基于 DeepLab 模型的水下图像分割

基于 DeepLab 模型的水下图像分割

实验目标：

使用神经网络的任意模型训练数据，探究光对图像分割的影响
使用给定的数据集：亮光条件下阀门和暗光条件下阀门

实验过程：

DeepLab V1 设置

项目	设置
数据集	亮光、暗光数据集
DCNN 模型	权重采用预训练的 VGG16
损失函数	Softmax Cross Entropy
训练器	SGD，Batch=2
学习率	0.001

训练条件：Tensorflow-gpu 1.4.0+Python3.5+Cuda8+windows10
数据集分为：亮光下的水下阀门和暗光下的水下阀门，其中，亮光下的阀门较清晰，识别比较容易，而暗光下的阀门则不然，不仅整体偏暗，而且还有一些明显的激光线，对模型训练和识别有着较大的影响

1. **选择模型**，部署到本机
   配置安装环境，安装 Tensorflow-gpu==1.4.0 和 Python3.5，由于最新版本的 Python 不支持 DeeplabV1，所以重新安装的 Python3.5.安装了对应版本的 CUDA8.0.最后成功运行

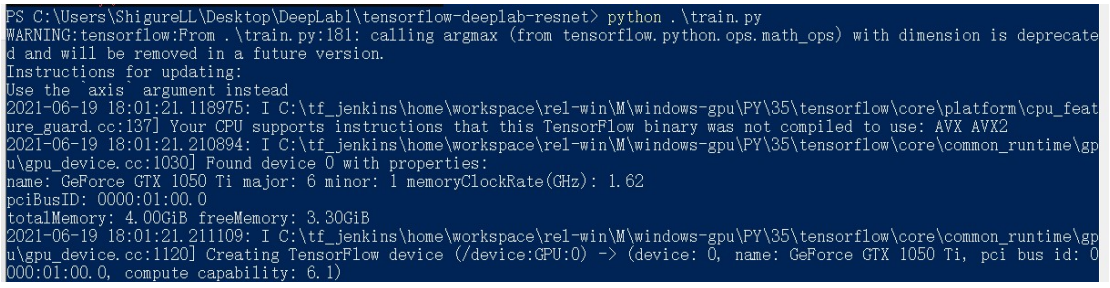

图 1　运行成功得到设备的信息

2. 将训练集的 label **转换成合适的格式**
   原始训练集使用的是 VOC2012 的 RGB 格式 MASK，黑色，红色，绿色分别代表背景，阀门和管道（图 2），但 deeplab 使用的 label 是需要转换的（图 3），使用以下代码即可实现格式转换。

图 2 MASK 图像

图 3 Deeplab 的 MASK 图像

```
def convert_from_color_segmentation(arr_3d):
    arr_2d = np.zeros((arr_3d.shape[0], arr_3d.shape[1]), dtype=np.uint8)
    for c, i in palette.items():
        m = np.all(arr_3d == np.array(c).reshape(1, 1, 3), axis=2)
        arr_2d[m] = i

    return arr_2d
```

代码 1.1 转换格式

3. **训练得到参数**，使用 evaluate.py 计算得到 Mean IoT
   亮光和暗光分别为 89.4%和 63.8%
   可以得出在亮光条件下使用该模型能够得到比较好的分割结果，暗光条件下分割效果不是很好。于是我将重点放到了暗光条件下的图像分割。
   在这里有两个选择，修改模型参数或更改模型，或者是对图像进行增强。由于 DeeplabV1 比较落后，很多函数属性与部分库不兼容，于是在这一步进展十分缓慢，也尝试修改过 loss 函数，无一例外都失败了，只好从图像入手。
4. **为了验证修改图像来提高分割准确度的可行性，我先使用简单的方法对图像进行快速地处理，得到的图像进行测试**
   对图像进行伽马变换，拉普拉斯算子增强和线性变换增强后，得到的结果并不好，甚至没能达到 50%以上，考虑水下环境干扰因素过多，因此不能简单的提高图像的亮度，于是通过查阅资料，我使用“暗通道去雾”的算法进行处理，（图 4）去雾算法是针对大气中的雾的，水中的光线传播模型和大气中有相似之处，因此首先想到的是根据物理模型实现图像增强。

图 4 原图（左），去雾（右）

多次调整参数后，发现去雾后效果不明显。于是使用其他的同类算法，其中“自适应色阶和对比度”算法得到的结果比之前更好。（图 5）

图 5 原图（左） 自适应色阶和对比度（右）

5. **尝试对暗光条件下的图像中的激光部分进行处理，**光对图像分割识别等影响很大，于是我想到可以将图像中的光去掉，或者平衡一下整体的亮度。
   首先是将图像中最亮的部分选出，通过调整阈值可以得到图像中的光线，再将去掉的部分使用图像修复技术尝试修复。
   a） 利用 opencv 中的阈值函数，得到图中光线的 mask 区域。（图 6）

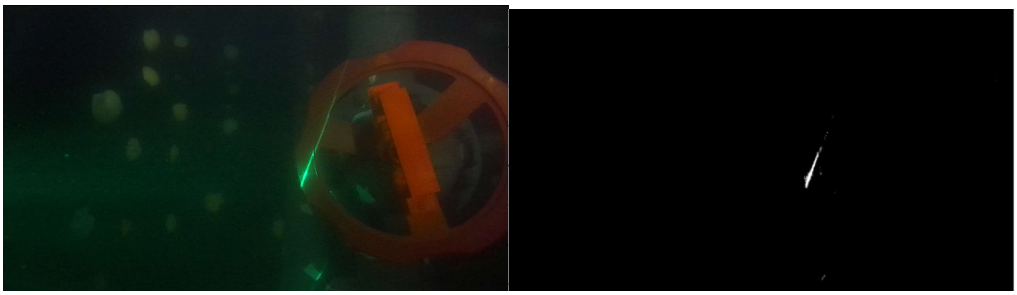

图 6 原图像（左） 光线（右）

B） 使用形态学图形学将该区域膨胀，利用 inpaint 方法，将 mask 区域中的图像进行修复。最终得到修复后的图像。(图 7)

图 6 原图（左） 去高光（右）

将处理后的图像进行测试，再一次得到 MeanIoT：65.80%

比一开始的 63.8%提高的一点。

6. 使用去高光的数据集训练，将训练集的所有图像进行去高光处理，再次进行训练

训练完成后再次运行 evaluate.py 计算 Mean IoT：72.70%

比一开始的 63.8%提高了 8.9%

实验结果：

训练集	操作	测试集	操作	Mean IoT	其他情况低于50%	
亮光	无	亮光	无	89.40%		
亮光	无	暗光	无	54.90%		
亮光	无	暗光	拉普拉斯算子增强	59%		
亮光	无	暗光	线性变换增强	61.30%		
暗光	无	暗光	无	63.80%		
暗光	无	暗光	去除高光	65.80%		
暗光	去除高光	暗光	无	72.60%		
暗光	去除高光	暗光	去除高光	72.70%		8.90%

图 7 各个实验结果

下一步计划方向：

McGlamery 和 Jaffe 的研究表明, 在图像平面的一个点上的反射模型主要有三个组成部分: 直接照射分量，前向照射分量和后向散射分量。(图 8)

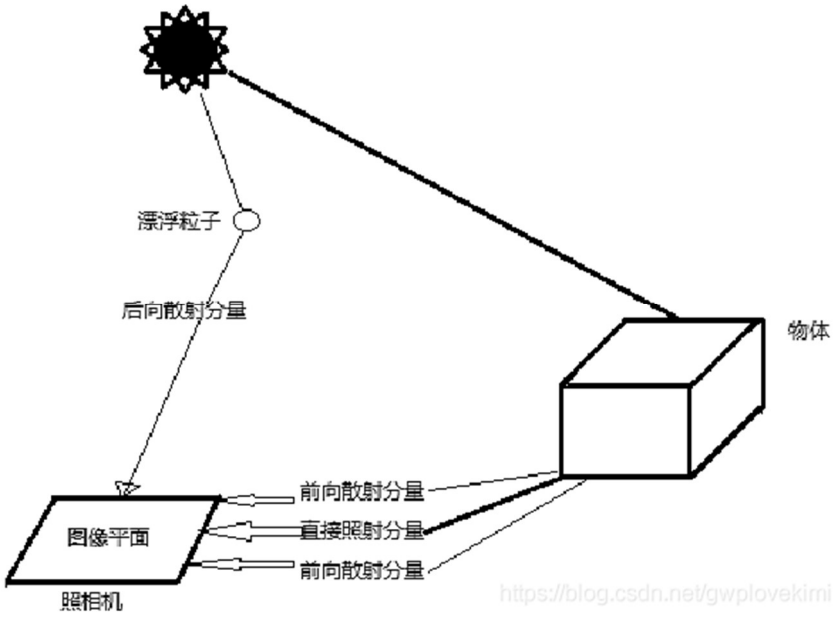

图 8 水下散射模型

在本次实验环境中，直接照射分量是阀门和管道直接反射到相机的光的分量，

$$E_D(x) = J(x)e^{-\beta d(x)} = J(x)t(x) \qquad (2\text{-}1)$$

$J(x)$目标场景的直接反射部分，β为光在水中的衰减系数，$d(x)$表示场景深度，指数项$e^{-\beta d(x)}$也就是 $t(x)$为光在水中的透射率。

受到水中杂质等影响，光线进入镜头时会与微粒碰撞发生散射，这是前向散射。由已知实验可以确定它的影响可以通过点扩散函数来近似，，点扩散函数受图像平面和物体之间的距离的影响。水下分割样本中，物体与镜头距离较近，收到前向散射影响有限。

后向散射是由于人造光撞击水粒子，并反射回相机。显然在暗光数据集中几乎所有图片都受到激光的影响。后向散射的数学公式如下:

$$E_{BS}(x) = B_\infty(x)(1 - e^{-\beta d(x)}) \qquad (2\text{-}2)$$

$B_\infty(x)$为反向散射光的颜色矢量。

图 9 实际图像 暗光（左）亮光（右）

由公式（2-2）结合实际图像分析，激光颜色矢量为绿色，经后向散射使得镜头接收的到光偏绿色，这也导致去掉激光线后周围依旧是偏绿，如果忽略前向散射部分，水下散射模型可

以被简化为下式：

$$I(x) = J(x)e^{-\beta d(x)} + B_{\infty}(x)(1 - e^{-\beta d(x)}) \qquad (2\text{-}3)$$

此公式与大气散射模型相似，根据上式与图 9 图像分析：亮光条件激光影响几乎可以忽略，式 2-3 中，如果某一点只受直接照射分量，那后向散射对图像的影响可以忽略；暗光下直接照射分量太小不足以抵消激光的影响。在一定程度上，该公式可以解释此次实验的水下数据集。

下一步计划是考虑如何通过现有的水下光散射及漫反射物理模型，对激光的漫反射部分进行处理。使得激光对图像的影响降到最小。

实验总结：

结论：光线对图像分割的影响：

在暗光下，在本次实验中影响最大的光是激光（图 8），首先图像的 label 中并没有排除激光的影响，或者说水下激光因为折射漫反射等原因，影响到了相邻区域，并最终影响图像分割

但是通过对图像中的激光进行处理，如去除激光后再次分割，分割的准确率确实能提升，原因可能是在卷积时激光是高频信号，更容易保留在 feature map 中，不断累积，最终影响分割。去除激光后有提升，可能是去掉了高频信号，最后更少的信息保留在 feature map 中，但进步空间有限，可能是仅处理了激光的部分，但并没有完全消除漫反射的光，图像复原时周围的漫反射影响了修复，此时再进行图像增强（伽马变换，分段线性变换等），漫反射的光亮度会提升，影响其他区域。

因此，在这次实验中一开始使用去雾算法，自适应色阶和对比度等效果不好，最大的原因是激光影响较大，大气中光的漫反射有物理方程，现阶段也有研究用于解决大气中的光影响，水中的光和大气中在物理上有相似性，如果能去除激光及其漫反射的光，再次使用图像增强的方法，还原物体本来的颜色，最终的分割效果或许会更好。

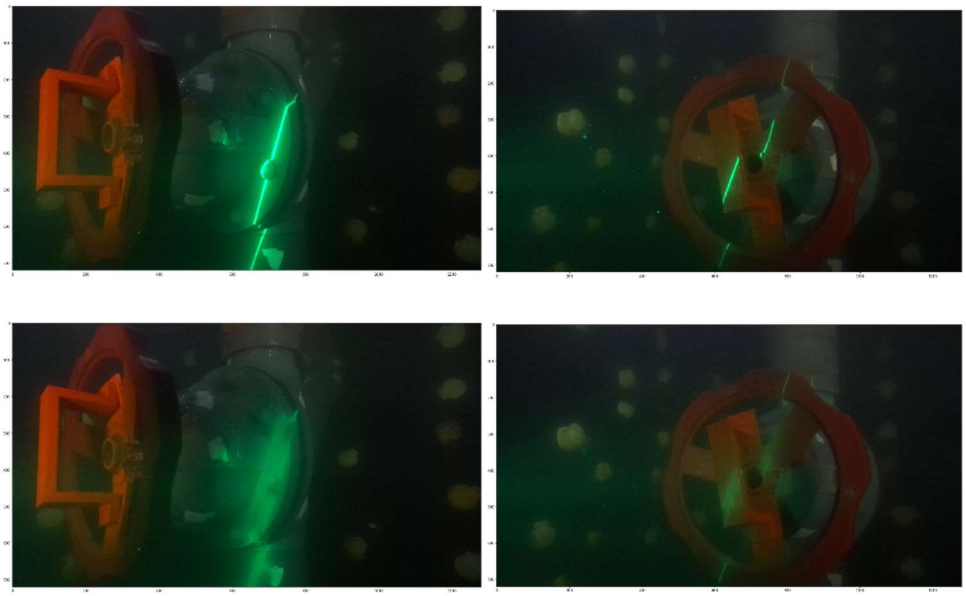

图 8

1. 为什么使用该模型：
    a) 第一是设备限制，本机由于配置原因只能运行要求比较低的模型，在训练时也降低了训练规模。在尝试运行了多个模型后最终成功运行了该模型。之后的实验也是使用了这个模型
    b) DeepLab 是结合了深度卷积神经网络（DCNNs）和概率图模型（DenseCRFs）的方法。在实验中发现 DCNNs 做语义分割时精准度不够的问题，根本原因是 DCNNs 的高级特征的平移不变性（即高层次特征映射）。DeepLab 解决这一问题的方法是通过将 DCNNs 层的响应和完全连接的条件随机场（CRF）结合。同时模型创新性的将 Hole（即空洞卷积）算法应用到 DCNNs 模型上，在现代 GPU 上运行速度达到了 8FPS。
    c) 速度更快，训练时间更短

2. 遇到的问题及解决方法
    a） 一开始我选择的是 FaceBook 的 Detectron2 框架进行实验，但是需要 COCO 格式的数据集，而实验提供的数据集是 VOC 格式，前者是包含坐标点构成的多边形的 json 格式标签，后者是 RGB 的图像。需要将该格式 label 转换成 COCO，而找遍全网也没有转换的方法，只好按照原理写了个方法。成功将 rgb 的 mask 转换成 COCO。
    b） 即使数据集没有问题，训练时又出现各种问题，比如网络不稳定，Google 的 colab 实验室限制了显卡的使用，使训练速度大大降低。最后无法完成训练。解决方法就是更换了模型。使用本机的显卡加速训练。

心得：这次实验最大的感受是不知道从何处下手，模型不知道如何选择，图像分割的模型有 DeepLab，FCN，VGG 等，模型选择是一个问题，这一次也是边做边学，课外也花了时间看网课，了解了卷积神经网络的原理和其他知识，里面的数学原理大部分在机器学习课上已经讲过，比如交叉熵损失函数，softmax 函数，以及一些基础的如前向传播，反向传播的含义等。其次是不知道如何修改参数，无论是 github 还是 stackoverflow，关于这个模型的讨论也是有限的，期间遇到过许多问题找不到答案，比如如何自定义损失函数，网上的实例与该版本不兼容，无法正常运行。最后对图像的处理，即使使用更复杂的处理方式，最后的识别率也没有到 80%以上，可能使用更好的模型，如 deeplabv3 效果可能会更好。

附录：

[1] 非均匀介质传播中的图像复原方法研究_陈田田
[2] 基于颜色补偿的水下图像增强_温伟清
[3] S. K. Nayar and S. G. Narasimhan, "Vision in bad weather," in IEEE Int'l Conf. Computer Vision, 1999.

#将 mask 图像转换成 deeplab 中的 label 图像

```
def convert_from_color_segmentation(arr_3d):
    arr_2d = np.zeros((arr_3d.shape[0], arr_3d.shape[1]), dtype=np.uint
8)
    for c, i in palette.items():
        m = np.all(arr_3d == np.array(c).reshape(1, 1, 3), axis=2)
        arr_2d[m] = i

    return arr_2d
for l_f in tqdm(label_files):
    arr = np.array(Image.open(label_dir + l_f).convert("RGB"))
    arr_2d = convert_from_color_segmentation(arr)
    Image.fromarray(arr_2d).save(new_label_dir + l_f)
```

#将 RGB 的 MASK 图像转换成 COCO 数据集格式

```
def close_contour(contour):
    if not np.array_equal(contour[0], contour[-1]):
        contour = np.vstack((contour, contour[0]))
    return contour
def create_sub_masks(mask_image):#将三个子 mask 分离出来
    width, height = mask_image.size

    # Initialize a dictionary of sub-masks indexed by RGB colors
    sub_masks = {}
    for x in range(width):
        for y in range(height):
            pixel = mask_image.getpixel((x,y))[:3]
            pixel_str = str(pixel)
            sub_mask = sub_masks.get(pixel_str)
            if sub_mask is None:

                sub_masks[pixel_str] = Image.new('1', (width+2, height+
2))

            sub_masks[pixel_str].putpixel((x+1, y+1), 1)

    maps = {"(0, 0, 0)":"background","(0, 128, 0)":"pipe","(128, 0, 0)"
:"valve"}
    for v in list(sub_masks.keys()):
        sub_masks[maps[v]] = sub_masks.pop(v)
```

```
    return sub_masks
def mask2polygon(mask):
    contours, hierarchy = cv2.findContours((mask).astype(np.uint8), cv2
.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    # mask_new, contours, hierarchy = cv2.findContours((mask).astype(np
.uint8), cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    segmentation = []
    for contour in contours:
        contour_list = contour.flatten().tolist()
        if len(contour_list) > 4:# and cv2.contourArea(contour)>10000
            segmentation.append(contour_list)
    return segmentation
def bounding_box(img):
    #计算包围盒
    rows = np.any(img, axis=1)
    cols = np.any(img, axis=0)
    rmin, rmax = np.where(rows)[0][[0, -1]]
    cmin, cmax = np.where(cols)[0][[0, -1]]

    return rmin, rmax, cmin, cmax
def countArea(box):
    #计算 area
    return abs((box[1]-box[0])*(box[3]-box[2]))
```

#去掉图像的高光部分

```
#找亮光位置
def create_mask(img):
    _, mask = cv2.threshold(img, 100, 255, cv2.THRESH_BINARY)
    return mask
#修复图片
def restore(img,gray):
    src_ = img
    mask = create_mask(gray)

    res_s = cv2.resize(src_,None,fx=0.6, fy=0.6, interpolation = cv2.IN
TER_CUBIC)
    mask_s = cv2.resize(mask,None,fx=0.6, fy=0.6, interpolation = cv2.I
NTER_CUBIC)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15,20))
    kernel0 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))
    mask2 = cv2.morphologyEx(mask,cv2.MORPH_OPEN,kernel0,iterations=1)

    mask1 = cv2.dilate(mask,kernel,iterations=3)
```

```
    contours, hierarchy = cv2.findContours(mask2,cv2.RETR_TREE,cv2.CHAI
N_APPROX_SIMPLE)
    cont = cv2.drawContours(img.copy(),contours,-1,(0,0,255),3)
    #bounds = cv2.boundingRect(mask)

    mask2 = np.zeros((img.shape[0],img.shape[1])).astype(np.uint8) * 25
5
    pts = []
    for contour in contours:
        if len(contour) < 5 :
            continue
        bound_circle = cv2.fitEllipse(contour)
        bound_pts = cv2.boxPoints(bound_circle).astype(np.int32)
        pts.append(bound_pts)
    m2 = cv2.fillPoly(mask2.copy(), pts,255)

    mask3 = cv2.dilate(m2.copy(),kernel,iterations=3)



    img3 = cv2.cvtColor(cont, cv2.COLOR_BGR2RGB)

#     for x in range(bounds[1],bounds[1]+bounds[3]):
#         for y in range(bounds[0],bounds[0]+bounds[2]):
#             mask2[x,y] = 255

    dst = cv2.inpaint(img, mask1, 15,flags=cv2.INPAINT_TELEA)

    figure(figsize=(30,15))
    subplot(1,2,1)
    imshow(mask1)
    subplot(1,2,2)
    imshow(mask)
    return dst

for im in imgs:
    img = cv2.imread("./imgs/"+im)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    res = restore(img,gray)
    img_Guassian = cv2.GaussianBlur(res,(5,5),0)
    cv2.imwrite("./results/"+im,img_Guassian)
```

#模型训练代码

```
from __future__ import print_function
```

```
import argparse
from datetime import datetime
import os
import sys
import time

import tensorflow as tf
import numpy as np

from deeplab_resnet import DeepLabResNetModel, ImageReader, decode_labe
ls, inv_preprocess, prepare_label

IMG_MEAN = np.array((104.00698793,116.66876762,122.67891434), dtype=np.
float32)

BATCH_SIZE = 2
DATA_DIRECTORY = './dataset'
DATA_LIST_PATH = './dataset/train_dark.txt'
IGNORE_LABEL = 255
INPUT_SIZE = '256,256'
LEARNING_RATE = 1e-4
MOMENTUM = 0.9
NUM_CLASSES = 3
NUM_STEPS = 2000
POWER = 0.9
RANDOM_SEED = 1234
RESTORE_FROM = './deeplab_resnet.ckpt'
SAVE_NUM_IMAGES = 2
SAVE_PRED_EVERY = 10
SNAPSHOT_DIR = './snapshots/'
WEIGHT_DECAY = 0.0005
def save(saver, sess, logdir, step):
   '''Save weights.

   Args:
     saver: TensorFlow Saver object.
     sess: TensorFlow session.
     logdir: path to the snapshots directory.
     step: current training step.
   '''
   model_name = 'model.ckpt'
   checkpoint_path = os.path.join(logdir, model_name)

   if not os.path.exists(logdir):
```

```
      os.makedirs(logdir)
   saver.save(sess, checkpoint_path, global_step=step)
   print('The checkpoint has been created.')

def load(saver, sess, ckpt_path):
    '''Load trained weights.

    Args:
      saver: TensorFlow Saver object.
      sess: TensorFlow session.
      ckpt_path: path to checkpoint file with parameters.
    '''
    saver.restore(sess, ckpt_path)
    print("Restored model parameters from {}".format(ckpt_path))

def main():
    """Create the model and start the training."""
    args = get_arguments()

    h, w = map(int, args.input_size.split(','))
    input_size = (h, w)

    tf.set_random_seed(args.random_seed)

    # Create queue coordinator.
    coord = tf.train.Coordinator()

    # Load reader.
    with tf.name_scope("create_inputs"):
        reader = ImageReader(
            args.data_dir,
            args.data_list,
            input_size,
            args.random_scale,
            args.random_mirror,
            args.ignore_label,
            IMG_MEAN,
            coord)
        image_batch, label_batch = reader.dequeue(args.batch_size)

    # Create network.
    net = DeepLabResNetModel({'data': image_batch}, is_training=args.is
_training, num_classes=args.num_classes)
    # For a small batch size, it is better to keep
```

```
    # the statistics of the BN layers (running means and variances)
    # frozen, and to not update the values provided by the pre-
trained model.
    # If is_training=True, the statistics will be updated during the tr
aining.
    # Note that is_training=False still updates BN parameters gamma (sc
ale) and beta (offset)
    # if they are presented in var_list of the optimiser definition.

    # Predictions.
    raw_output = net.layers['fc1_voc12']
    # Which variables to load. Running means and variances are not trai
nable,
    # thus all_variables() should be restored.
    restore_var = [v for v in tf.global_variables() if 'fc' not in v.na
me or not args.not_restore_last]
    all_trainable = [v for v in tf.trainable_variables() if 'beta' not
in v.name and 'gamma' not in v.name]
    fc_trainable = [v for v in all_trainable if 'fc' in v.name]
    conv_trainable = [v for v in all_trainable if 'fc' not in v.name] #
 lr * 1.0
    fc_w_trainable = [v for v in fc_trainable if 'weights' in v.name] #
 lr * 10.0
    fc_b_trainable = [v for v in fc_trainable if 'biases' in v.name] #
lr * 20.0
    assert(len(all_trainable) == len(fc_trainable) + len(conv_trainable
))
    assert(len(fc_trainable) == len(fc_w_trainable) + len(fc_b_trainabl
e))

    # Predictions: ignoring all predictions with labels greater or equa
l than n_classes
    raw_prediction = tf.reshape(raw_output, [-1, args.num_classes])
    label_proc = prepare_label(label_batch, tf.stack(raw_output.get_sha
pe()[1:3]), num_classes=args.num_classes, one_hot=False) # [batch_size,
 h, w]
    raw_gt = tf.reshape(label_proc, [-1,])
    indices = tf.squeeze(tf.where(tf.less_equal(raw_gt, args.num_classe
s - 1)), 1)
    gt = tf.cast(tf.gather(raw_gt, indices), tf.int32)
    prediction = tf.gather(raw_prediction, indices)

    # Pixel-wise softmax loss.
```

```
    loss = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=prediction,labels=gt)

    l2_losses = [args.weight_decay * tf.nn.l2_loss(v) for v in tf.trainable_variables() if 'weights' in v.name]
    reduced_loss = tf.reduce_mean(loss) + tf.add_n(l2_losses)

    # Processed predictions: for visualisation.
    raw_output_up = tf.image.resize_bilinear(raw_output, tf.shape(image_batch)[1:3,])
    raw_output_up = tf.argmax(raw_output_up, dimension=3)
    pred = tf.expand_dims(raw_output_up, dim=3)

    # Image summary.
    images_summary = tf.py_func(inv_preprocess, [image_batch, args.save_num_images, IMG_MEAN], tf.uint8)
    labels_summary = tf.py_func(decode_labels, [label_batch, args.save_num_images, args.num_classes], tf.uint8)
    preds_summary = tf.py_func(decode_labels, [pred, args.save_num_images, args.num_classes], tf.uint8)

    total_summary = tf.summary.image('images',
                                     tf.concat(axis=2, values=[images_summary, labels_summary, preds_summary]),
                                     max_outputs=args.save_num_images) # Concatenate row-wise.
    summary_writer = tf.summary.FileWriter(args.snapshot_dir,
                                           graph=tf.get_default_graph())

    # Define loss and optimisation parameters.
    base_lr = tf.constant(args.learning_rate)
    step_ph = tf.placeholder(dtype=tf.float32, shape=())
    learning_rate = tf.scalar_mul(base_lr, tf.pow((1 - step_ph / args.num_steps), args.power))

    opt_conv = tf.train.MomentumOptimizer(learning_rate, args.momentum)
    opt_fc_w = tf.train.MomentumOptimizer(learning_rate * 10.0, args.momentum)
    opt_fc_b = tf.train.MomentumOptimizer(learning_rate * 20.0, args.momentum)

    grads = tf.gradients(reduced_loss, conv_trainable + fc_w_trainable + fc_b_trainable)
```

```
    grads_conv = grads[:len(conv_trainable)]
    grads_fc_w = grads[len(conv_trainable) : (len(conv_trainable) + len
(fc_w_trainable))]
    grads_fc_b = grads[(len(conv_trainable) + len(fc_w_trainable)):]

    train_op_conv = opt_conv.apply_gradients(zip(grads_conv, conv_train
able))
    train_op_fc_w = opt_fc_w.apply_gradients(zip(grads_fc_w, fc_w_train
able))
    train_op_fc_b = opt_fc_b.apply_gradients(zip(grads_fc_b, fc_b_train
able))

    train_op = tf.group(train_op_conv, train_op_fc_w, train_op_fc_b)


    # Set up tf session and initialize variables.
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    sess = tf.Session(config=config)
    init = tf.global_variables_initializer()

    sess.run(init)

    # Saver for storing checkpoints of the model.
    saver = tf.train.Saver(var_list=tf.global_variables(), max_to_keep=
10)

    # Load variables if the checkpoint is provided.
    if args.restore_from is not None:
        loader = tf.train.Saver(var_list=restore_var)
        load(loader, sess, args.restore_from)

    # Start queue threads.
    threads = tf.train.start_queue_runners(coord=coord, sess=sess)

    # Iterate over training steps.
    for step in range(args.num_steps):
        start_time = time.time()
        feed_dict = { step_ph : step }
        if step % args.save_pred_every == 0:
            loss_value, images, labels, preds, summary, _ = sess.run([r
educed_loss, image_batch, label_batch, pred, total_summary, train_op],
feed_dict=feed_dict)
            summary_writer.add_summary(summary, step)
```

```
            save(saver, sess, args.snapshot_dir, step)
        else:
            loss_value, _ = sess.run([reduced_loss, train_op], feed_dic
t=feed_dict)
        duration = time.time() - start_time
        print('step {:d} \t loss = {:.3f}, ({:.3f} sec/step)'.format(st
ep, loss_value, duration))
    coord.request_stop()
    coord.join(threads)

if __name__ == '__main__':
    main()
```