Segmetation

陈乐偲

1.摘要

基于pytorch实现几个简单的语义分割模型,并做相关实验

- fcn
- segnet
- linknet
- unet
- pspnet
- icnet

数据集采用cityscape，只对道路和非道路进行二分类/分割

2.代码

2.0 block

定义计算机视觉中常用的一些模块，如CBR单元： Seqential(Conv, Batchnorm, Relu)

- CBR/CR/CB... 利用卷积进行下采样
- DCBR 利用反卷积等进行上采样

```
class CBR(nn.Module):
    def __init__(self,in_channels,out_channels,k_size,stride,padding,inplace=True):
        super().__init__()
        in_channels = int(in_channels)
        out_channels = int(out_channels)
        conv = nn.Conv2d(in_channels, out_channels, k_size, stride=stride, padding=padding)
        self.cbr = nn.Sequential(conv, nn.BatchNorm2d(out_channels), nn.ReLU(inplace=inplace))

    def forward(self,x):
        return self.cbr(x)
```

```
class CR(nn.Module):
    def __init__(self,in_channels,out_channels,k_size,stride,padding,inplace=True):
        super().__init__()
        in_channels = int(in_channels)
        out_channels = int(out_channels)
        conv = nn.Conv2d(in_channels, out_channels, k_size, stride=stride, padding=padding)
        self.cr = nn.Sequential(conv, nn.ReLU(inplace=inplace))

    def forward(self,x):
        return self.cr(x)

class CB(nn.Module):
    def __init__(self,in_channels,out_channels,k_size,stride,padding):
        super().__init__()
        in_channels = int(in_channels)
        out_channels = int(out_channels)
        conv = nn.Conv2d(in_channels,out_channels,k_size,stride,padding)
        self.cb = nn.Sequential(conv, nn.BatchNorm2d(out_channels))
    def forward(self,x):
        return self.cb(x)

class DCBR(nn.Module):
    def __init__(self, in_channels, out_channels, k_size, stride, padding,inplace=True):
        super().__init__()

        self.dcbr = nn.Sequential(
            nn.ConvTranspose2d(
                int(in_channels),
                int(out_channels),
                kernel_size=k_size,
                padding=padding,
                stride=stride,
            ),
            nn.BatchNorm2d(int(out_channels)),
            nn.ReLU(inplace=inplace),
        )

    def forward(self, inputs):
        outputs = self.dcbr(inputs)
        return outputs
```

2.1 fcn

实现了全卷积网络fcn32，fcn16，fcn8。

fcn32基于vgg16进行特征提取，只不过将全连接层改为全卷积层。

fcn16在fcn32的基础上，多考虑了最后一层的高层特征信息，而fcn8考虑了最后两层的高层特征信息。

```
class FCN32(nn.Module):
    def __init__(self, n_classes=2):
```

```
        super().__init__()
        self.n_classes = n_classes
        #self.loss = functools.partial(cross_entropy2d, size_average=False)

        self.conv_block1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, stride=2, ceil_mode=True),
        )

        self.conv_block2 = nn.Sequential(
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, stride=2, ceil_mode=True),
        )

        self.conv_block3 = nn.Sequential(
            nn.Conv2d(128, 256, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, stride=2, ceil_mode=True),
        )

        self.conv_block4 = nn.Sequential(
            nn.Conv2d(256, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, stride=2, ceil_mode=True),
        )

        self.conv_block5 = nn.Sequential(
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, stride=2, ceil_mode=True),
        )

        self.classifier = nn.Sequential(
            nn.Conv2d(512, 4096, 7, padding=3),
            nn.ReLU(inplace=True),
            nn.Dropout2d(),
            nn.Conv2d(4096, 4096, 1),
            nn.ReLU(inplace=True),
            nn.Dropout2d(),
            nn.Conv2d(4096, self.n_classes, 1),
        )

        self.upsample = nn.UpsamplingBilinear2d(scale_factor=32)
```

```
    def forward(self, x):
        conv1 = self.conv_block1(x)
        conv2 = self.conv_block2(conv1)
        conv3 = self.conv_block3(conv2)
        conv4 = self.conv_block4(conv3)
        conv5 = self.conv_block5(conv4)
        out = self.classifier(conv5)
        out = self.upsample(out)
        return out

    #复制vgg的卷积层参数
    def init_vgg16_params(self):
        vgg16 = models.vgg16(pretrained=True).to(device)
        blocks = [
            self.conv_block1,
            self.conv_block2,
            self.conv_block3,
            self.conv_block4,
            self.conv_block5,
        ]

        #复制vgg16的参数
        ranges = [[0, 4], [5, 9], [10, 16], [17, 23], [24, 29]]
        features = list(vgg16.features.children())

        for idx, conv_block in enumerate(blocks):
            for l1, l2 in zip(features[ranges[idx][0] : ranges[idx][1]], conv_block):
                if isinstance(l1, nn.Conv2d) and isinstance(l2, nn.Conv2d):
                    assert l1.weight.size() == l2.weight.size()
                    assert l1.bias.size() == l2.bias.size()
                    l2.weight.data = l1.weight.data
                    l2.bias.data = l1.bias.data

class FCN16(FCN32):
    def __init__(self, n_classes=2):
        super().__init__()
        self.n_classes = n_classes
        self.score_pool4 = nn.Conv2d(512, self.n_classes, 1)

    def forward(self, x):
        conv1 = self.conv_block1(x)
        conv2 = self.conv_block2(conv1)
        conv3 = self.conv_block3(conv2)
        conv4 = self.conv_block4(conv3)
        conv5 = self.conv_block5(conv4)

        score = self.classifier(conv5)
        score_pool4 = self.score_pool4(conv4)

        #将conv4的信息连结
        score = F.interpolate(score, score_pool4.size()[2:],mode="bilinear",align_corners=True)
        score += score_pool4
        out = F.interpolate(score, x.size()[2:],mode="bilinear",align_corners=True)

        return out

class FCN8(FCN32):
    def __init__(self, n_classes=2):
        super().__init__()
        self.n_classes = n_classes
```

```
        self.score_pool4 = nn.Conv2d(512, self.n_classes, 1)
        self.score_pool3 = nn.Conv2d(256, self.n_classes, 1)

    def forward(self, x):
        conv1 = self.conv_block1(x)
        conv2 = self.conv_block2(conv1)
        conv3 = self.conv_block3(conv2)
        conv4 = self.conv_block4(conv3)
        conv5 = self.conv_block5(conv4)

        score = self.classifier(conv5)

        score_pool4 = self.score_pool4(conv4)
        score_pool3 = self.score_pool3(conv3)
        score = F.interpolate(score, score_pool4.size()[2:], mode='bilinear',
align_corners=True)
        score += score_pool4
        score = F.interpolate(score, score_pool3.size()[2:], mode='bilinear',
align_corners=True)
        score += score_pool3
        out = F.interpolate(score, x.size()[2:], mode='bilinear', align_corners=True)

        return out
```

2.2 segnet

编码-解码器结构，主要使用了最大值池化作为下采样手段，并且记录下池化的索引值，在上采样的过程中沿着索引的方向进行反池化。

```
class segnetDown2(nn.Module):
    def __init__(self, in_size, out_size):
        super(segnetDown2, self).__init__()
        self.conv1 = CBR(in_size, out_size, 3, 1, 1)
        self.conv2 = CBR(out_size, out_size, 3, 1, 1)
        #在最大值池化中记录index
        self.maxpool_with_argmax = nn.MaxPool2d(2, 2, return_indices=True)

    def forward(self, inputs):
        outputs = self.conv1(inputs)
        outputs = self.conv2(outputs)
        unpooled_shape = outputs.shape
        outputs, indices = self.maxpool_with_argmax(outputs)
        return outputs, indices

class segnetDown3(nn.Module):
    def __init__(self, in_size, out_size):
        super(segnetDown3, self).__init__()
        self.conv1 = CBR(in_size, out_size, 3, 1, 1)
        self.conv2 = CBR(out_size, out_size, 3, 1, 1)
        self.conv3 = CBR(out_size, out_size, 3, 1, 1)
        self.maxpool_with_argmax = nn.MaxPool2d(2, 2, return_indices=True)

    def forward(self, inputs):
```

```
        outputs = self.conv1(inputs)
        outputs = self.conv2(outputs)
        outputs = self.conv3(outputs)
        unpooled_shape = outputs.shape
        outputs, indices = self.maxpool_with_argmax(outputs)
        return outputs, indices


class segnetUp2(nn.Module):
    def __init__(self, in_size, out_size):
        super(segnetUp2, self).__init__()
        self.conv1 = CBR(in_size, in_size, 3, 1, 1)
        self.conv2 = CBR(in_size, out_size, 3, 1, 1)

    def forward(self, inputs, indices):
        #在unpool的过程中传入indice索引
        outputs = F.max_unpool2d(inputs, indices, kernel_size=2, stride=2)
        outputs = self.conv1(outputs)
        outputs = self.conv2(outputs)
        return outputs

class segnetUp3(nn.Module):
    def __init__(self, in_size, out_size):
        super(segnetUp3, self).__init__()
        self.conv1 = CBR(in_size, in_size, 3, 1, 1)
        self.conv2 = CBR(in_size, in_size, 3, 1, 1)
        self.conv3 = CBR(in_size, out_size, 3, 1, 1)

    def forward(self, inputs, indices):
        outputs = F.max_unpool2d(inputs, indices, kernel_size=2, stride=2)
        outputs = self.conv1(outputs)
        outputs = self.conv2(outputs)
        outputs = self.conv3(outputs)
        return outputs

#segnet网络
class SEGNET(nn.Module):
    def __init__(self, n_classes=2, in_channels=3, is_unpooling=True):
        super().__init__()

        self.in_channels = in_channels
        self.is_unpooling = is_unpooling

        #与vgg中的conv大小基本一致
        #使用两个down2，三个down3
        self.down1 = segnetDown2(self.in_channels, 64)
        self.down2 = segnetDown2(64, 128)
        self.down3 = segnetDown3(128, 256)
        self.down4 = segnetDown3(256, 512)
        self.down5 = segnetDown3(512, 512)

        self.up5 = segnetUp3(512, 512)
        self.up4 = segnetUp3(512, 256)
        self.up3 = segnetUp3(256, 128)
        self.up2 = segnetUp2(128, 64)
        self.up1 = segnetUp2(64, n_classes)

    def forward(self, inputs):
        #在下采样时记录最大值池化的index和池化后的大小
        down1, indices_1 = self.down1(inputs)
        down2, indices_2 = self.down2(down1)
```

```
        down3, indices_3 = self.down3(down2)
        down4, indices_4 = self.down4(down3)
        down5, indices_5 = self.down5(down4)

        up5 = self.up5(down5, indices_5)
        up4 = self.up4(up5, indices_4)
        up3 = self.up3(up4, indices_3)
        up2 = self.up2(up3, indices_2)
        up1 = self.up1(up2, indices_1)

        return up1

    #同样用vgg16的参数初始化
    def init_vgg16_params(self):
        vgg16 = vgg16 = models.vgg16(pretrained=True).to(device)
        blocks = [self.down1, self.down2, self.down3, self.down4, self.down5]

        features = list(vgg16.features.children())

        vgg_layers = []
        for _layer in features:
            if isinstance(_layer, nn.Conv2d):
                vgg_layers.append(_layer)

        merged_layers = []
        for idx, conv_block in enumerate(blocks):
            #区分down2还是down3
            if idx < 2:
                units = [conv_block.conv1.cbr,conv_block.conv2.cbr]
            else:
                units = [conv_block.conv1.cbr,conv_block.conv2.cbr,conv_block.conv3.cbr,]
            for _unit in units:
                for _layer in _unit:
                    if isinstance(_layer, nn.Conv2d):
                        merged_layers.append(_layer)

        for l1, l2 in zip(vgg_layers, merged_layers):
            if isinstance(l1, nn.Conv2d) and isinstance(l2, nn.Conv2d):
                l2.weight.data = l1.weight.data
                l2.bias.data = l1.bias.data
```

2.3 unet

unet采用优美的U型结构，本质为解码-编码器的结合，解码器或编码器的每个子单元为两层CBR单元，并且在编码器编码的过程中加入解码器原本信息的跳层连接。

```
class UNET(nn.Module):
    def __init__(self, feature_scale=4, n_classes=2, in_channels=1, is_deconv=True,
 is_batchnorm=True):
        super(unet, self).__init__()
```

```
        self.in_channels = in_channels
        self.is_deconv = is_deconv
        self.is_batchnorm = is_batchnorm
        self.feature_scale = feature_scale

        filters = [64, 128, 256, 512, 1024]
        filters = [int(x / self.feature_scale) for x in filters]

        # 下采样，默认使用BN
        self.conv1 = unetConv2(self.in_channels, filters[0], self.is_batchnorm)
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        self.conv2 = unetConv2(filters[0], filters[1], self.is_batchnorm)
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)

        self.conv3 = unetConv2(filters[1], filters[2], self.is_batchnorm)
        self.maxpool3 = nn.MaxPool2d(kernel_size=2)

        self.conv4 = unetConv2(filters[2], filters[3], self.is_batchnorm)
        self.maxpool4 = nn.MaxPool2d(kernel_size=2)

        #UNet最中间一层
        self.center = unetConv2(filters[3], filters[4], self.is_batchnorm)

        # 上采样，默认采用含有可学习参数的反卷积
        self.up_concat4 = unetUp(filters[4], filters[3], self.is_deconv)
        self.up_concat3 = unetUp(filters[3], filters[2], self.is_deconv)
        self.up_concat2 = unetUp(filters[2], filters[1], self.is_deconv)
        self.up_concat1 = unetUp(filters[1], filters[0], self.is_deconv)

        #用卷积实现
        self.final = nn.Conv2d(filters[0], n_classes, 1)

    def forward(self, inputs):
        conv1 = self.conv1(inputs)
        maxpool1 = self.maxpool1(conv1)

        conv2 = self.conv2(maxpool1)
        maxpool2 = self.maxpool2(conv2)

        conv3 = self.conv3(maxpool2)
        maxpool3 = self.maxpool3(conv3)

        conv4 = self.conv4(maxpool3)
        maxpool4 = self.maxpool4(conv4)

        center = self.center(maxpool4)
        up4 = self.up_concat4(conv4, center)
        up3 = self.up_concat3(conv3, up4)
        up2 = self.up_concat2(conv2, up3)
        up1 = self.up_concat1(conv1, up2)

        final = self.final(up1)

        return final

#unet中的卷积单元，每个卷积单元由两个卷积组成，默认使用BN
class unetConv2(nn.Module):
    def __init__(self, in_size, out_size, is_batchnorm):
        super(unetConv2, self).__init__()
```

```
        if is_batchnorm:
            self.conv1 = nn.Sequential(nn.Conv2d(in_size, out_size, 3, 1, 0),
nn.BatchNorm2d(out_size), nn.ReLU())
            self.conv2 = nn.Sequential(nn.Conv2d(out_size, out_size, 3, 1, 0),
nn.BatchNorm2d(out_size), nn.ReLU())
        else:
            self.conv1 = nn.Sequential(nn.Conv2d(in_size, out_size, 3, 1, 0), nn.ReLU())
            self.conv2 = nn.Sequential(nn.Conv2d(out_size, out_size, 3, 1, 0), nn.ReLU())

    def forward(self, inputs):
        outputs = self.conv1(inputs)
        outputs = self.conv2(outputs)
        return outputs

#unet中的上采样模块，将通道提升为两倍
class unetUp(nn.Module):
    def __init__(self, in_size, out_size, is_deconv):
        super(unetUp, self).__init__()
        self.conv = unetConv2(in_size, out_size, False)
        if is_deconv:
            self.up = nn.ConvTranspose2d(in_size, out_size, kernel_size=2, stride=2)
        else:
            self.up = nn.UpsamplingBilinear2d(scale_factor=2)

    def forward(self, inputs1, inputs2):
        outputs2 = self.up(inputs2)
        offset = outputs2.size()[2] - inputs1.size()[2]
        padding = 2 * [offset // 2, offset // 2]
        outputs1 = F.pad(inputs1, padding)
        return self.conv(torch.cat([outputs1, outputs2], 1))
```

2.4 linknet

类似于resnet，定义了残差卷积模块，在传统的CBR单元的基础上，在CB与R之间插入残差模块。并且同样使用了跳层连接。

```
class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1,inplace=True):
        super().__init__()
        in_channels = int(in_channels)
        out_channels = int(out_channels)

        self.cbr = CBR(in_channels, out_channels, 3, stride, 1)
        self.cb = CB(out_channels, out_channels, 3, 1, 1)
        #在CB R之间引入残差单元
        self.relu = nn.ReLU(inplace=inplace)

    def forward(self, x):
        residual = x
        out = self.cbr(x)
        out = self.cb(out)
        out += residual
        out = self.relu(out)
        return out
```

```
#带下采样的残差卷积
class ResDownBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, inplace=True):
        super().__init__()
        in_channels = int(in_channels)
        out_channels = int(out_channels)

        self.cbr = CBR(in_channels, out_channels, 3, stride, 1)
        self.cb = CB(out_channels, out_channels, 3, 1, 1)
        self.downsample = CBR(in_channels, out_channels,3, stride, 1)
        self.relu = nn.ReLU(inplace=inplace)

    def forward(self, x):
        residual = self.downsample(x)
        out = self.cbr(x)
        out = self.cb(out)

        out += residual
        out = self.relu(out)
        return out

#上采样，通过反卷积操作
class linknetUp(nn.Module):
    def __init__(self, in_channels, out_channels,inplace=True):
        super().__init__()

        in_channels = int(in_channels)
        out_channels = int(out_channels)
        self.cbr1 = CBR(in_channels, out_channels/2, k_size=1, stride=1, padding=0,
inplace=inplace)
        self.dcbr2 = DCBR(out_channels/2, out_channels/2, k_size=2, stride=2, padding=0,
inplace=inplace)
        self.cbr3 = CBR(out_channels/2, out_channels, k_size=1, stride=1, padding=0,
inplace=inplace)

    def forward(self, x):
        x = self.cbr1(x)
        x = self.dcbr2(x)
        x = self.cbr3(x)
        return x

#LINKNET将编码器和解码器相连接
class LINKNET(nn.Module):
    def __init__(self, n_classes=2, in_channels=3):
        super().__init__()

        self.firstconv = CBR(in_channels, 16, 7, stride=2, padding=3)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        #编码器
        self.encoder1 = nn.Sequential(ResBlock(16,16,1,False), ResBlock(16,16,1,False))
        self.encoder2 = nn.Sequential(ResDownBlock(16,32,2,False), ResBlock(32,32,1,False))
        self.encoder3 = nn.Sequential(ResDownBlock(32,64,2,False), ResBlock(64,64,1,False))
        self.encoder4 = nn.Sequential(ResDownBlock(64,128,2,False), ResBlock(128,128,1,False))

        #解码器，将编码器中的信息连接到解码器上

        #跳层连接的过程中，relu的inplace应该设置为False
        self.decoder4 = linknetUp(128, 64, inplace=False)
        self.decoder3 = linknetUp(64, 32, inplace=False)
        self.decoder2 = linknetUp(32, 16, inplace=False)
```

```
        self.decoder1 = linknetUp(16, 16, inplace=False)

        #分割产生器
        self.finaldeconv1 = nn.Sequential(
            nn.ConvTranspose2d(16, 8, 2, 2, 0),
            nn.BatchNorm2d(8),
            nn.ReLU(inplace=True),
        )
        self.finalconv2 = CBR(8,8,3,1,1)
        self.finalconv3 = nn.Conv2d(8, 2, 3, 1, 1)

    def forward(self, x):
        #编码, encoder234逐次下采样
        x = self.firstconv(x)
        x = self.maxpool(x)
        e1 = self.encoder1(x)
        e2 = self.encoder2(e1)
        e3 = self.encoder3(e2)
        e4 = self.encoder4(e3)

        #将四个编码器的结果加到解码器上
        d4 = self.decoder4(e4)

        d4 += e3
        d3 = self.decoder3(d4)
        d3 += e2
        d2 = self.decoder2(d3)
        d2 += e1
        d1 = self.decoder1(d2)

        #产生分割结果
        f1 = self.finaldeconv1(d1)
        f2 = self.finalconv2(f1)
        f3 = self.finalconv3(f2)

        return f3
```

2.5 pspnet

pspnet使用了金字塔池化，将给定的图像卷积后池化为不同大小，再上采样恢复原形状后在通道上拼接。

网络其实得到了小于原图大小的分割图，并且与以前网络逐次卷积后上采样不同，该网络直接进行上采样。

```
#psp中使用到的池化模块定义
class PyramidPool(nn.Module):

    def __init__(self, in_features, out_features, pool_size):
        super().__init__()

        #使用自适应池化变为给定pool_size的大小，再使用双线性插值上采样为原尺寸
        self.features = nn.Sequential(
            nn.AdaptiveAvgPool2d(pool_size),
            nn.Conv2d(in_features, out_features, 1, bias=False),
```

```
            nn.BatchNorm2d(out_features, momentum=0.95),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        size=x.size()
        output=F.interpolate(self.features(x), size[2:], mode='bilinear', align_corners=True)
        return output


class PSPNET(nn.Module):

    def __init__(self, num_classes=2):
        super().__init__()
        #使用resnet作为预训练的骨干网络，使用layer1-4提取特征，大小为原图大小的1/8
        self.resnet = torchvision.models.resnet50(pretrained = True)

        #layer5使用金字塔池化并拼接
        self.layer5a = PyramidPool(2048, 512, 1)
        self.layer5b = PyramidPool(2048, 512, 2)
        self.layer5c = PyramidPool(2048, 512, 3)
        self.layer5d = PyramidPool(2048, 512, 6)

        self.final = nn.Sequential(
            nn.Conv2d(4096, 512, 3, padding=1, bias=False),
            nn.BatchNorm2d(512, momentum=0.95),
            nn.ReLU(inplace=True),
            nn.Dropout(0.1),
            nn.Conv2d(512, num_classes, 1),
        )


    def forward(self, x):

        size=x.size()
        x = self.resnet.conv1(x)
        x = self.resnet.bn1(x)
        x = self.resnet.relu(x)

        x = self.resnet.layer1(x)
        x = self.resnet.layer2(x)
        x = self.resnet.layer3(x)
        x = self.resnet.layer4(x)

        #得到每个子layer具有512个通道
        f5a = self.layer5a(x)
        f5b = self.layer5b(x)
        f5c = self.layer5c(x)
        f5d = self.layer5d(x)

        #在通道上拼接
        y = torch.cat([x,f5a,f5b,f5c,f5d], 1)
        x = self.final(y)

        #直接通过双线性插值恢复为原大小
        x = F.interpolate(x,size[2:],mode='bilinear',align_corners=True)

        return x
```

2.6 icnet

icnet，图像级联网络，为了提高分割的实时性而提出。

用不同分辨率的图片同时进行训练，得到不同分辨率的分割结果。

在训练高分辨率图片的分割的时候，利用CCF单元融合了低分辨率图片特征与高分辨率图片特征。

得到了比原图小得多的图片，但同样直接使用双线性插值进行上采样。

```
class CBR(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3, stride=1, padding=1):
        super().__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
        self.bn = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(True)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.relu(x)
        return x

#PSPNET中的金字塔池化，默认参数和PSPNET相同，为[1,2,3,6]
class PyramidPooling(nn.Module):
    def __init__(self, pyramids=[1,2,3,6]):
        super().__init__()
        self.pyramids = pyramids

    def forward(self, input):
        feat = input
        height, width = input.shape[2:]
        for bin_size in self.pyramids:
            x = F.adaptive_avg_pool2d(input, output_size=bin_size)
            x = F.interpolate(x, size=(height, width), mode='bilinear', align_corners=True)
            feat  = feat + x
        return feat

# CCF单元，级联上下层特征，将底层特征通过分类器输出辅助分类结果，并将低层与高层特征进行结合
class CCF(nn.Module):

    def __init__(self, low_channels, high_channels, out_channels, nclass):
        super().__init__()
        #使用dilation进行空洞卷积
        self.conv_low = nn.Sequential(
            nn.Conv2d(low_channels, out_channels, 3, padding=2, dilation=2, bias=False),
            nn.BatchNorm2d(out_channels)
        )
        self.conv_high = nn.Sequential(
            nn.Conv2d(high_channels, out_channels, 1, bias=False),
            nn.BatchNorm2d(out_channels)
        )
        #分类器
        self.conv_low_cls = nn.Conv2d(out_channels, nclass, 1, bias=False)

    def forward(self, x_low, x_high):
        #将x_low插值经过上采样插值为x_high的大小
        x_low = F.interpolate(x_low, size=x_high.size()[2:], mode='bilinear',
align_corners=True)
```

```
        x_low = self.conv_low(x_low)
        x_high = self.conv_high(x_high)
        x = x_low + x_high
        x = F.relu(x, inplace=True)
        x_low_cls = self.conv_low_cls(x_low)

        return x, x_low_cls

class ICHead(nn.Module):
    def __init__(self, nclass):
        super().__init__()
        self.cff_12 = CCF(128, 64, 128, nclass)
        self.cff_24 = CCF(2048, 512, 128, nclass)
        self.conv_cls = nn.Conv2d(128, nclass, 1, bias=False)

    def forward(self, x_sub1, x_sub2, x_sub4):
        outputs = list()
        x_cff_24, x_24_cls = self.cff_24(x_sub4, x_sub2)
        outputs.append(x_24_cls)
        x_cff_12, x_12_cls = self.cff_12(x_cff_24, x_sub1)
        outputs.append(x_12_cls)

        up_x2 = F.interpolate(x_cff_12, scale_factor=2, mode='bilinear', align_corners=True)
        up_x2 = self.conv_cls(up_x2)
        outputs.append(up_x2)
        up_x8 = F.interpolate(up_x2, scale_factor=4, mode='bilinear', align_corners=True)
        outputs.append(up_x8)
        # 输出不同大小的图片，1 -> 1/4 -> 1/8 -> 1/16
        outputs.reverse()

        return outputs

class ICNET(nn.Module):
    #使用resnet作为预训练的模型
    def __init__(self, nclass = 2):
        super().__init__()
        self.backbone = models.resnet50(pretrained = True)
        #使用步长为2的卷积
        self.conv_sub1 = nn.Sequential(CBR(3, 32, 3, 2),CBR(32, 32, 3, 2),CBR(32, 64, 3, 2))
        self.ppm = PyramidPooling()
        self.head = ICHead(nclass)

    #输入图片x，使用backbone提取各层特征
    def getFeatures(self,x):
        x = self.backbone.conv1(x)
        x = self.backbone.bn1(x)
        x = self.backbone.relu(x)
        x = self.backbone.maxpool(x)
        c1 = self.backbone.layer1(x)
        c2 = self.backbone.layer2(c1)
        c3 = self.backbone.layer3(c2)
        c4 = self.backbone.layer4(c3)
        return c1, c2, c3, c4

    def forward(self, x):
        size = x.size()[2:]
        #原大小,提取特征为1/8 eg.[32,64]
        x_sub1 = self.conv_sub1(x)

        #1/2图的特征，提取特征为1/16 eg.[16,32]
        x_sub2 = F.interpolate(x, scale_factor=0.5, mode='bilinear', align_corners=True)
```

```
        _, x_sub2, _, _ = self.getFeatures(x_sub2)

        #1/4图，并加入图像金字塔池化
        x_sub4 = F.interpolate(x, scale_factor=0.25, mode='bilinear', align_corners=True)
        _, _, _, x_sub4 = self.getFeatures(x_sub4)
        x_sub4 = self.ppm(x_sub4)
        #通过IChead分类头输出结果

        outputs = self.head(x_sub1, x_sub2, x_sub4)

        for i in range(len(outputs)):
            outputs[i] = F.interpolate(outputs[i], size=size, mode='bilinear',
align_corners=True)

        return outputs

#定义级联损失，为不同分辨率图片损失之和
class ICLoss(nn.Module):
    def __init__(self,weights=[1.0,0.4,0.4,0.2]):
        super().__init__()
        self.critirion = nn.CrossEntropyLoss()
        self.weights =  weights

    def forward(self,outputs,y):
        loss = 0
        for out,w in zip(outputs,self.weights):
            loss += w * self.critirion(out, y)
        return loss
```

2.7 metric

评价指标选取了fps和miou，（同时输出每个像素准确率也可作为辅助指标）

```
def getMetrics(self,dataloader):
        correct = 0
        tot_pixel = 0
        tot_time = 0
        tot_num = 0

        tot_intersect = 0
        tot_union = 0

        for i,data in enumerate(dataloader):
            #选取部分计算评价指标
            if i > 5:
                break
            x,y = data
            B = x.shape[0]
            x = x.to(device)
            y = y.to(device)

            tot_num += 1

            start_time = time.time()
            #ICNET取第一个元素为输出
```

```
            out = self.model(x)
            if isinstance(out,list):
                out = out[0]
            end_time = time.time()
            tot_time += end_time - start_time

            pred = torch.argmax(out,dim=1)

            correct += torch.sum(pred ==  y)
            tot_pixel += torch.sum((y>=0))

            pred = pred.bool()
            y = y.bool()

            tot_intersect += torch.sum(pred & y)
            tot_union += torch.sum(pred | y)

        acc = (correct / tot_pixel).item()
        miou = (tot_intersect / tot_union).item()
        fps = (tot_num*BATCH) / tot_time

        return acc,miou,fps
```

3.对比

超参数	数值
batch size	8
epoch	10
lr	1e-4

模型	MIOU	FPS	总大小(M)	准确率
FCN32	-	-	-	-
FCN16	-	-	-	-
FCN8	80.97	1913.68	1108.24	92.97
FCN8-VGG	87.63	1857.00	1108.24	95.9
SEGNET	82.22	835.10	89192777.82	92.98
SEGNET-VGG	**89.36**	177.72?	89192777.82	96.37

模型	MIOU	FPS	总大小(M)	准确率
UNET	78.50	1208.15	445.41	92.31
LINKNET	74.59	795.27	159.95	90.6
PSPNET	**89.49**	427.43	3030.57	96.48
ICNET	85.13	221.57	137438962078.62	93.6

foggy数据集下，

模型	MIOU	FPS	总大小(M)	准确率
FCN32	-	-	-	-
FCN16	-	-	-	-
FCN8	-	-	-	-
FCN8-VGG	**90.46**	2155	-	96.7
SEGNET	-	-	-	-
SEGNET-VGG	84.78	1006.10	-	94.68
UNET	77.23	1033.62	-	91.48
LINKNET	80.5	631.16	-	92.7
PSPNET	80.82	368.27	-	93.68
ICNET	86.30	185.51	-	95.26

4.实验

代码可实现的实验

- 不同网络结构的各方面对比（参数量、miou、fps）
- BN层的影响
- 不同上采样方式的影响（插值、反卷积）
- 普通卷积和空洞卷积的影响
- 迁移预训练模型参数的影响
- 不同骨干网络的影响（resnet，vgg）
- 残差卷积和普通卷积的影响
- 是否跳层连接的影响
- ICNET中图片级联的影响

5.样例

使用FCN8取部分训练集（BATCH=8，取训练集前50BATCH，并将图片尺寸缩减为256x512）进行训练，训练10轮的结果如下：（上一行为输出，下一行为真实标记，白色为道路，黑色为其他）

取数据集的1/5，每轮训练时间大约1min，总计不到10min

运行结果示例

参考

[1] https://github.com/meetshah1995/pytorch-semseg/

附录

torchsummary的部分,展示一些网络结构（应该没什么用）

```
FCN8

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 256, 512]           1,792
              ReLU-2         [-1, 64, 256, 512]               0
            Conv2d-3         [-1, 64, 256, 512]          36,928
              ReLU-4         [-1, 64, 256, 512]               0
         MaxPool2d-5         [-1, 64, 128, 256]               0
            Conv2d-6        [-1, 128, 128, 256]          73,856
              ReLU-7        [-1, 128, 128, 256]               0
            Conv2d-8        [-1, 128, 128, 256]         147,584
              ReLU-9        [-1, 128, 128, 256]               0
        MaxPool2d-10         [-1, 128, 64, 128]               0
           Conv2d-11         [-1, 256, 64, 128]         295,168
             ReLU-12         [-1, 256, 64, 128]               0
           Conv2d-13         [-1, 256, 64, 128]         590,080
             ReLU-14         [-1, 256, 64, 128]               0
           Conv2d-15         [-1, 256, 64, 128]         590,080
             ReLU-16         [-1, 256, 64, 128]               0
        MaxPool2d-17          [-1, 256, 32, 64]               0
           Conv2d-18          [-1, 512, 32, 64]       1,180,160
             ReLU-19          [-1, 512, 32, 64]               0
```

```
           Conv2d-20          [-1, 512, 32, 64]       2,359,808
             ReLU-21          [-1, 512, 32, 64]               0
           Conv2d-22          [-1, 512, 32, 64]       2,359,808
             ReLU-23          [-1, 512, 32, 64]               0
        MaxPool2d-24          [-1, 512, 16, 32]               0
           Conv2d-25          [-1, 512, 16, 32]       2,359,808
             ReLU-26          [-1, 512, 16, 32]               0
           Conv2d-27          [-1, 512, 16, 32]       2,359,808
             ReLU-28          [-1, 512, 16, 32]               0
           Conv2d-29          [-1, 512, 16, 32]       2,359,808
             ReLU-30          [-1, 512, 16, 32]               0
        MaxPool2d-31           [-1, 512, 8, 16]               0
           Conv2d-32          [-1, 4096, 8, 16]     102,764,544
             ReLU-33          [-1, 4096, 8, 16]               0
        Dropout2d-34          [-1, 4096, 8, 16]               0
           Conv2d-35          [-1, 4096, 8, 16]      16,781,312
             ReLU-36          [-1, 4096, 8, 16]               0
        Dropout2d-37          [-1, 4096, 8, 16]               0
           Conv2d-38             [-1, 2, 8, 16]           8,194
           Conv2d-39            [-1, 2, 16, 32]           1,026
           Conv2d-40            [-1, 2, 32, 64]             514
================================================================
Total params: 134,270,278
Trainable params: 134,270,278
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.50
Forward/backward pass size (MB): 594.54
Params size (MB): 512.20
Estimated Total Size (MB): 1108.24
```

```
FCN16

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 256, 512]           1,792
              ReLU-2         [-1, 64, 256, 512]               0
            Conv2d-3         [-1, 64, 256, 512]          36,928
              ReLU-4         [-1, 64, 256, 512]               0
         MaxPool2d-5         [-1, 64, 128, 256]               0
            Conv2d-6        [-1, 128, 128, 256]          73,856
              ReLU-7        [-1, 128, 128, 256]               0
            Conv2d-8        [-1, 128, 128, 256]         147,584
              ReLU-9        [-1, 128, 128, 256]               0
        MaxPool2d-10         [-1, 128, 64, 128]               0
           Conv2d-11         [-1, 256, 64, 128]         295,168
             ReLU-12         [-1, 256, 64, 128]               0
           Conv2d-13         [-1, 256, 64, 128]         590,080
             ReLU-14         [-1, 256, 64, 128]               0
           Conv2d-15         [-1, 256, 64, 128]         590,080
             ReLU-16         [-1, 256, 64, 128]               0
        MaxPool2d-17          [-1, 256, 32, 64]               0
           Conv2d-18          [-1, 512, 32, 64]       1,180,160
             ReLU-19          [-1, 512, 32, 64]               0
           Conv2d-20          [-1, 512, 32, 64]       2,359,808
             ReLU-21          [-1, 512, 32, 64]               0
           Conv2d-22          [-1, 512, 32, 64]       2,359,808
```

```
             ReLU-23          [-1, 512, 32, 64]               0
        MaxPool2d-24          [-1, 512, 16, 32]               0
           Conv2d-25          [-1, 512, 16, 32]       2,359,808
             ReLU-26          [-1, 512, 16, 32]               0
           Conv2d-27          [-1, 512, 16, 32]       2,359,808
             ReLU-28          [-1, 512, 16, 32]               0
           Conv2d-29          [-1, 512, 16, 32]       2,359,808
             ReLU-30          [-1, 512, 16, 32]               0
        MaxPool2d-31           [-1, 512, 8, 16]               0
           Conv2d-32          [-1, 4096, 8, 16]     102,764,544
             ReLU-33          [-1, 4096, 8, 16]               0
        Dropout2d-34          [-1, 4096, 8, 16]               0
           Conv2d-35          [-1, 4096, 8, 16]      16,781,312
             ReLU-36          [-1, 4096, 8, 16]               0
        Dropout2d-37          [-1, 4096, 8, 16]               0
           Conv2d-38             [-1, 2, 8, 16]           8,194
           Conv2d-39            [-1, 2, 16, 32]           1,026
================================================================
Total params: 134,269,764
Trainable params: 134,269,764
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.50
Forward/backward pass size (MB): 594.51
Params size (MB): 512.20
Estimated Total Size (MB): 1108.21
----------------------------------------------------------------
```

```
FCN8

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 256, 512]           1,792
              ReLU-2         [-1, 64, 256, 512]               0
            Conv2d-3         [-1, 64, 256, 512]          36,928
              ReLU-4         [-1, 64, 256, 512]               0
         MaxPool2d-5         [-1, 64, 128, 256]               0
            Conv2d-6        [-1, 128, 128, 256]          73,856
              ReLU-7        [-1, 128, 128, 256]               0
            Conv2d-8        [-1, 128, 128, 256]         147,584
              ReLU-9        [-1, 128, 128, 256]               0
        MaxPool2d-10         [-1, 128, 64, 128]               0
           Conv2d-11         [-1, 256, 64, 128]         295,168
             ReLU-12         [-1, 256, 64, 128]               0
           Conv2d-13         [-1, 256, 64, 128]         590,080
             ReLU-14         [-1, 256, 64, 128]               0
           Conv2d-15         [-1, 256, 64, 128]         590,080
             ReLU-16         [-1, 256, 64, 128]               0
        MaxPool2d-17          [-1, 256, 32, 64]               0
           Conv2d-18          [-1, 512, 32, 64]       1,180,160
             ReLU-19          [-1, 512, 32, 64]               0
           Conv2d-20          [-1, 512, 32, 64]       2,359,808
             ReLU-21          [-1, 512, 32, 64]               0
           Conv2d-22          [-1, 512, 32, 64]       2,359,808
             ReLU-23          [-1, 512, 32, 64]               0
        MaxPool2d-24          [-1, 512, 16, 32]               0
           Conv2d-25          [-1, 512, 16, 32]       2,359,808
```

```
              ReLU-26          [-1, 512, 16, 32]               0
            Conv2d-27          [-1, 512, 16, 32]       2,359,808
              ReLU-28          [-1, 512, 16, 32]               0
            Conv2d-29          [-1, 512, 16, 32]       2,359,808
              ReLU-30          [-1, 512, 16, 32]               0
         MaxPool2d-31           [-1, 512, 8, 16]               0
            Conv2d-32          [-1, 4096, 8, 16]     102,764,544
              ReLU-33          [-1, 4096, 8, 16]               0
         Dropout2d-34          [-1, 4096, 8, 16]               0
            Conv2d-35          [-1, 4096, 8, 16]      16,781,312
              ReLU-36          [-1, 4096, 8, 16]               0
         Dropout2d-37          [-1, 4096, 8, 16]               0
            Conv2d-38             [-1, 2, 8, 16]           8,194
UpsamplingBilinear2d-39          [-1, 2, 256, 512]               0
================================================================
Total params: 134,268,738
Trainable params: 134,268,738
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.50
Forward/backward pass size (MB): 596.50
Params size (MB): 512.19
Estimated Total Size (MB): 1110.20
----------------------------------------------------------------
```