DIP REPORT

Made By: -

Sparsh Dalal 21ucs208

Namit Kapoor 21ucs135

Assignment 2

• Given the attached image, write a program to implement a scheme that will extract only the elephant from the image. The output image will **only have the elephant (in colour) and a white background**.

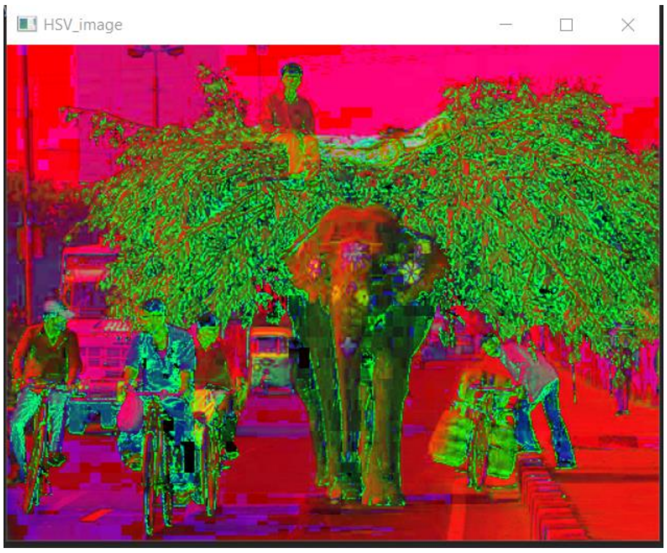
The language used – Python.

Libraries used – OpenCV, Pandas.

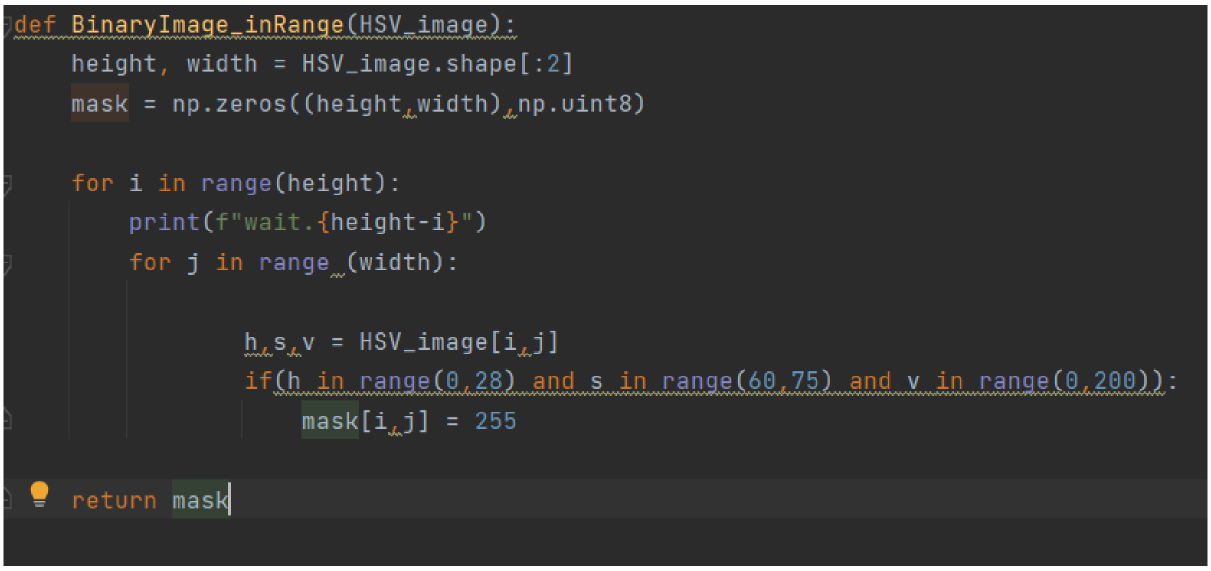
Outline of Steps performed: -

1. Conversion of the image from RGB to HSV
2. Finding the appropriate HSV range of the elephant via mouse handler.
3. Creation of a binary image, with the pixels in range to be white, the rest black.
4. Applied dilation to the obtained image.
5. Found out the largest connected component of the image (the elephant) and applied various algorithms for hole filing of the component.
6. Took the 'AND' operator of the obtained mask with the original image, to extract Elephant from the image.

1. To convert the image into an HSV image, we have an inbuilt function that returns the desired image.

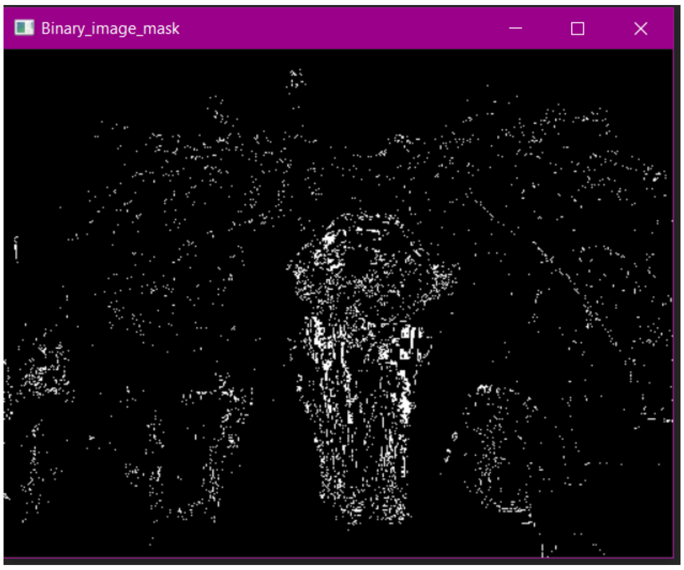


2. We used the mouse handler code to find various HSV Values of the elephant. We then used this data to finalize a range of values in which the elephant lies.
   The range is as follows –
   Hue – 0 to 28
   Saturation - 60 to 75
   Intensity – 0 to 200
3. All the pixels in this range were converted to white, while the rest were converted to black.

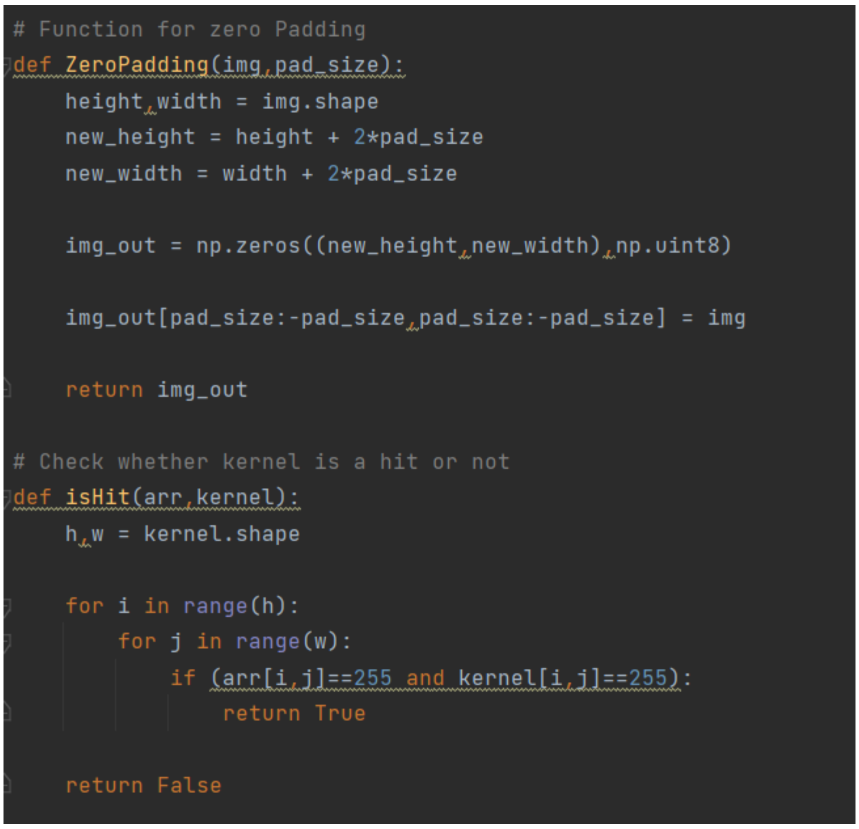


```
def BinaryImage_inRange(HSV_image):
    height, width = HSV_image.shape[:2]
    mask = np.zeros((height,width),np.uint8)

    for i in range(height):
        print(f"wait.{height-i}")
        for j in range (width):

                h,s,v = HSV_image[i,j]
                if(h in range(0,28) and s in range(60,75) and v in range(0,200)):
                    mask[i,j] = 255

    return mask
```

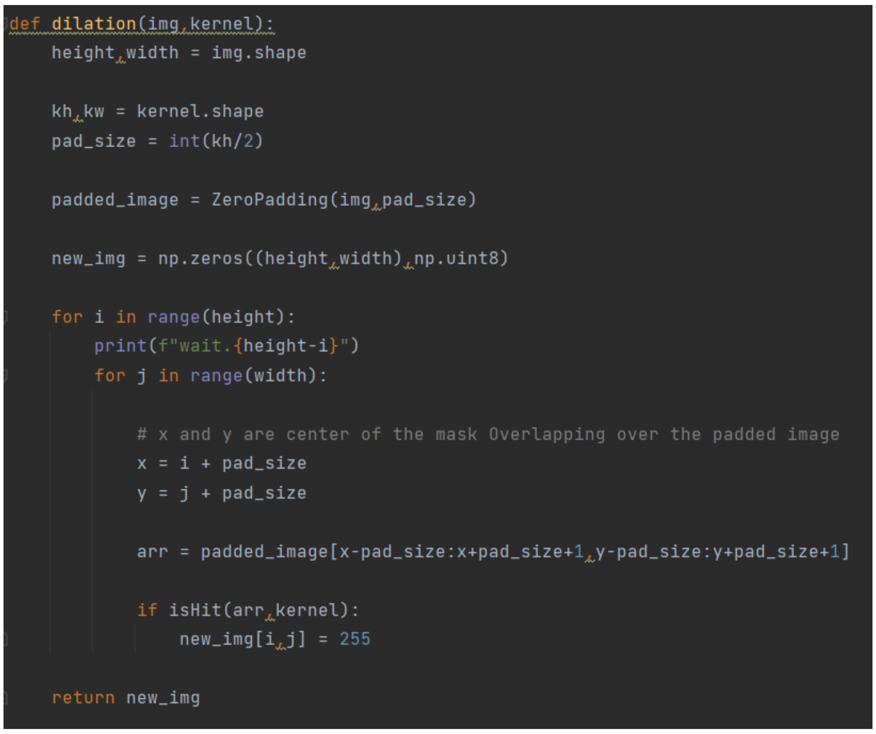
This gave us a binary image, on which we could perform further functions.



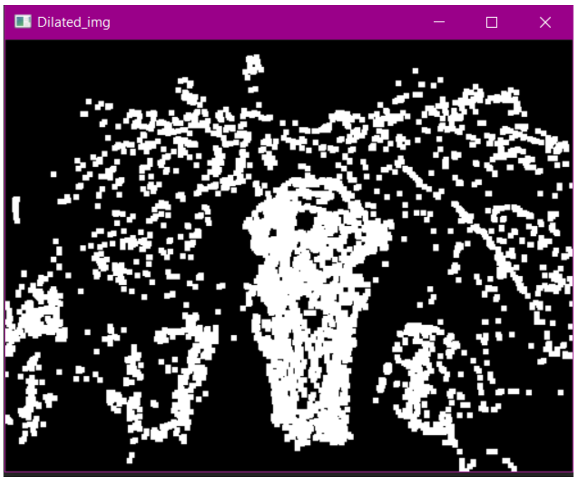
4. As the white dots in the picture were very apart to perform maximum connected component, we performed dilation on the binary image.
   This resulted in a more comprehensive binary image.



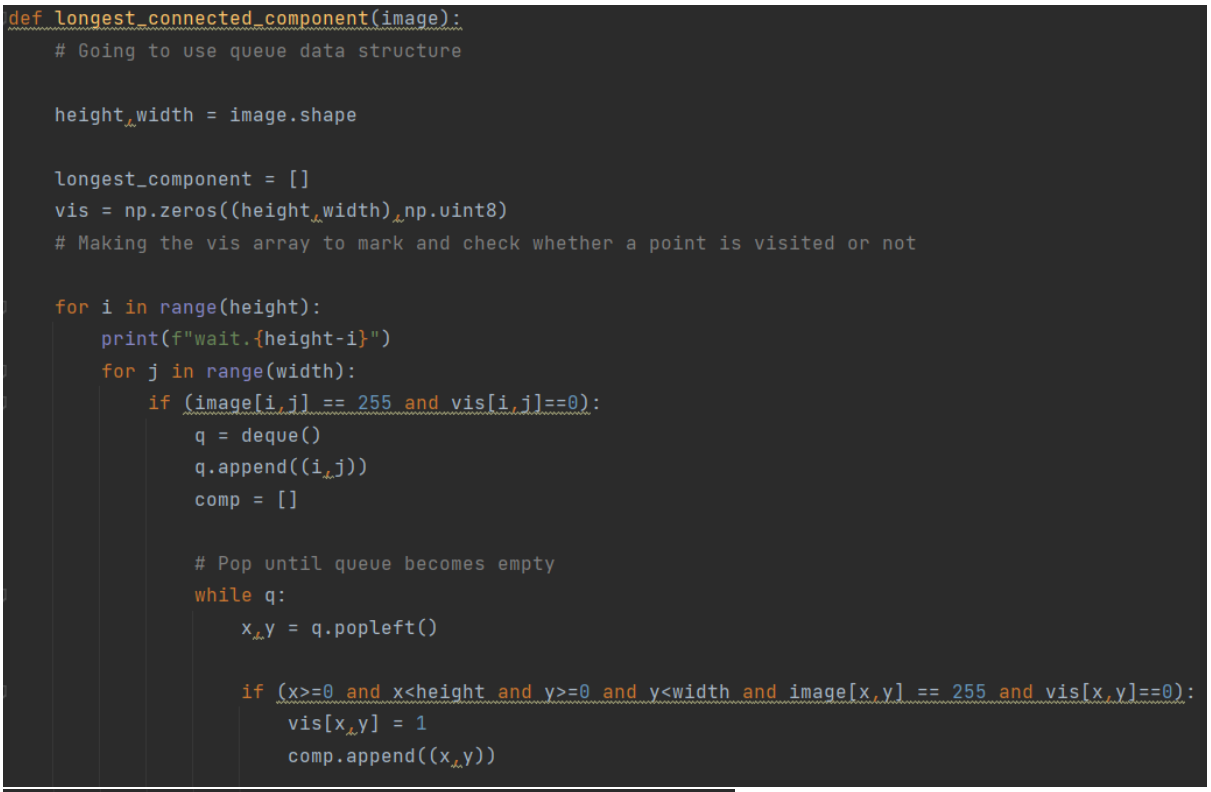
```python
# Function for zero Padding
def ZeroPadding(img,pad_size):
    height,width = img.shape
    new_height = height + 2*pad_size
    new_width = width + 2*pad_size

    img_out = np.zeros((new_height,new_width),np.uint8)

    img_out[pad_size:-pad_size,pad_size:-pad_size] = img

    return img_out

# Check whether kernel is a hit or not
def isHit(arr,kernel):
    h,w = kernel.shape

    for i in range(h):
        for j in range(w):
            if (arr[i,j]==255 and kernel[i,j]==255):
                return True

    return False
```



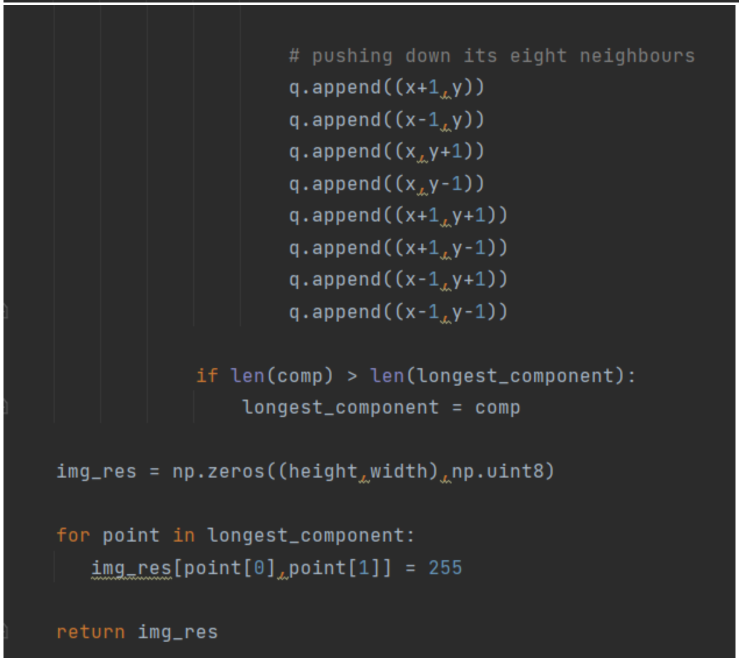
```python
def dilation(img,kernel):
    height,width = img.shape

    kh,kw = kernel.shape
    pad_size = int(kh/2)

    padded_image = ZeroPadding(img,pad_size)

    new_img = np.zeros((height,width),np.uint8)

    for i in range(height):
        print(f"wait.{height-i}")
        for j in range(width):

            # x and y are center of the mask Overlapping over the padded image
            x = i + pad_size
            y = j + pad_size

            arr = padded_image[x-pad_size:x+pad_size+1,y-pad_size:y+pad_size+1]

            if isHit(arr,kernel):
                new_img[i,j] = 255

    return new_img
```



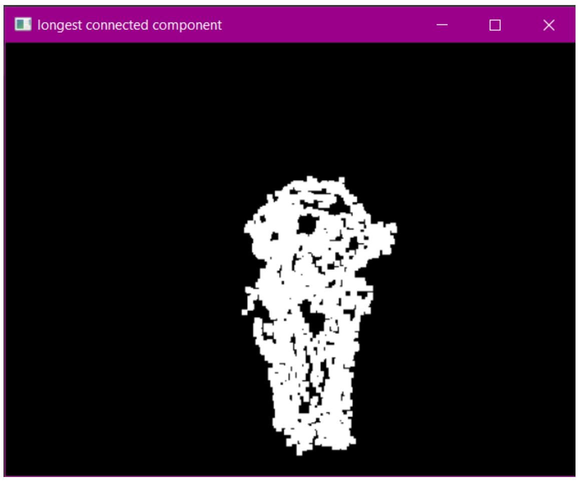
5. We could figure out in the previous image; the elephant is the largest connected component. As a result, we performed the largest connected component function and found the boundary of our elephant.



```python
def longest_connected_component(image):
    # Going to use queue data structure

    height,width = image.shape

    longest_component = []
    vis = np.zeros((height,width),np.uint8)
    # Making the vis array to mark and check whether a point is visited or not

    for i in range(height):
        print(f"wait.{height-i}")
        for j in range(width):
            if (image[i,j] == 255 and vis[i,j]==0):
                q = deque()
                q.append((i,j))
                comp = []

                # Pop until queue becomes empty
                while q:
                    x,y = q.popleft()

                    if (x>=0 and x<height and y>=0 and y<width and image[x,y] == 255 and vis[x,y]==0):
                        vis[x,y] = 1
                        comp.append((x,y))
```



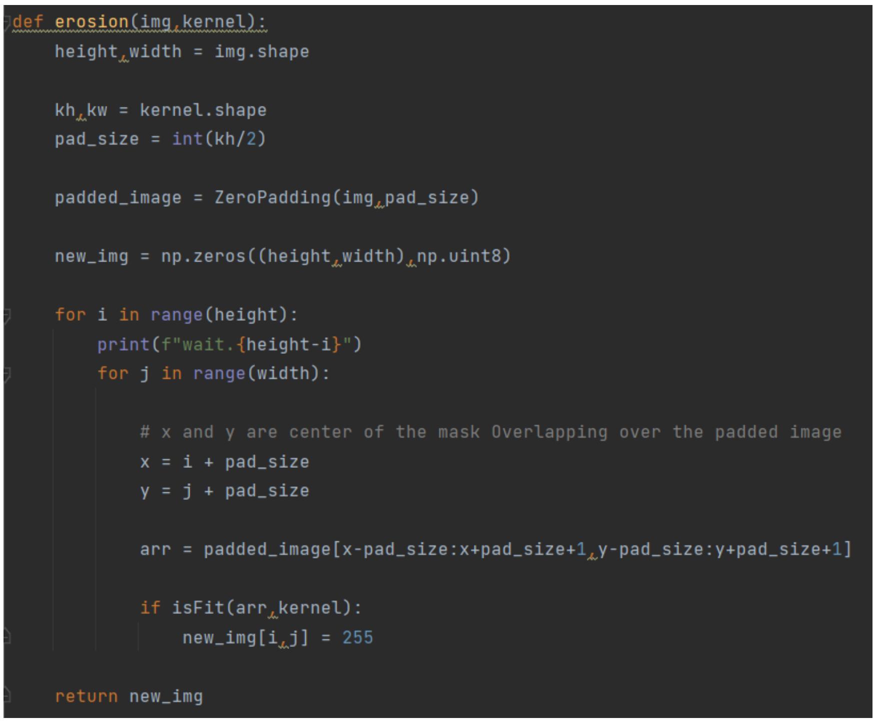
```python

                        # pushing down its eight neighbours
                        q.append((x+1,y))
                        q.append((x-1,y))
                        q.append((x,y+1))
                        q.append((x,y-1))
                        q.append((x+1,y+1))
                        q.append((x+1,y-1))
                        q.append((x-1,y+1))
                        q.append((x-1,y-1))

                if len(comp) > len(longest_component):
                    longest_component = comp

    img_res = np.zeros((height,width),np.uint8)

    for point in longest_component:
        img_res[point[0],point[1]] = 255

    return img_res
```



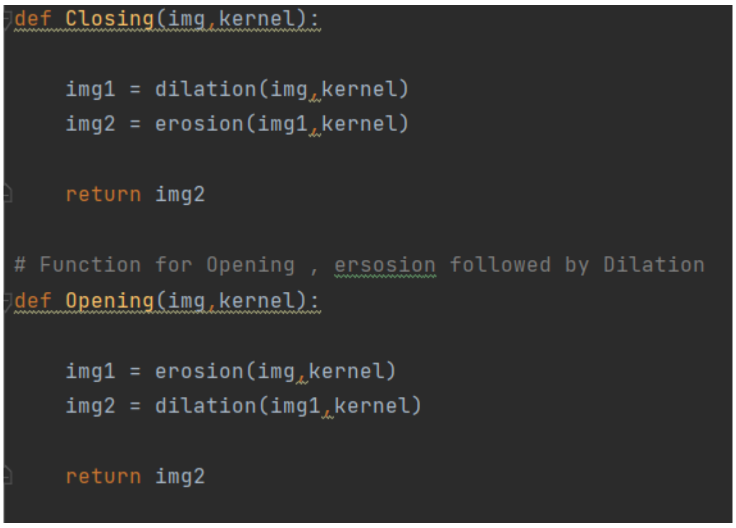
Although as noticed in the image, we have quite a few holes.
We had **3 different methods** to solve this problem.

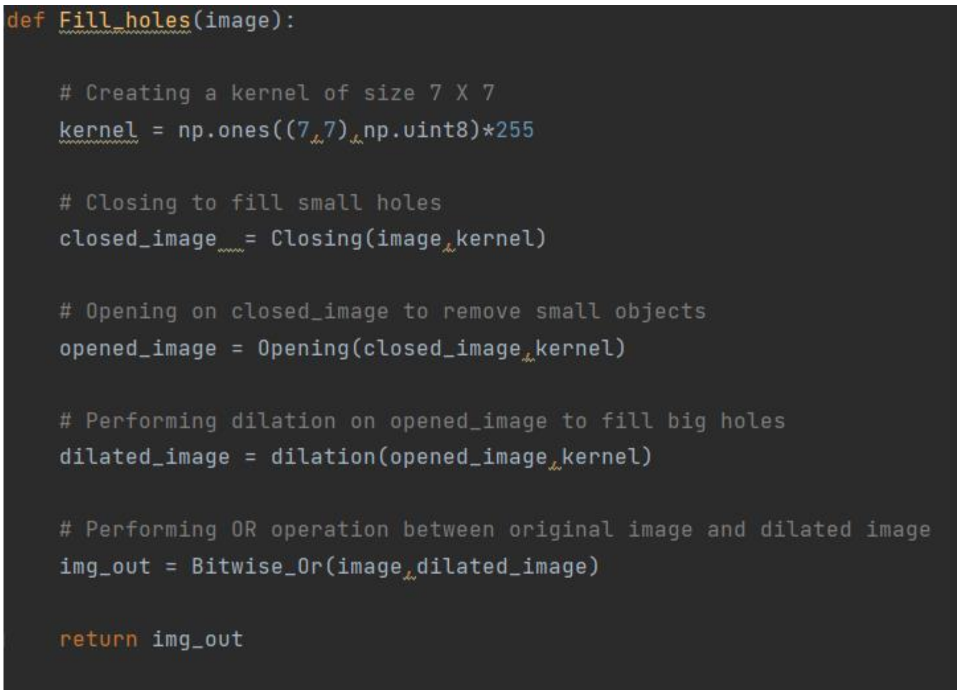
Method 1:

We performed a series of erosion and dilation functions.
We got rid of the holes, but it also resulted in an extra region being included around the elephant.
This method also resulted in more time complexity of the code.
As a result, this method was discarded by us.



```
def erosion(img,kernel):
    height,width = img.shape

    kh,kw = kernel.shape
    pad_size = int(kh/2)

    padded_image = ZeroPadding(img,pad_size)

    new_img = np.zeros((height,width),np.uint8)

    for i in range(height):
        print(f"wait.{height-i}")
        for j in range(width):

            # x and y are center of the mask Overlapping over the padded image
            x = i + pad_size
            y = j + pad_size

            arr = padded_image[x-pad_size:x+pad_size+1,y-pad_size:y+pad_size+1]

            if isFit(arr,kernel):
                new_img[i,j] = 255

    return new_img
```


```
def Closing(img,kernel):

    img1 = dilation(img,kernel)
    img2 = erosion(img1,kernel)

    return img2

# Function for Opening , ersosion followed by Dilation
def Opening(img,kernel):

    img1 = erosion(img,kernel)
    img2 = dilation(img1,kernel)

    return img2
```

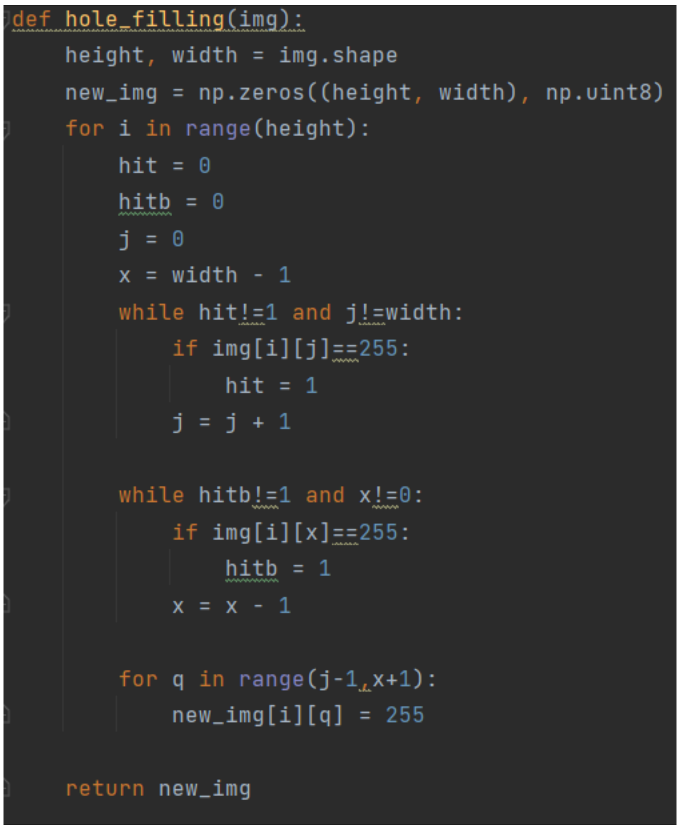

```
def Fill_holes(image):

    # Creating a kernel of size 7 X 7
    kernel = np.ones((7,7),np.uint8)*255

    # Closing to fill small holes
    closed_image  = Closing(image,kernel)

    # Opening on closed_image to remove small objects
    opened_image = Opening(closed_image,kernel)

    # Performing dilation on opened_image to fill big holes
    dilated_image = dilation(opened_image,kernel)

    # Performing OR operation between original image and dilated image
    img_out = Bitwise_Or(image,dilated_image)

    return img_out
```

Method 2:

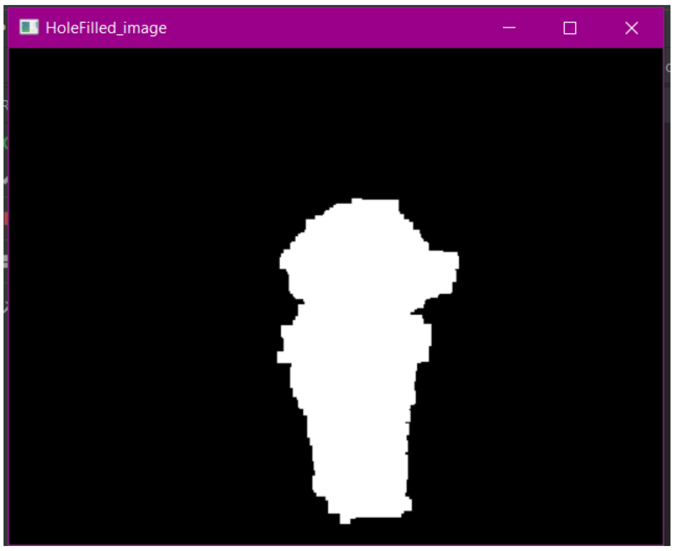
Instead of performing a maximum connected component for the white pixels, we can perform a maximum connected component for the background.
This will first give us a proper elephant, with holes filled, but will also result in other regions being present.
In order to remove other regions, we shall perform the longest connected component again to extract the largest white connected component (the elephant).

Method 3:

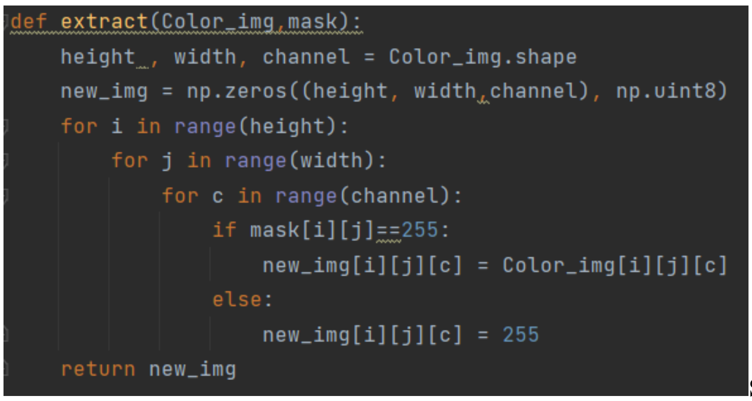
We found a simple method for hole filing; we iterated through the image vertically and found out 1st and last white pixel in each row. Using this, we made each of the pixels in between white.
As the elephant was the only white component in the image, we did not receive any other noise, and we got the desired image on the first try, and in the least time complexity. We performed an additional dilation function for smoothening the result, but it was optional.



```python
def hole_filling(img):
    height, width = img.shape
    new_img = np.zeros((height, width), np.uint8)
    for i in range(height):
        hit = 0
        hitb = 0
        j = 0
        x = width - 1
        while hit!=1 and j!=width:
            if img[i][j]==255:
                hit = 1
            j = j + 1

        while hitb!=1 and x!=0:
            if img[i][x]==255:
                hitb = 1
            x = x - 1

        for q in range(j-1,x+1):
            new_img[i][q] = 255

    return new_img
```

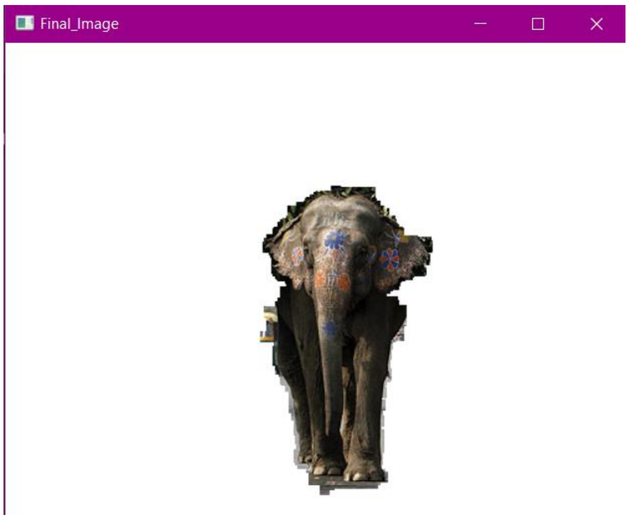


6. We took the bitwise AND of the binary and original images to get the desired output.



```
def extract(Color_img,mask):
    height_, width, channel = Color_img.shape
    new_img = np.zeros((height, width,channel), np.uint8)
    for i in range(height):
        for j in range(width):
            for c in range(channel):
                if mask[i][j]==255:
                    new_img[i][j][c] = Color_img[i][j][c]
                else:
                    new_img[i][j][c] = 255
    return new_img
```

s



Challenge Faced:

1. The most challenging part of the assignment was to fill the holes of the elephant after obtaining the largest connected components.