

# 目 录

第 1 章 基于 C#和 OpenVINO2023.0 部署 YOLOv8 全系列模型 .....	1
1.1 项目简介.....	1
1.1.1 OpenVINO™ .....	1
1.1.2 YOLOv8.....	1
1.1.3 项目内容.....	2
1.2 OpenVinoSharp .....	3
1.2.1 OpenVINO™ 2023.0 安装配置 .....	3
1.2.2 C++ 动态链接库.....	4
1.2.3 C#构建 Core 推理类.....	6
1.2.4 NuGet 安装 OpenVinoSharp.....	7
1.3 获取和转换 YOLOv8 模型 .....	7
1.3.1 安装 ultralytics.....	7
1.3.2 导出 yolo v8 模型.....	8
1.3.3 安装 OpenVINO™ Python 版 .....	8
1.3.4 YOLOv8 模型量化 .....	9
1.3.5 Benchmark_app 测试 YOLOv8 模型 .....	9
1.4 OpenVinoSharp 部署 YOLOv8 模型 .....	9
1.4.1 YOLOv8-det 预测结果 .....	10
1.4.2 YOLOv8-cl s 预测结果.....	11
1.4.3 YOLOv8-seg 预测结果 .....	12
1.4.4 YOLOv8-pose 预测结果 .....	12
1.5 总结.....	13

## 第1章 基于 C#和 OpenVINO2023.0 部署 YOLOv8 全系列模型

### 1.1 项目简介

#### 1.1.1 OpenVINO™

英特尔发行版 OpenVINO™ 工具套件基于 oneAPI 而开发,可以加快高性能计算机视觉和深度学习视觉应用开发速度工具套件,适用于从边缘到云的各种英特尔平台上,帮助用户更快地将更准确的真实世界结果部署到生产系统中。通过简化的开发工作流程, OpenVINO™ 可赋能开发者在现实世界中部署高性能应用程序和算法。

在推理后端,得益于 OpenVINO™ 工具套件提供的“一次编写,任意部署”的特性,转换后的模型能够在不同的英特尔硬件平台上运行,而无需重新构建,有效简化了构建与迁移过程。可以说,如果开发者希望在英特尔平台上实现最佳的推理性能,并具备多平台适配和兼容性, OpenVINO™ 是不可或缺的部署工具首选。

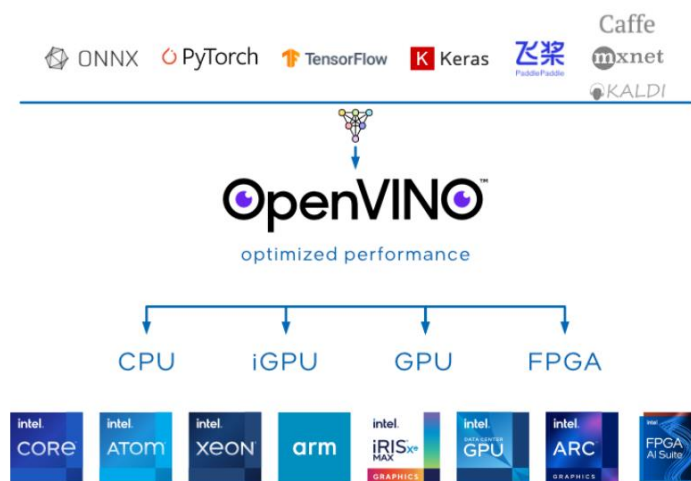


图 1 OpenVINO™

《OpenVINO™ 5 周年重头戏！2023.0 版本持续升级 AI 部署和加速性能》(<https://mp.weixin.qq.com/s/UWIA1EeX7M2-XhoJ5OfT7w>) 文章中介绍了 OpenVINO™ 最新版本 2023.0, 引入了一系列旨在增强开发人员体验的新功能、改进和弃用, 突出亮点是通过最大限度地减少离线转换、扩大模型支持和推进硬件优化来改善开发者之旅。

#### 1.1.2 YOLOv8

由 Ultralytics 开发的 Ultralytics YOLOv8 是一种尖端的, 最先进的 (SOTA) 模型, 它建立在以前的 YOLO 版本成功的基础上, 并引入了新功能和改进, 以进一步提高性能和灵活性。YOLOv8 设计为快速、准确且易于使用, 使其成为各种对象检测、图像分割和图像分类任务的绝佳选择。

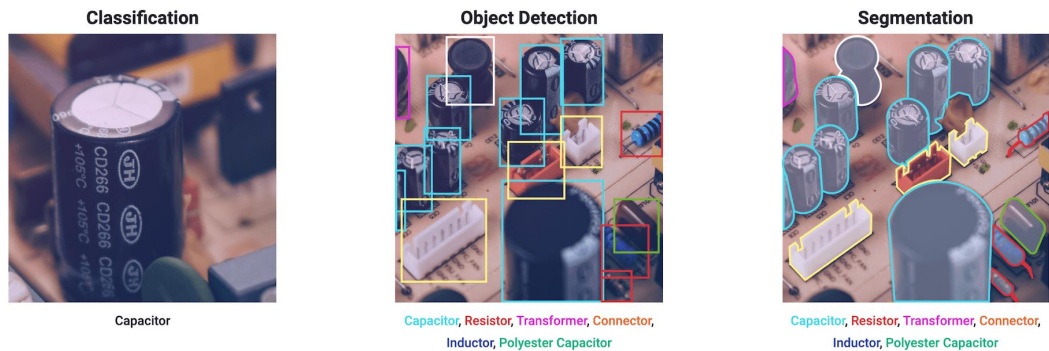


图 2 YOLOv8

YOLOv8 是 YOLO 系列目标检测算法的最新版本，相比于之前的版本，YOLOv8 具有更快的推理速度、更高的精度、更加易于训练和调整、更广泛的硬件支持以及原生支持自定义数据集等优势。这些优势使得 YOLOv8 成为了目前业界最流行和成功的目标检测算法之一。

模型	尺寸 (像素)	mAP <sub>val</sub> 50-95	速度 CPU ONNX (ms)	速度 A100 TensorRT (ms)	参数 (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

图 3 YOLOv8-det 模型

YOLOv8 提供了 YOLOv8-det、YOLOv8-seg、YOLOv8-pose 和 YOLOv8-cls 预训练模型，每种模型官方还提供了 n、s、m、l、x 五种模型格式。

### 1.1.3 项目内容

该项目基于 OpenVINO™ 2023.0 模型推理库，在 C# 语言下，调用封装的 OpenVINO™ 动态链接库，实现 YOLOv8 模型部署推理，实现了在 C# 平台调用 OpenVINO™ 部署 YOLOv8 模型。

为了防止复现代码出现问题，列出以下代码开发环境，可以根据自己需求设置，注意 OpenVINO™ 一定是 2022 版本，其他依赖项可以根据自己的设置修改。

- 操作系统：Windows 11
- OpenVINO™：2023.0
- OpenCV：4.5.5
- Visual Studio：2022
- C# 框架：.NET 6.0
- OpenCvSharp：OpenCvSharp4

项目所有代码已经在开源代码仓开源，开源地址：

GitHub: <https://github.com/guojin-yan/OpenVinoSharp.git>

Gitee: <https://gitee.com/guojin-yan/OpenVinoSharp.git>

## 1.2 OpenVinoSharp

C#是微软公司发布的一种由 C 和 C++衍生出来的面向对象的编程语言、运行于.NET Framework 和.NET Core(完全开源, 跨平台)之上的高级程序设计语言。但 OpenVINO™ 未提供 C#语言接口, 因此无法直接在 C#中使用, 之前发布的文章《在 C#中调用 OpenVINO™ 模型》([https://mp.weixin.qq.com/s/0ZbWIWSjQLTA3f\\_sfaCl1w](https://mp.weixin.qq.com/s/0ZbWIWSjQLTA3f_sfaCl1w))通过动态链接库特性, 实现了在 C#中调用 OpenVINO™ 部署深度学习模型, 简称: OpenVinoSharp。目前根据日常使用以及 OpenVINO™ 升级, OpenVinoSharp 也进行了相应的改进。

目前 OpenVinoSharp 2.1 已经推出, 支持 OpenVINO™ 2023.0 版本, 图 4 中给出了 OpenVinoSharp 实现原理。

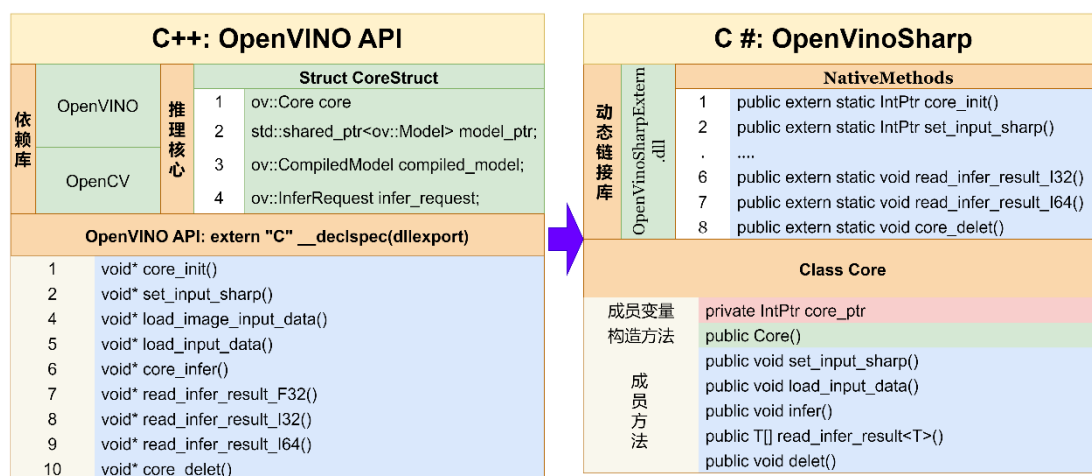


图 4 OpenVinoSharp 实现原理

### 1.2.1 OpenVINO™ 2023.0 安装配置

访问 OpenVINO™(openvino.ai)官网, 进入到下载页面, 按照图 5 进行选择, 最后进行下载, 下载后将文件解压到 C:\Program Files (x86)\Intel 文件夹下, 并在 Path 环境变量下增加线面的环境变量:

```
C:\Program Files (x86)\Intel\openvino_2023.0.0.10926\runtime\bin\intel64\Debug
C:\Program Files (x86)\Intel\openvino_2023.0.0.10926\runtime\bin\intel64\Release
C:\Program Files (x86)\Intel\openvino_2023.0.0.10926\runtime\3rdparty\tbb\bin
```

对于 C++项目, 增加以下配置:

Release 模式下:

包含目录:

```
C:\Program Files (x86)\Intel\openvino_2023.0.0.10926\runtime\include
C:\Program Files (x86)\Intel\openvino_2023.0.0.10926\runtime\include\ie
```

库目录

```
C:\Program Files (x86)\Intel\openvino_2023.0.0.10926\runtime\lib\intel64\Release
```

附加依赖项

```
openvino.lib
```

Debug 模式

包含目录

C:\Program Files (x86)\Intel\openvino\_2023.0.0.10926\runtime\include  
C:\Program Files (x86)\Intel\openvino\_2023.0.0.10926\runtime\include\ie  
库目录  
C:\Program Files (x86)\Intel\openvino\_2023.0.0.10926\runtime\lib\intel64\Debug  
附加依赖项  
openvinod.lib

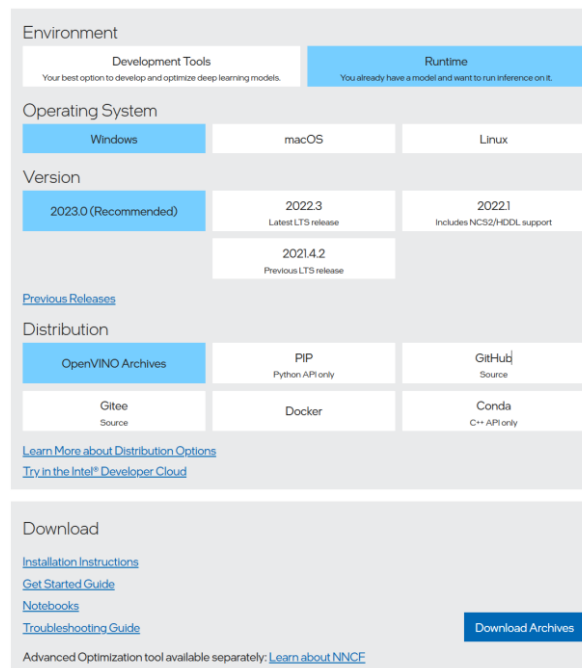


图 5 下载选择

## 1.2.2 C++ 动态链接库

C++可以封装 dll 动态链接库，由于 C#是基于 C/C++发展来的一门语言，因此 C#可以通过动态链接库的形式调用 C++代码。下面简单对动态链接库接口进行介绍，具体实现可以参考源码。

### (1) 推理引擎结构体

Core 是 OpenVINO™ 工具套件里的推理核心类，该类下包含多个方法，可用于创建推理中所使用的其他类。在此处，需要在各个方法中传递模型信息，因此选择构建一个推理引擎结构体，用于存放各个变量。

```
// @brief 推理核心结构体
typedef struct openvino_core {
    ov::Core core; // core 对象
    std::shared_ptr<ov::Model> model_ptr; // 读取模型指针
    ov::CompiledModel compiled_model; // 模型加载到设备对象
    ov::InferRequest infer_request; // 推理请求对象
} CoreStruct;
```

其中 Core 是 OpenVINO™ 工具套件里的推理机核心； `shared_ptr<ov::Model>`是读取本地模型的方法，可支持读取 Paddlepaddle 飞桨模型、ONNX 模型以及 IR 模型；CompiledModel 主要是将读取的本地模型映射到计算内核，由所指定的设备编译模型；

InferRequest 是一个推理请求类，在推理中主要用于对推理过程的操作。CoreStruct 结构体指针会贯穿下面的各个接口方法，实现模型信息的传递。

## (2) 初始化推理模型

OpenVINO™ 推理引擎结构体是联系各个方法的桥梁，后续所有操作都是在推理引擎结构体中的变量上操作的，为了实现数据在各个方法之间的传输，因此在创建推理引擎结构体时，采用的是创建结构体指针，并将创建的结构体地址作为函数返回值返回。推理初始化接口主要整合了原有推理的初始化 Core 对象、读取本地推理模型、载入模型到执行硬件和创建推理请求步骤，并将这些步骤所创建的变量放在推理引擎结构体中。

初始化推理模型接口方法为：

```
EXTERN_C __MIDL_DECLSPEC __DLLEXPORT void* STDMETHODCALLTYPE core_init(  
    const wchar_t* w_model_dir, const wchar_t* w_device, const wchar_t* w_cache_dir);
```

其中 w\_model\_dir 为推理模型本地地址，w\_device 为模型运行设备名，w\_cache\_dir 是缓存文件路径。

## (3) 配置输入数据形状

OpenVINO™ 2022.1 之后的版本支持了模型的动态输入并且支持动态设置模型输入形状，因此此处增加了模型形状设置接口：

```
EXTERN_C __MIDL_DECLSPEC __DLLEXPORT void* STDMETHODCALLTYPE set_input_shape(  
    void* core_ptr, const wchar_t* w_node_name, size_t* input_shape, int input_size);
```

其中 core\_ptr 是 CoreStruct 结构体指针，在读取模型后，后续所有操作都是基于该结构体实现的；w\_node\_name 是待修改模型节点，input\_shape 是模型节点，input\_size 是形状长度。

## (4) 配置输入数据

模型读取后就需要在模型中加载推理数据，常见的推理数据为图片数据，图片一般为一个三通道的二维数据，因此无法直接在 C# 与 C++ 中传递，在该项目中主要是将图片数据转为二进制数据进行传递，下面接口实现了图片数据的加载：

```
EXTERN_C __MIDL_DECLSPEC __DLLEXPORT void* STDMETHODCALLTYPE load_image_input_data(  
    void* core_ptr, const wchar_t* w_node_name, uchar* image_data, size_t image_size, int type);
```

其中 w\_node\_name 为模型输入节点名，image\_data 为图片数据的二进制数组，image\_size 为图片数据长度，type 为数据处理方式，目前已经实现了常规变换、仿射变换、均方差归一化和常规归一化等方式。除了图片输入，一些模型还要求其他数据的输入，因此在该方法中，还增加了常规数据的加载方法：

```
EXTERN_C __MIDL_DECLSPEC __DLLEXPORT void* STDMETHODCALLTYPE load_input_data(  
    void* core_ptr, const wchar_t* w_node_name, float* input_data);
```

## (5) 模型推理

将数据加载到模型后，就可以进行模型的推理，获取模型的推理结果，下面方法中封装了模型推理的方法，可以实现加载的数据进行推理。

```
EXTERN_C __MIDL_DECLSPEC __DLLEXPORT void* STDMETHODCALLTYPE core_infer(void* core_ptr);
```

## (6) 读取推理结果

在进行完模型推理后，就可以获取模型的推理结果，由于不同模型要求的数据格式不同，此处封装了 float 和 int 数据的读取方式：

```
EXTERN_C __MIDL_DECLSPEC __DLLEXPORT void STDMETHODCALLTYPE read_infer_result_I32(  
    void* core_ptr, int node_id, int* result);
```

```
void* core_ptr, const wchar_t* w_node_name, int* infer_result);
EXTERN_C __MIDL_DECLSPEC_DLLEXPORT void STDMETHODCALLTYPE read_infer_result_I64(
void* core_ptr, const wchar_t* w_node_name, long long* infer_result);
```

其中 w\_node\_name 为输出节点名称，infer\_result 为推理结果数组指针。

### (7) 删除推理核心结构体指针

推理完成后，我们需要将在内存中创建的推理核心结构地址删除，防止造成内存泄露，影响电脑性能，其接口该方法为：

```
EXTERN_C __MIDL_DECLSPEC_DLLEXPORT void STDMETHODCALLTYPE core_delet(void* core_ptr);
```

## 1.2.3 C#构建 Core 推理类

上一步中我们将模型推理方法封装到了 dll 动态链接库中，生成了 OpenVinoSharpExtern.dll 文件，在 C#中可以通过[DllImport()]方式通过函数名称与参数对应，将 C++中的方法引入当 C#中。

```
private const string dll_extern = "OpenVinoSharpExtern.dll";
[DllImport(dll_extern, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
public extern static IntPtr core_init(string model_dir, string device, string w_cache_dir);
```

上述代码通过 core\_init 名称对应和参数返回指定应，将 core\_init()方法引入到 C#中，同样的方式，可以将其他方法引入到 C#中。

上一步我们引入了封装的 OpenVINO™ 动态链接库，为了更方便的使用，将其封装到 Core 类中。在不同方法之间，主要通过推理核心结构体指针在各个方法之间传递，在 C# 是没有指针这个说法的，不过可以通过 IntPtr 结构体来接收这个指针，为了防止该指针被篡改，将其封装在类中作为私有成员使用。

根据模型推理的步骤，构建模型推理类，表 1 中除了构建的 Core 模型推理类的 API 接口，具体代码实现可以参考源码。

表 1 OpenVinoSharp API 接口

序号	API		参数解释	说明
1	方法	Core()	构造函数	初始化推理核心，
	参数	string model	构造函数	
		string device	设备名称	
		string cache_dir	缓存路径	
2	方法	void set_input_shape()	设置输入节点形状	根据节点维度设置
	参数	string node_name	输入节点名称	
		int[] input_shape	形状数组	
3	方法	void load_input_data()	设置图片/普通输入数据	<b>type = 0:</b> 方差归一化、常规缩放 <b>type = 1:</b> 普通归一化(1/255)、常规缩放 <b>type = 2:</b> 不归一化、常规缩放 <b>type = 0:</b> 方差归一化、仿射变换 <b>type = 1:</b> 普通归一化(1/255)、仿射变换 <b>type = 2:</b> 不归一化、仿射变换
	参数	string node_name	输入节点名称	
		float[] input_data	输入数据	
	参数	string node_name	输入节点名称	
		byte[] image_data	图片数据	
		int type	数据处理类型：	
4	方法	void infer()	模型推理	
5	方法	void T[] read_infer_result <T>()	读取推理结果数据	支持读取 Float32、Int32、Int64



	参数	<code>string output_name</code>	输出节点名	格式数据
		<code>int data_size</code>	输出数据长度	
6	方法	<code>void delet()</code>	删除内存地址	

## 1.2.4 NuGet 安装 OpenVinoSharp

为了方便使用 OpenVinoSharp，该项目提供了 NuGet 包，使用者可以直接通过 NuGet 包管理器直接安装使用，程序包中包含 OpenVINO2023.0 版本的库文件，因此使用时无需再安装 OpenVINO2023.0，直接安装 NuGet 包即可使用。



图 6 NuGet 包安装

## 1.3 获取和转换 YOLOv8 模型

### 1.3.1 安装 ultralytics

为了更好的管理 python 库，此处采用 conda 安装 ultralytics 环境，使用 CMD 窗口输入如下指令，进行环境创建并安装 ultralytics 依赖项：

```
conda create -n ultralytics python==3.10
conda activate ultralytics
pip install ultralytics
```

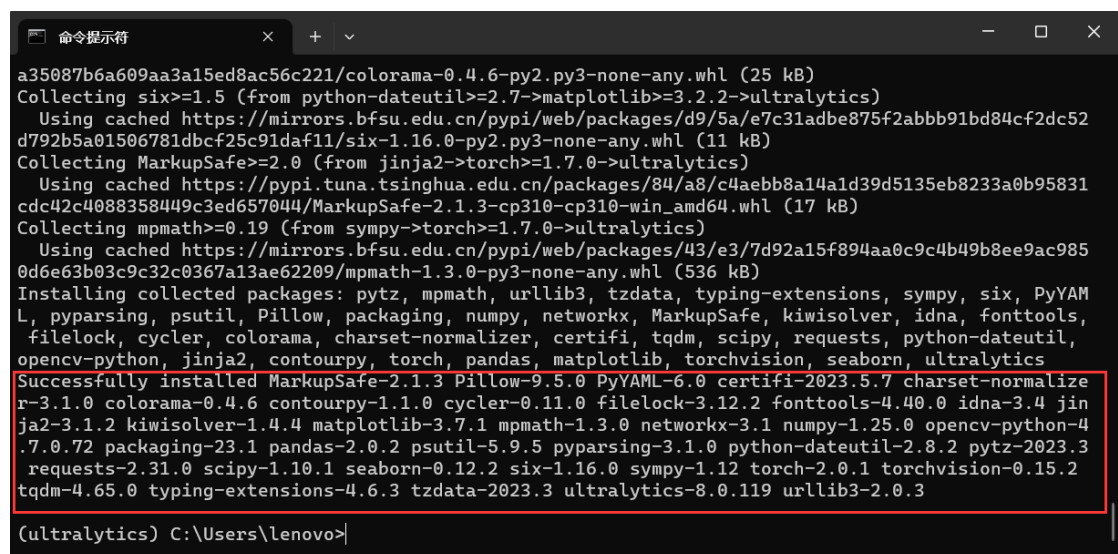


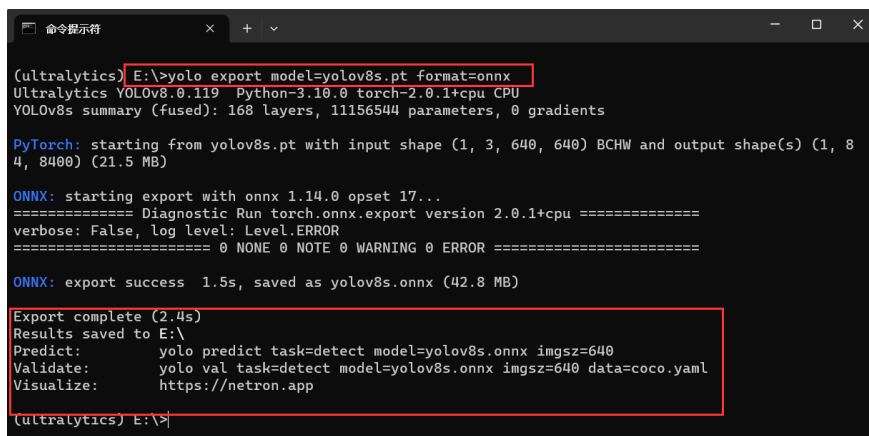


图 7 ultralytics 安装结果

### 1.3.2 导出 yolov8 模型

Ultralytics 支持直接导出预训练模型，并且可以直接导出 onnx 格式的模型，以 Yolov8-det 模型为例，直接导出 yolov8s.onnx 文件，在命令行中输入以下命令

```
yolo export model=yolov8s.pt format=onnx
```



```
(ultralytics) E:\>yolo export model=yolov8s.pt format=onnx
Ultralytics YOLOv8.0.119 Python-3.10.0 torch-2.0.1+cpu CPU
YOLOv8s summary (fused): 168 layers, 11156544 parameters, 0 gradients

PyTorch: starting from yolov8s.pt with input shape (1, 3, 640, 640) BCHW and output shape(s) (1, 8, 4, 8400) (21.5 MB)

ONNX: starting export with onnx 1.14.0 opset 17...
===== Diagnostic Run torch.onnx.export version 2.0.1+cpu =====
verbose: False, log level: Level.ERROR
===== 0 NONE 0 NOTE 0 WARNING 0 ERROR =====

ONNX: export success 1.5s, saved as yolov8s.onnx (42.8 MB)

Export complete (2.4s)
Results saved to E:\
Predict:      yolo predict task=detect model=yolov8s.onnx imgsz=640
Validate:     yolo val task=detect model=yolov8s.onnx imgsz=640 data=coco.yaml
Visualize:    https://netron.app

(ultralytics) E:\>
```

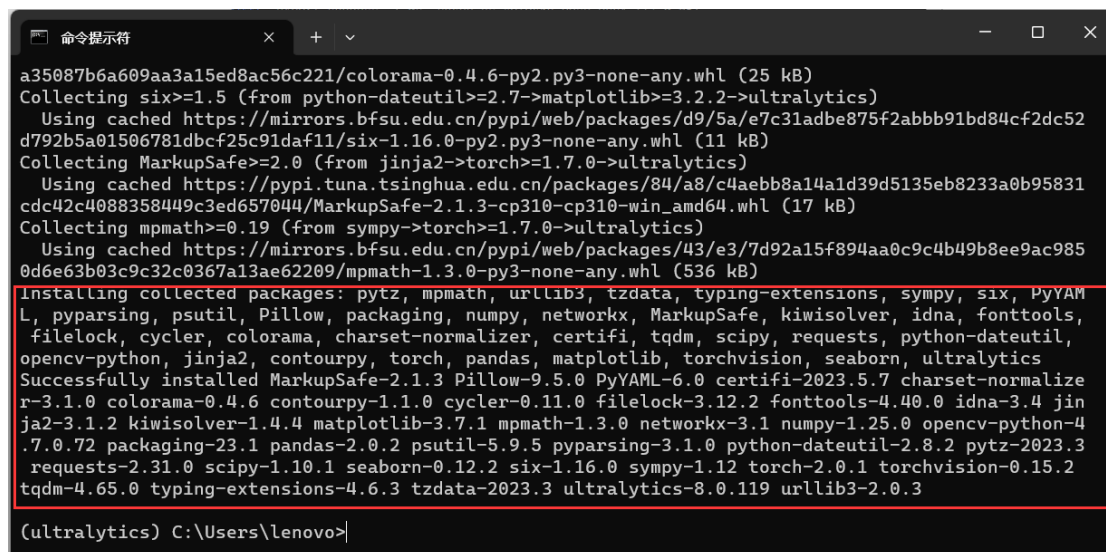
图 8 yolov8 模型导出

通过上述命令可以获得 yolov8s.onnx 模型文件，同样的方式可以将 Yolov8-seg、Yolov8-pose 和 Yolov8-cls 的模型文件。

### 1.3.3 安装 OpenVINO™ Python 版

OpenVINO™ 提供了 Python 语言接口，因此是可以直接安装 Python 版本使用，且 Python 版本至此模型的转换和量化，此处采用 conda 安装 OpenVINO™ 环境，使用 CMD 窗口输入如下指令，进行环境创建并安装 OpenVINO™ 依赖项：

```
conda create -n openvino2023_0 python=3.10
conda activate openvino2023_0
python -m pip install --upgrade pip
pip install openvino-dev[ONNX,pytorch,tensorflow2]==2023.0.0
```



```
(ultralytics) C:\Users\lenovo>pip install openvino-dev[ONNX,pytorch,tensorflow2]==2023.0.0
a35087b6a609aa3a15ed8ac56c221/colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting six>=1.5 (from python-dateutil>=2.7->matplotlib>=3.2.2->ultralytics)
Using cached https://mirrors.bfsu.edu.cn/pypi/web/packages/d9/5a/e7c31adbe875f2abbb91bd84cf2dc52
d792b5a01506781dbcf25c91daf11/six-1.16.0-py2.py3-none-any.whl (11 kB)
Collecting MarkupSafe>=2.0 (from jinja2->torch>=1.7.0->ultralytics)
Using cached https://pypi.tuna.tsinghua.edu.cn/packages/84/a8/c4aebb8a14a1d39d5135eb8233a0b95831
cdc42c4088358449c3ed657044/MarkupSafe-2.1.3-cp310-cp310-win_amd64.whl (17 kB)
Collecting mpmath>=0.19 (from sympy->torch>=1.7.0->ultralytics)
Using cached https://mirrors.bfsu.edu.cn/pypi/web/packages/43/e3/7d92a15f894aa0c9c4b49b8ee9ac985
0d6e63b03c9c32c0367a13ae62209/mpmath-1.3.0-py3-none-any.whl (536 kB)
Installing collected packages: pytz, mpmath, urllib3, tzdata, typing-extensions, sympy, six, PyYAM
L, pyparsing, psutil, Pillow, packaging, numpy, networkx, MarkupSafe, kiwisolver, idna, fonttools,
filelock, cycler, colorama, charset-normalizer, certifi, tqdm, scipy, requests, python-dateutil,
opencv-python, jinja2, contourpy, torch, pandas, matplotlib, torchvision, seaborn, ultralytics
Successfully installed MarkupSafe-2.1.3 Pillow-9.5.0 PyYAML-6.0 certifi-2023.5.7 charset-normalize
r-3.1.0 colorama-0.4.6 contourpy-1.1.0 cycler-0.11.0 filelock-3.12.2 fonttools-4.40.0 idna-3.4 jin
ja2-3.1.2 kiwisolver-1.4.4 matplotlib-3.7.1 mpmath-1.3.0 networkx-3.1 numpy-1.25.0 opencv-python-4
.7.0.72 packaging-23.1 pandas-2.0.2 psutil-5.9.5 pyparsing-3.1.0 python-dateutil-2.8.2 pytz-2023.3
requests-2.31.0 scipy-1.10.1 seaborn-0.12.2 six-1.16.0 sympy-1.12 torch-2.0.1 torchvision-0.15.2
tqdm-4.65.0 typing-extensions-4.6.3 tzdata-2023.3 ultralytics-8.0.119 urllib3-2.0.3

(ultralytics) C:\Users\lenovo>
```

图 9 OpenVINO™ 安装

### 1.3.4 Yolov8 模型量化

OpenVINO™ 提供了模型优化的工具，yolov8 模型精度默认为 f32 格式，该工具可以直接使用命令将模型转为 f16，以 Yolov8-det 模型 yolov8s.onnx 文件为例，在命令行中输入：

```
mo -m yolov8s.onnx --compress_to_fp16
```

```
(OpenVino2023_0) E:\Text_Model\yolov8>mo -m yolov8s.onnx --compress_to_fp16
[ INFO ] Generated IR will be compressed to FP16. If you get lower accuracy, please consider disabling compression by removing argument --compress_to_fp16 or set it to false --compress_to_fp16=False.
Find more information about compression to FP16 at https://docs.openvino.ai/latest/openvino_docs_MO_DG_FP16_Compression.html
[ INFO ] The model was converted to IR v11, the latest model format that corresponds to the source DL framework input/output format. While IR v11 is backwards compatible with OpenVINO Inference Engine API v1.0, please use API v2.0 (as of 2022.1) to take advantage of the latest improvements in IR v11.
Find more information about API v2.0 and IR v11 at https://docs.openvino.ai/latest/openvino_2_0_transition_guide.html
[ SUCCESS ] Generated IR version 11 model.
[ SUCCESS ] XML file: E:\Text_Model\yolov8\yolov8s.xml
[ SUCCESS ] BIN file: E:\Text_Model\yolov8\yolov8s.bin
```

图 10 Yolov8 模型量化

通过上述命令可以获得 f16 精度的 IR 格式的模型：yolov8s.xml、yolov8s.bin。同样的方式可以将 Yolov8-seg、Yolov8-pose 和 Yolov8-cla 模型的精度转为 f16。

### 1.3.5 Benchmark\_app 测试 Yolov8 模型

benchmark\_app 是 OpenVINO™ 工具套件自带的 AI 模型推理计算性能测试工具，可以指定在不同的计算设备上，在同步或异步模式下，测试出不带前后处理的纯 AI 模型推理计算性能。以 Yolov8-det 模型为例，在命令行中输入以下命令：

```
benchmark_app -m yolov8s.xml -d CPU
```

```
[ INFO ] Execution Devices: ['CPU']
[ INFO ] Count: 1276 iterations
[ INFO ] Duration: 60193.75 ms
[ INFO ] Latency:
[ INFO ] Median: 187.53 ms
[ INFO ] Average: 188.32 ms
[ INFO ] Min: 163.55 ms
[ INFO ] Max: 224.57 ms
[ INFO ] Throughput: 21.20 FPS
```

图 11 Yolov8-det 模型推理计算性能

上图为 Yolov8-det 模型在 CPU:AMD R7 5800 异步推理计算性能，最多可以实现 21 帧推理。

## 1.4 OpenVinoSharp 部署 Yolov8 模型

为了更好的演示 Yolov8 模型在 C# 中部署，此处使用 WinForm 搭建了一个简单的软件进行展示，软件搭建详细代码可以参考源码，下面代码演示了 OpenVinoSharp 部署 Yolov8 模型的核心代码：

```
// 配置图片数据
Mat image = new Mat(image_path); // 读取本地图片
int max_image_length = image.Cols > image.Rows ? image.Cols : image.Rows;
Mat max_image = Mat.Zeros(new OpenCvSharp.Size(max_image_length, max_image_length),
MatType.CV_8UC3);
```

```

Rect roi = new Rect(0, 0, image.Cols, image.Rows);
image.CopyTo(new Mat(max_image, roi));
float[] factors = new float[2]; // 图片缩放比例
factors[0] = factors[1] = (float)(max_image_length / 640.0);
// 读取模型，初始化推理核心
Core core = new Core(model_path, "CPU");
// 将图片转为二进制数据
byte[] image_data = max_image.ImEncode(".bmp");
// 存储 byte 的长度
ulong image_size = Convert.ToUInt64(image_data.Length);
// 加载推理图片数据
core.load_input_data("images", image_data, image_size, 1);
// 模型推理
core.infer();
// 读取推理结果
float[] result_array = core.read_infer_result<float>("output0", 8400 * 84);
// 清理内存占用
core.delete();
// 创建结果处理类
DetectionResult result_pro = new DetectionResult(classer_path, factors);
// 处理推理结果
result_image = result_pro.draw_result(result_pro.process_result(result_array), image.Clone());

```

### 1.4.1 YOLOv8-det 预测结果

用 Netron 软件打开 yolov8s.onnx，可以看到模型的输入和输出：

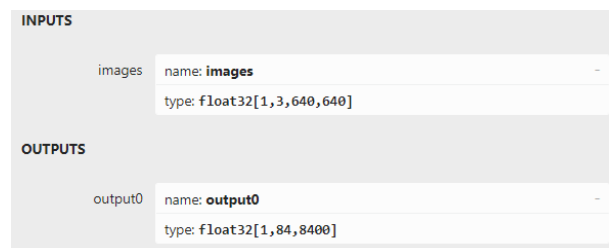


图 12 YOLOv8-det 模型节点

模型的输入节点为“images”，数据类型为 float32，输入为[1,3,640,640]大小的张量，代表输入为一个 640×640 大小的图片，图片加载模型前需要对数据进行归一化处理。

模型的输出节点为“output0”，数据类型为 float32，输出为[1,84,8400]的张量，其中“84”的定义为：cx,cy,h,w 和 80 种类别的分数。“8400”是指 YOLOv8 的 3 个检测头在图像尺寸为 640 时，有 640/8=80, 640/16=40, 640/32=20, 80x80+40x40+20x20=8400 个输出单元格。

下图为 YOLOv8-det 模型的推理结果：



图 13 YOLOv8-det 模型预测结果

#### 1.4.2 YOLOv8-cls 预测结果

用 Netron 软件打开 yolov8s-cls.onnx，可以看到模型的输入和输出：



图 14 YOLOv8-cls 模型节点

模型的输入节点为“images”，数据类型为 float32，输入为[1,3,224,224]大小的张量，代表输入为一个 224×224 大小的图片，图片加载模型前需要对数据进行归一化处理。

模型的输出节点为“output0”，数据类型为 float32，输出为[1,1000]的张量，其中“1000”代表 1000 个分类情况的置信值，后续直接通过最大值索引和类别标签判断结果。

下图为 YOLOv8-cls 模型的推理结果：

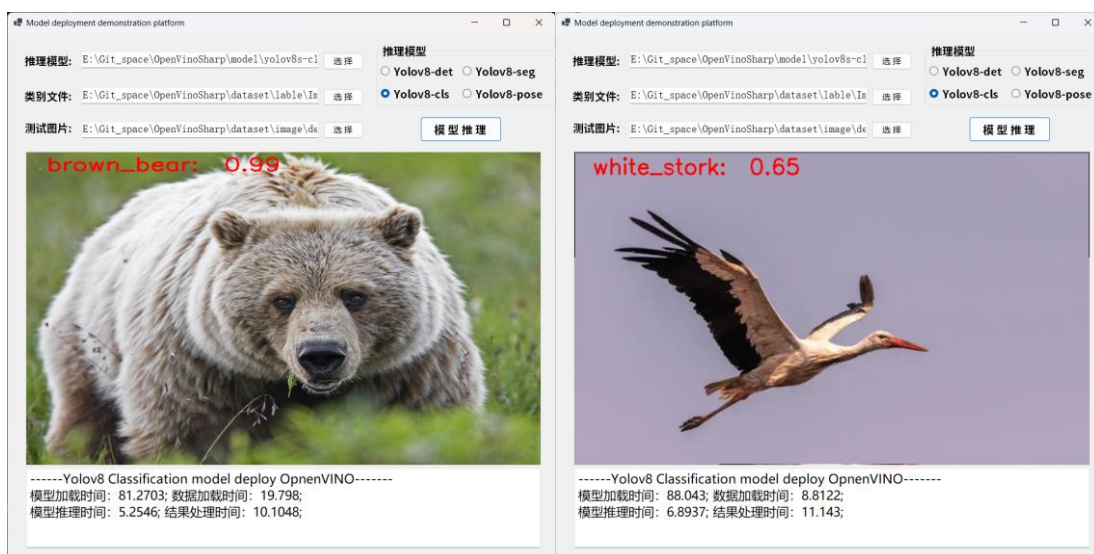


图 15 Yolov8-cls 模型预测结果

### 1.4.3 Yolov8-seg 预测结果

用 Netron 软件打开 yolov8s-seg.onnx，可以看到模型的输入和输出：

INPUTS	
images	name: images type: float32[1,3,640,640]
OUTPUTS	
output0	name: output0 type: float32[1,116,8400]
output1	name: output1 type: float32[1,32,160,160]

图 16 Yolov8-seg 模型节点

模型的输入节点与 Yolov8 一致。模型的输出节点有两个，输出节点 1 为“output0”，数据类型为 float32，输出为[1,116,8400]的张量，其中 116 的前 84 个字段跟 YOLOv8 目标检测模型输出定义完全一致，后 32 个字段用于计算 mask。输出节点 2 为“output1”，数据类型为 float32，输出为[1,32,160,160]的张量，output0 后 32 个字段与 output1 的数据做矩阵乘法后得到的结果，即为对应目标的 mask。

下图为 Yolov8-seg 模型的推理结果：



图 17 Yolov8-seg 模型预测结果

### 1.4.4 Yolov8-pose 预测结果

用 Netron 软件打开 yolov8s-pose.onnx，可以看到模型的输入和输出：

INPUTS	
images	name: images type: float32[1,3,640,640]
OUTPUTS	
output0	name: output0 type: float32[1,56,8400]



图 18 YOLOv8-pose 模型节点

模型的输入节点与 YOLOv8 一致。模型的输出节点为 “output0”，数据类型为 float32，输出为 [1,56,8400] 的张量，“56” 指人体的预测框 (cx,cy,w,h,score) 和人体 17 个关键点 (x,y,score) 的预测结果， $5+17\times 3=56$ 。

下图为 YOLOv8-pose 模型的推理结果：

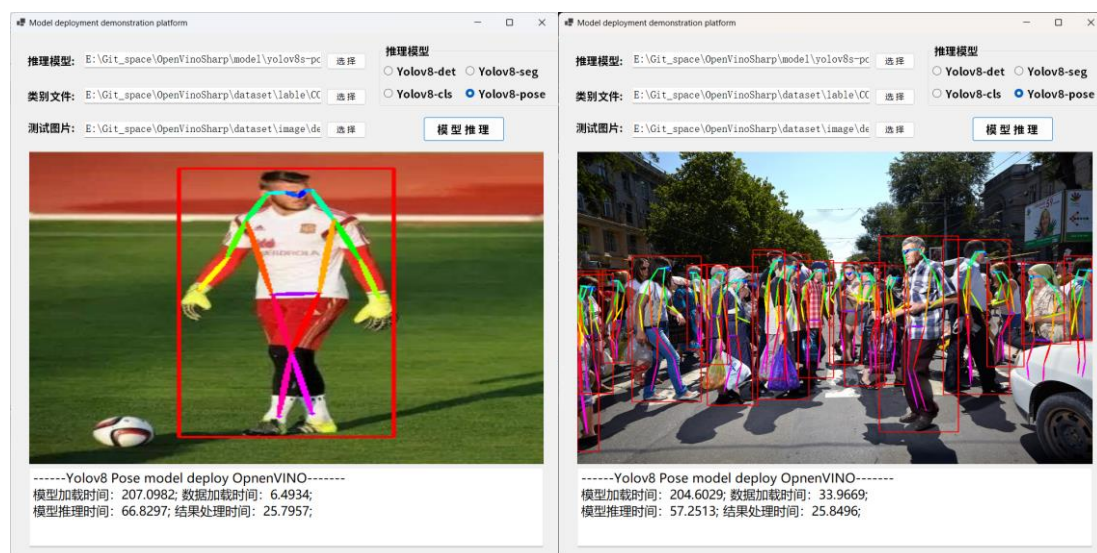


图 19 YOLOv8-pose 模型预测结果

## 1.5 总结

C#作为一门比较高级的编程语言，在 Windows 平台上有很好的支持，可以很容易实现桌面软件的开发，并且拥有自动的垃圾回收机制，可以实现自动内存管理，极大的减轻了编码过程，因此在当前工业领域有很广泛的应用。而 OpenVINO™凭借着齐强的推理性能和硬件支持，在模型部署领域大放异彩。

因此实现在 C#平台使用 OpenVINO™部署深度学习模型，打破了 OpenVINO™与 C#之间的壁垒，会使得模型部署在工业领域应用变得更加容易。