Computer Vision보고서 12주차

임형찬, 202102559, 인공지능학과

[과제]

- 구현코드 및 코드설명

```python
# Pre-Activation 사용
class convolution_block(nn.Module):
    def __init__(self, input, out):
        super(convolution_block,self).__init__()
        self.conv = nn.Sequential(
            nn.BatchNorm2d(input),
            nn.Mish(),
            nn.Conv2d(input, out, kernel_size=3,stride=1,padding=1,bias=True),

            nn.BatchNorm2d(out),
            nn.Mish(),
            nn.Conv2d(out, out, kernel_size=3,stride=1,padding=1,bias=True),
        )

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

# Convolution Block과 대칭되어야 해서 Post-Activation 사용
class upsampling(nn.Module):
    def __init__(self, input, out):
        super(upsampling, self).__init__()
        self.up = nn.Sequential(
            nn.Upsample(scale_factor=2),
            nn.Conv2d(input, out, kernel_size=3, stride=1, padding=1, bias=True),
            nn.BatchNorm2d(out),
            nn.Mish()
        )
    def forward(self, x):
        x = self.up(x)
        return x
```

사실 이번 KSC에 제출할 아이디어로 잡았었는데 선행연구로 Attention with Unet이 이미 존재하여 해당 모델을 기반으로 구현을 진행했습니다.

자세한 설명은 주석을 참고해주세요.

Conv Block의 경우, Unet의 반복되는 부분을 대체하기 위한 부분입니다.

Upsampling의 경우, 2배로 증가하도록 하기 위해 scale factor를 2로 잡았습니다.

```python
class AttentionBlock(nn.Module):
    def __init__(self, global_channels, local_channels, channels):
        super(AttentionBlock, self).__init__()

        # Global Attention에 대한 가중치 행렬 W_g
        self.global_conv = nn.Sequential(
            nn.Conv2d(global_channels, channels, kernel_size=1, stride=1, padding=0, bias=True),
            nn.BatchNorm2d(channels)
        )

        # Local Attention에 대한 가중치 행렬 W_x
        self.local_conv = nn.Sequential(
            nn.Conv2d(local_channels, channels, kernel_size=1, stride=1, padding=0, bias=True),
            nn.BatchNorm2d(channels)
        )

        # 어텐션을 제어하는 파라미터 alpha
        self.alpha = nn.Sequential(
            nn.Conv2d(channels, 1, kernel_size=1, stride=1, padding=0, bias=True),
            nn.BatchNorm2d(1),
            nn.Sigmoid()
        )

        # ReLU 활성화 함수
        self.relu = nn.ReLU(inplace=True)

    def forward(self, global_feat_map, local_feat_map):
        # Global Attention과 Local Attention에 대한 각각의 가중치 행렬 계산
        global_out = self.global_conv(global_feat_map)
        local_out = self.local_conv(local_feat_map)

        # 두 결과를 더하고 ReLU 함수를 적용하여 어텐션을 제어하는 파라미터 alpha 계산
        alpha = self.relu(global_out + local_out)
        # alpha를 이용하여 Local Attention에 가중치를 적용하여 출력
        alpha = self.alpha(alpha)
        return local_feat_map * alpha
```

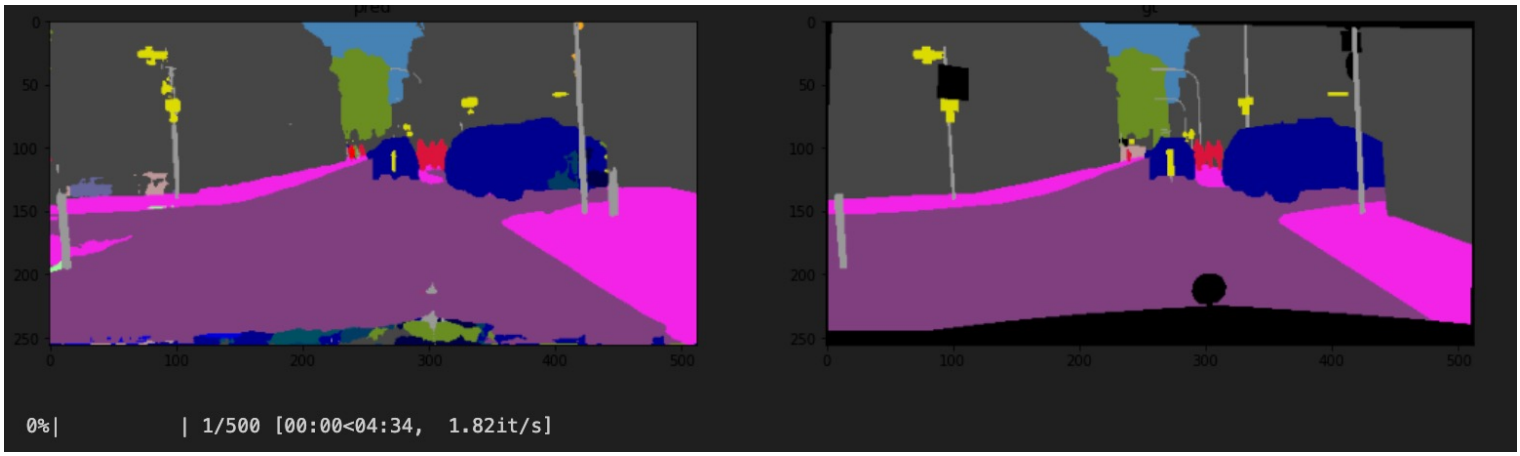
Attention Block의 경우, global과 local로 나누었는데
이는 CNN의 특징상 층이 늘어날수록 global feature에 집중되기 때문에 이를 alpha변수로 조정하기 위함입니다.

```python
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.Maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.Conv1 = convolution_block(3, 64)
        self.Conv2 = convolution_block(64, 128)
        self.Conv3 = convolution_block(128, 256)
        self.Conv4 = convolution_block(256, 512)
        self.Conv5 = convolution_block(512, 1024)

    def forward(self, x):
        x1 = self.Conv1(x)
        x2 = self.Maxpool(x1)
        x2 = self.Conv2(x2)
        x3 = self.Maxpool(x2)
        x3 = self.Conv3(x3)
        x4 = self.Maxpool(x3)
        x4 = self.Conv4(x4)
        x5 = self.Maxpool(x4)
        x5 = self.Conv5(x5)
        return x5, x4, x3, x2, x1
```

Encoder의 경우, 기존 Unet과 동일합니다.

```python
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.Upsampling1 = upsampling(1024, 512)
        self.Att1 = AttentionBlock(512, 512, 256)
        self.Conv1 = convolution_block(1024, 512)

        self.Upsampling2 = upsampling(512, 256)
        self.Att2 = AttentionBlock(256, 256, 128)
        self.Conv2 = convolution_block(512, 256)

        self.Upsampling3 = upsampling(256, 128)
        self.Att3 = AttentionBlock(128, 128, 64)
        self.Conv3 = convolution_block(256, 128)

        self.Upsampling4 = upsampling(128, 64)
        self.Att4 = AttentionBlock(64, 64, 32)
        self.Conv4 = convolution_block(128, 64)

        self.Conv = nn.Conv2d(64, 19, kernel_size=1, stride=1, padding=0)
    def forward(self, x5, x4, x3, x2, x1):
        d5 = self.Upsampling1(x5)
        x4 = self.Att1(d5, x4)
        d5 = torch.cat((x4, d5), dim=1)
        d5 = self.Conv1(d5)

        d4 = self.Upsampling2(d5)
        x3 = self.Att2(d4, x3)
        d4 = torch.cat((x3, d4), dim=1)
        d4 = self.Conv2(d4)

        d3 = self.Upsampling3(d4)
        x2 = self.Att3(d3, x2)
        d3 = torch.cat((x2, d3), dim=1)
        d3 = self.Conv3(d3)

        d2 = self.Upsampling4(d3)
        x1 = self.Att4(d2, x1)
        d2 = torch.cat((x1, d2), dim=1)
        d2 = self.Conv4(d2)

        d1 = self.Conv(d2)

        return d1


class UNet_with_Attention(nn.Module):
    def __init__(self, input=3, classes=19):
        super(UNet_with_Attention, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

    def forward(self, x):
        x5, x4, x3, x2, x1 = self.encoder(x)
        output = self.decoder(x5, x4, x3, x2, x1)
        return output
```

Decoder의 경우, Upsampling한 후 AttentionBlock을 추가한다는 점을 제외하면 기존 Unet과 동일합니다.



89	0.0625061652352733	52.435897966761026
90	0.05404243642284024	51.805676684998936
91	0.050385969420594555	51.18181592027635
92	0.049794981987165504	51.925396911386756
93	0.051814384187661836	51.31090809991971
94	0.054354806520765825	32.99319160175301
95	0.1779852427381982	50.571309713216536
96	0.07943747127528793	51.16314882993518
97	0.05623862637026656	52.189258820817706
98	0.049621909130765224	52.629548515397886
99	0.04704979187258149	51.667091385934164
100	0.04732232652003727	51.207748653300676