**Open CV Assignment 5**

**1. What exactly is the HOG Filter?**

A. Histogram of Oriented Gradients, also known as HOG, is a feature descriptor like the Canny Edge Detector, SIFT (Scale Invariant and Feature Transform). It is used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in the localized portion of an image. This method is quite similar to Edge Orientation Histograms and Scale Invariant Feature Transformation (SIFT). The HOG descriptor focuses on the structure or the shape of an object. It is better than any edge descriptor as it uses magnitude as well as angle of the gradient to compute the features. For the regions of the image it generates histograms using the magnitude and orientations of the gradient.

**2. Explain how to use OpenCV to enhance images.**

A. We will follow the below steps:

**Step 1:** Import the necessary libraries

**Step 2**: Load the image and display the sample image.

**Step 3**: Convert the images into a grayscale image.

**Step 4**: Find out the histograms of the grayscale image and look for the distribution of intensities.

**Step 5**: Use cv2.equalizeHist() function with the purpose of equalizing the contrast of a given grayscale image. cv2.equalizeHist() function normalizes the brightness and also increases the contrast.

**Step 6**: Display the Gray Scale Histogram equalized images

**Step 7**: Contrast Limited Adaptive Histogram Equalization

This algorithm can be applied to improve the contrast of the images. This algorithm works by creating several histograms of the image and uses all of these histograms to redistribute the lightness of the image. CLAHE can be applied to greyscale as well as colour images. There are 2 parameters to tune.

1. clip limit which sets the threshold for contrast limiting. The default value is 40
2. tileGridsize which sets the number of titles in the row and column. While applying CLAHE image is divided into small blocks called tiles (8\*8) in order to perform calculations.

**Step 8**: Thresholding Techniques

Thresholding is a simple, yet effective method for image partitioning into a foreground and background. The simplest thresholding methods replace each pixel in the source image with a black pixel if the pixel intensity is less than some predefined constant(the threshold value)or a white pixel if the pixel intensity is greater than the threshold value. Different types of Thresholding are:-

cv2.THRESH\_BINARY, cv2.THRESH\_BINARY\_INV, cv2.THRESH\_TRUNC, cv2.THRESH\_TOZERO, cv2.THRESH\_TOZERO\_INV, cv2.THRESH\_OTSU, cv2.THRESH\_TRIANGLE

**Step 9**: Adaptive Thresholding

The obtained results were not very good due to the different illumination conditions in the different areas of the image. In these cases, you can try adaptive thresholding. In OpenCV, the adaptive thresholding is performed by the **cv2.adapativeThreshold()** function

This function applies an adaptive threshold to the src array (8­bit single­channel image). The maxValue parameter sets the value for the pixels in the dst image for which the condition is satisfied. The adaptiveMethod parameter sets the adaptive thresholding algorithm to use .

cv2.ADAPTIVE\_THRESH\_MEAN\_C: The T(x, y) threshold value is calculated as the **mean** of the blockSize x blockSize neighbourhood of (x, y) minus the C parameter.  
cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C: The T(x, y) threshold value is calculated as the **weighted sum** of the blockSize x blockSize neighbourhood of (x, y) minus the C parameter.

The blockSize parameter sets the size of the neighbourhood area used to calculate a threshold value for the pixel, and it can take the values 3, 5, 7,… and so forth.

The C parameter is just a constant subtracted from the means or weighted means (depending on the adaptive method set by the adaptiveMethod parameter). Commonly, this value is positive, but it can be zero or negative.

**Step 10**: OTSU Binarization

Otsu’s binarization algorithm, which is a good approach when dealing with bimodal images. A bimodal image can be characterized by its histogram containing two peaks. Otsu’s algorithm automatically calculates the optimal threshold value that separates both peaks by maximizing the variance between two classes of pixels. Equivalently, the optimal threshold value minimizes the intra­class variance. Otsu’s binarization algorithm is a statistical method, because it relies on statistical information derived from the histogram (for example, mean, variance, or entropy)

**3. What is the name of the operator? With an example, explain any one of the operators.**

A. There are many operators like: Arithmetic, Bitwise, Logical,etc

Lets say, about bitwise operators: In bitwise operators we have AND, OR, XOR, NOT operators.

Lets consider we have a rectangle and square, and we can create them as below:

# import the necessary packages

import numpy as np

import cv2

# draw a rectangle

rectangle = np.zeros((300, 300), dtype="uint8")

cv2.rectangle(rectangle, (25, 25), (275, 275), 255, -1)

cv2.imshow("Rectangle", rectangle)

# draw a circle

circle = np.zeros((300, 300), dtype = "uint8")

cv2.circle(circle, (150, 150), 150, 255, -1)

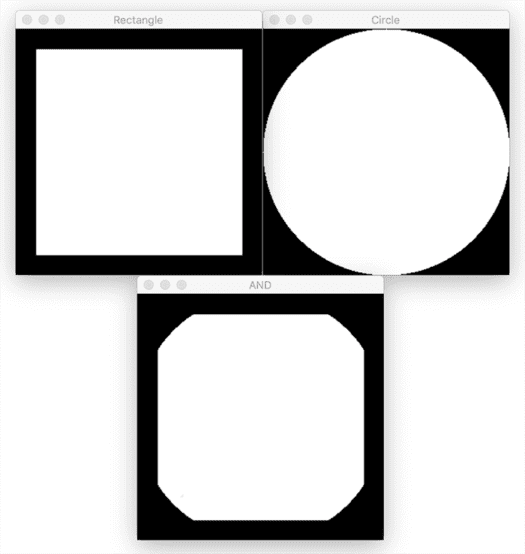
cv2.imshow("Circle", circle)

To utilize bitwise functions, we assume that we are comparing two pixels (the only exception is the NOT function). We’ll compare each of the pixels and then construct our bitwise representation. Let’s quickly review our binary operations:

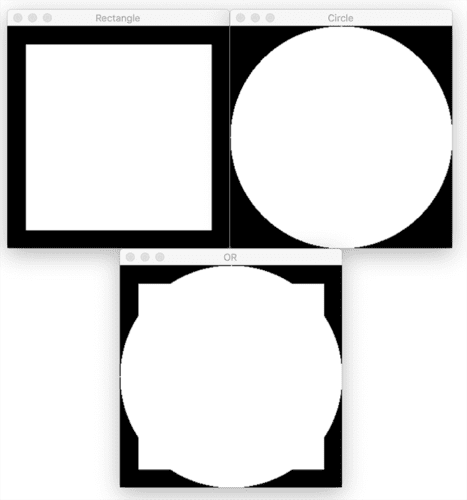
* **AND:** A bitwise AND is true *if and only if* both pixels are greater than zero.
* **OR:** A bitwise OR is true *if either* of the two pixels is greater than zero.
* **XOR:** A bitwise XOR is true *if and only if* one of the two pixels is greater than zero, *but not both.*
* **NOT:** A bitwise NOT inverts the “on” and “off” pixels in an image.

Now lets use these on the rectangle and circle:

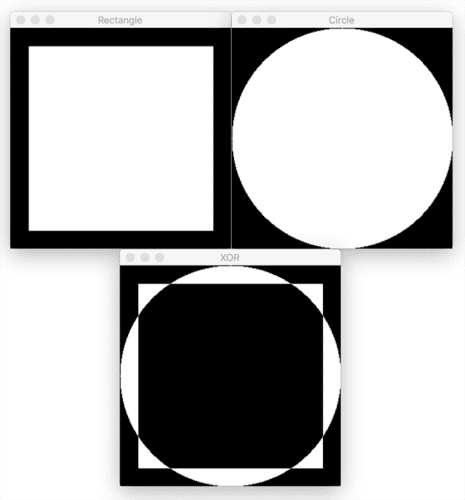
1. AND: bitwiseAnd = cv2.bitwise\_and(rectangle, circle)



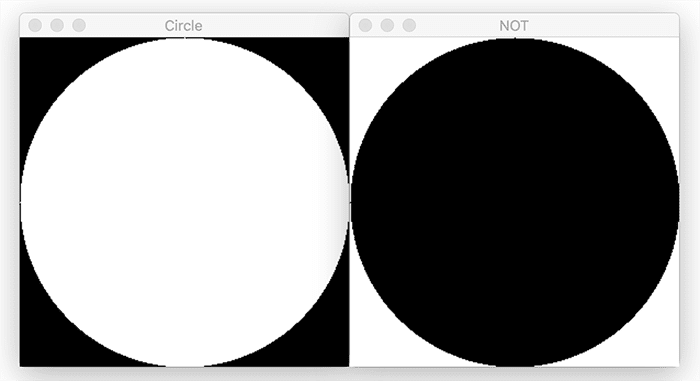
2. OR: bitwiseOr = cv2.bitwise\_or(rectangle, circle)



3. XOR: bitwiseXor = cv2.bitwise\_xor(rectangle, circle)

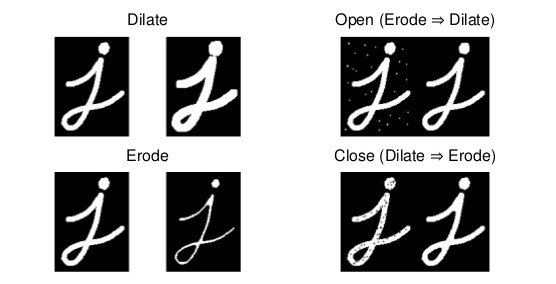


4. NOT: bitwiseNot = cv2.bitwise\_not(circle)



**4. With an example, explain the importance of erosion and dilation.**

A.



**Importance of Erosion and Dilation:**

1. Erosion:
   * It is useful for removing small white noises i.e, here in the opening, the noises are removed and the image becomes thinner and will become thicker on dilation
   * Used to detach two connected objects etc.
2. Dilation:
   * In cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won’t come back, but our object area increases.
   * It is also useful in joining broken parts of an object i.e, here in closing, the j is having some broken parts which are corrected by dilation which will close broken parts with white pixels, but the j becomes thicker so we perform erosion

**5. Explain how the haar cascade will be used to detect faces.**

A. Haar Cascade classifiers are an effective way for object detection. This method was proposed by Paul Viola and Michael Jones in their paper [Rapid Object Detection using a Boosted Cascade of Simple Features](https://www.researchgate.net/publication/3940582_Rapid_Object_Detection_using_a_Boosted_Cascade_of_Simple_Features). Haar Cascade is a machine learning-based approach where a lot of positive and negative images are used to train the classifier.

* **Positive images –** These images contain the images which we want our classifier to identify.
* **Negative Images –** Images of everything else, which do not contain the object we want to detect.

Steps to use Haar Cascade to detect objects:

1. Import libraries
2. Load Haar Cascade
3. Load the image
4. Convert image into grayscale
5. Detecting coordinates around the detected object
6. Drawing a bounding box using the detected coordinates

We can write the code as:

import numpy as np

import cv2

#---loading haarcascade detector---face\_detector=cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

#---Loading the image-----

img = cv2.imread('team\_india.jpg')

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

faces = face\_detector.detectMultiScale(gray, 1.3, 5)

for (x,y,w,h) in faces:

cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)

cv2.imshow('img',img)

cv2.waitKey(0)

cv2.destroyAllWindows()