East West University

Department of CSE

Lab Report 04

CSE 438

Digital Image Processing

Submitted To:

Md. Mahir Ashhab

Lecturer

Department of Computer Science and Engineering

Submitted By:

Adri Saha

ID: 2019-1-60-024

Submission Date: 22 August 2022

Bitwise operations:

Images can be altered using bitwise techniques. These bitwise techniques can be used to create masks of the image, apply watermarks to existing images, and perform other operations on existing images in computer vision applications.

In these operations first we used single channel. Then for area selection we used multiple channels which is shown in masking.

Rectangle

Blank is single channel. Size & shape will not match. That's why we can't directly apply bitwise AND/OR operation with blank. To sort this

- We can use bitwise operator to 3 channel images with 3 channel image and select carrier depending on 1D circle or rectangle.
- But we used 1 channel masking. So initialized blank variable first.

```
blank= np.zeros((400,400),dtype='uint8')

rectangle=cv.rectangle(blank.copy(),(30,30),(370,370),255,-1)

cv.imshow('rectangle',rectangle)
```

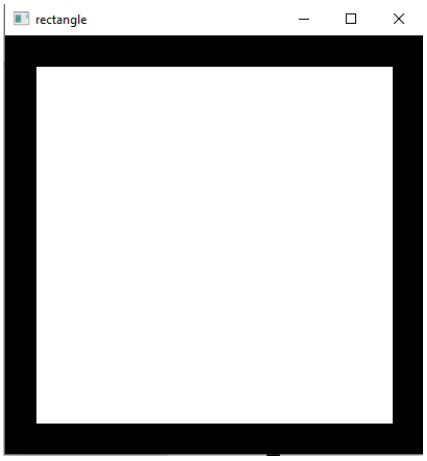

Figure 1: Rectangle

Circle

We also used blank here for 1 channel masking.

```
circle= cv.circle(blank.copy(),(200,200),200,255,-1)

cv.imshow('circle',circle)
```

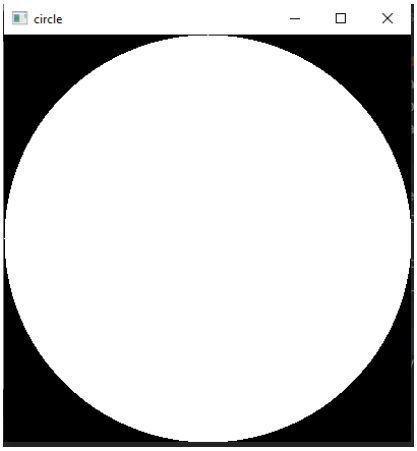

Figure 2: Circle

Bitwise AND

bit_and = cv.bitwise_and(rectangle,circle)

cv.imshow('bitwise_AND',bit_and)

Circle and rectangle in the center, white. the rest is entirely black. Using the cv2.bitwise and function, we applied a bitwise AND to our rectangle and circle pictures. Only when both pixels are greater than zero is a bitwise AND considered to be true. Our bitwise AND's output is visible. Since our rectangle does not cover the same area as the circle, both square's borders are lost, which makes sense given that neither pixel is "on."

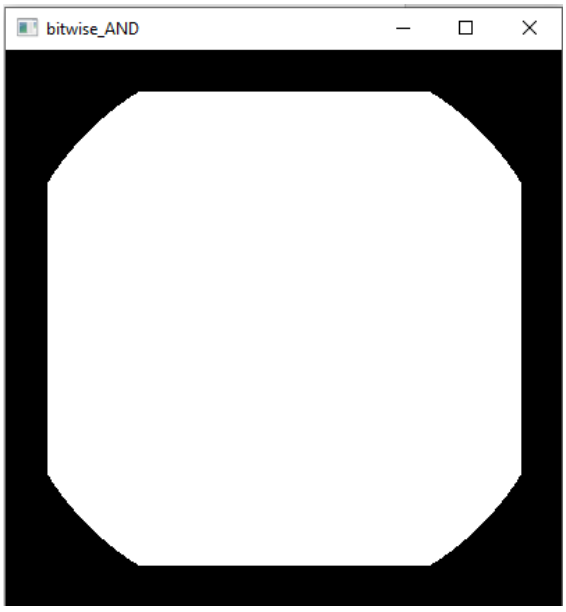

Figure 3: Bitwise AND on circle and rectangle

In this case, our square and rectangle have been combined.

Bitwise OR

A bitwise OR is true if either of the two pixels is greater than zero. We apply a bitwise OR using the cv2.bitwise_or function.

bit_or = cv.bitwise_or(rectangle,circle)

cv.imshow('bitwise_OR',bit_or)

Circle and rectangle side white, rest of all black.

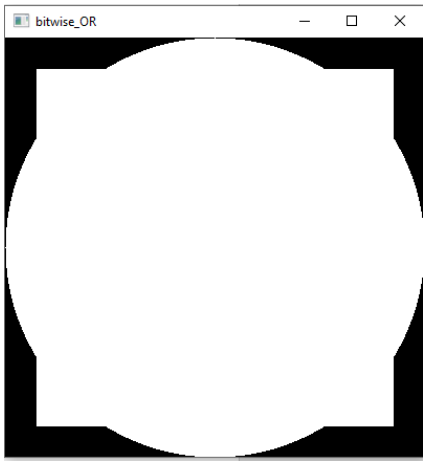

Figure 3: Bitwise OR on circle and rectangle

Bitwise X-OR:

One of the two pixels must be greater than zero for an XOR operation to be valid, but neither pixel may be greater than zero.

Using the cv2.bitwise xor function, we applied the bitwise XOR.

bit_xor = cv.bitwise_xor(rectangle,circle)

cv.imshow('bitwise_XOR',bit_xor)

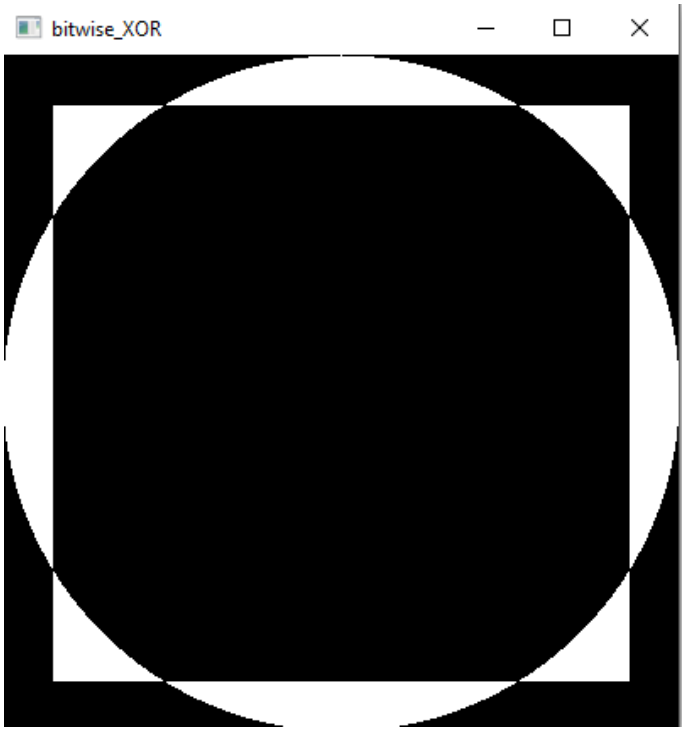

Figure 4: Bitwise XOR on circle and rectangle

The square's center has been cut out in this instance, as can be seen. Again, this makes sense because two pixels cannot be greater than zero in an XOR operation.

Circle bitwise NOT: An image's "on" and "off" pixels are inverted by a bitwise NOT. Using the cv2.bitwise not function, we apply a bitwise NOT. The bitwise NOT function essentially flips pixel values. The value of zero is applied to all pixels greater than zero, while the value of 255 is applied to all pixels equal to zero.

```
circle_bit_not = cv.bitwise_not(rectangle,circle)

cv.imshow('circle bitwise_NOT',circle_bit_not)
```

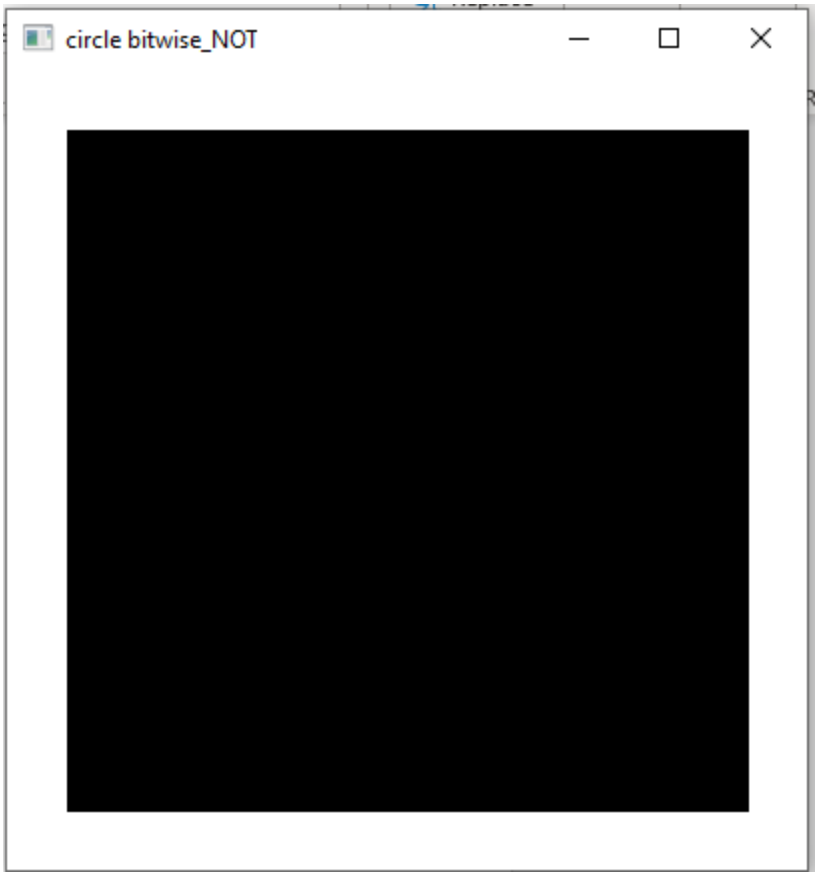

Figure 5: Bitwise NOT on circle and rectangle

The circle was once white on a black background, but now it is black on a white background. From this we can see how our circle and rectangular have been flipped.

Actual image

Masking: A common method for obtaining the Region of Interest is masking (ROI). Using the draw function and bitwise operation in openCV, any masking shape may be created. A post-processing method called image masking is used to separate particular areas of an image. This is helpful if you want to alter your photos selectively or if you want to hide a particular area of your image.

A binary image of zero- and non-zero values is referred to as a mask. All pixels that are zero in the mask are set to zero in the output image when a mask is applied to another binary or to a grayscale image of the same size.

To divide a picture on a specific segment we used circle masked and rectangle masked here.

Circle masked

Initialize blank to have the same dimensions as our original image and to be filled with zeros.

initially only having one channel. likewise, a bit operation on an image with an image. We send the rectangle of the mask to choose area. Then c onvert three blank channels to BGR. then use it to mask.

In this case, bitwise AND was used. We created a mask out of black.

Three blank channels were combined to create a three-channel BGR picture. on three channels, image conversion So I created a white image by setting the parameters B= 255, G= 255, and R= 255. Here we used 3 channels for image masking. R, G, B=(255,255,255) by relating image with image.

```
blank=np.zeros(img.shape[:2], dtype='uint8')

circle=cv.circle(blank.copy(),(img.shape[1]//2,img.shape[0]//2),150,(255,255,255),-1).

shape= cv.bitwise_and(circle,rectangle)

circle_masked= cv.bitwise_and(img,img,mask=circle)
```

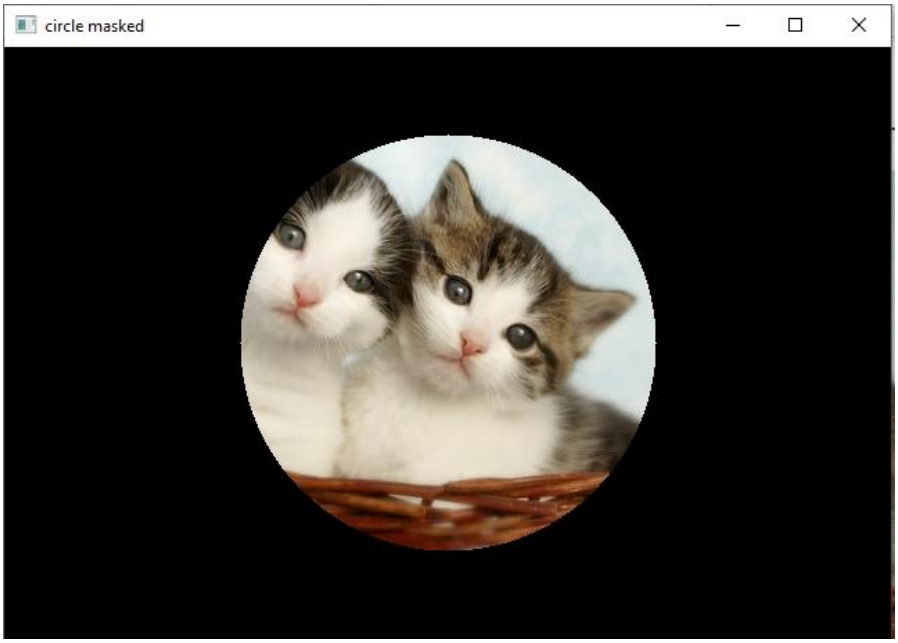

Figure 6: Circle masked using bitwise AND

So, output is a circle on black picture.

Rectangle masked

We could remove only the area of the image containing the individual and ignore the rest using our rectangle mask.
We re-initialize our mask with the same dimensions as our original image and a zero-filled area.

```
rectangle= cv.rectangle(blank.copy(),(30,30),(370,370),(255,255,255),-1)
shape= cv.bitwise_and(circle,rectangle)
rectangle_masked=cv.bitwise_and(img,img,mask=rectangle)
```

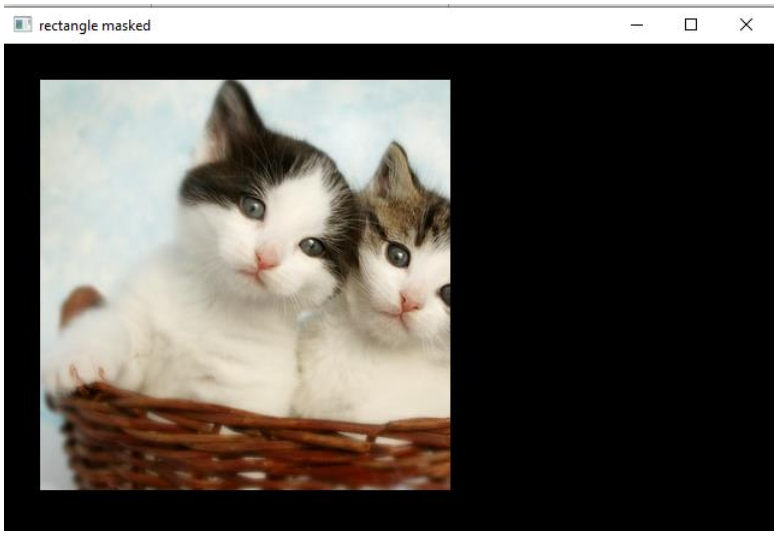

Figure 7: Rectangle masked using bitwise AND