 

**智能信息系统综合实践**

**实验报告**

|  |  |
| --- | --- |
| **题 目：** | 目标检测 |
| **年 级：** | **2020级** |
| **专 业：** | **软件工程** |
| **姓 名：** | **庞晓宇** |

1. **题目（原题目）**

****

1. **解题步骤（思路+代码）**

from \_\_future\_\_ import absolute\_import  
from help\_utils import tools  
from libs.box\_utils import show\_box\_in\_tensor  
from data.io.read\_tfrecord import next\_batch  
from libs.networks import build\_whole\_network  
from libs.configs import cfgs  
from \_\_future\_\_ import print\_function  
from \_\_future\_\_ import division  
  
import tensorflow as tf  
import tensorflow.contrib.slim as slim  
import os  
import sys  
import numpy as np  
import time  
sys.path.append("../")

os.environ["CUDA\_VISIBLE\_DEVICES"] = cfgs.GPU\_GROUP

def train():  
  
 faster\_rcnn = build\_whole\_network.DetectionNetwork(base\_network\_name=cfgs.NET\_NAME,  
 is\_training=True)  
  
 with tf.name\_scope('get\_batch'):  
 img\_name\_batch, img\_batch, gtboxes\_and\_label\_batch, num\_objects\_batch = \  
 next\_batch(dataset\_name=cfgs.DATASET\_NAME, # 'pascal', 'coco'  
 batch\_size=cfgs.BATCH\_SIZE,  
 shortside\_len=cfgs.IMG\_SHORT\_SIDE\_LEN,  
 is\_training=True)  
 gtboxes\_and\_label = tf.reshape(gtboxes\_and\_label\_batch, [-1, 5])  
  
 biases\_regularizer = tf.no\_regularizer  
 weights\_regularizer = tf.contrib.layers.l2\_regularizer(cfgs.WEIGHT\_DECAY)  
  
 # list as many types of layers as possible, even if they are not used now  
 with slim.arg\_scope([slim.conv2d, slim.conv2d\_in\_plane,  
 slim.conv2d\_transpose, slim.separable\_conv2d, slim.fully\_connected],  
 weights\_regularizer=weights\_regularizer,  
 biases\_regularizer=biases\_regularizer,  
 biases\_initializer=tf.constant\_initializer(0.0)):  
 final\_bbox, final\_scores, final\_category, loss\_dict = faster\_rcnn.build\_whole\_detection\_network(  
 input\_img\_batch=img\_batch,  
 gtboxes\_batch=gtboxes\_and\_label)  
  
 # ----------------------------------------------------------------------------------------------------build loss  
 # weight\_decay\_loss = tf.add\_n(slim.losses.get\_regularization\_losses())  
 # weight\_decay\_loss = tf.add\_n(tf.losses.get\_regularization\_losses())  
 rpn\_location\_loss = loss\_dict['rpn\_loc\_loss']  
 rpn\_cls\_loss = loss\_dict['rpn\_cls\_loss']  
 rpn\_total\_loss = rpn\_location\_loss + rpn\_cls\_loss  
  
 fastrcnn\_cls\_loss = loss\_dict['fastrcnn\_cls\_loss']  
 fastrcnn\_loc\_loss = loss\_dict['fastrcnn\_loc\_loss']  
 fastrcnn\_total\_loss = fastrcnn\_cls\_loss + fastrcnn\_loc\_loss  
  
 total\_loss = rpn\_total\_loss + fastrcnn\_total\_loss  
 # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_build loss  
  
 # ---------------------------------------------------------------------------------------------------add summary  
 tf.summary.scalar('RPN\_LOSS/cls\_loss', rpn\_cls\_loss)  
 tf.summary.scalar('RPN\_LOSS/location\_loss', rpn\_location\_loss)  
 tf.summary.scalar('RPN\_LOSS/rpn\_total\_loss', rpn\_total\_loss)  
  
 tf.summary.scalar('FAST\_LOSS/fastrcnn\_cls\_loss', fastrcnn\_cls\_loss)  
 tf.summary.scalar('FAST\_LOSS/fastrcnn\_location\_loss', fastrcnn\_loc\_loss)  
 tf.summary.scalar('FAST\_LOSS/fastrcnn\_total\_loss', fastrcnn\_total\_loss)  
  
 tf.summary.scalar('LOSS/total\_loss', total\_loss)  
 # tf.summary.scalar('LOSS/regular\_weights', weight\_decay\_loss)  
  
 gtboxes\_in\_img = show\_box\_in\_tensor.draw\_boxes\_with\_categories(img\_batch=img\_batch,  
 boxes=gtboxes\_and\_label[:, :-1],  
 labels=gtboxes\_and\_label[:, -1])  
 if cfgs.ADD\_BOX\_IN\_TENSORBOARD:  
 detections\_in\_img = show\_box\_in\_tensor.draw\_boxes\_with\_categories\_and\_scores(img\_batch=img\_batch,  
 boxes=final\_bbox,  
 labels=final\_category,  
 scores=final\_scores)  
 tf.summary.image('Compare/final\_detection', detections\_in\_img)  
 tf.summary.image('Compare/gtboxes', gtboxes\_in\_img)  
  
 # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_add summary  
  
 global\_step = slim.get\_or\_create\_global\_step()  
 lr = tf.train.piecewise\_constant(global\_step,  
 boundaries=[np.int64(cfgs.DECAY\_STEP[0]), np.int64(  
 cfgs.DECAY\_STEP[1])],  
 values=[cfgs.LR, cfgs.LR / 10., cfgs.LR / 100.])  
 tf.summary.scalar('lr', lr)  
 optimizer = tf.train.MomentumOptimizer(lr, momentum=cfgs.MOMENTUM)  
  
 # ---------------------------------------------------------------------------------------------compute gradients  
 gradients = faster\_rcnn.get\_gradients(optimizer, total\_loss)  
  
 # enlarge\_gradients for bias  
 if cfgs.MUTILPY\_BIAS\_GRADIENT:  
 gradients = faster\_rcnn.enlarge\_gradients\_for\_bias(gradients)  
  
 if cfgs.GRADIENT\_CLIPPING\_BY\_NORM:  
 with tf.name\_scope('clip\_gradients\_YJR'):  
 gradients = slim.learning.clip\_gradient\_norms(gradients,  
 cfgs.GRADIENT\_CLIPPING\_BY\_NORM)  
 # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_compute gradients  
  
 # train\_op  
 train\_op = optimizer.apply\_gradients(grads\_and\_vars=gradients,  
 global\_step=global\_step)  
 summary\_op = tf.summary.merge\_all()  
 init\_op = tf.group(  
 tf.global\_variables\_initializer(),  
 tf.local\_variables\_initializer()  
 )  
  
 restorer, restore\_ckpt = faster\_rcnn.get\_restorer()  
 saver = tf.train.Saver(max\_to\_keep=30)  
  
 config = tf.ConfigProto()  
 config.gpu\_options.allow\_growth = True  
  
 with tf.Session(config=config) as sess:  
 sess.run(init\_op)  
 if not restorer is None:  
 restorer.restore(sess, restore\_ckpt)  
 print('restore model')  
 coord = tf.train.Coordinator()  
 threads = tf.train.start\_queue\_runners(sess, coord)  
  
 summary\_path = os.path.join(cfgs.SUMMARY\_PATH, cfgs.VERSION)  
 # tools.mkdir(summary\_path)  
 if not os.path.exists(summary\_path):  
 os.makedirs(summary\_path)  
 summary\_writer = tf.summary.FileWriter(summary\_path, graph=sess.graph)  
  
 for step in range(cfgs.MAX\_ITERATION):  
 training\_time = time.strftime(  
 '%Y-%m-%d %H:%M:%S', time.localtime(time.time()))  
  
 if step % cfgs.SHOW\_TRAIN\_INFO\_INTE != 0 and step % cfgs.SMRY\_ITER != 0:  
 \_, global\_stepnp = sess.run([train\_op, global\_step])  
  
 else:  
 if step % cfgs.SHOW\_TRAIN\_INFO\_INTE == 0 and step % cfgs.SMRY\_ITER != 0:  
 start = time.time()  
  
 \_, global\_stepnp, img\_name, rpnLocLoss, rpnClsLoss, rpnTotalLoss, \  
 fastrcnnLocLoss, fastrcnnClsLoss, fastrcnnTotalLoss, totalLoss = \  
 sess.run(  
 [train\_op, global\_step, img\_name\_batch, rpn\_location\_loss, rpn\_cls\_loss, rpn\_total\_loss,  
 fastrcnn\_loc\_loss, fastrcnn\_cls\_loss, fastrcnn\_total\_loss, total\_loss])  
  
 end = time.time()  
 print(""" {}: step{} image\_name:{} |\t  
 rpn\_loc\_loss:{} |\t rpn\_cla\_loss:{} |\t rpn\_total\_loss:{} |  
 fast\_rcnn\_loc\_loss:{} |\t fast\_rcnn\_cla\_loss:{} |\t fast\_rcnn\_total\_loss:{} |  
 total\_loss:{} |\t per\_cost\_time:{}s"""  
 .format(training\_time, global\_stepnp, str(img\_name[0]), rpnLocLoss, rpnClsLoss,  
 rpnTotalLoss, fastrcnnLocLoss, fastrcnnClsLoss, fastrcnnTotalLoss, totalLoss,  
 (end - start)))  
 else:  
 if step % cfgs.SMRY\_ITER == 0:  
 \_, global\_stepnp, summary\_str = sess.run(  
 [train\_op, global\_step, summary\_op])  
 summary\_writer.add\_summary(summary\_str, global\_stepnp)  
 summary\_writer.flush()  
  
 if (step > 0 and step % cfgs.SAVE\_WEIGHTS\_INTE == 0) or (step == cfgs.MAX\_ITERATION - 1):  
  
 save\_dir = os.path.join(cfgs.TRAINED\_CKPT, cfgs.VERSION)  
 if not os.path.exists(save\_dir):  
 os.makedirs(save\_dir)  
  
 save\_ckpt = os.path.join(  
 save\_dir, 'voc\_' + str(global\_stepnp) + 'model.ckpt')  
 saver.save(sess, save\_ckpt)  
 print(' weights had been saved')  
  
 coord.request\_stop()  
 coord.join(threads)

train()

from \_\_future\_\_ import absolute\_import  
from help\_utils import tools  
from libs.box\_utils import draw\_box\_in\_img  
from libs.networks import build\_whole\_network  
from libs.configs import cfgs  
from data.io.image\_preprocess import short\_side\_resize\_for\_inference\_data  
from \_\_future\_\_ import print\_function  
from \_\_future\_\_ import division  
  
import os  
import sys  
import tensorflow as tf  
import time  
import cv2  
import argparse  
import numpy as np  
sys.path.append("../")

def detect(det\_net, inference\_save\_path, real\_test\_imgname\_list):  
  
 # 1. preprocess img  
 img\_plac = tf.placeholder(dtype=tf.uint8, shape=[  
 None, None, 3]) # is RGB. not GBR  
 img\_batch = tf.cast(img\_plac, tf.float32)  
 img\_batch = short\_side\_resize\_for\_inference\_data(img\_tensor=img\_batch,  
 target\_shortside\_len=cfgs.IMG\_SHORT\_SIDE\_LEN,  
 length\_limitation=cfgs.IMG\_MAX\_LENGTH)  
 img\_batch = img\_batch - tf.constant(cfgs.PIXEL\_MEAN)  
 img\_batch = tf.expand\_dims(img\_batch, axis=0) # [1, None, None, 3]  
  
 detection\_boxes, detection\_scores, detection\_category = det\_net.build\_whole\_detection\_network(  
 input\_img\_batch=img\_batch,  
 gtboxes\_batch=None)  
  
 init\_op = tf.group(  
 tf.global\_variables\_initializer(),  
 tf.local\_variables\_initializer()  
 )  
  
 restorer, restore\_ckpt = det\_net.get\_restorer()  
  
 config = tf.ConfigProto()  
 config.gpu\_options.allow\_growth = True  
  
 with tf.Session(config=config) as sess:  
 sess.run(init\_op)  
 if not restorer is None:  
 restorer.restore(sess, restore\_ckpt)  
 print('restore model')  
  
 for i, a\_img\_name in enumerate(real\_test\_imgname\_list):  
  
 raw\_img = cv2.imread(a\_img\_name)  
 start = time.time()  
 resized\_img, detected\_boxes, detected\_scores, detected\_categories = \  
 sess.run(  
 [img\_batch, detection\_boxes,  
 detection\_scores, detection\_category],  
 # cv is BGR. But need RGB  
 feed\_dict={img\_plac: raw\_img[:, :, ::-1]}  
 )  
 end = time.time()  
 # print("{} cost time : {} ".format(img\_name, (end - start)))  
  
 raw\_h, raw\_w = raw\_img.shape[0], raw\_img.shape[1]  
  
 xmin, ymin, xmax, ymax = detected\_boxes[:, 0], detected\_boxes[:, 1], \  
 detected\_boxes[:, 2], detected\_boxes[:, 3]  
  
 resized\_h, resized\_w = resized\_img.shape[1], resized\_img.shape[2]  
  
 xmin = xmin \* raw\_w / resized\_w  
 xmax = xmax \* raw\_w / resized\_w  
  
 ymin = ymin \* raw\_h / resized\_h  
 ymax = ymax \* raw\_h / resized\_h  
  
 detected\_boxes = np.transpose(np.stack([xmin, ymin, xmax, ymax]))  
  
 show\_indices = detected\_scores >= cfgs.SHOW\_SCORE\_THRSHOLD  
 show\_scores = detected\_scores[show\_indices]  
 show\_boxes = detected\_boxes[show\_indices]  
 show\_categories = detected\_categories[show\_indices]  
 final\_detections = draw\_box\_in\_img.draw\_boxes\_with\_label\_and\_scores(raw\_img - np.array(cfgs.PIXEL\_MEAN),  
 boxes=show\_boxes,  
 labels=show\_categories,  
 scores=show\_scores)  
 nake\_name = a\_img\_name.split('/')[-1]  
 # print (inference\_save\_path + '/' + nake\_name)  
 cv2.imwrite(inference\_save\_path + '/' + nake\_name,  
 final\_detections[:, :, ::-1])  
  
 tools.view\_bar('{} image cost {}s'.format(  
 a\_img\_name, (end - start)), i + 1, len(real\_test\_imgname\_list))

def test(test\_dir, inference\_save\_path):  
  
 test\_imgname\_list = [os.path.join(test\_dir, img\_name) for img\_name in os.listdir(test\_dir)  
 if img\_name.endswith(('.jpg', '.png', '.jpeg', '.tif', '.tiff'))]  
 assert len(test\_imgname\_list) != 0, 'test\_dir has no imgs there.' \  
 ' Note that, we only support img format of (.jpg, .png, and .tiff) '  
  
 faster\_rcnn = build\_whole\_network.DetectionNetwork(base\_network\_name=cfgs.NET\_NAME,  
 is\_training=False)  
 detect(det\_net=faster\_rcnn, inference\_save\_path=inference\_save\_path,  
 real\_test\_imgname\_list=test\_imgname\_list)

def parse\_args():  
 """  
 Parse input arguments  
 """  
 parser = argparse.ArgumentParser(  
 description='TestImgs...U need provide the test dir')  
 parser.add\_argument('--data\_dir', dest='data\_dir',  
 help='data path',  
 default='demos', type=str)  
 parser.add\_argument('--save\_dir', dest='save\_dir',  
 help='demo imgs to save',  
 default='inference\_results', type=str)  
 parser.add\_argument('--GPU', dest='GPU',  
 help='gpu id ',  
 default='0', type=str)  
  
 if len(sys.argv) == 1:  
 parser.print\_help()  
 sys.exit(1)  
  
 args = parser.parse\_args()  
  
 return args

args = parse\_args()  
print('Called with args:')  
print(args)  
os.environ["CUDA\_VISIBLE\_DEVICES"] = args.GPU  
test(args.data\_dir,  
 inference\_save\_path=args.save\_dir)

1. **总结（心得体会）**

这次的机器学习实验是有关于对有丝分裂细胞进行检测，通过提供的数据集，使用Faster RCNN模型进行训练和测试，并得到更好的检测结果。

在这个实验中，我对Faster RCNN代码进行了一些修改以适应新的数据集。首先，我进行了数据预处理，将图像进行裁剪和大小调整，以便于输入Faster RCNN网络中进行训练。然后，在训练过程中，我使用了313张图片进行了训练，同时对模型进行了调参，以达到更好的性能。最后，在测试阶段，我使用80张图片进行了测试，并根据指标计算方法计算了AP和Recall值，以衡量模型的性能。

通过这个实验，我学到了很多有关于机器学习的知识，例如数据预处理、模型训练和测试等。我意识到对于一个良好的机器学习模型，必须要经过充分的数据处理和调参，才能得到更好的性能。另外，我也发现了，不同的指标可以反映出不同的检测效果，例如AP和Recall值，因此在实际应用中需要根据具体需求选择合适的指标。

总的来说，这个实验让我深入地理解了机器学习的原理和方法，同时也让我更加熟练地使用相关的工具和技术。这对我的未来学习和工作都将有很大的帮助。