西北大学

智能信息系统综合实践
实验报告

题　　目：　目标检测

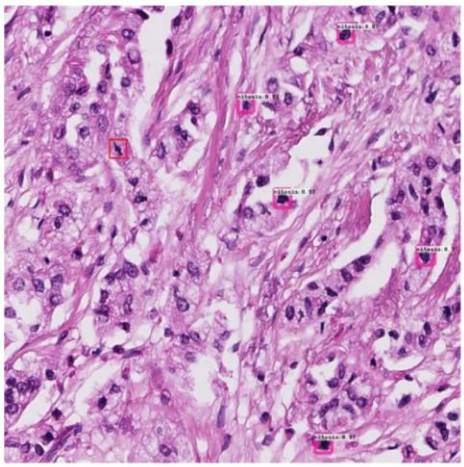
年　　级：　2020

专　　业：　软件工程

姓　　名：　何铖俊

一、题目（原题目）

有丝分裂细胞检测



提供数据集
训练集：313张图像
测试集：80张图像
标注格式：.xml，可再提供xml转为txt/csv的代码
（这三种格式几乎支持开源你的目标检测程序）

代码：
提供Faster RCNN代码（包含数据预处理、训练、测试、指标计算），tensorflow

目标：
请在测试集中达到更好的结果
AP，recall

二、解题步骤（思路+代码）

什么是目标检测？

所谓目标检测就是在一张图像中找到我们关注的目标，并确定它的类别和位置，这是计算机视觉领域最核心的问题之一。由于各类目标不同的外观，颜色，大小以及在成像时光照，遮挡等具有挑战性的问题，目标检测一直处于不断的优化和研究中。

传统的目标检测算法有：SIFT（尺度不变特征变换）、HOG（方向梯度直方图）、DPM（一种基于组件的图像检测算法）等。

基于深度学习的目标检测算法可以分为两类：二阶算法（Two Stage）和一阶算法（One Stage）。

二阶算法：先生成区域候选框，再通过卷积神经网络进行分类和回归修正。常见算法有 RCNN、SPPNet、Fast RCNN，Faster RCNN 和 RFCN 等。二阶算法检测结果更精确。

一阶算法：不生成候选框，直接在网络中提取特征来预测物体的分类和位置。常见算法有 SSD、YOLO 系列 和 RetinaNet 等。一阶算法检测速度与更快。

在了解 faster RCNN 之前，先了解下他们的前身 RCNN 和 fast RCNN

RCNN

RCNN 继承了传统目标检测的思想，将目标检测当做分类问题进行处理，先提取一系列目标的候选区域，然后对候选区域进行类。

其具体算法流程包含以下 4 步：

（1）生成候选区域：

采用一定区域候选算法（如 Selective Search）将图像分割成小区域，然后合并包含同一物体可能性高的区域作为候选区域输出，这里也需要采用一些合并策略。不同候选区域会有重合部分。

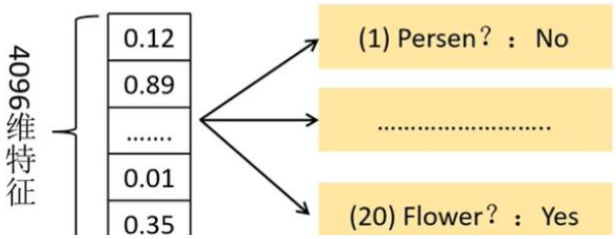
（2）对每个候选区域用 CNN 进行特征提取：

这里要事先选择一个预训练神经网络（如 AlexNet、VGG），并重新训练全连接层，即 fintune 技术的应用。

将候选区域输入训练好的 AlexNet CNN 网络，得到固定维度的特征输出。

（3）用每一类的 SVM 分类器对 CNN 的输出特征进行分类：

此处以 PASCAL VOC 数据集为例，该数据集中有 20 个类别，因此设置 20 个 SVM 分类器。



将 2000×4096 的特征与 20 个 SVM 组成的权值矩阵 4096×20 相乘，获得 2000×20 维矩阵，表示 2000 个候选区域分别属于 20 个分类的概率，因此矩阵的每一行之和为 1。

分别对上述 2000×20 维矩阵中每一列（即每一类）进行非极大值抑制剔除重叠建议框，得到该列即该类中概率最大的一些候选框。

非极大值抑制剔除重叠建议框的具体实现方法是：

第一步：定义 IoU 指数(Intersection over Union)，即 (A∩B) / (AUB) ，即 AB 的重合区域面积与 AB 总面积的比。直观上来讲 IoU 就是表示 AB 重合的比率， IoU 越大说明 AB 的重合部分占比越大，即 A 和 B 越相似。

第二步：找到每一类中 2000 个候选区域中概率最高的区域，计算其他区域与该区域的 IoU 值，删除所有 IoU 值大于阈值的候选区域。这样可以只保留少数重合率较低的候选区域，去掉重复区域。

（4）使用回归器精修候选区域的位置：

通过 Selective Search 算法得到的候选区域位置不一定准确，因此用 20 个回归器对上述 20 个类别中剩余的建议框进行回归操作，最终得到每个类别的修正后的目标区域。

RCNN 的缺点

（1）训练和测试速度慢，需要多步训练，非常繁琐。

（2）由于涉及分类中的全连接网络，因此输入 CNN 的候选区域尺寸是固定的，造成了精度的降低。

（3）候选区域需要提前提取并保存，占用的空间很大。对于非常深的网络，如VGG16,从VOC07训练集上的5000张图片上提取的特征需要数百GB的存储空间，这个问题是致命的。

RCNN 成为了当时目标检测领域的 SOAT 算法，尽管现在已经不怎么用了，但其思想仍然值得我们借鉴和学习。

fast RCNN

在RCNN之后,SPPNet解决了重复卷积计算和固定输出尺寸两个问题,SPPNet的主要贡献是在整张图像上计算全局特征图，然后对于特定的建议候选框，只需要在全局特征图上取出对应坐标的特征图就可以了。但 SPPNe 仍然存在一些弊端，如仍然需要将特征保存在磁盘中，速度还是很慢。

Fast RCNN 算法是 2015 年 Ross Girshick（还是这位大佬）提出的，在 RCNN 和 SPPNet 的基础上进行了改进。根据名字就知道，Fast RCNN 更快更强。其训练步骤实现了端到端，基于 CGG16 网络，其训练速度比 RCNN 快了 9 倍，测试速度快了 213 倍。

Fast RCNN 算法流程

（1）一张图像生成 1K~2K 个候选区域(使用 Selective Search 算法，简称 SS 算法)，我们将某个候选区域称为 ROI 区域。

（2）将图像输入网络得到相应的特征图，将 SS 算法生成的候选框投影到特征图上获得相应的特征矩阵。

R-CNN vs Fast-RCNN：

R-CNN 依次将 2000 个候选框区域输入卷积神经网络得到特征，存在大量冗余，提取时间很长。

Fast-RCNN 将整张图像送入网络，一次性计算整张图像特征，这样就可以根据特征图的坐标获得想要的候选区域的特征图，不需要重复计算。

（3）将每个特征矩阵通过 ROI pooling 层缩放到 7x7 大小的特征图。
前面讲到 RCNN 需要将候选区域归一化到固定大小，而 Fast RCNN 并不需要这样的操作。

（4）将特征图展平（reshape）为向量，通过一系列全连接层和 softmax 得到预测结果。

Fast RCNN 的缺点

1、尽管用到了 GPU，但 Region proposal 还是在 CPU 上实现的。在 CPU 中，用 SS 算法提取一张图片的候选框区域大约需要 2s，而完成整个 CNN 则只需要 0.32s，因此 Fast RCNN 计算速度的瓶颈是 Region proposal。

2、无法满足实时应用，没有真正实现端到端训练测试

Faster RCNN

Faster RCNN 是作者 Ross Girshick 继 RCNN 和 Fast RCNN 后的又一力作。同样使用 VGG16 作为网络的 backbone，推理速度在 GPU 上达到 5fps(包括候选区域的生成)，准确率也有进一步的提升。

Faster RCNN 算法流程：Faster RCNN = RPN + Fast RCNN

（RPN 是指 Region Proposal Network，建议区域生成网络。 Faster RCNN 中用 RPN 来代替了 Fast RCNN 中的 SS 算法。）

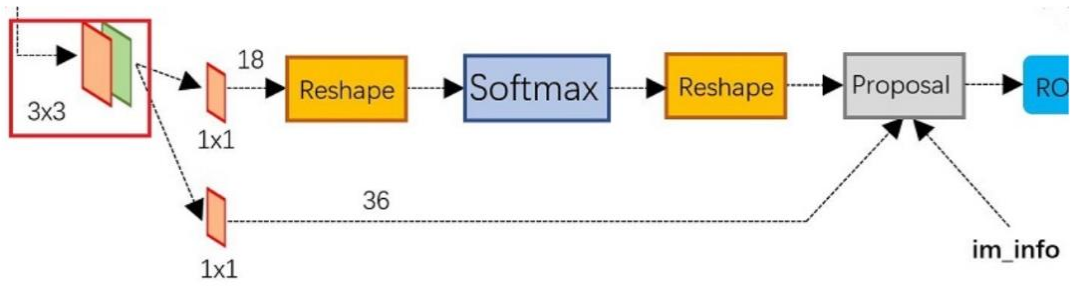
算法流程：

（1）将图像输入网络得到相应的特征图。

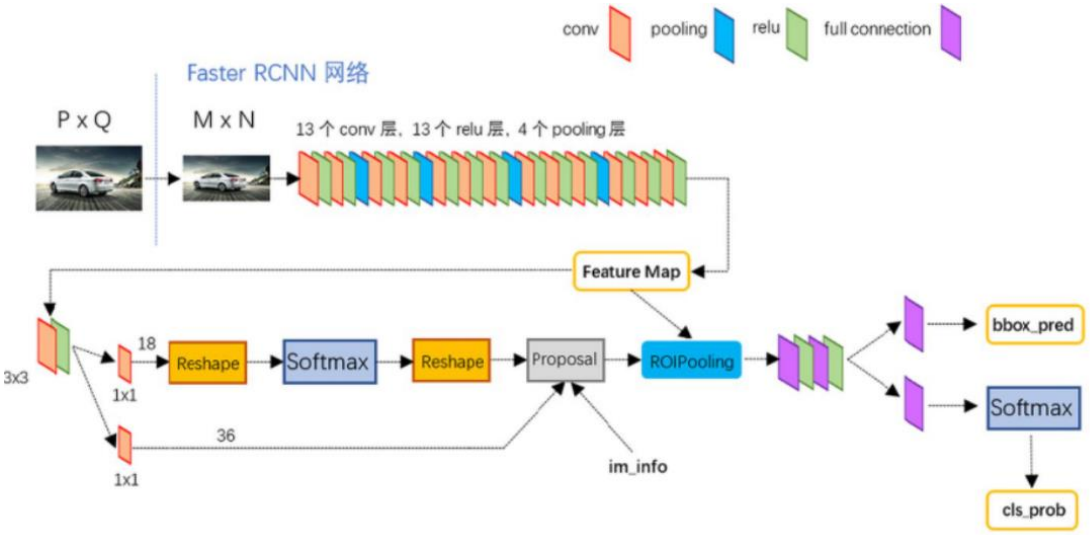
（2）使用 RPN 网络生成候选框，将 RPN 生成的候选框投影到特征图上获得 ROI 区域的特征矩阵。

（3）将每个 ROI 区域的特征矩阵通过 ROI pooling 层缩放到 7x7 大小的特征图，接着将特征图展平为 vector，之后通过一系列全连接层得到预测结果。

RPN 网络架构



Faster RCNN 网络架构



Conv layers

该 Backbone 层主要用来提取输入图像中的特征，生成 Feature Map 以供后两个模块使用。

Region Proposal Networks(RPN)

RPN 模块用来训练提取出原图中的 Region Proposal 区域，是整个网络模型中最重要的一个模块。

RoiHead

当通过 RPN 模块确定了 RP 后，就可以训练 Fast R-CNN 网络了，完成对 RP 区域的分类与 bbox 框的微调。

综上述可见，细心的人会发现，Conv layers＋RoiHead 就是 Fast R-CNN。所以，Faster R-CNN 网络实际上就是 RPN + Fast R-CNN，也就是 two-stage，训练时也是对两个模块分开训练，测试时先由 RPN 生成 RP，再将带有 RP 的 Feature Map 输入进 Fast R-CNN 中完成分类和预测框回归任务。下面，我将依次对三个模块进行详细讲解。

可以看到，从 RCNN、Fast RCNN 到 Faster RCNN，网络框架越来越简洁，目标检测效果也越来越好。

三者的优缺点比较：

	使用方法	缺点	改进
R-CNN (Region-based Convolutional Neural Networks)	1、SS提取RP； 2、CNN提取特征； 3、SVM分类； 4、BB盒回归。	1、 训练步骤繁琐（微调网络+训练SVM+训练bbox）； 2、 训练、测试均速度慢； 3、 训练占空间	1、 从DPM HSC的34.3%直接提升到了66%（mAP）； 2、 引入RP+CNN
Fast R-CNN (Fast Region-based Convolutional Neural Networks)	1、SS提取RP； 2、CNN提取特征； 3、softmax分类； 4、多任务损失函数边框回归。	1、 依旧用SS提取RP(耗时2-3s，特征提取耗时0.32s)； 2、 无法满足实时应用，没有真正实现端到端训练测试； 3、 利用了GPU，但是区域建议方法是在CPU上实现的。	1、 由66.9%提升到70%； 2、 每张图像耗时约为3s。
Faster R-CNN (Fast Region-based Convolutional Neural Networks)	1、RPN提取RP； 2、CNN提取特征； 3、softmax分类； 4、多任务损失函数边框回归。	1、 还是无法达到实时检测目标； 2、 获取region proposal，再对每个proposal分类计算量还是比较大。	1、 提高了检测精度和速度； 2、 真正实现端到端的目标检测框架； 3、 生成建议框仅需约10ms。

代码解析（课程提供代码）

Train.py

一个用于训练目标检测模型的 Python 函数。它首先创建一个 Faster R-CNN 网络，并从数据集中获取批量数据。接下来，它定义了一系列层，并使用它们来构建整个目标检测网络，该网络将图像输入作为输入，并输出目标框的位置、类别和得分等信息。

```
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.
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)
```

gtboxes_and_label_batch: Any = (next_batch(dataset_name=cfgs.DATASET_NAME, # 'pascal', 'coco'.

，它计算了它还计算了学习率，并使用动量优化器进行梯度下降训练。最后，

```
# 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)
```

```
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)
```

它记录了一些摘要信息，如损失、学习率和检测框，以便在训练期间进行可视化和监控。整个训练过程由一个 for 循环控制，每一步都会更新权重和损失，并将它们记录到 TensorBoard 中。

在这个 train.py 中调用了很多库，分别是

```
import tensorflow as tf
import tf_slim as slim
import os, sys
import numpy as np
import time
sys.path.append("../")
from libs.configs import cfgs
from libs.networks import build_whole_network
```

```
from data.io.read_tfrecord import next_batch
from libs.box_utils import show_box_in_tensor
from help_utils import tools
```

cfgs

这是一个用于目标检测的 Faster RCNN 模型的配置文件。它使用 TensorFlow 实现，包括训练和测试的配置选项。
在该配置文件中，可以设置版本号、网络名称、GPU 编号等。还可以设置一些路径，如输出目录、测试结果目录、保存权重目录等。
在训练过程中，可以设置一些超参数，如学习率、最大迭代次数、RPN 定位损失权重、RPN 分类损失权重、Fast RCNN 定位损失权重、Fast RCNN 分类损失权重等。还可以设置一些训练数据的预处理选项，如数据集名称、像素均值、图像短边长度等。
在网络配置方面，可以设置一些参数，如批量大小、初始化器、权重衰减等。此外，还可以设置锚框相关的参数，如基本锚框大小列表、锚框步长、锚框比例、锚框缩放因子等。
在 RPN 配置方面，可以设置一些参数，如卷积核大小、正样本 IoU 阈值、负样本 IoU 阈值、RPN 最小批量大小、RPN 正样本比例、RPN NMS IoU 阈值等。
在 Fast RCNN 配置方面，可以设置一些参数，如 ROI 池化大小、ROI 池化核大小、是否使用 dropout 等。
总之，这个配置文件提供了许多可以调整的选项，可以帮助用户优化 Faster RCNN 模型的性能。

NET_NAME：基础网络架构的名称，可以是“resnet_v1_101”或“MobilenetV2”。
RESTORE_FROM_RPN：是否从以前训练的 RPN 模型中恢复权重。
IMG_SHORT_SIDE_LEN 和 IMG_MAX_LENGTH:调整大小后输入图像的最短和最长边。
CLASS_NUM：数据集中的类数。
BASE_ANCHOR_SIZE_LIST，ANCHOR_STRIDE，ANCHOR_SCALES 和 ANCHOR_RATIOS：为生成 RPN 锚框的配置参数。
RPN_IOU_POSITIVE_THRESHOLD 和 RPN_IOU_NEGATIVE_THRESHOLD：确定 RPN 正负样本的锚框的阈值。
FAST_RCNN_NMS_IOU_THRESHOLD:在 Fast-RCNN 中进行非极大值抑制时使用的 IOU 阈值。
FAST_RCNN_NMS_MAX_BOXES_PER_CLASS：在 Fast-RCNN 的非极大值抑制期间保留每个类别的最大框数。

build_whole_network

__init__ 函数初始化了一些变量，包括基础网络的名称以及模型是否处于训练模式。
build_base_network 函数通过调用 ResNet 或 MobileNetV2 函数中的一个来构建基础网络，具体取决于提供的基础网络名称。

postprocess_fastrcnn 函数对 Faster R-CNN 模型的 Fast R-CNN 分支的输出进行后处理。具体来说，它解码预测的边界框坐标，将它们剪裁到图像边界内，并执行非最大抑制（NMS）以消除重复的检测结果。最终的输出包括预测的边界框、分数和类别。如果模型处于训练模式，则仅保留得分高于某个阈值的检测结果以供在 TensorBoard 中显示。

roi_pooling 函数实现 ROI 池化操作，用于对给定的特征图和 ROIs（感兴趣区域）进行裁剪和重采样，输出固定大小的特征图。

build_fastrcnn 函数是整个 Fast R-CNN 的主要函数，其中包含了 roi_pooling 函数，以及基于 ROIs 的分类和回归模块。

add_anchor_img_smry 函数是一个可选的辅助函数，用于将训练过程中的正负样本的 anchor 绘制在图像上进行可视化，便于检查模型的训练情况。

值得注意的是，在 Fast R-CNN 的分类和回归模块中，使用了 slim 库中的 fully_connected 函数来定义全连接层，并使用了 L2 正则化进行参数约束。同时，为了初始化权重，使用了 slim 库中的 variance_scaling_initializer 函数来对权重进行初始化。

add_roi_batch_img_smry(self, img, rois, labels)：该函数接收一个图片，一系列感兴趣区域（Regions of Interest, RoIs）和它们的标签，并在 Tensorboard 中显示这些 RoIs。具体来说，函数会将正例 RoIs 和负例 RoIs 用不同的颜色在图片上画出来，并通过 Tensorboard 显示这张图片。

build_loss(self, rpn_box_pred, rpn_bbox_targets, rpn_cls_score, rpn_labels, bbox_pred, bbox_targets, cls_score, labels)：该函数接收一些网络输出结果和标签，并计算 RPN 和 Fast R-CNN 的损失函数。具体来说，该函数会分别计算 RPN 的分类和回归损失，以及 Fast R-CNN 的分类和回归损失。其中，RPN 的分类损失使用 softmax 交叉熵，回归损失使用 smooth L1 损失;Fast R-CNN 的分类损失同样使用 softmax 交叉熵，回归损失也使用 smooth L1 损失。此外，如果在 Fast R-CNN 中使用了 OHEM（Online Hard Example Mining），则会进行一些额外的操作。

build_whole_detection_network(self,input_img_batch,gtboxes_batch)：该函数接收一批图片和它们对应的标注框，输出整个检测网络的输出结果。具体来说，该函数首先使用基础网络提取特征，并通过 RPN 产生一系列 RoIs；然后，该函数将这些 RoIs 分别送入 Fast R-CNN 中进行分类和回归，最终输出检测结果。如果是在训练阶段，还会计算网络的损失函数。

__init__(self, cfgs)：初始化函数，创建 Faster R-CNN 模型，并定义了一些变量和占位符。

build_base_network(self, input_img_batch, scope_name)：创建基础网络，该函数根据配置文件中指定的网络名称和版本创建一个 CNN 网络。

build_RPN(self, net_out, gt_boxes_batch, gt_labels_batch, gt_boxes_num)：创建 RPN（Region Proposal Network）网络，用于生成候选框。

run_RPN(self, sess, data)::在输入图像上运行 RPN 网络，返回生成的候选框。

build_fast_rcnn(self, rois, roi_scores, img_shape, feature_maps, gt_boxes_batch, gt_labels_batch, gt_boxes_num, is_training)：创建

Fast R-CNN 网络，用于检测目标。

run_fast_rcnn(self, sess, data, rois, roi_scores, img_shape)：在输入图像上运行 Fast R-CNN 网络，返回检测到的目标框。

get_restorer(self)：加载模型参数函数，用于从预训练模型或已训练模型中加载参数。

get_gradients(self, optimizer, loss)：计算梯度函数，用于计算训练时的梯度。

enlarge_gradients_for_bias(self, gradients)：放大偏置梯度函数，用于对偏置的梯度进行调整。

next_batch

```
def next_batch(dataset_name, batch_size, shortside_len, is_training):
    '''
    :return:
    img_name_batch: shape(1, 1)
    img_batch: shape:(1, new_imgH, new_imgW, C)
    gtboxes_and_label_batch: shape(1, Num_Of_objects, 5] .each row is [x1, y1, x2, y2, label]
    '''
    assert batch_size == 1, "we only support batch_size is 1.We may support large batch_size in the future"

    # if dataset_name not in ['ship', 'spacenet', 'pascal', 'coco','CXR']:
    #     raise ValueError('dataSet name must be in pascal, coco spacenet and ship')

    if is_training:
        pattern = os.path.join(cfgs.DATA_DIR, dataset_name + '_train*')
    else:
        pattern = os.path.join(cfgs.DATA_DIR, dataset_name + '_test*')

    print('tfrecord path is -->', os.path.abspath(pattern))

    filename_tensorlist = tf.compat.v1.train.match_filenames_once(pattern)

    filename_queue = tf.compat.v1.train.string_input_producer(filename_tensorlist)

    img_name, img, gtboxes_and_label, num_obs = read_and_prepocess_single_img(filename_queue, shortside_len,
```

```
print('tfrecord path is -->', os.path.abspath(pattern))

filename_tensorlist = tf.compat.v1.train.match_filenames_once(pattern)

filename_queue = tf.compat.v1.train.string_input_producer(filename_tensorlist)

img_name, img, gtboxes_and_label, num_obs = read_and_prepocess_single_img(filename_queue, shortside_len,
                                                                          is_training=is_training)
img_name_batch, img_batch, gtboxes_and_label_batch, num_obs_batch = \
    tf.train.batch(
                  [img_name, img, gtboxes_and_label, num_obs],
                  batch_size=batch_size,
                  capacity=1,
                  num_threads=1,
                  dynamic_pad=True)
return img_name_batch, img_batch, gtboxes_and_label_batch, num_obs_batch
```

它返回预处理图像及其对应的 ground truth 边界框和标签的批处理数据。该函数有四个参数：

dataset_name：表示要使用的数据集名称的字符串。
batch_size：表示每个批次中要处理的图像数量的整数。目前，该函数仅支持批量大小为 1。
shortside_len：表示缩放后图像最短边的长度的整数。函数将调整图像大小，同时保持宽高比，使较短的一边长度为 shortside_len。
is_training：表示该函数是否用于训练或测试的布尔值。

该函数首先基于 dataset_name 和 is_training 参数构建文件名模式，使用该模式匹配指定目录中的 TFRecord 文件。然后创建一个字符串输入生产者以匹配文件名，并使用 read_and_preprocess_single_img 函数从文件名队列中读取和预处理单个图像。

接下来，函数使用 tf.train.batch 函数批处理多个输入张量的实例。在这种情况下，函数将 img_name，img，gtboxes_and_label 和 num_obs 张量批处理在一起。批次大小由 batch_size 参数指定，容量和 num_threads 参数设置为 1。

最后，函数返回 img_name_batch，img_batch，gtboxes_and_label_batch 和 num_obs_batch 张量。img_name_batch 是一个形状为(1,1)的张量，表示图像文件的名称。img_batch 是一个形状为(1,new_imgH,new_imgW,C)的张量，表示具有高度 new_imgH，宽度 new_imgW 和 C 个颜色通道的预处理图像。gtboxes_and_label_batch 是一个形状为(1,Num_Of_objects,5)的张量，其中每行表示图像中一个对象的 ground truth 边界框和标签。num_obs_batch 是一个形状为(1,)的张量，表示图像中对象的数量。

show_box_in_tensor

这些是用于在图像上绘制不同详细级别的框的 TensorFlow 函数。它们都以图像批次和一组框作为输入，并返回在其上绘制了框的图像。
only_draw_boxes 函数仅绘制框本身，没有任何标签或分数。
draw_boxes_with_scores 函数绘制具有相应分数的框。
draw_boxes_with_categories 函数绘制具有相应类别标签的框。
draw_boxes_with_categories_and_scores 函数绘制具有类别标签和相应分数的框。

所有这些函数都使用 draw_box_in_img 模块中的 draw_boxes_with_label_and_scores 函数，该函数以图像张量、框、它们的标签和分数作为输入，并返回在其上绘制了框的图像张量。

请注意，所有输入张量都首先转换为 float32 并使用 tf.stop_gradient 停止它们的梯度。此外，由于 draw_boxes_with_label_and_scores 函数不是 TensorFlow 的一部分，因此使用 tf.py_func 调用它，该函数允许从 TensorFlow 中调用任意 Python 函数。最后，输出张量被重塑为匹配输入图像批次的形状。

Tools

这是两个 Python 函数。

view_bar 函数用于在终端中显示进度条，以反映处理任务的完成情况。它有三个参数：

message：字符串，表示任务的名称或描述。

num：整数，表示已完成的任务数量。

total：整数，表示总共需要完成的任务数量。

函数通过计算完成比例，并根据比例绘制相应长度的进度条，将任务的完成情况显示在终端上。此外，它还显示任务的名称和百分比，以及已完成的任务数量和总共需要完成的任务数量。最后，函数通过调用 sys.stdout.write 和 sys.stdout.flush 将信息打印到终端上。

mkdir 函数用于创建一个新目录。如果目录不存在，则函数创建它。如果目录已存在，则函数不做任何操作。函数有一个参数：

path：字符串，表示要创建的目录的路径。

函数使用 os.makedirs 函数创建目录。如果目录已存在，则 os.makedirs 函数不会覆盖它，因此函数不会删除或修改现有目录。

以上是 train.py 的代码构成和代码解析

Test.py

这是一个 Python 脚本，用于测试一个 Faster R-CNN 模型在一个图像目录上的表现。脚本导入了必要的软件包，例如 OpenCV、TensorFlow 和 NumPy。它定义了一个名为“detect”的函数，该函数对图像进行预处理，通过 Faster R-CNN 网络运行它们，并在原始图像上绘制边界框。该函数以一个 Faster R-CNN 模型、一个保存推理结果的路径和一个图像路径列表作为输入。

脚本还定义了一个“test”函数，该函数读取一个图像目录，并对每个图像调用“detect”函数。该“test”函数接受一个图像目录和一个保存推理结果的路径作为输入。此外，还有一个“parse_args”函数，该函数为脚本定义了命令行参数。

在主块中，脚本调用“parse_args”函数来检索命令行参数，设置可见的 GPU 设备，并使用指定的输入参数调用“test”函数。

总的来说，这个脚本提供了一个简单的方法，可以在一个图像目录上测试 Faster R-CNN 模型并保存结果。

```
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],
                feed_dict={img_plac: raw_img[:, :, ::-1]}  # cv is BGR. But need RGB
            )
        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))
```

detect 函数其作用是使用 Faster R-CNN 模型对一组图片进行目标检测，并将检测结果保存到指定的路径。函数的输入参数包括 Faster R-CNN 模型对象、保存推理结果的路径、待检测图片的文件路径列表。

具体实现上，该函数首先对待检测图片进行预处理，并将其输入到 Faster R-CNN 网络中进行推理。推理完成后，将检测结果从缩放后的图片空间映射回原始图片空间，并将结果绘制在原始图片上。最终，将带有检测结果的图片保存到指定的路径中。

该函数的实现基于 TensorFlow 框架，并依赖于 OpenCV 和 NumPy 等第三方库。具体地，该函数使用 TensorFlow 构建了 Faster R-CNN 模型的推理图，使用 OpenCV 加载和保存图片，并使用 NumPy 对图片和检测结果进行处理。

整个函数的实现过程可以分为以下几个步骤：

1、预处理待检测图片，并将其输入到 Faster R-CNN 网络中进行推理；

2、将缩放后的检测结果映射回原始图片空间；

3、将检测结果绘制在原始图片上；

4、保存带有检测结果的图片到指定的路径中。

该函数主要用于测试 Faster R-CNN 模型在给定数据集上的性能，可以方便地进行推理和结果可视化。

```
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)
```

Test 函数

这个函数是整个脚本的主函数，主要实现的是调用 "detect" 函数对指定目录中的图片进行 Faster R-CNN 模型的测试，并将检测结果保存到指定的输出路径。

具体来说，该函数首先获取指定目录中所有支持的图片文件路径，然后创建一个 Faster R-CNN 模型对象（"DetectionNetwork"），并将其作为参数传递给 "detect" 函数。"detect" 函数会逐张读取图片，对每张图片进行预处理，然后将其输入到 Faster R-CNN 模型中进行检测，最后将检测结果绘制在原图上并保存到指定的输出路径。

总之，这个函数的作用就是通过调用 "detect" 函数来对指定目录中的图片进行 Faster R-CNN 模型的测试，并将检测结果保存到指定的输出路径

在 test.py 中还引入了许多库

```
from data.io.image_preprocess import
short_side_resize_for_inference_data
from libs.configs import cfgs
from libs.networks import build_whole_network
from libs.box_utils import draw_box_in_img
from help_utils import tools
```

其中 cfgs、build_whole_network、tools 已经在 train.py 中解释

short_side_resize_for_inference_data

```
def short_side_resize_for_inference_data(img_tensor, target_shortside_len, length_limitation=1200):
    img_h, img_w = tf.shape(img_tensor)[0], tf.shape(img_tensor)[1]

    new_h, new_w = tf.cond(tf.less(img_h, img_w),
                           true_fn=lambda: (target_shortside_len,
                                            max_length_limitation(target_shortside_len * img_w // img_h, length_limitation)),
                           false_fn=lambda: (max_length_limitation(target_shortside_len * img_h // img_w, length_limitation),
                                             target_shortside_len))

    img_tensor = tf.expand_dims(img_tensor, axis=0)
    img_tensor = tf.image.resize_bilinear(img_tensor, [new_h, new_w])

    img_tensor = tf.squeeze(img_tensor, axis=0)  # ensure image tensor rank is 3
    return img_tensor
```

这个函数它接受图像张量、目标短边长度和长度限制作为输入，并返回一个经过调整大小的图像张量。该函数使用 TensorFlow 操作来根据原始图像的长宽比和目标短边长度计算图像的新高度和宽度。新的高度和宽度被限制在长度限制参数定义的最大值内，以防止图像变得过大。

以下是该函数的详细步骤：

该函数接收图像张量、目标短边长度和长度限制作为输入。

使用 TensorFlow 的 tf.shape()函数获取输入图像的高度和宽度。

然后使用 TensorFlow 的 tf.cond()函数根据高度或宽度哪个更短来计算图像的新高度和宽度。

如果高度更短，则新的高度设置为目标短边长度，并且新的宽度通过将原始图像的宽高比乘以目标短边长度来计算。新的宽度被限制在长度限制参数内。

如果宽度更短，则新的宽度设置为目标短边长度，并且新的高度通过将原始图像的宽高比乘以目标短边长度来计算。新的高度被限制在长度限制参数内。

输入图像张量被扩展为 4 个维度，通过在张量的开头添加一个新维度。

使用 tf.image.resize_bilinear()函数将图像张量调整为新的高度和宽度。

收缩调整大小后的图像张量，以删除添加的维度，得到一个 3 维张量。

返回调整大小后的图像张量作为函数的输出。

计算 AP 和 recall

```
def compute_AP_Recall(ground_truth, predicted_scores):
    """
    计算平均准确率（Average Precision，AP）和召回率（Recall）

    参数：
        ground_truth: 字典类型，键为类别名，值为该类别的真实标注，格式为{类别名: [真实标注列表]}，
                      真实标注列表中的元素为正整数，代表该类别的一个样本
        predicted_scores: 字典类型，键为类别名，值为该类别的预测得分，格式为{类别名: [预测得分列表]}，
                          预测得分列表中的元素为实数，代表该类别的一个样本的预测得分

    返回值：
        AP: 字典类型，键为类别名，值为该类别的平均准确率
        Recall: 字典类型，键为类别名，值为该类别的召回率
    """

    # 计算每个类别的AP和Recall
    AP = {}
    Recall = {}
    for category, ground_truth_labels in ground_truth.items():
        predicted_scores_list = predicted_scores.get(category, [])
```

```python
# 计算每个类别的AP和Recall
AP = {}
Recall = {}
for category, ground_truth_labels in ground_truth.items():
    predicted_scores_list = predicted_scores.get(category, [])

    # 如果预测得分为空，则将AP和Recall设为0
    if not predicted_scores_list:
        AP[category] = 0
        Recall[category] = 0
        continue

    # 将预测得分从高到低排序，并记录每个样本的真实标注和预测得分
    sorted_indices = sorted(range(len(predicted_scores_list)), key=lambda k: predicted_scores_list[k], reverse=True)
    sorted_ground_truth_labels = [ground_truth_labels[i] for i in sorted_indices]
    sorted_predicted_scores = [predicted_scores_list[i] for i in sorted_indices]

    # 计算每个位置处的精确率（Precision）、召回率（Recall）和阈值（Threshold）
    num_correct_predictions = 0
    num_ground_truth_labels = len(ground_truth_labels)
    num_predicted_labels = len(predicted_scores_list)
    Precision = [0] * num_predicted_labels
    Recall = [0] * num_predicted_labels
    Threshold = [0] * num_predicted_labels
    for i in range(num_predicted_labels):
```

```python
    for i in range(num_predicted_labels):
        if sorted_ground_truth_labels[i] == 1:
            num_correct_predictions += 1
        Precision[i] = num_correct_predictions / (i + 1)
        Recall[i] = num_correct_predictions / num_ground_truth_labels
        Threshold[i] = sorted_predicted_scores[i]

    # 计算AP
    AP[category] = 0
    for i in range(num_predicted_labels):
        if i == 0 or Threshold[i] != Threshold[i - 1]:
            AP[category] += Precision[i] * Recall[i]

    # 计算Recall
    Recall[category] = Recall[num_predicted_labels - 1]

return AP, Recall
```

以上为 train.py 和 test.py 的代码解析

由于代码复杂程度较高，个人能力不足无法在有限时间内调试和补全代码，只能通过分析重要代码来学习 faster RCNN 代码的重要知识点。

四、总结（心得体会）

1、实验过程中的不足之处：代码调试能力不足；没有完整的代码结构的意识；对 faster RCNN 算法了解不够充足

2、经过这次对 faster RCNN 代码的详细分析，对算法有了一定程度的了解，希望在实验外时间可以对实验进行完善和补足。