

Section 1: Fundamentals of Computer Vision with OpenCV

Loading Image: To perform operations on an image(s), it is important to first load the image(s) using OpenCV `cv2.imread(path)`: path A string representing the path of the image to be read. `image = cv2.imread(imgPath)`

Getting and setting arrays: Pixels are the raw building blocks of an image. Every image consists of a set of pixels. The image read with `imread` method is numpy array. The top-left corner of the image is the origin with `[0,0]` which gives the (b,r,g) value

```
(b, g, r) = image[0, 0]
```

General format is `image[y,x]` not `[x,y]`

To replace a particular location

```
Image[23,56] = (0, 0, 255)
```

Rotating: To rotate an image by using `cv2`. It is needed to first get the rotating Matrix `M` = using `cv2.getRotationMatrix2D` taking in three arguments. The first (cx, cy) the point to rotate the image about, the second is the angle of rotation, and last argument is the scale.

Then the actual rotation done by `cv2.warpAffine(img, M, dsize)`

Img is the image to be rotated

M is the rotating Matrix from `cv2.getRotationMatrix2D`

Size is the size of the rotated image u would want to see

```
# rotate our image by 45 degrees around the center of the image
M = cv2.getRotationMatrix2D((cX, cY), 45, 1.0)
rotated = cv2.warpAffine(image, M, (w, h))
cv2.imshow("Rotated by 45 Degrees", rotated)
```

Resizing: `cv2.resize()` method is used with necessary arguments. `cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]])` : src the image already read by `cv2`, dsize is the desired size of the output image, given as tuple, fx is the scaling factor along X-axis or Horizontal axis, fy is the scaling factor along Y-axis or Vertical axis, interpolation could be one of the following values.

`INTER_NEAREST`, `INTER_LINEAR`, `INTER_AREA`, `INTER_CUBIC`, `INTER_LANCZOS4`

Resizing can be done in three ways 1) using the scaling factor to keep the aspect ratio 2) scaling horizontally only 3) scaling vertically only. Upscaling using a scale of 1.5

```
img_scale_up = cv2.resize(img, (0, 0), fx=1.5, fy=1.5)
```

Downscaling using a scale of 0.5: `img_scale_up = cv2.resize(img, (0, 0), fx=0.5, fy=0.5)`
Horizontal scaling: `img_scale_up = cv2.resize(img, (new_width, img.shape[0]))` Vertical
Scaling: `img_scale_up = cv2.resize(img, (img.shape[1], new_height))`

Flipping: This can be done either horizontally or vertically or both ways. Using `cv2.flip(img, diriection)` direction=0 for vertical, 1 for horizontal and -1 for both

Cropping: Image cropping with OpenCV is accomplished via simple NumPy array in startY: endY, startX: endX order.

```
face = image[85:250, 85:220]  
cv2.imshow("Face", face)  
cv2.waitKey(0)
```

Image Arithmetic: Addition can subtraction can be done on an image with another numpy array using `cv2.add()`, `cv2.subtract()`

Image Masking: Image masking means to apply some other image as a mask on the original image or to change the pixel values in the image. First step is to use create the mask array and get the image to be masked but making sure they of the same size. Then using cv2 bitwise operators to perform the masking

```
mask = np.zeros(image.shape[:2], dtype="uint8")  
print(mask)  
cv2.rectangle(mask, (0, 90), (290, 450), 255, -1)  
cv2.imshow("Rectangular Mask", mask)  
  
# apply our mask -- notice how only the person in the image is  
# cropped out  
masked = cv2.bitwise_and(image, image, mask=mask)  
cv2.imshow("Mask Applied to Image", masked)  
cv2.waitKey(0)
```

Splitting and merging image channel: `cv2.split()` is used to **split** an image into three different intensity arrays for each color **channel**. It **return tuple of numpy array of each channel**. Image channels can be merge using the `cv2.merge()` method

`(B, G, R) = cv2.split(image)`

`merged = cv2.merge([B, G, R])`

