2월 25일 발표본

SpectraNeura

Sunjun Hwang
RAISE Lab

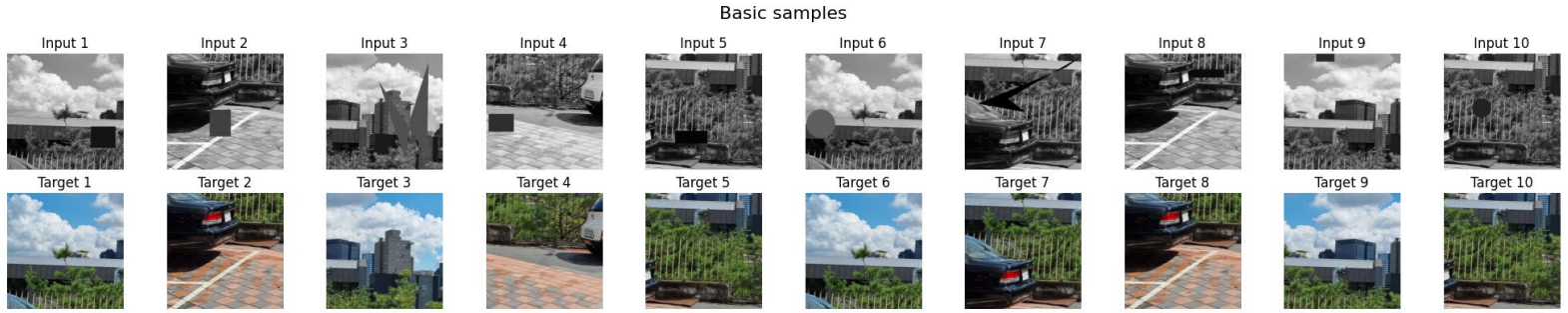
데이터 전처리

기본 전처리 정의

먼저, 기본 전처리에서는 단순하게 이미지의 크기를 256×256으로 Resize를 한 후, 텐서로 변환하는 작업을 수행

```
basic_transform_input = transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.ToTensor(),
])
basic_transform_target = transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.ToTensor(),
])
```

이 부분은 입력 이미지와 타깃이미지에 각각 동일한 기본 전처리를 적용하기 위한 준비 단계



데이터 전처리

JointAugmentation class 정의

JointAugmentation 클래스는 입력 이미지와 타깃 이미지에 동시에 같은 변환(랜덤 회전 및 랜덤 크롭)을 적용하기 위해 설계 되었다.

- 랜덤 회전
  `random.uniform(-self.rotation_range, self.rotation_range)`로 랜덤 회전 각도를 구하고, `TF.rotate` 함수를 사용하여 두 이미지에 동일하게 회전 시킴.
- 랜덤 크롭
  `transforms.RandomCrop.get_params`를 통해 동일한 크롭 파라미터를 생성하고, `TF.crop`을 사용하여 두 이미지에 동일하게 크롭을 수행.

데이터 전처리

JointAugmentation class 정의 - 계속

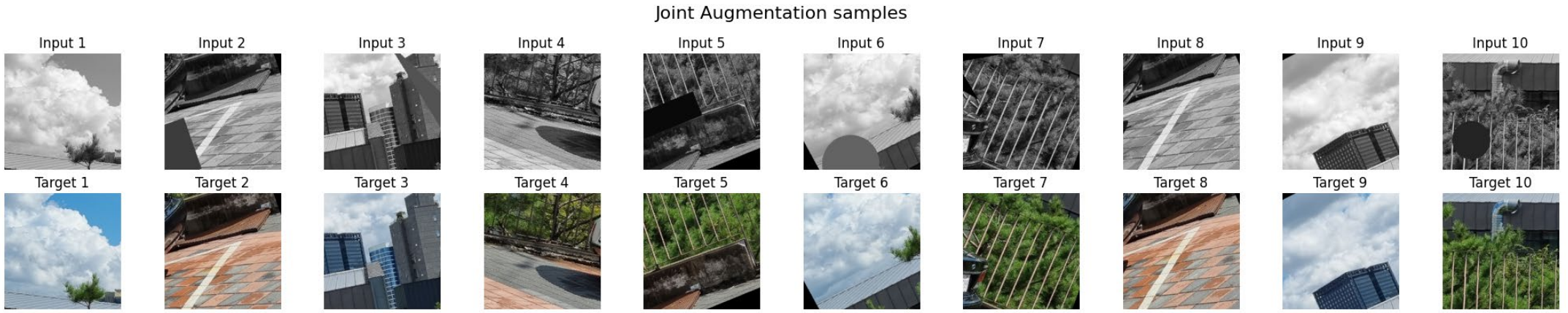
```
class JointAugmentation:
        def __init__(self, rotation_range=30, crop_size=(256, 256)):
                self.rotation_range = rotation_range
                self.crop_size = crop_size

        def __call__(self, input_img, target_img):
                angle = random.uniform(-self.rotation_range, self.rotation_range)
                input_img = TF.rotate(input_img, angle)
                target_img = TF.rotate(target_img, angle)
                i, j, h, w = transforms.RandomCrop.get_params(input_img,
output_size=self.crop_size)
                input_img = TF.crop(input_img, i, j, h, w)
                target_img = TF.crop(target_img, i, j, h, w)
                return input_img, target_img
```

이 부분은 입력 이미지와 타깃 이미지 간의 변환 동기화를 보장하여, 데이터 증강 시 라벨(타깃)과의 일관성을 유지한다.

데이터 전처리

JointAugmentation class 정의 - 계속



데이터 전처리

RandomFlipAugmentation class 정의

RandomFlipAugmentation 클래스는 이미지에 대해 랜덤 수평 및 수직 플립을 적용한다.

- 수평 플립
  `random.random() < self.horizontal_prob` 조건에 따라 입력 및 타깃 이미지를 좌우 반전시킨다.
- 수직 플립
  `vertical_prob`에 따라 상하 반전시킨다.(기본 값은 0이므로, 수직 플립은 기본적으로 적용되지 않는다.)

데이터 전처리

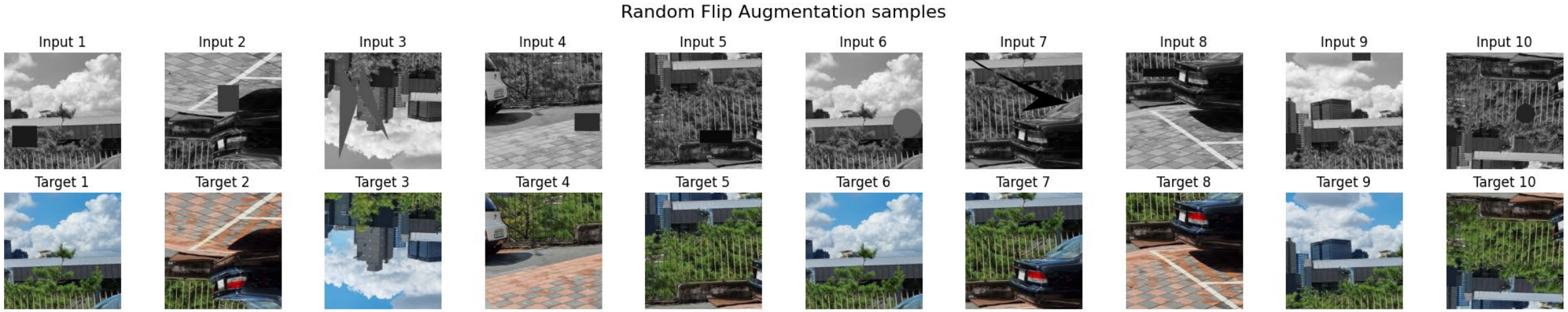
RandomFlipAugmentation class 정의 - 계속

```
class RandomFlipAugmentation:
        def __init__(self, horizontal_prob=0.5, vertical_prob=0.0):
                self.horizontal_prob = horizontal_prob
                self.vertical_prob = vertical_prob
        def __call__(self, input_img, target_img):
                if random.random() < self.horizontal_prob:
                        input_img = TF.hflip(input_img)
                        target_img = TF.hflip(target_img)
                if random.random() < self.vertical_prob:
                        input_img = TF.vflip(input_img)
                        target_img = TF.vflip(target_img)
                return input_img, target_img
```

두 이미지에 대해 독립적이 아니라, 동일한 조건으로 플립을 적용해, 변환 결과의 일관성을 유지할 수 있다.

데이터 전처리

RandomFlipAugmentation class 정의 - 계속



데이터 전처리

CombinedAugmentation class 정의

CombinedAugmentation 클래스는 앞서 정의한 JointAugmentation과 RandomFlipAugmentation을 순차적으로 적용하는 class이다.

- 먼저, joint_aug를 통해 회전 및 크롭을 적용하고
- 이후, filp_aug를 통해 플립을 적용한다.

데이터 전처리

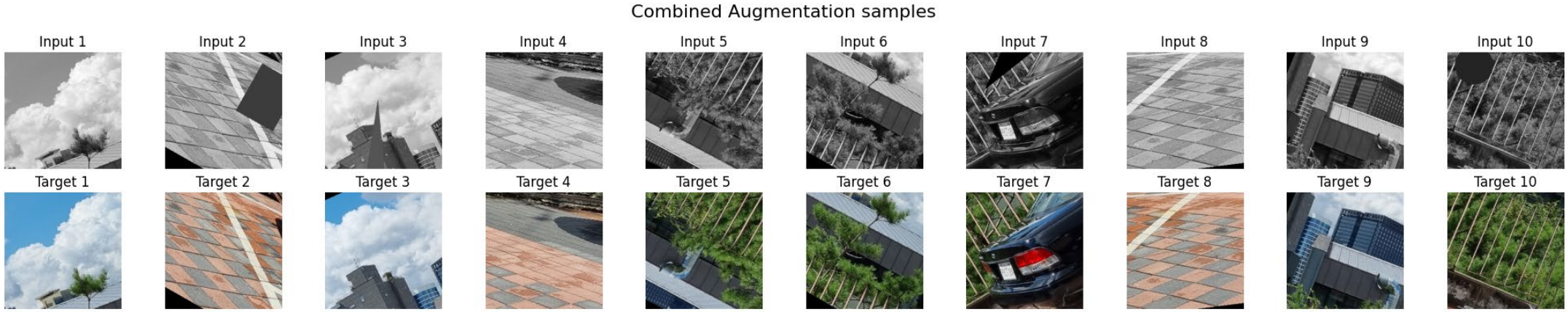
CombinedAugmentation class 정의 - 계속

```
class CombinedAugmentation:
        def __init__(self, joint_aug, flip_aug):
                self.joint_aug = joint_aug
                self.flip_aug = flip_aug
        def __call__(self, input_img, target_img):
                input_img, target_img = self.joint_aug(input_img, target_img)
                input_img, target_img = self.flip_aug(input_img, target_img)
                return input_img, target_img
```

이 클래스는 두 가지 증강 기법을 동시에 활용하여 데이터 다양성을 더욱 극대화할 수 있게 해준다.

데이터 전처리

CombinedAugmentation class 정의 - 계속



RestorationDataset class

RestorationDataset 클래스는 CSV 파일에 저장된 이미지 경로 정보를 바탕으로 이미지 데이터를 불러오고 전처리를 적용하는 커스텀 PyTorch 데이터 셋이다.

주요 구성 요소

- Csv_file: 이미지 경로가 기록된 CSV 파일 경로
- Input_dir & target_dir 실제 이미지 파일이 저장된 디렉터리
- Transform_input & transform_target: 입력 이미지와 타깃 이미지에 각각 개별적으로 적용할 전처리
- Joint_transform: 입력 이미지와 타깃 이미지에 동시에 적용할 전처리

이미지 로드 과정

1. CSV 파일에서 파일명을 추출하여 지정된 디렉터리에서 이미지를 불러온다.
2. 입력 이미지는 흑백(‘L’)으로, 타깃 이미지는 컬러(‘RGB’)로 변환한다.
3. 만약 joint_transform이 지정되면, 입력과 타깃이 이미지에 동일하게 증강을 적용한다.
4. 이후 각각의 transform을 적용하여 텐서 변환등의 작업을 수행

RestorationDataset class

```
class RestorationDataset(Dataset):
        def __init__(self, csv_file, input_dir, target_dir=None, transform_input=None,
transform_target=None,joint_transform=None):
                self.data = pd.read_csv(csv_file)
                self.input_dir = input_dir
                self.target_dir = target_dir
                self.transform_input = transform_input
                self.transform_target = transform_target
                self.joint_transform = joint_transform
        def __len__(self):
                return len(self.data)
        def __getitem__(self, idx):
                input_file = os.path.basename(self.data.iloc[idx]['input_image_path'])
        input_path = os.path.join(self.input_dir, input_file)
        input_img = Image.open(input_path).convert('L') # 흑백 변환
        if self.target_dir is not None:
                target_file = os.path.basename(self.data.iloc[idx]['gt_image_path'])
                target_path = os.path.join(self.target_dir, target_file)
                target_img = Image.open(target_path).convert('RGB') # 컬러 변환
        else:
                target_img = None
        if self.joint_transform is not None and target_img is not None:
                input_img, target_img = self.joint_transform(input_img, target_img)
        if self.transform_input:
                input_img = self.transform_input(input_img)
        if self.transform_target and target_img is not None:
                target_img = self.transform_target(target_img)
        return input_img, target_img
```

Thanks!

Do you have any questions?

sunjun7559012@yonsei.ac.kr
010 -8240-7559 | https://sites.google.com/view/yohanko