S1154007 賴宥瑋 計算機視覺作業

Table of contents

一、Threshold

1-1. Local method

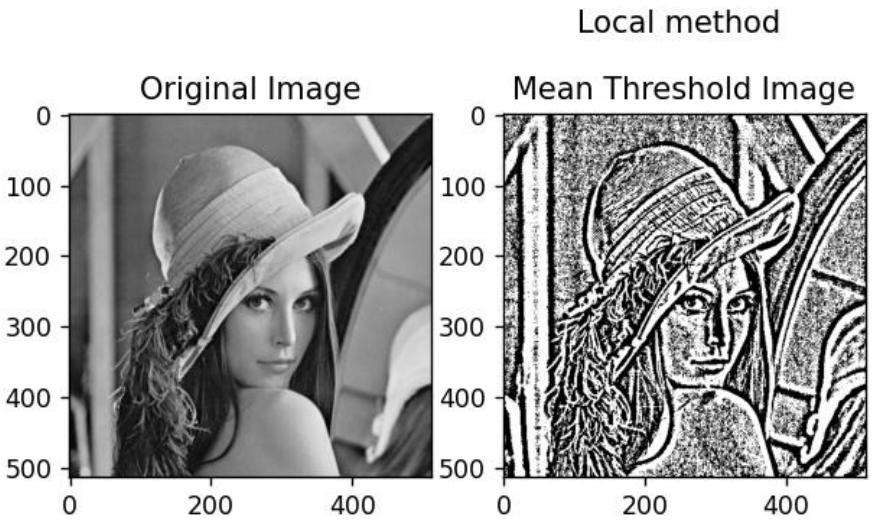
1-1-1. Mean thresholding:

做法:利用 blur 做平均強度，當影像數值>=local_mean，變成白色；反之則變成黑色

```
def local_mean_thresholding(image, window_size=15):
    # 如果影像是彩色的，轉換成灰階
    if len(image.shape) > 2:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # 計算局部的平均強度
    local_mean = cv2.blur(image, (window_size, window_size))

    # 使用局部平均強度作為閾值進行二值化處理
    binary_image = np.where(image >= local_mean, 255, 0).astype(np.uint8)

    return binary_image # 返回圖片
```

```
local_mean_img = local_mean_thresholding(image)
```

```
plt.figure(figsize=(15, 5))
plt.subplot(1, 5, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')

plt.subplot(1, 5, 2)
plt.title('Local method \n\nMean Threshold Image')
plt.imshow(local_mean_img, cmap='gray')
```



1-1-2. Niblack's method

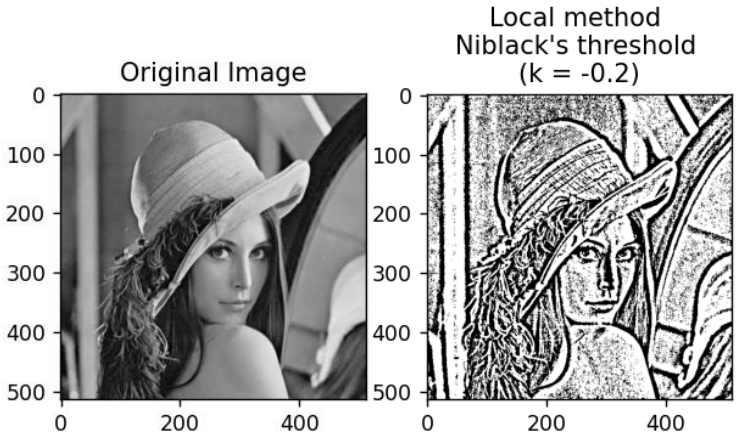
做法:

1. 利用 blur 計算 mean 和 mean 平方
2. 計算標準差
3. 利用公式計算 T(T = μ + k * σ)
4. 當影像數值>=T，變成白色；反之則變成黑色

```python
def niblack_thresholding(image, window_size=15, k=-0.2):
    # 如果影像是彩色的，轉換成灰階
    if len(image.shape) > 2:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # 計算局部的平均強度
    local_mean = cv2.blur(image, (window_size, window_size))

    # 計算局部的標準差
    local_sq_mean = cv2.blur(image**2, (window_size, window_size)) # 平方均值
    local_std_dev = np.sqrt(local_sq_mean - local_mean**2) # 局部的標準差

    # 計算 Niblack 閾值 T = μ + k * σ
    niblack_threshold = local_mean + k * local_std_dev

    # 使用 Niblack 閾值進行二值化處理
    binary_image = np.where(image >= niblack_threshold, 255, 0).astype(np.uint8)

    return binary_image #返回圖片
```

```python
niblack_img = niblack_thresholding(image)
```

```python
plt.subplot(1, 5, 3)
plt.title('Local method \nNiblack\'s threshold \n(k = -0.2)')
plt.imshow(niblack_img, cmap='gray')
```



1-2 Global method

1-2-1. Variance-based thresholding : Otsu algorithm

做法:(按照講義步驟)

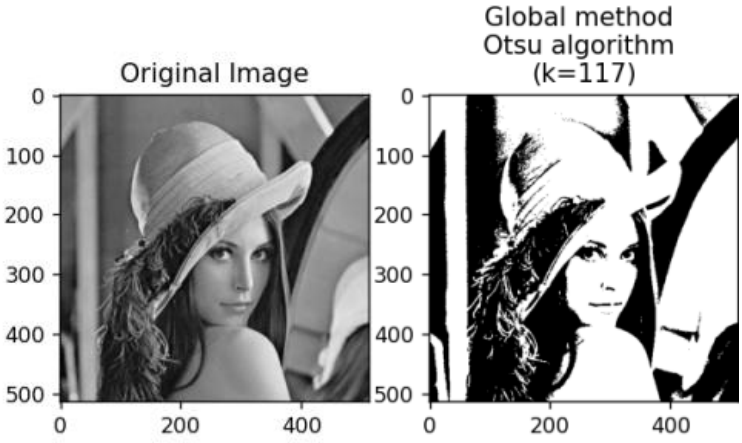
1. 計算各像素的累積值並計算分布機率
2. 計算前景的「累積」機率(Pi)　使用 cumsum:[1,2,3,4] -> [1,3,6,9]
3. 計算累積均值(i*Pi)
4. 計算全域均值(mG)
5. 遍歷所有可能的閥值(0~255)，找到最大 between-class variance 時，T 的值
6. 當影像數值>= T 的**平均**，變成白色；反之則變成黑色

```python
def otsu_thresholding(image):
    # Step 1: 計算正規化的histogram
    histogram, _ = np.histogram(image, bins=256, range=(0, 256))  # 計算直方圖
    histogram = histogram / float(np.sum(histogram))  # 計算機率分布

    # Step 2: 計算累積和 P1(k)
    P1 = np.cumsum(histogram)  # 前景累積機率

    # Step 3: 累積均值 m(k)
    mean_k = np.cumsum(histogram * np.arange(256))  # 累積均值(i*Pi)

    # Step 4: 計算全域均值 mG
    mG = mean_k[-1]  # 全域均值(mean_k的最後一項)

    # Step 5: 初始化變數
    max_sigma_b_squared = -np.inf  # 最大類間方差
    best_thresholds = []  # 用來存儲對應最大sigma_b_squared的閾值
    epsilon = 1e-10  # 避免除以0
```

```python
    # Step 6: 遍歷所有可能的閾值
    for T in range(256):
        sigma_b_squared = ((mG * P1[T] - mean_k[T]) ** 2) / (P1[T] * (1 - P1[T]) + epsilon)

        # 如果找到新的最大sigma_b_squared，清除舊的最佳閾值，存入新的閾值
        if sigma_b_squared > max_sigma_b_squared:
            max_sigma_b_squared = sigma_b_squared
            best_thresholds = [T]

        # 如果sigma_b_squared等於當前最大值，將閾值加入列表
        elif sigma_b_squared == max_sigma_b_squared:
            best_thresholds.append(T)

    # 如果發現多個最佳閾值，返回它們的平均值
    if len(best_thresholds) > 1:
        best_otsu_threshold = np.mean(best_thresholds)
    else:
        best_otsu_threshold = best_thresholds[0]

    return best_otsu_threshold
```

```
otsu_threshold_value = otsu_thresholding(image)  # 獲取 Otsu 閾值
otsu_img = np.where(image >= otsu_threshold_value, 255, 0)  # 使用 Otsu 閾值進行二值化
```

```
plt.subplot(1, 5, 4)
plt.title(f'Global method \nOtsu algorithm \n(k={otsu_threshold_value})')
plt.imshow(otsu_img, cmap='gray')
```



1-2-2. Entropy-based thresholding

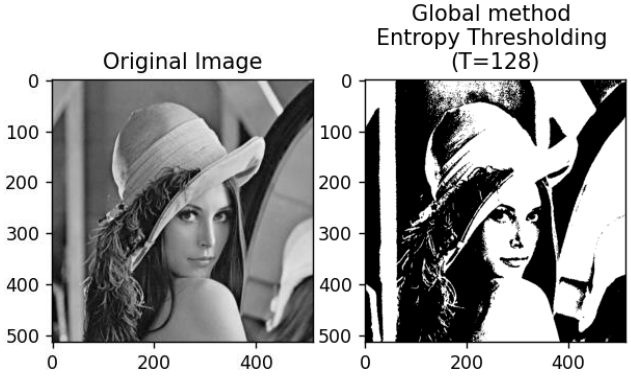
做法:

1. 計算各像素的累積值並計算分布機率
2. 計算前景和背景的「累積」機率
3. 遍歷所有可能的閥值(0~255),計算前景 entropy 跟背景 entropy
4. 找出前景 entropy+背景 entropy 最大時，T 的值
5. 當影像數值>= T，變成白色；反之則變成黑色

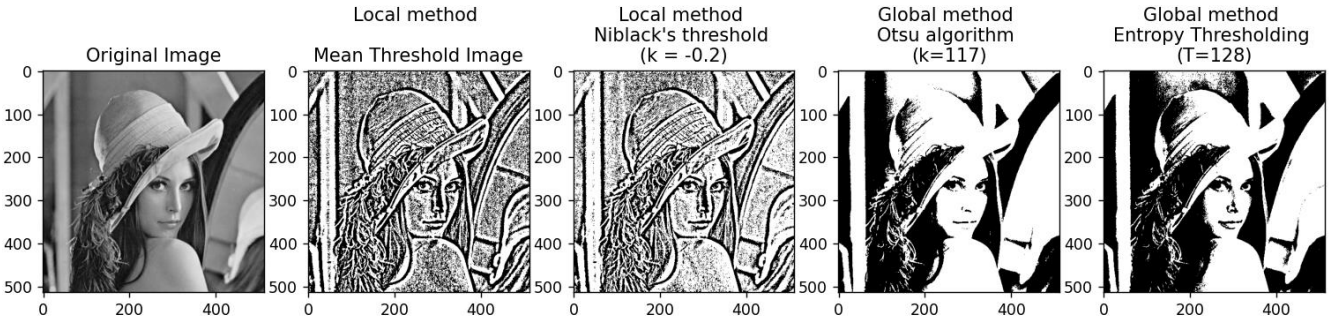
```python
def entropy_thresholding(image):
    # Step 1: 計算直方圖並正規化
    histogram, _ = np.histogram(image, bins=256, range=(0, 256))
    histogram = histogram / float(np.sum(histogram)) # 計算機率分布

    # Step 2: 計算前景和背景的累積機率
    P1 = np.cumsum(histogram)  # 前景累積機率
    P2 = 1 - P1  # 背景累積機率

    max_total_entropy = -np.inf
    best_entropy_threshold = 0  # 最佳閾值
    epsilon = 1e-10 # 避免log2(0)
    # Step 3: 遍歷所有可能的閾值
    for T in range(256):
        foreground_entropy = - (P1[T] * np.log2(P1[T] + epsilon) )  # foreground entropy
        background_entropy = - (P2[T] * np.log2(P2[T] + epsilon) )  # background entropy

        total_entropy = foreground_entropy + background_entropy

        # 找到最大total_entropy時的閾值
        if total_entropy > max_total_entropy:
            max_total_entropy = total_entropy
            best_entropy_threshold = T

    return best_entropy_threshold
```

```python
entropy_threshold_value = entropy_thresholding(image) # 獲取 entropy 閾值
entropy_img = np.where(image >= entropy_threshold_value, 255, 0).astype(np.uint8) #使用 entropy 閾值進行二值化
```

```python
plt.subplot(1, 5, 5)
plt.title(f'Global method \nEntropy Thresholding \n(T={entropy_threshold_value})')
plt.imshow(entropy_img, cmap='gray')


plt.show()
```



1-3. All Result



二、Noise reduction in color image

2-1. RGB to HSV

Functions:

(1) RGB -> HSV：將圖片從 RGB 換到 HSV

```python
def rgb_to_hsv(r, g, b): # RGB -> HSV
    r_prime = r / 255.0
    g_prime = g / 255.0
    b_prime = b / 255.0

    c_max = max(r_prime, g_prime, b_prime)
    c_min = min(r_prime, g_prime, b_prime)
    delta = c_max - c_min

    # RGB -> HSV
    # 計算 Hue
    if delta == 0:
        h = 0
    elif c_max == r_prime:
        h = 60 * (((g_prime - b_prime) / delta) % 6)
    elif c_max == g_prime:
        h = 60 * (((b_prime - r_prime) / delta) + 2)
    elif c_max == b_prime:
        h = 60 * (((r_prime - g_prime) / delta) + 4)
    # 計算 Saturation
    if c_max == 0:
        s = 0
    else:
        s = delta / c_max

    # 計算 Value
    v = c_max
```

(2) HSV -> RGB：將圖片從 HSV 換到 RGB

```python
def hsv_to_rgb(h, s, v): # HSV -> RGB
    c = v * s
    h_prime = h / 60
    x = c * (1 - abs(h_prime % 2 - 1))

    if 0 <= h_prime < 1:
        r1, g1, b1 = (c, x, 0)
    elif 1 <= h_prime < 2:
        r1, g1, b1 = (x, c, 0)
    elif 2 <= h_prime < 3:
        r1, g1, b1 = (0, c, x)
    elif 3 <= h_prime < 4:
        r1, g1, b1 = (0, x, c)
    elif 4 <= h_prime < 5:
        r1, g1, b1 = (x, 0, c)
    elif 5 <= h_prime < 6:
        r1, g1, b1 = (c, 0, x)

    m = v - c
    r, g, b = (r1 + m) * 255, (g1 + m) * 255, (b1 + m) * 255

    return r, g, b
```

(3) 高斯濾波(2D)數值計算 : 產生高斯濾波矩陣

```
def gaussian_filter_2d(shape): # 高斯濾波(2D)數值計算
    m, n = [(ss-1)//2 for ss in shape] # m, n為產生網格用
    x, y = np.ogrid[-m:m+1, -n:n+1] # 產生網格座標 : 例如kernel_size = 3*3 -> (-1, -1) (0, -1) (1, -1) .....
    h = np.exp(-(x*x + y*y) / (2.*sigma*sigma)) # 公式計算
    h = h / (2. * np.pi * sigma * sigma) # 公式計算
    return h / h.sum() # Normalization
```

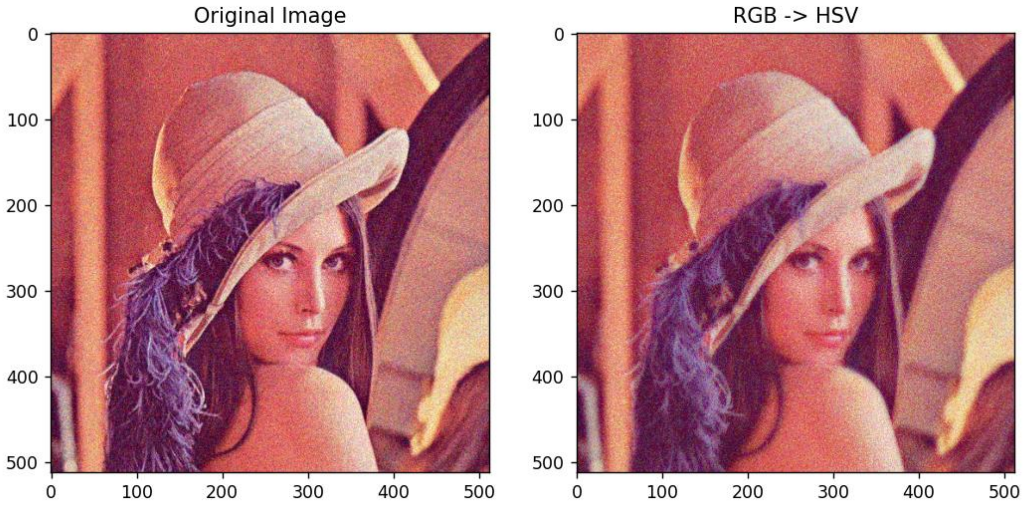
(4) 將高斯濾波應用於 **v 維度**

```
def apply_gaussian_to_v(hsv_image): # 將高斯濾波應用於v維度
    v_channel = hsv_image[:, :, 2] # 將HSV的V channel取出

    # 將高斯濾波應用於v維度
    gaussian_filter = gaussian_filter_2d((kernel_size, kernel_size))
    v_filtered = cv2.filter2D(v_channel, -1, gaussian_filter) # -1 : 輸出圖像的數據類型與輸入圖像相同

    hsv_filtered = hsv_image.copy() # 複製原圖片
    hsv_filtered[:, :, 2] = v_filtered # 將濾完波的V channel跟原本的H S合併
    return hsv_filtered
```

(5)結合(RGB -> HSV -> 高斯濾波 -> RGB)

```
# 讀取圖像
image = cv2.imread('noise.bmp')  # 替換為你的圖像路徑
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # BGR 色彩空間轉換為 RGB

# RGB to HSV , HSV 數值為小數
hsv_image = np.zeros_like(image, dtype=np.float32)
# 兩個for迴圈是指圖片大小
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        r, g, b = image[i, j] # 將RGB取出
        h, s, v = rgb_to_hsv(r, g, b)
        hsv_image[i, j] = [h, s, v]

kernel_size = 19
sigma = 3
# 將高斯濾波應用於v維度
hsv_filtered = apply_gaussian_to_v(hsv_image)
# HSV to RGB , RGB為0-255的數值
filtered_image = np.zeros_like(image, dtype=np.uint8)
# 兩個for迴圈是指圖片大小
for i in range(hsv_filtered.shape[0]):
    for j in range(hsv_filtered.shape[1]):
        h, s, v = hsv_filtered[i, j] # 將HSV取出
        r, g, b = hsv_to_rgb(h, s, v)
        filtered_image[i, j] = [r, g, b]

# 顯示原圖和濾波後的圖
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(filtered_image)
plt.title('RGB -> HSV')

plt.show()
```

Result



2-2. Vector (Median) filtering

做法:

1.需要填充(padding)，採用的是 BORDER_REFLECT 的方法

2.將 kernel 從最左上角開始放

3.將窗口內每個像素的 c 通道展開，以便計算歐式距離

4.開始遍歷窗口內的每個像素，計算到其餘 8 個點的距離

5.找到最小值後替換成該向量

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

def vector_median_filter(image):
    h, w, c = image.shape # 獲取圖像的尺寸
    pad_size = kernel_size // 2 # padding用
    padded_image = cv2.copyMakeBorder(image, pad_size, pad_size, pad_size, pad_size, cv2.BORDER_REFLECT) # padding，使用BORDER_REFLECT的方法
    output_image = np.zeros_like(image) # 創建一個shape為 h*w*c 的全0陣列

    for i in range(h):
        for j in range(w):
            window = padded_image[i:i+kernel_size, j:j+kernel_size] # 獲取窗口(從填充的邊界算)
            window_vectors = window.reshape(-1, 3)  # 將窗口展平為每個像素的向量，以便計算歐式距離

            min_distance_sum = float('inf') # 距離總和
            median_vector = None # 初始化向量

            # 計算每個點到其餘8個點的距離總和
            for v in window_vectors:
                # 計算距離總和
                distances = np.linalg.norm(window_vectors - v, axis=1)  # 使用歐氏距離(對於列 axis=1)
                distance_sum = np.sum(distances) # 距離總和
                if distance_sum < min_distance_sum: # 找出距離總和最小的
                    min_distance_sum = distance_sum
                    median_vector = v

            output_image[i, j] = median_vector # 替換成距離總和最小的向量

    return output_image
```

```python
# 讀取原始圖像
image = cv2.imread('noise.bmp')

# 設定窗口大小
kernel_size = 3

# 應用向量中值濾波
vector_median_image = vector_median_filter(image)

# 將圖像從 BGR 轉換為 RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
filtered_image_rgb = cv2.cvtColor(vector_median_image, cv2.COLOR_BGR2RGB)

# 顯示原圖和濾後圖像
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(filtered_image_rgb)
plt.title('Vector (Median) filtering')

plt.show()
```

Result: