哈爾濱工業大學

人工智能软件开发与实践

大实验报告

题　　目	基于 YOLOv7 的目标检测
学　　院	计算机科学与技术
专　　业	人工智能
学　　生	张智雄 韩晨烨 祝文鑫 门春廷
任 课 教 师	武小荷

哈尔滨工业大学计算机科学与技术学院

2023.9

大实验：基于 YOLOv7 的目标检测

1、实验内容或者文献情况介绍

1.1 研究现状

目标检测任务是找出图像或视频中人们感兴趣的物体，并同时检测出它们的位置和大小。不同于图像分类任务，目标检测不仅要解决分类问题，还要解决定位问题，是属于 Multi-Task 的问题。

目标检测的发展脉络可以划分为两个周期：传统目标检测算法时期(1998 年-2014 年)和基于深度学习的目标检测算法时期(2014 年至今)。而基于深度学习的目标检测算法又发展成了两条技术路线：Anchor based 方法(一阶段，二阶段)和 Anchor free 方法。而本文使用的 YOLOv7 算法就是一种基于 Anchor 的方法。

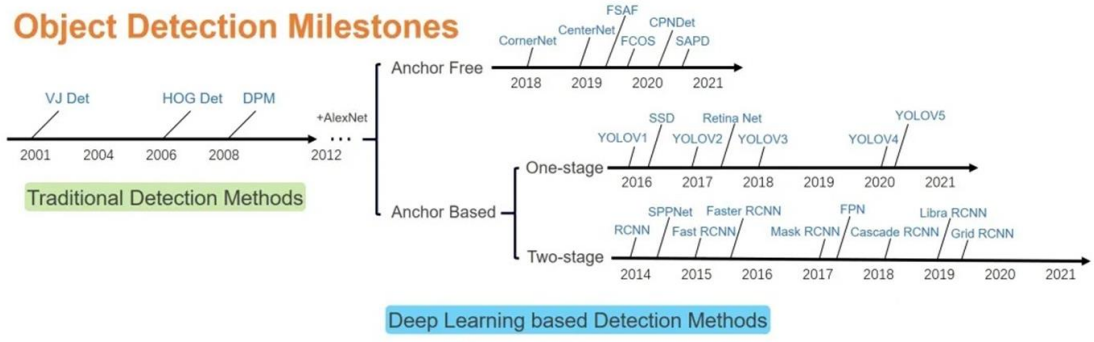

图 1 目标检测的发展

作为计算机视觉的基本问题之一，目标检测构成了许多其它视觉任务的基础，例如实例分割，图像标注和目标跟踪等等；从检测应用的角度看：行人检测、面部检测、文本检测、交通标注与红绿灯检测，遥感目标检测统称为目标检测的五大应用。

1.2 实验内容

本小组拟基于 YOLOv7 实现对标准 COCO 数据集的检测以及特定数据集图像中特定目标的检测，实验内容主要分三步：

第一步，利用 YOLOv7 模型对 COCO128 数据集进行目标检测。

第二步，将 YOLOv7 模型用于自建数据集进行目标检测。

第三步，改进 YOLOv7 模型，添加注意力集中机制。

2、算法简介及其实现细节

2.1 YOLOv7 模型

经典的 YOLOv7 的整体模型如下图 2 所示，首先模型将输入的图片调整为

640×640大小，然后将其输入到 backbone 网络中，之后经 head 层网络输出三层不同尺寸的特征图像，通过 Rep 和 conv 层后输出预测结果。

以 COCO 数据集为例，输出为80个类别，然后每个输出为(x, y, w, h, o)，即坐标位置和前后背景，3指的是 anchor 数量，因此每一层的输出为$(80 + 5) \times 3 = 255$，再乘上特征图像的大小即为最终的输出。

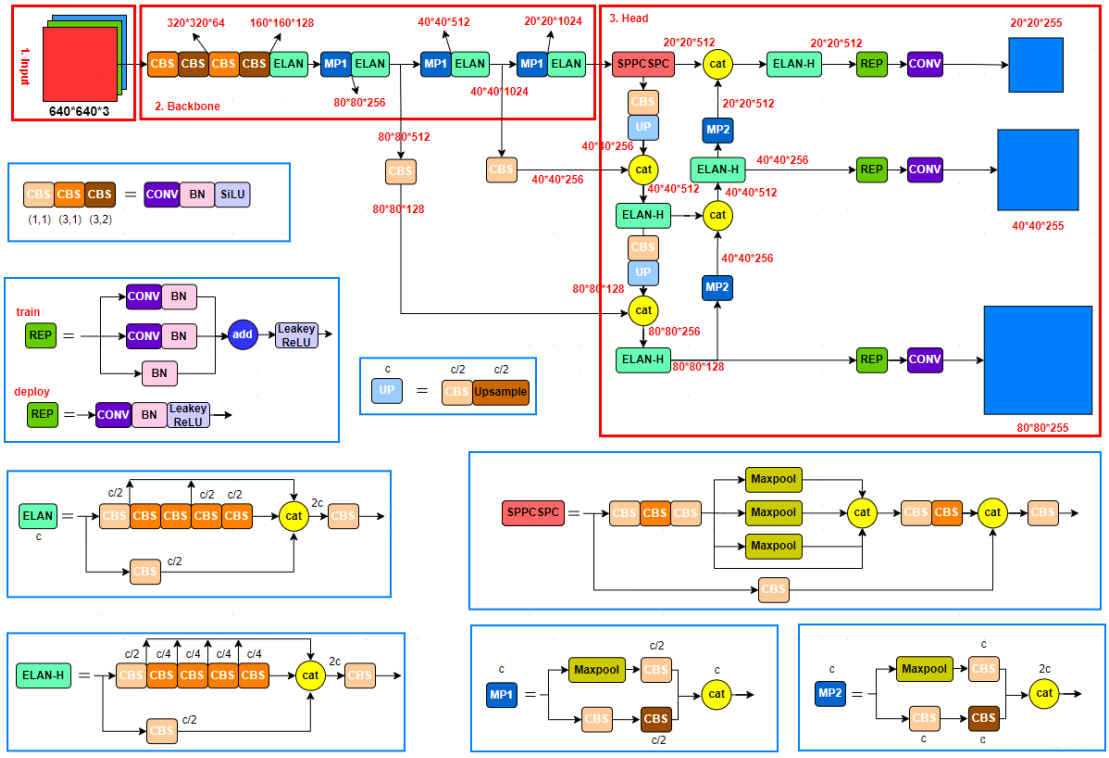

图 2 YOLOv7 模型整体框架

2.1.1 Backbone 模块

YOLOv7 的 Backbone 模块共有 50 层，图中标注了关键的几层。首先是经过 4 层卷积层，CBS 层主要由卷积层、批归一化层和激活函数 SiLU 构成。经过 4 个 CBS 层后，特征图变为$160 \times 160 \times 128$大小。随后会经过 ELAN 模块，ELAN 由多个 CBS 构成，其输入输出特征大小保持不变，通道数在开始的两个 CBS 会有变化，后面的几个输入通道都是和输出通道保持一致的，并经过最后一个 CBS 输出为需要的通道。

2.1.2 Head 模块

Head 模块就是一个 pafpn（PANet+FPN）的结构，和之前的 YOLOv4，YOLOv5 一样，区别在于将 YOLOv5 中的 CSP 模块换成了 ELAN-H 模块，同时下采样变成 MP2 层。

受计算资源的限制，我们所采用的模型是基于 YOLOv7-tiny 模型，它采用了更紧凑的网络架构和优化的训练策略，实现了模型的轻量化的同时保持了良好的

目标检测性能。它与 YOLOv7 结构基本相似，在 backbone 模块中减少了两个卷积层的堆叠，同时将激活函数由 SiLU 改为计算更快的 LeakyReLU 函数。

2.2 基于注意力机制的 YOLOv7 模型

随着深度学习的发展，越来越多的网络拓扑被用于目标检测任务，当我们将 CNN 扩展到多个卷积层时，它表现出显著的增强学习特征表示的能力。然而，它会导致堆栈更多的深度卷积副本，并需要消耗大量计算资源。作为一种替代方法，注意力机制的方法能够合理利用有限的视觉信息处理资源，有利于学习更具判别性的特征表示，而且可以插入 CNN 结构中，能够一定程度上在简化模型的基础上提高模型的性能。

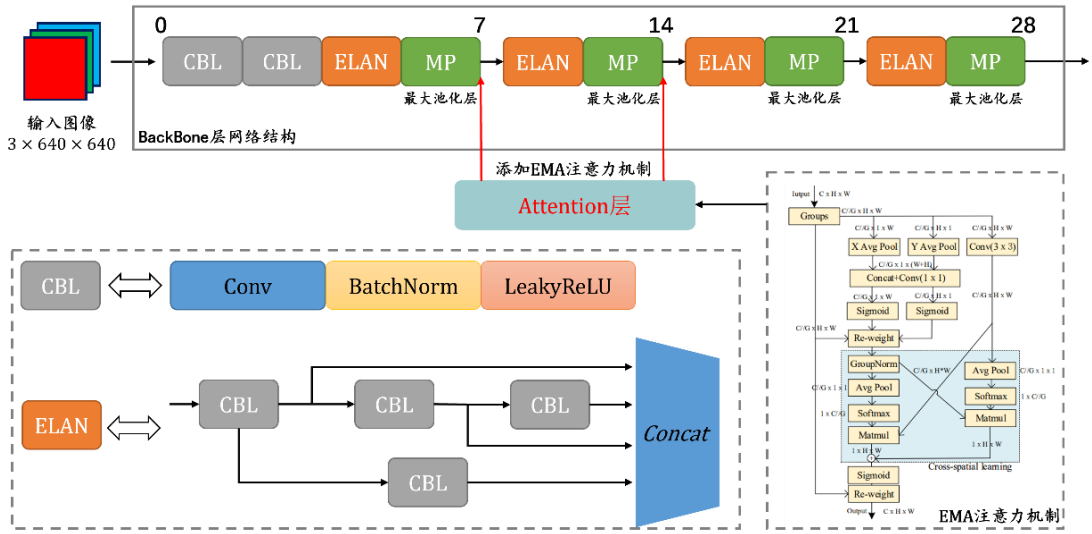

图 3 Efficient Multi-Scale Attention Module(EMA)模块框架

Efficient Multi-Scale Attention Module(EMA)采用以下方式实现多层次、多尺度的注意力学习感知：

1) **并行子结构**：可以帮助网络避免更多的顺序处理，为了聚合多尺度空间结构信息，将一个3×3分支和1×1分支平行放置以进行快速响应。对于任意给定的输入特征图，EMA 将其分为G个子特征跨通道进行学习，并利用注意力权重描述符来加强每个子特征中值得注意的区域的特征表示。

2) **三条平行路径提取分组特征的注意力权重描述符**：其中两条路为1×1分支，剩余一条为3×3分支。一方面，EMA 在1×1分支中分别沿两个空间方向编码特征，最终输出分解为两个向量后，采用两个非线性 Sigmoid 函数拟合线性卷积的二位二项分布。另一方面，3×3分支通过卷积扩大特征空间。这样，EMA 不仅对信道间信息进行编码以调整不同信道的重要性，而且精准的将空间结构信息保留在信道中。

3) **跨空间学习**：在跨空间学习中我们仍然引入两个张量，一个为 1x1 分支输

出，另一个为3×3分支输出。然后利用$z_\epsilon = \frac{1}{H*W}\sum_j^H \sum_i^W x_\epsilon(i,j)$二维全局平均池化对$1 \times 1$输出进行编码，从而实现对全局信息进行编码并对长依赖关系进行建模。

通过上述处理，得出第一个空间注意力分布图。此外，利用二维全局平均池化编码3×3分支的全局信息，1×1分支在此之前直接转化为对应维度，通过通道特征的机理，就得到第二张保留空间位置信息的空间注意力图。最后将每个组内的输出特种映射为两个空间注意权重值的集合，然后使用 Sigmoid 函数，捕获像素级成对关系，并突出图像的全局上下文。

让 YOLOv7 使用 EMA 作为并行子结构来聚合多尺度的空间结构信息，通过注意力权重描述符来提升每个子特征中重要区域的表示能力，相比于传统的 YOLOv7 模型，它可以在特定的数据集上更好地学习到目标对象的重要特征，从而提升检测性能和准确度。

3、实验设置及结果分析（包括实验数据集）

3.1 实验数据集及数据预处理

3.1.1 MS COCO 数据集

MS COCO(Microsoft Common Objects in Context)是一个大型的、丰富的物体检测，分割和字幕数据集。这个数据集以场景理解为目标，主要从复杂的日常场景中截取，图像中的目标通过精确的分割进行位置的标定，包含 80 类物体目标，328,000 影像和 2,500,000 个标签。

3.1.2 草莓数据集

根据实际采集的草莓图像使用 Labelimg 工具进行框选标注构建包含 3 类目标，约 1500 个标签的数据集。

3.2 模型设计

在本次实验中，仿照 YOLOv7-tiny 的模型结构，结合 github 上代码实现了通用和私有数据集上的目标检测，网络的定义是重写 nn.Module 实现的。

3.2.1 YOLOv7-tiny 模型结构

YOLOv7-tiny 模型结构中的标准卷积层包含一个卷积层和一个批归一化层并使用了 SiLU 作为激活函数。并包含了拼接层和最大池化层。

```python
class Conv(nn.Module):
    # Standard convolution
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  # ch_in, ch_out, kernel, stride, padding, groups
        super(Conv, self).__init__()
        self.conv = nn.Conv2d(c1, c2, k, s, autopad(k, p), groups=g, bias=False)
        self.bn = nn.BatchNorm2d(c2)
        self.act = nn.SiLU() if act is True else (act if isinstance(act, nn.Module) else nn.Identity())

    def forward(self, x):
        return self.act(self.bn(self.conv(x)))

    def fuseforward(self, x):
        return self.act(self.conv(x))
```

图 4 卷积 Conv 模块

并包含了拼接层用于将多个输入张量在指定维度上进行连接或拼接，形成一个更大的张量。

```python
class Concat(nn.Module):
    def __init__(self, dimension=1):
        super(Concat, self).__init__()
        self.d = dimension

    def forward(self, x):
        return torch.cat(x, self.d)
```

图 5 拼接 Concat 模块

以及最大池化层进行下采样用于特征提取，特征降维等操作。

```python
class MP(nn.Module):
    def __init__(self, k=2):
        super(MP, self).__init__()
        self.m = nn.MaxPool2d(kernel_size=k, stride=k)

    def forward(self, x):
        return self.m(x)


class SP(nn.Module):
    def __init__(self, k=3, s=1):
        super(SP, self).__init__()
        self.m = nn.MaxPool2d(kernel_size=k, stride=s, padding=k // 2)

    def forward(self, x):
        return self.m(x)
```

图 6 池化 MaxPooling 模块

3.2.2 EMA 注意力层结构

EMA 注意力层结构首先，将输入张量 x 重塑成[b*factor,c//factor,h,w]的形状。接着，通过自适应平均池化操作 pool_h 和 pool_w，将输入张量分别池化为高度和宽度的均值。将池化后的结果拼接在一起，并通过1×1卷积 conv1x1 进行处理。将处理后的结果分割为两部分，其中一部分用于计算权重 x1，另一部分用于计算 x2。对 x1 和 x2 进行 Softmax 操作，得到权重。

```python
class EMA(nn.Module):
    def __init__(self, channels, factor=8):
        super(EMA, self).__init__()
        self.groups = factor
        assert channels // self.groups > 0
        self.softmax = nn.Softmax(-1)
        self.agp = nn.AdaptiveAvgPool2d((1, 1))
        self.pool_h = nn.AdaptiveAvgPool2d((None, 1))
        self.pool_w = nn.AdaptiveAvgPool2d((1, None))
        self.gn = nn.GroupNorm(channels // self.groups, channels // self.groups)
        self.conv1x1 = nn.Conv2d(channels // self.groups, channels // self.groups, kernel_size=1, stride=1, padding=0)
        self.conv3x3 = nn.Conv2d(channels // self.groups, channels // self.groups, kernel_size=3, stride=1, padding=1)

    def forward(self, x):
        b, c, h, w = x.size()
        group_x = x.reshape(b * self.groups, -1, h, w)  # b*g,c//g,h,w
        x_h = self.pool_h(group_x)
        x_w = self.pool_w(group_x).permute(0, 1, 3, 2)
        hw = self.conv1x1(torch.cat([x_h, x_w], dim=2))
        x_h, x_w = torch.split(hw, [h, w], dim=2)
        x1 = self.gn(group_x * x_h.sigmoid() * x_w.permute(0, 1, 3, 2).sigmoid())
        x2 = self.conv3x3(group_x)
        x11 = self.softmax(self.agp(x1).reshape(b * self.groups, -1, 1).permute(0, 2, 1))
        x12 = x2.reshape(b * self.groups, c // self.groups, -1)  # b*g, c//g, hw
        x21 = self.softmax(self.agp(x2).reshape(b * self.groups, -1, 1).permute(0, 2, 1))
        x22 = x1.reshape(b * self.groups, c // self.groups, -1)  # b*g, c//g, hw
        weights = (torch.matmul(x11, x12) + torch.matmul(x21, x22)).reshape(b * self.groups, 1, h, w)
        return (group_x * weights.sigmoid()).reshape(b, c, h, w)
```

图 7 注意力机制 EMA 模块

3.3 在 COCO128 数据集上的目标检测

在本次实验中，分别使用 YOLOv7 和 YOLOv7-tiny 模型在 COCO128 数据集上进行训练，设置训练超参数*epoch*为300，*batch size*为 16，结果对照如下：

表 1 YOLOv7 模型 YOLOv7-tiny 模型对比结果

	YOLOv7	YOLOv7-tiny
Precision 精确率	0.856	0.793
Recall 召回率	0.169	0.094
mAP@.5	0.0544	0.0237

在 YOLOv7 模型运行及绘制图像如下：

```
     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
   293/299     14.9G   0.05047   0.03496   0.04341    0.1288       234       640: 100%|██████████| 8/8 [00:03<00:00,  2.43it/s]
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  2.92it/s]
                  all        128        929      0.715     0.0482     0.0463     0.0206

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
   294/299     14.9G   0.04991   0.03412   0.04263    0.1267       249       640: 100%|██████████| 8/8 [00:03<00:00,  2.31it/s]
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  2.76it/s]
                  all        128        929      0.667     0.0593      0.046     0.0197

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
   295/299     14.9G   0.05012   0.03628   0.04269    0.1291       280       640: 100%|██████████| 8/8 [00:03<00:00,  2.48it/s]
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  3.02it/s]
                  all        128        929      0.663     0.0572     0.0429      0.019

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
   296/299     14.9G   0.04822   0.03607   0.04283    0.1271       253       640: 100%|██████████| 8/8 [00:03<00:00,  2.40it/s]
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  2.79it/s]
                  all        128        929      0.743     0.0417     0.0433      0.019

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
   297/299     14.9G   0.04972   0.03317   0.04289    0.1258       264       640: 100%|██████████| 8/8 [00:03<00:00,  2.52it/s]
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  2.25it/s]
                  all        128        929      0.729     0.0391     0.0321     0.0158

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
   298/299     14.9G    0.0484   0.03639   0.04241    0.1272       206       640: 100%|██████████| 8/8 [00:03<00:00,  2.39it/s]
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  2.22it/s]
                  all        128        929      0.708     0.0418     0.0309     0.0153

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
   299/299     14.9G   0.04734   0.03608   0.04381    0.1272       215       640: 100%|██████████| 8/8 [00:03<00:00,  2.31it/s]
                Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%|██████████| 4/4 [00:02<00:00,  1.43it/s]
                  all        128        929      0.686     0.0454     0.0362      0.017
300 epochs completed in 0.511 hours.

Optimizer stripped from runs/train/yolov7t8/weights/last.pt, 75.7MB
Optimizer stripped from runs/train/yolov7t8/weights/best.pt, 75.7MB
```

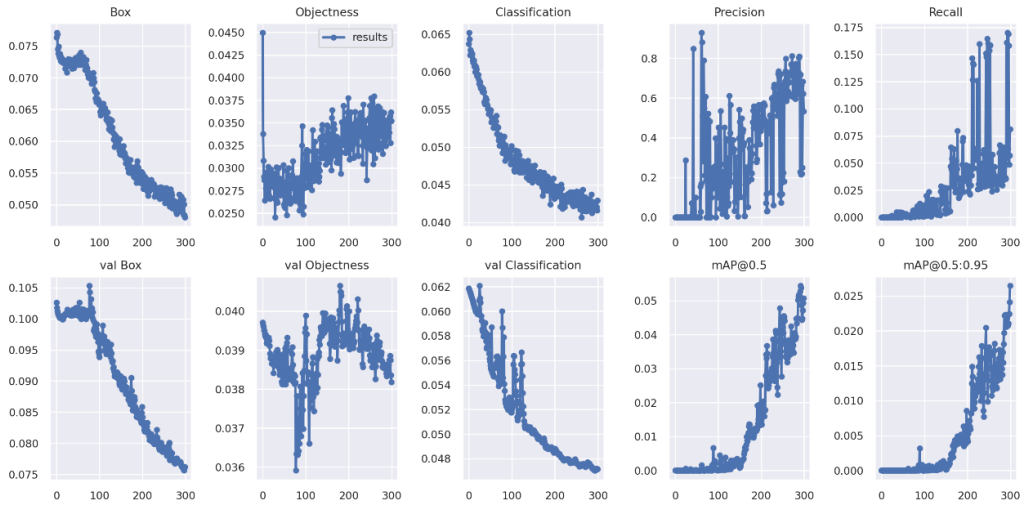

图 8 YOLOv7 模型 COCO 测试结果

在 YOLOv7-tiny 模型运行及绘制图像如下：

```
     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
293/299     6.78G   0.05463   0.03204   0.03951    0.1262       152       640: 100%|██████████| 8/8 [00:02<00:00,  3.75it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  3.02it/s]
                 all         128         929       0.687       0.059      0.0654      0.0247

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
294/299     6.78G   0.05528   0.03332   0.03895    0.1275       183       640: 100%|██████████| 8/8 [00:01<00:00,  4.08it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  3.03it/s]
                 all         128         929       0.595      0.0779      0.0677      0.0267

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
295/299     6.78G   0.05416   0.03324   0.04154    0.1289       251       640: 100%|██████████| 8/8 [00:02<00:00,  3.86it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  3.01it/s]
                 all         128         929       0.616      0.0728       0.068      0.0262

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
296/299     6.78G   0.05389   0.03212   0.04005    0.1261       219       640: 100%|██████████| 8/8 [00:01<00:00,  4.05it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  3.09it/s]
                 all         128         929       0.625        0.07      0.0618      0.0249

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
297/299     6.78G   0.05512   0.03272   0.04012     0.128       203       640: 100%|██████████| 8/8 [00:02<00:00,  3.92it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  3.00it/s]
                 all         128         929       0.638      0.0632      0.0642      0.0248

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
298/299     6.78G   0.05544   0.03246   0.03967    0.1276       142       640: 100%|██████████| 8/8 [00:02<00:00,  3.66it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 4/4 [00:01<00:00,  2.91it/s]
                 all         128         929       0.656      0.0627       0.066      0.0258

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
299/299     6.78G   0.05499   0.02781   0.03983    0.1226       238       640: 100%|██████████| 8/8 [00:02<00:00,  3.90it/s]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 4/4 [00:02<00:00,  1.45it/s]
                 all         128         929       0.464       0.146      0.0692       0.027
300 epochs completed in 0.302 hours.

Optimizer stripped from runs/train/yolov7-tiny2/weights/last.pt, 12.7MB
Optimizer stripped from runs/train/yolov7-tiny2/weights/best.pt, 12.7MB
```

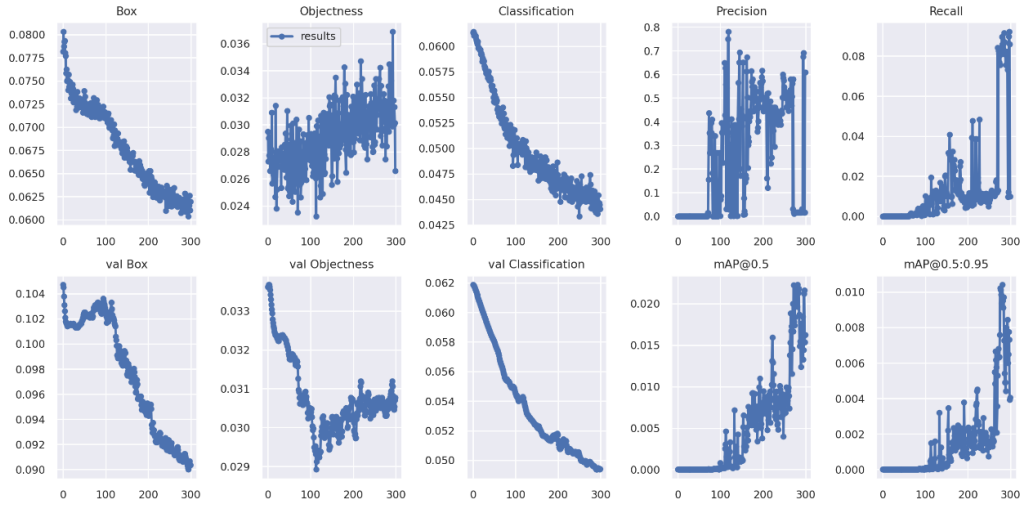

图 9 YOLOv7-tiny 模型 COCO 测试结果

根据上述实验结果可以发现，在相同数据集上 YOLOv7 相较于 YOLOv7-tiny 有较为明显的性能提升，但是受限于数据集检测类别较多、训练数据较小的情况出现训练反复波动的情况。

3.4 对自建数据集上的目标检测

在本次实验中，在自建的草莓成熟度数据集上分别使用加入和不加入 EMA 注意力层的模型进行生成训练，并设置训练超参数*epoch*为300，*batch size*为 16，具体结果如下。

表 2　YOLOv7-tiny 模型有无注意力对比结果

	YOLOv7-tiny(default)	YOLOv7-tiny(with EMA)
Precision 精确率	0.931	0.942
Recall 召回率	0.781	0.799
mAP@.5	0.873	0.899
mAP@.5:.95	0.664	0.666

在默认 YOLOv7-tiny 模型运行及绘制图像如下：

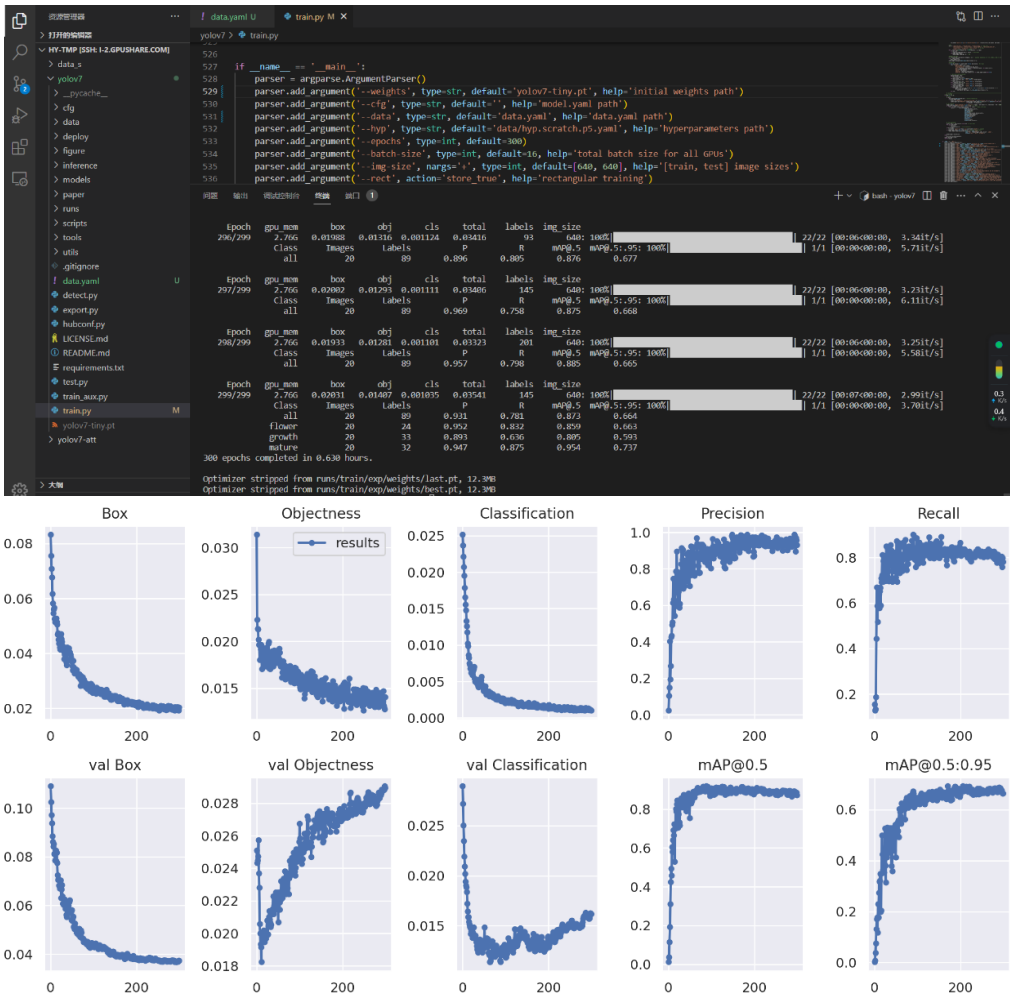

图 10 YOLOv7-tiny 模型无注意力测试结果

在加入 EMA 注意力机制的 YOLOv7-tiny 模型运行及绘制图像如下：

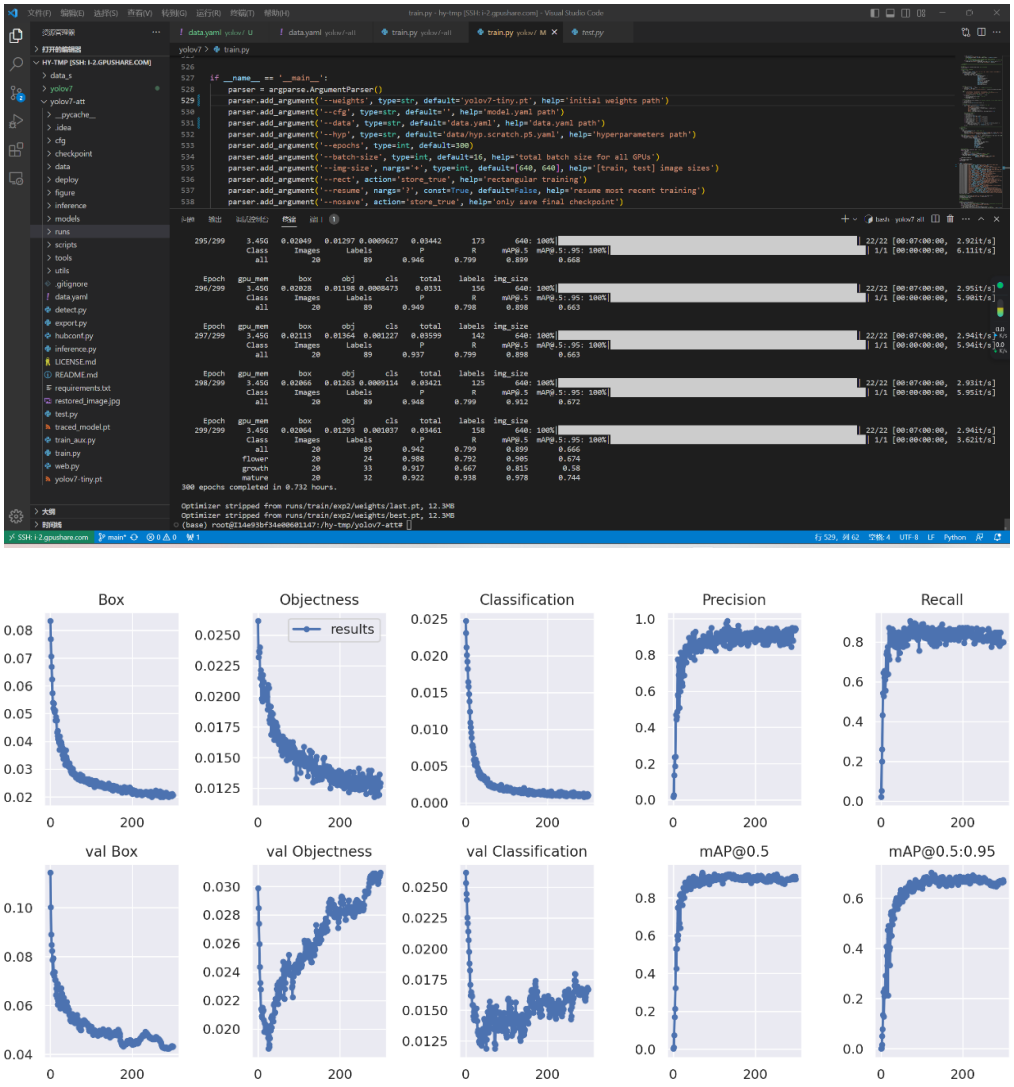

图 11 YOLOv7-tiny 模型加入 EMA 注意力测试结果

分析上述实验结果可以发现，在此草莓数据集上加入注意力机制能够一定程度上提高模型检测的性能，但是具体效果因数据集而定。总体上，此数据集数据量较小，两种模型都能在规定训练过程中实现收敛。

3.5 系统可视化界面设计

项目基于 Python 的 Gradio 库设计实现系统的可视化交互，能够切换三种模型下的目标检测，并提供示例图像与模型组。

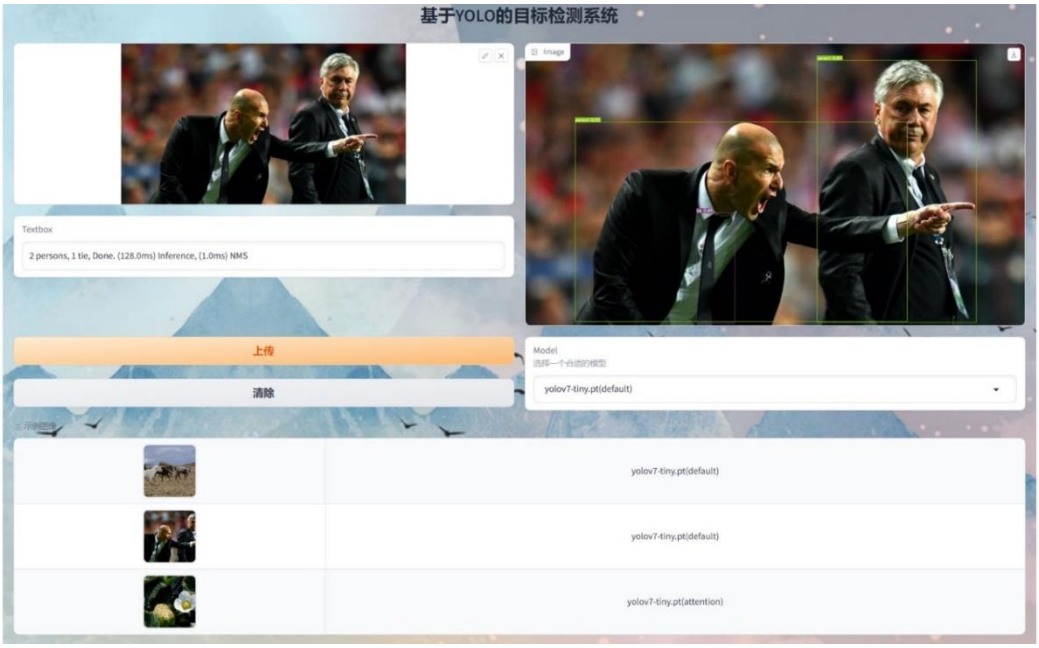

图 12 系统可视化界面

4、总结

YOLOv7 作为深度学习算法的一种，通过将输入图像调整为固定大小，并经过主干网络、头部网络以及注意力机制的处理，实现了对目标的准确检测和识别。

在实验部分，我们使用了 COCO128 数据集和自定义草莓数据集进行了目标检测的实验。在 COCO128 数据集上，我们使用了 YOLOv7 模型进行目标检测，并获得了一定的精度。在自定义草莓数据集上，我们进一步研究了加入注意力机制后的目标检测效果，结果显示注意力机制可以减少其他因素的干扰从而提高目标检测的准确性。

5、组内分工

姓名	分工	组内互评
张智雄	YOLOv7-tiny 模型注意力机制的添加以及两种模型在草莓数据集上的对比测试	5
韩晨烨	YOLOv7 模型和 YOLOv7-tiny 模型的实现以及在 COCO128 数据集上的目标检测	4
祝文鑫	查阅目标检测发展相关资料，基于 Gradio 库实现系统可视化界面的设计	4
门春廷	查阅注意力机制相关资料，基于 Gradio 库实现系统可视化界面的设计	3

参考文献

[1]. Wang, Chien-Yao et al. “YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors.” 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2022): 7464-7475.

[2]. Ouyang, Daliang et al. “Efficient Multi-Scale Attention Module with Cross-Spatial Learning.” ArXiv abs/2305.13563 (2023): n. pag.