<모델 구축>

Backbone, convolution조작, loss function, deep supervision..??

[efficientnet-b3]

입력 크기와 채널수가 정해져 있음.

컨볼루션할 때 텐서 크기랑 입력 채널 수 못 맞춤.

입력 크기가 작아서 pooling 연산 후에 너무 작게 계산돼서 오류.

[Resnet-50]

```
class ResNetBackbone(nn.Module):
    def __init__(self):
        super(ResNetBackbone, self).__init__()

        # ResNet-50을 백본으로 사용
        resnet = models.resnet50(weights='ResNet50_Weights.DEFAULT')

        # ResNet의 마지막 두 레이어를 제거하여 feature map을 얻습니다.
        self.features = nn.Sequential(*list(resnet.children())[:-2])

        self.upsample = nn.Upsample(size=(224, 224), mode='bilinear', align_corners=True)
        self.res_down1 = double_conv(2048,64)

    def forward(self, x):
        features = self.features(x)
        features = self.upsample(features)
        features = self.res_down1(features)
        return features
```

resnet50 => 사전 학습된 가중치를 가져옴

마지막 두 개의 레이어를 제외한 나머지 레이어를 feature맵(학습된 특징 결과)으로 만듦

feature맵을 (224,224)크기로 업샘플링

feature맵의 채널 수를 줄여서 최종 feature 맵 반환

resnet을 백본으로 가져와서 채널 수 증가 (인코딩)

채널 수 감소 (디코딩)

```python
class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()
        self.backbone = ResNetBackbone()

        self.dconv_down1 = double_conv(64, 64)
        self.dconv_down2 = double_conv(64, 128)
        self.dconv_down3 = double_conv(128, 256)
        self.dconv_down4 = double_conv(256, 512)

        self.maxpool = nn.MaxPool2d(2)
        self.upsample = nn.Upsample(scale_factor=2, mode='biline

        self.dconv_up3 = double_conv(256 + 512, 256)
        self.dconv_up2 = double_conv(128 + 256, 128)
        self.dconv_up1 = double_conv(128 + 64, 64)

        self.conv_last = nn.Conv2d(64, 1, 1)
```

여기서 컨볼루션할 때 사이사이에 배치정규화, 드롭 아웃(20%)해봄

배치정규화 => 레이어마다 입력이 달라지면서 학습이 어려워질 수 있는데, 그 변화를 감소시키고, 정규화 함으로써 학습률을 낮추니까 학습 시간도 단축

```python
def double_conv(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, 3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True),
        nn.Dropout2d(p=0.2),
        nn.Conv2d(out_channels, out_channels, 3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
    )
```

Deep supervision(최종 출력만이 아닌 중간 레이어의 출력도 손실 계산) 앙상블

여러 레이어에서 출력값 발생하고 타겟 텐서(실제값)하고 비교해야되는데, 자꾸 크기가 안맞고 차원이 안맞고 torchvision resize함수로 조정하는데 계속 오류.

<Target size (torch.Size([8, 224, 224])) must be the same as input size (torch.Size([1, 224, 224]))>

Loss function

실제값과 예측값 차이가 났을 때 얼마인지 계산해주는 함수

입력값 -> 순전파-> 예측값 ⇔ 실제값과 차이 => 옵티마이저로 역전파 진행 매개변수 업데이트

수치형 데이터 => 회귀 분석 (MSE) Mean Square || 범주형 데이터 =>분류

Cross entropy = 클래스 불균형하면 적절하지 않을 수 있음

Focal loss = 잘 분류한 샘플에는 loss값을 줄여주고, 잘못 분류된 샘플에는 높은 가중치를 부여

불균형한 데이터셋에 좋음

Dice Loss = 예측된 마스크와 실제 마스크의 유사성을 측정, 위성 사진 같은 불균형한 클래스(배경, 건물)

Lovasz loss = 예측된 마스크와 실제 마스크 간의 거리를 측정해서 학습, 경계 픽셀 처리에 좋음

기존 BCE에 Focal loss 추가

alpha= 클래스 간 균형 조절 가중치(양성 값), gamma= 강조 강도, 즉 어려운 샘플에 대한 가중치 조절 파라미터(영향력)

입력과 정답 레이블(target)을 받아서 loss계산

BCE를 계산 후에, 확률 값(pt)를 계산하여 focal loss계산(이진 엔트로피 손실에 알파와 베타를 곱하고, 확률 값을 거듭제곱) 여기서 alpha 와 gamma 조절

Focal을 손실함수로 사용하여 모든 손실 값을 평균해서 학습 중에 손실 계산

```
class FocalLoss(nn.Module):
    def __init__(self, alpha=0.5, gamma=2):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma

    def forward(self, inputs, targets):
        bce_loss = nn.BCEWithLogitsLoss(reduction='none')(inputs, targets)

        pt = torch.exp(-bce_loss)  # 확률 값 계산
        focal_loss = self.alpha * (1 - pt) ** self.gamma * bce_loss

        return focal_loss.mean()
```