2023 암호분석경진대회

1번 문제

[1-1번 문제 답안]

1. PGD attack 설명

1.1 PGD 소개 및 수식

적대적 공격은 특정 노이즈 값 (perturbation)을 이용해 의도적으로 오분류를 이끌어내는 것이다.
대표적으로 FGSM과 PGD 방법이 존재한다.

$$x^{i} = (x^{i-1} + \alpha \,.\, \mathrm{sign}(\nabla_x l(h(x^{i-1}), y))$$

그림 1 PGD 방법 수식

alpha는 step size (noise magnitude)를 의미하고 x^i는 i번째 adversarial 이미지이다 (x는 인풋 이미지, l은 cost function, h는 NN model을 의미하며, ∇는 gradient를 의미한다.)
step이 T개 라면, step size는 $\varepsilon/T \leq \alpha < \varepsilon$ 사이의 값을 가지게 된다.
한편, 적대적 공격에서는 노이즈의 Lp norm은 다음 처럼 사이의 값을 가질 것을 요구한다
($x*$ 은 x의 vicinity임을 가정, p=0,1,2…).

$$\|x* - x\|_p \leq \varepsilon,$$

그림 2 Lp norm 값

PGD는 'constrained optimization problem' 에 직면하게 되는데, model의 loss를 최대화하면서 perturbation을 특정 값(epsilon) 보다 작게 설정해야 한다. 이를 주로 흔히 L infinity norm이라고 부른다. 해당 노이즈를 육안으로는 구분이 거의 되지 않게 실제 이미지에 더한다.

1.2 PGD 공격 육안 구분 증명

epsilon 값을 조절함에 따라, 육안으로 구분이 되는 적대적 이미지가 생성된다는 것을 알 수 있다.
perturbation 범위가 epsilon에 따라 설정되고, perturbation을 실제 이미지에 더하기 때문에, 허용 값을 크게 설정한다면 실제 이미지와 구분이 되는 적대적 이미지가 생성 될 것이다.

alpha, step 수 등 이외의 값을 고정한 후, eps만 조절한다면 아래와 같은 결과를 확인할 수 있다. 확인 결과, eps = 0.02 부터 육안으로 거의 구분이 불가능 했으며 eps = 0.01은 육안으로 구분이 불가능했다. 따라서 해당 값들로 고정 한 후, PGD 공격을 수행하면 육안으로는 공격 전후 이미지를 거의 구분할 수 없다.

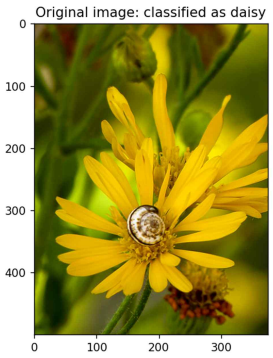

그림 3 원본 사진

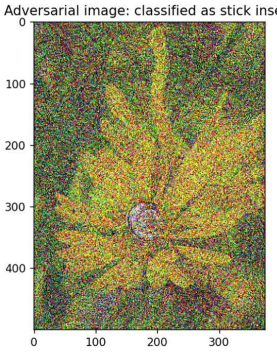

그림 4 eps = 1.0

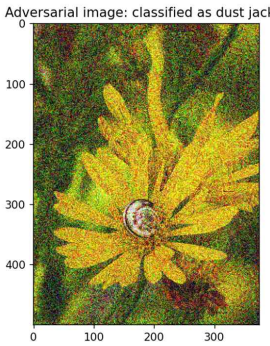

그림 5 eps = 0.5

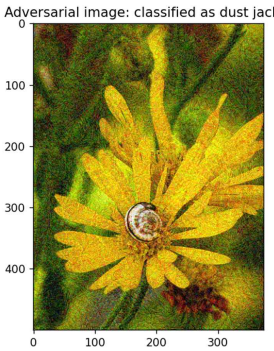

그림 6 eps = 0.25

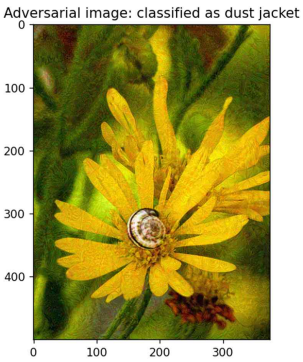

그림 7 eps = 0.1

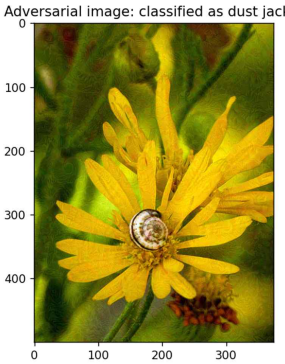

그림 8 eps = 0.05

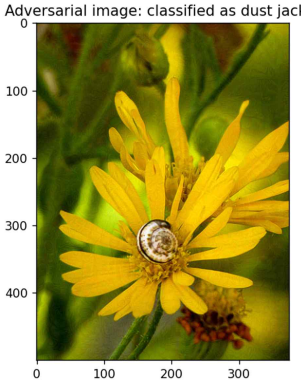

그림 9 eps = 0.02

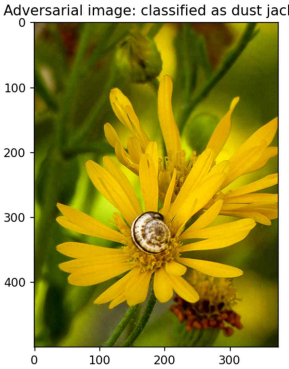

그림 10 eps = 0.01

2. 공격 코드 설명

2.1 코드 개괄

공격 코드는 크게 모델과 데이터 로드 하는 부분과, 이미지 형태를 변환 후 공격 전 분류 값 추출, 공격 수행 후 분류 값 추출, 공격 검증으로 나뉜다.

1) 우선, RestNet-18 모델을 준비해준다.

모델의 경우, 문제에서 주어진 대로 ResNet-18을 이용하였다. torchvision.models.resnet18을 통해 Pre trained 된 RestNet-18을 로드 한다. 마찬가지로, ResNet-18 분류 라벨도 json 형태로 raw.githubuser content.com/anishathalye/imagenet-simple-labels/master/imagenet-simple-labels.json에서 받아온다. ResNet-18 모델은 주어진 데이터를 해당 라벨로 분류하게 된다.

2) 다음으로, ImageNet 데이터를 로드 한다.

dataset폴더 하위에 있는 이미지 파일을 데이터로 선정한다. dataset폴더에는 https://figshare.com/articl es/dataset/RGB_Image_Dataset/14542434에서 다운로드 가능한 ImageNet 이미지 데이터셋들을 포함시킨다. 구현한 코드에서는 다양한 이미지들 중 하나를 랜덤으로 선정해 공격을 수행하도록 하였다.

3) 이미지 형태를 변환 시킨다.

ToTensor() 을 이용해 4차원 텐서 형식으로 변환한다. PyTorch 에서는 주로 텐서 형태의 데이터로 작업한다. 필수는 아니지만 이미지 정규화 후 공격을 진행하고 싶은 경우, Normalize() 함수를 이용한다. mean은 평균을 의미 하고, std는 표준편차를 의미하는데, Pretrained 된 모델은 ImageNet 데이터셋을 학습했기에 정해진 값들을 mean 과 std로 고정해 정규화 가능하다.

4) 공격 전 분류 값을 추출한다.

이미지 데이터셋이 주어졌을 때 Pretrained 모델이 예측하는 값 상위 5개 결과를 받아와서, 분류 라벨에도 존재하는 결과 중 가장 상위에 있는 값을 공격 전 분류 값으로 판정한다.

5) PGD 공격을 수행한다.

1번에 나와있는 수식을 구현해 PGD 공격을 수행한다.

6) 공격 후 분류 값을 추출한다.

4번과 동일한 방식으로, 이미지는 PGD 공격이 수행된 적대적 이미지가 주어졌을 때의 분류 값을 판정한다.

7) 공격 성공을 검증한다.

분류 전후 값이 다르면 공격에 성공했다고 판정한다.

적대적 이미지가 원본 이미지와 육안 구분이 되지 않는다는 점은 PGD 설명 부분에서 증명되었다.

8) (출력) 원본, 노이즈, 공격 이미지를 출력시킨다.

가장 좌측에서는 원본 사진, 가운데는 추가한 노이즈, 가장 우측에는 적대적 이미지를 출력시킨다.

2.2 공격 구현 설명

1) 각종 변수값

코드 구현에서는 ITER (step 수), alpha, epsilon 값을 고정해두었다.

https://arxiv.org/pdf/1706.06083.pdf 에 의하면, step 수는 주로 7, 40과 같은 값들이 사용되는데, 40으로 고정하였고, 앞서 언급된 수식의 조건을 만족하고 육안으로 구분이 안되도록 epsilon과 alpha를 설정했다.

2) Perturbation 생성

```
if random_start:
    adv_image =image +torch.empty_like(image).uniform_(-eps,eps)
    adv_image =torch.clamp(adv_image,0,1)
    print(f"[*] Start image' = image + ε, (|ε| < {eps})")
else:
    adv_image =torch.clamp(image,0,1)
    print(f"[*] Start image' = image + ε, (ε = 0)")
```

입력 이미지와 동일한 크기를 가지지만 채널을 1개 사용하는 노이즈를 생성하는 부분이다.

노이즈 값은 음수가 될 수 있기 때문에 평균이 0 인 uniform랜덤 값으로 설정한다.

원본 이미지에 노이즈를 더하고 결과가 0과 1사이에 속하도록 자르게 된다.

3) Iteration 과정

```
device =torch.device("cuda:0"if torch.cuda.is_available()else "cpu")
model =model.to(device)
image =image.to(device)
_,target =torch.max(prediction,1)
target =target.to(device)

loss =torch.nn.CrossEntropyLoss()

ITER =40
for i in range(0,ITER):
    adv_image.requires_grad =True
    output =model(adv_image)

    model.zero_grad()
    cost =loss(output,target).to(device)
    cost.backward()

    attack_images =adv_image +alpha*adv_image.grad.sign()
    eta =torch.clamp(attack_images -image,min=-eps,max=eps)
    print(f"[*] η = ε + α * sign(image')")
    adv_image =torch.clamp(image +eta,min=0,max=1).detach_()
    print(f"[*] image* = image + η")

perturbation =adv_image -image
```

loss 함수를 설정하고, iteration에 들어가게 된다.
required_grad 속성 값을 True로 해서 torch.Tensor연산을 추적한 뒤, 앞서 원본 이미지에 노이즈를 더한 이미지에 대한 모델의 판단 결과를 확인한다.
손실 함수를 미분해 기울기를 계산한다. 우리는 untargeted attack을 진행했기 때문에 앞서 소개한 수식에 의해 기울기를 이용하여 손실 함수가 증가하는 방향으로 변화시켰다. 이후, 노이즈가 eps 범위 안에 들도록 설정한다.

adv_image는 따라서, 최종적으로 만들어진 공격 이미지를 의미한다. 이후 perturbation은 결과를 보여줄 때 사용하기 위해 최종 공격 이미지 - 원본 이미지를 하여 구해두었다.

3. 공격 결과

3.1 실행 방법

1) dataset폴더 하위에 공격 대상 이미지 데이터셋을 옮긴다.

2) TARGET_DIR에 dataset 폴더경로를 명시한다.

3) solver.py를 실행시킨다.

```
# ResNet-18 분류
model = torchvision.models.resnet18(weights='DEFAULT').eval()
print(f"[*] Loaded pretrained ResNet-18.")

# ResNet-18 분류 라벨
url = 'https://raw.githubusercontent.com/anishathalye/imagenet-simple-labels/master/imagenet-simple-labels.json'
labels = requests.get(url).json()
print(f"[*] Loaded human readable output label mapping (total {len(labels)} labels)")

# ImageNet 데이터 로드
# 아래 폴더에서 임의의 사진 1개 로드
TARGET_DIR = "dataset 경로"
files = os.listdir(TARGET_DIR)
file = random.choice(files)
img = Image.open(os.path.join(TARGET_DIR, file))
print(f"[*] Loaded random image from ImageNet: {file}")

# 이미지 형태를 4차원 텐서로 변환
# 이미지 normalization 후 공격 진행하고 싶은 경우 아래 변수 수정
DO_NORMALIZE = False
if DO_NORMALIZE:
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\zzzmi\OneDrive\바탕 화면\CryptoChallange23\1>  c:; cd 'c:\Users\zzzmi\OneDrive\바탕 화면\CryptoChallange23\1'; & '
C:\Users\zzzmi\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\zzzmi\.vscode\extensions\ms-python.python-2023.12
.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '63962' '--' 'C:\Users\zzzmi\OneDrive\바탕 화면\CryptoChalla
nge23\1\pngwna\1\solver.py'
[*] Loaded pretrained ResNet-18.
[*] Loaded human readable output label mapping (total 1000 labels)
[*] Loaded random image from ImageNet: ILSVRC2017_test_00000801.JPEG
[*] Preprocessing with Image -> ToTensor -> Tensor
[*] Original image is classified as 'giant panda'
[*] Start image' = image + ε, (|ε| < 0.01)
[*] η = ε + α * sign(image')
[*] image* = image + η
[*] Adversarial image is classified as 'handkerchief'
[+] PGD perturbation attack successful! (giant panda != handkerchief)
```

그림 11 실행 화면

3.2 ImageNet 대상 결과

코드 실행 결과, 다음과 같은 화면을 확인할 수 있다. 해당 이미지는 ILSVRC2017_test_00000801.JPEG 로, https://figshare.com/articles/dataset/RGB_Image_Dataset/14542434 의 이미지넷 데이터셋 중 하나다. 또한 해당 url에서 제공하는 파일 중 ILSVRC_eval 폴더 내 데이터셋 분류 결과를 모두 모아 표로 정리했다.

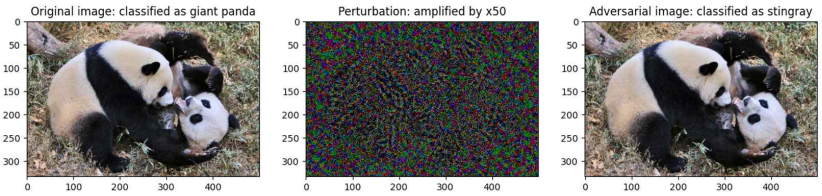

그림 12 ILSVRC2017_test_00000801.JPEG 대상 공격 실행 결과 화면

Pretrained model이 원본 이미지 분류를 바르게 수행한 26건의 ImageNet 데이터셋 대상 PGD 공격을 성공적으로 수행할 수 있었다. PGD 수식에 의해 공격이 바르게 수행됐으므로 육안 구분 시 공격 전 후 이미지가 동일해 보인다는 것도 직접 확인해봤다.

파일명	원본 이미지 분류 결과	적대적 이미지 분류 결과	육안 확인시 동일 여부
ILSVRC2017_test_00000021.JPEG	daisy	dust jacket	O
ILSVRC2017_test_00000101.JPEG	cowboy hat	mosquito net	O
ILSVRC2017_test_00000105.JPEG	pot pie	mosquito net	O
ILSVRC2017_test_00000114.JPEG	giant panda	mosquito net	O
ILSVRC2017_test_00000117.JPEG	red panda	paper towel	O
ILSVRC2017_test_00000118.JPEG	dromedary	mosquito net	O
ILSVRC2017_test_00000121.JPEG	shoal	American alligator	O
ILSVRC2017_test_00000140.JPEG	lacewing	pillow	O
ILSVRC2017_test_00000146.JPEG	dragonfly	pillow	O
ILSVRC2017_test_00000173.JPEG	husky	paper towel	O
ILSVRC2017_test_00000179.JPEG	tray	hippopotamus	O
ILSVRC2017_test_00000196.JPEG	dust jacket	automated teller machine	O
ILSVRC2017_test_00000201.JPEG	mixing bowl	plunger	O
ILSVRC2017_test_00000801.JPEG	giant panda	stingray	O
ILSVRC2017_test_00000802.JPEG	soccer ball	handkerchief	O
ILSVRC2017_test_00000806.JPEG	Alpine ibex	lampshade	O
ILSVRC2017_test_00000809.JPEG	window shade	rocking chair	O
ILSVRC2017_test_00000816.JPEG	beaver	shoal	O
ILSVRC2017_test_00000818.JPEG	zebra	stingray	O
ILSVRC2017_test_00000823.JPEG	harp	refrigerator	O
ILSVRC2017_test_00000827.JPEG	dragonfly	pillow	O
ILSVRC2017_test_00000838.JPEG	lab coat	bathtub	O
ILSVRC2017_test_00000841.JPEG	red panda	mosquito net	O
ILSVRC2017_test_00000845.JPEG	microwave oven	dome	O
ILSVRC2017_test_00000849.JPEG	wing	pillow	O
ILSVRC2017_test_00000850.JPEG	gas mask	mosquito net	O

4. 전체 코드 (solver.py)

코드 실행 시 필요한 requirements.txt는 다음과 같다.

```
torch
torchvision
pillow
requests
matplotlib
```

```
import os
import random

import torch
import torchvision
import torchvision.transforms as transforms

import requests

from PIL import Image

import matplotlib.pyplot as plt

# 모델, 테스트셋 준비
# ResNet-18 준비
model =torchvision.models.resnet18(weights='DEFAULT').eval()
print(f"[*] Loaded pretrained ResNet-18.")

# ResNet-18 분류 라벨
url ='https://raw.githubusercontent.com/anishathalye/imagenet-simple-labels/master/imagenet-simple-labels.json'
labels =requests.get(url).json()
print(f"[*] Loaded human readable output label mapping (total {len(labels)} labels)")

# ImageNet 데이터 로드
# 아래 폴더에서 임의의 사진 1개 로드
TARGET_DIR ="dataset"
files =os.listdir(TARGET_DIR)
file =random.choice(files)
img =Image.open(os.path.join(TARGET_DIR,file))
print(f"[*] Loaded random image from ImageNet: {file}")


# 이미지 형태를 4차원 텐서로 변환
# 이미지 normalization 후 공격 진행하고 싶은 경우 아래 변수 수정
DO_NORMALIZE =False
if DO_NORMALIZE:
    MEAN =[0.485,0.456,0.406]
    STD =[0.229,0.224,0.225]
    transform =transforms.Compose(
        [
            transforms.ToTensor(),
            transforms.Normalize(MEAN,STD)]
        )
    print(f"[*] Preprocessing with Image -> ToTensor -> Normalize -> Tensor")
else:
    transform =transforms.Compose(
        [transforms.ToTensor()]
        )
    print(f"[*] Preprocessing with Image -> ToTensor -> Tensor")
image =transform(img).unsqueeze(0)
```

```
# 공격 전 분류값 추출
prediction =model(image)
values,indexes =torch.topk(prediction,5)
classifications =[labels[index]for index in indexes[0].tolist()]
origin_top1 =classifications[0]
print(f"[*] Original image is classified as '{origin_top1}'")

# PGD 공격 파라미터
eps =0.01
alpha =2/255
random_start =True

# PGD 공격 수행
# 데이터 로드
device =torch.device("cuda:0"if torch.cuda.is_available()else "cpu")
model =model.to(device)
image =image.to(device)
_,target =torch.max(prediction,1)
target =target.to(device)

loss =torch.nn.CrossEntropyLoss()

if random_start:
    adv_image =image +torch.empty_like(image).uniform_(-eps,eps)
    adv_image =torch.clamp(adv_image,0,1)
    print(f"[*] Start image' = image + ε, (|ε| < {eps})")
else:
    adv_image =torch.clamp(image,0,1)
    print(f"[*] Start image' = image + ε, (ε = 0)")

ITER =40
for i in range(0,ITER):
    adv_image.requires_grad =True
    output =model(adv_image)

    model.zero_grad()
    cost =loss(output,target).to(device)
    cost.backward()

    attack_images =adv_image +alpha*adv_image.grad.sign()
    eta =torch.clamp(attack_images -image,min=-eps,max=eps)
    print(f"[*] η = ε + α * sign(image')")
    adv_image =torch.clamp(image +eta,min=0,max=1).detach_()
    print(f"[*] image* = image + η")

perturbation =adv_image -image

# 공격 후 분류값 추출
prediction =model(adv_image)
values,indexes =torch.topk(prediction,5)
classifications =[labels[index]for index in indexes[0].tolist()]
adv_top1 =classifications[0]
print(f"[*] Adversarial image is classified as '{adv_top1}'")

# 공격 성공 검증
if origin_top1 !=adv_top1:
    print(f"[+] PGD perturbation attack successful! ({origin_top1} != {adv_top1})")
else:
    print(f"[+] PGD perturbation attack failed! ({origin_top1} == {adv_top1})")

# 원본, 노이즈, 공격 이미지 출력
```

```
_,figures =plt.subplots(1,3,figsize=(15,5))

figures[0].imshow(image.squeeze(0).permute(1,2,0).clamp(0,1))
figures[0].set_title(f"Original image: classified as {origin_top1}")

MULTIPLIER =50
figures[1].imshow(perturbation.squeeze(0).permute(1,2,0).clip(0,255)*MULTIPLIER)
figures[1].set_title(f"Perturbation: amplified by x{MULTIPLIER}")

figures[2].imshow(adv_image.squeeze(0).permute(1,2,0).clamp(0,1))
figures[2].set_title(f"Adversarial image: classified as {adv_top1}")

plt.show()
```

5. 참고 문헌

[1] https://arxiv.org/pdf/1706.06083.pdf

[2] https://www.scitepress.org/Papers/2022/109384/109384.pdf

[3] https://github.com/Harry24k/PGD-pytorch/tree/master

[4] https://wikidocs.net/157285

[5] https://adversarial-ml-tutorial.org/adversarial_examples/

[6] https://github.com/ndb796/Deep-Learning-Security-Basic-for-KISA/blob/master/Shadow_Attack_Tutorial.ipynb

[7] https://arxiv.org/pdf/1706.06083.pdf

[1-2번 문제 답안]

1. Patch attack 설명

논문에서 제시한 Adversarial Patch attack의 경우, 이미지와 무관한, 매우 "자명한" 패치를 생성하는 식으로 공격을 진행한다. "자명하다"는 것은 가장 핵심인 오브젝트를 의미한다.
따라서 Patch attack은 사진에 담긴 여러 물체 중에서, 가장 "자명한" 오브젝트를 탐지해 하나의 target label만 참으로 판정하는 이미지 분류의 특성을 exploit 하는 공격이다.

2. 사례 제시

2.1.1 원본 교통 표지판 사진

pretrained 모델에 의해 traffic sign으로 제대로 인식되는 2가지 사진을 사용했다. 각기 다른 방법으로 패치를 붙였다.

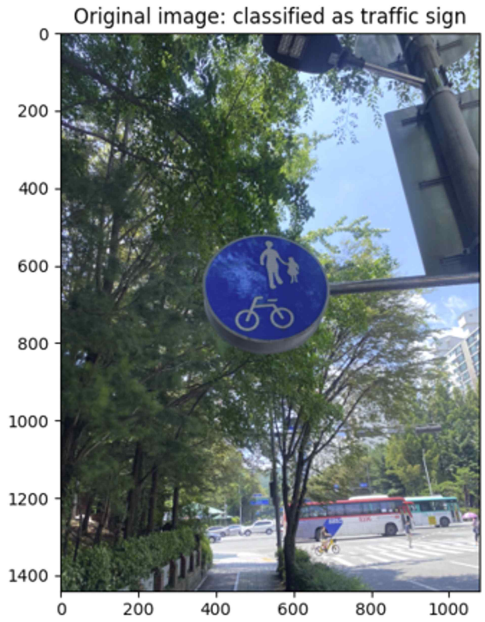

그림 13 원본 사진 1

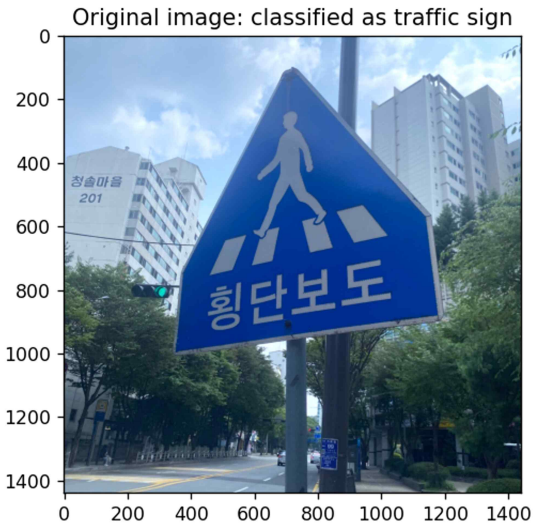

그림 14 원본 사진 2

표지판과 함께 패치를 촬영해 오인식시키는 방법과, 촬영된 사진에 패치를 붙이는 방법을 사용했다.

2.1.2 패치가 붙은 사진

그림 15 촬영 사진에 패치를 붙이는 방법

그림 16 패치를 함께 촬영하는 방법

좌측은 촬영된 사진에 패치를 붙이는 방법, 우측은 패치와 함께 사진을 촬영하는 방법이다.

2.2 적대적 이미지 생성

문제에서 제시한 Figure 5를 컬러로 프린트 후, 뒷부분에 하드보드지를 덧대고 교통표지판과 직접 촬영하기 위해 뒤에 나무막대를 붙여 실험을 진행했다.

Figure 5: Printable sticker illustrating the attack. For best results keep stickers within 20 degrees of the vertical alignment shown here. This patch was generated by the white-box ensemble method described in Section 3. We observed some transferability of this patch to the third party Demitasse application (the patch was not designed to fool this application). However, in order to be effective the size of the patch needs to be larger than what is demonstrated in Figure 1, which is a white box attack on the models described in Section 3.

그림 17 논문에서 주어진 패치

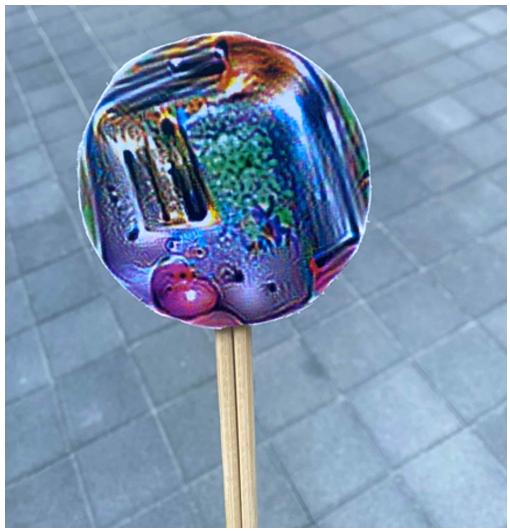

그림 18 패치를 변형해 실험 진행

영상에서는 바닥에 오인식시킬 물건을 함께 두고 사진을 촬영할 수 있으나, 교통표지판 특성상 사진을 더 간편하고 정확하게 촬영하기 위해 아래와 같은 패치를 만들었다. 논문에서 설명한 무늬가 위로 오도록 하였다.

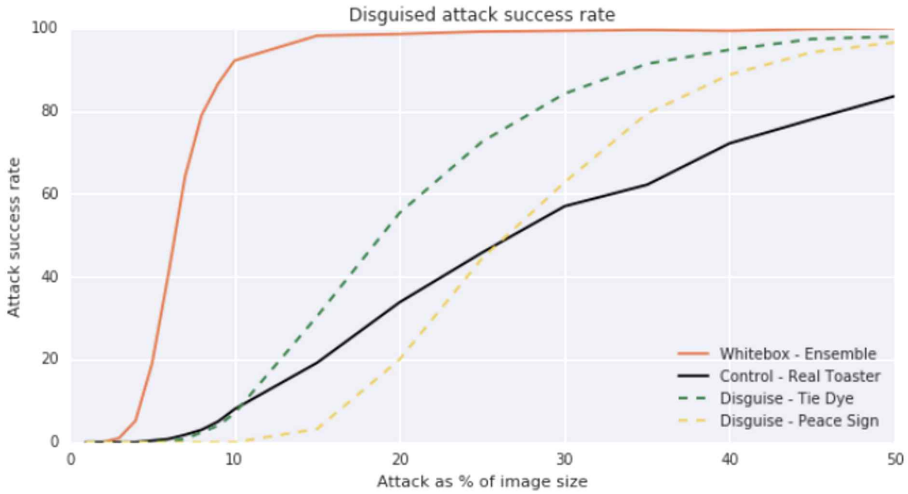

그림 19 패치 크기 관련 자료

이때, 문제에서 주어진 whitebox-ensemble 사진의 경우 전체 이미지의 20퍼센트 크기로 패치를 제작했을 때 attack 성공률이 가장 높다는 것을 확인할 수 있다.

3. 오인식 결과

3.1 사용 코드

공격 대상 모델은 VGG16 이다.

```
import os
import random

import torch
import torchvision
import torchvision.transforms as transforms

import requests

from PIL import Image

import matplotlib.pyplot as plt

# 모델, 테스트셋 준비
# VGG16 준비
model =torchvision.models.vgg16(weights='DEFAULT').eval()
print(f"[*] Loaded pretrained VGG16.")


# 분류 라벨
url ='https://raw.githubusercontent.com/anishathalye/imagenet-simple-labels/master/imagenet-simple-labels.json'
labels =requests.get(url).json()
print(f"[*] Loaded human readable output label mapping (total {len(labels)} labels)")

# 원본, 공격 이미지 로드
TARGET_DIR ="eval"
ORIGIN ="orig_image.png"
ADV ="adv_image.png"
PATCH ="patch.png"
orig_img =Image.open('원본 사진 경로').convert("RGB")
adv_img =Image.open('패치가 붙은 사진 경로').convert("RGB")
patch =Image.open('패치 사진 경로').convert("RGB")
print(f"[*] Loaded original, adversarial image, and patch image")
```

```
# 이미지 형태를 4차원 텐서로 변환
# 이미지 normalization 후 공격 진행하고 싶은 경우 아래 변수 수정
MEAN =[0.485,0.456,0.406]
STD =[0.229,0.224,0.225]
transform =transforms.Compose(
    [
        transforms.ToTensor(),
        transforms.Normalize(MEAN,STD)]
    )
print(f"[*] Preprocessing with Image -> ToTensor -> Normalize -> Tensor")

orig_image =transform(orig_img).unsqueeze(0)
adv_image =transform(adv_img).unsqueeze(0)

# 공격 전 분류값 추출
prediction =model(orig_image)
values,indexes =torch.topk(prediction,5)
classifications =[labels[index]for index in indexes[0].tolist()]
origin_top1 =classifications[0]
print(f"[*] Original image is classified as '{origin_top1}'")

# 공격 전 분류값 추출
prediction =model(adv_image)
values,indexes =torch.topk(prediction,5)
classifications =[labels[index]for index in indexes[0].tolist()]
adv_top1 =classifications[0]

# 공격 성공 검증
if origin_top1 !=adv_top1:
    print(f"[+] Patch attack successful! ({origin_top1} != {adv_top1})")
else:
    print(f"[+] Patch attack failed! ({origin_top1} == {adv_top1})")

# 이미지 denormalize 진행
for t,m,s in zip(orig_image,MEAN,STD):
    t.mul_(s).add_(m)
orig_image =orig_image.squeeze(0).permute(1,2,0).clip(0,1)

for t,m,s in zip(adv_image,MEAN,STD):
    t.mul_(s).add_(m)
adv_image =adv_image.squeeze(0).permute(1,2,0).clip(0,1)


# 원본, 노이즈, 공격 이미지 출력
_,figures =plt.subplots(1,3,figsize=(15,5))

figures[0].imshow(orig_image)
figures[0].set_title(f"Original image: classified as {origin_top1}")

figures[1].imshow(patch)
figures[1].set_title(f"Adversarial patch")

figures[2].imshow(adv_image)
figures[2].set_title(f"Adversarial image: classified as {adv_top1}")

plt.show()
```

3.2 코드 결과 (인식 결과)

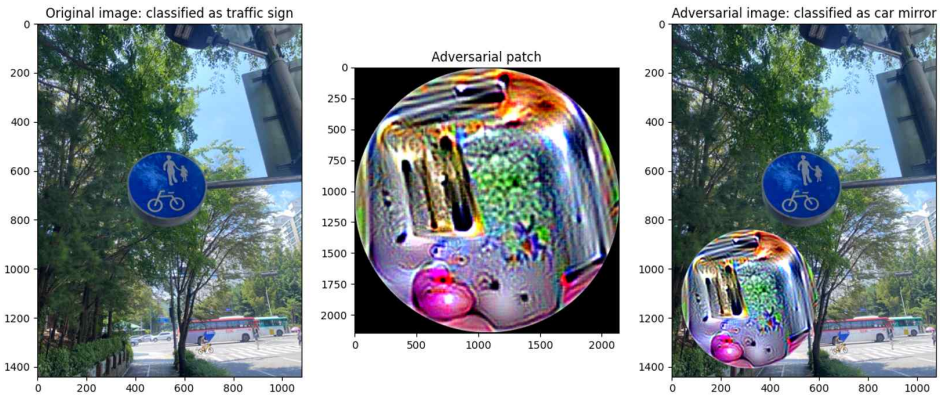

그림 20 원본 사진 1에 대한 patch attack 결과

좌측은 교통 표지판 사진을 촬영했고, 우측 사진은 앞서 설명한 패치를 사진에 붙인 것이다. 좌측 사진은 traffic sign으로 분류 되었고, 우측 사진은 car mirror로 분류되었다.

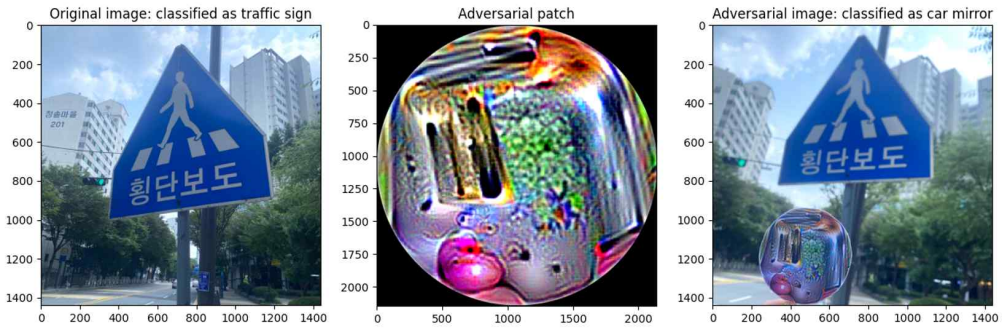

그림 21 원본 사진 2에 대한 patch attack 결과

좌측은 교통 표지판 사진을 촬영했고, 우측 사진은 앞서 설명한 패치와 함께 사진 촬영을 한 것이다. 좌측 사진은 traffic sign으로 분류 되었고, 우측 사진은 마찬가지로 car mirror로 분류되었다.

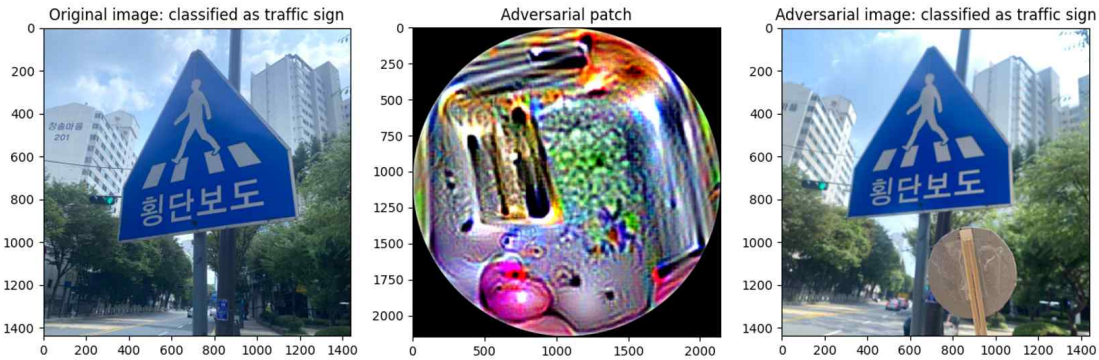

그림 22 잘못된 patch를 사용했을 때의 결과

adversarial patch가 효과가 있는지 확인하기 위해, 논문에서 제공된 패치 대신, 같은 크기의 임의의 스티커와 함께 촬영된 우측 사진의 경우, 동일하게 traffic sign으로 분류 됐음을 확인할 수 있었다. 따라서 패치 없이는 공격에 성공할 수 없음을 확인했다.

패치가 붙은 교통 표지판 사진을 모두 동일하게 car mirror이라는 제3의 오브젝트로 분류됐음을 확인했을때, 공격이 어느정도 성공했음을 확인할 수 있다.

4. 참고 자료

https://arxiv.org/pdf/1712.09665.pdf