Assignment – LeNet5

1. 概述

a. LeNet 是由 Yann LeCun 團隊提出的網路架構，是卷積神經網路的始祖。其架構由兩個卷積層、池化層、全連接層以及最後一層 Gaussian 連接層所組成，早期用來辨識手寫數字圖像

b. 由下圖可以看到 LeNet 的網路架構共有七層：卷積層 (Convolutions, C1)、池化層 (Subsampling, S2)、卷積層 (C3)、池化層 (S4)、全連接卷積層 (C5)、全連接層 (F6)、Gaussian 連接層 (output)

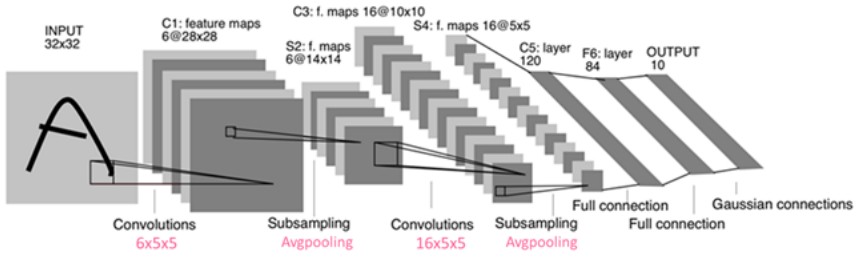

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

2. LeNet5 程式碼說明

在訓練方面，有 10 種動物類別，總共先使用 100 張圖面作為訓練的資料，每個動物類別皆有 10 張圖片；測試方面，亦放入 10 總動物的類別，總共先使用 50 張圖片作為測試的資料，每個動物類別皆有 5 張圖片．圖面類別如下所示：

Bear leopard tiger dog cat

lion fox meerkat polar bear wolf

訓練及測試的平台為 google 的 colab 上做訓練及測試，由於目前的圖片資料量來看，若放過多的圖片，會使得在 colab 上運行時，會產生記憶體不足的現象，故目前所放的圖片量較少．相關程式碼說明如下：

a. 載入圖片與儲存圖片 pixel 置矩陣內。

```
# 取得檔案名稱, 將檔案名稱的第一個字元, 作為標籤(答案)的方式, 最後 return list labels.
def get_training_data(data_dir):
    images = []
    labels = []
    files = os.listdir(data_dir)
    #print(" -- files: \n", files)
    random.shuffle(files)
    #i = 0
    for f in files:
        # path + file name
        img = cv2.imread(os.path.join(data_dir, f), cv2.IMREAD_GRAYSCALE)
        print(" -- img shape: \n: ", img.shape)
        img = cv2.resize(img, (32, 32))
        img = img.astype(np.float32).reshape(32, 32, 1) / 255.0
        images.append(img)

        # bear -> 0,    leopard    -> 1,    tiger   -> 2
        # dog  -> 3,    cat.       -> 4,    lion.   -> 5
        # fox. -> 6,    polar_bear -> 7,    meerkat -> 8
        # wolf -> 9,

        # use first letter to define label's answer.
        if "bear" in f:
            num = 0
        elif "leopard" in f:
            num = 1
        elif "tiger" in f:
            num = 2
        elif "dog" in f:
            num = 3
        elif "cat" in f:
            num = 4
        elif "lion" in f:
            num = 5
        elif "fox" in f:
            num = 6
        elif "polar_bear" in f:
            num = 7
        elif "meerkat" in f:
            num = 8
        elif "wolf" in f:
            num = 9

        label = np.zeros(10, dtype=np.float32)
        # num 表示答案, 將那個答案填入升起的 flag = 1, 作為標籤答案的表示法.
        label[num] = 1
        labels.append(label)
    return (np.array(images), np.array(labels))
```

b. 執行訓練主程式

```
if __name__ == '__main__':
    x, y = get_training_data("./images/train/")
    lenet = LeNet()
    lenet.train(x, y)
    lenet.save("./lenet.npy"
```

c. 執行測試主程式

```
import os
import cv2
import numpy as np
from lenet import LeNet

data_dir = "./images/test/"
net = LeNet()
net.load("./lenet.npy")
files = os.listdir(data_dir)
print("files: \n", files)
images = []
labels = []
for f in files:
    img = cv2.imread(os.path.join(data_dir, f), cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (32, 32))
    img = img.astype(np.float32).reshape(32, 32, 1) / 255.0
    images.append(img)

    # bear -> 0,    leopard    -> 1,    tiger   -> 2
    # dog  -> 3,    cat.       -> 4,    lion.   -> 5
    # fox. -> 6,    polar_bear -> 7,    meerkat -> 8
    # wolf -> 9,
    print("f: \n", f)
    if "bear" in f and "polar_bear" not in f:
      num = 0
    elif "leopard" in f:
      num = 1
    elif "tiger" in f:
      num = 2
    elif "dog" in f:
      num = 3
    elif "cat" in f:
      num = 4
    elif "lion" in f:
      num = 5
    elif "fox" in f:
      num = 6
    elif "polar_bear" in f:
      num = 7
    elif "meerkat" in f:
      num = 8
    elif "wolf" in f:
      num = 9

    labels.append(num)

x = np.array(images)
y = np.array(labels)
print("  -- y: \n", y)
predict = net.predict(x)
tp = np.sum(predict == y)
accuracy = float(tp) / len(files)
print("accuracy=%f" % accuracy
```

3. 訓練及測試結果

a. 訓練

```
step 0: loss=3.561156, accuracy=0.0900, lr=0.0003
step 1: loss=3.543641, accuracy=0.0900, lr=0.0002997
step 2: loss=3.513316, accuracy=0.1100, lr=0.0002994
step 3: loss=3.473693, accuracy=0.1200, lr=0.000299101
step 4: loss=3.429314, accuracy=0.1400, lr=0.000298802
step 5: loss=3.384855, accuracy=0.1200, lr=0.000298503
step 6: loss=3.344236, accuracy=0.1400, lr=0.000298204
step 7: loss=3.309901, accuracy=0.1200, lr=0.000297906
step 8: loss=3.281117, accuracy=0.1200, lr=0.000297608
step 9: loss=3.256747, accuracy=0.1100, lr=0.000297311
step 10: loss=3.237259, accuracy=0.1100, lr=0.000297013
step 11: loss=3.221760, accuracy=0.1500, lr=0.000296716
step 12: loss=3.210008, accuracy=0.1600, lr=0.00029642
step 35: loss=3.091626, accuracy=0.2000, lr=0.000289677

                                        ......

step 186: loss=2.334085, accuracy=0.5200, lr=0.000249059
step 187: loss=2.327128, accuracy=0.5300, lr=0.00024881
step 188: loss=2.320301, accuracy=0.5300, lr=0.000248561
step 189: loss=2.313511, accuracy=0.5300, lr=0.000248312
step 190: loss=2.306746, accuracy=0.5300, lr=0.000248064
step 191: loss=2.299893, accuracy=0.5400, lr=0.000247816
step 192: loss=2.292888, accuracy=0.5400, lr=0.000247568
step 193: loss=2.285807, accuracy=0.5400, lr=0.000247321
step 194: loss=2.278806, accuracy=0.5400, lr=0.000247073
step 195: loss=2.271881, accuracy=0.5400, lr=0.000246826
step 196: loss=2.265034, accuracy=0.5400, lr=0.000246579
step 197: loss=2.258204, accuracy=0.5400, lr=0.000246333
step 198: loss=2.251280, accuracy=0.5400, lr=0.000246087
step 199: loss=2.244301, accuracy=0.5400, lr=0.00024584
```

b. 測試

```
files:
 ['888_leopard.JPEG', '887_leopard.JPEG', '890_leopard.JPEG', '889_leopard.JPEG',
'891_leopard.JPEG', '889_bear.JPEG', '890_bear.JPEG', '891_bear.JPEG',
'888_bear.JPEG', '887_bear.JPEG', '888_meerkat.JPEG', '891_meerkat.JPEG',
'890_meerkat.JPEG', '889_meerkat.JPEG', '887_meerkat.JPEG', '888_tiger.JPEG',
'889_tiger.JPEG', '891_tiger.JPEG', '890_tiger.JPEG', '887_tiger.JPEG',
'889_wolf.JPEG', '888_wolf.JPEG', '891_wolf.JPEG', '890_wolf.JPEG',
'887_wolf.JPEG', '889_fox.JPEG', '887_fox.JPEG', '888_fox.JPEG', '891_fox.JPEG',
'890_fox.JPEG', '887_lion.JPEG', '891_lion.JPEG', '889_lion.JPEG',
'888_lion.JPEG', '890_lion.JPEG', '889_polar_bear.JPEG', '891_polar_bear.JPEG',
'888_polar_bear.JPEG', '890_polar_bear.JPEG', '887_polar_bear.JPEG',
'890_dog.JPEG', '888_dog.JPEG', '891_dog.JPEG', '889_dog.JPEG', '887_dog.JPEG',
'889_cat.JPEG', '887_cat.JPEG', '891_cat.JPEG', '888_cat.JPEG', '890_cat.JPEG']
  -- y:
 [1 1 1 1 1 0 0 0 0 0 8 8 8 8 8 2 2 2 2 2 9 9 9 9 9 6 6 6 6 6 5 5 5 5 5 7 7
 7 7 7 3 3 3 3 3 4 4 4 4 4]
accuracy=0.120000
```

4. 結論

以 100 張圖片為訓練基礎，並且 step 設定 200，其測試的結果準確度為 0.12，結果不是很理想。