# 基于深度学习 Unet 架构实现户型图分割

## 一、项目介绍

### 1.1 基本介绍

图像分割是计算机视觉研究中图像理解领域关注的一个热点，同时也是图像分析的第一步，是计算机视觉的基础，是图像理解的重要组成部分，同时也是图像处理中最困难的问题之一。

其中户型图分割是其在计算机视觉在建筑与室内设计领域的典型应用，旨在将户型图中的墙体、门窗、功能区等元素进行像素级划分。传统方法依赖人工标注或规则算法，效率低且难以处理复杂场景。而深度学习经典架构Unet凭借其编码器-解码器结构和跳跃连接机制，在医学图像分割中已展现出了低成本高效率的优势，是迁移至户型图分割的一个非常经典的深度学习方法。由此本项目也是构建基于 Unet 的深度学习模型，实现对户型图中不同功能区的分割。

### 1.2 成员及分工

杨智权：数据处理、环境布置、代码编写
邱海涛：寻找资料、编写报告
黄堉轩：模型构建、代码编写
谢智华：代码编写、可视化

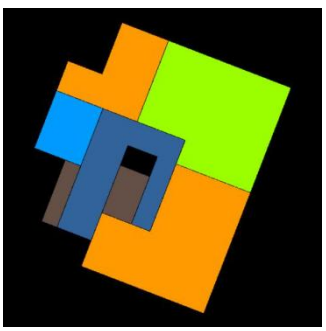
## 二、实验设计

### 2.1 项目环境

编程环境：Kaggle 的云工作台 notebook、Python 语言、GPU T4

### 2.2 数据准备（来源在附录中）

节取自 huggingface 某户型图中的 400 个数据图以及对应的标签图



### 2.3 实验方法

U-Net 在户型图分割任务中扮演着重要的角色，它通过编码器-解码器（Encoder-Decoder）

结构和跳跃连接（Skip Connection）来实现高精度的语义分割。　首先，U-Net 的编码器部分类似于经典的卷积神经网络（CNN），由多个卷积层和池化层（如最大池化）组成，逐步提取图像的高级特征，并减少空间分辨率，使网络能够学习到更抽象的语义信息。然后，解码器部分则通过上采样（Up-sampling）或转置卷积（Transposed Convolution）来逐步恢复图像的空间信息，使得最终的分割结果能够保持较高的分辨率。　同时，U-Net 的一个关键特性是跳跃连接（Skip Connection），它将编码器中不同层级的特征直接传输到解码器的相应层，这样不仅能够弥补由于池化导致的空间信息损失，还能使得网络在恢复高分辨率特征时更加精准。　户型图通常包含多个相邻但结构不同的房间，而跳跃连接能够帮助 U-Net 充分利用低级和高级特征，从而准确地分割不同房间区域。

在户型图分割任务中，U-Net 的输入是一张户型图，而输出是一个分割掩码（Segmentation Mask），其中每个像素点都被分类为不同的房间类别，比如卧室、客厅、厨房、卫生间等。 数据集需要包含带有人工标注的户型图。 训练时，使用损失函数交叉熵损失（Cross-Entropy Loss）和 Dice 损失（Dice Loss），其中交叉熵适用于多类别分割，而 Dice 损失更适合类别不均衡的情况。 优化器一般选择 Adam 或 SGD，以保证模型的稳定收敛。此外，评估分割效果的指标主要包括 IoU（Intersection over Union）和 Dice 系数，IoU 衡量预测区域和真实区域的重叠程度，而 Dice 系数则更关注匹配的精确度，尤其适用于不均衡数据集。

## 2.4 实验步骤

### 2.4.1 数据预处理

先从该 huggingface 网页获取带标注的户型图数据集，每张图对应一个标注掩膜（Mask）选取其中的 400 个，由于数据已经标准化，故不再进行过多数据增强等操作，后续就是归一化后进行简单的尺寸调整，统一输入尺寸（256x256）。

### 2.4.2 定义 Unet 模型

```
        # 定义卷积块：Conv2d + BatchNorm2d + ReLU 的组合
        def CBR2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=True):
            layers = []
            # 添加卷积层
            layers += [nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
kernel_size=kernel_size,stride=stride, padding=padding,bias=bias)]
            # 添加批归一化层
            layers += [nn.BatchNorm2d(num_features=out_channels)]
            # 添加激活函数
            layers += [nn.ReLU()]
            return nn.Sequential(*layers)  # 将层组合成序列
        ### 编码器（下采样）
        # 第 1 个编码阶段（输入分辨率最高）
        self.enc1_1 = CBR2d(in_channels, 64)    # 输入通道→64 通道
        self.enc1_2 = CBR2d(64, 64)                  # 保持 64 通道
        self.pool1 = nn.MaxPool2d(kernel_size=2)  # 下采样（高宽减半）
        .....
        # 第 5 个编码阶段（最底层，分辨率最小）
```

self.enc5_1 = CBR2d(512, 1024)
self.enc5_2 = CBR2d(1024, 1024)

### 解码器（上采样）
# 第 5 个解码阶段（最底层开始上采样）
self.dec5_1 = CBR2d(1024, 512)
self.dec5_2 = CBR2d(512, 512)
# 转置卷积实现上采样（分辨率加倍）
self.unpool4 = nn.ConvTranspose2d(512, 512, kernel_size=2, stride=2)
.....
# 第 1 个解码阶段（输出层）
self.dec1_2 = CBR2d(128, 64)　　# 处理 64 上采样 + 64 编码特征 → 128 通道
self.dec1_1 = CBR2d(64, out_channels)　# 最终输出层（无激活函数，通常后续接 Sigmoid 或 Softmax

后接 forward 层的定义，此处省略。

2.4.3 模型训练

首先是初始化之前的 Unet 模型做准备，接着定义函数（选择了 CrossEntropyLoss）与优化器（Adam 优化器，初始学习率 0.001），定义训练 epoch 为 20。

后续再遍历数据集进行训练，过程中计算并打印 epoch 平均损失。

```
criterion = nn.CrossEntropyLoss()  # 使用交叉熵损失函数
optimizer = optim.Adam(model.parameters(), lr=0.001)  # Adam优化器，初始学习率0.001

num_epochs = 20  # 定义训练轮数（可根据训练情况调整）

# 开始训练循环
for epoch in range(num_epochs):
    model.train()  # 设置模型为训练模式（启用BatchNorm/Dropout）
    running_loss = 0.0  # 累计每个epoch的损失

    # 遍历训练数据集
    for images, masks in train_loader:

        images = images.to(device)
        masks = masks.to(device)

        optimizer.zero_grad()  # 梯度清零

        outputs = model(images)
        loss = criterion(outputs, masks)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()  # 累计当前batch的损失值

    # 计算并打印epoch平均损失
    epoch_loss = running_loss / len(train_loader)
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}')
```

2.4.4 模型预测

模型预测阶段首先完成训练模型的加载与初始化配置，将训练好的 UNet 模型参数从保存路径加载至预设设备（GPU 优先），并调用 model.eval()切换为评估模式，关闭训练中的 Dropout 和 BatchNorm 层动态特性以保证预测稳定性。随后定义可视化函数，通过 matplotlib

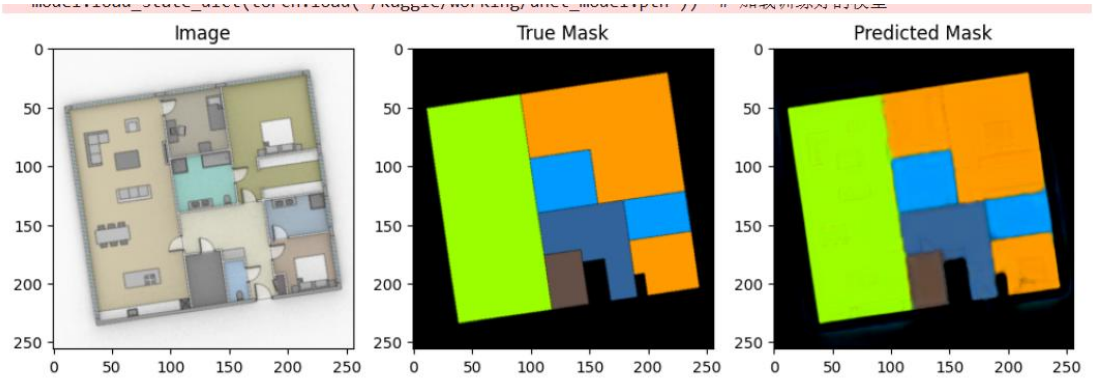
将输入图像、真实分割掩膜与预测结果并排展示，直观对比三者的空间一致性。

```
# 定义保存路径
save_path = '/kaggle/working/unet_model.pth'

# 确保目录存在
os.makedirs(os.path.dirname(save_path), exist_ok=True)

# 保存模型状态字典
torch.save(model.state_dict(), save_path)
print(f'Model saved to {save_path}')
```

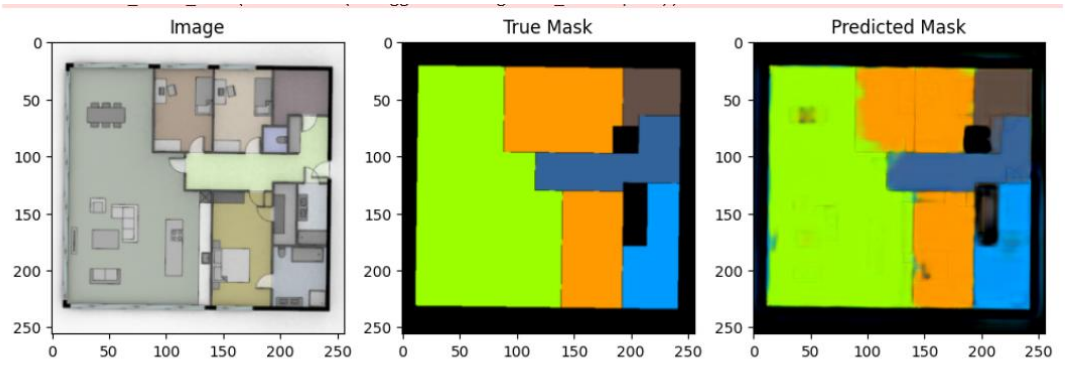
```
# 设置设备为GPU（如果可用），否则使用CPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# 初始化UNet模型并加载训练好的模型参数
model = UNet().to(device)
model.load_state_dict(torch.load('/kaggle/working/unet_model.pth'))  # 加载训练好的模型
model.eval()  # 将模型设置为评估模式

# 定义可视化函数，用于显示图像、真实掩码和预测掩码
def visualize(image, mask, pred_mask):
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 3, 1)
    plt.title('Image')
    plt.imshow(make_grid(image, nrow=1).permute(1, 2, 0).numpy())  # 显示图像
    plt.subplot(1, 3, 2)
    plt.title('True Mask')
    plt.imshow(make_grid(mask, nrow=1).permute(1, 2, 0).numpy())  # 显示真实掩码
    plt.subplot(1, 3, 3)
    plt.title('Predicted Mask')
    plt.imshow(make_grid(pred_mask, nrow=1).permute(1, 2, 0).numpy())  # 显示预测掩码
    plt.show()
```

# 三、实验结果

## 3.1 模型效果

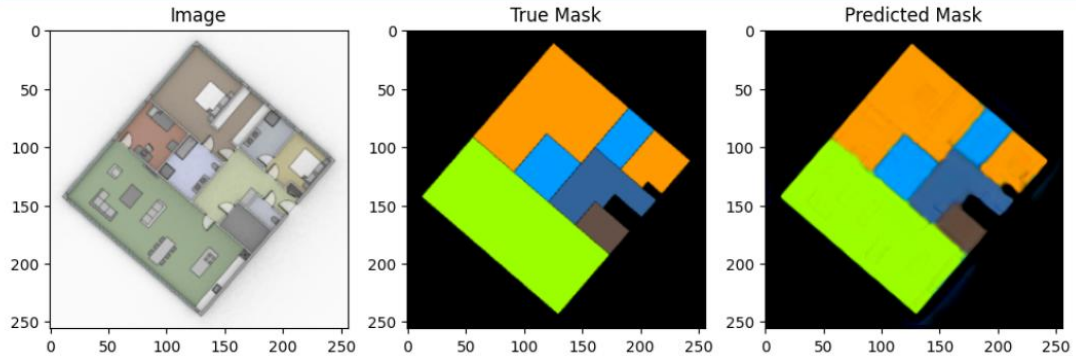



+ Code + Markdown

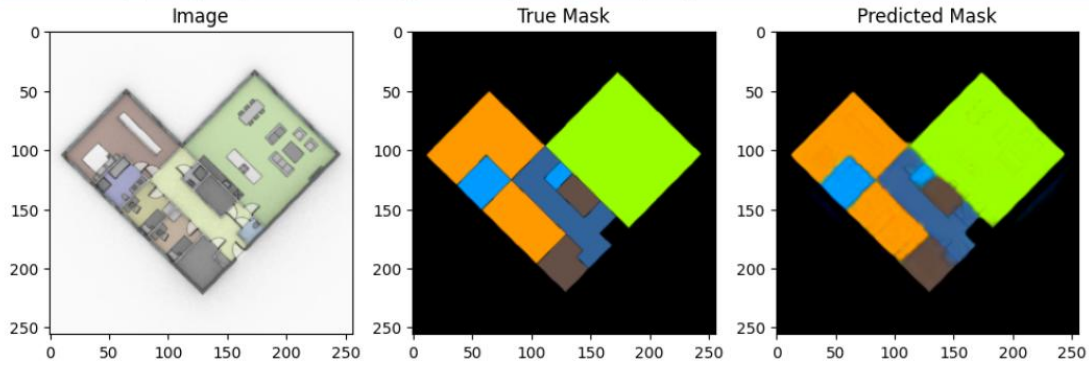



+ Code + Markdown

```
model.load_state_dict(torch.load('/kaggle/working/unet_model.pth'))  # 加载训练好的模型
```




+ Code + Markdown




+ Code + Markdown

# 四、参考文献

## 5.1 数据来源

https://huggingface.co/datasets/zimhe/pseudo-floor-plan-12k

## 5.2 项目 github 链接

https://github.com/seekingstar/CV_floor_plan_split