Core operations with OpenCV Python

Code Repository :- https://github.com/Shiven-saini/OpenCV-Samples

This section is mostly about how to manipulate core properties of an image object. A strong understanding of numpy library, which is implemented to be computed way faster than what normal math functions in python allows to, is desired to progress ahead.

Things to Learn :-

- Basic Operations on Images
- Arithmetic Operations on Images
- Performance Measurement and Improvement Techniques

1. Basic Operations on Images

Things to learn :-

- Access pixel values and modify them
- Access image properties
- Set a region of interest ROI
- Split and Merge images

Accessing and Modifying pixel values

Load the image to be sampled first :-

```
img = cv.imread("samples/messi5.jpg")
```

Accessing the pixel value, since image is nothing but a $m*n*3$ array.

```
px = img[100, 100]
```

Pass it the x, y coordinates of the pixel you want to access. If the image is of type RGB 8-Bit, It will return a 1-D array of 3 elements (B, G, R) respectively. Optionally, to access only a single color-bit, add 3rd parameter of either 0, 1, 2 (blue, green and red respectively.)

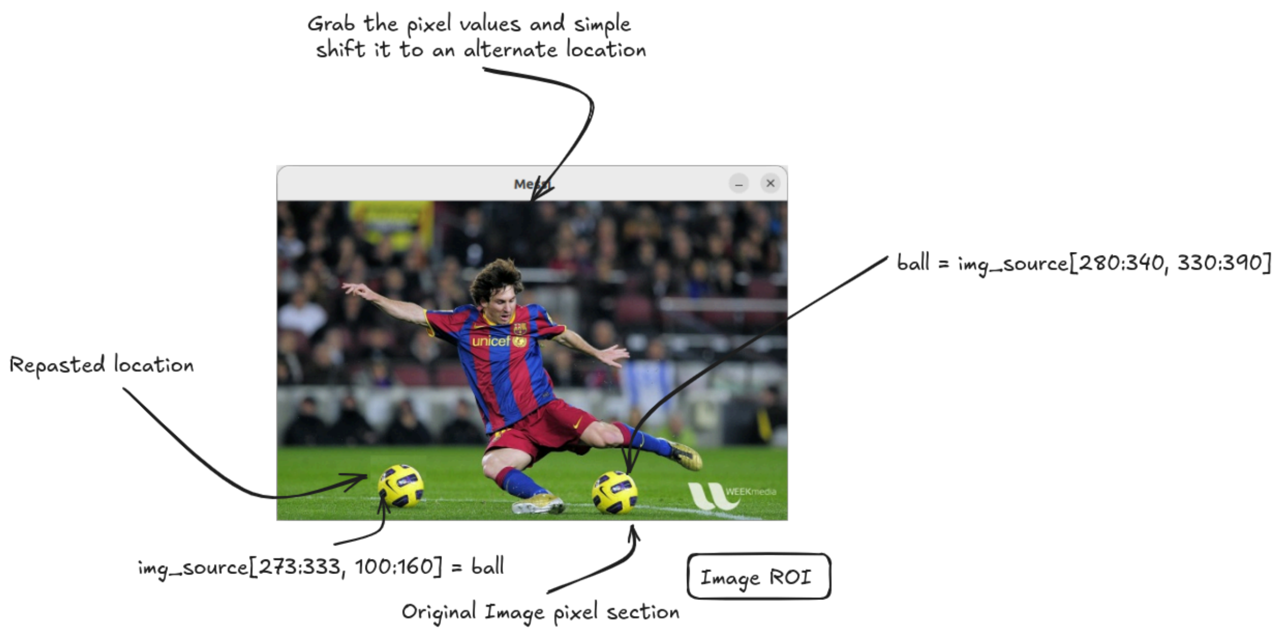
> Remember, OpenCV always operates in the format of B, G, R instead of standard RGB

To get the properties of an image

```
print(img.shape)           # returns number of rows, columns and channels
print(img.size)            # returns the total number of pixels.
print(img.dtype)           # returns data type of an image.
```

> Most of the errors in OpenCV python code are caused by invalid datatype. So, make sure to use .dtype as much as you want while debugging the error.

Image ROI (Region of Images/ Region of interest)


Grab the pixel values and simple shift it to an alternate location
ball = img_source[280:340, 330:390]
Repasted location
img_source[273:333, 100:160] = ball
Original Image pixel section
Image ROI

Sometimes we have to manipulate or are interested in only a smaller section of the image rather than the whole. For example, in a program of eye detection, we are only interested in the person's face rather than the whole image. For this, we can slice off only those range pixels and apply the algorithm on those instead.

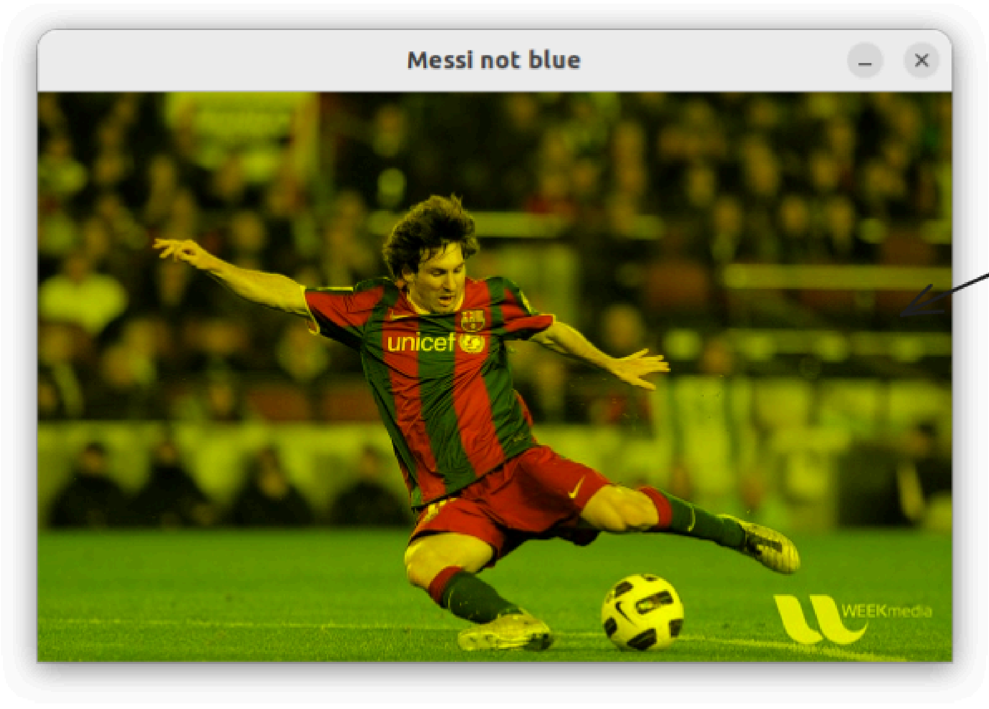
For example in the above image, i grabbed the football pixels from original coordinates (280:340 and 330:390) and repasted it to somewhere else. Please make sure that the coordinates of new location match the ball shape.

Splitting and Merging Image Channels

We can split B, G, R channels, when we want to work with each channel individually. There are two commands available for these operations :-
`b, g, r = cv.split(img)`
`img = cv.merge((b, g, r))`

To manipulate or select only a single channel across an image, use :-

```
b = img[:, :, 0]
g = img[:, :, 1]
```



set every pixel's blue channel to value 0.

2. Arithmetic Operations on Images

Things to learn :-

- Several arithmetic operations on images, like addition, subtraction bitwise operations etc.
- Learn functions : `cv.add()` , `cv.addWeighted()`

Image Addition

We can add two images by using either *OpenCV Function* cv.add() or simply by numpy operation result = img +img2

Difference between OpenCV addition and numpy addition :-

- OpenCV addition is a saturated operation (i.e. fixes the value to maximum possible)
- Numpy Addition is a modulo operation (i.e. use the modulo operando value)
- OpenCV .add function is always preferred over numpy addition.

In the example below :-

```
x = np.uint8([250])
y = np.uint8([20])

print(cv.add(x, y)) => 250 + 20 => 255
print(x+y) => 250 + 20 => 14
```

Image Blending

Also a type of image addition, but different weights are given to images in order to give a blending or transparency effect.

`cv.addWeighted(img1, alpha1, img2, alpha2, scalar)` method can be used to blend images together.

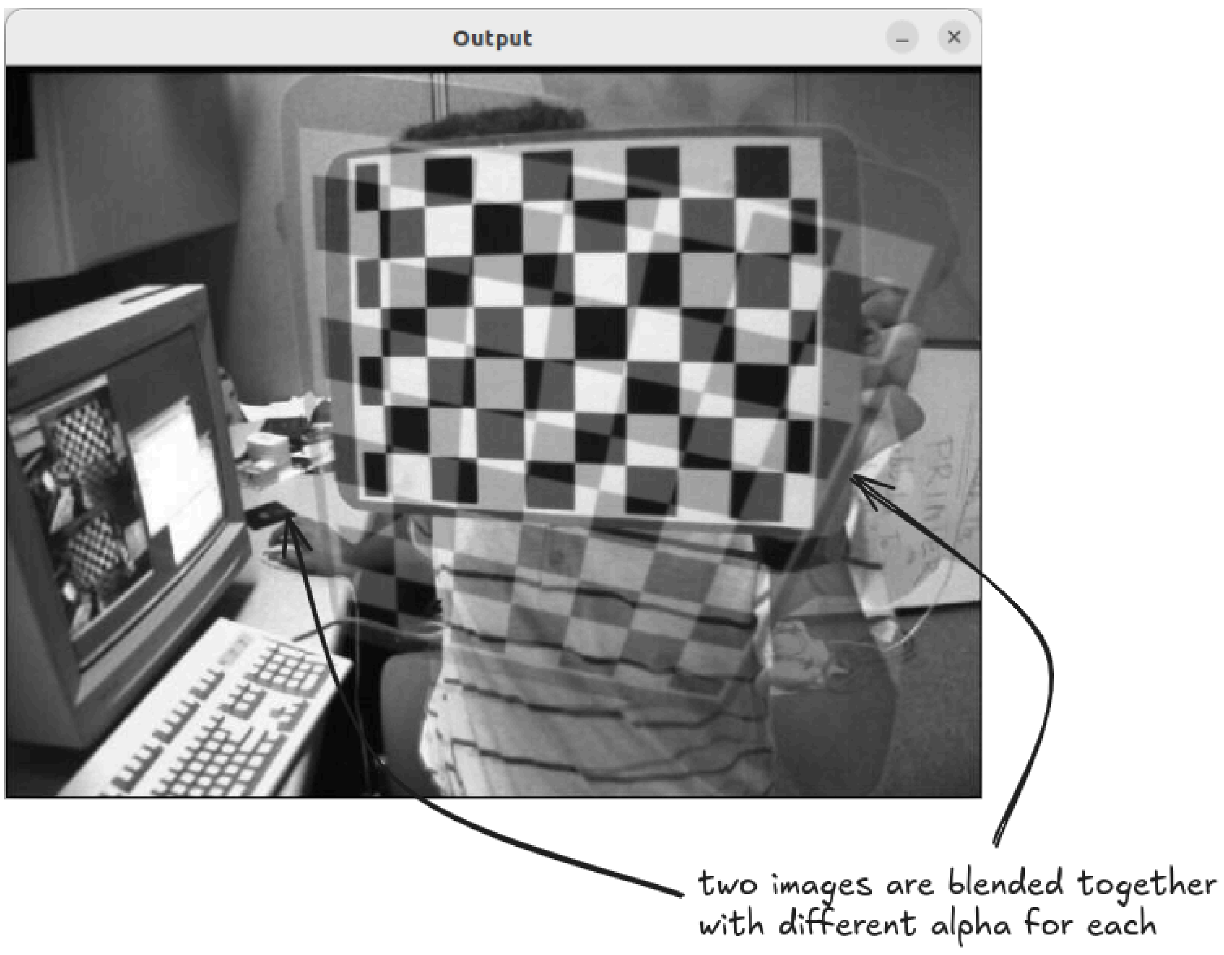

Output
two images are blended together
with different alpha for each

Image pixels are manipulated as per :-

$$h(x) = (1 - a) * f(x) + a * g(x)$$

By varying a from 0 $->$ 1, different transitions can be applied on test image.

```
img1 = cv.imread("samples/left01.jpg")
img2 = cv.imread("samples/left02.jpg")

img_output = cv.addWeighted(img1, 0.7, img2, 0.3, 0)
cv.imshow("Output", img_output)
cv.waitKey(0)
cv.destroyAllWindows()
```

Bitwise Operations

These include bitwise operations like AND, OR, NOT and XOR. They can be used to extract a section of an image (like I did in ROI Section), but the fact is that these operations work best for non-rectangular ROIs.

So, to extract any non-rectangular ROIs => Use Bitwise Operations!

`cv.bitwise_not(), cv.bitwise_and()` etc.

Reference

OpenCV-Python Docs => https://docs.opencv.org/4.x/d3/df2/tutorial_py_basic_ops.html