assignment3

September 9, 2018

1 Machine Learning and Computer Vision

1.1 Assigment 3

This assignment contains 2 programming exercises.

1.2 Problem 1: Order-statistic filtering

Order-statistic filters (OSF) are local filters that are only based on the ranking of pixel values inside a sliding window. 1. Create in imstack(img,s1,s2) function that creates a stack xstack of size n1 Œn2 Œs, which s = (2s1 +1)(2s2 +1) from the n1 Œn2 image x, such that xstack(i,j,:) contains all the values of x in the neighborhood (s1, s1) Œ (s2, s2). This function should take into account the four possible boundary conditions.

```
Hint: you can use imshift, which we implemented in assignment 1, and only two loops for s1 <= l
```

2. Create in imosf() function function imosf(x, type, s1, s2) that implements order-statistic filters, returns xosf. imosf should first call imstack, next sort the entries of the stack with respect to the third dimension, and create the suitable output xosf according to the string type as follows:

   'median': select the median value,

   'erode': select the min value,

   'dilate': select the max value,

   'trimmed': take the mean after excluding at least 25% of the extreme values on each side.

3. Create in imopening() and imclosing() function that performs the opening and closing by the means of OSF filters.

4. Load castle.png. Write a script to test imosf() that loads the image x = castle and create a corrupted version of image x with 10% of impulse noise (salt and pepper)

Apply your OSF filters and zoom on the results to check that your results are consistent with the following ones:

In [1]:
```
import random
import numpy as np
from scipy.misc import imread
import matplotlib.pyplot as plt


def imstack(img, s1, s2):
    n1, n2 = img.shape
    s = (2*s1+1)*(2*s2+1)
    xstack = np.empty((n1, n2, 0))
    for k in range(-s1,s1+1):
        for l in range(-s2,s2+1):
            xshift = np.expand_dims(imshift(img, -k, -l), axis=-1)
            xstack = np.append(xshift,xstack, axis=-1)
    return xstack

def imshift(x, k, l):
    h,w = x.shape
    xshifted = np.zeros((h,w))
    if k == 0:
        k = h
    if l == 0:
        l = w
    xshifted[-k:,-l:] = x[:k,:l]
    xshifted[:-k,:-l] = x[k:,l:]
    xshifted[:-k,-l:] = x[k:,:l]
    xshifted[-k:,:-l] = x[:k,l:]
    return xshifted


def sort(simg):
    n1, n2 = simg.shape
    for h in range(n1):
        for w in range(n2):
            simg[h, w].sort()
    return simg


def imosf(x, typ, s1, s2):
    n1, n2 = x.shape
    xosf = np.zeros((n1, n2))
    xstack = imstack(x, s1, s2)
    if typ == "median":
        xosf = np.median(xstack, axis=-1)
    elif typ =="erode":
        xosf = np.min(xstack, axis=-1)
    elif typ == "dilate":
        xosf = np.max(xstack, axis=-1)
```

```
    elif typ == "trimmed":
        xosf = np.sort(xstack, axis=-1)
        s = 2*(s1+1)*2*(s2+1)
        xosf = np.mean(xosf[:, :, int(s*0.25):int(s*0.75)], axis=-1)
    elif typ == "close":
        tem = np.max(xstack, axis=-1)
        tem = imstack(tem, s1, s2)
        xosf =  np.min(tem, axis=-1)
    elif typ == "open":
        tem = np.min(xstack, axis=-1)
        tem = imstack(tem, s1, s2)
        xosf =  np.max(tem, axis=-1)
    return xosf


def pepper(x, rate):
    n1, n2 = x.shape
    noise_p = int(n1*n2*rate)
    for r in range(noise_p):
        x[random.randint(0, n1-1),random.randint(0, n2-1)] = random.choice([0, 255])
    return x


s1 = 2
s2 = 2
img = imread('castle.png')
img_noise = pepper(img, 0.05)

plt.subplot(1, 5, 1)
plt.title("noise")
plt.axis('off')
plt.imshow(img_noise ,cmap = plt.get_cmap('gray'))

img1 = imosf(img_noise,"median",s1,s2)
plt.subplot(1, 5, 2)
plt.title("median")
plt.axis('off')
plt.imshow(img1 ,cmap = plt.get_cmap('gray'))

img2 = imosf(img_noise,"trimmed",s1,s2)
plt.subplot(1, 5, 3)
plt.title("trimmed")
plt.axis('off')
plt.imshow(img2 ,cmap = plt.get_cmap('gray'))


img11 = imosf(img_noise,"close",s1,s2)
plt.subplot(1, 5, 4)
```

```
        plt.title("closing")
        plt.axis("off")
        plt.imshow(img11 ,cmap = plt.get_cmap('gray'))

        img22 = imosf(img_noise,"open",s1,s2)
        plt.subplot(1, 5, 5)
        plt.title("opening")
        plt.axis("off")
        plt.imshow(img22 ,cmap = plt.get_cmap('gray'))

        plt.show()
```

```
C:\study\anaconda\lib\site-packages\ipykernel_launcher.py:74: DeprecationWarning: `imread` is d
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.


<matplotlib.figure.Figure at 0x1eeddb6d2b0>
```

1.3 Problem 2: Bilateral filter

Now, we will discuss a non-local filter, Bilateral filter.

The bilateral filter is a denoising algorithm that reads as:

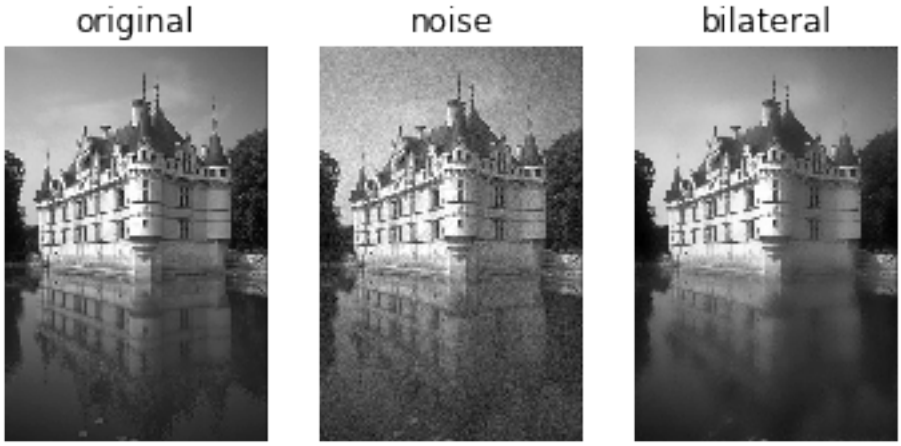
1. Create a test_imbilateral(img, sigma) function that loads the image x = castle and adds additive white Gaussian noise of standard deviation = 10
2. Create in imbilateral_naive(img, sigma, s1, s2, h), a function that implements the bilateral filter (except around boundaries) with four loops.
3. test your function on y with s1 = s2 = 10 and h = 1. Zoom on the results to check that your functions are consistent with the following ones:
4. Create function imbilateral(y, sigma, s1, s2, h)that implements the bilateral filter including around boundaries. The idea is again to switch the k, l loops with the i, j loops, and then make use of imshift. The final code should read with only two loops and deal with boundary conditions.
5. Compare the computation times.
6. Increase the noise level, and play with the search window sizes s1 and s2 and filtering parameter h.

```
In [2]: import random
        import numpy as np
        from scipy.misc import imread
        import matplotlib.pyplot as plt
        import time
```

```
def GaussianNoise(x, means, sigma):
    n1, n2 = x.shape
    noise_img = np.zeros((n1,n2))
    for i in range(n1):
        for j in range(n2):
            noise_img[i, j] = x[i, j] + random.gauss(means, sigma)
            if noise_img[i, j] < 0:
                noise_img[i, j] = 0
            elif noise_img[i, j] > 255:
                noise_img[i, j] = 255
    return noise_img


def test_imbilateral():
    sigma = 10
    H = 1
    s1 = s2 = 10
    img = imread('castle.png')

    plt.subplot(1, 3, 1)
    plt.title("original")
    plt.axis('off')
    plt.imshow(img, cmap=plt.get_cmap('gray'))

    img_noise = GaussianNoise(img, 0, 10)
    plt.subplot(1, 3, 2)
    plt.title("noise")
    plt.axis('off')
    plt.imshow(img_noise, cmap=plt.get_cmap('gray'))

    t0 = time.time()
    img1 = imbilateral(img_noise, s1, s2, sigma, H)
    print("time cost:",time.time()-t0)
    plt.subplot(1, 3, 3)
    plt.title("bilateral")
    plt.axis('off')
    plt.imshow(img1, cmap=plt.get_cmap('gray'))
    plt.show()



def imbialteral_navie(img, s1, s2, sigma, H):
    n1, n2 = img.shape
    phi = lambda a: np.exp(-(max(a - 2 * sigma ** 2, 0) / (16 * H * sigma ** 2)))
    z = np.zeros((n1, n2))
    x = np.zeros((n1, n2))
    for i in range(n1):
```

```
            for j in range(n2):
                for k in range(-s1, s1 + 1):
                    for l in range(-s2, s2 + 1):
                        if (i+k)>(n1-1) or (j+l)>(n2-1):
                            continue
                        x[i, j] += phi((img[(i + k), (j + l)] - img[i, j]) ** 2) * img[(i +
                        z[i, j] += phi((img[(i + k), (j + l)] - img[i, j]) ** 2)

        x = x / z
        return x



    def imbilateral(img, s1, s2, sigma, H):
        n1, n2 = img.shape
        phi = lambda a: np.exp(-(np.maximum(a - 2 * sigma ** 2, 0) / (16 * H * sigma ** 2))
        z = np.zeros((n1, n2))
        x = np.zeros((n1, n2))
        for k in range(-s1, s1 + 1):
            for l in range(-s2, s2 + 1):
                xshift = imshift(img, -k, -l)
                x += phi((xshift - img) ** 2) * xshift
                z += phi((xshift - img) ** 2)
        x = x / z
        return x


    def imshift(x, k, l):
        h, w = x.shape
        xshifted = np.zeros((h, w))
        if k == 0:
            k = h
        if l == 0:
            l = w
        xshifted[-k:, -l:] = x[:k, :l]
        xshifted[:-k, :-l] = x[k:, l:]
        xshifted[:-k, -l:] = x[k:, :l]
        xshifted[-k:, :-l] = x[:k, l:]
        return xshifted


    test_imbilateral()
```

```
C:\study\anaconda\lib\site-packages\ipykernel_launcher.py:25: DeprecationWarning: `imread` is d
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.


cost: 4.520999431610107
```



1.4 Conclusion

Have you accomplished all parts of your assignment? What concepts did you used or learned in this assignment? What difficulties have you encountered? Explain your result for each section. Please wirte one or two short paragraph in the below Markdown window (double click to edit).

**** Your Conclusion: ****

-- For this time, I learn many things, such as OSF and some other filter such as Bilateral filter. The first time I just don't know how to deal with the matrix transmission. Then I search google and finally find some useful meathods in np.

** Submission Instructions**

Remember to submit you pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via LaTeX