1.卷积神经网络

卷积神经网络一般有卷积层和池化层，卷积层就是通过一个小正方型的过滤器，其深度与当前层的深度一样，通过一定的步长对图像进行扫描，输出的矩阵可以长宽与被卷积层一样，也可以比它小，而输出的深度可以人为设定，该深度就是下一层卷积层的深度。

卷积的目的就是减少图像长宽，增加图像的深度，提取更高的特征，所以池化层一般就是输出的长宽为输入的一半，而深度不变。

经典结构：卷积层，池化层，卷积层，池化层，全连接层，全连接处，全连接层。

卷积层：

filter\_weight= tf.get\_variable(‘weight’,[5,5,3,16],initializer=tf.truncater\_normal\_initializer(stddev=0.1))

过滤器长宽5x5，当前层深度3，输出层深度16

Biase = tf.get\_variable(‘biase’,[16],initializer=tf.constant\_initializer(0.1))

Conv= tf.nn.conv2d(input,filter\_weight,strides=[1,1,1,1,],padding=’same’)

Input为四维矩阵，第1参数代表batch，第2,3,4对应一个节点矩阵，步长1，填充0使输出大小不变。

提供专门的加偏置项函数

Bias = tf.nn.biase\_add(conv,biase)

Actived\_conv = tf.nn.relu(biases)

池化层：

Pool= tf.nn.max\_pool(Actived\_conv,ksize=[1,3,3,1],strides=[1,2,2,1,],padding=’same’)

过滤器3x3，步长2，填充0，所以输出长宽为输入的一半。

池化层与全连接层维度不一样，所以输出需要改变维度

Pool\_shape = pool.get\_shape().as\_list()

Nodes = pool\_shape[1]\* pool\_shape[2]\* pool\_shape[3]

Reshape = tf.reshape(pool,[ pool\_shape[0],nodes])

更简洁的卷积层函数，自动默认步长1，填充0。且自动加偏置项去线性化。

Net = slim.conv2d(input,32,[3,3],scope=’convd3\_3’)

实现矩阵拼接，在维度3拼接

Branch\_1 = tf.concat(3,

[slim.conv2d(branch\_1,384,[1,3],scop=’conv2d\_1x3’),

slim.conv2d(branch\_1,384,[3,1],scop=’conv2d\_3x1’)

])

2.图像处理

除了对神经网络的优化，对图像的处理也可以提升训练效果。

对图像处理就是随机改变图像的大小，亮度等一系列可调参数来作为训练材料，在不增加图片数量来增加训练量，且减小图片亮度对比度等对训练效果的影响。

#获取图像原始数据

Image\_raw\_data= tf.gfile.FastGFile(“/path/to/picture”,’r’).read()

#解码

Imag\_data = tf.image.decode\_jpeg(Image\_raw\_data)

#转化成实数

Imag\_data = tf.image.convert\_image\_dtype(Imag\_data, dtype=tf.float32)

Encoded\_image = tf.image.encode\_jpeg(Imag\_data)

With tf.gfile.GFile(“/path/to/output”,”wb”) as f:

f.write(Encoded\_image.eval())

TFRecord格式是tensorflow提供的图片统一保存格式

3.队列与多线程

q=tf.FIFOQueue(2,”int32”)

Init=q.enqueue\_many(([0,10],))

X=q.dequeue()

q\_inc=q.enqueue([y])

多线程：

def myloop(coord ,worker\_id):

……

cord = tf.train.Coordinator()

Threads = [

threading.Thread(target = myloop, args = (coord,I,) ) for I in xrang(5)

]

for t in threats ：

t.start()

coord.join(threads)#等待所有线程退出

管理多线程

queue = tf.FIFOQueue(100,”float”)

enqueue\_op = queue.enqueue([tf.random\_normal([1])])

#创建5个线程对队列执行操作

qr = tf.train.QueueRunner(queue,[enqueue\_op]\*5)

#加入默认集合的tf.GraphKeys.QUEUE\_RUNNER

Tf.train.add\_queue\_runner(qr)

coord = tf.train.Coordinator()

#用tf.train.QueueRunner，需要明确调用tf.train.start\_queue\_runners#来启动所有线程。

threads = tf.train.start\_queue\_runners(sess=sess, coord=coord)

coord.request\_stop()#请求

coord.should\_stoop()#当发出请求，should\_stop返回值为True

4.输入文件队列

#获取文件列表

files = tf.train.match\_filenames\_one(“/path/to/data.tfrecprds-\*”)

#生成输入队列

Filename\_queue = tf.train.string\_input\_producer(files, shuffle = False)