Digital Image Processing Project3

By

賴知榆
310512025

CODE (DEMO.PY)

```
import numpy as np
import math
import cv2
import matplotlib.pyplot as plt
from plot_util import plot_and_save, show_hist

def Gaussian_Filter(img_size, lowpass=True, D0=100):
    '''
    Return filtered img and filter H
    '''
    # Create Gaussin Filter: Low Pass Filter in frequenct 
domain
    M, N = img_size[0], img_size[1]
    H = np.zeros((M,N), dtype=np.float32)
    D0 = D0 # Cutoff frequency

    xv, yv = np.meshgrid(range(M), range(N), sparse=False)
    H = np.exp(-((xv - M/2)**2 + (yv - N/2)**2) / (2. * D0 * 
D0))

    if lowpass:
        pass
    else:
        H = 1 - H
    return H

class alpha_trimmed_mean_filter:
    def __init__(self, mask_size=5, alpha=16):
        self.mask_size = mask_size
        self.alpha = alpha

    def padding(self, img):
        pad_size = self.mask_size // 2
        pad_img = np.pad(img, pad_width=pad_size)
        return pad_img

    def filter(self, img):
        img = np.array(img)
        img_size = img.shape
        pad_img = self.padding(img)
        result_img = np.empty(img_size)
        kernel_size = self.mask_size
```

```
        f_i = []
        for i in range(img_size[0]):
            f_j = []
            for j in range(img_size[1]):
                startx = i
                starty = j
                f = pad_img[startx : startx + kernel_size,
starty : starty + kernel_size]
                f_j.append(f.flatten())
            f_i.append(f_j)
        f = np.array(f_i).reshape(img_size[0], img_size[1],
kernel_size**2)
        f = np.sort(f, axis=-1)
        result_img = np.mean(f[:, :, (self.alpha // 2): -
(self.alpha // 2)], axis=-1)

        return result_img

class inverse_filter:
    def __init__(self, degradation_model):
        self.H = degradation_model

    def inverse(self, s):
        s = np.array(s)
        assert s.shape == self.H.shape
        S = np.fft.fft2(s)
        S = np.fft.fftshift(S)
        F = S / self.H
        F = np.fft.ifftshift(F)
        f = np.fft.ifft2(F)
        return np.real(f)

def demo():
    '''
    Assume system: \n
    Assume h is Gaussian blur \n
        f * h = s \n
        s + n = g \n
    Try to find f from g \n
    '''
    img_file = 'Kid2 degraded.tiff'
    img = cv2.imread(img_file, 0)
    g = img
```

```
    D0s = np.linspace(100, 250, 4, dtype=int)
    for D0 in D0s:
        # Denoise (g -> s)
        mask_size = 5
        alpha = 16
        filter = alpha_trimmed_mean_filter(mask_size, alpha)
        s = filter.filter(g)

        # Inverse filter (s -> f)
        H = Gaussian_Filter(s.shape, lowpass=True, D0=D0)
        inv_filter = inverse_filter(degradation_model=H)
        f = inv_filter.inverse(s)

        # Plot results
        name_list = []
        img_list = []
        name_list.append('origin_img'), img_list.append(g)
        name_list.append('denoise_img'), img_list.append(s)
        name_list.append('inv_filter_img'), img_list.append(f)
        plot_and_save(f'Kid_restoration_D0={str(D0)}',
img_list, name_list, dpi=200, plot=False, save=True)
        show_hist(f'Kid_restoration_D0={str(D0)}', img_list,
name_list, plot=False, save=True, density=False)

    # Anaylize noise
    n_hat = g - s
    n_hist = show_hist('Kid_restoration', [n_hat],
filenames=['noise_hist'], plot=False, save=True)[0]
    n_mean = np.sum(n_hist[0] * n_hist[1][:-1])
    n_var = np.sum(n_hist[0] * (n_hist[1][:-1] - n_mean)**2)
    print(f'noise mean: {n_mean}')
    print(f'noise variance: {n_var}')

if __name__ == "__main__":
    demo()
```

CODE (PLOT_UTIL.PY)

```
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt

def plot_and_save(demo_name, gray_imgs, filenames, dpi=200,
plot=True, save=True, cv_plot=False):
    size = len(gray_imgs)
    if save:
        root = 'results'
        result_path = os.path.join(root, demo_name)
        imgs_path = os.path.join(result_path, 'imgs')
        if not os.path.exists(root):
            os.mkdir(root)
        if not os.path.exists(result_path):
            os.mkdir(result_path)
        if not os.path.exists(imgs_path):
            os.mkdir(imgs_path)
        for i in range(size):
            img_path = imgs_path + '/' + filenames[i] + '.png'
            plt.figure()
            plt.title(filenames[i])
            plt.tight_layout()
            plt.imshow(gray_imgs[i], cmap='gray')
            plt.savefig(img_path, bbox_inches='tight',
dpi=dpi)
        plt.close('all')

    if plot and not cv_plot:
        plt.figure(figsize=(16, 8))
        img_in_row = 4
        r = size // img_in_row + (size % img_in_row > 0)
        c = img_in_row
        for i in range(size):
            plt.subplot(r, c, i+1)
            plt.title(filenames[i])
            plt.imshow(gray_imgs[i], cmap='gray')
        plt.suptitle(demo_name + ' images')
        plt.tight_layout()
        plt.show()
        plt.close('all')
```

```
    if cv_plot:
        c = size
        p = 0
        concated_img = []
        for i in range(size):
            cv2.putText(gray_imgs[i], filenames[i], (40, 40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (128, 128, 128), 1, cv2.LINE_AA)
        blanks = 4 - size % 4
        for _ in range(blanks):
            gray_imgs.append(np.zeros(gray_imgs[0].shape,
dtype='uint8'))
            cv2.putText(gray_imgs[-1], 'BLANK IMG',
(gray_imgs[0].shape[1]//2-80, gray_imgs[0].shape[0]//2),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 1, cv2.LINE_AA)
        while c > 0:
            concated_img.append(cv2.hconcat(gray_imgs[p:p+4]))
            c -= 4
            p += 4
        final_img = cv2.vconcat(concated_img)
        final_img = cv2.resize(final_img, (1280, 720))
        cv2.imshow('imgs', final_img)
        cv2.moveWindow('imgs', 0, 0)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

def show_hist(demo_name, gray_imgs, filenames, plot=True,
save=True, density=True):
    size = len(gray_imgs)
    if save:
        root = 'results'
        result_path = os.path.join(root, demo_name)
        hist_path = os.path.join(result_path, 'hist')
        if not os.path.exists(root):
            os.mkdir(root)
        if not os.path.exists(result_path):
            os.mkdir(result_path)
        if not os.path.exists(hist_path):
            os.mkdir(hist_path)
        for i in range(size):
            img_path = hist_path + '/' + filenames[i] + '.png'
            plt.figure()
            plt.title(filenames[i])
            plt.hist(gray_imgs[i].ravel(), 256, [0,256])
            plt.tight_layout()
```

```
            plt.savefig(img_path, bbox_inches='tight')
        plt.close('all')

    if plot:
        plt.figure(figsize=(16, 9))
        r = size // 4 + (size % 4 > 0)
        c = 4
        for i in range(size):
            plt.subplot(r, c, i+1)
            plt.title(filenames[i])
            plt.hist(gray_imgs[i].ravel(), 256, [0,256])
        plt.suptitle(demo_name + ' histograms')
        plt.tight_layout()
        plt.show()
        plt.close('all')

    hists = []
    for i in range(size):
        hists.append(np.histogram(gray_imgs[i].ravel(), 256,
[0, 256], density=density))
    return hists
```

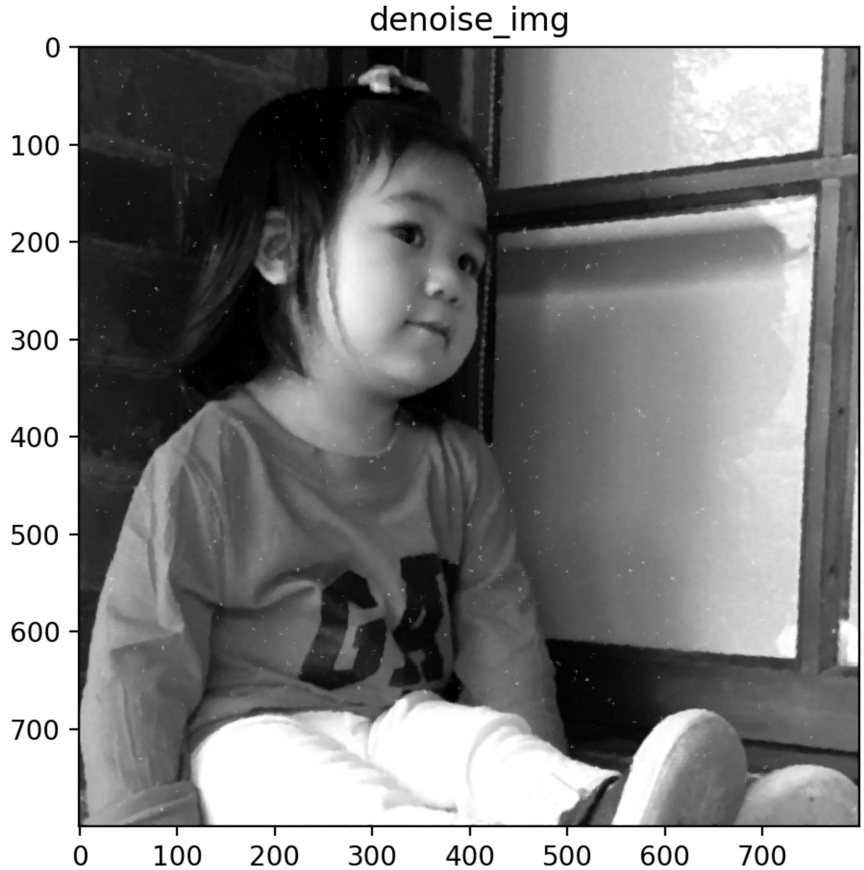
RESULTS OF NOISE MODEL AND MODEL PARAMETERS

Model: pepper and salt noise model

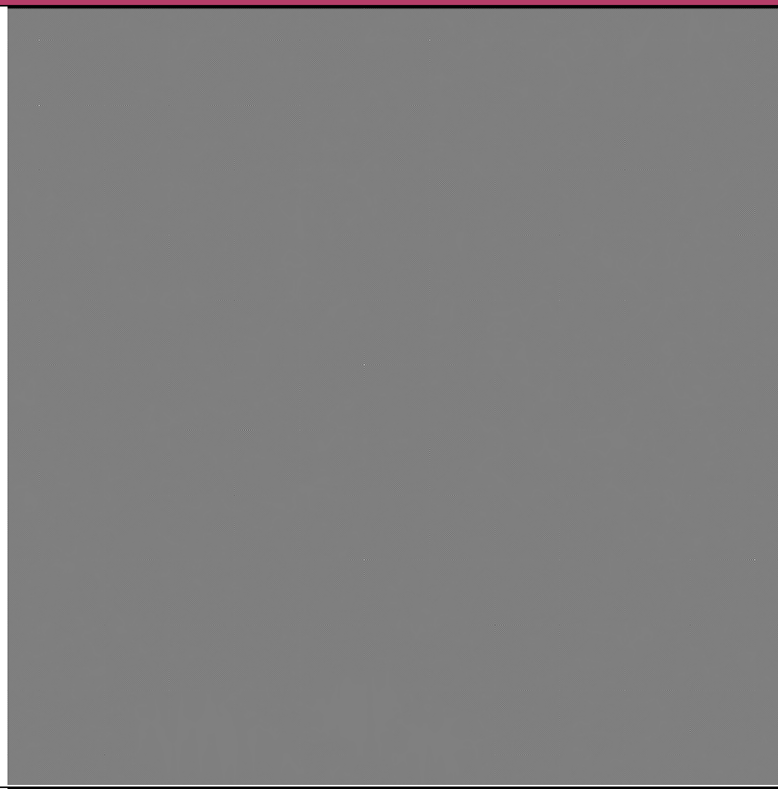
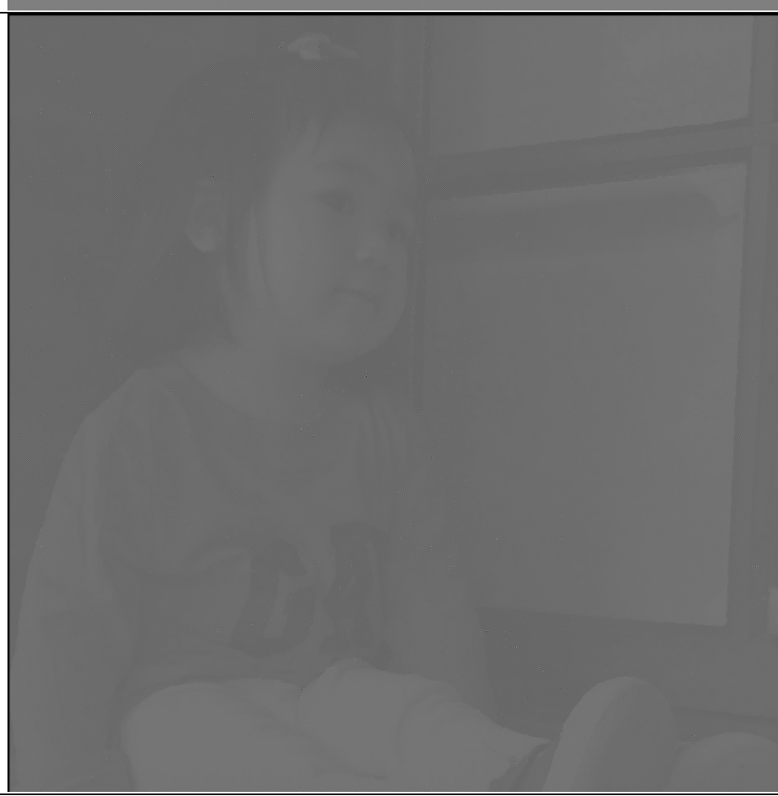
Noise mean: 43.815218160831094

Noise variance: 5811.754854517623

DE-NOISED IMAGE BY ALPHA-TRIMMED MEAN FILTER USING 5X5 MASK AND ALPHA= 16



IMAGES RECONSTRUCTED BY ESTIMATED INVERSE FILTER

D0 = 100	
D0 = 150	

D0 = 200	
D0 = 250	

DESCRIBE THE PARAMETERS USED FOR DECONVOLUTION, INCLUDING D0 OF THE INVERSE FILTER AS WELL AS THE ORDER N AND CUTOFF FREQUENCY OF BUTTERWORTH LP

Before we reconstructed the images, we assumed that the degradation model is Gaussian model. When we use inverse filter (inverse Gaussian filter) to reconstruct the images, we will get images closer to the original images if we guess more accurate about the degradation model.

Here we assume the degradation model is Gaussian, so the parameter is D0. When the D0 is same as the degradation model (Gaussian model), we can get original image.

The degraded image looks not heavily blurred, so the D0 should not be too small. If it does, we won't get a good result, like previous part D0=100, we get a bad result.