系列文章

OpenVINO™ 工具套件

OpenVINO™工具套件 实现图像实例分割

实例分割概念

图像实例分割是在对象检测的基础上进一步细化，分离对象的前景与背景，实现像素级别的对象分离。所以图像实例分割是基于对象检测的基础上进一步提升。图像实例分割在目标检测、人脸检测、表情识别、医学图像处理与疾病辅助诊断、视频监控与对象跟踪、零售场景的货架空缺识别等场景下均有应用。很多人会把图像语义分割跟实例分割搞混淆、其实图像的语义分割(Semantic Segmentation)与图像的实例分割(Instance Segmentation)是两个不同的概念，看下图:

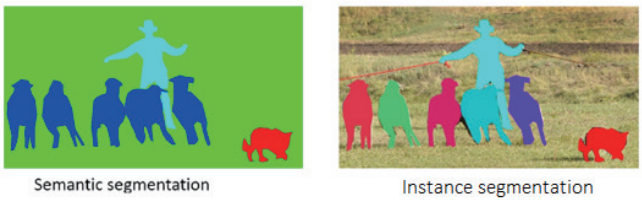


图1(来自COCO数据集论文)

左侧是图像语义分割的结果，几个不同的类别人、羊、狗、背景分别用不同的颜色表示；右侧是图像实例分割的结果，对每只羊都用不同的颜色表示，而且把每个对象从背景中分离出来。这个就是语义分割跟实例分割的区别，直白点可以说就是语义分割是对每个类别、实例分割是针对每个对象（多个对象可能属于同一个类别）。

常见的实例分割网络

1.Mask-RCNN实例分割网络

图像实例分割是在对象检测的基础上再多出个基于ROI的分割分支，基于这样思想的实例分割MasK-RCNN就是其经典代表，它的网络结构如下:

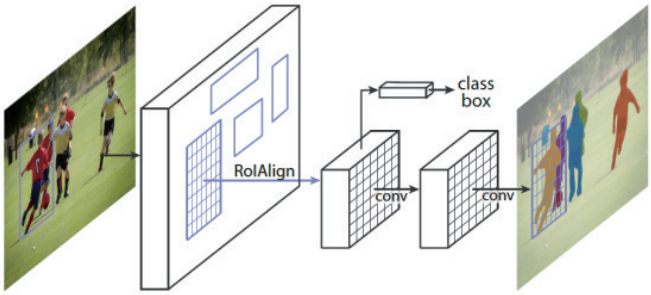


图2(来自Mask-RCNN的论文)

Mask-RCNN可以简单的认为是Faster-RCNN得基础上加上一个实例分割分支。

2.RetinaMask实例分割网络

RetinaMask可以看出RetinaNet对象检测网络跟Mask-RCNN实例分割网络的两个优势组合，基于特征金字塔实现了更好的Mask预测，网络结构图示如下:

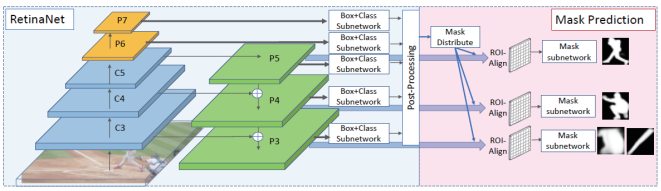


图3(来自RetinaMask论文)

3.PANet实例分割网络

PANet主要工作是基于Mask-RCNN网络上改进所得，作者通过改进Backbone部分提升了特征提取能力，通过自适应的池化操作得到更多融合特征，基于全链接融合产生mask，最终取得了比Mask-RCNN更好的实例分割效果，该模型的结构如下：

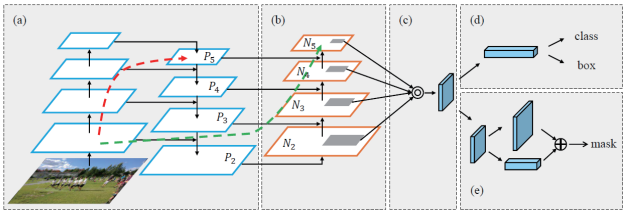


Illustration of our framework. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling. (d) branch. (e) Fully-connected fusion. Note that we omit channel dimension of feature maps in (a) and (b) for brevity.

图4(来自PANet论文)

其中全链接特征融合mask分支如下图：

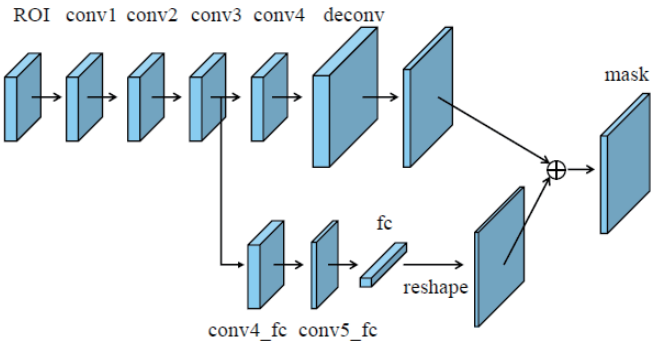
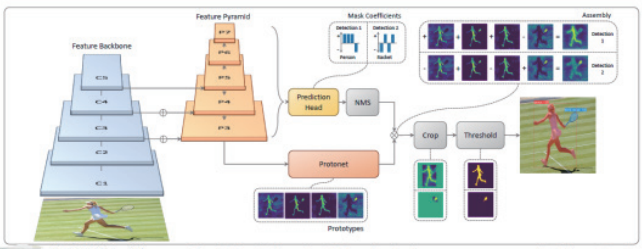


图5(来自PANet论文)

4.YOLACT实例分割网络

该实例分割网络也是基于RetinaNet对象检测网络的基础上，添加一个Mask分支，不过在添加Mask分支的时候它的Mask分支设计跟RetinaMask有所不同，该网络的结构图示如下：



YOLACT Architecture Blue/yellow indicates low/high values in the prototypes, gray nodes indicate functions that are not trained, and $k = 4$ in this example. We base this architecture off of RetinaNet [27] using ResNet-101 + FPN.

图5(来自YOLACT作者论文)

5.CenterMask实例分割网络

该实例网络是基于FCOS对象检测框架的基础上，设计一个Mask分支输出，该Mask分支被称为空间注意力引导蒙板(Spatial Attention Guided Mask)，该网络的结构如下：

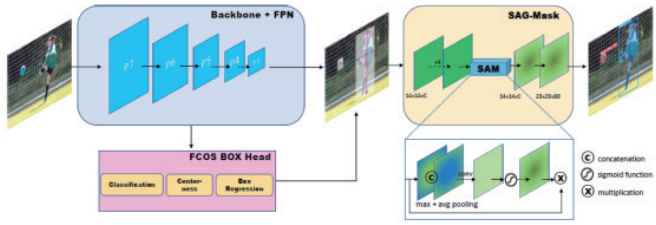


图7(来自CenterMask论文)

OpenVINO™ 支持Mask-RCNN模型

OpenVINO™ 中支持两种实例分割模型分别是Mask-RCNN与YOLACT模型，其中Mask-RCNN模型支持来自intel官方库文件、而YOLACT则来自公开的第三方提供。我们这里以官方的Mask-RCNN模型instance-segmentation-security-0050为例说明，该模型基于COCO数据集训练，支持80个类别的实例分割，加上背景为81个类别。

OpenVINO™ 支持部署Faster-RCNN与Mask-RCNN网络时候输入的解析都是基于两个输入层，它们分别是：

im_data : NCHW=[1x3x480x480]

im_info: 1x3 三个值分别是H、W、Scale=1.0

输出有四个，名称与输出格式及解释如下：

name: classes, shape: [100,] 预测的100个类别可能性，值在[0~1]之间

name: scores: shape: [100,] 预测的100个Box可能性，值在[0~1]之间

name: boxes, shape: [100, 4] 预测的100个Box坐标，左上角与右下角，基于输入的480x480

name: raw_masks, shape: [100, 81, 28, 28] Box ROI区域的实例分割输出，81表示类别（包含背景），28x28表示ROI大小，注意：此模型输出大小为14x14

OpenVINO™ 支持Mask-RCNN模型

因为模型的加载与推理部分的代码跟前面系列文章的非常相似，这里就不再给出。代码演示部分重点在输出的解析，为了简化，我用了两个for循环设置了输入与输出数据精度，然后直接通过hardcode的输出层名称来获取推理之后各个输出层对应的数据部分，首先获取类别，根据类别ID与Box的索引，直接获取实例分割mask，然后随机生成颜色，基于mask实现与原图BOX ROI的叠加，产生了实例分割之后的效果输出。解析部分的代码首先需要获取推理以后的数据，获取数据的代码如下：

```
float w_rate = static_cast<float>(im_w) / 480.0;
float h_rate = static_cast<float>(im_h) / 480.0;

auto scores = infer_request.GetBlob("scores");
auto boxes = infer_request.GetBlob("boxes");
auto clazzes = infer_request.GetBlob("classes");
auto raw_masks = infer_request.GetBlob("raw_masks");
const float* score_data = static_cast<PrecisionTrait<Precision::FP32>::value_type*>(scores->buffer());
const float* boxes_data = static_cast<PrecisionTrait<Precision::FP32>::value_type*>(boxes->buffer());
const float* clazzes_data = static_cast<PrecisionTrait<Precision::FP32>::value_type*>(clazzes->buffer());
const auto raw_masks_data = static_cast<PrecisionTrait<Precision::FP32>::value_type*>(raw_masks->buffer());
const SizeVector scores_outputDims = scores->getTensorDesc().getDims();
const SizeVector boxes_outputDims = boxes->getTensorDesc().getDims();
const SizeVector mask_outputDims = raw_masks->getTensorDesc().getDims();
const int max_count = scores_outputDims[0];
const int object_size = boxes_outputDims[1];
printf("mask NCHW=[%d, %d, %d, %d]\n", mask_outputDims[0], mask_outputDims[1], mask_outputDims[2], mask_outputDims[3]);
int mask_h = mask_outputDims[2];
int mask_w = mask_outputDims[3];
size_t box_stride = mask_h * mask_w * mask_outputDims[1];
```

然后根据输出数据格式开始解析Box框与Mask，这部分的代码如下：

```
for (int n = 0; n < max_count; n++) {
    float confidence = score_data[n];
    float xmin = boxes_data[n*object_size] * w_rate;
    float ymin = boxes_data[n*object_size + 1] * h_rate;
    float xmax = boxes_data[n*object_size + 2] * w_rate;
    float ymax = boxes_data[n*object_size + 3] * h_rate;
    if (confidence > 0.5) {

cv::Scalar color(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));

cv::Rect box;
    float x1 = std::min(std::max(0.0f, xmin), static_cast<float>(im_w));
    float y1 = std::min(std::max(0.0f, ymin), static_cast<float>(im_h));
    float x2 = std::min(std::max(0.0f, xmax), static_cast<float>(im_w));
```

```
float y2 = std::min(std::max(0.0f, ymax), static_cast<float>(im_h));
box.x = static_cast<int>(x1);
box.y = static_cast<int>(y1);

box.width = static_cast<int>(x2 - x1);

box.height = static_cast<int>(y2 - y1);
int label = static_cast<int>(clazzes_data[n]);

std::cout << "confidence: " << confidence << " class name: " << coco_labels[label] << std::endl;
// 解析mask
float* mask_arr = raw_masks_data + box_stride * n + mask_h * mask_w * label;

cv::Mat mask_mat(mask_h, mask_w, CV_32FC1, mask_arr);

cv::Mat roi_img = src(box);

cv::Mat resized_mask_mat(box.height, box.width, CV_32FC1);

cv::resize(mask_mat, resized_mask_mat, cv::Size(box.width, box.height));


cv::Mat uchar_resized_mask(box.height, box.width, CV_8UC3, color);


roi_img.copyTo(uchar_resized_mask, resized_mask_mat <= 0.5);
cv::addWeighted(uchar_resized_mask, 0.7, roi_img, 0.3, 0.0f, roi_img);

cv::putText(src, coco_labels[label].c_str(), box.tl() + (box.br() - box.tl()) / 2, cv::FONT_HERSHEY_PLAIN, 1.0, cv::Scalar(0, 0, 255), 1, 8);
}
}
```

其中Mask部分的时候有个技巧的地方，首先获取类别，然后根据类别，直接获取Mask中对应的通道数据生成二值Mask图像，添加上颜色，加权混合到ROI区域即可得到输出结果。最终的代码运行如下：

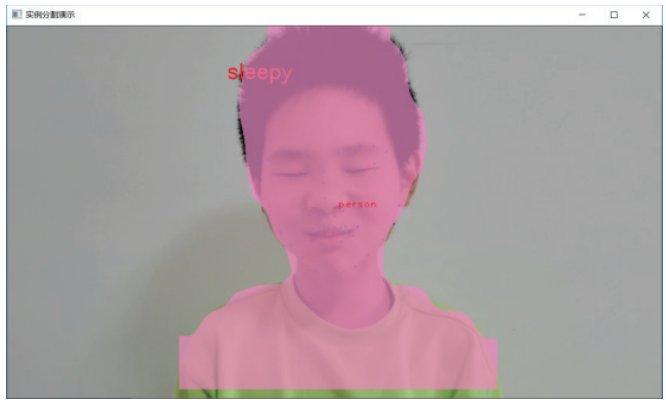


图3(实例分割)

如欲了解更多OpenVINO™开发资料，
请扫描下方二维码，我们会把最新资料及时推送给您。

请访问www.Intel.com/PerformanceIndex了解负载及参数。结果可能不同。
性能结果基于截至配置中显示的日期的测试，可能无法反映所有公开可用的更新。有关配置的详细信息，请参见备份。没有任何产品或组件能够做到绝对安全。
成本及结果均不同。
英特尔技术可能需要支持的硬件、软件或服务得以激活。
英特尔并不控制或审计第三方数据。请您咨询其他来源，并确认提及数据是否准确。