Untitled17

January 16, 2022

[ ]:
```
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 13 08:38:35 2021

@author: tiitu
"""



import os
import sys
sys.path.append(os.getcwd())

from matplotlib.pyplot import imread
from skimage.transform import resize as imresize
#from scipy.misc import imresize  # deprecated, may work with older versions of␣
 ↪scipy
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage.filters import convolve as conv2
from scipy.ndimage.filters import convolve1d as conv1
from utils import imnoise, gaussian2, bilateral_filter



# Load test images and convert to double precision in the interval [0,1].
im = imread('einsteinpic.jpg') / 255.
im = imresize(im, (256, 256))

# Generate noise
imns = imnoise(im, 'salt & pepper', 0.05) * 1.              # "salt and pepper"␣
 ↪noise
imng = im + 0.05*np.random.randn(im.shape[0],im.shape[1])  # zero-mean Gaussian␣
 ↪noise



# Apply a Gaussian filter with a standard deviation of 2.5
sigmad = 2.5
g, _, _, _, _, _, = gaussian2(sigmad)
```

```
gflt_imns = conv2(imns, g, mode='reflect')
gflt_imng = conv2(imng, g, mode='reflect')


# Instead of directly filtering with g, make a separable implementation
# where you use horizontal and vertical 1D convolutions.
# Store the results again to gflt_imns and gflt_imng, use conv1 instead.
# The result should not change.
# See Szeliski's Book chapter 3.2.1 Separable filtering, numpy.linalg.svd and␣
 ↪scipy.ndimage.filters convolve1d

##--your-code-starts-here--##
## 1d-gaussian

def gaussian1(sigma, N=None):
    if N is None:
        N = 2  * np.maximum(4, np.ceil(6*sigma)) + 1
    k = (N - 1) / 2.
    x = np.arange(-k, k+1)
    g = 1/(np.sqrt(2 * np.pi * sigma**2)) * np.exp(-(x**2) / (2 * sigma ** 2))
    return g

g1d=gaussian1(sigmad)

gflt_imns_x = conv1(imns, g1d, mode="reflect", axis=0)
gflt_imns_xy = conv1(gflt_imns_x, g1d, mode='reflect', axis=1)

gflt_imns_y = conv1(imns, g1d, mode="reflect", axis=1)
gflt_imns_yx = conv1(gflt_imns_y, g1d, mode='reflect', axis=0)

#gflt_imns = conv1(imns, g, mode='reflect')
#gflt_imng = conv1(imng, g, mode='reflect')
##--your-code-ends-here--##


# Median filtering is done by extracting a local patch from the input image
# and calculating its median
def median_filter(img, wsize):
    nrows, ncols = img.shape
    output = np.zeros([nrows, ncols])
    k = (wsize - 1) / 2

    for i in range(nrows):
        for j in range(ncols):
            # Calculate local region limits
            iMin = int(max(i - k, 0))
            iMax = int(min(i + k, nrows - 1))
```

```
            jMin = int(max(j - k, 0))
            jMax = int(min(j + k, ncols - 1))

            # Use the region limits to extract a patch from the image,
            # calculate the median value (e.g using numpy) from the extracted
            # local region and store it to output using correct indexing.

            ##--your-code-starts-here--##

            ##--your-code-ends-here--##

    return output

# Apply median filtering, use neighborhood size 5x5
# Store the results in medflt_imns and medflt_imng
# Use the median_filter function above

##--your-code-starts-here--##

##--your-code-ends-here--##


# Apply bilateral filter to each image with window size 11.
# See section 3.3.1 of Szeliski's book
# Use sigma value 2.5 for the domain kernel and 0.1 for range kernel.

wsize = 11
sigma_d = 2.5
sigma_r = 0.1

bflt_imns = bilateral_filter(imns, wsize, sigma_d, sigma_r)
bflt_imng = bilateral_filter(imng, wsize, sigma_d, sigma_r)

# Display filtering results
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16,8))
ax = axes.ravel()
ax[0].imshow(imns, cmap='gray')
ax[0].set_title("Noisy input image")
ax[1].imshow(gflt_imns, cmap='gray')
ax[1].set_title("Result of gaussian filtering")
#ax[2].imshow(medflt_imns, cmap='gray')
#ax[2].set_title("Result of median filtering")
ax[3].imshow(bflt_imns, cmap='gray')
ax[3].set_title("Result of bilateral filtering")
ax[4].imshow(imng, cmap='gray')
ax[5].imshow(gflt_imng, cmap='gray')
#ax[6].imshow(medflt_imng, cmap='gray')
```

```
ax[7].imshow(bflt_imng, cmap='gray')
plt.suptitle("Filtering results", fontsize=20)
plt.show()
```