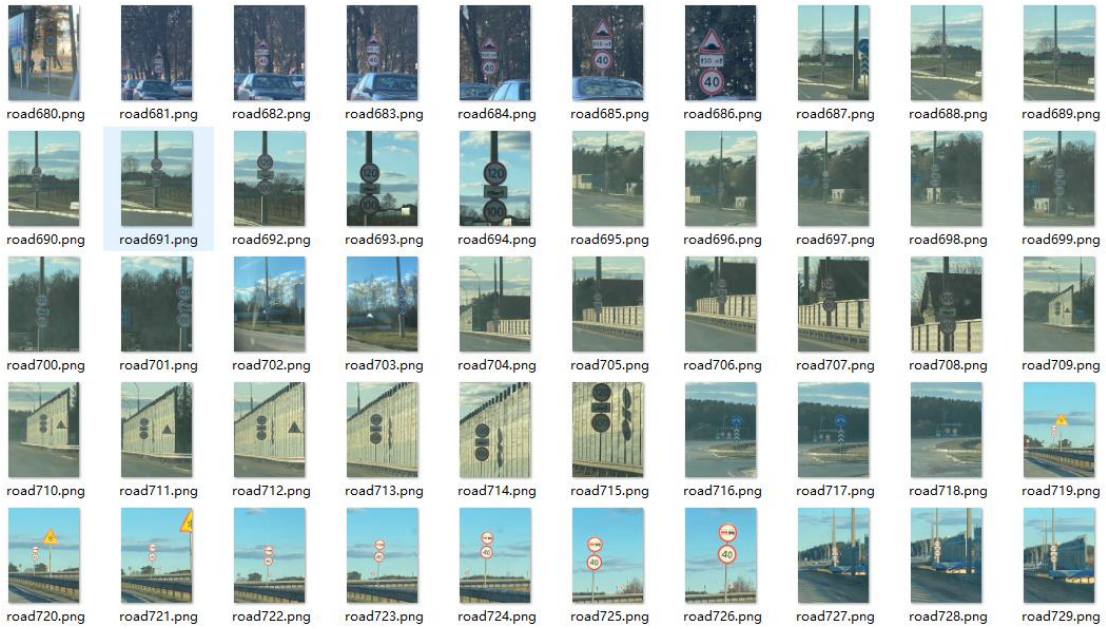
Final Project 报告

210010101
房煊梓

一、数据准备

1.**选择的数据集**：Kaggle 道路标志检测数据集，包含 877 张含有道路标志的图像。



2.**数据预处理**：对于数据的预处理操作包括统一图像和边界框的大小，随机旋转，随机裁剪，中心裁剪和归一化，实现了对数据的增强。

```python
# 统一图像和边界框的大小
def resize_image_and_bb(read_path, write_path, bb, size):
    # 读取图像
    image_read = cv2.imread(str(read_path))
    image = cv2.cvtColor(image_read, cv2.COLOR_BGR2RGB)

    # 定义新的大小
    new_size = int(1.45 * size)
    # 调整图像大小
    image_resized = cv2.resize(image, (new_size, size))
    # 获取掩码并调整掩码大小
    mask = get_mask(bb, image)
    mask_resized = cv2.resize(mask, (new_size, size))
    # 根据掩码获取对应的调整后的边界框
    bb_resized = get_bb_by_mask(mask_resized)

    # 获取输出的路径
    output_path = write_path / read_path.name
    # 保存图像
    cv2.imwrite(str(output_path), cv2.cvtColor(image_resized, cv2.COLOR_RGB2BGR))

    return output_path, bb_resized
```

```python
# 整体变换
def transforms(image_path, bounding_box, apply_transforms):
    # 读取图像
    image_read = cv2.imread(str(image_path)).astype(np.float32)
    image = cv2.cvtColor(image_read, cv2.COLOR_BGR2RGB) / 255.0

    # 创建掩码
    mask = get_mask(bounding_box, image)

    if apply_transforms:
        # 旋转
        image, mask = rotate(image, mask)
        # 随机裁剪
        image, mask = random_crop(image, mask)
    else:
        # 中心裁剪
        image, mask = center_crop(image, mask)

    # 将图像转换为PyTorch张量
    return image, get_bb_by_mask(mask)
```

```python
# 使用之前定义的变换对图像进行预处理
image, y_bb = transforms(path, self.bb[index], self.apply_transforms)
# 归一化
image = (image - [0.485, 0.456, 0.406])/[0.229, 0.224, 0.225]
```

二、模型构建

1.**特征提取器：**使用 ResNet18 的神经网络模型的前 8 个子模块作为特征提取器，采用的权重为使用 ResNet18 模型的默认预训练权重。

```python
# 使用resnet18作为特征提取器，使用的权重为默认的参数
resnet = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
self.layers = nn.Sequential(*list(resnet.children())[:8])
```

2.**分类器和边界框回归器：**使用全连接层作为分类器和边界框回归器。

```python
# 分类器
self.classifier = nn.Sequential(
    nn.BatchNorm1d(512),
    nn.Linear(512, 4)
)

# 边界框回归器
self.bb = nn.Sequential(
    nn.BatchNorm1d(512),
    nn.Linear(512, 4)
)
```

三、超参数选择

1.**批量大小**：batch_size 为 32，即每个训练批次包含的样本数为 32。
2.**损失函数**：分类部分采用交叉熵损失函数，边界框回归部分采用 L1 损失函数，并且除以常数 C 进行加权。

```
# 类别损失
loss_class = F.cross_entropy(out_class, y_class, reduction="sum")
# 边界框坐标损失
loss_bb = F.l1_loss(out_bb, y_bb, reduction="none").sum(1).sum()
loss = loss_class + loss_bb/C
```

3.**优化器**：采用 Adamax 函数作为优化器，初始学习率设置为 0.005。

```
optimizer = torch.optim.Adamax(parameters, lr=0.005)
```

四、训练过程

1.**训练模型**：对于每个 epoch，需要进行前向传播，损失计算，反向传播和参数更新，并且评估模型在验证集上的损失和准确率。

```
# 对于每个epoch
for epoch in range(epochs):
    model.train()
    # 初始化
    total_samples = 0
    total_loss = 0
    # 对于训练集的每个batch进行训练
    for x, y_class, y_bb in dataloader_train:
        batch = y_class.shape[0]
        x = x.float()
        y_bb = y_bb.float()

        # 获取预测的类别和边界框
        out_class, out_bb = model(x)

        # 计算类别损失
        loss_class = F.cross_entropy(out_class, y_class, reduction="sum")

        # 边界框坐标损失
        loss_bb = F.smooth_l1_loss(out_bb, y_bb, reduction="none").sum(1).sum()
        loss = loss_class + loss_bb/C

        # 梯度清零
        optimizer.zero_grad()
        # 反向传播
        loss.backward()
        # 更新参数
        optimizer.step()
        total_samples += batch
        total_loss += loss.item()
    # 计算训练损失
    train_loss = total_loss/total_samples
    # 计算损失和准确率
    val_loss, val_acc = val_metrics(model, dataloader_val, C)
    print("train loss: %.4f; val loss: %.4f; val acc: %.4f" % (train_loss, val_loss, val_acc))
```

```
# 训练模型
model = MyModel()
parameters = filter(lambda p: p.requires_grad, model.parameters())
optimizer = torch.optim.Adamax(parameters, lr=0.005)
train(model, optimizer, dataloader_train, dataloader_val, epochs=10)
```

2.**更新学习率进行训练：**调用 update_lr 函数对学习率进行更新，再调用 train 函数对模型进行训练。

```
# 更新优化器的学习率
def update_lr(optimizer, lr):
    for i, param_group in enumerate(optimizer.param_groups):
        param_group["lr"] = lr
```

```
# 更新学习率进行训练
update_lr(optimizer, 0.002)
train(model, optimizer, dataloader_train, dataloader_val, epochs=10)
```

五、实验结果分析

1.**训练模型：**训练 10 个 epoch，结果如下。

```
train loss: 1.9039; val loss: 71.4865; val acc: 0.1989
train loss: 1.4607; val loss: 1.8874; val acc: 0.7557
train loss: 1.3263; val loss: 1.4356; val acc: 0.7443
train loss: 1.2083; val loss: 1.3230; val acc: 0.7784
train loss: 1.0060; val loss: 0.9904; val acc: 0.7727
train loss: 0.9572; val loss: 0.8066; val acc: 0.8068
train loss: 0.8142; val loss: 0.9095; val acc: 0.7614
train loss: 0.7552; val loss: 1.0197; val acc: 0.8011
train loss: 0.8200; val loss: 1.1013; val acc: 0.7159
train loss: 0.7226; val loss: 0.7707; val acc: 0.7898
```

训练集损失：对于 train loss，随着 epoch 增加，除了第 9 个 epoch 的 train loss 有所上升之外，train loss 整体呈现降低的趋势，说明模型在训练过程中逐渐有效地学习到了数据集的特征。

验证集损失：对于 val loss，第 1 个 epoch 的 val loss 比其他 epoch 大许多，第 2-6 个 epoch 中 val loss 呈现降低趋势，第 7-9 个 epoch 中 val loss 呈现上升趋势，但在第 10 个 epoch 中又出现了降低，说明初始时模型未能对验证集进行有效的泛化，而在后续的训练中性能有所提升但不够稳定。

验证集准确率：对于 val acc，第 1 个 epoch 的 val acc 较低，在后续的 epoch 中 val acc 较高，并且在 0.7 和 0.8 左右波动，说明初始时模型的随机性较大，而在后续的训练中有不错的性能但是没有明显进步，可能出现过拟合现象。

2.**更新学习率进行训练：**将学习率更新为 0.002 后训练 10 个 epoch，结果如下。

```
train loss: 0.6939; val loss: 0.7792; val acc: 0.8011
train loss: 0.6202; val loss: 0.7008; val acc: 0.8125
train loss: 0.5928; val loss: 0.6530; val acc: 0.8523
train loss: 0.5140; val loss: 0.6188; val acc: 0.8409
train loss: 0.5369; val loss: 0.9003; val acc: 0.8352
train loss: 0.5186; val loss: 0.6280; val acc: 0.8295
train loss: 0.4508; val loss: 0.8936; val acc: 0.8580
train loss: 0.4241; val loss: 0.6147; val acc: 0.8523
train loss: 0.4680; val loss: 0.6895; val acc: 0.8864
train loss: 0.4066; val loss: 0.4816; val acc: 0.8807
```

训练集损失：对于 train loss，数值上比之前更小，并且整体呈现降低的趋势，说明模型在降低学习率后，在训练过程中能更加有效地学习数据集的特征。
验证集损失：对于 val loss，数值上比之前更小，并且整体呈现降低的趋势，说明模型在降低学习率后，在验证集上的表现变得更好。
验证集准确率：对于 val acc，在 10 个 epoch 中稳定超过 0.8，并且大约每 3 个 epoch 出现上升和下降趋势的波动，最后达到 0.8807 的准确率，说明模型在降低学习率之后可以进一步提升在验证集上的准确率，但也可能存在过拟合现象。

3.**选一个例子查看检测结果**：选取一个图片（这里为 road76.png），利用训练的模型进行目标检测，并可视化检测的结果。

可看到蓝色的边界框基本将“STOP”标志牌框住，可知该目标检测模型具有良好的准确性。

六、总结

本项目在 kaggle 道路标志检测数据集上实现了目标检测任务，经过训练的模型可以较为准确地检测到道路标志并用蓝色边界框进行指示。
本项目让我在处理数据集和编写代码的过程中学习到了目标检测任务的流程，明白了处理数据、构建模型、训练模型和评估模型的具体实现，加深了对计算机视觉相关任务的认识，收获颇丰。