DIP HW 3-2

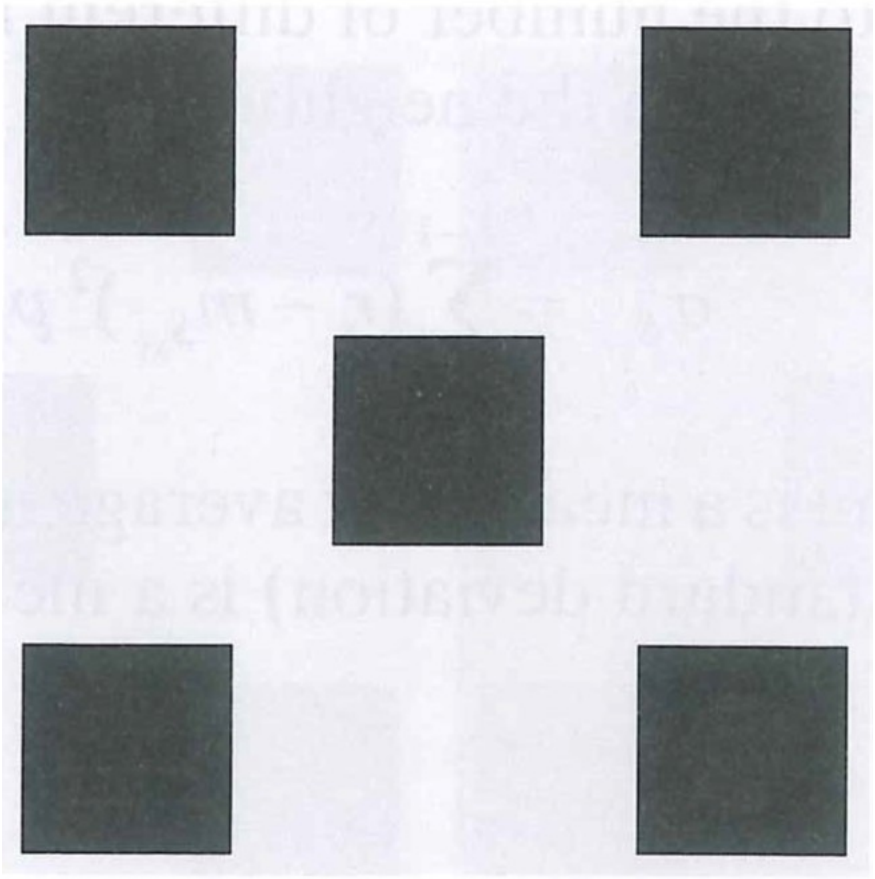
人工智慧碩二 311505011 黃丰楷

1. Please use a Histogram Statistics method and a local enhancement method to extract the hidden image of the image, ‘hidden object.jpg.’ Please describe the your parameters and method in detail and print out the source code? (40+40)

Because the image provided for the assignment seems to be slightly different from the one in the textbook, there are hidden texts in the background, and extracting the text within the boxes is more challenging after processing. Therefore, I will process and present both the image provided for the assignment and the one from the textbook below.

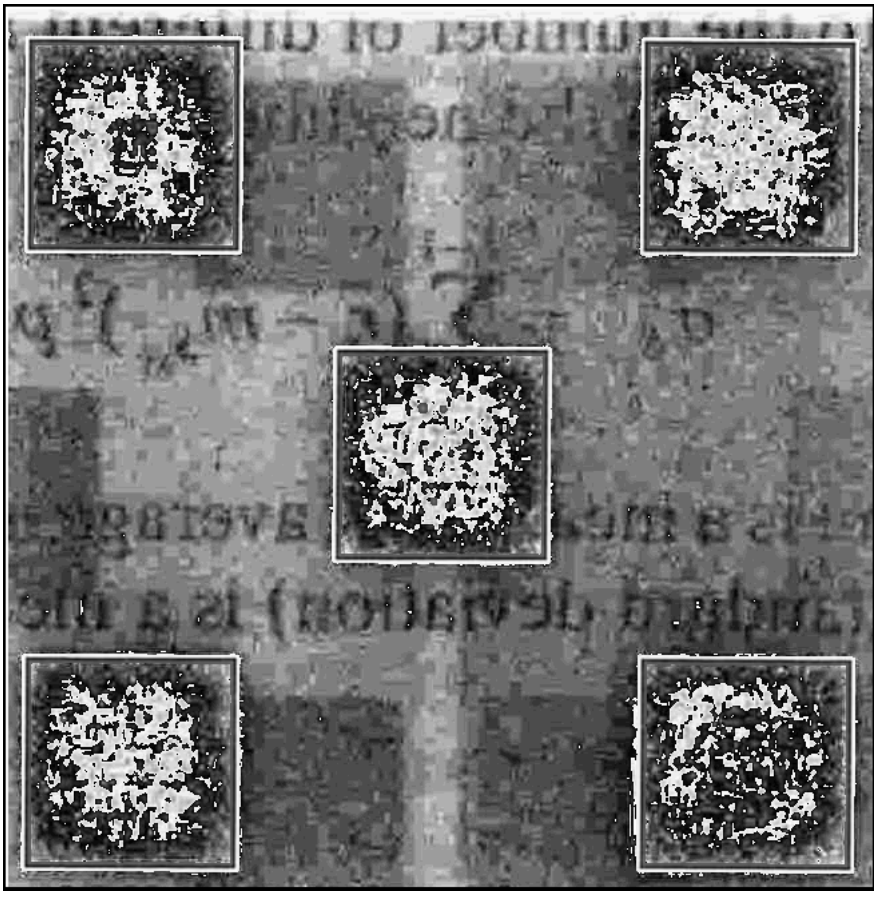
The image provided for the assignment

- Original image



- Histogram Statistics method

  I processed this image using histogram statistics. After calculating the global mean, denoted as m, and global standard deviation, denoted as σ. I set the local mean m_s in the range of $0.025m \leq m_s \leq 2m$, and the local standard deviation σ_s in the range of $0 \leq \sigma_s \leq 0.1\sigma$. Throughout the process, I used a kernel size of 5. Finally, I multiplied the grayscale intensity of the selected range of pixels by 10 to emphasize the text that is closer to the background color. The processed image is shown below:



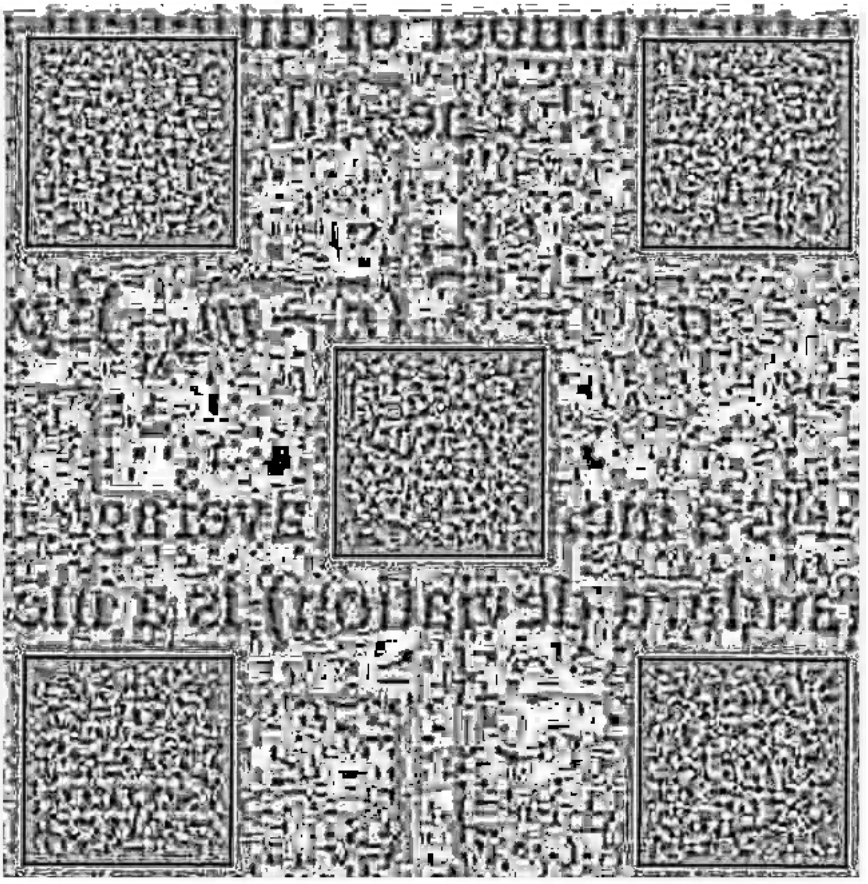
We can see the texts in the background are now more obvious. And the code is shown below:

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


img = cv2.imread("hidden object.jpg", cv2.IMREAD_GRAYSCALE)  # shape: (665, 652, 3)
cv2.imwrite("ori.png", img)


global_mean = np.mean(img)
print(global_mean)

global_std = np.std(img)
print(global_std)


kernel_size = 5
output_img = np.zeros_like(img)
h, w = img.shape
print(img.shape)
img = np.pad(img, ((kernel_size//2,kernel_size//2),(kernel_size//2,kernel_size//2)))
print(img.shape)

k0, k1, k2, k3 = 0.025,2, 0, 0.1


for y in np.arange(kernel_size//2, h - (kernel_size//2+1)):  #652
    for x in np.arange(kernel_size//2, w - kernel_size//2+1):

        area = img[(y-(kernel_size//2)) : (y+(kernel_size//2+1)), (x-(kernel_size//2)) : (x+(kernel_size//2+1))]
        local_mean = np.mean(area)
        local_std =np.std(area)

        if (local_mean >= k0 *global_mean) and (local_mean <= k1 *global_mean) and (local_std >= k2*global_std ) and (local_std <=

            output_img[y, x] = img[y, x]* 10

        else:
            output_img[y, x] = img[y, x]
```

```
cv2.imwrite("ori_3_2_hist_stat.png", output_img)
```

- local enhancement method

  I performed local histogram equalization on the image. The kernel size I used was 13, with the aim of making the text in the background easier to distinguish by increasing the kernel size. The processed image is shown below:



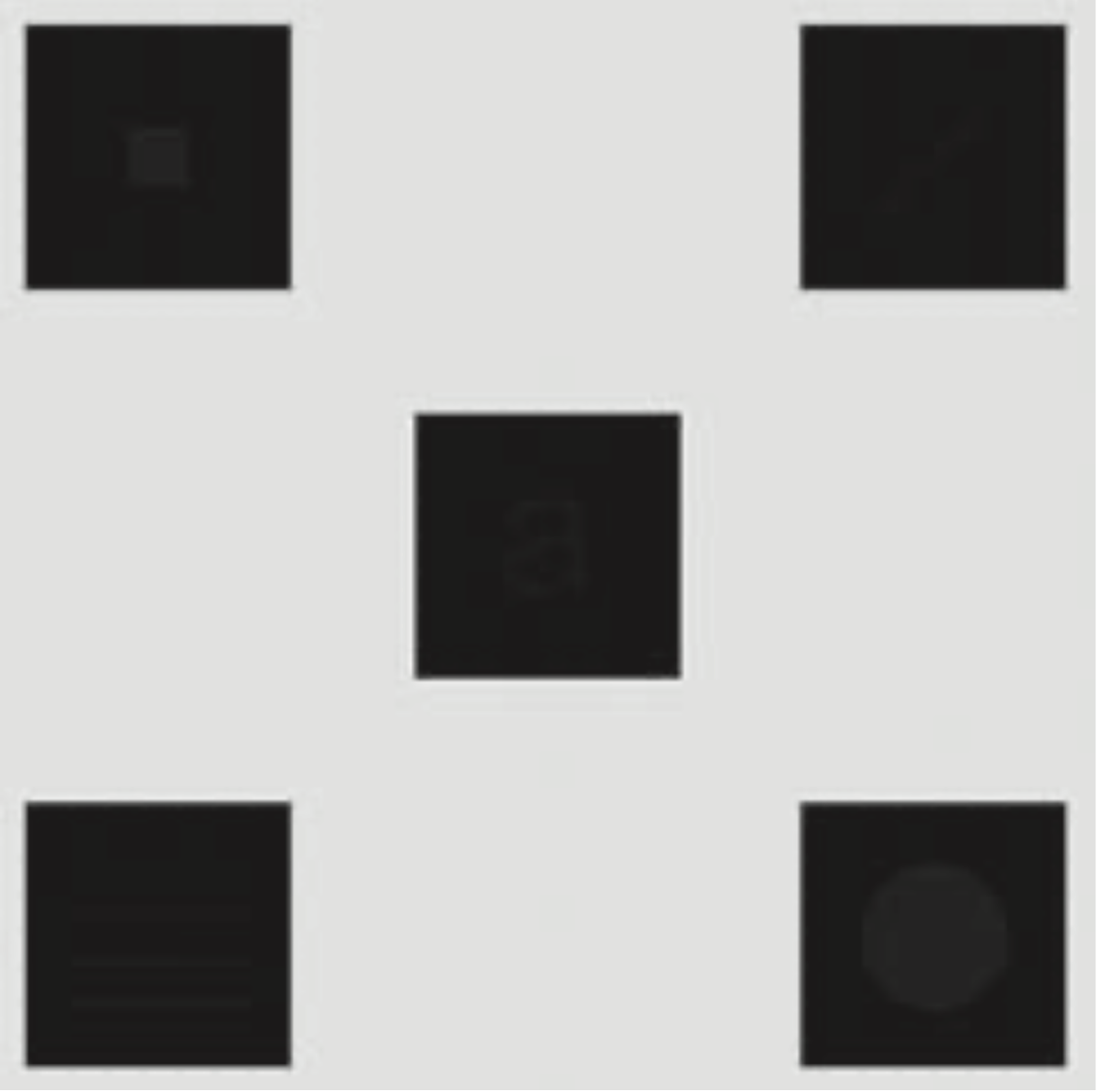
Through local enhancement, it can be observed that the edges of the text in the background have become clearer. The corresponding code is as follows:

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



img = cv2.imread("hidden object.jpg", cv2.IMREAD_GRAYSCALE)  # shape: (665, 652, 3)




kernel_size = 13
stride = 1
output_img = np.empty_like(img)
h, w = img.shape
print(img.shape)
img = np.pad(img, ((kernel_size//2,kernel_size//2),(kernel_size//2,kernel_size//2)))
print(img.shape)

for y in np.arange(kernel_size//2, h - (kernel_size//2)+1, stride):
    for x in np.arange(kernel_size//2, w - kernel_size//2 +1, stride):
        y1 = (y-(kernel_size//2))
        y2 = (y+(kernel_size//2)+1)
        x1 = (x-(kernel_size//2))
```

```
        x2 = (x+(kernel_size//2)+1)
        area = img[y1 : y2, x1 : x2 ]
        equalized_area = cv2.equalizeHist(area)
        output_img[y, x] = equalized_area[kernel_size//2, kernel_size//2]

print(output_img.shape)
cv2.imwrite("ori_3_2_local_enhance.png", output_img)
```

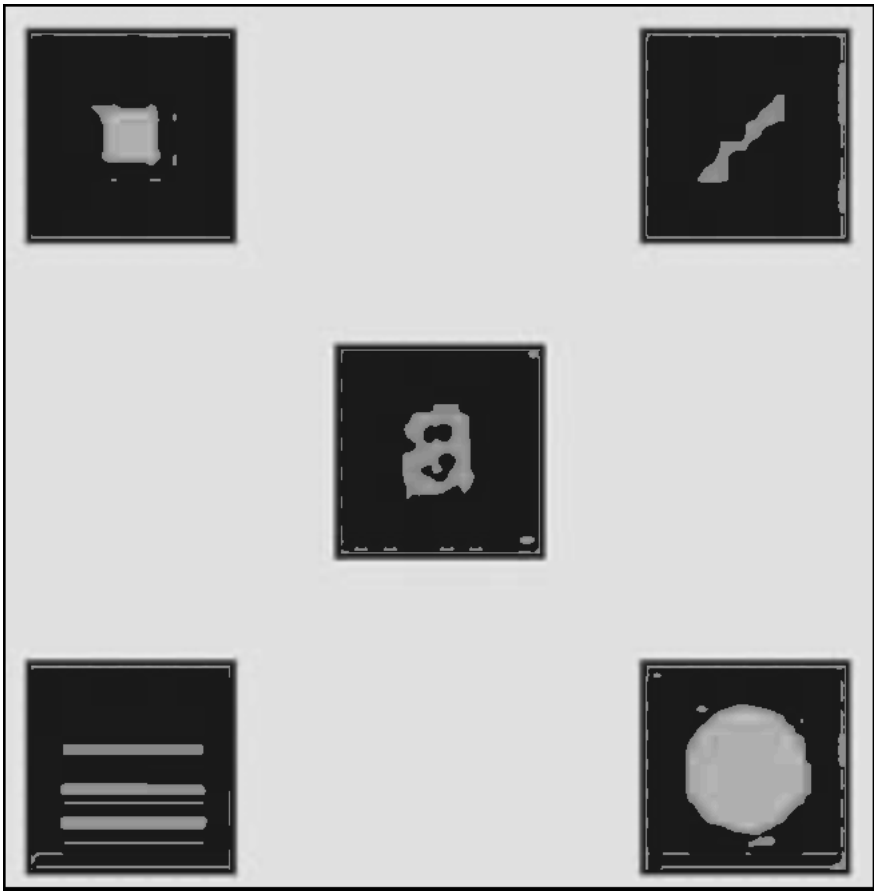
The image from the text book

- Original image



- Histogram Statistics method

  I processed this image using histogram statistics. The image provided for the assignment is different, as it has a higher resolution after being captured. Therefore, I resized the image before processing. After calculating the global mean, denoted as m, and global standard deviation, denoted as σ. I set the local mean m_s in the range of $0.16m \leq m_s \leq 0.3m$, and the local standard deviation σ_s in the range of $0 \leq \sigma_s \leq 0.1\sigma$. Throughout the process, I used a kernel size of 5. Finally, I multiplied the grayscale intensity of the selected range of pixels by 5. After multiple attempts at adjusting the parameters, the pattern inside the boxes can be distinctly separated. The processed image is shown below:



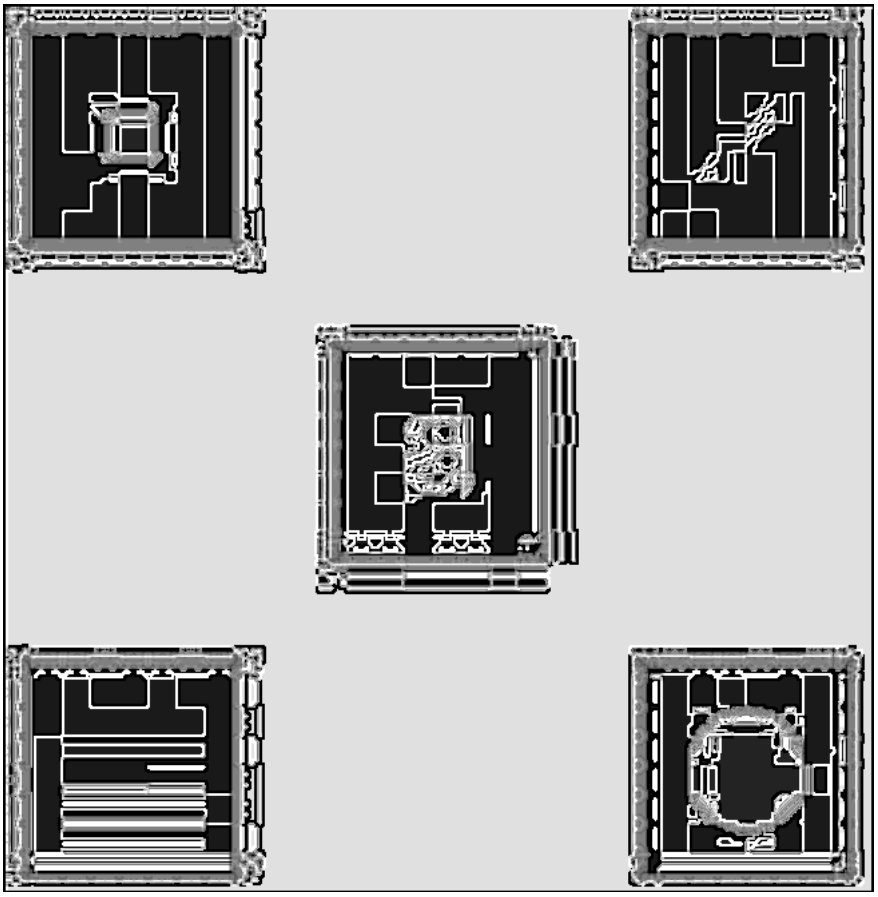
We can see the graphs in the black boxes are now more obvious. And the code is shown below:

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


img = cv2.imread("hidden object2.png", cv2.IMREAD_GRAYSCALE)  # shape: (665, 652, 3)

img = cv2.resize(img, (652,665),interpolation=cv2.INTER_AREA)
cv2.imwrite("ori.png", img)


global_mean = np.mean(img)
print(global_mean)

global_std = np.std(img)
print(global_std)


kernel_size = 5
output_img = np.zeros_like(img)
h, w = img.shape
print(img.shape)
img = np.pad(img, ((kernel_size//2,kernel_size//2),(kernel_size//2,kernel_size//2)))
print(img.shape)

k0, k1, k2, k3 = 0.16, 0.3, 0, 0.1


for y in np.arange(kernel_size//2, h - (kernel_size//2+1)):  #652
    for x in np.arange(kernel_size//2, w - kernel_size//2+1):

        area = img[(y-(kernel_size//2)) : (y+(kernel_size//2+1)), (x-(kernel_size//2)) : (x+(kernel_size//2+1))]
        local_mean = np.mean(area)
        local_std =np.std(area)

        if (local_mean >= k0 *global_mean) and (local_mean <= k1 *global_mean) and (local_std >= k2*global_std ) and (local_std <=

            output_img[y, x] = img[y, x]* 5
```

```
        else:
            output_img[y, x] = img[y, x]


cv2.imwrite("3_2_hist_stat.png", output_img)
```

- local enhancement method

  I performed local histogram equalization on the image. Also, I did the resize process first, and the kernel size I used was 5, with the aim of making the graphs in the black boxes easier to distinguish by increasing the kernel size. The processed image is shown below:



Through local enhancement, it can be observed that the edges of the edge of the graphs have become clearer. The corresponding code is as follows:

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



img = cv2.imread("hidden object2.png", cv2.IMREAD_GRAYSCALE)  # shape: (665, 652, 3)

img = cv2.resize(img, (652,665),interpolation=cv2.INTER_AREA)



kernel_size = 5
stride = 1
output_img = np.empty_like(img)
h, w = img.shape
print(img.shape)
img = np.pad(img, ((kernel_size//2,kernel_size//2),(kernel_size//2,kernel_size//2)))
print(img.shape)
```

```
for y in np.arange(kernel_size//2, h - (kernel_size//2)+1, stride):
    for x in np.arange(kernel_size//2, w - kernel_size//2 +1, stride):
        y1 = (y-(kernel_size//2))
        y2 = (y+(kernel_size//2)+1)
        x1 = (x-(kernel_size//2))
        x2 = (x+(kernel_size//2)+1)
        area = img[y1 : y2, x1 : x2 ]
        equalized_area = cv2.equalizeHist(area)
        output_img[y, x] = equalized_area[kernel_size//2, kernel_size//2]


print(output_img.shape)
cv2.imwrite("3_2_local_enhance.png", output_img)
```

2. Please comment and compare Histogram Statistics method and a local enhancement method?

- Histogram statistics processing enhances images by:
    1. Separating elements or text with similar grayscale values.
    2. Selecting pixels based on local mean range.
    3. Enhancing edges through local standard deviation range.
- Local enhancement:
    - Stronger effect on edges.
    - Hidden figures' edges become clearly visible.
    - Introduces slightly more intricate lines, adding detail.
- Overall impact:
    - Significantly improves image quality.
    - Facilitates extraction of meaningful information.
    - Useful for nuanced analysis and interpretation.