UNIVERSIDAD DE GRANADA

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Práctica 0: Introducción a OpenCV

Visión por Computador

Autor:
Lugli, Valentino Glauco · YB0819879

Índice

1. Introducción

Esta práctica consistió en la realización de 5 ejercicios para familiarizarse con el entorno de desarrollo propuesto para realizar el resto de prácticas de Visión por Computador y comprendieron el manejo de las estructuras de datos y métodos relacionados con la manipulación, carga y dibujado de imágenes dentro del lenguaje de programación Python principalmente con la librería OpenCV y de manera secundaria con las librerías Matplotlib y Numpy.

2. Ejercicios

2.1. Escribir una función que lea el fichero de una imagen y permita mostrarla tanto en grises como en color

Para la realización de este ejercicio, se utilizó la función `imread()` de la biblioteca OpenCV puesto es lo más directo y posee el parámetro que interesa para este ejercicio: que permita que una imagen sea leída en escala de grises (monobanda) como en colores (tribanda).

Esta función se incluye dentro de la función pedida `leeImagen(ruta, flagColor)`, donde el parámetro `flagColor` es un booleano indicando la manera en que se leerá la imagen, `True` o 1 para que sea en color y escala de grises de lo contrario.

Se puede comprobar que la imagen se la leído correctamente, además de intentar dibujarla, por medio de la función `type()` de Python, si indica que es de la clase `numpy.ndarray` se puede estar seguro de una lectura correcta debido a que internamente OpenCV trabaja con las matrices de Numpy, adicionalmente se puede verificar que las dimensiones sean correctas con el atributo `shape` de esa matriz de Numpy, también se puede acceder a la matriz e imprimir rangos de valores.

Se imprimió la imagen `orapple.jpg` de dos maneras: leída como una imagen a color y a escala de grises, ver Figura 1.

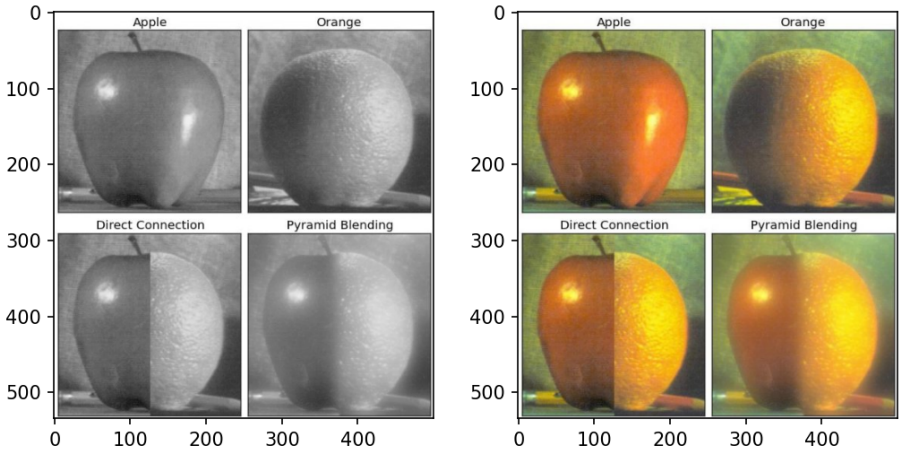

Figura 1: Resultado de lectura en escala de grises y color

2.2. Escribir una función que permita visualizar una matriz de números reales cualquiera/arbitraria, tanto monobanda como tribanda

Para realizar este ejercicio se utilizará la librería Matplotlib para mostrar las imágenes, puesto que Google Colab tiene mejor compatibilidad para pintar imágenes que con la función de OpenCV equivalente. Sin embargo, OpenCV y Numpy se utilizarán para el resto de operaciones de manipulación de las imágenes.

La función `pintaI(im, title=None)` acepta matrices de Numpy con números arbitrarios, ya sea a escala de grises (monobanda) o a color (tribanda): Para diferenciarlas, se utiliza la lontigud de la tupla `shape` de la matriz, puesto que las imágenes monobanda poseen solamente dos dimensiones en la tupla, mientras que las tribanda poseen tres, se asume que no se trabajará con matrices con otro número distinto de bandas.

- Si es una matriz monobanda, lo que daría como resultado una imagen de escala de grises, se pueden utilizar los parámetros que posee `imshow()` para normalizar la imagen directamente, primero se indica que el mapa de color debe ser gris con `cmap='gray'` y luego se indica que debe normalizarse con `Norm=clr.Normalize()`, que por defecto toma los valores máximos y mínimos de la matriz y linealmente normaliza los valores entre $[0, 1]$, lo cual asegura que no se está perdiendo información.
- Si es una imagen a color, entonces se realiza manualmente la normalización con la ecuación (1), donde p es el valor de una celda, p_{min} es el valor mínimo en la matriz y p_{max} el máximo.

$$p' = \frac{p - p_{min}}{p_{max} - p_{min}} \tag{1}$$

  El resultado p' será el valor escalado entre $[0, 1]$ sin pérdida de información. Este proceso se puede realizar en una línea puesto que Numpy se encarga de realizar esta operación para cada celda de la matriz. Una vez normalizado, para imprimir se invierte la última componente de la tupla utilizando el indexado de las matrices de Numpy. Esta inversión no es realmente necesaria puesto que esta función tiene el objetivo de imprimir una matriz arbitraria, no una imagen, pero se ha decidido seguir el convenio de OpenCV para evitar posibles confusiones futuras.

Adicionalmente y por estética, se puede incluir un título a la imagen, aunque este es un parámetro totalmente opcional y para dibujar una imagen en una ventana diferente, se hace uso de la función `figure()` de Matplotlib.

Una vez realizado esto, se crearon dos matrices con valores aleatorios y dos matrices con valores fijos para poder comprobar manualmente el funcionamiento.

```
np.array([[-100, -50, 0],
          [0, 50, 100]])
```

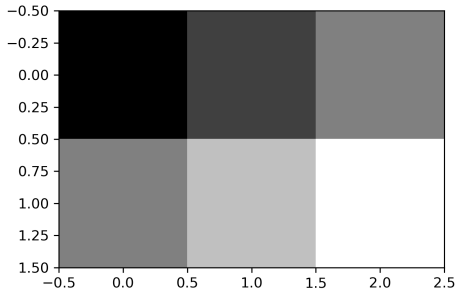

Figura 2

```
np.array([[[-20, -20, -20],
           [-20, -20, 20],
           [20,  -20, -20]],
           [[-20, 20, -20],
           [0,    0  , 0],
           [20,  20,  20]]])
```

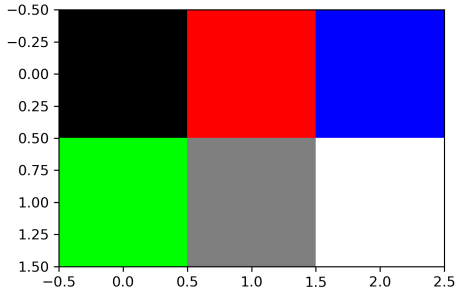

Figura 3

Como se observa en la Figura 2, que es una matriz monobanda, el color negro se obtiene con el valor mínimo de `-100` y a medida que los valores se acercan al máximo de `100`, se van aclarando las celdas hasta llegar al blanco.

En la Figura 3 se tiene el mismo ejemplo con una matriz tribanda, y de nuevo puede observarse que los colores corresponden con los valores mínimos y máximos, esto demuestra que la normalización se está realizando, escalando los valores entre $[0, 1]$ puesto que no se está perdiendo información.

2.3. Escribir una función que visualice varias imágenes distintas a la vez

Para realizar la función `pintaIM(vim, title=None)` se tomaron varias decisiones puesto que el planteamiento deja ciertos aspectos ambiguos:

- Las imágenes fueron concatenadas de manera horizontal, de manera que la primera imagen del vector será la que esté en el extremo izquierdo y la última la que está al extremo derecho.
- Las imágenes no se redimensionan, por lo tanto, aquellas imágenes que no tengan la altura máxima, son rellenadas con color blanco verticalmente hasta llegar a la altura máxima.
- La función acepta tanto imágenes de escala de grises como de colores, se creó una función adicional, `Gray2Color()` que convierte las imágenes de escala de grises a BGR, para poderlas concatenar sin problemas. Se utiliza la función `cvtColor()` que está diseñada para este propósito indicando la imagen fuente y la conversión, en este caso de escala de grises a BGR.

- Acepta un título opcional.

Con esta aclaración, el funcionamiento del subprograma es el siguiente:

En primer lugar, se determina cual es la altura máxima: aquella de la imagen o imágenes con mayor dimensión vertical.

El proceso comienza con la primera imagen de la lista, si esta está en escala de grises, se convierte a BGR, y luego se asigna a la variable `strip` la cual contendrá todas las imágenes concatenadas, luego:

- Si tiene altura máxima, se asigna directamente a la variable `strip`.
- Si no, entonces se utiliza la función `copyMakeBorder()` la cual como indica su nombre genera una copia de la imagen con un borde añadido. La imagen resultante posee 0 bordes excepto por debajo, puesto que se quiere que las imágenes estén alineadas por el borde superior, por lo tanto el relleno es la diferencia de la altura máxima y la altura de la imagen. Se utiliza `cv.BORDER_CONSTANT` para indicar que se desea un color plano de borde, y la tupla al final indica que se rellenará con blanco.

Una vez finalizado esto, se comienza el bucle en el siguiente ítem del vector, donde se realizan las mismas operaciones descritas anteriormente con la adición de `hconcat()`, que es para concatenar horizontalmente `strip` con la siguiente imagen `i` del vector.

Una vez finalizado, se dibuja la imagen. Puede observarse un ejemplo de esta función en la Figura 4, donde se puede observar que la primera imagen es la que posee mayor dimensión vertical del grupo, y el resto de imágenes están concatenadas alineadas por el borde superior, también notar que hay una mezcla de imágenes en escala de grises y a color.

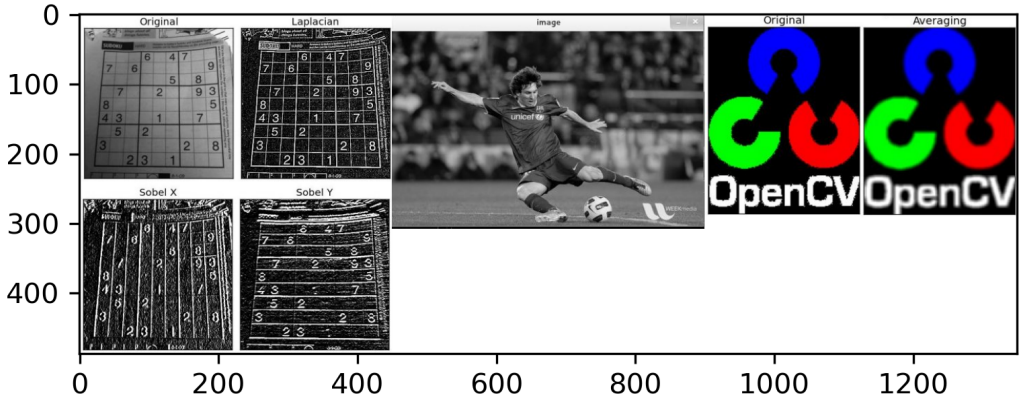

Figura 4: Lista de imágenes concatenadas

2.4. Escribir una función que modifique el color en la imagen de cada uno de los elementos de una lista de coordenadas de píxeles

Para la realización de este ejercicio se creó la función `cambiarColor(imagen, listaPuntos, color)` con los parámetros que se pensaron son ideales para realizar lo pedido.

Puesto que se indica que la función reciba una lista de coordenadas, lo que realiza es tomar cada coordenada e ir reemplazando el color que posee con el color que deseado, comprobando que esta coordenada se encuentra dentro de los límites de la imagen.

Se tomó la decisión de aceptar imágenes en escala de grises, las cuales son convertidas a BGR como en el ejercicio anterior.

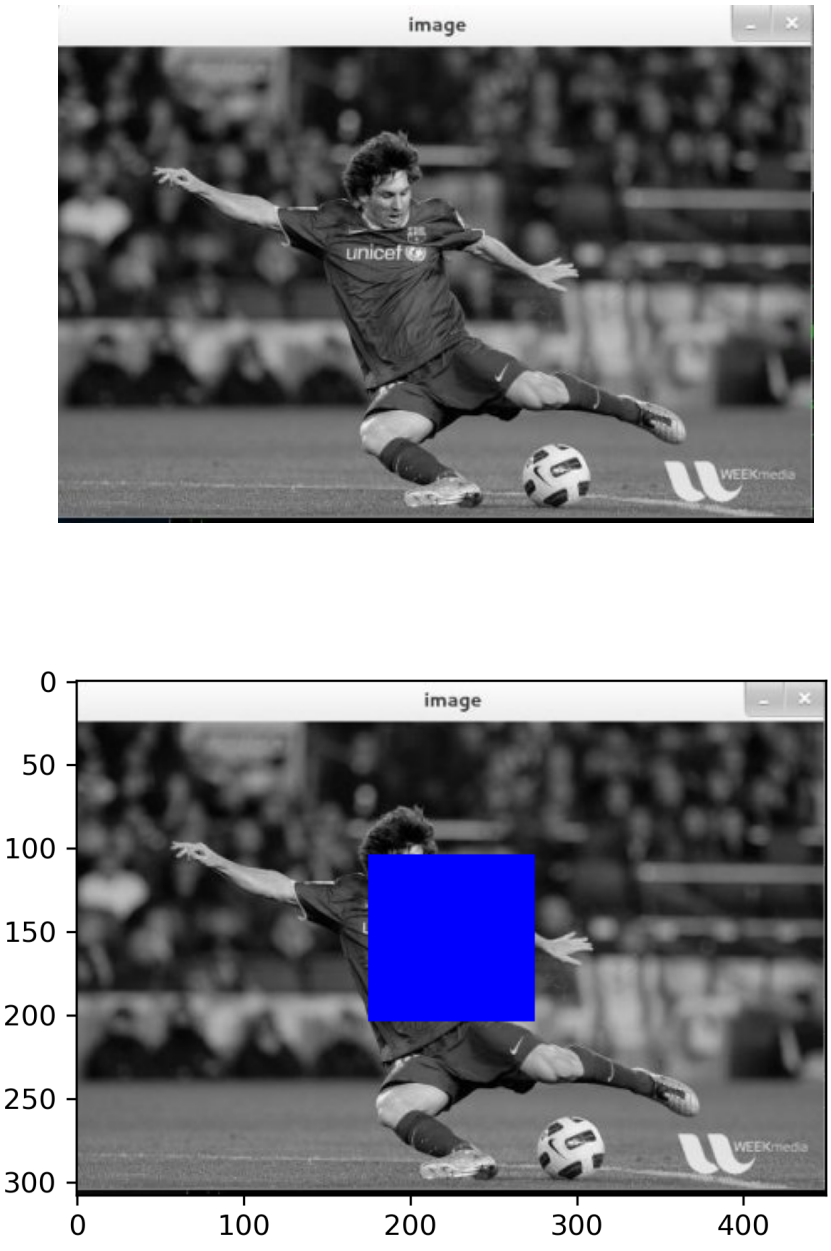

Figura 5: La imagen original y modificada

Concretamente se pide dibujar un cuadrado de 100x100 píxeles: para generar los puntos primero se declaran las variables `xStart` y `yStart` las cuales son un sesgo: es el punto de inicio del cuadrado, y por lo tanto, es el punto que está 50 píxeles antes que el centro de la imagen en cada dimensión. Luego en un bucle `for` anidado se rellena el vector generando matrices Numpy de dos componentes con el sesgo sumado al valor actual del índice que va de 0 a 100, resultando en un vector de 100x100 puntos centrado en la imagen, esto se le pasa como parámetro a la función junto con la imagen a modificar y el color indicado como una 3-tupla, la función luego retorna la imagen con la modificación realizada, el resultado puede verse en la Figura 5.

2.5. Una función que sea capaz de representar varias imágenes con sus títulos en una misma ventana

Para la realización de este ejercicio se creó la función `pintaIMVentana(dictIm)`. Se tomaron varias decisiones puesto que el planteamiento deja ciertos aspectos ambiguos:

- Puesto que no se indican los parámetros del procedimiento, se tomó la decisión de utilizar un diccionario de Python como la estructura de datos que recibe la misma. El diccionario se compone del título de la imagen como clave y la imagen en sí como el valor.
- Al igual que el ejercicio 3, se concatenarán las imágenes horizontalmente.
- Se aceptarán imágenes en color y en escala de grises, utilizando nuevamente la función `Gray2Color()`.

Puesto que para mostrar imágenes se está utilizando Matplotlib, se utilizarán los métodos que posee. En primer lugar se crea un objeto `figure`, esto se puede pensar es la ventana en su totalidad.

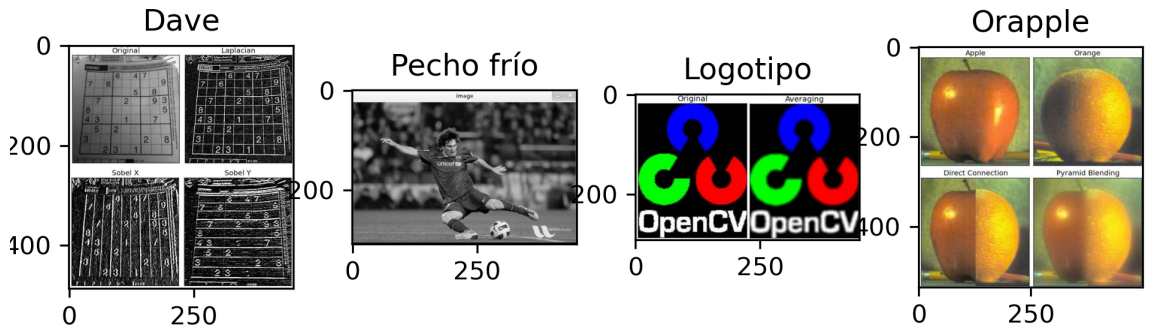

Figura 6: Lista de imágenes con sus títulos

Luego se determina la longitud del diccionario y prosigue un bucle `for` que recorre dicho diccionario, dentro del bucle se crea un `subplot`, el cual puede pensarse es una subdivisión de la ventana, se

indica que será horizontalmente de tamaño de la longitud del diccionario y se insertará cada imagen en el orden que viene en el diccionario, se añade la imagen a la ventana llamando a la función `imshow()`, además se le añade el título con `title()`. Una vez se vuelve a llamar a `subplot()` se empezará a tratar otra subdivisión diferente.

Finalizado el bucle, se indica una opción de `tight_layout()` para espaciar mejor las imágenes y finalmente `show()` para dibujar la ventana con todas las imágenes y sus títulos. Un ejemplo puede observarse en la Figura 6.