2022 MOAI Challenge

Body morphometry AI Segmentation

Team. Accelers

Members. 김상호 / 김유진

01 Introduction

1. Framework
- PyTorch

2. IDE
- 작업 수행 및 리팩토링 : Visual Studio Code
- 디버거 : Jupyter Notebook

3. Used Model
- U-Net (MICCAI 2015)

4. Skills
- Data Augmentation
- Learning Rate Scheduler
- Customize Loss Functions
- Various Optimizers and hyperparameters

02 Methods

Data 전처리와 분할 방법

Data preprocessing

- **mritopng** module을 활용해 dcm format을 png format으로 변환

Dataset split

- Overfitting 여부 확인 목적으로 분할
- 총 100개의 데이터 중 Train-set은 80개, Valid-set은 20개를 사용하였음
- 하이퍼파라미터마다 정확하고 객관적인 성능 비교를 위해 Valid-set은 고정하였음

02 Methods

적은 수의 train-set만으로 robust한 성능을 가지는 모델을 구축하려면 augmentation이 필요

<Include Classes>

- Random HorizontalFlip
- Random Rotation
- Random Scaling
- Random Crop

```
class RandomHorizontalFlip(object):
  """
    Option:
    probability of 0.5
  """
class RandomRotate(object):
  """
    Option:
    probability of 0.5
  """
class RandomScale(object):
  """
    Option:
    range of (0.5, 0.75, 1., 1.25, 1.5)
  """
class RandomCrop(object):
  """
    Option:
    size of (512, 512)
  """
```

02 Methods

Model

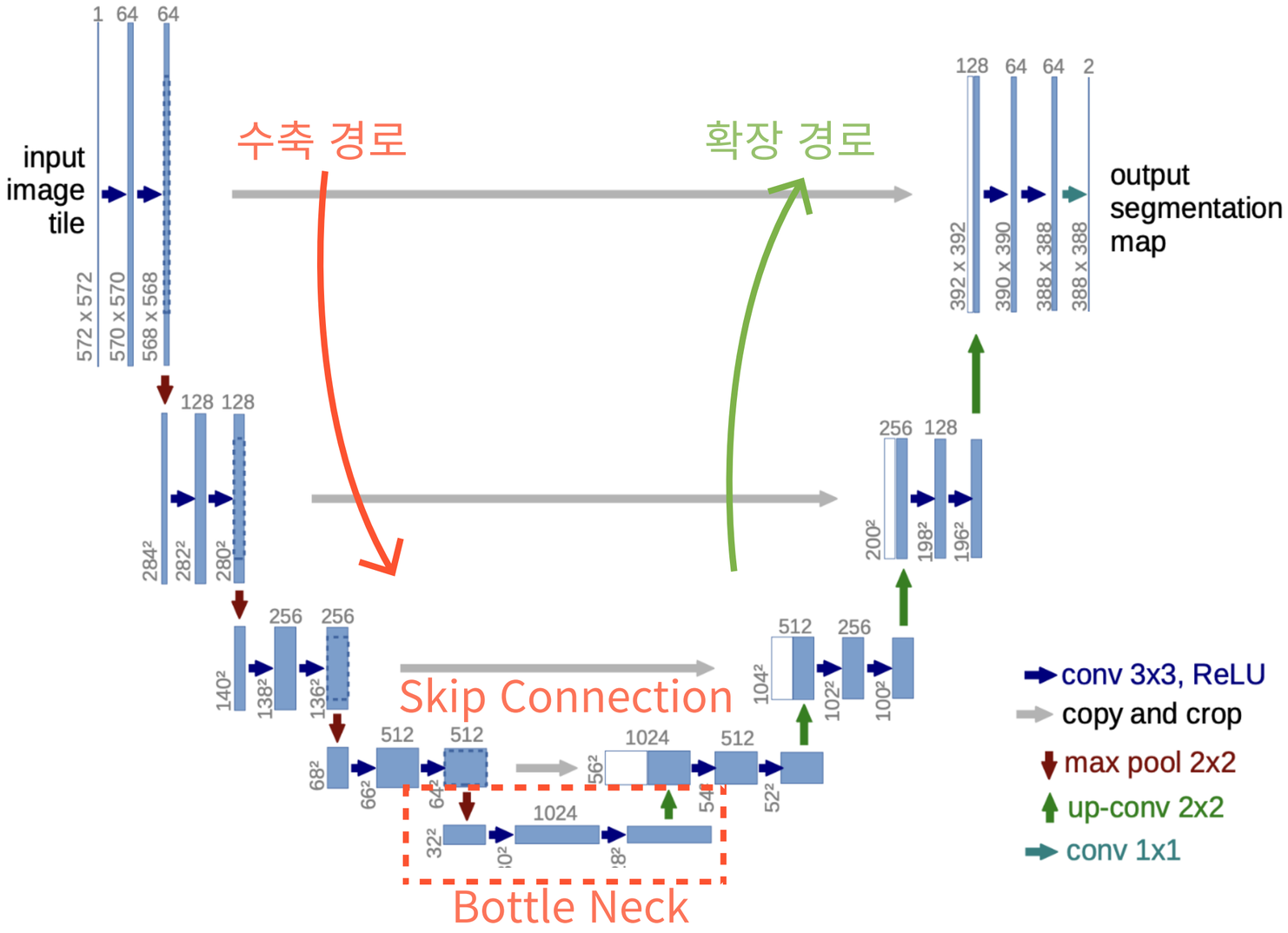
The architecture of U-Net

1. Biomedical 분할에 적합한 모델

2. Contracting(수축) Path

3. Expanding(확장) Path

4. Bottle Neck



02 Methods

Customize Model

1. 논문 상 기본적으로 padding을 사용하지 않지만 모델의 성능 향상을 위해
   padding을 사용해서 모델이 구조적으로 완벽한 대칭을 이루도록 설계하였음

2. 4개의 **Encoding layer**, 1개의 **BottleNeck layer** 그리고 4개의 **Decoding layer**를 구축

3. 마지막 layer의 output channel 수를 4로 **fine-tuning**

02 Methods

Loss functions

- **Dice Loss**
  - 이번 대회의 평가 기준인 Dice metric을 따라 loss function은 1-dice로 설정

- **Ohem CE Loss**
  - 일정 threshold를 설정
  - 그 이상의 loss 값을 가지는 index에 대해서만 backpropagation을 수행
  - 그 이후의 연산은 일반적인 Cross Entropy Loss의 계산과 동일

- **Weighted loss functions with above losses**
  - $\lambda_1 * (1 - dice\ loss) + \lambda_2 * ohem\ ce\ loss$
  - 여러 방법으로 학습하면서 최적의 하이퍼파라미터 λ_1, λ_2 를 탐색

02 Methods

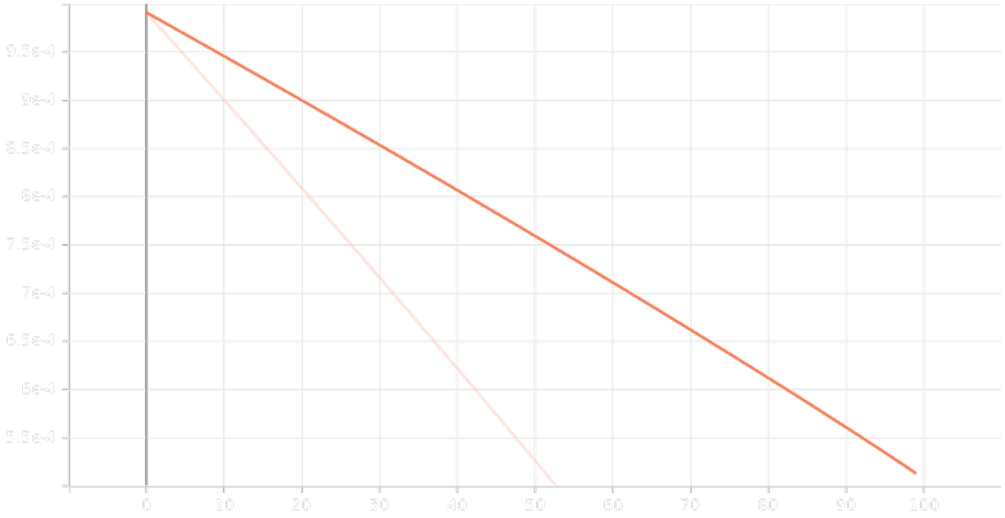
Learning Rate Scheduler

· 높은 값의 학습률(LR)로 시작해 점점 값을 감소시키면서 좋은 솔루션을 빠르게 발견하기 위한 전략 사용

· Polynomial LR Decay Scheduler

- $(1 - \frac{iter}{max\_iter})^{power}$
- where iter is training epoch, max_iter is total epochs and power is 0.9

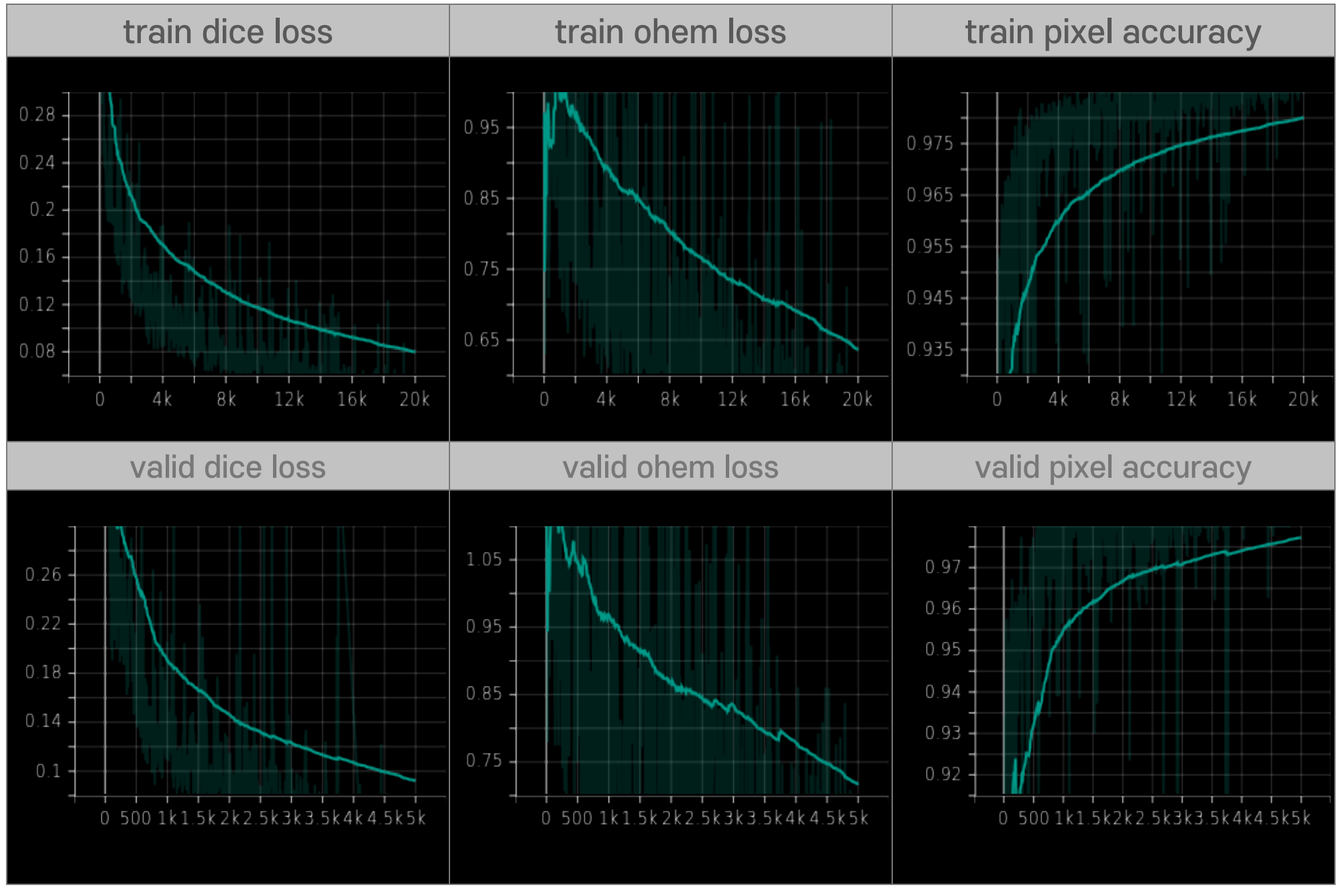
- figure



03 Results of Experiment

Experiment 1

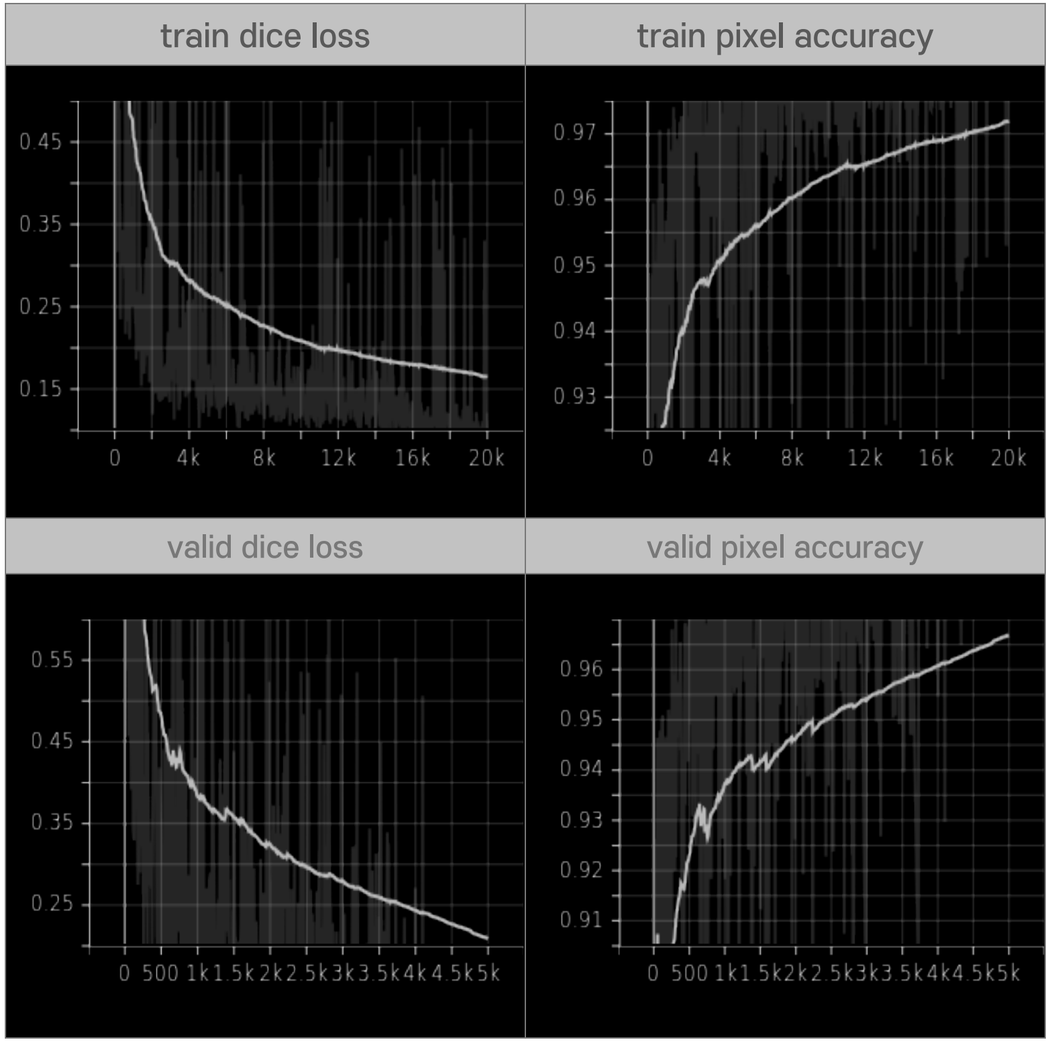
- optimizer = **RAdam**
- initial lr = **5e-3** and **end lr: 1e-5**
- epochs = **1000**
- batch size = 4
- weight decay = 5e-4
- special note
  - loss function은 다음과 같이 구성
  - Loss = 2 * (1 - dice) + 0.5 * ohem
- performance = 0.95071



03 Results of Experiment

Experiment 2

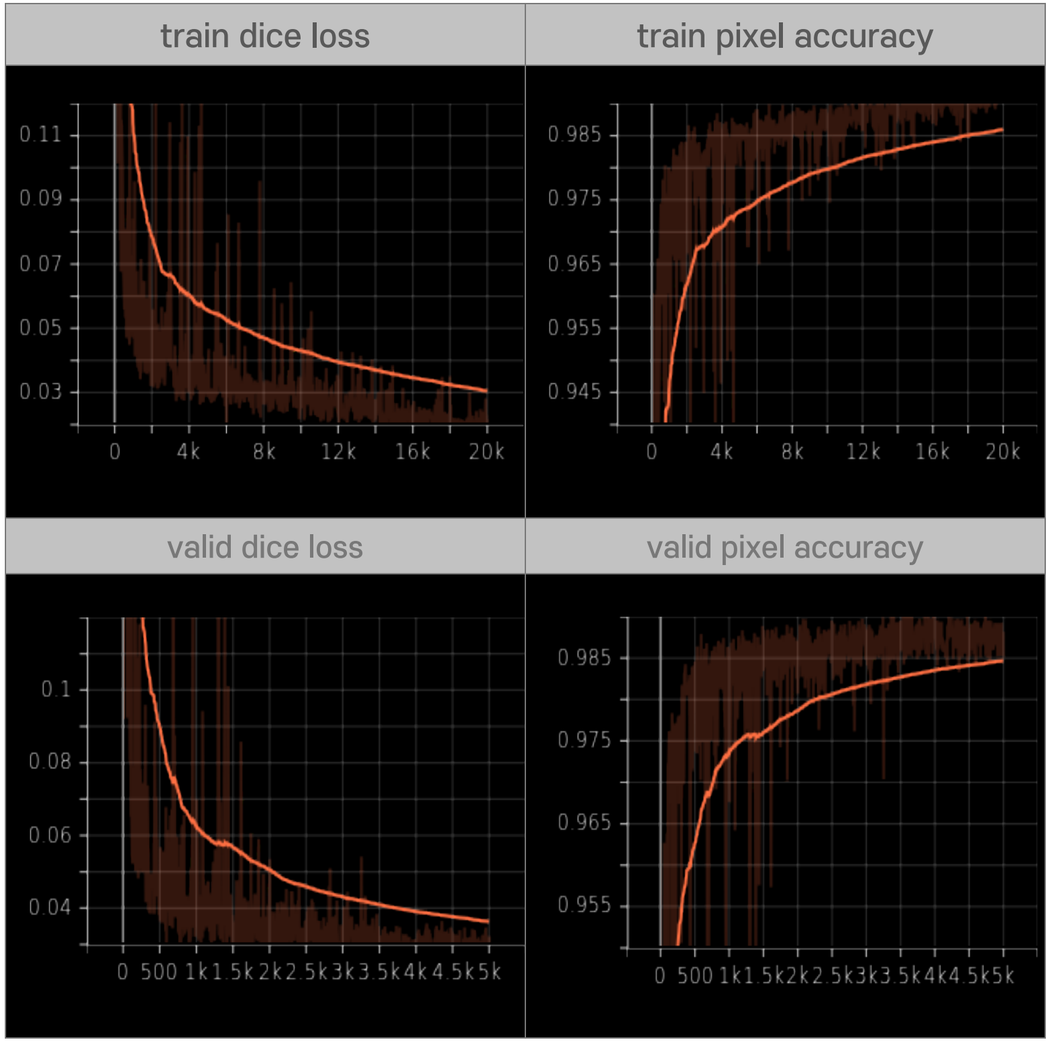
- optimizer= **RAdam**
- initial lr= **5e-3** and **end lr= 1e-5**
- epochs= **1000**
- batch size= 4
- weight decay= 5e-4
- special note
  - weighted value of loss 사용
  - Loss= 3 * (1 - dice) + 0. * ohem
- performance = 0.95023



03 Results of Experiment

Experiment 3

- optimizer= NAdam
- initial lr= 1e-3 and end lr= 1e-5
- epochs= 1000
- batch size= 4
- weight decay= 5e-4
- special note
  - NAdam optimizer 사용
  - 학습률을 낮추고 dice loss만 사용
  - Loss= 1. * (1 - dice) + 0. * ohem
- performance = 0.95867(2nd best score)

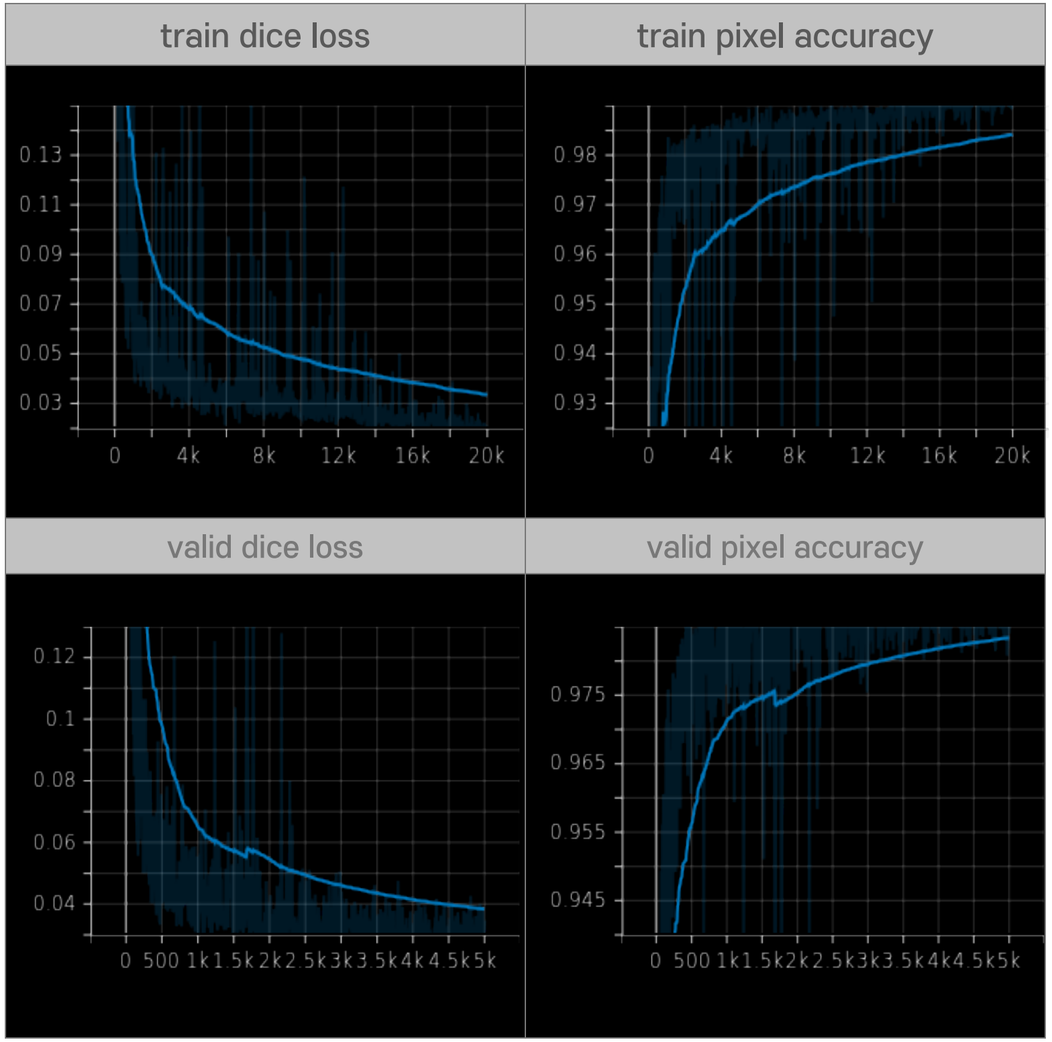


03 Results of Experiment

Experiment 4

- optimizer= NAdam
- initial lr= 1e-3 and end lr= 1e-7
- epochs= 1000
- batch size= 4
- weight decay= 1e-6
- special note
  - 필터 개수를 32에서 64로 증가

- performance = 0.95528

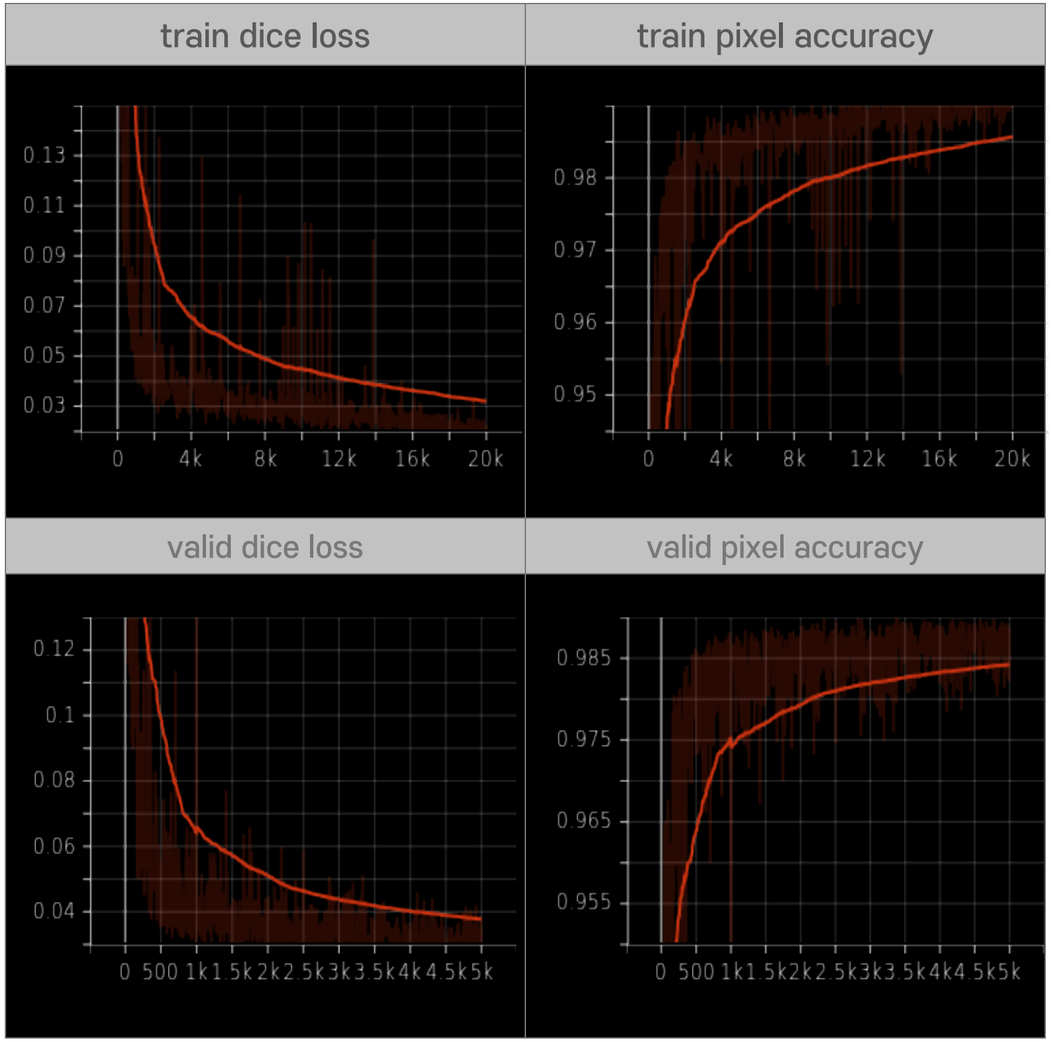


03 Results of Experiment

Experiment 5

- optimizer= NAdam
- initial lr= 1e-3 and end lr= 1e-7
- epochs= 1000
- batch size= 4
- weight decay= 1e-6
- special note
  - 필터 개수를 64에서 16으로 감소
  - 나머지 하이퍼파라미터는 experiment 4와 동일

- performance = 0.95231



03 Results of Experiment

Experiment 6

- optimizer= **AdamW**
- initial lr= **1e-3** and **end lr= 1e-12**
- epochs= 3000
- batch size= 4
- weight decay= 1e-2
- special note
  - end lr 대폭 감소
  - weight decay 증가
  - AdamW optimizer 사용
  - epochs 3000으로 대폭 증가
- performance = 0.95978(1st best score)