《计算机视觉》
作业报告

大作业

图像目标检测任务

学　院: 计算机科学与技术

姓　名: 朱超群

学　号: 210110321

专　业: 计算机类

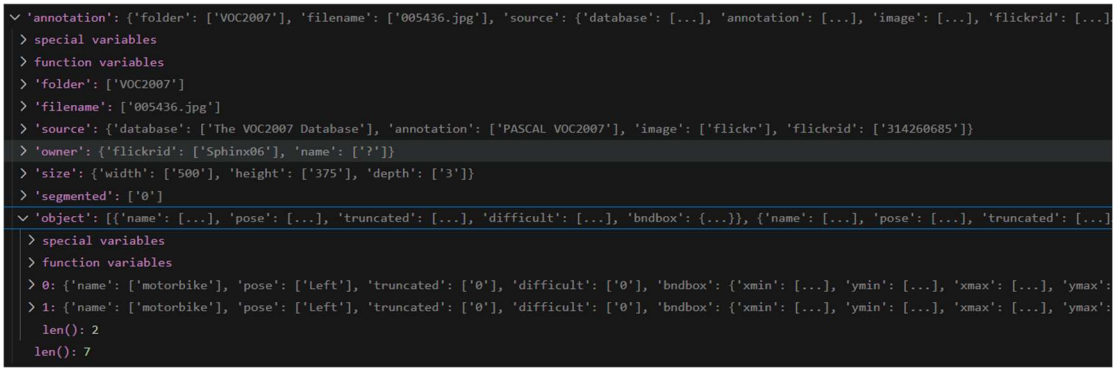
日　期: 2024 年 6 月

数据集情况

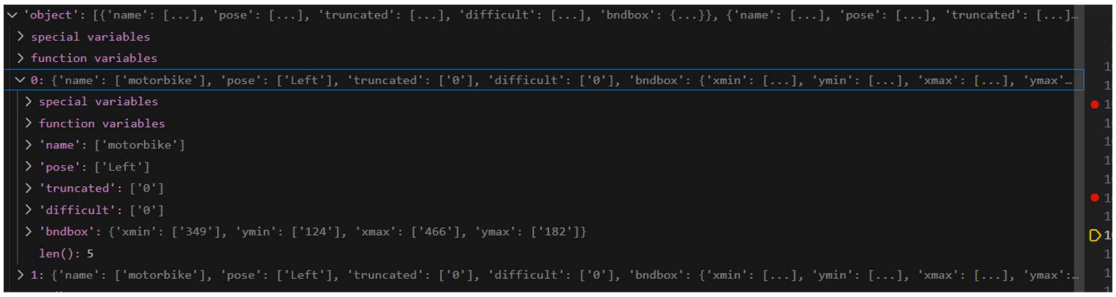
总体描述：采用了 pascal_voc_2007 数据集: 训练集总共有 2501 张图像,测试集共有 2510 张图像，共有 20 个类别(不包括背景)

图像大小：每张图像均为 3 通道的 rbg 图像。但是 h 和 w 不固定，大小在 1000 以内，500 左右。

标注情况：



载入数据集以后可以得知：每个标注包括了 7 个内容，它们分别是：folder,filename,source,owner,size,segmented,object,前 6 项表示图像和标注的基本信息，包括文件夹，文件名，数据库来源，所有者，大小，以及是否可用图像分割标注。最后一项表示标注的目标，可以看到，在这个图像中有两个目标被标注。下面将对其中的第 0 个进行分析。



可以看到，被标注的信息有 name,pose,truncated,difficult,bndbox，他们分别代表：name—目标类别名称，这里标注的是一辆摩托车、pose—目标的姿势，这里标注姿势为偏左、truncated—目标是否被截断，这里表示目标没有被截断，

也就是可以完整地看到目标、difficult—目标是否能清晰地被人眼识别，这里表示能够被人眼识别、bndbox—目标的 bounding box，指目标的框的值，这里的格式为(x1,y1,x2,y2)类型，有的数据集为(h,w,x,y)类型,所以我们需要对于标签进行预处理。

标注的选择：在本次任务中，只筛选 annotation 的目标中 difficult == 0 的 bndbox 和 name。注：在筛选 name 的时候会进行一个映射，将字符串映射成为数字。映射如下：

```
pascal_object_dict = {
        "aeroplane": 0,
        "bicycle": 1,
        "bird": 2,
        "boat": 3,
        "bottle": 4,
        "bus": 5,
        "car": 6,
        "cat": 7,
        "chair": 8,
        "cow": 9,
        "diningtable": 10,
        "dog": 11,
        "horse": 12,
        "motorbike": 13,
        "person": 14,
        "pottedplant": 15,
        "sheep": 16,
        "sofa": 17,
        "train": 18,
        "tvmonitor": 19
        }
```

数据集预处理

对于输入图像，如图所示

```python
    # 首先进行数据预处理
    transform = torchvision.transforms.Compose([
        torchvision.transforms.Resize((448, 448)),
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
```

首先将数据预处理成为 448*448 大小的图片，然后将其转化为 tensor，最后进行归一化。

对于标签的预处理，如图所示：

```python
# 返回的target采用了7*7*30的格式，其中前4个是坐标信息，第5个是置信度 预测两个box，接下来的20个是类别信息
def target_transform(target):
    # 新建一个tensor，大小为(7, 7, 30)
    target_tensor = torch.zeros((7, 7, 30))
    objects = target['annotation']['object']
    size = target['annotation']['size']
    for obj in objects:
        name = obj['name']
        bbox = obj['bndbox']
        xmin = int(bbox['xmin'])
        ymin = int(bbox['ymin'])
        xmax = int(bbox['xmax'])
        ymax = int(bbox['ymax'])
        # resize了图片，所以要重新计算坐标
        xmin = xmin * 448 / int(size['width'])
        xmax = xmax * 448 / int(size['width'])
        ymin = ymin * 448 / int(size['height'])
        ymax = ymax * 448 / int(size['height'])
        # 计算中心坐标
        x_center = (xmin + xmax) / 2
        y_center = (ymin + ymax) / 2
        # 计算宽高
        width = xmax - xmin
        height = ymax - ymin
        # 计算所在的网格
        grid_x = int(x_center // 64)
        grid_y = int(y_center // 64)
        # # 计算相对于网格的坐标
        # x_center = x_center %64 / 64
        # y_center = y_center %64 / 64
        # width = width / 448
        # height = height / 448
        # 计算类别
        class_index = pascal_object_dict[name]
        # 将目标信息填入tensor
        target_tensor[grid_y, grid_x, 0] = x_center
        target_tensor[grid_y, grid_x, 1] = y_center
        target_tensor[grid_y, grid_x, 2] = width
        target_tensor[grid_y, grid_x, 3] = height
        target_tensor[grid_y, grid_x, 4] = 1
        target_tensor[grid_y, grid_x, 5] = x_center
        target_tensor[grid_y, grid_x, 6] = y_center
        target_tensor[grid_y, grid_x, 7] = width
        target_tensor[grid_y, grid_x, 8] = height
        target_tensor[grid_y, grid_x, 9] = 1
        target_tensor[grid_y, grid_x, 10 + class_index] = 1
    return target_tensor.view(-1)
```

对于原始的标注信息,取出其中的 bounding_box 以及其类别,最终采用的标

注格式为 7*7*30 的格式，7*7 是指将图像分割为 7*7 的格子,如果由目标的中心落入某个格子，那么就由这个格子来负责此目标的检测。30 是由 2*5+20 组成，其中选择了预测两个 bounding box, 每个 bounding box 格式为(x,y,w,h,confidence)。另外 20 为类别。其中原始的标注格式为(xmin,ymin,xmax,ymax)，这里需要将其转化为(x,y,w,h)格式

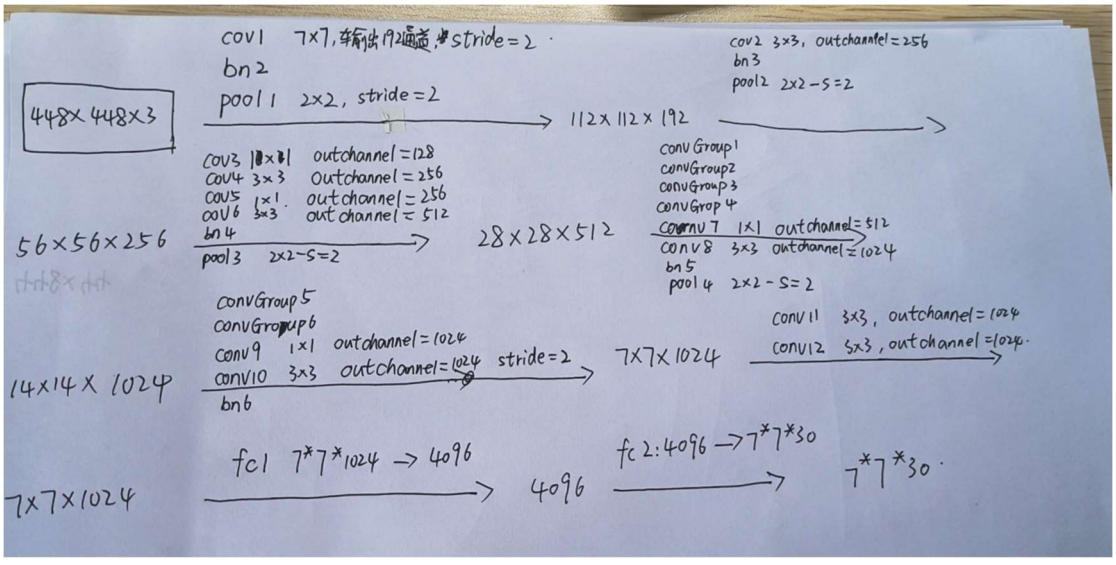
模型结构

模型采用 448*448*3 的输入，输出为 7*7*30。模型旨在将输入图像分割为 7*7 的网格,由每个物体的中心的网格负责检测该物体的 boundingbox。所以输出的前两维为 7*7，而每一个网格负责预测两个 boundingbox，boundingbox 的格式为(x,y,w,h,confidence),以及该 box 的图像的类别(voc 图像总共 20 个类别)，所以第三维的大小为 2*5+20 = 30。

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=192, kernel_size=7, stride=2, padding=3, bias=True)
        self.bn2 = nn.BatchNorm2d(192)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        self.conv2 = nn.Conv2d(in_channels=192, out_channels=256, kernel_size=3, stride=1, padding=1, bias=True)
        self.bn3 = nn.BatchNorm2d(256)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        self.conv3 = nn.Conv2d(in_channels=256, out_channels=128, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv4 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv5 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv6 = nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, stride=1, padding=1, bias=True)
        self.bn4 = nn.BatchNorm2d(512)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        self.convGroup1 = ConvModule1()
        self.convGroup2 = ConvModule1()
        self.convGroup3 = ConvModule1()
        self.convGroup4 = ConvModule1()
        self.conv7 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv8 = nn.Conv2d(in_channels=512, out_channels=1024, kernel_size=3, stride=1, padding=1, bias=True)
        self.bn5 = nn.BatchNorm2d(1024)
        self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)


        self.convGroup5 = ConvModule2()
        self.convGroup6 = ConvModule2()
        self.conv9 = nn.Conv2d(in_channels=1024, out_channels=1024, kernel_size=1, stride=1, padding=0, bias=True)
        self.conv10 = nn.Conv2d(in_channels=1024, out_channels=1024, kernel_size=3, stride=2, padding=1, bias=True)
        self.bn6 = nn.BatchNorm2d(1024)

        self.conv11 = nn.Conv2d(in_channels=1024, out_channels=1024, kernel_size=3, stride=1, padding=1, bias=True)
        self.conv12 = nn.Conv2d(in_channels=1024, out_channels=1024, kernel_size=3, stride=1, padding=1, bias=True)

        self.fc1 = nn.Linear(7*7*1024, 4096)
        self.fc2 = nn.Linear(4096, 7*7*30)
```

```
Model(
  (conv1): Conv2d(3, 192, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3))
  (bn2): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(192, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
  (conv6): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (convGroup1): ConvModule1(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (conv2): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (convGroup2): ConvModule1(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (conv2): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (convGroup3): ConvModule1(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (conv2): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (convGroup4): ConvModule1(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (conv2): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv7): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
  (conv8): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (convGroup5): ConvModule2(
    (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1))
    (conv2): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (convGroup6): ConvModule2(
    (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1))
    (conv2): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv9): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1))
  (conv10): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (bn6): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv11): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv12): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=50176, out_features=4096, bias=True)
  (fc2): Linear(in_features=4096, out_features=1470, bias=True)
)
```

模型的总体结构如图所示，总的来说，模型由 24 个卷积层，13 个 batchnorm 层，4 个池化层组成。



如上图所示：这里给出了模型的具体结构：

1. 输入图像为 448*448*3 的大小,经过一个 7*7 步长为 2,输出通道为 192 的卷积层,batchnorm 层以及一个 2*2 步长为 2 的最大池化层后得到了 112*112*192 的输出
2. 112*112*192 的输入经过一个 3*3 输出通道为 256 的卷积层,batchnorm 层以及一个 2*2 步长为 2 的池化层后得到了 56*56*256 的输出
3. 56*56*256 的输入经过 1*1,out_channel=128; 3*3,out_channel=256; 1*1,out_channel=256; 1*1,out_channel=512; 的连续卷积后再经过 batchnorm 层和 2*2 步长为 2 的池化层得到 28*28*512 的输出。
4. 28*28*512 的输入经过 4 个卷积组，每个卷积组由 1*1,outchannel=256;3*3,outchannel=512 的卷积层和一个 batchnorm 层组成。然后再经过 1*1,outchannel=512;3*3,outchannel=1024 的卷积层和 batchnorm 层以及一个 2*2 步长为 2 的池化层得到了 14*14*1024 输出。
5. 14*14*1024 的输入经过两个卷积组，每个卷积组由 1*1,outchannel=512 和 3*3,outchannel=1024 的卷积层和一个 batchnorm 层组成。然后再经过 1*1,outchannel=1024;3*3,outchannel=1024 步长为 2 的卷积层和一个 batchnorm 层得到了 7*7*1024 的输出。
6. 7*7*1024 的输入经过两个 3*3,outchannel=1024 的卷积层得到 7*7*1024 的输出
7. 7*7*1024 的输入经过一个输入为 7*7*1024,输出为 4096 全连接层得到 4096 的输出
8. 4096 的输入经过一个输入为 4096,输出为 7*7*30 的全连接层得到一个

7*7*30 的输出。

损失函数

损失函数设计:

损失函数由四部分组成:

1. 对于目标的 boundingbox 的损失

   对于 x,y 采用均方误差求损失，对于 w,h 的损失是采用$\sqrt{w},\sqrt{h}$的均方误差求损失。其系数为 coord,选择为 5

2. 对于非目标的检测框的置信度损失：采用均方误差求损失，其系数为 noobj，选择为 0.5

3. 对于目标的置信度损失：采用均方误差求损失。

4. 对于目标的类别损失：采用交叉熵损失。

实现细节如下图所示

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

    def forward(self, pred, target):
        # pred: (batch, 7*7*30)
        # target: (batch, 7*7*30)
        pred = pred.view(-1, 7, 7, 30)
        target = target.view(-1, 7, 7, 30)

        # 对于包含物体的检测框，计算坐标损失
        loss_coord = torch.sum(((target[:, :, :, 0] - pred[:, :, :, 0])**2 + (target[:, :, :, 1] - pred[:, :, :, 1])**2)* target[:, :, :, 4])
        loss_coord += torch.sum(((target[:, :, :, 5] - pred[:, :, :, 5])**2 + (target[:, :, :, 6] - pred[:, :, :, 6])**2)* target[:, :, :, 9])
        # 对于torch.sqrt(pred[:, :, :, 2])，如果pred[:, :, :, 2]为负数，会报错，所以需要使用torch.sqrt(pred[:, :, :, 2])，这样可以保证pred[:, :, :, 2]为正数
        loss_coord += torch.sum(((torch.sqrt(target[:, :, :, 2]) - torch.sqrt(torch.clamp(pred[:, :, :, 2], min=0)))**2 + (torch.sqrt(target[:, :, :, 3]) - torch.sqrt(torch.clamp(pred[:, :, :, 3], min=0)))**2)* target
        loss_coord += torch.sum(((torch.sqrt(target[:, :, :, 7]) - torch.sqrt(torch.clamp(pred[:, :, :, 7], min=0)))**2 + (torch.sqrt(target[:, :, :, 8]) - torch.sqrt(torch.clamp(pred[:, :, :, 8], min=0)))**2)* target

        # 对于包含物体的检测框，计算置信度损失
        loss_conf = torch.sum((target[:, :, :, 4] - pred[:, :, :, 4])**2 * target[:, :, :, 4] + (target[:, :, :, 9] - pred[:, :, :, 9])**2 * target[:, :, :, 9])

        # 对于不包含物体的检测框，计算置信度损失
        loss_noobj = torch.sum((target[:, :, :, 4] - pred[:, :, :, 4])**2 * (1 - target[:, :, :, 4]) + (target[:, :, :, 9] - pred[:, :, :, 9])**2 * (1 - target[:, :, :, 9]))

        # 类别损失，仅对包含物体的检测框计算，使用交叉熵损失
        # 先将target[:, :, :, 10:] 使用softmax归一化
        target_has_obj = target[target[:, :, :, 4] == 1]
        pred_has_obj = pred[target[:, :, :, 4] == 1]
        loss_class = nn.CrossEntropyLoss()(pred_has_obj[:, 10:], target_has_obj[:, 10:].max(1)[1])


        res = coord*loss_coord + loss_conf + noobj * loss_noobj + loss_class
        return res
```

超参数选择

学习率：0.001，随 epoch 动态变化：每 70 个 epoch 学习率*0.2

损失函数的参数:

对于目标的损失函数的参数 coord = 5

对于非目标的损失函数的参数 noobj = 0.5

训练 epoch 300

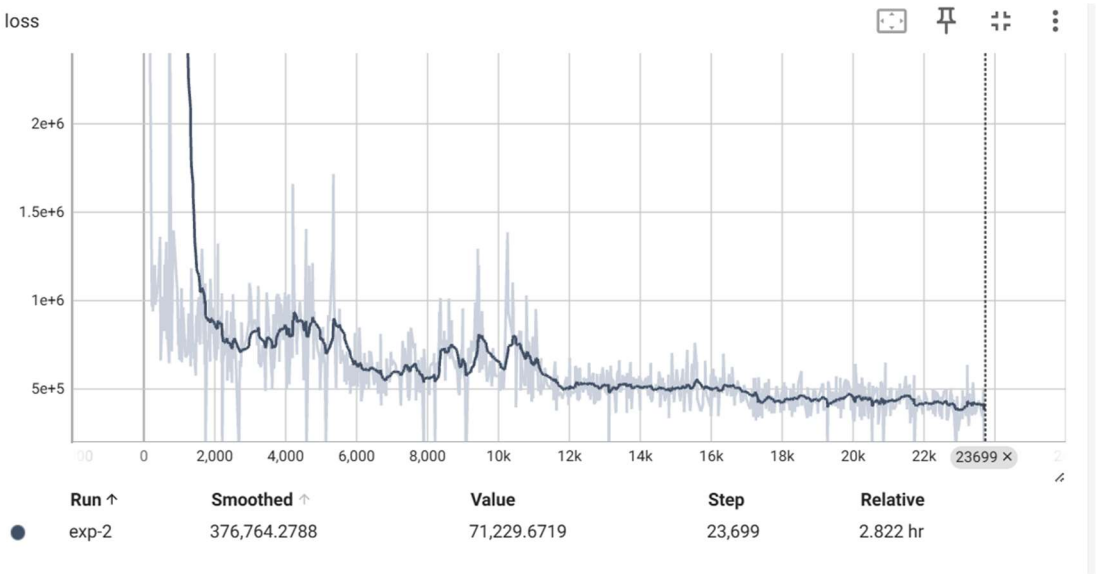
非极大值抑制阈值：0.4

置信度阈值：0.4

训练情况统计：使用 tensorboard

性能分析

训练：



在 tensorboard 可视化中可以看到总训练时长为 2.822 小时，在开始阶段损失是快速下降的，但是在后面损失就是震荡着下降了。

评估：

通过对测试集的前 100 个数据分析得到其 Precision,Recall,F1 score 和 IoU 如下图所示: (在 detect.ipynb 文件中有体现)

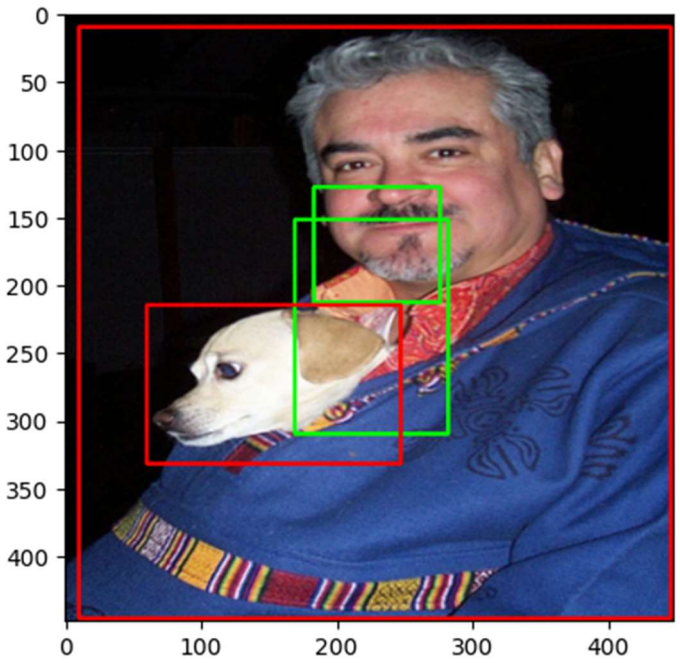
```
Precision: 0.4636363636363636
Recall: 0.40823970037453183
F1 Score: 69.4875
average IOU: tensor(0.2080)
```

可以看到,在预测为正类的样本中，实际上为正类的样本数占比为 46%，在实际为正类的样本中，被正确判断为正类的比例为 40%，F1 Score 为 69.4857，平均 IoU 为 0.2。

根据以上信息可以分析出，本次任务的模型对于正确样本的判断准确率大概到 40%-50%，而且对于预测正确的样本来说，boundingbox 与真实值之间的交并比的平均值为 0.2 这可能是因为训练批次不够造成的,也有可能是模型训练的损失函数的参数设置问题。

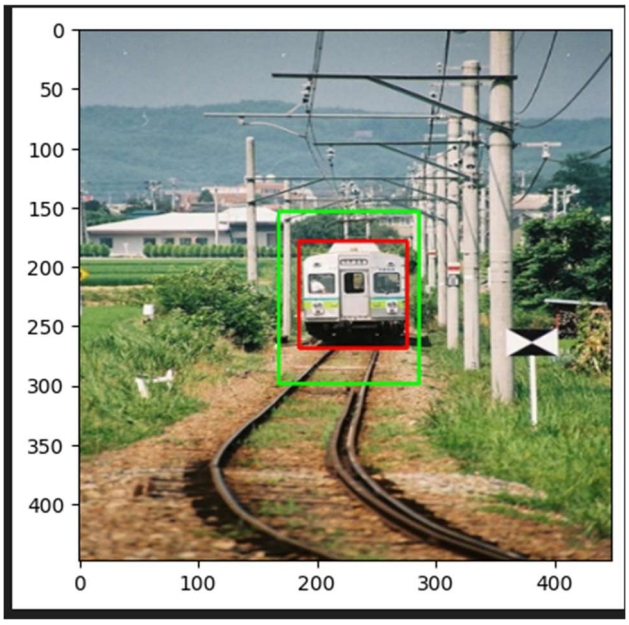
实验结果展示

注: 红色代表真实标签，绿色是模型的预测值，此处展示了部分的预测结果

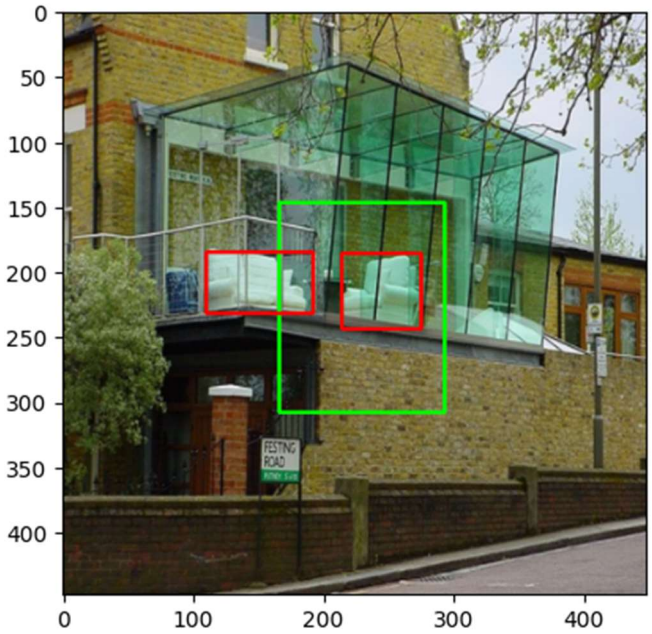


分析：对于人来说,模型只识别出了脸部的一部分，对于狗来说，模型识

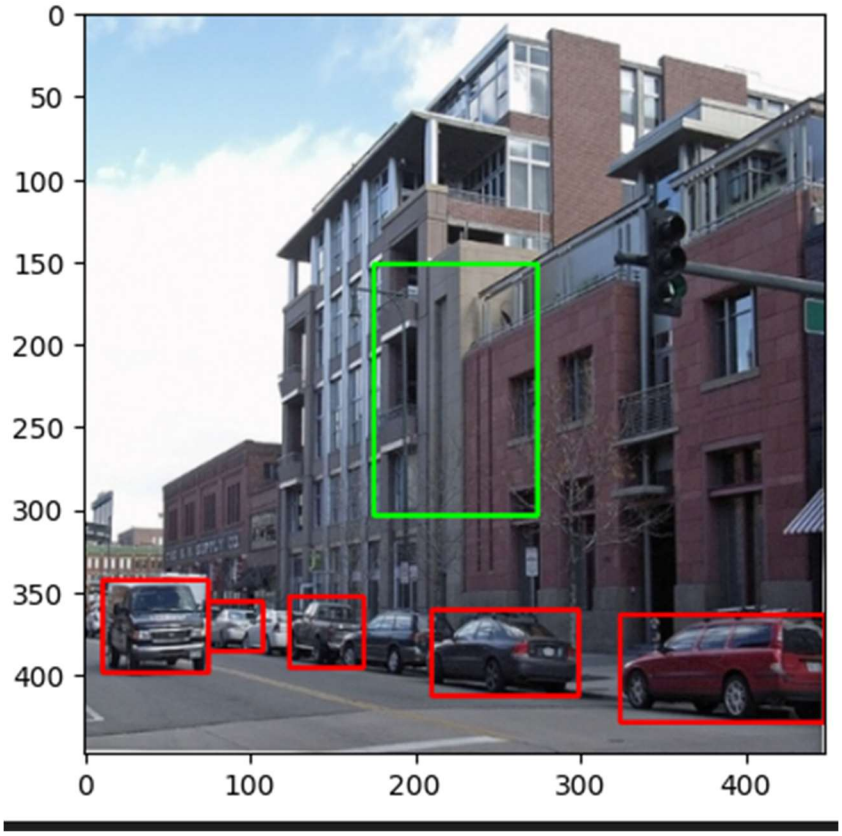
别出了耳朵那一块，但是没有把头识别成功。这可能与分类的损失函数构造有关



分析：对于这个结果来说，模型成功地将火车识别，但是检测框却过于大了。这可能与判断检测框的大小的损失函数或者是训练批次不够有关。



分析：这两张沙发由于是透过玻璃来看的，模型很可能会认为沙发和玻璃是一体的，于是就将两张沙发识别成为了一个物体。



分析: 这一张图片是比较失败的,失败原因在于模型的结构上，由于模型将原图像分割为 7*7 的栅格，每个格子负责预测一个物体，那么就造成了模型在预测小体积的物体上会有很大的误差现象，这张图中的几个汽车均是小体积物体，模型就没有识别出来。