# VOC2012 语义分割

本次课程项目的代码可在 我的 Github 上找到。

## 数据准备

本次任务选取 PASCAL VOC 2012 segmentation 数据集进行语义分割任务。数据集包含 20 个类别的图像和对应的分割标注。原有的数据集已完成了预处理，包含训练集、验证集。

可从 `torchvision` 库中直接下载 VOC2012 数据集如下：

```
import torchvision
dataset = torchvision.datasets.VOCSegmentation(
    root='data/VOC2012',
    year='2012',
    image_set='train', # [train | val | trainval]
    download=True,
)
```

通过实现 `torch.utils.data.Dataset` 接口，可以自定义数据集类来加载图像和分割标注。参考 awesome-semantic-segmentation-pytorch 的实现，我将图像裁剪的变换嵌入在该类中，而将标准化和张量化操作作为参数传入。

主体逻辑如下：

```
class VocDataSet(Dataset):
    def __init__(self, root='data/VOCdevkit', split='train', transform=None,
base_size=512, crop_size=320):
        ...
    def __getitem__(self, index):
        img = Image.open(self.images[index]).convert('RGB')
        mask = Image.open(self.masks[index])
        # synchronized transform
        if self.split == 'train':
            img, mask = self._sync_transform(img, mask)
        else:
            img, mask = self._val_sync_transform(img, mask)
        # general resize, normalize and toTensor
        if self.transform is not None:
            img = self.transform(img)
        return img, mask, os.path.basename(self.images[index])
    ...
```

其中，`_sync_transform` 和 `_val_sync_transform` 方法实现了图像裁剪和变换。

本次任务传入的变换如下：

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

其中， `mean` 和 `std` 分别是 ImageNet 数据集的均值和标准差，用于对图像进行标准化处理。

数据集处理完毕后，将其传入 `torch.utils.data.DataLoader` 中进行批处理和加载如下：

```
val_dataset = dataset.VocDataSet(split='val', transform=transform)
train_dataset = dataset.VocDataSet(split='train', transform=transform)
```

```
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False, num_workers=1)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True, num_workers=1)
```

## 模型结构

语义分割任务与传统分类任务的主要区别在于输出的标签是每个像素点的类别，而不是整个图像的类别。因此，语义分割模型通常需要一个编码器-解码器结构。在解码器部分，通常会使用上采样操作将特征图恢复到原始图像的大小。

因此，在 CNN 将原始图像编码为特征图后，解码器部分需要将特征图上采样到原始图像的大小，并通过卷积层将通道数转换为类别数。如果只进行一步到位上采样，则很可能会导致上采样后的特征图信息丢失。因此，通常会使用多次上采样和卷积操作来逐步恢复特征图的空间分辨率。

本次任务使用了 U-Net 模型结构。U-Net 是一种常用的语义分割模型，具有编码器-解码器结构。其主要特点是通过跳跃连接将编码器部分的特征图与解码器部分的特征图进行融合，从而保留更多的空间信息。

模型结构大致如下：

```
class UNet(nn.Module):
    def __init__(self, n_channels, n_classes, bilinear=False):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear

        self.inc = (DoubleConv(n_channels, 32))
        self.down1 = (Down(32, 64))
        self.down2 = (Down(64, 128))
        self.down3 = (Down(128, 256))
        factor = 2 if bilinear else 1
        self.down4 = (Down(256, 512 // factor))
        self.up1 = (Up(512, 256 // factor, bilinear))
        self.up2 = (Up(256, 128 // factor, bilinear))
        self.up3 = (Up(128, 64 // factor, bilinear))
        self.up4 = (Up(64, 32, bilinear))
        self.outc = (OutConv(32, n_classes))

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        logits = self.outc(x)
        return logits
```

其中，`Down` 层为常规的卷积层和池化层，`Up` 层为上采样和卷积层，上采样使用反向卷积进行，`DoubleConv` 层为连续的卷积层，`OutConv` 层为输出层，将通道数转换为类别数。

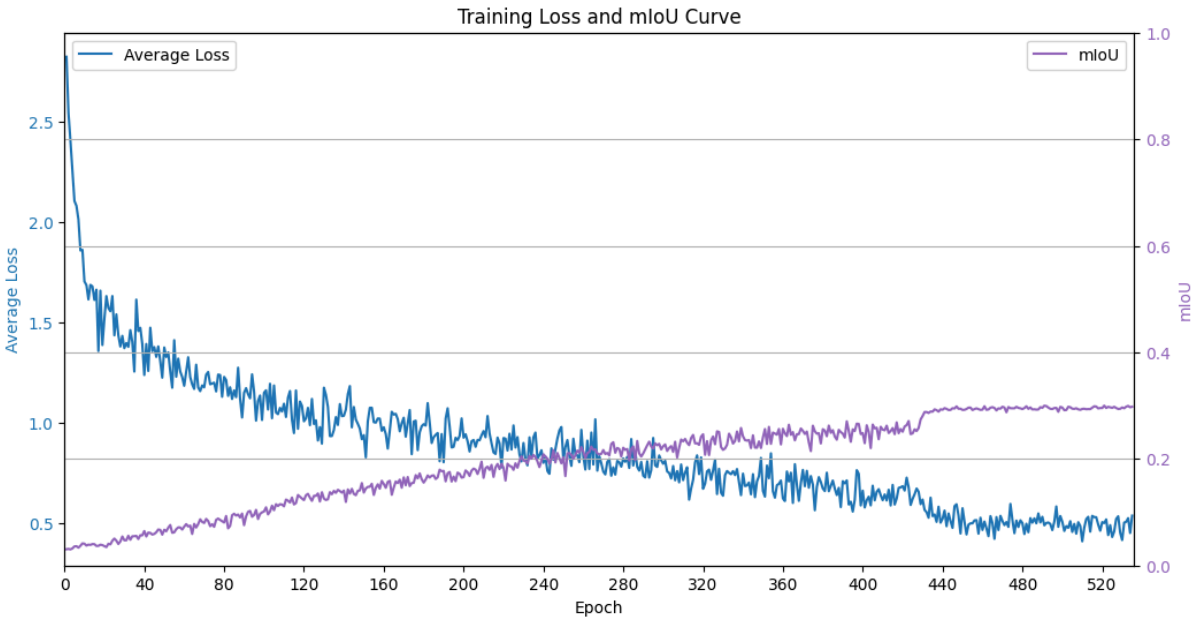
## 训练过程

训练过程配置如下：

1. GPU: NVIDIA L40
2. Batch size: 16
3. Optimizer: AdamW
4. Learning rate: 1e-5 — 1e-4
5. Loss function: CrossEntropyLoss
6. Epochs: 600 max

验证集采用的指标为 mIoU（mean Intersection over Union），即每个类别的 IoU 的平均值。同时计算 Pixel Accuracy（像素准确率）。

训练过程的 Loss 和 mIoU 曲线如下所示：




初始学习率为 1e-4，在 400 个 epoch 时，模型的 mIoU 达到了 0.29，此时调整学习率为 1e-5 后 mIoU 上升了一个点，但此后不再有明显提升(sad)。

## 实验结果与分析

经过 535 个 epoch 的训练，模型在验证集上的最佳指标如下：

Epoch: 521 PixAcc: 0.7914, mIoU: 0.3000

对验证集的部分图像进行预测，结果如下：

由结果可知，模型对类别所在区域的预测较为准确，但对更细分的区域（车轮内部）以及较为阴暗的区域预测较差。另外，类别的预测出错的概率也较高。目前来看，该模型的性能仍有较大提升空间。

我认为需要进一步改进模型结构或训练策略，例如：

1. **数据增强**：增加数据集的多样性，例如随机裁剪、旋转、翻转等操作，以提高模型的泛化能力。
2. **模型结构**：尝试更复杂的模型结构，例如 DeepLabV3、PSPNet 等，这些模型在语义分割任务中表现较好。
3. **损失函数**：尝试使用更适合语义分割任务的损失函数，例如 Focal Loss、Dice Loss 等，这些损失函数可以更好地处理类别不平衡问题。
4. **学习率调度**：使用学习率调度器，根据验证集的指标动态调整学习率，以提高模型的收敛速度和性能。

另外还需要进一步分析模型的错误预测，找出模型在不同类别的预测上存在的问题，并针对性地进行改进。

## 总结

本次任务实现了基于 U-Net 模型的 PASCAL VOC 2012 数据集的语义分割。通过自定义数据集类和数据加载器，成功加载了数据集并进行了预处理。模型训练过程中，使用了 AdamW 优化器和 CrossEntropyLoss 损失函数，经过 535 个 epoch 的训练，模型在验证集上的 mIoU 达到了 0.30。

这次课程项目使我充分体验了神经网络的训练过程和语义分割任务的挑战。虽然模型的性能仍有提升空间，但我对语义分割任务有了更深入的理解，并掌握了如何使用 PyTorch 实现自定义数据集和模型结构。

由于时间和资源的限制，本次实验未能达到更高的性能指标，但我相信通过进一步的改进和优化，模型的