使用MNIST手写数字识别的代码：

#encoding=utf-8

import tensorflow as tf

import numpy as np

from tensorflow.examples.tutorials.mnist import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

def weight\_variable(shape):

initial = tf.truncated\_normal(shape,stddev=0.1) #截断正态分布，此函数原型为尺寸、均值、标准差

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') # strides第0位和第3为一定为1，剩下的是卷积的横向和纵向步长

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x,ksize = [1,2,2,1],strides=[1,2,2,1],padding='SAME')# 参数同上，ksize是池化块的大小

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

y\_ = tf.placeholder("float", shape=[None, 10])

# 图像转化为一个四维张量，第一个参数代表样本数量，-1表示不定，第二三参数代表图像尺寸，最后一个参数代表图像通道数

x\_image = tf.reshape(x,[-1,28,28,1])

# 第一层卷积加池化

w\_conv1 = weight\_variable([5,5,1,32]) # 第一二参数值得卷积核尺寸大小，即patch，第三个参数是图像通道数，第四个参数是卷积核的数目，代表会出现多少个卷积特征

b\_conv1 = bias\_variable([32])

h\_conv1 = tf.nn.relu(conv2d(x\_image,w\_conv1)+b\_conv1)

h\_pool1 = max\_pool\_2x2(h\_conv1)

# 第二层卷积加池化

w\_conv2 = weight\_variable([5,5,32,64]) # 多通道卷积，卷积出64个特征

b\_conv2 = bias\_variable([64])

h\_conv2 = tf.nn.relu(conv2d(h\_pool1,w\_conv2)+b\_conv2)

h\_pool2 = max\_pool\_2x2(h\_conv2)

# 原图像尺寸28\*28，第一轮图像缩小为14\*14，共有32张，第二轮后图像缩小为7\*7，共有64张

w\_fc1 = weight\_variable([7\*7\*64,1024])

b\_fc1 = bias\_variable([1024])

h\_pool2\_flat = tf.reshape(h\_pool2,[-1,7\*7\*64]) # 展开，第一个参数为样本数量，-1未知

f\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat,w\_fc1)+b\_fc1)

# dropout操作，减少过拟合

keep\_prob = tf.placeholder(tf.float32)

h\_fc1\_drop = tf.nn.dropout(f\_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)) # 定义交叉熵为loss函数

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy) # 调用优化器优化

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

sess = tf.InteractiveSession()

sess.run(tf.initialize\_all\_variables())

for i in range(2000):

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[0:500], y\_: mnist.test.labels[0:500], keep\_prob: 1.0}))

运行结果：