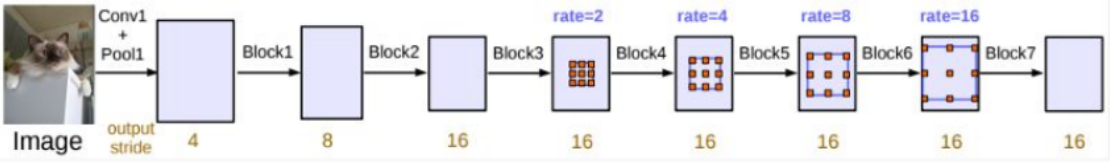
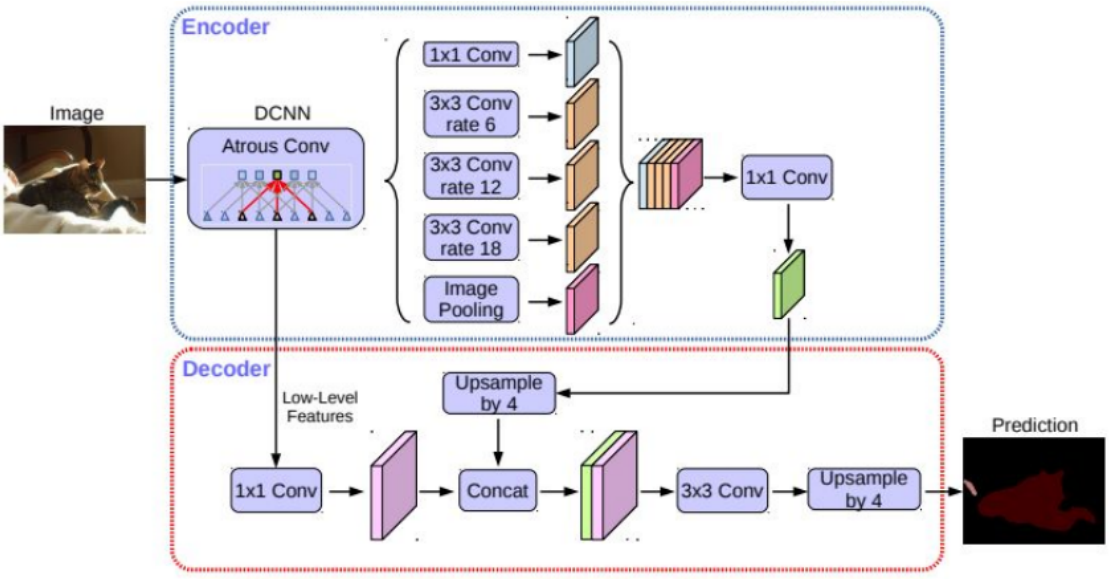
[使用Tensorflow deeplabv3+ 训练自己的数据集] [https://blog.csdn.net/heiheiya/article/details/88535576]

1.DeepLabV3与DeepLabV3+

> DeepLabV3 是另一种无需加设系数的多尺度处理方法。
>
> 这个模型十分轻量级。我们再次从一个特征提取前端开始，从第四次下采样后的特征入手处理。现在的分辨率已经很低（比输入图片小了16倍）所以我们就从这里入手就很好！不太好的一点是，在这样一种低分辨率的情况下，由于像素的低准确度，很难得到很好的定位。
>
> 这就是体现 DeepLabV3 的突出贡献之处了，对多孔卷积的巧妙运用。普通的卷积只能处理局部信息，因为权值总是一个挨着一个。例如，在一个标准的3*3卷积中，两个权重值之间的距离只有一个步长/像素。
>
> 有了多孔卷积，我们可以直接增加卷积权重值之间的空间，而实际上在操作中不增加权重值的总数。所以我们仍然只有3*3也就是9个为参数总量，我们只是把它们分布得更开了。我们把每个权重值间的距离称作扩张率。下面的模型图解很好的阐释了这个思想。
>
> 当我们使用一种低扩张率时，我们会得到非常局部/低尺度的信息。当我们采用高扩张率时，我们会处理到更多全局/高尺度的信息。因此 DeepLabV3 模型融合了不同扩张率的多孔卷积来获取多尺度信息。

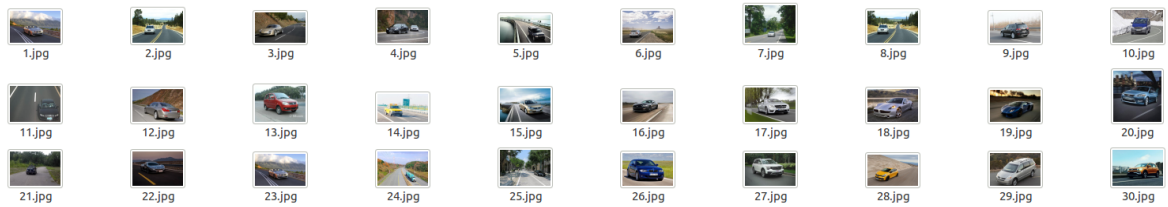


> DeepLabV3+ 模型就像它名字所说的那样，是 DeepLabV3 的一个快速延伸，借用了它之前工作的一些优势。如我们之前看到的那样，如果我们仅仅在最后利用双线性差值升尺度的话就会遇到潜在的瓶颈。事实上，原来的 DeepLabV3 在最后把尺度放大了16倍！
>
> 为了处理这件事， DeepLabV3+ 在 DeepLabV3 上加入了中间的解码模块，通过 DeepLabV3 处理后，首先特征图会被放大4倍。之后它们会和前端特征提取的原特征图一起处理，之后再放大4倍。这减轻了后端网络的负担，并提供了一条从前端特征提取到网络后面部分的捷径。



2.制作语义分割数据集

1.从网上下载30张道路和车辆的图片



2.labelme的使用

> labelme是麻省理工（MIT）的计算机科学和人工智能实验室（CSAIL）研发的图像标注工具，人们可以使用该工具创建定制化标注任务或执行图像标注，项目源代码已经开源。

项目开源地址：https://github.com/CSAILVision/LabelMeAnnotationTool

labelMe项目地址：http://labelme.csail.mit.edu/Release3.0/

安装

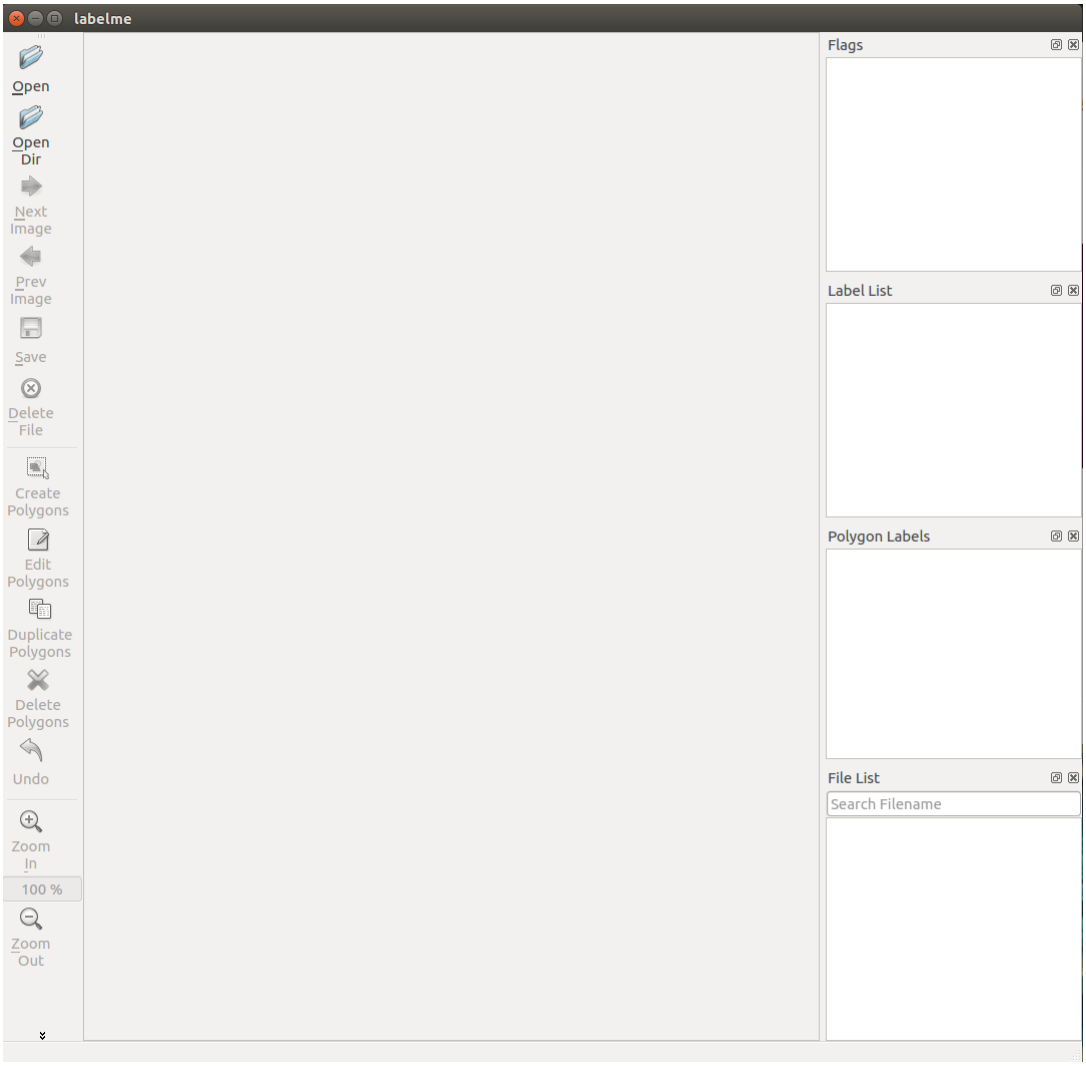
```
sudo apt-get install python-pyqt5
sudo pip install labelme
```

使用

```
Traceback (most recent call last):
  File "/usr/local/lib/python2.7/dist-packages/labelme/app.py", line 1622, in op
enDirDialog
    self.importDirImages(targetDirPath)
  File "/usr/local/lib/python2.7/dist-packages/labelme/app.py", line 1642, in im
portDirImages
    for filename in self.scanAllImages(dirpath):
  File "/usr/local/lib/python2.7/dist-packages/labelme/app.py", line 1666, in sc
anAllImages
    if file.lower().endswith(tuple(extensions)):
UnicodeDecodeError: 'ascii' codec can't decode byte 0xe6 in position 0: ordinal
not in range(128)
已放弃 (核心已转储)
```

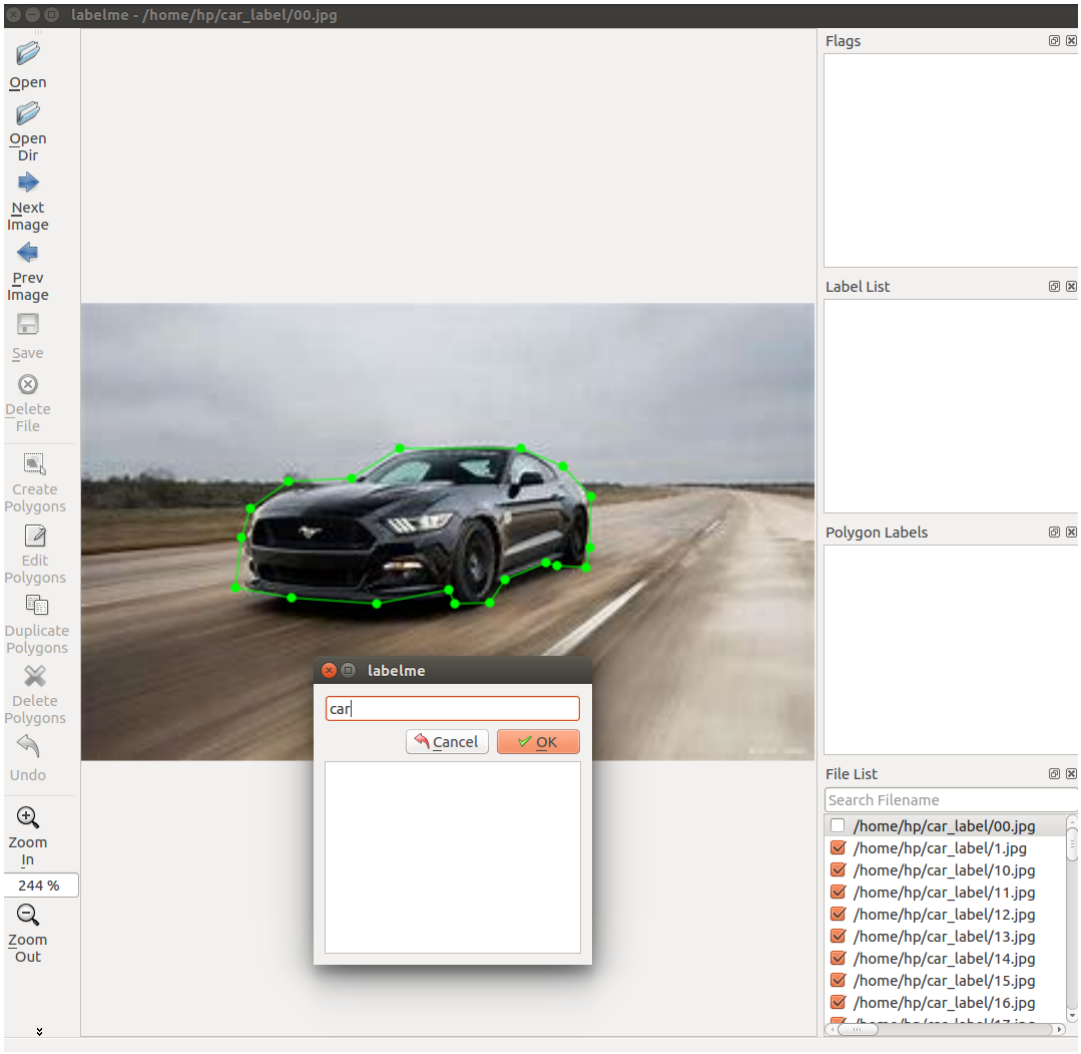
> 报错1：路径文件夹名称不要包含中文

初始界面

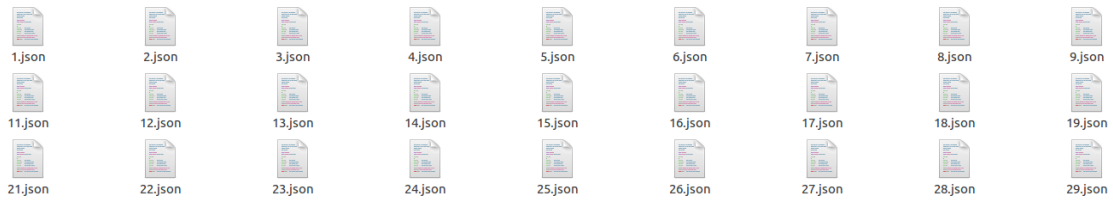


OpenDir可以直接批量处理文件夹内的图片

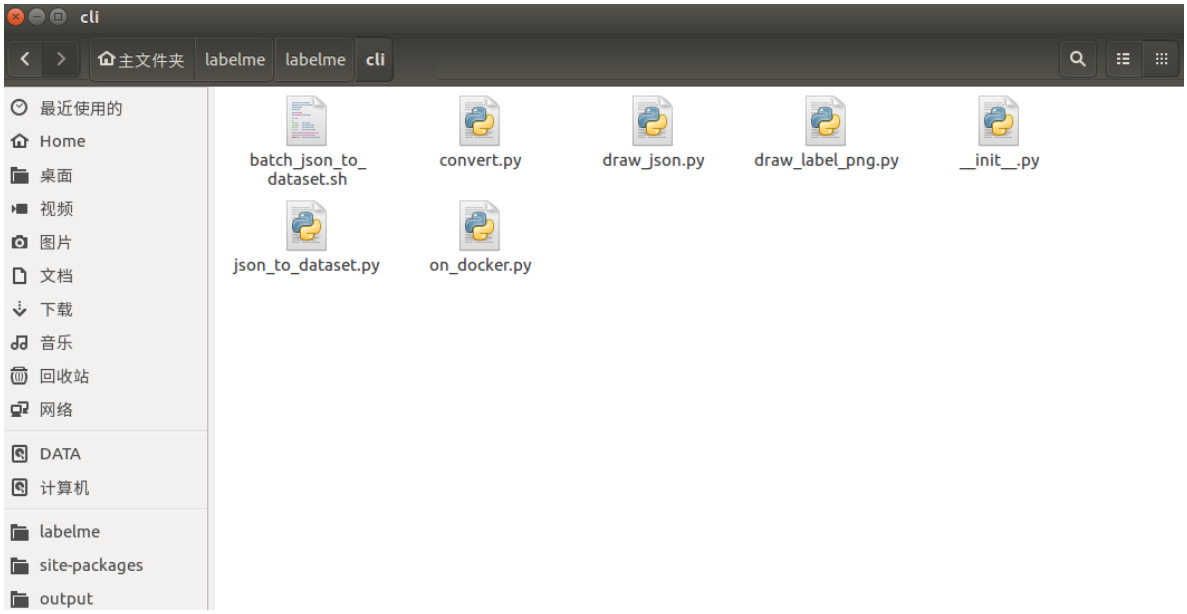
使用CreatePolygons进行多边形框选，框选目标后可以设置标签名称



完成后会生成json文件



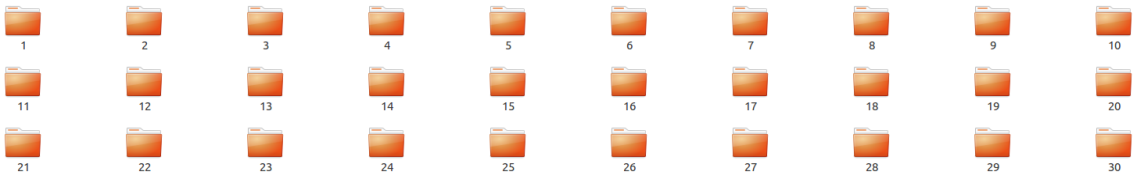
labelme下面的cli文件夹下有一个json_to_dataset.py，执行该文件，解析json文件



使用脚本对文件批量处理

```
num=31
for ((i=1;i<num;i++))
do
  python json_to_dataset.py /home/hp/car_label/$i.json -o
/home/hp/car_label/output/$i
done
```

处理后



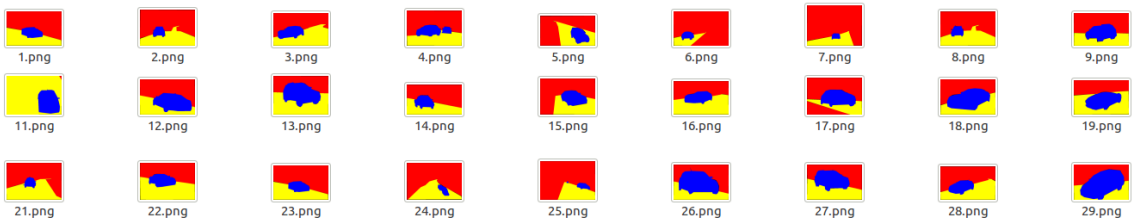
每个文件夹对应一张图片，文件夹内有五个文件

接下来为标注出来的label.png进行着色

写一个python程序，命名为convert.py

```
import PIL.Image
import numpy as np
from skimage import io,data,color
import matplotlib.pyplot as plt

num=31
for i in range(num):
    img=PIL.Image.open("/home/hp/car_label/output/%d/label.png"%i)
    img=np.array(img)
    dst=color.label2rgb(img,bg_label=0,bg_color=(0,0,0))
    io.imsave("/home/hp/car_label/output/dstlabel/%d.png"%i,dst)
```

着色后的结果



刚刚着色之后的图是24位png图，deeplab中的标注图片需要是灰度图

原博主使用matlab进行转换

```
dirs=dir('car_label/output/*.png');
for n=1:numel(dirs)
    strname=strcat('car_label/output/',dirs(n).name);
    img=imread(strname);
    [x,map]=rgb2ind(img,256);
    newname=strcat('car_label/output/dstlabel/',dirs(n).name);
    imwrite(x,newname);
end
```

这是生成的结果

因为图像的像素值其实是像素的类别，是非常小的，看起来是一片黑的

	1	2	3	4	5	6	7	8	9	10	11	12
237	0	0	2	2	2	2	2	2	2	2	2	
238	0	0	2	2	2	2	2	2	2	2	2	
239	0	0	2	2	2	2	2	2	2	2	2	
240	0	0	2	2	2	2	2	2	2	2	2	
241	0	0	2	2	2	2	2	2	2	2	2	
242	0	0	2	2	2	2	2	2	2	2	2	
243	0	0	2	2	2	2	2	2	2	2	2	
244	0	0	2	2	2	2	2	2	2	2	2	
245	0	0	2	2	2	2	2	2	2	2	2	
246	0	0	2	2	2	2	2	2	2	2	2	
247	0	0	2	2	2	2	2	2	2	0	0	
248	0	0	2	2	2	2	0	0	0	0	0	
249	0	0	2	0	0	0	0	0	0	0	0	
250	0	0	0	0	0	0	0	0	0	0	0	

使用matlab可以看出区别

至此，我们的原始数据集基本制作完成

3.使用 deepv3+ 训练数据集

编写python程序 clist.py 生成训练集的 train.txt

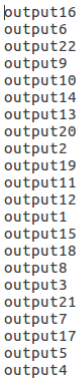
```
import os
import random

trainfilepath = 'train'
txtsavepath = 'txt'
train_file = os.listdir(trainfilepath)

num=len(train_file)
list = range(num)

train = random.sample(list, num)

os.chdir(txtsavepath)

ftrain = open('train.txt', 'w')

for i in list :
```

```
  name =train_file[i][:-4] + '\n'
  ftrain.write(name)
ftrain.close()
```

按照同样的方法生成 val.txt

本次使用22张图片作为训练集，5张图片作为验证集，3张图片作为测试集

生成的train.txt如图所示



下面生成tfrecord文件

> TFRecord 是什么?
>
> TFRecord 是谷歌推荐的一种二进制文件格式，理论上它可以保存任何格式的信息
>
> 下面是Tensorflow 的官网给出的文档结构，整个文件由文件长度信息、长度校验码、数据、数据校验码组成
>
> ```
> uint64 length
> uint32 masked_crc32_of_length
> byte   data[length]
> uint32 masked_crc32_of_data
> ```
>
> 但对于我们普通开发者而言，我们并不需要关心这些，Tensorflow 提供了丰富的 API 可以帮助我们轻松读写 TFRecord 文件

> 优点：
>
> 1、它特别适应于 Tensorflow ，或者说它就是为 Tensorflow 量身打造的。
> 2、因为 Tensorflow开发者众多，统一训练时数据的文件格式是一件很有意义的事情。也有助于降低学习成本和迁移成本。

使用models_master/research/deeplab/datasets/build_voc2012_data.py来生成

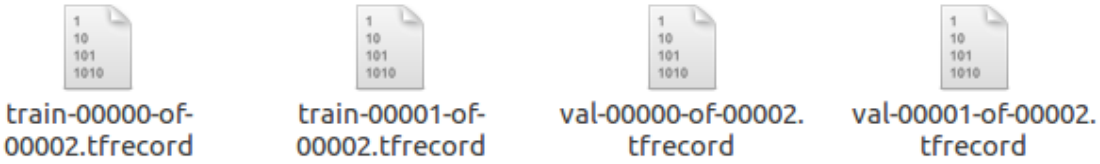
按照要求整理文件夹

```
+ pascal_voc_seg
  - build_data.py
  - build_voc2012_data.py (current working directory).
  + VOCdevkit
    + VOC2012
      + JPEGImages
      + SegmentationClass
      + ImageSets
        + Segmentation
```

```
    + tfrecord

Image folder:
  ./VOCdevkit/VOC2012/JPEGImages

Semantic segmentation annotations:
  ./VOCdevkit/VOC2012/SegmentationClass

list folder:
  ./VOCdevkit/VOC2012/ImageSets/Segmentation
```

由于图片数量比较少，所以将81行的_NUM_SHARDS = 2，默认是4

生成的tfrecord如下图所示



使用现有模型训练时，首先要注册自己的数据集

在models_master/research/deeplab/datasets/segmentation_dataset.py中110行的地方添加如下内容

```
_SEGTEST_INFORMATION = DatasetDescriptor(
    splits_to_sizes={
        'train': 22,  # num of samples in images/training
        'val': 5,  # num of samples in images/validation
    },
    num_classes=4,
    ignore_label=255,
)
```

`ignore_label`是不参与计算loss的，在mask中将`ignore_label`的灰度值标记为`255`

在120行把自己的数据集注册进去

```
_DATASETS_INFORMATION = {
    'cityscapes': _CITYSCAPES_INFORMATION,
    'pascal_voc_seg': _PASCAL_VOC_SEG_INFORMATION,
    'ade20k': _ADE20K_INFORMATION,
    'segtest': _SEGTEST_INFORMATION
}
```

最后进行训练

```
WORK_DIR="./"
DATASET_DIR="./datasets/"


# Set up the working directories.
SEG_FOLDER="mysegtest"
EXP_FOLDER="exp/train_on_trainval_set"
INIT_FOLDER="${DATASET_DIR}/${SEG_FOLDER}/init_models"
TRAIN_LOGDIR="${DATASET_DIR}/${SEG_FOLDER}/${EXP_FOLDER}/train"
```

```
EVAL_LOGDIR="${DATASET_DIR}/${SEG_FOLDER}/${EXP_FOLDER}/eval"
VIS_LOGDIR="${DATASET_DIR}/${SEG_FOLDER}/${EXP_FOLDER}/vis"
EXPORT_DIR="${DATASET_DIR}/${SEG_FOLDER}/${EXP_FOLDER}/export"
mkdir -p "${INIT_FOLDER}"
mkdir -p "${TRAIN_LOGDIR}"
mkdir -p "${EVAL_LOGDIR}"
mkdir -p "${VIS_LOGDIR}"
mkdir -p "${EXPORT_DIR}"


SEG_DATASET="${DATASET_DIR}/tfrecord"

# Train 10 iterations.
NUM_ITERATIONS=500
python "${WORK_DIR}"/train.py \
  --logtostderr \
  --train_split="train" \
  --model_variant="xception_65" \
  --atrous_rates=6 \
  --atrous_rates=12 \
  --atrous_rates=18 \
  --output_stride=16 \
  --decoder_output_stride=4 \
  --train_crop_size=513 \
  --train_crop_size=513 \
  --train_batch_size=1 \
  --training_number_of_steps="${NUM_ITERATIONS}" \
  --fine_tune_batch_norm=true \
  --dataset="segtest" \
  --tf_initial_checkpoint="./backbone/deeplabv3_cityscapes_train/model.ckpt" \
  --train_logdir="${TRAIN_LOGDIR}" \
  --dataset_dir="${SEG_DATASET}" \
  --initialize_last_layer=False \
  --last_layers_contain_logits_only=True
```

原博主`train_batch_siaze`设置为4，运行时发现超出内存容量，改为1后正常运行

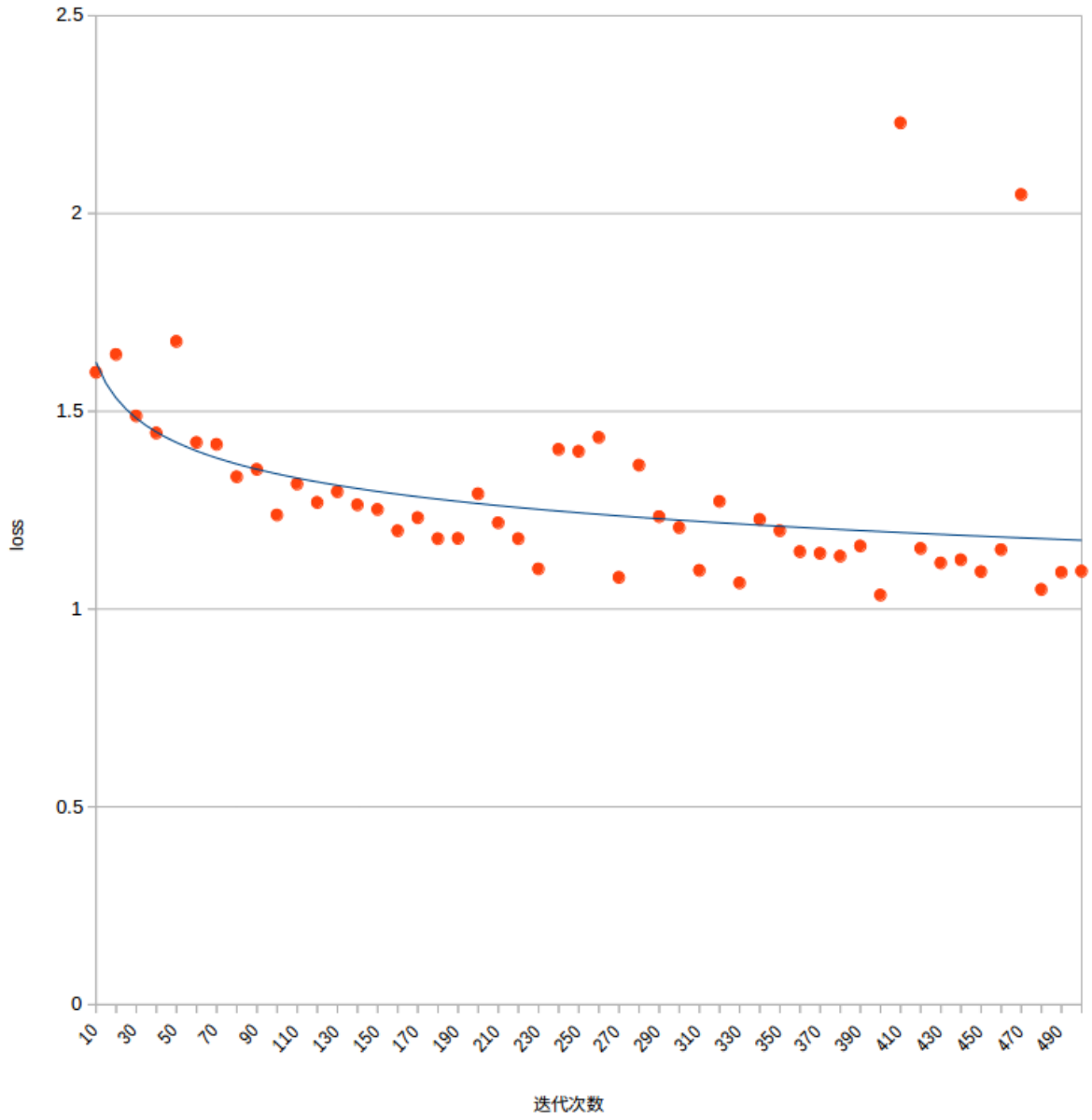
运行时每10次输出一个loss值

```
INFO:tensorflow:global step 10: loss = 1.5983 (9.208 sec/step)
INFO:tensorflow:global step 20: loss = 1.6434 (9.235 sec/step)
INFO:tensorflow:global step 30: loss = 1.4879 (9.397 sec/step)
INFO:tensorflow:global step 40: loss = 1.4448 (9.191 sec/step)
INFO:tensorflow:global step 50: loss = 1.6763 (9.159 sec/step)
INFO:tensorflow:global step 60: loss = 1.4209 (9.273 sec/step)
INFO:tensorflow:global_step/sec: 0.10505
INFO:tensorflow:Recording summary at step 63.
INFO:tensorflow:global step 70: loss = 1.4162 (9.224 sec/step)
INFO:tensorflow:global step 80: loss = 1.3340 (9.255 sec/step)
INFO:tensorflow:global step 90: loss = 1.3530 (9.179 sec/step)
INFO:tensorflow:global step 100: loss = 1.2377 (9.195 sec/step)
INFO:tensorflow:global step 110: loss = 1.3160 (9.173 sec/step)
INFO:tensorflow:global step 120: loss = 1.2693 (9.211 sec/step)
INFO:tensorflow:Saving checkpoint to path ./datasets//mysegtest/exp/train_on_trainval_set/train/model.ckpt
INFO:tensorflow:global_step/sec: 0.108332
INFO:tensorflow:Recording summary at step 128.
INFO:tensorflow:global step 130: loss = 1.2964 (9.207 sec/step)
INFO:tensorflow:global step 140: loss = 1.2632 (9.185 sec/step)
INFO:tensorflow:global step 150: loss = 1.2515 (9.238 sec/step)
INFO:tensorflow:global step 160: loss = 1.1976 (9.236 sec/step)
INFO:tensorflow:global step 170: loss = 1.2308 (9.297 sec/step)
INFO:tensorflow:global step 180: loss = 1.1777 (9.215 sec/step)
INFO:tensorflow:global step 190: loss = 1.1784 (9.172 sec/step)
INFO:tensorflow:global_step/sec: 0.108335
INFO:tensorflow:Recording summary at step 193.
INFO:tensorflow:global step 200: loss = 1.2912 (9.152 sec/step)
INFO:tensorflow:global step 210: loss = 1.2178 (9.200 sec/step)
INFO:tensorflow:global step 220: loss = 1.1777 (9.158 sec/step)
INFO:tensorflow:global step 230: loss = 1.1014 (9.213 sec/step)
INFO:tensorflow:global step 240: loss = 1.4035 (9.168 sec/step)
INFO:tensorflow:global step 250: loss = 1.3984 (9.250 sec/step)
INFO:tensorflow:Saving checkpoint to path ./datasets//mysegtest/exp/train_on_trainval_set/train/model.ckpt
INFO:tensorflow:global_step/sec: 0.108333
INFO:tensorflow:Recording summary at step 258.
```

```
INFO:tensorflow:global step 260: loss = 1.4337 (9.154 sec/step)
INFO:tensorflow:global step 270: loss = 1.0797 (9.204 sec/step)
INFO:tensorflow:global step 280: loss = 1.3634 (9.224 sec/step)
INFO:tensorflow:global step 290: loss = 1.2335 (9.228 sec/step)
INFO:tensorflow:global step 300: loss = 1.2057 (9.211 sec/step)
INFO:tensorflow:global step 310: loss = 1.0976 (9.188 sec/step)
INFO:tensorflow:global step 320: loss = 1.2720 (9.235 sec/step)
INFO:tensorflow:global_step/sec: 0.106667
INFO:tensorflow:Recording summary at step 323.
INFO:tensorflow:global step 330: loss = 1.0659 (9.170 sec/step)
INFO:tensorflow:global step 340: loss = 1.2265 (9.201 sec/step)
INFO:tensorflow:global step 350: loss = 1.1978 (9.199 sec/step)
INFO:tensorflow:global step 360: loss = 1.1449 (9.231 sec/step)
INFO:tensorflow:global step 370: loss = 1.1406 (9.251 sec/step)
INFO:tensorflow:global step 380: loss = 1.1334 (9.207 sec/step)
INFO:tensorflow:Saving checkpoint to path ./datasets//mysegtest/exp/train_on_trainval_set/train/model.ckpt
INFO:tensorflow:global_step/sec: 0.108298
INFO:tensorflow:Recording summary at step 388.
INFO:tensorflow:global step 390: loss = 1.1591 (9.160 sec/step)
INFO:tensorflow:global step 400: loss = 1.0351 (9.135 sec/step)
INFO:tensorflow:global step 410: loss = 2.2288 (9.237 sec/step)
INFO:tensorflow:global step 420: loss = 1.1530 (9.175 sec/step)
INFO:tensorflow:global step 430: loss = 1.1163 (9.083 sec/step)
INFO:tensorflow:global step 440: loss = 1.1244 (9.089 sec/step)
INFO:tensorflow:global step 450: loss = 1.0944 (9.082 sec/step)
INFO:tensorflow:global_step/sec: 0.110036
INFO:tensorflow:Recording summary at step 453.
INFO:tensorflow:global step 460: loss = 1.1501 (9.085 sec/step)
INFO:tensorflow:global step 470: loss = 2.0476 (9.071 sec/step)
INFO:tensorflow:global step 480: loss = 1.0493 (9.097 sec/step)
INFO:tensorflow:global step 490: loss = 1.0926 (9.106 sec/step)
INFO:tensorflow:global step 500: loss = 1.0955 (9.062 sec/step)
INFO:tensorflow:Stopping Training.
INFO:tensorflow:Finished training! Saving model to disk.
```

共运行500次，平均每次约9.1s，共耗时一个半小时左右

loss的变化趋势



训练之后进行验证

```
python "${WORK_DIR}"/eval.py \
```

```
  --logtostderr \
  --eval_split="val" \
  --model_variant="xception_65" \
  --atrous_rates=6 \
  --atrous_rates=12 \
  --atrous_rates=18 \
  --output_stride=16 \
  --decoder_output_stride=4 \
  --eval_crop_size=514 \
  --eval_crop_size=794 \
  --dataset="segtest" \
  --checkpoint_dir="${TRAIN_LOGDIR}" \
  --eval_logdir="${EVAL_LOGDIR}" \
  --dataset_dir="${SEG_DATASET}" \
  --max_number_of_evaluations=1
```

```
INFO:tensorflow:Starting evaluation at 2019-10-11-08:11:17
INFO:tensorflow:Evaluation [1/5]
INFO:tensorflow:Evaluation [2/5]
INFO:tensorflow:Evaluation [3/5]
INFO:tensorflow:Evaluation [4/5]
INFO:tensorflow:Evaluation [5/5]
INFO:tensorflow:Finished evaluation at 2019-10-11-08:11:37
miou_1.0[0.428180873]
```

图片尺寸可能会报错

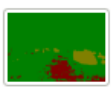
根据自己的数据集中最大的图片尺寸更改`eval_crop_size`即可

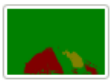
最后进行可视化

```
python "${WORK_DIR}"/vis.py \
  --logtostderr \
  --vis_split="val" \
  --model_variant="xception_65" \
  --atrous_rates=6 \
  --atrous_rates=12 \
  --atrous_rates=18 \
  --output_stride=16 \
  --decoder_output_stride=4 \
  --vis_crop_size=514 \
  --vis_crop_size=794 \
  --dataset="segtest" \
  --checkpoint_dir="${TRAIN_LOGDIR}" \
  --vis_logdir="${VIS_LOGDIR}" \
  --dataset_dir="${SEG_DATASET}" \
  --max_number_of_iterations=1
```

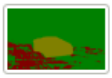
在`mysegtest/exp/train_on_trainval_set/vis/segmentation_results`可以找到可视化之后的结果


000000_prediction.png


000003_prediction.png


000004_prediction.png


000002_prediction.png


000001_prediction.png

根据结果进行模型修正等工作