```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data   #手写数字相关的数据包
```

```
# 载入数据集
mnist = input_data.read_data_sets("MNIST_data",one_hot=True)    #载入数据，{数据集包路径，把标签转化为只有0和1的形式}

#定义变量，即每个批次的大小
batch_size = 100    #一次放100章图片进去
n_batch = mnist.train.num_examples // batch_size   #计算一共有多少个批次；训练集数量（整除）一个批次大小


#初始化权值
def weight_variable(shape):
    initial = tf.truncated_normal(shape,stddev=0.1)   #生成一个截断的正态分布
    return tf.Variable(initial)

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

#卷积层
def conv2d(x,W):
    #x input tensor of shape '[batch, in_height, in_width, in_channels]'
    #W filter / kernel tensor of shape [filter_height, filter_width, in_channels, out_channels]
    #'strides[0] = strides[3] = 1', strides[1]代表x方向的步长，strides[2]代表y方向的步长
    #padding:A 'string' frome: '"SAME"（补0）, "VALID"（不补0）'
    return tf.nn.conv2d(x,W,strides=[1,1,1,1],padding='SAME')

#池化层
def max_pool_2x2(x):
    #ksize [1,x,y,1]（窗口大小）
    return tf.nn.max_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')


#定义两个placeholder
x = tf.placeholder(tf.float32,[None,784])    #[行不确定，列为784]：28*28
y = tf.placeholder(tf.float32,[None,10])    #数字为0-9，则为10

#改变x的格式转为4D的向量[batch, in_height, in_width, in_channels]
x_image = tf.reshape(x,[-1,28,28,1])

#初始化第一个卷积层的权值和偏置
W_conv1 = weight_variable([5,5,1,32])  #5*5的采样窗口，32个卷积核从1个平面抽取特征
b_conv1 = bias_variable([32])  #每个卷积核一个偏置
```

```
43
44  #把x_image和权值向量进行卷积，再加上偏置值，然后应用于relu激活函数
45  h_conv1 = tf.nn.relu(conv2d(x_image,W_conv1) + b_conv1)
46  h_pool1 = max_pool_2x2(h_conv1)   #进行max-pooling
47
48  #初始化第二个卷积层的权值和偏置
49  W_conv2 = weight_variable([5,5,32,64])  #5*5的采样窗口，32个卷积核从1个平面抽取
    特征
50  b_conv2 = bias_variable([64])  #每个卷积核一个偏置
51
52  #把h_pool1和权值向量进行卷积，再加上偏置值，然后应用于relu激活函数
53  h_conv2 = tf.nn.relu(conv2d(h_pool1,W_conv2) + b_conv2)
54  h_pool2 = max_pool_2x2(h_conv2)   #进行max-pooling
55
56  #28*28的图片第一次卷积后还是28*28，第一次池化后变为14*14
57  #第二次卷积后为14*14，第二次池化后变为7*7
58  #经过上面的操作后得到64张7*7的平面
59
60  #初始化第一个全连接层的权值
61  W_fc1 = weight_variable([7*7*64,1024])  #上一层有7*7*64个神经元，全连接层有1024
    个神经元
62  b_fc1 = bias_variable([1024])  #1024个节点
63
64  #把池化层2的输出扁平化为1维
65  h_pool2_flat = tf.reshape(h_pool2,[-1,7*7*64])
66  #求第一个全连接层的输出
67  h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat,W_fc1) + b_fc1)
68
69  #keep_prob用来表示神经元的输出概率
70  keep_prob = tf.placeholder(tf.float32)
71  h_fc1_drop = tf.nn.dropout(h_fc1,keep_prob)
72
73  #初始化第二个全连接层
74  W_fc2 = weight_variable([1024,10])
75  b_fc2 = bias_variable([10])
76
77  #计算输出
78  prediction = tf.nn.softmax(tf.matmul(h_fc1_drop,W_fc2) + b_fc2)
79
80
81  #定义交叉熵代价函数
82  cross_entropy =
    tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=pred
    iction))
83  #使用AdamOptimizer进行优化
84  train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
85
86  #准确数，结果存放在一个布尔型列表中
87  correct_prediction = tf.equal(tf.argmax(prediction,1),tf.argmax(y,1))   #比
    较两个参数大小是否相同，同则返回为true，不同则返回为false；argmax()：返回张量中最大的值
    所在的位置
88
89  #求准确率
90  accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
    #cast()：将布尔型转换为32位的浮点型；（比方说9个T和1个F，则为9个1，1个0，即准确率为
    90%）
91
92  with tf.Session() as sess:
```

```
    sess.run(tf.global_variables_initializer())
    for epoch in range(21):
        for batch in range(n_batch):
            batch_xs,batch_ys = mnist.train.next_batch(batch_size)
            sess.run(train_step,feed_dict={x:batch_xs,y:batch_ys,keep_prob:0.7})

        acc = sess.run(accuracy,feed_dict={x:mnist.test.images,y:mnist.test.labels,keep_prob:1.0})
        print("Iter" + str(epoch) + ",Testing Accuracy" + str(acc))
```

```
Extracting MNIST_data\train-images-idx3-ubyte.gz
Extracting MNIST_data\train-labels-idx1-ubyte.gz
Extracting MNIST_data\t10k-images-idx3-ubyte.gz
Extracting MNIST_data\t10k-labels-idx1-ubyte.gz
Iter0,Testing Accuracy0.863
Iter1,Testing Accuracy0.8757
Iter2,Testing Accuracy0.881
Iter3,Testing Accuracy0.8833
```

```
---------------------------------------------------------------------------

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-11-b59272fba4b4> in <module>
     95         for batch in range(n_batch):
     96             batch_xs,batch_ys = mnist.train.next_batch(batch_size)
---> 97             sess.run(train_step,feed_dict={x:batch_xs,y:batch_ys,keep_prob:0.7})
     98
     99         acc = sess.run(accuracy,feed_dict={x:mnist.test.images,y:mnist.test.labels,keep_prob:1.0})
```

```
D:\anaconda\lib\site-packages\tensorflow\python\client\session.py in run(self, fetches, feed_dict, options, run_metadata)
    948     try:
    949       result = self._run(None, fetches, feed_dict, options_ptr,
--> 950                          run_metadata_ptr)
    951       if run_metadata:
    952         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)
```

```
D:\anaconda\lib\site-packages\tensorflow\python\client\session.py in _run(self, handle, fetches, feed_dict, options, run_metadata)
   1171     if final_fetches or final_targets or (handle and
feed_dict_tensor):
   1172       results = self._do_run(handle, final_targets, final_fetches,
-> 1173                              feed_dict_tensor, options, run_metadata)
   1174     else:
   1175       results = []
```

```
D:\anaconda\lib\site-packages\tensorflow\python\client\session.py in
_do_run(self, handle, target_list, fetch_list, feed_dict, options,
run_metadata)
   1348     if handle is None:
   1349       return self._do_call(_run_fn, feeds, fetches, targets, options,
-> 1350                            run_metadata)
   1351     else:
   1352       return self._do_call(_prun_fn, handle, feeds, fetches)
```

```
D:\anaconda\lib\site-packages\tensorflow\python\client\session.py in
_do_call(self, fn, *args)
   1354   def _do_call(self, fn, *args):
   1355     try:
-> 1356       return fn(*args)
   1357     except errors.OpError as e:
   1358       message = compat.as_text(e.message)
```

```
D:\anaconda\lib\site-packages\tensorflow\python\client\session.py in
_run_fn(feed_dict, fetch_list, target_list, options, run_metadata)
   1339       self._extend_graph()
   1340       return self._call_tf_sessionrun(
-> 1341           options, feed_dict, fetch_list, target_list, run_metadata)
   1342
   1343     def _prun_fn(handle, feed_dict, fetch_list):
```

```
D:\anaconda\lib\site-packages\tensorflow\python\client\session.py in
_call_tf_sessionrun(self, options, feed_dict, fetch_list, target_list,
run_metadata)
   1427     return tf_session.TF_SessionRun_wrapper(
   1428         self._session, options, feed_dict, fetch_list, target_list,
-> 1429         run_metadata)
   1430
   1431   def _call_tf_sessionprun(self, handle, feed_dict, fetch_list):
```

```
KeyboardInterrupt:
```

```

```