浙江大学

计算机视觉作业报告

作业名称：实现 LeNet-5

姓　　名：陈润健

学　　号：3160103989

电子邮箱：3160103989@zju.edu.cn

联系电话：18868104871

导　　师：潘纲

2019 年　　1 月　　8　日

实现 LeNet-5

一、 作业已实现的功能简述及运行简要说明

1. 实现 LeNet-5。
2. 用 MNIST 的训练数据训练，测试数据进行测试，得到准确率。
3. 自己手写 100 个数字，用训练好的网络进行识别，得到准确率。

二、 作业的开发与运行环境

IDE：Xcode 10.1

操作系统：MacOS 10.14.1

SDK：macosx10.14

三、 基本思路，用到的函数及流程

1. LeNet-5 的原理：

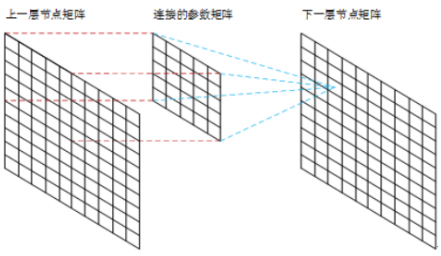
1) 网络中各个部分的介绍：

a) 卷积层（Conv）：

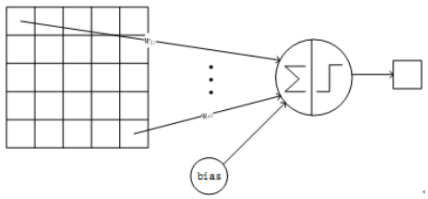
卷积运算的目的是提取输入的不同特征，第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等层级，更多层的网络能从低级特征中迭代提取更复杂的特征。

卷积层采用的都是 5x5 大小的卷积核，且卷积核每次滑动一个像素，一个特征图谱使用同一个卷积核。

每个上层节点的值乘以连接上的参数，把这些乘积及一个偏置参数相加得到一个和，把该和输入激活函数，激活函数的输出即是下一层节点的值。



卷积神经网络连接与矩阵卷积的对应关系



一个卷积节点的连接方式

图示

b） 降采样层（池化层，pooling）：

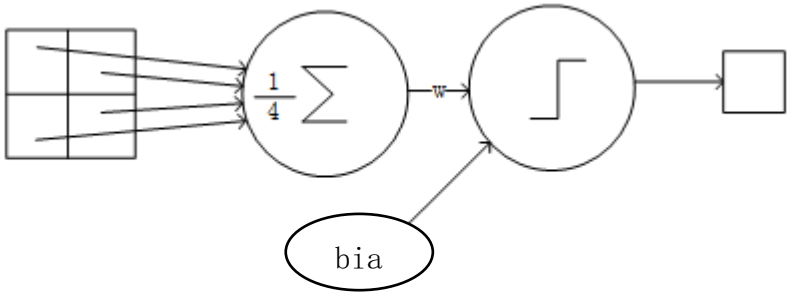
作用是降低图片的分辨率。

降采样在 2*2 的小方块中实现，有两种降采样的方法：平均降采样法，最大值降采样法。

平均值降采样法：将 2*2 的方块中的平均值作为降采样之后的值。

最大值降采样法：将 2*2 的方块中的最大值作为降采样之后的值。

下抽样层采用的是 2x2 的输入域，即上一层的 4 个节点作为下一层 1 个节点的输入，且输入域不重叠，即每次滑动 2 个像素



图示

c） 全连接层：

跟普通的神经网络一样的连接，位于最后几层。

2） 网络结构：一共 7 层，包括两个卷积层，两个降采样层，三个全连接层。

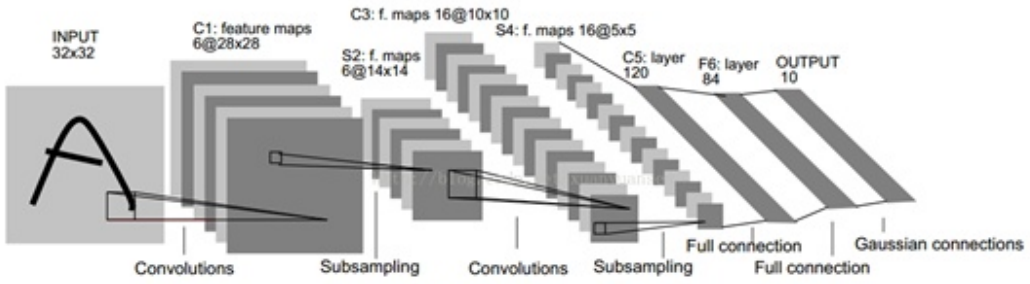


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

a） 卷积层 C1：输入为 32x32 的图像，输出为卷积得到的 6 张 28x28 的图像。因此卷积核的大小是 5x5，每个特征图谱内参数共享，

即每个特征图谱内只使用一个共同卷积核，卷积核有 5x5 个连接参数加上 1 个偏置共 26 个参数。C1 层共有 26x6=156 个训练参数。

b) 池化层 S2：C1 层的 6 个 28x28 的特征图谱分别进行以 2x2 为单位的下抽样得到 6 个 14x14 的图。

c) 卷积层 C3：卷积核和 C1 相同，不同的是 C3 的每个节点与 S2 中的多个图相连。C3 层输出有 16 个 10x10 的图。该层有 (5x5x3+1)x6 + (5x5x4 + 1) x 3 + (5x5x4 +1)x6 + (5x5x6+1)x1 = 1516 个训练参数。

d) 池化层 S4：S4 是一个下采样层。C3 层的 16 个 10x10 的图分别进行以 2x2 为单位的下抽样得到 16 个 5x5 的图。

e) 全连接层 C5：由于 S4 层的 16 个图的大小为 5x5，与卷积核的大小相同，所以卷积后形成的图的大小为 1x1。这里形成 120 个卷积结果。每个都与上一层的 16 个图相连。所以共有 (5x5x16+1)x120 = 48120 个参数

f) 全连接层 F6: F6 层有 84 个节点，对应于一个 7x12 的比特图

g) 全连接层 Output：共有 10 个节点，分别代表数字 0 到 9，如果节点 i 的输出值为 0，则网络识别的结果是数字 i。

3) 训练过程：

a) 前向计算：从样本集中取一个样本（X，Y），将 X 输入网络，计算对应的输出。

b) BP：计算实际输出与理想输出 Y 的差，按极小化误差的方法调整权重矩阵。

c) 可以自己定义训练阶段的个数（epochs），在每个阶段训练之后，会对 validation 数据集进行一次识别，以验证训练的成果。

2. 用 python 调用 tensorflow 进行网络的搭建：

1) Lenet 类的定义，网络结构的构建，loss 等的定义都在类的初始化函数中完成：

```python
# importing packages

import tensorflow as tf
from tensorflow.contrib.layers import flatten
from sklearn.utils import shuffle
from tqdm import tqdm,trange

# define class ChenRJ_lenet5

class ChenRJ_lenet5():

    # initial
    def __init__(self, training_data, traning_label, testing_data, testing_label, validation_data=None,
        validation_label=None,mean=0,stddev=0.3,learning_rate=0.001):

        # training data
        self.training_data = training_data
        self.training_label = traning_label

        # assert if input data does not match requirement
        assert (len(self.training_data) == len(self.training_label))
        assert (self.training_data[0].shape[0] == 32 and self.training_data[0].shape[1] == 32)

        # testing data
        self.testing_data = testing_data
        self.testing_label = testing_label

        # assert if input data does not match requirement
        assert (len(self.testing_data) == len(self.testing_label))
        assert (self.testing_data[0].shape[0] == 32 and self.testing_data[0].shape[1] == 32)

        # validation data
        self.validation_data = validation_data
        self.validation_label = validation_label

        # assert if input data does not match requirement
        assert (len(self.validation_data) == len(self.validation_label))
        assert (self.validation_data[0].shape[0] == 32 and self.validation_data[0].shape[1] == 32)

        # set number for outputs
        self.output_num = 10


        self.X = tf.placeholder(tf.float32, shape=(None, 32, 32, 1))
        self.Y = tf.placeholder(tf.int32, shape=(None, self.output_num))

        # parameter for kernels
        self.mul = mean
        self.sigma = stddev

        # network setting

        # Layer 1: Input 32x32x1, Output 28x28x6
        self.conv1_kernels = tf.Variable(tf.truncated_normal(shape=[5, 5, 1, 6], mean=self.mul, stddev=self.sigma))
        self.conv1_biases = tf.get_variable(name="conv1_biases", shape=[6],
                                            initializer=tf.random_normal_initializer(stddev=0.3))
        self.conv1 = tf.nn.conv2d(self.X, self.conv1_kernels, [1, 1, 1, 1], padding='VALID') + self.conv1_biases

        # Pooling -> from 28x28 to 14x14
        self.pool1 = tf.nn.max_pool(self.conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')
        # Activation
        self.conv1 = tf.nn.relu(self.pool1)

        # Layer 2: Input 14x14x6, Output 10x10x16
        self.conv2_kernels = tf.Variable(tf.truncated_normal(shape=[5, 5, 6, 16], mean=self.mul, stddev=self.sigma))
        self.conv2_biases = tf.get_variable(name="conv2_biases", shape=[16],
                                            initializer=tf.random_normal_initializer(stddev=self.sigma))
        self.conv2 = tf.nn.conv2d(self.conv1, self.conv2_kernels, [1, 1, 1, 1], padding='VALID') + self.conv2_biases

        # Pooling -> from 10x10x16 to 5x5x16
        self.pool2 = tf.nn.max_pool(self.conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

        # Activation 2
        self.conv2 = tf.nn.relu(self.pool2)

        # Flatten -> from 5x5x16 to 400x1
        self.flattened = flatten(self.conv2)

        # Fully Connected Layer n.1
        self.fcl1_weights = tf.Variable(tf.truncated_normal(shape=[400, 120], mean=self.mul, stddev=self.sigma))
        self.fcl1_biases = tf.get_variable(name="fc1_biases", shape=[120],
                                           initializer=tf.random_normal_initializer(stddev=self.sigma))
        self.fcl1 = tf.matmul(self.flattened, self.fcl1_weights) + self.fcl1_biases
```

```
        # Activation 3
        self.fcl1 = tf.nn.relu(self.fcl1)

        # Fully Connected Layer n.2
        self.fcl2_weights = tf.Variable(tf.truncated_normal(shape=[120, 84], mean=self.mul, stddev=self.sigma))
        self.fcl2_biases = tf.get_variable(name="fc2_biases", shape=[84],
                                           initializer=tf.random_normal_initializer(stddev=self.sigma))
        self.fcl2 = tf.matmul(self.fcl1, self.fcl2_weights) + self.fcl2_biases
        # Activation 4
        self.fcl2 = tf.nn.relu(self.fcl2)

        # Fully Connected Layer n.3
        self.fcl3_weights = tf.Variable(tf.truncated_normal(shape=[84, 10], mean=self.mul, stddev=self.sigma))
        self.fcl3_biases = tf.get_variable(name="fc3_biases", shape=[10],
                                           initializer=tf.random_normal_initializer(stddev=self.sigma))
        self.logits = tf.matmul(self.fcl2, self.fcl3_weights) + self.fcl3_biases

        # Loss and metrics
        self.cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(logits=self.logits, labels=self.Y)
        self.loss_op = tf.reduce_mean(self.cross_entropy)
        self.optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
        self.training_step = self.optimizer.minimize(self.loss_op)

        self.correct_prediction = tf.equal(tf.argmax(self.logits, 1), tf.argmax(self.Y, 1))
        self.accuracy_operation = tf.reduce_mean(tf.cast(self.correct_prediction, tf.float32))
```

2） 训练函数：

```
    def train(self, epochs, batch_size):
        assert (epochs > 0 and batch_size > 0)

        num_examples = len(self.training_data)

        print('Training . . .')

        with tf.Session() as session:
            session.run(tf.global_variables_initializer())
            total_steps = trange(epochs)
            for epoch in total_steps:
                self.training_data, self.training_label = shuffle(self.training_data, self.training_label)
                for offset in range(0, num_examples, batch_size):
                    end = offset + batch_size
                    X_batch, y_batch = self.training_data[offset:end], self.training_label[offset:end]

                    _, acc, cross = session.run([self.training_step, self.accuracy_operation, self.cross_entropy],
                                                feed_dict={self.X: X_batch, self.Y: y_batch})

                if self.validation_data.any() != None:
                    validation_accuracy = self.evaluate(self.validation_data, self.validation_label, batch_size)
                    total_steps.set_description(
                        "Epoch {}th - validation accuracy {:.3f} ".format(epoch + 1, validation_accuracy))

                    total_steps.set_description(
                        "Epoch {}th - validation accuracy {:.3f} ".format(epoch + 1, validation_accuracy))

            test_accuracy = self.evaluate(self.testing_data, self.testing_label, batch_size=batch_size)
            return test_accuracy
```

3） 识别函数：

```
    def evaluate(self, X_data, y_data, batch_size):
        examples = len(X_data)
        total_accuracy = 0
        sess = tf.get_default_session()
        for offset in range(0, examples, batch_size):
            batch_x, batch_y = X_data[offset:offset + batch_size], y_data[offset:offset + batch_size]
            accuracy = sess.run(self.accuracy_operation, feed_dict={self.X: batch_x, self.Y: batch_y})
            total_accuracy += (accuracy * len(batch_x))
        return total_accuracy / examples
```

3. 调用 tensorflow 的函数读取 MNIST 数据集：

```
# read data from mnist
mnist_data = input_data.read_data_sets("MNIST_data/", reshape=False, one_hot=True)

# classify data using module in tensorflow

# training data
train_X, train_Y = mnist_data.train.images, mnist_data.train.labels

# data for validating the trained network
validation_X, validation_Y = mnist_data.validation.images, mnist_data.validation.labels

# data for testing
test_X, test_Y = mnist_data.test.images, mnist_data.test.labels
```

4. 将自己手写的数字制作成可以被调用的形式。

```
# Testing my own hand_script
My_Hand_Script = np.zeros((100,28,28,1))

My_Lable = np.zeros((100,10))


for i in range(10):
    for j in range(10):
        # path for my own handscript
        filename = '/Users/chenrj/Desktop/my_handscript2/' + str(i) + '/' + str(j) + '.jpg'
        # read images
        image = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
        # image reshape to [28,28,1] to match the requirement of dataset
        image = np.array(image).reshape(28,28,1)
        # add new image to my data set
        My_Hand_Script[i*10+j]=image
        # lable my data set
        My_Lable[i*10+j][i]=1
```

5. 训练以及得到准确率。

```
# Lenet-5 initialing and training
my_lenet_network = ChenRJ_lenet5(train_X, train_Y, My_Hand_Script, My_Lable, validation_X, validation_Y)

# Testing and show the quality of the network
accuracy = my_lenet_network.train(epochs=10,batch_size=100)
print("Accuracy on mnist test set: {:.3f}".format(accuracy))
```

四、 实验结果与分析：

1. MNIST 测试数据的识别率：97.9%。

```
Input image shape: (28, 28, 1)
Size of training set: 55000
Size of validation set: 5000
Size of test set: 10000
Training the model . . .
2019-01-09 23:43:31.827167: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFl
ow binary was not compiled to use: AVX2 FMA
Epoch 10 - validation accuracy 0.982 : 100%|██████████| 10/10 [01:55<00:00, 12.12s/it]
Accuracy on MNIST test set: 0.979
```

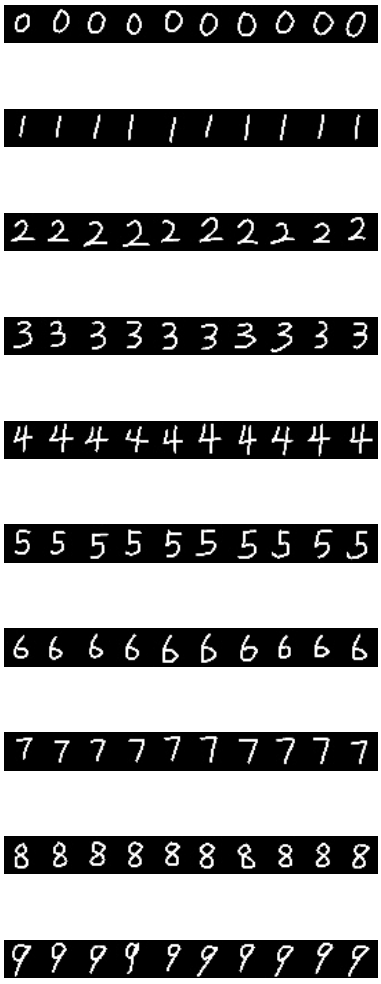
2. 自己手写的数字：一开始识别率比较低（40%左右）。

```
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
Input image shape: (28, 28, 1)
Size of training set: 55000
Size of validation set: 5000
Size of test set: 10000
Training the model . . .
Epoch 10 - validation accuracy 0.973 : 100%|███| 10/10 [02:02<00:00, 12.34s/it]
Accuracy on my own test set: 0.400
```

对比自己手写的数字跟 MNIST 数据集的数字之后，发现 0，2，5 和 9 这四个数字由于书写习惯的问题差别很大，导致了识别率比较低，因此重新进行了书写。

然后再观察写出来的数字，发现有断开或者笔画不连续的问题，因此再对手写的原始数据进行一定的预处理（膨胀操作等），最终使用的数据集贴在下面，识别率为 81%。由于书写习惯不同，还是存在一定的误差率，认为是正常的现象。

```
Input image shape: (28, 28, 1)
Size of training set: 55000
Size of validation set: 5000
Size of test set: 10000
Training the model . . .
2019-01-10 11:02:06.205326: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFl
ow binary was not compiled to use: AVX2 FMA
Epoch 10 - validation accuracy 0.952 : 100%|████████████████████████████████████| 10/10 [01:54<00:00, 11.59s/it]
Accuracy on my own test set: 0.810
```



3. 在测试的过程中发现，实现结果有一定的随机性，比如 MNIST 的测试数据集，在 97.5%到 98.5%之间波动。自己的数据集会在 80%左右波动。思考之后认为是在参数优化的过程中（下降法等），优化方式带有一定的随机性导致的。

4. 在运行的过程中，存在这样的一个警告：

```
Training the model . . .
2019-01-10 10:58:57.310041: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFl
ow binary was not compiled to use: AVX2 FMA
```

查询资料之后，发现是可以调用 CPU 的 AVX2，FMA 提高训练速度，应用以下代码调用：

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

调用之后，训练速度提高了，但是发现识别率明显降低，因此没有使用这种方法。

五、 结论与心得体会

在这一次的实验中，我实现了一个基础的 CNN：Lenet5。做了多组实验，实现效果比较好。

在本学期的课程中，我们学习了理论知识，也进行了很多代码的实现，能够用自己的代码实现一些很好玩的东西，确实是一件令人兴奋的事情，特别感谢老师和助教老师的辛苦付出！