oolyjoo38的博客

基于Tensorflow的用MNIST手写数字做图像检索

标签： 机器学习 cnn 深度学习 图像检索 mnist手写体数字识别

2017-05-12 09:10 258人阅读 评论(0) 收藏

学机器学习或者CNN的大神们入门都必研究的就是MNIST手写数字识别，而MNIST数据集通常在各种平台上（Tensorflow,caffee,theao...）做分类的，而本人研究的是基于CNN的图像检索问题，怎么办呢？先从MNIST手写数字开始吧！

一开始我用的Tensorlayer去做检索（不了解tl的可以进入tl中文文档），用tl实现MNIST也是及其简单的，相比于用tf实现来讲代码上简单了些许，源代码戳这里。然而改代码的时候就发现了问题，因为在tl上一切变量都是以Tensor的形式运行的，用eval()转换也试过，总是提示错误。所以提取特征的时候没办法转换成数组，也就没办法计算距离。

本人使用的平台是Tensorflow，语言python，网络是最简单的LENET，源代码在github上能搜到，我是基于LENET把源代码改成图像检索的，检索的思路有两种方式，第一种是基于分类的检索，简单的来讲就是先做分类，然后输入一张测试图像，判别属于某一类，在把测试集中所有属于这一类的保存即可，这个比较简单，会分类的应该都会，本文不做介绍。第二种是基于特征的检索，简单的讲就是全连接层会输出维度为1024的特征（LENET），输入一张测试图片提取其特征，再将库里所有图像的特征都提取出来，进行特征距离的计算，然后根据实际情况设定一个阈值，低于这个阈值的图像保存即可。下面为大家讲解具体如何实现的。

我的运行环境是linux，python2.7，在win10下我也安装了Anaconda，python3.5，运行会提示出错，所以建议大家在python2.7下运行。

源代码的地址我找不到了，因此贴出：

```
import tensorflow as tf
import numpy as np

#导入input_data用于自动下载和安装MNIST数据集
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

#创建一个交互式Session
sess = tf.InteractiveSession()

#创建两个占位符，x为输入网络的图像，y_为输入网络的图像类别
x = tf.placeholder("float", shape=[None, 784])
y_ = tf.placeholder("float", shape=[None, 10])

#权重初始化函数
def weight_variable(shape):
    #输出服从截尾正态分布的随机值
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

#偏置初始化函数
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

#创建卷积op
#x 是一个4维张量，shape为[batch,height,width,channels]
#卷积核移动步长为1。填充类型为SAME,可以不丢弃任何像素点
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1,1,1,1], padding="SAME")

#创建池化op
#采用最大池化，也就是取窗口中的最大值作为结果
#x 是一个4维张量，shape为[batch,height,width,channels]
#ksize表示pool窗口大小为2x2,也就是高2，宽2
#strides，表示在height和width维度上的步长都为2
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1,2,2,1],
                          strides=[1,2,2,1], padding="SAME")


#第1层，卷积层
#初始化W为[5,5,1,32]的张量，表示卷积核大小为5*5，第一层网络的输入和输出神经元个数分别为1和32
```

关闭

```
W_conv1 = weight_variable([5,5,1,32])
#初始化b为[32],即输出大小
b_conv1 = bias_variable([32])

#把输入x(二维张量,shape为[batch, 784])变成4d的x_image，x_image的shape应该是[batch,28,28,1]
#-1表示自动推测这个维度的size
x_image = tf.reshape(x, [-1,28,28,1])

#把x_image和权重进行卷积，加上偏置项，然后应用ReLU激活函数，最后进行max_pooling
#h_pool1的输出即为第一层网络输出，shape为[batch,14,14,1]
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

#第2层，卷积层
#卷积核大小依然是5*5，这层的输入和输出神经元个数为32和64
W_conv2 = weight_variable([5,5,32,64])
b_conv2 = weight_variable([64])

#h_pool2即为第二层网络输出，shape为[batch,7,7,1]
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

#第3层, 全连接层
#这层是拥有1024个神经元的全连接层
#W的第1维size为7*7*64，7*7是h_pool2输出的size，64是第2层输出神经元个数
W_fc1 = weight_variable([7*7*64, 1024])
b_fc1 = bias_variable([1024])

#计算前需要把第2层的输出reshape成[batch, 7*7*64]的张量
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

#Dropout层
#为了减少过拟合，在输出层前加入dropout
keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

#输出层
#最后，添加一个softmax层
#可以理解为另一个全连接层，只不过输出时使用softmax将网络输出值转换成了概率
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

#预测值和真实值之间的交叉墒
```

关闭

```
cross_entropy = -tf.reduce_sum(y_ * tf.log(y_conv))

#train op, 使用ADAM优化器来做梯度下降。学习率为0.0001
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

#评估模型，tf.argmax能给出某个tensor对象在某一维上数据最大值的索引。
#因为标签是由0,1组成了one-hot vector，返回的索引就是数值为1的位置
correct_predict = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))

#计算正确预测项的比例，因为tf.equal返回的是布尔值，
#使用tf.cast把布尔值转换成浮点数，然后用tf.reduce_mean求平均值
accuracy = tf.reduce_mean(tf.cast(correct_predict, "float"))

#初始化变量
sess.run(tf.global_variables_initializer())

#开始训练模型，循环20000次，每次随机从训练集中抓取50幅图像
for i in range(20000):
    batch = mnist.train.next_batch(50)
    if i%100 == 0:
        #每100次输出一次日志
        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})
```

好了好了，言归正传，我们要如何把分类改成检索呢？

其实很简单。首先你要训练好这个网络然后保存一下权重，然后加载权重。

```
#保存权重
saver = tf.train.Saver()
save_path = saver.save(sess, "/home/jsj/LYJ/QZ1/LYJ.ckpt")
print "Model saved in file: ", save_path
#加载权重
saver = tf.train.Saver()
saver.restore(sess, "/home/jsj/LYJ/QZ1/LYJ.ckpt")
print "Model restored."
```

关闭

第二步，随机输入一张测试图片，提取其特征。

```
batch2 = mnist.test.next_batch(1)
b = h_fc1.eval(feed_dict={x:batch2[0],y_:batch2[1],keep_prob:1.0})
print batch2[1]#打印检索图像的标签
#h_fc1为全连接层的输出,eval()把特征tensor转成数组向量
```

下面写一个简单的循环，一次随机进50张，当然我的50张是直接从训练集里提取的，因为我个人认为检索毕竟不是分类，你只要把相似的列出就可以了，如果你想从测试集里提取也可以，把train改成test就ok了！

```
for i in range(50):
    batch1 = mnist.train.next_batch(1)#随机进50张作为检索图像库
    if i%1 == 0:
        a = h_fc1.eval(feed_dict={x:batch1[0],y_:batch1[1],keep_prob:1.0})#提取特征
        a1 = tf.reshape(batch1[0],[28,28]).eval()#转成向量
        a2 = 255*a1
        a3 = a2.astype(np.uint64)#a3为进入这50张的原图像
        dist = sum(abs(a-b))#计算距离公式

#我用的是绝对值距离，比较简单，当然也可以用欧氏距离（dist = np.linalg.norm(a - b) ），不过阈值要改一下

        print dist#如果你想看一下就打印，不看就不打印
        if dist<=350:#根据实际情况自设阈值
          cv2.imwrite('/home/jsj/LYJ/22/'+str(i)+'.jpg',a3)
            i=i+1#把符合条件的图像保存到文件夹中，最前面别忘记导入opencv库,加import cv2
```

那么现在问题来了，当我库里只有50张的时候，可以手动查出我检出了几个检对了几个，如果200张呢？500张呢？我都要一张一张查吗？当然不是！我需要添加几行代码进去，计算查全率和查准率，还有F值，通过这些参数去调整我的阈值，让整个系统的检索准确率提高。

下面是把测试集中所有图像都导入网络，提取特征。这个地方有一点不方便的就是首先你需要判定你输入的那一张测试图像，即检索的图像是属于哪一类别的。

关闭

```
count = 0#相关总数
count1 = 0#检索到相关的
count2 = 0#所有检索出来的
for i in range(1):
    batch1 = mnist.test.next_batch(10000)
    a = h_fc1.eval(feed_dict={x:batch1[0],y_:batch1[1],keep_prob:1.0})
    B = tf.argmax(batch1[1], 1).eval()
    for j in range(10000):
        if B[j] ==7:#检索图像为数字7
            count += 1
    a1 = tf.reshape(batch1[0],[10000,28,28]).eval()
```

```
    a2 = 255*a1
    a3 = a2.astype(np.uint64)
    for k in range(10000):
        dist = np.linalg.norm(a[k] - b)#使用欧氏距离
        if dist<=25:#阈值25
            count2 +=1
            if B[k] ==7:#检索图像为数字7
                count1 += 1

R = count1/count#查全率
P = count1/count2#查准率
F = R*P*2/R+P#F值
```

好了，完成！贴出完整代码：

```
import tensorflow as tf
import numpy as np
import cv2

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

sess = tf.InteractiveSession()

x = tf.placeholder("float", shape=[None, 784])
y_ = tf.placeholder("float", shape=[None, 10])

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)

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1,1,1,1], padding="SAME")

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

W_conv1 = weight_variable([5,5,1,32])
b_conv1 = bias_variable([32])

```

关闭

```
x_image = tf.reshape(x, [-1,28,28,1])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

W_conv2 = weight_variable([5,5,32,64])
b_conv2 = weight_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

W_fc1 = weight_variable([7*7*64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

cross_entropy = -tf.reduce_sum(y_ * tf.log(y_conv))

train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

correct_predict = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))

accuracy = tf.reduce_mean(tf.cast(correct_predict, "float"))

sess.run(tf.initialize_all_variables())

for i in range(20000):
    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})

saver = tf.train.Saver()
save_path = saver.save(sess, "/home/jsj/LYJ/QZ1/LYJ.ckpt")
```

关闭

```
print "Model saved in file: ", save_path

saver = tf.train.Saver()
saver.restore(sess, "/home/jsj/LYJ/QZ1/LYJ.ckpt")
print "Model restored."

batch2 = mnist.test.next_batch(1)
b = h_fc1.eval(feed_dict={x:batch2[0],y_:batch2[1],keep_prob:1.0})
print b
print batch2[1]

count = 0
count1 = 0
count2 = 0
for i in range(1):
    batch1 = mnist.test.next_batch(10000)
    a = h_fc1.eval(feed_dict={x:batch1[0],y_:batch1[1],keep_prob:1.0})
    B = tf.argmax(batch1[1], 1).eval()
    for j in range(10000):
        if B[j] ==7:
            count += 1
    a1 = tf.reshape(batch1[0],[10000,28,28]).eval()
    a2 = 255*a1
    a3 = a2.astype(np.uint64)
    for k in range(10000):
        dist = np.linalg.norm(a[k] - b)
        if dist<=25:
            cv2.imwrite('/home/jsj/LYJ/22/'+str(k)+'.jpg',a3[k])
            count2 +=1
            if B[k] ==7:
                count1 += 1

R = count1/count
P = count1/count2
F = R*P*2/R+P
```

建议训练和测试分开运行，训练完毕以后保存权重，测试时候把网络架构加载一下，再加载一下权重即可。

- 上一篇 斯坦福CS20SI：基于Tensorflow的深度学习研究课程笔记，Lecture note3：TensorFlow上的线性回归和逻辑回归

相关文章推荐

- 斯坦福CS20SI：基于Tensorflow的深度学习研究课…
- Presto的服务治理与架构在京东的实践与应用--王哲…
- 斯坦福CS20SI：基于Tensorflow的深度学习研究课…
- 深入掌握Kubernetes应用实践--王渊命
- [翻译]斯坦福CS20SI:基于Tensorflow的深度学习研…
- Python基础知识汇总
- 基于Tensorflow的用MNIST手写数字做图像检索
- Android核心技术详解
- tensorflow-mnist手写数字识别
- Retrofit 从入门封装到源码解析
- Tensorflow的Helloword：使用简单Softmax Regressi…
- 自然语言处理工具Word2Vec
- TensorFlow学习_02_CNN卷积神经网络_Mnist手写…
- 基于tensorflow的MNIST手写数字识别（三）--神经…
- Tensorflow系列之（二）：详解CNN识别MNIST手…
- TensorFlow实现机器学习的“Hello World”--Mnist手…

查看评论

暂无评论