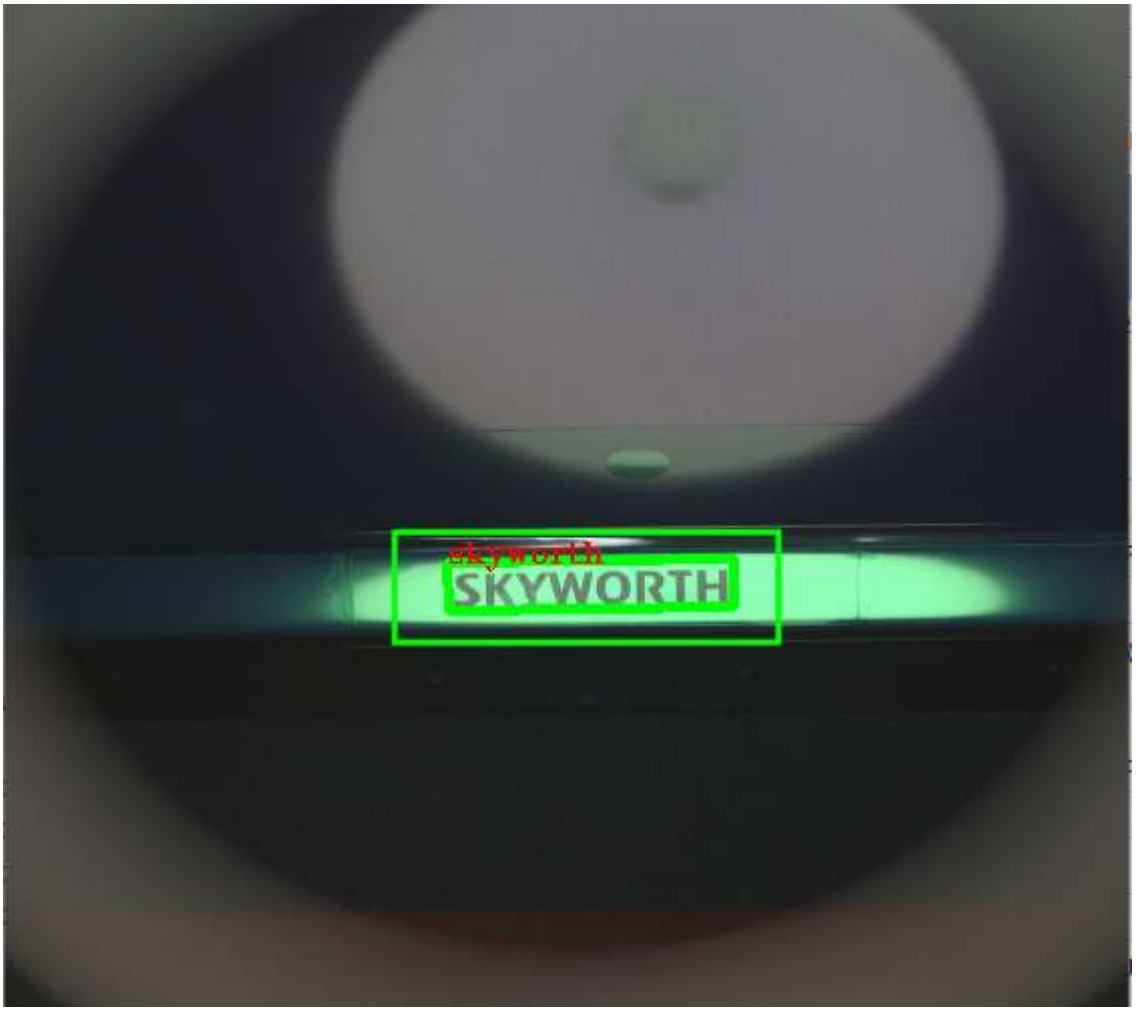
OCR识别：EAST+CRNN在c#上的调用模块

make by haifengye

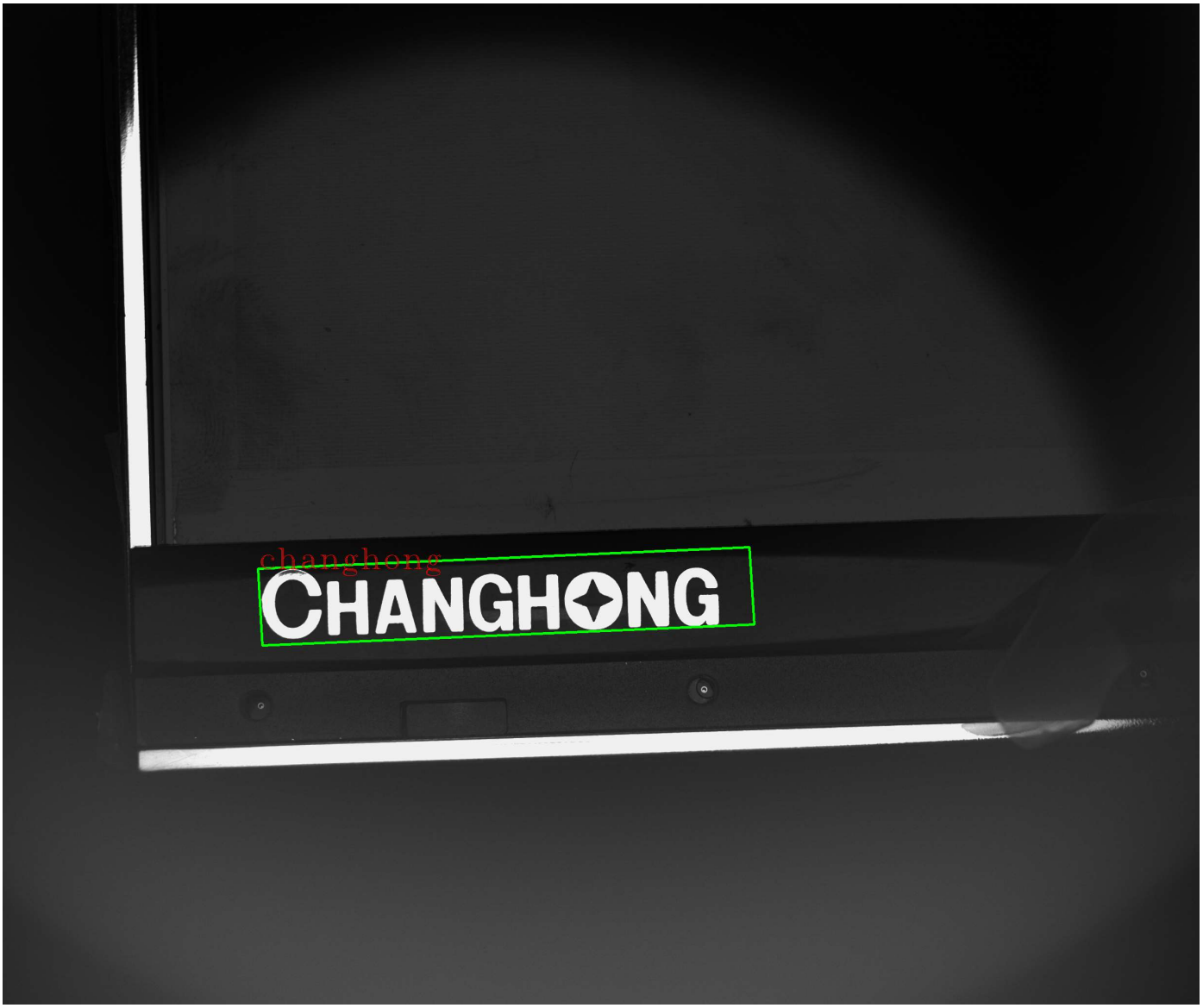
使用示例

In [ ]:

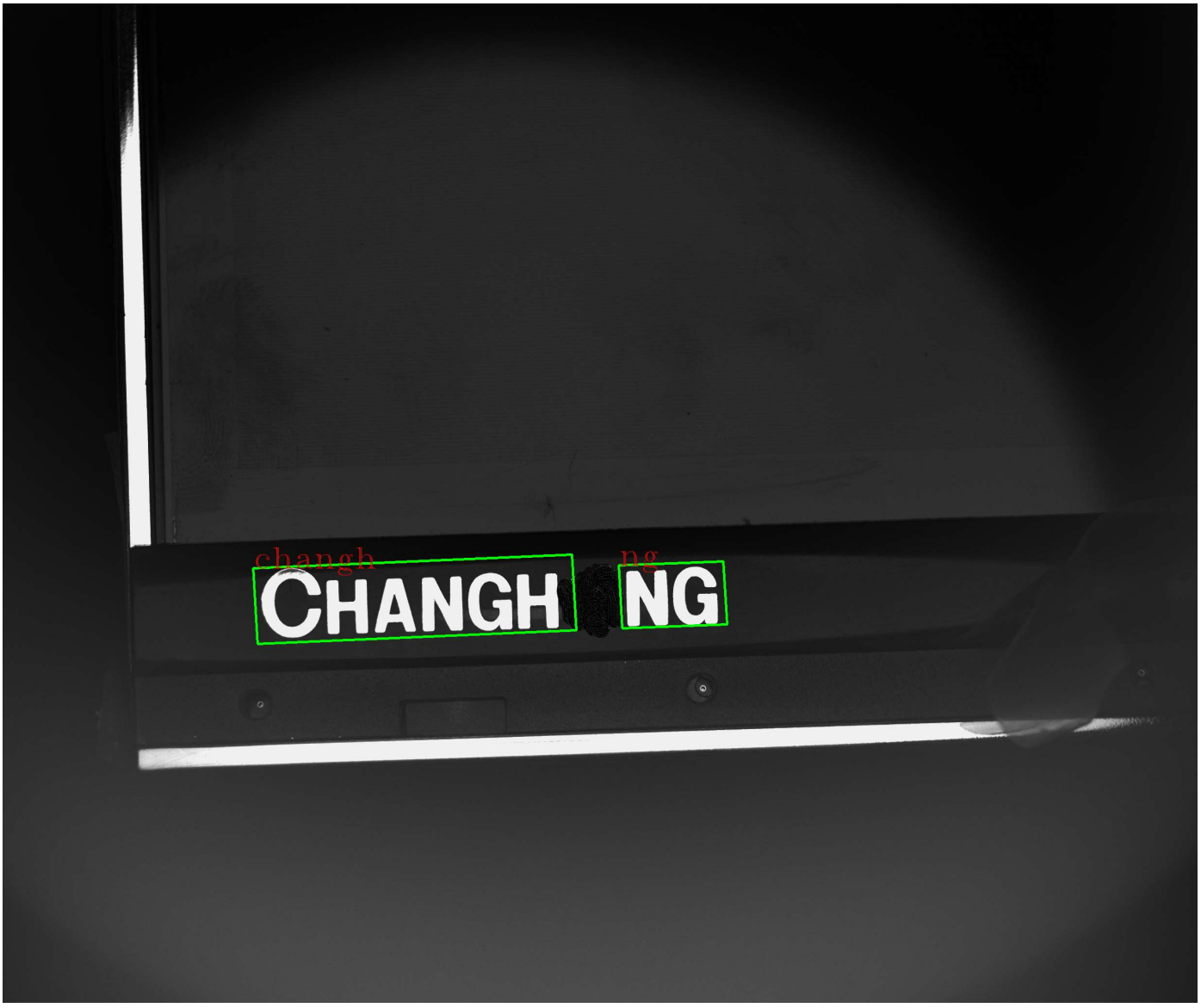
```
private void Demo()
        {
            string dir = System.IO.Directory.GetCurrentDirectory();
            string dectecteoPath = System.IO.Path.Combine(dir, "frozen_east_text_detection.pb");
            string recognitionPath = System.IO.Path.Combine(dir, "CRNN_VGG_BiLSTM_CTC.onnx");
            string labelFile = System.IO.Path.Combine(dir, "Alphabet.txt");
            string imgpath = System.IO.Path.Combine(dir, "1.bmp");

            //初始化
            OCR_CRNNDetect ocr = new OCR_CRNNDetect(dectecteoPath, recognitionPath, labelFile);


            //检测
            OcrResult[] results = ocr.Detect(imgpath);

            foreach (var result in results)
            {
                Console.WriteLine("字符，区域坐标[第一点(x1,y1)、第二点(x2,y2)、第三点(x3,y3)、第四点
                Console.WriteLine(result.Test + "  " + "[" +
                     "(" + result.Vertices[0].X.ToString() + "," + result.Vertices[0].Y.ToString() +
                     "(" + result.Vertices[1].X.ToString() + "," + result.Vertices[1].Y.ToString() +
                     "(" + result.Vertices[2].X.ToString() + "," + result.Vertices[2].Y.ToString() +
                     "(" + result.Vertices[3].X.ToString() + "," + result.Vertices[3].Y.ToString() +
            }
        }
```


skyworth
SKYWORTH


沪A·Z0007


changhong
CHANGHONG


hanghon
HANGHON




```
字符，区域坐标[第一点(x1,y1)、第二点(x2,y2)、第三点(x3,y3)、第四点(x4,y4)]
skyworth  [(221.0813,302.3392)(220.7168,278.178)(363.3603,276.4623)(363.7248,300.6236)]
```

在代码文件夹下工程项目中的示例

1先初始化：

In [ ]:

```
        private void Init3()
        {

            dir = System.IO.Directory.GetCurrentDirectory();
            string dectecteoPath = System.IO.Path.Combine(dir, "frozen_east_text_detection.pb");
            string recognitionPath = System.IO.Path.Combine(dir, "CRNN_VGG_BiLSTM_CTC.onnx");
            string labelFile = System.IO.Path.Combine(dir, "Alphabet.txt");

            ocr = new OCR_CRNNDetect(dectecteoPath, recognitionPath, labelFile);

        }
```

2、在做检测：

In [ ]:

```
        private void Detect3()
        {
            string imgpath = System.IO.Path.Combine(dir, "1.bmp");
            OcrResult[] results = ocr.Detect(imgpath);

            foreach (var result in results)
            {
                Console.WriteLine("字符，区域坐标[第一点(x1,y1)、第二点(x2,y2)、第三点(x3,y3)、第四点
                Console.WriteLine(result.Test + "  " + "[" +
                    "(" + result.Vertices[0].X.ToString() + "," + result.Vertices[0].Y.ToString() +
                    "(" + result.Vertices[1].X.ToString() + "," + result.Vertices[1].Y.ToString() +
                    "(" + result.Vertices[2].X.ToString() + "," + result.Vertices[2].Y.ToString() +
                    "(" + result.Vertices[3].X.ToString() + "," + result.Vertices[3].Y.ToString() +
            }
        }
```

先按《m模型初始化》按键，在按《检测》按键

OCR_CRNNDetect ocr = new OCR_CRNNDetect(dectecteoPath, recognitionPath, labelFile);

其中，dectecteoPath为OCR定位模型(EAST)的模型路径，recognitionPath为OCR识别模型(CRNN)模型路径，labelFile为模型的标签文件路径

OcrResult[] results = ocr.Detect(imgpath);

imgpath是待检测图像的路径

结果保存在OcrResult格式中，参考示例，其提供检测的基本信息：字符，区域坐标[第一点(x1,y1)、第二点

(x2,y2)、第三点(x3,y3)、第四点(x4,y4)]

In [ ]:

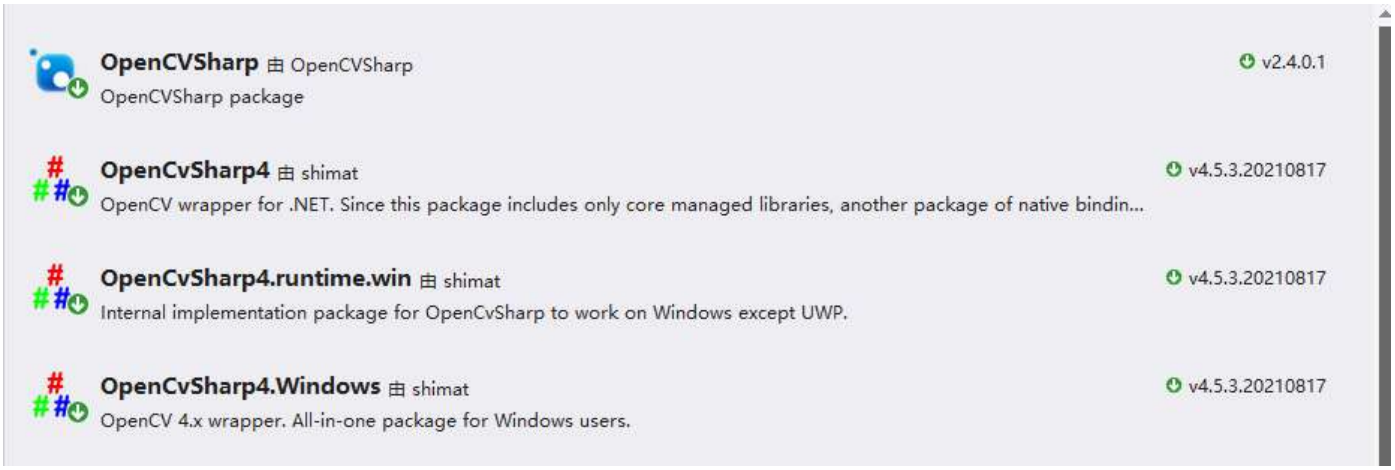
```
1、string Test
2、Point2f[] Vertices
```

In [ ]:

```
除了使用图像路径的方式进行检测
OcrResult[] Detect(string imgpath)
还可以使用图像本身进行检测
OcrResult[] Detect(System.Drawing.Bitmap img)
要求使用System.Drawing.Bitmap 格式的图像。
```

环境配置

在NutGet管理上搜索安装OpenCVSharp、OpenCvSharp4、OpenCvSharp4.runtime.win、OpenCvSharp4.Windows四个包



说明OpencvSharp是opencv为C#提供的源码与接口，接下来我们只要使用OpencvSharp中dnn模块中来实现yolo模型的调用。

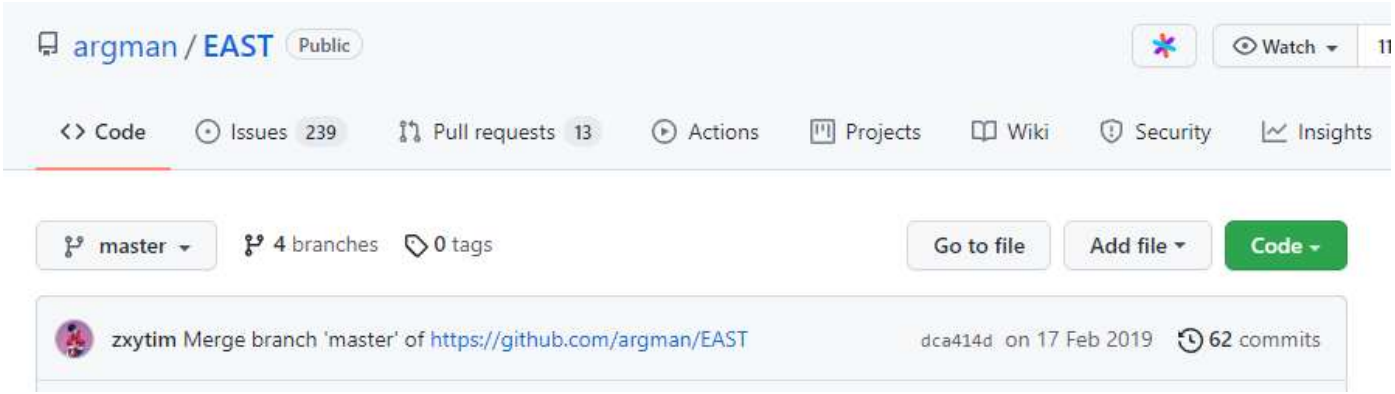
模块参数介绍

OCR_CRNNConfig参数

In [ ]:

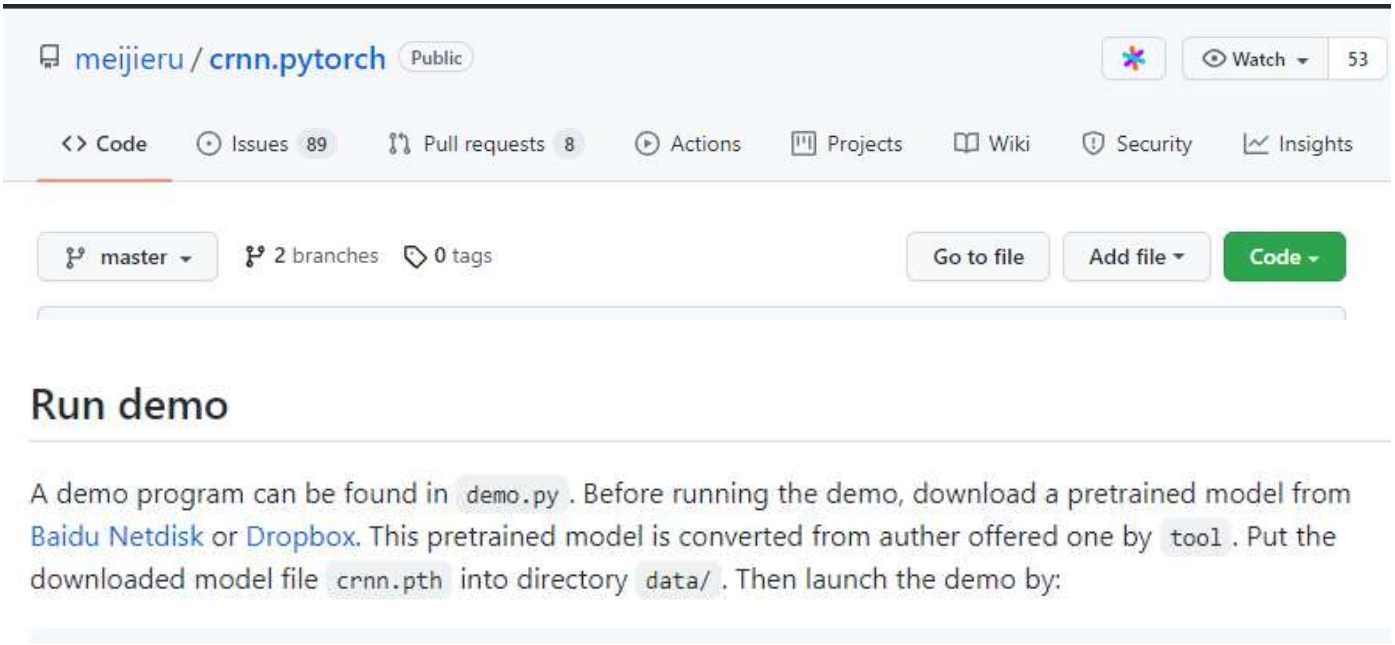
```
    public class OCR_CRNNConfig
{
    /// <summary>
    /// 文本检测模型文件路径
    /// </summary>
    public string DetectorModelPath { set; get; }


    /// <summary>
    /// 文本识别模型文件路径
    /// </summary>
    public string RecognitionModelPath { set; get; }

    /// <summary>
    /// 置信度阈值
    /// </summary>
    public float ConfThreshold { set; get; }

    /// <summary>
    /// nms 阈值
    /// </summary>
    public float NmsThreshold { set; get; }

    /// <summary>
    /// 模型要求的的图片大小
    /// </summary>
    public int ImgWidth { set; get; }

    /// <summary>
    ///模型要求的的图片大小
    /// </summary>
    public int ImgHight { set; get; }

    /// <summary>
    /// 设置分类
    /// </summary>
    public string Alphabet { set; get; }

    /// <summary>
    /// 是否显示图像
    /// </summary>
    public bool IsDraw { set; get; }
}
```

其中DetectorModelPath、RecognitionModelPath、Alphabet即为EAST的模型路径、CRNN模型路径，Alphabet为标签
可以在EAST的github网站 https://github.com/argman/EAST (https://github.com/argman/EAST) 上下载模型。 可以在CRNN的github网站 https://github.com/meijieru/crnn.pytorch (https://github.com/meijieru/crnn.pytorch) 上下载模型。



Download

1. Models trained on ICDAR 2013 (training set) + ICDAR 2015 (training set): BaiduYun link GoogleDrive
2. Resnet V1 50 provided by tensorflow slim: slim resnet v1 50



Run demo

A demo program can be found in `demo.py`. Before running the demo, download a pretrained model from Baidu Netdisk or Dropbox. This pretrained model is converted from auther offered one by `tool`. Put the downloaded model file `crnn.pth` into directory `data/`. Then launch the demo by:

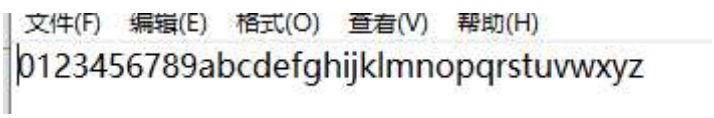
其中RCNN为pytorch模型，需要将其转化为ONNX格式。

当然我们不是为了直接使用其提供的模型，而是根据需求，修改并训练出满足自己需求的模型，但训练的模型与下载网站的格式是一致的，接下来我们使用网站模型进行检测。

Threshold是最小置信度阈值

NmsThreshold是nms 阈值

LabelsFile是标签文件路径，对应与debug中的Alphabet.txt，存储这对应的类别、标签



ImgWidth与ImgHight为模型对图片要求的大小，对应大小为320×320

OcrResult 结果存储格式

In [ ]:

```
public class OcrResult
{
    /// <summary>
    /// 字符区域的位置
    /// </summary>
    public Point2f[] Vertices { set; get; }

    /// <summary>
    /// 识别出的字符
    /// </summary>
    public string Test { set; get; }

    internal static OcrResult Add(Point2f[] points,string text)
    {
        return new OcrResult()
        {
            Vertices = points,
            Test = text
        };
    }
}
```

模块介绍

对接接口

In [ ]:

```
    /// <summary>
    /// 初始化
    /// </summary>
    /// <param name="dectecteoPath"> 检测模型的路径</param>
    /// <param name="recognitionPath">识别模型的路径</param>
    /// <param name="分类">标签路径</param>
    /// <param name="imgWidth">模型要求的的图片大小</param>
    /// <param name="imgHigh">模型要求的的图片大小</param>
    /// <param name="threshold">置信度阈值</param>
    /// <param name="nms">nms 阈值</param>
    public OCR_CRNNDetect(string dectecteoPath, string recognitionPath, string labelsFile,
                  int imgWidth = 320, int imgHigh = 320, float threshold = 0.5f, float nms = 0.
```

In [ ]:

```
    /// <summary>
    /// 检测
    /// </summary>
    /// <param name="imgpath">图像路径</param>
    /// <returns></returns>
     public OcrResult[] Detect(string imgpath)
```

In [ ]:

```
        /// <summary>
        /// 检测
        /// </summary>
        /// <param name="img"></param>
        /// <returns></returns>
        public OcrResult[] Detect(System.Drawing.Bitmap img)
```

内部方法实现

定位模型

In [ ]:

```
        /// <summary>
        ///  文本检测模型图片预处理
        /// </summary>
        /// <param Mat img</param>
        /// <returns></returns>
        private Mat DetectoterPretreatment(Mat img)

        /// <summary>
        /// 初始化文本检测打开pd模型
        /// </summary>
        private Net InitializeDetectoterModel(string pathModel)

        /// <summary>
        /// 检测的后处理
        /// </summary>
        /// <param name="image"></param>
        /// <param name="results">1、字符位置；2、置信度分数</param>
        /// <returns></returns>
        private OcrResult[] Postprocess(Mat image, Mat[] results)

        /// <summary>
        /// 解码
        /// 该模型需要进行解码操作：属于ocr模型中编码-解码网络结构
        /// </summary>
        /// <param name="scores">置信度分数</param>
        /// <param name="geometry">字符位置</param>
        /// <param name="boxes">输出 最终位置框</param>
        /// <param name="confidences">输出：最终得分</param>
        private void Decode(Mat scores, Mat geometry,
            out IList<RotatedRect> boxes, out IList<float> confidences)

        /// <summary>
        /// 将结果在图像上画出
        /// </summary>
        /// <param name="image"></param>
        /// <param name="vertices"> 位于图像中的位置</param>
        /// <param name="text"> 识别结果</param>
        private void Draw(Mat image, Point2f[] vertices, string text)

        /// <summary>
        /// 旋转图像并剪切字符区域
        /// </summary>
        /// <param name="image"></param>
        /// <param name="vertices"></param>
        /// <param name="crops"></param>
        private void GetCropImage(Mat image, Point2f[] vertices, out Mat crops)

        /// <summary>
        /// 检测流程
        /// </summary>
        /// <param name="img"></param>
        /// <returns></returns>
        private OcrResult[] Process(Mat img)

        /// <summary>
        /// 模型初始化
        /// </summary>
        private void InitDetectorModel()
```

识别模型

In [ ]:

```
        /// <summary>
        ///  文本识别模型模型图片预处理
        /// </summary>
        /// <param Mat img</param>
        /// <returns></returns>
        private Mat RecognitionPretreatment(Mat img)

        /// <summary>
        /// 初始化识别模型 打开pd模型
        /// </summary>
        private Net InitializeRecognitionModel(string pathModel)

        /// <summary>
        /// 读入标签
        /// </summary>
        /// <param name="pathLabels"></param>
        /// <returns></returns>
        private string ReadLabels(string pathLabels)

        /// <summary>
        /// 检测的后处理
        /// </summary>
        /// <param name="results"></param>
        /// <returns></returns>
        private string Postprocess(Mat scores)

        /// <summary>
        /// 识别流程
        /// </summary>
        /// <param name="img"></param>
        /// <returns></returns>
        private string RecognitionProcess(Mat img)

        /// <summary>
        /// 模型初始化
        /// </summary>
        private void InitRecognitionModel()
```

原理介绍

定位模型

图片预处理

1、EAST模型的输入图像进行mean操作，使数据分布均匀，加速学习
2、EAST模型一般对图像输入大小由格式要求为320×320

In [ ]:

```
private Mat DetectoterPretreatment(Mat img)
{
    double scale = 1.0;

    //变为320×320
    Size size = new Size(_config.ImgWidth, _config.ImgHight);

    //数据居中：mean常规操作
    Scalar scalar = new Scalar(123.68, 116.78, 103.94);

    //网络结果是使用同大小一图像进行训练的所以需要将图像转为对应的大小
    Mat blob = CvDnn.BlobFromImage(img, scale, size, scalar, true, false);

    return blob;
}
```

```
[[[[0.28235295 0.27058825 0.2784314  ... 0.25490198 0.23137257
    0.21176472]
   [0.28627452 0.26666668 0.27450982 ... 0.2627451  0.2392157
    0.21960786]
   [0.29803923 0.2784314  0.28627452 ... 0.28235295 0.25882354
    0.227451  ]
   ...
   [0.14509805 0.14509805 0.15686275 ... 0.16078432 0.16078432
    0.15686275]
   [0.15686275 0.14901961 0.15686275 ... 0.14901961 0.14117648
    0.14509805]
   [0.15294118 0.14509805 0.16078432 ... 0.14901961 0.13725491
    0.13725491]]
```

读入标签

```
"0123456789abcdefghijklmnopqrstuvwxyz",
```

初始化模型

对于.pd格式模型使用opencvsharp中dnn模型的ReadNet函数

Net ReadNet(string model, string config = "", string framework = "");

由于没有GPU，只能使用CPU，所以需要设置net.SetPreferableTarget(Target.CPU)

检测的后处理

EAST 模型输出output分为两层，
置信度分数：[1, 1, 80, 80]形式
字符位置：[1, 5, 80, 80]形式

置信度分数：
其中一维，二维的1是在这里没有意义，第三为中80代表共检测出80个可能的对象，会随图像变化数值
第四维80含义：代表着可能对象的得分。

字符位置：
其中一维在这里没有意义，
二维的5分别是角度、字符区域的四个顶角，
第三为中80代表共检测出80个可能的对象，会随图像变化数值
第四维80含义：代表着可能对象的得分。

模型最终输出数据sharp：[1, 1, 80, 80]、[1, 5, 80, 80]

In [ ]:

```
将输出的两个数据取出：
Mat scores = results[0];                    //置信度得分
Mat geometry = results[1];                  //字符位置
```

In [ ]:

```
//取置信度
//从[1, 1, 80, 80]中取出第三、第四维：[80, 80]
//从[80, 80]中取第二维数据[80]
var scoresData = Enumerable.Range(0, height).Select(
    row => scores.At<float>(0, 0, y, row)).ToArray();
```

In [ ]:

```
//取位置
//从[1, 5, 80, 80]中取出第三、第四维：[80, 80] 以此排列
//从[80, 80]中取第二维数据[80]
var x0Data = Enumerable.Range(0, height).Select(
    row => geometry.At<float>(0, 0, y, row)).ToArray();
var x1Data = Enumerable.Range(0, height).Select(
    row => geometry.At<float>(0, 1, y, row)).ToArray();
var x2Data = Enumerable.Range(0, height).Select(
    row => geometry.At<float>(0, 2, y, row)).ToArray();
var x3Data = Enumerable.Range(0, height).Select(
    row => geometry.At<float>(0, 3, y, row)).ToArray();
var anglesData = Enumerable.Range(0, height).Select(
    row => geometry.At<float>(0, 4, y, row)).ToArray();
```

In [ ]:

```
//从[80]中取出每一个得分
var score = scoresData[x];
```

In [ ]:

```
if (score >= _config.ConfThreshold)
//大于设置的置信度得分则获取该对于的位置
```

In [ ]:

```
// 解码操作：
// Multiple by 4 because feature maps are 4 time less than input image.
float offsetX = x * 4.0f;
float offsetY = y * 4.0f;
float angle = anglesData[x];

float cosA = (float)Math.Cos(angle);
float sinA = (float)Math.Sin(angle);
float x0 = x0Data[x];
float x1 = x1Data[x];
float x2 = x2Data[x];
float x3 = x3Data[x];
```

In [ ]:

```
角度计算及顶点计算
var value1 = offsetX + cosA * x1 + sinA * x2;
var value2 = offsetY - sinA * x1 + cosA * x2;
Point2f offset = new Point2f(value1, value2);
Point2f p1 = new Point2f(-sinA * h, -cosA * h) + offset;
Point2f p3 = new Point2f(-cosA * w, sinA * w) + offset;
RotatedRect r = new RotatedRect(new Point2f(0.5f * (p1.X + p3.X),
        0.5f * (p1.Y + p3.Y)), new Size2f(w, h), (float)(-angle * 180.0f / Math.PI));
```

完整代码

In [ ]:

```
private void Decode(Mat scores, Mat geometry, out IList<RotatedRect> boxes, out IList<float> confi
{
    confidences = new List<float>();                          //可能对象的置信度集合
    boxes       = new List<RotatedRect>();                    //可能对象的方框集合

    if ((scores == null || scores.Dims != 4 || scores.Size(0) != 1 || scores.Size(1) != 1) ||
        (geometry == null || geometry.Dims != 4 || geometry.Size(0) != 1 || geometry.Size(1) != 5)
        (scores.Size(2) != geometry.Size(2) || scores.Size(3) != geometry.Size(3)))
    {
        return;
    }

    var height = scores.Size(2);                              //图像的宽高
    var width = scores.Size(3);

    for (var y = 0; y < height; ++y)
    {
        //取置信度
        //从[1, 1, 80, 80]中取出第三、第四维：[80, 80]
        //从[80, 80]中取第二维数据[80]
        var scoresData = Enumerable.Range(0, height).Select(row => scores.At<float>(0, 0, y, row)).

        //取位置
        //从[1, 5, 80, 80]中取出第三、第四维：[80, 80] 以此排列
        //从[80, 80]中取第二维数据[80]
        var x0Data = Enumerable.Range(0, height).Select(row => geometry.At<float>(0, 0, y, row)).To
        var x1Data = Enumerable.Range(0, height).Select(row => geometry.At<float>(0, 1, y, row)).To
        var x2Data = Enumerable.Range(0, height).Select(row => geometry.At<float>(0, 2, y, row)).To
        var x3Data = Enumerable.Range(0, height).Select(row => geometry.At<float>(0, 3, y, row)).To
        var anglesData = Enumerable.Range(0, height).Select(row =>
                                                    geometry.At<float>(0, 4, y, row)).ToArra


        for (var x = 0; x < width; ++x)
        {
            //从[80]中取出每一个得分
            var score = scoresData[x];
            if (score >= _config.ConfThreshold)       //大于设置的置信度得分则获取该对于的位置
            {
                // 解码操作：
                // Multiple by 4 because feature maps are 4 time less than input image.
                float offsetX = x * 4.0f;
                float offsetY = y * 4.0f;
                float angle = anglesData[x];

                float cosA = (float)Math.Cos(angle);
                float sinA = (float)Math.Sin(angle);
                float x0 = x0Data[x];
                float x1 = x1Data[x];
                float x2 = x2Data[x];
                float x3 = x3Data[x];

                var h = x0 + x2;
                var w = x1 + x3;


                var value1 = offsetX + cosA * x1 + sinA * x2;
                var value2 = offsetY - sinA * x1 + cosA * x2;
                Point2f offset = new Point2f(value1, value2);
```

```
                Point2f p1 = new Point2f(-sinA * h, -cosA * h) + offset;
                Point2f p3 = new Point2f(-cosA * w, sinA * w) + offset;
                RotatedRect r = new RotatedRect(new Point2f(0.5f * (p1.X + p3.X), 0.5f *
                    (p1.Y + p3.Y)), new Size2f(w, h), (float)(-angle * 180.0f / Math.PI));

                confidences.Add(score);
                boxes.Add(r);
            }
        }
    }
}
```

将结果在图像上画出

由于检测的字符区域可能不是水平的，所以存在 for (int j = 0; j < 4; ++j),即是用四个顶点画图

In [ ]:

```
    /// <summary>
    /// 将结果在图像上画出
    /// </summary>
    /// <param name="image"></param>
    /// <param name="vertices"> 位于图像中的位置</param>
    /// <param name="text"> 识别结果</param>
    private void Draw(Mat image, Point2f[] vertices, string text)
    {
        //区域
        for (int j = 0; j < 4; ++j)
        {
            Cv2.Line(image, (int)vertices[j].X, (int)vertices[j].Y, (int)vertices[(j + 1) % 4].X, (i
        }

        //字符
        Cv2.PutText(image, text, (Point)vertices[1], HersheyFonts.HersheyTriplex, 0.5, Scalar.Red);
        Cv2.ImShow("图片展示：", image);
    }
```

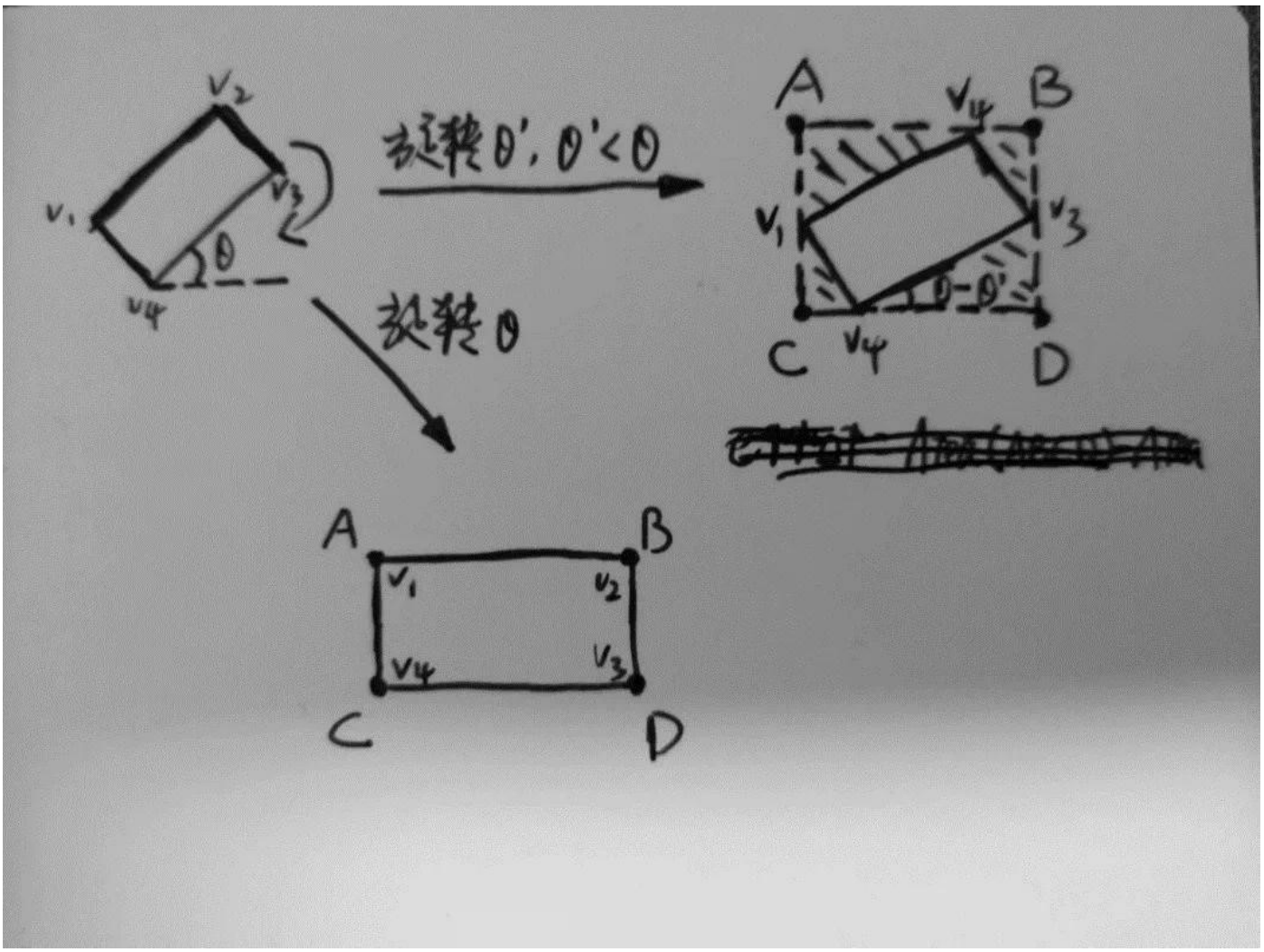
EAST整个处理过程

In [ ]:

```
private OcrResult[] Process(Mat img)
{
    Mat blob = DetectoterPretreatment(img);

    //推理
    _model.DetectoterNet.SetInput(blob);
    _model.DetectoterNet.Forward(_model.OutputBlobs, _model.OutputBlobNames);

    return Postprocess(img, _model.OutputBlobs);
}
```

其中关键语句是 net.Forward()，用于图像的推理工作。

旋转图像并剪切字符区域



In [ ]:

```
/// <summary>
        /// 旋转图像并剪切字符区域
        /// </summary>
        /// <param name="image"></param>
        /// <param name="vertices"></param>
        /// <param name="crops"></param>
        private void GetCropImage(Mat image, Point2f[] vertices, out Mat crops)
        {
            OpenCvSharp.Size outputSize = new OpenCvSharp.Size(100, 32);

            Point2f[] dsts = new Point2f[4];
            dsts[0] = new Point2f(0, outputSize.Height - 1);
            dsts[1] = new Point2f(0, 0);
            dsts[2] = new Point2f(outputSize.Width - 1, 0);
            dsts[3] = new Point2f(outputSize.Width - 1, outputSize.Height - 1);

            //计算旋转矩阵
            Mat rotationMatrix = Cv2.GetPerspectiveTransform(vertices, dsts);

            //旋转图像
            Mat rotatImag = new Mat();
            Cv2.WarpPerspective(image, rotatImag,rotationMatrix,outputSize);

            //
            Cv2.CvtColor(rotatImag, rotatImag, ColorConversionCodes.BGR2BGRA);

            crops = rotatImag;
        }
```

识别模型

预处理

1、CRNN模型的输入图像进行标准/归一化操作

In [ ]:

```
private Mat RecognitionPretreatment(Mat img)
{
    double scale = 1.0 / 127.5;
    Scalar scalar = new Scalar(127.5, 127.5, 127.5);

    Mat blob = CvDnn.BlobFromImage(img, scale, new Size(), scalar);

    return blob;
}
```

加载模型

由于模型是onnx格式，使用dnn中ReadNetFromOnnx函数来读取模型

In [ ]:

```
Net ReadNetFromOnnx(string onnxFile);
```

后处理

模型检测头输出格式：[N,1,37]
其中37仅是由于这个模型仅训练了37个字符

从[N,1,37]变为[N,37]，其中第一维度表示识别到可能的对象，第二维度是没有意义的

In [ ]:

```
//resharp from[N,1,37] to [N,37]
Mat scoresMat = scores.Reshape(1, scores.Size(0));
```

从[37]中的得分中取出最大的对象对于的序列号

In [ ]:

```
//从每一个[37]中获取最大置信度对于的序列号
Point max;
Cv2.MinMaxLoc(scoresMat.Row(i), out _, out _, out _, out max)
```

对于实际生活中字符的间隔不定问题，在CTC模型中将空的区域当成‘-’
其他的则在标签中寻找对于的字符

In [ ]:

```
 if(max.X >0)
{
    elements.Add(_config.Alphabet[max.X - 1]);
}
else
{
    //对于实际生活中字符的间隔不定问题，在CTC模型中将空的区域当成‘-’
    elements.Add('_');
}
```

CRNN整个处理过程

In [ ]:

```
private string RecognitionProcess(Mat img)
{
    Mat blob = RecognitionPretreatment(img);

    _model.RecognitionNet.SetInput(blob);
    var prob = _model.RecognitionNet.Forward();

    return Postprocess(prob);
}
```

模型持久化变量

在这次中，为了加速检测，节省检测时间，使用变量持久化

对象：在检测中涉及到需要较大内存的变量
目的：将这些变量的从检测流程中提取出，移动到初始化中，
将整个流程分为初始化与检测两个部分 一次初始化，多次使用，加速检测部分.

In [ ]:

```
public class ModelKeepValue
{
    /// <summary>
    /// 定位模型
    /// </summary>
    public Net DetectoterNet { set; get; }

    /// <summary>
    /// 检测头对应的输出层名字
    /// </summary>
    public string[] OutputBlobNames { set; get; }

    /// <summary>
    /// 模型检测头
    /// </summary>
    public Mat[] OutputBlobs { set; get; }

    /// <summary>
    /// 识别模型
    /// </summary>
    public Net RecognitionNet { set; get; }
}
```