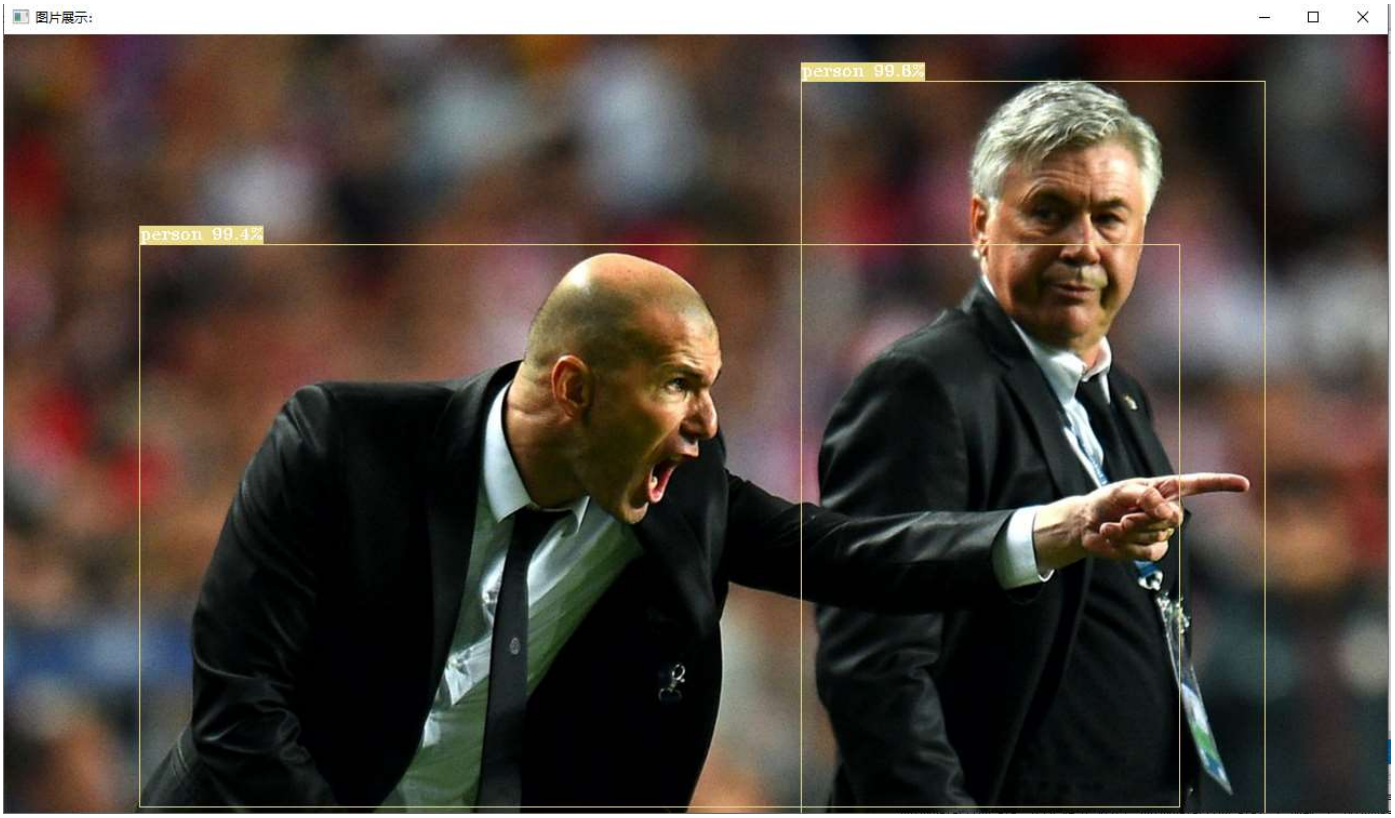
# yolo模型在c#上的调用模块

make by haifengye

## 使用示例

In [ ]:

```
private void Demo()
        {
            string dir              = System.IO.Directory.GetCurrentDirectory();
            string path             = dir + "\\model\\";
            string modelWeights     = System.IO.Path.Combine(path, "yolov3.weights");
            string modelConfiguaration = System.IO.Path.Combine(path, "yolov3.cfg");
            string labelFile        = System.IO.Path.Combine(path, "coco.names");
            string imgpath          = System.IO.Path.Combine(path, "zidane.jpg");

            YoloV3 yoloV3 = new YoloV3(modelWeights, modelConfiguaration, labelFile, 320, 320, 0.5f)
            NetResult[] netResults  = yoloV3.Detect(imgpath);

            foreach (var result in netResults)
            {
                Console.WriteLine("类别，置信度，方框坐标[左边、顶点、长、宽]");
                Console.WriteLine(result.Label + "  " + result.Probability.ToString() + " [" +
                    result.Rectangle.Left.ToString() + " " + result.Rectangle.Top.ToString() + " " +
                    result.Rectangle.Width.ToString() + " " + result.Rectangle.Height.ToString() + "
            }
        }
```



```
类别，置信度，方框坐标[左边、顶点、长、宽]
person  0.998163282871246 [737 43 429 690]
类别，置信度，方框坐标[左边、顶点、长、宽]
person  0.993507623672485 [125 194 961 519]
```

YoloV3 yoloV3 = new YoloV3(modelWeights, modelConfiguaration, labelFile, 320, 320, 0.5f);

其中，modelWeights为yolo模型权重路径，modelConfiguaration为模型参数路径，labelFile为模型的标签文件路径，imgpath为要检测的图片路径，

320，320是yolo模型对输入图像大小的转换要求，0.5f是你需要设置的最低置信度阈值

结果保存在NetResult格式中，参考示例，其提供检测的基本信息：类别，置信度，方框坐标[左边、顶点、长、宽]

In [ ]:

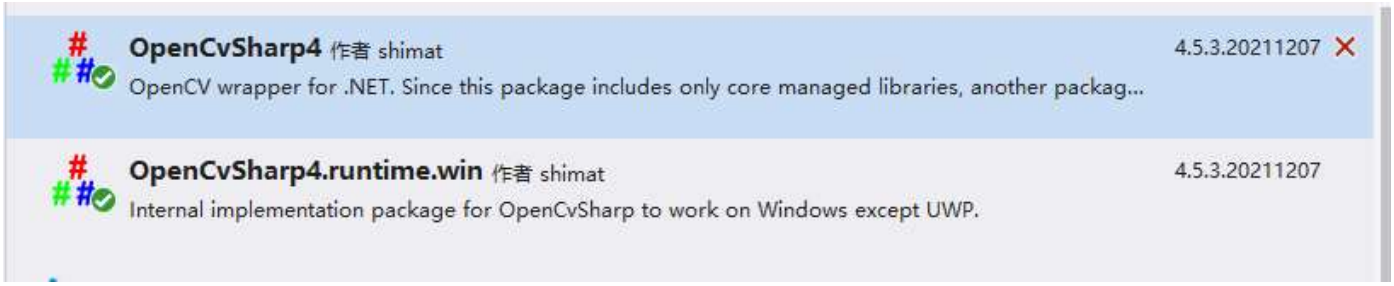
```
1、string Label
2、double Probability
3、Rectangle Rectangle
```

In [ ]:

```
除了使用图像路径的方式进行检测
NetResult[] netResults  = yoloV3.Detect(imgpath);
还可以使用图像本身进行检测
NetResult[] netResults  = yoloV3.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OpenCvSharp4、OpenCvSharp4.runtime.win两个包


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。

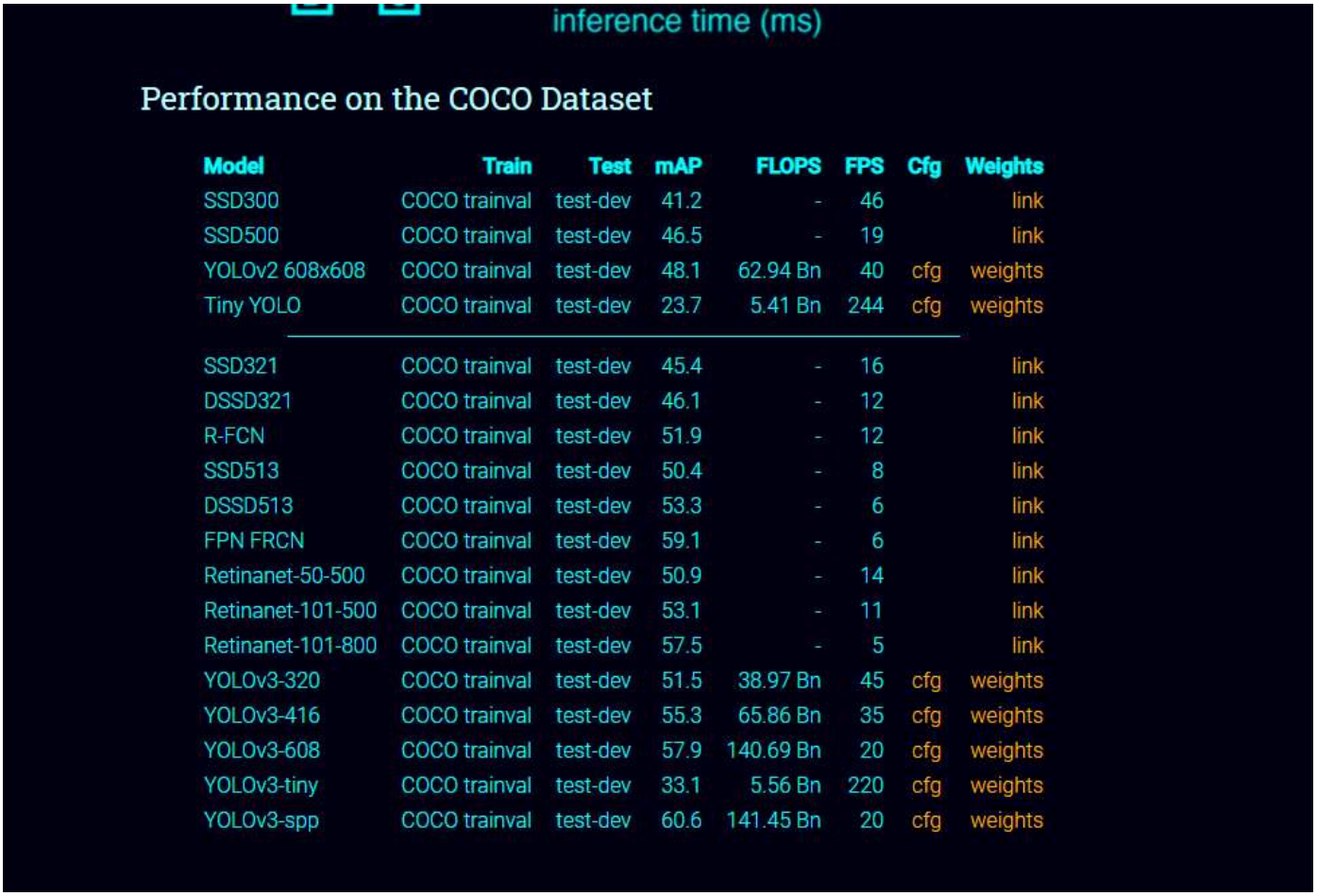
# 模块参数介绍

## YoloV3Config参数

In [ ]:

```
    /// <summary>
    /// 参数
    /// </summary>
    public class YoloV3Config
    {
        /// <summary>
        /// 模型权重文件路径
        /// </summary>
        public string ModelWeights { set; get; }


        /// <summary>
        /// 模型权重参数文件路径
        /// </summary>
        public string ModelConfiguaration { set; get; }

        /// <summary>
        /// 标签文件路径
        /// </summary>
        public string LabelsFile { set; get; }

        /// <summary>
        /// 标签
        /// </summary>
        public string[] Labels { set; get; }

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


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

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

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

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

        /// <summary>
        /// 画图的颜色
        /// </summary>
        public Scalar[] Colors;
    }
```

其中ModelWeights、ModelConfiguaration即为模型权重与模型参数的路径，
可以在yolo的官方网站 https://pjreddie.com/darknet/yolo/ (https://pjreddie.com/darknet/yolo/)
上下载模型权重与模型参数。



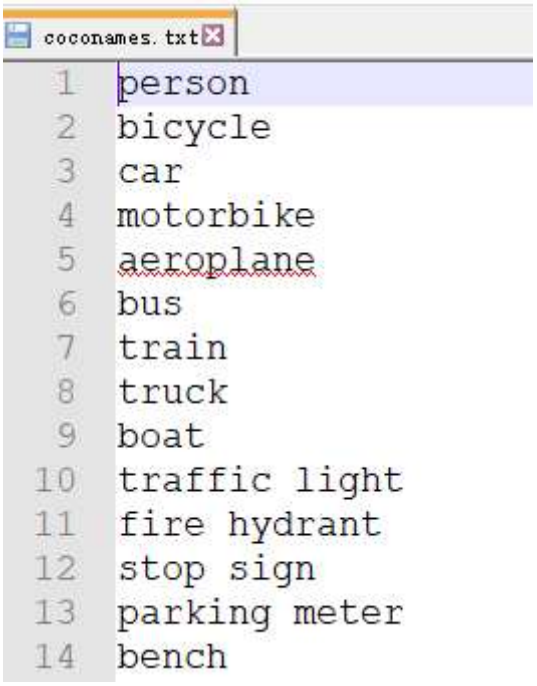

当然我们不是为了直接使用其提供的模型，而是根据需求，修改并训练出满足自己需求的模型，但训练的模型与下载网站的格式是一致的，接下来我们使用网站模型进行检测。

我们在在程序debug中存储的是YOLOV3-320模型,对应与文件中yolov3.weights、yolov3.cfg。

Threshold是最小置信度阈值

NmsThreshold是yolo3特有的nms 阈值，一般不需要设置

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




ImgWidth与ImgHight为yolo模型对图片要求的大小，YOLOV3-320中-320即使图像大小的标注。YOLOV3-410对应大小为416×416、YOLOV3-608对应大小为608×608，其他模型一般在416×416，608×608

## NetResult 结果存储格式

In [ ]:

```
public class NetResult
{
    /// <summary>
    /// Bounding Box  方框
    /// </summary>
    public Rectangle Rectangle { get; set; }

    /// <summary>
    /// 置信度
    /// </summary>
    public double Probability { get; set; }

    /// <summary>
    /// 类别
    /// </summary>
    public string Label { get; set; }

     /// <summary>
    /// 添加
    /// </summary>
    /// <param name="left"> 方框的左边 </param>
    /// <param name="top">方框的顶点</param>
    /// <param name="width">方框的宽度</param>
    /// <param name="height">方框的高度</param>
    /// <param name="label">对应的类别</param>
    /// <param name="probability">置信度</param>
    /// <returns></returns>
    internal static NetResult Add(int left, int top, int width, int height, string label, double
    {
        return new NetResult()
        {
            Label = label,
            Probability = probability,
            Rectangle = new Rectangle(left, top, width, height)
        };
    }
}
```

# 模块介绍

## 对接接口

In [ ]:

```
        /// <summary>
        /// 初始化
        /// </summary>
        /// <param name="pathModel"> 权重路径</param>
        /// <param name="pathConfig">权重参数路径</param>
        /// <param name="labelsFile">标签路径</param>
        /// <param name="imgWidth">yolo模型要求的的图片大小</param>
        /// <param name="imgHigh">yolo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 YoloV3(string pathModel, string pathConfig, 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 NetResult[] Detect(string imgpath)
```

In [ ]:

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

## 内部方法实现

In [ ]:

```
        /// <summary>
        /// 图片预处理
        /// </summary>
        /// <param Mat img</param>
        /// <returns></returns>
        private Mat ImagePretreatment(Mat img)

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

        /// <summary>
        /// 初始化模型
        /// </summary>
        private Net InitializeModel(string pathModel, string pathConfig)

        /// <summary>
        /// 检测的后处理
        /// </summary>
        /// <param name="image"></param>
        /// <param name="results"></param>
        /// <returns></returns>
        private NetResult[] Postprocess(ref Mat image, Mat[] results)

        /// <summary>
        /// 将结果在图像上画出
        /// </summary>
        /// <param name="image"></param>
        /// <param name="classes"> 在标签中的序号</param>
        /// <param name="confidence">置信度</param>
        /// <param name="left">对象框左边距离</param>
        /// <param name="top">对象框顶边距离</param>
        /// <param name="width">对象框宽度</param>
        /// <param name="height">对象框高度</param>
        private void Draw(ref Mat image, int classes, float confidence, double left, double top, dou

        /// <summary>
        /// yolo整个处理过程
        /// </summary>
        /// <param name="img"></param>
        /// <returns></returns>
        private NetResult[] Process(Mat img)
```

# 原理介绍

### 图片预处理

1、yolo模型的输入图像中的像素大小为0~~1，所以需要对0~~255的图像进行转化
2、yolo模型一般对图像输入大小由格式要求，一般为320×320、416×416，608×608

In [ ]:

```
        private Mat ImagePretreatment(Mat img)
        {
            double scale = 1.0 / 255;      //像素大小由0~255范围变为0~1，深度学习中的输入都是0~1范围
            Size size = new Size(_config.ImgWidth, _config.ImgHight);

            //yolo的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
            //生成blob, 块尺寸可以是320/416/608
            Mat blob = CvDnn.BlobFromImage(img, scale, size, new 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

```
['person', 'bicycle', 'car', 'motorbike', 'aeroplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking
```

### 初始化模型

使用opencvsharp中dnn模型的ReadNetFromDarknet函数

In [ ]:

```
Net ReadNetFromDarknet([NullableAttribute(1)] string cfgFile, string darknetModel = null);
```

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

YOLO3 COCO 模型输出output的格式：矩阵形式其中行代表检测出可能的对象其中列为每一个对象的信息，共

一个85维[x1,x2,x3,....x85]形式列的各个维的信息为：
0 1 : center 对象框的中心点
2 3 : w/h 对象框的宽与高
4 : confidence 对象框的置信度
5 ~ 84 : class probability 每一类的置信度

```
对象矩阵形式： (300, 85)
对象矩阵形式： (1200, 85)
```

```
85维： [9.7254866e-01 9.6308517e-01 4.6136118e-02 1.1547550e-01 8.5952079e-06
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
```

In [ ]:

```
foreach (var item in results) 是为了取出可能的对象
```

In [ ]:

```
var confidence = item.At<float>(i, 4);   是为了取出第四维：置信度
```

In [ ]:

```
if (confidence > _config.Threshold)  目的是为了与最低置信度最比较，是否满足我们的要求。 其中 _con
```

In [ ]:

```
double maxVal, minVal;
Point min, max;
Cv2.MinMaxLoc(item.Row(i).ColRange(prefix, item.Cols),    //取出5 ~ 84维
    out minVal, out maxVal, out min, out max);           //求5 ~ 84维中最大的置信度的那个 返回max.

是为了取出5~85维度的数据，并比较哪一个类的置信度值最大，取出来
```

In [ ]:

```
var classes = max.X;
var probability = item.At<float>(i, classes + prefix);     //取出max.X对应的置信度
取出置信度值最大对应类的序列号及对应的置信度
```

In [ ]:

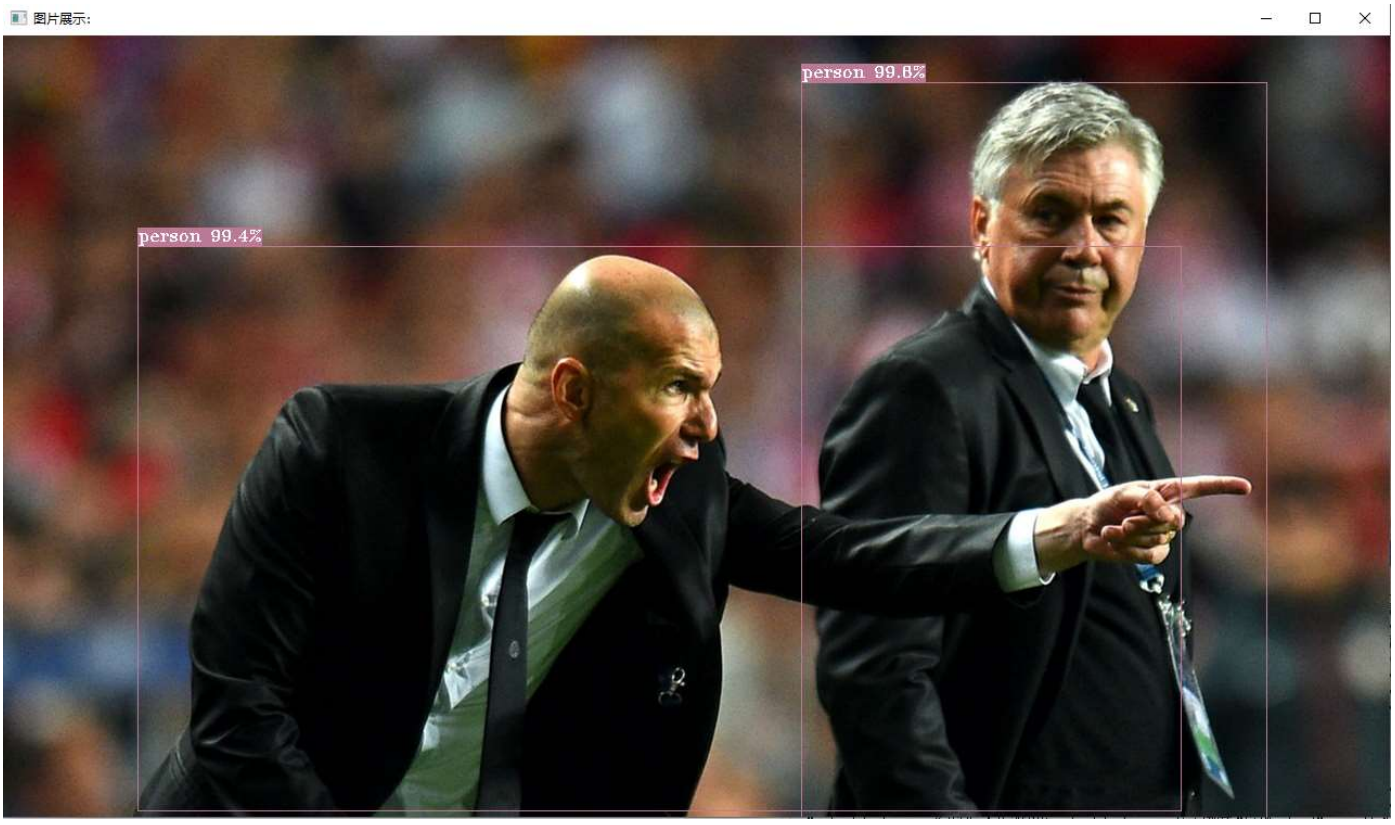
```
var centerX     = item.At<float>(i, 0) * w;
var centerY     = item.At<float>(i, 1) * h;
var width       = item.At<float>(i, 2) * w;
var height      = item.At<float>(i, 3) * h;
取出对应的对象框的中心点与宽高。同时yolo中这些参数仅是相对于输入图片的比例，所以需要乘以相应的宽高进
```

In [ ]:

```
//nms(非极大值抑制)提取分数最高的
//去除重叠和低置信度的目标框
CvDnn.NMSBoxes(boxes, confidences, _config.Threshold, _config.NmsThreshold, out int[] indices);
```

非极大值抑制前


非极大值抑制后:




In [ ]:

```
string label        = _config.Labels[classIds[i]];
double probability  = (double)confidences[i];
int left            = (int)box.X;
int top             = (int)box.Y;
int width           = (int)box.Width;
int height          = (int)box.Height;
netResults.Add(NetResult.Add(left, top, width, height, label, probability));
```

为了将结果存储起来，存到NetResult格式内，当然你也可以定义自己的存储格式。

## 完整代码

In [ ]:

```
private NetResult[] Postprocess(ref Mat image, Mat[] results)
        {
            var netResults = new List<NetResult>();          //存储检测结果

            var classIds = new List<int>();                  //可能对象在标签中对应的序列号集合
            var confidences = new List<float>();             //可能对象的置信度集合
            var boxes = new List<Rect2d>();                  //可能对象的方框集合

            var w = image.Width;                             //图像的宽高
            var h = image.Height;
            /*
             YOLO3 COCO 模型输出output的格式：矩阵形式
             其中行代表检测出可能的对象
             其中列为每一个对象的信息，共一个85维[x1,x2,x3,....x85]形式
             列的各个维的信息为：
             0 1 : center 对象框的中心点                    2 3 : w/h  对象框的宽与高
             4 : confidence  对象框的置信度                 5 ~ 84 : class probability  每一类的置信
            */
            const int prefix = 5;                            //分类概率

            foreach (var item in results)
            {
                for (var i = 0; i < item.Rows; i++)          //取出每一个可能的对象
                {

                    var confidence = item.At<float>(i, 4);                          //第四维：置信度
                    if (confidence > _config.Threshold)
                    {
                        double maxVal, minVal;
                        Point min, max;
                        Cv2.MinMaxLoc(item.Row(i).ColRange(prefix, item.Cols),     //取出5 ~ 84维
                            out minVal, out maxVal, out min, out max);             //求5 ~ 84维中最大

                        var classes = max.X;
                        var probability = item.At<float>(i, classes + prefix);     //取出max.X对应的

                        if (probability > _config.Threshold)                        //more accuracy,
                        {
                            //x,y,width,height 都是相对于输入图片的比例，所以需要乘以相应的宽高进行
                            var centerX     = item.At<float>(i, 0) * w;
                            var centerY     = item.At<float>(i, 1) * h;
                            var width       = item.At<float>(i, 2) * w;
                            var height      = item.At<float>(i, 3) * h;
                            var left        = centerX - width / 2;
                            var top         = centerY - height / 2;

                            //准备nms(非极大值抑制)数据
                            classIds.Add(classes);
                            confidences.Add(confidence);
                            boxes.Add(new Rect2d(left, top, width, height));
                        }
                    }
                }
            }

            //nms(非极大值抑制)提取分数最高的
            //去除重叠和低置信度的目标框
            CvDnn.NMSBoxes(boxes, confidences, _config.Threshold, _config.NmsThreshold, out int[] in
```

```
            foreach (var i in indices)
            {
                //画出目标方框并标注置信度和分类标签
                var box = boxes[i];

                //Build NetResult
                string label        = _config.Labels[classIds[i]];
                double probability  = (double)confidences[i];
                int left            = (int)box.X;
                int top             = (int)box.Y;
                int width           = (int)box.Width;
                int height          = (int)box.Height;
                netResults.Add(NetResult.Add(left, top, width, height, label, probability));

                if (true == _config.IsDraw)
                {
                    Draw(ref image, classIds[i], confidences[i], box.X, box.Y, box.Width, box.Height
                }

            }

            return netResults.ToArray();        // 返回检测结果
        }
```

**将结果在图像上画出**

由于我们需要的信息在上面已经完成，这一步不是必要的，提供一个画图示例。

In [ ]:

```
private void Draw(ref Mat image, int classes, float confidence, double left, double top, double wid
        {
            //标签字符串
            var label = string.Format("{0} {1:0.0}%", _config.Labels[classes], confidence * 100);
            //画方框
            Cv2.Rectangle(image, new Point(left, top), new Point(left + width, top + height), _confi

            //标签字符大小
            var textSize = Cv2.GetTextSize(label, HersheyFonts.HersheyTriplex, 0.5, 1, out var basel

            //画标签背景框
            var x1 = left < 0 ? 0 : left;
            Cv2.Rectangle(image, new Rect(new Point(x1, top - textSize.Height - baseline),
                new Size(textSize.Width, textSize.Height + baseline)), _config.Colors[classes], Cv2.
            Cv2.PutText(image, label, new Point(x1, top - baseline), HersheyFonts.HersheyTriplex, 0.

            Cv2.ImShow("图片展示：", image);
        }
```

In [ ]:

```
画方框python：
cv2.rectangle(frame, (left, top), (left + width, top + height), (0, 0, 255))
```

**yolo整个处理过程**

In [ ]:

```
private NetResult[] Process(Mat img)
        {
            Mat blob = ImagePretreatment(img);

            Net net = InitializeModel(_config.ModelWeights, _config.ModelConfiguaration);
            _config.Labels = ReadLabels(_config.LabelsFile);


            net.SetInput(blob);                                          // 输入数据

            var outNames = net.GetUnconnectedOutLayersNames();     //获得输出层名

            var outs = outNames.Select(_ => new Mat()).ToArray(); //转换成 Mat[]或者List<Mat> outpu

            net.Forward(outs, outNames);                               //Execute all out layers

            NetResult[] netResults = Postprocess(ref img, outs);


            return netResults;
        }
```

其中关键语句是 net.Forward(outs, outNames)，用于图像的推理工作。

In [ ]:

```
对于
var outNames = net.GetUnconnectedOutLayersNames();
ar outs = outNames.Select(_ => new Mat()).ToArray();
是为了获得 Mat[]，用于保存推理结果
```