p5-1

July 17, 2020

0.0.1 Mohammadreza Osouli - 610395077 - Phase 5.1

[1]:
```
import math
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

Convolution on 2d matrix function

[2]:
```
def convolution2d(image, kernel):
    m, n = kernel.shape
    y, x = image.shape
    y = y - m + 1
    x = x - m + 1
    new_image = np.zeros((y, x))
    for i in range(y):
        for j in range(x):
            new_image[i][j] = np.sum(image[i:i+m, j:j+m]*kernel)
    return new_image
```

DoG kernel function:

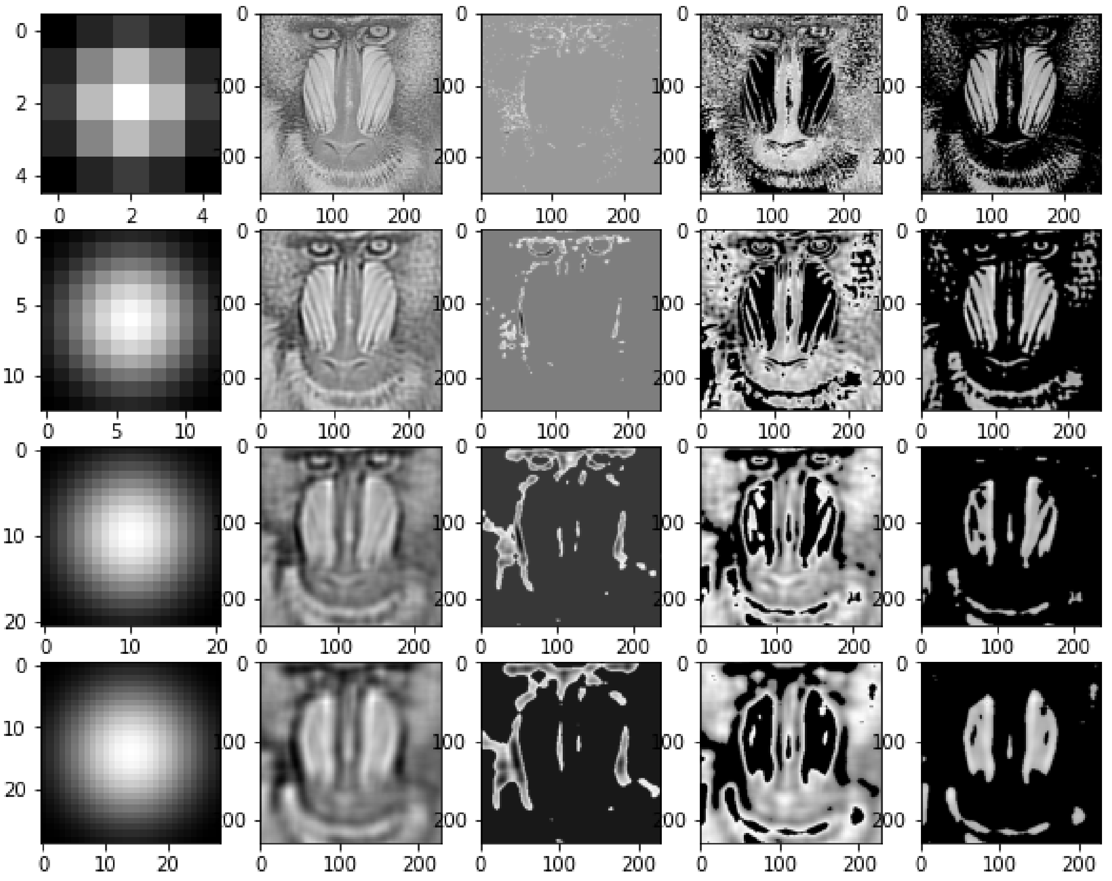
[3]:
```
def get_dog_kernel(sigma1, sigma2, kernel_size):
    if kernel_size % 2 == 0:
        raise ValueError('kernel_size should be an odd number like 3, 5 and ...
 ↪')
    result = np.zeros((kernel_size, kernel_size))
    for i in range(kernel_size):
        for j in range(kernel_size):
            x, y = i - kernel_size // 2, j - kernel_size // 2
            g1 = (1 / sigma1) * math.exp(-(x * x + y * y) / (2 * sigma1 *␣
 ↪sigma1))
            g2 = (1 / sigma2) * math.exp(-(x * x + y * y) / (2 * sigma2 *␣
 ↪sigma2))
            result[i][j] = (1 / (math.sqrt(2 * math.pi)) * (g1 - g2))
    return result
```

Main task: In this task first, we read a file from file and convert it to grayscale mode (to have one channel instead of three), then get all DoG kernels we need (I decide to have sigma values related to kernel size) and after convolve it with source image, show them as result.

Any row of result contains kernel, result image and segmented result images to 3 parts (can be changed in `parts` variable).

[10]:
```
gray = np.asarray(Image.open("data/gorill.png").convert('L'))
kernels = [get_dog_kernel(size / 3.2, size / 2, size) for size in [5, 13, 21,␣
 ↪29]]
filtered_images = [convolution2d(gray, kernel) for kernel in kernels]

parts = 3
fig, axes = plt.subplots(len(kernels), 2 + parts, figsize=(10, 8))
for i in range(len(kernels)):
    axes[i][0].imshow(kernels[i], "gray")
    axes[i][1].imshow(filtered_images[i], "gray")
    min_element, max_element = np.min(filtered_images[i]), np.
 ↪max(filtered_images[i])
    for j in range(parts):
        min_thresh = min_element + (max_element - min_element) * j / parts
        max_thresh = min_element + (max_element - min_element) * (j +1) / parts
        axes[i][2 + j].imshow((min_thresh <= filtered_images[i]) *␣
 ↪(filtered_images[i] < max_thresh) * filtered_images[i], "gray")
plt.show()
```


0
2
4
0
5
10
20
100
200