Medición del rendimiento con OpenCV

import cv2 as cv

La función cv.getTickCount devuelve el número de ciclos de reloj después de un evento de referencia (como el momento en que se encendió la máquina) hasta el momento en que se llama a esta función. Entonces, si lo llama antes y después de la ejecución de la función, obtendrá la cantidad de ciclos de reloj utilizados para ejecutar una función.

La función cv.getTickFrequency devuelve la frecuencia de los ciclos de reloj o el número de ciclos de reloj por segundo. Entonces, para encontrar el tiempo de ejecución en segundos, puedes hacer lo siguiente:

e1 = cv.getTickCount ()  
# ejecución de tu código  
e2 = cv.getTickCount ()  
tiempo = (e2 - e1)/ cv.getTickFrequency ()

img1 = cv.imread('resources/messi5.jpg')  
assert img1 is not None, "file could not be read, check with os.path.exists()"  
e1 = cv.getTickCount()  
for i in range(5,49,2):  
 img1 = cv.medianBlur(img1,i)  
e2 = cv.getTickCount()  
t = (e2 - e1)/cv.getTickFrequency()  
print( t )

0.388329385

Optimización predeterminada en OpenCV

Muchas de las funciones de OpenCV están optimizadas usando SSE2, AVX, etc. También contiene el código no optimizado. Entonces, si nuestro sistema admite estas funciones, deberíamos explotarlas (casi todos los procesadores modernos las admiten). Está habilitado de forma predeterminada durante la compilación. Entonces OpenCV ejecuta el código optimizado si está habilitado; de lo contrario, ejecuta el código no optimizado. Puede usar cv.useOptimized() para verificar si está habilitado/deshabilitado y cv.setUseOptimized() para habilitarlo/deshabilitarlo. Veamos un ejemplo sencillo.

cv.useOptimized()

%timeit res = cv.medianBlur(img,49)

cv.setUseOptimized(False)

cv.useOptimized()

%timeit res = cv.medianBlur(img,49)

Medición del rendimiento en IPython

x = 5

%tiempoit y=x\*\*2

%tiempoit y=x\*x

z = np.uint8([5])

%tiempoit y=z\*z

%timeit y=np.square(z)

Puedes ver que, x = 5; y = x\*x es más rápido y es aproximadamente 20 veces más rápido en comparación con Numpy. Si consideramos también la creación de matrices, puede llegar a ser hasta 100 veces más rápida. ¿Guay, verdad? Las operaciones escalares de Python son más rápidas que las operaciones escalares de Numpy. Entonces, para operaciones que incluyen uno o dos elementos, el escalar de Python es mejor que las matrices Numpy. Numpy tiene la ventaja cuando el tamaño de la matriz es un poco mayor. Probaremos un ejemplo más. Esta vez, compararemos el rendimiento de cv.countNonZero() y np.count\_nonzero() para la misma imagen.

%timeit z = cv.countNonZero (img)  
  
%timeit z = np.count\_nonzero(img)