注意：Deeplab于2019年11月在github上进行了更新，文档中部分地方区分了旧版（2019年11月以前）和新版（更新后的github版本）.

学员可使用百度网盘旧版deeplab目录下载旧版models.tar.gz或者使用从github上克隆新版deeplab进行项目实践。

1 图像标注工具labelme的安装与使用

1) 安装图像标注工具labelme

(1.1) Ubuntu Linux下的安装:

建议使用Python 2.7和Qt4安装。Python 3和Qt5安装容易出问题。

Python 2 + Qt4安装

```
# Python2
sudo apt-get install python-qt4  # PyQt4
```

```
sudo pip install labelme
```

如果安装过程中提示缺少某个包，可安装上，如:

```
pip install  pyyaml
```

安装好后，可仍然使用python3

(1.2) Anaconda下安装

Anaconda下安装相对容易，执行命令:

```
conda install  scikit-image
```

```
pip install labelme -i https://pypi.tuna.tsinghua.edu.cn/simple
```

2）使用labelme进行图像标注

执行:

```
labelme
```

标注后生成json文件

课程roadscene项目案例的数据集为1280*720的图片，136张用于训练，16张用于验证。

3）图像标注后的数据转换

重要说明:

- **json文件需要解析之后才能得到标签图片。这里使用labelme2voc.py生成voc数据集格式的数据集，其中产生的label图可以保证每一类的编号都是一致的。**
- **该脚本转换完成的label图是8-bit的RGB图，需要再转换成灰度图。**

转换步骤：

（1）训练数据集生成标签图

建立文件夹/home/bai/dataset/dataset_train，并dataset_train下建立子文件夹data_annotated。

把训练数据集图像和labelme标注的json文件放置到dataset_train/data_annotated目录下。

在/home/bai/dataset目录下执行:

```
python labelme2voc.py dataset_train/data_annotated
dataset_train/data_dataset_voc --labels labels.txt
```

其中，labels.txt中是需要分割的物体的类别。本项目包括：

```
__ignore__
_background_
pothole
car
midlane
rightlane
dashedline
```

执行后生成：

- data_dataset_voc/JPEGImages
- data_dataset_voc/SegmentationClass
- data_dataset_voc/SegmentationClassVisualization

（2）测试数据集生成标签图

建立文件夹/home/bai/dataset/dataset_val，并dataset_val下建立子文件夹data_annotated。

把测试数据集图像和labelme标注的json文件放置到dataset_train/data_annotated目录下。

在/home/bai/dataset目录下执行:

```
python labelme2voc.py dataset_val/data_annotated dataset_val/data_dataset_voc --
labels labels.txt
```

(3) mask灰度值的转换:

mask的灰度值设置

CamVid有11个分类，项目roadscene自己的数据集有6个分类。生成的mask必须为单通道(灰度图)，为png格式。

注意，在制作mask时，对所有object的灰度像素值有要求。

对于所有objects包括background在内，在mask中要将灰度值标注为0, 1,...n。虽然产生的图片肉眼很难看出区别，但是这对训练是有效的。注意，不要把object的灰度值标注成10,20,100...。

因为自己的数据集中没有ignore_label，所以没有点设置为255(自己的数据集和CamVid都没有。如果存在ingore_label,在视觉上是白色的)。

关于图像的mask，设置如下：

将background的灰度值设置为0 object1,2,3,4,5的灰度值设置为1,2,3,4,5.

去除mask的colormap

使用~/models/research/deeplab/datasets/remove_gt_colormap.py

deeplab/datasets 中自带的 `remove_gt_colormap.py` 脚本仅适用于带有 colormap 的单通道png图像。

如果是这种类型的png图像，则可以使用自带的 remove_gt_colormap.py 脚本进行转换。

（1）对训练集mask去除colormap:

在/home/bai/dataset下执行：

```
python ~/models/research/deeplab/datasets/remove_gt_colormap.py --
original_gt_folder dataset_train/data_dataset_voc/SegmentationClassPNG --
output_dir dataset_train/data_dataset_voc/SegmentationClassPNG-raw
```

（2）对测试集mask去除colormap:

在/home/bai/dataset下执行：

```
python ~/models/research/deeplab/datasets/remove_gt_colormap.py --
original_gt_folder dataset_val/data_dataset_voc/SegmentationClassPNG --
output_dir dataset_val/data_dataset_voc/SegmentationClassPNG-raw
```

2 数据集处理

数据集处理分成三步：

- 标注数据（见上节）
- 制作指引文件
- 将数据打包成TFRecord

RoadScene数据集下载

下载地址：百度网盘链接

```
链接：https://pan.baidu.com/s/1YDwnIpyh-X4TY31teKMDfg
提取码：nhrp
```

数据集的文件夹结构为：

├── test

├── testannot

├── train

├── trainannot

├── val

└── valannot

由上到下分别是测试集、测试集标签、训练集、训练集标签、验证集、验证集标签。

对于RoadScene, 其中训练集、验证集中的图片数目分别为train 136， val 16。

制作指引文件

TF 提供了一种统一输入数据的格式—— TFRecord 它有两个特别好的优点：

1. 可以将一个样本的所有信息统一起来存储，这些信息可以是不同的数据类型。其内部使用“Protocol Buffer”二进制数据编码方案。
2. 利用文件队列的多线程操作，使得数据的读取和批量处理更加方便快捷。

在制作TFRecord之前，需要有文件指引将数据集分类成训练集、测试集、验证集，故需要创建指引文件。

将所有图片和mask分在两个文件夹下，设置如下：

/home/bai/dataset/RoadScene/image: 存放所有的输入图片，共有152张，这其中包括训练集、测试集、验证集的图片。

/home/bai/dataset/RoadScene/mask：存放所有的标签图片，共有152张，和image文件夹下的图片是一一对应的。

对于RoadScene数据集，创建了一个目录/home/bai/dataset/RoadScene/index，该目录下包含三个.txt文件：

- train.txt：所有训练集的文件名称
- trainval.txt：所有验证集的文件名称
- val.txt：所有测试集的文件名称

将数据转换成TFRecord

创建文件夹tfrecord:

```
mkdir tfrecord
```

将上述制作的数据集打包成TFRecord，使用的是build_voc2012_data.py

执行:

```
python ~/models/research/deeplab/datasets/build_voc2012_data.py \
  --image_folder="/home/bai/dataset/RoadScene/image" \
  --semantic_segmentation_folder="/home/bai/dataset/RoadScene/mask" \
  --list_folder="/home/bai/dataset/RoadScene/index" \
  --image_format="jpg" \
  --output_dir="/home/bai/dataset/RoadScene/tfrecord"
```

- image_folder ： 数据集中原输入数据的文件目录地址
- semantic_segmentation_folder：数据集中标签的文件目录地址
- list_folder : 将数据集分类成训练集、验证集等的指示目录文件目录
- image_format : 输入图片数据的格式，RoadScene是jpg格式
- output_dir：制作的TFRecord存放的目录地址(自己创建)

3 网络训练

3.1 修改训练脚本

在DeepLabv3+模型的基础上，主要需要修改以下两个文件

- `data_generator.py`文件
- `train_utils.py`

添加数据集描述

在datasets/data_generator.py文件中，添加自己的数据集描述：

```
_MYDATA_INFORMATION = DatasetDescriptor(
    splits_to_sizes={
        'train': 136,  # num of samples in images/training
        'val': 16,  # num of samples in images/validation
    },
    num_classes=6,
    ignore_label=255,
)
```

自己的数据集共有6个classes，算上了background。由于没有使用`ignore_label`，没有算上`ignore_label`。

注册数据集

同时在datasets/data_generator.py文件，添加对应数据集的名称：

```
_DATASETS_INFORMATION = {
    'cityscapes': _CITYSCAPES_INFORMATION,
    'pascal_voc_seg': _PASCAL_VOC_SEG_INFORMATION,
    'ade20k': _ADE20K_INFORMATION,
    'camvid':_CAMVID_INFORMATION, #camvid示例
    'mydata':_MYDATA_INFORMATION, #自己的数据集
}
```

修改train_utils.py

对应的utils/train_utils.py中，将`exclude_list`的设置修改(新版第210行；旧版第159行)，作用是在使用预训练权重时候，不加载该`logit`层：

```
exclude_list = ['global_step','logits']
if not initialize_last_layer:
exclude_list.extend(last_layers)
```

如果想在DeepLab的基础上fine-tune其他数据集， 可在deeplab/train.py中修改输入参数。

一些选项：

- 使用预训练的所有权重，设置initialize_last_layer=True
- 只使用网络的backbone，设置initialize_last_layer=False和last_layers_contain_logits_only=False
- 使用所有的预训练权重，除了logits。因为如果是自己的数据集，对应的classes不同(这个我们前面已经设置不加载logits),可设置initialize_last_layer=False和last_layers_contain_logits_only=True

这里使用的设置是：

initialize_last_layer=False #157行 (新版)

last_layers_contain_logits_only=True #160行 （新版）

initialize_last_layer=False #133行 (旧版)

last_layers_contain_logits_only=True #136行 （旧版）

3.2 下载预训练模型

在model_zoo上下载预训练模型：

下载地址:https://github.com/tensorflow/models/blob/master/research/deeplab/g3doc/model_zoo.md

下载的预训练权重为xception_cityscapes_trainfine

文件大小为439M

下载到deeplab目录下，然后解压：

tar -zxvf deeplabv3_cityscapes_train_2018_02_06.tar.gz

需要注意对应的解压文件目录为：

```
/home/bai/models/research/deeplab/deeplabv3_cityscapes_train
```

注意：在训练CamVid时，没有考虑类别之间的的平衡问题，所以没有做imblance的修正。如果你是二分类或者存在严重的imblance情况，要考虑类别不平衡修正方法。

类别不平衡修正

roadscene分割项目案例中的数据集，因为是6分类问题，其中background占了非常大的比例，设置的权重比例为1，10，15, 10, 10, 10等。

注意：权重的设置对最终的分割性能有影响。权重的设置因数据集而异。

新版权重修改：

在train_utils.py文件中add_softmax_cross_entropy_loss_for_each_scale函数中第一句加上你设置的权重,例如 loss_weight=[1.0, 10.0, 15.0, 10.0, 10.0, 10.0] 其中1.0是背景权重，后面是5个类别物体的权重。

即增加一句（74行）：

```
 loss_weight=[1.0, 10.0, 15.0, 10.0, 10.0, 10.0]
```

其中

1. 0 对应灰度值0，即background; 10.0, 15.0, 10.0, 10.0, 10.0 分别对应object1, object2,object3,object4,object5。

旧版权重修改：

在train_utils.py的89行修改权重如下：

```
ignore_weight = 0
```

```
label0_weight = 1  # 对应灰度值0，即background
label1_weight = 10  # 对应object1, mask 中灰度值1
label2_weight = 15  # 对应object2,.mask 中灰度值2
label3_weight = 10  # 对应object3, mask 中灰度值3
label4_weight = 10  # 对应object4,.mask 中灰度值4
label5_weight = 10  # 对应object5,.mask 中灰度值5

not_ignore_mask = \
tf.to_float(tf.equal(scaled_labels, 0)) * label0_weight + \
tf.to_float(tf.equal(scaled_labels, 1)) * label1_weight + \
tf.to_float(tf.equal(scaled_labels, 2)) * label2_weight + \
tf.to_float(tf.equal(scaled_labels, 3)) * label0_weight + \
tf.to_float(tf.equal(scaled_labels, 4)) * label1_weight + \
tf.to_float(tf.equal(scaled_labels, 5)) * label2_weight + \
tf.to_float(tf.equal(scaled_labels, ignore_label)) * ignore_weight

tf.losses.softmax_cross_entropy(
    one_hot_labels,
    tf.reshape(logits, shape=[-1, num_classes]),
    weights=not_ignore_mask,
    scope=loss_scope)
```

3.3 执行训练指令

网络训练注意如下几个参数：

- tf_initial_checkpoint：预训练的权重，因为CamVid和自己的RoadScene数据集都和CityScapes类似，所以使用的是CityScapes的预训练权重
- train_logdir: 训练产生的文件存放位置
- dataset_dir: 数据集的TFRecord文件
- dataset：设置为在data_generator.py文件设置的数据集名称

在目录 ~/models/research/deeplab下执行

```
 python train.py \
    --logtostderr \
    --training_number_of_steps=3000 \
    --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=513,513 \
    --train_batch_size=4 \
    --dataset="mydata" \
    --
tf_initial_checkpoint='/home/bai/models/research/deeplab/deeplabv3_cityscapes_train/model.ckpt' \
    --train_logdir='/home/bai/models/research/deeplab/exp/mydata_train/train' \
    --dataset_dir='/home/bai/dataset/RoadScene/tfrecord'
```

Q: 如何设置train_crop_size的值?

A: output_stride * k + 1, where k is an integer. For example, we have 321x321， 513x513 。

4 网络测试

4.1 测试结果可视化

在目录 ~/models/research/deeplab下执行

```
python 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=720,1280\
    --dataset="mydata" \
    --colormap_type="pascal" \
    --checkpoint_dir='/home/bai/models/research/deeplab/exp/mydata_train/train'
\
    --vis_logdir='/home/bai/models/research/deeplab/exp/mydata_train/vis' \
    --dataset_dir='/home/bai/dataset/RoadScene/tfrecord'
```

- vis_split：设置为测试集
- vis_crop_size:设置720,1280为图片的大小
- dataset：设置为我们在data_generator.py文件设置的数据集名称
- dataset_dir：设置为创建的TFRecord
- colormap_type：可视化标注的颜色

  可到目录deeplab/exp/mydata_train/vis下查看可视化结果

4.2 性能评估：

```
python 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=720,1280\
    --dataset="mydata" \
    --checkpoint_dir='/home/bai/models/research/deeplab/exp/mydata_train/train'
\
    --eval_logdir='/home/bai/models/research/deeplab/exp/mydata_train/eval' \
    --dataset_dir='/home/bai/dataset/RoadScene/tfrecord' \
    --max_number_of_evaluations=1
```

- eval_split：设置为测试集
- crop_size：同样设置为720和1280
- dataset：设置为mydata

- dataset_dir：设置为我们创建的数据集

查看mIoU值：

tensorboard --logdir /home/bai/models/research/deeplab/exp/mydata_train/eval

查看训练过程的loss：

tensorboard --logdir /home/bai/models/research/deeplab/exp/mydata_train/train