**CTC Loss:**

为了解决预测结果与标签难以对齐而提出的一种损失函数计算方式。

**Tensorflow 中的CTC Loss:**

**训练:**

**1.三个输入:**

A.图片数据:

inputs = tf.placeholder(tf.float32, [None, input\_height,input\_width,3],name="inputs")

B.图片标签:

sparse\_groundtrouth = tf.sparse\_placeholder(tf.int32,name="sparse\_gt")

以稀疏矩阵的形式作为输入,以(indices,values,shape)的形式存放,通过get\_sparse\_labels函数对便签列表-[“1524253”,”4353255”......]进行稀疏解析

C.每个样本的序列长度:

seq\_len\_placeholder = tf.placeholder(tf.int32, [None],name="seq\_len\_gt")

其在feed时是一个长度为batch\_size的列表，列表中每个元素值是time\_steps(即一个样本的序列个数,这里以每张图片的每列作为一个序列,所以每个样本的序列个数是time\_steps=input\_width)

eg. seq\_len = np.ones(batch\_size) \* input\_width可以用于feed变量seq\_len\_placeholder

**2.一个输出:**

logits = ctcRecognizer.inference(processed\_inputs,seq\_len\_placeholder)

无论是灰度图还是三通道图作为输入,又或是网络架构采用CNN或RNN或者CNN和RNN的组合，最终都要保证输出logits的shape是(time\_steps,batch\_size,num\_class)，后面方可通过logits和sparse\_groundtrouth计算损失

**3.Loss计算:**

通过tf.nn.ctc\_beam\_search\_decoder(logits,seq\_len,merge\_repeated=is\_merge)将logitsg变为解析后的decoded\_logits,再通过:

loss=tf.nn.ctc\_loss(labels=sparse\_groundtrouth,inputs=logits,sequence\_length=seq\_len)

loss\_mean=tf.reduce\_mean(loss)

至此即可得到最终的损失值，并使用优化器迭代式地最小化该损失

**测试:**

1. **稀疏矩阵形式-----不推荐**

通过sess.run得到decode\_logits[0]的值，即可得到该批样本预测结果的稀疏形式-test\_decode\_logits,再放入decode\_sparse\_tensor函数进行解析得到最终的预测结果。

1. **非稀疏矩阵形式-----推荐**

为了避免在测试的时候还要做稀疏矩阵的解析，则可以将解析过程放在预测的过程中(即Tensorflow空间):

dense\_predictions = tf.sparse\_to\_dense (decode\_logits[0].indices,

[tf.shape(inputs,out\_type =tf.int32)[0], input\_width],

decode\_logits[0].values, default\_value=-1,

name='dense\_predictions')

即可在tensorflow空间内将decode\_logits由稀疏矩阵形式得到密集矩阵形式，再sess.run得到dense\_predictions的值就已是该批测试样本的预测标签了eg.[“123112”,”837463”...]

**OpenVino下的CTC 训练和测试:**

**训练：**

因为OpenVino只支持一个placeholder作为输入,而由以上所说，为了训练CTC，共有三个输入，那么除了必须的图片数据输入inputs ，其他两个sparse\_groundtrouth和seq\_len\_placeholder 是否可以消除或者隐藏呢？事实是不仅sparse\_groundtrouth和seq\_len\_placeholder可以隐藏，就连inputs都可以隐藏。这一点我们通过tensorflo的数据读取机制就可以做到,我们不使用placeholder作为输入，而是使用tf.train.shuffle\_batch作为数据读取器每次读取inputs和未稀疏化的标签labels,至于seq\_len\_placeholder，经过分析，其就是一个列表，通过seq\_len\_placeholder = tf.fill([tf.shape(train\_image\_batch)[0]], tf.shape(train\_image\_batch)[2])即可达到效果，而对于sparse\_groundtrouth，其是labels的稀疏化形式，通过sparse\_gt = tf.py\_func(get\_sparse\_labels, [labels], [tf.int64, tf.int64, tf.int64])即可得到。(在我理解下:tf.py\_func可以在TensorFlow图形的中间运行任意的Python代码，即打通了tensorflow空间和numpy空间)

**测试:**

至此整个模型中没有了placeholder,训练出来的模型也是不能使用的，因为必须要有且仅有一个placeholder,这时我们即可通过重新读取模型加载参数，但是对于inputs通过placeholder得到，具体见freezing\_graph.py。(相当于使用原来的模型参数和模型大致架构，但是使用新的placeholder进行inference)这里有个好处，就是比如原来的模型中is\_training=True,这里可以将is\_training变为False,这是很有帮助的。(原来训练MobileNetV1时有slim.batch\_norm,,如果在测试的时候不将is\_training置为Flase,那么预测时会产生极大的精度差异，而且在转化成openVino模型时也会出问题)

**其他细节:**

1. 通过 tf.decode\_csv从txt文件中读取数据时,是字节类型的,必须将label给decode成字符串类型，不然标签读取错误，后面训练的时候编辑距离最后会一直稳定在1.000。
2. Sess.run(tf.global\_variables\_initializer())放在saver.restore之前,这样才能加载之前的模型接着训练

3.通过tf.train.shuffle\_batch读取数据并训练时，是不好在训练过程中进行精度计算的，因为通过train\_image\_batch\_,train\_label\_batch\_,dense\_predictions\_ = sess.run([train\_image\_batch,train\_label\_batch, dense\_predictions])得到的train\_image\_batch\_其实和计算dense\_predictions\_使用的不是同一批数据。