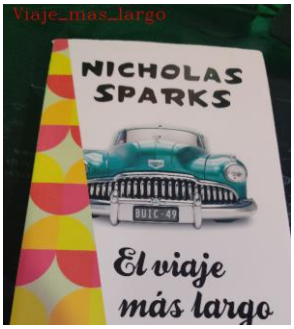
## DESCRIPCIÓN DE LOS RESULTADOS

Abajo podemos ver los resultados del descriptor ORB obteniendo un 100% de eficacia, detectando así todos los objetos correctamente y, además, el objeto que no estaba asociado a ningún descriptor no fue clasificado correctamente, pero se le asigno “N/A”. Con este descriptor decidí hacer una prueba, redimensionando las imágenes, las hice más pequeñas y ejecuté el programa, los resultados obtenidos fue que solo un objeto fue clasificado correctamente y a los demás se les asignó “N/A” (Captura en Anexos/ORB_Redimensiones).

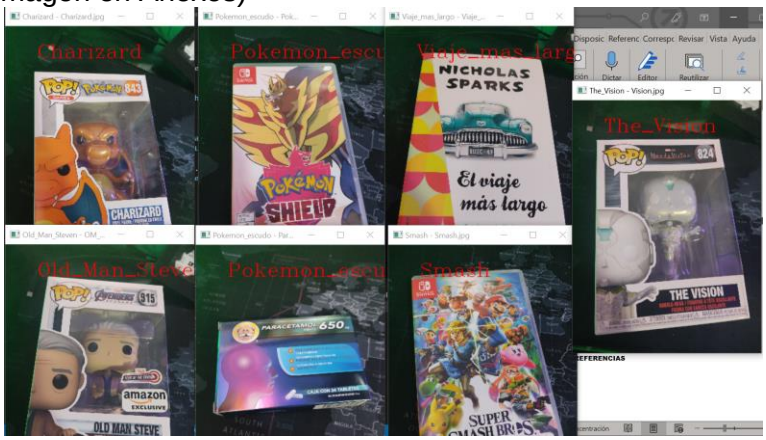




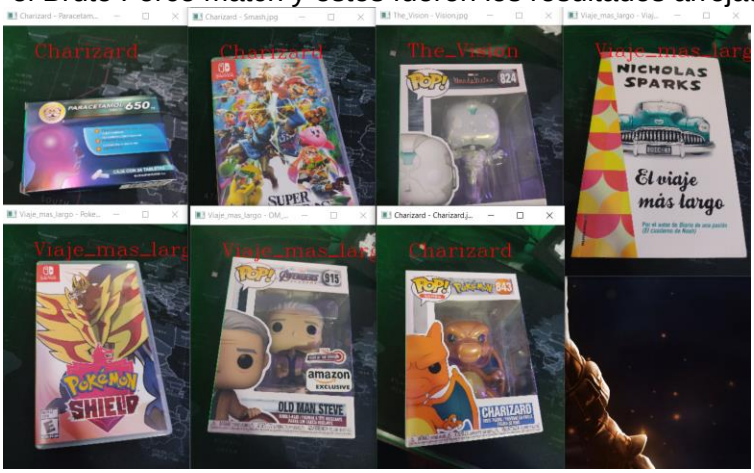





La siguiente imagen corresponde a los resultados del descriptor SIFT, en donde, se le aplico un redimensionamiento a las imágenes de 600x300 y como podemos ver, no hubo cambio, en los resultados, los objetos que tenían un descriptor fueron exitosamente clasificados, el objeto extra que se agrego no fue clasificado con éxito y se identifico como otro que no corresponde.  (Imagen en Anexos)




Con respecto a los resultados en HOG, podemos ver que no tuvo la precisión deseada, si bien, logro detectar a todos los objetos, solo 3 de ellos fueron correctos. Esto se puede deber a que no se implemento el SVM debido a que no lo pude entender muy bien dado esto, decidí usar el Brute Force match y estos fueron los resultados arrojados:



## DISCUSIÓN

Dentro de la comparación entre SIFT y ORB podríamos dar mejores comentarios de ORB pues en la prueba principal logro clasificar todos los objetos correctamente y el objeto que no tenía descriptor lo marco como NA lo cual es un casi 100% de efectividad. El SIFT al contrario, podemos decir que solo fallo en el que no tenía un descriptor relacionado y todos los demás los clasifico de manera correcta. La fortaleza o el punto a favor que podríamos dar en SIFT es que, como su nombre lo indica trabaja aun con variabilidad de la escala, pues, aunque se redimensionaron las imágenes el detector obtuvo los mismos resultados que con las imágenes originales caso contrario con ORB que al momento de redimensionar las imágenes sus resultados bajaron la efectividad y solo uno fue detectado correctamente. Comparando los resultados con el descriptor HOG, este es el que sale perdiendo más, pues solo logro detectar correctamente 3 objetos, no hubo un mejor resultado en el experimento creado. Esto no dudo que se pueda mejorar con la implementación y entrenamiento del SVM personalizado para los objetos elegidos.

## CONCLUSIÓN

Creo que la realización de este proyecto fue de mucho provecho, ya que me ayudo a entender de una manera más clara y objetiva el funcionamiento de los descriptores pues en la teoría muchas veces nos quedamos con muy poco, pero llevarlo a la practica ayuda más. Sabemos que OpenCV nos ayuda demasiado pues el que tenga ya las funciones implementadas para nosotros solo tener que saber como y cuando implementarlas que es otro proceso pero que ayuda a un mejor entendimiento. El descriptor HOG tuvo una mayor complicación al no encontrar de manera muy clara como es la implementación. Creo que el HOG se puede mejorar en un gran porcentaje si se usa y se entrena una SVM, cosa que no pude lograr en la realización de este proyecto. El video proporcionado fue de mucha utilidad para los dos primeros descriptores.

## REFERENCIAS


https://learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/
https://stackoverflow.com/questions/6090399/get-hog-image-features-from-opencv-python
https://www.youtube.com/watch?v=nnH55-zD38I
https://github.com/spmallick/learnopencv/blob/0a1f555fee10eba9644d840463ca9d99338e07cd/digits-classification/train_digits.py#L37
https://docs.opencv.org/4.x/d5/d33/structcv_1_1HOGDescriptor.html
https://stackoverflow.com/questions/28390614/opencv-hogdescripter-python
https://docs.opencv.org/2.4/modules/gpu/doc/object_detection.html