

입력 변환 기반 Adversarial Examples 생성 방법 연구

김서연 · 김원준 · 류완철 · 안태영 · 황정순 · 최윤희
(부산대학교 1학년) (지도교사) (지도교수)

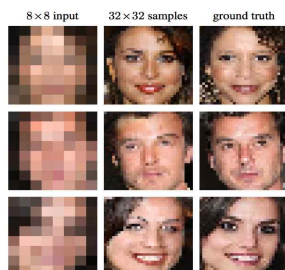
요약

최근 심층 신경망 모델이 사람의 눈에 보이지 않는 섭동을 추가해 모델의 오분류를 유도하는 adversarial attack에 취약함이 밝혀지면서 관련된 다양한 연구가 이루어지고 있다. 이러한 공격은 심층 신경망 기반 보안에 큰 문제를 일으킬 수 있음에도 불구하고 현재 adversarial attack 시뮬레이터는 연구를 위한 형태로만 제공되고 있고 서비스 관점에서는 전무하다. 본 과제에서는 머신 러닝 프레임워크 'Tensorflow'와 오픈소스 라이브러리 'cleverhans'를 활용하여 다양한 공격 기법을 모듈 단위로 개발 후, 시각화하여 사용자가 이미지를 비교할 수 있는 시뮬레이터를 개발하고자 하였다. 또한, 원본, 공격, 섭동 이미지의 출력을 통해 사용자가 공격의 영향을 쉽게 확인할 수 있도록 하였다.

주제어: 심층 신경망, 보안, 오픈소스 라이브러리, 적대적 공격, 머신 러닝

I. 서론

올해 세계 AI 시장 규모가 작년 73억 5000만 달러에서 112억 8000만 달러 규모로 성장했으며, 2025년에는 약 20배 성장한 898억 5000만 달러 규모로의 성장이 추정되고 있다[1]. UN 미래 보고서에 따르면 2050년까지 현존하는 20억 개의 일자리가 AI로 대체될 전망이다[2]. 이러한 AI에 사용되는 핵심 기술은 심층 신경망(Deep Neural Network)이며, 입력층과 출력층 사이에 다중의 은닉층을 포함하여 여러 비선형적인 관계들을 학습해 많은 분야에서 뛰어난 성능을 제공하는 기술을 통칭한다. 심층 신경망의 활용 예시 중에는 이미지 인식 관련 분야가 높은 비중을 차지하고 있는데, 대표적으로 [Fig. 1]과 같은 픽셀 복원 프로그램[3]을 들 수 있다.



[Fig. 1] Pixel recursive super resolution example

[Fig. 1]은 “Google Brain”의 연구진들이 개발한 심층 신경망 모델을 이용하여 얼굴 이미지를 저해상도로 변환시킨 후, 각

이미지가 무엇과 유사한 형태를 보이는지 예측하고 사진의 해상도를 극명하게 높이는 Pixel recursive super resolution의 예시이다. 즉, 이 프로그램은 8x8 pixel 사진이 심층 신경망 모델에 입력되면 원본이 어떻게 생겼는지 예측하는 동작을 수행한다.

이처럼 심층 신경망은 이미지 “복원”의 목적으로써 많은 분야에 활용되고 있다. 하지만, 이러한 심층 신경망 활용 사례들은 보안에 매우 취약한 문제점이 존재하며 이러한 취약점을 기반으로 한 다양한 공격이 등장하였다. 그 중에서도, 공격자가 의도적으로 조작시킨 입력 값들로 사람의 육안으로는 구분할 수 없는 미세한 차이를 만들어 심층 신경망 모델이 잘못된 결과를 산출하도록 이미지를 변환시키는 adversarial attack이 지속적으로 발전하고 있는 추세이다. 이러한 adversarial attack은 실험 및 검증을 위해서 각 공격 기법을 연구 관점에서의 시뮬레이터 형태로 제공하고 있으나, 다양한 공격 기법의 성능 비교 및 특정 이미지의 변화 과정 시각화 등을 위한 서비스 관점에서의 시뮬레이터는 전무한 실정이다. 따라서 본 과제에서는 Google에서 제공하는 머신 러닝 프레임워크 Tensorflow와 오픈 소스 라이브러리 cleverhans를 이용해 다양한 adversarial attack의 기법들을 모듈 단위로 개발한 후, 시각화하여 사용자가 공격 전후의 이미지를 비교할 수 있도록 하는 프로그램을 개발할 것이다. 이를 통해서 기초 프로그래밍 능력과 소프트웨어 알고리즘 설계 능력을 배양하고, 심층 신경망의 취약점 분석과 이를 이용한 공격 기법에 대해 학습 및 개발함으로써 심층 신경망 전반에 대한 견해를 넓히

고자 한다.

II. 이론적 배경

1. 머신 러닝 (Machine learning)

머신 러닝은 입력 데이터를 바탕으로 예측 값을 도출하는 모델 및 시스템을 통칭한다. 이러한 머신 러닝은 패턴 인식, 이상 감지, 예측 문제를 해결하기 위해 주로 사용된다. 패턴 인식의 경우 음성 단어 인식, 사람 얼굴 인식 등과 같은 분야에 사용되며, 이상 감지의 경우 네트워크 공격 탐지, 신용카드 거래 감지 분야에 사용된다. 예측의 경우에는 환율 예측, 주식 변동 예측 등 다양한 분야에 사용되고 있다. 머신 러닝에 사용되는 알고리즘은 크게 지도 학습, 준지도 학습 그리고 비지도(자율) 학습으로 나뉜다.

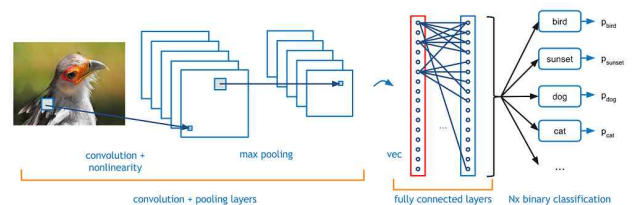
지도 학습 알고리즘은 한 세트의 사례들을 기반으로 예측을 수행한다. 지도 학습에는 기존에 이미 분류된 학습용 데이터로 구성된 입력 변수와 원하는 출력 변수가 수반되며, 알고리즘을 이용해 학습용 데이터를 분석함으로써 입력 변수를 출력 변수와 매핑시키는 함수를 찾을 수 있다. 이렇게 추론된 함수는 학습용 데이터로부터 일반화를 통해 알려지지 않은 새로운 사례들을 매핑하고, 눈에 보이지 않는 상황 속에서 결과를 예측한다. 회귀 분석, 인공 신경망 등이 지도 학습에 포함된다. 지도 학습은 데이터 분류(labeling) 작업에 많은 비용과 시간이 소모될 수 있다는 단점을 지닌다. 따라서 분류된 자료가 한정적일 때에는 지도 학습을 개선하기 위해 미분류(unlabeled) 사례를 이용하기도 하며, 이때 기계는 온전히 지도받지 않기 때문에 Semi-supervised라 표현한다. 준지도 학습은 학습 정확성을 개선하기 위해 미분류 사례와 함께 소량의 분류(labeled) 데이터를 이용한다. 위의 두 경우와는 다르게 비지도 학습을 수행할 때 기계는 미분류 데이터만을 제공받으며 클러스터링 구조(clustering structure), 저차원 다양체(low-dimensional manifold), 희소 트리 및 그래프(a sparse tree and graph) 등과 같은 데이터의 기저를 이루는 고유 패턴을 발견하도록 설정된다.

2. 인공 신경망 (Artificial neural network)

인공 신경망 (Artificial Neural Network, ANN)은 생물에서 신경 전달을 담당하는 기관인 뉴런(Neuron) 및 뉴런들을 연결하는 형태를 본떠서 만든 수학적 모델이다. 인공 신경망은 완전히 다른 두 성격을 가진 Input vector와 Output vector 간의 연관 관계를 학습하며, 또한 임의의 확률 분포를 학습할 수 있다. 인공 신경망은 입력층, 은닉층, 출력층 3개의 층으로 구성되어 있고, 각각의 층들 사이에는 노드들의 그룹이 형성되어 있다.

이를 발전시킨 것으로는 심층 인공 신경망 (Deep Neural Network, DNN)이 있는데, 심층 인공 신경망은 일반적인 인공 신경망과 달리 여러 개의 은닉층으로 구성되어 있다. 인공 신경망

의 경우에는 은닉층이 단 하나밖에 존재하지 않기 때문에 단일 단계의 특징만을 추출할 수 있지만, 심층 인공 신경망은 이를 보완하여 여러 개의 은닉층을 이용하여 여러 단계에서 더 높은 수준의 특징을 추출할 수 있도록 만들어졌다. 높은 수준의 특징은 낮은 수준의 특징보다 더 많은 정보를 가지고 있기에 이는 정보를 처리하는 데에 있어서 큰 도움을 준다[4]. 또한, 기존 인공 신경망이 특징 추출 단계와 분류(회귀) 모델을 따로 형성한 것에 비교해 심층 인공 신경망은 이를 하나의 모델로 처리할 수 있기 때문에 효율적인 작업 수행이 가능하다. 하지만 심층 인공 신경망의 경우에는 학습 데이터를 과하게 학습하여 학습 데이터에 대해서는 오차가 감소하는 경향을 보이지만 실제 데이터로 테스트를 시행할 경우 오차가 오히려 증가하는 과적합 (Overfitting) 현상이 자주 발생하여 이후 합성곱 신경망으로 대체되었다.



[Fig. 2] Structure of Convolutional Neural Network

합성곱 신경망(Convolutional Neural Network, CNN)은 심층 인공 신경망의 발전 형태로 최소한의 전처리 과정을 사용하도록 설계되었다. 합성곱(Convolution)이란 하나의 함수와 다른 함수를 반전하여 이동한 값을 곱한 다음, 특정 구간에 대해서 적분하여 새로운 함수를 만드는 것을 말한다. [Fig. 2]에서와 같이[5], 합성곱 신경망은 이전의 심층 신경망들과는 다르게 하나 또는 여러 개의 합성곱 layer와 일반적으로 개발된 다른 인공 신경망을 섞어서 사용하며 가중치 등을 추가로 활용한다. 이러한 구조로 인해 합성곱 신경망은 2차원 행렬 구조의 입력 데이터 (이미지, 파일 등)를 충분히 활용할 수 있고 다른 신경망 구조들에 비해서 탁월한 성능을 보여준다.

3. Adversarial Example & Attack

Adversarial attack은 악의적인 입력을 통해 모델을 속이기 위해 시도하는 신경망 분야에서 사용되는 공격 기술이다. 이 기법은 다양한 방향으로 적용될 수 있는데, 가장 일반적인 것은 신경망 모델의 오작동을 유발하는 것이다. 이 기법은 어떻게 공격자가 입력 데이터를 은밀하게 조작하여 신경망 알고리즘의 특정한 취약점을 이용하고, 또한 훼손할 수 있는지를 보여준다. Adversarial attack이 적용되어 생성된 입력 데이터값을 adversarial example 이라고 한다.

Gradient based optimization은 adversarial example의 개념을 소개한 최초의 공격 기법이다. 이 방식은 신경망을 자극하는

특성에서 경사 하강법을 기반으로 한 최적화된 접근 방식을 사용하여 심층 신경망에 대한 adversarial example을 제시한다. Gradient based optimization 공격은 다음 식을 이용하여 수행된다고 볼 수 있다.

$$loss(f(x+r), l) + c|r|$$

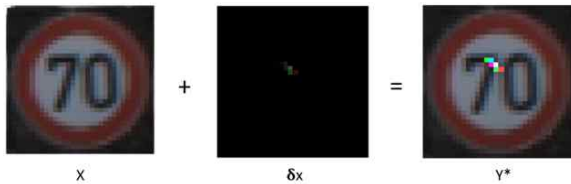
위 공식에서 x 는 입력 이미지, r 은 adversarial example을 생성하기 위한 픽셀의 변경(이에 따라 $x+r$ 은 새 이미지를 의미한다), l 은 원하는 결과의 클래스, 파라미터 c 는 이미지 사이의 거리와 모델 예측 사이의 거리를 균형 있게 조정하는 파라미터로 사용된다. 첫 번째 항은 adversarial example의 예측 결과와 원하는 등급 l 사이의 거리로, 두 번째 항은 adversarial example의 원래 이미지 사이의 거리를 측정한다. 이 공식은 반 사실적 설명을 생성하기 위해 손실 기능과 거의 동일하다. r 에는 픽셀 값이 0과 1 사이에 있도록 하는 추가적인 제약조건이 존재한다.

Fast Gradient Sign Method'는 non-iterative 기반의 공격 기법으로, adversarial example을 찾기 위해 신경망 모델의 gradient를 사용한다. gradient를 사용한다는 것은 신경망 모델이 분류를 실패하도록 이미지를 의도적으로 변경한다는 것을 의미한다. 주어진 입력 이미지 X 에 대해 섭동의 양은 다음과 같이 계산된다.

$$x' = x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$$

$\nabla_x J$ 는 원래 입력 픽셀 벡터 x 에 대한 모델 손실함수의 기울기(gradient), y 는 x 에 대한 실제 레이블 벡터, θ 는 모델 파라미터 벡터. gradient 벡터(입력 픽셀의 벡터인 경우)를 나타낸다.

Jacobian-based Saliency Map Attack은 이미지 분류 작업을 위한 심층 신경망과 같은 분류 모델을 속이는 공격방법이다. 주어진 이미지의 몇 개의 픽셀만을 최댓값 또는 최솟값으로 변경시킴으로써 JSMA는 모델이 결과적인 adversarial example을 잘못된 지정된 목표 클래스로 분류하게 할 수 있다. 아래 [Fig. 3]은 JSMA를 통해 최소한의 픽셀만을 조정한 것을 나타낸 것[6]이다.



[Fig. 3] JSMA attack example

III. 연구방법 및 절차

1. 코드 주제별 탐구

Adversarial attack simulator를 만들기 위한 코드 작성 과정은 이미지 저장, 이미지 출력, 모델 저장으로 크게 3가지로 나눌 수 있었다.

수 있었다.

<Table. 1> Python code of Image saving process.

```
#1. Pillow
from PIL import Image
import numpy
imagenam=Image.open(path).convert('L')
imagenam=Image.resize((28,28))
imagepixel=np.array(imagenam)

#2. OpenCV
import numpy
import cv2
imagenam=cv2.imread('path',
cv2.IMREAD_GRAYSCALE)
image2=cv2.resize(imagenam,dsiz=(28,28),interpolatio
n=cv2.INTER_AREA)
imagepixel=np.array(imagenam)

#3. Changing name of the image
from PIL import Image
myimage=Image.open(path)
myimage.save('newname.extension')
```

이미지 저장 과정에서는 Pillow와 OpenCV를 이용하여 외부의 이미지를 모델 내에 저장시킬 수 있었다. 또한, 이미지들 사이의 명확한 구분을 위해 이미지의 이름을 바꾸어 저장하는 기능도 추가하였다.

<Table. 2> Python code of Model saving process.

```
#1. Saving model with h5 extension
model = Sequential()
model.add(Dense(units = 64, input_dim = 28*28,
activation = 'relu'))
model.add(Dense(units = 10, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy',
optimizer = 'sgd', metrics = ['accuracy'])
model.fit(x_train, y_train, epochs = 5, batch_size = 32,
validation_data = (x_val, y_val))
loss_and_metrics = model.evaluate(x_test,y_test,
batch_size = 32)
print("")
print('loss_and_metrics : ' + str(loss_and_metrics))
from keras.models import load_model
model.save('mnist_mlp_model.h5')
```

```
#2. Saving model with checkpoint(ckpt) extension
with tf.Session(graph = g) as sess :
    # Saver.save(sess, ckpt_path)
    # Saver.restore(sess, ckpt_path)
    saver = tf.train.Saver()
    sess.run(tf.global_variables_initializer())
    ckpt_path = saver.save(sess, "saved/train")
    print("ckpt files were saved as : ", ckpt_path)
```

h5 확장자 형태의 모델을 저장할 경우 나중에 모델을 재구성하기 위한 모델의 구성 정보, 모델을 구성하는 각 뉴런들의 가중치, 손실함수 등의 학습 설정, 재학습을 할 수 있도록 마지막 학습상태로 저장되게 된다. 하지만 모델 그래프 자체가 저장되지는 않는다. h5 확장자 형태로 모델을 저장하는 과정은 크게 모델 학습과정 설정, 학습, 평가, 저장 4단계로 나누었다.

checkpoint 확장자로 저장을 하게 되는 경우 학습된 모델의 그래프가 그대로 저장되기 때문에 더욱 효율적으로 저장할 수 있다. 이를 저장하고 불러오으로써 학습된 모델의 재사용 및 추가적인 학습과 같은 작업이 가능해진다. 만약 checkpoint 형태로 모델을 저장하게 되면 코드 파일이 저장되어 있는 디렉터리에 checkpoint 파일과 함께 train.ckpt.data-filename-of-totalfile, train.ckpt.index, train.ckpt.meta가 저장된다.

<Table. 3> Python code of Image printing process.

```
#1. Pillow
from PIL import Image
def prt_image('original image', 'adversarial example'):
    image_org = Image.open('original image', jpg,95 or none)
    image_att = Image.open('adversarial example', jpg,95 or none)
    image_org = Image.resize(28,28)
    image_att = Image.resize(28,28)
    image_org.show()
    image_att.show()

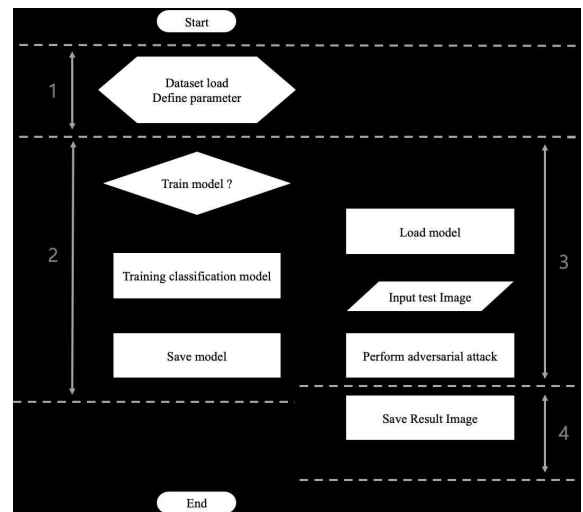
#2. OpenCV
import cv2
def prt_image('original image', 'adversarial example'):
    image_org = cv2.imread('original image',
    cv2.IMREAD_UNCHANGED)
    cv2.imshow(image_org)
    cv2.waitKey(0)
```

```
image_att = cv2.imread('adversarial example',
cv2.IMREAD_UNCHANGED)
cv2.imshow(image_att)
cv2.waitKey(0)
```

```
#3. matplotlib
import matplotlib.image as img
import matplotlib.pyplot as pp
file_org = 'original image'
fie_att = 'adversarial example'
def prt_img(file_org, file_att):
    image_org = img.imread(file_org)
    image_att = img.imread(file_att)
    pp.imshow(image_org)
    pp.imshow(image_att)
```

이미지 저장의 경우 앞에서 보았던 이미지 출력의 경우와 같이 Pillow와 OpenCV를 이용해서 저장할 수 있었다. 다른 방법으로는 matplotlib을 활용하는 방법이 있었는데, matplotlib은 Python에서 MATLAB과 유사한 그래프 표시를 가능케 하는 라이브러리다. 이 또한 이미지를 프로세싱 할 수 있는 기능들을 가지고 있어 이미지를 저장하는 데 유용하게 쓰일 수 있었다. 이러한 방법들 외에도 GUI를 이용하여 이미지를 출력하는 방법을 만들었으나 실제로 사용하지는 않았다.

2. Adversarial attack simulator 모델화



[Fig. 4] Schematic diagram of the research

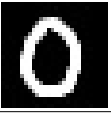


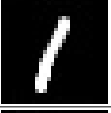








프로그램의 진행 과정 모식도는 [Fig. 4]와 같다. 먼저, MNIST 데이터셋과 같은 심층 신경망의 학습과 관련된 값들을 불러온 후 정의하고, 이후 train 모드와 test 모드를 사용자가 선택할 수 있도록 하여 train 모드에서는 MNIST 데이터셋을 이용

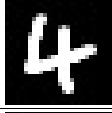












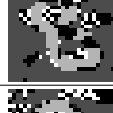




하여 심층 신경망 모델을 학습시키고 모델의 가중치를 저장하며, test 모드에서는 사용자로부터 공격을 수행할 이미지 파일을 입력 받은 후 저장된 심층 신경망 모델의 가중치를 불러와서 adversarial attack을 수행한 후 이미지와 그래프를 화면에 출력한다.

IV. 연구 결과

아래의 <Table. 4>는 사용자가 직접 숫자 10개를 손 글씨로 쓴 다음, png 파일 형태로 전환하여 본 연구의 코드를 통해 얻은 결과들을 나타내는 표이다. Original Image는 사용자가 입력한 원본 이미지, Adversarial Perturbation은 공격을 수행하기 위해 공격자가 원본 이미지에 추가한 섭동 이미지, Adversarial example은 실제로 공격이 수행된 이미지를 의미한다. Adversarial example 이미지를 보면 원본 이미지와 다소 차이가 나타나는 것을 볼 수 있다. 이러한 결과가 나오는 이유는 다음과 같다. 먼저, FGSM 공격은 초기 공격모델로써 최근 공격모델들과 비교하면 공격 성능이 떨어진다. 다음으로, 사용된 데이터셋이 MNIST 데이터셋인 점이 있다. MNIST 데이터셋은 28*28 크기의 사진들로, 하나의 픽셀이 바뀌어도 눈에 띄기 쉽기 때문이다. 이 프로그램이 공격 기법의 성능을 시험하는 시뮬레이터인 점을 생각한다면 다른 공격 기법이나 데이터셋을 이용한다면 충분히 다른 결과를 이끌어낼 수 있기 때문에 이미지에서 차이가 나타나는 것 자체만으로는 문제가 되지 않는다고 볼 수 있다.

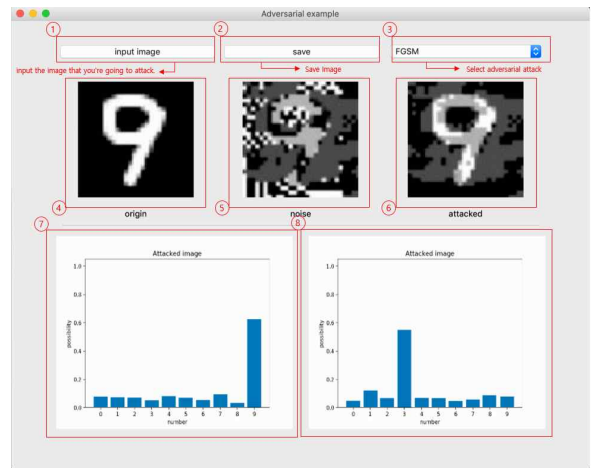
<Table. 4> Original Image, Adversarial Perturbation (Noise), Adversarial Example of 10 random handmade files which is similar to MNIST dataset. Fast Gradient Sign Method(FGSM) was used in this test case, so using another adversarial attack techniques can lead to a different result.

	Original Image	Adversarial Perturbation	Adversarial Example
0			
1			
2			
3			

4			
5			
6			
7			
8			
9			

아래의 [Fig. 5]는 우리가 최종적으로 구현한 Adversarial attack simulator를 실행시킨 모습이다. 그림에 표시된 요소들은 각각 다음과 같다.

- ① 공격에 사용할 이미지를 입력한다.
- ② 노이즈 이미지, 공격 이미지를 저장한다.
- ③ adversarial attack 기법을 선택한다.
- ④ 원본 이미지(Original image)가 출력되는 부분.
- ⑤ 노이즈 이미지(Adversarial perturbation)가 출력되는 부분
- ⑥ 공격 이미지(Adversarial example)가 출력되는 부분
- ⑦ 공격 전의 확률 값
- ⑧ 공격 후의 확률 값



[Fig. 5] Adversarial attack simulator, which is made by the research

이처럼 우리가 구현한 Adversarial attack simulator에서는 사용자가 28*28 크기의 흑백 손 글씨 이미지 (MNIST dataset의 형태를 가진 이미지)를 입력하고, 사용할 adversarial attack 기법을 입력하면 그에 맞게 공격이 실행되고, 원본, 노이즈, 공격 이미지와 원본과 공격 이미지에서의 확률 값이 출력되는 것을 볼 수 있다.

V. 고찰 및 결론

1. 고찰

현재까지 연구한 내용과 최종 목표인 본 연구의 주제를 함께 분석해 보았을 때, 현재 진행된 연구에서는 기초적인 데이터셋 ‘MNIST’를 이용하여 학습된 FGSM, JSMA의 공격 방식을 불러오는 공격 시뮬레이터를 구현하였다. 하지만 ‘MNIST’의 경우 28*28 크기의 흑백 이미지의 데이터셋으로 색 판별에 있어서 한계가 존재하기에, 사용자의 시뮬레이터 이용에 있어서 컬러 이미지를 사용하지 못한다는 제약이 있다. 그래서 추후에 28*28의 흑백 이미지가 아닌 32*32 크기의 컬러 이미지 데이터셋인 ‘CIFAR-10’을 사용하는 시뮬레이터를 구현한다면 사용에 있어서 더 넓은 스펙트럼을 가질 수 있을 것이라 보았다.

2. 결론

본 과제에서는 서비스 관점에서의 adversarial attack 시뮬레이터 개발을 위한 연구 및 개발을 수행하였다. 이를 위해 심층 신경망 및 adversarial attack에 대한 이론적인 연구를 수행하였으며, tensorflow 및 cleverhans 라이브러리 분석을 수행하였다. 우리가 개발한 시뮬레이터는 MNIST 데이터셋으로 심층 신경망 모델을 학습시키고 해당 모델을 초기 adversarial attack 모델로 공격하는 형태로 동작한다. 또한, 사용자가 원하는 이미지와 공격방법을 입력하면, adversarial attack 결과를 Original image, Adversarial perturbation, Adversarial example로 저장하고, 그래프로 표현된 정확도 값과 함께 시각화시키는 프로그램을 구현하였다. 이 과정을 통해, 기초 프로그래밍 능력과 소프트웨어 알고리즘 설계능력을 배양하고, 심층 신경망의 취약점 분석과 이를 이용하여 공격 기법에 대해 학습 및 개발함으로써 사용자가 좀 더 쉽게 adversarial attack을 사용할 수 있게 할 수 있었다.

3. 전망 및 활용성

연구의 목표에서 제시한 바와 같이 사용자가 adversarial example simulator를 통해서 Adversarial attack 기법들을 직접 시험하고, 효율성을 판단하는 작업을 통해서 Adversarial attack에 대한 이해도를 높이고 보안 측면에 있어서 더 다양한 대책들을 빠르게 수립할 수 있을 것으로 판단된다. 또한, 교육, 연구 및 산업적인 측면에서도 이 시뮬레이터의 활용을 통해서 심층 신경망 전반에 대한 심도 있는 후속 연구와 기술 발전이 있을 것이라

고 기대하는 바이다.

참고 문헌

- [1] 뉴시스. AI 시장 매년 급성장...美·中 경쟁에 韓 가세. (2018. September 10). Retrieved June 6, 2019. from http://www.newsis.com/view/?id=NISX20180817_0000393625
- [2] 박영숙. 제롬 글렌(2016). 세계미래보고서 2050. 교보문고
- [3] Ryan Dahl. Mohammad Norouzi. & Jonathon Shlens. (2017). Pixel recursive super resolution.
- [4] Wikipedia. Artificial neural network. (2019. November 8). Retrieved November 11, 2019. http://en.wikipedia.org/wiki/Artificial_neural_network
- [5] CS231n Convolutional Neural Networks for Visual Recognition. Convolutional Neural Networks (CNNs / ConvNets). (2019. June 4). Retrieved August 28, 2019. <http://cs231n.github.io/convolutional-networks/>
- [6] Arkar Min Aung. Yousef Fadila. Radian Gondokaryono. & Luis Gonzalez. (2017). Building Robust Deep Neural Networks for RoadSign Detection.