Họ tên: Nguyễn Phú Thành, MSSV: 18110014

Bài thực hành 02 môn Phân tích và xử lý ảnh

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
import cv2
from skimage import feature, filters
from skimage.color.adapt_rgb import adapt_rgb, each_channel, hsv_value
from skimage.exposure import rescale_intensity
from skimage import transform
```

Bài tập 01

Viết thành một function cho biết thông tin của bức ảnh : Ảnh màu hay ảnh xám và giá trị min, max tương ứng với từng kênh màu

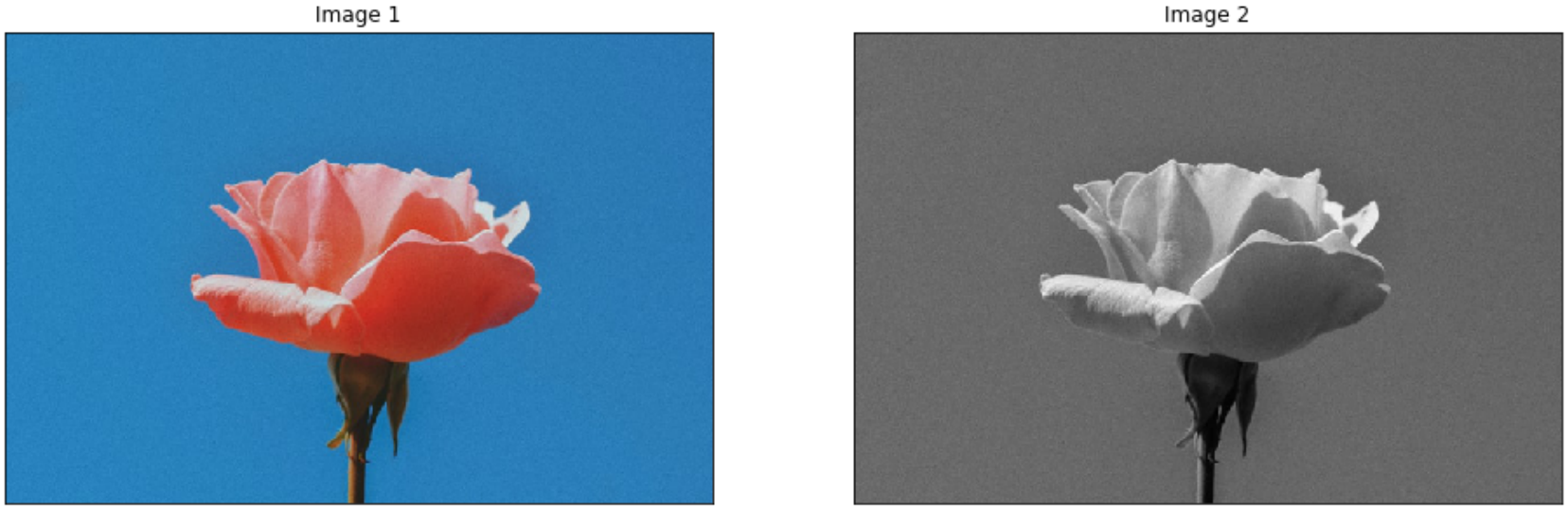
In [2]:
```
class Image:

    def __init__(self, imgName = None, img = None):
        self.image = img if imgName is None else plt.imread(imgName)
        self.shape = self.image.shape
        self.width, self.height = self.shape[:2]
        self.channel = len(self.shape)

    def info(self, verbose = True):
        info_dict = {
            'Width' : self.width,
            'Height' : self.height,
            'Channel' : self.channel
        }
        if verbose is True:
            print(f'>> Width= {self.width}, Height = {self.height}, Channel = {self.channel}')
            if self.channel == 2:
                minVal, maxVal = np.min(self.image), np.max(self.image)
                info_dict['Min intensity'], info_dict['Max intensity'] = minVal, maxVal
                print(f'>> Min intensity = {minVal}, Max intensity = {maxVal}')
            else:
                colors = ['Red', 'Green', 'Blue']
                for channel, color in enumerate(colors):
                    minVal = np.min(self.image[:, :, channel])
                    maxVal = np.max(self.image[:, :, channel])
                    info_dict[f'Channel {color}'] = (minVal, maxVal)
                    print(f'>> {color}: Min intensity = {minVal}, Max intensity = {maxVal}')
        return info_dict

    def convertColor(self, cv2ColorConverter):
        return Image(img = cv2.cvtColor(self.image, cv2ColorConverter))

    def segmentByMask(self, mask):
        mask = Image(img = mask.image.astype(np.uint8))
        result = cv2.bitwise_and(self.image, self.image, mask = mask.image)
        return Image(img = result)

    def rotateImage(self, angle, resize = False):
        return Image(img = transform.rotate(self.image, angle, resize = resize))

    def flipImage(self, how = 'vertical'):
        if how not in ['vertical', 'horizontal', 'both']:
            raise ValueError(' `how` parameter must be either `vertical`, `horizontal` or `both`')
```

```
        if how == 'vertical':
            return Image(img = self.image[::-1, :].copy())
        elif how == 'horizontal':
            return Image(img = self.image[:, ::-1].copy())
        else:
            return self.rotateImageImage(angle = 180)

    def getInverted(self, max_values = [255, 255, 255]):
        result_img = self.image.copy()
        for channel, max_value in enumerate(max_values):
            result_img[:, :, channel] = max_value - result_img[:, :, channel]
        return Image(img = result_img)

    def enhanceContrast(self, v_min = None, v_max = None):
        if v_min is None:
            v_min = np.percentile(self.image, 5)
        if v_max is None:
            v_max = np.percentile(self.image, 95)
        return Image(img = rescale_intensity(self.image, in_range = (v_min, v_max)))

    def adjustGamma(self, gamma = None, gain = None):
        if gamma is None:
            gamma = 1
        if gain is None:
            gain = 1
        return Image(img = gain * ((self.image/255)**gamma))

    def adjustLog(self, gain = None):
        if gain is None:
            gain = 1
        return Image(img = gain * np.log2(1 + self.image/255))
```

In [3]:
```
def imShows(Images, Labels = None, rows = 1, cols = 1):
    imagesArray = list(Images)
    labelsArray = [f"Image {i + 1}" for i in range(len(imagesArray))] if Labels is None else list(Labels)
    figsize = (20, 20) if ((rows == 1) and (cols == 1)) else (cols * 8, rows * 5)
    fig = plt.figure(figsize = figsize)
    for i in range(1, rows * cols + 1):
        ax = fig.add_subplot(rows, cols, i)
        image = imagesArray[i - 1].image
        cmap = plt.cm.gray if (imagesArray[i - 1].channel < 3) else None
        ax.imshow(image, cmap = cmap)
        ax.set(title = labelsArray[i - 1], xticks = [], yticks = [])
    plt.show()

def showThreeImages(IM1, IM2, IM3):
    imShows([IM1, IM2, IM3], rows = 1, cols = 3)
def showTwoImages(IM1, IM2):
    imShows([IM1, IM2], rows = 1, cols = 2)
def showOneImage(IM1):
    imShows([IM1])
def showListImages(listImage, rows, cols):
    imShows(listImage, rows = rows, cols = cols)
```

In [4]:
```
imageColor = Image(imgName = 'rose.jpg')
imageGray = imageColor.convertColor(cv2.COLOR_RGB2GRAY)
```

In [5]:
```
showTwoImages(imageColor, imageGray)
```



In [6]:
```
_ = imageColor.info()
```

```
>> Width= 427, Height = 640, Channel = 3
>> Red: Min intensity = 6, Max intensity = 243
>> Green: Min intensity = 22, Max intensity = 230
>> Blue: Min intensity = 0, Max intensity = 255
```

In [7]:
```
_ = imageGray.info()
```

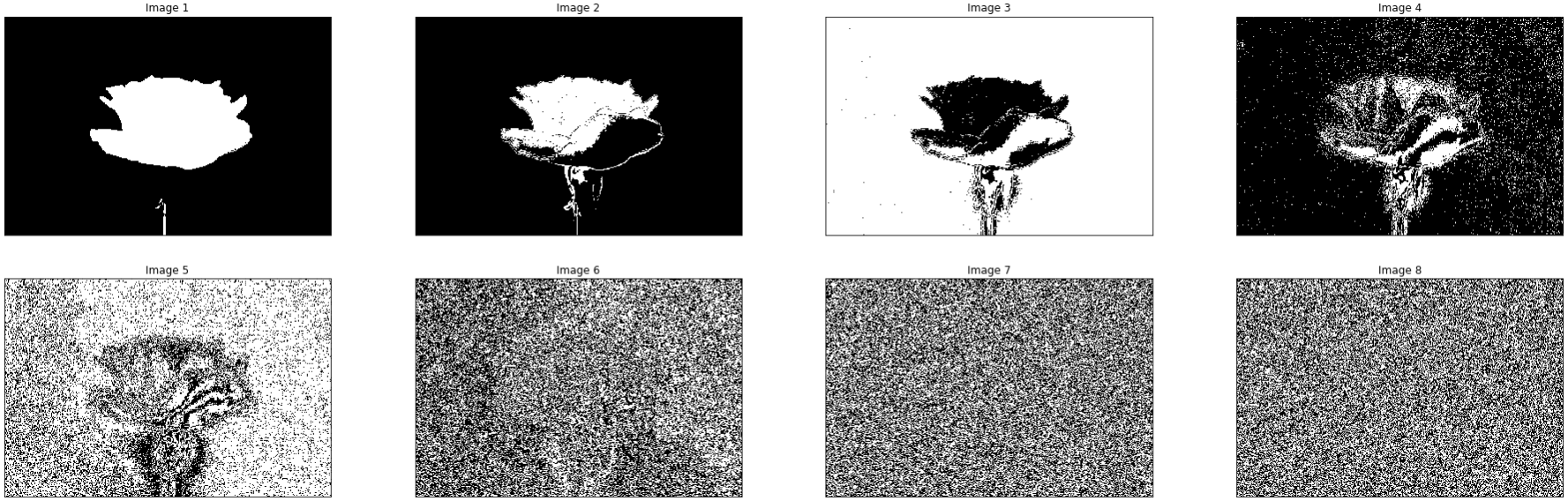
```
>> Width= 427, Height = 640, Channel = 2
>> Min intensity = 25, Max intensity = 230
```

In [8]:
```
def convertToBitArray(img):
    rows, cols = img.shape
    bitArray = []
    for i in range(rows):
        for j in range(cols):
            bitArray.append(np.binary_repr(img[i,j], width = 8))
    return bitArray

def bitPlane(bitImgVal, img1D):
    bitList = [int(i[bitImgVal]) for i in img1D]
    return bitList

def getBitImage(index, image2D):
    ImageIn1D = convertToBitArray(image2D)
    Imagebit = np.array(bitPlane(index, ImageIn1D)).reshape(image2D.shape)
    return Imagebit

def getAllBitImage(image2D):
    image2D_Bit = list()
    for i in range(8):
        image2D_Bit.append(Image(img = getBitImage(i, image2D)))
    return image2D_Bit

def getAllBitColorImage(image, channels):
    bitImages = {}
    for index, channel in enumerate(channels):
        print(f'>> Channel: {channel}')
        bitImage = getAllBitImage(image.image[:, :, index])
        bitImages[channel] = bitImage
        showListImages(bitImage, rows = 2, cols = 4)
    return bitImages
```

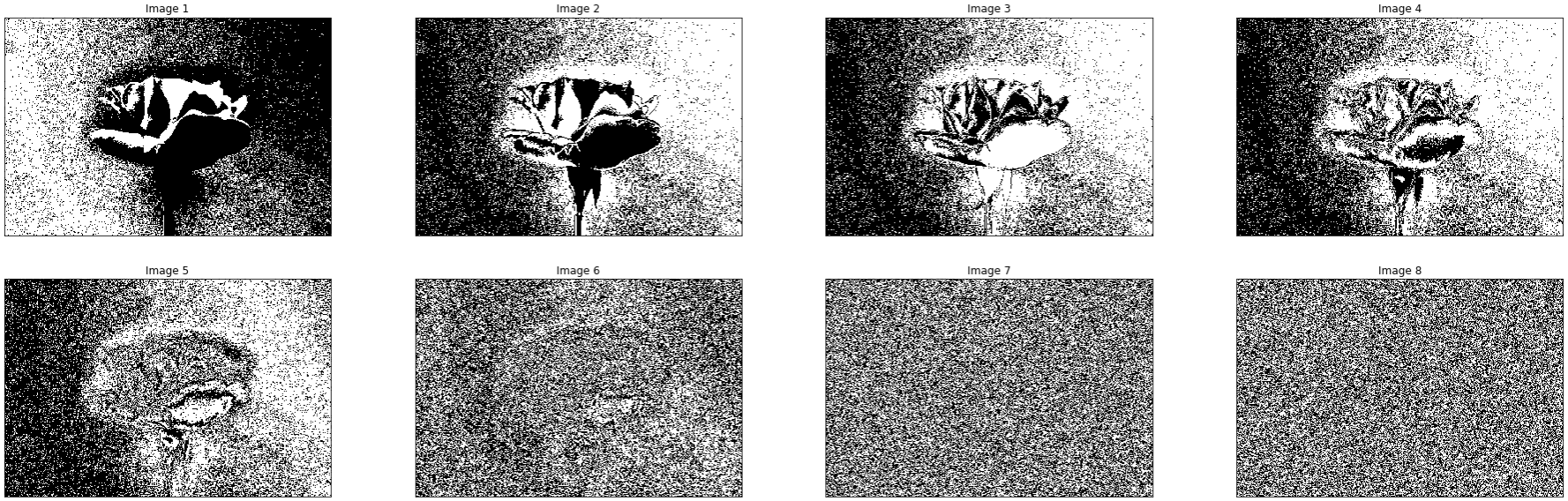
Hiển thị 24 bức ảnh theo bit của từng kênh màu R, G, B

In [9]:
```
channels = ['R', 'G', 'B']
RGB_bitImage = getAllBitColorImage(imageColor, channels)
```

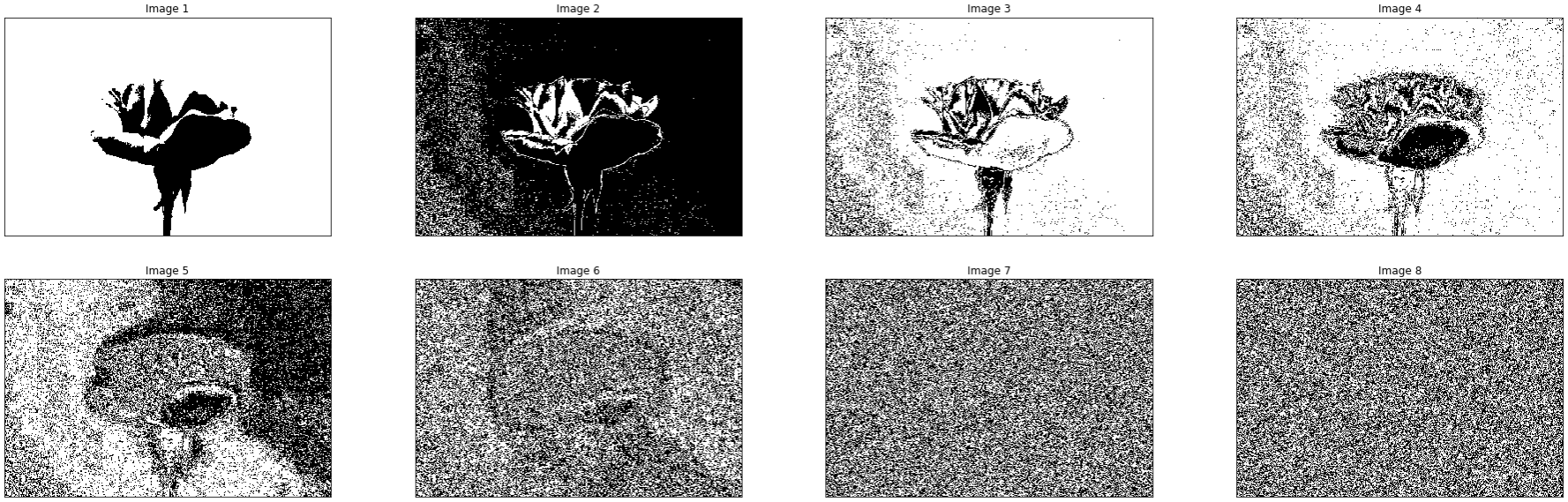
>> Channel: R



>> Channel: G



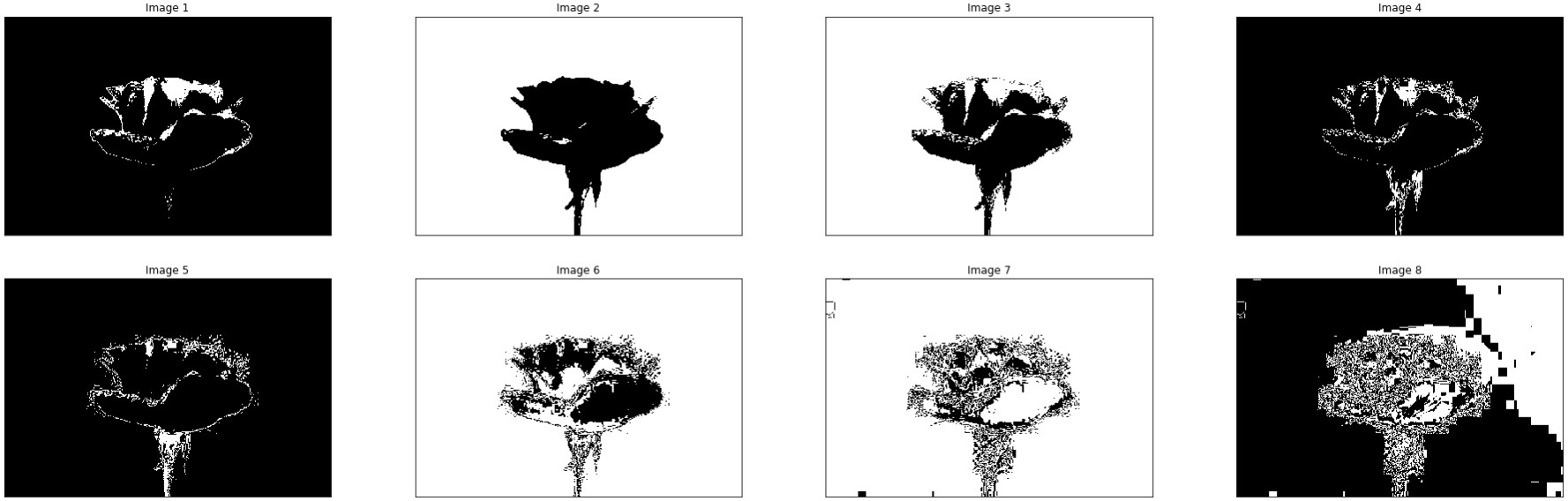
>> Channel: B



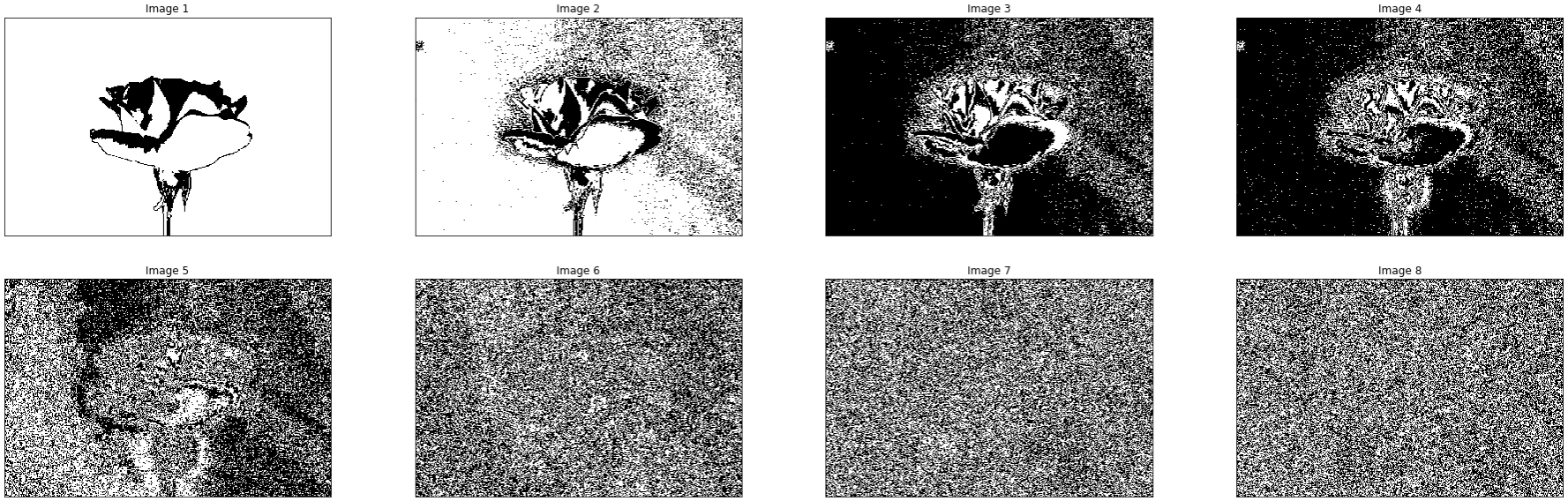
Hiển thị 24 bức ảnh theo bit của từng kênh màu H,S,V

In [10]:
```
imageHsv = imageColor.convertColor(cv2.COLOR_RGB2HSV)
channels = ['H', 'S', 'V']
HSV_bitImage = getAllBitColorImage(imageHsv, channels)
```

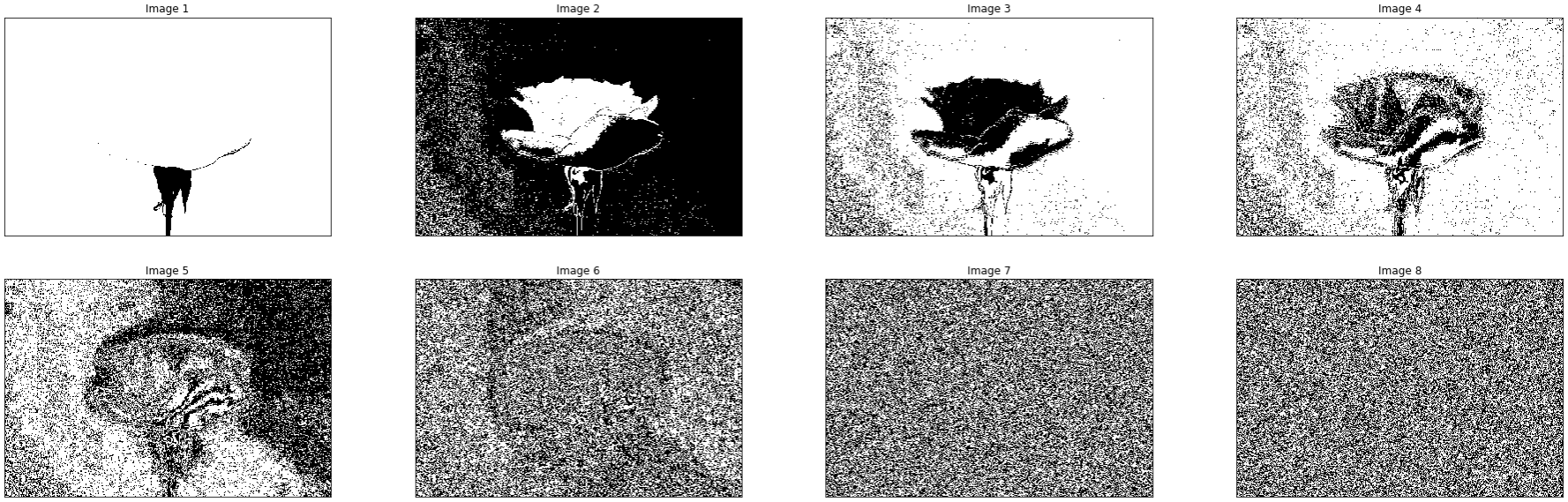
>> Channel: H



>> Channel: S

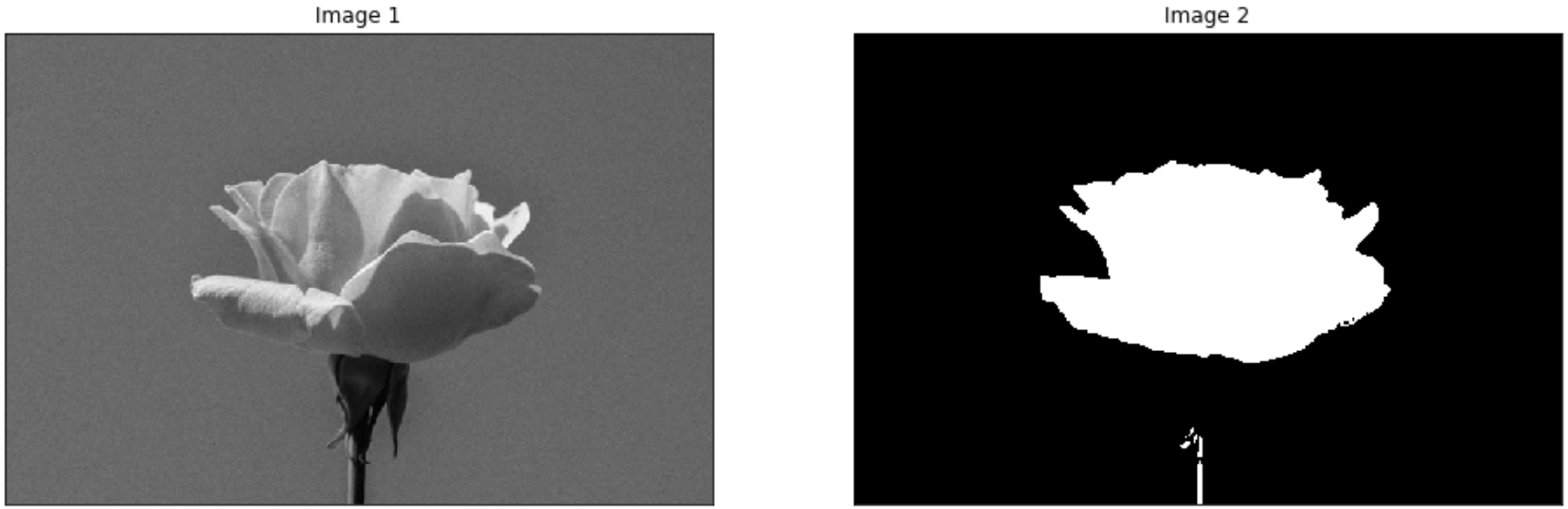


>> Channel: V



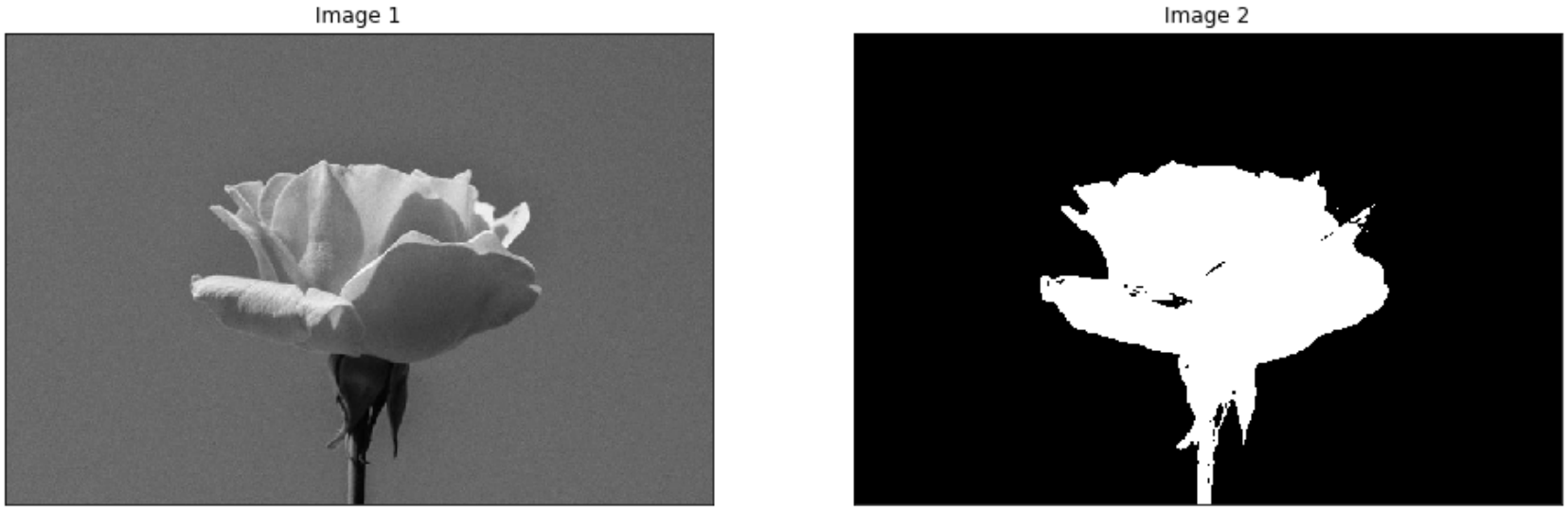
Cho biết trong 24 ảnh bit theo kênh R, G, B có ảnh nào có thể dùng để trích xuất đối tượng hay không. Nếu có thì hiển thị ảnh bit đó ra

In [11]:
```
bitMask01 = RGB_bitImage['R'][0]
showTwoImages(imageGray, bitMask01)
```

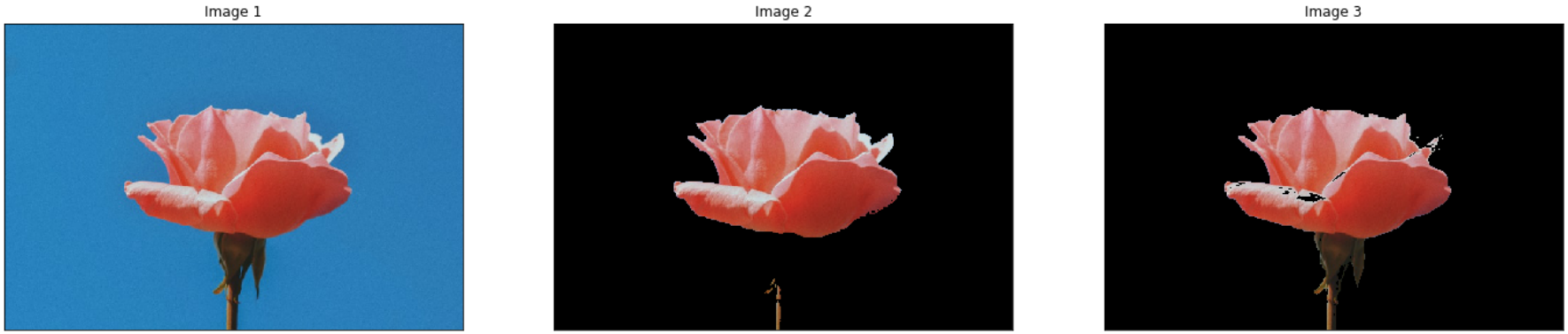


Cho biết trong 24 ảnh bit theo kênh H, S, V có ảnh nào có thể dùng để trích xuất đối tượng hay không. Nếu có thì hiển thị ảnh bit đó ra

In [12]:
```
bitMask02 = Image(img = 1 - HSV_bitImage['H'][1].image)
showTwoImages(imageGray, bitMask02)
```



In [13]:
```
mask01 = imageColor.segmentByMask(bitMask01)
mask02 = imageColor.segmentByMask(bitMask02)
showThreeImages(imageColor, mask01, mask02)
```



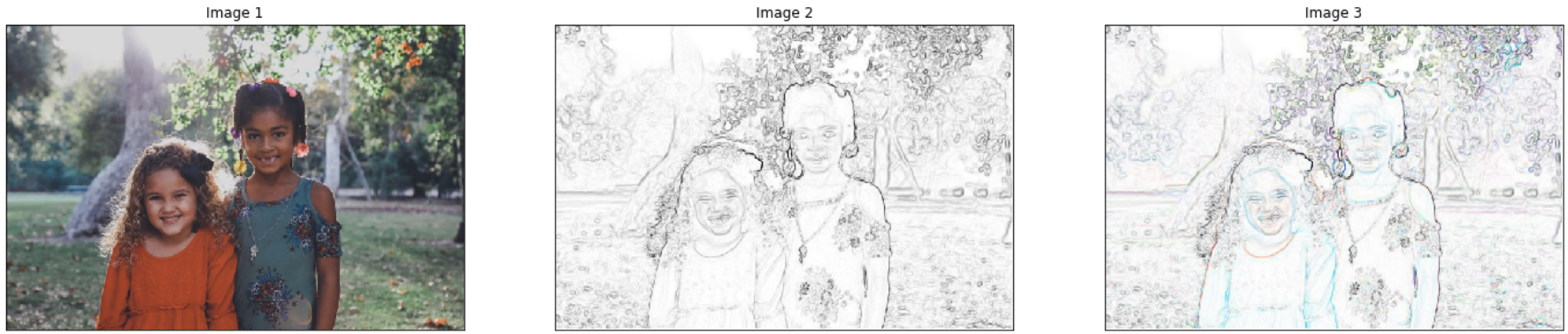
Bài tập 02

In [14]:
```
imageColor = Image('faces.jpg')
imageGray = imageColor.convertColor(cv2.COLOR_RGB2GRAY)
showTwoImages(imageColor, imageGray)
```

Viết 1 function input ảnh màu và xuất ảnh chân dung xám và chân dùng màu của ảnh đó

In [15]:
```
@adapt_rgb(each_channel)
def sobel_each(image):
    return filters.sobel(image)

def getPortrait(imgColor):
    imgGray = imgColor.convertColor(cv2.COLOR_RGB2GRAY)
    edges_gray = filters.sobel(imgGray.image)
    edges_color = sobel_each(imgColor.image)
    return Image(img = (1 - edges_gray)), Image(img = rescale_intensity(1 - edges_color))
```

In [16]:
```
portraitGray, portraitColor = getPortrait(imageColor)
showThreeImages(imageColor, portraitGray, portraitColor)
```

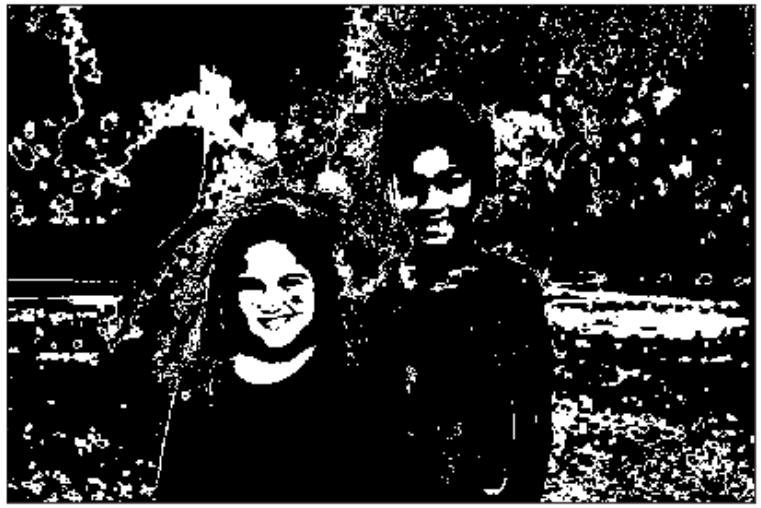


Tìm chọn range màu lower và upper để trích xuất được các khuôn mặt trong ảnh màu

In [17]:
```
lower = np.array([127,79,72])
upper = np.array([183, 158, 164])
mask_face = Image(img = cv2.inRange(imageColor.image, lower, upper))
showTwoImages(imageColor, mask_face)
```

Image 1

Image 2



In [18]:
```
mask_rgb = imageColor.segmentByMask(mask_face)
showTwoImages(imageColor, mask_rgb)
```

Bài tập 03

In [19]:
```
imageColor = Image('flower.jpg')
imageGray = imageColor.convertColor(cv2.COLOR_RGB2GRAY)
showTwoImages(imageColor, imageGray)
```

Xuất ảnh xoay các góc 30,45,60 độ

In [20]:
```
angles = [30, 45, 60]
for angle in angles:
    rotatedImage = imageColor.rotateImage(angle)
    showTwoImages(imageColor, rotatedImage)
```

Image 1

Image 2

Image 1

Image 2

Xuất ảnh đối xứng qua trục hoành và trục tung đi qua tâm ảnh

In [21]:
```
flips = ['horizontal', 'vertical']
for flip in flips:
    flippedImage = imageColor.flipImage(how = flip)
    showTwoImages(imageColor, flippedImage)
```

Xuất ảnh inversion

In [22]:
```
invertedImage = imageColor.getInverted()
showTwoImages(imageColor, invertedImage)
```

Xuất ảnh enhance contrast

In [23]:
```
contrastImg = imageColor.enhanceContrast()
showTwoImages(imageColor, contrastImg)
```

Xuất ảnh gamma và log enhancement

In [24]:
```
gammaAdjusted = imageColor.adjustGamma(gamma = 1.2, gain = 0.9)
showTwoImages(imageColor, gammaAdjusted)
```

In [25]:
```
logAdjust = imageColor.adjustLog(gain = 1)
showTwoImages(imageColor, logAdjust)
```