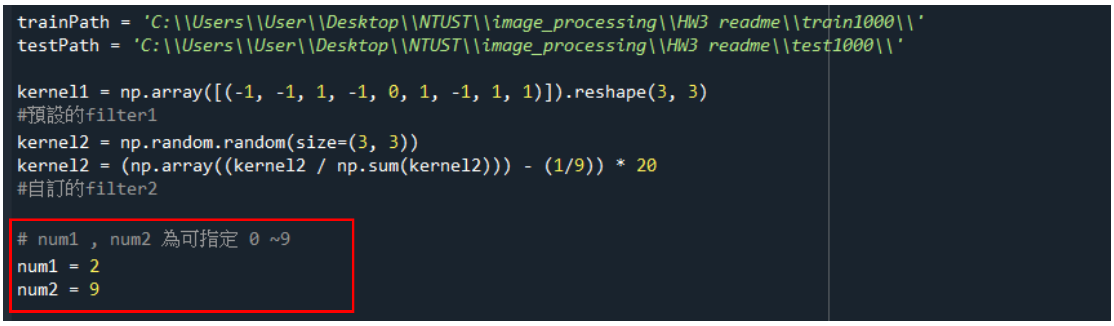
彩色影像處理

作業 3：簡化的 CNN 數值辨識

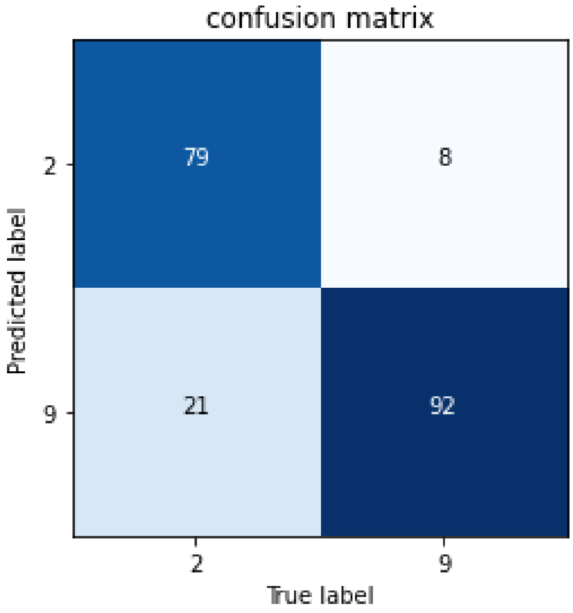
M10915045 施信宏

完成了可以隨意指定 0~9 之間兩類別的比較，若要更改數值，請在下圖地方更改，路徑改成存放位置的地方。其餘程式碼部分已有註解，有疑問再請詳看。

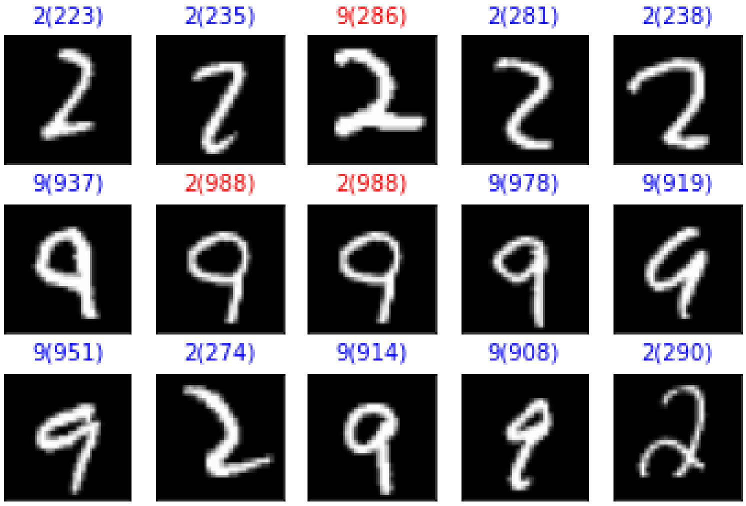


```
trainPath = 'C:\\Users\\User\\Desktop\\NTUST\\image_processing\\HW3 readme\\train1000\\'
testPath = 'C:\\Users\\User\\Desktop\\NTUST\\image_processing\\HW3 readme\\test1000\\'

kernel1 = np.array([(-1, -1, 1, -1, 0, 1, -1, 1, 1)]).reshape(3, 3)
#預設的filter1
kernel2 = np.random.random(size=(3, 3))
kernel2 = (np.array((kernel2 / np.sum(kernel2))) - (1/9)) * 20
#自訂的filter2

# num1 , num2 為可指定 0 ~9
num1 = 2
num2 = 9
```

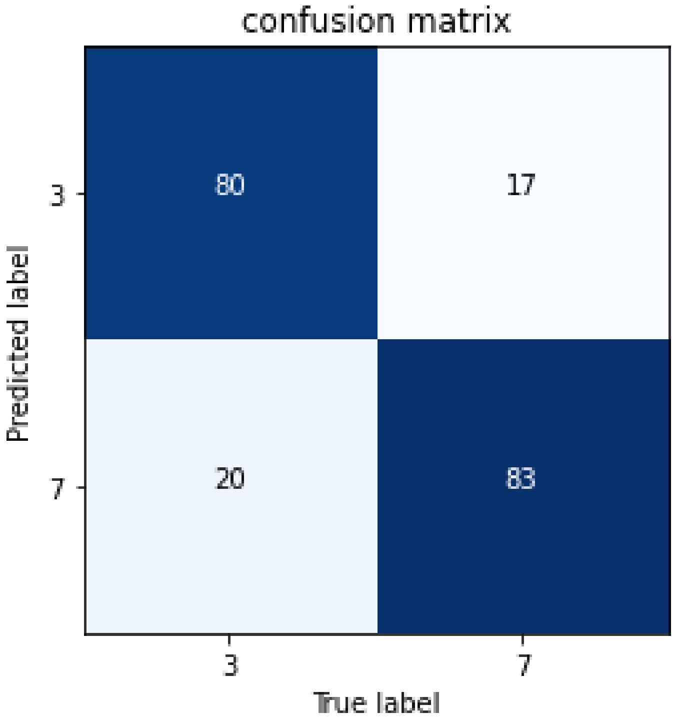
類別 ： 2、9

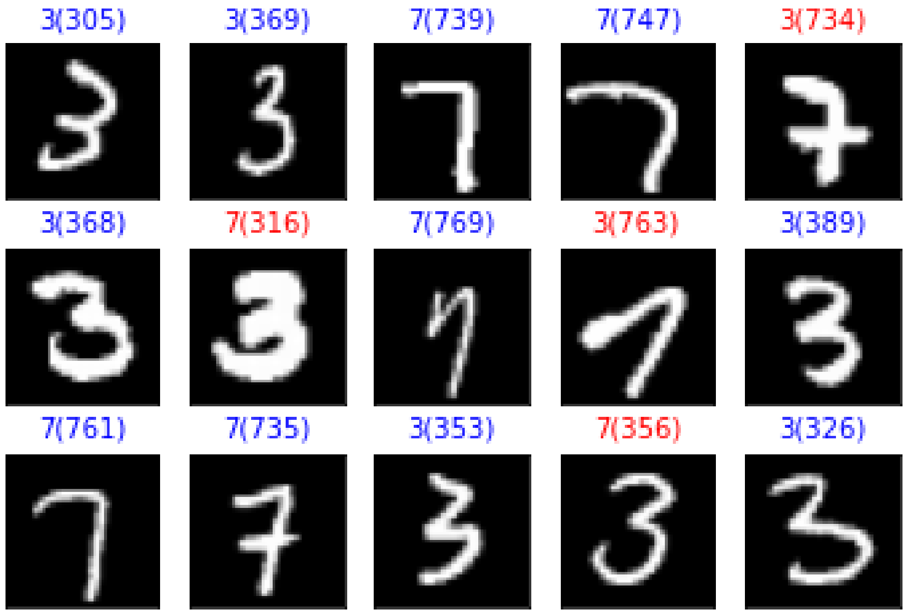


隨機 15 個影像分類結果


2(223)
2(235)
9(286)
2(281)
2(238)
9(937)
2(988)
2(988)
9(978)
9(919)
9(951)
2(274)
9(914)
9(908)
2(290)

類別 3、7


confusion matrix
Predicted label
3
7
80
17
20
83
3
7
True label



Code :

```
import numpy as np
import cv2
from sklearn.metrics import classification_report
import itertools
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import random

trainPath = 'C:\\Users\\User\\Desktop\\NTUST\\image_processing\\HW3 readme\\train1000\\'
testPath = 'C:\\Users\\User\\Desktop\\NTUST\\image_processing\\HW3 readme\\test1000\\'

kernel1 = np.array([(-1, -1, 1, -1, 0, 1, -1, 1, 1)]).reshape(3, 3)
#預設的 filter1
kernel2 = np.random.random(size=(3, 3))
kernel2 = (np.array((kernel2 / np.sum(kernel2))) - (1/9)) * 20
#自訂的 filter2

# num1 , num2  為可指定  0 ~9
num1 = 2
```

```
num2 = 9

#讀取檔案，只取 train 的部分即可完成作業需求
def load_image(num1 , num2):

    train = []
    test = []

    num1 = num1 * 100
    num2 = num2 * 100

    for i in range(num1 , num1 + 100):
        img = cv2.imread(trainPath + str(i + 1) + ".png", 0)
        train.append(cv2.resize(img, (8, 8)))

    for i in range(num2,num2 + 100):
        img = cv2.imread(trainPath + str(i + 1) + ".png", 0)
        train.append(cv2.resize(img, (8, 8)))

    return train

#max pooling
def pooling(inputMap, poolSize=3, poolStride=2, mode='max'):
        """INPUTS:
                  inputMap - input array of the pooling layer
                  poolSize - X-size(equivalent to Y-size) of receptive field
                  poolStride - the stride size between successive pooling squares

           OUTPUTS:
                   outputMap - output array of the pooling layer

           Padding mode - 'edge'
        """
        # inputMap sizes
        in_row, in_col = np.shape(inputMap)

        # outputMap sizes
        out_row, out_col = int(np.floor(in_row / poolStride)), int(np.floor(in_col /
poolStride))
        row_remainder, col_remainder = np.mod(in_row, poolStride), 
np.mod(in_col, poolStride)
        if row_remainder != 0:
            out_row += 1
        if col_remainder != 0:
            out_col += 1
        outputMap = np.zeros((out_row, out_col))

        # padding
        temp_map = np.lib.pad(inputMap, ((0, poolSize - row_remainder), (0, 
poolSize - col_remainder)), 'edge')

        # max pooling
        for r_idx in range(0, out_row):
            for c_idx in range(0, out_col):
                startX = c_idx * poolStride
                startY = r_idx * poolStride
                poolField = temp_map[startY:startY + poolSize, startX:startX + 
poolSize]
                poolOut = np.max(poolField)
                outputMap[r_idx, c_idx] = poolOut

        # retrun outputMap
        return outputMap
#第一層捲積及完成 max-pooling
def conv_pool_layer1(img, kernel):
    # Conv_and_Pooling
    conv_img = cv2.filter2D(img, -1, kernel)
    max_img = pooling(conv_img, 2)
    return max_img
#第二層捲積及完成 max-pooling
def conv_pool_layer2(img, kernel):
    # Conv_and_Pooling
    conv_img = cv2.filter2D(img[:, :, 0], -1, kernel) + cv2.filter2D(img[:, :, 1], -1, 
kernel)
    max_img = pooling(conv_img, 2)
    return max_img
```

```
def model_train(num1, num2,train,kernel1,kernel2):

        Y = np.zeros(200)
        Y[:100] += num1
        Y[100:] += num2
        X = []
        # Y true label , X predict  的結果
        #經過兩層 conv,max_pooling 後的結果加入 X
        for img in train:
                img = np.array([conv_pool_layer1(img, kernel1), conv_pool_layer1(img, 
kernel2)]).transpose(1, 2, 0)
                img = np.array([conv_pool_layer2(img, kernel1), conv_pool_layer2(img, 
kernel2)]).transpose(1, 2, 0)
                img = list(img.reshape(-1))
                #平坦化  2*2*2 -> 8
                img.append(1)
                #  常數項加 1
                X.append(np.array(img))
        X = np.array(X)
        try:
                A = np.dot(np.linalg.inv(np.dot(X.T, X)), (np.dot(X.T, Y)))
        except np.linalg.LinAlgError:
                print("LinAlgError")
        Yp = np.dot(X, A)
        y_pred = np.where(np.abs(Yp - num1) > np.abs(Yp - num2), num2, num1)
        #將算出來的結果進行分類
        y_true = Y
        cnf_matrix = confusion_matrix(y_true, y_pred)
        acc = cnf_matrix[0][0] + cnf_matrix[1][1]
        print(acc)
        return y_true , y_pred
#畫 confusion_matrix
def plot_confusion_matrix(cm, classes,
                                                            normalize=False,
                                                            title='Confusion matrix',
                                                            cmap=plt.cm.Blues):
        """
```

```
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    cm = cm.T
    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
        horizontalalignment="center",
        color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()

#隨機找 15 張圖，看其預測結果
def plot_images_labels_prediction(labels, prediction, num1, num2):
    # fig = plt.gcf()
    # fig.set_size_inches(12, 14)
    plt.figure()
    for i in range(0, 15):
        r = random.randint(0, 199)
        png = num1 * 100 + r if r < 100 else (num2 - 1) * 100 + r
        img = cv2.imread(testPath + str(png + 1) + ".png", 0)
        ax = plt.subplot(3, 5, 1 + i)
        ax.imshow(img, cmap='gray')
        title = str(prediction[r]) + "(" + str(png) + ")"
        if labels[r] == prediction[r]:
            color = 'b'
        else:
            color = 'r'

        ax.set_title(title, fontsize=10, color=color)
        ax.set_xticks([])
        ax.set_yticks([])

def generate_confusion_mat(y_true, y_pred):
    target_names = [str(num1), str(num2)]
    cnf_matrix = confusion_matrix(y_true, y_pred)
    plot_confusion_matrix(cnf_matrix, classes=target_names, normalize=False,
                          title='confusion matrix')
    plot_images_labels_prediction(y_true, y_pred, num1, num2)
    plt.show()

if __name__ == '__main__':
    train    = load_image(num1 , num2)
    y_true, y_pred = model_train(num1,num2,train,kernel1,kernel2)
    generate_confusion_mat(y_true, y_pred)
```