第一部分「景深擴張」

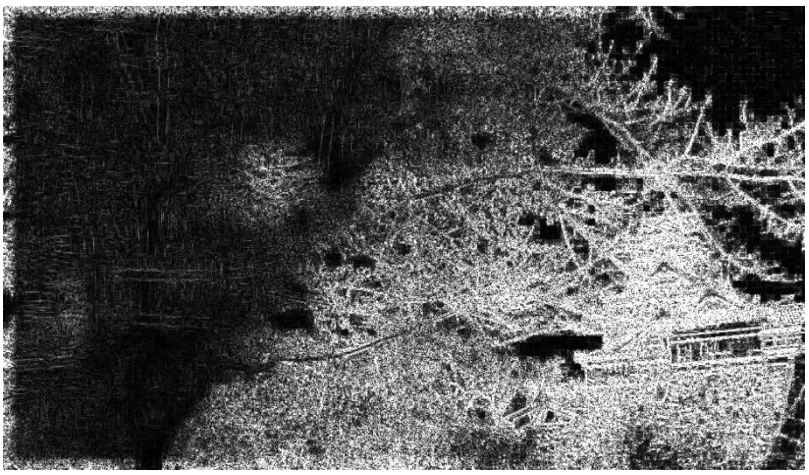
```
#匯入此作業使用到的模組
import cv2 #OpenCV 函式庫，用於圖像處理
import numpy as np #NumPy 函式庫，用於數值計算
import matplotlib.pyplot as plt #Matplotlib 函式庫，用於圖像顯示

#將圖像轉換為浮點數格式
def img_float(img):
    float_img = img / 255 #將像素值從 0-255 轉換為 0-1
    float_img = float_img.astype('float32') #轉換為 32 位元浮點數

    return float_img

#對圖像進行二值化處理
def image_threshold(img, thrsdh, maxval):
    height, width = img.shape[:2] #獲取圖像的高度和寬度

    #用雙層迴對每個像素進行二值化
    for i in range(height):
        for j in range(width):
            if img[i][j] > 0:
                img[i][j] = maxval #大於 threshold 的像素設為最大值
            else:
                img[i][j] = thrsdh #小於 threshold 的像素設為 threshold

    return img

#Laplacian 高通濾波
def laplacian(img):
    float_img = img_float(img) #將圖像轉換為浮點數格式
    img_gray = cv2.cvtColor(float_img, cv2.COLOR_RGB2GRAY) #轉換為灰度圖像

    #自訂八方向 Laplacian 濾鏡
    laplacian_f = np.array([[-1, -1, -1],
                            [-1,  8, -1],
                            [-1, -1, -1]])

    #應用拉普拉斯濾波器
    laplacian_img = cv2.filter2D(img_gray, -1, laplacian_f)
    laplacian_img = np.abs(laplacian_img) #取絕對值

    laplacian_img = np.clip(laplacian_img * 3, 0, 1) # 強化濾波結果並限制範圍 0~1
    return laplacian_img
```

```python
#處理前景和背景的高通濾波結果，生成遮罩
def mask_process(fg_hipass, bg_hipass):
    mask = fg_hipass - bg_hipass #計算前景遮罩

    #應用均值濾波
    kernel_size = 15
    mean_f_kernal = np.ones((kernel_size, kernel_size), np.float32) / (kernel_size ** 2)
    mean_f_mask = cv2.filter2D(mask, -1, mean_f_kernal)

    #對遮罩進行二值化處理
    threshold_mask = image_threshold(mean_f_mask, 0, 255)

    #顯示並保存二值化後的遮罩
    plt.imshow(threshold_mask, cmap='gray')
    plt.title('二值化後的前景遮罩')
    plt.axis('off')
    plt.savefig('threshold_mask.jpg', dpi=300, bbox_inches='tight', pad_inches=0)
    plt.show()

    return threshold_mask

if __name__ == "__main__":
    #讀取前景和背景圖像
    fg_img = './depth of field/5fg.JPG'
    bg_img = './depth of field/5bg.JPG'

    #讀取並轉換色彩空間從 BGR 到 RGB
    fg = cv2.cvtColor(cv2.imread(fg_img, cv2.IMREAD_COLOR), cv2.COLOR_BGR2RGB)
    bg = cv2.cvtColor(cv2.imread(bg_img, cv2.IMREAD_COLOR), cv2.COLOR_BGR2RGB)

    #設置 matplotlib 支援中文顯示
    plt.rcParams['font.sans-serif'] = ['Microsoft JhengHei']
    plt.rcParams['axes.unicode_minus'] = False

    #顯示原始的前景和背景圖像
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(fg)
    plt.title('對焦在前景(fg)')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(bg)
```

```
plt.title('對焦在背景(bg)')
plt.axis('off')
plt.show()

#對前景圖像進行 Laplacian 高通濾波，顯示儲存結果影像
fg_hipass = laplacian(fg)
plt.imshow(fg_hipass, cmap='gray')
plt.title('fg_hipass')
plt.axis('off')
plt.savefig('fg_hipass.jpg', dpi=300, bbox_inches='tight', pad_inches=0)
plt.show()

#對背景圖像進行 Laplacian 高通濾波，顯示並儲存結果影像
bg_hipass = laplacian(bg)
plt.imshow(bg_hipass, cmap='gray')
plt.title('bg_hipass')
plt.axis('off')
plt.savefig('bg_hipass.jpg', dpi=300, bbox_inches='tight', pad_inches=0)
plt.show()

#生成並處理遮罩，轉換為 uint8 格式
threshold_mask = mask_process(fg_hipass, bg_hipass).astype('uint8')

#將遮罩應用到前景與背景
foreground = cv2.bitwise_and(fg, fg, mask=threshold_mask)
background = cv2.bitwise_and(bg, bg, mask=cv2.bitwise_not(threshold_mask))

#合併前景和背景影像
combine_img = foreground + background

#顯示並保存最終合併結果
plt.imshow(combine_img)
plt.title('景深擴增影像')
plt.axis('off')
plt.savefig('景深擴增影像.jpg', dpi=300, bbox_inches='tight', pad_inches=0)
plt.show()
```

第一部分「景深擴張」執行結果

bg_hipass



fg_hipass

二值化後的前景遮罩

景深擴增影像

第二部分「以多維空間分析影像的差異」

```
#匯入此作業使用到的模組
import cv2 #OpenCV 函式庫，用於圖像處理
import os #檔案和目錄操作
import numpy as np #NumPy 函式庫，用於數值計算
import matplotlib.pyplot as plt #Matplotlib 函式庫，用於圖像顯示

#將圖像轉換為浮點數格式
def img_float(img):
    float_img = img / 255 #將像素值從 0-255 轉換為 0-1
    float_img = float_img.astype('float32') #轉換為 32 位元浮點數

    return float_img

#對圖像進行二值化處理
def image_threshold(img, thrsdh, maxval):
    height, width = img.shape[:2] #獲取圖像的高度和寬度

    #用雙層迴對每個像素進行二值化
    for i in range(height):
        for j in range(width):
            if img[i][j] > 0:
                img[i][j] = maxval #大於 threshold 的像素設為最大值
            else:
                img[i][j] = thrsdh #小於 threshold 的像素設為 threshold

    return img

#提取圖像的各種特徵
def img_feature(img):
    float_img = img_float(img) #將圖像轉換為浮點數格式

    #LAB 空間
    img_lab = cv2.cvtColor(float_img, cv2.COLOR_BGR2LAB) #轉換到 LAB 色彩空間
    L = img_lab[:, :, 0] #L 通道：亮度
    a = img_lab[:, :, 1] #a 通道：紅綠對立通道
    b = img_lab[:, :, 2] #b 通道：黃藍對立通道
    L_standard, L_mean = np.std(L), np.mean(L) #計算明度 L 的標準差和平均值
    a_mean, b_mean = np.mean(a), np.mean(b) #計算 a.b 通道的均值

    #HSV 空間
    img_hsv = cv2.cvtColor(float_img, cv2.COLOR_BGR2HSV) #轉換到 HSV 色彩空間
    H = img_hsv[:, :, 0] #色相通道
```

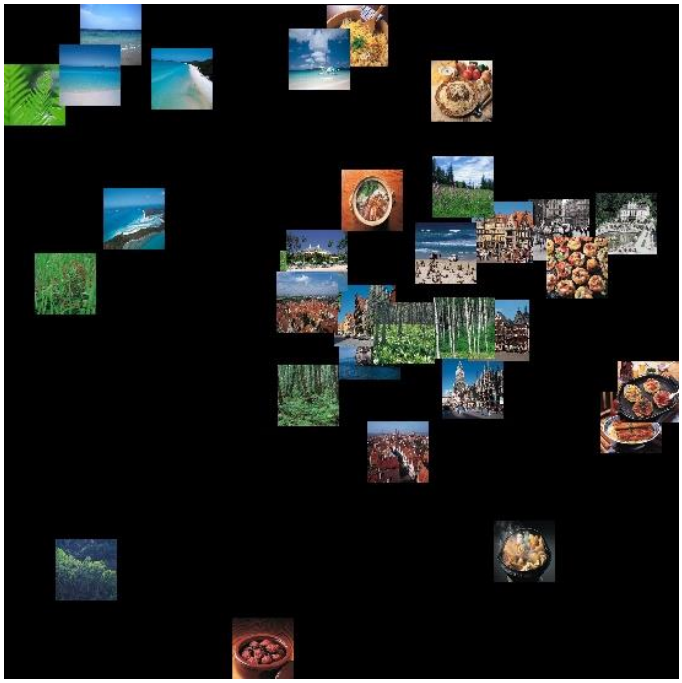
```
    S = img_hsv[:, :, 1] #飽和度通道
    H_mean, S_mean = np.mean(H), np.mean(S) #計算色相和飽和度平均值

    threshold = 5 #設定門檻值

    #高通濾波
    L_laplacian = cv2.Laplacian(L, -1, ksize=3)  #高通濾波
    L_laplacian = np.abs(L_laplacian)  #取絕對值
    _, L_laplacian_threshold = cv2.threshold(L_laplacian, threshold, 255, cv2.THRESH_BINARY)
#二值化
    L_laplacian_threshold_mean = np.mean(L_laplacian_threshold)  # 紋理特徵均值

    #Sobel 垂直線檢測
    L_sobel = cv2.Sobel(L, -1, 1, 0, ksize=3)  #Sobel 濾波器，X 方向（垂直邊緣）
    L_sobel = np.abs(L_sobel)  #取絕對值
    _, L_sobel_threshold = cv2.threshold(L_sobel, threshold, 255, cv2.THRESH_BINARY) #二值化
    L_sobel_threshold_mean = np.mean(L_sobel_threshold)  #垂直邊緣均值

    #返回所有提取的特徵
    return {
        'Contrast': L_standard,
        'Lightness': L_mean,
        'Hue': H_mean,
        'Saturation': S_mean,
        'Red-Green': a_mean,
        'Yellow-Blue': b_mean,
        'Texture': L_laplacian_threshold_mean,
        'Vertical edges': L_sobel_threshold_mean
    }

#對特徵值進行正規化處理，將其值縮放到 0-500 範圍內
def normalize_features(features_list):
    normalized = {}

    #遍歷每個特徵名稱
    for feature in features_list[0].keys():
        values = np.array([f[feature] for f in features_list]) #提取所有圖片在當前特徵上的值，並
轉換為 numpy 陣列
        min_val, max_val = values.min(), values.max() #計算當前特徵值的最小值和最大值

        #當特徵的最大值與最小值相等時，避免除以零的錯誤
        if min_val == max_val:
            #將所有值設置為範圍的中間值（500/2），以保證正規化結果的一致性
```

```
            normalized[feature] = np.full_like(values, 250)
        else:
            #正規化計算公式：將值縮放到 0~500 範圍
            normalized[feature] = np.clip(((values - min_val) / (max_val - min_val) *
500).astype(int), 0, 500)

    return normalized

#在黑色背景圖上繪製二維分析圖
def plot_img_feature(images, features, x_feature, y_feature, output):
    black_img = np.zeros((550, 550, 3), np.uint8) #創建黑色背景

    #對每張圖片進行處理
    for i, img in enumerate(images):
        #計算位置
        x = features[x_feature][i]  #轉換為整數
        y = 500 - features[y_feature][i]   #反轉 Y 軸

        #將圖縮小至 50x50 尺寸
        resize_img = cv2.resize(img, (50, 50))

        #防止超出邊界
        x = min(max(0, x), 500)
        y = min(max(0, y), 500)

        #貼上圖片
        black_img[y:y + 50, x:x + 50] = resize_img

    #儲存處理後的結果
    cv2.imwrite(output, black_img)

    #顯示圖片
    black_img = cv2.cvtColor(black_img, cv2.COLOR_BGR2RGB)
    plt.imshow(black_img)
    plt.xlabel(x_feature, fontsize=12) #設置 x 軸標籤
    plt.ylabel(y_feature, fontsize=12) #設置 y 軸標籤
    plt.xticks([])
    plt.yticks([])
    plt.show()

#針對資料夾中每個圖像做依序處理
def img_process(img_dir):
    images = []
```

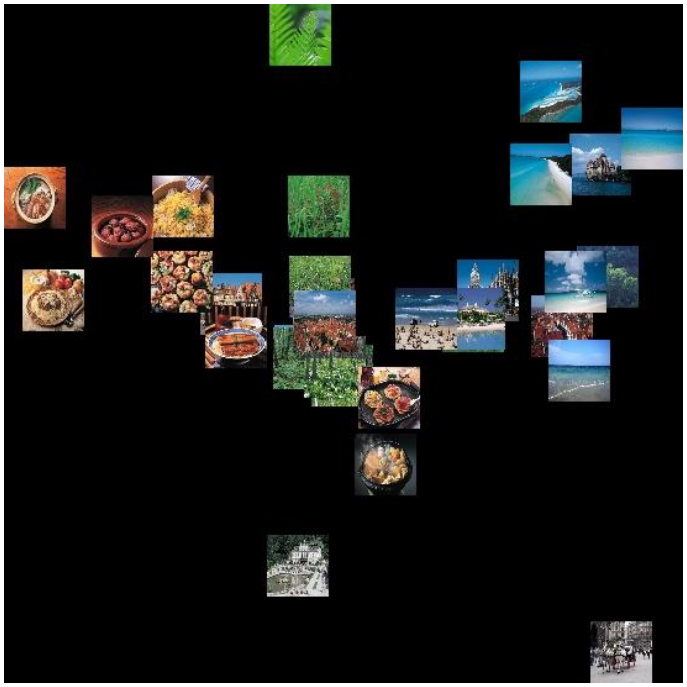
```
    features_list = []

    #讀取並處理所有圖像
    for img_name in sorted(os.listdir(img_dir)): #遍歷指定目錄中的所有影像檔案，並按名稱排序
        img_path = os.path.join(img_dir, img_name) #將影像檔案的名稱與目錄路徑結合
        img = cv2.imread(img_path) #讀取圖像檔案
        images.append(img) #將讀取的圖像添加到 images 陣列中
        features_list.append(img_feature(img)) #提取當前圖像的特徵，並將特徵字典添加到
features_list 陣列中

    #特徵值正規化
    norm_features = normalize_features(features_list)

    #生成不同特徵組合的二維分析圖結果
    plot_img_feature(images, norm_features, 'Contrast', 'Lightness',
'Contrast_Lightness.jpg')
    plot_img_feature(images, norm_features, 'Hue', 'Saturation', 'Hue_Saturation.jpg')
    plot_img_feature(images, norm_features, 'Red-Green', 'Yellow-Blue', 'Red-Green_Yellow-
Blue.jpg')
    plot_img_feature(images, norm_features, 'Texture', 'Vertical edges', 'Texture_Vertical
edges.jpg')

if __name__ == '__main__':
    #目標資料夾
    img_path = './imgbank'
    img_process(img_path)
```

第二部分「以多維空間分析影像的差異」執行結果

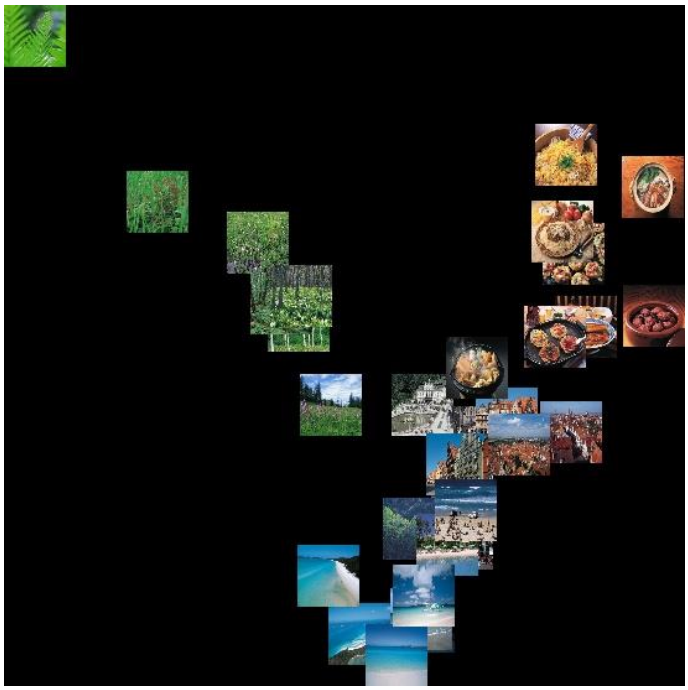
CIELAB 的明度標準差(Contrast)與明度平均值



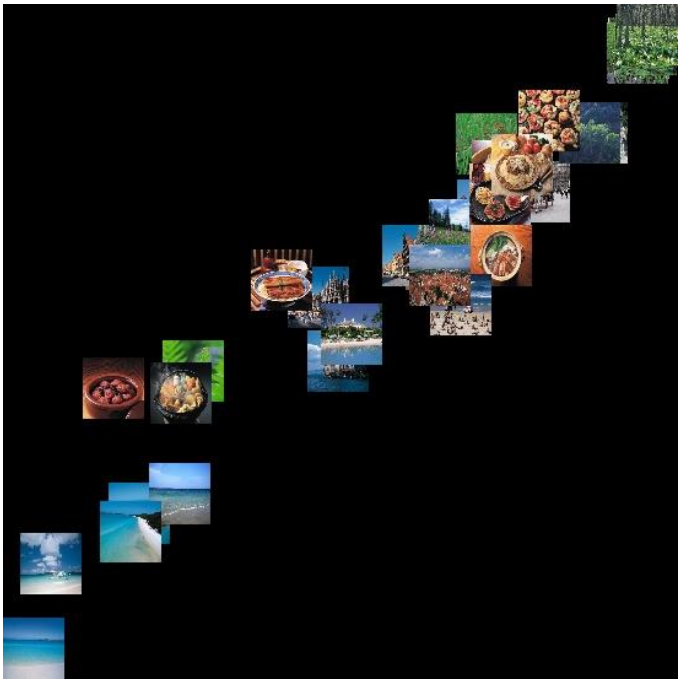
HSV 色空間下的色相(Hue)與飽和度(Saturation)



CIELAB 色空間下的紅綠(a*)與黃藍(b*)



紋理與垂直邊緣



第三部分「視覺異常模擬」

```
#匯入此作業使用到的模組
import cv2 #OpenCV 函式庫，用於圖像處理
import numpy as np #NumPy 函式庫，用於數值計算
import matplotlib.pyplot as plt #Matplotlib 函式庫，用於圖像顯示

#將圖像轉換為浮點數格式
def img_float(img):
    float_img = img / 255 #將像素值從 0-255 轉換為 0-1
    float_img = float_img.astype('float32') #轉換為 32 位元浮點數

    return float_img

#模擬紅綠色盲的視覺效果
def red_green_colorblindness(image):
    float_img = img_float(image) #將圖像轉換為浮點數格式
    float_img_lab = cv2.cvtColor(float_img, cv2.COLOR_RGB2LAB) #將 RGB 色彩空間轉換為 LAB 色彩空
間

    #分別得到 LAB 三個空間
    L = float_img_lab[:, :, 0] #L 通道：亮度
    a = float_img_lab[:, :, 1] #a 通道：紅綠對立通道
    b = float_img_lab[:, :, 2] #b 通道：黃藍對立通道

    #將 a 通道設為 0
    a[:] = 0

    #合併修改後的 LAB 空間
    lab_img_modify = np.zeros_like(float_img_lab)
    lab_img_modify[:, :, 0] = L
    lab_img_modify[:, :, 1] = a
    lab_img_modify[:, :, 2] = b

    #將 LAB 色彩空間轉回 RGB
    rgb_img = cv2.cvtColor(lab_img_modify, cv2.COLOR_LAB2RGB)

    #顯示並儲存結果
    plt.imshow(rgb_img)
    plt.axis('off')
    plt.title('紅綠色盲')
    plt.savefig('red_green_colorblindness.jpg', bbox_inches='tight', pad_inches=0) #儲存圖片並
將空白去除
    plt.show()
```

```
#模擬黃藍色盲的視覺效果
def yellow_blue_colorblindness(img):
    float_img = img_float(img) #將圖像轉換為浮點數格式
    float_img_lab = cv2.cvtColor(float_img, cv2.COLOR_RGB2LAB) #將 RGB 色彩空間轉換為 LAB 色彩空
間

    #分別得到 LAB 三個空間
    L = float_img_lab[:, :, 0] #L 通道：亮度
    a = float_img_lab[:, :, 1] #a 通道：紅綠對立通道
    b = float_img_lab[:, :, 2] #b 通道：黃藍對立通道

    #將 b 通道設為 0
    b[:] = 0

    #合併修改後的 LAB 空間
    lab_img_modify = np.zeros_like(float_img)
    lab_img_modify[:, :, 0] = L
    lab_img_modify[:, :, 1] = a
    lab_img_modify[:, :, 2] = b

    #將 LAB 色彩空間轉回 RGB
    rgb_img = cv2.cvtColor(lab_img_modify, cv2.COLOR_LAB2RGB)

    #顯示並儲存結果
    plt.imshow(rgb_img)
    plt.axis('off')
    plt.title('黃藍色盲')
    plt.savefig('yellow_blue_colorblindness.jpg', bbox_inches='tight', pad_inches=0)
    plt.show()

#模擬青光眼的視覺效果
def glaucoma(img):
    height, width = img.shape[0:2] #獲取圖像的高度和寬度
    sigma = 100 #設定 sigma

    #建立與影像同尺寸的 2D 高斯濾鏡
    gaussian = np.outer(cv2.getGaussianKernel(height, sigma), (cv2.getGaussianKernel(width,
sigma)).T)
    gaussian /= np.max(gaussian) #將濾鏡數值矩陣的每個數值除以其最大值

    # 對圖像應用高斯濾波
    gaussian_img = img.astype('float32')
```

```
    for color in range(3): #對每個色彩通道分別處理
        gaussian_img[:, :, color] *= gaussian

    #將結果裁減回 0~255 範圍，並轉換回 uint8 格式
    gaussian_img = np.clip(gaussian_img, 0, 255).astype('uint8')

    #顯示並儲存結果
    plt.imshow(gaussian_img)
    plt.axis('off')
    plt.title('青光眼')
    plt.savefig('glaucoma.jpg', bbox_inches='tight', pad_inches=0)
    plt.show()

if __name__ == "__main__":
    #讀取原始圖像並轉換色彩空間從 BGR 到 RGB
    original_img = cv2.imread('picture.jpg')
    original_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)

    #設置 matplotlib 支援中文顯示
    plt.rcParams['font.sans-serif'] = ['Microsoft JhengHei']
    plt.rcParams['axes.unicode_minus'] = False

    #顯示原始圖像
    plt.imshow(original_img)
    plt.axis('off') #關閉坐標軸
    plt.title('一般人')
    plt.show()

    red_green_colorblindness(original_img) #模擬紅綠色盲
    yellow_blue_colorblindness(original_img) #模擬黃藍色盲
    glaucoma(original_img) #模擬青光眼
```

第三部分「視覺異常模擬」執行結果

紅綠色盲

黃藍色盲

青光眼