目录

基于 OpenVINO 在 C++中部署 YOLOv5 模型

作者：英特尔物联网行业创新大使 王一凡

本文主要介绍在 C++中使用 OpenVINO 工具包部署 YOLOv5 模型，主要步骤有：

1. 配置 OpenVINO C++开发环境
2. 下载并转换 YOLOv5 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 预训练模型的详细步骤，请参考：《基于 OpenVINO™2022.2 和蝰蛇峡谷优化并部署 YOLOv5 模型》，本文所使用的 OpenVINO 是 2022.3 LTS 版。

完成上述步骤后，可以获得 YOLOv5 的 IR 模型文件：yolov5s.xml 和 yolov5s.bin，如下图所示：

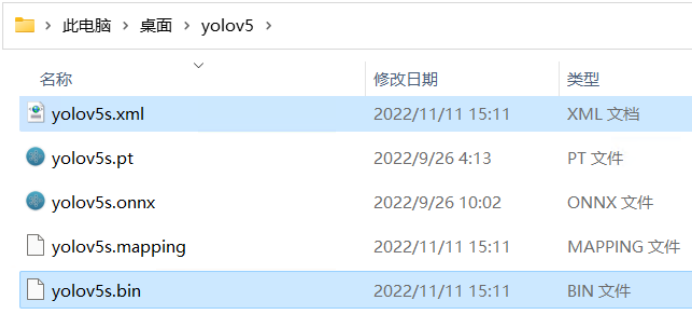

图 1-1 YOLOv5 IR 模型文件

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。

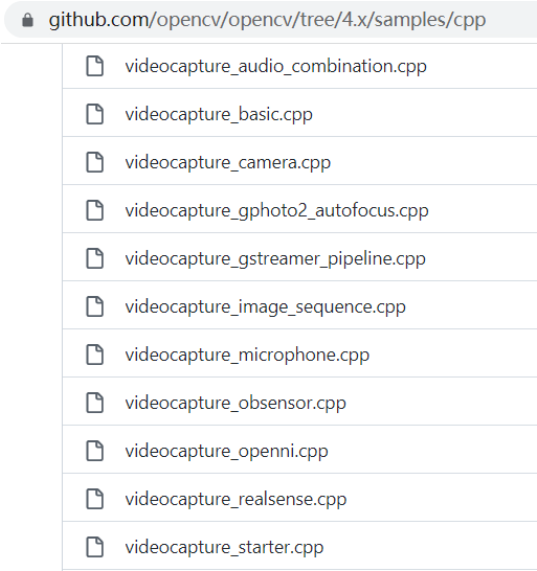

图 1-3 从视频流读取图像帧范例

1.3.2 YOLOv5 的图像预处理

图像数据输入 YOLOv5 模型前需要做预处理，其主要工作有：使用 Letterbox 算法对图像进行非变形放缩，然后完成转换颜色通道、归一化数据、更改数据布局和数值精度。

直接调用 OpenCV 的 cv::resize()函数将原始图像按照模型输入要求的尺寸进行放缩，虽然实现起来简单，但会导致图像中的被检测对象变形。Letterbox 算法一种**不会导致被检测对象变形的缩放**，主要步骤为：

1. 计算宽高缩放比例，选择较小那个缩放系数
2. 计算缩放后的尺寸，原始图片的长宽都乘以较小的缩放系数
3. 计算短边需要填充的灰边数，将短边的两边各自填充一半的灰行

参考 YOLOv5 的 Letterbox 算法实现方式，本文的 Letterbox 函数实现如下所示：

```
cv::Mat letterbox(cv::Mat& img, std::vector<int> new_shape = {640, 640}){
    // Get current image shape [height, width]
    // Refer to https://github.com/ultralytics/yolov5/blob/master/utils/augmentations.py#L111
    int img_h = img.rows;
    int img_w = img.cols;
    // Compute scale ratio(new / old) and target resized shape
```

```
    float scale = std::min(new_shape[1] * 1.0 / img_h, new_shape[0] * 1.0 / img_w);
    int resize_h = int(round(img_h * scale));
    int resize_w = int(round(img_w * scale));
    // Compute padding
    int pad_h = new_shape[1] - resize_h;
    int pad_w = new_shape[0] - resize_w;
    // Resize and pad image while meeting stride-multiple constraints
    cv::Mat resized_img;
    cv::resize(img, resized_img, cv::Size(resize_w, resize_h));
    // divide padding into 2 sides
    float half_h = pad_h * 1.0 / 2;
    float half_w = pad_w * 1.0 / 2;
    // Compute padding boarder
    int top = int(round(half_h - 0.1));
    int bottom = int(round(half_h + 0.1));
    int left = int(round(half_w - 0.1));
    int right = int(round(half_w + 0.1));
    // Add border
    cv::copyMakeBorder(resized_img, resized_img, top, bottom, left, right, 0, cv::Scalar(114,
114, 114));
    return resized_img;
}
```

letterbox 函数的运行结果如下图所示：

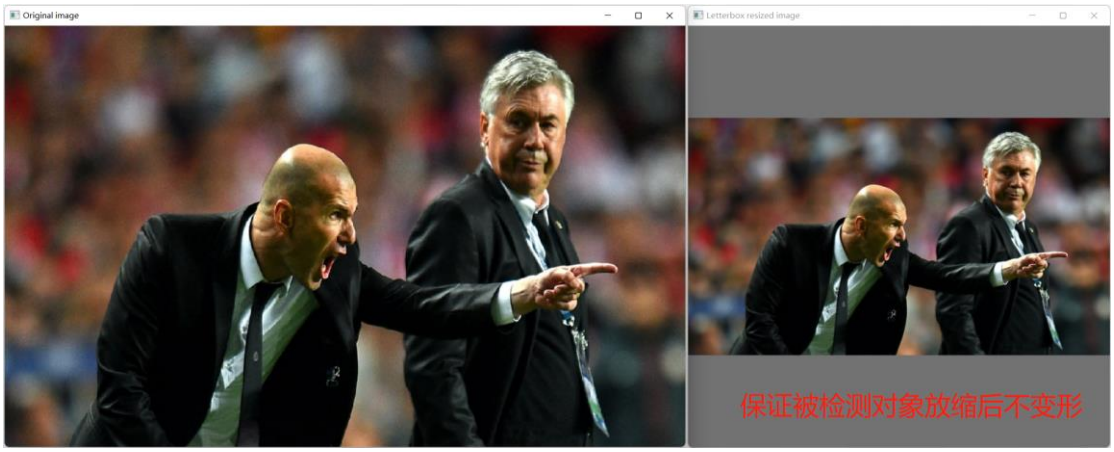

图 1-4 letterbox 放缩图片的效果

转换颜色通道、归一化数据、更改数据布局和数值精度的操作可以由 OpenCV 提供的 Mat cv::dnn::blobFromImage()函数实现，或者由 OpenVINO 的预处理 API 实现。为了简洁范例代码，本文选择调用 cv::dnn::blobFromImage()函数。

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
2. 编译模型
3. 创建推理请求 infer_request
4. 读取图像数据并做预处理
5. 将预处理后的 blob 数据传入模型输入节点
6. 调用 infer()方法执行推理计算
7. 获得推理结果

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, "CPU"); //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 output = infer_request.get_output_tensor(0);
```

```
auto output_shape = output.get_shape();

std::cout << "The shape of output tensor:"<<output_shape << std::endl;
```

1.3.4 推理结果进行后处理

对于目标检测应用，后处理主要是执行 NMS(非极大值抑制)算法去除多余的检测框，然后剩余的检测框中提取出检测框坐标(box)、置信度(confidence)和类别(class_id)。NMS 算法本文直接使用了 cv::dnn::NMSBoxes()。

经过后处理，获得了经过 NMS 过滤后的检测框坐标(box)、置信度(confidence)和类别(class_id)后，就可以将这些信息显示在图像上了。

完整的代码实现，请下载：yolov5_openvino_sync_dGPU.cpp

1.4 总结

配置 OpenVINO C++开发环境后，可以直接编译运行 yolov5_openvino_sync_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。

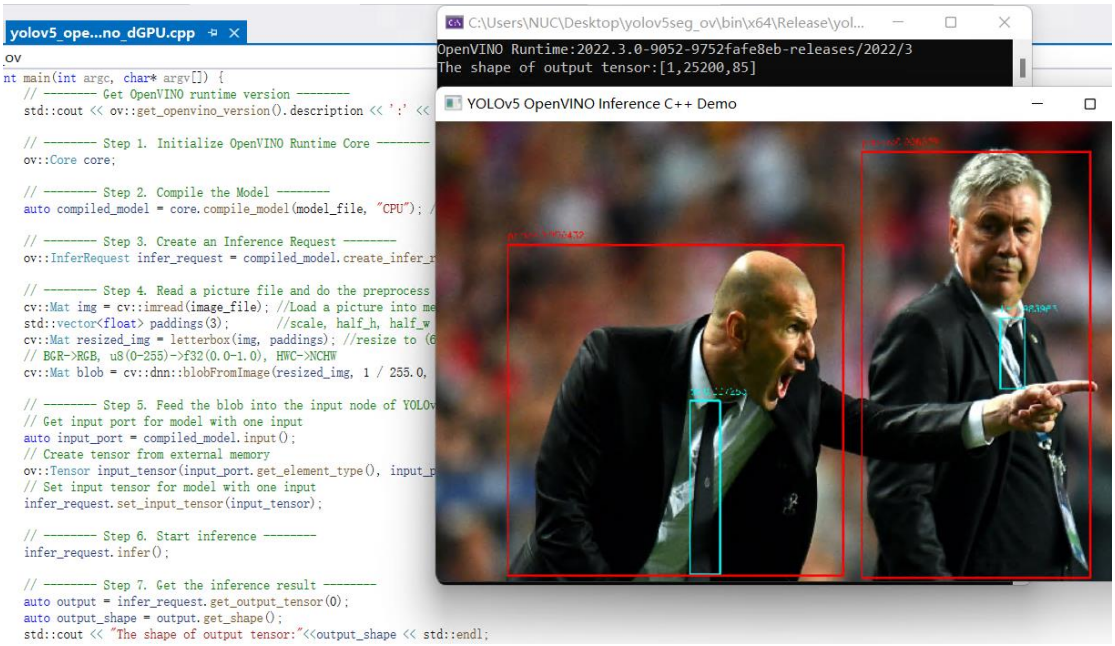

图 1-5 运行结果

，