

## 你

这篇论文介绍了一种名为"双信息交叉网络" (Dual Information Crossing Network, DICN) 的有效方法, 用于使用航拍图像和轨迹数据进行道路提取。DICN集成了源特定的特征编码网络、像素注意力融合模块和边缘预测分支, 以实现不同模态之间的特征交互。此外, 该模型还使用了像素感知的对比学习和迁移学习方法来提升其性能。在两个公开数据集BJRoad和Porto上的大量实验表明, 所提出的方法在从航拍图像和轨迹数据中提取道路方面超越了现有的方法, 设定了新的最先进水平。

读取下面的 pytorch 核心网络代码, 核心网络部分为 SegTrans, 给出这里是如何实现 gps 和 遥感图像 两个通道的融合, 在代码中如何体现

```
class SegTrans(nn.Module):
    def init(self, num_classes=1, num_channels=3, encoder_1dconv=0, decoder_1dconv=0, pretrain=True):
        super().init()
```

```
        filters = [64, 128, 256, 512]
        self.num_channels = num_channels
        resnet = models.resnet34(pretrained=True)
        if num_channels < 3:
            self.firstconv = nn.Conv2d(num_channels, 64, kernel_size=7, stride=2, padding=3,
                                         bias=False)
        else:
            self.firstconv = resnet.conv1
        self.firstbn = resnet.bn1
        self.firstrelu = resnet.relu
        self.firstmaxpool = resnet.maxpool

        if encoder_1dconv == 0:
            self.encoder1 = resnet.layer1
            self.encoder2 = resnet.layer2
            self.encoder3 = resnet.layer3
            self.encoder4 = resnet.layer4
        else:
            myresnet = ResnetBlock()
            layers = [3, 4, 6, 3]
            basicBlock = BasicBlock1DConv
            self.encoder1 = myresnet._make_layer(basicBlock, 64, layers[0])
            self.encoder2 = myresnet._make_layer(
                basicBlock, 128, layers[1], stride=2)
            self.encoder3 = myresnet._make_layer(
                basicBlock, 256, layers[2], stride=2)
            self.encoder4 = myresnet._make_layer(
                basicBlock, 512, layers[3], stride=2)

        gpsscnn = models.resnet34(pretrained=True)
        self.num_channels_gpsscnn = 3
        if self.num_channels_gpsscnn < 3:
            self.firstconv_gpsscnn = nn.Conv2d(self.num_channels_gpsscnn, 64, kernel_size=7, stride=2, padding=3,
                                                bias=False)
        else:
            self.firstconv_gpsscnn = gpsscnn.conv1
        self.firstbn_gpsscnn = gpsscnn.bn1
        self.firstrelu_gpsscnn = gpsscnn.relu
        self.firstmaxpool_gpsscnn = gpsscnn.maxpool
        self.encoder1_gpsscnn = gpsscnn.layer1
        self.encoder2_gpsscnn = gpsscnn.layer2
        self.encoder3_gpsscnn = gpsscnn.layer3
        self.encoder4_gpsscnn = gpsscnn.layer4

        if decoder_1dconv == 0:
            self.decoder = DecoderBlock
        elif decoder_1dconv == 2:
            self.decoder = DecoderBlock1DConv2
        elif decoder_1dconv == 4:
            self.decoder = DecoderBlock1DConv4

        self.decoder4 = self.decoder(filters[3], filters[2])
        self.decoder3 = self.decoder(filters[2], filters[1])
        self.decoder2 = self.decoder(filters[1], filters[0])
        self.decoder1 = self.decoder(filters[0], filters[0])

        self.decoder_gps = DecoderBlock
        self.decoder4_gps = self.decoder_gps(filters[3], filters[2])
        self.decoder3_gps = self.decoder_gps(filters[2], filters[1])
        self.decoder2_gps = self.decoder_gps(filters[1], filters[0])
        self.decoder1_gps = self.decoder_gps(filters[0], filters[0])

        self.finaldeconv1 = nn.ConvTranspose2d(filters[0] * 2, 32, 4, 2, 1)
        self.finalrelu1 = nonlinearity
        self.finalconv2 = nn.Conv2d(32, 32, 3, padding=1)
```

```

self.finalrelu2 = nonlinearity
self.finalconv3 = nn.Conv2d(32, num_classes, 3, padding=1)
self.proj_head = ProjectionHead(dim_in=32, proj_dim=32)
if self.num_channels > 3:
    self.addconv = nn.Conv2d(
        self.num_channels - 3, 64, kernel_size=7, stride=2, padding=3)

self.edge = Edge_Module()

self.avgpool_img = nn.AdaptiveAvgPool2d((32, 32))
self.avgpool_gps = nn.AdaptiveAvgPool2d((32, 32))
self.input_chs = [64, 128, 256, 512]
self.transformer1 = SAFM(n_embd=self.input_chs[0],
                          n_head=4,
                          block_exp=4,

```

## Copilot

在这段代码中, SegTrans 类的构造函数初始化了两个独立的编码器, 一个用于处理遥感图像 (resnet), 另一个用于处理GPS数据 (gpscnn)。每个编码器都有自己的第一层卷积 (firstconv 和 firstconv\_gpscnn)、批归一化 (firstbn 和 firstbn\_gpscnn)、激活函数 (firstrelu 和 firstrelu\_gpscnn) 和最大池化层 (firstmaxpool 和 firstmaxpool\_gpscnn), 以及后续的编码层 (encoder1 到 encoder4 和 encoder1\_gpscnn 到 encoder4\_gpscnn)。

融合发生在解码

你

```

self.transformer1 = SAFM(n_embd=self.input_chs[0], n_head=4, block_exp=4, n_layer=1, img_vert_anchors=32,
img_horz_anchors=32, trj_vert_anchors=32, trj_horz_anchors=32, embd_pdrop=0.1, attn_pdrop=0.1,
resid_pdrop=0.1, ) self.transformer2 = SAFM(n_embd=self.input_chs[1], n_head=4, block_exp=4, n_layer=1,
img_vert_anchors=32, img_horz_anchors=32, trj_vert_anchors=32, trj_horz_anchors=32, embd_pdrop=0.1,
attn_pdrop=0.1, resid_pdrop=0.1, ) self.transformer3 = SAFM(n_embd=self.input_chs[2], n_head=4, block_exp=4,
n_layer=1, img_vert_anchors=32, img_horz_anchors=32, trj_vert_anchors=32, trj_horz_anchors=32,
embd_pdrop=0.1, attn_pdrop=0.1, resid_pdrop=0.1, ) self.transformer4 = SAFM(n_embd=self.input_chs[3],
n_head=4, block_exp=4, n_layer=1, img_vert_anchors=32, img_horz_anchors=32, trj_vert_anchors=32,
trj_horz_anchors=32, embd_pdrop=0.1, attn_pdrop=0.1, resid_pdrop=0.1, ) self.transformer5 =
SAFM(n_embd=self.input_chs[2], n_head=4, block_exp=4, n_layer=1, img_vert_anchors=32, img_horz_anchors=32,
trj_vert_anchors=32, trj_horz_anchors=32, embd_pdrop=0.1, attn_pdrop=0.1, resid_pdrop=0.1, )
self.transformer6 = SAFM(n_embd=self.input_chs[1], n_head=4, block_exp=4, n_layer=1, img_vert_anchors=32,
img_horz_anchors=32, trj_vert_anchors=32, trj_horz_anchors=32, embd_pdrop=0.1, attn_pdrop=0.1,
resid_pdrop=0.1, ) self.transformer7 = SAFM(n_embd=self.input_chs[0], n_head=4, block_exp=4, n_layer=1,
img_vert_anchors=32, img_horz_anchors=32, trj

```

## Copilot

在这段代码中, SAFM (Self-Attention Fusion Module) 是用于实现GPS数据和遥感图像之间融合的关键组件。每个transformer变量都是一个SAFM实例, 它们分别对应于不同的特征通道大小 (`self

你

```

def forward(self, input_): channels = input_.shape[1] if channels == 4: gps_feature = input_[ :, 3, :,
: ].unsqueeze(1) gps_feature = gps_feature.repeat(1, 3, 1, 1) elif channels ==6: gps_feature = input_[ :, 3:,
:, : ] x = input_[ :, :3, :, : ]

# Encoder
if self.num_channels > 3:
    add = self.addconv(x.narrow(1, 3, self.num_channels - 3))
    x = self.firstconv(x.narrow(1, 0, 3))
    x = x + add
else:
    x = self.firstconv(x)

x = self.firstbn(x)
x = self.firstrelu(x)
x = self.firstmaxpool(x)
gps_feature = self.firstconv_gpscnn(gps_feature)
gps_feature = self.firstbn_gpscnn(gps_feature)
gps_feature = self.firstrelu_gpscnn(gps_feature)
gps_feature = self.firstmaxpool_gpscnn(gps_feature)

e1 = self.encoder1(x)
e1_gps = self.encoder1_gpscnn(gps_feature)
e1_trans = self.avgpool_img(e1)
e1_gps_trans = self.avgpool_gps(e1_gps)
e1_trans, e1_gps_trans = self.transformer1(e1_trans, e1_gps_trans, False)
e1_trans = F.interpolate(e1_trans, size=(e1.shape[2], e1.shape[3]), mode='bilinear', align_corners=False)
e1_gps_trans = F.interpolate(e1_gps_trans, size=(e1_gps.shape[2], e1_gps.shape[3]), mode='bilinear', align_corners=False)
e1 = e1 + e1_trans

```

```

e1_gps = e1_gps + e1_gps_trans

e2 = self.encoder2(e1)
e2_gps = self.encoder2_gpscnn(e1_gps)
e2_trans = self.avgpool_img(e2)
e2_gps_trans = self.avgpool_gps(e2_gps)
e2_trans, e2_gps_trans = self.transformer2(e2_trans, e2_gps_trans, False)
e2_trans = F.interpolate(e2_trans, size=(e2.shape[2], e2.shape[3]), mode='bilinear', align_corners=False)
e2_gps_trans = F.interpolate(e2_gps_trans, size=(e2_gps.shape[2], e2_gps.shape[3]), mode='bilinear', align_corners=False)
e2 = e2 + e2_trans
e2_gps = e2_gps + e2_gps_trans

e3 = self.encoder3(e2)
e3_gps = self.encoder3_gpscnn(e2_gps)
e3_trans = self.avgpool_img(e3)
e3_gps_trans = self.avgpool_gps(e3_gps)
e3_trans, e3_gps_trans = self.transformer3(e3_trans, e3_gps_trans, False)
e3_trans = F.interpolate(e3_trans, size=(e3.shape[2], e3.shape[3]), mode='bilinear', align_corners=False)
e3_gps_trans = F.interpolate(e3_gps_trans, size=(e3_gps.shape[2], e3_gps.shape[3]), mode='bilinear', align_corners=False)
e3 = e3 + e3_trans
e3_gps = e3_gps + e3_gps_trans

e4 = self.encoder4(e3)
e4_gps = self.encoder4_gpscnn(e3_gps)
e4_trans = self.avgpool_img(e4)
e4_gps_trans = self.avgpool_gps(e4_gps)

e4_trans, e4_gps_trans = self.transformer4(e4_trans, e4_gps_trans, False)
e4_trans = F.interpolate(e4_trans, size=(e4.shape[2], e4.shape[3]), mode='bilinear', align_corners=False)
e4_gps_trans = F.interpolate(e4_gps_trans, size=(e4_gps.shape[2], e4_gps.shape[3]), mode='bilinear', align_corners=False)
e4 = e4 + e4_trans
e4_gps = e4_gps + e4_gps_trans

# # Decoder
d4 = self.decoder4(e4) + e3
d4_gps = self.decoder4_gps(e4_gps) + e3_gps
d4_trans = self.avgpool_img(d4)
d4_gps_trans = self.avgpool_gps(d4_gps)
d4_trans, d4_gps_trans = self.transformer5(d4_trans, d4_gps_trans, False)
d4_trans = F.interpolate(d4_trans, size=(d4.shape[2], d4.shape[3]), mode='bilinear', align_corners=False)
d4_gps_trans = F.interpolate(d4_gps_trans, size=(d4_gps.shape[2], d4_gps.shape[3]), mode='bilinear', align_corners=False)
d4 = d4 + d4_trans
d4_gps = d4_gps + d4_gps_trans

d3 = self.decoder3(d4) + e2
d3_gps = self.decoder3_gps(d4_gps) + e2_gps
d3_trans = self.avgpool_img(d3)
d3_gps_trans = self.avgpool_gps(d3_gps)
d3_trans, d3_gps_trans = self.transformer6(d3_trans, d3_gps_trans, False)
d3_trans = F.interpolate(d3_tr

```

## Copilot

在forward函数中，首先根据输入的通道数来处理GPS特征。如果通道数为4，则只有一个GPS特征通道，它会被扩展为3个通道以匹配遥感图像的RGB通道。如果通道数为6，则意味着有3个GPS特征通道。接着，遥感图像特征（x）和GPS特征（gps\_feature）分别通过各自的第一层卷积、批归一化、激活函数和最大池化层。

在编码器部分，遥感图像特征和GPS特征通过各自的编码器层（encoder1到encoder4和encoder1\_gpscnn到encoder4\_gpscnn）进行特征提取。然后，这些特征通过平均池化层（avgpool\_img和avgpool\_gps）进行下采样，并输入到SAFM模块（transformer1到transformer4）进行特征融合。SAFM模块使用自注意力机制来实现不同模

你

结合以上代码，给出 SegTrans 的如何实现双通道融合的

## Copilot

SegTrans模型通过以下步骤实现了GPS和遥感图像两个通道的融合：

- 特征提取：**使用两个独立的ResNet编码器，一个用于遥感图像特征（resnet），另一个用于GPS特征（gpscnn）。每个编码器都包含卷积层、批归一化、激活函数和最大池化层，以及多个残差块。
- 特征融合：**在每个编码器级别之后，通过SAFM (Self-Attention Fusion Module) 模块实现特征融合。SAFM使用自注意力机制来处理和融合来自两个不同源的特征。
- 特征转换：**使用自适应平均池化（avgpool\_img和avgpool\_gps）将编码器输出的特征图下采样到统一的空间分辨率，然后通过SAFM模块进行融合。
- 特征上采样：**融合后的特征图通过双线性插值（F.interpolate）上采样到与原始特征图相同的分辨率。

5. **特征重组**: 将上采样后的融合特征与原始特征图相加, 以增强特征表示。

6. **解码器**: 使用解码器块 (decoder和decoder\_gps) 对融合后的特征进行进一步处理, 并逐步恢复特征图的分辨率。

7. **边缘预测**: Edge\_Module用于预测图像的边缘, 这有助于提高道路提取的精度。

8. **投影头**: ProjectionHead用于将特征映射到新的特征空间, 以便进行对比学习。

通过这种方式, SegTrans模型能够有效地结合来自不同源的信息, 提高道路提取任务的性能。这种融合策略允许模型利用GPS数据中的位置信息和遥感图像中的视觉信息, 从而提供更准确的道路网络预测。

你

关于 特征 提取的部分, 两个 resnet 编码器 分别在代码中什么地方体现

Copilot

在代码中, 两个ResNet编码器分别体现在以下部分:

#### 1. 遥感图像的ResNet编码器:

- self.firstconv、self.firstbn、self.firstrelu和self.firstmaxpool是对应于遥感图像的第一层卷积、批归一化、激活函数和最大池化层。
- self.encoder1、self.encoder2、self.encoder3和self.encoder4是遥感图像特征提取的四个编码器层。

#### 2. GPS特征的ResNet编码器:

- self.firstconv\_gpscnn、self.firstbn\_gpscnn、self.firstrelu\_gpscnn和self.firstmaxpool\_gpscnn是对应于GPS特征的第一层卷积、批归一化、激活函数和最大池化层。
- self.encoder1\_gpscnn、self.encoder2\_gpscnn、self.encoder3\_gpscnn和self.encoder4\_gpscnn是GPS特征提取的四个编码器层。

这两组编码器分别处理遥感图像和GPS数据, 然后通过SAFM模块进行特征融合, 以实现双通道信息的有效结合。这种结构允许模型独立学习每种类型的特征, 然后通过注意力机制来整合这些特征, 从而提高道路提取的准确性。