Computer Vision HW10: Zero Crossing Edge Detection

R10741015 鄭傑鴻

Nov. 22, 2022

Implement 2 Laplacian Mask, Minimum Variance Laplacian, Laplacian of Gaussian, and Difference of Gaussian(inhibitory sigma=3, excitatory sigma=1, kernel size 11x11).

- **Description**

  Since homework 10 have some similarities with homework 9, I revised and utilized some functions from my previous work.

- **Method / Algorithm**

  1. Set up the masks: Construct a class “Kernel” to store the related information of the masks, also define a method doKernelConv() to do convolution on the given picture position.
  2. Pad the input image based on the kernel size
  3. Iterate over the padded image and do convolution to get the intermediate result:
     - If the gradient value greater than threshold: assign 1
     - If the gradient values lesser than -1*threshold: assign -1
     - Else assign 0
  4. Pad the intermediate result based on the kernel size again
  5. Iterate over the padded intermediate result and decide whether it’s zero crossing:
     - If (the current pixel with a label of 1) and (there’s label -1 exist in its 8-connected neighbor): assign 0 (Black pixel)
     - Else assign 255(white pixel)

- **Main Code Segment**

  Class of kernel (for convolution purpose)

```
class Kernel():
    class KernelUnit():
        row = 0
        col = 0
        weight = 0

        def __init__(self, r, c, w):
            self.row = r
            self.col = c
            self.weight = w

    def __init__(self, *args, norm=1.0):
        self.length = int(np.sqrt(len(args)))
        self.d = int((self.length - 1) / 2)
        self.elements = []
        self.normalize = norm
        for i in range(self.length):
            for j in range(self.length):
                self.elements.append(self.KernelUnit(i - self.d, j - self.d, args[i * self.length + j]))

    def DoKernelCov(self, img, row, col):
```

```
        retVal = 0
        for elm in self.elements:
            retVal += (elm.weight * img[row + elm.row][col + elm.col])
        retVal *= self.normalize
        return retVal
```

Method to detect zero crossing:

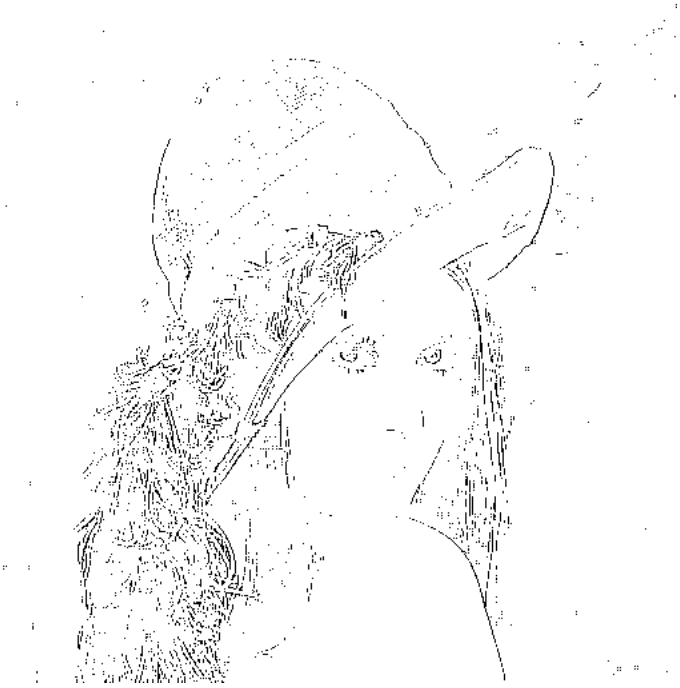
```
def DetectZeroCrossingEdge(self, inPic, Kernel, thr):
    laplaPic = np.zeros_like(inPic).astype(int)
    paddedPic = cv2.copyMakeBorder(inPic, Kernel.d, Kernel.d, Kernel.d, Kernel.d, cv2.BORDER_REPLICATE)

    for i in range(inPic.shape[0]):
        for j in range(inPic.shape[1]):
            resultVal = Kernel.DoKernelCov(paddedPic, i + Kernel.d, j + Kernel.d)
            if resultVal >= thr:
                laplaPic[i][j] = 1
            elif resultVal <= -1 * thr:
                laplaPic[i][j] = -1
            else:
                laplaPic[i][j] = 0

    def ISCrossing(img, row, col):
        for i in range(-1 * 1, 1 + 1):
            for j in range(-1 * 1, 1 + 1):
                if img[row + i][col + j] == -1: return True
        return False

    retPic = np.zeros_like(inPic)
    paddedLaplaPic = cv2.copyMakeBorder(laplaPic, Kernel.d, Kernel.d, Kernel.d, Kernel.d,
cv2.BORDER_REPLICATE)
    for i in range(inPic.shape[0]):
        for j in range(inPic.shape[1]):
            if paddedLaplaPic[i + Kernel.d][j + Kernel.d] != 1:
                retPic[i][j] = 255
            else:
                if ISCrossing(paddedLaplaPic, i + Kernel.d, j + Kernel.d):
                    retPic[i][j] = 0
                else:
                    retPic[i][j] = 255
    return retPic
```

- **Results**

a) Laplace Mask1 (0, 1, 0, 1, -4, 1, 0, 1, 0): 15

b) Laplace Mask2 (1, 1, 1, 1, -8, 1, 1, 1, 1)

c) Minimum variance Laplacian: 20



d) Laplace of Gaussian: 3000

e) Difference of Gaussian: 1