西北工業大學

基于深度学习的矿泉水瓶检测

专业名称：	自动化
学生姓名：	沈宝印
班级：	09011505
时间：	2018.05-2018.06

摘要

图像中的矿泉水瓶检测可以视为一个目标检测的问题。目标检测作为图像处理和计算机视觉领域中的经典课题，在交通监控、图像检索、人机交互等方面有着广泛的应用。它旨在一个静态图像（或动态视频）中检测出人们感兴趣的目标对象。传统的目标检测算法中特征提取和分类决策分开进行，对特征的要求就更加严格，在面对复杂场景的时候很难得到理想的效果。自 Hinton 教授提出深度学习理论，越来越多的研究学者尝试采用深度学习理念来解决目标检测问题，并提出了不同的模型。不同的模型应用也不尽相同，通常采用卷积神经网络来处理目标检测问题。相比于传统的目标检测算法，卷积神经网络中的特征提取和模式分类并行进行，而且随着层数的增多可以更好的处理复杂场景。本文的主要工作有：

（1）在 Google 开源的深度学习框架 tensorflow 下，使用 MobileNet 模型进行迁移学习，最终训练出可以在静态图像中检测矿泉水瓶的模型。

（2）对训练出的检测矿泉水瓶的模型进行实际测试，分析不足及改进措施。

目录

1 引言

1.1 选题背景与意义

随着社会发展，互联网渐渐走进千家万户，人们每天接收到的信息数以万计。有心理调查报告显示，人类通过视觉获得的信息占总信息量的 83%。由此可见，图像成为人们生活中最重要的数据信息，而对图像的处理成为今年来研究的热点。作为图像处理和计算机视觉领域的核心研究方向，目标检测以其广泛的应用需求，存在于各行各业，例如生物医疗、道路监控、航空航天、工业制造、文化展示等等。

传统的目标检测技术是指在给定的图像或视频中检测出人们感兴趣的目标对象，主要由图像预处理、特征提取、目标分类三部分组成。传统的目标检测算法要精确地检测复杂图像的目标，仍存在许多难点：

（1）传统目标检测技术目前没有形成一种统一有效的算法，可以针对所有类别的复杂图像。

（2）传统算法容易受外外界因素的影响，无法像人类一样读懂图片的内容，做到检测目标的精确性。

2006 年，由 Hinton 等人提出了深度学习的概念。深度学习来源于人工神经网络的研究，本质上是一种多层级、复杂的神经网络。目前深度学习已在图像处理、计算机视觉等领域取得了令人惊讶的成就，更在工程应用等方面表现出极大潜力。

1.2 研究现状

1.2.1 目标检测研究现状

简单的说，目标检测就是计算机模拟人眼通过图片获得自己感兴趣的目标。

传统的目标检测的过程一般分成以下几个步骤,

图 1-1 传统目标检测的过程

第一步:对图像的预处理。任何图像处理的算法都无法避免这一步。因为我们所获得的图像，其数量巨大，并不能保证其可以直接作为算法的输入图像。为方便后续的对图像的处理，对图像进行预处理，主要包括采用各种方法来消除原始图像的噪声和突变，统一每张图像的颜色、尺寸、光照等特征，消减无关特征加强目标特征。

第二步:特征提取。作为最关键的一步，特征的挑选直接影响到最后的结果。目前研究学者已经提出了很多特征提取的算法，例如基本的特征(颜色、纹理、形状等)，带有视觉信息的特征(边缘、角点)等等。但是还没有一种万能的方法可以处理一切图像检测问题，目前较好的特征提取算法有 Slaglan 等人提出的 GIST 特征提取方法，Lazebnik 等人提出的 SPM 方法，还有 BoW 方法。

第三步:分类检测。目前，研究学者已经提出了三种用于检测的分类器:人工神经网络、AdaBoost 分类法和 SVM 支持向量机。随着人工智能变得越来越强大，人工神经网络作为该领域的热点，也慢慢的揭开了它的神秘面纱。Adaboost 分类法在本质上属于机器学习，核心在于先训练不同的弱分类器，在集成出一个精确的最终分类器。SVM 支持向量机是一个有监督的机器学习模型，它建立在 VC 维学习理论和结构风险最小原理上，通过对学习精度和学习能力寻求折中达到分类要求。

传统的目标检测算法通常会把特征提取和分类检测分开进行。

1.2.2 深度学习研究现状

最早的神经网络算法出现在二十世纪八十年代，但由于当时的条件限制，严重制约了其发展规模，以致之后相当一段时间，工程界与学界仅仅用其作为单纯的分类器。直到 2012 年 10 月，Hinton 教授带着他的学生在著名的 ImageNet 大赛上采用深度学习模型拿到冠军，将误检测率降低到 15.3%，掀起了神经网络领域的第二次浪潮。

由于 Fine-tune 技术和 GPU 技术的发展，深度学习克服了长久以来样本需求量大的训练困难，成为神经网络领域的研究热点，并且广泛应用于各个不同的领域。

图像检测是深度学习应用最广泛的领域。在国内，百度公司将深度学习技术应用到人脸和自然图像检测领域，并推出了相应产品。目前，深度学习模型不仅提高了图像检测的精确度，而且也减少了时间人力，大大提升了图像检测的效率和准确率。

2 目标检测相关算法

2.1 基于深度学习的目标检测

深度学习的概念源于人工神经网络，是一种“深层化”的人工神经网络。

2.1.1 卷积神经网络

卷积神经网络(Convolutional Neural Networks ，CNN)模型是深度学习的一个非常成功的算法，并且能够训练多层网络结构的算法。卷积神经网络的特殊之处在于它的神经元之间的连接是非全连接的，而且有些神经元之间的连接的权值是共享的。

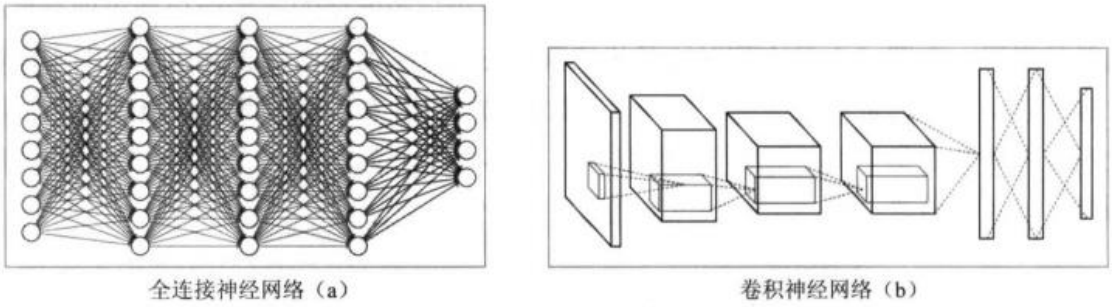


图 2-1 全连接神经网络与卷积神经网络结构示意图

使用全连接神经网络处理图像问题的最大问题在于全连接层的参数太多，参数增多除了导致计算速度减慢，还很容易导致过拟合问题，所以需要一个更合理的神经网络结构来有效地减少神经网络中的参数个数。卷积神经网络可以达到这个目的。

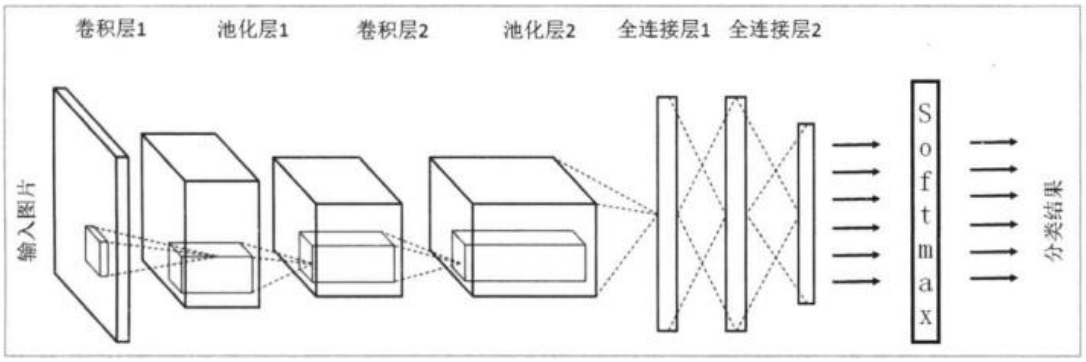


图 2-2 用于图像分类问题的一种神经网络架构图

在卷积神经网络的前几层中，每一层的节点都被组织成一个三维矩阵。一个神经网络主要由以下 5 种结构组成：

（1）输入层。输入层是整个神经网络的输入，在处理图像的卷积神经网络中，它一般代表了一张图片的像素矩阵。在图 2-2 中，最左侧的三维矩阵可以代表一张图片，其中三维矩阵的长和宽代表图像的大小，深度代表图像的色彩通道。从输入层开始，卷积神经网络通过不同的神经网络结构将上一层的三维矩阵转化为下一层的三维矩阵，直到最后的全连接层。

（2）卷积层。和传统全连接层不同，卷积层中每一个节点只是上一层神经网络的一小块，通常大小为 3*3 或 5*5。卷积层试图将神经网络中的每一小块进行更加深入的分析从而得到抽象程度更高的特征。一般来说，通过卷积层处理过的节点矩阵会变得更深。

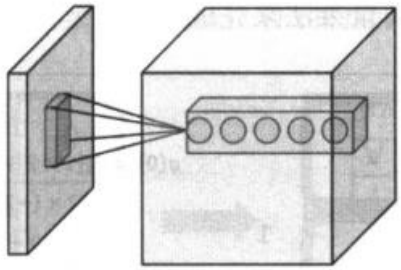


图 2-3 卷积层过滤器结构示意图

过滤器将当前层神经网络上的一个子节点矩阵转化为下一层神经网络上的一个单位节点矩阵。

过滤器的前向传播过程就是通过左侧小矩阵中的节点计算出右侧单位矩阵中节点的过程。假设使用 $w^{i}_{x,y,z}$ 来表示对于输出单位矩阵中的第 i 个节点，过滤器输入节点(x,y,z)的权重，使用 b^{i} 表示第 i 个输出节点对应的偏置项参数，那么单位矩阵中的第 i 个节点的取值 g(i)为：

$$g(i) = f(\sum_{x=1}^{2}\sum_{y=1}^{2}\sum_{z=1}^{3} a_{x,y,z} \times w^{i}_{x,y,z} + b^{i})$$

其中 $a_{x,y,z}$ 为过滤器中节点(x,y,z)的取值，f 为激活函数。

卷积层结构的前向传播过程就是通过一个过滤器从神经网络当前层的左上角移动到右下角，并且在移动中计算每一个对应的的单位矩阵。

在卷积神经网络中，每一个卷积层中使用的过滤器中的参数都是一样的，所以通过卷积层之后无论目标在图像上的哪个位置，得到的结果都一样。共享每一个卷积层中过滤器的参数可以巨幅减少神经网络上的参数。

TensorFlow 对卷积神经网络提供了非常好的支持。

（3）池化层。池化层神经网络不会改变三维矩阵的深度，但它可以缩小矩阵的大小。池化操作可以认为将一张分辨率较高的图片转化为分辨率较低的图片。通过池化层，可以进一步缩小最后全连接层中节点的个数，从而达到减少整个神经网络中参数的目的。

与卷积层类似，池化层前向传播的过程也是通过移动一个类似过滤器的结构完成。池化层过滤器中的计算不是节点的加权和，而是采用最大值或者平均值运算。

（4）全连接层。在经过多轮卷积层和池化层的处理后，在卷积神经网络的最后一般会是由 1 到 2 个全连接层来给出最后的分类结果。可以将卷积层和池化层看成自动图像特征提取的过程。

（5）Softmax 层。Softmax 层主要用于分类问题，通过 Softmax 层，可以得到当前样例属于不同种类的概率分布情况。

2.1.2 TensorFlow

TensorFlow 是世界上最受欢迎的开源机器学习框架，它具有快速、灵活并适合产品级大规模应用等特点，让每个开发者和研究者都能方便地使用人工智能来解决多样化的挑战。TensorFlow 能够让使用者直接解决各种机器学习任务。目标就是在一般情况下，无论遇到什么问题，TensorFlow 都可以在一定程度上提供 API 的支持。

借助于 TensorFlow 的框架，可以训练出含许多层，并且比早期的神经网络更复杂的模型。这就是深度学习时所指的"深度”。在这里'深度'指的就是，层与层之间更深层次的协调，以及随之产生的更加复杂的连接，最终的结果就是模型中有百万级别甚至十亿级别数量的神经元。因此通过深度神经网络得到的结果，能够极大地优于早期的手工构建并且手工调试的模型。

TensorFlow 能够在大型神经网络中将代码转换成操作图。而真正运行的正是这种图。在这些操作之间运行的数据叫做张量(Tensor)。模型以图的形式展现出来，可以推迟或者删除不必要的操作，甚至重用部分结果。基于所见的样本以及计算的误差，来更新模型中的连接强度的过程称为反向传播。

2.1.3 Mobilenet

Tensorflow 在更新 1.0 版本之后多了很多新功能，其中放出了很多用 tf 框架写的深度网络结构，大大降低了开发难度，利用现成的网络结构，无论 fine-tuning 还是重新训练方便了不少。

Object Detection API 提供了 5 种网络结构的预训练的权重，全部是用 COCO 数据集进行训练，这五种模型分别是 SSD+mobilenet、SSD+inception_v2、R-FCN+resnet101、faster RCNN+resnet101、faster RCNN+inception+resnet101。

各个模型的精度和计算所需时间如下，

Model name	Speed	COCO mAP	Outputs
ssd_mobilenet_v1_coco	fast	21	Boxes
ssd_inception_v2_coco	fast	24	Boxes
rfcn_resnet101_coco	medium	30	Boxes
faster_rcnn_resnet101_coco	medium	32	Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	slow	37	Boxes

图 2-4 5 种模型的精度和计算所需时间

MobileNets 是基于一个流线型的架构，它使用深度可分离的卷积来构建轻量级的深层神经网络。引入了两个简单的全局超参数，在延迟度和准确度之间有效地进行平衡。这两个超参数允许模型构建者根据问题的约束条件，为其应用选择合适大小的模型。进行资源和精度权衡的广泛实验，与 ImageNet 分类上的其他流行的网络模型相比，MobileNets 表现出很强的性能。MobileNets 在广泛的应用场景中的非常有效，包括物体检测，细粒度分类，人脸属性和大规模地理定位。

2.1.4 迁移学习

在真实生活中，很难收集到足够多的标注数据用于训练。而即使有海量的数据，训练一个复杂的卷积神经网络也需要几天甚至几周的时间。为了解决标注数据和训练时间的问题，可以使用迁移学习。

所谓迁移学习，就是将一个问题上训练好的模型通过简单的调整使其适用于一个新的问题。在本文中，我们利用 COCO 数据集上训练好的 ssd_mobilenet_V1 模型来解决矿泉水瓶的检测问题。

3 基于 TENSOEFLOW 的矿泉水瓶检测模型训练

3.1 训练准备

3.1.1 软硬件准备

（1）系统：Windows 10

（2）编程语言：Python 3.6

（3）Tensorflow:Tensorflow 1.5 CPU

（4）模型：ssd_mobilenet_V1

（5）protobuf 和 protoc:3.5.1

使用 protobuf 来配置模型和训练参数，所以 API 正常使用必须先编译 protobuf 库，并在模型目录中执行以下命令：

protoc object_detection/protos/*.proto --python_out=.

（6）主对象检测目录结构：

object detection

-data/

--test_labels.csv

--train_labels.csv

--test.record

--train.record

-images/

--test/

---testingimages.jpg

--train/

---testingimages.jpg

--...spring_water_images.jpg

-training

--object-detection.pbtxt

--ssd_mobilenet_v1_waters.config

-xml_to_csv.py

-generate_tfrecord.py

-ssd_mobilenet_v1_coco_11_06_2017/

-win_amwater_inference_graph/

3.1.2 数据准备

（1）矿泉水瓶图片收集。我在网上收集了大约 100 张带有康师傅/优悦矿泉水瓶的图片，其中包括单个的、多个的或一提的矿泉水瓶图片。

（2）标注图像。我使用 LabelImg 来标注图片中的矿泉水瓶，标注程序会自动创建一个描述图片中的对象的 XML 文件。

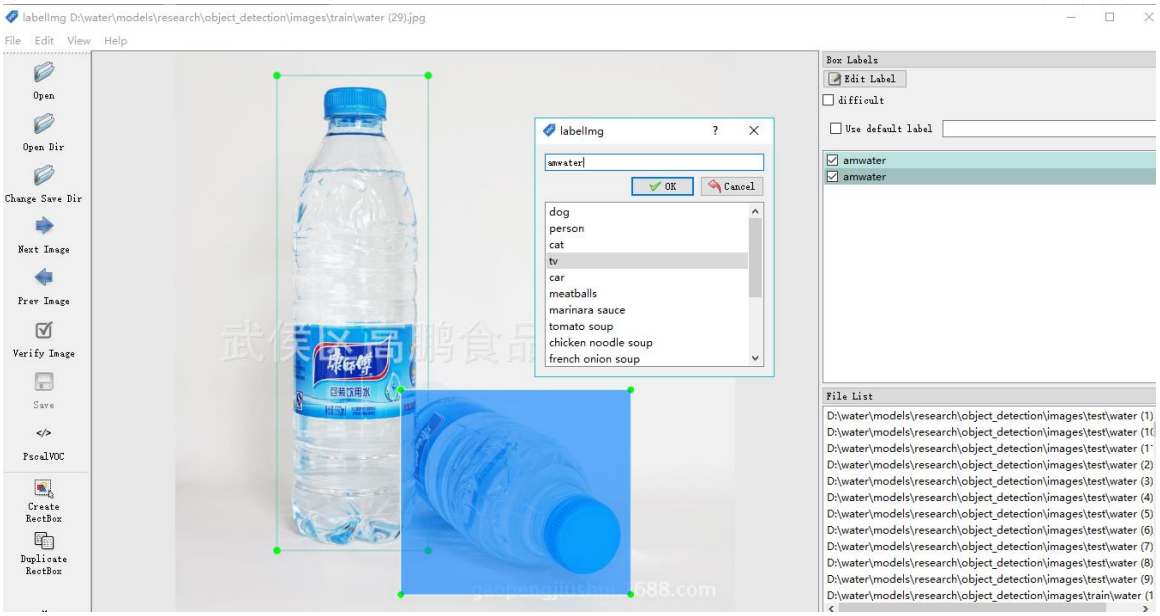


图 3-1 LabelImg 标注图片

（3）标记完毕后，将数据分解成训练/测试样本。因为总样本数据较少，所以我的训练样本就是我收集的所有矿泉水瓶图片，测试样本从所有矿泉水瓶图片中随机选出了大约 10%。

（4）创建 TFRecord。

第一步，将训练和测试样本的所有 XML 文件各自转换为单个 CSV 文件。代码（xml_to_csv.py）如下：

```
# -*- coding: utf-8 -*-
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET

def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df

def main():
    for directory in ['train','test']:
        image_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))
```

```
    xml_df = xml_to_csv(image_path)
    xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None)
    print('Successfully converted xml to csv.')

main()
```

第二步，将 csv 转换为 TFRecord 文件,代码（generate_tfrecord.py）如下：

```
# -*- coding: utf-8 -*-
from __future__ import division
from __future__ import print_function
from __future__ import absolute_import

import os
import io
import pandas as pd
import tensorflow as tf

from PIL import Image
from object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict

flags = tf.app.flags
flags.DEFINE_string('csv_input', '', 'Path to the CSV input')
flags.DEFINE_string('output_path', '', 'Path to output TFRecord')
FLAGS = flags.FLAGS

# TO-DO replace this with label map
def class_text_to_int(row_label):
    if row_label == 'amwater':
```

```
        return 1
    else:
        None


def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.keys(),
gb.groups)]


def create_tf_example(group, path):
    with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size


    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []


    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
```

```
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_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(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }))
    return tf_example

def main(_):
    writer = tf.python_io.TFRecordWriter(FLAGS.output_path)
    path = os.path.join(os.getcwd(), 'images')
    examples = pd.read_csv(FLAGS.csv_input)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())

    writer.close()
    output_path = os.path.join(os.getcwd(), FLAGS.output_path)
    print('Successfully created the TFRecords: {}'.format(output_path))
```

```
if __name__ == '__main__':
    tf.app.run()
```

运行两次 generate_tfrecord.py 脚本，一次用于训练 TFRecord，一次用于测试 TFRecord，命令如下：

```
python generate_tfrecord.py --csv_input=data/train_labels.csv --
output_path=data/train.record
```

```
python generate_tfrecord.py --csv_input=data/test_labels.csv --
output_path=data/test.record
```

3.2 训练矿泉水瓶检测器

（1）配置文件（ssd_mobilenet_v1_waters.config），代码如下：

```
model {
  ssd {
    num_classes: 1
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
```

```
    negatives_lower_than_unmatched: true
    force_match_for_each_row: true
  }
}
similarity_calculator {
  iou_similarity {
  }
}
anchor_generator {
  ssd_anchor_generator {
    num_layers: 6
    min_scale: 0.2
    max_scale: 0.95
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    aspect_ratios: 3.0
    aspect_ratios: 0.3333
  }
}
image_resizer {
  fixed_shape_resizer {
    height: 300
    width: 300
  }
}
box_predictor {
  convolutional_box_predictor {
    min_depth: 0
    max_depth: 0
    num_layers_before_predictor: 0
```

```
    use_dropout: false
    dropout_keep_probability: 0.8
    kernel_size: 1
    box_code_size: 4
    apply_sigmoid_to_scores: false
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.03
          mean: 0.0
        }
      }
      batch_norm {
        train: true,
        scale: true,
        center: true,
        decay: 0.9997,
        epsilon: 0.001,
      }
    }
  }
}
feature_extractor {
  type: 'ssd_mobilenet_v1'
  min_depth: 16
```

```
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
      l2_regularizer {
        weight: 0.00004
      }
    }
    initializer {
      truncated_normal_initializer {
        stddev: 0.03
        mean: 0.0
      }
    }
    batch_norm {
      train: true,
      scale: true,
      center: true,
      decay: 0.9997,
      epsilon: 0.001,
    }
  }
}
loss {
  classification_loss {
    weighted_sigmoid {
      anchorwise_output: true
    }
  }
  localization_loss {
    weighted_smooth_l1 {
```

```
        anchorwise_output: true
      }
    }
    hard_example_miner {
      num_hard_examples: 3000
      iou_threshold: 0.99
      loss_type: CLASSIFICATION
      max_negatives_per_positive: 3
      min_negatives_per_image: 0
    }
    classification_weight: 1.0
    localization_weight: 1.0
  }
  normalize_loss_by_num_matches: true
  post_processing {
    batch_non_max_suppression {
      score_threshold: 1e-8
      iou_threshold: 0.6
      max_detections_per_class: 100
      max_total_detections: 100
    }
    score_converter: SIGMOID
  }
 }
}

train_config: {
 batch_size: 10
 optimizer {
  rms_prop_optimizer: {
   learning_rate: {
```

```
      exponential_decay_learning_rate {
        initial_learning_rate: 0.004
        decay_steps: 800720
        decay_factor: 0.95
      }
    }
    momentum_optimizer_value: 0.9
    decay: 0.9
    epsilon: 1.0
  }
 }
 fine_tune_checkpoint: "ssd_mobilenet_v1_coco_11_06_2017/model.ckpt"
 from_detection_checkpoint: true
 data_augmentation_options {
  random_horizontal_flip {
  }
 }
 data_augmentation_options {
  ssd_random_crop {
  }
 }
}

train_input_reader: {
 tf_record_input_reader {
  input_path: "data/train.record"
 }
 label_map_path: "training/object-detection.pbtxt"
}

eval_config: {
```

```
  num_examples: 12
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "data/test.record"
  }
  label_map_path: "training/object-detection.pbtxt"
  shuffle: false
  num_readers: 1
}
```

（2）类别标签文件（object-detection.pbtxt），代码如下：

```
item {
  id: 1
  name: 'amwater'
}
```

（3）在 models\research\object_detection 下开始训练：

```
python train.py --logtostderr --train_dir=training/ --
pipeline_config_path=training/ssd_mobilenet_v1_waters.config
```

保留最新 5 次的训练模型，输出文件如下：

checkpoint	2018/6/7 8:55	文件	1 KB
events.out.tfevents.1528229076.ASUS-SBY	2018/6/7 9:07	ASUS-SBY 文件	313,717 KB
graph.pbtxt	2018/6/6 4:03	PBTXT 文件	15,496 KB
model.ckpt-3909.data-00000-of-00001	2018/6/7 8:15	DATA-00000-OF...	86,322 KB
model.ckpt-3909.index	2018/6/7 8:15	INDEX 文件	26 KB
model.ckpt-3909.meta	2018/6/7 8:18	META 文件	7,821 KB
model.ckpt-3933.data-00000-of-00001	2018/6/7 8:25	DATA-00000-OF...	86,322 KB
model.ckpt-3933.index	2018/6/7 8:25	INDEX 文件	26 KB
model.ckpt-3933.meta	2018/6/7 8:28	META 文件	7,821 KB
model.ckpt-3957.data-00000-of-00001	2018/6/7 8:35	DATA-00000-OF...	86,322 KB
model.ckpt-3957.index	2018/6/7 8:35	INDEX 文件	26 KB
model.ckpt-3957.meta	2018/6/7 8:38	META 文件	7,821 KB
model.ckpt-3982.data-00000-of-00001	2018/6/7 8:45	DATA-00000-OF...	86,322 KB
model.ckpt-3982.index	2018/6/7 8:45	INDEX 文件	26 KB
model.ckpt-3982.meta	2018/6/7 8:47	META 文件	7,821 KB
model.ckpt-4009.data-00000-of-00001	2018/6/7 8:55	DATA-00000-OF...	86,322 KB
model.ckpt-4009.index	2018/6/7 8:55	INDEX 文件	26 KB
model.ckpt-4009.meta	2018/6/7 9:00	META 文件	7,821 KB

图 3-2 模型训练输出

通过 tensorboard 查看训练过程的总损失变化：

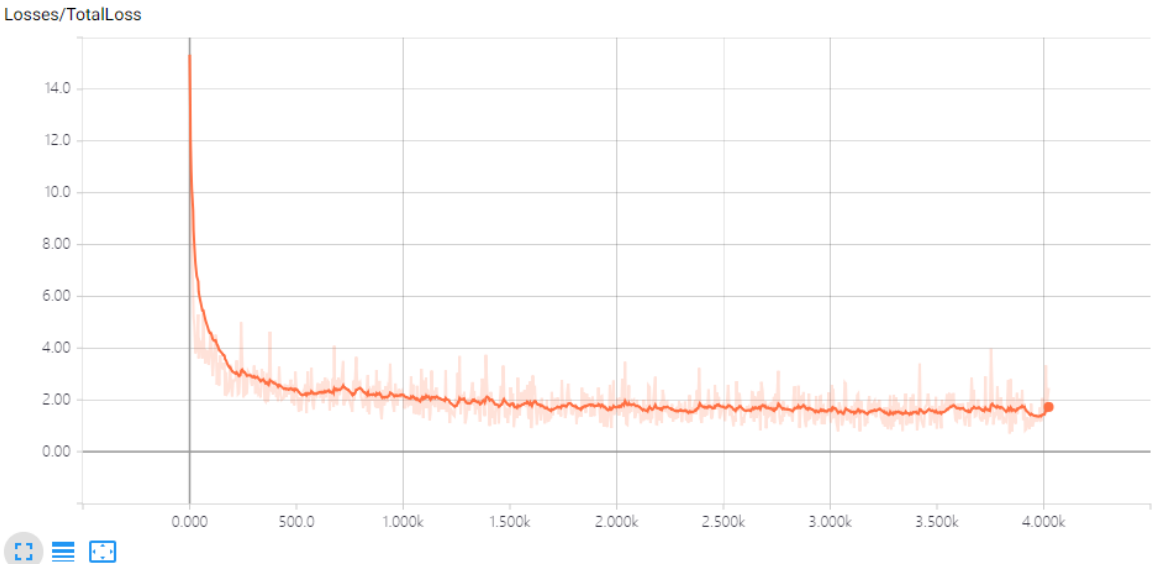


图 3-3 训练总损失图

3.3 测试矿泉水瓶检测器

（1）使用 export_inference_graph.py 脚本导出模型，命令如下：

```
python export_inference_graph.py --input_type image_tensor --pipeline_config_path
training/ssd_mobilenet_v1_waters.config --trained_checkpoint_prefix
training/model.ckpt-4009 --output_directory win_amwater_inference_graph
```

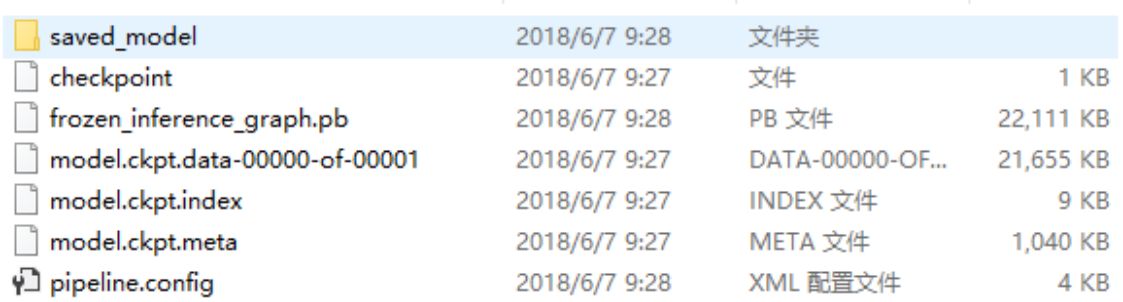


图 3-4 导出模型

（2）测试矿泉水瓶模型（Object Detection.py）

```
# -*- coding: utf-8 -*-
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops

if tf.__version__ < '1.4.0':
  raise ImportError('Please upgrade your tensorflow installation to v1.4.* or later!')
```

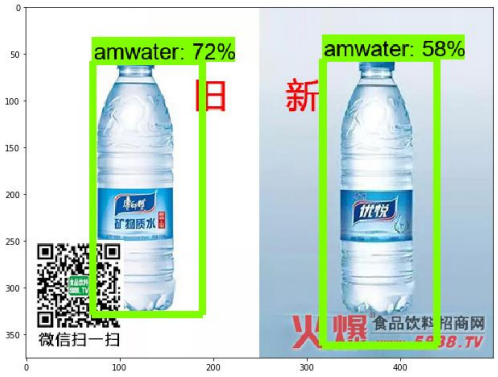
```
# This is needed to display the images.
#%matplotlib inline

from utils import label_map_util

from utils import visualization_utils as vis_util

# What model to download.
MODEL_NAME = 'win_amwater_inference_graph'

# Path to frozen detection graph. This is the actual model that is used for the object detection.
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('training', 'object-detection.pbtxt')

NUM_CLASSES = 1

detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
```

```
def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)


# For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
# If you want to test the code with your images, just add path to the images to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i in range(1, 15) ]


# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)


def run_inference_for_single_image(image, graph):
  with graph.as_default():
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in op.outputs}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
```

```
      tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
          tensor_name)
    if 'detection_masks' in tensor_dict:
      # The following processing is only for single image
      detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
      detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
      # Reframe is required to translate mask from box coordinates to image coordinates and fit the image size.
      real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
      detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
      detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1, -1])
      detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
          detection_masks, detection_boxes, image.shape[0], image.shape[1])
      detection_masks_reframed = tf.cast(
          tf.greater(detection_masks_reframed, 0.5), tf.uint8)
      # Follow the convention by adding back the batch dimension
      tensor_dict['detection_masks'] = tf.expand_dims(
          detection_masks_reframed, 0)
    image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

    # Run inference
    output_dict = sess.run(tensor_dict,
                           feed_dict={image_tensor: np.expand_dims(image, 0)})

    # all outputs are float32 numpy arrays, so convert types as appropriate
    output_dict['num_detections'] = int(output_dict['num_detections'][0])
    output_dict['detection_classes'] = output_dict[
        'detection_classes'][0].astype(np.uint8)
    output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
    output_dict['detection_scores'] = output_dict['detection_scores'][0]
    if 'detection_masks' in output_dict:
```

```
        output_dict['detection_masks'] = output_dict['detection_masks'][0]
    return output_dict


for image_path in TEST_IMAGE_PATHS:
    image = Image.open(image_path)
    # the array based representation of the image will be used later in order to prepare the
    # result image with boxes and labels on it.
    image_np = load_image_into_numpy_array(image)
    # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
    image_np_expanded = np.expand_dims(image_np, axis=0)
    # Actual detection.
    output_dict = run_inference_for_single_image(image_np, detection_graph)
    # Visualization of the results of a detection.
    vis_util.visualize_boxes_and_labels_on_image_array(
        image_np,
        output_dict['detection_boxes'],
        output_dict['detection_classes'],
        output_dict['detection_scores'],
        category_index,
        instance_masks=output_dict.get('detection_masks'),
        use_normalized_coordinates=True,
        line_thickness=8)
    plt.figure(figsize=IMAGE_SIZE)
    plt.imshow(image_np)
```

实际测试结果如下：


amwater: 72%
amwater: 58%
旧
新
微信扫一扫


水蜜桃
鲜果橙
¥8.8


amwater: 95%
amwater: 98%


¥8.8

4 结论及不足

4.1 结论

从总损失图及测试结果可以看出，基于深度学习的矿泉水瓶检测的研究训练的目标基本完成，通过迁移学习训练出的模型，基本可以检测出静态图像中的一个、多个或者一提康师傅/优悦矿泉水瓶。

4.2 不足和改进

经过多次测试发现以下问题：

（1）受 mobilenet 模型限制，对小目标检测效果不佳，可以尝试采用其他模型进行迁移学习；

（2）由于训练样本较少，对目标的各种情况覆盖不够全面，训练时间限制等因素，导致一些情况下不能有效识别出矿泉水瓶，比如只有上半个矿泉水瓶或者矿泉水瓶的背面，针对这种情况，在条件允许的情况下，可以加大训练样本，增加训练时间，以取得更低的损失，达到更好的检测效果。

5 附录

（1）目标检测 API 下载地址：

https://github.com/tensorflow/model

（2）样本标注程序 LabelImg 下载地址：

https://github.com/tzutalin/labelImg

（3）Mobilenet 检查点和配置文件下载地址：

https://raw.githubusercontent.com/tensorflow/models/master/object_detection/samples/configs/ssd_mobilenet_v1_pets.config

http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_11_06_2017.tar.gz

6 参考文献

[1]TensorFlow 目标检测：https://github.com/wizardforcel/py-ds-intro-tut-zh/blob/master/object-detection.md

[2] 付若楠. 基于深度学习的目标检测研究[D]. 北京交通大学, 2017.

[3] 深度解读谷歌 MobileNet: https://blog.csdn.net/t800ghb/article/details/78879612

[4]郑泽宇，顾思宇. TensorFlow:实战 Google 深度学习框架. 北京：电子工业出版社，2017.03.