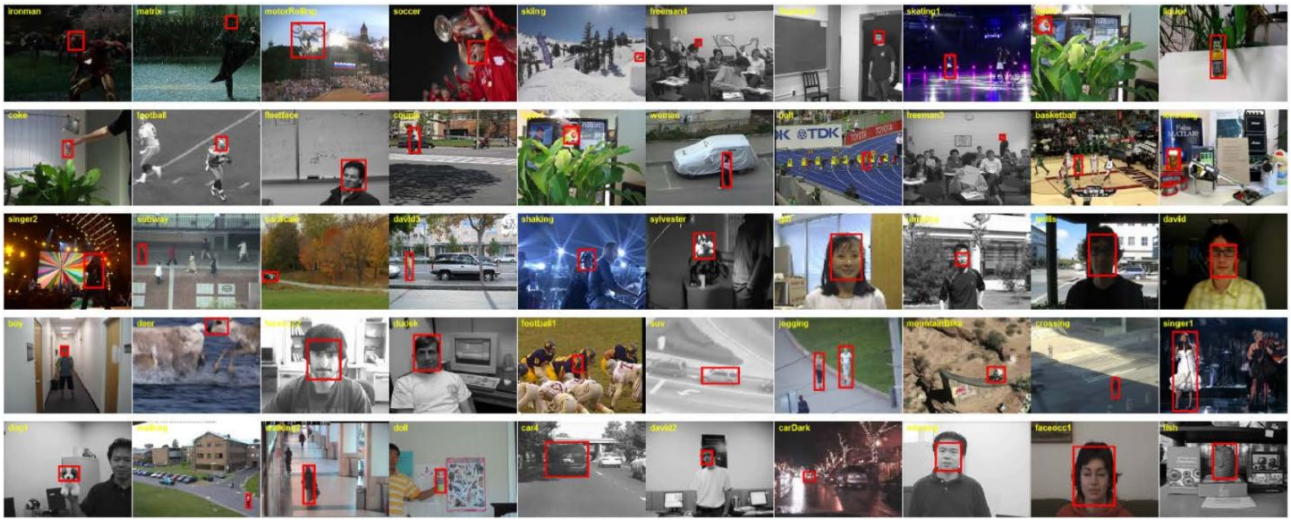
基于YoloV3及Sort算法实现目标跟踪

运行

更新于 2020-09-11 14:18:08 3 2

随着计算机视觉技术的发展，基于视频的目标跟踪算法也成为研究热点。目标跟踪技术通常依据视频中目标及背景的信息，对目标的形状、大小、位置、轨迹等运动状态进行预测。目标跟踪技术的应用领域非常广泛，包括视频监控、无人驾驶等多个领域，具有重要的研究价值。

本案例使用Yolov3算法进行目标检测，并采用Sort算法实现目标跟踪，取得了较好的输出结果。



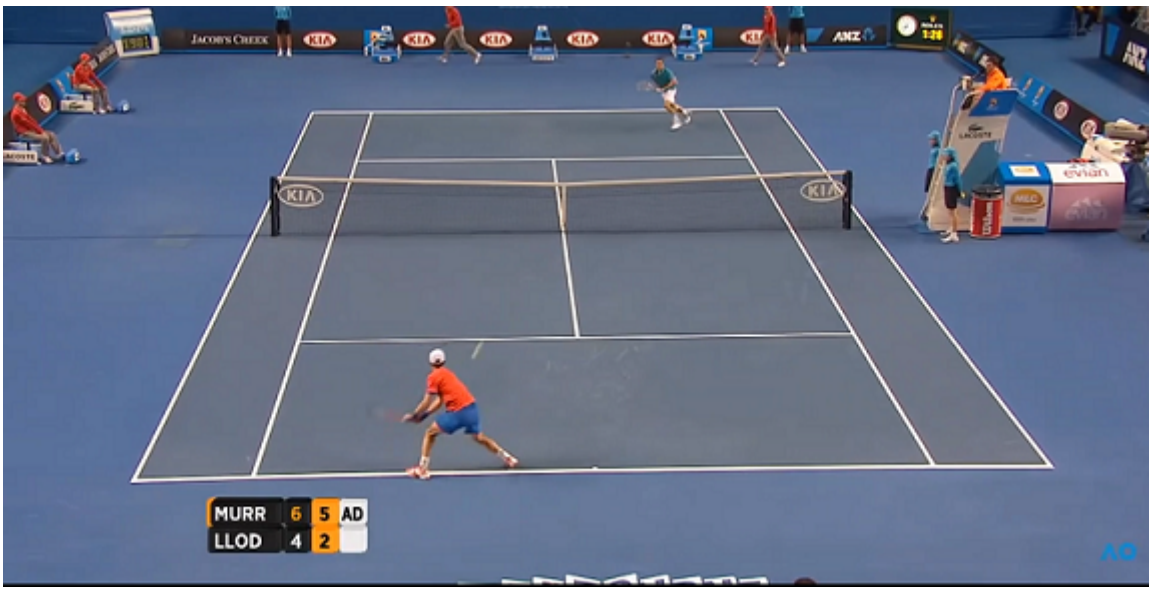
目录

1 数据集简介

本案例采用YoloV3模型实现目标检测，再进行目标跟踪。Yolo算法已经在COCO数据集上进行了预训练。COCO数据集是一个关注目标检测和分割的大型数据集。数据集中包含约20万张日常场景图像，标注出80类物体，如人、狗、沙发、飞机等，方便进行目标检测和分割。



本案例对一段视频中的物体进行目标跟踪。视频的内容是网球男子单打比赛，时长为6秒，数据大小为$2.74 MB$。主要跟踪对象为正在比赛的两名男子，我们希望检测到他们的行动轨迹，并标注出检测结果。

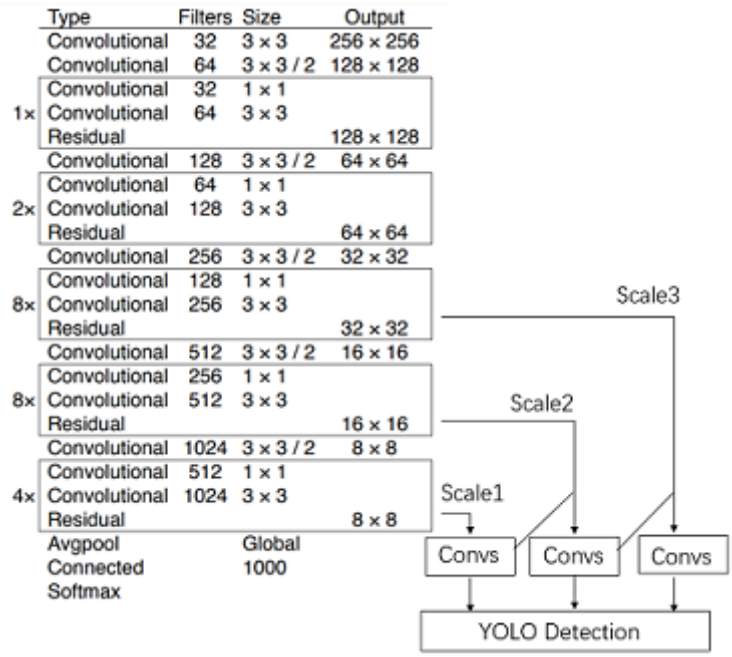


2 模型介绍

2.1 YoloV3模型

Darknet是一个用C语言和CUDA编写的开源神经网络框架。YoloV3基于Darknet实现，且在COCO数据集上进行了预训练。

YoloV3使用了Darknet-53网络，是一个全卷积网络，大量使用残差的跳层连接。并且删除池化层，使用步长为2的卷积核进行降采样。下图为YoloV3的网络结构图。



2.2 Sort模型

目标跟踪的算法思路是：算法预测物体的运动轨迹，并与物体的实际检测框相关联；然后输出检测框，作为目标跟踪结果。本案例采用Sort算法进行目标跟踪。

Sort算法是容易使用且计算速度较快的算法。它提出用卡尔曼滤波和匈牙利算法，来预测上一帧图像中物体的运动轨迹，并与实际检测框相匹配。

在多目标跟踪问题中，匈牙利算法的优化目标可以简单理解为：匹配前后两帧图像中同一物体位置。而卡尔曼滤波先对目标的轨迹进行预测，再使用确信度较高的跟踪结果进行预测结果的修正。

3 读取模型和数据

在进行目标跟踪前，我们需要读取上述介绍的YoloV3模型和Sort模型，以及视频数据。

首先加载需要使用的库。

```
# 导入包含辅助函数的python文件
%%bash
```

```
git clone https://github.com/XiwangLi/object-tracking-SORT-Pytorch.git
```

```
Cloning into 'object-tracking-SORT-Pytorch'...
```

```
cd object-tracking-SORT-Pytorch
cp utils/datasets.py ../
cp models.py ../
cp sort.py ../
```

```
/content/object-tracking-SORT-Pytorch
```

```
# 加载需要使用的库
from models import *
from utils import *
import os, sys, time, datetime, random
import torch
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from torch.autograd import Variable
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
from sort import *
import cv2
from IPython.display import clear_output
```

YoloV3算法已经在COCO数据集上进行了预训练，我们读取预训练得到的权重，在进行目标检测时调用模型。

```
#downloading yolo weight
!wget https://pjreddie.com/media/files/yolov3.weights  -O config/yolov3.weights
```

```
--2020-09-11 04:39:34--  https://pjreddie.com/media/files/yolov3.weights
Resolving pjreddie.com (pjreddie.com)... 128.208.4.108
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 248007048 (237M) [application/octet-stream]
Saving to: ‘config/yolov3.weights’

config/yolov3.weigh 100%[===================>] 236.52M   647KB/s    in 6m 24s

2020-09-11 04:45:58 (631 KB/s) - ‘config/yolov3.weights’ saved [248007048/248007048]
```

之后我们给定部分参数的值。因为图像需转换为一致大小以进行目标检测，参数 `img_size` 表示转换后的图像大小。 `conf_thres` 表示置信度阈值， `nms_thres` 表示

非极大值抑制算法阈值，二者用于YoloV3算法选择候选框。

```
config_path = '/content/object-tracking-SORT-Pytorch/config/yolov3.cfg'
weights_path = '/content/object-tracking-SORT-Pytorch/config/yolov3.weights'
class_path = '/content/object-tracking-SORT-Pytorch/config/coco.names'
# 图像大小
img_size = 416
# 置信度阈值
conf_thres = 0.8
# 非极大值抑制算法阈值
nms_thres = 0.4
```

现在我们加载YoloV3模型及预训练得到的权重。

```
# Darknet网络构造
model = Darknet(config_path, img_size=img_size)
# 加载权重
model.load_weights(weights_path)
# 使用GPU处理数据
model.cuda()
# 固定模型BN层和dropout层,不改变模型权值,调用模型进行预测
model.eval()
```

查看算法可识别的物体的类别，结果如下所示。可知YoloV3算法可以检测到的物体有80类，查看前5类物体，分别为：人、自行车、汽车、摩托车、飞机。

```
classes = utils.load_classes(class_path)
# 可检测的物体类别数
print("可检测的物体类别数 ",len(classes))
print("前5类物体 ",classes[:5])
```

```
可检测的物体类别数  80
前5类物体  ['person', 'bicycle', 'car', 'motorbike', 'aeroplane']
```

接着直接调用 `cv2.VideoCapture` 函数，即可读取视频数据，同时调用Sort算法，实例化为 `mot_tracker` 。

```
# 读取视频数据
videopath = '/content/basketball.mp4'
%pylab inline
cmap = plt.get_cmap('tab20b')
cap = cv2.VideoCapture(videopath)
# 实例化Sort算法函数
mot_tracker = Sort()
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
<matplotlib.colors.ListedColormap at 0x7f8e95913080>
```

4 目标跟踪

因为人眼每秒只能分辨出30帧的图像（30帧：每秒钟能够连续播放30副静止画面），所以我们读取视频数据时，每秒截取30幅图像进行处理，即可在视频中实现流畅的目标跟踪结果。

4.1 单帧图像处理

我们先以单帧图像为例，详细介绍本案例实现目标跟踪的思路。

首先定义函数 `convertMillis` ，函数能够返回每幅静止图像在视频中所处的时间。以此作为图像标题，更方便查看输出结果。

```
def convertMillis(millseconds):
    seconds, millseconds = divmod(millseconds, 1000)
    minutes, seconds = divmod(seconds, 60)
    hours, minutes = divmod(minutes, 60)
    day, hours = divmod(hours, 24)
    seconds = int(seconds + millseconds/10000)
    return f"{int(hours)}:{int(minutes)}:{int(seconds)}"
```

接下来我们定义函数 `detect_image` ，函数对图像的大小和数据格式进行了转换，并返回单张图像的目标检测结果。在将图像剪裁为一致大小时，保持了图像原本的长宽比，并对空白部分进行填充。

```
Tensor = torch.cuda.FloatTensor
def detect_image(img):
    # scale and pad image
    ratio = min(img_size/img.size[0], img_size/img.size[1])
    imw = round(img.size[0] * ratio)
    imh = round(img.size[1] * ratio)
    img_transforms = transforms.Compose([ transforms.Resize((imh, imw)),
         transforms.Pad((max(int((imh-imw)/2),0), max(int((imw-imh)/2),0), max
(int((imh-imw)/2),0), max(int((imw-imh)/2),0)),
                        (128,128,128)),
         transforms.ToTensor(),
         ])
    # 转换为float格式,便于进行计算
    image_tensor = img_transforms(img).float()
    image_tensor = image_tensor.unsqueeze_(0)
    input_img = Variable(image_tensor.type(Tensor))
    # 进行目标检测
```

```
    # 进行目标检测
    with torch.no_grad():
        detections = model(input_img)
        detections = utils.non_max_suppression(detections, 80, conf_thres, nms
_thres)
    return detections[0]
```

我们在检测框中添加物体的编号，并使用不同的颜色表示检测框，这样可以更清楚地展示输出结果。

```
colors = [cmap(i)[:3] for i in np.linspace(0, 1, 20)]
print("候选框颜色种类 ",len(colors))
```

候选框颜色种类 20

接下来，我们读取视频中第一帧图像，以这帧图像为例，详细介绍如何实现目标跟踪。

首先读取这一帧数据在视频中的时间位置,作为输出图像的标题。

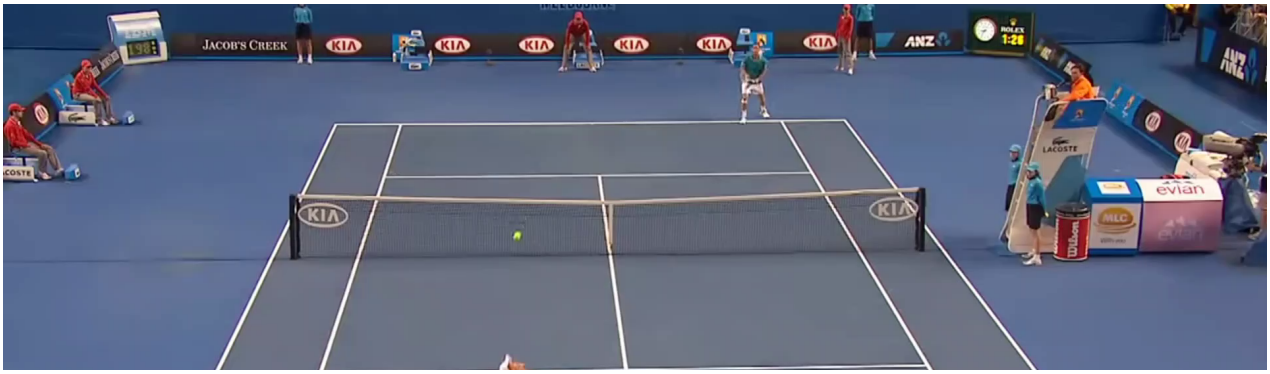
```
timestamp = cap.get(cv2.CAP_PROP_POS_MSEC)
time_report = convertMillis(timestamp)
time_report
```

'0:0:0'

接下来读取图像数据:

- vc.read() ：按帧读取视频;
- ret ：输出布尔值，如果读取帧是正确的，返回 True ；如果文件读取到结尾，返回 False ；
- frame ：输出每一帧的图像数据，是一个三维矩阵。

```
ret, frame = cap.read()
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
pilimg = Image.fromarray(frame)
pilimg
```



对这一帧图像进行目标检测，识别出物体，输出的结果为候选框的位置。

```
# 对这一帧图像进行目标检测
detections = detect_image(pilimg)
detections
```

```
tensor([[164.6796, 216.8698, 191.5544, 274.2080,   0.9882,   0.9999,   0.000
0],
        [240.5188, 111.2726, 253.1677, 140.6082,   0.9283,   0.9997,   0.000
0],
        [167.5435, 177.2674, 171.0233, 180.6892,   0.9877,   0.9994,  32.000
0]],
       device='cuda:0')
```

之后读取图像大小等数据,在后续为图像添加候选框时可以使用。

```
# 读取图像大小等数据
img = np.array(pilimg)
pad_x = max(img.shape[0] - img.shape[1], 0) * (img_size / max(img.shape))
pad_y = max(img.shape[1] - img.shape[0], 0) * (img_size / max(img.shape))
unpad_h = img_size - pad_y
unpad_w = img_size - pad_x
```

`mot_tracker` 为Sort算法的实例对象，调用 `updata` 函数，可以对目标的运动轨迹进行预测，更新目标候选框。

```
# 通过updata函数更新目标轨迹
tracked_objects = mot_tracker.update(detections.cpu())
tracked_objects
```

```
array([[167.54348755, 177.26742557, 171.02328491, 180.68923946,
        249.        ,  32.        ],
       [240.51881411, 111.27262861, 253.16770933, 140.60823076,
        248.        ,   0.        ],
       [164.67958037, 216.86981158, 191.55442842, 274.20800824,
        247.        ,   0.        ]])
```

此外，我们可以输出目标检测的结果，判断结果是否正确，从下列结果中，可知算法可以检

此外，我们可以输出目标检测的结果，判断结果是否正确。这个例子中，可知算法可以检测出两类目标：网球和打网球的人。

```
# 检测出的物体的类别
unique_labels = detections[:, -1].cpu().unique()
unique_labels_number = list(map(lambda x:int(x),unique_labels.numpy().tolist
()))
print("检测出的物体类别 ",list(map(lambda x:classes[x],unique_labels_number
)))
n_cls_preds = len(unique_labels)
print("检测出的物体类别数 ",n_cls_preds)
```

```
检测出的物体类别  ['person', 'sports ball']
检测出的物体类别数  2
```

将更新后的候选框添加到图像中，可以可视化目标跟踪的结果。对不同的目标，我们采用不同颜色的候选框，使图像更加清晰明了。

```
# 将检测框添加到图像上
i = 1
for x1, y1, x2, y2, obj_id, cls_pred in tracked_objects:
    print(i)
    i = i+1
    box_h = int(((y2 - y1) / unpad_h) * img.shape[0])
    box_w = int(((x2 - x1) / unpad_w) * img.shape[1])
    y1 = int(((y1 - pad_y // 2) / unpad_h) * img.shape[0])
    x1 = int(((x1 - pad_x // 2) / unpad_w) * img.shape[1])

    # 设定检测框的颜色
    color = colors[int(obj_id) % len(colors)]
    color = [i * 255 for i in color]
    cls = classes[int(cls_pred)]
    print("检测的物体",cls)
    print("检测框的RGB颜色值",color)

    # 将检测框添加到图像上
    cv2.rectangle(frame, (x1, y1), (x1+box_w, y1+box_h), color, 4)
    cv2.rectangle(frame, (x1, y1-35), (x1+len(cls)*19+60, y1), color, -1)
    cv2.putText(frame, cls + "-" + str(int(obj_id)), (x1, y1 - 10), cv2.FONT_H
ERSHEY_SIMPLEX, 1, (255,255,255), 3)
```

```
1
检测的物体 sports ball
检测框的RGB颜色值 [189.0, 158.0, 57.0]
2
检测的物体 person
检测框的RGB颜色值 [140.0, 109.0, 49.0]
3
检测的物体 person
检测框的RGB颜色值 [206.0, 219.0, 156.0]
```

```
# 展示图像
fig = figure(figsize=(12, 8))
title("Video Stream")
imshow(frame)
title(f"Video Stream: {time_report}")
frame_img = Image.fromarray(frame)
show()
```

发表您的讨论