



# 导入ImageNet图像分类模型



浙江大學城市學院  
ZHEJIANG UNIVERSITY CITY COLLEGE

- 选择预训练好的ImageNet图像识别模型 ➡ 经典网络介绍
- 导入模型 ➡ 模型的加载



## 模型的加载



Tensorflow提供了以下两种方式来存储和加载模型：

- 生成检查点文件 (checkpoint file) , 扩展名一般为.ckpt, 通过在tf.train.Saver对象上调用Saver.save()生成, 通过saver.restore()来加载。
- 生成图协议文件 (graph proto file) , 这是一个二进制文件, 扩展名一般为.bp, 用tf.train.write\_graph()保存, 然后使用tf.import\_graph\_def()来加载图。



## 图模型的加载



```
import tensorflow as tf  
#from tensorflow.python.platform import gfile
```

```
# 图的保存  
v = tf.Variable(1.0, name='my_variable')  
with tf.Session() as sess:  
    tf.train.write_graph(sess.graph_def, './tfmodel', 'test_pb.pb', as_text=False)
```

```
# 图的加载  
with tf.Session() as sess:  
    with tf.gfile.GFile('./tfmodel/test_pb.pb', 'rb') as f:  
        graph_def = tf.GraphDef()  
        graph_def.ParseFromString(f.read())  
        sess.graph.as_default()  
        tf.import_graph_def(graph_def, name='tf.graph')  
        print(graph_def)
```



# 导入ImageNet图像分类模型



浙江大學城市學院  
ZHEJIANG UNIVERSITY CITY COLLEGE

- 选择预训练好的ImageNet图像识别模型 ➡ Inception
- 导入模型 ➡ 通过图文件



# 导入Inception模型



浙江大學城市學院  
ZHEJIANG UNIVERSITY CITY COLLEGE

## 导入库

```
# coding: utf-8  
from __future__ import print_function  
import os  
from io import BytesIO  
import numpy as np  
from functools import partial  
import PIL.Image  
import scipy.misc  
import tensorflow as tf
```

## 创建图和会话

```
graph = tf.Graph()  
sess = tf.InteractiveSession(graph=graph)
```



## 导入Inception模型



```
model_fn = 'tensorflow_inception_graph.pb' #导入Inception网络  
# tensorflow_inception_graph.pb文件的下载 :  
# https://storage.googleapis.com/download.tensorflow.org/models/inception5h.zip
```

```
with tf.gfile.GFile(model_fn, 'rb') as f:  
    graph_def = tf.GraphDef()  
    graph_def.ParseFromString(f.read())
```

```
# 定义输入图像的占位符  
t_input = tf.placeholder(np.float32, name='input')  
  
#图像预处理——减均值  
imagenet_mean = 117.0 #在训练Inception模型时做了减均值预处理，此处也需减同样的均值以保持一  
  
#图像预处理——增加维度  
# 图像数据格式一般是 ( height , width , channels ) ，为同时将多张图片输入网络而在前面增加一维  
# 变为 ( batch,height,width,channel )  
t_preprocessed = tf.expand_dims(t_input - imagenet_mean, 0)
```

```
# 导入模型并将经预处理的图像送入网络中  
tf.import_graph_def(graph_def, {'input': t_preprocessed})
```



## 图像预处理——增加维度



- 使用的图像数据格式通常是 (height,width,channel) , 只能表示一张图像;
- 而Inception模型要求的输入格式却是 (batch, height, width, channel) , 即同时将多张图像送入网络

**tf.expand\_dims(input, dim, name=None)**

**Returns:**

A Tensor. Has the same type as input. Contains the same data as input, but its shape has an additional dimension of size 1 added.

向tensor中插入维度1, 插入位置就是参数代表的位置 (维度从0开始)



## 图像预处理——增加维度



`tf.expand_dims(input, dim, name=None)`

```
t = [[2,3,3],[1,5,5]]  
print("t_shape:",np.shape(t))
```

# 输出 (2, 3)

```
t1 = tf.expand_dims(t, 0)  
print("t1_shape:",np.shape(t1))
```

# 输出 (1, 2, 3)

```
t2 = tf.expand_dims(t, 1)  
print("t2_shape:",np.shape(t2))
```

# 输出 (2, 1, 3)

```
t3 = tf.expand_dims(t, 2)  
print("t3_shape:",np.shape(t3))
```

# 输出 (2, 3, 1)

```
t4 = tf.expand_dims(t, -1)  
print("t3_shape:",np.shape(t4))
```

# 输出 (2, 3, 1)





## 找到卷积层



```
layers = [op.name for op in graph.get_operations() if op.type == 'Conv2D' and 'import/' in op.name]
```

# 输出卷积层层数

```
print('Number of layers', len(layers))
```

# 输出所有卷积层名称

```
print(layers)
```

# 还可输出指定卷积层的参数

```
name = 'mixed4d_3x3_bottleneck_pre_relu'
```

```
print('shape of %s: %s' % (name1, str(graph.get_tensor_by_name('import/' + name1 + ':0').get_shape())))
```

```
#name2 = 'mixed4e_5x5_bottleneck_pre_relu'
```

```
#print('shape of %s: %s' % (name2, str(graph.get_tensor_by_name('import/' + name2 + ':0').get_shape())))
```



## 图的基本操作



- 建立图、获得默认图、重置默认图  
( `tf.Graph()`, `tf.get_default_graph()`, `tf.reset_default_graph()` )
- 获取张量
- 获取节点操作



## 建立图、获得图、重置图



```
In [1]: import numpy as np  
import tensorflow as tf
```

```
g = tf.Graph() # 创建新的图  
with g.as_default():  
    c1 = tf.constant(0.0) # 在新图中添加变量  
    print("c1.graph : ", c1.graph) # 可通过变量的'.graph'可获得其所在的图
```

```
c1.graph : <tensorflow.python.framework.ops.Graph object at 0x0000000004EA8DD8>
```

```
In [2]: tf.reset_default_graph() # 重置默认图  
g2 = tf.get_default_graph() # 获得默认图  
print("g2 : ", g2)
```

```
g2 : <tensorflow.python.framework.ops.Graph object at 0x0000000004EAFB00>
```



## 获取张量



```
In [3]: #先获取张量的名字  
print(c1.name)
```

Const:0

```
In [4]: # 然后将张量名字放到tf.Graph().get_tensor_by_name(name = "")中  
t = g.get_tensor_by_name(name = "Const:0")  
# 通过打印t验证get_tensor_by_name所获取的张量就是前面定义的张量c1  
print(t)
```

Tensor("Const:0", shape=(), dtype=float32)

- 不必花太多精力去关注TensorFlow中默认的命名规则。一般在需要使用名字时，都会在定义的同时为它指定好固定的名字。
- 如果真的不清楚某个元素的名字，可以将其打印出来，回填到代码中，再次运行即可。



## 获取节点操作

```
In [5]: a = tf.constant([[1.0, 2.0]])  
        b = tf.constant([[1.0], [3.0]])  
  
        tensor1 = tf.matmul(a, b, name='example_op')  
        print(tensor1)  
        print(tensor1.name)  
  
Tensor("example_op:0", shape=(1, 1), dtype=float32)  
example_op:0
```

```
In [6]: #先将op的名字打印出来  
        print(tensor1.op.name)  
  
example_op
```

```
In [7]: #然后使用get_operation_by_name函数  
        test_op = g2.get_operation_by_name("example_op")  
        print(test_op)
```

```
name: "example_op"  
op: "MatMul"  
input: "Const"  
input: "Const_1"  
attr {  
  key: "T"  
  value {  
    type: DT_FLOAT  
  }  
}  
attr {  
  key: "transpose_a"  
  value {  
    b: false  
  }  
}  
attr {  
  key: "transpose_b"  
  value {  
    b: false  
  }  
}
```

- 注意：上例中的`tensor1 = tf.matmul(a, b, name='example_op')`并不是OP，而是张量。
- OP其实是描述张量中的运算关系，是通过访问张量的属性找到的。