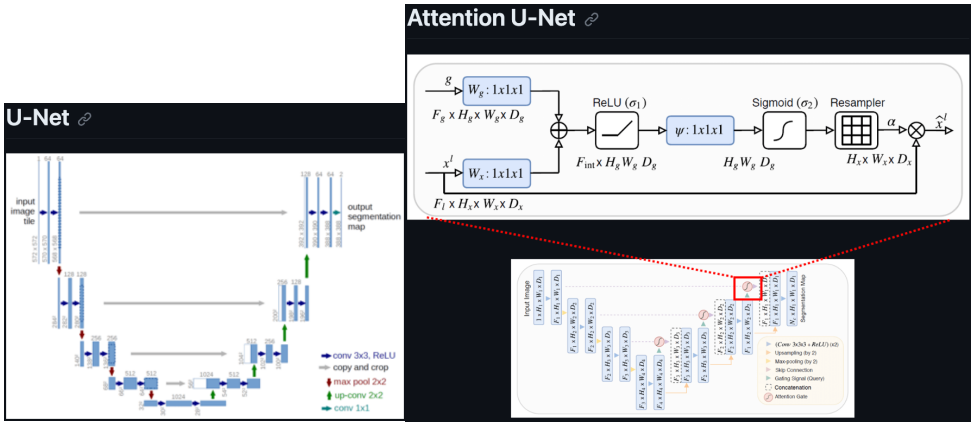
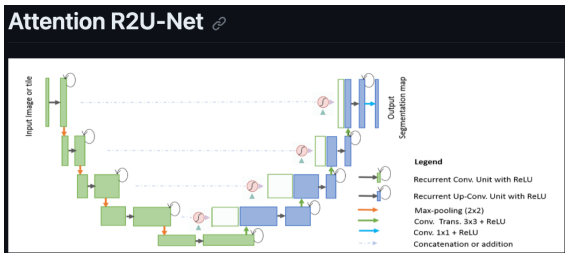
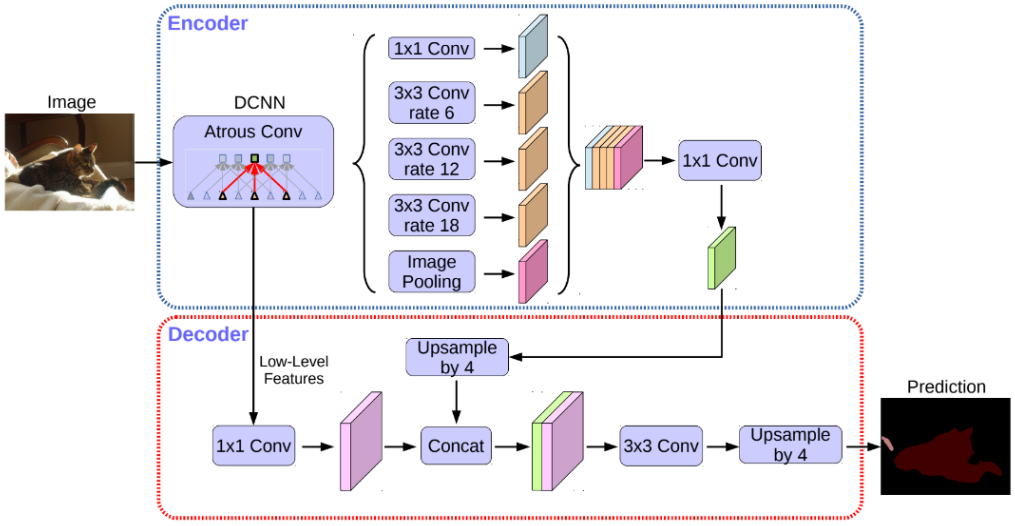
[AI00]_202102559_임형찬_HW08

📌 모델네트워크





! Best Model – DeepLabV3+



📌 기존성능보다 향상시킨방법(전처리,후처리등등)

❗[방법1.]: Data Augmentation 및 Batch_size조정

- Resize: CNN의 경우 input size를 맞춰주지않으면 학습이 불가능하기때문
- RandomHorizontalFip: 다른 crop, colorjitter등은 학습에 영향을 줄 수 있어서 하지않음.
- (Crop의 경우, Crack부분이 잘릴수도 있기에 진행X), batch를 줄일수록 성능은 증가함.

❗[방법2.]: Optimizer tuning, Early Stopping

처음사용한 optimizer: SGD, Adam

다만, 현재 많은 Transformer, LLM등에서 사용되는 AdamW를 사용.

추가적으로 learning rate scheduler를 사용.

ReduceLROnPlateau를 사용함(CosineAnealing, Heat, Step, Exp등 다양한 방법이 있지만 가장 정석적인 방법을 채택; 처음에는 좀 더 공격적인 학습으로 local minima에 빠질 위험을 줄이기위해 학습률을 AdamW의 default값인 1e-4보다 10배 높은 1e-3으로 설정. 이후 5 epoch동안 val_loss가 감소하지 않는다면 factor만큼 곱해 learning rate를 떨어뜨린다.)

```
def configure_optimizers(self):
    optimizer = torch.optim.AdamW(self.parameters(), lr=1e-3)
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
                                                           mode='min',  # Loss최소화,최대화 결정
                                                           factor=0.1,  # 학습률 감소율
                                                           patience=5,
                                                           verbose=True,)
    monitor_metric = 'val_loss'
    return {
        'optimizer': optimizer,
        'lr_scheduler': {
            'scheduler': scheduler,
            'monitor': monitor_metric  # Specify the metric you want to monitor
        }
    }
```

```
early_stopping_callback = EarlyStopping(
    monitor='val_loss',
    patience=8,
    verbose=True,
    mode='min'
)

trainer = pl.Trainer(devices=[0],max_epochs=100, logger=wandb_logger, callbacks=[early_stopping_callback])
trainer.fit(model, train_dataloaders=train_dataloader, val_dataloaders=val_dataloader)
```

Cf)처음에는 DiceLoss도 사용해보았으나 성능이 오히려 좋지않았음. (그냥 참고용으로 첨부)

```
def dice_coefficient(predicted, target):
    intersection = torch.sum(predicted * target)
    union = torch.sum(predicted) + torch.sum(target)
    dice = (2.0 * intersection + 1e-5) / (union + 1e-5)
    return dice

def dice_loss(predicted, target):
    return 1 - dice_coefficient(predicted, target)
```

❗[방법3.]: Model Architecture Change

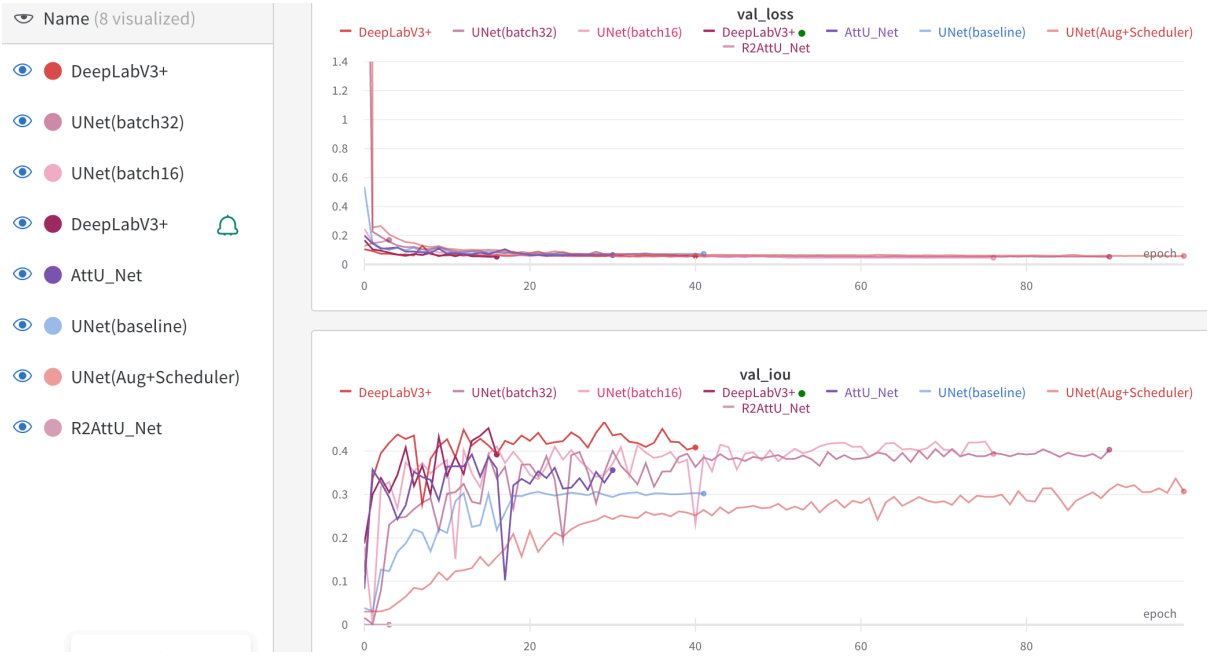
가장 성능을 끌어올리기 확실한 방법은 Model 을 바꾸는 것이다.

처음 사용해본 모델은 UNet 으로 만족할만한 성능을 보였으나

Trend	Dataset	Best Model	Paper	Code	Compare
	ADE20K	ONE-PEACE			See all
	Cityscapes test	InternImage-H			See all
	NYU Depth v2	EMSANet (2x ResNet-34 NBt1D, PanopticNDT version, finetuned)			See all
	ADE20K val	BEiT-3			See all
	Cityscapes val	InternImage-H			See all
	PASCAL Context	PlainSeg (EVA-02-L)			See all
	PASCAL VOC 2012 test	DeepLabv3+ (Xception-65-JFT)			See all
	S3DIS	PonderV2 + SparseUNet			See all
	S3DIS Area5	Swin3D-L			See all
	DensePASS	Trans4PASS+ (multi-scale)			See all

현재 paperswithcode 에 올라와있는 여러 Model 들 중 하나인 DeepLabV3+를 사용해 성능을 최대치로 끌어올리고자 하였다.

Iou 에 smooth 등을 추가하지 않아서 조금 빡빡하게 성능이 측정되었지만 만족할만한 성능을 얻은 것 같다. 아래는 결과이다.



📕 이번 실습에서 Segmentation 에 대해 알게된 점 기록

📌 Data Preprocessing

```
class segDataset(Dataset):
    def __init__(self, data_path, transforms=None):
        # cata path 설정 잘하기
        self.pos_imgs = sorted(glob(data_path + 'Positive/Image/*'))
        self.pos_labels = sorted(glob(data_path + 'Positive/Label/*'))
        self.neg_imgs = sorted(glob(data_path + 'Negative/Image/*'))
        self.neg_labels = sorted(glob(data_path + 'Negative/Label /*'))
        # positive와 negative를 합쳐서 불러오는 코드를 작성
        self.imgs = self.pos_imgs + self.neg_imgs
        self.labels = self.pos_labels + self.neg_labels
        self.transforms = transforms

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, item):
        img_path = self.imgs[item]
        label_path = self.labels[item]
        img = cv2.imread(img_path)
        label = cv2.imread(label_path, cv2.IMREAD_UNCHANGED)
        label = np.expand_dims(label, axis=2)

        concat = np.concatenate([img, label], axis=2)
        concat = torch.from_numpy(concat)
        concat = concat.permute(2,0,1) # (h,w,c -> c,h,w)


        if self.transforms:
            imgs = self.transforms(concat)

        X = imgs[:3].to(torch.float32)
        y = imgs[3].to(torch.float32)

        return {'X' : X/255, 'y': y/255}
```

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    # transforms.ToTensor(), # (h,w,c -> c,h,w) + Normalize
])
```

📌 Model 구현 part skeleton

```
from torchmetrics.classification import BinaryJaccardIndex

class DeepLabv3_plus(pl.LightningModule):

    def __init__(self, nInputChannels=3, n_classes=21, os=16, pretrained=False, _print=True):

        super(DeepLabv3_plus, self).__init__()

        self.jaccard = BinaryJaccardIndex()

        # Atrous Conv

        ...

    def forward(self, input):

        ...

    def configure_optimizers(self): { ... }


    def training_step(self, batch, batch_idx):

            x, y = batch['X'], batch['y']

            outputs = self(x).squeeze(dim=1)

            loss = nn.BCELoss()(outputs, y)


            predicted_masks = (outputs > 0.5).to(torch.uint8) # 0 아니면 1 로 바꾸는

            y = y.to(torch.uint8)


            iou = self.jaccard(predicted_masks, y)

            self.log('train_loss', loss, on_step=True, on_epoch=True, logger=True)

            self.log('train_iou', iou, on_step=True, on_epoch=True, logger=True)

            return loss
```