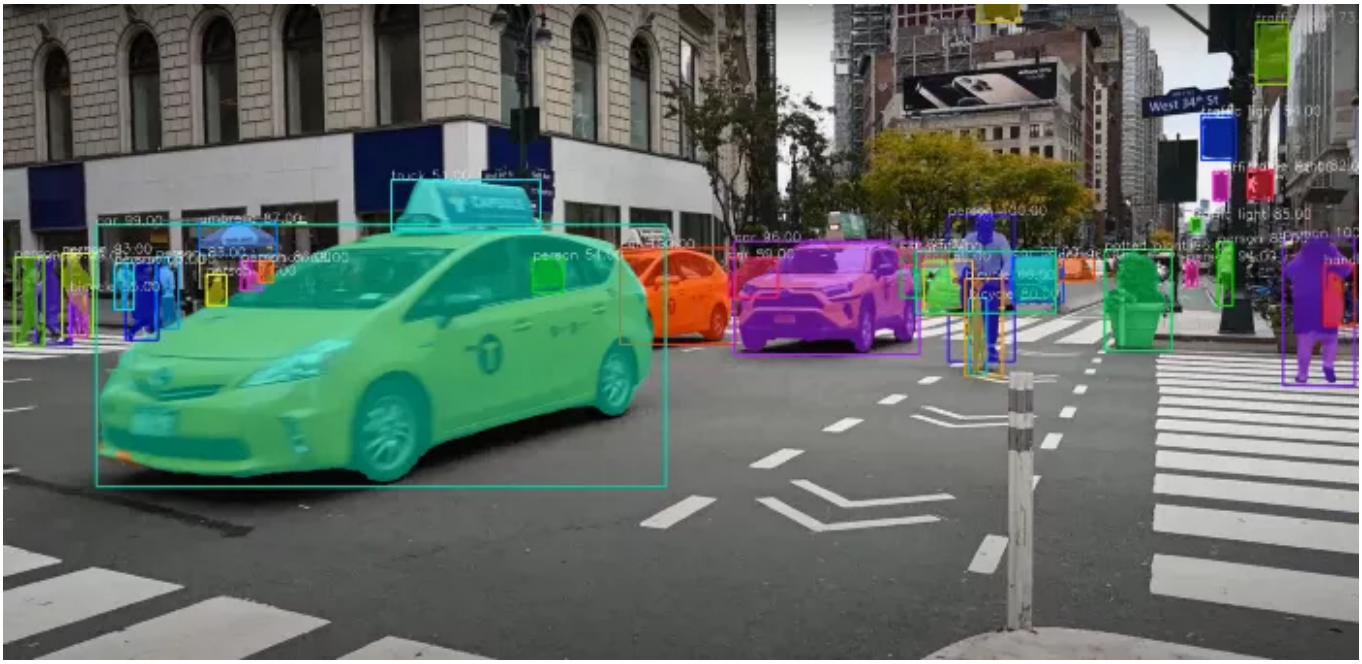
使用5行代碼的實時圖像分割

原創 磐焱 學習與計算機視覺 2022-01-16 23:58



目錄

1. 計算機視覺應用中的圖像分割
2. 圖像分割
3. 圖像分割
4. 視頻分割
5. 視頻的分割

計算機視覺應用中的圖像分割

計算機視覺是看到和分析他們所看到的東西的能力。圖像分割是計算機視覺的一個方面，它處理計算機將可視化對象的分割為不同的類別，以便更好地進行地分析。

圖像分割過程的一個好例子成為解決醫學圖像分析、背景編輯、自動駕駛汽車視覺和衛星分析等諸多計算機視覺問題的重要領域。

PixelLib庫是一個庫，它使用幾行python代碼輕鬆集成和視頻中的對象分割圖像。它支持許多驚人的功能，例如：

1. 和視頻中對象的圖像和畫面分割。
2. 細分模型的定制訓練。
3. 圖像和視頻中的背景編輯。
4. 抽取圖像和視頻中的對象。

實時圖像分割應用的需求

Pytorch Pixel Lib：是保持實時應用的精度和速度之間的距離之一。在性能方面，或計算機視覺領域的解決方案或更快、速度慢，或計算機應用程序中的最大速度、加速性能，這是一個兩難的問題。

以前版本的深度學習庫使用張量流深度學習庫作為後台，採用Mask R-CNN進行分割。Mask R-CNN實例是一種非常好的體系結構，但在實時應用中無法平衡精度和加速性能。

PixelLib现在支持pytorch后端，使用PointRend分割架构对图像和视频中的对象执行更快、更准确的分割和提取。

Alexander Kirillov等人的PointRend对象分割体系结构用于替代Mask R-CNN来执行对象的实例分割。PointRend是一种用于实现对象分割的先进神经网络。它生成精确的分割模板，并以高推理速度运行，以满足对精确和实时计算机视觉应用日益增长的需求。

多操作系统支持

> PixelLib是一个为支持不同操作系统而构建的库。我通过Detectron2将PixelLib与PointRend的python实现集成在一起，后者只支持Linux操作系统。

我对最初的Detectron2 PointRend实现进行了修改，以支持Windows操作系统。我很高兴告诉大家一个好消息，用于PixelLib的PointRend实现同时支持Linux和Windows操作系统。

上面的示例图像是PointRend与Mask RCNN分割结果差异的示例。很明显，与左侧的Mask R-CNN结果相比，右侧的PointRend图像结果是更好的分割输出。

注意：本文基于使用pytorch和PointRend执行实例分割。如果你想了解如何使用tensorflow和Mask R-CNN执行实例分割，请阅读以下文章：

https://towardsdatascience.com/image-segmentation-with-six-lines-0f-code-acb870a462e8

下载Python

PixelLib pytorch版本支持python 3.7及以上版本。下载兼容的python版本。

https://www.python.org/

安装PixelLib及其依赖项

安装Pytorch

PixelLib Pytorch版本支持Pytorch的这些版本（1.6.0、1.7.1、1.8.0和1.9.0）。

注意：Pytorch 1.7.0不受支持，请勿使用任何低于1.6.0的Pytorch版本。安装兼容的Pytorch版本。

https://pytorch.org/

安装Pycocotools

- *pip3 install pycocotools*

安装pixellib

- *pip3 install pixellib*

如果已安装，请使用以下软件升级至最新版本：

- *pip3 install pixellib — upgrade*

图像分割

PixelLib使用五行python代码在具有PointRend模型的图像和视频中执行对象分割。下载PointRend模型。这是图像分割的代码。

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
ins.segmentImage("image.jpg", show_bboxes=True, output_image_name="output_image.j
```

第1-4行：导入了PixelLib包，我们还从模块pixellib.torchbackend.instance导入了instanceSegmentation类（从pytorch支持导入实例分段类）。我们创建了该类的一个实例，并最终加载了下载的PointRend模型。

第5行：我们调用函数segmentImage来执行图像中对象的分割，并在函数中添加了以下参数：

- **image_path**：这是要分割的图像的路径。
- **show_bbox**：这是一个可选参数，用于显示带边框的分段结果。
- **output_image_name**：这是保存的分段图像的名称。

用于分割的样本图像


```
ins.segmentImage("image.jpg",output_image_name="output.jpg")
```

分割后的图像

```
The checkpoint state_dict contains keys that are not used by the model: proposal_
```

如果你正在运行分段代码，上面的日志可能会出现！这不是一个错误，代码可以正常工作！

获得分割结果

```
results, output = ins.segmentImage("image.jpg", show_bboxes=True, output_image_nar
print(results)
```

分割结果返回一个字典，其中包含许多与图像中分割的对象相关的值。打印的结果将采用以下格式：

```
{'boxes':  array([[ 579,  462, 1105,  704],
       [   1,  486,  321,  734],
       [ 321,  371,  423,  742],
       [ 436,  369,  565,  788],
       [ 191,  397,  270,  532],
       [1138,  357, 1197,  482],
       [ 877,  382,  969,  477],),

'class_ids': array([ 2,  2,  0,  0,  0,  0,  0,  2,  0,  0,  0,  0,  2, 24, 24,2,

'class_names': ['car', 'car', 'person', 'person', 'person', 'person', 'person', '

 'object_counts': Counter({'person': 15, 'car': 7, 'backpack': 2}),

'scores': array([100., 100., 100., 100.,  99.,  99.,  98.,  98.,  97.,  96.,  95.

'masks': array([[[False, False, False, ..., False, False, False],

[False, False, False, ..., False, False, False],

'extracted_objects': []
```

上面打印的结果值包括：

```
{'boxes': array([[5.790e+02, 4.620e+02, 1.105e+03, 7.050e+02],[1.000e+00, 4.870e+
```

boxes：这些是分割对象的边界框坐标。我没有显示所有框的坐标，这是因为列表太长了。

```
'class_ids': array([ 2,  2,  0,  0,  0,  0,  0,  2,  0,  0,  0,  0,  2, 24, 24,2,
```

class_ids：这些是分段对象的类ID。

```
'class_names': ['car', 'car', 'person', 'person', 'person', 'person', 'person', '
```

class_names：这些是分段对象的类名。

```
'object_counts': Counter({'person': 15, 'car': 7, 'backpack': 2}),
```

object_counts：这是图像中分割的每个类的计数。我使用python内置计数器对对象进行计数。在本例中，图像中分割了15个人、7辆汽车和2个背包。

```
'scores': array([100., 100., 100., 100.,  99.,  99.,  98.,  98.,  97.,  96.,  95.
```

scores：这些是每个分割对象的置信度分数。

```
'masks': array([[[False, False, False, ..., False, False, False],

[False, False, False, ..., False, False, False],
```

masks：这些是每个分割对象的掩码值。我没有显示所有的掩码值，这是因为列表太长了。

注意：返回的掩码的默认值以bolean为单位。可以使用新参数mask_points_values获取掩码的坐标。

```
ins.segmentImage("sample.jpg", show_bboxes=True, mask_points_values=True,  output_
```

mask_points_values参数已添加到segmentImage函数并设置为True，新的mask值将为：

```
[[array([[295, 497]])
  array([[422, 114],
       [421, 115],
       [417, 115],
       ...,
       [436, 115],
       [433, 115],
       [432, 114]])]]
extracted_objects: This is the container
```

提取的对象：如果我们提取对象，这是提取对象值的容器列表。它是空的，因为我们没有提取任何内容。我们将在本文后面讨论如何提取这些分段对象。

```
results, output = ins.segmentImage("image.jpg", show_bboxes=True, output_image_nam
```

访问分段结果呈现的值

边界框坐标值

```
results["boxes"]
```

类ID值

```
results["class_ids"]
```

类名值

```
results["class_names"]
```

对象计算值

```
results["object_counts"]
```

掩码值

```
results["masks"]
```

检测阈值

PixelLib使得确定目标分割的检测阈值成为可能。

```
ins.load_model("pointrend_resnet50.pkl", confidence = 0.3)
```

置信度：这是在load_model函数中引入的一个新参数，设置为0.3可将检测阈值设置为30%。我为检测阈值设置的默认值为0.5，可以使用置信度参数增加或减少该值。

速度记录

PixelLib使执行实时对象分割成为可能，并增加了调整推理速度以适应实时预测的能力。使用4GB容量的Nvidia GPU处理单个图像的默认推断速度约为0.26秒。

速度调整:

PixelLib支持速度调整，有两种速度调整模式，即快速模式和快速模式：

1 快速模式

```
ins.load_model("pointrend_resnet50.pkl", detection_speed = "fast")
```

在load_model函数中，我们添加了参数detection_speed并将该值设置为fast。快速模式处理单个图像的时间达到0.20秒。

快速模式检测的完整代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
```

```
ins.load_model("pointrend_resnet50.pkl", detection_speed = "fast")
ins.segmentImage("image.jpg", show_bboxes=True, output_image_name="output_image.j
```

2 快速模式

```
ins.load_model("pointrend_resnet50.pkl", detection_speed = "fast")
```

在load_model函数中，我们添加了参数detection_speed并将该值设置为rapid。快速模式处理单个图像的时间达到0.15秒。

快速模式检测的完整代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl", detection_speed = "rapid")
ins.segmentImage("image.jpg", show_bboxes=True, output_image_name="output_image.j
```

PointRend模型

有两种类型的PointRend模型用于对象分割，它们是resnet50变体和resnet101变体。

本文通篇都使用resnet50变体，因为它速度更快，准确性好。resnet101变型更精确，但比resnet50变型慢。根据Detectron2模型的官方报告，resnet50变型在COCO上的mAP达到38.3，resnet101变型在COCO上的mAP达到40.1。

Resnet101的速度记录：分段的默认速度为0.5秒，快速模式为0.3秒，而快速模式为0.25秒。

Resnet101变体的代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
```

```
ins.load_model("pointrend_resnet101.pkl", network_backbone="resnet101")
ins.segmentImage("sample.jpg",  show_bboxes = True, output_image_name="output.jpg
```

```
ins.load_model("pointrend_resnet101.pkl", network_backbone = "resnet101")
```

使用resnet101模型执行推断的代码是相同的，只是我们在load_model函数中加载了PointRend resnet101模型。从这里下载resnet101模型。我们还添加了一个额外的参数network_backbone，并将该值设置为resnet101。

注意：如果你希望实现高推断速度和高精度，请使用PointRend resnet50变体，但如果你更关心精度，请使用PointRend resnet101变体。所有这些推断报告都基于使用4GB容量的Nvidia GPU。

图像分割中的自定义目标检测

所使用的PointRend模型是一个预训练COCO模型，支持80类对象。PixelLib支持自定义对象检测，这使得过滤检测和确保目标对象的分割成为可能。我们可以从支持的80类对象中进行选择，以符合我们的目标。以下是支持的80类对象：

```
person, bicycle, car, motorcycle, airplane,

bus, train, truck, boat, traffic_light, fire_hydrant, stop_sign,

parking_meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra,

giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard,

sports_ball, kite, baseball_bat, baseball_glove, skateboard, surfboard, tennis_ra

bottle, wine_glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orang

broccoli, carrot, hot_dog, pizza, donut, cake, chair, couch, potted_plant, bed,

dining_table, toilet, tv, laptop, mouse, remote, keyboard, cell_phone, microwave,

oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy_bear, hair_

toothbrush.
```

我们想要过滤样本图像的检测结果，以便只检测图像中的人。

```
target_classes = ins.select_target_classes(person = True)

ins.segmentImage("image.jpg", segment_target_classes = target_classes, output_ima
```

调用函数select_target_classes来选择要分割的目标对象。函数segmentImage得到了一个新的参数segmente_target_classes，可以从目标类中进行选择，并根据它们过滤检测。

PixelLib仅检测图像中的人物。

自定义对象检测的完整代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
target_classes = ins.select_target_classes(person = True)
ins.segmentImage("image.jpg", show_bboxes=True, segment_target_classes = target_c
```

图像中的对象提取

PixelLib使提取和分析图像中分割的对象成为可能。

对象提取代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
ins.segmentImage("image.jpg", show_bboxes=True, extract_segmented_objects=True,
save_extracted_objects=True, output_image_name="output_image.jpg" )
```

图像分割的代码是相同的，只是我们在segmentImage函数中添加了额外的参数来执行对象提取。

```
ins.segmentImage("image.jpg", extract_segmented_objects = True, save_extracted_ob
```

- 提取分割对象：这是处理分割对象提取的参数。使用以下命令访问提取的对象值：

```
results, output = ins.segmentImage("image.jpg", show_bboxes=True, output_image_na

# 从结果中访问提取的对象
results["extracted_objects"]
```

- save_extracted_objects（保存提取的对象）：这是将每个提取的对象保存为图像的参数。每个分段对象将保存为分段对象索引，例如分段对象1。对象按提取顺序保存。

```
segmented_object_1.jpg
segmented_object_2.jpg
segmented_object_3.jpg
segmented_object_4.jpg
segmented_object_5.jpg
segmented_object_6.jpg
```

从掩码坐标提取对象

注意：图像中的所有对象都已提取，我选择仅显示其中两个对象。

边界框坐标的提取

默认提取方法从掩码的坐标中提取对象。提取只给我们提供有关对象本身的信息，不包括其周围环境。

考虑一个问题，我们要分析它在图像中的对象和区域。PixelLib能够通过使用边界框坐标提取分割对象及其在图像中的远程位置来解决此问题。

修改提取代码

```
ins.segmentImage("image.jpg", extract_segmented_objects = True, extract_from_box
```

我们引入了一个新的参数extract_from_box来提取从边界框坐标分割的对象。每个提取的对象将保存为对象提取索引，例如对象提取1。对象按提取顺序保存。

```
object_extract1.jpg
object_extract2.jpg
object_extract3.jpg
object_extract4.jpg
object_extract5.jpg
object_extract6.jpg
```

使用边界框坐标提取对象的完整代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
ins.segmentImage("image.jpg", show_bboxes=True, extract_segmented_objects=True, e
save_extracted_objects=True, output_image_name="output_image.jpg" )
```

图像分割输出可视化。

PixelLib可以根据图像的分辨率调节图像的可视化。

```
ins.segmentImage("sample.jpg", show_bboxes=True, output_image_name= "output.jpg")
```


可视化不可见，因为文本大小和框厚度太小。我们可以调整文本大小、厚度和框厚度来调整可视化效果。

为更好的可视化而进行的修改

```
ins.segmentImage(“sample.jpg”, show_bboxes=True, text_size=5, text_thickness=4, b
```

segmentImage函数接受了调整文本和边框厚度的新参数。

- **text_size**：默认文本大小为0.6，中等分辨率的图像也可以。它对于高分辨率的图像来说太小了。我将值增加到5。
- **text_thickness**：默认文本厚度为1。我将其增加到4以匹配图像分辨率。
- **box_thickness**：默认长方体厚度为2，我将其更改为10以匹配图像分辨率。

以更好的可视化效果输出图像

注意：根据图像的分辨率调整参数。如果图像分辨率较低，我用于分辨率为5760 x 3840的示例图像的值可能太大。如果有分辨率非常高的图像，则可以将参数值增加到我在本示例代码中设置的参数值之外。

text_thickness和box_thickness参数的值必须是整数，不能用浮点数表示。文本大小值可以用整数和浮点数表示。

批处理图像分割

PixelLib可以对位于同一文件夹中的一批图像执行预测。

批量分割代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
ins.segmentBatch("inputfolder", show_bboxes=True, output_folder_name="outputfolde
```

```
ins.segmentBatch("inputfolder",  show_bboxes=True, output_folder_name = "outputfo
```

批量分割的代码与单幅图像分割非常相似，只是我们用segmentBatch函数替换了segmentImage函数。

我们向segmentBatch添加了以下参数：

- **folder_path**：这是包含我们要分割的图像的文件夹。
- **output_folder_name**：这是我们将保存所有分割图像的文件夹的名称。

示例文件夹结构

```
--input_folder
    --test1.jpg
    --test2.jpg
    --test3.jpg

--output_folder
    --test1.jpg
    --test2.jpg
    --test3.jpg
```

批处理图像分割中的对象提取代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
```

```
ins.segmentBatch("inputfolder", show_bboxes=True, extract_segmented_objects=True,
save_extracted_objects=True, output_folder_name="outputfolder")
```

```
ins.segmentBatch("inputfolder", show_bboxes=True, extract_segmented_objects=True,
```

我们在segmentBatch函数中添加了**extract_segmented_objects**和**save_extracted_objects**参数，分别用于提取和保存提取的对象。输入文件夹中每个图像的提取对象将保存在单独的文件夹中，其名称为imagename_extracts。例如，如果图像名称为test1.jpg，这意味着提取的对象将保存在名为test1_extracts的文件夹中。

注意：提取对象的文件夹是在图像的同一输入文件夹中创建的。

```
--input_folder
    --test1.jpg
    --test1_extracts

    --test2.jpg
    --test2_extracts

    --test3.jpg
    --test3_extracts

--output_folder
    --test1.jpg
    --test2.jpg
    --test3.jpg
```

从边界框坐标提取对象的代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
ins.segmentBatch("inputfolder", show_bboxes=True, extract_segmented_objects=True,
save_extracted_objects=True, output_folder_name="outputfolder")
```

```
ins.segmentBatch("inputfolder", show_bboxes=True, extract_segmented_objects=True,
```

我们在segmentBatch函数中添加了**extract_from_box**和**save_extracted_objects**参数，以分别提取和保存提取的对象。

注意：从边界框坐标提取的对象的文件夹也在图像的同一输入文件夹中创建。示例文件夹结构

```
--input_folder
    --test1.jpg
    --test1_extracts

    --test2.jpg
    --test2_extracts

    --test3.jpg
    --test3_extracts

--output_folder
    --test1.jpg
    --test2.jpg
    --test3.jpg
```

批处理图像分割中自定义对象分割的代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
target_classes = ins.select_target_classes(person = True)
ins.segmentBatch("inputfolder", show_bboxes=True, segment_target_classes = target
```

```
target_classes = ins.select_target_classes(person = True)
ins.segmentBatch("inputfolder", show_bboxes=True, segment_target_classes = target
```

我们调用函数select_target_classes来选择要分割的目标对象。函数segmentBatch函数获得了一个新的参数segmente_target_classes，可以从目标类中进行选择，并根据它们过滤检测。

批处理图像分割中的快速模式检测代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl", detection_speed = "fast")
ins.segmentBatch("inputfolder", show_bboxes=True, output_folder_name="outputfolde
```

```
ins.load_model(“pointrend_resnet50.pkl”, detection_speed = "fast")
```

在load_model函数中，我们添加了参数detection_speed并将该值设置为fast。快速模式处理单个图像的时间达到0.20秒。

批处理图像分割中的快速模式检测代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl", detection_speed = "rapid")
ins.segmentBatch("inputfolder", show_bboxes=True, output_folder_name="outputfolde
```

```
ins.load_model(“pointrend_resnet50.pkl”, detection_speed = "rapid")
```

我们将推理的检测速度设置为快速模式。

在load_model函数中，我们添加了参数detection_speed并将该值设置为rapid。快速模式处理单个图像的时间达到0.15秒。

注：PixelLib所有支持功能的批处理图像分割的代码实现与单个图像分割相同，只是segmentImage函数被segmentBatch替换。

视频和摄影机中的对象分割

PixelLib使得在实时摄像机和视频文件中执行实时对象分割成为可能。

视频分割

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
ins.process_video("sample_video.mp4", show_bboxes=True, frames_per_second=3, outp
```

第1-4行：导入了PixelLib包，我们还从模块PixelLib.torchbackend.instance导入了instanceSegmentation类（从pytorch支持导入实例分段类）。我们创建了该类的一个实例，并最终加载了PointRend模型。如果模型尚未下载，请从此处下载。

https://github.com/ayoolaolafenwa/PixelLib/releases/download/0.2.0/pointrend_resnet50.pkl

第5行：我们调用函数process_video对视频中的对象进行分割，并将以下参数添加到该函数中：

- **video_path**：这是要分割的视频的路径。
- **show_bboxes**：这是一个可选参数，用于在结果中显示带边界框的分段对象。
- **frames_per_second**：此参数将设置已保存视频的每秒帧数。
- **output_video_name**：这是输出分段视频的名称。

```
ins.process_video("sample_video.mp4", show_bboxes=True, frames_per_second=3, outp
```

https://youtu.be/o4Ies6YEces

视频中的对象提取代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
ins.process_video("sample.mp4", show_bboxes=True, extract_segmented_objects=True,
save_extracted_objects=True, frames_per_second=3, output_video_name="output_video
```

```
ins.process_video(“sample_video.mp4”, show_bboxes=True,  extract_segmented_object
```

process_video函数具有新参数extract_segmented_objects和save_extracted_objects，分别用于提取和保存分割对象。

视频中边界框坐标的提取

修改提取代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
ins.process_video("sample.mp4", show_bboxes=True, extract_segmented_objects=True,
save_extracted_objects=True, frames_per_second=5, output_video_name="output_video
```

```
ins.process_video(“sample.mp4”, show_bboxes=True, extract_segmented_objects=True,
```

- 将extract_from_box添加到函数中，以提取从其边界框坐标分割的对象。

视频中的自定义对象分割

PixelLib可以在视频中执行自定义对象分割，以过滤未使用的检测并分割目标类。

视频中的自定义检测代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl")
target_classes = ins.select_target_classes(person = True, bicycle =True)
ins.process_video("sample_video.mp4", show_bboxes=True, segment_target_classes =
frames_per_second=5, output_video_name="output_video.mp4")
```

```
target_classes = ins.select_target_classes(person = True, bicycle =True)

ins.process_video("sample_video.mp4", show_bboxes=True, segment_target_classes =
```

调用函数select_target_classes来选择要分割的目标对象。函数process_video获得了一个新的参数segment_target_classes，可以从目标类中进行选择，并基于它们过滤检测。

https://youtu.be/dVde5fx654Y

视频分割中的快速模式检测代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
```

```
ins.load_model("pointrend_resnet50.pkl", detection_speed = "fast")
ins.process_video("sample_video.mp4", show_bboxes=True, frames_per_second=5, outp
```

```
ins.load_model(“pointrend_resnet50.pkl”, detection_speed = "fast")
```

我们将视频处理的检测速度设置为快速模式。

视频分割中的快速模式检测代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation

ins = instanceSegmentation()
ins.load_model("pointrend_resnet50.pkl", detection_speed = "rapid")
ins.process_video("sample_video.mp4", show_bboxes=True, frames_per_second=5, outp
```

```
ins.load_model(“pointrend_resnet50.pkl”, detection_speed = "rapid")
```

我们将视频处理的检测速度设置为快速模式。

实时摄影机中的对象分割

PixelLib为实时分割实时摄影机提供了极好的支持。

用于分割实时摄影机的代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation
import cv2

capture = cv2.VideoCapture(0)

segment_video = instanceSegmentation()
segment_video.load_model("pointrend_resnet50.pkl")
segment_video.process_camera(capture,  show_bboxes = True, frames_per_second= 5,
frame_name= "frame", output_video_name="output_video.mp4")
```

```
import cv2 capture = cv2.VideoCapture(0)
```

我们导入了cv2并包含了捕获相机帧的代码。

```
segment_video.process_camera(capture,  show_bboxes = True, frames_per_second= 5,
```

在执行分割的代码中，我们将视频的文件路径替换为捕获，也就是说，我们正在处理摄像机捕获的帧流。我们添加了额外的参数以显示相机的帧：

- show_frames：该参数用于处理分段摄影机帧的显示。
- frame_name：这是为显示的摄影机帧指定的名称。
- check_fps：这是在相机馈送处理结束时每秒打印帧数的参数。
- show_bboxes：这是一个可选参数，用于显示带有边界框的分段对象。
- frames_per_second：此参数用于设置保存的视频文件每秒帧数。在这种情况下，它被设置为5，即保存的视频文件每秒有5帧。
- output_video_name：这是保存的分段视频的名称。

实时相机馈送处理的速度调整

默认速度模式达到4fps。快速模式达到6fps，快速模式达到9fps。这些报告基于使用4GB容量的Nvidia GPU

摄像机馈送中的快速模式检测代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation
import cv2

capture = cv2.VideoCapture(0)

segment_video = instanceSegmentation()
segment_video.load_model("pointrend_resnet50.pkl", detection_speed = "fast")
```

```
segment_video.process_camera(capture,  show_bboxes = True, frames_per_second= 5,
frame_name= "frame", output_video_name="output_video.mp4")
```

```
ins.load_model(“pointrend_resnet50.pkl”, detection_speed = "fast")
```

我们将处理实时摄像机馈送的检测速度设置为快速模式，推断速度为6fps。

快速模式检测代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation
import cv2

capture = cv2.VideoCapture(0)

segment_video = instanceSegmentation()
segment_video.load_model("pointrend_resnet50.pkl", detection_speed = "rapid")
segment_video.process_camera(capture,  show_bboxes = True, frames_per_second= 5,
frame_name= "frame", output_video_name="output_video.mp4")
```

```
ins.load_model(“pointrend_resnet50.pkl”, detection_speed = "rapid")
```

我们将处理实时摄像机馈送的检测速度设置为快速模式，推断速度为9fps。

实时摄影机中自定义对象分割的代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation
import cv2

capture = cv2.VideoCapture(0)

segment_video = instanceSegmentation()
segment_video.load_model("pointrend_resnet50.pkl")
target_classes = segment_video.select_target_classes(person=True)
segment_video.process_camera(capture,  show_bboxes = True, frames_per_second= 5,
show_frames= True,frame_name= "frame", output_video_name="output_video.mp4")
```

```
target_classes = segment_video.select_target_classes(person=True)

segment_video.process_camera(capture,  show_bboxes = True, frames_per_second= 5,
```

调用函数select_target_classes来选择要分割的目标对象。函数process_camera获得了一个新的参数segment_target_classes，可以从目标类中进行选择，并基于它们过滤检测。

实时摄影机中的对象提取代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation
import cv2

capture = cv2.VideoCapture(0)

segment_video = instanceSegmentation()
segment_video.load_model("pointrend_resnet50.pkl")
segment_video.process_camera(capture,  show_bboxes = True, frames_per_second= 5,
show_frames= True,frame_name= "frame", output_video_name="output_video.mp4")
```

```
segment_video.process_camera(capture,  show_bboxes = True, frames_per_second= 5,
```

process_camera函数具有新参数extract_segmented_Object和save_extracted_Object，分别用于提取和保存分割对象。

- 将extract_from_box添加到函数中，以提取从其边界框坐标分割的对象。

从实时摄影机中的框坐标提取对象的代码

```
import pixellib
from pixellib.torchbackend.instance import instanceSegmentation
import cv2

capture = cv2.VideoCapture(0)
```

```
segment_video = instanceSegmentation()
segment_video.load_model("pointrend_resnet50.pkl")
segment_video.process_camera(capture,  show_bboxes = True, frames_per_second= 5,
save_extracted_objects=True, show_frames= True,frame_name= "frame", output_video_
```

```
segment_video.process_camera(capture,  show_bboxes = True, frames_per_second= 5,
```

- 将extract_from_box添加到函数中，以提取从其边界框坐标分割的对象。

在本文中，我们详细讨论了如何使用PixelLib Pytorch版本对图像和实时摄影机中的对象执行准确、快速的图像分割和提取。我们还介绍了使用PointRend网络体系结构添加到PixelLib的升级，这使得该库能够满足不断增长的需求，以平衡计算机视觉中的精度和速度性能。

注意：PixelLib pytorch版本不支持自定义训练，使用PointRend的自定义训练将很快发布。

访问PixelLib的官方github存储库：https://github.com/ayoolaolafenwa/PixelLib

访问PixelLib的官方文档：https://pixellib.readthedocs.io/en/latest/

查看这些关于如何使用PixelLib进行语义分割、实例分割、对象提取以及图像和视频中的背景编辑的文章。

https://towardsdatascience.com/video-segmentation-with-5-lines-of-code-87f798afb93

https://towardsdatascience.com/semantic-segmentation-of-150-classes-of-objects-with-5-lines-of-code-7f244fa96b6c

https://towardsdatascience.com/change-the-background-of-any-image-with-5-lines-of-code-23a0ef10ce9a

https://towardsdatascience.com/change-the-background-of-any-video-with-5-lines-of-code-7cc847394f5d

https://towardsdatascience.com/extraction-of-objects-in-images-and-videos-using-5-lines-of-code-6a9e35677a31

☆ END ☆

如果看到这里，说明你喜欢这篇文章，请转发、点赞。微信搜索「uncle_pn」，欢迎添加小编微信「woshicver」，每日朋友圈更新一篇高质量博文。

↓扫描二维码添加小编↓

喜欢此内容的人还喜欢

【工程化】1191- 结合代码实践，全面学习前端工程化

前端自习课