



csdn 第九周作业文档

崔萌

csdn 第九周作业文档	1
崔萌	1
作业要求:	4
文档内容	5
关于 tfrecord 的制作:	5
关于 tinymind 环境的配置:	6
关于 8x 的代码补全	7
关于结果的对比:	9
心得体会	10

作业要求：

数据集准备完成-20 分：

数据集中应包含 train 和 val 两个 tfrecord 文件，大小在 400MB 左右

模型训练完成-20 分：

在 tinymind 运行 log 的输出中，可以看到如下内容：

```
2018-01-04 11:11:20,088 - DEBUG - train.py:298 - step 1200 Current Loss: 101.153938293
```

```
2018-01-04 11:11:20,088 - DEBUG - train.py:300 - [23.54] imgs/s
```

```
2018-01-04 11:11:21,011 - DEBUG - train.py:307 - Model saved in file: ./out/train/model.ckpt-1200
```

```
2018-01-04 11:11:21,018 - DEBUG - train.py:314 - validation generated at step [1200]
```

```
2018-01-04 11:11:28,461 - DEBUG - train.py:298 - step 1210 Current Loss: 116.911231995
```

```
2018-01-04 11:11:28,461 - DEBUG - train.py:300 - [19.11] imgs/s
```

```
2018-01-04 11:11:35,356 - DEBUG - train.py:298 - step 1220 Current Loss: 90.7060165405
```

训练结果完成-20 分：

训练完成之后，可以在**/output/eval 下面生成验证的图片，其中 val_xx_prediction.jpg** 的图片为模型输出的预测结果，内容应可以对应相应的 annotation 和 img。根据验证图片的内容，结果可能会有区别，但是肯定可以看到输出的结果是明显有意义的。

模型代码补全 -20 分：

train.py 中可以看到 8x 代码的实现。形式可能会有区别，但是有比较明显的三个上采样过程，两个 2X，一个 8X，及其结果的融合。

心得体会-20 分

文档内容

关于 tfrecord 的制作:

补全 convert_fcn_dataset.py 的代码内容。

补全内容如下:

在函数 dict_to_tf_example 中:

Your code here, fill the dict

```
feature_dict = {
    'image/height': dataset_util.int64_feature(height),
    'image/width': dataset_util.int64_feature(width),
    'image/filename': dataset_util.bytes_feature(
        data.encode('utf8')),
    'image/encoded': dataset_util.bytes_feature(encoded_data),
    'image/label': dataset_util.int64_list_feature(encoded_label),
    'image/format': dataset_util.bytes_feature('jpeg'.encode('utf8')),
}
```





在 create_tf_record 中:

Your code here

```
print(output_filename)
with contextlib2.ExitStack() as tf_record_close_stack:
    writer = tf.python_io.TFRecordWriter(output_filename)
    for data,label in file_pars:
        #print(data)
        #print(label)
        example = dict_to_tf_example(data,label)
        if example == None:
            print(data + " is null!")
        else:
            writer.write(example.SerializeToString())
```

具体代码可详见码云

运行后生成两个文件, 如下所示:

 convert_fcn_dataset.py	2018/9/20 11:02	PY 文件	4 KB
 dataset.py	2018/3/18 22:34	PY 文件	4 KB
 fcn_train.record	2018/9/20 11:03	RECORD 文件	413,108 KB
 fcn_val.record	2018/9/20 11:04	RECORD 文件	409,602 KB

关于 tinymind 环境的配置：

上传下面 4 个代码到 tinymind

dataset.py	2018/9/21 15:29	PY 文件	4 KB
train.py	2018/9/21 15:30	PY 文件	14 KB
utils.py	2018/9/21 15:30	PY 文件	3 KB
vgg.py	2018/9/21 15:34	PY 文件	15 KB

由于 vgg16 的模型试过各种办法，下载的实在太慢了。。所以使用了课程提供的数据，参数配置如下：

learning_rate	batch_size	output_dir	checkpoint_path
0.0001	16	/output	/data/ai100/quiz-w9/vgg_16.ckpt

dataset_train	dataset_val
/data/ai100/quiz-w9/fcn_train.record	/data/ai100/quiz-w9/fcn_val.record

max_steps

3000

运行结果：

8x 的运行链接：<https://www.tinymind.com/executions/jynrk89s>

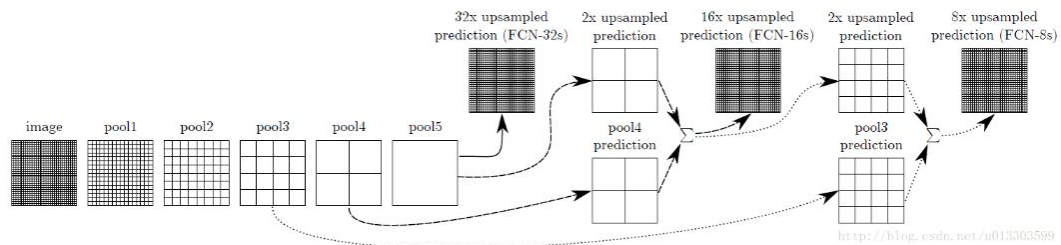
16x 的运行链接：<https://www.tinymind.com/executions/oz27vaja>

8x 的运行结果截图：

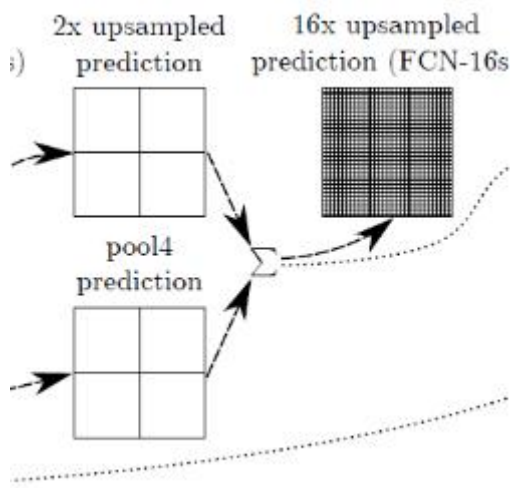
```
2018-09-25 10:34:39,971 - DEBUG - train.py:329 - step 4860 Current Loss: 41.21818923950195
2018-09-25 10:34:39,972 - DEBUG - train.py:331 - [9.61] imgs/s
2018-09-25 10:34:56,622 - DEBUG - train.py:329 - step 4870 Current Loss: 40.7941780090332
2018-09-25 10:34:56,623 - DEBUG - train.py:331 - [9.61] imgs/s
2018-09-25 10:35:12,914 - DEBUG - train.py:329 - step 4880 Current Loss: 102.94287109375
2018-09-25 10:35:12,914 - DEBUG - train.py:331 - [9.82] imgs/s
2018-09-25 10:35:29,569 - DEBUG - train.py:329 - step 4890 Current Loss: 65.7442855834961
2018-09-25 10:35:29,569 - DEBUG - train.py:331 - [9.61] imgs/s
2018-09-25 10:35:46,265 - DEBUG - train.py:329 - step 4900 Current Loss: 32.87675857543945
2018-09-25 10:35:46,265 - DEBUG - train.py:331 - [9.58] imgs/s
2018-09-25 10:35:48,230 - DEBUG - train.py:338 - Model saved in file: /output/train/model.ckpt-4900
2018-09-25 10:36:14,767 - DEBUG - train.py:361 - Model saved in file: /output/train/model.ckpt-4900
```

关于 8x 的代码补全

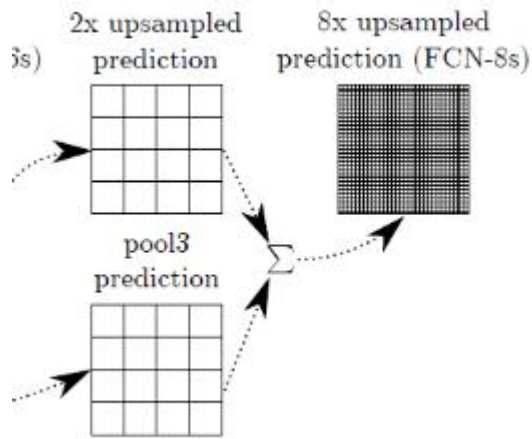
整体流程参照这个图示：



```
# Perform the upsampling
```

[illegible]

#-----my code-----



<http://blog.csdn.net/u013303599>

#获得 pool3 的预测结果

```
pool3_feature = end_points['vgg_16/pool3']
```

```
with tf.variable_scope('vgg_16/fc8'):
```

```
    aux_logits_8s = slim.conv2d(pool3_feature, number_of_classes, [1, 1],
                                activation_fn=None,
                                weights_initializer=tf.zeros_initializer,
                                scope='conv_pool3')
```

```
upsample_filter_tensor_x21 = tf.Variable(upsample_filter_np_x2, name='vgg_16/fc8/t_conv_x21')
```

```
upsampled_logits = tf.nn.conv2d_transpose(upsampled_logits, upsample_filter_tensor_x21,
                                           output_shape=tf.shape(aux_logits_8s),
                                           strides=[1, 2, 2, 1],
                                           padding='SAME')
```

#加和上一个的 2x 上采样结果

#sum

```
upsampled_logits = upsampled_logits + aux_logits_8s
```

#8x 的上采样，获得最终的 logits

#8x upsample

```
upsample_factor_8 = 8
```


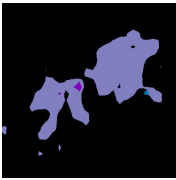

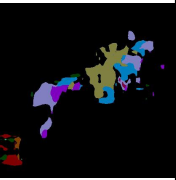



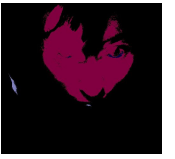

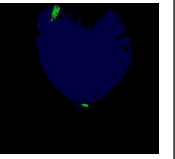










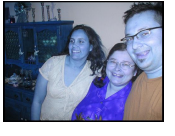




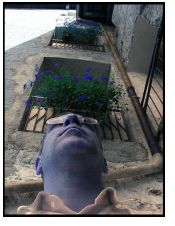
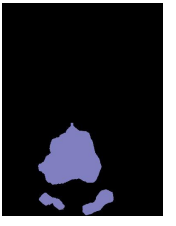
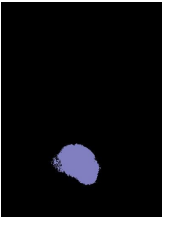
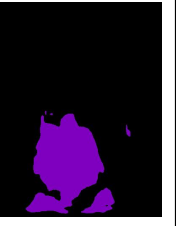
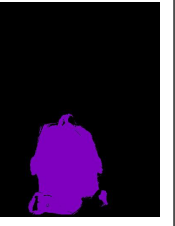
```
upsample_filter_np_x8 = bilinear_upsample_weights(upsample_factor_8,
                                                    number_of_classes)
```

```
upsample_filter_tensor_x8 = tf.Variable(upsample_filter_np_x8, name='vgg_16/fc8/t_conv_x8')
```

```
upsampled_logits = tf.nn.conv2d_transpose(upsampled_logits, upsample_filter_tensor_x8,
                                           output_shape=upsampled_logits_shape,
                                           strides=[1, upsample_factor_8, upsample_factor_8, 1],
                                           padding='SAME')
```

#-----my code-----

关于结果的对比：

Step	segmented	16x-pred	16x-pred-crf	8x-pred	8x-pred-crf
200					
400					
1000					
1400					
1600					
2600					

从中可以观察到，随着 step 的上升，8x 比 16x 关于物品的分类更加准确，边缘更加准确。

论文中也提到，对于 pool2 和 pool1 的上采样也是可以计算的，但是对于结果的提升帮助不大，这次作业中也没有尝试。

心得体会

通过完成这次作业，对 slim 框架的使用更加的熟练，更加强化了对预训练模型的使用方法，包括对预训练模型的结构修改，其中节点结果的获取等方法。

Tfrecord 的制作还是处在入门的位置吧，毕竟很多代码是作业中自带的。Slim 的使用也存在这样的问题，使用的更加熟练了，但是还没有信心能自己从零实现 fcn。

Crf 的理解上还不够透彻，baidu 后得知是随机场算法，在结果的显示帮助上表现为获得更主要的分类结果，过滤次要分类结果，不知道理解的到位不到位。

对于 fcn 的理解个人感觉还是十分到位的，毕竟 fcn 相比于 vgg，只是改变了一小部分，达到了更大的功能范畴。但是意义确是重大的。

最后还是谢谢老师！

——崔萌