浙江大学实验报告

专业：__数字媒体技术__
姓名：_____王涵___
学号：___3160102507__
日期：___2018/12/23__

课程名称：_计算机视觉___指导老师：__宋明黎_____成绩：_____

实验名称：__学习 CNN__

一、实验目的和要求（必填）

- 实验目的：利用 CNN 进行手写数字识别。
- 实验要求：框架用 TensorFlow，数据集用 MNIST 手写体数据集，网络结构为 LeNet-5。利用上述数据集、网络结构及框架实现手写数字的识别。
- 开发与运行环境：python opencv3.6，pycharm，windows10。

二、实验内容和原理（必填）

MINIST 数据集介绍：

MNIST 数据集是一个手写体数据集，数据集中每一个样本都是一个 0-9 的手写数字。该数据集由四部分组成，训练图片集，训练标签集，测试图片集和测试标签集。其中，训练集中有 60000 个样本，测试集中有 10000 个样本。每张照片均为 28*28 的二值图片，为方便存储，官方已对图片集进行处理，将每一张图片变成了维度为(1,784)的向量。

LeNet-5 介绍：

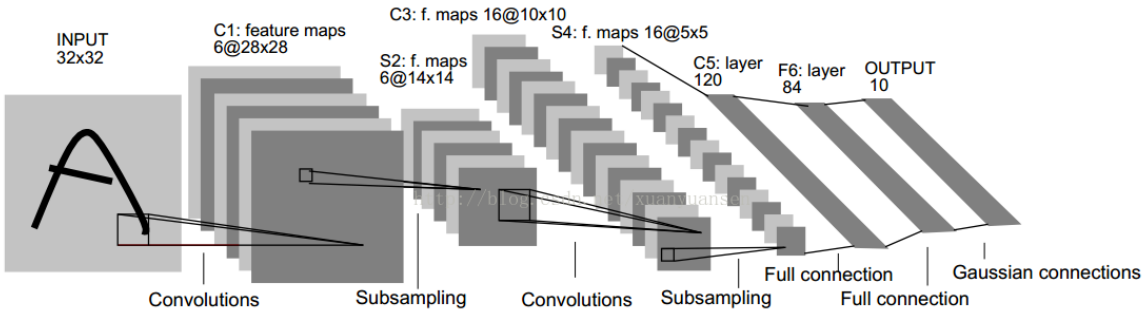

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet 是一种典型的卷积神经网络的结构，由 Yann LeCun 发明。它的网路结构如上。LeNet-5 共有 7 层（不包含输入），每层都包含可训练参数。

输入图像大小为 32x32，比 MNIST 数据集的图片要大一些，这么做的原因是希望潜在的明显特征如笔画断点或角能够出现在最高层特征检测子感受野（receptive field）的中心。因此在训练整个网络之前，需要对 28x28 的图像加上 paddings（即周围填充 0）。

C1 层：该层是一个卷积层。使用 6 个大小为 5x5 的卷积核对输入层进行卷积运算，特征图尺寸为 32-5+1=28，因此产生 6 个大小为 28x28 的特征图。这么做够防止原图像输入的信息掉到卷积核边界之外。

S2 层：该层是一个池化层（pooling，也称为下采样层）。这里采用 max_pool（最大池化），池化的 size 定为 2x2，经池化后得到 6 个 14x14 的特征图，作为下一层神经元的输入。

C3 层：该层仍为一个卷积层，我们选用大小为 5x5 的 16 种不同的卷积核。这里需要注意：

C3 中的每个特征图，都是 S2 中的所有 6 个或其中几个特征图进行加权组合得到的。输出为 16 个 10x10 的特征图。

S4 层：该层仍为一个池化层，size 为 2x2，仍采用 max_pool。最后输出 16 个 5x5 的特征图，神经元个数也减少至 16x5x5=400。

C5 层：该层我们继续用 5x5 的卷积核对 S4 层的输出进行卷积，卷积核数量增加至 120。这样 C5 层的输出图片大小为 5-5+1=1。最终输出 120 个 1x1 的特征图。这里实际上是与 S4 全连接了，但仍将其标为卷积层，原因是如果 LeNet-5 的输入图片尺寸变大，其他保持不变，那该层特征图的维数也会大于 1x1。

F6 层：该层与 C5 层全连接，输出 84 张特征图。

输出层：该层与 F6 层全连接，输出长度为 10 的张量，代表所抽取的特征属于哪个类别。（例如[0,0,0,1,0,0,0,0,0,0]的张量，1 在 index=3 的位置，故该张量代表的图片属于第三类）

三、实验结果

实验结果：Test accuracy 0.9812（使用激活函数为 sigmoid，dropout 的比例为 0.5）

四、代码

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

# read in MINIST data
mnist=input_data.read_data_sets("MNIST_data/", one_hot=True)
keep_prob = tf.placeholder("float")

def weight_variable(shape):
    initial=tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial=tf.constant(0.1,shape=shape)
    return tf.Variable(initial)

# convolution and pooling
def conv2d(x,W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='VALID')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')
```

```
# convolution layer
def lenet5_layer(layer, weight, bias):
    W_conv = weight_variable(weight)
    b_conv = bias_variable(bias)
    h_conv = conv2d(layer, W_conv)+b_conv
    return max_pool_2x2(h_conv)

# connected layer
def dense_layer(layer, weight, bias):
    W_fc = weight_variable(weight)
    b_fc = bias_variable(bias)
    return tf.matmul(layer, W_fc)+b_fc

def main():
    sess = tf.InteractiveSession()
    # input layer
    x = tf.placeholder("float", shape=[None, 784])
    y_ = tf.placeholder("float", shape=[None, 10])
    # first layer
    x_image = tf.pad(tf.reshape(x, [-1, 28, 28, 1]), [[0, 0], [2, 2], [2, 2], [0, 0]])
    layer = lenet5_layer(x_image, [5, 5, 1, 6], [6])
    # second layer
    layer = lenet5_layer(layer, [5, 5, 6, 16], [16])
    # third layer
    W_conv3 = weight_variable([5, 5, 16, 120])
    b_conv3 = bias_variable([120])
    layer = conv2d(layer, W_conv3) + b_conv3
    layer = tf.reshape(layer, [-1, 120])
    # all connected layer
    con_layer = dense_layer(layer, [120, 84], [84])
    # output
    con_layer = dense_layer(con_layer, [84, 10], [10])
    y_conv = tf.nn.softmax(tf.nn.dropout(con_layer, keep_prob))
    # train and evalute
    cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_,
```

```
    logits=y_conv))
    train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
    correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    sess.run(tf.global_variables_initializer())
    for i in range(30000):
        batch = mnist.train.next_batch(50)
        if i % 100 == 0:
            train_accuracy = accuracy.eval(feed_dict={
                x: batch[0], y_: batch[1], keep_prob: 1.0
            })
            print("step %d,training accuracy %g" % (i, train_accuracy))
        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
    print("Test accuracy %g" % accuracy.eval(feed_dict={
        x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0
    }))

if __name__=='__main__':
    main()
```