# 基于FCN模型的卫星图像分割

# 10.1 实验目的

1.       初步了解FCN检测框架的实现原理；

2.       初步了解语义分割相关知识；

# 10.2 实验要求

本次试验后，要求学生能：

1.       对FCN的实现原理有初步的了解；

2.       跟着教程实现本实验的语义分割任务；

# 10.3 案例讲解（SiftFlow数据集）

### **10.3.1****SiftFlow数据集**

本次试验使用的使用SiftFlow数据集，数据集共有2688张图像，每张图均为256\*256的彩色数据集。SiftFlow数据集示意图如下：



图10-1 SiftFlow数据集示意图

SiftFlow数据集下共有3个文件夹：Images、GeoLabels、SemanticLabels。

**Images** -- 保存图片数据

**semanticlabels** -- 保存语义分割标签,语义分割标签，除默认的0外，共有33类，类别如下：

01 awning、02 balcony、03 bird、04 boat、05 bridge、06 building、07 bus、08 car、09 cow、10 crosswalk、11 desert、12 door、13 fence、14 field、15 grass、16 moon、17 mountain、18 person、19 plant、20 pole、21 river、22 road、23 rock、24 sand、25 sea、26 sidewalk、27 sign、28 sky、29 staircase、30 streetlight、31 sun、32 tree、33 window。

**geolabels** -- 保存场景识别标签，一共4类，除去标签-1，就是3类。

**所以是分别训练了两个网络，网络的前七层一样。**

10.3.2 FCN（Fully Convolutional networks）

针对卫星图像分割我们选用全卷积网络（FCN）用于模型训练，参考了2015年的CVPR论文《Fully Convolutional Networks for Semantic Segmentation》。

### （1）从图像分类到语义分割

近些年，CNN的发展不断驱动着图像领域的进步，图像分类、关键点检测、物体检测等任务都取得了较好的进展。不同于以上领域，图像的语义分割是一种像素级别上的分类任务，需要对图像中每个像素点进行分类，语义分割效果如图10-1所示，其中一种颜色代表一个类别。



图10-1 语义分割效果图

优势：传统基于CNN的语义分割方法为了对单个像素分类，通常需要使用周围的一块区域作为网络的输入进行训练或预测。这种方法存在一些缺点，若对整张图像的所有像素进行分类，滑窗（即单个像素对应的周围区域）造成的存储开销较大，且相邻像素对应的滑窗之间重叠区域大，重复计算多，效率低下。此外像素对应的输入图块通常较小，只能提取局部特征，其分类效果也会有所限制。针对这些问题，加州大学伯克利大学的Jonathan Long等人提出了全卷积网络（FCN）的概念，该网络试图从抽象的特征中恢复出每个像素对应的类别，实现图像“end-to-end”的语义分割。

### （2）从CNN到FCN

CNN网络是深度学习过程中接触到最多，也是最常用的网络。经典的CNN流程如下：输入图像经过若干组卷积层、池化层提取图像特征，特征图再被全连接层映射成固定长度的特征向量。这种经典的CNN结构适用于图像级别的分类和回归任务，在网络最后通常会输出图像的类别和其对应的概率。

而FCN在CNN的基础上做了一些改变，它将传统CNN中的全连接层全部转化为卷积层，整个网络均靠网络连接，也因此叫做“全卷积网络”。FCN语义分割示意图如下：

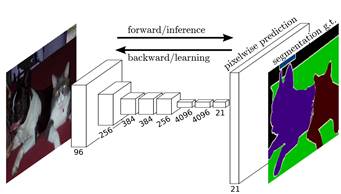


图10-2 FCN分类示意图

### （3）FCN的主要技术

FCN主要利用三种关键技术来优化网络性能，具体介绍如下：

A、卷积化（convolutionalization）：经典CNN架构通常在卷积层后接全连接层，输出每个分类对应的概率，使得提取的二维特征被全连接层拉伸为一维，从而丢失了图片的空间信息。而FCN中，CNN的全连接层均被替换成卷积层，避免了空间信息的损失。网络全连接层卷积化示意如图所示：

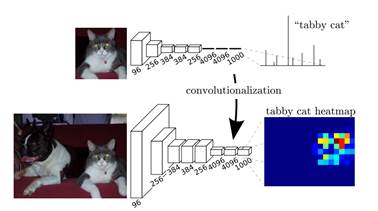


图10-3全连接层卷积化示意图

#### B、上采样（Upsampling）：FCN最后输出的是一张分割图。在CNN结构中通常会使用池化操作降低特征图维度，经过若干次卷积池化，特征图持续降低。为了在最后输出原图大小的分割图，作者在得到heatmap后加上了上采样操作，输入图像在进行了5次卷积池化之后，图像的大小依次缩小了2、4、8、16、32倍，于是在最后一层图像进行32倍的上采样即可得到原图的大小。这里的上采样实际是通过反卷积（deconvolution）实现的。

C、跳跃结构（Skip Architecture）：卷积化加上上采样，基本能在原图大小上实现像素级的语义分割，但此时的分割结果较为粗糙，采用跳跃结构用于优化分割的结果。由于只对第5层结果采用上采样结果不够精细，作者将第4层和第3层的输出也进行反卷积操作，分别使用了16和8倍的上采样。“Skip Architecture”网络结构和对应的采样效果如下图所示。

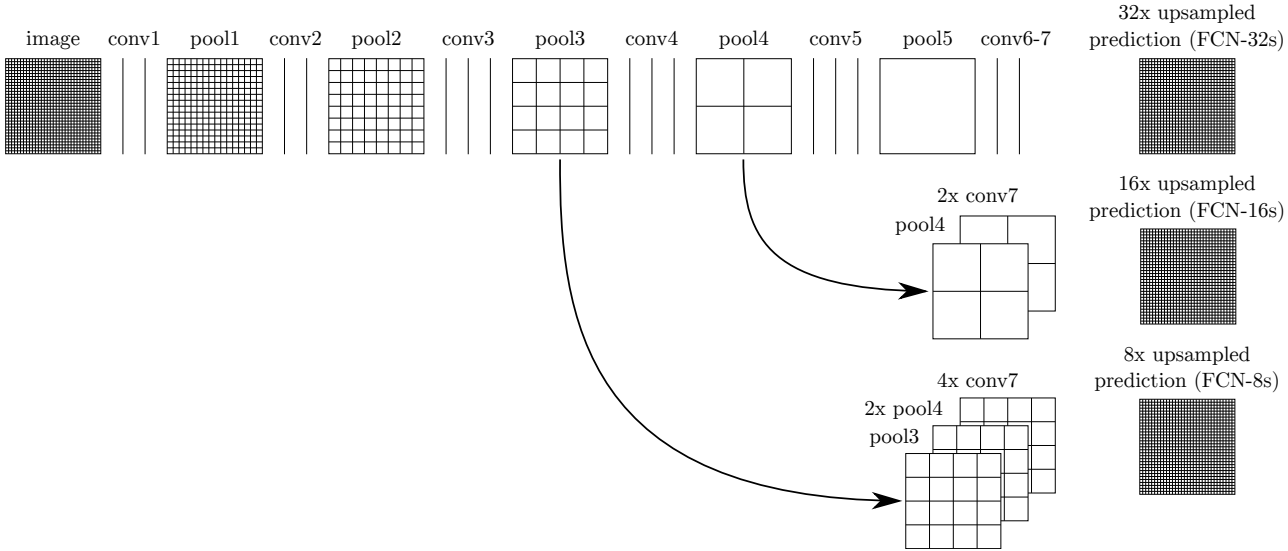
-

图10-4 “Skip Architecture”示意图

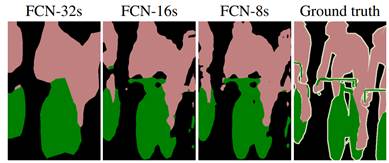


图10-5 不同倍数上采样效果图

# 10.4 实验步骤

此实验可使用Caffe镜像。

### **10.4.1获取项目源码**

从平台“相关下载”模块，获取源码，解压后可获取一个code文件夹，共四个文件：fcn.berkeleyvision.org.zip、vgg16-fcn.caffemodel、VGG\_ILSVRC\_16\_layers\_deploy.prototxt、deploy.prototxt。其中fcn.berkeleyvision.org.zip为本实验的源码框架。其余三个文件为实验中需要的其他资源，我们在后续的步骤中会介绍并使用。

将fcn.berkeleyvision.org.zip文件解压，解压方法如下：

# unzip fcn.berkeleyvision.org.zip

### **10.4.2数据集获取**

从平台“相关下载”模块获取，数据集文件，解压后可获取一个data文件夹，文件夹有两个文件，训练集SiftFlowDataset.zip和一张测试图片test.png。将SiftFlowDataset.zip放置于fcn.berkeleyvision.org/data/siftflow，目录下并解压，命令如下：

# unzip SiftFlowDataset.zip

### **10.4.3依赖安装**

# pip install scikit-image -i https://pypi.tuna.tsinghua.edu.cn/simple/

# pip install matplotlib -i https://pypi.mirrors.ustc.edu.cn/simple/

# pip install protobuf -i https://pypi.mirrors.ustc.edu.cn/simple/

# yum install tkinter

# pip install opencv\_python -i https://pypi.mirrors.ustc.edu.cn/simple/

# yum install libSM.x86\_64

### **10.4.4获取预训练模型和网络描述文件**

FCN模型训练使用了预训练模型，预训练模型包含在源码文件夹code中，从code文件夹下获取vgg16-fcn.caffemodel和网络描述文件VGG\_ILSVRC\_16\_layers\_deploy.prototxt，将其一起放置到fcn.berkeleyvision.org/ilsvrc-nets目录下。

### **10.4.5 训练模型**

### （1）关联Caffe和FCN

使用Caffe的python接口，需要编译pycaffe，进入Caffe目录下，打开Makefile.config，确认“WITH\_PYTHON\_LAYER := 1”的注释已经被取消，然后编译pycaffe，命令如下：

# /root/cDeep/Caffe/caffe-master

# make pycaffe

由于Caffe和FCN文件夹相互独立，所以需要将Caffe的Python路径添加到设为python路径。

# vim ~/.bashrc

加上export PYTHONPATH=/root/cDeep/Caffe/caffe-master/python

使配置文件生效

# source ~/.bashrc

### （2）进入fcn.berkeleyvision.org目录下，将需要的文件软链到相关文件夹

# cd siftflow-fcn32s

# ln -s ../surgery.py .

# ln -s ../score.py .

# ln -s ../siftflow\_layers.py .

新建文件夹，用于放置模型

# mkdir models

### （3）修改训练超参文件

# vim solver.prototxt

在其中添加snapshot（迭代多少次保存一次）和snapshot\_prefix（保存的路径），其中snapshot和snapshot\_prefix的参数并不固定，根据实际情况可以适当调整。修改后内容如下：

train\_net: "trainval.prototxt"

test\_net: "test.prototxt"

test\_iter: 200

# make test net, but don't invoke it from the solver itself

test\_interval: 999999999

display: 20

average\_loss: 20

lr\_policy: "fixed"

# lr for unnormalized softmax

base\_lr: 1e-10

# high momentum

momentum: 0.99

# no gradient accumulation

iter\_size: 1

max\_iter: 300000

weight\_decay: 0.0005

snapshot:5000  #迭代5000次保存一次模型

snapshot\_prefix:"models/fcn" #模型保存路径，需根据实际修改

test\_initialization: false

### （4）修改训练文件

# vim solve.py

由于训练的是fcn32网络，是使用迁移学习，在预训练模型的基础上训练的，模型训练的时候需要获取预训练模型卷积层的参数，所以需要修改solver.py中的相关内容，使其可以复用vgg16-fcn.caffemodel中的参数。

vgg16的权重使用transplant的方式获取，它和原本的copy\_from的区别在于拷贝参数的同时，会改造原本的网络。

并且caffe.set\_device(int(sys.argv[1]))中，需改为实际的gpu设备号，机器上的可获取的gpu设备号可以使用nvidia-smi查询，由于平台的每个账号默认只有一个GPU，因此这里只能使用0号GPU，即改为caffe.set\_device(int(0))。

最终修改如下：

#coding:utf-8

import caffe

import surgery, score

import numpy as np

import os

import sys

try:

    import setproctitle

    setproctitle.setproctitle(os.path.basename(os.getcwd()))

except:

    pass

vgg\_weights = '../ilsvrc-nets/vgg16-fcn.caffemodel'                 #预训练模型地址

vgg\_proto = '../ilsvrc-nets/VGG\_ILSVRC\_16\_layers\_deploy.prototxt'  #网络描述文件地址

#weights = '../ilsvrc-nets/vgg16-fcn.caffemodel'

# init

#caffe.set\_device(int(sys.argv[1]))  #此行注释掉

caffe.set\_device(int(0))      #修改为实际的GPU设备号，如果只有一个GPU，则设置为0

caffe.set\_mode\_gpu()

solver = caffe.SGDSolver('solver.prototxt')

vgg\_net=caffe.Net(vgg\_proto,vgg\_weights,caffe.TRAIN)  #用transplant的方式获取vgg16的权重

surgery.transplant(solver.net,vgg\_net)                  #

del vgg\_net                                       #

#solver.net.copy\_from(weights)                       #注释copy\_from方法

# surgeries

interp\_layers = [k for k in solver.net.params.keys() if 'up' in k]

surgery.interp(solver.net, interp\_layers)

# scoring

test = np.loadtxt('../data/siftflow/test.txt', dtype=str)

for \_ in range(50): #50\*2000=10w即solver.prototxt中maxiter最大迭代次数，在solver中设置没有用需要在这边设置

    solver.step(2000) #2000次时执行score.py中的begin seg测试获取loss、accuracy、mean等

    # N.B. metrics on the semantic labels are off b.c. of missing classes;

    # score manually from the histogram instead for proper evaluation

    score.seg\_tests(solver, False, test, layer='score\_sem', gt='sem')

    score.seg\_tests(solver, False, test, layer='score\_geo', gt='geo')

（5）训练模型

若直接训练：

# python solve.py

若后台训练，实时跟踪日志，方法如下：

# nohup python solve.py > train\_fcn\_sift32.log 2>&1 &

# tailf train\_fcn\_sift32.log

### **10.4.6 测试模型**

### （1）准备测试网络文件

测试网络描述文件deploy.prototxt，可从code文件夹下获取，获取后置于siftflow-fcn32s下。

### （2）修改测试文件

测试文件为主目录下的predictTest.py文件，其修改如下。其中Image.open对应测试图片路径，原始代码中的测试图片为Pascal VOC数据集的测试图片，由于我们使用的数据集并不是Pascal VOC，所以在Pascal VOC类型的数据上的测试效果并不好，所以这里的测试图片的路径改为SiftFlow数据集风格的图片。

net = caffe.Net后参数对应测试网络和模型文件所在位置，需要对应我们实际想测试的模型进行调整。下面的实例修改中，测试的模型为迭代了90000次的模型fcn\_iter\_90000.caffemodel。

此外由于代码停止维护的问题，需要稍作修改，将“out = net.blobs['score'].data[0].argmax(axis=0)”中的score修改为score\_sem。

#coding:utf-8

import numpy as np

from PIL import Image

import matplotlib           #增加画图需要import的包

matplotlib.use('Agg')         #

import matplotlib.pyplot as plt  #

import caffe

import cv2

#%matplotlib inline

# load image, switch to BGR, subtract mean, and make dims C x H x W for Caffe

**im = Image.open('****data/sift-flow/Images/spatial\_envelope\_256x256\_static\_8outdoorcategories/insidecity\_art1167.jpg')  #测试图片**

in\_ = np.array(im, dtype=np.float32)

in\_ = in\_[:,:,::-1]

in\_ -= np.array((104.00698793,116.66876762,122.67891434))

in\_ = in\_.transpose((2,0,1))

# load net

**net = caffe.Net('****siftflow-fcn32s/deploy.prototxt', '****siftflow-fcn32s/models/****fcn\_iter\_90000.caffemodel', caffe.TEST)#网络描述文件和模型路径**

# shape for input (data blob is N x C x H x W), set data

net.blobs['data'].reshape(1, \*in\_.shape)

net.blobs['data'].data[...] = in\_

# run net and take argmax for prediction

net.forward()

**out = net.blobs['****score\_sem'].data[0].argmax(axis=0)               #修改score为score\_sem**

#print "hello,python!"

plt.imshow(out)        #将生成的图片保存在当前目录下，为test\_result.png

plt.axis('off')               #

plt.savefig('test\_result.png')      #

修改后执行如下命令，结果在当前目录下，为test\_result.png。

# python predictTest.py

# 10.5 实验结果

测试模型时选用仅迭代90000次的模型，可以看出模型还是不够精细，若需要效果好，还需继续训练。此外，测试数据选用了sift-flow数据集中的图片，因为模型在sift-flow数据集上训练，若使用其他风格的图片，其测试效果会有所欠佳，测试图片和训练集风格接近，效果更好。若想测试其他风格的图片，请采用相似风格的数据集进行训练。

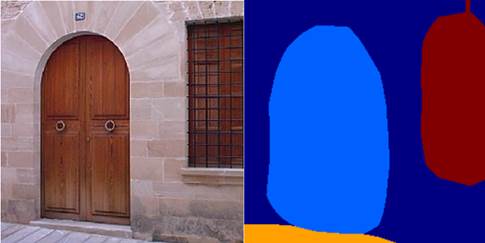


图10-6分割效果示意图

# 10.6 数据集制作

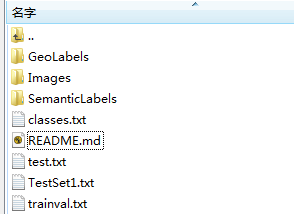
### **10.6.1 siftflow数据集内部结构**

siftflow是一共语义分割数据集，有两个标记：

1.semantic classes：语义类，如猫、狗

2.geometric classes：几何类，如水平、垂直和天空

参考classes.txt列出模型输出顺序中的类，参考../siftflow\_layers.py文件用于对这个数据集的python数据层



注意，数据集有许多问题，包括未注释的图像和测试集中缺少的类。所提供的分割排除未注释的图像。如文中所指出的，必须注意排除丢失的类进行适当的评估。

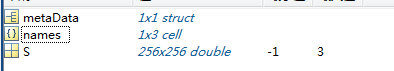
*SemanticLabels：*每个.mat打开是1\*33



33类：

'awning' 'balcony' 'bird' 'boat' 'bridge' 'building' 'bus' 'car' 'cow' 'crosswalk' 'desert' 'door' 'fence' 'field' 'grass' 'moon' 'mountain' 'person' 'plant' 'pole' 'river' 'road' 'rock' 'sand' 'sea' 'sidewalk' 'sign' 'sky' 'staircase' 'streetlight' 'sun' 'tree' 'window'

*GeoLabels：*几何类，1\*3：'sky' 'horizontal' 'vertical'



类别映射关系：

SemanticLabels: 0 --- 'void' GeoLabels: -1 --- 'void'  
 1 --- 'awning' 1 --- 'sky'  
 2 --- 'balcony' 2 --- 'horizontal'  
 3 --- 'bird' 3 --- 'vertical'  
 4 --- 'boat'   
 5 --- 'bridge'  
 6 --- 'building'  
 7 --- 'bus'  
 8 --- 'car'  
 9 --- 'cow'   
0. void ---> -1. void  
1. 'awning' ---> 3. 'vertical'  
2. 'balcony' ---> 3. 'vertical'  
3. 'bird' ---> 3. 'vertical'  
4. 'boat' ---> 3. 'vertical'  
5. 'bridge' ---> 3. 'vertical'  
6. 'building' ---> 3. 'vertical'  
7. 'bus' ---> 3. 'vertical'  
8. 'car' ---> 3. 'vertical'  
9. 'cow' ---> 3. 'vertical'  
10. 'crosswalk' ---> 2. 'horizontal'  
11. 'desert' ---> 2. 'horizontal'  
12. 'door' ---> 3. 'vertical'  
13. 'fence' ---> 3. 'vertical'  
14. 'field' ---> 2. 'horizontal'  
15. 'grass' ---> 2. 'horizontal'  
16. 'moo

### **10.6.2 仿照siftflow制作自己的数据集**

1.先由原图获取灰度类号将其映射进mat中S矩阵中，类别映射关系如下：

Semantic and geometric segmentation classes for scenes.Semantic: 0 is void and 1–10 are classes.

原图获取的类别 映射进 mat: 映射进 训练层

1 0 'void' 背景 0

2 1 'crops' 农作物 1

3 2 'trees' 树木 2

4 3 'roads' 道路 3

5 4 'buildings' 建筑物 4

6 5 'tracks' 铁路 5

7 6 'parking lot'停车场 6

8 7 'misc' 混杂建筑物 7

9 8 'hill' 丘陵 8

10 9 'standingwater'积水区 9

255 10 'waterways' 河流 10

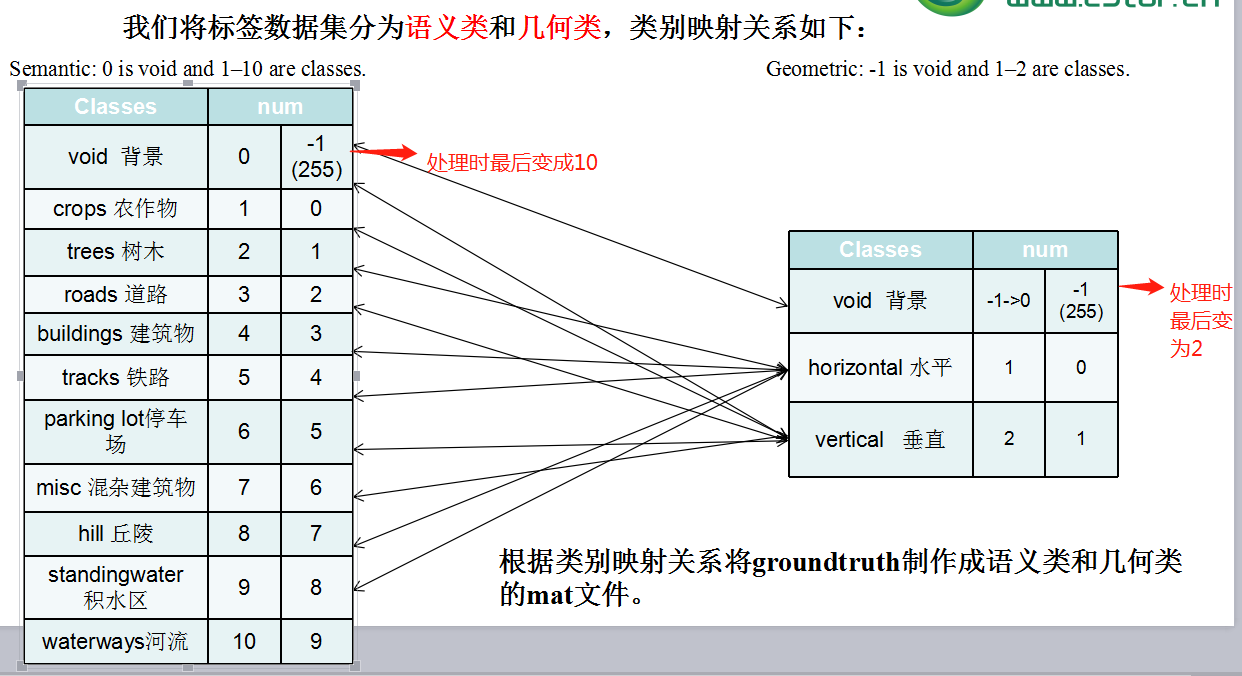
Geometric: -1 is void and 1–2 are classes.

自定义 -1 'void' 背景类 0

1 'horizontal' 水平类 1

2 'vertical' 垂直类 2

2.训练过程中在对map\_S处理：首先将几何类中背景序号-1变为0，在对几何类和语义类别所有都减去1。



*trainval.txt:*1-15000共15000张;*test.txt:*15001-18000共3000张

打乱test.txt与trainval.txt顺序：

shuf input\_file.txt -o output\_file.txt

### **10.6.3计算图片均值**

*create\_mean.py:*

#coding=utf-8  
import os  
import numpy as np  
from PIL import Image  
def createmean(img\_file):  
 R = 0  
 G = 0  
 B = 0  
 sum = len(img\_file)  
 for file in img\_file:  
 print "Calculating image:"  
 file = file.strip('\n') + ".tif"  
 img = Image.open("./Images/" + file) #获取trainval.txt中对应的图片  
 img\_array = np.array(img, dtype=np.float32) #将图片转为矩阵  
 img\_h = img\_array.shape[0] #获取图片长宽  
 img\_w = img\_array.shape[1]  
 r = 0  
 g = 0  
 b = 0  
 for i in range(img\_h):  
 for j in range(img\_w):  
 r += img\_array[i, j][0] #获取每张图片的RGB之和  
 g += img\_array[i, j][1]  
 b += img\_array[i, j][2]  
 R += r #将所有图片RGB分别求和  
 G += g  
 B += b  
 print(R, G, B)  
 R\_ = R / (sum \* img\_h \* img\_w) #平均后就是均值 /所有图片张数\*图片长\*宽  
 G\_ = G / (sum \* img\_h \* img\_w)  
 B\_ = B / (sum \* img\_h \* img\_w)  
 print "Results:", (R\_, G\_, B\_)  
if \_\_name\_\_ == '\_\_main\_\_':  
 with open("trainval.txt", "r") as f:  
 img\_file = f.readlines() #list:文件中所有行数，每行代表一张图片  
 createmean(img\_file)

运行结果：记得在predictTest.py和dstl\_layers.py中将R G B -->B G R。因为caffe是倒着读取。

(80.09322379964193, 90.55851244227091, 75.5052908510844)-->(75.5052908510844, 90.55851244227091, 80.09322379964193)

# 10.7调整模型并训练

总思路：

weights =dstl-fcn16s/100000.caffemodel

权重赋值给dstl\_fcn8s的模型

weights =dstl-fcn32s/100000.caffemodel

权重赋值给dstl\_fcn16s的模型

dstl\_fcn8s

dstl\_fcn16s

dstl\_fcn32s

训练：

调用py传参3个：

1.数据集路径dir

2.seed：1337

3.split：train和test列表

trainval.prototxt

dstl\_layers.py

solver.prototxt

训练

test.prototxt

score.py

solve.py

surgery.py

测试：

predictTest.py

deploy.prototxt

### **10.7.1 dstl\_fcn32s**

*solve.py:*

*solver.prototxt:*

*trainval.prototxt:prototxt中最后定义的c3和11个输出，对应与原图同样大小的11个图片，就是对11个类的预测结果，最后生成的图片就是各个像素位置取值最大的分类。如果忽略背景类写了10类，则预测时候不会输出背景类，背景类会被预测成其他的类别。*

*反卷积层生成了一个11类别的输出，把图片放大尺寸变为531\*531，添加crop层裁剪成与原图同样的大小*

#最后输出的卷积层

layer {

name: "score\_fr\_geo"

type: "Convolution"

bottom: "fc7"

top: "score\_fr\_geo"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

engine:CAFFE

num\_output: 2

pad: 0

kernel\_size: 1

}

}

#上采样层

layer {

name: "upscore\_geo"

type: "Deconvolution"

bottom: "score\_fr\_geo"

top: "upscore\_geo"

param {

lr\_mult: 0

}

convolution\_param {

engine:CAFFE

num\_output: 2 # 几何类别定义2类

bias\_term: false

kernel\_size: 64

stride: 32

}

}

#用反卷积层即上采样将图片放大，但是得到的结果与原图大小不一致，用crop层进行裁剪

layer {

name: "score\_geo"

type: "Crop"

bottom: "upscore\_geo"

bottom: "data"

top: "score\_geo"

crop\_param {

axis: 2 #对blob的第三个维度之后的维度进行裁剪，就是图片的宽高

offset: 19 #裁剪531-19=512

}

}

#loss层

layer {

name: "loss\_geo"

type: "SoftmaxWithLoss"

bottom: "score\_geo"

bottom: "geo"

top: "loss\_geo"

loss\_param {

ignore\_label:255 #忽略背景类别定义

normalize: false #若改为true，loss表示显示的小一点

}

}

*test.prototxt:*

*dstl\_layers.py:*

caffe中的数据按N\*C\*H\*W存储的:

N为batch size,top[0].reshape(1, \*self.data.shape)这里的1为batch size

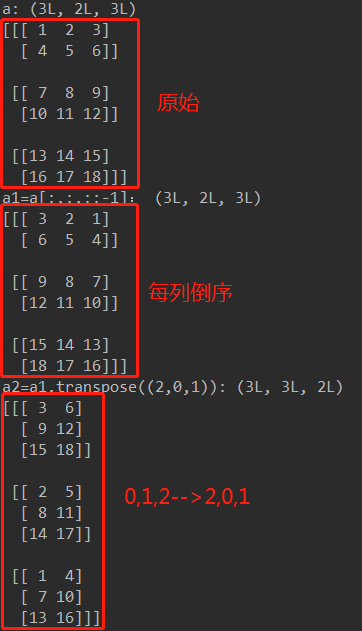
C为通道数RGB，in\_ = in\_[ : , : , : :-1]但是caffe读取要倒序读即为BGR，这里的-1即表示最后一维从后往前读取；

H和W分别为长和宽，分别是in\_ = in\_[ : , : , : :-1]中的第一维和第二维，最后需要转换位置in\_.transpose((2,0,1))，将通道数放在最前面

def test\_nparray():  
 a=np.array([  
 [[1,2,3],[4,5,6]],  
 [[7,8,9],[10,11,12]],  
 [[13,14,15],[16,17,18]]  
 ])  
 print "a:",a.shape #3L,2L,3L：表示三维，每维度都是2\*3列,L代表无符号整型，就是0-255的数  
 print a  
 a1=a[:,:,::-1] #[ , , ]对应3L,2L,3L：最后一个表示对每个维度中的2\*3的3进行操作倒序

print "a1=a[:,:,::-1]：",a1.shape  
 print a1  
 a2=a1.transpose((2,0,1))  
 print "a2=a1.transpose((2,0,1)):",a2.shape  
 print a2

结果：



注意：

label就是灰度图，只有一个通道,uint16指的是这个通道内的元素类型，其实一般都是用uint8表示，mat里虽然存储的是uint16，但是取值还是在0到255之间，所以可以用uint8表示，也不会损失精度，label转换成mat还是一通道、

**训练：**

nohup python solve.py > train\_dstl32.log 2>&1 & #后台训练

tailf train\_dstl32.log #实时查看日志

测试：

*predictTest.py:* 将灰度图变为彩图部分代码如下

# load image, switch to BGR, subtract mean, and make dims C x H x W for Caffe im = Image.open('/home/jhy/env/caffe/fcn/data/dstl/Images/2.tif') #测试图片

testimg\_dir='/home/jhy/env/caffe/fcn/data/dstl/Images/10000.tif'

im = Image.open(testimg\_dir) #测试图片

in\_ = np.array(im, dtype=np.float32)

in\_ = in\_[:,:,::-1]

in\_ -= np.array(( 74.68374324824015, 89.82805046920777,79.10120382537842) ,dtype=np.float32) #减均值(104.00698793,116.66876762,122.67891434)

in\_ = in\_.transpose((2,0,1))

# load net

net = caffe.Net('deploy.prototxt', './models/dstl\_iter\_100000.caffemodel', caffe.TEST) #网络描述文件和模型路径

# shape for input (data blob is N x C x H x W), set data

print "in.shape:", in\_.shape # (3,512,512)

net.blobs['data'].reshape(1, \*in\_.shape) #第一维：N，batch size=1;\*in\_.shape:\*表示接受任意多个参数放在一个元组中[1,(3,512,512)]

net.blobs['data'].data[...] = in\_

# run net and take argmax for prediction

net.forward()

out = net.blobs['score\_sem'].data[0].argmax(axis=0) #修改score为score\_sem，获得测试图像的矩阵

#用cv2保存图片直接为灰度图原尺寸输出

#cv2.imwrite("1\_32\_gray.tif",out)

#pillow保存图片默认为彩色，变为灰色则：plt.imshow(out,cmap='gray')。默认改变了输出图像尺寸

#plt.imshow(out) #将生成的图片保存在当前目录下，为test\_result.png

#plt.axis('off')

#plt.savefig('./15555\_32.tif') #将生成的图片保存在当前目录

#设置调色盘：将测试得到的图像out矩阵，每一个类定义调色板输出为彩图

arr=out.astype(np.uint8)

im=Image.fromarray(arr)

palette=[]

for i in range(256):

palette.extend( (i,i,i) )

palette[:3\*10]=np.array([[0,0,0],[128,0,0],[0,128,0],[0,0,128],[128,0,128],[0,128,128],[128,128,128],[64,0,0],[192,0,0],[64,128,0]],dtype='uint8').flatten()

im.putpalette(palette)

im.show()

im.save('10000\_32\_palette.tif')

*deploy.prototxt:将trainval.prototxt复制过来删除所有与loss有关的层，改变第一层如下*

|  |  |
| --- | --- |
| *trainval.prototxt:* | *deploy.prototxt:* |
| *layer {*  *name: "data"*  *type: "Python"*  *top: "data"*  *top: "sem"*  *top: "geo"*  *python\_param {*  *module: "dstl\_layers"*  *layer: "DstlSegDataLayer"*  *param\_str: "{\'dstl\_dir\': \'../data/dstl\',\'split\': \'trainval\'}"*  *}*  *}* | *layer {*  *name: "input"*  *type: "Input"*  *top: "data"*  *input\_param {*  *shape { dim: 1 dim: 3 dim:256 dim:256 }*  *}*  *}* |

**测试命令：**

python predictTest.py

生成result.tif

### **10.7.2 dstl\_fcn16s与dstl\_fcn8s**

*solve.py：需要改动，直接拿上一次训练得到的模型权重赋值给当前训练模型*

|  |  |
| --- | --- |
| fcn32s | fcn16s/fcn8s |
| vgg\_weights = '/home/jhy/env/caffe/fcn/ilsvrc-nets/vgg16-fcn.caffemodel' #预训练模型地址  vgg\_proto = '/home/jhy/env/caffe/fcn/ilsvrc-nets/VGG\_ILSVRC\_16\_layers\_deploy.prototxt'  #网络描述文件地址  # init  caffe.set\_device(int(0))#修改为实际的GPU设备号，如果只有一个GPU，则设置为0  caffe.set\_mode\_gpu()  solver=caffe.SGDSolver('solver.prototxt') #solver.prototxt:trainval.prototxt and test.prototxt  vgg\_net=caffe.Net(vgg\_proto,vgg\_weights,caffe.TRAIN) #用transplant的方式获取vgg16的权重  surgery.transplant(solver.net,vgg\_net)  del vgg\_net | weights = '../dstl-fcn32s/models/dstl\_iter\_100000.caffemodel'  # init  caffe.set\_device(int(0))  caffe.set\_mode\_gpu()  solver = caffe.SGDSolver('solver.prototxt')  solver.net.copy\_from(weights) |

训练：caffe.set\_device(int(sys.argv[1])) #运行时传参

nohup python solve.py 0 > train\_dstl16/8.log 2>&1 & #后台训练并将GPU序号传参过去

tailf train\_dstl16.log #实时查看日志

# 10.8 结果评估

### **10.8.1 评估标准**

评判标准：当用一个算法对图像进行分割后，量化的对分割效果进行评估。指标有：基于Groundtruth图像计算分割图像的分割精度、过分割率、欠分割等。

**GT（ground truth）图像:**也叫mask即含有理论分割结果的图像，用来和结果图像进行比较的参照图像，通常人工标注。

**分割精度：**分割准确的面积占GT图像中真实面积的百分比，Rs表示人工标注出的分割图像的参考面积，Ts表示算法分割得到的图像的真实面积，|Rs-Ts|表示错误分割的像素点个数。



**过分割率：**即分割在GT图像参考面积之外的像素点的比率，计算公式如下：



Os表示本不应该包含在分割结果中的像素点个数，实际却在分割结果中的像素点个数。即Os中的像素点出现在实际分割图像中，但不出现在理论分割图像Rs中。

**欠分割率：**即分割在GT图像参考面积之中欠缺的像素点的比率，计算的公式如下



含义同上，Us表示本应该包含在分割结果中的像素点个数，实际却不在分割结果中的像素点个数。即Us中的像素点出现在理论分割图像中，但不出现在实际分割图像中。

案例：分割头发



//为图像分割的结果提供评价标准,预先人工绘制GT图像作为参考，与分割图像进行过分割率和欠分割率的计算

#include <opencv2/opencv.hpp>

#include <iostream>

#include <stdlib.h>

void getCounterImage(IplImage\* src,IplImage\* test,IplImage\* dst);

using namespace std;

FILE \*fp;

int main(){

IplImage\* testImg; // 测试图像;

IplImage\* gtImg; //参照图像;

IplImage\* counterImg; //轮廓图像;

fp = fopen("meanshift 方法.txt","w+");

fprintf(fp,"%s","图像编号 ");

fprintf(fp,"%s","分割准确率 ");

fprintf(fp,"%s","过分割率 ");

fprintf(fp,"%s","欠分割率 ");

int i = 1;

while(i<=5){

fprintf(fp,"%s","\n");

fprintf(fp,"%d",i);

char fileName[100] = "E:\\testdata\\Judge\\om\_hair\_dilate";

char gtFileName[100] = "E:\\testdata\\Judge\\gtHair";

char NumStr[10];

char JpgStr[10]=".jpg";

char ResStr[100]="res";

itoa(i,NumStr,10);

strcat(fileName,NumStr);

strcat(fileName,JpgStr);

strcat(gtFileName,NumStr);

strcat(gtFileName,JpgStr);

testImg = cvLoadImage(fileName);

gtImg = cvLoadImage(gtFileName);

counterImg = cvCreateImage(cvGetSize(gtImg),8,1); //8:位深度，1表示单通道图像

cvZero(counterImg);

getCounterImage(gtImg,testImg,counterImg);

cvShowImage("testImage",testImg);

cvShowImage("gtImage",gtImg);

cvMoveWindow("gtImage",0,0);

cvReleaseImage(&counterImg);

char c=cvWaitKey();

if(32==c)

i++;

}

fclose(fp);

cvWaitKey(0);

}

void getCounterImage(IplImage\* src,IplImage\* test,IplImage\* dst)

{

double SegRate; //分割准确率;

double OverSegRate; //过分割率;

double LessSegRate; //欠分割率;

CvScalar BLACK = CV\_RGB(0,0,0);

CvScalar WHITE = CV\_RGB(255,255,255);

for (int i = 0; i <src->height;i++)

for (int j = 0; j <src->width;j++){

CvScalar pixel = cvGet2D(src,i,j);

if(pixel.val[2]>200 && pixel.val[0]<50 && pixel.val[1]<50){

cvSet2D(dst,i,j,BLACK);

}

else

cvSet2D(dst,i,j,WHITE);

}

cvThreshold(dst,dst,150,255,CV\_THRESH\_BINARY\_INV);

//查找轮廓;

CvSeq \*pContour = NULL;

CvSeq \*pConInner = NULL;

CvMemStorage \*pStorage = NULL;

if(dst){

pStorage = cvCreateMemStorage(0);

cvFindContours(dst,pStorage,&pContour,sizeof(CvContour), CV\_RETR\_EXTERNAL , CV\_CHAIN\_APPROX\_SIMPLE);

cvDrawContours(dst, pContour, CV\_RGB(255, 255, 255), CV\_RGB(255,255,255), -1, CV\_FILLED, 8, cvPoint(0, 0));

}

cvNamedWindow("counter",0);

cvThreshold(dst,dst,150,255,CV\_THRESH\_BINARY\_INV);

cvShowImage("counter",dst);

/\*\*开始计算过分割率和欠分割率\*\*/

int x,y;

int SegNum=0; //分割准确的像素点个数;

int OverSegNum=0; //过分割的像素点个数;

int LessSegNum=0; //欠分割的像素点个数;

int GTNum=0; //真实像素点个数;

for (int i = 0; i <dst->height;i++)

for (int j = 0; j <dst->width;j++){

x = cvGet2D(test,i,j).val[0]; //分割图像

y = cvGet2D(dst,i,j).val[0]; //参照图像

if(x==0)

SegNum++;

if(y==0){

GTNum++;

if(x!=0)

LessSegNum++;

}

else if(x==0)

OverSegNum++;

}

SegRate = 1-(double)(abs(GTNum-SegNum))/GTNum; //分割准确率;

OverSegRate = (double)OverSegNum/(GTNum+OverSegNum); //过分割率;

LessSegRate = (double)LessSegNum/(GTNum+OverSegNum); //欠分割率;

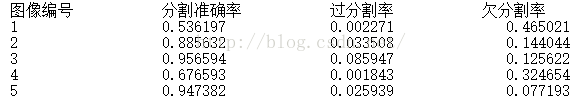
cout<<"分割准确率:"<<SegRate<<endl;

cout<<"过分割率："<<OverSegRate<<endl;

cout<<"欠分割率："<<LessSegRate<<endl;

}

结果：



### **10.8.2 卫星图像分割精度**

predictTest.py

#获取测试图片类别

print "test img size:",out.shape # (256,256)

labels = []

for i in range(out.shape[0]):

for j in range(out.shape[1]):

labels.append(out[i,j])

labelSet = []

for label in labels:

labelSet.append(label)

labelSet=list(set(labelSet)) #类别集合

print "label sum:",len(labelSet),"\nlabelSet:\n",labelSet

#获取语义类mat类别

def getClass(file):

labels = []

mat = scio.loadmat(file)

mat\_S = mat.get('S')

print "groundtruth(mat) size：", mat\_S.shape # (512L,512L)代表图片像素点

for i in range(mat\_S.shape[0]): # i代表横轴x上的像素

for j in range(mat\_S.shape[1]): # j代表纵轴y上的像素

labels.append(mat\_S[i, j]) # img[i,j]:输出一个数字代表的是灰度的类也就是语义类

labelSet = [] # 灰度类别集合，去除重复的

for label in labels:

labelSet.append(label-1)

labelSet = list(set(labelSet))

sum = labelSet.\_\_len\_\_()

print "labelSum:", sum

print "label:", labelSet

# 获取图像分割准确率

def getSegRate(groudtruth\_dir, out):

mat\_groundtruth =scio.loadmat(groundtruth\_dir)

mat\_S=mat\_groundtruth.get('S')

segNum = 0

gtNum = 512 \* 512 # 512\*512=262144

for i in range(512):

for j in range(512):

if out[i, j] == mat\_S[i,j]-1:

segNum += 1

print "SegNum:", segNum

SegRate = 1 - (float)(abs(gtNum - segNum)) / gtNum # 分割准确率

print "分割精度",SegRate

if \_\_name\_\_ == '\_\_main\_\_':

groundtruth\_dir="../data/dstl/graySemLabels/10000.mat" #获取对应的groundtruth标注的mat文件

getClass(groundtruth\_dir)

getSegRate(groundtruth\_dir,out)

运行结果：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 原图512\*512 | | train   1-15000 | | | | test 15001-18000 | | | |
| 1.tif | 2.tif | 3.tif | 13701.tif | 15555.tif | 16000.tif | 17000.tif | 17999.tif |
| ground  truth | | label：5  [0,1,2,3,8] | label：4  [0,1,4,8] | label：3  [2,4,10] | label：6  [0, 2, 4, 5, 8, 9] | label:6  [0, 1, 2, 4, 5, 8] | label:2  [4,8] |  |  |
| 测试图 | 32 | label：5  [0,1,2,3,8] | label：5  [0,1,2,4,8] | label：4  [0,2,4,8] | label：3  [2,8,9] | label:1  [8] | label:7  [0, 1, 2, 4, 5] |  |  |
| 0.714 | 0.863 | 0.951 | 0.419 | 0.131 | 0.014 |  |  |
| 16 |  |  | 0.964 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

### **10.8.3 调整策略**

loss不下降原因集合,train loss与test loss结果分析

train loss 不断下降，test loss不断下降，说明网络仍在学习;

train loss 不断下降，test loss趋于不变，说明网络过拟合;

train loss 趋于不变，test loss不断下降，说明数据集100%有问题;

train loss 趋于不变，test loss趋于不变，说明学习遇到瓶颈，需要减小学习率或批量数目;

train loss 不断上升，test loss不断上升，说明网络结构设计不当，训练超参数设置不当，数据集经过清洗等问题。

方案：

1.加入BN层：即在每个卷积层后面加上以下代码，除了全连接层以外。效果不佳，因为网络主框架是VGG在此基础微调。

layer {

bottom: "conv1\_1"

top: "conv1\_1"

name: "bn\_conv1\_1"

type: "BatchNorm"

batch\_norm\_param {

use\_global\_stats: true

}

}

layer {

bottom: "conv1\_1"

top: "conv1\_1"

name: "scale\_conv1\_1"

type: "Scale"

scale\_param {

bias\_term: true

}

}

1. 改变均值

注意：均值的获取必须是训练集中的原图，将所有图片中每个像素点的RGB相加然后除以15000张数目。

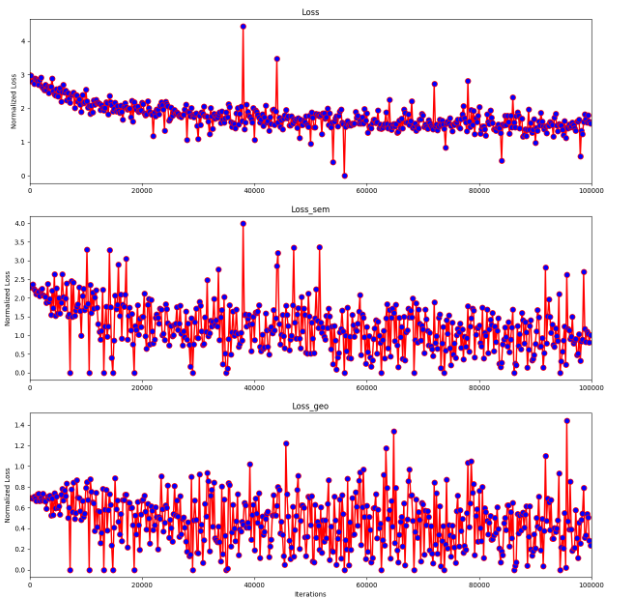
3.加上背景类训练：dstl\_layer.py中load\_label()函数中修改读取mat文件中类别映射到训练层中的类别处理，prototxt中最后loss层中的num\_output类别数目。

效果：加上背景类训练后平均精度提高，特别是含有背景类的图片平均精度大幅度提高，但是对于没有背景类的图片下降了很多。

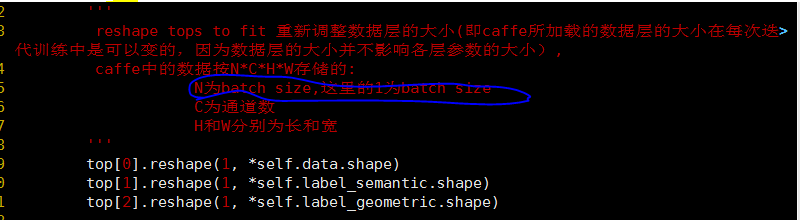
可能是因为背景类涵盖的种类太多，训练的不理想。

1. 增大学习率：fcn32s学习率是1e-10过小，震荡幅度太大，调整到1e-8变化幅度稍微小一点，总体loss有下降的趋势。

训练模型时注意学习率要逐渐变小，fcn为跳跃结构，fcn32s训练时base\_lr为1e-10,在fcn32s基础上训练fcn16s时base\_lr为1e-8



1. 改变batchsize:默认为1，由1改成5，效果不佳。



1. 改变主架模型

6.背景类的处理

loss层解读系列-softmax\_loss

Loss Function

softmax\_loss的计算包含2步：

（1）计算softmax归一化概率

（2）计算损失

这里以batchsize=1的2分类为例： 设最后一层的输出为[1.2 0.8],减去最大值后为[0 -0.4]，然后计算归一化概率得到[0.5987 0.4013]，假如该图片的label为1，则Loss=-log0.4013=0.9130.

**可选参数**

（1） *ignore\_label:*int型变量，默认为空。   
如果指定值，则label等于ignore\_label的样本将不参与Loss计算，并且反向传播时梯度直接置0.

（2） *normalize*:bool型变量，即Loss会除以参与计算的样本总数；否则Loss等于直接求和

（3） *normalization* :enum型变量，默认为VALID，具体代表情况如下面的代码。

enum NormalizationMode {

// Divide by the number of examples in the batch times spatial dimensions.

// Outputs that receive the ignore label will NOT be ignored in computing the normalization factor.

FULL = 0;

// Divide by the total number of output locations that do not take the

// ignore\_label. If ignore\_label is not set, this behaves like FULL.

VALID = 1;

// Divide by the batch size.

BATCH\_SIZE = 2;

NONE = 3;

}

归一化case的判断：

（1） 如果未设置normalization，但是设置了normalize。   
则有normalize==1 -> 归一化方式为VALID ;normalize==0 -> 归一化方式为BATCH\_SIZE

（2） 一旦设置normalization，归一化方式则由normalization决定，不再考虑normalize。

使用方法

layer {

name: "loss"

type: "SoftmaxWithLoss"

bottom: "fc1"

bottom: "label"

top: "loss"

top: "prob"

loss\_param{

ignore\_label：0

normalize: 1

normalization: FULL

}

}

扩展使用

（1） 如上面的使用方法中所示，softmax\_loss可以有2个输出，第二个输出为归一化后的softmax概率

（2） 最常见的情况是，一个样本对应一个标量label，但softmax\_loss支持更高维度的label。   
当bottom[0]的输入维度为N\*C\*H\*W时, 其中N为一个batch中的样本数量，C为channel通常等于分类数，H\*W为feature\_map的大小通常它们等于1.此时我们的一个样本对应的label不再是一个标量了，而应该是一个长度为H\*W的矢量，里面的数值范围为0——C-1之间的整数。 至于之后的Loss计算，则采用相同的处理。

# 基于Unet模型的卫星图像分割

# 11.1数据集

### **11.1.1数据集处理**

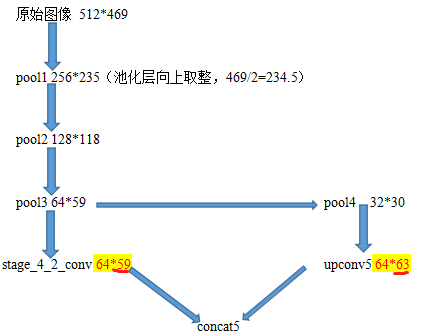
改用unet模型对卫星图像进行分割，对图像输入尺寸有要求这是因为unet模型中有concat层，concat连接的两个top输入必须尺寸维度必须一样，在处理数据集时我们遇到512\*469尺寸的图像报错是因为经过四次池化后上采样upconv5那一步与stage\_4\_2\_conv维度不一致。

输入图像尺寸必须为16的倍数，因为经过四次池化所以图片缩小了2^4=16倍。

代码如下：

if img\_h != 512 or img\_w != 512:  
 crop\_img=img[:,:448] # 指定坐标裁剪，第一个冒号表示长度不变，第二个表示宽度变为0-448

# crop\_img = cv2.resize(img, (512, 448), interpolation=cv2.INTER\_AREA) # resize是缩放会失真，清晰度会变差  
 cv2.imwrite(crop\_dir+file, crop\_img)



### **11.2.1数据集标注问题**

秦恩泉代码如下：

IpImage \*cimg=cvCreateImage(cvSize(3\*img->width,img->height) , 8,3) //cimg拼接图片：长=原图长\*3，宽度不变，8位，3通道

void combine\_imgs(IplImage \*img, IplImage \*img2, IplImage \*cimg){  
float color[256][3] = {{0,0,0}, {255,0,0},   { 0,255,0 },   { 0,0,255 },   { 255,0, 128 },{ 255,255, 0 }, { 0,255, 255 },  { 0,128, 200 },{ 200,128, 14 },  {35, 150, 78},                               {90, 78, 200 },};  
color[255][0] = 0;  
color[255][1] = 0;  
color[255][2] = 0;  
int w = img->width;  
int h = img->height;  
cvSetImageROI(cimg, cvRect(0, 0, w, h));  
cvCopy(img, cimg);  
cvResetImageROI(cimg);  
cvSetImageROI(cimg, cvRect(w, 0, w, h));  
cvCopy(img2, cimg);  
cvResetImageROI(cimg);  
int offx = 2 \* w;  
for(int i = 0; i < h; i++)  
 for (int j = 0; j < w; j++){  
 unsigned char b = img->imageData[i\*img->widthStep + 3 \* j];  
 unsigned char g = img->imageData[i\*img->widthStep + 3 \* j+1];  
 unsigned char r = img->imageData[i\*img->widthStep + 3 \* j+2];  
 unsigned char gray = img2->imageData[i\*img2->widthStep + 3 \* j];  
 unsigned char b2 = (b + color[gray][0]) \* 0.5;  
 unsigned char g2 = (g + color[gray][1]) \* 0.5;  
 unsigned char r2 = (r + color[gray][2]) \* 0.5;  
 cimg->imageData[i\*cimg->widthStep + 3 \* (j + offx)] = b2;  
 cimg->imageData[i\*cimg->widthStep + 3 \* (j + offx)+1] = g2;  
 cimg->imageData[i\*cimg->widthStep + 3 \* (j + offx)+2] = r2;  
 }  
}

# 11.2参数优化

调整学习率，loss层中加入归一化

1. 加入BN层
2. sgd+momentum
3. xavier