922 U0610 電腦視覺
Computer Vision

Homework 8

授課教師： 傅楸善 教授

學生系級： 資工所一年級

學生姓名： 姚嘉昇

學生學號： R06922002

I. INTRODUCTION

1.1. Descriptions of Problem

This homework is to do noise removal with following rules:

A. Generate Gaussian noise with amplitude of 10 and 30.
B. Generate salt-and-pepper noise with probability of 0.1 and 0.05.
C. Use the 3x3 and 5x5 box filter on noise images.
D. Use the 3x3 and 5x5 median filter on noise images.
E. Use opening-then-closing and closing-then-opening filter on noise images.
F. Calculate the signal-to-noise-ratio (SNR) of noise images.

1.2. Programming Tools

1.2.1. Programming Language: Python3

1.2.2. Programming IDE: Visual Studio Code

II. METHOD

2.1. Algorithms

2.1.1. Box filter

- 3x3 box filter
- 5x5 box filter

2.1.2. Median filter

- 3x3 median filter
- 5x5 median filter

2.1.3. Opening-then-closing

2.1.4. Closing-then-opening

2.2. Code Fragments

2.2.1. Code fragments of this homework

```python
def getGaussianNoiseImage(originalImage, amplitude):
    """
    :type originalImage: Image (from PIL)
    :type amplitude: float
    :return type: Image (from PIL)
    """
    from PIL import Image
    import random
    # Copy image from origianl image.
    gaussianNoiseImage = originalImage.copy()
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Get pixel value with gaussian noise.
            noisePixel = int(originalImage.getpixel((c, r)) + amplitude * random.gauss(0, 1))
            # Limit pixel value at 255.
            if noisePixel > 255:
                noisePixel = 255
            # Put pixel to noise image.
            gaussianNoiseImage.putpixel((c, r), noisePixel)
    return gaussianNoiseImage
```

Figure 2.2.1.1. Code of get Gaussian Noise image function.

```python
def getSaltAndPepperImage(originalImage, probability):
    """
    :type originalImage: Image (from PIL)
    :type probability: float
    :return type: Image (from PIL)
    """
    from PIL import Image
    import random
    # Copy image from origianl image.
    saltAndPepperImage = originalImage.copy()
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Get random value.
            randomValue = random.uniform(0, 1)
            if (randomValue <= probability):
                # Put black pixel(pepper) to image.
                saltAndPepperImage.putpixel((c, r), 0)
            elif (randomValue >= 1 - probability):
                # Put white pixel(salt) to image.
                saltAndPepperImage.putpixel((c, r), 255)
            else:
                # Put origianl pixel to image.
                saltAndPepperImage.putpixel((c, r), originalImage.getpixel((c, r)))
    return saltAndPepperImage
```

Figure 2.2.1.2. Code of get Salt-and-pepper image function.

```python
def boxFilter(originalImage, boxWidth, boxHeight):
    """
    :type originalImage: Image (from PIL)
    :type boxWidth: integer
    :type boxHeight: integer
    :return type: Image (from PIL)
    """
    # Calculate center of kernel.
    centerKernel = (boxWidth // 2, boxHeight // 2)
    # Copy image from origianl image.
    boxFilterImage = originalImage.copy()
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Create empty list.
            boxPixels = []
            # Scan each column in box.
            for x in range(boxWidth):
                # Scan each row in box.
                for y in range(boxHeight):
                    # Calculate destination x, y position.
                    destX = c + (x - centerKernel[0])
                    destY = r + (y - centerKernel[1])
                    # Avoid out of image range.
                    if ((0 <= destX < originalImage.size[0]) and \
                        (0 <= destY < originalImage.size[1])):
                        # Get pixel value in original image at (destX, destY).
                        originalPixel = originalImage.getpixel((destX, destY))
                        # Append pixel to list.
                        boxPixels.append(originalPixel)
            boxFilterImage.putpixel((c, r), int(sum(boxPixels) / len(boxPixels)))
    return boxFilterImage
```

Figure 2.2.1.3. Code of box filter.

```python
def medianFilter(originalImage, boxWidth, boxHeight):
    """
    :type originalImage: Image (from PIL)
    :type boxWidth: integer
    :type boxHeight: integer
    :return type: Image (from PIL)
    """
    # Calculate center of kernel.
    centerKernel = (boxWidth // 2, boxHeight // 2)
    # Copy image from origianl image.
    medianFilterImage = originalImage.copy()
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Create empty list.
            boxPixels = []
            # Scan each column in box.
            for x in range(boxWidth):
                # Scan each row in box.
                for y in range(boxHeight):
                    # Calculate destination x, y position.
                    destX = c + (x - centerKernel[0])
                    destY = r + (y - centerKernel[1])
                    # Avoid out of image range.
                    if ((0 <= destX < originalImage.size[0]) and \
                        (0 <= destY < originalImage.size[1])):
                        # Get pixel value in original image at (destX, destY).
                        originalPixel = originalImage.getpixel((destX, destY))
                        # Append pixel to list.
                        boxPixels.append(originalPixel)
            # Sort pixels in box.
            boxPixels.sort()
            # Get Median pixel.
            medianPixel = boxPixels[len(boxPixels) // 2]
            # Put median pixel to image.
            medianFilterImage.putpixel((c, r), medianPixel)
    return medianFilterImage
```

Figure 2.2.1.4. Code of median filter.

```python
def dilation(originalImage, kernel, centerKernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :type centerKernel: tuple
    :return type: Image (from PIL)
    """
    from PIL import Image
    # New image with the same size and 'grayscale' format.
    dilationImage = Image.new('L', originalImage.size)
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Record local max. pixel value.
            localMaxPixel = 0
            # Scan each column in kernel.
            for x in range(kernel.shape[0]):
                # Scan each row in kernel.
                for y in range(kernel.shape[1]):
                    # Only check value '1' in kernel.
                    if (kernel[x, y] == 1):
                        # Calculate destination x, y position.
                        destX = c + (x - centerKernel[0])
                        destY = r + (y - centerKernel[1])
                        # Avoid out of image range.
                        if ((0 <= destX < originalImage.size[0]) and \
                            (0 <= destY < originalImage.size[1])):
                            # Get pixel value in original image at (destX, destY).
                            originalPixel = originalImage.getpixel((destX, destY))
                            # Update local max. pixel value.
                            localMaxPixel = max(localMaxPixel, originalPixel)
            # Paste local max. pixel value on original image.
            dilationImage.putpixel((c, r), localMaxPixel)
    # Return dilation image.
    return dilationImage
```

Figure 2.2.1.5. Code of dilation.

```python
def erosion(originalImage, kernel, centerKernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :type centerKernel: tuple
    :return type: Image (from PIL)
    """
    from PIL import Image
    # New image with the same size and 'grayscale' format.
    erosionImage = Image.new('L', originalImage.size)
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Record local min. pixel value.
            localMinPixel = 255
            # Scan each column in kernel.
            for x in range(kernel.shape[0]):
                # Scan each row in kernel.
                for y in range(kernel.shape[1]):
                    # Only check value '1' in kernel.
                    if (kernel[x, y] == 1):
                        # Calculate destination x, y position.
                        destX = c + (x - centerKernel[0])
                        destY = r + (y - centerKernel[1])
                        # Avoid out of image range.
                        if ((0 <= destX < originalImage.size[0]) and \
                            (0 <= destY < originalImage.size[1])):
                            # Get pixel value in original image at (destX, destY).
                            originalPixel = originalImage.getpixel((destX, destY))
                            # Update local min. pixel value.
                            localMinPixel = min(localMinPixel, originalPixel)
            # Paste local min. pixel value on original image.
            erosionImage.putpixel((c, r), localMinPixel)
    # Return erosion image.
    return erosionImage
```

Figure 2.2.1.6. Code of erosion.

```python
def opening(originalImage, kernel, centerKernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :type centerKernel: tuple
    :return type: Image (from PIL)
    """
    return dilation(erosion(originalImage, kernel, centerKernel), kernel, centerKernel)

def closing(originalImage, kernel, centerKernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :type centerKernel: tuple
    :return type: Image (from PIL)
    """
    return erosion(dilation(originalImage, kernel, centerKernel), kernel, centerKernel)

def openingThenClosing(originalImage, kernel, centerKernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :type centerKernel: tuple
    :return type: Image (from PIL)
    """
    return closing(opening(originalImage, kernel, centerKernel), kernel, centerKernel)

def closingThenOpening(originalImage, kernel, centerKernel):
    """
    :type originalImage: Image (from PIL)
    :type kernel: numpy array
    :type centerKernel: tuple
    :return type: Image (from PIL)
    """
    return opening(closing(originalImage, kernel, centerKernel), kernel, centerKernel)
```

Figure 2.2.1.7. Code of opening and closing.

```python
def getSNR(signalImage, noiseImage):
    """
    :type signalImage: Image (from PIL)
    :type noiseImage: Image (from PIL)
    :return type: float
    """
    import math
    # Clear mu and power of signal and noise.
    muSignal = 0
    powerSignal = 0
    muNoise = 0
    powerNoise = 0

    # Scan each column in signal image.
    for c in range(signalImage.size[0]):
        # Scan each row in signal image.
        for r in range(signalImage.size[1]):
            muSignal = muSignal + signalImage.getpixel((c, r))
    # Average mu of signal.
    muSignal = muSignal / (signalImage.size[0] * signalImage.size[1])

    # Scan each column in noise image.
    for c in range(noiseImage.size[0]):
        # Scan each row in noise image.
        for r in range(noiseImage.size[1]):
            muNoise = muNoise + (noiseImage.getpixel((c, r)) - signalImage.getpixel((c, r)))
    # Average mu of noise.
    muNoise = muNoise / (noiseImage.size[0] * noiseImage.size[1])

    # Scan each column in signal image.
    for c in range(signalImage.size[0]):
        # Scan each row in signal image.
        for r in range(signalImage.size[1]):
            powerSignal = powerSignal + math.pow(signalImage.getpixel((c, r)) - muSignal, 2)
    # Average power of signal.
    powerSignal = powerSignal / (signalImage.size[0] * signalImage.size[1])

    # Scan each column in noise image.
    for c in range(noiseImage.size[0]):
        # Scan each row in noise image.
        for r in range(noiseImage.size[1]):
            powerNoise = powerNoise +  math.pow((noiseImage.getpixel((c, r)) - signalImage.getpixel((c, r))) - muNoise, 2)
    # Average mu of noise.
    powerNoise = powerNoise / (noiseImage.size[0] * noiseImage.size[1])
```

Figure 2.2.1.8. Code of SNR.

III. RESULTS

3.1. Original Image

Figure 3.1. Original lena.bmp.

3.2. Results of Gaussian Noise Image

Figure 3.2.1. gaussianNoise_10.bmp.

Figure 3.2.2. gaussianNoise_30.bmp.

3.3. Results of Salt-and-Pepper

Figure 3.3.1. saltAndPepper_0_05.bmp.

Figure 3.3.2. saltAndPepper_0_10.bmp.

3.4. Results of 3x3 Box Filter

Figure 3.4.1.
gaussianNoise_10_box_3x3.bmp.

Figure 3.4.2.
gaussianNoise_30_box_3x3.bmp.

Figure 3.4.3.
saltAndPepper_0_05_box_3x3.bmp

Figure 3.4.4.
saltAndPepper_0_10_box_3x3.bmp

3.5. Results of 5x5 Box Filter

Figure 3.5.1.
gaussianNoise_10_box_5x5.bmp.

Figure 3.5.2.
gaussianNoise_30_box_5x5.bmp.

Figure 3.5.3.
saltAndPepper_0_05_box_5x5.bmp

Figure 3.5.4.
saltAndPepper_0_10_box_5x5.bmp

3.6. Results of 3x3 Median Filter

Figure 3.6.1.
gaussianNoise_10_median_3x3.bmp.

Figure 3.6.2.
gaussianNoise_30_median_3x3.bmp.

Figure 3.6.3.
saltAndPepper_0_05_median_3x3.bmp

Figure 3.6.4.
saltAndPepper_0_10_median_3x3.bmp

3.7. Results of 5x5 Median Filter

Figure 3.7.1.
gaussianNoise_10_median_5x5.bmp.

Figure 3.7.2.
gaussianNoise_30_median_5x5.bmp.

Figure 3.7.3.
saltAndPepper_0_05_median_5x5.bmp

Figure 3.7.4.
saltAndPepper_0_10_median_5x5.bmp

3.8. Results of Opening-then-Closing

Figure 3.8.1.
gaussianNoise_10_openingThenClosing.bmp.

Figure 3.8.2.
gaussianNoise_30_openingThenClosing.bmp.

Figure 3.8.3.
saltAndPepper_0_05_openingThenClosing.bmp

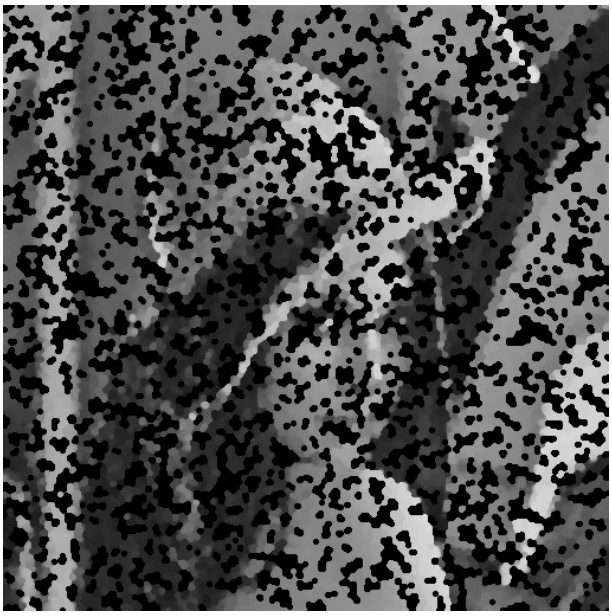

Figure 3.8.4.
saltAndPepper_0_10_openingThenClosing.bmp

3.9. Results of Closing-then-Opening

Figure 3.9.1.
gaussianNoise_10_closingThenOpening.bmp.

Figure 3.9.2.
gaussianNoise_30_closingThenOpening.bmp.

Figure 3.9.3.
saltAndPepper_0_05_closingThenOpening.bmp

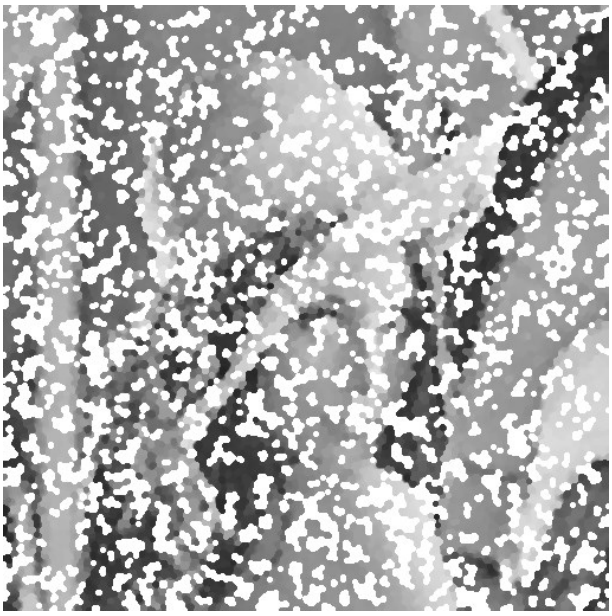

Figure 3.9.4.
saltAndPepper_0_10_closingThenOpening.bmp

3.10. Results of SNR

Filter		Gaussian Noise	
		Amplitude=10	Amplitude=30
No filter		13.577673848918188	4.179277588955975
Box	3x3	17.731140149360535	12.610573226583657
	5x5	14.857729621739733	13.311954047893185
Median	3x3	17.651055018705332	11.068588450952369
	5x5	15.986375902746365	12.923616645551498
Opening-then-Closing		13.246475409841953	11.107239975017215
Closing-then-Opening		13.587499304142945	11.161285816210508

Filter		Salt-and-Pepper	
		Probability=0.05	Probability=0.10
No filter		0.919599038789096	-2.1331972742191896
Box	3x3	9.437939435017782	6.276181353721542
	5x5	11.144625145075523	8.45026635583591
Median	3x3	19.038446617464842	14.871190557310829
	5x5	16.3801023822922	15.78842254312368
Opening-then-Closing		5.6121449335507565	-2.234575099322334
Closing-then-Opening		5.228390279049658	-2.538225164674737