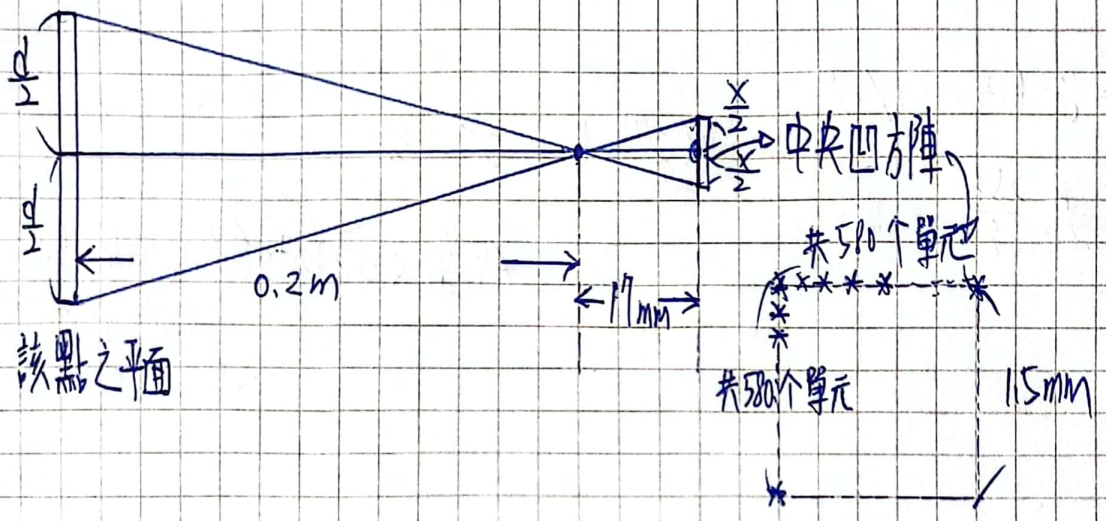
DIP HW　　資管四 410902934 蕭柏叡

Q1:

section 2.1 提到錐狀體約337,000單元，若是以方陣來表示則長寬約為580單位

其中中央四為1.15mm × 1.15mm的正方形陣列




因為單元間之間隔與單元大小一致，故1.15mm = (580個單元 + 519個間隔)

根據相似三角形 $\frac{d}{0.2m} = \frac{x}{17\times10^{-3}m}$

$x = 1.15\times10^{-3}/1159 \fallingdotseq 1.3\times10^{-6}m$

$\frac{d}{2}\times17\times10^{-3} = 0.2\times\frac{x}{2}\times1.3\times10^{-6}$

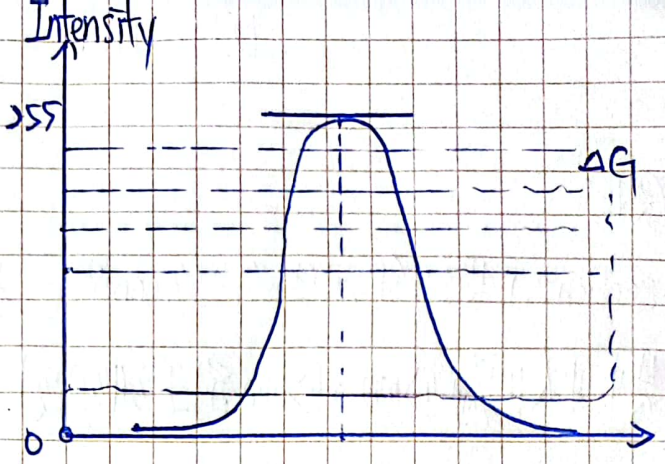
$\frac{2.6\times10^{-7}}{17\times10^{-3}} = \frac{216}{117}\times10^{-5} \fallingdotseq 1.53\times10^{-5}m$ #

Q2:

$$f(x,y) = i(x,y)\, r(x,y)$$

$$= 255\, e^{-[(x-x_0)^2+(y-y_0)^2]} \times 1.0$$

$$= 255\, e^{-[(x-x_0)^2+(y-y_0)^2]}$$




$\Delta G = 8 = (255+1)/2^m$

$\times 2^m = k = 2^5 = 32$

$2^m = \frac{256}{8} \Rightarrow m = 5$

A: 當 < 32 bits 會有 visible false contouring 的問題。

Q3:

∵ width : height = $\frac{16}{9}$ 已知垂直方向有1125 lines ⇒ 也就是1125 pixels

↳ width = $\frac{16}{9} \times 1125 = 2000$ pixels

算位操算:

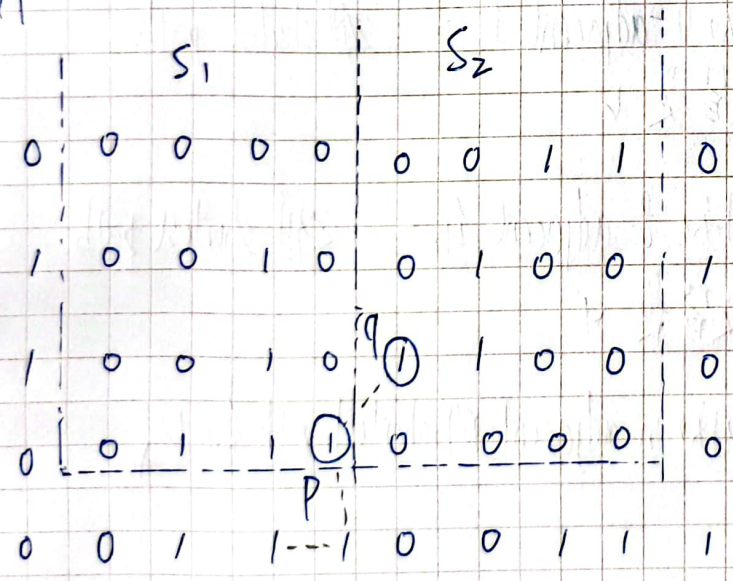
RGB各8个bits ⇒ $3 \times 8 = 24$ × 每个field $\frac{1}{60}$ s 共2个fields ⇒ $\frac{1}{60} \times 2 = \frac{1}{30}$ s

2 hrs = 7200 s

$$\underbrace{2000 \times 1125}_{\text{pixel}} \times \underbrace{24}_{\text{RGB}} \times \underbrace{7200}_{\text{hr}} / \underbrace{\frac{1}{30}}_{\frac{1}{30}\text{s}} \Rightarrow 2000 \times 1125 \times 24 \times 7200 \times 30 = 1.1664 \times 10^{13} \text{ bits} \#$$

A: $1.1664 \times 10^{13}$ bits

Q4:




先選 p, q (a) 可看出 $S_1$、$S_2$ 並非 4-connected 因為 p 的上下左右都不是1

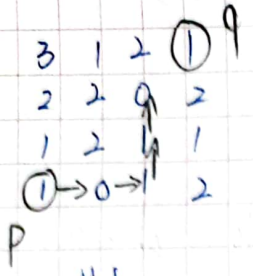
故和 4-~~connected~~ adjacent 違背及矛盾

(b) 可看出 $S_1$, $S_2$ 符合 8-connected ∵ q 在 $N_8(p)$ 這個集合。

(c) 可看出 $S_1$, $S_2$ 符合 m-connected ∵ q 在 $N_D(p)$ 內且 $N_4(p) \cap N_4(q)$ 是空的。

Q5:

5-(a) Let V = {0, 1}

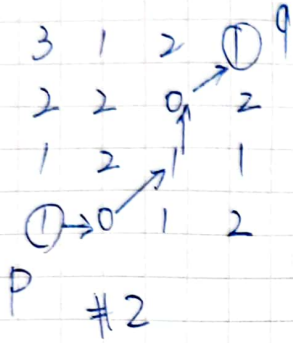



#1

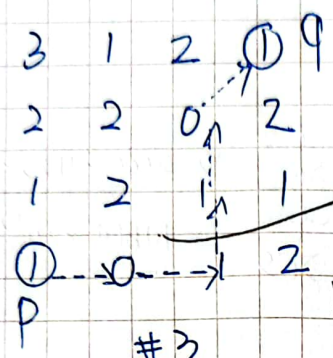
由#1可知並非4-adjacent，p到不了q

由#2可知 8-adjacent 的 shortest path 長度為4

由#3可知 m-adjacent 的 shortest path 長度為5




#2

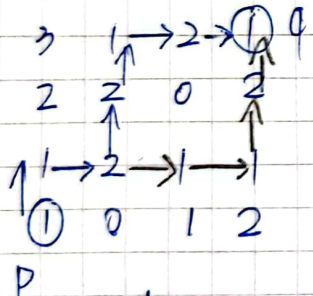



這邊不能算的原因

m-adjacent 的定義 $N_4(p) \cap N_4(q) \Rightarrow \phi$

#3

5-(b) Let V = {1, 2}




#4

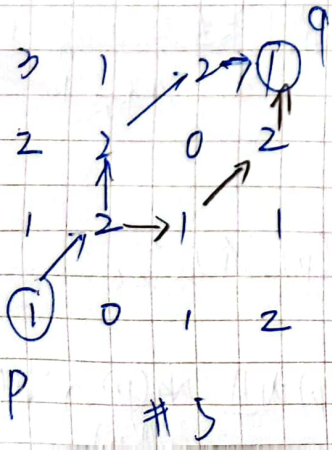
由#4可知 4-adjacent 有 → → 2種 shortest path

長度皆是 6

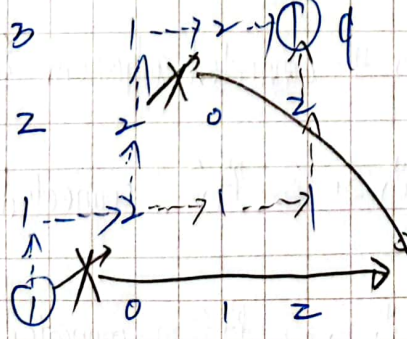
由#5可知 8-adjacent 有 → → 2種 shortest path

長度皆是 4

由#6可知 m-adjacent 的 shortest path 為 6 #




#5




$N_4(p) \cap N_4(q) \neq \phi$，不能算

#1

Q6:

Let $S_1 = \{3, -5, 8\}$, $S_2 = \{21, 25, 31\}$

$H(S_1) = 3$ ; $H(S_2) = 25$

$H(aS_1 + bS_2)$ 令 $a = 1$ ; $b = 1 \Rightarrow H(S_1 + S_2) = \text{median}\{24, 20, 39\} = 24$

又 $H(aS_1 + bS_2) \neq aH(S_1) + bH(S_2)$ ∴ 為 non-linear 的 #

$24 \neq 3 + 25$

Q7: i) $r(x, y, u, v) = e^{-j2\pi(ux/M + vy/N)}$

separable: $r(x, y, u, v) = e^{-j2\pi(ux/M + vy/N)}$

$= e^{-j2\pi(ux/M)} \times e^{-j2\pi(vy/N)}$

$= r_1(x, u) \times r_2(y, v)$ #

symmetric: ∵ $r_1(x, y)$ is functionally equal to $r_2(x, y)$

∴ $r_1(x, u)\, r_2(y, v) = r_1(x, u)\, r_1(y, v)$ #

---

ii) $s(x, y, u, v) = \frac{1}{MN} e^{j2\pi(ux/M + yv/N)}$

separable: $s(x, y, u, v) = \frac{1}{MN} \times e^{j2\pi(ux/M)} \times e^{j2\pi(vy/N)}$

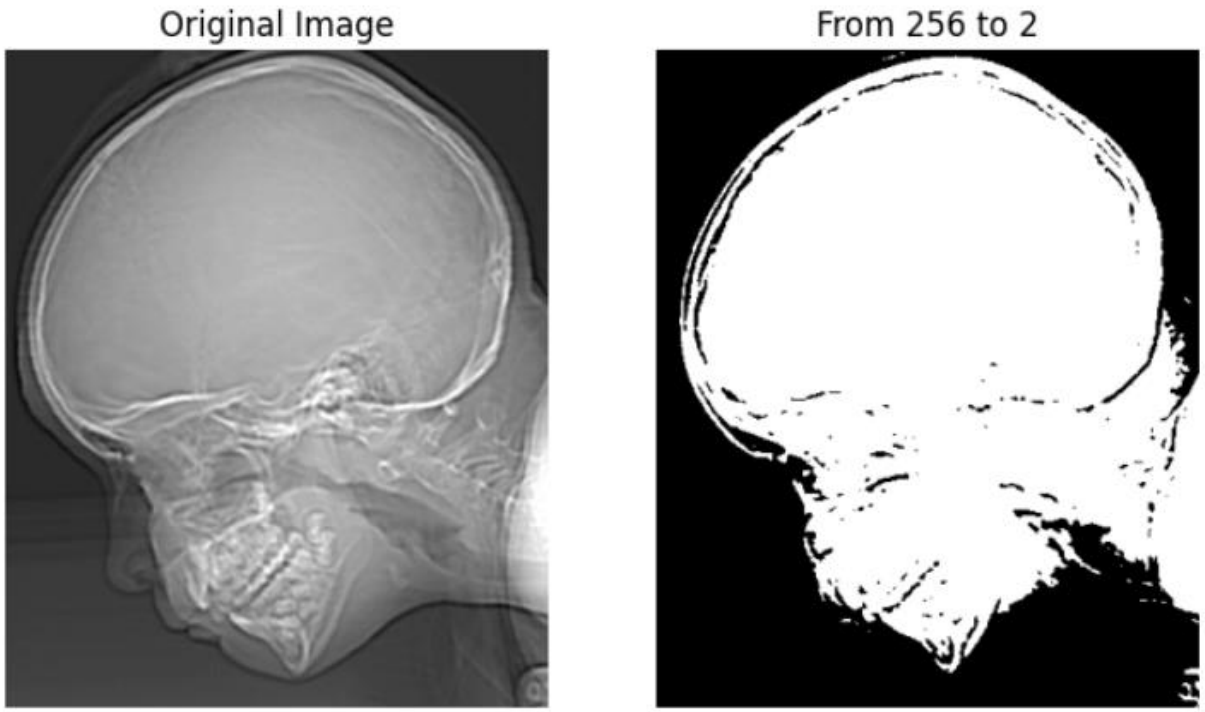
$= \frac{1}{MN} \times s_1(x, u) \times s_2(y, v)$ #

symmetric: ∵ $s_1(x, y)$ is functionally equal to $s_2(x, y)$

∴ $\frac{1}{MN} \times s_1(x, u)\, s_2(y, v) = \frac{1}{MN} s_1(x, u)\, s_1(y, v)$ #

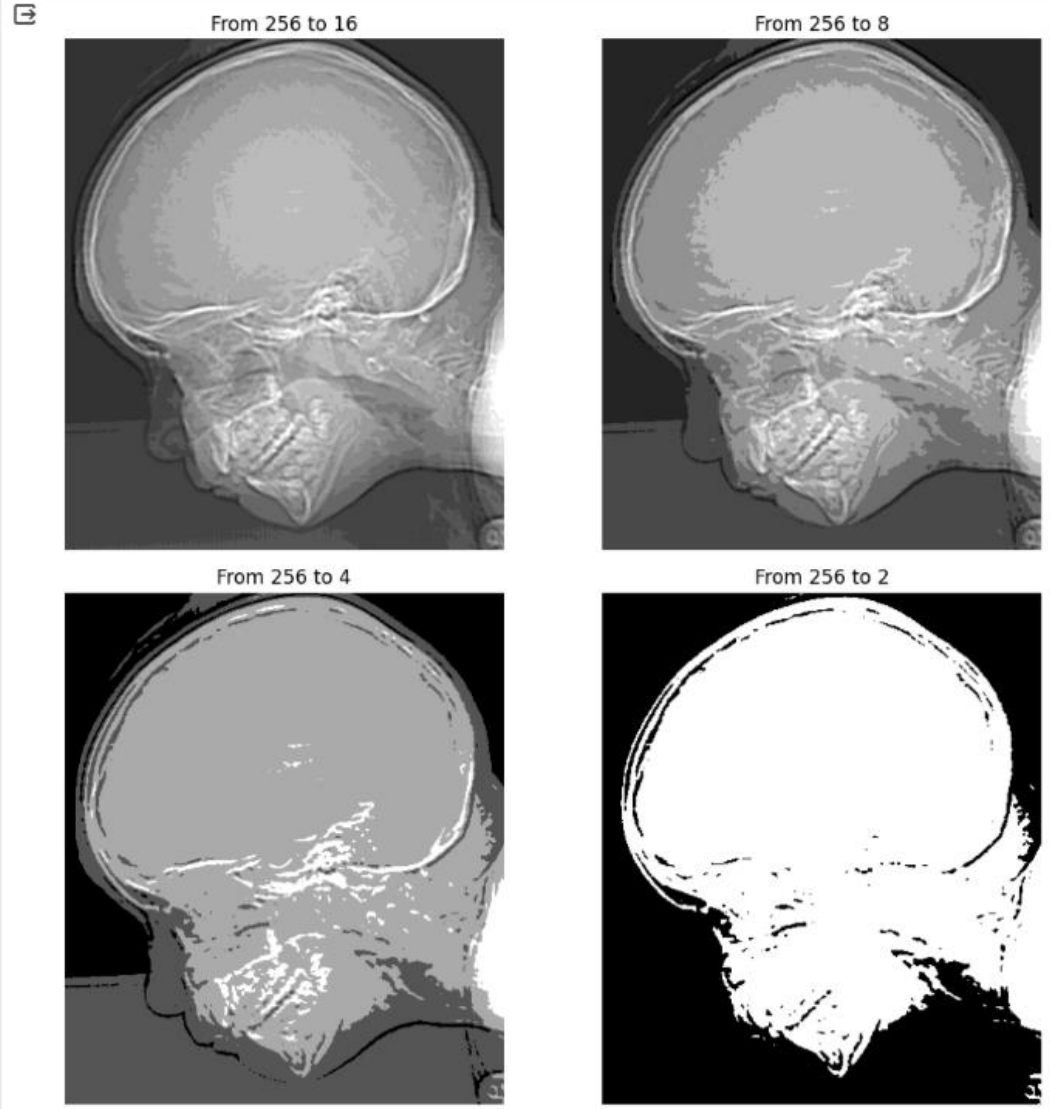
Q8: Please print out the simulation code along with your results for the problems below. 8. Reducing the Number of Intensity Levels in an Image (10%)
(a) Write a computer program capable of reducing the number of intensity levels in an image from 256 to 2, in integer powers of 2. The desired number of intensity levels needs to be a variable input to your program.

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

def reduce_intensity_levels(image_path, levels):
    # 讀取圖像
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    # 計算每個強度級別的間隔
    interval = 256 // levels
    # 減少強度級別
    reduced_image = (image // interval) * interval
    # 顯示原始圖像
    plt.figure(figsize=(8, 8))
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Original Image')
    plt.axis('off')
    # 顯示處理後的圖像
    plt.subplot(1, 2, 2)
    plt.imshow(reduced_image, cmap='gray')
    plt.title('From 256 to 2')
    plt.axis('off')
# 輸入圖像的檔案路徑和所需的強度級別
image_path = "/content/Fig0221(a)(ctskull-256).tif"
desired_levels = 2

# 呼叫函式進行圖像處理
reduce_intensity_levels(image_path, desired_levels)
```



(b) Download Fig. 2.21(a) from the course web site and duplicate the results shown in Fig. 2.21 of the book.

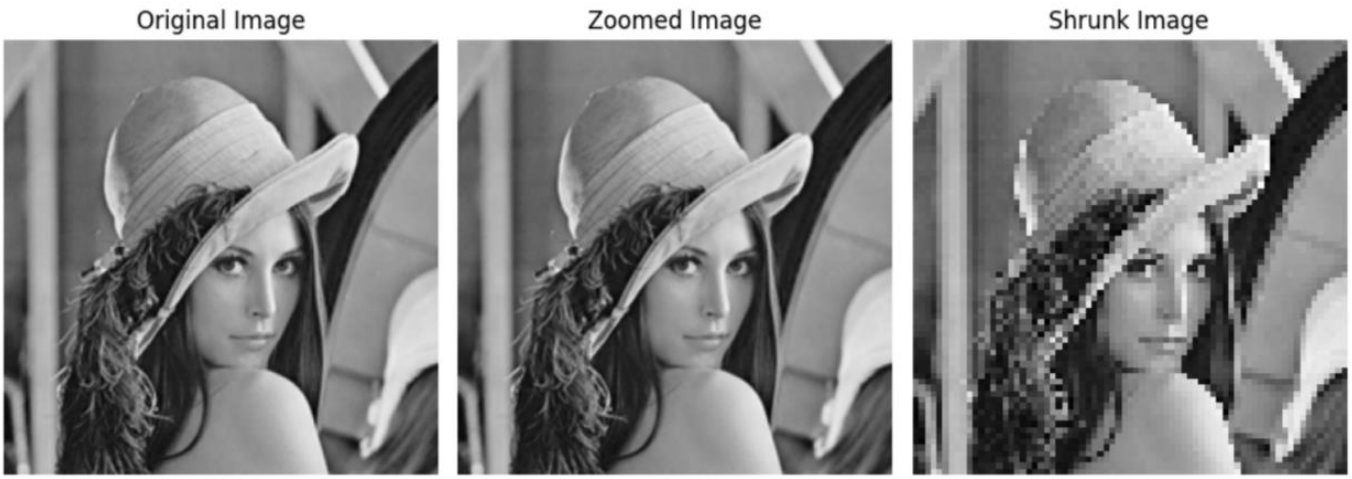
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

def reduce_intensity_levels(image_path, levels):
    # 讀取圖像
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # 計算每個強度級別的間隔
    interval = 256 // levels

    # 減少強度級別
    reduced_image = (image // interval) * interval

    return reduced_image

# 輸入圖像的檔案路徑和所需的強度級別
image_path = "/content/Fig0221(a)(ctskull-256).tif"
desired_levels = [16, 8, 4, 2]

# 呼叫函式進行圖像處理
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

for i, levels in enumerate(desired_levels):
    reduced_image = reduce_intensity_levels(image_path, levels)
    row = i // 2
    col = i % 2
    axs[row, col].imshow(reduced_image, cmap='gray')
    axs[row, col].set_title(f'From 256 to {levels}')
    axs[row, col].axis('off')

plt.tight_layout()
plt.show()
```


From 256 to 16
From 256 to 8
From 256 to 4
From 256 to 2

9. Zooming and Shrinking Images by Pixel Replication (10%) (a) Write a computer program capable of zooming and shrinking an image by pixel replication. Assume that the desired zoom/shrink factors are integers.

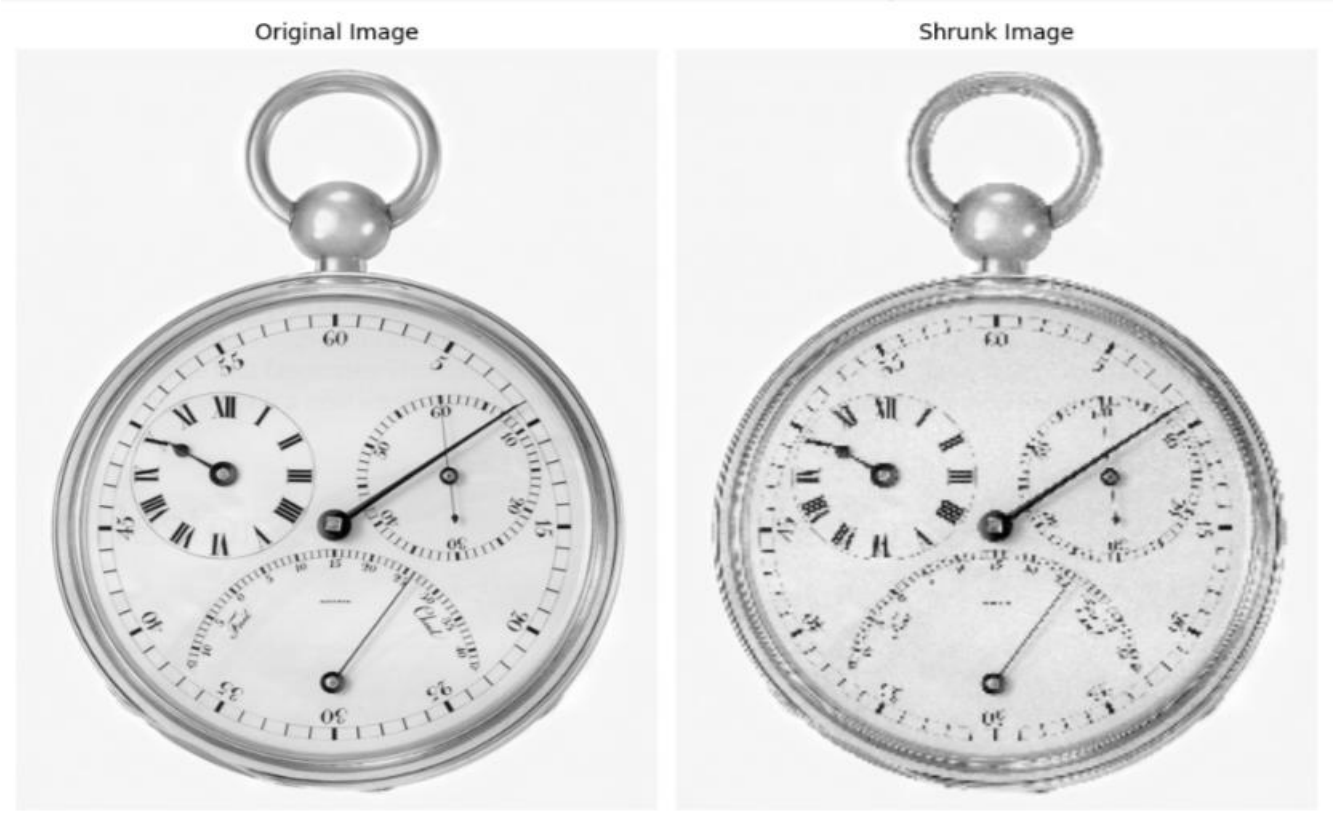
```python
import cv2
from matplotlib import pyplot as plt

def zoom_image(image, zoom_factor):
    height, width = image.shape[:2]
    zoomed_height = height * zoom_factor
    zoomed_width = width * zoom_factor

    zoomed_image = cv2.resize(image, (zoomed_width, zoomed_height), interpolation=cv2.INTER_NEAREST)
    return zoomed_image

def shrink_image(image, shrink_factor):
    height, width = image.shape[:2]
    shrunk_height = height // shrink_factor
    shrunk_width = width // shrink_factor

    shrunk_image = cv2.resize(image, (shrunk_width, shrunk_height), interpolation=cv2.INTER_NEAREST)
    return shrunk_image

# Load the image
image = cv2.imread('/content/Lena.png', cv2.IMREAD_GRAYSCALE)

# Zoom the image by pixel replication
zoom_factor = 20  # Adjust the zoom factor as needed
zoomed_image = zoom_image(image, zoom_factor)

# Shrink the image by pixel replication
shrink_factor = 3  # Adjust the shrink factor as needed
shrunk_image = shrink_image(image, shrink_factor)
```

```python
# Display the zoomed and shrunk images
plt.figure(figsize=(10, 8))
plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title('Zoomed Image')
plt.imshow(zoomed_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title('Shrunk Image')
plt.imshow(shrunk_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```




(b) Download Fig. 2.20(a) from the course web site and use your program to shrink the image by a factor of 12.

```python
import cv2
from matplotlib import pyplot as plt

def shrink_image(image, shrink_factor):
    height, width = image.shape[:2]
    shrunk_height = height // shrink_factor
    shrunk_width = width // shrink_factor

    shrunk_image = cv2.resize(image, (shrunk_width, shrunk_height), interpolation=cv2.INTER_NEAREST)
    return shrunk_image

# Load the image
image = cv2.imread('/content/Fig0220(a)(chronometer 3692x2812 2pt25 inch 1250 dpi).tif', cv2.IMREAD_GRAYSCALE)

# Shrink the image by pixel replication
shrink_factor = 12  # Adjust the shrink factor as needed
shrunk_image = shrink_image(image, shrink_factor)

# Display the original and shrunk images
plt.figure(figsize=(10, 8))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Shrunk Image')
plt.imshow(shrunk_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
# Save the shrunk image
cv2.imwrite('/content/shrunk_clock.png', shrunk_image)
```




True

(c) Use your program to zoom the image in (b) back to the resolution of the original. Explain the reasons for their differences.

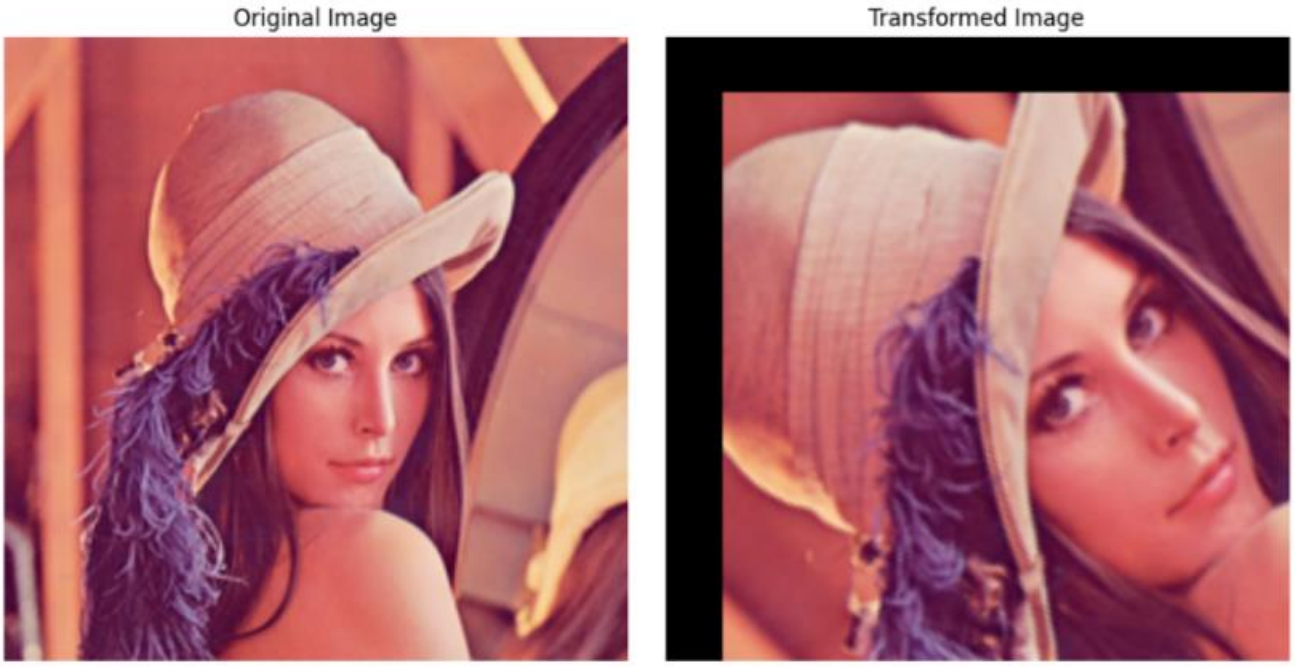
A: 會有落差的原因是因為當進行縮小或放大時會在進行內插的時候丟失一些特徵值，導致有些細節被丟失，所以就算縮放回原本大小也會和原圖產生落差。

```
import cv2
from matplotlib import pyplot as plt

def zoom_image(image, zoom_factor):
    height, width = image.shape[:2]
    zoomed_height = height * zoom_factor
    zoomed_width = width * zoom_factor

    zoomed_image = cv2.resize(image, (zoomed_width, zoomed_height), interpolation=cv2.INTER_NEAREST)
    return zoomed_image

# Load the image
image = cv2.imread('/content/shrunk_clock.png', cv2.IMREAD_GRAYSCALE)
origin = cv2.imread('/content/Fig0220(a)(chronometer 3692x2812 2pt25 inch 1250 dpi).tif', cv2.IMREAD_GRAYSCALE)
# Zoom the image by pixel replication
zoom_factor = 12  # Adjust the zoom factor as needed
zoomed_image = zoom_image(image, zoom_factor)
```

```
# Display the zoomed and shrunk images
plt.figure(figsize=(10, 8))
plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(origin, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title('After Shrunk and Zoomed Image')
plt.imshow(zoomed_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```


10. Image Rotation, Scaling, Translation and Intensity Interpolation (10%) (a) Write a computer program capable of rotating, scaling, and translating an image by specified degree, ratio, and pixels. The rotation degree, scaling ratio, and translating pixels need to be variable inputs to your program.

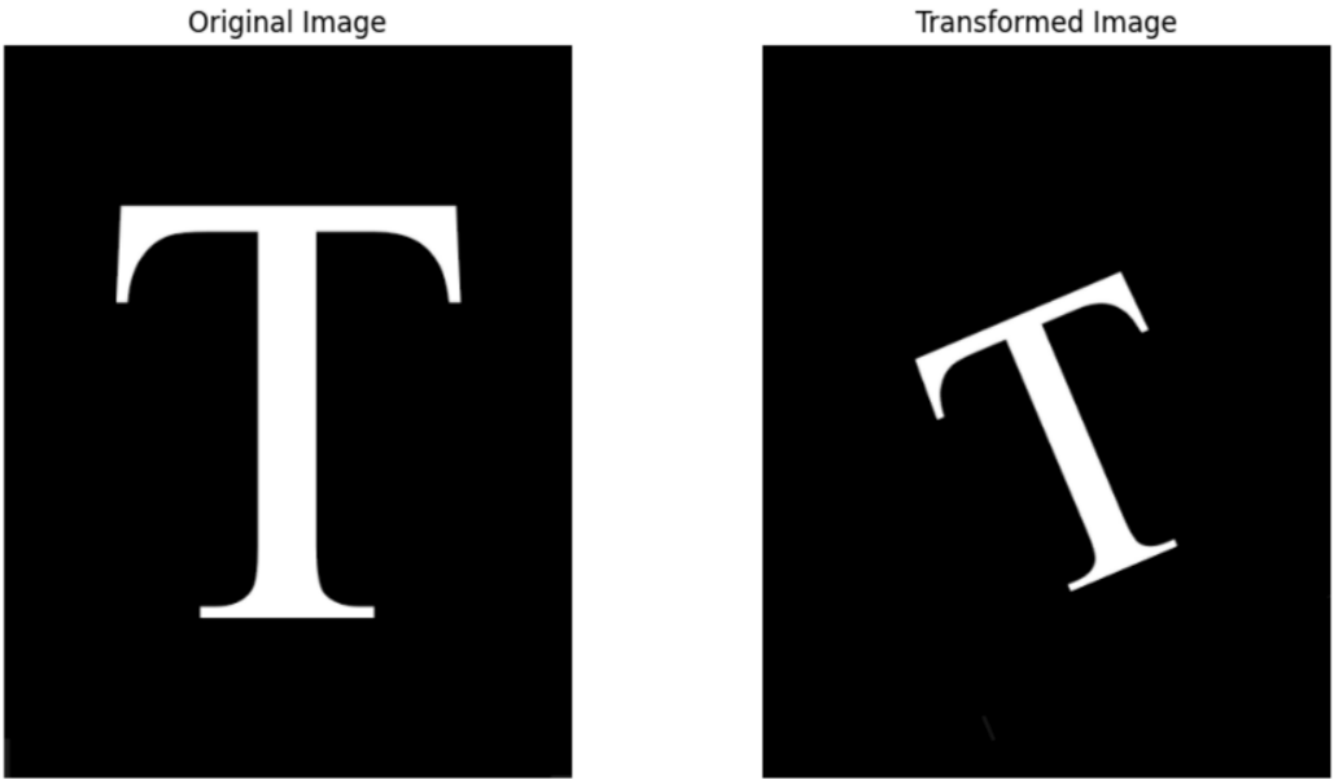
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

def transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels):
    # Read the image
    image = cv2.imread(image_path)

    # Get image dimensions
    height, width = image.shape[:2]

    # Calculate the rotation center
    center = (width // 2, height // 2)

    # Define the rotation matrix
    rotation_matrix = cv2.getRotationMatrix2D(center, rotation_degree, scaling_ratio)

    # Perform the rotation
    rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))

    # Define the translation matrix
    translation_matrix = np.float32([[1, 0, translation_pixels[0]], [0, 1, translation_pixels[1]]])

    # Perform the translation
    translated_image = cv2.warpAffine(rotated_image, translation_matrix, (width, height))

    # Display the original and transformed images
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(translated_image, cv2.COLOR_BGR2RGB))
    plt.title("Transformed Image")
    plt.axis("off")

    plt.tight_layout()
    plt.show()


image_path = "/content/Lena.png"

# Specify the rotation degree, scaling ratio, and translation pixels
rotation_degree = 45
scaling_ratio = 1.5
translation_pixels = (20, 20)

# Apply the transformations to the image
transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels)
```




(b) Download Fig. 2.36 (a) from the course web site and rotate the image 23 ° clockwise, scale it to 2/3 of original size, and shift it by 18 and 22 pixels in x and y directions, respectively. Show the results using three interpolation approaches mentioned in the textbook. Please also zoom in the results to compare the differences as shown in Figs. 2.36 (b) – (d).

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

def transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels):
    # Read the image
    image = cv2.imread(image_path)

    # Get image dimensions
    height, width = image.shape[:2]

    # Calculate the rotation center
    center = (width // 2, height // 2)

    # Define the rotation matrix
    rotation_matrix = cv2.getRotationMatrix2D(center, rotation_degree, scaling_ratio)

    # Perform the rotation
    rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))

    # Define the translation matrix
    translation_matrix = np.float32([[1, 0, translation_pixels[0]], [0, 1, translation_pixels[1]]])

    # Perform the translation
    translated_image = cv2.warpAffine(rotated_image, translation_matrix, (width, height))
```

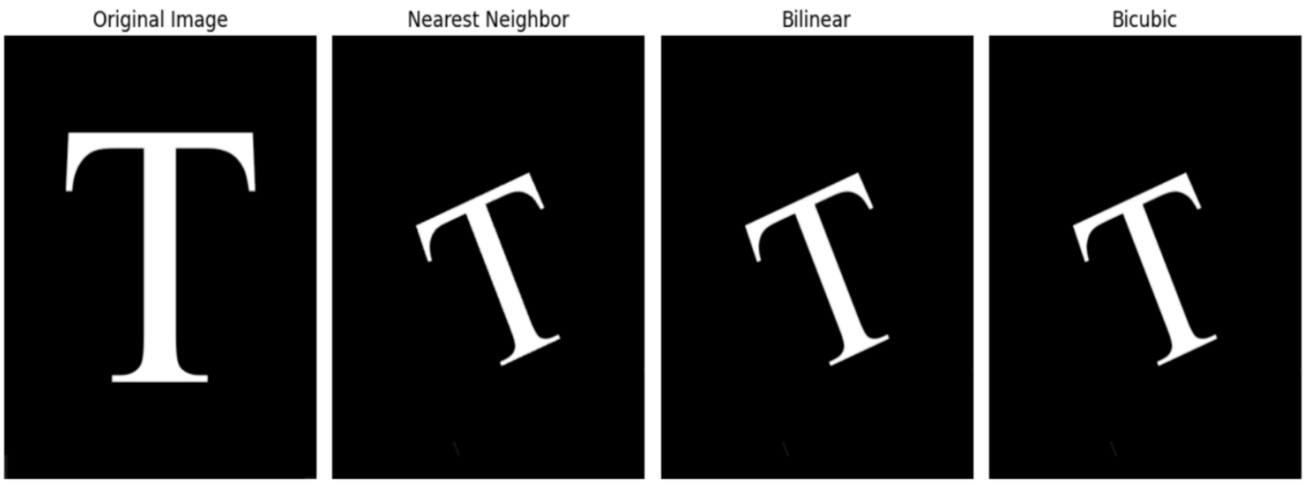
```
        # Display the original and transformed images
        plt.figure(figsize=(10, 5))
        plt.subplot(1, 2, 1)
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.title("Original Image")
        plt.axis("off")

        plt.subplot(1, 2, 2)
        plt.imshow(cv2.cvtColor(translated_image, cv2.COLOR_BGR2RGB))
        plt.title("Transformed Image")
        plt.axis("off")

        plt.tight_layout()
        plt.show()


image_path = "/content/Fig0236(a)(letter_T).tif"

# Specify the rotation degree, scaling ratio, and translation pixels
rotation_degree = 23
scaling_ratio = 2/3
translation_pixels = (18,22)

# Apply the transformations to the image
transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels)
```




下一頁使用課本介紹的三種內插法進行比較

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

def transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels, interpolation):
    # Read the image
    image = cv2.imread(image_path)

    # Get image dimensions
    height, width = image.shape[:2]

    # Calculate the rotation center
    center = (width // 2, height // 2)

    # Define the rotation matrix
    rotation_matrix = cv2.getRotationMatrix2D(center, rotation_degree, scaling_ratio)

    # Perform the rotation
    rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height), flags=interpolation)

    # Define the translation matrix
    translation_matrix = np.float32([[1, 0, translation_pixels[0]], [0, 1, translation_pixels[1]]])

    # Perform the translation
    translated_image = cv2.warpAffine(rotated_image, translation_matrix, (width, height), flags=interpolation)

    return translated_image

# Specify the image file path
image_path = "/content/Fig0236(a)(letter_T).tif"

# Specify the rotation degree, scaling ratio, and translation pixels
rotation_degree = 23
scaling_ratio = 2/3
translation_pixels = (18,22)

# Apply the transformations to the image using different interpolation methods
nearest_neighbor_image = transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels, cv2.INTER_NEAREST)
bilinear_image = transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels, cv2.INTER_LINEAR)
bicubic_image = transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels, cv2.INTER_CUBIC)

# Display the images
plt.figure(figsize=(12, 4))
```

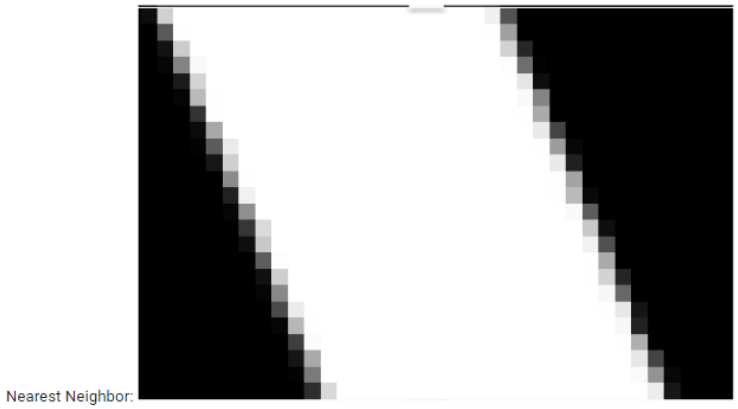
```
# Apply the transformations to the image using different interpolation methods
nearest_neighbor_image = transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels, cv2.INTER_NEAREST)
bilinear_image = transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels, cv2.INTER_LINEAR)
bicubic_image = transform_image(image_path, rotation_degree, scaling_ratio, translation_pixels, cv2.INTER_CUBIC)

# Display the images
plt.figure(figsize=(12, 4))

plt.subplot(1, 4, 1)
plt.imshow(cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 4, 2)
plt.imshow(cv2.cvtColor(nearest_neighbor_image, cv2.COLOR_BGR2RGB))
plt.title("Nearest Neighbor")
plt.axis("off")

plt.subplot(1, 4, 3)
plt.imshow(cv2.cvtColor(bilinear_image, cv2.COLOR_BGR2RGB))
plt.title("Bilinear")
plt.axis("off")

plt.subplot(1, 4, 4)
plt.imshow(cv2.cvtColor(bicubic_image, cv2.COLOR_BGR2RGB))
plt.title("Bicubic")
plt.axis("off")

plt.tight_layout()
plt.show()
```



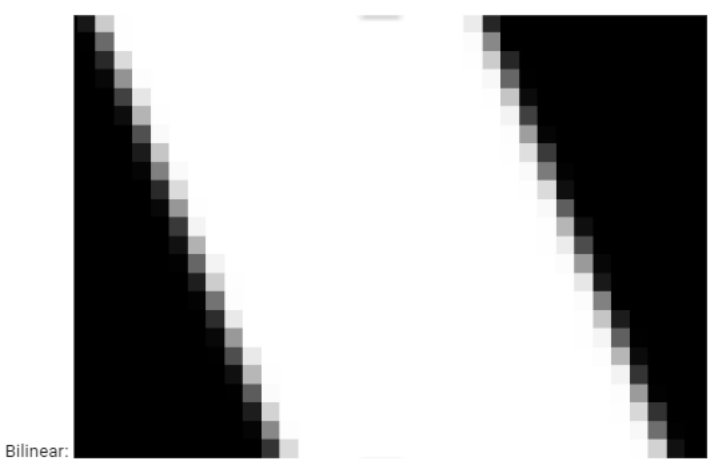
比較三者邊緣細節差異:

1) Nearest Neighbor:

比較三者邊緣細節




2) Bilinear:




3) Bicubic: