第一部分: 簡易灰階影像分類

```
#匯入此作業使用到的模組
import os #用於操作系統功能，這裡用在清除畫面
import cv2 #OpenCV函式庫，用於圖像處理
import types #用於類型判斷
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
from scipy.special import softmax #Scipy中的softmax函數，用於將數值轉為機率

#初始化
def initialize():
    #清畫面
    os.system('cls' if os.name == 'nt' else 'clear')

    #清變數
    for var in list(globals().keys()):
        if var not in ["__builtins__", "__name__", "__doc__",
"__package__"] and not isinstance(globals()[var], (type(os),
types.FunctionType)):
            del globals()[var]

    #關閉所有figure
    plt.close('all')

    path = 'HW3 instructions\imageset' #設圖庫路徑

    #設拉普拉斯的離散卷積核(3x3濾鏡數值)
    filter = np.array([[-1, -1, -1],
                       [-1,  8, -1],
                       [-1, -1, -1]])

    #建立分類目標值t數值陣列，1~40, 41~80, 81~120的分類編號分別是1,2,3。
    t = np.zeros(120, dtype = int)
    t[0:40] = 1
    t[40:80] = 2
    t[80:120] = 3

    return path, filter, t

#影像特徵擷取
def img_feature_extract(path, filter):
    feature = np.zeros((120, 5)) #120*5的特徵向量矩陣
```

```
    #迴圈依序讀取120個影像
    for i in range(1, 121):
        img_path = f'{path}\{i}.jpg' #影像路徑
        image = cv2.cvtColor(cv2.imread(img_path, cv2.IMREAD_COLOR),
cv2.COLOR_BGR2RGB) #讀取影像並轉換為RGB
        image_float = image.astype(np.float32) / 255.0 #轉換為浮點格式
        image_LAB = cv2.cvtColor(image_float, cv2.COLOR_RGB2LAB) #將影像
格式從RGB轉成LAB格式

        #提取L通道，調整L通道為2x1的大小
        L_channel = image_LAB[:, :, 0]
        L_channel_resize = cv2.resize(L_channel, (1, 2),
interpolation=cv2.INTER_AREA)
        x1, x2 = L_channel_resize[0, 0], L_channel_resize[1, 0] #影像上
半部與下半部的明度值

        #拉普拉斯濾波，取絕對值，調整濾波後的圖像為2x1的大小
        laplacian_LAB = cv2.filter2D(L_channel, -1, filter)
        laplacian_LAB = np.abs(laplacian_LAB)
        laplacian_LAB_resize = cv2.resize(laplacian_LAB, (1, 2),
interpolation=cv2.INTER_AREA)
        x3, x4 = laplacian_LAB_resize[0, 0], laplacian_LAB_resize[1, 0]
#影像上半部與下半部的紋理

        #儲存提取的特徵x1.x2.x3.x4以及常數項x5
        feature[i-1, :] = [x1, x2, x3, x4, 1]

    return feature

#影像分類訓練
def img_classfication_training(feature, t):
    loss_n = 0 #loss change 計數器 n 初始值
    min_loss = np.inf #最小損失 min_loss 值初始值為無限大
    loss_history = []
    accuracy = []

    #建立100萬次迭代(itr)的迴圈
    for itr in range(1, 1000000):
        #用rand函式生成三組5x1尺寸的隨機係數a1,a2,a3，數值範圍落在正負10之間
        a1 = np.random.uniform(-10, 10, (5, 1))
        a2 = np.random.uniform(-10, 10, (5, 1))
        a3 = np.random.uniform(-10, 10, (5, 1))
```

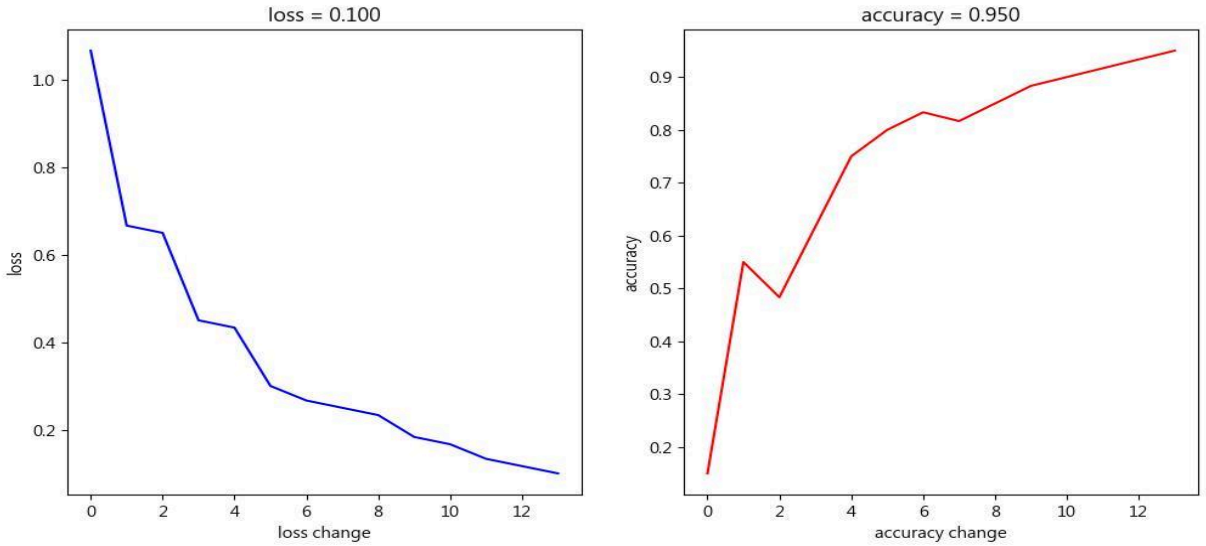
```
            #將a1,a2,a3分別乘上影像特徵向量
            y1 = np.dot(feature, a1)
            y2 = np.dot(feature, a2)
            y3 = np.dot(feature, a3)

            #用 softmax 函式，將每一筆資料的評估值，轉換算成[0 1]之間的機率值p
            p1, p2, p3 = softmax(y1), softmax(y2), softmax(y3)
            p = np.concatenate((p1, p2, p3), axis=1) #將p1.p2.p3三個120*1矩陣
合併為120*3

            tp = np.argmax(p, axis=1)+1 #獲得每一個影像機率最高的分類編號
            t_error = np.abs(t - tp) #計算t的預測誤差
            loss = np.mean(t_error) #以t_error的平均數，計算損失(loss)

            if loss < min_loss:
                loss_n += 1
                loss_history.append(loss) #記錄 loss 歷史變化
                acc = np.sum(t_error == 0) / len(t) #計算當下的正確率
                accuracy.append(acc) #記錄正確率歷史變化
                min_loss = loss #將當下的loss值存入min_loss
                best_tp = tp #將當下的tp值存入best_tp

    return loss_history, accuracy, min_loss, best_tp

#繪製loss及accuracy曲線
def plot_loss_accuracy(loss_history, accuracy):
    #設置matplotlib支援中文顯示
    plt.rcParams['font.sans-serif'] = ['Microsoft JhengHei'] #使用微軟正
黑體
    plt.rcParams['axes.unicode_minus'] = False

    #繪製loss歷史曲線
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(loss_history, color='blue')
    plt.xlabel('loss change') #x軸標籤
    plt.ylabel('loss') #y軸標籤
    plt.title(f'loss = {min_loss:.3f}')

    #繪製accuracy歷史曲線
    plt.subplot(1, 2, 2)
    plt.plot(accuracy, color='red')
    plt.xlabel('accuracy change') #x軸標籤
```

```
    plt.ylabel('accuracy') #y軸標籤
    plt.title(f'accuracy = {np.max(accuracy):.3f}')
    plt.savefig('loss and acccuracy.jpg') #儲存圖片
    plt.show()

#計算混淆矩陣函式
def compute_confusion_matrix(t, best_tp):
    #建立3*3混淆矩陣
    confusion_matrix = np.zeros((3, 3), dtype=int)

    #計算混淆矩陣
    for i in range(len(t)):
        true = t[i] #真實標籤
        pred = best_tp[i] #預測標籤
        confusion_matrix[true-1, pred-1] += 1

    #建立大小8*4的圖
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.axis('off') #關閉座標

    #建立表格資料
    table_data = [
        ['', '', '', '實際值', ''],
        ['混淆', '矩陣', '沙漠', '樹林', '風景'],
        ['' ,'沙漠'] + [str(x) for x in confusion_matrix[0]],
        ['預測值' ,'樹林'] + [str(x) for x in confusion_matrix[1]],
        ['' ,'風景'] + [str(x) for x in confusion_matrix[2]]
    ]

    #建立表格
    # cellText: 表格內容
    # cellLoc: 單元格內文字居中對齊
    # loc: 表格在圖中的位置
    # bbox: 表格的邊界框位置和大小
    table = ax.table(cellText=table_data,
                     cellLoc='center',
                     loc='center',
                     bbox=[0, 0, 1, 1])


    #調整表格對齊與框線顯示
    table[(1, 0)].set_text_props(ha='right', va='bottom')
    table[(1, 1)].set_text_props(ha='left', va='bottom')
```

```
    #設置表格邊框的可見性 T: 上邊框, L: 左邊框, R: 右邊框, B: 下邊框
    table[(0, 0)].visible_edges = 'TL'
    table[(0, 1)].visible_edges = 'T'
    table[(1, 0)].visible_edges = 'BL'
    table[(1, 1)].visible_edges = 'R'

    table[(2, 0)].visible_edges = 'TL'
    table[(3, 0)].visible_edges = 'L'
    table[(4, 0)].visible_edges = 'BL'

    table[(0, 2)].visible_edges = 'TL'
    table[(0, 3)].visible_edges = 'T'
    table[(0, 4)].visible_edges = 'TR'

    #調整文字大小
    table.auto_set_font_size(False) #關閉自動字體大小調整
    table.set_fontsize(16) #設置字體大小為16

    plt.savefig('confusion matrix.jpg') #儲存圖片
    plt.show()

if __name__ == '__main__':
    #初始化變數, 並獲得圖庫路徑、拉普拉斯的離散卷積核、目標標籤
    path, filter, t = initialize()

    #提取影像特徵, 根據指定的濾波器來提取影像的特徵
    feature = img_feature_extract(path, filter)

    #執行影像分類訓練, 返回loss歷史、準確率、最小損失和最佳預測結果
    loss_history, accuracy, min_loss, best_tp =
img_classfication_training(feature, t)

    #輸出最小損失值與準確率
    print(f'loss : {min_loss:.3f}')
    print(f'Accuracy : {np.max(accuracy):.3f}')

    #儲存best_tp變數
    np.save('best_tp.npy', best_tp)

    #繪製loss與準確率的曲線圖
    plot_loss_accuracy(loss_history, accuracy)
```

```
#計算並顯示混淆矩陣
compute_confusion_matrix(t, best_tp)
```

「影像分類」訓練的**loss** 歷史曲線(左)與準確度歷史曲線(右)



「影像分類」結果的混淆矩陣

混淆　矩陣		實際值		
		沙漠	樹林	風景
預測值	沙漠	40	0	0
	樹林	0	40	0
	風景	6	0	34

第二部分: 簡易色彩風格轉換(用梯度下降法優化參數)

```
#匯入此作業使用到的模組
import os #用於操作系統功能，這裡用在清除畫面
import cv2 #OpenCV函式庫，用於圖像處理
import types #用於類型判斷
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

#初始化
def initialize():
    #清畫面
    os.system('cls' if os.name == 'nt' else 'clear')

    #清變數
    for var in list(globals().keys()):
        if var not in ["__builtins__", "__name__", "__doc__",
"__package__"] and not isinstance(globals()[var], (type(os),
types.FunctionType)):
            del globals()[var]

    #關閉所有figure
    plt.close('all')

    path = 'HW3 instructions\imageset' #設圖庫路徑

    X = [] #X序列
    Xf = [] #Xf序列
    Y = [] #Y序列
    Yf = [] #Yf序列

    #迴圈依序產生1到40的影像編號
    for i in range(1, 41):
        img_x_path = f'{path}\\{i}.jpg' #x影像路徑
        img_x = cv2.cvtColor(cv2.imread(img_x_path, cv2.IMREAD_COLOR),
cv2.COLOR_BGR2RGB) #讀取x影像並轉換為RGB
        X.append(img_x) #將x影像存入X序列
        feature_x = cv2.resize(img_x, (1, 3)) #將x影像縮小至3x1x3特徵擷取
        Xf.append(feature_x) #將x影像特徵存入Xf序列

        img_y_path = f'{path}\\{i + 80}.jpg' #y影像路徑
        img_y = cv2.cvtColor(cv2.imread(img_y_path, cv2.IMREAD_COLOR),
cv2.COLOR_BGR2RGB) #讀取y影像並轉換為RGB
        Y.append(img_y) #將y影像存入Y序列
```

```
            feature_y = cv2.resize(img_y, (1, 3)) #將y縮小至3x1x3特徵擷取
            Yf.append(feature_y) #將y影像特徵存入Yf序列

    #轉換成numpy array
    X = np.array(X)
    Xf = np.array(Xf) / 255.0 #正規化
    Y = np.array(Y)
    Yf = np.array(Yf) / 255.0 #正規化

    return X, Xf, Y, Yf

#參數訓練
def parameter_training(Xf, Yf):
    w = np.zeros((3, 1, 3)) #初始化權重矩陣

    #學習率lr設定0.2
    lr = 0.2

    #儲存每個zone和channel的Loss
    loss_history = [[[[] for _ in range(3)] for _ in range(1)] for _ in
range(3)]

    #迴圈
    for zone in range(3): #高、中、低位置迴圈
        for ch in range(3): #RGB通道迴圈
            Xi = [Xf[i, zone, 0, ch] for i in range(len(Xf))] #從Xf序列
讀出參數相應的輸入值Xi
            Yi = [Yf[i, zone, 0, ch] for i in range(len(Xf))] #從Yf序列
讀出參數相應的輸入值Yi

            #迭代迴圈(相當於回合數epoch)設定為30
            for epoch in range(30):
                #計算L2Loss和梯度
                L2 = 0
                G = 0
                #計算L2 loss及誤差梯度向量
                for i in range(len(Xi)):
                    L2 += ((Yi[i] - w[zone, 0, ch] * Xi[i]) ** 2)
                    G += ((Yi[i] - w[zone, 0, ch] * Xi[i]) * 2 * Xi[i])
                L2 /= len(Xi)
                G /= -len(Xi)

                #權重增量向量
```

```
                delta_w = -lr * G

                #更新權重
                w[zone, 0, ch] += delta_w

                #儲存Loss
                loss_history[zone][0][ch].append(L2)

    return w, loss_history

#展示成果
def result(X, Xf, Y, Yf, w):
    w_img = np.zeros((3, 3, 3)) #創建3x3的影像，3個區域，3個顏色通道
    for ch in range(3):
        w_img[ch] = w[ch, 0] #將每個區域的RGB權重填入影像中

    #將3x3的權重影像使用線性內差方法放大到224x224
    w_resize = cv2.resize(w_img, (224, 224),
interpolation=cv2.INTER_LINEAR)

    #從0-39中隨機選擇前4個不重複的數字
    rand = np.random.permutation(40)[:4]

    #建立一個大的畫布，用於顯示所有影像
    plt.figure(figsize=(24, 16))

    #依序處理選中的影像索引
    for index, value in enumerate(rand):
        original_img = X[value] #原始影圖
        style_img = Y[value] #對應的風格影像

        #建立一個與原始影像相同大小的畫布，用於存放轉換後的影像，dtype設為float32
以支持浮點數運算
        transform_img = np.ones_like(original_img, dtype=np.float32)

        #對影像的每個像素進行處理
        for i in range(224):
            for j in range(224):
                for c in range(3):
                    #將原始影像的像素值與對應位置的權重相乘
                    transform_img[i, j, c] = original_img[i, j, c] *
w_resize[i, j, c]
```

```
            #將像素值限制在0-255範圍內，並轉換為整數類型
            transform_img = np.clip(transform_img, 0, 255).astype(np.uint8)

            #顯示原始影像
            plt.subplot(3, 4, index + 1)
            plt.imshow(original_img)
            plt.axis('off') #關閉座標軸
            plt.title(f'content X ({value + 1})')

            #顯示目標風格影像
            plt.subplot(3, 4, index + 5)
            plt.imshow(style_img)
            plt.axis('off') #關閉座標軸
            plt.title(f'style Y ({value + 81})')

            #顯示風格轉換後的影像
            plt.subplot(3, 4, index + 9)
            plt.imshow(transform_img)
            plt.axis('off') #關閉座標軸
            plt.title(f'X ({value + 1}) to Y')

    # 顯示圖像
    plt.tight_layout() #自動調整圖像間距
    plt.get_current_fig_manager().window.state('zoomed') #讓圖形全螢幕顯示
    plt.savefig('result.jpg')  #儲存圖片
    plt.show()

#繪製9個參數的 Loss 梯度下降歷史曲線
def plot_loss(loss_history):
    #繪製Loss曲線在同一個圖中
    fig, ax = plt.subplots(3, 3, figsize=(12, 12))

    #處理每個位置和通道，繪製對應的Loss曲線
    for zone in range(3): #高、中、低位置迴圈
        for ch in range(3): #RGB通道迴圈
            axs = ax[zone, ch] #當前子圖的axes對象
            axs.plot(loss_history[zone][0][ch]) #繪製當前位置和通道的Loss曲
線
            axs.set_title(f"zone {zone+1}, channel {['1', '2',
'3'][ch]}") #設置標題
            axs.set_xlabel('epoch') #x軸標籤
            axs.set_ylabel('loss') #y軸標籤
```

```
    plt.tight_layout(rect=[0, 0, 1, 0.95]) #調整整體布局
    plt.get_current_fig_manager().window.state('zoomed') #讓圖形全螢幕顯示
    plt.savefig('loss history.jpg') #儲存圖片
    plt.show()

if __name__ == '__main__':
    #初始化資料，載入影像資料並提取特徵
    X, Xf, Y, Yf = initialize()

    #執行參數訓練，使用Xf和Yf中的特徵來訓練，並返回最終的權重w和loss歷史
    w, loss_history = parameter_training(Xf, Yf)

    #將原影像、目標風格影像、以及轉換後的影像隨機展示
    result(X, Xf, Y, Yf, w)

    #繪製loss歷史的曲線圖
    plot_loss(loss_history)
```

「簡易色彩風格轉換」的成果

「簡易色彩風格轉換」的**loss**歷史曲線