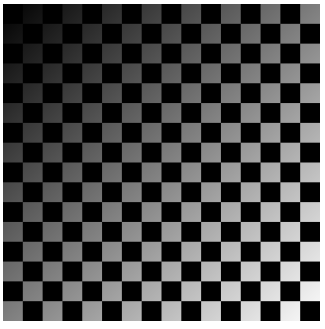
DIP HW 3-3

311505011 人工智慧碩二 黃丰楷

1. Please use a self-designed Lowpass Gaussian Filter Kernels to cancel the shaded pattern noise of the image, ‘checkerboard1024-shaded.tif’. Please describe the your lowpass Gaussian filter kernels, shading noise pattern as Fig. 3.42 (b), final corrected image without shading noise pattern as Fig. 3.42 (c) and print out the source code?(40)

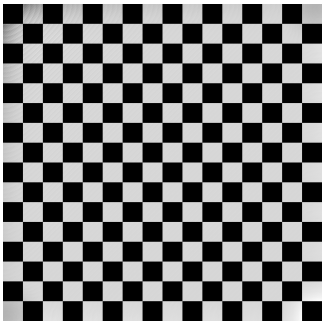
The image size is 1024x1024, so that the size of a small square is 128x128. I made the kernel size be equal to 4 times to a small square, $kernel\ size = (257, 257)$, to includes a white square and a black square in a row and column respectively. Then the σ **is set to 64**, which is half of the size of a small square. That is, the lowpass Gaussian filter can blur the image properly to estimate the shade in the back ground. Then the original image is divided by the estimated shade. After that, I normalize the corrected image to [0, 255].

- Original image



- Estimated shade

- Corrected image



- Code

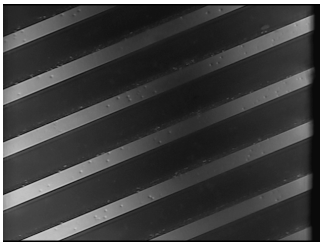
```
import cv2
import matplotlib.pyplot as plt
import numpy as np

img = cv2.imread("./checkerboard1024-shaded.tif",cv2.IMREAD_GRAYSCALE)
```

```
print(img.shape)


kernel_size = (257, 257)
sigmaX, sigmaY = 64, 64


shade = cv2.GaussianBlur(img,kernel_size, sigmaX=sigmaX, sigmaY=sigmaY)

titles = ['original_1', 'estimated_shade_1', "divided_1"]
# titles = ['original_2', 'estimated_shade_2', "divided_2"]


divided_img =  (img / shade)

divided_img = (divided_img-np.min(divided_img))*255/ (np.max(divided_img)-np.min(divided_img))


outputs = [img, shade, divided_img]

for idx, title in enumerate(titles):
    cv2.imwrite(f"./{title}.png", outputs[idx])
```

2. Repeat (1) steps in the image ‘N1.bmp’ (40)

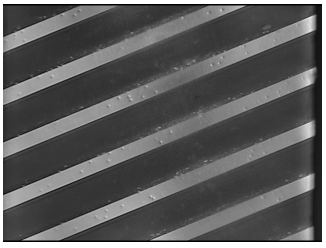
Different from (1), the image size of ‘N1.bmp’ is 480x640, which is a horizontal rectangle. Another difference is that the shadows in the background on the right side of the image are in the form of vertical strips. The square kernel cannot handle this type of shadows. Thus, I made the kernel be a vertical rectangle, $kernel\ size = (57, 357)$, which can handle the vertical shadows properly. Then the σ_x **is set to 40** and σ_y **is set to 60**, respectively. That is, the lowpass Gaussian filter can blur the image properly to estimate the shade in the back ground. And we can see the shadows on the right side of the image are predicted as well. Then the original image is divided by the estimated shade. After that, I normalize the corrected image to [0, 255].

- Original image



- Estimated shade

- Corrected image



- Code

```
import cv2
import matplotlib.pyplot as plt
```

```
import numpy as np

img = cv2.imread("./N1.bmp",cv2.IMREAD_GRAYSCALE)


print(img.shape)


kernel_size = (57, 357)
sigmaX, sigmaY = 40, 60

shade = cv2.GaussianBlur(img,kernel_size, sigmaX=sigmaX, sigmaY=sigmaY)

titles = ['original_2', 'estimated_shade_2', "divided_2"]


divided_img =  (img / shade)

divided_img = (divided_img-np.min(divided_img))*255/ (np.max(divided_img)-np.min(divided_img))


outputs = [img, shade, divided_img]

for idx, title in enumerate(titles):
    cv2.imwrite(f"./{title}.png", outputs[idx])
```

3. Please comment and compare your two self-designed Lowpass Gaussian Filter Kernels? (20)

Although both problems use the lowpass gaussian filter to deal with shadows, the shadows are not presented in the same way. In the first problem, the graph form shows a square distribution, and the shadows are gradually changing from the top left to the bottom right. The problem of this problem is that the kernel size should be large enough to blur the blocks in the foreground, and then predict the shadows. The problem with the second question is that the pattern in the foreground is distributed diagonally and the shadows in the background are distributed vertically. If we use a normal square kernel, we can't predict the vertical shadow on the right side of the image well. So I set the shape of the kernel to be a vertical strip, so that the shadows can be predicted well.