**实验五 行人检测**

1. 实验目的
2. 掌握基于卷积神经网络的目标检测的基本原理。
3. 了解迁移学习思想，在数据样本较小的情况下学会利用预训练模型提升自身模型精度。
4. 掌握如何组织TFRecord格式数据。
5. 掌握TensorFlow Object Detection API的使用。
6. 实验要求
7. 该实验基于TensorFlow Object Detection API实现对行人的检测，环境配置为Ubuntu 16.04， TensorFlow 1.9.0。
8. 对于TensorFlow Object Detection API的详细使用教程，请自行阅读官方提供的[文档](https://github.com/tensorflow/models/tree/master/research/object_detection)（点击查看），该实验指导书仅对实现行人检测相关的操作进行介绍。
9. 该指导书未涉及到Faster R-CNN原理性及代码实现的解释，详情请参考课件并自行查阅Faster R-CNN原始论文及开源代码。
10. 实验所用工具及数据集
11. 该实验需要下载TensorFlow Object Detection API的[源码](https://github.com/tensorflow/models)（该实验使用的API源码已给出，因为涉及环境的版本问题没有使用最新版的API源码，如果使用最新版本的API源码的话，应该需要使用TensorFlow 1.12.0，并按API安装文档进行相关配置）。之后拷贝model/research/目录下的object\_detection和slim目录到为该实验创建的目录下。
12. 该实验提供一个小型行人检测数据集[TownCentre](http://www.robots.ox.ac.uk/ActiveVision/Research/Projects/2009bbenfold_headpose/project.html" \l "datasets)，该数据集包含一个视频TownCentreXVID.avi和标签文件TownCentre-groundtruth.top。其中TownCentreXVID.avi一共5 min，每1 sec包含25帧图像（1920\*1080），因此一共包含7500帧图像；TownCentre-groundtruth.top包含前4500帧图像中行人的位置信息，每一行信息组织格式如下：

personNumber, frameNumber, headValid, bodyValid, headLeft, headTop, headRight, headBottom, bodyLeft, bodyTop, bodyRight, bodyBottom

* personNumber - A unique identifier for the individual person
* frameNumber - The frame number (counted from 0)
* headValid - 1 if the head region is valid, 0 otherwise
* bodyValid - 1 if the body region is valid, 0 otherwise
* headLeft,headTop,headRight,headBottom - The head bounding box in pixels
* bodyLeft,bodyTop,bodyRight,bodyBottom - The body bounding box in pixels

对于行人检测，我们主要需要上述标红的数据。

1. 下载Faster R-CNN预训练模型，[下载地址](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md)https://github.com/tensorflow/models/blob/master/research/object\_detection/g3doc/detection\_model\_zoo.md。参考下载faster\_rcnn\_inception\_v2\_coco，也可以下载其他模型进行实验。主要需要如下文件：

| - model.ckpt.meta

| - model.ckpt.data-00000-of-00001

| - model.ckpt.index

| - pipeline.config

1. 实验的组织目录如下：

| - Pedestrian-Detection

| - object\_dection // API源码

| - slim

| - TownCentre //原始数据集

| - TownCentreXVID.avi

| - TownCentre-groundtruth.top

| - pretrained //存储预训练Faster R-CNN模型

| - Dataset

| - images //训练用图像

| - test\_images //可用于测试

| - annotations

| - json

| - label\_map.pbtxt

| - trainval.txt

1. 实验步骤与方法
2. API的准备工作

参考API的官方使用教程，首先进行Protobuf编译，API使用Protobuf对模型和训练参数进行配置。运行如下命令：

protoc object\_detection/protos/\*.proto --python\_out=.

为保证代码正确运行，将object\_detection和slim加入环境变量：

export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

***注：***如果使用最新版的API源码，需要下载COCO API源码编译，并拷贝编译后的目录到Pedestrian-Detection下：

git clone <https://github.com/cocodataset/cocoapi.git>

cd cocoapi/PythonAPI

make

cp -r pycocotools < Pedestrian-Detection的路径>

1. 数据预处理：提取视频帧与行人位置信息
2. 提供的原始数据集为视频，而TensorFlow Object Detection API需要将数据组织为TFRecord格式进行训练。在此之前，可利用opencv的VideoCapture从TownCentreXVID.avi中抽取用于训练的图像帧，将其尺寸减半，存储到Dataset/images中，参考代码extract\_towncentre.py:

**def** video2im**(**video\_name**,** train\_path**=**'./Dataset/images'**,** test\_path**=**'./Dataset/test\_images'**,** factor**=**2**):**

"""

Extracts all frames from a video and saves them as jpgs

"""

frame **=** 0

cap **=** cv2**.**VideoCapture**(**video\_name**)**

length **=** int**(**cap**.**get**(**cv2**.**CAP\_PROP\_FRAME\_COUNT**))**

**print(**'Total Frame Count:'**,** length **)**

**while** **True:**

check**,** img **=** cap**.**read**()**

**if** check**:**

**if** frame **<** 4500**:**

path **=** train\_path

**else:**

path **=** test\_path

img **=** cv2**.**resize**(**img**,** **(**1920 **//** factor**,** 1080 **//** factor**))**

cv2**.**imwrite**(**os**.**path**.**join**(**path**,** str**(**frame**)** **+** ".jpg"**),** img**)**

frame **+=** 1

**print(**'Processed: '**,**frame**)**

**else:**

**break**

cap**.**release**()**

1. 从TownCentre-groundtruth.top提取前4500帧图像的行人位置信息，保存为json文件（也可保存为xml文件，且API提供了xml文件转TFRecord的代码），示例格式如下：

**{**

"size"**:** **{**

"width"**:** "960"**,**

"depth"**:** "3"**,**

"height"**:** "540"

**},**

"object"**:** **[**

**{**

"bndbox"**:** **{**

"xmin"**:** "118.0"**,**

"ymin"**:** "385.0"**,**

"ymax"**:** "550.0"**,**

"xmax"**:** "186.0"

**},**

"name"**:** "pedestrian"

**},**

**......(多个行人位置数据)**

**],**

"filename"**:** "0.jpg"

**}**

其中’size’保存了图像的尺寸信息，’object’包含了该帧中每一个行人的位置信息，’filename’保存了该帧的图像名，代码参考extract\_GT.py。

1. 将用于训练的所有帧的文件名写入Dataset/annotation/trainval.txt文件中，即将0-4499写入到文件中，一行一个数字。
2. 为满足TensorFlow Object Detection的使用需求，为行人指定一个id和标签，将以下信息写入到Dataset/annotation/**label\_map.pbtxt**文件中：

item {

id: 1

name: ‘pedestrian’

}

1. 将上述获取的images和json文件转为TFRecord文件

images中一共包含4500张图像，其中95%用于训练，剩余图像用于验证（参考代码create\_tf\_record.py）。主要需要组织保存如下数据：

example **=** tf**.**train**.**Example**(**features**=**tf**.**train**.**Features**(**feature**={**

'image/height'**:** dataset\_util**.**int64\_feature**(**height**),**

'image/width'**:** dataset\_util**.**int64\_feature**(**width**),**

'image/filename'**:** dataset\_util**.**bytes\_feature**(**

data**[**'filename'**].**encode**(**'utf8'**)),**

'image/source\_id'**:** dataset\_util**.**bytes\_feature**(**

data**[**'filename'**].**encode**(**'utf8'**)),**

'image/key/sha256'**:** dataset\_util**.**bytes\_feature**(**key**.**encode**(**'utf8'**)),**

'image/encoded'**:** dataset\_util**.**bytes\_feature**(**encoded\_jpg**),**

'image/format'**:** dataset\_util**.**bytes\_feature**(**'jpeg'**.**encode**(**'utf8'**)),**

'image/object/bbox/xmin'**:** dataset\_util**.**float\_list\_feature**(**xmin**),**

'image/object/bbox/xmax'**:** dataset\_util**.**float\_list\_feature**(**xmax**),**

'image/object/bbox/ymin'**:** dataset\_util**.**float\_list\_feature**(**ymin**),**

'image/object/bbox/ymax'**:** dataset\_util**.**float\_list\_feature**(**ymax**),**

'image/object/class/text'**:** dataset\_util**.**bytes\_list\_feature**(**classes\_text**),**

'image/object/class/label'**:** dataset\_util**.**int64\_list\_feature**(**classes**),**

'image/object/difficult'**:** dataset\_util**.**int64\_list\_feature**(**difficult\_obj**),**

'image/object/truncated'**:** dataset\_util**.**int64\_list\_feature**(**truncated**),**

'image/object/view'**:** dataset\_util**.**bytes\_list\_feature**(**poses**),**

**}))**

其中除了前面我们保存在json文件的信息外，还包括图像像素信息及前面label\_map.pbtxt文件中保存的信息。

我们会得到train.tfrecord和val.tfrecord两个文件。

1. 修改API训练所需的管道配置文件

即前面下载的预训练模型中的pipeline.config文件，只需要修改以下几项，用于指定预训练模型所在路径，即我们自己的TFRecord数据集所在路径（建议填写绝对路径，不然可能会报错无法找到文件）：

gradient\_clipping\_by\_norm**:** 10.0

fine\_tune\_checkpoint**:** "预训练模型model.ckpt的路径"

from\_detection\_checkpoint**:** true

num\_steps**:** 200000

**}**

train\_input\_reader **{**

label\_map\_path**:** "label\_map.pbtxt的路径"

tf\_record\_input\_reader **{**

input\_path**:** "train.record的路径"

**}**

**}**

eval\_config **{**

num\_examples**:** 8000

max\_evals**:** 10

use\_moving\_averages**:** false

**}**

eval\_input\_reader **{**

label\_map\_path**:** "label\_map.pbtxt的路径"

shuffle**:** false

num\_epochs**:** 1

num\_readers**:** 1

tf\_record\_input\_reader **{**

input\_path**:** "val.record的路径"

**}**

**}**

1. 模型训练

总结前面的工作，一是数据的预处理，把数据转换为API所需的TFRecord文件格式；二是API的准备，包括预训练模型下载，API所需文件的准备（如label\_map.pbtxt文件），预训练模型**pipeline.config**文件的修改。

准备完成之后，可直接利用API进行模型的训练，训练命令如下（训练会自动保存模型结果到train目录下，按Ctrl+C随时结束训练）：

python object\_detection/train.py \  
--logtostderr \  
--pipeline\_config\_path=pipeline.config路径 \  
--train\_dir=train

***注：***如果使用最新版API，模型的训练迁移到了model\_main.py，训练命令如下：

python object\_detection/model\_main.py \

--pipeline\_config\_path= pipeline.config路径 \

--model\_dir=train \

--num\_train\_steps=训练步数 \

--sample\_1\_of\_n\_eval\_examples=1 \

--alsologtostderr

1. 模型测试
2. 模型训练完成之后，会在train目录下得到一系列模型的checkpoint文件，选取最后的文件，将其转换为TensorFlow的pb文件（可理解为包含网络结构和参数的文件），API提供了转换代码，命令如下（22220修改为你训练结束得到的数字）：

python object\_detection/export\_inference\_graph.py \  
--input\_type=image\_tensor \  
--pipeline\_config\_path=pipeline.config \  
--trained\_checkpoint\_prefix=train/model.ckpt-**22220** \  
--output\_directory=pb\_output

1. 转换之后，可利用pb\_output目录下的frozen\_inference\_graph.pb文件对训练之后的模型进行测试。导入pb文件构造网络模型示例代码如下：

frozen\_graph **=** 'frozen\_inference\_graph.pb路径'

config **=** tf**.**ConfigProto**(**allow\_soft\_placement**=True)**

detection\_graph **=** tf**.**Graph**()**

**with** detection\_graph**.**as\_default**():**

od\_graph\_def **=** tf**.**GraphDef**()**

**with** tf**.**gfile**.**FastGFile**(**frozen\_graph**,** 'rb'**)** **as** fid**:**

serialized\_graph **=** fid**.**read**()**

od\_graph\_def**.**ParseFromString**(**serialized\_graph**)**

tf**.**import\_graph\_def**(**od\_graph\_def**,** name**=**''**)**

**with** tf**.**Session**(**graph**=**detection\_graph**,** config**=**config**)** **as** sess**:**

# image\_tensor为测试图像，大小为（1，X，Y，3）

image\_tensor **=** detection\_graph**.**get\_tensor\_by\_name**(**'image\_tensor:0'**)**

# 以下为行人检测结果

detection\_boxes **=** detection\_graph**.**get\_tensor\_by\_name**(**'detection\_boxes:0'**)**

detection\_scores **=** detection\_graph**.**get\_tensor\_by\_name**(**'detection\_scores:0'**)**

detection\_classes **=** detection\_graph**.**get\_tensor\_by\_name**(**'detection\_classes:0'**)**

num\_detections **=** detection\_graph**.**get\_tensor\_by\_name**(**'num\_detections:0'**)**

构造好模型之后，可使用OpenCV读取测试图像进行测试，或直接读取视频，对每一帧进行测试并显示，也可将测试结果保存为视频，参考视频test.avi。