案例三：基于传统目标检测算法识别目标并定位其位置

1. 案例目的

• 理解模型的保存与加载；

• 掌握传统目标检测算法流程；

• 掌握滑动窗口、图像金字塔、非极大值抑制等概念；

• 掌握resnet50分类模型的使用；

2. 案例内容

本案例将首先使用图像金字塔不断缩小图像规模得到多张图像，然后使用滑动窗口处理每张图像得到多个ROI，并将得到的一批ROI传入ResNet50分类模型做分类，最后使用非极大值抑制算法确定最终边界框，以得到最终目标检测结果。

3. 案例知识点

• Python语言编程；

• NumPy和OpenCV库的使用；

• 滑动窗口、图像金字塔、非极大值抑制的原理；

• ResNet50分类模型的使用

4. 案例时长

共4学时，具体安排如下：

• 定义目标检测算法（2学时）

• 预测检测结果（2学时）

5. 案例实验环境

**软件环境：**

• Python 3.7.4

• Tensorflow 2.3.0

• Keras 2.4.3

**开发环境与工具：**

• Spyder

6. 案例分析

本案例主要分为以下2部分：

1）定义目标检测算法；

传统目标检测算法包含4个部分：首先使用图像金字塔按比例不断缩小图像尺寸得到多张图像，然后对每张图像使用滑动窗口选取ROI区域，得到一批ROI后，将ROI传入分类模型做分类，最终使用非极大值抑制方法处理边界框。

2）预测检测结果；

调用1）目标检测算法中定义的4个方法，并使用ResNet50模型进行分类识别。

7. 案例实验过程

该案例可分为以下几个步骤：

1、 目标检测算法

1.1导入库

1.2滑动窗口

1.3 图像金字塔

1.4 图像分类识别（批处理）

1.5 非极大值抑制

2、 预测检测结果

2.1 导入库

2.2 设置全局变量

2.3 设置全局常量

2.4 加载ResNet50模型

2.5 存储预测结果

2.6 改变图片尺寸

2.7 初始化保存处理后的图像的变量

2.8 设置检测的起始时间

2.9 目标检测

2.10 处理遗留的ROI

2.11 输出检测用时

2.12 处理不存在检测目标的情况

2.13 非极大值抑制处理

7.1开发准备

在桌面上新建一个目录，名字叫obj\_det。本案例中所有的数据集和代码都放在该目录下。

7.1.1 获取数据

获取数据之前，在obj\_det目录下新建一个目录，命名为input。获取到的数据将存放在此目录下。

本案例需要单张图像。



将图像保存下来，并且重命名为dog\_1.jpg，然后复制到刚刚创建的obj\_det/input目录下。

7.1.2 打开Spyder软件

在桌面上双击Spyder图标打开Spyder，并把工作目录切换至obj\_det目录下。

7.1.3 新建代码文件

确保Spyder的工作目录为obj\_det。使用Spyder新建2个Python文件，分别命名为obj\_det\_algrithm.py和simple\_object\_detection.py。

7.1.4 新建图像输出目录

在obj\_det目录下新建一个目录，命名为output。检测结果图像会保存在该目录下。

至此，整个案例的目录结构如图1所示。

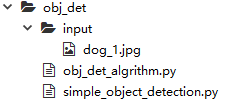


图1 案例目录结构

7.2目标检测算法

向obj\_det\_algrithm.py文件中添加代码，该模块对目标检测算法的每一步进行了定义。传统目标检测算法分为四个步骤：首先通过图像金字塔不断缩小图像规模得到多张图像，然后使用滑动窗口处理每张图像得到多个ROI，并将得到的一批ROI传入分类模型做分类，最后使用非极大值抑制算法确定最终边界框。因此，在该模块中定义了4个函数sliding\_window、image\_pyramid、classify\_batch、non\_max\_suppression分别对应目标检测的4步。

7.2.1 导入库

把所有用到的库都导入。

【代码7-2-1】obj\_det\_algrithm.py

# 导入库

from keras.applications import imagenet\_utils

import numpy as np

import imutils

7.2.2 滑动窗口

滑动窗口(Sliding Window)是宽度和高度固定的矩形区域 (ROI)，可在图像上“滑动”。

该部分定义滑动窗口方法，通过遍历图像选取固定大小的矩形区域，并返回x坐标、y坐标以及当前窗口。

【代码7-2-2】obj\_det\_algrithm.py

# 滑动窗口

# @image: 输入图像

# @step: 值一般为 4像素 或 8像素 或 16像素

# @ws: 窗口的高度和宽度

def sliding\_window(image, step, ws):

# slide a window across the image

for y in range(0, image.shape[0] - ws[0], step):# 遍历高度像素

for x in range(0, image.shape[1] - ws[1], step): # 遍历宽度像素

# 返回x坐标、y坐标、当前窗口

yield (x, y, image[y:y + ws[0], x:x + ws[1]])

7.2.3 图像金字塔

图像金字塔(Image Pyramid)即对图像进行一定比例的缩放。图像金字塔的作用在于解决目标检测中的尺度问题。在比较早的时候，是通过改变滑动窗口的形式来检测图像中大小不一的物体，而目前阶段，更多的是采用滑动窗口规格不变，改变图片大小来检测图像中尺度不一致的物体。金字塔的层级越多，计算量更大，花费的时间会更多，但是，在某种程度上有获得更准确的结果。

图像金字塔的具体实现如下：输入图像后，每次变化scale倍，当小于可接受物体的最小尺寸minSize结束。

【代码7-2-3】obj\_det\_algrithm.py

# 图像金字塔

# @image: 输入图像

# @scale: 图像金字塔每次变化的倍数

# @minSize: 可接受的物体的最小尺寸

def image\_pyramid(image, scale=1.5, minSize=(224, 224)):

# yield 原始图像

yield image

# 不断遍历，得到金字塔的所有图片

while True:

# 计算下一个金字塔的图片

w = int(image.shape[1] / scale)

image = imutils.resize(image, width=w)

# 如果不符合最小尺寸了，就终止循环

if image.shape[0] < minSize[1] or image.shape[1] < minSize[0]:

break

# yield 下一个金字塔的图片

yield image

7.2.4 图像分类识别

为了提高目标检测速度，我们传入分类模型的数据不是单个ROI，而是一批ROI，并且过滤掉识别概率较低的ROI。具体实现过程如下：

【代码7-2-4】obj\_det\_algrithm.py

# 图像分类识别

# @model：图像分类模型

# @batchROIs：批量roi

# @ batchLocs：批量x、y坐标

# @labels：key: 预测的物体；value: 坐标区域和预测的概率

# @minProb：最低识别概率

# @top：得到预测的前top个结果

# @dims：图像分类模型需要的图片尺寸

def classify\_batch(model, batchROIs, batchLocs, labels, minProb=0.5, top=10, dims=(224, 224)):

# 预测

preds = model.predict(batchROIs)

P = imagenet\_utils.decode\_predictions(preds, top=top) # "vgg16", "vgg19", "resnet"都需要这行代码

# 循环预测的值

for i in range(0, len(P)):

for (\_, label, prob) in P[i]:

# 过滤掉识别概率较低的记录

if prob > minProb:

# 得到检测区域

(pX, pY) = batchLocs[i]

box = (pX, pY, pX + dims[0], pY + dims[1])

# 改变label的值

L = labels.get(label, [])

L.append((box, prob))

labels[label] = L

# 返回label的值

return labels

7.2.5 非极大值抑制

非极大值抑制(Non-maxima Suppression, NMS)抑制不是极大值的元素，可以理解为局部最大搜索，也可以理解为只取置信度最高的一个识别结果。

当滑动窗口ROI接近物体时，会识别出物体。ROI越接近物体，置信度越高。

过程：对于Bounding Box的列表B及其对应的置信度S，采用下面的计算方式：选择具有最大score的检测框M，将其从B集合中移除并加入到最终的检测结果D中。通常将B中剩余检测框中与M的IoU大于阈值Nt的框从B中移除。重复这个过程，直到B为空。

结论：从上面的列表中挑选一个置信度最高的为最终的结果。

【代码7-2-5】obj\_det\_algrithm.py

# 非极大值抑制

# @boxes：检测到的区域

# @probs：识别的物体的概率

def non\_max\_suppression(boxes, probs=None, overlapThresh=0.3):

# 如果boxes没有值，返回空List

if len(boxes) == 0:

return []

# 如果像素数据是int，转换为float类型

if boxes.dtype.kind == "i":

boxes = boxes.astype("float")

# initialize the list of picked indexes

pick = []

# 得到boxes的坐标

x1 = boxes[:, 0]

y1 = boxes[:, 1]

x2 = boxes[:, 2]

y2 = boxes[:, 3]

# 计算检测区域的面积，然后按识别概率由高到低排序

area = (x2 - x1 + 1) \* (y2 - y1 + 1)

idxs = y2

if probs is not None:

idxs = probs

idxs = np.argsort(idxs)

# 循环遍历idxs，直至idxs没有元素

while len(idxs) > 0:

# 得到idxs是最后一个元素

last = len(idxs) - 1

i = idxs[last]

pick.append(i)

# 找到最大的bounding的起始坐标(x1,y1)，找到最小的bounding的终止坐标(x2,y2)

xx1 = np.maximum(x1[i], x1[idxs[:last]])

yy1 = np.maximum(y1[i], y1[idxs[:last]])

xx2 = np.minimum(x2[i], x2[idxs[:last]])

yy2 = np.minimum(y2[i], y2[idxs[:last]])

# 计算 bounding 的宽度和高度

w = np.maximum(0, xx2 - xx1 + 1)

h = np.maximum(0, yy2 - yy1 + 1)

# 计算重叠的比率

overlap = (w \* h) / area[idxs[:last]]

# 只要重叠率大于重叠阈值的，从idxs中删掉这个元素

idxs = np.delete(idxs, np.concatenate(([last], np.where(overlap > overlapThresh)[0])))

# 只返回我们选中的box

return boxes[pick].astype("int")

7.2.1、7.2.2、7.2.3、7.2.4、7.2.5的代码只需要添加在obj\_det\_algrithm.py文件中，不需要执行，执行simple\_object\_detection.py文件时会调用该文件中的函数。

7.3预测检测结果

接下来向simple\_object\_detection.py文件中添加代码，该模块的作用是调用obj\_det\_algrithm.py文件中定义的目标检测算法，以及使用ResNet50分类模型，检测单张RGB图像，并输出目标检测结果。

7.3.1 导入库

把所有用到的库都导入。

【代码7-3-1】simple\_object\_detection.py

# 导入包

from obj\_det\_algrithm import image\_pyramid

from obj\_det\_algrithm import sliding\_window

from obj\_det\_algrithm import classify\_batch

from obj\_det\_algrithm import non\_max\_suppression

from keras.applications import ResNet50

from keras.preprocessing.image import img\_to\_array

from keras.applications import imagenet\_utils

import numpy as np

import time

import cv2

该模块需要调用obj\_det\_algrithm.py文件中定义的目标检测算法，因此需要导入sliding\_window、image\_pyramid、classify\_batch、non\_max\_suppression四个函数。

from keras.applications import ResNet50是为了调用ResNet50的预训练模型。

执行代码块。

7.3.2 设置全局变量

该变量用于指定源图像的路径。

【代码7-3-2】simple\_object\_detection.py

# 全局变量

image\_file = 'input/dog\_1.jpg'

执行代码块。

此时建议切换到Spyder的Variable explorer面板，查看变量信息。

7.3.3 设置全局常量

这些常量都是目标检测用到的一些指定参数。如滑动窗口移动多少像素、图像金字塔每次变化的倍数、ROI的大小、批的大小等。

【代码7-3-3】simple\_object\_detection.py

# 全局常量

CONFIDENCE = 0.5 # 图像分类的最低准确率

INPUT\_SIZE = (350, 350) # 把输入图片统一处理成这个尺寸，如果图片过大提高处理速度

WIN\_STEP = 16 # 滑动窗口每次移动几个像素

PYR\_SCALE = 1.5 # 图像金字塔每次变化的倍数

ROI\_SIZE = (224, 224) # 感兴趣区域的尺寸，因为ResNet的输入图像的尺寸是 (224, 224)

BATCH\_SIZE = 64 # 每满 BATCH\_SIZE 条记录，便检测一次

执行代码块。

7.3.4 加载ResNet50模型

传统目标检测算法的第三步是对ROI进行分类识别，此处我们只需要调用预训练好的CNN分类模型即可，以ResNet50为例。

【代码7-3-4】simple\_object\_detection.py

# 加载CNN模型-ResNet50

print("[INFO] 加载CNN模型...")

model = ResNet50(weights="imagenet", include\_top=True)

执行代码块。

7.3.5 存储预测结果

定义变量，用于存储识别结果。变量是字典类型，key是预测的物体，value是坐标区域和预测的概率。样式如图2所示：



图2 labels变量样式

【代码7-3-5】simple\_object\_detection.py

# 存储预测结果

# key: 预测的物体

# value: 坐标区域和预测的概率

labels = {}

执行代码块。

7.3.6 改变图片尺寸

由于传统目标检测算法的执行速度较慢，尤其对于较大图像，需要花费较长的时间，因此，通过resize缩小图像尺寸，提高算法的执行速度。

【代码7-3-6】simple\_object\_detection.py

# 读取图片，并改变图片尺寸

orig = cv2.imread(image\_file) # 读取图片

#(h, w) = orig.shape[:2] # 获取图片的高度和宽度

resized = cv2.resize(orig, INPUT\_SIZE, interpolation=cv2.INTER\_CUBIC) # 改变图片尺寸

执行代码块。

7.3.7 初始化保存处理后的图像的变量

初始化保存处理后的图像变量，便于后续使用。

【代码7-3-7】simple\_object\_detection.py

# 初始化保存处理后的图像的变量

batchROIs = None # 批量感兴趣区域(ROI)

batchLocs = [] # 批量(x, y)坐标

执行代码块。

7.3.8 设置检测的起始时间

由于传统的目标检测程序检测速度较慢，需要花数十秒乃至几分钟，所以有必要记录程序执行前的时间和执行后的时间。

【代码7-3-8】simple\_object\_detection.py

# start the timer

print("[INFO] 开始检测图片中的物体...")

start = time.time() # 记录检测物体时的起始时间

执行代码块。

7.3.9目标检测

该部分正式做目标检测。步骤如下：首先循环遍历图像金字塔，然后对金字塔的每一张图像循环遍历滑动窗口，再把遍历后的图像区域填充到batch中，当batch满了之后，就把batch中的数据做图像分类。

【代码7-3-9】simple\_object\_detection.py

# 循环遍历图像金字塔

batch\_rois = None # 此行代码纯属为了理解 batchROIs 的值

for image in image\_pyramid(resized, scale=PYR\_SCALE, minSize=ROI\_SIZE):

# 循环遍历滑动窗口

for (x, y, roi) in sliding\_window(resized, WIN\_STEP, ROI\_SIZE):

roi = img\_to\_array(roi) # 图片像素类型由uint8转换为float32

roi = np.expand\_dims(roi, axis=0) # 扩展成4维，第1维是样本的条数，即1

roi = imagenet\_utils.preprocess\_input(roi) # "vgg16", "vgg19", "resnet"都需要这行代码

# 如果batchROIs的值是空，那么用roi赋值。否则，将roi的值添加在batchROIs下面

# 第1次循环，batchROIs的尺寸是(1, 224, 224, 3)，第2次循环，尺寸是(2, 224, 224, 3)

# 第3次循环，尺寸是(3, 224, 224, 3)

if batchROIs is None:

batchROIs = roi

else:

batchROIs = np.vstack([batchROIs, roi])

# 更新 batchLocs 的值

batchLocs.append((x, y))

# 检测batch是否满了。如果满了就识别

if len(batchROIs) == BATCH\_SIZE:

# 对batch里的样本做图像分类识别

labels = classify\_batch(model, batchROIs, batchLocs, labels, minProb=CONFIDENCE)

batch\_rois = batchROIs # 此行代码纯属为了理解 batchROIs 的值

# 清空batchROIs和batchLocs的值，准备下一个batch

batchROIs = None

batchLocs = []

执行代码块。

7.3.10 处理遗留的ROI

由于定义全局常量中已定义batch\_size的大小为64，即每满足64个ROI就会检测，若仅有65个ROI，就会有一个ROI没有被检测，因此，需判断是否有ROI没有被检测，若有遗留，则送入分类模型识别。

【代码7-3-10】simple\_object\_detection.py

# 判断是否有遗留的 ROIs 没有被图像分类模型识别。如果有，就识别

if batchROIs is not None:

labels = classify\_batch(model, batchROIs, batchLocs, labels, minProb=CONFIDENCE)

执行代码块。

7.3.11 输出检测用时

7.3.8中已经记录了检测开始时间，该部分用于记录检测结束时间，并用结束时间减开始时间，以得到检测用时，并输出。

【代码7-3-11】simple\_object\_detection.py

# 目标检测程序结束，输出时间

end = time.time()

print("[INFO] 检测用了 {:.4f} 秒".format(end - start))

执行代码块。

7.3.12 处理不存在检测目标的情况

若图像中不存在标签中所包含的类别，或者没有正确检测出图像中包含的类别，则输出“没有检测到物体”。

【代码7-3-12】simple\_object\_detection.py

# 如果没有检测到物体

if not labels:

print('没有检测到物体')

执行代码块。

7.3.13 非极大值抑制处理

该部分主要应用非极大值抑制算法对边界框进行处理。首先循环遍历所有的类别，对于每一个类别都调用non\_max\_suppression函数进行非极大值一直处理，并且根据处理后区域的坐标，画矩形框，并将预测的物体类别和概率显示出来。最终，将图像检测结果显示出来。

【代码7-3-13】simple\_object\_detection.py

# 循环遍历 labels，处理每一个类别

for k in labels.keys():

clone = resized.copy() # 用于显示

# 使用非极大值抑制算法一直弱的、重叠的检测区域

boxes = np.array([p[0] for p in labels[k]])

proba = np.array([p[1] for p in labels[k]])

boxes = non\_max\_suppression(boxes, proba)

# 在每一个识别区域上画上边框，画出坐标区域，并把预测到的物体写在上面

for (xA, yA, xB, yB) in boxes:

cv2.rectangle(clone, (xA, yA), (xB, yB), (0, 0, 255), 2)

display\_label = '%s: %.3f' %(k, proba[np.argmax(proba)])

print(display\_label)

cv2.putText(clone, display\_label, (10,25), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,0,255),2) # 写上物体及概率

# 显示图像

print("[INFO] {}: {}".format(k, len(boxes)))

cv2.imwrite 'output/dog\_1.jpg',clone)

执行代码块。

代码执行完毕后，图像检测结果图保存在output目录下，共有检测三种结果，检测结果1是Old\_English\_sheepdog，如图3所示。其中文名字为古英国牧羊犬，实际图片如图4所示。检测概率为0.546，案例图片中的犬和古英国牧羊犬确实有一定的相似度。



图3 检测结果1



图4 古英国牧羊犬

检测结果2是Bouvier\_des\_Flandres，如图5所示。其中文名字为法兰德斯牧牛犬，实际图片如图6所示。检测概率为0.543，案例图片中的犬和法兰德斯牧牛犬确实有一定的相似度。

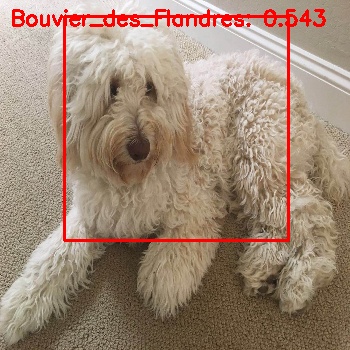


图 5 检测结果2



图6 法兰德斯牧牛犬

检测结果3是soft-coated\_wheaten\_terrier，如图7所示。其中文名字为爱尔兰软毛麦色梗，实际图片如图8所示。检测概率为0.929，案例图片中的犬和爱尔兰软毛麦色梗相似度最高，且概率也最高。

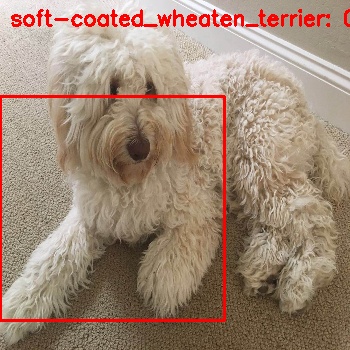


图7 检测结果3



图8 软毛麦色梗

此外，控制台会打印检测用时，目标类别以及检测概率，如图9所示

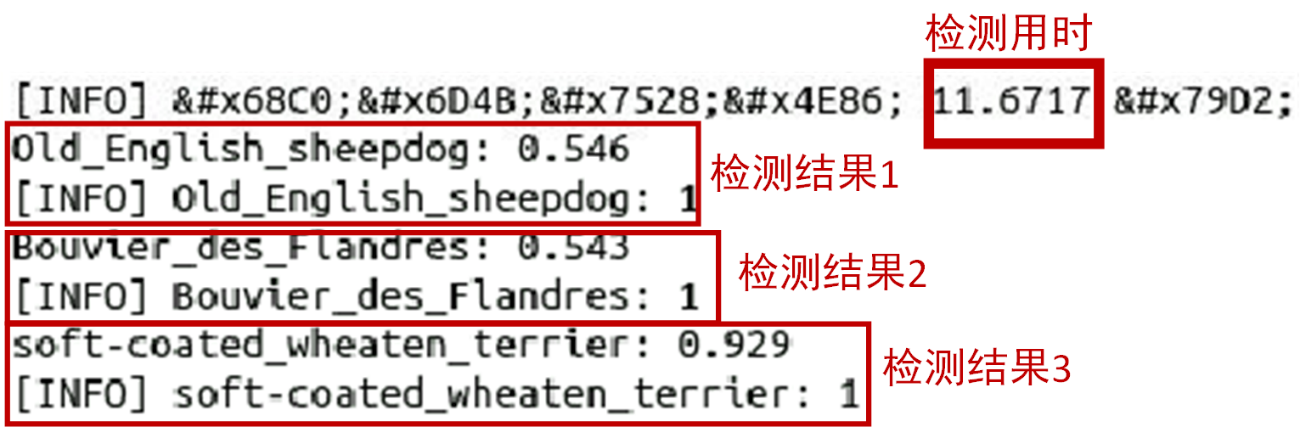


图9 控制台输出结果

结论：从结果中可以看到，该目标检测算法检测出了三种结果，其中soft-coated\_wheaten\_terrier类别的概率为0.929，是检测出的三类中最高的。出现三种检测结果的原因是，滑动窗口处理一张图片时会得到不同的ROI，每一个ROI都需要输入ResNet50中进行分类，由于实际三种犬比较相似，因此出现了多种结果。这也就说明了传统目标检测算法对于相似度比较高的目标，检测的准确性不够高。此外，检测用时为11.6717秒，速度较慢。本案例是为了让同学们了解传统目标检测算法流程，掌握滑动窗口、图像金字塔、非极大值抑制等的python实现。目前已提出多种one-stage的目标检测算法，速度较快，且准确性较高，后面的课程会练习更好的目标检测模型，同学们如果有时间也可以查阅资料进行了解。

8. 案例代码

【案例代码】obj\_det\_algrithm.py

# -\*- coding: utf-8 -\*-

'''

目标检测算法

'''

# 导入库

from keras.applications import imagenet\_utils

import numpy as np

import imutils

# 滑动窗口

# @image: 输入图像

# @step: 值一般为 4像素 或 8像素 或 16像素

# @ws: 窗口的高度和宽度

def sliding\_window(image, step, ws):

# slide a window across the image

for y in range(0, image.shape[0] - ws[0], step):# 遍历高度像素

for x in range(0, image.shape[1] - ws[1], step): # 遍历宽度像素

# 返回x坐标、y坐标、当前窗口

yield (x, y, image[y:y + ws[0], x:x + ws[1]])

# 图像金字塔

# @image: 输入图像

# @scale: 图像金字塔每次变化的倍数

# @minSize: 可接受的物体的最小尺寸

def image\_pyramid(image, scale=1.5, minSize=(224, 224)):

# yield 原始图像

yield image

# 不断遍历，得到金字塔的所有图片

while True:

# 计算下一个金字塔的图片

w = int(image.shape[1] / scale)

image = imutils.resize(image, width=w)

# 如果不符合最小尺寸了，就终止循环

if image.shape[0] < minSize[1] or image.shape[1] < minSize[0]:

break

# yield 下一个金字塔的图片

yield image

# 图像分类识别

# @model：图像分类模型

# @batchROIs：批量roi

# @ batchLocs：批量x、y坐标

# @labels：key: 预测的物体；value: 坐标区域和预测的概率

# @minProb：最低识别概率

# @top：得到预测的前top个结果

# @dims：图像分类模型需要的图片尺寸

def classify\_batch(model, batchROIs, batchLocs, labels, minProb=0.5, top=10, dims=(224, 224)):

# 预测

preds = model.predict(batchROIs)

P = imagenet\_utils.decode\_predictions(preds, top=top) # "vgg16", "vgg19", "resnet"都需要这行代码

# 循环预测的值

for i in range(0, len(P)):

for (\_, label, prob) in P[i]:

# 过滤掉识别概率较低的记录

if prob > minProb:

# 得到检测区域

(pX, pY) = batchLocs[i]

box = (pX, pY, pX + dims[0], pY + dims[1])

# 改变label的值

L = labels.get(label, [])

L.append((box, prob))

labels[label] = L

# 返回label的值

return labels

# 非极大值抑制

# @boxes：检测到的区域

# @probs：识别的物体的概率

def non\_max\_suppression(boxes, probs=None, overlapThresh=0.3):

# 如果boxes没有值，返回空List

if len(boxes) == 0:

return []

# 如果像素数据是int，转换为float类型

if boxes.dtype.kind == "i":

boxes = boxes.astype("float")

# initialize the list of picked indexes

pick = []

# 得到boxes的坐标

x1 = boxes[:, 0]

y1 = boxes[:, 1]

x2 = boxes[:, 2]

y2 = boxes[:, 3]

# 计算检测区域的面积，然后按识别概率由高到低排序

area = (x2 - x1 + 1) \* (y2 - y1 + 1)

idxs = y2

if probs is not None:

idxs = probs

idxs = np.argsort(idxs)

# 循环遍历idxs，直至idxs没有元素

while len(idxs) > 0:

# 得到idxs是最后一个元素

last = len(idxs) - 1

i = idxs[last]

pick.append(i)

# 找到最大的bounding的起始坐标(x1,y1)，找到最小的bounding的终止坐标(x2,y2)

xx1 = np.maximum(x1[i], x1[idxs[:last]])

yy1 = np.maximum(y1[i], y1[idxs[:last]])

xx2 = np.minimum(x2[i], x2[idxs[:last]])

yy2 = np.minimum(y2[i], y2[idxs[:last]])

# 计算 bounding 的宽度和高度

w = np.maximum(0, xx2 - xx1 + 1)

h = np.maximum(0, yy2 - yy1 + 1)

# 计算重叠的比率

overlap = (w \* h) / area[idxs[:last]]

# 只要重叠率大于重叠阈值的，从idxs中删掉这个元素

idxs = np.delete(idxs, np.concatenate(([last], np.where(overlap > overlapThresh)[0])))

# 只返回我们选中的box

return boxes[pick].astype("int")

【案例代码】simple\_object\_detection.py

# -\*- coding: utf-8 -\*-

'''

使用传统目标检测算法识别狗并定位其位置

'''

# 导入包

from obj\_det\_algrithm import image\_pyramid

from obj\_det\_algrithm import sliding\_window

from obj\_det\_algrithm import classify\_batch

from obj\_det\_algrithm import non\_max\_suppression

from keras.applications import ResNet50

from keras.preprocessing.image import img\_to\_array

from keras.applications import imagenet\_utils

import numpy as np

import time

import cv2

# 全局变量

image\_file = 'input/dog\_1.jpg'

# 全局常量

CONFIDENCE = 0.5 # 图像分类的最低准确率

INPUT\_SIZE = (350, 350) # 把输入图片统一处理成这个尺寸，如果图片过大提高处理速度

WIN\_STEP = 16 # 滑动窗口每次移动几个像素

PYR\_SCALE = 1.5 # 图像金字塔每次变化的倍数

ROI\_SIZE = (224, 224) # 感兴趣区域的尺寸，因为ResNet的输入图像的尺寸是 (224, 224)

BATCH\_SIZE = 64 # 每满 BATCH\_SIZE 条记录，便检测一次

# 加载CNN模型-ResNet50

print("[INFO] 加载CNN模型...")

model = ResNet50(weights="imagenet", include\_top=True)

# 存储预测结果

# key: 预测的物体

# value: 坐标区域和预测的概率

labels = {}

# 读取图片，并改变图片尺寸

orig = cv2.imread(image\_file) # 读取图片

#(h, w) = orig.shape[:2] # 获取图片的高度和宽度

resized = cv2.resize(orig, INPUT\_SIZE, interpolation=cv2.INTER\_CUBIC) # 改变图片尺寸

# 初始化保存处理后的图像的变量

batchROIs = None # 批量感兴趣区域(ROI)

batchLocs = [] # 批量(x, y)坐标

# start the timer

print("[INFO] 开始检测图片中的物体...")

start = time.time() # 记录检测物体时的起始时间

# 循环遍历图像金字塔

batch\_rois = None # 此行代码纯属为了理解 batchROIs 的值

for image in image\_pyramid(resized, scale=PYR\_SCALE, minSize=ROI\_SIZE):

# 循环遍历滑动窗口

for (x, y, roi) in sliding\_window(resized, WIN\_STEP, ROI\_SIZE):

roi = img\_to\_array(roi) # 图片像素类型由uint8转换为float32

roi = np.expand\_dims(roi, axis=0) # 扩展成4维，第1维是样本的条数，即1

roi = imagenet\_utils.preprocess\_input(roi) # "vgg16", "vgg19", "resnet"都需要这行代码

# 如果batchROIs的值是空，那么用roi赋值。否则，将roi的值添加在batchROIs下面

# 第1次循环，batchROIs的尺寸是(1, 224, 224, 3)，第2次循环，尺寸是(2, 224, 224, 3)

# 第3次循环，尺寸是(3, 224, 224, 3)

if batchROIs is None:

batchROIs = roi

else:

batchROIs = np.vstack([batchROIs, roi])

# 更新 batchLocs 的值

batchLocs.append((x, y))

# 检测batch是否满了。如果满了就识别

if len(batchROIs) == BATCH\_SIZE:

# 对batch里的样本做图像分类识别

labels = classify\_batch(model, batchROIs, batchLocs, labels, minProb=CONFIDENCE)

batch\_rois = batchROIs # 此行代码纯属为了理解 batchROIs 的值

# 清空batchROIs和batchLocs的值，准别下一个batch

batchROIs = None

batchLocs = []

# 判断是否有遗留的 ROIs 没有被图像分类模型识别。如果有，就识别

if batchROIs is not None:

labels = classify\_batch(model, batchROIs, batchLocs, labels, minProb=CONFIDENCE)

# 目标检测程序结束，输出时间

end = time.time()

print("[INFO] 检测用了 {:.4f} 秒".format(end - start))

# 如果没有检测到物体

if not labels:

print('没有检测到物体')

# 循环遍历 labels，处理每一个类别

for k in labels.keys():

clone = resized.copy() # 用于显示

# 使用非极大值抑制算法一直弱的、重叠的检测区域

boxes = np.array([p[0] for p in labels[k]])

proba = np.array([p[1] for p in labels[k]])

boxes = non\_max\_suppression(boxes, proba)

# 在每一个识别区域上画上边框，画出坐标区域，并把预测到的物体写在上面

for (xA, yA, xB, yB) in boxes:

cv2.rectangle(clone, (xA, yA), (xB, yB), (0, 0, 255), 2)

display\_label = '%s: %.3f' %(k, proba[np.argmax(proba)])

print(display\_label)

cv2.putText(clone, display\_label, (10,25), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,0,255),2) # 写上物体及概率

# 显示图像

print("[INFO] {}: {}".format(k, len(boxes)))

cv2.imwrite('output/%s.jpg' %(k),clone)