연구 노트 4주차

▼ 2팀 : 20191467 이재형

✅연구 주제 : Adative data enginering Learning framework

✅4주차 Topic : Reinforce Tuning for FT Model training Using LAPA Dataset

> 💡 지난 3주차는 Face Transformer를 Baseline 모델로 설정하고, 간단한 TNBC dataset을 통해 reinforce tuning의 가능성과 효용성을 검증하였다.
> 이번 4주차는
> **LAPA Face Parsing Dataset**을 직접 활용하기 위하여 코드 작업을 진행하였습니다.

---

1. LAPA Dataset

- LAPA Dataset을 통한 학습을 위해 LAPA Dataset**[1]**을 다운받았다.
- 학습을 위해 DataLoader를 생성하기 위해, LAPA Dataset의 폴더 구조를 먼저 살펴보았다.

```
LAPA/
├── train/
│   ├── images/
│   ├── labels/
│   └── landmarks/
├── val/
│   ├── images/
│   ├── labels/
│   └── landmarks/
└── test/
    ├── images/
    ├── labels/
    └── landmarks/
```

- LAPA는 총 3가지 폴더 train, test, val으로 나누어져 있었다.
  - ‘images’ 폴더 : 입력 이미지
  - ‘labels’ 폴더 : mask 이미지(레이블)
  - ‘landmarks’ 폴더 : 랜드마크 데이터 포함

- 우리는 ‘images’와 ‘labels’ 폴더를 사용하여 모델을 학습할 것이다.

---

2. DataLoader Code 작성

▼ LAPA Dataset을 위한 데이터 로더를 작성한 코드에 대해 설명하겠다.

2-1. LAPADataset 클래스

- LAPADataset 클래스는 PyTorch의 Dataset 클래스를 상속받아, LAPA 데이터셋의 이미지를 로드하고 전처리하는 역할을 수행한다.
- 이 클래스는 이미지와 그에 대응되는 mask를 로드하고, 이 들을 모델의 입력으로 제공할 수 있는 형태로 변환한다.

```
import os
from PIL import Image
from torch.utils.data import Dataset
import torchvision.transforms as transforms

class LAPADataset(Dataset):
    def __init__(self, images_dir, labels_dir, transform=None):
        self.images_dir = images_dir
        self.labels_dir = labels_dir
        self.transform = transform
        # 이미지 디렉토리에서 모든 이미지 파일을 리스트로 저장
        self.image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg') or f.endswit

    def __len__(self):
        # 데이터셋의 전체 샘플 수를 반환
        return len(self.image_files)

    def __getitem__(self, idx):
        img_name = os.path.join(self.images_dir, self.image_files[idx])
        label_name = os.path.join(self.labels_dir, self.image_files[idx].replace('.jpg', '.png')

        image = Image.open(img_name).convert("RGB")
        label = Image.open(label_name).convert("L")

        if self.transform:
            image = self.transform(image)
            label = self.transform(label)

        return image, label
```

[주요 메서드]

1. `'__init__'`

- 클래스 초기화 메서드
- images_dir (str): 이미지가 저장된 디렉토리 경로
  labels_dir (str): 마스크가 저장된 디렉토리 경로
  transform (callable, optional): 이미지와 마스크에 적용할 변환 함수
- 주어진 이미지 디렉토리에서 모든 이미지 파일을 리스트로 저장

2. `'__len__'`

- 데이터셋의 전체 샘플 수를 반환

3. `'__getitem__'`

- 주어진 인덱스에 해당하는 샘플(이미지와 그에 대응하는 mask)을 로드하고, 변환을 적용하여 반환

2-2. LAPASegmentationDataModule 클래스

- LAPASegmentationDataModule 클래스는 PyTorch Lightning의 LightningDataModule 클래스를 상속받아, 학습, 검증 및 테스트 데이터 로더를 관리하는 역할을 수행한다.

```
from pytorch_lightning import LightningDataModule

class LAPASegmentationDataModule(LightningDataModule):
    def __init__(self, data_dir: str, batch_size: int = 32):
        """
        Args:
            data_dir (str): 데이터셋 디렉토리 경로
            batch_size (int, optional): 배치 크기 (기본값: 32)
        """
        super().__init__()
        self.data_dir = data_dir
        self.batch_size = batch_size
        self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.5], std=[0.5])
        ])

    def setup(self, stage: Optional[str] = None):
        train_images_dir = os.path.join(self.data_dir, 'train', 'images')
        train_labels_dir = os.path.join(self.data_dir, 'train', 'labels')
        val_images_dir = os.path.join(self.data_dir, 'val', 'images')
        val_labels_dir = os.path.join(self.data_dir, 'val', 'labels')

        self.train_set = LAPADataset(images_dir=train_images_dir, labels_dir=train_labels_dir, t
        self.val_set = LAPADataset(images_dir=val_images_dir, labels_dir=val_labels_dir, transfo

    def train_dataloader(self):
        return DataLoader(self.train_set, batch_size=self.batch_size, shuffle=True)

    def val_dataloader(self):
        return DataLoader(self.val_set, batch_size=self.batch_size)
```

[주요 메서드]

1. `'__init__'`

- 클래스 초기화 메서드
- 데이터 디렉토리 경로와 배치 크기를 인자로 받음

2. `'setup'`

- 데이터셋을 설정하는 메서드
- 학습 및 검증 데이터를 나누고 각각의 'LAPADataset' 인스턴스를 생성

3. `'train_dataloader'`

- 학습 데이터 로더 반환

4. `'val_dataloader'`

- 검증 데이터 로더 반환

```
####################LAPADataLoader#######################
import os
from PIL import Image
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
from typing import Optional
from pytorch_lightning import LightningDataModule

class LAPADataset(Dataset):
    def __init__(self, images_dir, labels_dir, transform=None):
        self.images_dir = images_dir
        self.labels_dir = labels_dir
        self.transform = transform
        self.image_files = [f for f in os.listdir(images_dir) if f.endswith('.jpg') or f.endswit

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

    def __getitem__(self, idx):
        img_name = os.path.join(self.images_dir, self.image_files[idx])
        label_name = os.path.join(self.labels_dir, self.image_files[idx].replace('.jpg', '.png')

        image = Image.open(img_name).convert("RGB")
        label = Image.open(label_name).convert("L")

        if self.transform:
            image = self.transform(image)
            label = self.transform(label)

        return image, label

class LAPASegmentationDataModule(LightningDataModule):
    def __init__(self, data_dir: str, batch_size: int = 32):
        super().__init__()
        self.data_dir = data_dir
        self.batch_size = batch_size
        self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.5], std=[0.5])
        ])
```

```
    def setup(self, stage: Optional[str] = None):
        train_images_dir = os.path.join(self.data_dir, 'train', 'images')
        train_labels_dir = os.path.join(self.data_dir, 'train', 'labels')
        val_images_dir = os.path.join(self.data_dir, 'val', 'images')
        val_labels_dir = os.path.join(self.data_dir, 'val', 'labels')

        self.train_set = LAPADataset(images_dir=train_images_dir, labels_dir=train_labels_dir, t
        self.val_set = LAPADataset(images_dir=val_images_dir, labels_dir=val_labels_dir, transfo

    def train_dataloader(self):
        return DataLoader(self.train_set, batch_size=self.batch_size, shuffle=True)

    def val_dataloader(self):
        return DataLoader(self.val_set, batch_size=self.batch_size)
```

3. FT Train Code 작성

▼ 위에서 작성한 데이터 모듈을 사용하여 학습을 진행하는 코드에 대해 설명하겠다.

3-1. DataLoader import

```
from LAPA_dataloader import LAPADataset, LAPASegmentationDtaModule
```

- 필요한 라이브러리와 모듈이 다양하지만, 위에서 정의한 데이터 로더를 import하여 데이터 로더를 설정한다.

3-2. Model 클래스

- Pytorch Lightning의 LightningModule을 상속받아 모델을 정의한다.
- 모델 정의 부분에서 가장 핵심이 되는 부분인 reward 함수 부분을 설명하겠다.

```
def shared_step(self, batch, mode="train"):
                #1
        x, y = batch

        pred = self.net(x)

                #2
        dist = Bernoulli(logits=pred)
        sample = dist.sample()
        baseline = dist.sample()

                #3
        reward = binary_f1_score(sample, y) - binary_f1_score(baseline, y)
```

```
        #4
log_prob = torch.mean(dist.log_prob(sample), dim=[1, 2])
loss = torch.mean(-log_prob * reward)

        #5
metrics = getattr(self, f"{mode}_metrics")(pred, y.long())

self.log(f"{mode}_loss", loss, on_epoch=True, prog_bar=True)
for k, v in metrics.items():
    self.log(f"{mode}_{k.lower()}", v, on_epoch=True)

return loss
```

1. **입력 및 예측값 계산**
   - x는 입력 이미지, y는 타겟 라벨(mask)이다
   - self.net(x)는 입력 이미지를 통해 예측값 pred를 생성

2. **확률 분포 및 샘플링**
   - Bernoulli는 예측값을 logit으로 하는 베르누이 분포를 생성
   - sample과 baseline은 이 분포에서 샘플링된 값으로 실제 예측값과 기준값을 의미

3. **reward 함수 계산**

$$reward = F1(sample, y) - F1(baseline, y)$$

   - `binary_f1_score(sample, y)`: 예측값 sample과 실제값 y사이의 F1 score 계산
   - `binary_f1_score(baseline, y)`: 기준값 baselne과 실제값 y사이의 F1 score 계산
   - `reward`: 두 스코의 차이, 샘플이 기준값보다 얼마나 더 좋은지를 나타냄

4. **로그 확률 및 Loss 계산**
   - `dist.log_prob(sample)`: 분포에서 샘플링된 값의 로그 확률을 계산
   - `log_prob`: 로그 확률의 평균
   - 손실(loss)은 보상과 샘플의 로그 확률을 곱한 값을 음수로 변환한 것

5. **metric 계산 및 logging**
   - `metrics`: 현재 모드('train', 'val', 'test')에 따른 성능 metric을 계산
   - 손실 값과 metric값을 logging함

3-3. 학습

```
if __name__ == "__main__":
    data_dir = "C:\Users\jaehy\FTproject\cv-rl\LAPA\LaPa"
    batch_size = 32
```

```
    data_module = LAPASegmentationDataModule(data_dir=data_dir, batch_size=batch_size)

    MyLightningCLI(
        BinaryF1SegmentationModel,
        data_module,
        save_config_kwargs={"overwrite": True},
        trainer_defaults={"check_val_every_n_epoch": None},
    )
```

1. **데이터 디렉토리와 배치 크기 설정**
    - `data_dir`: 실제 LAPA 데이터셋이 위치한 경로를 지정
    - `batch_size`: 배치 크기를 설정

2. **데이터 모듈 생성**
    - `LAPASegmentationDataModule`을 인스턴스화하여 데이터 모듈을 생성
    - `data_dir`과 `batch_size`를 인자로 받음

3. **모델 학습**
    - `MyLightningCLI`를 사용하여 모델 학습을 시작
    - `BinaryF1SegmentationModel`과 `data_module`을 인자로 전달
    - `save_config_kwargs`와 `trainer_defaults`는 학습 설정을 정의함

- LAPA 데이터셋을 이용하여 Face Transforemr의 reinforce tuning의 기반 코드를 완성하였다.
- 가상 환경 세팅, 데이터 전처리, 코드 작업에 시간을 소요하여 학습을 진행하지 못하였음.
- 다음 주차에는 학습을 진행하고, 모델 성능을 평가함과 동시에, reward에 따른 성능 차이까지 살펴볼 계획이다.

Reference

[1] https://www.kaggle.com/datasets/kiranraghavendrauci/lapa-face-parsing-dataset?resource=download-directory

[2]

[3]