Digital Image Processing
Project2

By

賴知榆
310512025

CODE (DEMO.PY)

```
from turtle import left
import numpy as np
import cv2
import os, argparse, time
import csv
from matplotlib import pyplot as plt

def get_parser():
    parser = argparse.ArgumentParser(description='Filters
demo')
    parser.add_argument('--no_save_img', dest='save_img',
action='store_false', default=True)
    parser.add_argument('--no_show_img', dest='show_img',
action='store_false', default=True)
    return parser

# Function to map each intensity level to output intensity
level.
def pixelVal(pix, r1, s1, r2, s2):
    if (0 <= pix and pix <= r1):
        if r1 == 0:
            return s1
        else:
            return (s1 / r1)*pix
    elif (r1 < pix and pix <= r2):
        return ((s2 - s1)/(r2 - r1)) * (pix - r1) + s1
    else:
        if r2 == 255:
            return s2
        else:
            return ((255 - s2)/(255 - r2)) * (pix - r2) + s2

# Map (rmin, rmax) to (0, 255)
def prep_plot(img):
    # Find min and max value of img pixels
    rmin = np.amin(img)
    rmax = np.amax(img)
    # Define parameters.
    r1 = rmin
    s1 = 0
    r2 = rmax
    s2 = 255
```

```
    # Vectorize the function to apply it to each value in the
Numpy array.
    pixelVal_vec = np.vectorize(pixelVal)
    # Apply contrast stretching.
    contrast_stretched_img = pixelVal_vec(img, r1, s1, r2, s2)
    return np.array(contrast_stretched_img, dtype = 'uint8')

def plot_and_save(demo_name, gray_imgs, filenames, dpi=150,
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 Gaussian_Filter(img, lowpass=True, D0=100):
    '''
    Return filtered img and filter H
    '''
    # Create Gaussin Filter: Low Pass Filter in frequenct
domain
    M, N = img.shape
    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 np.multiply(img, H), H
```

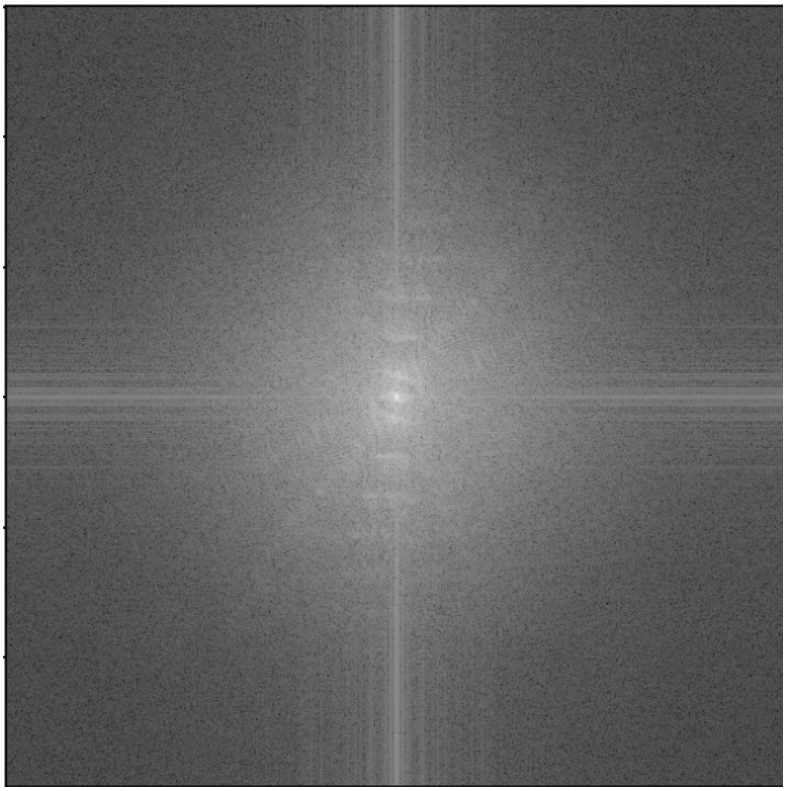
```
def padding_img(img, times):
    M, N = img.shape
    zeros_array = np.zeros((M, N * (times-1)),
dtype=np.float32)
    img = np.hstack([img, zeros_array])
    zeros_array = np.zeros((M * (times-1), N * (times)),
dtype=np.float32)
    img = np.vstack([img, zeros_array])
    return img

def create_minus_xy_array(img):
    M, N = img.shape
    xv, yv = np.meshgrid(range(M), range(N), sparse=False)
    ret = ((xv + yv) % 2) * 2 - 1
    return ret

def find_most_freq(F, num):
    '''
    Find the Top most appearing frequencies
    Rank `num` of them and return rank list
    '''
    M, N = F.shape
    left_half_F = np.abs(F[:M, :N//2])
    pixels = left_half_F.ravel()    # Flatten

    indices =  np.argpartition(pixels, -2)[-num:]
    indices = np.vstack(np.unravel_index(indices,
left_half_F.shape)).T
    return indices

def main(parser):
    args = parser.parse_args()

    demo_names = ['kid', 'fruit']
    files_type = ['.tif', '.tif']

    for demo_name, file_type in zip(demo_names, files_type):
        img_list = []
        img_name = []

        file = demo_name + file_type
        print(f'Running {file}')
```

```
        # origin img
        img = cv2.imread(file, 0)
        origin_size = img.shape
        img_list.append(img)
        img_name.append(f'(a) origin {demo_name}')
        # Padding img to 2M, 2N size
        img = padding_img(img, times=2)
        img_list.append(img)
        img_name.append(f'(b) padding {demo_name}')
        # img * (-1)^(x+y)
        ret = create_minus_xy_array(img)
        img = np.multiply(img, ret)
        img_list.append(img)
        img_name.append(f'(c) Multiply by (-1)^(x+y)')
        # FFT
        ft_img = np.fft.fft2(img)
        img_list.append(20*np.log(np.abs(ft_img)))
        img_name.append(f'(d) {demo_name} magnitude spectra')
        # Gaussian filtering
        LP_ft_img, filter1 = Gaussian_Filter(ft_img,
lowpass=True, D0=100*2)
        HP_ft_img, filter2 = Gaussian_Filter(ft_img,
lowpass=False, D0=100*2)
        img_list.append(filter1)
        img_list.append(filter2)
        img_name.append('(e) Gaussian Lowpass filter')
        img_name.append('(e) Gaussian Highpass filter')
        img_list.append((np.abs(LP_ft_img)))
        img_list.append((np.abs(HP_ft_img)))
        img_name.append(f'(f) {demo_name} Gaussian Lowpass
filtered')
        img_name.append(f'(f) {demo_name} Gaussian Highpass
filtered')
        # Inverse FFT
        LP_ft_img = np.multiply(LP_ft_img, ret)
        HP_ft_img = np.multiply(HP_ft_img, ret)
        LP_ft_img = np.fft.ifft2(LP_ft_img)
        HP_ft_img = np.fft.ifft2(HP_ft_img)
        LP_img = np.abs(np.fft.ifftshift(LP_ft_img))
        HP_img = np.abs(np.fft.ifftshift(HP_ft_img))
        img_list.append(LP_img)
        img_list.append(HP_img)
        img_name.append(f'(g) {demo_name} Lowpass filtered
img')
```

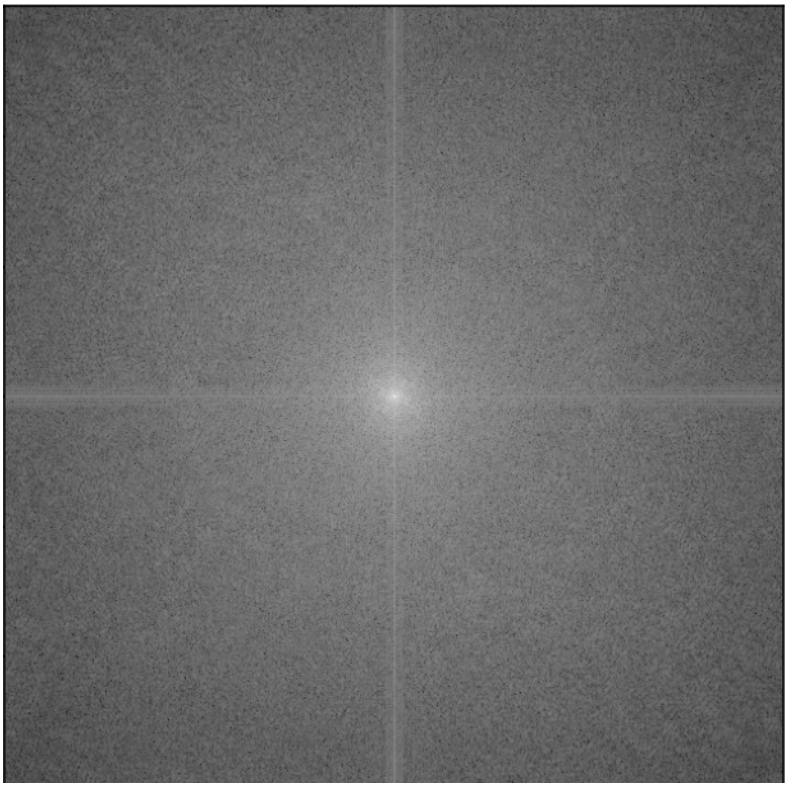
```
        img_name.append(f'(g) {demo_name} Highpass filtered
img')
        # Crop to original image size
        LP_img = LP_img[:origin_size[0], :origin_size[1]]
        HP_img = HP_img[:origin_size[0], :origin_size[1]]
        img_list.append(LP_img)
        img_list.append(HP_img)
        img_name.append(f'(h) {demo_name} Lowpass cropped
img')
        img_name.append(f'(h) {demo_name} Highpass cropped
img')

        most_list = find_most_freq(ft_img, 25)
        root = 'results'
        result_path = os.path.join(root, demo_name)
        if not os.path.exists(root):
            os.mkdir(root)
        if not os.path.exists(result_path):
            os.mkdir(result_path)
        with open(os.path.join(result_path,
'most_freq(u,v).csv'), 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(['u', 'v'])
            writer.writerows(most_list)

        plot_and_save(
            demo_name,
            img_list,
            img_name,
            dpi=150,
            plot=args.show_img,
            save=args.save_img,
            cv_plot=False,
        )

if __name__ == "__main__":
    main(get_parser())
```

FOURIER MAGNITUDE SPECTRA (IN LOG SCALE) OF KID AND FRUIT IMAGE (600X600 DFT)

Kid:

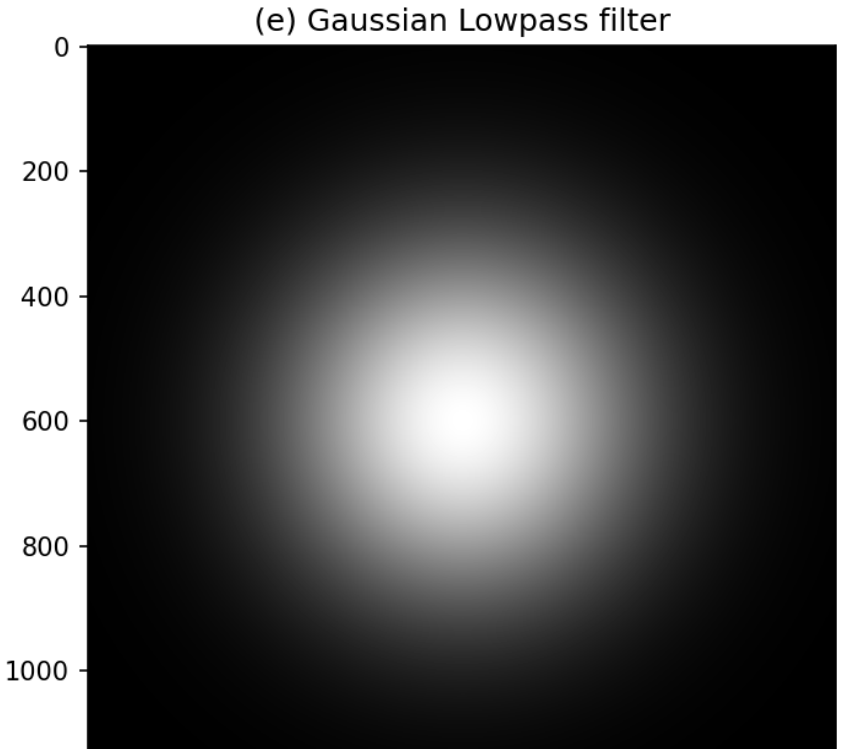


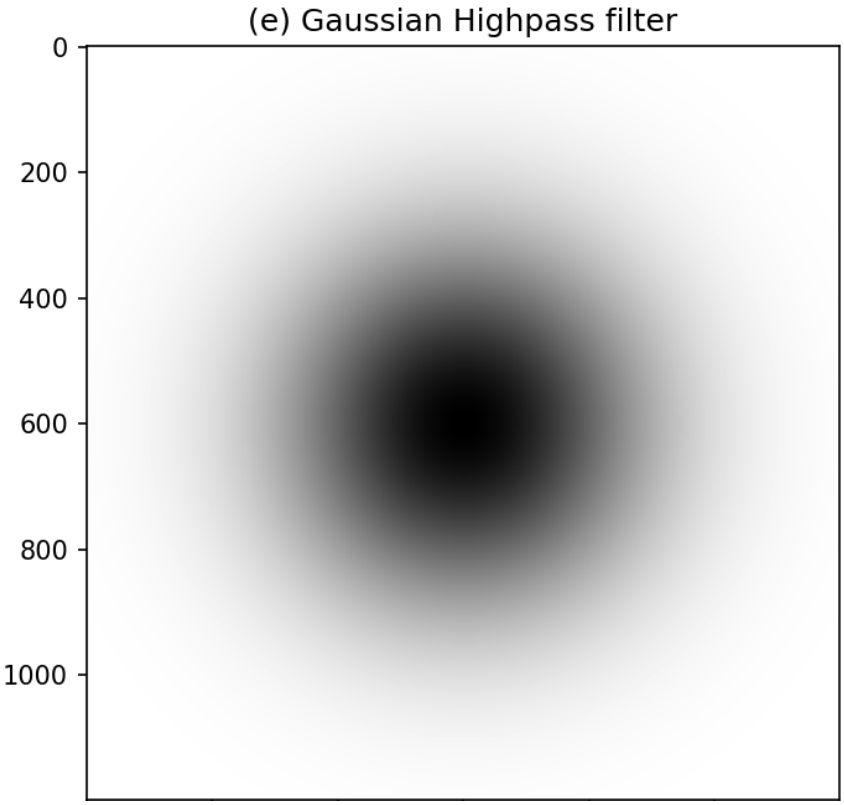
Fruit:



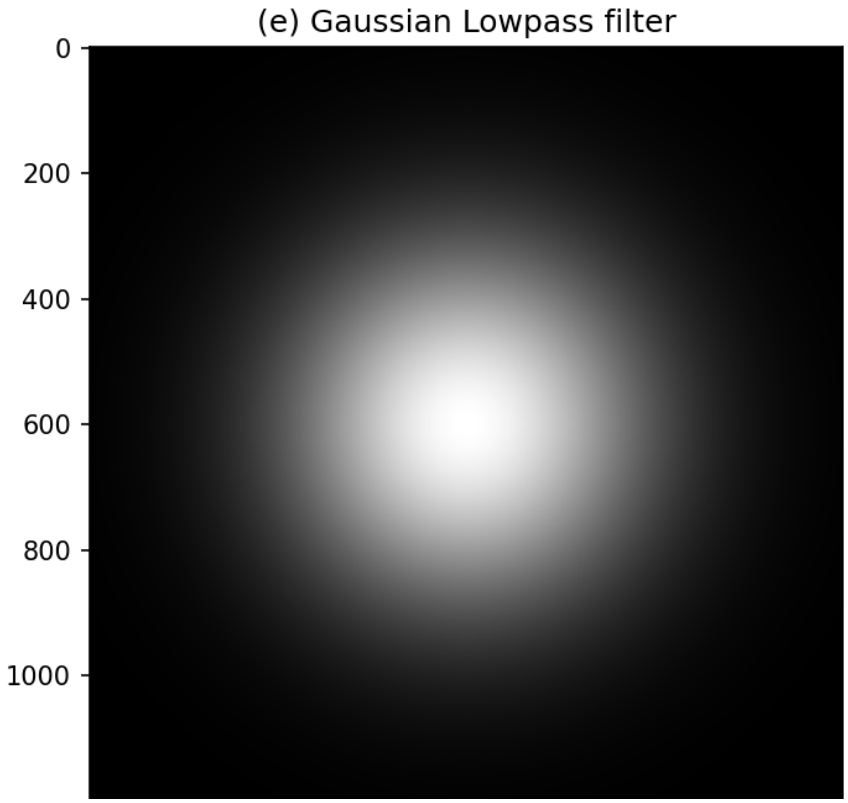
MAGNITUDE RESPONSES OF GAUSSIAN LPF AND HPF (1200X1200 DFT)

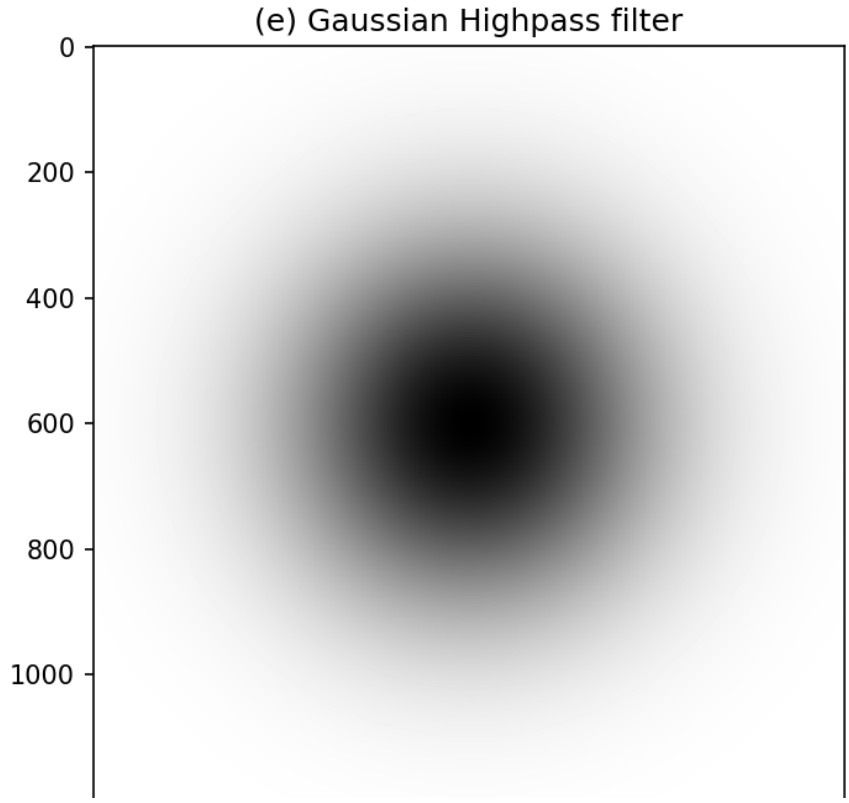
Kid:



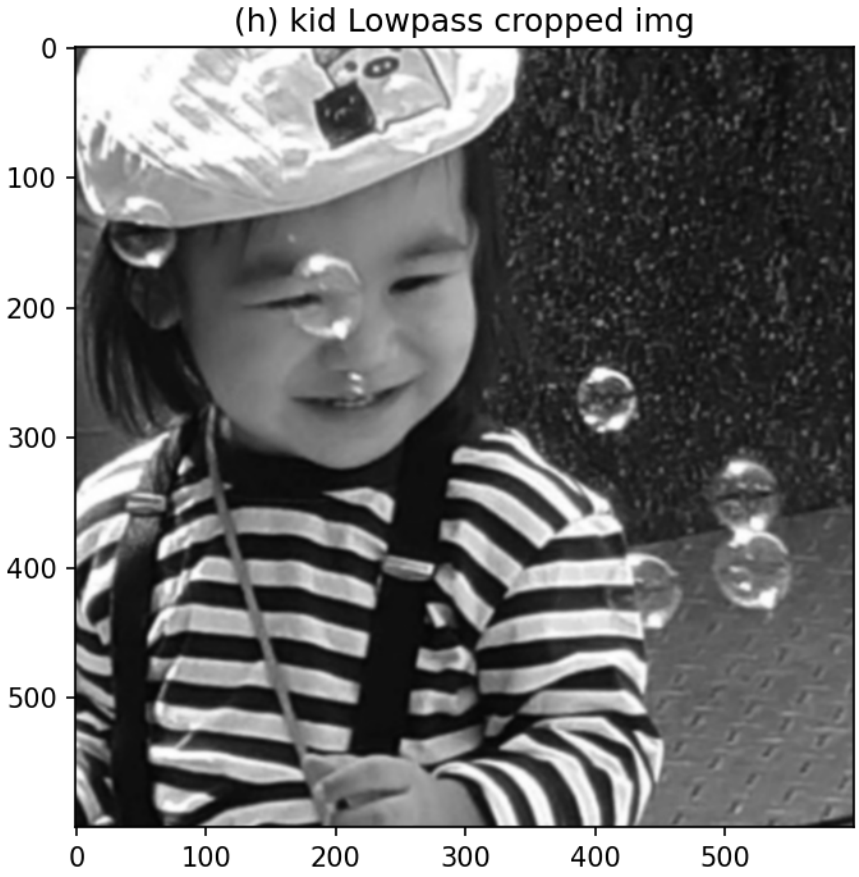


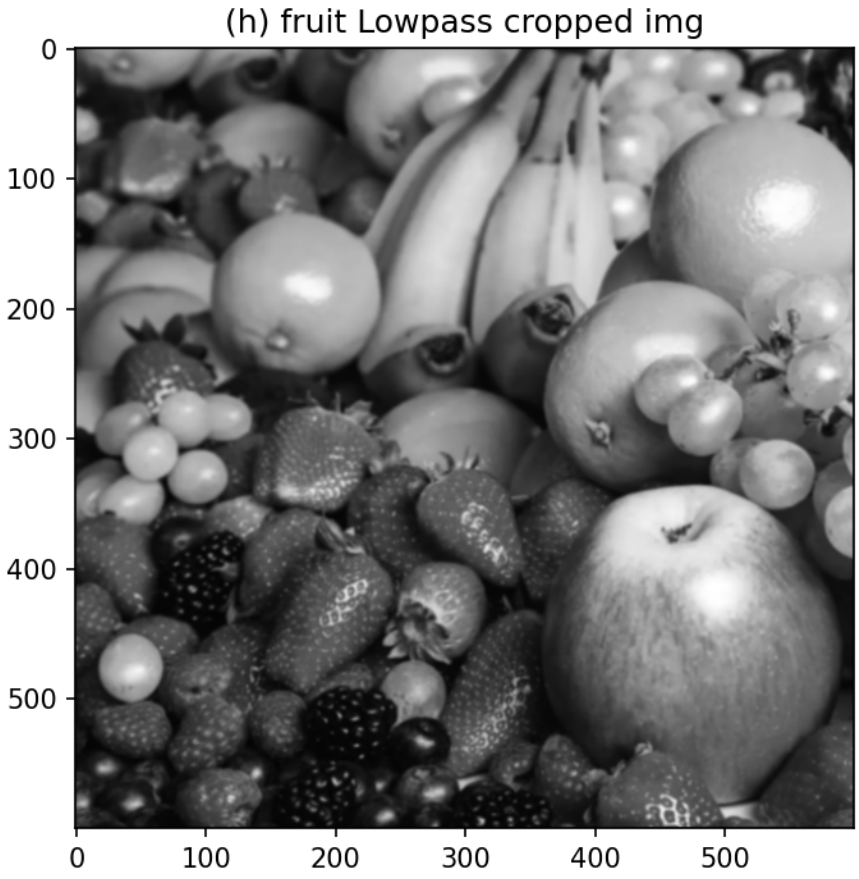
Fruit:

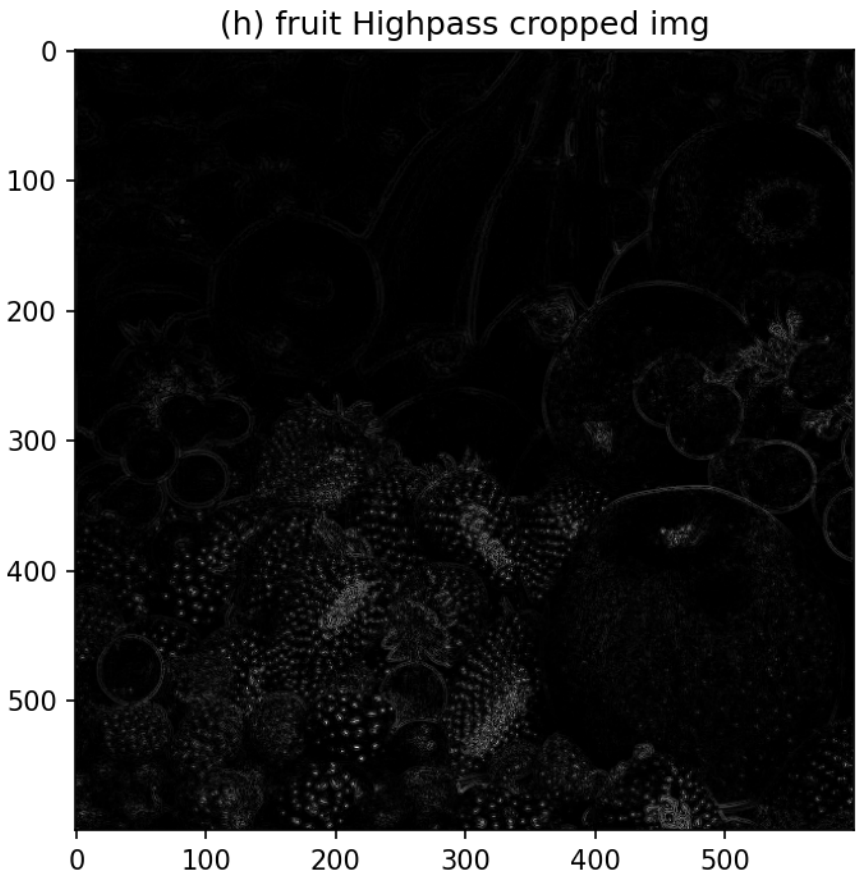

(e) Gaussian Lowpass filter
0
200
400
600
800
1000

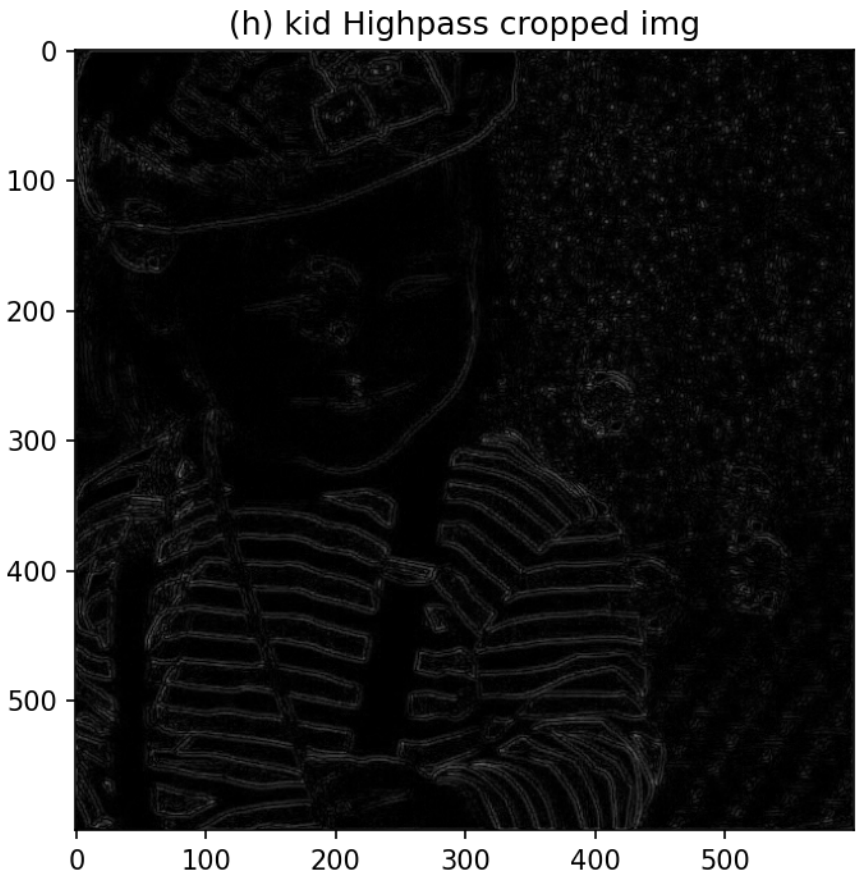

(e) Gaussian Highpass filter
0
200
400
600
800
1000

FOUR OUTPUT IMAGES




(h) fruit Lowpass cropped img
0
100
200
300
400
500
0
100
200
300
400
500


(h) fruit Highpass cropped img
0
100
200
300
400
500
0
100
200
300
400
500


(h) kid Highpass cropped img
0
100
200
300
400
500
0
100
200
300
400
500

TABLES OF TOP 25 DFT FREQUENCIES (U,V) OF B) IN THE LEFT HALF FREQUENCY REGION (0<=U<=M-1, 0<=V<=N/2-1)

Kid:

1	u	v
2	301	296
3	316	298
4	300	298
5	300	295
6	300	294
7	299	298
8	299	297
9	299	294
10	302	299
11	298	298
12	298	297
13	298	294
14	298	292
15	297	299
16	297	296
17	296	298
18	296	296
19	288	299
20	315	297
21	315	298
22	316	296
23	298	299
24	299	299
25	300	299
26	301	299

Fruit:

1	u	v
2	295	291
3	296	298
4	296	297
5	296	296
6	295	296
7	296	293
8	296	292
9	299	298
10	302	297
11	295	299
12	304	299
13	297	298
14	298	299
15	300	295
16	303	299
17	302	299
18	296	294
19	301	294
20	299	299
21	296	299
22	303	297
23	300	298
24	300	297
25	301	297
26	300	299