CSE404 Project 1

201811118 이 구

1. Discussion

이미지의 크기가 N x M, spatial domain에서 사용한 filter의 크기가 n x m 이라 할 때, filtering를 적용하기 전에는 mirror padding을 적용하여 (N+n-1) x (M+m-1) 크기로 만든 후 계산하였고, frequency domain에서의 filtering을 적용하기 전에는 wraparound error를 막기 위해 zero padding을 적용하여 (2N x 2M) 크기로 만든 후 계산하였다. 또, 주어진 이미지는 모두 RGB image인 반면, 수업시간에 배운 모든 method들은 grayscale image를 대상으로 한 것이었으므로, Rec. 601에 따라 아래의 수식을 이용하여 RGB image를 grayscale image로 변환하였다.

$$Y = 0.299R + 0.587G + 0.114B$$

마지막으로, sobel filter의 일반적인 사용 방식은 이미지에 x, y 방향의 filter를 적용한 결과 각각의 제곱을 더한 후 root를 취하거나, x, y 방향의 filter를 적용한 결과 각각에 절댓값을 취한 후 더해주어 computational cost를 줄이는 식으로 사용한다. 하지만, 실험적으로 x, y 각각의 결과에 minmax normalization을 취한 후 더해주면 더욱 깔끔하게 edge 영역이 검출되는 것을 확인하였고, 아래에서 기술한 모든 sobel 연산은 이 방식으로 수행되었다.

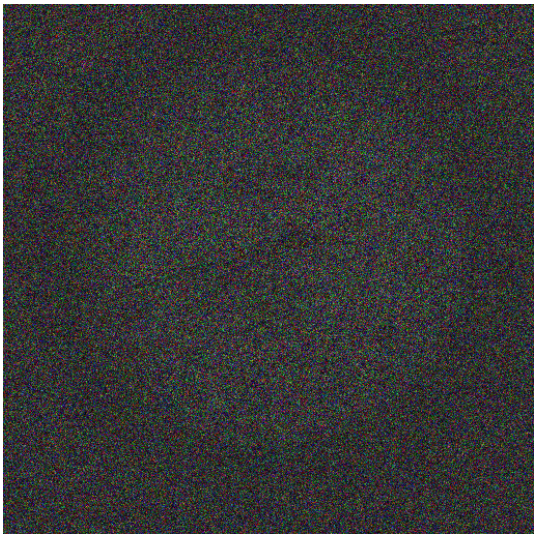
각 이미지 별로 수행한 연산 과정은 아래와 같다.

1) clock_noise1.png : 이미지에 전체적으로 심한 noise가 존재하였으므로, filter size가 7인 average filter를 연속으로 두 번 적용하였다. 이후, 전체적으로 contrast가 낮았으므로 histogram equalization을 적용하였다. 이후, 여전히 남아있는 noise들을 없애기 위해, frequency domain에서 lowpass filter를 적용하였다. 사용한 lowpass filter는 Butterworth lowpass filter이고, cutoff frequency는 100, order는 3으로 설정하여 사용하였다. 이후, 다시 한 번 filter size가 3인 average filter를 한 번 적용하여 최종 reduced noise image를 얻었다. 이후, sobel filter를 적용하여 highlighted edge image를 얻었다.
2) cars1_noise.png : 이미지의 contrast가 전체적으로 낮았으므로, histogram equalization을 적용하였다. 이후, 남아있는 noise들을 제거하기 위해, filter size가 3인 average filter를 연속으로 3번 적용하여, 최종적인 reduced noise image를 얻었다. 이후, sobel filter를 적용하여 highlighted edge image를 계산하였다.
3) cars2_noise.png: 이미지에 salt-and-pepper noise가 눈에 띄게 존재하였으므로, median filter를 적용하였다. 이후, 이미지의 전체적인 contrast가 높았으므로 histogram equalization을 적용하였다. 이후, 남아있는 noise들을 제거하기 위해, frequency domain에서 lowpass filter를 적용하였다. 사용한 lowpass filter는 Butterworth lowpass filter이고, cutoff frequency는 500, order는 3으로 설정한 후 적용하여 최종적인 reduced noise image를 얻었다. 이후, sobel filter를 적용하여 highlighted edge image를 계산하였다.

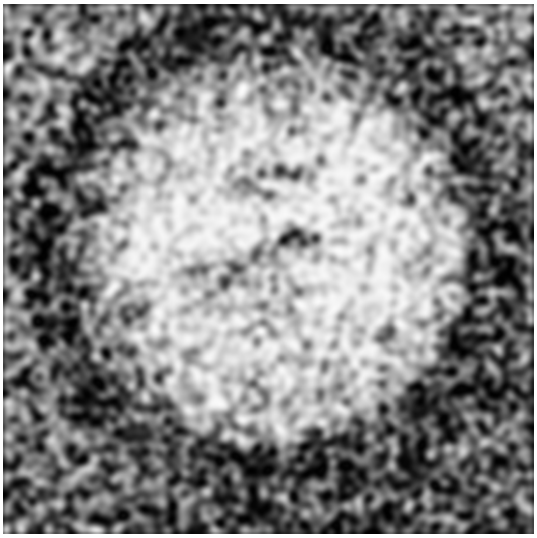
2. Result

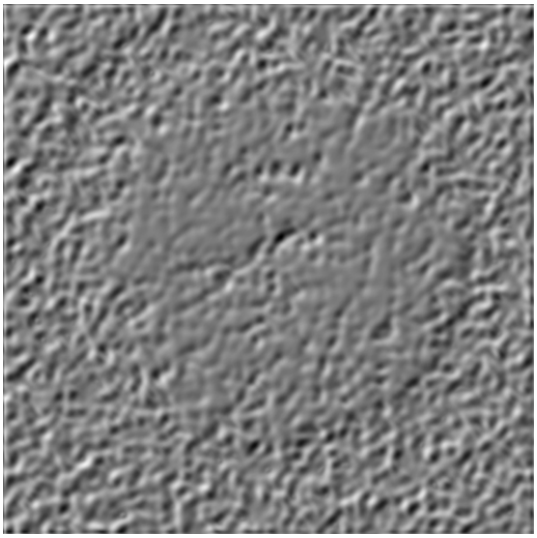
1) clock_noise1.png

- input



- reduced noise image (left) / highlighted edge image (right)





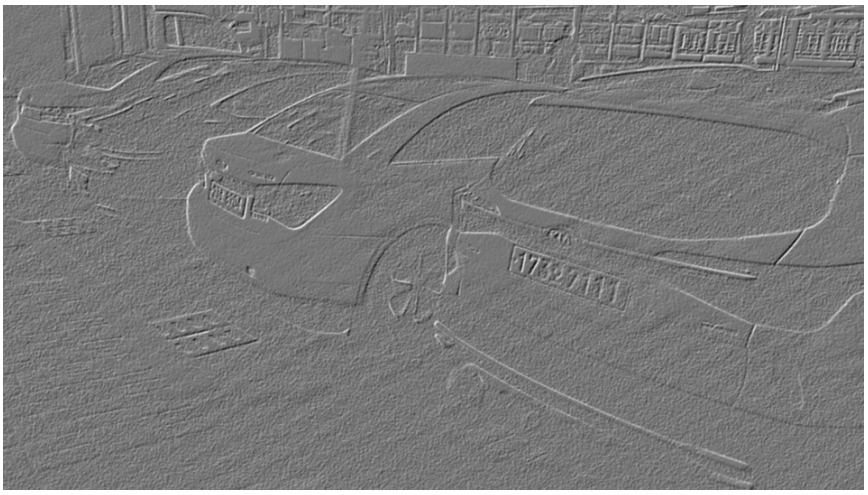
전체적인 noise 의 정도가 심해 완벽하게 noise 를 제거할 수는 없었지만, 시침, 분침이 좌측 하단을 가리키는 것은 분명하게 식별할 수 있다.

2) cars1_noise.png

- input

- reduced noise image

- highlighted edge image



3) cars2_noise.png

- input

- reduced noise image

- highlighted edge image

3. Code

각 이미지 별로 사용한 코드는 아래와 같다.

1) clock_noise1.png

```
"""
2024 Spring, Introduction to Computer Vision.
Project 1 - clock_noise.png
"""

import numpy as np
import os
from PIL import Image as pilimg
from matplotlib import pyplot as plt


def readimage(img_name):
    global row, col
    im = pilimg.open(img_name)
    cimg = np.array(im)
    col, row = im.size
    return cimg, row, col


def saveimage(cimg, name):
    cimg=np.uint8(cimg)
    im=pilimg.fromarray(cimg)
    im.save(name)
    return im

def medianfiltering(img2d, ms):
    row, col = img2d.shape
    buff2d = np.full((row, col), 0)
    hs = ms//2
    for i in range(hs, row-hs):
        for j in range(hs, col-hs):
            temp=img2d[i-hs:i+hs+1, j-hs:j+hs+1].flatten()
            temp=np.sort(temp)
            buff2d[i, j] = temp[len(temp)//2]
    return buff2d[hs:-hs, hs:-hs]

def minmax_normalization(img, min, max):
    return 255.0 * (img-min) / (max-min)

def z_standard_normalization(img):
    # z_std = 1 keeping about 70%
    # z_std = 2 keeping about 95%
```

```
    z_std = 1
    mean = np.mean(img)
    std = np.std(img)
    z_score = np.clip((img-mean) / std, -z_std, z_std)
    return minmax_normalization(z_score, -z_std, z_std)

def histogram_equalization(img):
    histogram = [0 for _ in range(256)]

    for i in range(row):
        for j in range(col):
            histogram[img[i,j]] += 1

    pdf = np.array(histogram) / (row*col)
    cdf = np.cumsum(pdf)
    result = np.zeros_like(img)

    for i in range(row):
        for j in range(col):
            result[i,j] = 255 * cdf[img[i, j]]

    return np.uint8(result)


def imagefft(img, center):
    img_fft = np.fft.fft2(img)
    if center:
        img_fft = np.fft.fftshift(img_fft)
    return img_fft


def imageifft(img_fft, center=True):
    if center:
        img_fft = np.fft.ifftshift(img_fft)
    img = np.fft.ifft2(img_fft)
    return img


def make_ideal_LPF_mask_f(img_fft, d0):
    row, col = np.shape(img_fft)
    mask = np.full((row,col), 0.0).astype(np.float32)
    center_row = row // 2
    center_col = col // 2

    for i in range (row):
        for j in range(col):
            dist = np.sqrt((center_row-i)**2+(center_col-j)**2)
            if dist < d0:
```

```
                mask[i, j] = 1
            else:
                mask[i, j]= 0
    return mask


def make_gaussian_LPF_mask_f(img_fft, std):
    row, col = np.shape(img_fft)
    mask = np.full((row,col), 0.0).astype(np.float32)
    center_row = row // 2
    center_col = col // 2

    for i in range (row):
        for j in range(col):
            dist = np.sqrt((center_row-i)**2+(center_col-j)**2)
            mask[i, j] = np.exp(- (dist ** 2) / (2 * (std**2)))
    return mask


def make_butterworth_LPF_mask_f(img_fft, cutoff, n):
    row, col = np.shape(img_fft)
    mask = np.full((row,col), 0.0).astype(np.float32)
    center_row = row // 2
    center_col = col // 2

    for i in range (row):
        for j in range(col):
            dist = np.sqrt((center_row-i)**2+(center_col-j)**2)
            mask[i, j] = 1 / (1 + (dist / cutoff)**(2*n))
    return mask


def average_filtering(img2d, msize):
    row, col = img2d.shape
    buff2d = np.full((img2d.shape[0], img2d.shape[1]), 0)
    mask = np.ones((msize, msize)) / (msize*msize)
    hs = msize // 2

    for i in range(hs, row-hs):
        for j in range(hs, col-hs):
            buff2d[i, j] = np.sum(img2d[i-hs:i+hs+1, j-hs:j+hs+1] * mask)
    return buff2d[hs:-hs, hs:-hs]


def maskfiltering(img2d, mask):
    row, col = img2d.shape
    buff2d = np.full((row, col), 0)
    mr, mc = mask.shape[0], mask.shape[1]
```

```
    hs = mr // 2

    for i in range(hs, row-hs):
        for j in range(hs, col-hs):
            buff2d[i, j] = np.sum(img2d[i-hs:i+hs+1, j-hs:j+hs+1] * mask)
    return buff2d[hs:-hs, hs:-hs]


if __name__=="__main__":
    img, row, col = readimage("input/clock_noise1.png")

    # make grayscale img
    rcimg = img[:,:,0] # r-channel img
    gcimg = img[:,:,1] # g-channel img
    bcimg = img[:,:,2] # b-channel img

    gimg = np.uint8(0.299 * rcimg + 0.587 * gcimg + 0.114 * bcimg) # Convert a color image
to a grayscale image
    saveimage(gimg, "output/clock/0_grayscale.png")

    img = gimg

    # applying padding and average filtering in spatial domain
    average_filter_size=7
    padded_img = np.lib.pad(img, (((average_filter_size-1)//2, (average_filter_size-1)//2),
                                  ((average_filter_size-1)//2, (average_filter_size-1)//2)),
'reflect')
    img = average_filtering(padded_img, msize=average_filter_size)
    padded_img = np.lib.pad(img, (((average_filter_size-1)//2, (average_filter_size-1)//2),
                                  ((average_filter_size-1)//2, (average_filter_size-1)//2)),
'reflect')
    img = average_filtering(padded_img, msize=average_filter_size)
    saveimage(img, f"output/clock/1_average_filter_{average_filter_size}.png")

    # applying hist eq.
    # img = np.uint8(img)
    img = histogram_equalization(img)
    saveimage(img, "output/clock/2_histogram_equalization.png")

    # applying lowpass filter in frequency domain
    half_row = row // 2
    half_col = col // 2
    padded_img = np.lib.pad(img, ((half_row, half_row),(half_col, half_col)), 'constant',
constant_values=0)
    img_fft = imagefft(padded_img, center=True)
    b_mask = make_butterworth_LPF_mask_f(img_fft, cutoff=100, n=3)
    saveimage(b_mask*255, f"output/clock/3_LPF_mask_f.png")
```

```
    img_fft_filterd = img_fft * b_mask
    img = imageifft(img_fft_filterd, center=True)
    img = img[half_row:-half_row, half_col:-half_col] # center crop
    img = np.abs(img)
    max = img.max()
    min = img.min()
    img = (img - min) / (max-min) * 255
    saveimage(img, "output/clock/4_BLPF_result.png")



    # applying padding and average filtering in spatial domain
    average_filter_size=3
    padded_img = np.lib.pad(img, (((average_filter_size-1)//2, (average_filter_size-1)//2),
                                  ((average_filter_size-1)//2, (average_filter_size-1)//2)),
'reflect')
    img = average_filtering(padded_img, msize=average_filter_size)
    saveimage(img, f"output/clock/5_average_filter_{average_filter_size}.png")

    # applying sobel operator
    sobel_x = np.array([[-1, -2, -1],
                        [0, 0, 0],
                        [1, 2, 1]])
    sobel_y = np.array([[-1, 0, 1],
                        [-2, 0, 2],
                        [-1, 0, 1]])

    sobel_filter_size = 3
    padded_img = np.lib.pad(img, (((sobel_filter_size-1)//2, (sobel_filter_size-
1)//2),((sobel_filter_size-1)//2, (sobel_filter_size-1)//2)), 'reflect')
    grad_x = maskfiltering(padded_img, sobel_x)
    max = grad_x.max()
    min = grad_x.min()
    grad_x = (grad_x - min) / (max-min) * 255
    saveimage(grad_x, "output/clock/6_sobel_x.png")

    grad_y = maskfiltering(padded_img, sobel_y)
    max = grad_y.max()
    min = grad_y.min()
    grad_y = (grad_y - min) / (max-min) * 255
    saveimage(grad_y, "output/clock/7_sobel_y.png")

    grad_img = np.sqrt((grad_x)**2 + grad_y**2)
    max = grad_img.max()
    min = grad_img.min()
    grad_img = (grad_img - min) / (max-min) * 255
    saveimage(grad_img, "output/clock/8_highlighted_edge.png")
```

2) cars1_noise.png

```
"""
2024 Spring, Introduction to Computer Vision.
Project 1 - cars1_noise.png
"""

import numpy as np
import os
from PIL import Image as pilimg
from matplotlib import pyplot as plt


def readimage(img_name):
    global row, col
    im = pilimg.open(img_name)
    cimg = np.array(im)
    col, row = im.size
    return cimg, row, col


def saveimage(cimg, name):
    cimg=np.uint8(cimg)
    im=pilimg.fromarray(cimg)
    im.save(name)
    return im

def medianfiltering(img2d, ms):
    row, col = img2d.shape
    buff2d = np.full((row, col), 0)
    hs = ms//2
    for i in range(hs, row-hs):
        for j in range(hs, col-hs):
            temp=img2d[i-hs:i+hs+1, j-hs:j+hs+1].flatten()
            temp=np.sort(temp)
            buff2d[i, j] = temp[len(temp)//2]
    return buff2d[hs:-hs, hs:-hs]

def minmax_normalization(img, min, max):
    return 255.0 * (img-min) / (max-min)

def z_standard_normalization(img):
    # z_std = 1 keeping about 70%
    # z_std = 2 keeping about 95%
    z_std = 1
    mean = np.mean(img)
    std = np.std(img)
    z_score = np.clip((img-mean) / std, -z_std, z_std)
```

```
    return minmax_normalization(z_score, -z_std, z_std)

def histogram_equalization(img):
    histogram = [0 for _ in range(256)]

    for i in range(row):
        for j in range(col):
            histogram[img[i,j]] += 1

    pdf = np.array(histogram) / (row*col)
    cdf = np.cumsum(pdf)
    result = np.zeros_like(img)

    for i in range(row):
        for j in range(col):
            result[i,j] = 255 * cdf[img[i, j]]

    return np.uint8(result)


def imagefft(img, center):
    img_fft = np.fft.fft2(img)
    if center:
        img_fft = np.fft.fftshift(img_fft)
    return img_fft


def imageifft(img_fft, center=True):
    if center:
        img_fft = np.fft.ifftshift(img_fft)
    img = np.fft.ifft2(img_fft)
    return img


def make_ideal_LPF_mask_f(img_fft, d0):
    row, col = np.shape(img_fft)
    mask = np.full((row,col), 0.0).astype(np.float32)
    center_row = row // 2
    center_col = col // 2

    for i in range (row):
        for j in range(col):
            dist = np.sqrt((center_row-i)**2+(center_col-j)**2)
            if dist < d0:
                mask[i, j] = 1
            else:
                mask[i, j]= 0
    return mask
```

```
def make_gaussian_LPF_mask_f(img_fft, std):
    row, col = np.shape(img_fft)
    mask = np.full((row,col), 0.0).astype(np.float32)
    center_row = row // 2
    center_col = col // 2

    for i in range (row):
        for j in range(col):
            dist = np.sqrt((center_row-i)**2+(center_col-j)**2)
            mask[i, j] = np.exp(- (dist ** 2) / (2 * (std**2)))
    return mask


def make_butterworth_LPF_mask_f(img_fft, cutoff, n):
    row, col = np.shape(img_fft)
    mask = np.full((row,col), 0.0).astype(np.float32)
    center_row = row // 2
    center_col = col // 2

    for i in range (row):
        for j in range(col):
            dist = np.sqrt((center_row-i)**2+(center_col-j)**2)
            mask[i, j] = 1 / (1 + (dist / cutoff)**(2*n))
    return mask

def average_filtering(img2d, msize):
    row, col = img2d.shape
    buff2d = np.full((img2d.shape[0], img2d.shape[1]), 0)
    mask = np.ones((msize, msize)) / (msize*msize)
    hs = msize // 2

    for i in range(hs, row-hs):
        for j in range(hs, col-hs):
            buff2d[i, j] = np.sum(img2d[i-hs:i+hs+1, j-hs:j+hs+1] * mask)
    return buff2d[hs:-hs, hs:-hs]

def maskfiltering(img2d, mask):
    row, col = img2d.shape
    buff2d = np.full((row, col), 0)
    mr, mc = mask.shape[0], mask.shape[1]
    hs = mr // 2

    for i in range(hs, row-hs):
        for j in range(hs, col-hs):
            buff2d[i, j] = np.sum(img2d[i-hs:i+hs+1, j-hs:j+hs+1] * mask)
    return buff2d[hs:-hs, hs:-hs]
```

```
if __name__=="__main__":
    img, row, col = readimage("input/cars1_noise.png")

    # make grayscale img
    rcimg = img[:,:,0] # r-channel img
    gcimg = img[:,:,1] # g-channel img
    bcimg = img[:,:,2] # b-channel img
    gimg = np.uint8(0.299 * rcimg + 0.587 * gcimg + 0.114 * bcimg) # Convert a color image 
to a grayscale image
    saveimage(img, "output/cars1/0_grayscale.png")

    img = gimg

    # applying histogram_equalization
    img = histogram_equalization(img)
    saveimage(img, "output/cars1/1_hist_eq.png")

    # applying padding and average filtering in spatial domain
    average_filter_size=3
    padded_img = np.lib.pad(img, (((average_filter_size-1)//2, (average_filter_size-1)//2),
                                  ((average_filter_size-1)//2, (average_filter_size-1)//2)), 
'reflect')
    img = average_filtering(padded_img, msize=average_filter_size)
    padded_img = np.lib.pad(img, (((average_filter_size-1)//2, (average_filter_size-1)//2),
                                  ((average_filter_size-1)//2, (average_filter_size-1)//2)), 
'reflect')
    img = average_filtering(padded_img, msize=average_filter_size)
    padded_img = np.lib.pad(img, (((average_filter_size-1)//2, (average_filter_size-1)//2),
                                  ((average_filter_size-1)//2, (average_filter_size-1)//2)), 
'reflect')
    img = average_filtering(padded_img, msize=average_filter_size)

    saveimage(img, f"output/cars1/2_final_reduced_noise.png")

    # applying sobel operator
    sobel_x = np.array([[-1, -2, -1],
                        [0, 0, 0],
                        [1, 2, 1]])
    sobel_y = np.array([[-1, 0, 1],
                        [-2, 0, 2],
                        [-1, 0, 1]])

    sobel_filter_size = 3
    padded_img = np.lib.pad(img, (((sobel_filter_size-1)//2, (sobel_filter_size-1)//2),
                                  ((sobel_filter_size-1)//2, (sobel_filter_size-1)//2)), 
'reflect')
```

```
    grad_x = maskfiltering(padded_img, sobel_x)
    max = grad_x.max()
    min = grad_x.min()
    grad_x = (grad_x - min) / (max-min) * 255 # normalized
    saveimage(grad_x, "output/cars1/3_sobel_x.png")

    grad_y = maskfiltering(padded_img, sobel_y)
    max = grad_y.max()
    min = grad_y.min()
    grad_y = (grad_y - min) / (max-min) * 255 # normalized
    saveimage(grad_y, "output/cars1/4_sobel_y.png")

    grad_img = np.sqrt((grad_x)**2 + grad_y**2)
    max = grad_img.max()
    min = grad_img.min()
    grad_img = (grad_img - min) / (max-min) * 255 # calc from normalized img
    saveimage(grad_img, "output/cars1/5_highlighted_edge.png")
```

3) cars2_noise.png

```
"""
2024 Spring, Introduction to Computer Vision.
Project 1 - cars2_noise.png
"""

import numpy as np
import os
from PIL import Image as pilimg
from matplotlib import pyplot as plt


def readimage(img_name):
    global row, col
    im = pilimg.open(img_name)
    cimg = np.array(im)
    col, row = im.size
    return cimg, row, col


def saveimage(cimg, name):
    cimg=np.uint8(cimg)
    im=pilimg.fromarray(cimg)
    im.save(name)
    return im

def medianfiltering(img2d, ms):
    row, col = img2d.shape
    buff2d = np.full((row, col), 0)
    hs = ms//2
    for i in range(hs, row-hs):
        for j in range(hs, col-hs):
            temp=img2d[i-hs:i+hs+1, j-hs:j+hs+1].flatten()
            temp=np.sort(temp)
            buff2d[i, j] = temp[len(temp)//2]
    return buff2d[hs:-hs, hs:-hs]

def minmax_normalization(img, min, max):
    return 255.0 * (img-min) / (max-min)

def z_standard_normalization(img):
    # z_std = 1 keeping about 70%
    # z_std = 2 keeping about 95%
    z_std = 1
    mean = np.mean(img)
    std = np.std(img)
    z_score = np.clip((img-mean) / std, -z_std, z_std)
```

```
    return minmax_normalization(z_score, -z_std, z_std)

def histogram_equalization(img):
    histogram = [0 for _ in range(256)]

    for i in range(row):
        for j in range(col):
            histogram[img[i,j]] += 1

    pdf = np.array(histogram) / (row*col)
    cdf = np.cumsum(pdf)
    result = np.zeros_like(img)

    for i in range(row):
        for j in range(col):
            result[i,j] = 255 * cdf[img[i, j]]

    return np.uint8(result)


def imagefft(img, center):
    img_fft = np.fft.fft2(img)
    if center:
        img_fft = np.fft.fftshift(img_fft)
    return img_fft


def imageifft(img_fft, center=True):
    if center:
        img_fft = np.fft.ifftshift(img_fft)
    img = np.fft.ifft2(img_fft)
    return img


def make_ideal_LPF_mask_f(img_fft, d0):
    row, col = np.shape(img_fft)
    mask = np.full((row,col), 0.0).astype(np.float32)
    center_row = row // 2
    center_col = col // 2

    for i in range (row):
        for j in range(col):
            dist = np.sqrt((center_row-i)**2+(center_col-j)**2)
            if dist < d0:
                mask[i, j] = 1
            else:
                mask[i, j]= 0
    return mask
```

```python
def make_gaussian_LPF_mask_f(img_fft, std):
    row, col = np.shape(img_fft)
    mask = np.full((row,col), 0.0).astype(np.float32)
    center_row = row // 2
    center_col = col // 2

    for i in range (row):
        for j in range(col):
            dist = np.sqrt((center_row-i)**2+(center_col-j)**2)
            mask[i, j] = np.exp(- (dist ** 2) / (2 * (std**2)))
    return mask


def make_butterworth_LPF_mask_f(img_fft, cutoff, n):
    row, col = np.shape(img_fft)
    mask = np.full((row,col), 0.0).astype(np.float32)
    center_row = row // 2
    center_col = col // 2

    for i in range (row):
        for j in range(col):
            dist = np.sqrt((center_row-i)**2+(center_col-j)**2)
            mask[i, j] = 1 / (1 + (dist / cutoff)**(2*n))
    return mask


def average_filtering(img2d, msize):
    row, col = img2d.shape
    buff2d = np.full((img2d.shape[0], img2d.shape[1]), 0)
    mask = np.ones((msize, msize)) / (msize*msize)
    hs = msize // 2

    for i in range(hs, row-hs):
        for j in range(hs, col-hs):
            buff2d[i, j] = np.sum(img2d[i-hs:i+hs+1, j-hs:j+hs+1] * mask)
    return buff2d[hs:-hs, hs:-hs]

def maskfiltering(img2d, mask):
    row, col = img2d.shape
    buff2d = np.full((row, col), 0)
    mr, mc = mask.shape[0], mask.shape[1]
    hs = mr // 2

    for i in range(hs, row-hs):
        for j in range(hs, col-hs):
            buff2d[i, j] = np.sum(img2d[i-hs:i+hs+1, j-hs:j+hs+1] * mask)
```

```
    return buff2d[hs:-hs, hs:-hs]

if __name__=="__main__":
    img, row, col = readimage("input/cars2_noise.png")

    # make grayscale img
    rcimg = img[:,:,0] # r-channel img
    gcimg = img[:,:,1] # g-channel img
    bcimg = img[:,:,2] # b-channel img

    gimg = np.uint8(0.299 * rcimg + 0.587 * gcimg + 0.114 * bcimg) # Convert a color image
to a grayscale image
    saveimage(gimg, "output/cars2/0_grayscale.png")

    img = gimg

    # applying median filter
    median_filter_size = 3
    padded_img = np.lib.pad(img, (((median_filter_size-1)//2, (median_filter_size-1)//2),
                                  ((median_filter_size-1)//2, (median_filter_size-1)//2)),
'reflect')
    img = medianfiltering(padded_img, median_filter_size)
    saveimage(img, f"output/cars2/1_median_filtering_{median_filter_size}.png")

    # applying hist eq
    img = histogram_equalization(img)
    saveimage(img, "output/cars2/2_histogram_equalization.png")

    # applying zero padding and BLPF in frequency domain.
    half_row = row // 2
    half_col = col // 2
    padded_img = np.lib.pad(img, ((half_row, half_row),(half_col, half_col)), 'constant',
constant_values=0)
    saveimage(np.abs(padded_img), "output/cars2/3_padded.png")

    img_fft = imagefft(padded_img, center=True)
    b_mask = make_butterworth_LPF_mask_f(img_fft, cutoff=500, n=3)
    saveimage(b_mask*255, f"output/cars2/4_mask_in_frequency_domain.png")

    img_fft_filterd = img_fft * b_mask

    img = imageifft(img_fft_filterd, center=True)
    img = img[half_row:-half_row, half_col:-half_col]

    img = np.abs(img)
    max = img.max()
    min = img.min()
    img = (img - min) / (max-min) * 255
```

```
    saveimage(img, f"output/cars2/5_final_reduced_noise.png")

    # applying sobel operator
    sobel_x = np.array([[-1, -2, -1],
                        [0, 0, 0],
                        [1, 2, 1]])
    sobel_y = np.array([[-1, 0, 1],
                        [-2, 0, 2],
                        [-1, 0, 1]])

    sobel_filter_size = 3
    padded_img = np.lib.pad(img, (((sobel_filter_size-1)//2, (sobel_filter_size-1)//2),
                                  ((sobel_filter_size-1)//2, (sobel_filter_size-1)//2)),
'reflect')
    grad_x = maskfiltering(padded_img, sobel_x)
    max = grad_x.max()
    min = grad_x.min()
    grad_x = (grad_x - min) / (max-min) * 255
    saveimage(grad_x, "output/cars2/6_sobel_x.png")

    grad_y = maskfiltering(padded_img, sobel_y)
    max = grad_y.max()
    min = grad_y.min()
    grad_y = (grad_y - min) / (max-min) * 255
    saveimage(grad_y, "output/cars2/7_sobel_y.png")

    grad_img = np.sqrt((grad_x)**2 + grad_y**2)
    max = grad_img.max()
    min = grad_img.min()
    grad_img = (grad_img - min) / (max-min) * 255
    saveimage(grad_img, "output/cars2/8_highlighted_edge.png")
```