캡스톤 디자인
‘딥페이크 탐지’

#6. 딥페이크 탐지 성능 및 안티포렌식 기법 조사

김지수, 김민지, 민지민

지난 캡스톤 회의 내용 -딥페이크 탐지기 및 안티포렌식 데이터셋 생성-

Anti-Forensic 데이터 셋 생성 (Sharpening, Denoising, JPEG Compression)

데이콘 딥페이크 탐지대회 Xception model (Center cropping 이용)

캐글 딥페이크 탐지대회 (FaceForenics에서 사전 학습된 모델로 detection)

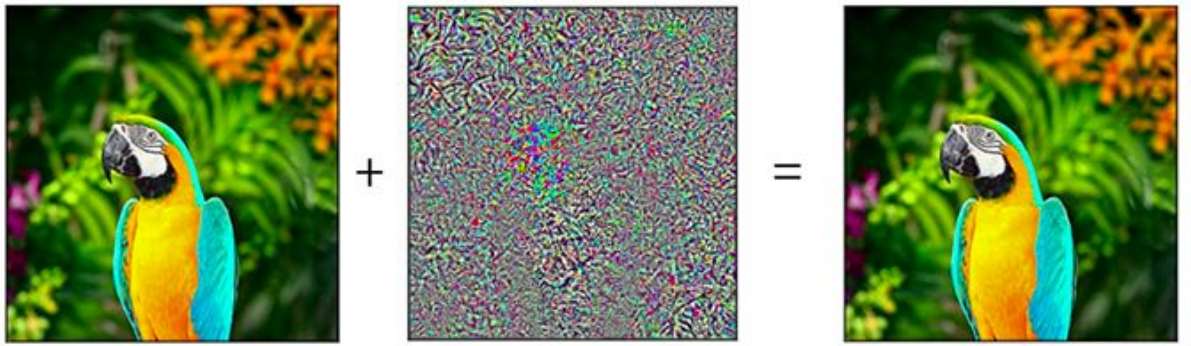
=> 이번주 목표

- 딥페이크 탐지기 성능 올리기
- 안티포렌식 기법 조사

FGSM

신경망을 통해 이미지를 앞으로 전파 후 손실을 계산하고, 그라디언트를 이미지에 역전파하고 손실 값을 최대화하는 방향으로 이미지 픽셀을 조금씩 이동

⇒ 이미지에서 노이즈가 눈에 띄는 정도는 엡실론에 따라 다르다. 값이 클수록 노이즈가 더 눈에 띄고, 엡실론을 높이면 네트워크가 잘못된 예측을 할 가능성도 높아짐.



FGSM with COLAB

TensorFlow를 사용하여 mobileNet V2 모델 로드

다양한 엡실론 값 적용

이미지 로드 -> 실행 -> 손실 기울기 얻음

signed_grad : 그라디언트가 이미지에 방향 효과만 적용함

그라디언트로 손실을 최대화하는 방향으로 이미지 픽셀을 조금씩 움직임

앱실론 값이 증가하면 노이즈가 더 잘 보이고 잘못된 예측에 대한 신뢰도가 높아진다

결과: 모델을 성공적으로 속임...!!

FGSM 이용한 데이터셋 생성

clean 이미지를 엡실론 0.02로 설정하여 fgsm 설정한 이미지

안티포렌식 도구

1. Steganography Studio : 다양한 필터로 구성할 수 있는 알고리즘 구현
2. CryptaPix : 이미지 파일 관리 및 암호화 프로그램

CryptaPix 사용해서 암호화해봤는데 암호화하면 파일 크기가 커진다는 것을 알 수 있음.

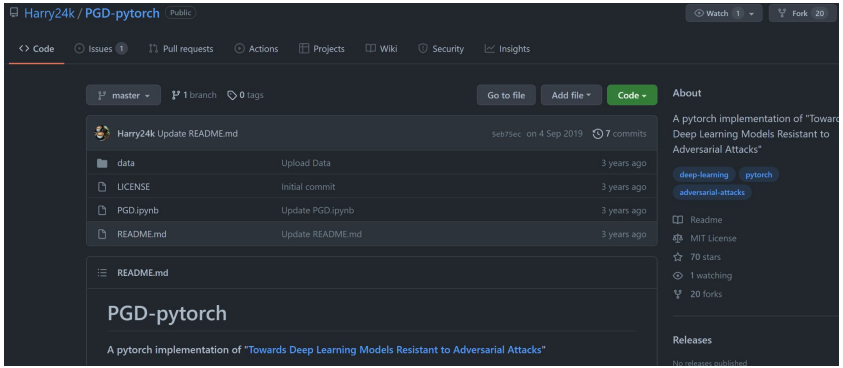
PGD(Projected Gradient Descent)

- FGSM 이후 3년 뒤에 나온 공격방법
- FGSM 방법을 조금 응용
- n번의 step만큼 공격을 반복해서 inner maximization을 수행
- local maxima를 찾는 최적해를 구하기 위해 first-order만을 사용한 공격 중에 PGD를 이용하는 것이 효과적
- FSGM은 1step gradient를 계산, PGD는 step 수에 따라 공격 강도가 강해짐

https://github.com/Harry24k/PGD-pytorch

PGD(Projected Gradient Descent)

<GitHub>



- real 이미지, fake 이미지 데이터셋을 입력하면 loss가 높아지는 방향으로 noise 추가해서 최종 생성한 이미지 저장

<주요 코드>

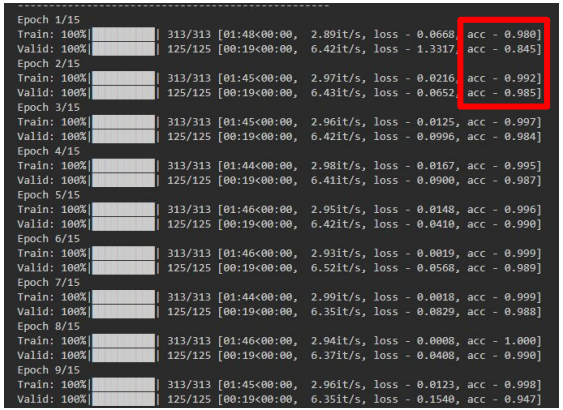
```python
# PGD Attack
# MNIST init
def pgd_attack(model, images, labels, eps=0.3, alpha=2/255, iters=40) :
    images = images.to(device)
    labels = labels.to(device)
    loss = nn.CrossEntropyLoss()

    ori_images = images.data

    for i in range(iters) :
        images.requires_grad = True
        outputs = model(images)

        model.zero_grad()
        cost = loss(outputs, labels).to(device)
        cost.backward()

        adv_images = images + alpha*images.grad.sign()
        eta = torch.clamp(adv_images - ori_images, min=-eps, max=eps)
        images = torch.clamp(ori_images + eta, min=0, max=1).detach_()

    return images
```

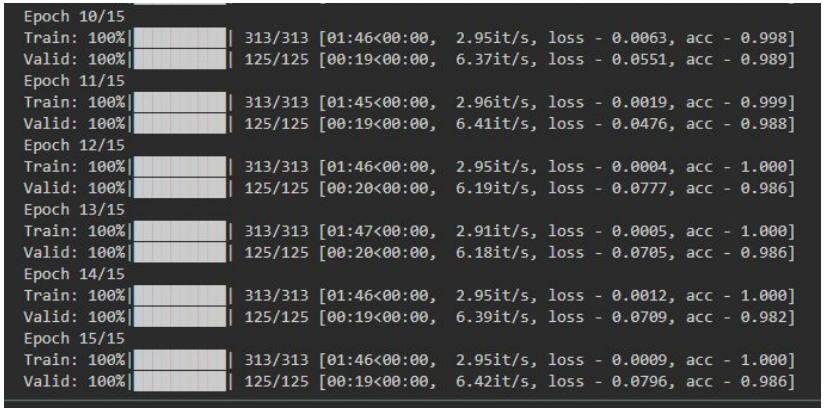
PGD(Projected Gradient Descent)

- real 이미지, fake 이미지 데이터셋을 입력하면 loss가 높아지는 방향으로 noise 추가해서 최종 생성한 이미지 저장

Deepfake Detector

- Dacon 데이터셋 전부 넣어서 학습 X → 1에폭 train에 약 12시간 소요
- Dacon 데이터셋 중 **train 5000장, test 2000장**만으로 성능 측정
- **epoch 15**로 변경, 추가 변경사항 없음
- validation accuracy → **0.98~0.99**
- 1 ~2에폭만으로 충분히 학습 가능을 확인





Deepfake Detector

- 지난 주 **cv2 내장 함수 사용**해서 생성한 anti-forensics 데이터셋으로 성능 측정
- 두가지 방법으로 측정
    1) dacon 참조한 코드의 **inference 코드**
    2) train/valid 측정 코드에서 **validation 데이터셋을 변경**

- **성능 하락하지 않음**을 확인
- 지난주 사용한 변형 방법이 성능에 영향을 끼치지 않는 것 같음(불확실)

Deepfake Detector

1) inference 코드

```
## overfit을 막기 위하여 한 번만 학습
criterion = nn.CrossEntropyLoss().cuda()

print('-' * 50)

acc = validate(valid_loader, model, criterion)
```

```
--------------------------------------------------
Valid: 100%|██████████| 1110/1110 [00:18<00:00, 61.39it/s, loss - 0.0009, acc - 1.000]
```

2) validation 데이터 변경

```
-----------------------------------------------
Epoch 1/15
Train: 100%|██████████| 313/313 [04:34<00:00,  1.14it/s, loss - 0.0664, acc - 0.979]
Valid: 100%|██████████| 35/35 [00:13<00:00,  2.56it/s, loss - 0.0369, acc - 0.979]
Epoch 2/15
Train: 100%|██████████| 313/313 [01:51<00:00,  2.81it/s, loss - 0.0247, acc - 0.989]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.82it/s, loss - 0.0000, acc - 1.000]
Epoch 3/15
Train: 100%|██████████| 313/313 [01:50<00:00,  2.84it/s, loss - 0.0179, acc - 0.994]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.73it/s, loss - 0.0002, acc - 1.000]
Epoch 4/15
Train: 100%|██████████| 313/313 [01:49<00:00,  2.87it/s, loss - 0.0245, acc - 0.993]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.73it/s, loss - 0.0000, acc - 1.000]
Epoch 5/15
Train: 100%|██████████| 313/313 [01:48<00:00,  2.89it/s, loss - 0.0148, acc - 0.996]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.73it/s, loss - 0.0064, acc - 0.998]
Epoch 6/15
Train: 100%|██████████| 313/313 [01:49<00:00,  2.85it/s, loss - 0.0037, acc - 0.999]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.74it/s, loss - 0.0069, acc - 0.997]
Epoch 7/15
Train: 100%|██████████| 313/313 [01:47<00:00,  2.90it/s, loss - 0.0040, acc - 0.999]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.87it/s, loss - 0.0000, acc - 1.000]
Epoch 8/15
Train: 100%|██████████| 313/313 [01:48<00:00,  2.90it/s, loss - 0.0013, acc - 1.000]
Valid: 100%|██████████| 35/35 [00:08<00:00,  3.93it/s, loss - 0.0000, acc - 1.000]
Epoch 9/15
Train: 100%|██████████| 313/313 [01:46<00:00,  2.95it/s, loss - 0.0002, acc - 1.000]
Valid: 100%|██████████| 35/35 [00:08<00:00,  3.90it/s, loss - 0.0000, acc - 1.000]
Epoch 10/15
Train: 100%|██████████| 313/313 [01:48<00:00,  2.87it/s, loss - 0.0017, acc - 1.000]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.57it/s, loss - 0.0005, acc - 1.000]
Epoch 11/15
Train: 100%|██████████| 313/313 [01:56<00:00,  2.69it/s, loss - 0.0013, acc - 0.999]
Valid: 100%|██████████| 35/35 [00:08<00:00,  3.90it/s, loss - 0.0005, acc - 1.000]
Epoch 12/15
Train: 100%|██████████| 313/313 [01:48<00:00,  2.89it/s, loss - 0.0001, acc - 1.000]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.74it/s, loss - 0.0000, acc - 1.000]
Epoch 13/15
Train: 100%|██████████| 313/313 [01:46<00:00,  2.94it/s, loss - 0.0001, acc - 1.000]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.78it/s, loss - 0.0001, acc - 1.000]
Epoch 14/15
Train: 100%|██████████| 313/313 [01:48<00:00,  2.90it/s, loss - 0.0001, acc - 1.000]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.72it/s, loss - 0.0001, acc - 1.000]
Epoch 15/15
Train: 100%|██████████| 313/313 [01:48<00:00,  2.87it/s, loss - 0.0003, acc - 1.000]
Valid: 100%|██████████| 35/35 [00:09<00:00,  3.70it/s, loss - 0.0002, acc - 1.000]
```