Assignment – LeNet5 (Pytorch + Tensorflow)

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)

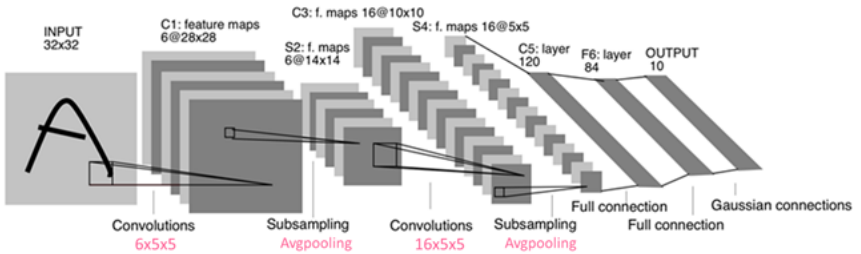

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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. load_data()函式：載入圖片的 RGB pixel 資料，並分別儲存到 train 與 test 的 list 裡。

```
def load_data():
    data_dir_train = "./images/train"
    data_dir_test  = "./images/test"

    files_train = os.listdir(data_dir_train)
    files_test  = os.listdir(data_dir_test)

    num_train_samples = len(files_train)
    num_test_samples  = len(files_test)
    x_train = np.empty((num_train_samples, 32, 32, 3), dtype = "float32")
    y_train = np.zeros((num_train_samples, 10), dtype = "float32")
    x_test  = np.empty((num_test_samples, 32, 32, 3), dtype = "float32")
    y_test  = np.zeros((num_test_samples, 10), dtype = "float32")

    i = 0
    for f_train in files_train:
        img = cv2.imread(os.path.join(data_dir_train, f_train))
        print("data_dir_train + f_train: \n", os.path.join(data_dir_train,
f_train))
        print("img.shape: \n", img.shape)

        img_resize = np.resize(img, (32, 32, 3))

        x_train[i, :, :, :] = img_resize

        if "bear" in f_train:
            num = 0
        elif "leopard" in f_train:
            num = 1
        elif "tiger" in f_train:
            num = 2
        elif "dog" in f_train:
            num = 3
        elif "cat" in f_train:
            num = 4
        elif "lion" in f_train:
            num = 5
        elif "fox" in f_train:
            num = 6
        elif "polar_bear" in f_train:
            num = 7
        elif "meerkat" in f_train:
            num = 8
        elif "wolf" in f_train:
            num = 9

        # 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.
        y_train[i, num] = 1

        i += 1
```

```
    j = 0
    for f_test in files_test:
        img = cv2.imread(os.path.join(data_dir_test, f_test))
        print("data_dir_test + f_test: \n", os.path.join(data_dir_test, f_test))
        print("img.shape: \n", img.shape)
        img_resize = np.resize(img, (32, 32, 3))

        x_test[j, :, :, :] = img_resize

        #print(" -- f \n", f_test)
        if "bear" in f_test:
            num = 0
        elif "leopard" in f_test:
            num = 1
        elif "tiger" in f_test:
            num = 2
        elif "dog" in f_test:
            num = 3
        elif "cat" in f_test:
            num = 4
        elif "lion" in f_test:
            num = 5
        elif "fox" in f_test:
            num = 6
        elif "polar_bear" in f_test:
            num = 7
        elif "meerkat" in f_test:
            num = 8
        elif "wolf" in f_test:
            num = 9

        # 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.
        y_test[j, num] = 1
        j += 1

    x_train = x_train / 255
    #print("x_train / 255: \n", x_train)
    #print("y_train: \n", y_train)

    x_test  = x_test / 255
    #print("x_test / 255: \n", x_test)
    #print("y_test: \n", y_test)

    return x_train, y_train, x_test, y_test
```

b. 執行訓練與測試程式

```
xtrain, ytrain, xtest, ytest = load_data()

# Parameters
num_epoch = 4000
batch_size = 128

# layer 0: input data
x = tf.placeholder("float", [None,32,32,3])
#x = tf.placeholder("float", [None,64,64,3])
y = tf.placeholder("float", [None,10])

# layer 1: convolution
# filter size = 5x5, input channel = 1, output channel = 32
conv1_w = tf.get_variable("conv1_w", [5,5,3,32],
initializer=tf.truncated_normal_initializer(stddev=0.1))
#conv1_w = tf.get_variable("conv1_w", [5,5,3,64],
initializer=tf.truncated_normal_initializer(stddev=0.1))
conv1_b = tf.get_variable("conv1_b", [32],
initializer=tf.constant_initializer(value=0))
#conv1_b = tf.get_variable("conv1_b", [64],
initializer=tf.constant_initializer(value=0))
conv1 = tf.nn.conv2d(x, conv1_w, strides=[1,1,1,1], padding='SAME')
relu1 = tf.nn.relu( tf.nn.bias_add(conv1, conv1_b) )

# layer 2: max pool
# filter size = 2x2, stride = 2
pool1 = tf.nn.max_pool(relu1, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

# layer 3: convolution
# filter size = 5x5, input channel = 32, output channel = 64
conv2_w = tf.get_variable("conv2_w", [5,5,32,64],
initializer=tf.truncated_normal_initializer(stddev=0.1))
#conv2_w = tf.get_variable("conv2_w", [5,5,64,64],
initializer=tf.truncated_normal_initializer(stddev=0.1))
conv2_b = tf.get_variable("conv2_b", [64],
initializer=tf.constant_initializer(value=0))
conv2 = tf.nn.conv2d(pool1, conv2_w, strides=[1,1,1,1], padding='SAME')
relu2 = tf.nn.relu( tf.nn.bias_add(conv2, conv2_b) )

# layer 4: max pool
pool2 = tf.nn.max_pool(relu2, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
```

```
# layer 5: fully connected
fc1_w = tf.get_variable("fc1_w", [8 * 8 * 64, 1024],
initializer=tf.truncated_normal_initializer(stddev=0.1))
fc1_b = tf.get_variable("fc1_b", [1024],
initializer=tf.constant_initializer(value=0.1))
pool2_vector = tf.reshape(pool2, [-1, 8 * 8 * 64])
fc1 = tf.nn.relu( tf.matmul(pool2_vector, fc1_w) + fc1_b )

# dropout layer
fc1_dropout = tf.nn.dropout(fc1, 1.0)

# layer 6: fully connected
fc2_w = tf.get_variable("fc2_w", [1024, 10],
initializer=tf.truncated_normal_initializer(stddev=0.1))
fc2_b = tf.get_variable("fc2_b", [10],
initializer=tf.constant_initializer(value=0.1))
y_hat = tf.matmul(fc1_dropout, fc2_w) + fc2_b

# layer 7: softmax, output layer
pred = tf.nn.softmax(y_hat)

# define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=y_hat,
labels=y))
optimizer = tf.train.AdamOptimizer()
train_op = optimizer.minimize(loss_op)

# evaluate model
correct = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(num_epoch):
        xbatch, ybatch = next_batch(batch_size, xtrain, ytrain)
        sess.run(train_op, feed_dict={x: xbatch, y: ybatch})
        if ((epoch + 1) % 100 == 0):
            loss, acc = sess.run([loss_op, accuracy], feed_dict={x: xtest, y:
ytest})
            print("epoch " + str(epoch+1) + ", loss= " + "{:.4f}".format(loss) +
", acc= " + "{:.3f}".format(acc))
    # Calculate accuracy for MNIST test images
    acc = sess.run(accuracy, feed_dict={x: xtest, y: ytest})
    print('test acc=' + '{:.3f}'.format(acc))
```

c. 測試結果的準確率

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-
packages/tensorflow/python/util/dispatch.py:201: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in
a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
epoch 100, loss= 6.1622, acc= 0.240
epoch 200, loss= 7.2166, acc= 0.240
epoch 300, loss= 7.7948, acc= 0.220
epoch 400, loss= 8.2040, acc= 0.200
epoch 500, loss= 8.5052, acc= 0.200
epoch 600, loss= 8.7532, acc= 0.200
epoch 700, loss= 8.9626, acc= 0.200
epoch 800, loss= 9.1423, acc= 0.200
epoch 900, loss= 9.3021, acc= 0.200
epoch 1000, loss= 9.4455, acc= 0.200
epoch 1100, loss= 9.5790, acc= 0.200
epoch 1200, loss= 9.7031, acc= 0.200
epoch 1300, loss= 9.8175, acc= 0.200

                                        ......

epoch 2700, loss= 10.9632, acc= 0.180
epoch 2800, loss= 11.0282, acc= 0.180
epoch 2900, loss= 11.0890, acc= 0.180
epoch 3000, loss= 11.1514, acc= 0.180
epoch 3100, loss= 11.2124, acc= 0.180
epoch 3200, loss= 11.2695, acc= 0.180
epoch 3300, loss= 11.3284, acc= 0.180
epoch 3400, loss= 11.3871, acc= 0.180
epoch 3500, loss= 11.4433, acc= 0.200
epoch 3600, loss= 11.4981, acc= 0.200
epoch 3700, loss= 11.5509, acc= 0.200
epoch 3800, loss= 11.6053, acc= 0.200
epoch 3900, loss= 11.6576, acc= 0.200
epoch 4000, loss= 11.7113, acc= 0.200
test acc=0.200
```

3. 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 0.2，結果不是很理想，分析原因如下：

a. 訓練所使用的圖片數量不夠多
b. 圖片的 size 經由切割後，採用 32 x 32 x 3 的 pixel 數，應該可以用更高 pixel 樹的圖片，更接近原始圖片大小的 pixel 去做訓練，