YOLOv4 目标检测复现

摘要

使用大量的特征信息作 CNN 卷积神经网络的训练可以提高神经网络预测精确度。这一点需要在大型的数据集上对其进行实际验证测试。其中，有些特征信息只能在特定的模型及问题上产生作用或只能适用于小规模数据集的训练及测试。而一些如批处理和剩余连接的特征可适用于大多数模型、数据集当中。YoLov4 目标检测网络采用了一系列能够帮助提取通用特征的机制：加权残差连接 WRC、跨阶段局部连接 CSP、跨小批量归一化 CmBn、自对抗训练 SAT、激活函数 mish、mosaic 数据增强、DropBlock 正则化和 CIoU 损失，最后实现在单块显卡上的目标检测性能提升。

关键词：CNN 卷积神经网络；目标检测；特征提取

1 引言

选题背景：计算机视觉是一门研究如何使机器“看”的科学，更进一步的说，就是指用摄影机和电脑代替人眼对目标进行识别、跟踪和测量等机器视觉，并进一步做图形处理，经过电脑处理成为更适合人眼观察或传送给仪器检测的图像。作为一个科学学科，计算机视觉研究相关的理论和技术，试图建立能够从图像或者多维数据中获取”“信息”的人工智能系统。其中，目标检测领域是计算机视觉的一大热门的研究领域。目标检测是计算机视觉众多邻域中的一项基础工作，无论是作物体追踪还是关键点检测都需要运用到目标检测技术。目标检测是一种基于目标几何和统计特征的图像分割。它能够将目标的分割和识别合二为一。对一张图片的内容进行目标检测工作，需要进行三个层次工作：分类、检测和分割。

选题依据：深度学习模型的训练往往需要依赖性能较高的多块显卡，而 YOLOv4[1]实现的目标检测模型可在单个 GPU 上进行训练，使得在缺乏显卡硬件情况下的模型训练更加便利。

选题意义：目前大部分的目标检测模型都应用于推荐系统。在推荐系统中应用的目标检测模型往往是低速但精确度较高的，然而在遇到一些对实时性较高的应用场景（比如汽车的碰撞预警）时，则需要快速但精确度适中的目标检测模型。通过提高这类具备实时性的目标检测算法的检测精确度可以进一步实现独立的流程管理及控制，以此减少人工干预的成本开销，使实时性的目标检测不再局限于提供警示信息的推荐系统。然而目前准确率较高的神经网络模型不支持实时任务，并且还需要使用大量 GPU 资源对模型进行训练。而 YOLOv4 首次通过优化网络特征提取结构的方式实现了能够在单个 GPU 上实时运行的 CNN 模型，同时该模型只需一个 GPU 进行训练。YOLOv4 使用单个传统 GPU 进行训练和测试就可以实现实时、高质量和可行的目标检测结果。能够在单个 GPU 上训练模型，这对于实际的生产应用带来了巨大的帮助与增益。通过学习 YOLOv4, 对其模型进行复现，可以帮助我们较好地了解目标检测的总体实现过程。

2 相关工作

常见的运行在 GPU 上的目标检测网络的主干网络有 VGG[2]、ResNet[3]、ResNeXt[4]或 DenseNet[5]。

常见的运行在 CPU 上的目标检测网络的主干网络有在 CPU 平台上运行的检测器，它们的骨干可以是 SqueezeNet[6]、MobileNet[7]或 ShuffleNet[8]。

常见的 head 预测网络，通常分为两种，即一阶段目标检测器和两阶段目标检测器。最具代表性的两阶段目标检测器是 R-CNN[9]系列，包括 fast R-CNN[10]、faster R-CNN[11]。至于单阶段物体检测器，最有代表性的模型是 YOLO[12] 。

3 本文方法

YOLOv4 论文的复现主要根据网络构成来实现。YOLOv4 的目标检测网络由三个网络组成：backbone、neck 和 head 部分。其中 backbone 部分采用的是 CSPDarkNet53 网络来进行输入图像的特征提取，Neck 部分采用的是 SPP 与 PANet 网络的组合，head 部分采用 YOLOv3 的 head 网络.

首先，需要复现的是 Backbone 的特征提取网络，实现对输入图像的特征信息的提取。接着，需要复现的是 Neck 的 SPP 网络。最后，实现的是 head 的预测网络。

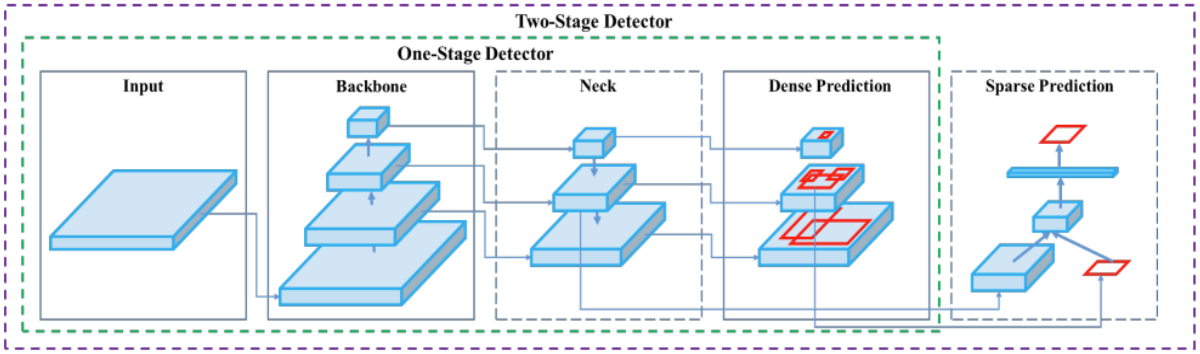

图 1: YOLOv4 目标检测器结构

4 复现细节

4.1 与已有开源代码对比

本文的 YOLOv4 目标检测网络的复现过程参考了部分网上代码。参考的代码包含主干网络设计部分的代码。参考代码在使用过程中未出现问题。

复现过程中参考的网络设计部分的代码：

```
class YoloBody(nn.Module):
    def __init__(self, anchors_mask, num_classes, pretrained=False):
        super(YoloBody, self).__init__()

        self.backbone    = darknet53(pretrained)

        self.conv1       = make_three_conv([512,1024],1024)
        self.SPP         = SpatialPyramidPooling()
        self.conv2       = make_three_conv([512,1024],2048)

        self.upsample1   = Upsample(512,256)
        self.conv_for_P4 = conv2d(512,256,1)
        self.make_five_conv1 = make_five_conv([256, 512],512)

        self.upsample2   = Upsample(256,128)
        self.conv_for_P3 = conv2d(256,128,1)
        self.make_five_conv2 = make_five_conv([128, 256],256)

        # 3*(5+num_classes) = 3*(5+20) = 3*(4+1+20)=75
        self.yolo_head3  = yolo_head([256, len(anchors_mask[0]) * (5 + num_classes)],128)

        self.down_sample1    = conv2d(128,256,3,stride=2)
        self.make_five_conv3 = make_five_conv([256, 512],512)

        # 3*(5+num_classes) = 3*(5+20) = 3*(4+1+20)=75
        self.yolo_head2  = yolo_head([512, len(anchors_mask[1]) * (5 + num_classes)],256)

        self.down_sample2    = conv2d(256,512,3,stride=2)
        self.make_five_conv4 = make_five_conv([512, 1024],1024)

        # 3*(5+num_classes)=3*(5+20)=3*(4+1+20)=75
        self.yolo_head1  = yolo_head([1024, len(anchors_mask[2]) * (5 + num_classes)],512)
```

图 2: yolobody 网络结构

```
class CSPDarkNet(nn.Module):
    def __init__(self, layers):
        super(CSPDarkNet, self).__init__()
        self.inplanes = 32
        # 416,416,3 -> 416,416,32
        self.conv1 = BasicConv(3, self.inplanes, kernel_size=3, stride=1)
        self.feature_channels = [64, 128, 256, 512, 1024]

        self.stages = nn.ModuleList([
            # 416,416,32 -> 208,208,64
            Resblock_body(self.inplanes, self.feature_channels[0], layers[0], first=True),
            # 208,208,64 -> 104,104,128
            Resblock_body(self.feature_channels[0], self.feature_channels[1], layers[1], first=False),
            # 104,104,128 -> 52,52,256
            Resblock_body(self.feature_channels[1], self.feature_channels[2], layers[2], first=False),
            # 52,52,256 -> 26,26,512
            Resblock_body(self.feature_channels[2], self.feature_channels[3], layers[3], first=False),
            # 26,26,512 -> 13,13,1024
            Resblock_body(self.feature_channels[3], self.feature_channels[4], layers[4], first=False)
        ])

        self.num_features = 1
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
```

图 3: CspDarkNet 网络结构

4.2 实验环境搭建

pytorch:1.7.1+cu110

torchvision:0.8.2+cu110

cuda:cuda 11.0

os:windows 11

4.3 主干特征提取网络 cspdarknet53 的构建

4.3.1 激活函数

YOLOv4 采用了 mish 激活函数,mish 是光滑的非单调激活函数，具有较好的泛化能力和结果的有效优化能力，可以提高结果的质量。mish 激活函数公式为：

$$mish = x \times tanh(ln(1 + e^x)) \tag{1}$$

```
class Mish(nn.Module):
    def __init__(self):
        super(Mish, self).__init__()

    def forward(self, x):
        return x * torch.tanh(F.softplus(x))
```

图 4: mish 函数实现

4.3.2 构造网络卷积块

一个卷积块由卷积函数、标准化函数以及激活函数构成。

```
class BasicConv(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride=1):
        super(BasicConv, self).__init__()

        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size, stride, kernel_size//2, bias=False)
        self.bn = nn.BatchNorm2d(out_channels)
        self.activation = Mish()

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.activation(x)
        return x
```

图 5: 网络卷积块实现

4.3.3 构造内部堆叠的残差块

对输入进行 1*1 和 3*3 大小的卷积块的卷积处理后，将输入与卷积处理后的输入进行相加。

```python
class Resblock(nn.Module):
    def __init__(self, channels, hidden_channels=None):
        super(Resblock, self).__init__()

        if hidden_channels is None:
            hidden_channels = channels

        self.block = nn.Sequential(
            BasicConv(channels, hidden_channels, 1),
            BasicConv(hidden_channels, channels, 3)
        )

    def forward(self, x):
        return x + self.block(x)
```

图 6: 堆叠残差块实现

4.3.4 构建 cspnet 的结构块

YOLOv4 对残差块的堆叠进行了拆分，左边部分为一个大的残差边，只进行少量处理。右边部分进行正常的残差块的堆叠。最后。将左边部分的大残差边与右边部分残差块的堆叠结果进行堆叠。

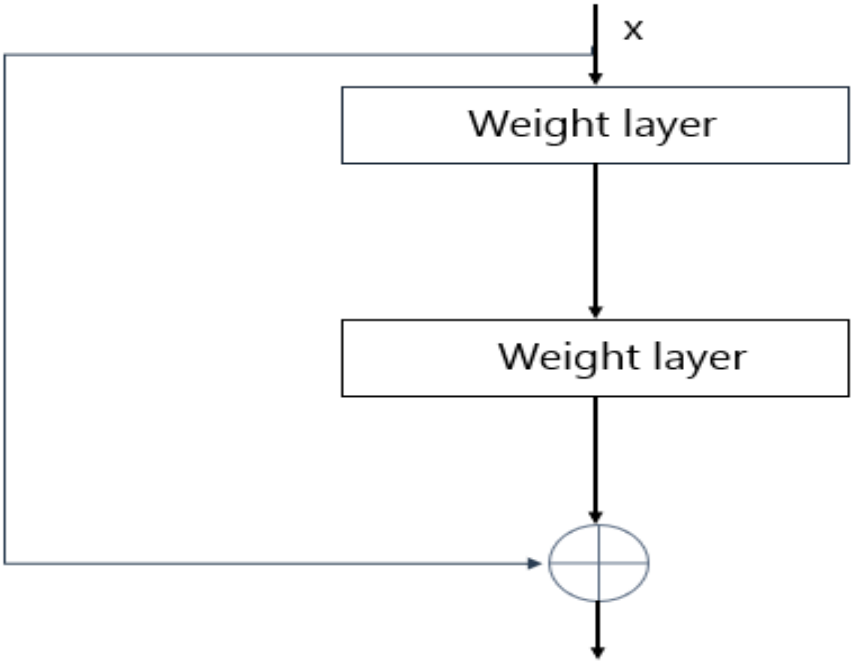

图 7: 堆叠残差块

4.4 Neck 网络构建

4.4.1 SPP 网络实现

首先，对 SPP 结构的输入进行 3 次卷积，得到 3 种有效特征层。使用 5*5、9*9、13*13 的最大池化和对输入的有效特征层进行最大池化，最后将三种池化结果与一条短接边进行堆叠形成 SPP 结构的输出。最后，对 SPP 结构的输出进行 3 次卷积运算。

```python
class SpatialPyramidPooling(nn.Module):
    def __init__(self, pool_sizes=[5, 9, 13]):
        super(SpatialPyramidPooling, self).__init__()

        self.maxpools = nn.ModuleList([nn.MaxPool2d(pool_size, 1, pool_size//2) for pool_size in pool_sizes])

    def forward(self, x):
        features = [maxpool(x) for maxpool in self.maxpools[::-1]]
        features = torch.cat(features + [x], dim=1)

        return features
```

图 8: SPP 网络实现

4.4.2 PANet 网络实现

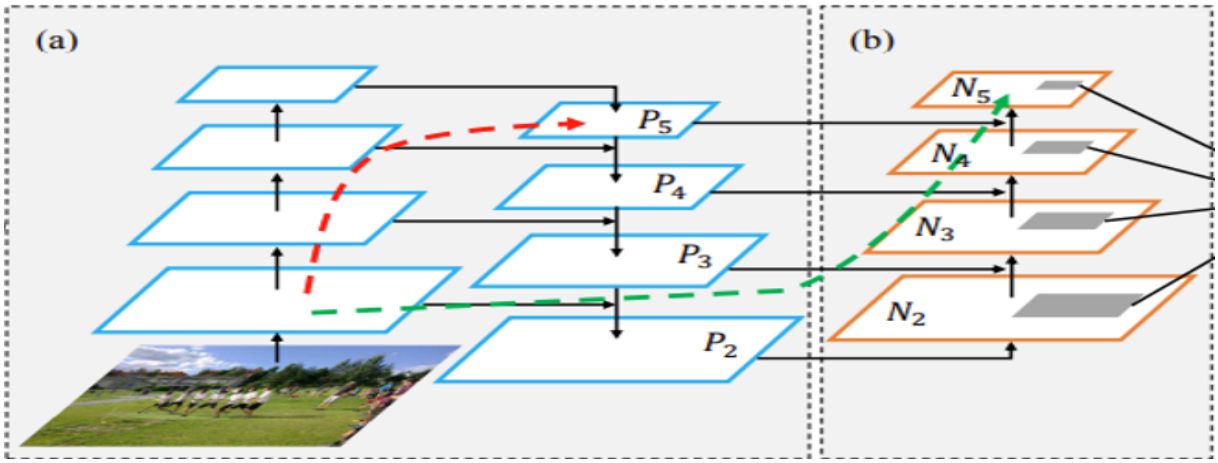

图 9: PANet 网络

PANet 是一种用于实例分割的路径聚合网络。特点是对输入的特征进行反复提取。上图 a 是一个传统的特征金字塔结构。FPN 网络对最底层的图像特征进行向上采样，并与该底层特征进行融合，得到高分辨率、强语义的特征（即加强了特征的提取）。YOLOv4 当中使用 FPN 对 52*52*256、26*26*512、13*13*1024 的三个有效特征层进行特征提取。在使用 FPN 进行自顶向下的特征提取之后，PANet 再进行一次自底向上的特征提取。

4.5 Head 网络构建

```python
def yolo_head(filters_list, in_filters):
    m = nn.Sequential(
        conv2d(in_filters, filters_list[0], 3),
        nn.Conv2d(filters_list[0], filters_list[1], 1),
    )
    return m
```

图 10: head 网络实现

4.6 创新点

1.YOLOv4 设计了一个可在单个传统 GPU 上实时运行的 CNN 模型，同时该模型只需要一个 GPU 进行训练。YOLOv4 使用单个传统 GPU 进行训练和测试就可以实现实时，高质量和可行的目标检测结果。

2.YOLOv4 对 YOLOv3 网络进行了优化改进，相较于 YOLOv3 模型，AP 和 FPS 分别有 10% 和 12% 的提升。

5 实验结果分析

采用 voc 2007 数据集作为训练集对 YOLOv4 目标检测模型进行训练，训练的参数 batchsize=2,epoch=50.

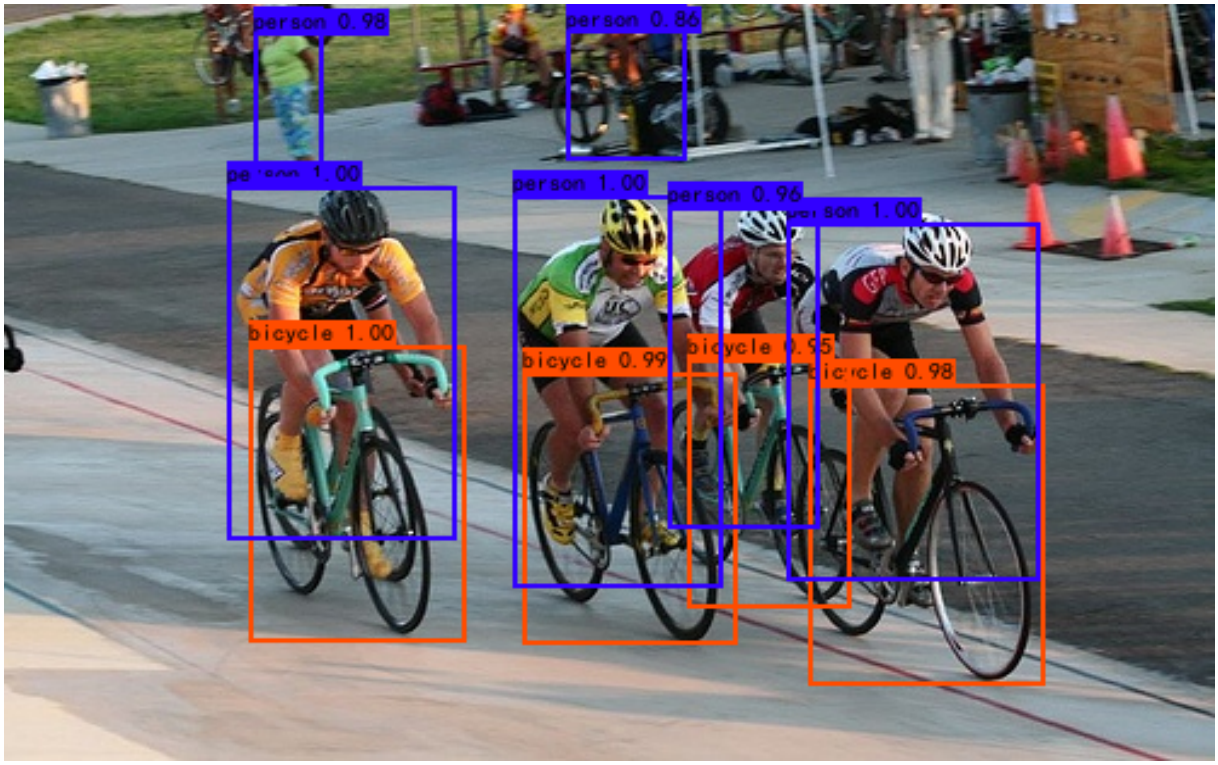

图 11: 目标检测实验结果示意

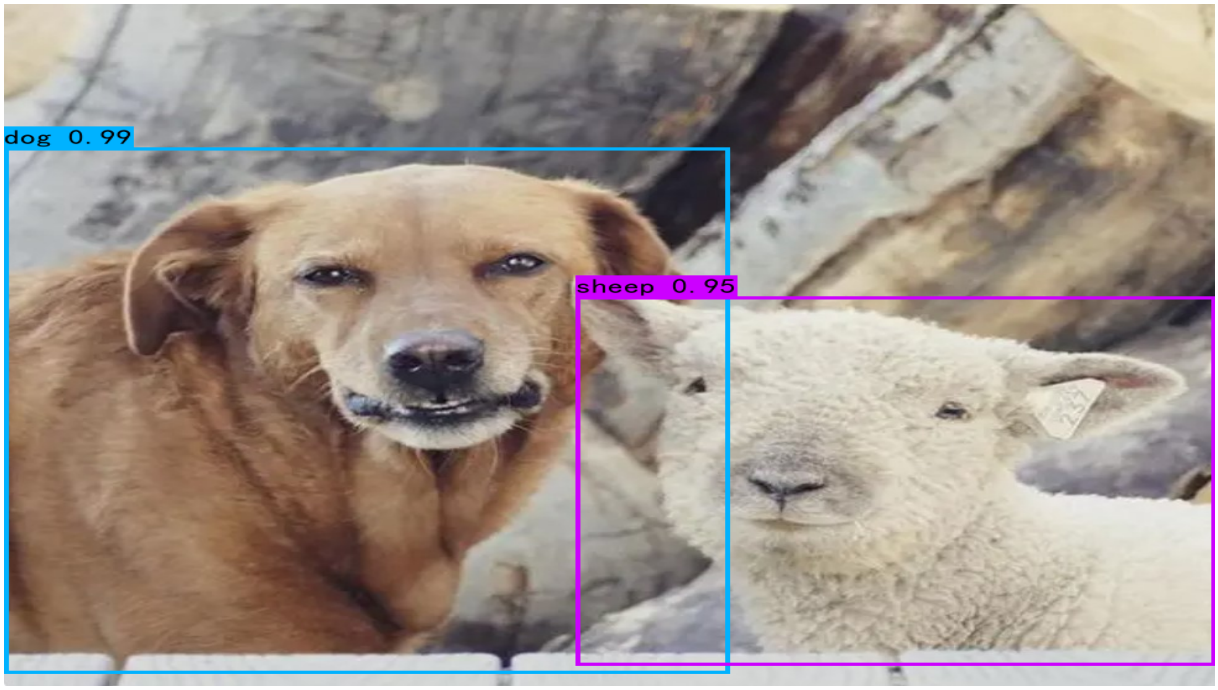

图 12: 目标检测实验结果示意

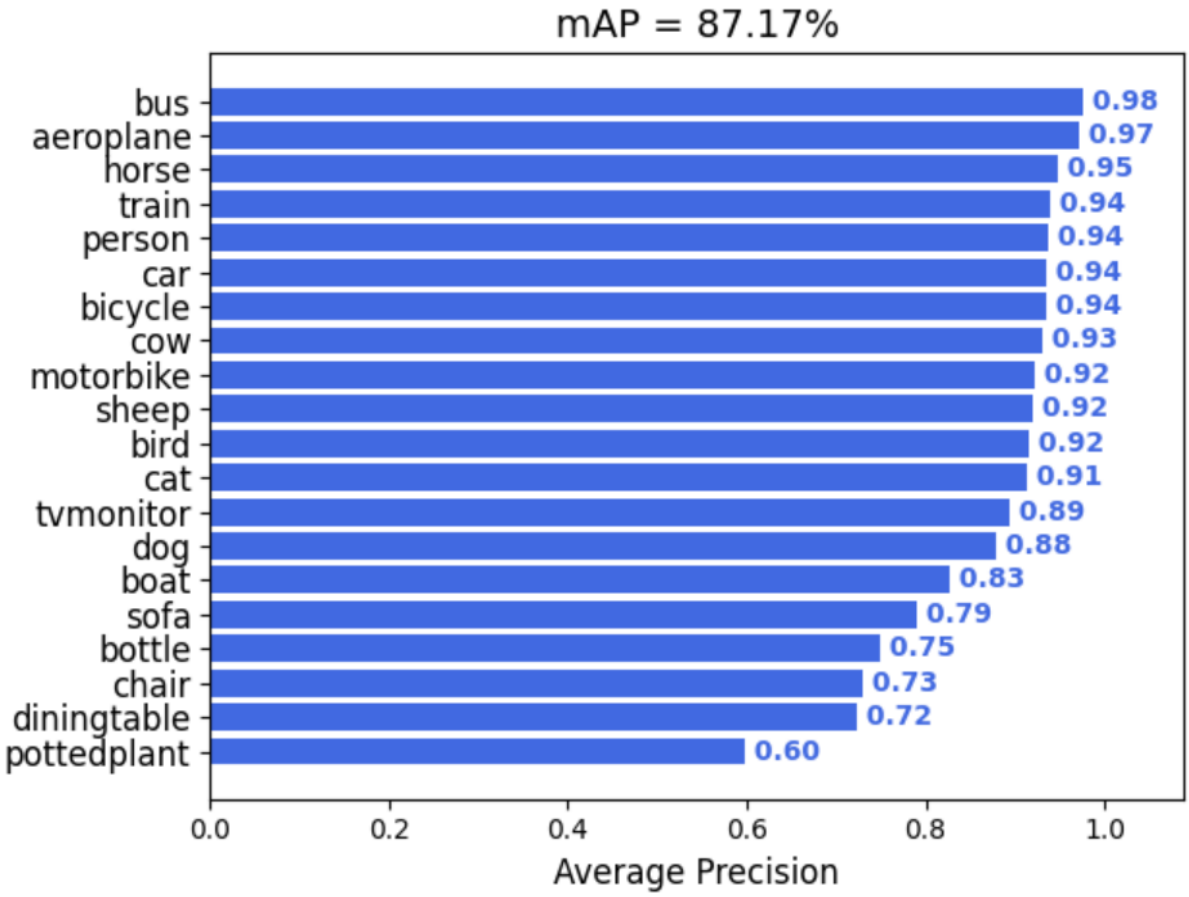

图 13: 目标检测测试 mAP 结果

从测试结果的 mAP 数值可以看出复现的 YOLOv4 目标检测模型的预测精确度比较好。

6 总结与展望

YOLOv4 实现了较好的目标检测速度与检测准确度，但检测速度与精确度还可进一步提升。在未来可以对 YOLOv4 的网络进行进一步的优化，进一步提升其目标检测的速度与准确率，让 YOLO 目标检测网络可以更加适用于实时性的应用场景，进一步普及相关产业的目标检测应用，给相关产业带来可观的经济效益。

参考文献

[1] Bochkovskiy A, Wang C Y, Liao H Y M. Yolov4: Optimal speed and accuracy of object detection[J]. arXiv preprint arXiv:2004.10934, 2020.

[2] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014

[3] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[4] Xie S, Girshick R, Dollár P, et al. Aggregated residual transformations for deep neural networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 1492-1500.

[5] Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4700-4708.

[6] Iandola F N, Han S, Moskewicz M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size[J]. arXiv preprint arXiv:1602.07360, 2016.

[7] Howard A G, Zhu M, Chen B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861, 2017.

[8] Zhang X, Zhou X, Lin M, et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 6848-6856.

[9] Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2014: 580-587.

[10] Girshick R. Fast r-cnn[C]//Proceedings of the IEEE international conference on computer vision. 2015: 1440-1448.

[11] Ren S, He K, Girshick R, et al. Faster r-cnn: Towards real-time object detection with region proposal networks[J]. Advances in neural information processing systems, 2015, 28.

[12] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 779-788.