目 录

第1章 基于 C# 和 OpenVINO 部署飞桨 PP-Human

作者：英特尔物联网行业创新大使 杨雪锋

本文将详细介绍基于 OpenVINO™ 工具包，在 C#语言下，部署飞桨 PP-Human 的全流程，帮助开发者快速掌握并部署产业级 AI 人体分析解决方案。

1.1 飞桨实时行人分析工具 PP-Human

PP-Human 是飞桨目标检测套件 PaddleDetection 中开源的实时行人分析工具，提供了五大异常行为识别和四大产业级功能：人体属性分析、人流计数、跨镜 ReID，如下图所示：

图 1-1 PP-Human v2 全功能全景图

1.1.1 PP-Human 技术架构

PP-Human 支持单张图片、图片文件夹单镜头视频和多镜头视频输入，经目标检测以及特征关联，实现属性识别、关键点检测、轨迹/流量计数以及行为识别等功能，如下图所示。本文将以行人摔倒识别为例，基于 OpenVINO™ 进行多种模型联合部署。

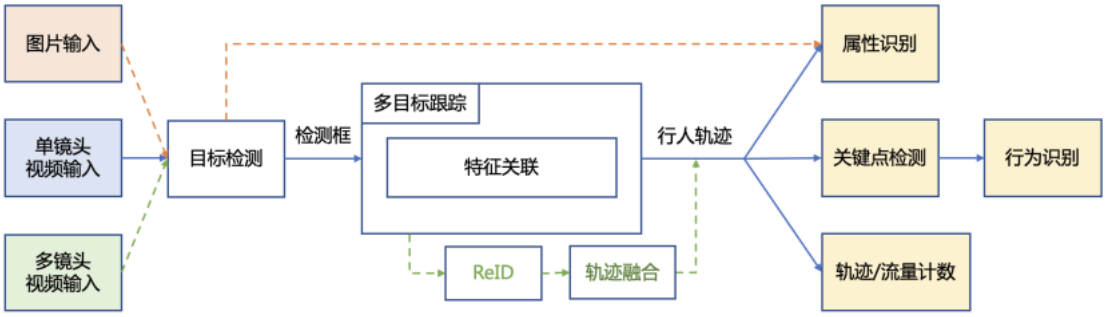

图 1-2 PP-Human 技术架构

1.1.2 构建 C#开发环境

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™：2022.3

- OpenCV：4.5.5
- Visual Studio：2022 Community
- C#框架：.NET 6.0
- OpenCvSharp：OpenCvSharp4

1.1.3 项目完整代码已开源

项目所涉及的源码已在 Gitee 上开源，直接克隆到本地即可使用：

```
git clone https://github.com/guojin-yan/Csharp_and_OpenVINO_deploy_PP-Human.git
```

1.2 C#中调用 OpenVINO™ 实现

1.2.1 构建 OpenVINO™ 动态链接库

由于 OpenVINO™ 只有 C++和 Python 接口，无法直接在 C#中使用 OpenVINO™ 部署模型，为了实现在 C#中使用，通过动态链接库的方式实现。具体教程参考《在 C#中调用 OpenVINO™ 模型》。

1.2.2 在 C#中引入动态链接库文件

在 C#中需要使用[DllImport()]方法引入动态链接库文件，其完整的使用方式如以下代码所示：

```
[DllImport(openvino_dll_path, CharSet = CharSet.Unicode, CallingConvention =
CallingConvention.Cdecl)]
public extern static IntPtr set_input_image_sharp(IntPtr inference_engine, string
input_node_name, ref ulong input_size);
```

针对[DllImport()]括号中的内容：

- openvino_dll_path 为 dll 文件路径
- CharSet = CharSet.Unicode 代表支持中文编码格式字符串
- CallingConvention = CallingConvention.Cdecl 指示入口点的调用约定为调用方清理堆栈

在声明动态链接库后，就可以引入动态链接库中的方法，由于我们在 C++环境下生成的动态链接库，为了让编译器识别，需要方法名、变量类型一一对应，才可以引入成功：

表 1 C++与 C#方法对应关系

	C++	C#
返回值类型	void*	IntPtr
方法名	set_input_image_sharp	set_input_image_sharp
参数 1	void*	IntPtr
参数 2	wchar_t*	string
参数 3	size_t *	ref ulong

基于以上方法，我们将动态链接库中的所有方法引入到 C#中。

1.2.3 C#构建 Core 类

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：

（1）构造函数

```
public Core(string model_file, string device_name){
    // 初始化推理核心
    ptr = NativeMethods.core_init(model_file, device_name);
}
```

在该方法中，主要是调用推理核心初始化方法，初始化推理核心，读取本地模型，将模型加载到设备、创建推理请求等模型推理步骤。

（2）设置模型输入形状

```
// @brief 设置推理模型的输入节点的大小
// @param input_node_name 输入节点名
// @param input_size 输入形状大小数组
public void set_input_sharp(string input_node_name, ulong[] input_size) {
    // 获取输入数组长度
    int length = input_size.Length;
    if (length == 4) {
        // 长度为 4，判断为设置图片输入的输入参数，调用设置图片形状方法
        ptr = NativeMethods.set_input_image_sharp(ptr, input_node_name, ref input_size[0]);
    }
    else if (length == 2) {
        // 长度为 2，判断为设置普通数据输入的输入参数，调用设置普通数据形状方法
        ptr = NativeMethods.set_input_data_sharp(ptr, input_node_name, ref input_size[0]);
    }
    else {
        // 为防止输入发生异常，直接返回
        return;
    }
}
```

OpenVINO™ 2022.3 支持模型动态输入，读入模型可以不固定输入大小，在使用时固定模型的输入大小，并且可以随时修改输入形状。当前设置情况下，至此设置二维、以及四维的输入形状，在当前模型中足够使用。

（3）加载推理数据

```
// @brief 加载推理数据
// @param input_node_name 输入节点名
// @param input_data 输入数据数组
public void load_input_data(string input_node_name, float[] input_data) {
    ptr = NativeMethods.load_input_data(ptr, input_node_name, ref input_data[0]);
```

```
        }
        // @brief 加载图片推理数据
        // @param input_node_name 输入节点名
        // @param image_data 图片矩阵
        // @param image_size 图片矩阵长度
        public void load_input_data(string input_node_name, byte[] image_data, ulong image_size, int type) {
            ptr = NativeMethods.load_image_input_data(ptr, input_node_name, ref image_data[0], image_size,
type);
        }
```

加载推理数据主要包含图片数据和普通的矩阵数据，其中对于图片的预处理，也已经在 C++中进行封装，保证了图片数据在传输中的稳定性。

（5）模型推理

```
        // @brief 模型推理
        public void infer() {
            ptr = NativeMethods.core_infer(ptr);
        }
```

（6）读取推理结果数据

```
        // @brief 读取推理结果数据
        // @param output_node_name 输出节点名
        // @param data_size 输出数据长度
        // @return 推理结果数组
        public T[] read_infer_result<T>(string output_node_name, int data_size) {
            // 获取设定类型
            string t = typeof(T).ToString();
            // 新建返回值数组
            T[] result = new T[data_size];
            if (t == "System.Int32") { // 读取数据类型为整形数据
                int[] inference_result = new int[data_size];
                NativeMethods.read_infer_result_I32(ptr, output_node_name, data_size, ref
inference_result[0]);
                result = (T[])Convert.ChangeType(inference_result, typeof(T[]));
                return result;
            }
            else { // 读取数据类型为浮点型数据
                float[] inference_result = new float[data_size];
                NativeMethods.read_infer_result_F32(ptr, output_node_name, data_size, ref
inference_result[0]);
                result = (T[])Convert.ChangeType(inference_result, typeof(T[]));
                return result;
            }
        }
```

在读取模型推理结果时，支持读取整形数据和浮点型数据，且需要知晓模型输出数据的大小，这就要求我们对自己所使用的模型有很好的把握。

（7）清除地址

```
// @brief 删除创建的地址
public void delet() {
    NativeMethods.core_delet(ptr);
}
```

此处的清除地址需要调用 fengzhuangd 额地址删除方法实现，不可以直接删除 C#中创建的 IntPtr，这样会导致内存泄漏，影响程序性能。

通过上面的封装，比可以在 C#平台下，调用 Core 类，间接调用 OpenVINO™ 工具包署自己的模型了。

下一篇，我们将继续介绍：构建行人识别 C# PP-YOLOE 类和人体姿势识别 TinyPose 类。