目 录

第 1 章 基于 C#和 OpenVINO 部署 PaddleOCR 模型 …… 2
1.1 项目概述…… 2
1.1.1 OpenVINO™ …… 2
1.1.2 PaddleOCR …… 2
1.1.3 项目介绍…… 3
1.1.4 项目编码环境 …… 4
1.1.5 源码下载方式 …… 4
1.1.6 模型下载方式 …… 4
1.2 C#中调用 OpenVINO™ 实现…… 5
1.2.1 OpenVINO™ 动态链接库…… 5
1.2.2 C#引入动态链接库文件…… 5
1.2.3 C#构建 Core 类…… 6
1.3 PaddleOCR 文字区域识别 …… 8
1.3.1 模型介绍-Detection model …… 8
1.3.2 模型下载与转换 …… 9
1.3.3 文字区域识别 C#实现模型推理 …… 9
1.3.4 文字区域识别 C#实现模型结果处理 …… 10
1.4 PaddleOCR 文字内容识别 …… 13
1.4.1 模型介绍- Recognition model…… 13
1.4.2 模型下载与转换 …… 13
1.4.3 文字内容识别 C#实现模型推理 …… 14
1.4.4 文字内容识别 C#实现模型结果处理 …… 16
1.5 PaddleOCR 文字识别模型在 OpenVINO™ 部署效果…… 17
1.5.1 文字识别效果 …… 17
1.5.2 时间测试…… 18
1.6 总结…… 18

第1章 基于 C#和 OpenVINO 部署 PaddleOCR 模型

1.1 项目概述

1.1.1 OpenVINO™

OpenVINO™是英特尔基于自身现有的硬件平台开发的一种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，用于快速开发应用程序和解决方案，以解决各种任务（包括人类视觉模拟、自动语音识别、自然语言处理和推荐系统等）。

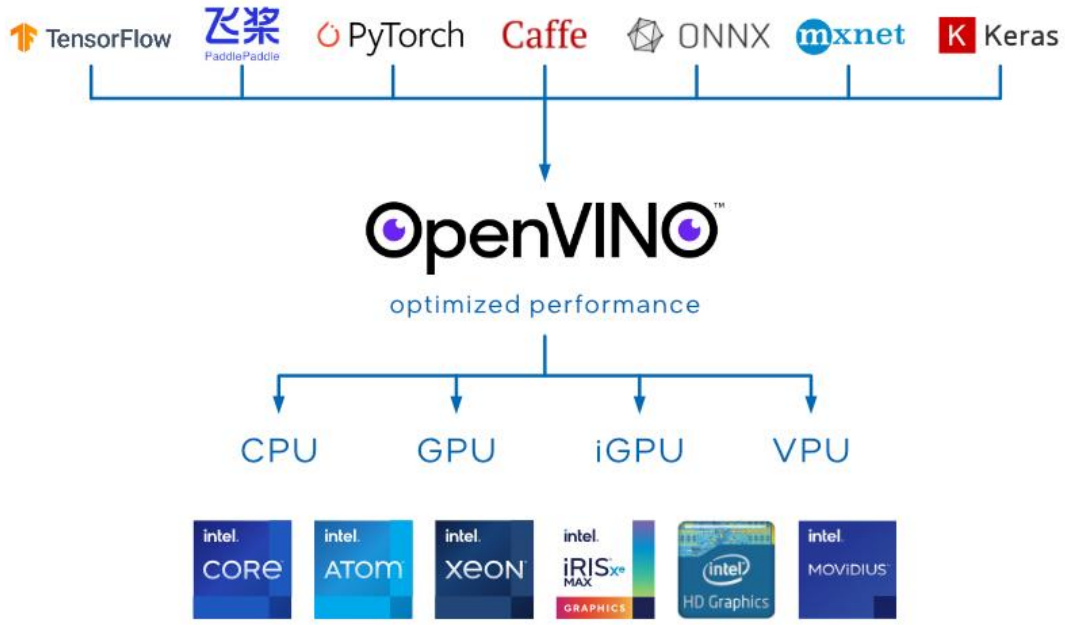

图 1 OpenVINO™ 工具结构图

该工具套件基于最新一代的人工神经网络，包括卷积神经网络 (CNN)、递归网络和基于注意力的网络，可扩展跨英特尔® 硬件的计算机视觉和非视觉工作负载，从而最大限度地提高性能。它通过从边缘到云部署的高性能、人工智能和深度学习推理来为应用程序加速，并且允许直接异构执行。极大的提高计算机视觉、自动语音识别、自然语言处理和其他常见任务中的深度学习性能；使用使用流行的框架（如 TensorFlow，PyTorch 等）训练的模型；减少资源需求，并在从边缘到云的一系列英特尔®平台上高效部署；支持在 Windows 与 Linux 系统，且官方支持编程语言为 Python 与 C++语言。

OpenVINO™ 工具套件 2022.1 版于 2022 年 3 月 22 日正式发布，与以往版本相比发生了重大革新，提供预处理 API 函数、ONNX 前端 API、AUTO 设备插件，并且支持直接读入飞桨模型，在推理中中支持动态改变模型的形状，这极大地推动了不同网络的应用落地。2022 年 9 月 23 日，OpenVINO™ 工具套件 2022.2 版推出，对 2022.1 进行了微调，以包括对英特尔最新 CPU 和离散 GPU 的支持，以实现更多的人工智能创新和机会。

1.1.2 PaddleOCR

飞桨(PaddlePaddle)是集深度学习核心框架、工具组件和服务平台为一体的技术先进、

功能完备的开源深度学习平台，已被中国企业广泛使用，深度契合企业应用需求，拥有活跃的开发者社区生态。提供丰富的官方支持模型集合，并推出全类型的高性能部署和集成方案供开发者使用。是中国首个自主研发、功能丰富、开源开放的产业级深度学习平台。

OCR（optical character recognition）文字识别是指电子设备（例如扫描仪或数码相机）检查纸上打印的字符，然后用字符识别方法将形状翻译成计算机文字的过程。如何除错或利用辅助信息提高识别正确率，是 OCR 最重要的课题。PaddleOCR 由飞桨平台推出，旨在打造一套丰富、领先、且实用的 OCR 工具库，助力开发者训练出更好的模型，并应用落地。

图 2　PaddleOCR 识别效果图

1.1.3 项目介绍

该项目基于 OpenVINO™ 模型推理库，在 C#语言下，调用封装的 OpenVINO™ 动态链接库，部署推理 PP-OCR 中的文字识别模型；实现了在 C#平台调用 OpenVINO™ 部署 PP-OCR 文字识别模型。

如图 3 所示，对于一个完整的文字识别流程主要分为三个流程：识别文字区域、识别文字方向、识别文字内容。其中识别文字区域主要是将一张图片上的所有文字区域识别出来，并将其处理成一个个小的文字区域；文本方向识别主要是识别当前文字的方向，并调整文本方向到正确方向；识别文字内容主要是将前两步处理后的文字区域中的文字识别出来。通过以上三个步骤，就可以实现提取一张图片上文字内容。

当前新项目主要实现了文本区域识别以及文本内容识别两个步骤，由于所识别的文字都是正向。因此对文本方向未作过多的研究。下面的讲解主要围绕文本区域识别与文本内容识别展开。

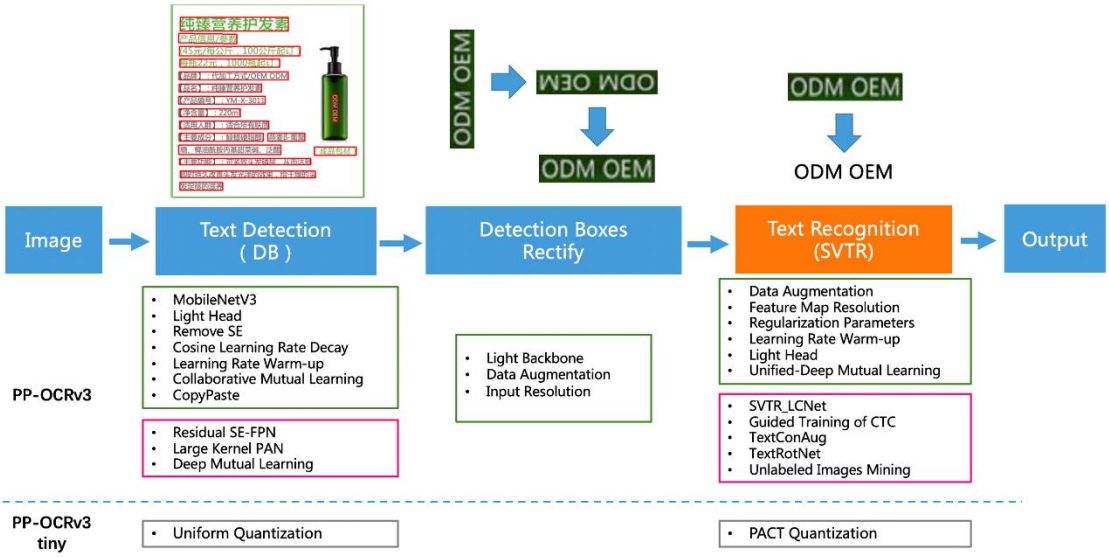

图 3 PP-OCRv3 模型文字识别流程

1.1.4 项目编码环境

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™：2022.2.0
- OpenCV：4.5.5
- Visual Studio：2022
- C#框架：.NET 6.0
- OpenCvSharp：OpenCvSharp4

1.1.5 源码下载方式

项目所使用的源码均已经在 Github 和 Gitee 上开源，

Github:

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

Gitee:

```
git clone https://gitee.com/guojin-yan/Csharp_and_OpenVINO_deploy_PaddleOCR.git
```

1.1.6 模型下载方式

由于模型大小超 Github 上限，因此无法直接上传到代码厂库，可根据下文题都的防止自行下载和转换，如果有啥问题，可以直接下载转换好的模型：

链接：https://pan.baidu.com/s/1VVO1v4hCmPjNHFlLu8r1UA

提取码：9966

1.2 C#中调用 OpenVINO™ 实现

1.2.1 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。

该方式具体实现教程，可以参考下面这一篇 Github：

```
https://github.com/guojin-yan/OpenVinoSharp.git
```

为了方便直接使用，该项目中已经包含了上述项目创建动态链接库的代码，并将 OpenVINO™ 升级到了 2022.2 版本，在使用时可以直接配置使用。

1.2.2 C#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 1C++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。

```
private const string openvino_dll_path = @"E:\Git_space\基于Csharp和OpenVINO部署PaddleOCR模型\CppOpenVinoAPI\dll\OpenVinoSharp.dll";

[DllImport(openvino_dll_path, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
1 个引用
public extern static IntPtr core_init(string model_file, string device_name);

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

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

[DllImport(openvino_dll_path, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
1 个引用
public extern static IntPtr load_image_input_data(IntPtr inference_engine, string input_node_name, ref byte image_data, ulong image_size, int type);

[DllImport(openvino_dll_path, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
1 个引用
public extern static IntPtr load_input_data(IntPtr inference_engine, string input_node_name, ref float input_data);

[DllImport(openvino_dll_path, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
1 个引用
public extern static IntPtr core_infer(IntPtr inference_engine);

[DllImport(openvino_dll_path, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
1 个引用
public extern static void read_infer_result_F32(IntPtr inference_engine, string output_node_name, int data_size, ref float inference_result);

[DllImport(openvino_dll_path, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
1 个引用
public extern static void read_infer_result_I32(IntPtr inference_engine, string output_node_name, int data_size, ref int inference_result);

[DllImport(openvino_dll_path, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
1 个引用
public extern static void core_delet(IntPtr inference_engine);
```

图 4 动态链接库引入

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™ 2022.1 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™ 推理套件部署自己的模型了。

1.3 PaddleOCR 文字区域识别

1.3.1 模型介绍-Detection model

文字区域识别主要使用的是 Detection model，该模型可以直接从 PaddleOCR 上下载训练好的模型，并可以直接使用。

表 2 ppocr_v3_det 模型信息

	Input	Output
名称	x	sigmoid_0.tmp_0
形状	[bath_size,3,640,640]	[bath_size,1,640,640]
数据类型	Float32	Float32

表 2 为 ONNX 格式下模型的输入与输出相关信息，如果切换模型格式一定要注意模型的输出与输入节点名称。文字区域识别模型输入与输出已经进行了固定，为 640×640 大小，后续使用时不用再进行更改。数据归一化时采用标准差归一化。

1.3.2 模型下载与转换

（1）PaddlePaddle 模型下载方式：

命令行直接输入以下代码，或者浏览器输入后面的网址即可。

```
wget https://paddleocr.bj.bcebos.com/PP-OCRv3/chinese/ch_PP-OCRv3_det_infer.tar
```

下载好后将其解压到文件夹中，便可以获得 Paddle 格式的推理模型。

（2）转换为 ONNX 格式：

该方式需要安装 paddle2onnx 和 onnxruntime 模块。在命令行中输入以下指令进行转换：

```
cd E:\Paddle\PaddleOCR
paddle2onnx --model_dir ./ch_PP-OCRv3_det_infer --model_filename inference.pdmodel --params_filename
inference.pdiparams --save_file ./inference/det_v3_onnx/model.onnx --opset_version 10 --
input_shape_dict="{'x':[-1,3,640,640]}" --enable_onnx_checker True
```

其中在指定模型输入大小时，-1 代表不指定，可以模型推理部署时在进行指定。运行上述指令后在./inference/det_onnx/路径下可以找到该模型

（3）转换为 IR 格式

利用 OpenVINO™ 模型优化器，可以实现将 ONNX 模型转为 IR 格式。

在 OpenVINO™ 环境下，切换到模型优化器文件夹，直接使用下面指令便可以进行转换。

```
cd .\openvino\tools
mo –input_model det_onnx/model.onnx
```

经过上述指令模型转换后，可以在当前文件夹下找到转换后的三个文件。

1.3.3 文字区域识别 C#实现模型推理

经过测试，paddle、ONNX 以及 IR 模型都可以使用，此处使用 IR 模型进行展示。

（1）导入模型相关信息

```
// 模型路径
string model_file_path_det = @"E:\Text_Model\ppocr_model_v3\dec_onnx\model.onnx";
// 模型输入节点
string input_node_name_det = "x";
// 模型输出节点
string output_node_name_det = "softmax_5.tmp_0";
// 设备名
string device_name = "CPU";
```

对于模型路径，此处要使用全英文的路径，尽量不使用中文，容易出错。由于本机设备未搭载 Intel 的显卡，因此只在 CPU 上进行测试。对于有显卡设备的可以将设备名改为 GPU，或者使用 AUTO 让套件自行根据实际情况选择。其中模型的输入与输出节点名一定要正确，前后不得有空格。

（2）初始化推理核心

```
Core pridector_det = new Core(model_file_path_det, device_name);
```

通过初始化上一节构建的推理类，实现模型的本地读取、加载设备以及创建推理通道。

（3）设置输入节点形状

```
// 设置模型节点形状
ulong[] input_size_det = new ulong[] { 1, 3, 640, 640 };
pridector_det.set_input_sharp(input_node_name_det, input_size_det);
```

输入节点形状的设置需要根据模型或者数据来，对于该模型，其输入形状已经固定，bath_size 还未设置，新版 OpenVINO™ 支持动态设置形状，但在推理前要保证不存在动态形状。

（4）加载推理数据

```
// 设置输入数据
byte[] image_data_det = new byte[2048 * 2048 * 3];
ulong image_size_det = new ulong();
image_data_det = image.ImEncode(".bmp");
image_size_det = Convert.ToUInt64(image_data_det.Length);
// 将图片数据加载到模型
pridector_det.load_input_data(input_node_name_det, image_data_det, image_size_det, 0);
```

由于我们使用的 OpenCV 处理的图片，Mat 数据格式不是编程语言的基本数据类型，是一个类，因此无法再 C++与 C#之间传递，为了保证图片数据的传输，将图片数据编码为 byte 数据格式，在 C++中再将数据进行解码，经过测试，虽然经过编解码操作，但在时间上不会由太大的差距。

对于图片的预处理（包括转换 RGB、缩放处理、归一化处理）已经封装到 dll 文件中，所以在此处无需对图片做过多的处理，直接调用接口函数进行数据的加载。

（5）模型推理

```
// 模型推理
pridector_det.infer();
```

（6）读取推理结果数据

```
int result_det_length = 640 * 640;
float[] result_det = pridector_det.read_infer_result<float>(output_node_name_det, result_det_length);
```

从前面的模型信息中我们可以得知，模型输出的数据格式与大小，按照要求进行读取数据到一维数组中。

1.3.4 文字区域识别 C#实现模型结果处理

（1）重构结果图像

```
// 将模型输出转为 byte 格式
byte[] result_det_byte = new byte[result_det_length];
for (int i = 0; i < result_det_length; i++)
{
    result_det_byte[i] = (byte)(result_det[i] * 255);
}
Mat image_det = new Mat(640, 640, MatType.CV_8UC1, result_det_byte);
```

文字区域识别模型输出实际为一通道的灰度图，首先将其转为 8 位无符号整形数据，

此处需要将输出结果乘 255，原因是输出的数据为归一化的数据。

图 5 文字区域识别输出结果与输入结果对比

通过图 5 可以看出，想要提取文字区域，只需要提取模型输出结果中的白色位置即可。

（2）提取文字区域矩形框

```
// 查找文字区域框
Rect[] text_rects = find_rect(image_det);
```

为了提取矩形框，此处封装了一个方法，专门用来提取二值图像中的白色区域，其主要实现是利用 OpenCV 的方法实现的，源代码可以参考图 6 或下载源码查看。

```
/// <summary>
/// 获取文字区域的矩形框信息
/// </summary>
/// <param name="source_image">图片</param>
/// <returns>矩形框</returns>
1 个引用
public static Rect[] find_rect(Mat source_image)
{
    Mat image = source_image.Clone();
    //中值滤波或腐蚀去除很小的点
    Cv2.MedianBlur(image, image, 3);
    Mat element = Cv2.GetStructuringElement(MorphShapes.Rect, new Size(1, 1), new Point(-1, -1));
    Cv2.Erode(image, image, element, new Point(-1, -1), 1, BorderTypes.Default, new Scalar());
    Mat element2 = Cv2.GetStructuringElement(MorphShapes.Rect, new Size(2, 2), new Point(-1, -1));
    Cv2.Dilate(image, image, element2, new Point(-1, -1), 1, BorderTypes.Default, new Scalar());

    Point[][] contours;
    HierarchyIndex[] hierarchy; //轮廓拓扑结构变量
    Cv2.FindContours(image, out contours, out hierarchy, RetrievalModes.External,
        ContourApproximationModes.ApproxNone);

    Rect[] rects = new Rect[contours.Length];
    for (int i = 0; i < contours.Length; i++)
    {
        Rect rect = Cv2.BoundingRect(contours[i]);
        rect = enlarge_rect(rect);
        rects[i] = rect;
    }
    return rects;
}
```

图 6 find_rect()函数方法

（3）裁剪文字区域

上一步已经实现了文本区域的位置，由于文本内容识别模型是一张张识别文字区域的，因此需要经文字区域裁剪下来。

```
Mat image_rect = image.Clone();
Cv2.Resize(image_rect, image_rect, new Size(640, 640));
//裁剪文字区域
Mat[] text_images = cut_image_roi(image_rect, text_rects);
```

```
/// <summary>
/// 裁剪文字区域
/// </summary>
/// <param name="source_image">原图片</param>
/// <param name="rects">矩形区域数组</param>
/// <returns>文字区域mat</returns>
1 个引用
public static Mat[] cut_image_roi(Mat source_image, Rect[] rects)
{
    Mat image = source_image.Clone();
    Mat[] rois = new Mat[rects.Length];

    for (int r = 0; r < rects.Length; r++)
    {
        Mat roi = new Mat(image, rects[r]);
        rois[r] = roi;
    }
    return rois;

}
```

图 7 cut_image_roi()方法

文字区域裁剪主要通过封装图 7 方法实现，由于模型输出位 640×640，因此在裁剪前，需要将原图片缩放到指定大小。

最终经过裁剪，会生成若干个文字区域，如图 8 所示，或许文字识别需要将这些文字区域带入逐一识别。

图 8 文本区域裁剪结果

1.4 PaddleOCR 文字内容识别

1.4.1 模型介绍- Recognition model

文字区域识别主要使用的是 Recognition model，该模型可以直接从 PaddleOCR 上下载训练好的模型，并可以直接使用，目前官网有多种识别模型，此处下载位 Chinese and English general PP-OCR model。

表 3 ppocr_v3_rec 模型信息

	Input	Output
名称	x	sigmoid_5.tmp_0
形状	[bath_size, 3, 32, -1]	[bath_size, -1, 6625]
数据类型	Float32	Float32

表 3 为 ONNX 格式下模型的输入与输出相关信息，可以看出，该模型和文字区域识别模型相差很多。此处有几个注意点：

（1）模型的输出形状与输入有关：

可以看出模型输入的形状第一个维度形状是没有固定的，这个维度形状和输出的第二个维度形状成比例关系，输出的大小为输入的四分之一向下取整。

（2）动态调整模型的输入：

此处主要针对不同文本长度的输入，因为输入的宽度决定了最多能识别的字符数量，因此在识别时，输入图形的宽度是要根据文字的数量动态调整的，在使用时主要是通过对比输入图片的长宽比实现的。

1.4.2 模型下载与转换

（1）PaddlePaddle 模型下载方式：

命令行直接输入以下代码，或者浏览器输入后面的网址即可。

```
wget https://paddleocr.bj.bcebos.com/PP-OCRv3/chinese/ch_PP-OCRv3_rec_infer.tar
```

下载好后将其解压到文件夹中，便可以获得 Paddle 格式的推理模型。

（2）转换为 ONNX 格式：

该方式需要安装 paddle2onnx 和 onnxruntime 模块。在命令行中输入以下指令进行转换：

```
cd E:\Paddle\PaddleOCR
paddle2onnx --model_dir ./ch_PP-OCRv3_rec_infer --model_filename inference.pdmodel --params_filename inference.pdiparams --save_file ./inference/rec_v3_onnx/model.onnx --opset_version 10 --input_shape_dict="{'x':[-1,3,32,-1]}" --enable_onnx_checker True
```

其中在指定模型输入大小时，此处需要将第一个维度指定为-1。运行上述指令后在./inference/det_onnx/路径下可以找到该模型

（3）转换为 IR 格式

利用 OpenVINO™ 模型优化器，可以实现将 ONNX 模型转为 IR 格式。

在 OpenVINO™ 环境下，切换到模型优化器文件夹，直接使用下面指令便可以进行转换。

```
cd .\openvino\tools
mo –input_model rec_onnx/model.onnx
```

经过上述指令模型转换后，可以在当前文件夹下找到转换后的三个文件。

1.4.3 文字内容识别 C#实现模型推理

经过测试， ONNX 以及 IR 模型都可以使用，paddle 模型由于尺寸已经固定，无法动态调整，所以使用起来有些问题，此处使用 IR 模型进行展示。

（1）导入模型相关信息

```
// 模型路径
string model_file_path_det = @"E:\Text_Model\ppocr_model_v3\rec_onnx\model.onnx";
// 模型输入节点
string input_node_name_det = "x";
// 模型输出节点
string output_node_name_det = "softmax_5.tmp_0";
// 字符字典
string dict_path = @"E:\Git_space\基于 Csharp 和 OpenVINO 部署 PaddleOCR 模型\model\ppocr_keys_v1.txt";
```

此处需要导入一个字典文件，用于根据索引值查找文字内容，后面将其读取到一个字符串列表中。

（2）初始化推理核心

```
Core pridector_rec = new Core(model_file_path_rec, device_name);
```

通过初始化推理类，实现模型的本地读取、加载设备以及创建推理通道。

这一步已经够建好了文字内容识别推理类，下面的话就可以实现逐张识别文字内容，在一个程序中，我们只需要构建一次推理类即可，后面的逐张识别就可以。

（3）动态调整带识别文字图片形状

动态调整图片形状主要是为了适应含不同文字数量的图片。由于文字数量我们未知，所以只能通过对输入图片长宽等比例缩放，这样能够保证文字在输入时不会发生大的变形，其还能识别更多的文字。主要通过如图 9 所示方法实现。

```
Mat text_image = text_images[epoch].Clone();
text_image = adjust_image_size(text_image);
```

```
/// <summary>
/// 调整文字识别图片大小
/// </summary>
/// <param name="source_image"></param>
/// <returns></returns>
1 个引用
public static Mat adjust_image_size(Mat source_image)
{
    if (source_image.Width * 1.5 < source_image.Height)
    {
        Cv2.Transpose(source_image, source_image);
        Cv2.Flip(source_image, source_image, 0);
    }
    int img_W = source_image.Width;
    int img_H = 32;
    double scale_size = (double)img_W / (double)source_image.Height;

    int max_W = (int)scale_size * 32;
    if (scale_size * img_H > max_W)
    {
        img_W = max_W;
    }
    else
    {
        img_W = (int)scale_size * img_H;
    }
    Cv2.Resize(source_image, source_image, new Size(img_W, img_H));
    return source_image;
}
```

图 9 adjust_image_size()方法

（4）设置输入节点形状

```
// 设置模型节点形状
ulong[] input_size_rec = new ulong[] { 1, 3, (ulong)text_image.Height, (ulong)text_image.Width };
pridector_rec.set_input_sharp(input_node_name_rec, input_size_rec);
```

此处模型输入节点设置参数要以输入图片的尺寸为准，主要就是图片的宽度。

（5）加载推理数据

```
// 设置输入数据
byte[] image_data_rec = new byte[500 * 200 * 3];
ulong image_size_rec = new ulong();
image_data_rec = text_image.ImEncode(".bmp");
image_size_rec = Convert.ToUInt64(image_data_rec.Length);
// 将图片数据加载到模型
pridector_rec.load_input_data(input_node_name_rec, image_data_rec, image_size_rec, 1);
```

此处加载推理数据与上一步中相同，主要就是该步图片归一化方式不同，注意选择模型对应的归一化方式。

（6）模型推理

```
// 模型推理
pridector_rec.infer();
```

（6）读取推理结果数据

```
int text_size = text_image.Width / 4;
int result_rec_length = text_size * 6625;
float[] result_rec = pridector_rec.read_infer_result<float>(output_node_name_rec, result_rec_length);
```

在读取模型输出结果时，需要注意结果形状大小是在变化的，时输入图片宽度四分之一。

1.4.4 文字内容识别 C#实现模型结果处理

（1）结果数据分析

当 bath_size = 1 时，模型输出为：1×text_size×6625，实际就是一个 text_size×6625 的二维数组。其中将索引字典读取到列表后，其长度为 6623，经求证，模型在训练时，会在字典前后补一位，使字典总数为 6625。因此模型输出的代表意义为每个字符串的概率。

（2）划分数组、求每一组的最大值和索引值

以 6625 个为一组，分别求每一组的最大子索引以及最大值，如果识别的字体为有效字体的索引，并将其记录下来。

```
List<float> confindences = new List<float>() ;
List<int> indexs = new List<int>();
int num = 0;
for (int r = 0; r < text_size; r++)
{
    float[] temp = new float[6625];
    for (int j = 0; j < 6625; j++)
    {
        temp[j] = result[r * 6625 + j];
    }
    int index = 1;
    float max = max_index(temp, ref index);
    if (index > 1 && index < 6624)
    {
        indexs.Add(index);
        confindences.Add(max);
        num++;
    }

}
```

（3）动态调整字符识别阈值

为了防止部分文字因为识别度太低导致在过滤时被过滤掉，此处使用一个动态阈值，主要是先求取当前所有文字内容的置信值平均，在平均值的基础上设置阈值大小。

```
List<string> list = new List<string>();
if (confindences.Count>0) {
    float aver_confindence = confindences.Average();

    for (int r = 0; r < num; r++)
    {
        if (text_size < 5 && confindences[r] > 0)
        {
            list.Add(dict[indexs[r] - 1]);
        }
        else
        {
            if (confindences[r] > aver_confindence-0.5)
            {
                if (list.Count > 0 && list[list.Count - 1] == dict[indexs[r] - 1])
                {
                    if (confindences[r] - confindences[r - 1] > 0.05 || confindences[r - 1] - confindences[r] > 0.05){}
                    else {
                        list.Add(dict[indexs[r] - 1]);
                    }

                }
                else {
                    list.Add(dict[indexs[r] - 1]);
                }

            }
        }
    }
}
```

最后将符合条件的结果字符存放到列表中。

1.5 PaddleOCR 文字识别模型在 OpenVINO™ 部署效果

1.5.1 文字识别效果

当我们输入一张带有文字的图片时，该图片经过文字区域识别，获取文字区域，并将文字区域使用矩形框标注出来，最后将文字区域带入文本识别模型，便可以获得输入图片上的所有文字。如图 10 所示，为方便观察，将图片识别出的文字放在了一续航相同大小的图片上与原图相同的区域。

图 10 PaddleOCR 文字识别效果

可以看出，经过 PaddleOCR 模型的推理，已经能够将图片上的中英文字符完全提取出

来，并且可以获取的位置与内容

1.5.2 时间测试

为了比较 C++与 C#之间的模型运行性能，此处对其进行了测试。测试模型均采用上文中所下载的服务端模型，ONNX 格式以及 IR 格式；测试图片采用同一张图片，有 16 个文本区域。测试采用 100 次求平均，获得以下结果，如表 4 所示。

表 4 不同平台以及不同模型格式 PaddleOCR 模型运行时间(ms)

平台	模型格式	Detection model				Recognition model							总计
		模型读取	加载数据	模型推理	结果处理	模型读取	加载数据		模型推理		结果处理		
							文字区域	总计	文字区域	总计	文字区域	总计	
C++	ONNX	192	4	52	2	173	0.13	2	11.19	179	0.75	12	616
	IR	125	4	51	2	101	0.13	2	10.63	170	0.69	11	466
C#	ONNX	197	6	54	3	176	0.31	5	12.13	194	0.88	14	649
	IR	126	6	52	3	101	0.31	5	11.81	189	0.88	14	496

注： 模型读取：读取本地模型，加载到设备，创建推理通道；

加载数据：将待推理数据进行处理并加载到模型输入节点；

模型推理：模型执行推理运算；

结果处理：在模型输出节点读取输出数据，并转化为我们所需要的结果数据。

通过表 4 可以看出，在 C++和 C#中，利用 OpenVINO 部署 PaddleOCR 模型并没有太大的时间差异，说明动态链接库的使用并没有影响程序进程；其次，将模型转为 OpenVINO™ 的 IR 格式，会加快模型推理速度，降低模型运行时间。

1.6 总结

在该项目中，基于 C#和 OpenVINO 部署 PaddleOCR 模型，实现了图片文本内容的提取和识别，打通了 C#、OpenVINO、PaddleOCR 模型之间的障碍。经过时间测试，在 C++和 C#中，利用 OpenVINO 部署 PaddleOCR 模型并没有太大的时间差异，这对我们开发 C#软件调用 PaddleOCR 模型提供了选择。

在该项目中，除了利用动态链接库在 C#中实现调用 OpenVINO 推理套件这一个难点之外，其主要的难点就是模型的结果处理。其主要表现两个方面：

第一点在文字区域识别结果提取上，保证文字区域提取精确这会直接影响文字内容识别。在该项目上主要通过寻找合适的文字框缩放比例来实现，经过验证该方式适合大多数情况。

第二点在文字内容识别的输入上，在该项目中，采用的逐个文字区域识别，因此每个文字区域输入模型时会有不同的大小，这会导致每一张图片都要进行一次模型推理。比较好的解决办法是对文本区域大小进行划分，将一定范围打下的图片和为一组，缩放为相同大小，带入模型，进行多 bath_size 的模型推理，这会大大减少模型推理的次数，降低推理时间。