JSS Science & Technological University

Professor: Shruthi N M

DIGITAL IMAGE PROCESSING LAB

Y PRAHASITH

Program to enhance image using image arithmetic and logical operations

```
from PIL import Image

def arithmetic_addition(img1, img2):

    for i in range(img1.width):
        for j in range(img1.height):
            px1 = img1.getpixel((i, j))
            if i < img2.width and j < img2.height:
                px2 = img2.getpixel((i, j))
                tmp = []
                for k in range(len(px1)):
                    tmp.append(min(255, px1[k]+px2[k]))
                px1 = tuple(tmp)
            img1.putpixel((i, j), px1)

    # img1.show()
    img1.save('addition.png')

def arithmetic_subtraction(img1, img2):

    for i in range(img1.width):
        for j in range(img1.height):
            px1 = img1.getpixel((i, j))
            if i < img2.width and j < img2.height:
```

```
                px2 = img2.getpixel((i, j))
                tmp = []
                for k in range(len(px1)):
                    tmp.append(max(0, px1[k]-px2[k]))
                px1 = tuple(tmp)
            img1.putpixel((i, j), px1)

    # img1.show()
    img1.save('subtraction.png')

def arithmetic_multiplication(img1, img2):

    for i in range(img1.width):
        for j in range(img1.height):
            px1 = img1.getpixel((i, j))
            if i < img2.width and j < img2.height:
                px2 = img2.getpixel((i, j))
                tmp = []
                for k in range(len(px1)):
                    tmp.append(min(255, px1[k]*px2[k]))
                px1 = tuple(tmp)
            img1.putpixel((i, j), px1)

    # img1.show()
    img1.save('multiplication.png')

def arithmetic_division(img1, img2):

    for i in range(img1.width):
        for j in range(img1.height):
            px1 = img1.getpixel((i, j))
            if i < img2.width and j < img2.height:
                px2 = img2.getpixel((i, j))
                tmp = []
                for k in range(len(px1)):
                    if px2[k] > 0:
                        tmp.append(max(0, px1[k]//px2[k]))
                    else:
                        tmp.append(255)
                px1 = tuple(tmp)
            img1.putpixel((i, j), px1)

    # img1.show()
    img1.save('division.png')
```

```
def logical_and(img1, img2):

    for i in range(img1.width):
        for j in range(img1.height):
            px1 = img1.getpixel((i, j))
            if i < img2.width and j < img2.height:
                px2 = img2.getpixel((i, j))
                tmp = []
                for k in range(len(px1)):
                    tmp.append(px1[k]&px2[k])
                px1 = tuple(tmp)
            img1.putpixel((i, j), px1)

    # img1.show()
    img1.save('and.png')

def logical_or(img1, img2):

    for i in range(img1.width):
        for j in range(img1.height):
            px1 = img1.getpixel((i, j))
            if i < img2.width and j < img2.height:
                px2 = img2.getpixel((i, j))
                tmp = []
                for k in range(len(px1)):
                    tmp.append(px1[k]|px2[k])
                px1 = tuple(tmp)
            img1.putpixel((i, j), px1)

    # img1.show()
    img1.save('or.png')

def logical_xor(img1, img2):

    for i in range(img1.width):
        for j in range(img1.height):
            px1 = img1.getpixel((i, j))
            if i < img2.width and j < img2.height:
                px2 = img2.getpixel((i, j))
                tmp = []
                for k in range(len(px1)):
                    tmp.append(px1[k]^px2[k])
                px1 = tuple(tmp)
```

```
            img1.putpixel((i, j), px1)

    # img1.show()
    img1.save('xor.png')

if __name__ == '__main__':

    print('1. Addition\n2. Subtraction\n3. Multiplication\n4. Division\n5.
And\n6. Or\n7. Xor')
    choice = int(input('Pick required operation: '))

    img_name1 = input('Image 1 filename: ')
    img1 = Image.open(img_name1).convert('RGB')

    img_name2 = input('Image 2 filename: ')
    img2 = Image.open(img_name2).convert('RGB')

    if choice == 1:
        arithmetic_addition(img1, img2)

    elif choice == 2:
        arithmetic_subtraction(img1, img2)

    elif choice == 3:
        arithmetic_multiplication(img1, img2)

    elif choice == 4:
        arithmetic_division(img1, img2)

    elif choice == 5:
        logical_and(img1, img2)

    elif choice == 6:
        logical_or(img1, img2)

    elif choice == 7:
        logical_xor(img1, img2)

    else:
        print('Invalid choice!')
```

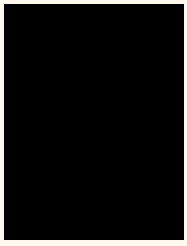
Input Image

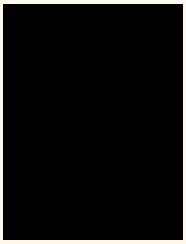
We need to give 2 input images of the same dimensions, but now for the sake of demonstration I am considering two same pictures as the inputs.

Input 1  Input 2 

Output Images

Addition  And  Division 

Multiplication  Or  Subtraction 

XOR 

Program for an image enhancement using pixel operations

```
from PIL import Image
import math

def linear_indentity(img):

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            img.putpixel((i, j), px)

    # img.show()
    img.save('identity.png')

def linear_negative(img):

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            img.putpixel((i, j), (px[1]-px[0], px[1]))

    # img.show()
    img.save('negative.png')

def logarithmic(img, c=31.875):

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            img.putpixel((i, j), (int(c * math.log(px[0]+1, 2)), px[1]))

    # img.show()
    img.save('logarithmic.png')

def power(img, gamma, c=31.875):

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            img.putpixel((i, j), (int(c * (px[0] ** (1/gamma))), px[1]))
```

```
    # img.show()
    img.save('power.png')

def piecewise_contrast_stretching(img, a, b, l, m, n, v, w):

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            if px[0] < a:
                img.putpixel((i, j), (int(l*px[0]), px[1]))
            elif a <= px[0] <= b:
                img.putpixel((i, j), (int(m*(px[0]-a)+v), px[1]))
            else:
                img.putpixel((i, j), (int(n*(px[0]-b)+w), px[1]))

    # img.show()
    img.save('contrast_stretching.png')

def piecewise_clipping(img, a, b):

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            if px[0] < a:
                img.putpixel((i, j), (0, px[1]))
            elif a <= px[0] <= b:
                img.putpixel((i, j), (int(((px[1]/(b-a))*px[0])), px[1]))
            else:
                img.putpixel((i, j), (px[1], px[1]))

    # img.show()
    img.save('clipping.png')

def piecewise_thresholding(img, t):

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            if px[0] >= t:
                img.putpixel((i, j), px)
            else:
                img.putpixel((i, j), (0, px[1]))
```

```
    # img.show()
    img.save('thresholding.png')

if __name__ == '__main__':

    img_name = input('Image filename: ')
    img = Image.open(img_name).convert('LA')

    print('1. Identity\n2. Negative\n3. Logarithmic\n4. Power\n5. Contrast
Stretching\n6. Clipping\n7. Thresholding')
    choice = int(input('Pick required operation: '))

    if choice == 1:
        linear_indentity(img)

    elif choice == 2:
        linear_negative(img)

    elif choice == 3:
        c = float(input('c = '))
        logarithmic(img, c)

    elif choice == 4:
        c = float(input('c = '))
        gamma = float(input('Gamma = '))
        power(img, gamma, c)

    elif choice == 5:
        a, b = map(int, input('Input range = ').split())
        l, m, n = map(int, input('3 slopes = ').split())
        v, w = map(int, input('Output range = ').split())
        piecewise_contrast_stretching(img, a, b, l, m, n, v, w)

    elif choice == 6:
        a, b = map(int, input('Range = ').split())
        piecewise_clipping(img, a, b)

    elif choice == 7:
        threshold = int(input('Threshold = '))
        piecewise_thresholding(img, threshold)

    else:
        print('Invalid choice!')
```

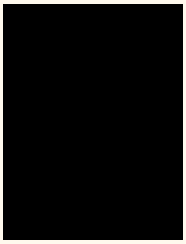
Input Image

We have considered the same image for all operations . For the processes we have consider, c = 31.875 , Gamma = 0.85 , input range = 0 - 255 , output range = 0 - 255 , 3 slopes as {0,-1,2} and threshold = 8

Input 

Output Images

Clipping 

Contrast Stretching 

Identity 

Logarithmic 

Negative 

Power

Thresholding 

Program for gray level slicing with and without background

```
from PIL import Image

def gray_level_slicing_with_background(img, a, b):

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            if a <= px[0] <= b:
                img.putpixel((i, j), (px[1], px[1]))

    #img.show()
    img.save('gray_level_slicing_with_background.png')

def gray_level_slicing_without_background(img, a, b):

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            if a <= px[0] <= b:
                img.putpixel((i, j), (px[1], px[1]))
            else:
                img.putpixel((i, j), (0, px[1]))

    #img.show()
    img.save('gray_level_slicing_without_background.png')

if __name__ == '__main__':
    a = int(input('Lower value: '))
    b = int(input('Upper value: '))

    img_name = input('Image filename: ')
    img = Image.open(img_name).convert('LA')

    print('1. Gray level slicing with background\n2. Gray level slicing
without background')

    choice = int(input('Pick required operation: '))

    if choice == 1:
        gray_level_slicing_with_background(img, a, b)
```

```
    elif choice == 2:
        gray_level_slicing_without_background(img, a, b)

    else:
        print('Invalid choice!')
```

Input Image

We have considered the same image for all operations . For the processes we have consider, Lower value = 80 and Upper value = 202.

Input 

Output Images

Gray Level Slicing with Background 

Gray Level Slicing without Background 

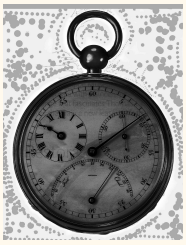
Program for image enhancement using histogram equalization

```
from PIL import Image

def histogram_equalization(img):

    pixels = img.width * img.height
    grey_levels = img.getpixel((0, 0))[1] + 1

    arr = [0] * grey_levels

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            arr[px[0]] += 1

    for i in range(grey_levels):
        arr[i] = arr[i] / pixels

    for i in range(1, grey_levels):
        arr[i] += arr[i-1]

    for i in range(grey_levels):
        arr[i] *= (grey_levels - 1)

    for i in range(grey_levels):
        arr[i] = round(arr[i])

    # print(arr)

    for i in range(img.width):
        for j in range(img.height):
            px = img.getpixel((i, j))
            img.putpixel((i, j), (arr[px[0]], px[1]))

    # img.show()
    img.save('histogram_equalization.png')

if __name__ == '__main__':

    img_name = input('Image filename: ')
```

```
    img = Image.open(img_name).convert('LA')

    histogram_equalization(img)
```

Input Image

Output Images



Program to filter an image using averaging low pass filter in spatial domain and median filter

```
from PIL import Image

def low_pass_mean_filter(img):

    mean_img = Image.new('L', (img.width, img.height))

    for i in range(img.width):
        for j in range(img.height):
            total = 0
            n = 0
            for m in [-1, 0, 1]:
                for n in [-1, 0, 1]:
```

```
                    if 0 <= i+m < img.width and 0 <= j+n < img.height:
                        total += img.getpixel((i, j))
                        n += 1
            mean_img.putpixel((i, j), round(total/n))

    # mean_img.show()
    mean_img.save('mean.png')

def low_pass_median_filter(img):

    median_img = Image.new('L', (img.width, img.height))

    for i in range(img.width):
        for j in range(img.height):
            arr = []
            n = 0
            for m in [-1, 0, 1]:
                for n in [-1, 0, 1]:
                    if 0 <= i+m < img.width and 0 <= j+n < img.height:
                        arr.append(img.getpixel((i, j)))
                        n += 1
            arr.sort()
            median_img.putpixel((i, j), arr[n//2])

    # median_img.show()
    median_img.save('median.png')

if __name__ == '__main__':

    img_name = input('Image filename: ')
    img = Image.open(img_name).convert('L')

    print('1. Box/Mean Filter\n2. Median Filter')
    choice = int(input('Pick required operation: '))

    if choice == 1:
        low_pass_mean_filter(img)

    elif choice == 2:
        low_pass_median_filter(img)

    else:
        print('Invalid choice!')
```

Input Image

Output Images

Median 

Mean 

Program to sharpen an image using 2-D laplacian high pass filter in spatial domain.

```
from PIL import Image

def high_pass_laplacian_filter(img):

    laplacian_img = Image.new('L', (img.width, img.height))

    for i in range(img.width):
        for j in range(img.height):
            n = 0
            total = 0
            if i-1 >= 0:
                total += img.getpixel((i-1, j))
                n += 1
            if i+1 < img.width:
                total += img.getpixel((i+1, j))
```

```
                n += 1
            if j-1 >= 0:
                total += img.getpixel((i, j-1))
                n += 1
            if j+1 < img.height:
                total += img.getpixel((i, j+1))
                n += 1
            px = max(0, total - (n * img.getpixel((i, j))))
            laplacian_img.putpixel((i, j), px)

    # laplacian_img.show()
    laplacian_img.save('laplacian.png')

if __name__ == '__main__':

    img_name = input('Image filename: ')
    img = Image.open(img_name).convert('L')

    high_pass_laplacian_filter(img)
```

Input Image

Output Images

Program for detecting edges in an image using Roberts cross gradient operator and sobel operator.

```
from PIL import Image
import math

def robert_operator(img):
    Gx = [[1, 0], [0, -1]]
    Gy = [[0, 1], [-1, 0]]

    robert = Image.new('L', (img.width, img.height))

    for i in range(img.width-1):
        for j in range(img.height-1):
            x, y = 0, 0
            for p in range(2):
                for q in range(2):
                    x += (img.getpixel((i+p, j+q)) * Gx[p][q])
                    y += (img.getpixel((i+p, j+q)) * Gy[p][q])
            robert.putpixel((i, j), int(math.sqrt(((x**2)+(y**2)))))

    # robert.show()
    robert.save('robert.png')

def prewitt_operator(img):
    Gx = [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]
    Gy = [[1, 1, 1], [0, 0, 0], [-1, -1, -1]]

    prewitt = Image.new('L', (img.width, img.height))

    for i in range(1, img.width-1):
        for j in range(1, img.height-1):
            x, y = 0, 0
            for p in [-1, 0, 1]:
                for q in [-1, 0, 1]:
                    x += (img.getpixel((i+p, j+q)) * Gx[p+1][q+1])
                    y += (img.getpixel((i+p, j+q)) * Gy[p+1][q+1])
            prewitt.putpixel((i, j), int(math.sqrt(((x**2)+(y**2)))))

    # prewitt.show()
    prewitt.save('prewitt.png')
```

```
def sobel_operator(img):
    Gx = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]
    Gy = [[1, 2, 1], [0, 0, 0], [-1, -2, -1]]

    sobel = Image.new('L', (img.width, img.height))

    for i in range(1, img.width-1):
        for j in range(1, img.height-1):
            x, y = 0, 0
            for p in [-1, 0, 1]:
                for q in [-1, 0, 1]:
                    x += (img.getpixel((i+p, j+q)) * Gx[p+1][q+1])
                    y += (img.getpixel((i+p, j+q)) * Gy[p+1][q+1])
            sobel.putpixel((i, j), int(math.sqrt(((x**2)+(y**2)))))

    # sobel.show()
    sobel.save('sobel.png')

if __name__ == '__main__':

    img_name = input('Image filename: ')
    img = Image.open(img_name).convert('L')

    print('1. Robert operator\n2. Prewitt operator\n3. Sobel operator')
    choice = int(input('Pick required operation: '))

    if choice == 1:
        robert_operator(img)

    elif choice == 2:
        prewitt_operator(img)

    elif choice == 3:
        sobel_operator(img)

    else:
        print('Invalid choice!')
```

Input Image

Output Images

Robert 

Prewitt 

Sobel 

Programs for morphological image operations-Erosion, Dilation, Opening, Closing, Thinning, Thickening, Skeletons and Pruning, Boundary extraction, Hole filling, Extraction of connected components.

```
import imageio
import numpy as np

image = imageio.imread("horse.png", "PNG", pilmode="1")


def erode(img, se):
    se_size_rows = se.shape[0]
    se_size_cols = se.shape[1]
    row_pad = (se_size_rows - 1) // 2
    col_pad = (se_size_cols - 1) // 2

    padded_img = np.pad(
        array=img,
        pad_width=((row_pad, row_pad), (col_pad, col_pad)),
```

```
        mode="constant",
        constant_values=0,
    )
    padded_img_size_rows = padded_img.shape[0]
    padded_img_size_cols = padded_img.shape[1]

    result = np.zeros(padded_img.shape, dtype=np.uint8)

    for i, row in enumerate(img):
        for j, _ in enumerate(row):
            match = True

            for img_px, se_px in zip(
                padded_img[i : i + 2 * row_pad + 1, j : j + 2 * col_pad +
1].flatten(),
                se.flatten(),
            ):
                if se_px != 1:
                    continue

                if img_px != se_px:
                    match = False
                    break

            if match:
                result[i + row_pad][j + col_pad] = 1

    return result[
        row_pad : padded_img_size_rows - row_pad,
        col_pad : padded_img_size_cols - col_pad,
    ]


def dilate(img, se):
    se_size_rows = se.shape[0]
    se_size_cols = se.shape[1]
    row_pad = (se_size_rows - 1) // 2
    col_pad = (se_size_cols - 1) // 2

    padded_img = np.pad(
        array=img,
        pad_width=((row_pad, row_pad), (col_pad, col_pad)),
        mode="constant",
        constant_values=0,
```

```
    )
    padded_img_size_rows = padded_img.shape[0]
    padded_img_size_cols = padded_img.shape[1]

    result = np.zeros(padded_img.shape, dtype=np.uint8)

    for i, row in enumerate(img):
        for j, _ in enumerate(row):
            for img_px, se_px in zip(
                padded_img[i : i + 2 * row_pad + 1, j : j + 2 * col_pad +
1].flatten(),
                se.flatten(),
            ):
                if se_px != 1:
                    continue

                if img_px == se_px:
                    result[i + row_pad][j + col_pad] = 1
                    break

    return result[
        row_pad : padded_img_size_rows - row_pad,
        col_pad : padded_img_size_cols - col_pad,
    ]


def opening(img, se):
    return dilate(erode(img, se), se)


def closing(img, se):
    return erode(dilate(img, se), se)


def hit_miss_transform(img, se):
    se_size_rows = se.shape[0]
    se_size_cols = se.shape[1]
    row_pad = (se_size_rows - 1) // 2
    col_pad = (se_size_cols - 1) // 2

    padded_img = np.pad(
        array=img,
        pad_width=((row_pad, row_pad), (col_pad, col_pad)),
        mode="constant",
```

```
        constant_values=0,
    )
    padded_img_size_rows = padded_img.shape[0]
    padded_img_size_cols = padded_img.shape[1]

    result = np.zeros(padded_img.shape, dtype=np.uint8)

    for i, row in enumerate(img):
        for j, _ in enumerate(row):
            if (
                padded_img[i : i + 2 * row_pad + 1, j : j + 2 * col_pad +
1] == se
            ).all():
                result[i + row_pad][j + col_pad] = 1

    return result[
        row_pad : padded_img_size_rows - row_pad,
        col_pad : padded_img_size_cols - col_pad,
    ]


def thinning(img, se):
    return (img - hit_miss_transform(img, se)).clip(min=0, max=1)


def thickening(img, se):
    return (img + hit_miss_transform(img, se)).clip(min=0, max=1)


def skeleton(img, se):
    max_k = 0
    empty_erosion = False

    erosion = img
    skeletons = []
    while not empty_erosion:
        skeleton = (erosion - opening(erosion, se)).clip(min=0, max=1)
        skeletons.append(skeleton)

        erosion = erode(erosion, se)

        # Check if erosion is null set
        empty_erosion = not np.any(erosion)
```

```
        if not empty_erosion:
            max_k += 1

    return sum(skeletons).clip(min=0, max=1)


def pruning(img, ses, num_thinning_iter, num_dilation_iter):
    thinning_result = img  # x1

    for _ in range(num_thinning_iter):
        for se in ses:
            thinning_result = thinning(thinning_result, se)

    hmt_result = np.zeros(thinning_result.shape, dtype=np.uint8)  # x2
    for se in ses:
        hmt_result += hit_miss_transform(thinning_result, se)

    hmt_result = hmt_result.clip(min=0, max=1)

    dilation_se = np.array(
        [
            [1, 1, 1],
            [1, 1, 1],
            [1, 1, 1],
        ]
    )  # h

    dilation_result = hmt_result  # x3
    for _ in range(num_dilation_iter):
        dilation_result = np.bitwise_and(dilate(dilation_result,
dilation_se), img)

    return (thinning_result + dilation_result).clip(min=0, max=1)


def extract_boundary(img, se):
    return (img - erode(img, se)).clip(min=0, max=1)


def fill_holes(img, se, holes):
    result = holes
    inverse = 255 - img
    terminate = False
```

```
    while not terminate:
        new_result = np.bitwise_and(dilate(result, se), inverse)

        if (new_result == result).all():
            terminate = True

        result = new_result

    return (img + result).clip(min=0, max=1)


def connected_components(img, se, comps):
    result = comps
    terminate = False

    while not terminate:
        new_result = np.bitwise_and(dilate(result, se), img)

        if (new_result == result).all():
            terminate = True

        result = new_result

    return result


structuring_element = np.array(
    [
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
    ]
)

image = (image / 255).astype(np.uint8)

result = erode(image, structuring_element) * 255
imageio.imwrite("erosion.png", result, "PNG")

result = dilate(image, structuring_element) * 255
imageio.imwrite("dilation.png", result, "PNG")
```

```
result = opening(image, structuring_element) * 255
imageio.imwrite("opening.png", result, "PNG")

result = closing(image, structuring_element) * 255
imageio.imwrite("closing.png", result, "PNG")

result = hit_miss_transform(image, structuring_element) * 255
imageio.imwrite("hit_miss_transform.png", result, "PNG")

result = thinning(image, structuring_element) * 255
imageio.imwrite("thinning.png", result, "PNG")

result = thickening(image, structuring_element) * 255
imageio.imwrite("thickening.png", result, "PNG")

result = skeleton(image, structuring_element) * 255
imageio.imwrite("skeleton.png", result, "PNG")

pruning_ses = [
    np.array([[0, 0, 0], [1, 1, 0], [0, 0, 0]]),
    np.array([[0, 0, 0], [1, 1, 0], [1, 0, 0]]),
    np.array([[1, 0, 0], [1, 1, 0], [0, 0, 0]]),
    np.array([[1, 0, 0], [1, 1, 0], [1, 0, 0]]),
    np.array([[0, 1, 0], [0, 1, 0], [0, 0, 0]]),
    np.array([[0, 1, 1], [0, 1, 0], [0, 0, 0]]),
    np.array([[1, 1, 0], [0, 1, 0], [0, 0, 0]]),
    np.array([[1, 1, 1], [0, 1, 0], [0, 0, 0]]),
    np.array([[0, 0, 0], [0, 1, 1], [0, 0, 0]]),
    np.array([[0, 0, 0], [0, 1, 1], [0, 0, 1]]),
    np.array([[0, 0, 1], [0, 1, 1], [0, 0, 0]]),
    np.array([[0, 0, 1], [0, 1, 1], [0, 0, 1]]),
    np.array([[0, 0, 0], [0, 1, 0], [0, 1, 0]]),
    np.array([[0, 0, 0], [0, 1, 0], [0, 1, 1]]),
    np.array([[0, 0, 0], [0, 1, 0], [1, 1, 0]]),
    np.array([[0, 0, 0], [0, 1, 0], [1, 1, 1]]),
    np.array([[1, 0, 0], [0, 1, 0], [0, 0, 0]]),
    np.array([[0, 0, 1], [0, 1, 0], [0, 0, 0]]),
    np.array([[0, 0, 0], [0, 1, 0], [0, 0, 1]]),
    np.array([[0, 0, 0], [0, 1, 0], [1, 0, 0]]),
]

digit_img = (imageio.imread("digit.png", "PNG", pilmode="1") / 255).astype(
    np.uint8
)
```

```
result = pruning(digit_img, pruning_ses, 2, 2) * 255
imageio.imwrite("pruning.png", result, "PNG")

boundary_se = np.array(
    [
        [1, 1, 1],
        [1, 1, 1],
        [1, 1, 1],
    ]
)

result = extract_boundary(image, boundary_se) * 255
imageio.imwrite("boundary_extraction.png", result, "PNG")

hole_filling_se = np.array(
    [
        [0, 1, 0],
        [1, 1, 1],
        [0, 1, 0],
    ]
)

boundary = result
boundary = (boundary / 255).astype(np.uint8)
holes = np.zeros(boundary.shape, dtype=np.uint8)
holes[160, 200] = 1
result = fill_holes(boundary, hole_filling_se, holes) * 255
imageio.imwrite("hole_filling.png", result, "PNG")

comps = np.zeros(image.shape, dtype=np.uint8)
comps[160, 200] = 1
result = connected_components(image, boundary_se, comps) * 255
imageio.imwrite("connected_components.png", result, "PNG")
```

Input Image

We have considered 2 input images here , namely horse.png and digit.png . The horse.png is used to perform Erosion, Dilation, Opening, Closing, Thinning, Thickening, Skeletons and Pruning. The digit.png is used to perform the rest operations.

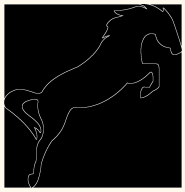
Input 1 

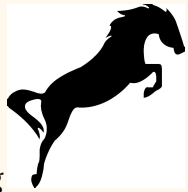
Input 2 

Output Images

Opening 

Thickening 

Thinning 

Closing 

Dilation 

Erosion 

Hit-miss Transform 

Program for segmentation of an image: To detect lines, edges, boundaries.

```
# Points, lines and edges detection
import imageio
import numpy as np

image = imageio.imread("input.png", "PNG", pilmode="L")

# fmt: off

point_laplacian_kernel = np.array(
    [
        [1,  1,  1],
        [1, -8,  1],
        [1,  1,  1],
    ]
)

line_horizontal_kernel = np.array(
    [
        [-1, -1, -1],
        [ 2,  2,  2],
        [-1, -1, -1],
    ]
)

line_vertical_kernel = np.array(
    [
        [-1,  2, -1],
        [-1,  2, -1],
        [-1,  2, -1],
    ]
)

line_plus_45_kernel = np.array(
    [
        [ 2, -1, -1],
        [-1,  2, -1],
        [-1, -1,  2],
    ]
)
```

```
line_minus_45_kernel = np.array(
    [
        [-1, -1,  2],
        [-1,  2, -1],
        [ 2, -1, -1],
    ]
)

# fmt: on

row_pad = 1
column_pad = 1

# Pad the image with sufficient number of zeros
padded_image = np.pad(
    array=image,
    pad_width=((column_pad, column_pad), (row_pad, row_pad)),
    mode="edge",
)


def threshold_image(img, threshold):
    result = np.zeros(img.shape, dtype=np.uint8)

    for i, row in enumerate(img):
        for j, value in enumerate(row):
            if value > threshold:
                result[i][j] = 255

    return result


# Create arrays of same shape as padded image
points = np.zeros(padded_image.shape, dtype=int)
lines_horizontal = np.zeros(padded_image.shape, dtype=int)
lines_plus_45 = np.zeros(padded_image.shape, dtype=int)
lines_vertical = np.zeros(padded_image.shape, dtype=int)
lines_minus_45 = np.zeros(padded_image.shape, dtype=int)

# Convolve image with kernels
for i, row in enumerate(image):
    for j, _ in enumerate(row):
        points[i + row_pad][j + column_pad] = (
            point_laplacian_kernel
```

```
            * padded_image[i : i + 2 * row_pad + 1, j : j + 2 * column_pad
+ 1]
        ).sum(dtype=int)

        lines_horizontal[i + row_pad][j + column_pad] = (
            line_horizontal_kernel
            * padded_image[i : i + 2 * row_pad + 1, j : j + 2 * column_pad
+ 1]
        ).sum(dtype=int)

        lines_vertical[i + row_pad][j + column_pad] = (
            line_vertical_kernel
            * padded_image[i : i + 2 * row_pad + 1, j : j + 2 * column_pad
+ 1]
        ).sum(dtype=int)

        lines_plus_45[i + row_pad][j + column_pad] = (
            line_plus_45_kernel
            * padded_image[i : i + 2 * row_pad + 1, j : j + 2 * column_pad
+ 1]
        ).sum(dtype=int)

        lines_minus_45[i + row_pad][j + column_pad] = (
            line_minus_45_kernel
            * padded_image[i : i + 2 * row_pad + 1, j : j + 2 * column_pad
+ 1]
        ).sum(dtype=int)

points = threshold_image(points[1:-1, 1:-1].clip(min=0,
max=255).astype(np.uint8), 200)
lines_horizontal = threshold_image(
    lines_horizontal[1:-1, 1:-1].clip(min=0, max=255).astype(np.uint8), 200
)
lines_vertical = threshold_image(
    lines_vertical[1:-1, 1:-1].clip(min=0, max=255).astype(np.uint8), 200
)
lines_plus_45 = threshold_image(
    lines_plus_45[1:-1, 1:-1].clip(min=0, max=255).astype(np.uint8), 200
)
lines_minus_45 = threshold_image(
    lines_minus_45[1:-1, 1:-1].clip(min=0, max=255).astype(np.uint8), 200
)

imageio.imwrite("points.png", points, "PNG")
```

```
imageio.imwrite("lines_horizontal.png", lines_horizontal, "PNG")
imageio.imwrite("lines_vertical.png", lines_vertical, "PNG")
imageio.imwrite("lines_plus_45.png", lines_plus_45, "PNG")
imageio.imwrite("lines_minus_45.png", lines_minus_45, "PNG")

# Edge detection: Roberts, Prewitt, Sobel operators program in
'07_edge_detection_roberts_prewitt_sobel' directory
```

Input Image

Output Images

Horizontal Lines 

Lines(-45) 

Lines(+45) 

Vertical Lines 

Points