이상치 탐지 알고리즘 경진대회

C조 김동연, 김유환, 정다예

생각의 과정

1. baseline을 참고하여 작동하는 코드를 작성
2. 적용 가능한 이미지 사이즈를 늘리는 작업
3. 최적의 모델을 선정
4. 이미지 증강
5. epoch, batchsize, learning rate 최적화

data split

- train과 valid data를 7대 3 비율로 분리
- 인덱스를 같은 비율로 분리해 할당
- num_workers를 설정해 속도 향상

```
# validation 데이터로 사용할 비율
valid_size = 0.3

# validation idx
num_train = len(train_dataset)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_size * num_train))
train_idx, valid_idx = indices[split:], indices[:split]
```

```
13 # sampler 반환
14 train_sampler = SubsetRandomSampler(train_idx)
15 valid_sampler = SubsetRandomSampler(valid_idx)
16
17 # train_loader
18 train_loader = torch.utils.data.DataLoader(train_dataset,
19                                            batch_size=batch_size,
20                                            sampler=train_sampler,
21                                            num_workers=2)
22
23 # valid_loaer
24 valid_loader = torch.utils.data.DataLoader(train_dataset,
25                                            batch_size=batch_size,
26                                            sampler=valid_sampler,
27                                            num_workers=2)
```

이미지 사이즈 최적화

- 1024x1024이미지를 학습 할때마다 코드에서 리사이즈하니 시간이 매우 오래걸려서 로컬에서 이미지 리사이즈하기로 함
- powertoys라는 유틸을 사용하여 여러 이미지를 한번에 리사이즈하고 공유 폴더에서 다같이 사용

램 정리

- 512x512사이즈 파일, 256x256사이즈 파일을 만들어 사용했지만, 512사이즈는 colab램 부족이 자꾸 발생하여 램 정리하는 코드를 추가함
- del 변수, import gc, gc.collect()를 실행하면 변수가 할당되어 있지 않은 램을 정리함
-

```
1   train_loader = DataLoader(train_dataset, shuffle=True, batch_size=batch_size, num_workers=2)
```

```
1   del train_dataset
```

```
1   gc.collect()
```

```
0
```

```python
t_train_crop = T.Compose([
                    T.RandomCrop(400, 512), # 랜덤 크롭
                    T.Resize((512, 512)), #리사이즈
                    T.ToTensor(), #Tensor로 변환
                    T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)) # 이미지 Normalize
                    ])

t_train_rotate = T.Compose([
                    T.RandomRotation(360), # 랜덤 회전
                    T.ToTensor(),
                    T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
                    ])

t_train_vflip = T.Compose([
                    T.RandomVerticalFlip(1.0), # 랜덤 상하 반전
                    T.ToTensor(),
                    T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
                    ])

t_train_hflip = T.Compose([
                    T.RandomHorizontalFlip(1.0), # 랜덤 좌우 반전
                    T.ToTensor(),
                    T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
                    ])

t_test = T.Compose([
                    T.ToTensor(),
                    T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
                    ])
```

모델 선택

- timm 모듈을 이용하여 적용
- 다른 모델들보다 파라미터와 연산을 적게 사용하고 좋은 정확도를 내는 efficientnet을 사용
- timm.create_model('모델 이름')을 사용하면 원하는 모델을 바로 만들 수 있다.

```
class Network(nn.Module):
    def __init__(self):
        super(Network, self).__init__()
        self.model = timm.create_model('efficientnet_b2', pretrained=True, num_classes=88)
```

```
model_names = timm.list_models(pretrained=True)
print(model_names)

'''
['adv_inception_v3',
 'cspdarknet53',
 'cspresnext50',
 'densenet121',
 'densenet161',
 'densenet169',
 'densenet201',
 'densenetblur121d',
 'dla34',
 'dla46_c',
...
]
'''
```

이미지 증강

- test이미지들을 직접 확인 해 보니 회전한 이미지, 좌우를 잘라낸 이미지, 플립한 이미지 등이 있어서 그러한 부분을 증강해서 추가하기로 함
- 처음에는 로컬에서 라벨 별로 직접 증강했으나 GPU의 문제로 해당 방법으로 학습하는데에 어려움을 겪음
- state가 good이 아닌 항목 648개의 이미지만 증강하여 학습을 시도했지만 유의미한 스코어 상승을 느끼지 못함
- train data 총 21,385장
- 이상치만 증강하여 샘플 수의 밸런스를 맞추는 방법을 마지막에 더 시도해보자 했으나 시간부족으로 못함 <<< 나중에 개인적으로라도 도전해 보면 좋을듯

이미지 증강

- transform.compose를 5개로 만들어서 사용 default는 텐서 변환만, vflip은 상하반전, hflip은 좌우반전, crop은 좌우 잘라내기, rotate는 회전
- 1024x1024크기에서 가로 800사이즈로 잘라낸 이미지가 두드러지게 많아서 512x512를 400x512사이즈로 크롭함
- 이렇게 데이터셋을 5개 만들어서 torch.utils.data.ConcatDataset을 사용하여 하나로 합침

```
batch_size = 24
epochs = 40

train_dataset_default = Custom_dataset(np.array(train_imgs), np.array(train_labels), transform=t_test)
train_dataset_crop = Custom_dataset(np.array(train_imgs), np.array(train_labels), transform=t_train_crop)
train_dataset_rotate = Custom_dataset(np.array(train_imgs), np.array(train_labels), transform=t_train_rotate)
train_dataset_vflip = Custom_dataset(np.array(train_imgs), np.array(train_labels), transform=t_train_vflip)
train_dataset_hflip = Custom_dataset(np.array(train_imgs), np.array(train_labels), transform=t_train_hflip)
train_dataset = torch.utils.data.ConcatDataset([train_dataset_default, train_dataset_crop, train_dataset_rotate, train_dataset_vflip, train_dataset_
```

```
del train_imgs, train_dataset_default, train_dataset_crop, train_dataset_rotate, train_dataset_vflip, train_dataset_hflip, train_labels
```

```
gc.collect()
```

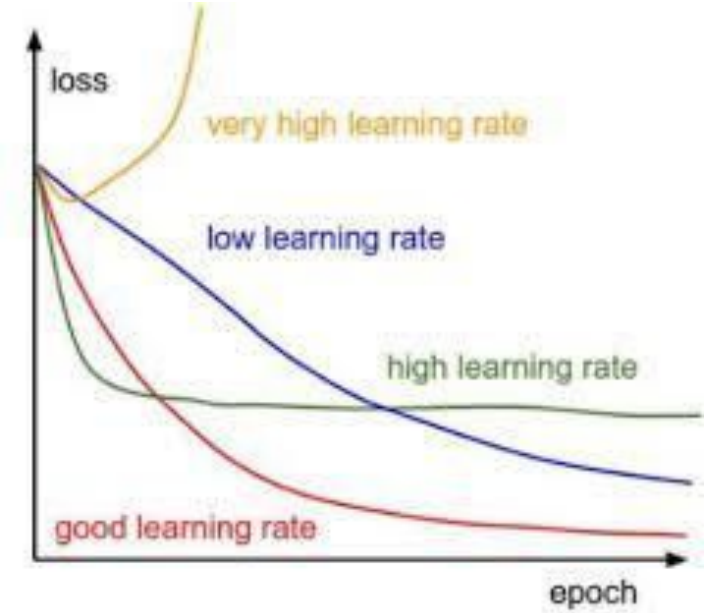
```
133
```

epoch 최적화

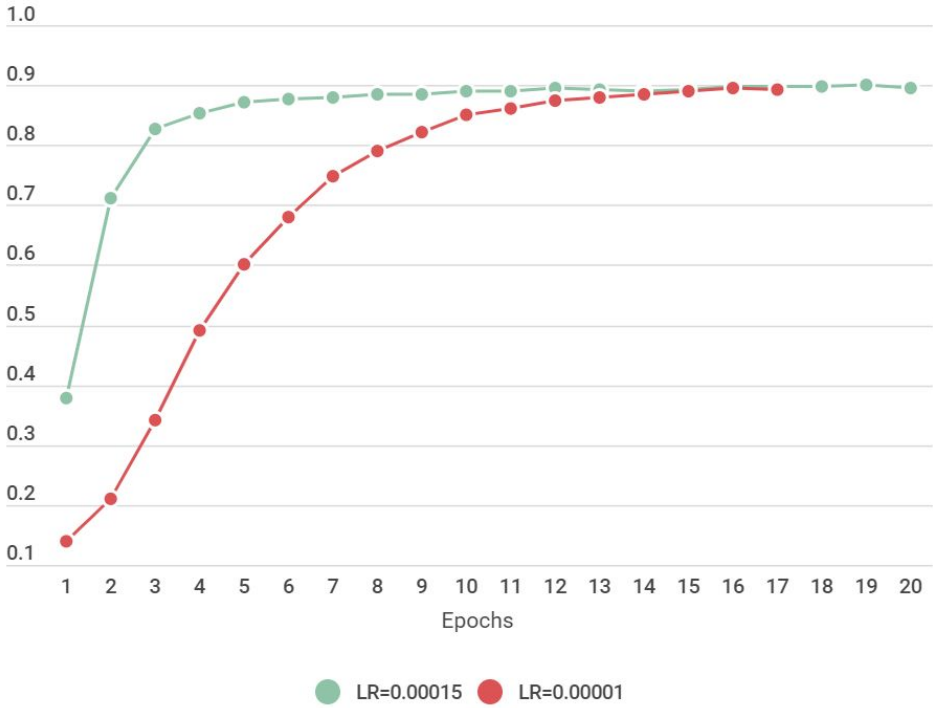
- 처음에 많으면 많을수록 좋다고 생각했지만 어느 시점부터 score가 수렴하는것을 확인
- 강사님이 주신 earlystop처럼 최고점이 2번 갱신 안될때 중지하기로 함

batch size, lr 최적화

- 처음에는 gpu램이 버티는 최대치로 batch설정 (36)
- 이후 batch와 LR의 관계에 대한 문서도 찾아보고 여러 문서에서 LR로 설정되어 있던 1e-3또는 1.5e-4같은 지수표기법 숫자가 몇을 의미하는지 알게 되어 조정함
- 최종적으로 batch = 16, lr = 0.0001으로 설정



Learning Rate & F1 score



should test 0.0001

더 공부해볼 점 / 아쉬운 점

- 사용 가능한 GPU가 한정되어 있어 다양한 시도를 해보지 못한것이 아쉽다.
- batch size와 lr과의 관계를 더 공부해보면 좋을 것 같다.

818954	230316_1.csv aug = transform.compose > concat efficientNetb1 batch=16, epoch=25 lr = 0.0001 edit	2023-03-17 07:28:43	0.8409057168 0.8485668704

감사합니다