2023 samsung AI Challenge

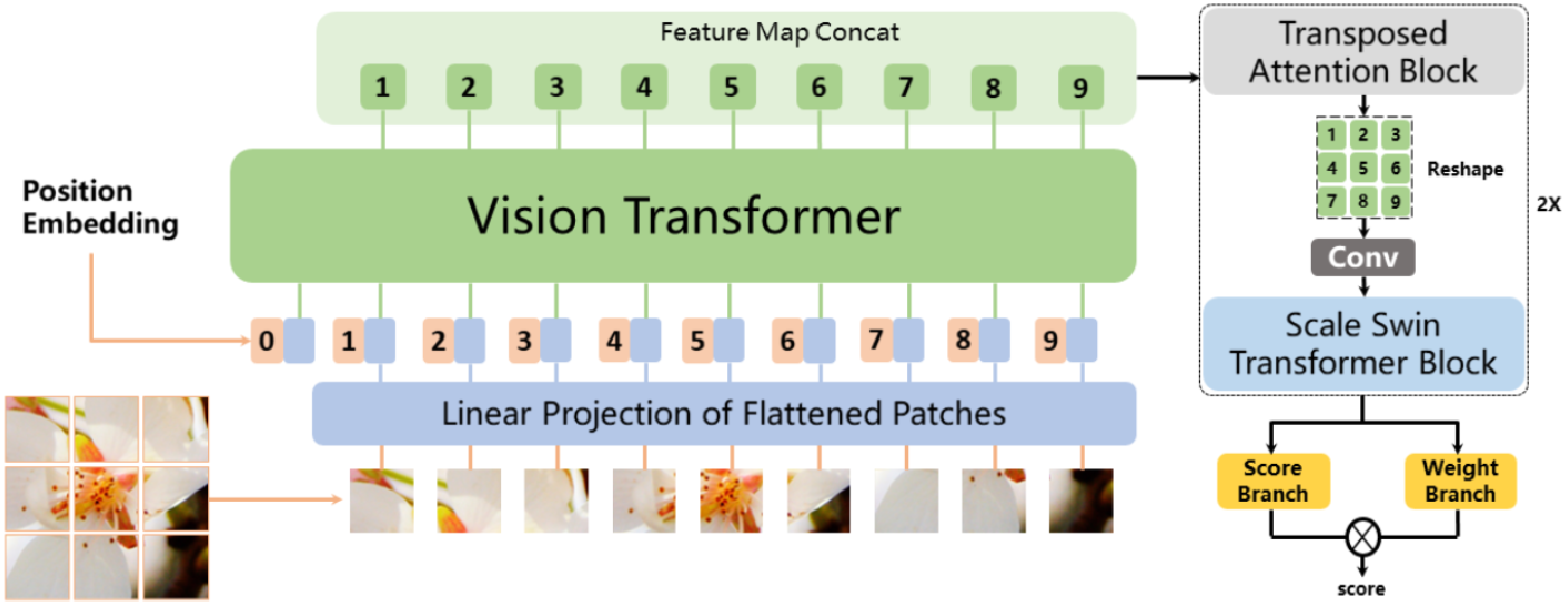
: Image Quality Assessment

박주용, Inha univ

Contents

01 Model

MANIQA : Multi-dimension Attention Network for No-Reference Image Quality Assessment



출처 :https://github.com/IIGROUP/MANIQA

01 Model

Sol1) MANIQA 모델 중, input image resolution을 크게 주기 위해서 vit 모델의 input size를 변경

■ 이유 : MANIQA 논문 코드에서는 input size가 (224,224)로 crop을 통해 주지만, training dataset의 대부분이 (224,224)보다 크다는 것을 확인했기 때문에 training data를 resize하는 저희 팀의 방법에서는 (224, 224) 보다 input size가 컸을 때 성능이 오를 것이라고 예상

■ 방법 : model input size를 384, 448, 640 으로 받을 수 있는 모델 재구성

출처 :https://github.com/IIGROUP/MANIQA

01 Model

Sol1) Code Details <384 model>

```python
    ):
        super().__init__()
        self.img_size = img_size
        self.patch_size = patch_size
        self.input_size = img_size // patch_size
        self.patches_resolution = (img_size // patch_size, img_size // patch_size)
        self.vit = timm.create_model("vit_base_patch16_384", pretrained=True)
        self.save_output = SaveOutput()
        hook_handles = []
```

■ vit 부분을 384 model로 변경

출처 :https://github.com/IIGROUP/MANIQA

01 Model

Sol1) Code Details <448, 640 model>

```python
def vit_base_patch16_448(pretrained=False, **kwargs):
    model_args = dict(
        patch_size=16, embed_dim=768, depth=12, num_heads=12, img_size=448
    )
    model = _create_vision_transformer(
        "vit_base_patch16_384", pretrained=pretrained, **dict(model_args, **kwargs)
    )
    return model


def vit_base_patch16_640(pretrained=False, **kwargs):
    model_args = dict(
        patch_size=16, embed_dim=768, depth=12, num_heads=12, img_size=640
    )
    model = _create_vision_transformer(
        "vit_base_patch16_384", pretrained=pretrained, **dict(model_args, **kwargs)
    )
    return model
```

■ 448, 640 모델은 제공하는 코드가 없어서 단순하게 돌리면 embedding layer에서 에러 발생, model_args의 img_size 인자에 448, 640을 추가로 주고, vit class를 새로 구성하는 것으로 해결

■ 왼쪽의 코드에서 먼저 함수를 정의하고, 이후 MANIQA 모델 정의

출처 :https://github.com/IIGROUP/MANIQA

01 Model

Sol1) Code Details <448, 640 model>

```python
class MANIQA_640(nn.Module):
    def __init__(
        self,
        embed_dim=72,
        num_outputs=1,
        patch_size=8,
        drop=0.5,
        depths=[2, 2],
        window_size=4,
        dim_mlp=768,
        num_heads=[4, 4],
        img_size=224,
        num_tab=2,
        scale=0.8,
        **kwargs
    ):
        super().__init__()
        self.img_size = img_size
        self.patch_size = patch_size
        self.input_size = img_size // patch_size
        self.patches_resolution = (img_size // patch_size, img_size // patch_s
        self.vit = vit_base_patch16_640(pretrained=True)
        self.save_output = SaveOutput()
```

```python
class MANIQA_448(nn.Module):
    def __init__(
        self,
        embed_dim=72,
        num_outputs=1,
        patch_size=8,
        drop=0.5,
        depths=[2, 2],
        window_size=4,
        dim_mlp=768,
        num_heads=[4, 4],
        img_size=224,
        num_tab=2,
        scale=0.8,
        **kwargs
    ):
        super().__init__()
        self.img_size = img_size
        self.patch_size = patch_size
        self.input_size = img_size // patch_size
        self.patches_resolution = (img_size // patch_size, img_size // patch_size)
        self.vit = vit_base_patch16_448(pretrained=True)
        self.save_output = SaveOutput()
```

■ 448, 640 모델 class의 self.vit를 앞에서 정의한 함수로 받는 코드

출처 :https://github.com/IIGROUP/MANIQA

01 Model

Sol2) 과적합 방지를 위해 마지막 layer 중 {dropout 0.1 -> 0.5}, {hidden dim : 768 // 2 -> 786 // 8}

■ 논문과 다르게 linear layer의 일부 하이퍼파라미터 변경

```python
    def __init__(
        self,
        embed_dim=72,
        num_outputs=1,
        patch_size=8,
        drop=0.5,
        depths=[2, 2],
        window_size=4,
        dim_mlp=768,
```

```python
        )
        self.fc_score = nn.Sequential(
            nn.Linear(embed_dim // 2, embed_dim // 8),
            nn.ReLU(),
            nn.Dropout(drop),
            nn.Linear(embed_dim // 8, num_outputs),
            nn.ReLU(),
        )
        self.fc_weight = nn.Sequential(
            nn.Linear(embed_dim // 2, embed_dim // 8),
            nn.ReLU(),
            nn.Dropout(drop),
            nn.Linear(embed_dim // 8, num_outputs),
            nn.Sigmoid(),
        )
```

출처 :https://github.com/IIGROUP/MANIQA

02 Training Details

1. Data
- Training Data : 주어진 전체 Train Data 중 80%
- Validation Data : 주어진 전체 Train Data 중 20%

- Batch_size : 32 (384 model), 16 (448, 640 model)
- num_worker : 4

2. Optimizer
- torch.optim.Adam

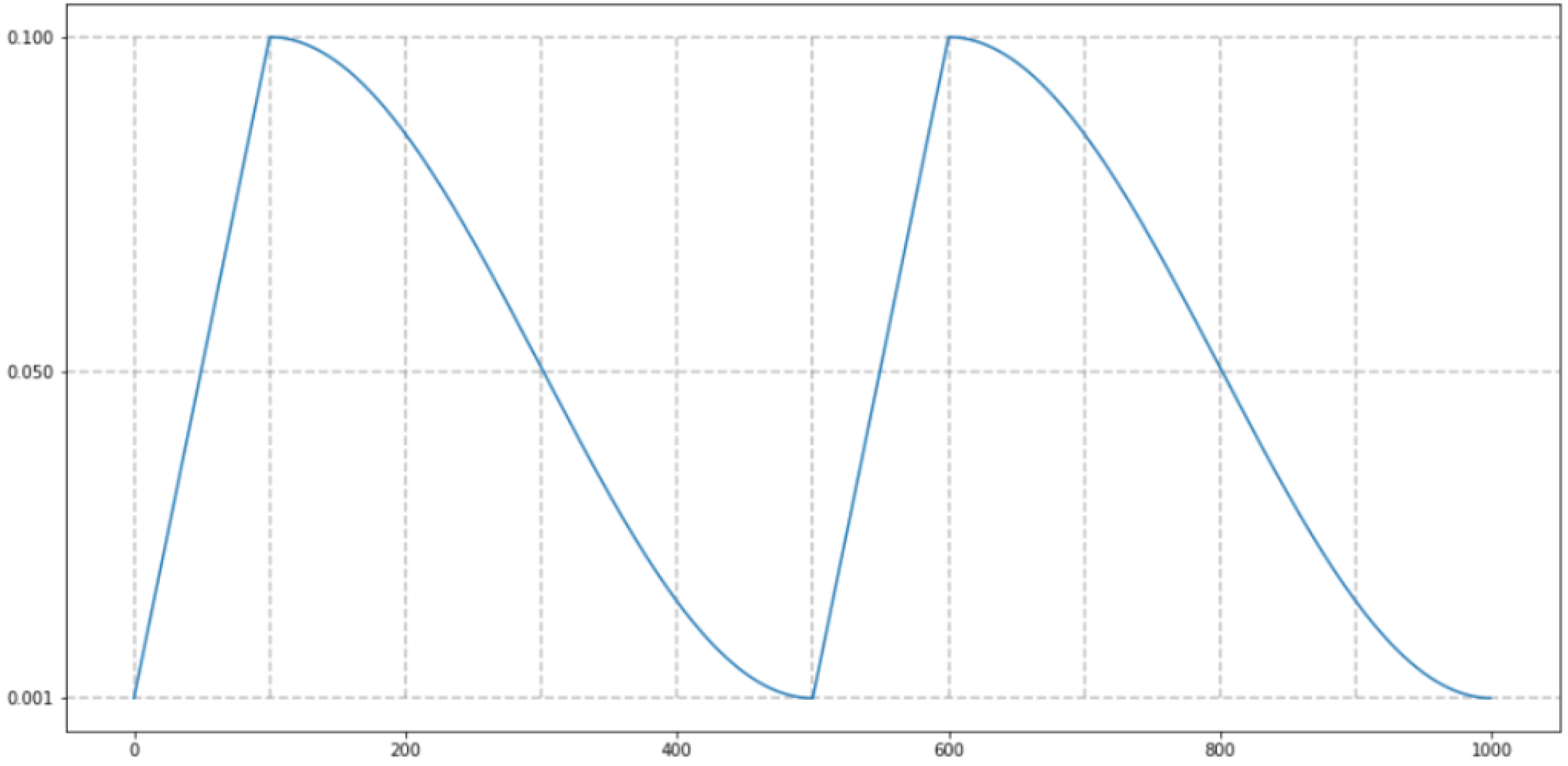
> learning_rate : 1e-5
> weight_decay : 1e-5

3. Scheduler
- cosine_annealing_warmpup.CosineAnnealingWarmupRestarts

> first_cycle_steps: 800
> cycle_mult: 1.0
> max_lr: 1e-5
> min_lr: 1e-10
> warmup_steps: 200
> gamma: 0.9

02 Training Details

cosine_annealing_warmpup.CosineAnnealingWarmupRestarts



step에 따라서 learning rate 조절

출처 : https://github.com/katsura-jp/pytorch-cosine-annealing-with-warmup

02 Training Details

4. 성능 측정

- Challenge Metric과 동일한 PLCC + SRCC 값을 Metric으로 사용
- scipy 라이브러리의 spearmanr, pearsonr 사용

- Metric 값을 기준으로 best score를 기록하면 Weight 저장
- 후에 Inference 할 때, best score를 기록한 epoch의 Weight를 사용하여 Inference

5. Seed

- Hydra-lightning-template을 사용했는데, trainer.yaml 파일에 있는 seed를 4420으로 주고 사용 (모든 모델 동일)

02 Training Details

```python
rho_s, _ = spearmanr(
    np.squeeze(self.train_pred_epoch), np.squeeze(self.train_labels_epoch)
)
rho_p, _ = pearsonr(
    np.squeeze(self.train_pred_epoch), np.squeeze(self.train_labels_epoch)
)
```

```yaml
  # debugging config (enable through command line, e.g. `python train.py debug=default)
  - debug: null

# task name, determines output directory path
task_name: "train"

# tags to help you identify your experiments
# you can overwrite this in experiment configs
# overwrite from command line with `python train.py tags="[first_tag, second_tag]"`
tags: ["dev"]

# set False to skip model training
train: True

# evaluate on test set, using best model weights achieved during training
# lightning chooses best weights based on the metric specified in checkpoint callback
test: False

# simply provide checkpoint path to resume training
ckpt_path: null

# seed for random number generators in pytorch, numpy and p
seed: 4420
```

Python 언어에 대한 권장되는 Microsoft의 'Python'
설치하시겠습니까?
설치

03 K-Fold

< 5-Fold 사용 >

1. 데이터를 Mos 기준으로 내림차순 Sort

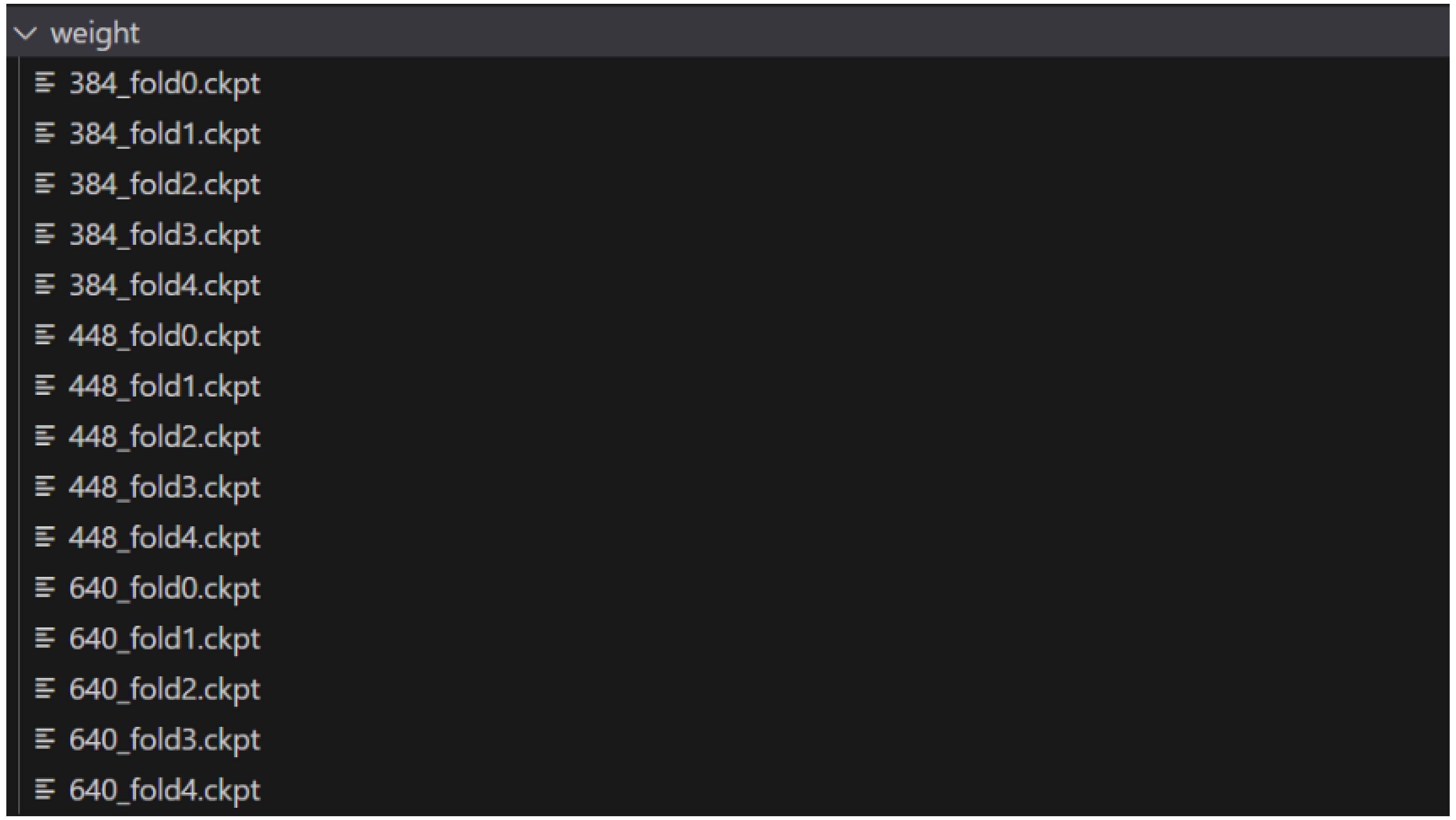
2. Sort된 데이터에 번호를 부여

- 첫번째 fold : 0 0 0 0 1 0 0 0 0 1 ...
- 두번째 fold : 0 0 0 1 0 0 0 0 1 0 ...
- 세번째 fold : 0 0 1 0 0 0 0 1 0 0 ...
- 네번째 fold : 0 1 0 0 0 0 1 0 0 0 ...
- 다섯째 fold : 1 0 0 0 0 1 0 0 0 0 ...

- 위와 같은 방식으로 0이면 train data, 1이면 validation data로 나눔 (8:2로 나눠짐)

3. 총 5개의 fold를 각각의 모델 (384, 448, 640)으로 training 한 뒤, best score를 기록한 weight를 기준으로 inference, 5x3 = 15 개의 inference mos 값을 mean 하여 최종 final mos 제출

03 K-Fold



weight 파일, ckpt 형식으로 저장됨

03 K-Fold

```
root@b9f7cbd38afb:~# cd dacon/lightning-hydra-template/src
root@b9f7cbd38afb:~/dacon/lightning-hydra-template/src# python eval.py ckpt_path=../weight/384_fold0.ckpt model=maniqa_384_model data=maniqa_384_data model.name=384_fold0
```

중요 : weight 파일을 통해서 dacon/data/hydra-lightning-template/src 에서 위와 같은 명령 실행
model.name을 통해서 저장할 csv파일의 이름 지정

384

```
ex) 384모델 fold0 weight inference
python eval.py ckpt_path=../weight/384_fold0.ckpt model=maniqa_384_model data=maniqa_384
_data model.name=384_fold0

ex) 384모델 fold1 weight inference
python eval.py ckpt_path=../weight/384_fold1.ckpt model=maniqa_384_model data=maniqa_384
_data model.name=384_fold1

ex) 384모델 fold2 weight inference
python eval.py ckpt_path=../weight/384_fold2.ckpt model=maniqa_384_model data=maniqa_384
_data model.name=384_fold2

ex) 384모델 fold3 weight inference
python eval.py ckpt_path=../weight/384_fold3.ckpt model=maniqa_384_model data=maniqa_384
_data model.name=384_fold3

ex) 384모델 fold4 weight inference
python eval.py ckpt_path=../weight/384_fold4.ckpt model=maniqa_384_model data=maniqa_384
_data model.name=384_fold4
```

448

```
ex) 448모델 fold0 weight inference
python eval.py ckpt_path=../weight/448_fold0.ckpt model=maniqa_448_model data=maniqa_448
_data model.name=448_fold0

ex) 448모델 fold1 weight inference
python eval.py ckpt_path=../weight/448_fold1.ckpt model=maniqa_448_model data=maniqa_448
_data model.name=448_fold1

ex) 448모델 fold2 weight inference
python eval.py ckpt_path=../weight/448_fold2.ckpt model=maniqa_448_model data=maniqa_448
_data model.name=448_fold2

ex) 448모델 fold3 weight inference
python eval.py ckpt_path=../weight/448_fold3.ckpt model=maniqa_448_model data=maniqa_448
_data model.name=448_fold3

ex) 448모델 fold4 weight inference
python eval.py ckpt_path=../weight/448_fold4.ckpt model=maniqa_448_model data=maniqa_448
_data model.name=448_fold4
```

640

```
ex) 640모델 fold0 weight inference
python eval.py ckpt_path=../weight/640_fold0.ckpt model=maniqa_640_model data=maniqa_640
_data model.name=640_fold0

ex) 640모델 fold1 weight inference
python eval.py ckpt_path=../weight/640_fold1.ckpt model=maniqa_640_model data=maniqa_640
_data model.name=640_fold1

ex) 640모델 fold2 weight inference
python eval.py ckpt_path=../weight/640_fold2.ckpt model=maniqa_640_model data=maniqa_640
_data model.name=640_fold2

ex) 640모델 fold3 weight inference
python eval.py ckpt_path=../weight/640_fold3.ckpt model=maniqa_640_model data=maniqa_640
_data model.name=640_fold3

ex) 640모델 fold4 weight inference
python eval.py ckpt_path=../weight/640_fold4.ckpt model=maniqa_640_model data=maniqa_640
_data model.name=640_fold4
```

```python
import pandas as pd


test_640_fold_0 = pd.read_csv("./640_fold0.csv")
test_640_fold_1 = pd.read_csv("./640_fold1.csv")
test_640_fold_2 = pd.read_csv("./640_fold2.csv")
test_640_fold_3 = pd.read_csv("./640_fold3.csv")
test_640_fold_4 = pd.read_csv("./640_fold4.csv")

test_448_fold_0 = pd.read_csv("./448_fold0.csv")
test_448_fold_1 = pd.read_csv("./448_fold1.csv")
test_448_fold_2 = pd.read_csv("./448_fold2.csv")
test_448_fold_3 = pd.read_csv("./448_fold3.csv")
test_448_fold_4 = pd.read_csv("./448_fold4.csv")

test_384_fold_0 = pd.read_csv("./384_fold0.csv")
test_384_fold_1 = pd.read_csv("./384_fold1.csv")
test_384_fold_2 = pd.read_csv("./384_fold2.csv")
test_384_fold_3 = pd.read_csv("./384_fold3.csv")
test_384_fold_4 = pd.read_csv("./384_fold4.csv")

path = test_640_fold_0["img_name"]
cap = test_640_fold_0["comments"]


last_csv = pd.DataFrame(path, columns=["img_name"])
last_csv["mos"] = (
    test_640_fold_0["mos"]
    + test_640_fold_1["mos"]
    + test_640_fold_2["mos"]
    + test_640_fold_3["mos"]
    + test_640_fold_4["mos"]
    + test_448_fold_0["mos"]
    + test_448_fold_1["mos"]
    + test_448_fold_2["mos"]
    + test_448_fold_3["mos"]
    + test_448_fold_4["mos"]
    + test_384_fold_0["mos"]
    + test_384_fold_1["mos"]
    + test_384_fold_2["mos"]
    + test_384_fold_3["mos"]
    + test_384_fold_4["mos"]
) / 15
last_csv["comments"] = cap
```

Infrence된 csv들을 통해서 최종 mos를 뽑는 result_csv/15fold.py

End Page