Digital Image Processing Project 3

Name: 彭晨益

Student ID: 310512054

(a) Source Code

```
import os
import argparse
import cv2
import numpy as np
import numpy.fft as fft
import matplotlib.pyplot as plt
from numpy.fft import ifft2, ifft, fft2, fftshift, ifftshift


def check_folder(path):
    if not os.path.exists(path):
        os.makedirs(path)
    return path


def fft_2d(img, n_x, n_y):
    """


    Parameters
    ------------
    img: input image
    n_x: DFT dimension of x
    n_y: DFT dimension of y

    Returns
    -----------
    return 2d fft result shape: (n_x, n_y)
    """
    f = np.fft.fft2(img,(n_x, n_y))
    fshift = np.fft.fftshift(f)

    return fshift
```

```python
def ifft_2d(img):
    """
    Parameters
    ------------
    img: input image

    Returns
    ------------
    return 2d ifft result shape: (n_x, n_y)
    """
    fshift = np.fft.ifftshift(img)
    result = np.fft.ifft(fshift)
    return result


def calMeanVar(img):

    hist, pixel_value = np.histogram(img, 256, [0,256], density=True)

    mean = np.sum(hist * pixel_value[:-1])
    var = np.sum((pixel_value[:-1] - mean)**2 * hist)

    return mean, var

def generate_GLPF(shape=(800, 800), d_0=100):
    """
    generate 2-D Gaussian lowpass filter


    Parameters
    ---------------
    shape: dimension of the gaussian lowpass filter (M, N)
    d_0: cutoff frequency


    Returns
    ---------------
    Gaussian lowpass filter shape: (M, N) with cutoff frequency d_0
```

```
    """
    m, n = shape
    x = np.linspace(0, m, num=m)
    y = np.linspace(0, n, num=n)
    xv, yv = np.meshgrid(x, y, sparse=False)
    h = np.exp(-((xv - m/2)*(xv - m/2) + (yv - n/2)*(yv - n/2)) / (2. *
d_0 * d_0) )

    return h

def save_img(src, title, save_path):
    global SAVE
    plt.figure()
    plt.imshow(np.real(src), cmap="gray")
    plt.title(title)
    plt.axis("off")

    if (os.path.exists(save_path + title + ".png")):
        return
    if SAVE:
        check_folder((save_path))
        plt.savefig(save_path + title + ".png", dpi=200,
bbox_inches='tight')

def save_hist(src, title, save_path):
    global SAVE
    plt.figure()
    plt.hist(np.real(src).ravel(), 256, [0, 256])
    plt.title(title)
    plt.xlabel("Brightness")

    if (os.path.exists(save_path + title + "-histogram.png")):
        return
    if SAVE:
        check_folder((save_path))
        plt.savefig(save_path + title + "-histogram.png", dpi=200,
bbox_inches='tight')
```

```python
def AlphaTrimFilter(img, n, alpha):

    """
    Implement the Alpha-Trimmed mean filter

    Params
    -----
    img: img source
    n: filter size nxn
    alpha: numbers of trimmed elements
    """

    v = int((n - 1) / 2)

    # add zero padding outside the image to prevent NaN values during the trimming
    pad_img = np.pad(img, pad_width=v)

    vector_i = []
    for i in range(0, img.shape[0]):
        vector_j = []
        for j in range(0, img.shape[1]):
            # Slice the sub-region of the image
            block = pad_img[i:i+n, j:j+n]

            # Flatten the sub-region
            vector_j.append(block.flatten())

        vector_i.append(vector_j)
    # Do the sort() in once
    final = np.array(vector_i).reshape(img.shape[0], img.shape[1], n**2)
    final = np.sort(final, axis=-1)
    result_img = np.mean(final[:,:, (alpha // 2): -(alpha // 2)], axis=-1)

    return result_img
```

```python
def inverse_filter(image, filter):
    """
    Do deconvolution to the image with given filter

    Params
    ------
    image: image source
    filter: filter in frequency domain
    """
    return ifft2(ifftshift((fftshift(fft2(image)) * (1 / filter))))



def main():

    # pipeline
    # s(x, y) + n(x, y) = g(x, y)
    # g(x, y) - n(x, y) = s(x, y)
    # f(x, y) convolution with h(x, y) = s(x, y)
    # f(x, y) = s(x, y) / h(x, y)

    image = cv2.imread('Kid2_degraded.tiff', 0)



    denoise_img = AlphaTrimFilter(image, 5, 16)

    glpf = generate_GLPF((image.shape[0],image.shape[1]), 250)

    recover_result = inverse_filter(denoise_img, glpf)

    noise =  image -(denoise_img)
    mean, var = calMeanVar(noise)
    print("========Parameters of Noise Model==========")
    print(f'mean = {mean:.4f}, variance = {var:.4f}')

    save_img(image, "Original image", 'results/')
    save_hist(image, 'Original_image', 'results/')
    save_img(denoise_img, 'Denoise image', 'results/')
    save_hist(denoise_img, "denoise_image", 'results/')
```

```
    save_img(recover_result, "Deconvolution image", 'results/')
    save_hist(recover_result, "Deconvolution result_histogram",
'results/')

    save_hist(noise, "Noise", 'results/')

    plt.show()
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--save", help="Whether to save the image",
action="store_true")
    args = parser.parse_args()
    SAVE = args.save
    main()
```

Result of noise model and model parameters

Model: pepper and salt noise model

```
========Parameters of Noise Model=========
mean = 43.8152, variance = 5811.7549
```

Denoised image by alpha-trimmed mean filter using 5x5 mask and alpha=16

Denoise image

Image reconstructed by estimated inverse filter

D0 =100	D0=100 
D0 =150	D0=150 

D0 =200	D0=200 
D0 =250	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 LPF

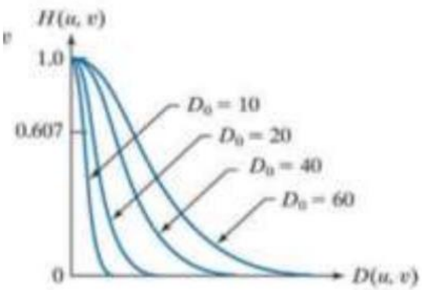
Ans:

We first assume the degradation model is Gaussian model. Then if we apply the Inverse Gaussian filter to the image, we would get the good result, which means the image would look more close to the original one.

D0 determined the variance of the gaussian filter, large D0 means the filter would cover more pixels, which means the image would me more clear. Instead if we choose small D0 then the blur effect would be more heavily.

In here, since the original degradation image is not that blurred, so if we choose the small D0 we would get bad result, we should choose the large D0 to get the better result like the D0=250’s result.

$$H(u,v)=e^{-\frac{D^2(u,v)}{2\sigma^2}} \xrightarrow{\sigma=D_0} H(u,v)=e^{-\frac{D^2(u,v)}{2D_0^2}}$$



Cross sections for D_0=10, 20, 40, 100