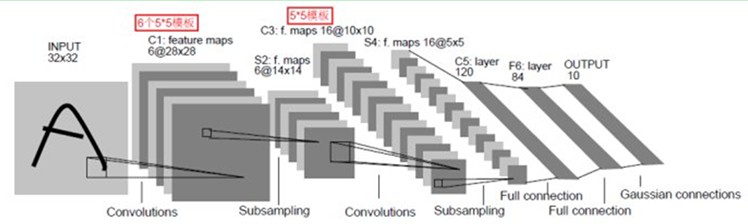
在之前的[tensorflow笔记：流程，概念和简单代码注释](http://blog.csdn.net/u014595019/article/details/52677412" \t "_blank) 文章中，已经大概解释了tensorflow的大概运行流程，并且提供了一个mnist数据集分类器的简单实现。当然，因为结构简单，最后的准确率在91%左右。似乎已经不低了？其实这个成绩是非常不理想的。现在mnist的准确率天梯榜已经被刷到了99.5%以上。为了进一步提高准确率，官网还提供了一个多层的CNN分类器的代码。相比之前的一层神经网络，这份代码的主要看点倒不是多层，而是CNN，也就是卷积神经网络。

CNN的具体内容不再详述，概述可以参考[这里](http://blog.csdn.net/zouxy09/article/details/8781543" \t "_blank)，详细信息可以参见[Convolutional Networks](http://www.deeplearningbook.org/contents/convnets.html)。一般来说，CNN网络的前几层为卷积层和采样层（或者说池化层），在若干层卷积和池化以后，还有若干层全连接层（也就是传统神经网络），最后输出分类信息。大概的结构示意图如下图所示



可以看到**，CNN相比与传统神经网络，最大的区别就是引入了卷积层和池化层**。这也是我们在代码中要着重看的地方。   
在下面的代码中，卷积是使用tf.nn.conv2d, 池化使用tf.nn.max\_pool,下面来详细的讲解一下这两个函数的用法。

**tf.nn.conv2d**

这个函数的功能是：给定4维的input和filter，计算出一个2维的卷积结果。函数的定义为：

def conv2d(input, filter, strides, padding, use\_cudnn\_on\_gpu=None,

data\_format=None, name=None):

前几个参数分别是input, filter, strides, padding, use\_cudnn\_on\_gpu, …下面来一一解释   
**input**：待卷积的数据。格式要求为一个张量，**[batch, in\_height, in\_width, in\_channels]**.   
分别表示 批次数，图像高度，宽度，输入通道数。   
**filter**： 卷积核。格式要求为**[filter\_height, filter\_width, in\_channels, out\_channels]**.   
分别表示 卷积核的高度，宽度，输入通道数，输出通道数。   
**strides** :一个长为4的list. 表示每次卷积以后卷积窗口在input中滑动的距离   
**padding** ：有SAME和VALID两种选项，表示是否要保留图像边上那一圈不完全卷积的部分。如果是SAME，则保留   
**use\_cudnn\_on\_gpu** ：是否使用cudnn加速。默认是True

**tf.nn.max\_pool**   
进行最大值池化操作,而avg\_pool 则进行平均值池化操作.函数的定义为：

def max\_pool(value, ksize, strides, padding, data\_format="NHWC", name=None):

* 1

**value:** 一个4D张量，格式为**[batch, height, width, channels]**，与conv2d中input格式一样   
**ksize:** 长为4的list,表示池化窗口的尺寸   
**strides:** 池化窗口的滑动值，与conv2d中的一样   
**padding:** 与conv2d中用法一样。

具体的代码注释如下：

# Copyright 2015 The TensorFlow Authors. All Rights Reserved.

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

# http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

# ==============================================================================

"""A very simple MNIST classifier.

See extensive documentation at

http://tensorflow.org/tutorials/mnist/beginners/index.md

"""

from \_\_future\_\_ import absolute\_import

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

# Import data

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

import tensorflow as tf

flags = tf.app.flags

FLAGS = flags.FLAGS

flags.DEFINE\_string('data\_dir', '/tmp/data/', 'Directory for storing data') # 第一次启动会下载文本资料，放在/tmp/data文件夹下

print(FLAGS.data\_dir)

mnist = input\_data.read\_data\_sets(FLAGS.data\_dir, one\_hot=True)

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1) # 变量的初始值为截断正太分布

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

"""

tf.nn.conv2d功能：给定4维的input和filter，计算出一个2维的卷积结果

前几个参数分别是input, filter, strides, padding, use\_cudnn\_on\_gpu, ...

input 的格式要求为一个张量，[batch, in\_height, in\_width, in\_channels],批次数，图像高度，图像宽度，通道数

filter 的格式为[filter\_height, filter\_width, in\_channels, out\_channels]，滤波器高度，宽度，输入通道数，输出通道数

strides 一个长为4的list. 表示每次卷积以后在input中滑动的距离

padding 有SAME和VALID两种选项，表示是否要保留不完全卷积的部分。如果是SAME，则保留

use\_cudnn\_on\_gpu 是否使用cudnn加速。默认是True

"""

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

def max\_pool\_2x2(x):

"""

tf.nn.max\_pool 进行最大值池化操作,而avg\_pool 则进行平均值池化操作

几个参数分别是：value, ksize, strides, padding,

value: 一个4D张量，格式为[batch, height, width, channels]，与conv2d中input格式一样

ksize: 长为4的list,表示池化窗口的尺寸

strides: 窗口的滑动值，与conv2d中的一样

padding: 与conv2d中用法一样。

"""

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1],

strides=[1, 2, 2, 1], padding='SAME')

sess = tf.InteractiveSession()

x = tf.placeholder(tf.float32, [None, 784])

x\_image = tf.reshape(x, [-1,28,28,1]) #将输入按照 conv2d中input的格式来reshape，reshape

"""

# 第一层

# 卷积核(filter)的尺寸是5\*5, 通道数为1，输出通道为32，即feature map 数目为32

# 又因为strides=[1,1,1,1] 所以单个通道的输出尺寸应该跟输入图像一样。即总的卷积输出应该为?\*28\*28\*32

# 也就是单个通道输出为28\*28，共有32个通道,共有?个批次

# 在池化阶段，ksize=[1,2,2,1] 那么卷积结果经过池化以后的结果，其尺寸应该是？\*14\*14\*32

"""

W\_conv1 = weight\_variable([5, 5, 1, 32]) # 卷积是在每个5\*5的patch中算出32个特征，分别是patch大小，输入通道数目，输出通道数目

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

h\_conv1 = tf.nn.elu(conv2d(x\_image, W\_conv1) + b\_conv1)

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

"""

# 第二层

# 卷积核5\*5，输入通道为32，输出通道为64。

# 卷积前图像的尺寸为 ?\*14\*14\*32， 卷积后为?\*14\*14\*64

# 池化后，输出的图像尺寸为?\*7\*7\*64

"""

W\_conv2 = weight\_variable([5, 5, 32, 64])

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

h\_conv2 = tf.nn.elu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

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

# 第三层 是个全连接层,输入维数7\*7\*64, 输出维数为1024

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.elu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

keep\_prob = tf.placeholder(tf.float32) # 这里使用了drop out,即随机安排一些cell输出值为0，可以防止过拟合

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

# 第四层，输入1024维，输出10维，也就是具体的0~9分类

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) # 使用softmax作为多分类激活函数

y\_ = tf.placeholder(tf.float32, [None, 10])

cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(y\_conv), reduction\_indices=[1])) # 损失函数，交叉熵

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy) # 使用adam优化

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1)) # 计算准确度

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

sess.run(tf.initialize\_all\_variables()) # 变量初始化

for i in range(20000):

batch = mnist.train.next\_batch(50)

if i%100 == 0:

# print(batch[1].shape)

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}))