Section 2 — Basic Image Processing Operations

Convolutions with OpenCV and Python: Image convolution is simply an element-wise multiplication of two matrices followed by a sum.We can think of an image as a big matrix and kernel or convolutional matrix as a tiny matrix that is used for blurring, sharpening, edge detection, and other image processing functions.cv2.filter2D is use for convolution in opencv. The image can be a grayscale or multichannel image. The kernel can be either custom, low pass filter and high pass filter

```
opencvOutput = cv2.filter2D(image, -1, kernel)
```

Morphological Operations: Morphological operations are simple transformations applied to binary or grayscale images. More specifically, we apply morphological operations to *shapes* and *structures* inside of images. There are different morphological operation erosion, dilation, opening, closing, morphological gradient, black hat and top hat/white hat. Steps involved are 1)convert the image to grayscale using cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) 2)Applying the necessary morphological operation to the grayscale image.

Erosion: An erosion in an image "erodes" the foreground object and makes it smaller. eroded =cv2.erode(imageGray, None, iterations).Erosion is useful for removing small blobs in an image or disconnecting two connected objects.
Dilation: Dilations *increase* the size of foreground objects and are especially useful for joining broken parts of an image together. Dilated dilated=cv2.dilate(imageGray, None, iterations)

Opening: An opening is an *erosion* followed by a *dilation*.

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, kernelSize)
opening = cv2.morphologyEx(gray, cv2.MORPH_OPEN, kernel)
```

The cv2.getStructuringElement function requires two arguments: the first is the type of structuring element we want which is rectangular in this case, and the second is the size of the structuring element. The actual opening operation is performed using cv2.morphologyEx(imageGray, cv2.MORPH_OPEN, kernel).The second argument set to cv2.MORPH_OPEN

Closing: The exact opposite to an opening would be a *closing*. A closing is a *dilation* followed by an *erosion*. The same code process use in opening can be use in closing just the second argument of cv2.morphologyEx() is changed to cv2.MORPH_CLOSE

Morphological Gradient: A morphological gradient is the difference between a *dilation* and *erosion*. It is useful for determining the outline of a particular object of an image.The same code process use in opening can be use in gradient just the second argument of cv2.morphologyEx() is changed to cv2.MORPH_GRADIENT.

Top hat: A *top hat* (also known as a *white hat*) morphological operation is the difference between the original (grayscale/single channel) *input image* and the *opening*.

A top hat operation is used to reveal bright regions of an image on dark backgrounds.

cv2.MORPH_TOPHAT is used
Black hat: The black-hat operation is used to do the opposite, enhance dark objects of interest in a bright background.cv2.MORPH_BLACKHAT

Smoothing and Blurring: This is a low filter operation needed to be taken to remove noise and edge(high frequency content) from an image.It should be applied on an image before edge detection or thresholding we make the color transition from one side of an edge in the image to another smooth rather than sudden. Types of blurring are simple average,gaussian blurring, weighted average, median filtering and bilateral filtering.

average_blur = cv.blur(img,kernel_size)
gaussian_blur = cv2.GaussianBlur(image, kernel_size),) is the standard deviation of the gaussian distribution by default it is 0
median_blur= cv2.medianBlur(image, kernel_size)
bilateral_blurring= cv2.bilateralFilter(image, diameter, sigmaColor, sigmaSpace)

Color space: Color space means the use of a specific color model or system that turns colors into numbers. There are different color spaces that can be used in opencv 1)brg(rgb)2)HSV 3)L*a*b 4)grayscale. To convert from one color space to the other

cv2.cvtColor(image, code) code It is the color space conversion code

```
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Thresholding (cv2.threshold): Thresholding is a very popular segmentation technique, used for separating an object considered as a foreground from its background. this technique of thresholding is done on grayscale images. In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255). Two methods are
1)Basic thresholding where you have to manually supply a threshold value, *T*
2)Otsu's thresholding, which automatically determines the threshold value

cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)
(T, thresh) = cv2.threshold(blurredImage, 200, 255, cv2.THRESH_BINARY)-for basic thresholding
(T, threshInv) = cv2.threshold(blurred, 0, 255,cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU) -otsu thresholding

Adaptive thresholding: Adaptive thresholding is the method where the threshold value is calculated for smaller regions.

```
cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY_INV, 21, 10)
thresh = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, 21, 4)
```

Image gradient: Image gradient is nothing but directional change in image intensity. By computing gradient for a small area of image and repeating the process for entire image, we can detect edges in images. Hence we can use it to detect edges by finding contours and outline of objects in images. Histogram of Oriented Gradients and SIFT are built upon image gradient.

```
# load the input image and convert it to grayscale
image = cv2.imread(args["image"])
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# compute gradients along the x and y axis, respectively
gX = cv2.Sobel(gray, cv2.CV_64F, 1, 0)
gY = cv2.Sobel(gray, cv2.CV_64F, 0, 1)
# compute the gradient magnitude and orientation
magnitude = np.sqrt((gX ** 2) + (gY ** 2))
orientation = np.arctan2(gY, gX) * (180 / np.pi) % 180
```

For scharr **cv2.Scharr()** is used.

Canny Edge detection: Detecting edges in computer vision are very important. There are different types of edges 1) step2) ramp3) roof 4) ridge. This edge can be detected using image gradient but gradient are sensitive to noise. Canny edge detection does edge detection using an algorithm
1)Gaussian smoothing 2) Computing the gradient magnitude and orientation 3) Non-maxima suppression 4) Hysteresis thresholding

```
image = cv2.imread(args["image"])
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
# compute a "wide", "mid-range", and "tight" threshold for the edges
# using the Canny edge detector
wide = cv2.Canny(blurred, 10, 200)
mid = cv2.Canny(blurred, 30, 150)
tight = cv2.Canny(blurred, 240, 250)
```

Zero-parameter, automatic Canny edge detection with Python and OpenCV

Canny in open cv is cv2.canny(image, lower, upper), where the lower and upper threshold are mean to be provided. Questions on which lower and upper threshold to be used needs to be answered, what is the optimal threshold to be used especially when multiple images are involved

with different lighting condition. The answer to all these give rise to automatic canny edge detection.

To automatically do canny edge detection

1. Find the median intensity of the grayscale image
2. Using sigma value calculate the lower and upper threshold
3. Then use the calculated thresholds cv2.canny(image, lower, upper)

```
def auto_canny(image, sigma=0.33):
    # compute the median of the single channel pixel intensities
    v = np.median(image)
    # apply automatic Canny edge detection using the computed median
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    edged = cv2.Canny(image, lower, upper)
    # return the edged image
    return edged

auto = auto_canny(blurred)
```