DIGITAL IMAGE PROCESSING COURSE - 505060
PRACTICE LABS

LAB 03. BINARY IMAGE PROCESSING

Requirements

(1) Follow the instructions with the help from your instructor.
(2) Finish all the exercises in class and do the homework at home. You can update your solutions after class and re-submit all your work together with the homework.
(3) Grading
Total score = 50% * Attendance + 50% * Exercises
Rules:
- If the number of finished exercises is less than **80% total number of excercises**, you will get **zero** for the lab.
- Name a source file as "**src_XX.py**" where XX is the exercise number, for ex., "src_03.py" is the source code for the Exercise 3.
- Add the text of your Student ID to each of the output image.
- Name an output image as "**image_XX_YY.png**" where XX is the exercise number and YY is the order of output images in the exercise, for ex., "image_03_02.png" is the second output image in the Exercise 3.
- Submit the source code and output image files directly to Google classroom assignment, donot compress the files.

If you submit the exercises with wrong rules, you will get **zero** for the lab or the corresponding exercises.

(4) Plagiarism check
If any 2 of the students have the same output images, then all will get zero for the corresponding exercises.

INTRODUCTION

In this Lab, you will learn how to
- Bitwise Operations
- Masking of images
- Alpha Blending with OpenCV
- Image thresholding

INSTRUCTIONS

1. Bitwise Operations

OpenCV: Operations on arrays: ***bitwise_and()***
```
dst = cv2.bitwise_and(src1, src2[, dst[, mask]])
```

`cv2.bitwise_and()` is a function that performs bitwise AND processing as the name suggests. The AND of the values for each pixel of the input images `src1` and `src2` is the pixel value of the output image.

Other Bitwise operation in Python (OR, XOR, NOT, SHIFT)

```
dst = cv2.bitwise_or(src1, src2[, dst[, mask]])
dst = cv2.bitwise_xor(src1, src2[, dst[, mask]])
dst = cv2.bitwise_not(src[, dst[, mask]])
```

Source: https://www.digitalocean.com/community/tutorials/arithmetic-bitwise-and-masking-python-opencv

Let's create a bitwise square and a bitwise circle through which we can use the bitwise operations.

```
# creating a rectangle
rectangle = np.zeros((300, 300), dtype="uint8")
cv2.rectangle(rectangle, (25, 25), (275, 275), 255, -1)
cv2.imshow("Rectangle : ", rectangle)

# creating a circle
circle = np.zeros((300, 300), dtype="uint8")
cv2.circle(circle, (150, 150), 150, 255, -1)
cv2.imshow("Circle : ", circle)

# the bitwise_and function executes the AND operation
# on both the images
bitwiseAnd = cv2.bitwise_and(rectangle, circle)
cv2.imshow("AND", bitwiseAnd)
cv2.waitKey(0)

# the bitwise_or function executes the OR operation
# on both the images
bitwiseOr = cv2.bitwise_or(rectangle, circle)
cv2.imshow("OR", bitwiseOr)
cv2.waitKey(0)

# the bitwise_xor function executes the XOR operation
# on both the images
bitwiseXor = cv2.bitwise_xor(rectangle, circle)
cv2.imshow("XOR", bitwiseXor)
cv2.waitKey(0)

# the bitwise_not function executes the NOT operation
# on both the images
bitwiseNot = cv2.bitwise_not(rectangle, circle)
cv2.imshow("NOT", bitwiseNot)
cv2.waitKey(0)
```

2. Masking of images

Masking is used in Image Processing to output the Region of Interest, or simply the part of the image that we are interested in. We tend to use bitwise operations for masking as it allows us to discard the parts of the image that we do not need.

We have three steps in masking.

1. Creating a **black** canvas with ***the same dimensions*** as the image, and naming it as mask.
2. Changing the values of the mask by drawing any figure in the image and providing it with a **white** color.
3. Performing the **bitwise AND** operation on the image with the mask.

Example:

Masking with a rectangle mask.

```
# creating a mask of that has the same dimensions of
the image
# where each pixel is valued at 0
mask = np.zeros(image.shape[:2], dtype="uint8")

# creating a rectangle on the mask
# where the pixels are valued at 255
cv2.rectangle(mask, (0, 90), (290, 450), 255, -1)
cv2.imshow("Mask", mask)

# performing a bitwise_and with the image and the mask
masked = cv2.bitwise_and(image, image, mask=mask)
cv2.imshow("Mask applied to Image", masked)
cv2.waitKey(0)
```

Masking with a circle mask.

```
# creating a mask of that has the same dimensions of
the image
# where each pixel is valued at 0
mask = np.zeros(image.shape[:2], dtype="uint8")

# creating a rectangle on the mask
# where the pixels are valued at 255
cv2.circle(mask, (145, 200), 100, 255, -1)
cv2.imshow("Mask", mask)

# performing a bitwise_and with the image and the mask
masked = cv2.bitwise_and(image, image, mask=mask)
cv2.imshow("Mask applied to Image", masked)
cv2.waitKey(0)
```

3. Alpha Blending with OpenCV

Use `cv2.addWeighted()` to do alpha blending with OpenCV.

```
dst = cv2.addWeighted(src1, alpha, src2, beta, gamma[, dst[, dtype]])
```

It is calculated as follows according to parameters. The fifth parameter `gamma` is the value to be added to all pixel values.

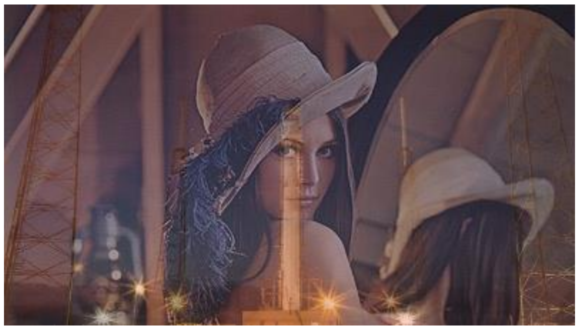
```
dst = src1 * alpha + src2 * beta + gamma
```

The two images need to be the same size, so resize them.

```
import cv2
src1 = cv2.imread('lena.jpg')
src2 = cv2.imread('rocket.jpg')
src2 = cv2.resize(src2, src1.shape[1::-1])
```

The image is alpha blended according to the values of the second parameter `alpha` and the fourth parameter `beta`. Although images are saved as files here, if you want to display them in another window, you can use `cv2.imshow()` (eg: `cv2.imshow('window_name', dst)`). The same is true for the following sample code.

```
dst = cv2.addWeighted(src1, 0.5, src2, 0.5, 0)
cv2.imwrite('opencv_add_weighted.jpg', dst)
```



4. Image thresholding

Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting grayscale images into binary images. Image thresholding is most effective in images with high levels of contrast.

The basic Thresholding technique is Binary Thresholding. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value. The different Simple Thresholding Techniques are:

```
th, dst = cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)
```

- cv2.*THRESH_BINARY*

  If pixel intensity is greater than the set threshold, value set to *maxval*, else set to 0 (black)
- cv2.*THRESH_BINARY_INV*

  Inverted case of cv2.THRESH_BINARY
- cv.*THRESH_TRUNC*

  If pixel intensity value is greater than threshold, it is truncated to the threshold. The pixel values are set to be the same as the threshold. All other values remain the same.
- cv.*THRESH_TOZERO*

  Pixel intensity is set to 0, for all the pixels intensity, less than the threshold value.
- cv.*THRESH_TOZERO_INV*

  Inverted case of cv2.THRESH_TOZERO.

Binary

$$\mathtt{dst}(x,y) = \begin{cases} \mathtt{maxval} & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Inverted Binary

$$\mathtt{dst}(x,y) = \begin{cases} 0 & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ \mathtt{maxval} & \text{otherwise} \end{cases}$$

Truncated

$$\mathtt{dst}(x,y) = \begin{cases} \mathtt{threshold} & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ \mathtt{src}(x,y) & \text{otherwise} \end{cases}$$

To Zero

$$\mathtt{dst}(x,y) = \begin{cases} \mathtt{src}(x,y) & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

To Zero Inverted

$$\mathtt{dst}(x,y) = \begin{cases} 0 & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ \mathtt{src}(x,y) & \text{otherwise} \end{cases}$$

```python
# import opencv
import cv2

# Read image
src = cv2.imread("threshold.png", cv2.IMREAD_GRAYSCALE);

# Basic threhold example
th, dst = cv2.threshold(src, 0, 255, cv2.THRESH_BINARY);
cv2.imwrite("opencv-threshold-example.jpg", dst);

# Thresholding with maxValue set to 128
th, dst = cv2.threshold(src, 0, 128, cv2.THRESH_BINARY);
cv2.imwrite("opencv-thresh-binary-maxval.jpg", dst);

# Thresholding with threshold value set 127
th, dst = cv2.threshold(src,127,255, cv2.THRESH_BINARY);
cv2.imwrite("opencv-thresh-binary.jpg", dst);

# Thresholding using THRESH_BINARY_INV
th, dst = cv2.threshold(src,127,255, cv2.THRESH_BINARY_INV);
cv2.imwrite("opencv-thresh-binary-inv.jpg", dst);
```

Source: https://learnopencv.com/opencv-threshold-python-cpp/

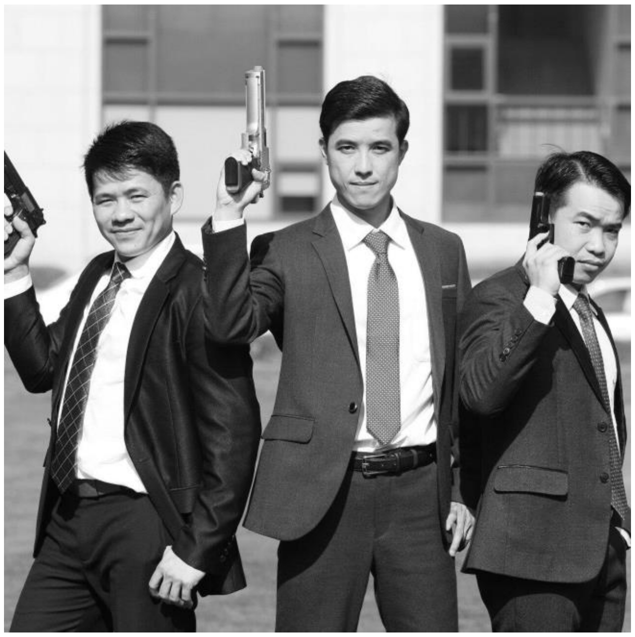
EXERCISE

Ex1. Masking image

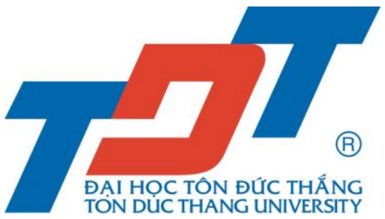
Extracting the faces from the input image using a circular mask.

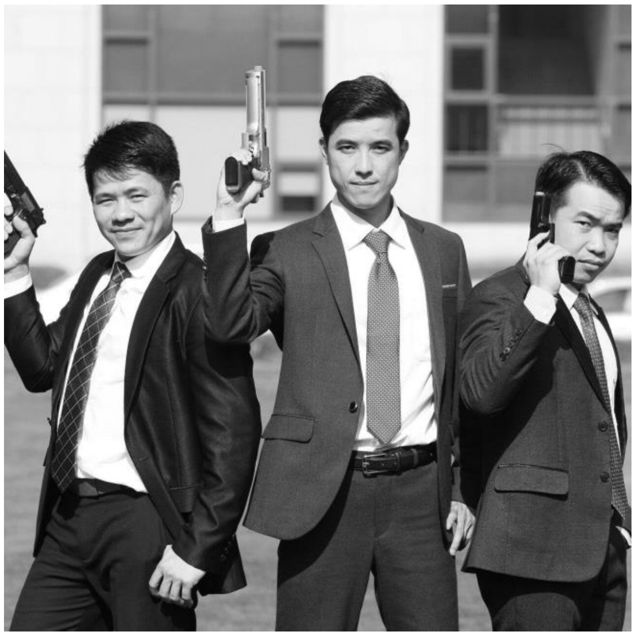
Output is 3 of face images extracted from the input image.

Hint: Using circular masks as in the instructions above.



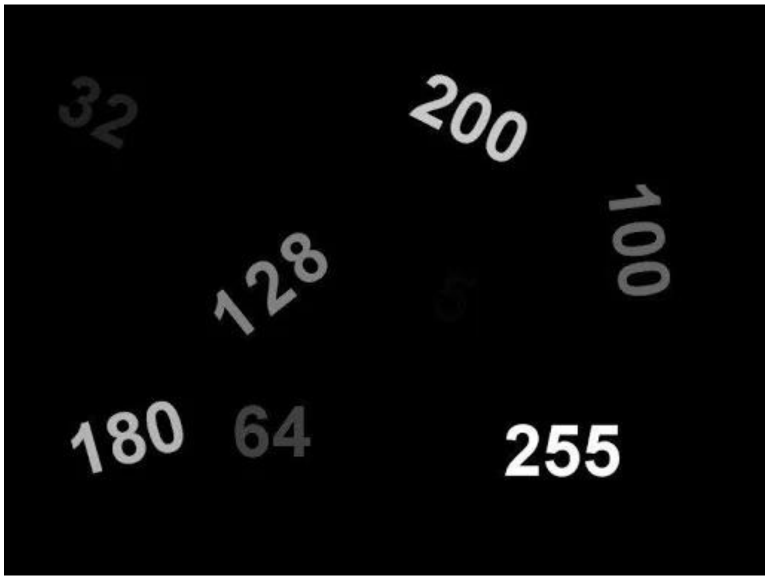
Ex2. Blend the following 2 images to create a new image





Ex3. Image thresholding

Given an input image as follows (numbers are pixel intensities)



1) Convert the image into 02 binary images:
    a) 1st output : numbers greater than or equals to 180 are in black
    b) 2nd output: numbers less than 180 are in white
    Hints: Using different thresholds and binarization techniques.
2) Extract each number in the input image as in separated images.

HOMEWORK

Create a video from your webcam by inserting the TDT logo to images streaming from your webcam.