# 目录

基于 OpenVINO 在 C++中部署 YOLOv5-Seg 实例分割模型 ..... 1
1.1 配置 OpenVINO C++开发环境 ..... 1
1.2 下载并转换 YOLOv5 预训练模型 ..... 1
1.3 使用 OpenVINO Runtime C++ API 编写推理程序 ..... 1
1.3.1 采集图像&图像解码 ..... 2
1.3.2 YOLOv5-Seg 模型的图像预处理 ..... 2
1.3.3 执行 AI 推理计算 ..... 4
1.3.4 推理结果进行后处理 ..... 5
1.4 总结 ..... 5

# 基于 OpenVINO 在 C++中部署 YOLOv5-Seg 实例分割模型

作者：英特尔物联网行业创新大使 王一凡

YOLOv5 兼具速度和精度，工程化做的特别好，Git clone 到本地即可在自己的数据集上实现目标检测任务的训练和推理，在产业界中应用广泛。开源社区对 YOLOv5 支持实例分割的呼声高涨，YOLOv5 在 v7.0 中正式官宣支持实例分割。

在前期文章中，已发布基于 OpenVINO 的 YOLOv5 模型的 Python 版本和 C++版本推理程序，以及 YOLOv5-Seg 模型的 Python 版推理程序，本文主要介绍在 C++中使用 OpenVINO 工具包部署 YOLOv5-Seg 模型，主要步骤有：

1. 配置 OpenVINO C++开发环境
2. 下载并转换 YOLOv5-Seg 预训练模型
3. 使用 OpenVINO Runtime C++ API 编写推理程序

下面，本文将依次详述

## 1.1 配置 OpenVINO C++开发环境

配置 OpenVINO C++开发环境的详细步骤，请参考《在 Windows 中基于 Visual Studio 配置 OpenVINO C++开发环境》。

## 1.2 下载并转换 YOLOv5 预训练模型

下载并转换 YOLOv5-seg 预训练模型的详细步骤，请参考：《在英特尔独立显卡上部署 YOLOv5 v7.0 版实时实例分割模型》，本文所使用的 OpenVINO 是 2022.3 LTS 版。

首先，运行命令获得 yolov5s-seg ONNX 格式模型：yolov5s-seg.onnx：

```
python export.py --weights yolov5s-seg.pt --include onnx
```

然后运行命令获得 yolov5s-seg IR 格式模型：yolov5s-seg.xml 和 yolov5s-seg.bin，如下图所示

```
mo -m yolov5s-seg.onnx --compress_to_fp16
```

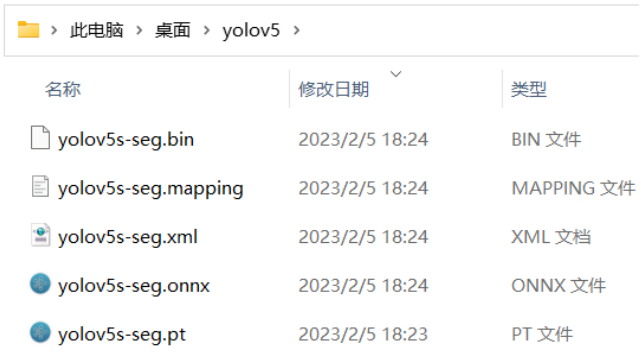


图 1-1 yolov5-seg ONNX 格式和 IR 格式模型

## 1.3 使用 OpenVINO Runtime C++ API 编写推理程序

一个端到端的 AI 推理程序，主要包含五个典型的处理流程：

1. 采集图像&图像解码
2. 图像数据预处理
3. AI 推理计算

4. 对推理结果进行后处理
5. 将处理后的结果集成到业务流程


图 1-2 端到端的 AI 推理程序处理流程

### 1.3.1 采集图像&图像解码

OpenCV 提供 imread()函数将图像文件载入内存，

```
Mat cv::imread (const String &filename, int flags=IMREAD_COLOR)
```

若是从视频流(例如，视频文件、网络摄像头、3D 摄像头(Realsense)等)中，一帧一帧读取图像数据到内存，则使用 cv::VideoCapture 类，对应范例代码请参考 OpenCV 官方范例代码：https://github.com/opencv/opencv/tree/4.x/samples/cpp。

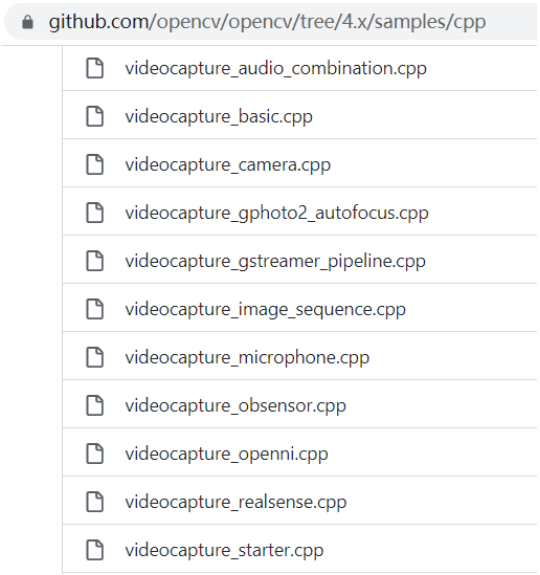


图 1-3 从视频流读取图像帧范例

### 1.3.2 YOLOv5-Seg 模型的图像预处理

YOLOv5-Seg 模型构架是在 YOLOv5 模型构架基础上，增加了一个叫“Proto”的小型卷积神经网络，用于输出检测对象掩码(Mask)，如下图所示：

```
92  class Segment(Detect):
93      # YOLOv5 Segment head for segmentation models
94      def __init__(self, nc=80, anchors=(), nm=32, npr=256, ch=(), inplace=True):
95          super().__init__(nc, anchors, ch, inplace)
96          self.nm = nm  # number of masks
97          self.npr = npr  # number of protos
98          self.no = 5 + nc + self.nm  # number of outputs per anchor
99          self.m = nn.ModuleList(nn.Conv2d(x, self.no * self.na, 1) for x in ch)  # output conv
100         self.proto = Proto(ch[0], self.npr, self.nm)  # protos
101         self.detect = Detect.forward
102
103     def forward(self, x):
104         p = self.proto(x[0])
105         x = self.detect(self, x)
106         return (x, p) if self.training else (x[0], p) if self.export else (x[0], p, x[1])
```

图 1-4　YOLOv5-Seg 模型输出的代码定义

详细参看：https://github.com/ultralytics/yolov5/blob/master/models/yolo.py#L92

由此可知，YOLOv5-Seg 模型对数据预处理的要求跟 YOLOv5 模型一模一样，YOLOv5-Seg 模型的预处理代码可以复用 YOLOv5 模型的 C++预处理代码。

另外，从代码可以看出 YOLOv5-Seg 模型的输出有两个张量，一个张量输出检测结果，一个张量输出 proto，其形状可以用 Netron 打开 yolov5-seg.onnx 查知，如下图所示。

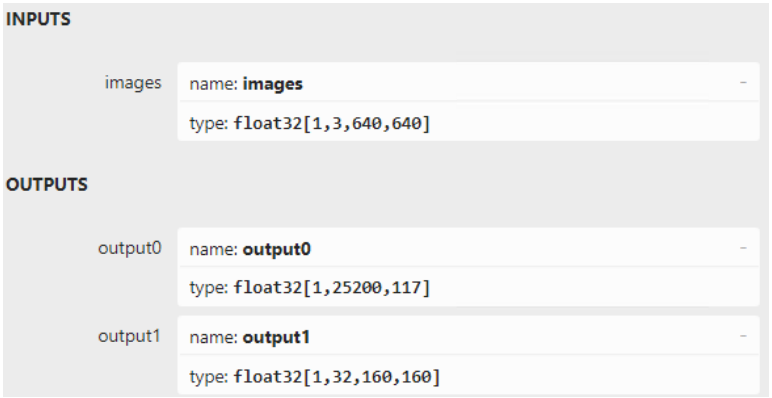


图 1-5　YOLOv5-Seg 模型的输入和输出

“output0”是检测输出，第一个维度表示 batch size，第二个维度表示 25200 条输出，第三个维度表示有 117 个字段，其中前 85 个字段(0~84)表示：cx、cy、w、h、confidence 和 80 个类别分数，后 32 个字段与”output1”做矩阵乘法，可以获得尺寸为 160x160 的检测目标的掩码(mask)，如下图所示。

图 1-6　检测目标的掩码

### 1.3.3 执行 AI 推理计算

基于 OpenVINO Runtime C++ API 实现 AI 推理计算主要有两种方式：一种是同步推理方式，一种是异步推理方式，本文主要介绍同步推理方式。

主要步骤有：

1. 初始化 Core 类：`ov::Core core;`
2. 编译模型：`core.compile_model()`
3. 创建推理请求 infer_request：`compiled_model.create_infer_request()`
4. 读取图像数据并做预处理：`letterbox()`
5. 将预处理后的 blob 数据传入模型输入节点：`infer_request.set_input_tensor()`
6. 调用 infer()方法执行推理计算：`infer_request.infer()`
7. 获得推理结果：`infer_request.get_output_tensor()`

基于 OpenVINO Runtime C++API 的同步推理代码如下所示：

```
    // -------- Step 1. Initialize OpenVINO Runtime Core --------
    ov::Core core;
    // -------- Step 2. Compile the Model --------
    auto compiled_model = core.compile_model(model_file, "GPU.1"); //GPU.1 is dGPU A770
    // -------- Step 3. Create an Inference Request --------
    ov::InferRequest infer_request = compiled_model.create_infer_request();
    // -------- Step 4. Read a picture file and do the preprocess --------
    cv::Mat img = cv::imread(image_file); //Load a picture into memory
    std::vector<float> paddings(3);       //scale, half_h, half_w
    cv::Mat resized_img = letterbox(img, paddings); //resize to (640,640) by letterbox
    // BGR->RGB, u8(0-255)->f32(0.0-1.0), HWC->NCHW
    cv::Mat blob = cv::dnn::blobFromImage(resized_img, 1 / 255.0, cv::Size(640, 640),
cv::Scalar(0, 0, 0), true);
    // -------- Step 5. Feed the blob into the input node of YOLOv5 -------
    // Get input port for model with one input
    auto input_port = compiled_model.input();
    // Create tensor from external memory
    ov::Tensor input_tensor(input_port.get_element_type(), input_port.get_shape(),
blob.ptr(0));
    // Set input tensor for model with one input
    infer_request.set_input_tensor(input_tensor);
    // -------- Step 6. Start inference --------
    infer_request.infer();
    // -------- Step 7. Get the inference result --------
    auto detect = infer_request.get_output_tensor(0);
```

```
auto detect_shape = detect.get_shape();
std::cout << "The shape of Detection tensor:"<< detect_shape << std::endl;
auto proto = infer_request.get_output_tensor(1);
auto proto_shape = proto.get_shape();
std::cout << "The shape of Proto tensor:" << proto_shape << std::endl;
```

### 1.3.4 推理结果进行后处理

后处理工作主要是从”detect ”输出张量中拆解出检测框的位置和类别信息，并用cv::dnn::NMSBoxes()过滤掉多于的检测框；从”detect ”输出张量的后 32 个字段与”proto”输出张量做矩阵乘法，获得每个检测目标的形状为 160x160 的掩码输出，最后将 160x160 的掩码映射回原始图像完成所有后处理工作。

完整的代码实现，请下载：https://gitee.com/ppov-nuc/yolov5_infer/blob/main/yolov5seg_openvino_dGPU.cpp

## 1.4 总结

配置 OpenVINO C++开发环境后，可以直接编译运行 yolov5seg_openvino_dGPU.cpp，结果如下图所示。使用 OpenVINO Runtime C++ API 函数开发 YOLOv5 推理程序，简单方便，并可以任意部署在英特尔 CPU、集成显卡和独立显卡上。

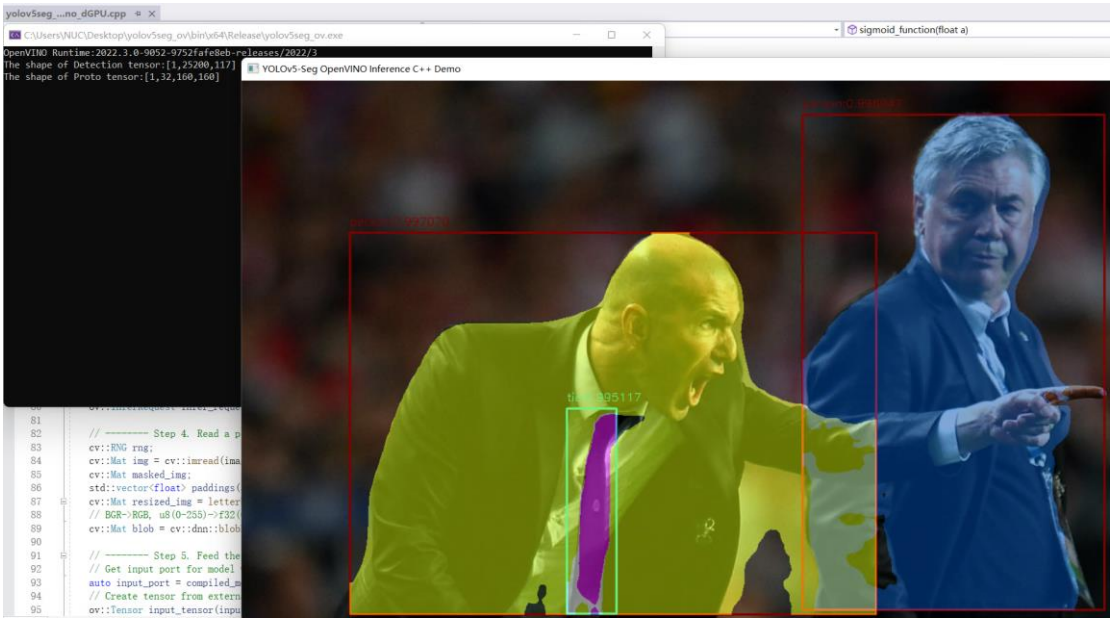


图 1-7 运行结果

，