Introducción a OpenCV

Visión por Computador

Ángel Cabeza Martín
75571222F

1. Introducción

El objetivo principal de esta práctica es conocer la biblioteca OpenCV, el entorno de desarrollo Spyder, empezar a trabajar con imágenes y aprender a visualizarlas, para ello se proponen una serie de ejercicios interesantes para comenzar a entender cómo trabaja OpenCV con las imágenes internamente, estas cuestiones se irán abordando conforme vayan apareciendo a lo largo de la memoria. Es importante tener estos conceptos claros porque cuando empecemos a trabajar con cosas más complicadas, si tenemos la base clara, nos ahorrará mucho tiempo.

2. Ejercicio 1

Este ejercicio nos pide crear una función para poder leer una función y mostrarla tanto en grises como a color. Para ello he utilizado dos métodos, uno de OpenCV para leer la imágen y para visualizarla uno de la biblioteca Matplotlib:

1. imread: con este método se lee la imágen. Como argumentos le he pasado la ruta donde tengo almacenada la imágen y un flag que indicará si queremos visualizar la foto a color (flag = 1) o en escala de grises (flag = 0).
2. imshow: con este método se muestra la imágen. Como argumentos le he pasado la imágen que hemos leído con imread.

De esta forma para leer y mostrar una imágen debemos hacer uso de estos dos métodos, sin embargo si lo hacemos así y dibujamos una imágen a color obtenemos el resultado de la Figura 1 [1]. Como se aprecia en la figura, los colores de la imágen están invertidos, esto se debe a que OpenCV guarda las imágenes como un tensor (una matriz de 3 dimensiones), en formato BGR (Blue Green Red), en vez de en el formato estándar RGB (Red Green Blue). Para solucionar esto he optado por recorrer la imágen al revés de forma que el formato BGR lo leo como RGB y he obtenido los resultados apreciados en la Figura 2 [2] y en la Figura 3 [3].

Como vemos estos resultados sí son correctos porque una imágen a color, si indicamos que la lea en escala de grises (flag = 0) la muestra en escala de grises y si le indicamos que la lea a color (flag = 1) la muestra a color y con el color de la imágen original y no con los colores invertidos.

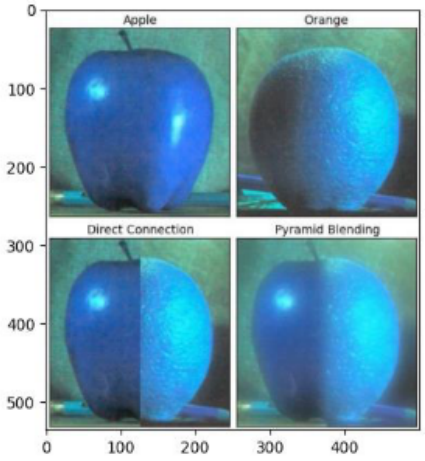

Figura 1: Imágen mostrada con los colores invertidos (es errónea).

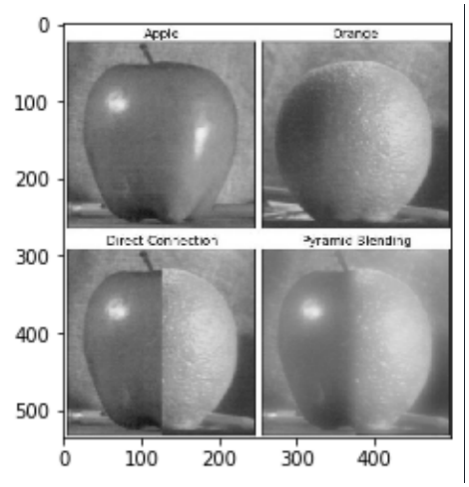

Figura 2: Ejemplo de visualización de una imágen en blanco y negro.

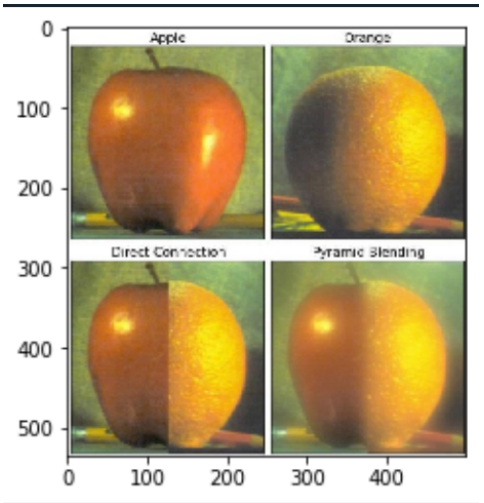

Figura 3: Ejemplo de visualización de una imagen a color.

3. Ejercicio 2.

En este ejercicio se nos pide crear una función que visualice una matriz de números reales tanto monobanda como tribanda.

Para este ejercicio he hecho dos funciones:

- **normalizamatriz**: recibe una imagen y devuelve otra imagen **tribanda** normalizada en el rango [0,1].
- **pintal:** recibe una imagen que normaliza usando la función anterior y la muestra.

Para la función **normalizamatriz**, lo primero que he hecho es convertir la imagen a tribanda si esta no lo es. Para ello triplico en profundidad la única banda existente con la ayuda del método dstack de numpy. Una vez hecho esto calculo el máximo y mínimo de cada canal con las funciones max y min de numpy y aplico la siguiente fórmula:

$$x_i = \frac{(x_i - min)}{(max - min)}$$

de esta forma normalizamos cada canal con su mínimo y máximo sin pérdidas de información.

La función **pintal** simplemente llama a la función normalizamatriz descrita anteriormente y la muestra.

De esta forma obtenemos los resultados de la Figura 4 [4] y la Figura 5 [5], como vemos la Figura 4 [4] es una matriz monobanda y por ello la imágen está en escala de grises, sin embargo la Figura 5 [5] es una matriz tribanda y por ello en la imágen se pueden observar puntos de distintos colores.

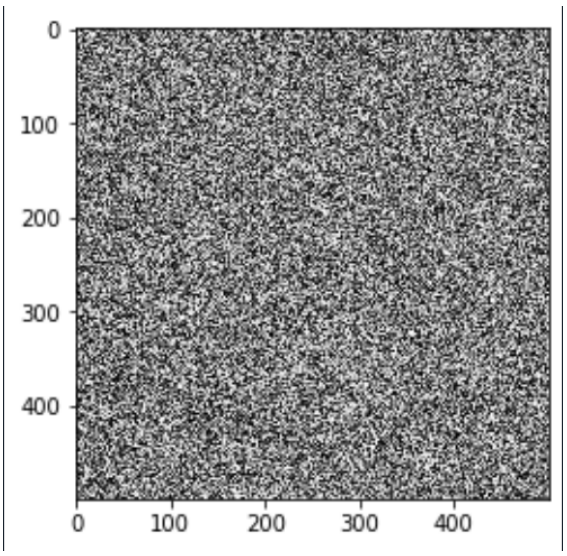

Figura 4: Matriz aleatoria monobanda normalizada en el intervalo [0,1]

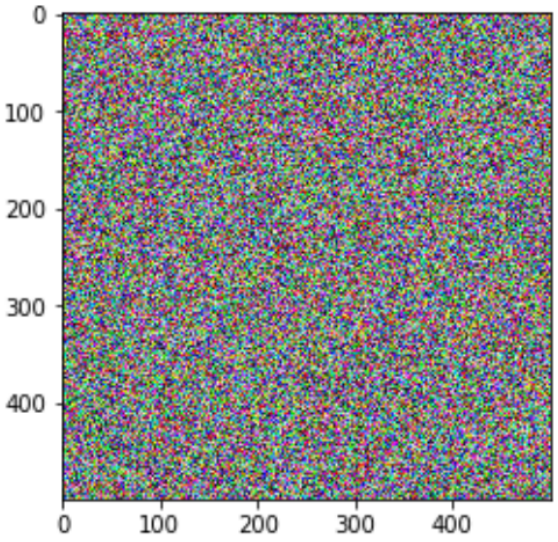

Figura 5: Matriz aleatoria tribanda normalizada en el intervalo [0,1]

4. Ejercicio 3.

En este ejercicio se nos pide crear una función que concatene varias imágenes y las muestre en una sola.

Para concatenar las imágenes he usado el método **hconcat** de OpenCV y me he encontrado con los siguientes problemas:

- Las imágenes tenían que tener el mismo tamaño.
- Las imágenes tenían que ser o todas monobanda o todas tribanda.
- Las imágenes pueden utilizar distintos niveles de gris o de colores.

Para solucionar el primer problema se puede optar por dos estrategias, hacer un resize de las imágenes para que tengan la dimensión de la máxima anchura y la máxima altura de todas las imágenes o hacer “zero-padding”, es decir, introducir una franja negra en la parte superior e inferior de la imágen para que todas tengan el mismo tamaño.

La ventaja del método “zero-padding” es que las imágenes no se deforman, cosa que sí ocurre con el otro método, sin embargo, la desventaja es que introducir la franja negra puede resultar un tanto molesta para el usuario y, sobre todo, si tienes una imágen muy grande y otra muy pequeña, la franja negra introducida puede llegar a ser más grande que la imágen. Dicho esto yo he decidido implementar las dos para poder ver físicamente las ventajas y desventajas que acabamos de comentar.

Para solucionar el segundo problema lo que he decidido es convertir todas las imágenes monobanda a tribanda. He hechor esto porque no veía otra forma lógica de poder pintar tanto imágenes a color como imágenes en escala de grises a la vez, ya que, si convertía las imágenes a color en monobanda estas se pintan en escala de grises y eso no nos sirve.

La desventaja de esto es que si la lista de imágenes que nos pasan como entrada solo contiene imágenes monobanda. vamos a gastar algo de tiempo de cómputo en convertirlas a tribanda. Y la ventaja que tiene es la comentada anteriormente, podemos dibujar las imágenes a color junto con las que están en escala de grises.

Por último, para solucionar el tercer problema la solución que he usado es utilizar la función del ejercicio anterior y escalar las imágenes al rango [0,1] de esta forma todas las imágenes usan los mismo niveles de gris y colores.

Para implementar la primera opción, he guardado la altura y anchura máximas de todas las imágenes y con el método resize de OpenCV las he escalado a esas dimensiones. Finalmente usando el método hconcat las he juntado todas y las he mostrado.

Para la implementación del método zero-padding, he guardado la altura máxima de todas las imágenes. Una vez guardada, recorremos cada imágen y miramos si la altura de la imágen es menor que la altura máxima, si es menor tenemos que crear las franjas negras y si no simplemente agregamos la imagen a las ya existentes con hstack de numpy.

Para crear las franjas he creado dos matrices de ceros con el tamaño siguiendo esta fórmula:

$$tam\ franja\ =\ (altura\ imagen\ -\ altura\ maxima)\ /\ 2$$

además a la franja superior le agrego el resto de la división por si la diferencia no es divisible entre 2. Después, las convertimos a tribanda y la agregamos a la imágen original, de esta forma la imagen original pasa a tener la altura máxima y ya la podemos concatenar con las imágenes anteriores.

Tras solucionar estos problemas e implementar ambas funciones obtenemos los resultados de la Figura 6 [6] aplicando el resize y de la Figura 7 [7] con zero-padding.

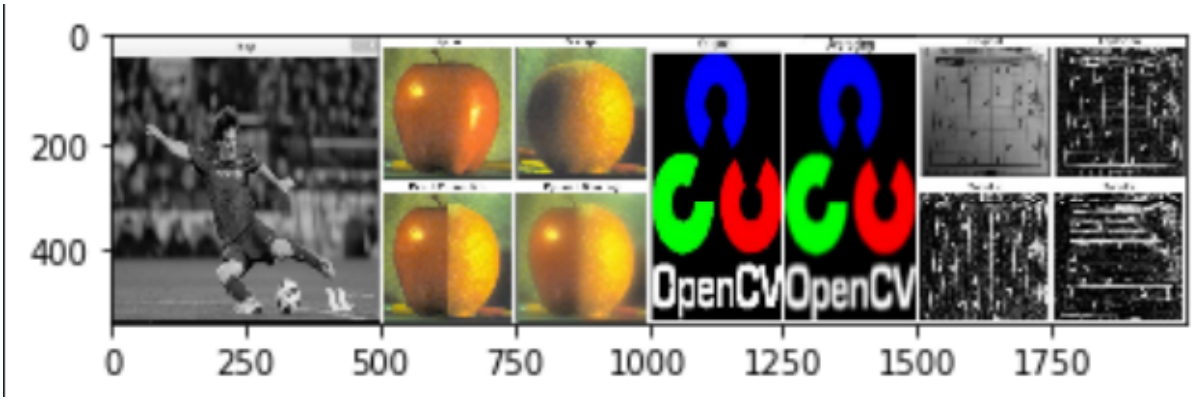

Figura 6: Varias imágenes concatenadas en una sola. Podemos apreciar como hay algunas deformadas.

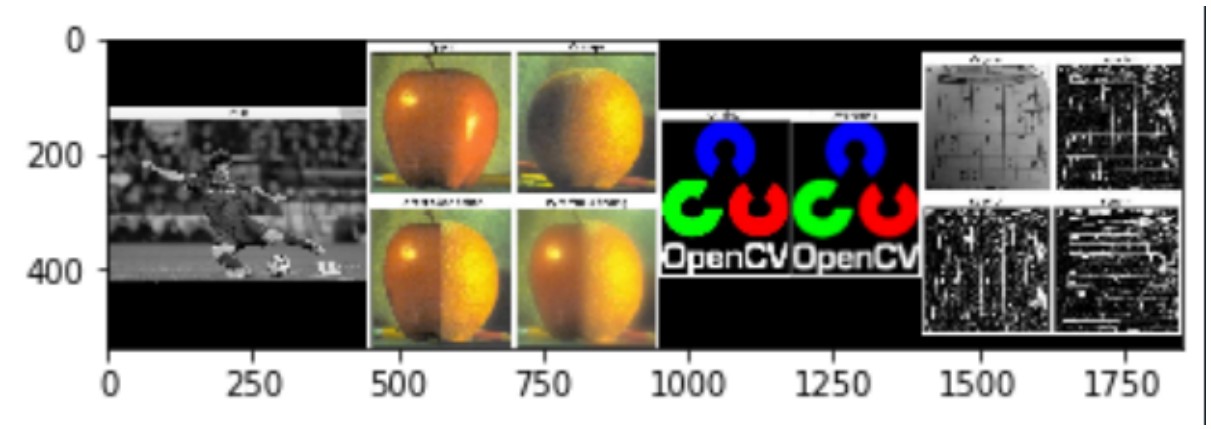

Figura 7: Varias imágenes concatenadas en una sola usando la técnica zero-padding. Podemos observar que han aparecido franjas negras pero las imágenes ya no están deformadas

5. Ejercicio 4

En este ejercicio se nos pide crear una función que permita modificar el color de una imágen dada unas coordenadas (en concreto las coordenadas de un cuadrado de dimensión 50x50 situado en el centro de la imágen).

Para realizar este ejercicio he creado una función **modificacolor** que recibe como parámetros la lista de coordenadas y una tupla de 3 elementos que corresponderán al nuevo color de la imágen **en formato BGR**. Hay que pasarlo en este formato porque, como ya sabemos, la representación interna de OpenCV de una imágen es en este formato.

La única consideración que hay que realizar, a parte de la ya mencionada del formato BGR, es tener en cuenta que en una imágen las filas y las columnas están invertidas, por tanto fila=y columna=x.

Teniendo esto en cuenta, hemos obtenido como resultado la Figura 8 [8]. Como podemos ver el cuadrado tiene dimensiones 50x50 (es fácil de observar mirando los ejes de la imágen), está en el centro y es de color azul.

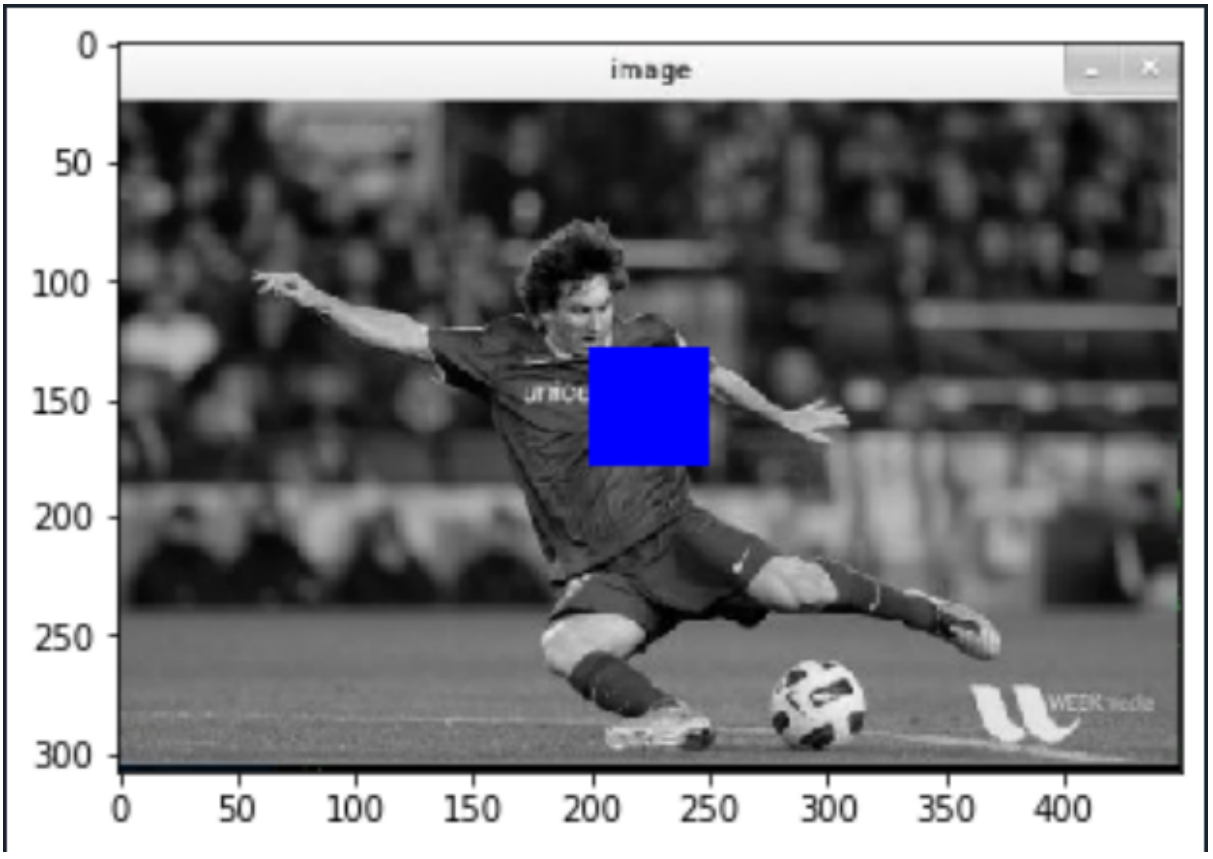

Figura 8: Imagen obtenida de la modificación del color de parte de la imagen..

6. Ejercicio 5

En este ejercicio se nos pide visualizar las imágenes dentro de la misma ventana junto con el título correspondiente.

Para realizar esto he decidido jugar con la biblioteca matplotlib. He creado una figura de dimensión 10x7 pulgadas (la dimensión la he decidido probando distintas y me he quedado con la que queda mejor), y sobre ella he hecho varios subplots (método add_subplot de matplotlib) con las distintas imágenes agregándoles su correspondiente título (método title de matplotlib), siempre teniendo en cuenta que hay que recorrer las imágenes al revés para que salgan en formato RGB. El resultado es el obtenido en la Figura 9 [9].

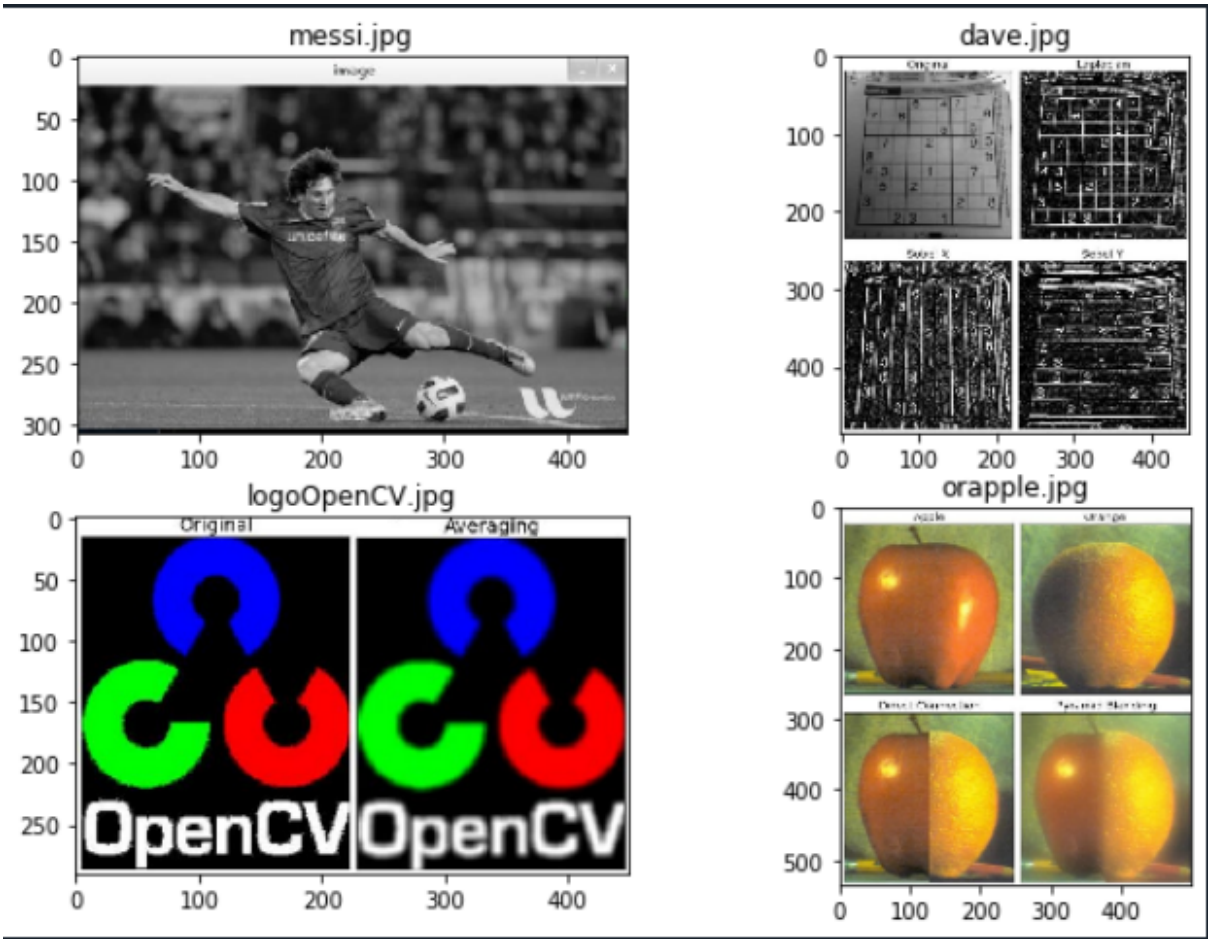

Figura 9: Ventana con varias imágenes y sus títulos

6. Bibliografía

Documentación oficial OpenCV flags de lectura y escritura de imágenes
Documentación oficial OpenCV función imread
Documentación min numpy
Documentación max numpy
OpenCV hconcat
Dstack numpy
Numpy hstack