

# 개별연구 보고서

학생 개별 작성용

수행 학기	2021학년도 1학기			
교과목 정보	교과목명	개별연구 (AI모델 보안 강화 방법 연구)	분반	-분반
교과담당교수	소속	컴퓨터공학전공	교수명	손윤식 교수님
학생정보	이름	조양진	학번	2019112130
	학과	컴퓨터공학과	전화번호	01038583942
	학년	3학년	이메일	choyj1222@gmail.com
요약 보고서				
작품명 (프로젝트명)	AI모델 Evasion Attack 보안 강화 방법 연구			
1. 연구·개발 동기/목적/ 필요성 및 개발 목표	<input type="checkbox"/> OECD 신뢰가능한 AI의 5원칙 중 하나인 견고성, 보안성 및 안전성 (Robustness, Security and Safety)를 지키기 위해 필요한 기초 개념을 익힌다. <input type="checkbox"/> AI 모델을 응용하여 안정성이 중요한 분야에 적용하기 위한 AI 모델 보안의 필요성을 익힌다.			
2. 최종 결과물 소개	<input type="checkbox"/> 오픈소스로 공개되어 있는 머신러닝 모델 3가지를 각각 다른 Evasion Attack을 IBM의 adversarial-robustness-tool로 실험 후 성능 평가 및 분석 <input type="checkbox"/> 해당 머신러닝 모델을 공격으로부터 강인하게 만들 수 있는 방법을 제시 및 해당 방법을 적용한 모델의 성능 평가 및 분석			
3. 프로젝트 추진 내용	<input type="checkbox"/> MNIST, CIFAR-10, 그리고 IRIS 데이터 셋을 이용하여 머신러닝 모델 3가지를 생성 후, ART Toolbox로 실험(Evasion 공격)과 성능 평가 진행 <input type="checkbox"/> 선정 모델을 공격으로부터 강인하게 만들 수 있는 방법 제시 <input type="checkbox"/> 해당 방법을 적용한 모델을 다시 ART Toolbox로 실험 및 성능 평가 진행			
4. 기대효과	<input type="checkbox"/> 앞으로 계속 진화해 나갈 AI 모델과 그 AI 모델의 보안을 지키기 위한 방법을 연구하여 제로데이 취약점을 제외한 익스플로잇에 대한 대비 할 수 있는 방법을 터득한다.			

## 1. 서론

요즘 IT 업계의 뜨거운 감자, "AI"는 컴퓨터 공학을 전공하는 학부생 입장에서 정말 빼놓을 수 없는 중요한 주제이다. AI는 벌써 우리 생활에 밀접하게 적용되어 많은 삶의 질 향상에 도움을 주고 있으며, 몇 달 전에는 "AI 챗봇 이루다"라는 페이스북 메신저를 기반으로 사용자들의 질의에 머신 러닝으로 학습된 대화를 진행해 나가는 챗 봇이 무단으로 사람들의 메신저 기록을 수집하여 기계 학습에 사용했다는 것이 밝혀져 전국적으로 큰 물의를 일으키기도 했다. AI가 우리의 삶의 질을 높여주는 대신 우리는 그 AI를 잘 활용할 수 있도록, AI를 설계하는데 있어서 정말 많은 시간과 노력을 통해 reliable, stable한 결과값을 얻어낼 수 있도록 해야 한다.

최근 들어 의료영상처리에서 컴퓨터 보조진단 시스템을 사용하여 환자의 병을 치료하는 일이 많아졌다. 공학계와 임상계 간의 꾸준한 협력을 통하여 근 30년간 많은 질병들의 컴퓨터 보조진단 시스템을 개발했으나, 수년 전까지만 해도 현장에서 직접 사용할 수 있는 시스템들은 몇 존재하지 않았으나, 딥러닝을 통한 컴퓨터 보조진단 시스템을 사용함에 따라 의사의 판단과 거의 동일한 결과를 유도할 수 있다는 점에 있어서 미국 식품의약청 (US Food and Drug Administration, FDA)에서는 이미 딥러닝을 응용하여 개발한 컴퓨터 보조진단 의료기기를 승인, 대한민국에서도 이미 비슷한 선례가 존재한다.<sup>1)</sup> 따라서 의학계에서도 사용되는 딥러닝 (이하 AI라 칭함)에게 있어서 AI 모델의 보안은 정말 중대한 사항으로 발전할 수 밖에 없다. 해당 기기를 사용하여 질병을 검사하였는데, AI의 보안에 중대한 문제가 생겨 이를 사용한 결과값에 의존한 환자에게 비극이 생긴다면 이에 대한 책임을 져야 할 사람들이 많이 생길 것이다.

공학은 여러 가지 분야와 긴밀하게 상호작용하기 때문에, 시간이 지남에 따라 앞으로도 계속 우리의 일상에 AI가 존재하게 될 것이다. 이러한 의미에서 우리 사회가 딥러닝을 사용하여 세상을 이롭게 하려면 AI의 연구 개발에 집중할 뿐만 아니라, AI 모델의 보안에도 많은 관심과 노력이 필요할 것이다.

## 2. 관련/배경 연구

### 2.1 Fast Gradient Sign Method (FGSM)

Generative Adversarial Network의 저자로 유명한 Ian J. Goodfellow의 adversarial attack 기법 중 하나인 Fast Gradient Sign Method (FGSM)은 Deep Neural Network (DNN)의 linear 한 성질 때문에 교란을 통해 이를 무력화 할 수 있다고 가정하였고, 이를 증명하기 위해 FGSM을 제안하게 되었다. 공격에 필요한 모든 자원을 최소화 하기 위해 모든 차원에서의 공격 강도를 동일하게 설정하고 교란 신호 정도를 조정하기 위해 상수 매개변수, 입실론을 사용한다. FGSM은 non targeted attack이며, 수식은 다음과 같다.<sup>2)</sup>

### 2.2 DeepFool

DeepFool 알고리즘은 classifier (이미 훈련된 AI 모델)이 분류에 실패하는데 가장 최소한의 교란

$$\Delta(x, x') = \epsilon \cdot \text{sign}(\nabla_x J(g(x), y))$$

그림 1 Fast Gradient Sign Method 수식

신호를 기하학적으로 탐색하는 알고리즘으로 Decision boundary가 linear plane인 linear classifier의 출력 결과를 묘사하는 공간을 다면체로 표현하는데, DeepFool은 분류자가 결정을 바꿀 수 있는 교란 신호의 최소값을 이 다면체 내에서 탐색한다. Greedy Algorithm이므로 성능의 측면에서 불리하지만 Jacobian Saliency Map Attack 보다 훨씬 빠르고, FGSM보다 정확하게 공격할 수 있기에 유용하다. 또한 공격에 필요한 최소 교란 값을 알 수 있다는 특징이 매력이다.<sup>3)</sup>

$$\Delta(x, x') := \arg \text{ subject to: } g(x+z) \neq g(x)$$

그림 2 DeepFool 수식

### 2.3 NewtonFool

NewtonFool 알고리즘은 입력 값 (주로 이미지)에서 클래스 l에 매우 높은 확률로 속하는 화소 x가 있고, 반대로 같은 클래스 l에 매우 높은 확률로 속하지 않는 화소 x'가 있다 가정하여 gradient-descent (경사 하강) 알고리즘과 비선형 방정식을 풀기 위한 뉴턴 방법을 활용한 적대적 공격 알고리즘이다. DeepFool 알고리즘과 마찬가지로 classifier가 분류에 실패하도록 유도하기 위해 최소한의 교란 신호를 탐색하는 방법이다. JSMA, DeepFool보다 훨씬 빠르고, FGSM보다 느리지만 효율이 좋은 것으로 알려져, 각광받고 있는 알고리즘이다.<sup>4)</sup>

$$\delta_i = \min \left\{ \eta \|x_0\| \|\nabla F_s^l(x_i)\|, F_s^l(x_i) - \frac{1}{|C|} \right\}, \eta \in (0, 1)$$

$$\Delta(x, x') = - \frac{\delta_i \nabla F_s^l(x_i)}{\|\nabla F_s^l(x_i)\|^2}$$

그림 3 NewtonFool 수식

### 2.4 Jacobian Saliency Map Attack (JSMA)

Jacobian Saliency Map Attack (JSMA) 알고리즘은 입력 값 (주로 이미지)의 일부 화소들을 수정하여 적대적 공격이 성립하게 되는 화소의 최소 개수를 찾는 알고리즘이다. Saliency map (돌출지도)는 입력 이미지 x에서 DNN 모델의 Jacobian 기울기 행렬로부터 계산하여 각 화소가 이미지를 특정 클래스로 분류하는 결과에 미치는 영향을 모델링한다. 따라서 Saliency score (돌출점수)가 높으면 해당 레이블이 타겟이 될 확률이 높게 되고, 돌출지도에서 표적이 되는 클래스의 확률을 증가시키기 위해, 해당 화소를 선택하고 반복적으로 교란 신호를 추가한다. 마찬가지로 Greedy 알고리즘이므로 매우 느리지만

표적 공격시 최소한으로 공격할 화소의 개수를 알 수 있다는 점이 JSMA의 특징이다.<sup>5)</sup>

## 2.5 Adversarial Training

Adversarial 예제들은 머신 러닝 모델들을 속여 모델이 제 역할을 하지 못하도록 설계된 예제이다. Adversarial Training의 경우, 일부러 adversarial example을 injection하여 robustness를 증가시킬 수 있도록 하는 학습 방법으로, 해당 Adversarial training을 통해서 적대적 공격에 대한 방어에 관한 NIPS 2017 대회 첫 라운드에서 승리했던 전적이 있으나, 추후 정교한 블랙박스 공격이 어택 가능성을 높이고 모델의 정확도를 떨어뜨릴 수 있다는 것이 밝혀지기도 했다.<sup>6)</sup>

## 3. 본론

아래 진행된 모든 결과값은 Lenovo사의 ThinkPad E14 Gen2에서 진행됐다.

CPU: Ryzen 5 4500U@Boosted 3.9GHz  
RAM: Samsung DDR4 16G@3200MHz  
GPU: AMD Radeon Graphics@1500MHz  
OS: Windows 10 Education 20H2 19042.1052  
IDE: Pycharm Professional 2021.1.2  
Python: Python 3.7 venv w/tensorflow 1.15.5 & Keras 2.3.1

### 3.1 MNIST – FGSM 공격

- 데이터 세트 설명

손글씨로 작성된 0에서 9 사이의 숫자 이미지로, 각각의 이미지는 다양한 모양이지만 사람이라면 쉽게 인지할 수 있는 이미지로 구성되어 있다.

- 공격 방법

FGSM

- 소스 코드 (01\_MNIST\_FGSM.py)

```
# 2019112130 Yangjin Cho
# 01_MNIST_FGSM
# Independant Capstone AI Model Security
import keras
import time
import datetime
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
import numpy as np
from art.attacks.evasion import FastGradientMethod
from art.estimators.classification import KerasClassifier
from art.utils import load_mnist
start = time.time()
# Step 1: Load the MNIST dataset
(x_train, y_train), (x_test, y_test), min_pixel_value, max_pixel_value = load_mnist()
# Step 2: Create the model
```

```

model = Sequential()
model.add(Conv2D(filters =4 , kernel_size =(5 , 5 ), strides =1 , activation ="relu", input_shape =(28
, 28 , 1 )))
model.add(MaxPooling2D(pool_size =(2 , 2 )))
model.add(Conv2D(filters =10 , kernel_size =(5 , 5 ), strides =1 , activation ="relu", input_shape =(23
, 23 , 4 )))
model.add(MaxPooling2D(pool_size =(2 , 2 )))
model.add(Flatten())
model.add(Dense(100 , activation ="relu"))
model.add(Dense(10 , activation ="softmax"))
model.compile(
    loss =keras.losses.categorical_crossentropy, optimizer =keras.optimizers.Adam(lr =0.01 ), metrics
=["accuracy"]
)
# Step 3: Create the ART classifier
classifier = KerasClassifier(model =model, clip_values =(min_pixel_value, max_pixel_value), use_logits
=False )
# Step 4: Train the ART classifier
classifier.fit(x_train, y_train, batch_size =64 , nb_epochs =20 )
# Step 5: Evaluate the ART classifier on benign test examples
predictions = classifier.predict(x_test)
accuracy = np.sum(np.argmax(predictions, axis =1 ) == np.argmax(y_test, axis =1 )) / len (y_test)
print ("정상적으로 학습시킨 MNIST 모델의 정확도: {}".format(accuracy * 100 ))
# Step 6: Generate adversarial test examples
attack = FastGradientMethod(estimator =classifier, eps =0.2 )
x_test_adv = attack.generate(x =x_test)
# Step 7: Evaluate the ART classifier on adversarial test examples
predictions = classifier.predict(x_test_adv)
accuracy = np.sum(np.argmax(predictions, axis =1 ) == np.argmax(y_test, axis =1 )) / len (y_test)
print ("MNIST에 FGSM 공격을 가한 후 정확도: {}".format(accuracy * 100 ))
sec = time.time() - start
times = str (datetime.timedelta(seconds =sec)).split(".")
times = times[0 ]
print (times)

```

### 3.2 CIFAR-10 – DeepFool 공격

- 데이터 세트 설명

CIFAR-10 데이터 세트는 32\*32 픽셀의 6만개의 컬러이미지가 포함되어 있으며, 각 클래스는 총 10개의 클래스로 라벨링이 되어 있다.

- 공격 방법

DeepFool

- 소스 코드 (03\_CIFAR10\_DEEPFOOL.py)

```

# 2019112130 Yangjin Cho
# 03_CIFAR10_DEEPFOOL
# Indendant Capstone AI Model Security
import keras
import datetime
import time
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Activation, Dropout

```

```

import numpy as np
from art.attacks.evasion import DeepFool
from art.estimators.classification import KerasClassifier
from art.utils import load_cifar10
start = time.time()
# Step 1: Load the CIFAR-10 dataset
(x_train , y_train ), (x_test , y_test ), min_pixel_value , max_pixel_value = load_cifar10()
# Step 2: Create the model
model = Sequential()
model.add(Conv2D(32 , (3 , 3 ) , padding ="same" , input_shape =x_train .shape[1 :]))
model.add(Activation("relu"))
model.add(Conv2D(32 , (3 , 3 )))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size =(2 , 2 )))
model.add(Dropout(0.25 ))
model.add(Conv2D(64 , (3 , 3 ) , padding ="same"))
model.add(Activation("relu"))
model.add(Conv2D(64 , (3 , 3 )))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size =(2 , 2 )))
model.add(Dropout(0.25 ))
model.add(Flatten())
model.add(Dense(512 ))
model.add(Activation("relu"))
model.add(Dropout(0.5 ))
model.add(Dense(10 ))
model.add(Activation("softmax"))
model.compile(loss ="categorical_crossentropy" , optimizer ="adam" , metrics =[ "accuracy" ])
# Step 3: Create the ART classifier
classifier = KerasClassifier(model =model , clip_values =(min_pixel_value , max_pixel_value ) , use_logits
=False )
# Step 4: Train the ART classifier
classifier.fit(x_train , y_train , batch_size =64 , nb_epochs =20 )
# Step 5: Evaluate the ART classifier on benign test examples
predictions = classifier.predict(x_test )
accuracy = np.sum (np.argmax (predictions , axis =1 ) == np.argmax (y_test , axis =1 )) / len
(y_test )
print ("정상적으로 학습시킨 CIFAR-10 모델의 정확도: {}".format (accuracy * 100 ))
# Step 6: Generate adversarial test examples
attack = DeepFool(classifier =classifier , max_iter =5 , epsilon =0.2 )
x_test_adv = attack.generate(x =x_test )
# Step 7: Evaluate the ART classifier on adversarial test examples
predictions = classifier.predict(x_test_adv )
accuracy = np.sum (np.argmax (predictions , axis =1 ) == np.argmax (y_test , axis =1 )) / len
(y_test )
print ("CIFAR-10에 DeepFool 공격을 가한 후 정확도: {}".format (accuracy * 100 ))
sec = time.time() - start
times = str (datetime.timedelta (seconds =sec )).split (".")
times = times [0 ]
print (times )

```

### 3.3 IRIS – JSMA 공격

- 데이터 세트 설명

IRIS 데이터 세트는 붓꽃 데이터로, 레코드 수 150개, 필드 개수 5개이며 꽃잎의 각 부분의 너비와 길이들을 측정한 데이터이다.

- 공격 방법

JSMA

- 소스 코드 (05\_IRIS\_JSMA.py)

```
# 2019112130 Yangjin Cho
# 05_IRIS_JSMA_DEFEND
# Independant Capstone AI Model Security
import keras
import time
import datetime
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
import numpy as np
from art.attacks.evasion import SaliencyMapMethod
from art.estimators.classification import KerasClassifier
from art.utils import load_iris
from art.defences.trainer import AdversarialTrainer
start = time.time()
# Step 1: Load the IRIS dataset
(x_train , y_train ), (x_test , y_test ), min_pixel_value , max_pixel_value = load_iris()
# Step 2: Create the model
# https://www.kaggle.com/rushabhwadkar/deep-learning-with-keras-on-iris-dataset
model = Sequential()
model.add(Dense(10 ,input_dim =4 , activation = 'relu',kernel_initializer = 'he_normal'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dropout(0.3 ))
model.add(Dense(7 , activation = 'relu', kernel_initializer = 'he_normal',kernel_regularizer =keras
.regularizers.l1_l2(l1 =0.001 , l2 =0.001 )))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dropout(0.3 ))
model.add(Dense(5 , activation = 'relu', kernel_initializer = 'he_normal',kernel_regularizer =keras
.regularizers.l1_l2(l1 =0.001 , l2 =0.001 )))
model.add(Dense(3 ,activation = 'softmax'))
model.compile(
    loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy']
)
# Step 3: Create the ART classifier
classifier = KerasClassifier(model =model , clip_values =(min_pixel_value , max_pixel_value ), use_logits
=False )
# Step 4: Train the ART classifier
classifier.fit(x_train , y_train , batch_size =7 , nb_epochs =700 )
# Step 5: Evaluate the ART classifier on benign test examples
predictions = classifier.predict(x_test )
accuracy = np .sum (np .argmax (predictions , axis =1 ) == np .argmax (y_test , axis =1 )) / len
(y_test )
print ("정상적으로 학습시킨 IRIS 모델의 정확도: {}".format (accuracy * 100 ))
# Step 6: Generate adversarial test examples
attack = SaliencyMapMethod(classifier =classifier )
x_test_adv = attack .generate(x =x_test )
# Step 7: Evaluate the ART classifier on adversarial test examples
predictions = classifier.predict(x_test_adv )
```

```

accuracy = np.sum(np.argmax(predictions, axis=1) == np.argmax(y_test, axis=1)) / len(y_test)
print("IRIS에 JSMA 공격을 가한 후 정확도: {}".format(accuracy * 100))
sec = time.time() - start
times = str(datetime.timedelta(seconds=sec)).split(".")
times = times[0]
print(times)

```

## 4. 실험결과

### 4.1 MNIST – FGSM 공격 결과 및 개선 방안과 평가

#### - 공격 결과

```

Run: 01_MNIST_FGSM (1) ×
840/937 [=====>...] - ETA: 0s - loss: 0.0617 - accuracy: 0.9853
847/937 [=====>...] - ETA: 0s - loss: 0.0616 - accuracy: 0.9853
854/937 [=====>...] - ETA: 0s - loss: 0.0615 - accuracy: 0.9853
861/937 [=====>...] - ETA: 0s - loss: 0.0616 - accuracy: 0.9853
868/937 [=====>...] - ETA: 0s - loss: 0.0613 - accuracy: 0.9854
875/937 [=====>...] - ETA: 0s - loss: 0.0613 - accuracy: 0.9854
882/937 [=====>...] - ETA: 0s - loss: 0.0614 - accuracy: 0.9853
889/937 [=====>...] - ETA: 0s - loss: 0.0615 - accuracy: 0.9853
896/937 [=====>...] - ETA: 0s - loss: 0.0613 - accuracy: 0.9854
903/937 [=====>...] - ETA: 0s - loss: 0.0612 - accuracy: 0.9853
910/937 [=====>...] - ETA: 0s - loss: 0.0610 - accuracy: 0.9854
917/937 [=====>...] - ETA: 0s - loss: 0.0607 - accuracy: 0.9854
924/937 [=====>...] - ETA: 0s - loss: 0.0606 - accuracy: 0.9855
931/937 [=====>...] - ETA: 0s - loss: 0.0607 - accuracy: 0.9855
937/937 [=====] - 7s 8ms/step - loss: 0.0606 - accuracy: 0.9
정상적으로 학습시킨 MNIST 모델의 정확도: 98.14%
MNIST에 FGSM 공격을 가한 후 정확도: 68.27%
0:02:31
|
Process finished with exit code 0

```

그림 4 MNIST FGSM Attack

MNIST 데이터 셋을 학습시킨 후 해당 AI 모델을 Fast Gradient Sign Method로 adversarial attack하여, 기존의 AI 모델의 정확도와 FGSM 공격 당한 모델의 정확도를 비교하였다. 기존의 AI 모델의 정확도는 98.14%, 공격 후 정확도는 68.27%로 낮아졌다. 이는 약 29.87%p가 낮아진 것이다.

#### - 개선 방안



Adversarial Trainer를 사용하여 해당 모델을 학습시킨 후, 기존의 일반적으로 학습시킨 모델의 정확도, FGSM을 공격을 가한 테스트 값을 일반 모델로 predict한 정확도 그리고 Adversarial Trainer를 사용하여 학습시킨 모델로 FGSM 공격을 한 테스트 값의 predict 값을 비교하여 Adversarial Trainer를 사용함이 실제로 AI 모델 공격 보안에 효과가 있는지를 확인한다.

- 개선 방안 적용 후 공격 (02\_MNIST\_FGSM\_DEFEND.py)

```
# 2019112130 Yangjin Cho
# 02_MNIST_FGSM_DEFEND
# Independant Capstone AI Model Security
import keras
import time
import datetime
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
import numpy as np
from art.attacks.evasion import FastGradientMethod
from art.estimators.classification import KerasClassifier
from art.utils import load_mnist
from art.defences.trainer import AdversarialTrainer
start = time.time()
# Step 1: Load the MNIST dataset
(x_train , y_train ), (x_test , y_test ), min_pixel_value , max_pixel_value = load_mnist()
# Step 2: Create the model
model = Sequential()
model.add(Conv2D(filters =4 , kernel_size =(5 , 5 ), strides =1 , activation ="relu", input_shape =(28 , 28 , 1 )))
model.add(MaxPooling2D(pool_size =(2 , 2 )))
model.add(Conv2D(filters =10 , kernel_size =(5 , 5 ), strides =1 , activation ="relu", input_shape =(23 , 23 , 4 )))
model.add(MaxPooling2D(pool_size =(2 , 2 )))
model.add(Flatten())
model.add(Dense(100 , activation ="relu"))
model.add(Dense(10 , activation ="softmax"))
model.compile(
    loss =keras .losses.categorical_crossentropy, optimizer =keras .optimizers.Adam(lr =0.01 ), metrics
    =["accuracy"]
)
# Step 3: Create the ART classifier
classifier = KerasClassifier(model =model , clip_values =(min_pixel_value , max_pixel_value ), use_logits
=False )
# Step 4: Train the ART classifier
classifier .fit(x_train , y_train , batch_size =64 , nb_epochs =20 )
# Step 5: Evaluate the ART classifier on benign test examples
predictions = classifier .predict(x_test )
accuracy = np .sum (np .argmax (predictions , axis =1 ) == np .argmax (y_test , axis =1 )) / len
(y_test )
print ("정상적으로 학습시킨 MNIST 모델의 정확도: {}".format (accuracy * 100 ))
# Step 6: Generate adversarial test examples
attack = FastGradientMethod(estimator =classifier , eps =0.2 )
x_test_adv = attack .generate(x =x_test )
# Step 7: Evaluate the ART classifier on adversarial test examples
predictions = classifier .predict(x_test_adv )
```

```

accuracy = np.sum(np.argmax(predictions, axis =1 ) == np.argmax(y_test, axis =1 )) / len(y_test)
print ("MNIST에 FGSM 공격을 가한 후 정확도: {}".format(accuracy * 100 ))
# Step 8: Train with AdversarialTrainer for accuracy comparision
# Paper link: https://arxiv.org/abs/1705.07204
AdversarialTrainer(classifier =classifier , attacks =attack , ratio =0.5 ).fit(x =x_train , y =y_train , batch_size =64 , nb_epochs =20 )
# Step 9: Evaluate the ART classifier on adversarial test examples
predictions = AdversarialTrainer(classifier =classifier , attacks =attack , ratio =0.5 ).predict(x =x_test_adv )
accuracy = np.sum(np.argmax(predictions, axis =1 ) == np.argmax(y_test, axis =1 )) / len(y_test)
print ("AdversarialTrainer - FGSM을 방어한 후의 모델 정확도: {}".format(accuracy * 100 ))
sec = time.time () - start
times = str(datetime.timedelta(seconds =sec)).split(".")
times = times [0 ]
print (times )

```

Adversarial Trainer를 사용하여 동일한 조건으로 모델을 학습시킨 후, 공격 받은 테스트 셋을 predict하여 기존의 일반적인 학습 이후 공격 받은 테스트 셋의 predict 값이랑 비교한다.

## - 개선 전후 비교

```

Python Console - IndependantCapstone
Python Console × 02_MNIST_FGSM_DEFEND ×

807/937 [=====>.....] - ETA: 1s - loss: 0.0575 - accuracy: 0.9857
815/937 [=====>....] - ETA: 1s - loss: 0.0573 - accuracy: 0.9858
823/937 [=====>....] - ETA: 0s - loss: 0.0570 - accuracy: 0.9859
831/937 [=====>....] - ETA: 0s - loss: 0.0575 - accuracy: 0.9859
839/937 [=====>....] - ETA: 0s - loss: 0.0573 - accuracy: 0.9859
845/937 [=====>...] - ETA: 0s - loss: 0.0579 - accuracy: 0.9858
853/937 [=====>...] - ETA: 0s - loss: 0.0576 - accuracy: 0.9859
861/937 [=====>...] - ETA: 0s - loss: 0.0574 - accuracy: 0.9859
869/937 [=====>...] - ETA: 0s - loss: 0.0577 - accuracy: 0.9858
877/937 [=====>..] - ETA: 0s - loss: 0.0575 - accuracy: 0.9859
885/937 [=====>..] - ETA: 0s - loss: 0.0579 - accuracy: 0.9859
892/937 [=====>..] - ETA: 0s - loss: 0.0581 - accuracy: 0.9857
900/937 [=====>..] - ETA: 0s - loss: 0.0584 - accuracy: 0.9857
908/937 [=====>.] - ETA: 0s - loss: 0.0590 - accuracy: 0.9857
916/937 [=====>.] - ETA: 0s - loss: 0.0594 - accuracy: 0.9857
924/937 [=====>.] - ETA: 0s - loss: 0.0596 - accuracy: 0.9857
931/937 [=====>.] - ETA: 0s - loss: 0.0596 - accuracy: 0.9857
937/937 [=====] - 8s 8ms/step - loss: 0.0595 - accuracy: 0.9
정상적으로 학습시킨 MNIST 모델의 정확도: 97.7%
MNIST에 FGSM 공격을 가한 후 정확도: 61.7%
Precompute adv samples: 100%|██████████| 1/1 [00:00<?, ?it/s]
Adversarial training epochs: 100%|██████████| 20/20 [07:01<00:00, 21.07s/it]
AdversarialTrainer - FGSM을 방어한 후의 모델 정확도: 94.56%
0:09:31

>>>

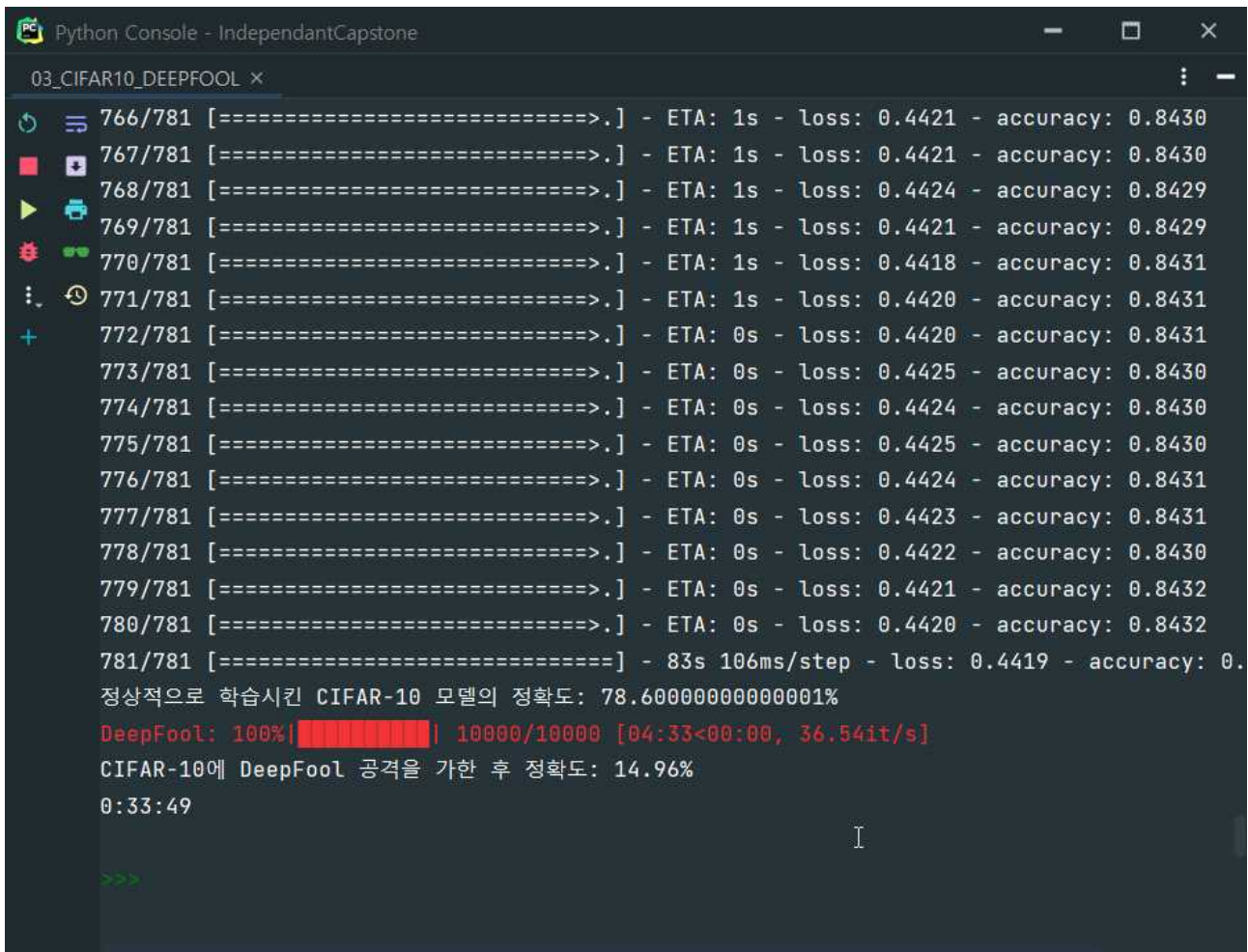
```

그림 5 MNIST Adversarial Trainer FGSM Defend

정상적으로 학습시킨 MNIST 모델의 정확도는 97.7%, MNIST에 FGSM 공격을 가한 후 정확도는 61.7%이다. 이후 Adversarial Trainer를 통해 새로 모델을 학습시키고 이후 FGSM 공격을 받았던 테스트 셋을 predict 한 값은 94.56%가 나왔다. 이를 통해 Adversarial Trainer를 사용하여 61.7%였던 정확도를 94.56%까지 끌어올렸음을 알 수 있다.

## 4.2 CIFAR-10 – DeepFool 공격 결과 및 개선 방안과 평가

### - 공격 결과



```
Python Console - IndependantCapstone
03_CIFAR10_DEEPFOOL x
766/781 [=====>.] - ETA: 1s - loss: 0.4421 - accuracy: 0.8430
767/781 [=====>.] - ETA: 1s - loss: 0.4421 - accuracy: 0.8430
768/781 [=====>.] - ETA: 1s - loss: 0.4424 - accuracy: 0.8429
769/781 [=====>.] - ETA: 1s - loss: 0.4421 - accuracy: 0.8429
770/781 [=====>.] - ETA: 1s - loss: 0.4418 - accuracy: 0.8431
771/781 [=====>.] - ETA: 1s - loss: 0.4420 - accuracy: 0.8431
772/781 [=====>.] - ETA: 0s - loss: 0.4420 - accuracy: 0.8431
773/781 [=====>.] - ETA: 0s - loss: 0.4425 - accuracy: 0.8430
774/781 [=====>.] - ETA: 0s - loss: 0.4424 - accuracy: 0.8430
775/781 [=====>.] - ETA: 0s - loss: 0.4425 - accuracy: 0.8430
776/781 [=====>.] - ETA: 0s - loss: 0.4424 - accuracy: 0.8431
777/781 [=====>.] - ETA: 0s - loss: 0.4423 - accuracy: 0.8431
778/781 [=====>.] - ETA: 0s - loss: 0.4422 - accuracy: 0.8430
779/781 [=====>.] - ETA: 0s - loss: 0.4421 - accuracy: 0.8432
780/781 [=====>.] - ETA: 0s - loss: 0.4420 - accuracy: 0.8432
781/781 [=====] - 83s 106ms/step - loss: 0.4419 - accuracy: 0.
정상적으로 학습시킨 CIFAR-10 모델의 정확도: 78.60000000000001%
DeepFool: 100%|██████████| 10000/10000 [04:33<00:00, 36.54it/s]
CIFAR-10에 DeepFool 공격을 가한 후 정확도: 14.96%
0:33:49
>>>
```

그림 6 CIFAR-10 DeepFool Attack

CIFAR-10 데이터 셋을 학습시킨 후 해당 AI 모델을 Deep Fool로 adversarial attack하여, 기존의 AI 모델의 정확도와 FGSM 공격 당한 모델의 정확도를 비교하였다. 기존의 AI 모델의 정확도는 78.60%, 공격 후 정확도는 14.96%로 낮아졌다. 이는 약 63.64%p가 낮아진 것이다.

### - 개선 방안

Adversarial Trainer를 사용하여 해당 모델을 학습시킨 후, 기존의 일반적으로 학습 시킨 모델의 정확도, FGSM을 공격을 가한 테스트 값을 일반 모델로 predict한 정확도 그리고 Adversarial Trainer를 사용하여 학습시킨 모델로 DeepFool 공격을 한 테스트 값의 predict 값을 비교하여 Adversarial Trainer를 사용함이 실제로 AI 모델 공격 보안에 효과가 있는지를 확인한다.

- 개선 방안 적용 후 공격 (04\_CIFAR10\_DEEPFOOL\_DEFEND.py)

```
# 2019112130 Yangjin Cho
# 04_CIFAR10_DEEPFOOL_DEFEND
# Independant Capstone AI Model Security
import keras
import time
import datetime
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Activation, Dropout
import numpy as np
from art.attacks.evasion import DeepFool
from art.estimators.classification import KerasClassifier
from art.utils import load_cifar10
from art.defences.trainer import AdversarialTrainer
start = time.time()
# Step 1: Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test), min_pixel_value, max_pixel_value = load_cifar10()
# Step 2: Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), padding="same", input_shape=x_train.shape[1:]))
model.add(Activation("relu"))
model.add(Conv2D(32, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(Conv2D(64, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation("softmax"))
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
# Step 3: Create the ART classifier
classifier = KerasClassifier(model=model, clip_values=(min_pixel_value, max_pixel_value), use_logits=False)
# Step 4: Train the ART classifier
classifier.fit(x_train, y_train, batch_size=64, nb_epochs=20)
# Step 5: Evaluate the ART classifier on benign test examples
predictions = classifier.predict(x_test)
accuracy = np.sum(np.argmax(predictions, axis=1) == np.argmax(y_test, axis=1)) / len(y_test)
print("정상적으로 학습시킨 CIFAR-10 모델의 정확도: {}".format(accuracy * 100))
# Step 6: Generate adversarial test examples
attack = DeepFool(classifier=classifier, max_iter=5, epsilon=0.2)
x_test_adv = attack.generate(x=x_test)
# Step 7: Evaluate the ART classifier on adversarial test examples
predictions = classifier.predict(x_test_adv)
accuracy = np.sum(np.argmax(predictions, axis=1) == np.argmax(y_test, axis=1)) / len
```

```

(y_test )
print ("CIFAR-10에 DeepFool 공격을 가한 후 정확도: {}".format (accuracy * 100 ))
# Step 8: Train with AdversarialTrainer for accuracy comparision
# Paper link: https://arxiv.org/abs/1705.07204
AdversarialTrainer(classifier =classifier , attacks =attack , ratio =0.5 ).fit(x =x_train , y =y_train
, batch_size =64 , nb_epochs =20 )
# Step 9: Evaluate the ART classifier on adversarial test examples
predictions = AdversarialTrainer(classifier =classifier , attacks =attack , ratio =0.5 ).predict(x
=x_test_adv )
accuracy = np .sum (np .argmax (predictions , axis =1 ) == np .argmax (y_test , axis =1 )) / len
(y_test )
print ("AdversarialTrainer - DeepFool을 방어한 후의 모델 정확도: {}".format (accuracy * 100 ))
sec = time .time () - start
times = str (datetime .timedelta (seconds =sec )).split (".")
times = times [0 ]
print (times )

```

Adversarial Trainer를 사용하여 동일한 조건으로 모델을 학습시킨 후, 공격 받은 테스트 셋을 predict하여 기존의 일반적인 학습 이후 공격 받은 테스트 셋의 predict 값이랑 비교해본다.



## - 개선 전후 비교

```

Python Console - IndependentCapstone
04_CIFAR10_DEEPFOOL_DEFEND (1) x

765/781 [=====>.] - ETA: 1s - loss: 0.4117 - accuracy: 0.8560
766/781 [=====>.] - ETA: 1s - loss: 0.4116 - accuracy: 0.8560
767/781 [=====>.] - ETA: 1s - loss: 0.4116 - accuracy: 0.8559
768/781 [=====>.] - ETA: 1s - loss: 0.4115 - accuracy: 0.8559
769/781 [=====>.] - ETA: 1s - loss: 0.4115 - accuracy: 0.8559
770/781 [=====>.] - ETA: 1s - loss: 0.4113 - accuracy: 0.8560
771/781 [=====>.] - ETA: 1s - loss: 0.4112 - accuracy: 0.8560
772/781 [=====>.] - ETA: 0s - loss: 0.4113 - accuracy: 0.8559
773/781 [=====>.] - ETA: 0s - loss: 0.4112 - accuracy: 0.8559
774/781 [=====>.] - ETA: 0s - loss: 0.4114 - accuracy: 0.8558
775/781 [=====>.] - ETA: 0s - loss: 0.4113 - accuracy: 0.8558
776/781 [=====>.] - ETA: 0s - loss: 0.4112 - accuracy: 0.8559
777/781 [=====>.] - ETA: 0s - loss: 0.4113 - accuracy: 0.8559
778/781 [=====>.] - ETA: 0s - loss: 0.4113 - accuracy: 0.8558
779/781 [=====>.] - ETA: 0s - loss: 0.4112 - accuracy: 0.8559
780/781 [=====>.] - ETA: 0s - loss: 0.4109 - accuracy: 0.8560
781/781 [=====] - 82s 106ms/step - loss: 0.4108 - accuracy: 0.8560
정상적으로 학습시킨 CIFAR-10 모델의 정확도: 78.97999999999999%
DeepFool: 100%|██████████| 10000/10000 [04:53<00:00, 34.10it/s]
CIFAR-10에 DeepFool 공격을 가한 후 정확도: 14.41%
Precompute adv samples: 100%|██████████| 1/1 [00:00<?, ?it/s]
Adversarial training epochs: 100%|██████████| 20/20 [6:28:35<00:00, 1165.77s/it]
AdversarialTrainer - DeepFool을 방어한 후의 모델 정확도: 52.76%
7:02:08

>>>
  
```

그림 7 CIFAR-10 Adversarial Trainer DeepFool Defend

정상적으로 학습시킨 CIFAR-10 모델의 정확도는 78.98%, MNIST에 FGSM 공격을 가한 후 정확도는 14.41%이다. 이후 Adversarial Trainer를 통해 새로 모델을 학습시키고 이후 FGSM 공격을 받았던 테스트 셋을 predict 한 값은 52.76%가 나왔다. 이를 통해 Adversarial Trainer를 사용하여 14.1%였던 정확도를 52.76%까지 끌어올렸음을 알 수 있다. 또한 CIFAR-10의 경우에는 일반적으로 학습시킨 모델의 정확도가 그렇게 높지 못하여 epoch 값을 20으로 크게 잡아서 adversarial trainer의 epoch 값도 20으로 잡게 되었는데, epoch 값을 조금 적게 잡는다면 상대적으로 adversarial trainer가 조금 더 좋은 성능을 보여준다.

## 4.3 IRIS - JSMA 공격 결과 및 개선 방안과 평가

### - 공격 결과

```
Python Console - IndependantCapstone
03_CIFAR10_DEEPFOOL x 05_IRIS_JSMA x

Epoch 697/700
1/15 [=>.....] - ETA: 0s - loss: 0.2722 - accuracy: 1.0000
15/15 [=====] - 0s 2ms/step - loss: 0.3178 - accuracy: 0.9048
Epoch 698/700
1/15 [=>.....] - ETA: 0s - loss: 0.1626 - accuracy: 1.0000
15/15 [=====] - 0s 2ms/step - loss: 0.3665 - accuracy: 0.8762
Epoch 699/700
1/15 [=>.....] - ETA: 0s - loss: 0.9641 - accuracy: 0.5714
15/15 [=====] - 0s 2ms/step - loss: 0.2758 - accuracy: 0.9238
Epoch 700/700
1/15 [=>.....] - ETA: 0s - loss: 0.1291 - accuracy: 1.0000
15/15 [=====] - 0s 2ms/step - loss: 0.2959 - accuracy: 0.8762
정상적으로 학습시킨 IRIS 모델의 정확도: 95.55555555555556%
JSMA: 100%|██████████| 45/45 [00:01<00:00, 43.73it/s]
IRIS에 JSMA 공격을 가한 후 정확도: 33.33333333333333%
0:00:21
>>>
```

그림 8 IRIS JSMA Attack

IRIS 데이터 셋을 학습시킨 후 해당 AI 모델을 Deep Fool로 adversarial attack하여, 기존의 AI 모델의 정확도와 FGSM 공격 당한 모델의 정확도를 비교하였다. 기존의 AI 모델의 정확도는 78.60%, 공격 후 정확도는 14.96%로 낮아졌다. 이는 약 63.64%p가 낮아진 것이다.

#### - 개선 방안

Adversarial Trainer를 사용하여 해당 모델을 학습시킨 후, 기존의 일반적으로 학습 시킨 모델의 정확도, JSMA를 공격을 가한 테스트 값을 일반 모델로 predict한 정확도 그리고 Adversarial Trainer를 사용하여 학습시킨 모델로 JSMA 공격을 한 테스트 값의 predict 값을 비교하여 Adversarial Trainer를 사용함이 실제로 AI 모델 공격 보안에 효과가 있는지를 확인해 보도록 한다.

#### - 개선 방안 적용 후 공격 (06\_IRIS\_JSMA\_DEFEND.py)

```
# 2019112130 Yangjin Cho
# 06_IRIS_JSMA_DEFEND
# Independant Capstone AI Model Security
import keras
import datetime
import time
```



```

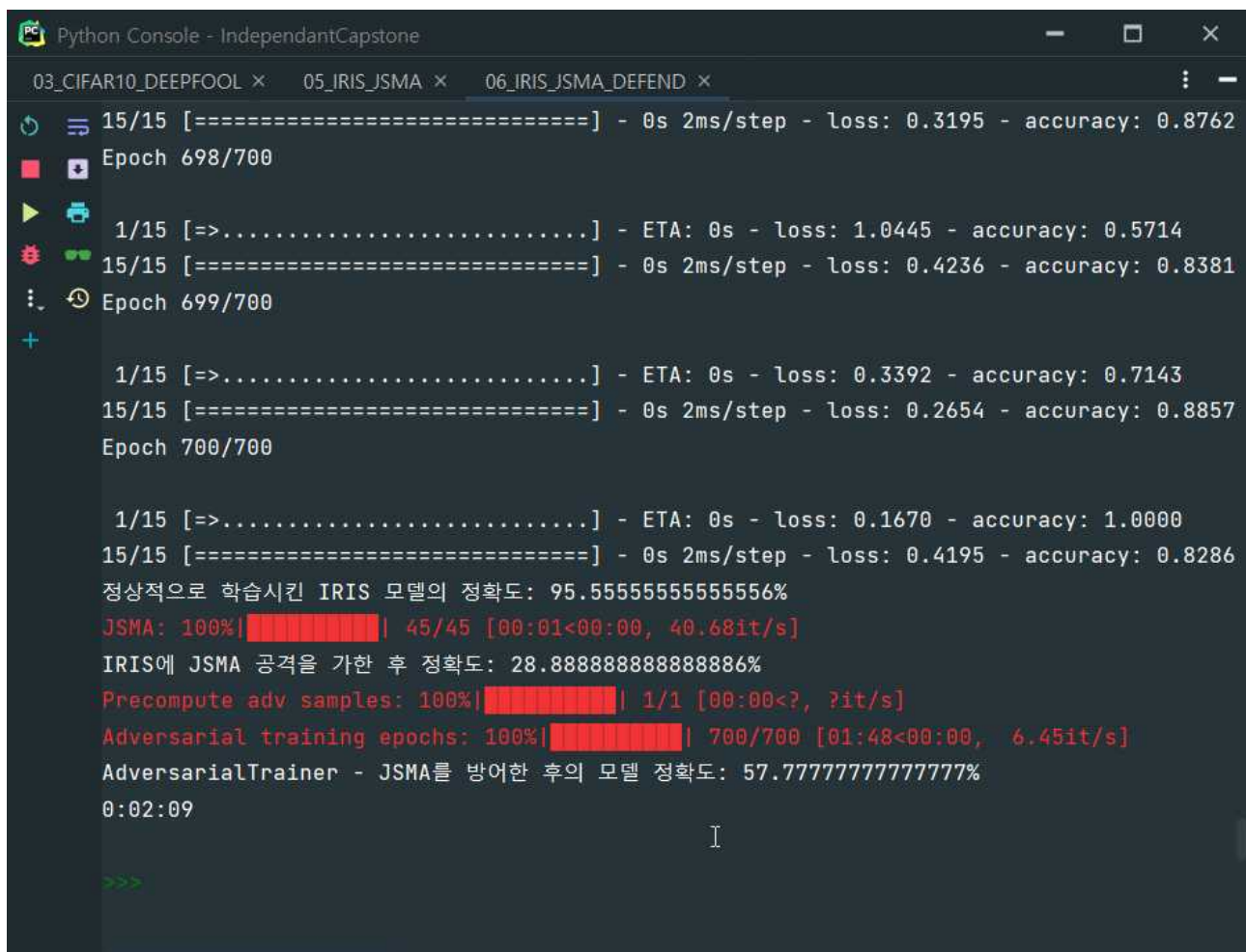
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
import numpy as np
from art.attacks.evasion import SaliencyMapMethod
from art.estimators.classification import KerasClassifier
from art.utils import load_iris
from art.defences.trainer import AdversarialTrainer
start = time.time ()
# Step 1: Load the IRIS dataset
(x_train , y_train ), (x_test , y_test ), min_pixel_value , max_pixel_value = load_iris()
# Step 2: Create the model
# https://www.kaggle.com/rushabhwdkar/deep-learning-with-keras-on-iris-dataset
model = Sequential()
model.add(Dense(10 ,input_dim =4 , activation = 'relu',kernel_initializer = 'he_normal'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dropout(0.3 ))
model.add(Dense(7 , activation = 'relu', kernel_initializer = 'he_normal',kernel_regularizer = keras
.regularizers.l1_l2(l1 =0.001 , l2 =0.001 )))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dropout(0.3 ))
model.add(Dense(5 , activation = 'relu', kernel_initializer = 'he_normal',kernel_regularizer = keras
.regularizers.l1_l2(l1 =0.001 , l2 =0.001 )))
model.add(Dense(3 ,activation = 'softmax'))
model.compile(
    loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy']
)
# Step 3: Create the ART classifier
classifier = KerasClassifier(model =model , clip_values =(min_pixel_value , max_pixel_value
), use_logits =False )
# Step 4: Train the ART classifier
classifier.fit(x_train , y_train , batch_size =7 , nb_epochs =700 )
# Step 5: Evaluate the ART classifier on benign test examples
predictions = classifier.predict(x_test )
accuracy = np.sum (np.argmax (predictions , axis =1 ) == np.argmax (y_test , axis =1 )) / len
(y_test )
print ("정상적으로 학습시킨 IRIS 모델의 정확도: {}".format (accuracy * 100 ))
# Step 6: Generate adversarial test examples
attack = SaliencyMapMethod(classifier =classifier )
x_test_adv = attack.generate(x =x_test )
# Step 7: Evaluate the ART classifier on adversarial test examples
predictions = classifier.predict(x_test_adv )
accuracy = np.sum (np.argmax (predictions , axis =1 ) == np.argmax (y_test , axis =1 )) / len
(y_test )
print ("IRIS에 JSMA 공격을 가한 후 정확도: {}".format (accuracy * 100 ))
# Step 8: Train with AdversarialTrainer for accuracy comparision
# Paper link: https://arxiv.org/abs/1705.07204
AdversarialTrainer(classifier =classifier , attacks =attack , ratio =0.5 ).fit(x =x_train , y =y_train
, batch_size =7 , nb_epochs =700 )
# Step 9: Evaluate the ART classifier on adversarial test examples
predictions = AdversarialTrainer(classifier =classifier , attacks =attack , ratio =0.5 ).predict(x
=x_test_adv )
accuracy = np.sum (np.argmax (predictions , axis =1 ) == np.argmax (y_test , axis =1 )) / len

```

```
(y_test )
print ("AdversarialTrainer - JSMA를 방어한 후의 모델 정확도: {}".format (accuracy * 100 ))
sec = time .time () - start
times = str (datetime .timedelta (seconds =sec )).split (".")
times = times [0 ]
print (times )
```

Adversarial Trainer를 사용하여 동일한 조건으로 모델을 학습시킨 후, 공격 받은 테스트 셋을 predict하여 기존의 일반적인 학습 이후 공격 받은 테스트 셋의 predict 값이랑 비교해본다.

#### - 개선 전후 비교



```
Python Console - IndependantCapstone
03_CIFAR10_DEEPFOOL × 05_IRIS_JSMA × 06_IRIS_JSMA_DEFEND ×
15/15 [=====] - 0s 2ms/step - loss: 0.3195 - accuracy: 0.8762
Epoch 698/700
1/15 [=>.....] - ETA: 0s - loss: 1.0445 - accuracy: 0.5714
15/15 [=====] - 0s 2ms/step - loss: 0.4236 - accuracy: 0.8381
Epoch 699/700
1/15 [=>.....] - ETA: 0s - loss: 0.3392 - accuracy: 0.7143
15/15 [=====] - 0s 2ms/step - loss: 0.2654 - accuracy: 0.8857
Epoch 700/700
1/15 [=>.....] - ETA: 0s - loss: 0.1670 - accuracy: 1.0000
15/15 [=====] - 0s 2ms/step - loss: 0.4195 - accuracy: 0.8286
정상적으로 학습시킨 IRIS 모델의 정확도: 95.55555555555556%
JSMA: 100%|██████████| 45/45 [00:01<00:00, 40.68it/s]
IRIS에 JSMA 공격을 가한 후 정확도: 28.888888888888886%
Precompute adv samples: 100%|██████████| 1/1 [00:00<?, ?it/s]
Adversarial training epochs: 100%|██████████| 700/700 [01:48<00:00, 6.45it/s]
AdversarialTrainer - JSMA를 방어한 후의 모델 정확도: 57.77777777777777%
0:02:09
>>>
```

그림 9 IRIS Adversarial Trainer JSMA Defend

정상적으로 학습시킨 IRIS 모델의 정확도는 95.56%, MNIST에 FGSM 공격을 가한 후 정확도는 28.89%이다. 이후 Adversarial Trainer를 통해 새로 모델을 학습시키고 이후 FGSM 공격을 받았던 테스트 셋을 predict 한 값은 57.78%가 나왔다. 이를 통해 Adversarial Trainer를 사용하여 28.89%였던 정확도를 57.78%까지 끌어올렸음을 알 수 있다.

프로세스   성능   앱 기록   시작프로그램   사용자   세부 정보   서비스					
이름	상태	100% CPU	65% 메모리	0% 디스크	0% 네트워크
PC PyCharm(5)		92.5%	4,025.5MB	0MB/s	0Mb/s
Python		45.2%	1,777.6MB	0MB/s	0Mb/s
Python		30.1%	649.3MB	0MB/s	0Mb/s
Python		16.6%	237.6MB	0MB/s	0Mb/s
PyCharm		0.6%	1,360.8MB	0MB/s	0Mb/s
Filesystem events processor		0%	0.2MB	0MB/s	0Mb/s
System		3.5%	0.1MB	0.1MB/s	0Mb/s
Discord(32비트)(6)		2.0%	252.9MB	0MB/s	0.1Mb/s
작업 관리자		0.7%	24.0MB	0.1MB/s	0Mb/s
콘솔 창 호스트		0.4%	3.8MB	0MB/s	0Mb/s
콘솔 창 호스트		0.2%	5.7MB	0MB/s	0Mb/s
Antimalware Service Executable		0.2%	138.1MB	0.1MB/s	0Mb/s
데스크톱 창 관리자		0.2%	17.0MB	0MB/s	0Mb/s
winpty-agent		0.1%	1.0MB	0MB/s	0Mb/s
System Interrupts		0.1%	0MB	0MB/s	0Mb/s
Windows 탐색기		0.1%	33.6MB	0MB/s	0Mb/s
Microsoft Office Click-to-Run (...)		0.1%	19.0MB	0MB/s	0Mb/s

간단히(D)
작업 끝내기(E)

그림 10 메모리 리소스 사용률

우선 리소스 사용률 비교 차원에서, 이는 개선 방법과는 관계가 없지만 공격 기법과 더욱 관련이 있는 내용으로 위 사진에서 위에서 아래 순서대로 DeepFool, JSMA, FGSM 순서로 프로세스가 진행되고 있다. Pycharm Professional에서는 여러 프로세스를 venv에서 같이 실행할 수 있지만, 대신 CPU의 자원을 서로 나누어 가져가므로 CPU 리소스 사용률은 크게 의미가 없는 값이다. 대신 메모리 사용량은 의미가 있다. DeepFool은 약 1.8기가, JSMA는 약 650메가, FGSM은 약 240메가의 메모리를 사용한다는 것을 알 수 있었고, 이는 프로세스를 하나만 실행 시켰을 때도 동일했다. (CPU의 경우는 무조건 100%를 사용하게끔 고정되었다.) 따라서 공격 방법 중에서 DeepFool -> JSMA -> FGSM 순서로 메모리를 많이 사용함을 알 수 있었다.

이후 Adversarial Trainer의 성능을 평가하기 위해, adversarial attack의 공격 정도 (eg. 엡실론 혹은 세타 값)을 조정해가면서 실험해 보았다. FGSM, DeepFool, JSMA 모두 시간적으로 각자 비슷한 시간이 소요되어 (학습에 걸리는 시간은 비슷하기 때문) 이는 무의미한 데이터라 생각하여 생략했다.

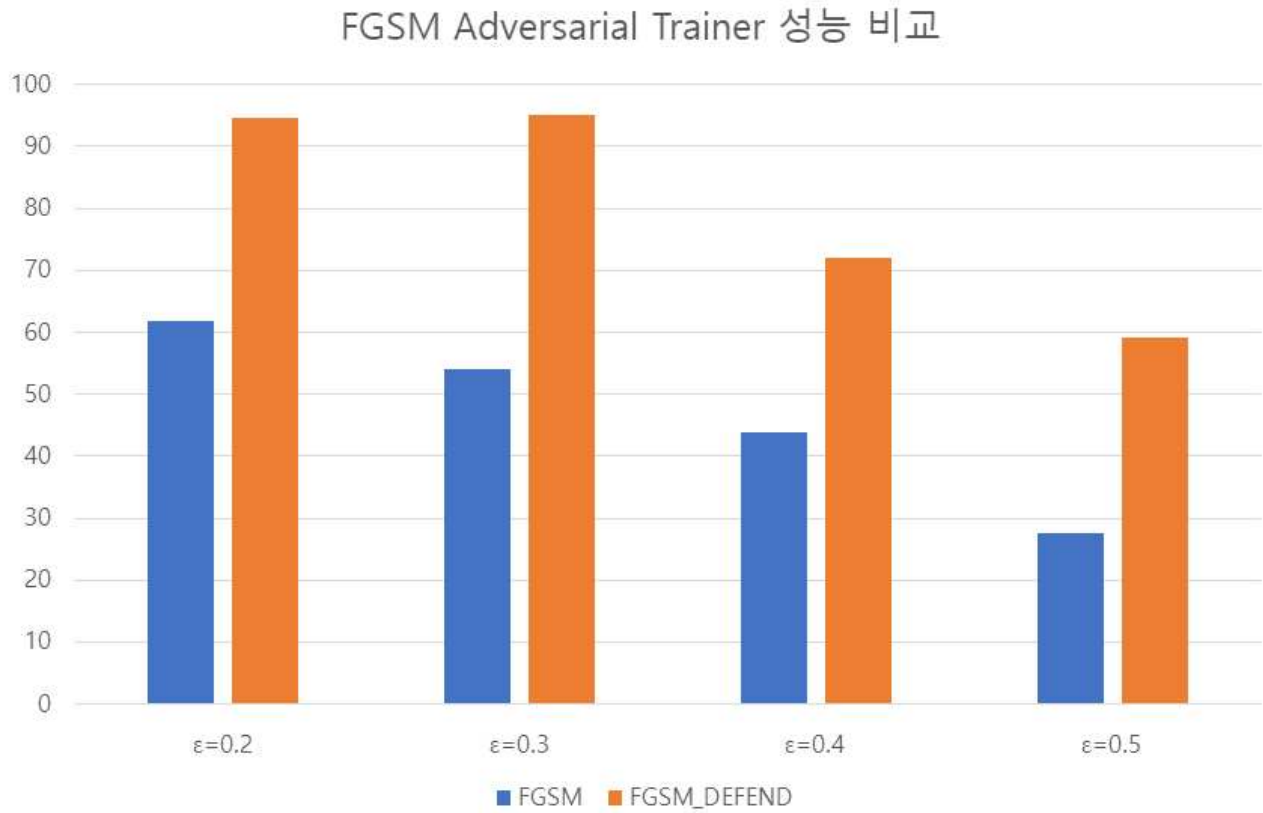


그림 11 FGSM Adversarial Trainer 성능 비교

최종적으로 결과를 정리하기 위해 앞서, FGSM의 경우 엡실론 값을 0.1씩 점진적으로 증가시켰다. 표의 FGSM이 FGSM 공격을 시행한 테스트 셋을 일반 학습 시킨 모델에 넣은 predict 값, FGSM\_DEFEND는 FGSM 공격을 시행한 테스트 셋을 Adversarial trainer에 학습 시킨 모델에 넣은 predict 값이다.

	0.2	0.3	0.4	0.5
FGSM	61.7	53.97	43.8	27.48
FGSM_DEFEND	94.56	94.97	72.07	59.05

표 1 FGSM Adversarial Trainer 성능 비교

엡실론 값이 0.3일 때까지는 좋은 성능을 보여주다, 0.4가 되는 순간 72.07%로 많이 하락하고, 0.5가 되면 60퍼센트 미만으로 내려갔다. 특히 Adversarial attack 논문을 확인해보면 MNIST 데이터셋을 상정하고 작성한 부분도 있는 것에 (기대에) 비해 아주 높지는 못했다. 그래도 엡실론 값이 0.4가 되면 실제로 데이터셋에 눈에 띄게 잡음 (노이즈)가 들어가있는 것을 육안으로도 확인할 수 있으니 이는 당연한 결과일 수도 있다.

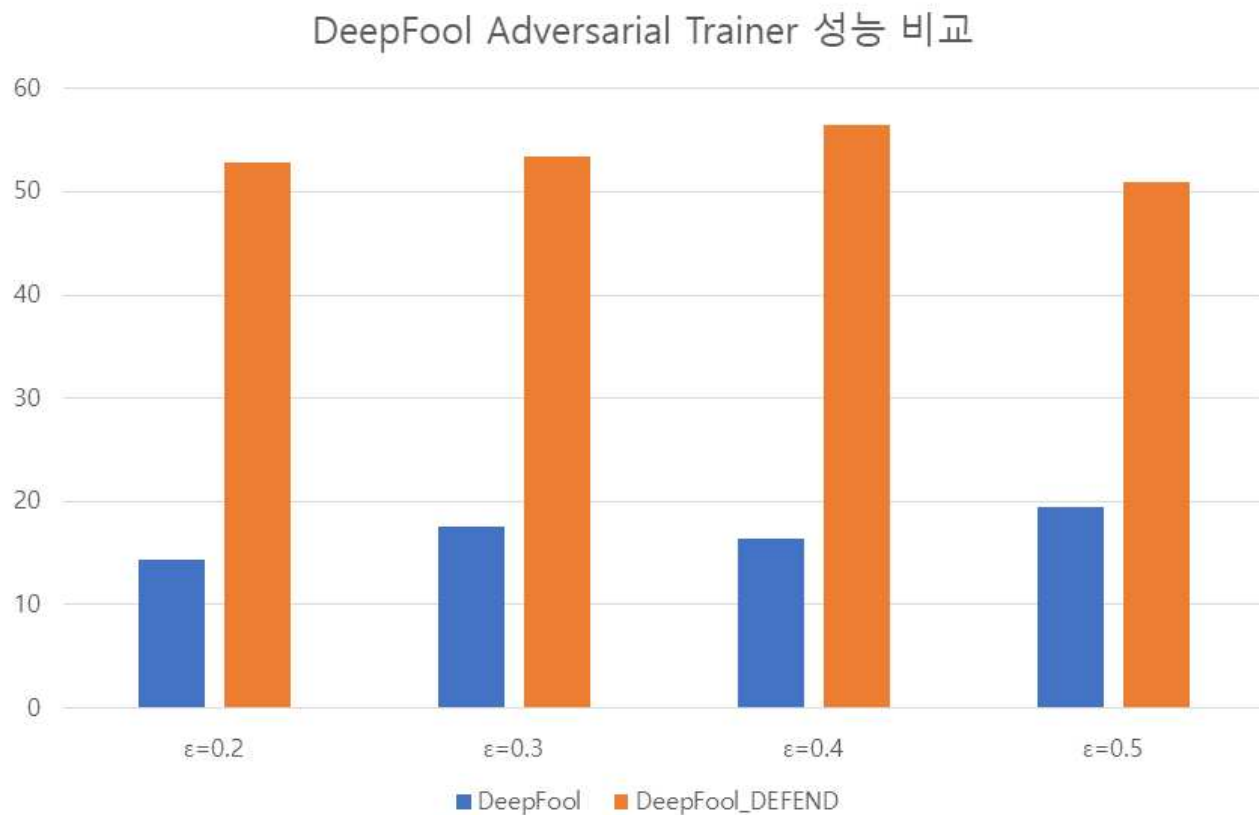


그림 12 DeepFool Adversarial Trainer 성능 비교

두 번째로, DeepFool의 경우 엡실론 값을 0.1씩 점진적으로 증가시켰다. 표의 DeepFool이 DeepFool 공격을 시행한 테스트 셋을 일반 학습시킨 모델에 넣은 predict 값, DeepFool\_DEFEND는 DeepFool 공격을 시행한 테스트 셋을 Adversarial trainer에 학습시킨 모델에 넣은 predict 값이다.

	0.2	0.3	0.4	0.5
DeepFool	14.41	17.55	16.43	19.41
DeepFool_DEFEND	52.76	53.46	56.43	50.87

표 2 DeepFool Adversarial Trainer 성능 비교

DeepFool 공격의 경우, FGSM과는 다르게 꾸준히 방어한 값이 50퍼센트 대를 유지함을 확인할 수 있었다. 또한 일정한 패턴을 보인다고 보다, 오히려 DeepFool 공격을 받은 테스트 셋의 predict 값이 점진적으로 증가하는 듯한 모습을 보이기 까지 하였다.

마지막으로 JSMA의 경우 세타 값을 0.1 씩 점진적으로 증가시켰다 (기본 값=0.1). 이는 FGSM과 DeepFool의 엡실론과는 다르게 공격 강도의 지표가 아닌, 변수를 주어 방어 값이 어떻게 달라지는가를 확인하기 위함이다. 표의 JSMA이 JSMA 공격을 시행한 테스트 셋을 일반 학습시킨 모델에 넣은 predict 값, JSMA\_DEFEND는 JSMA 공격을 시행한 테스트 셋을 Adversarial trainer에 학습시킨 모델에 넣은 predict 값이다.

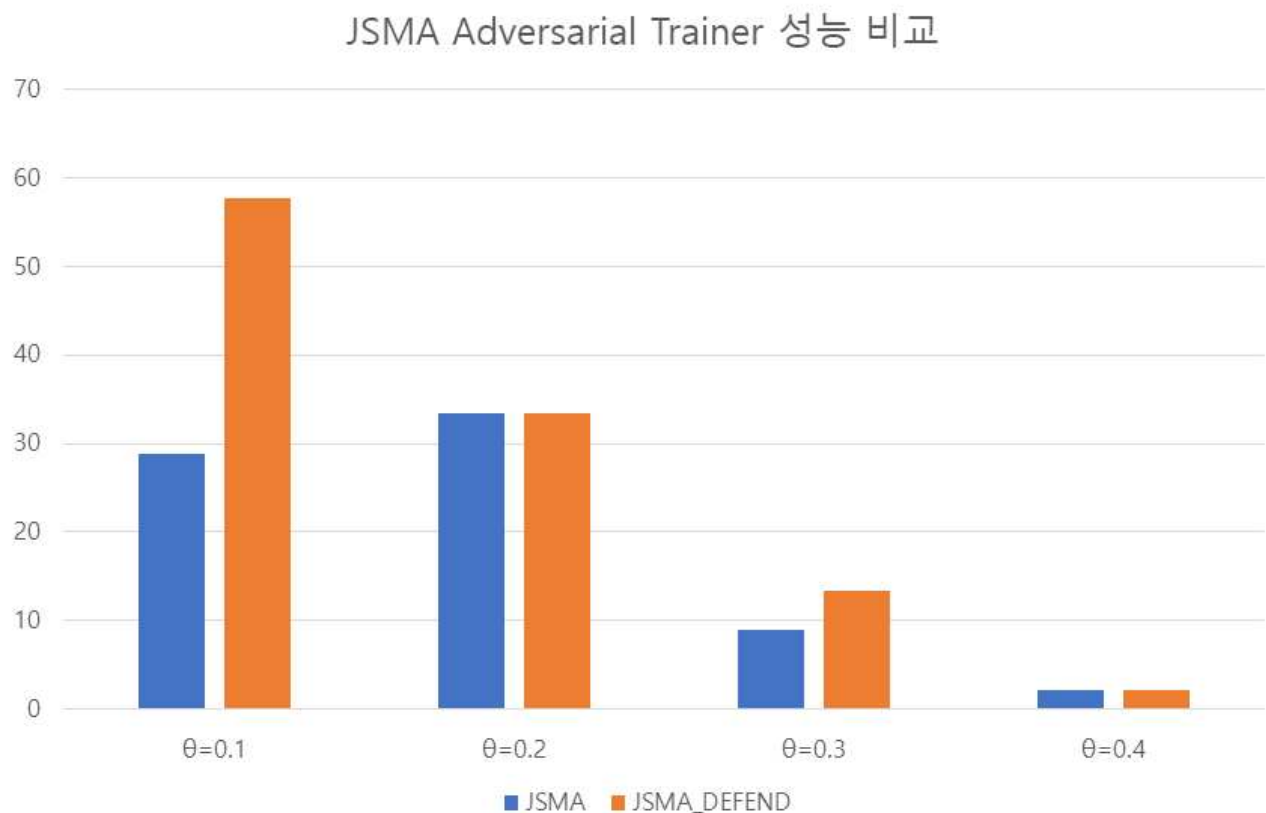


그림 13 JSMA Adversarial Trainer 성능 비교

JSMA의 경우는 세타 값을 조정한 결과 아쉬운 결과값을 얻을 수 있었다. 0.1의 경우에는 확실히 방어가 되는 듯 했으나, 0.2, 0.4의 경우에는 공격 받았던 그 자체 그 대로의 결과 값을 보여줬다.

	0.1	0.2	0.3	0.4
JSMA	28.89	33.34	8.89	2.22
JSMA_DEFEND	57.78	33.33	13.33	2.22

표 3 JSMA Adversarial Trainer 성능 비교

FGSM과 DeepFool에서는 어느정도 방어가 보장되는 Adversarial Trainer가 JSMA에서만 전혀 방어를 하지 못하고 있는 결과에 의구심을 갖게되어, 다른 방어책인 Adversarial Trainer Madry PGD<sup>7)</sup>를 사용해보기로 하였다. 해당 Adversarial Trainer는 기존의 FGSM 방식의 트레이너가 아닌 PGD (Projected Gradient Descent) 공격 법을 통해 모델의 robustness를 지키는 방법으로, PGD도 마찬가지로 evasion attack의 한 종류이다. 본래 해당 트레이너의 최대 성능을 확인하기 위해서는 PGD로 공격한 데이터셋을 가져와 정확도를 측정해보는 것이 제일이지만, JSMA에 적당한 방어법을 찾지 못하였기 때문에 이를 대신 사용해 보기로 하였다.

- 2차 개선 방안 적용 후 공격 (07\_IRIS\_JSMA\_DEFEND(2).py)



```

# 2019112130 Yangjin Cho
# 07_IRIS_JSMA_DEFEND(2)
# Independant Capstone AI Model Security
import keras
import datetime
import time
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
import numpy as np
from art.attacks.evasion import SaliencyMapMethod
from art.estimators.classification import KerasClassifier
from art.utils import load_iris
from art.defences.trainer import AdversarialTrainerMadryPGD
start = time.time()
# Step 1: Load the IRIS dataset
(x_train, y_train), (x_test, y_test), min_pixel_value, max_pixel_value = load_iris()
# Step 2: Create the model
# https://www.kaggle.com/rushabhwadkar/deep-learning-with-keras-on-iris-dataset
model = Sequential()
model.add(Dense(10, input_dim=4, activation='relu', kernel_initializer='he_normal'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dropout(0.3))
model.add(Dense(7, activation='relu', kernel_initializer='he_normal', kernel_regularizer=keras.regularizers.L1_L2(l1=0.001, l2=0.001)))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dropout(0.3))
model.add(Dense(5, activation='relu', kernel_initializer='he_normal', kernel_regularizer=keras.regularizers.L1_L2(l1=0.001, l2=0.001)))
model.add(Dense(3, activation='softmax'))
model.compile(
    loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']
)
# Step 3: Create the ART classifier
classifier = KerasClassifier(model=model, clip_values=(min_pixel_value, max_pixel_value), use_logits=False)
# Step 4: Train the ART classifier
classifier.fit(x_train, y_train, batch_size=7, nb_epochs=700)
# Step 5: Evaluate the ART classifier on benign test examples
predictions = classifier.predict(x_test)
accuracy = np.sum(np.argmax(predictions, axis=1) == np.argmax(y_test, axis=1)) / len(y_test)
print("정상적으로 학습시킨 IRIS 모델의 정확도: {}".format(accuracy * 100))
# Step 6: Generate adversarial test examples
attack = SaliencyMapMethod(classifier=classifier, theta=0.4)
x_test_adv = attack.generate(x=x_test)
# Step 7: Evaluate the ART classifier on adversarial test examples
predictions = classifier.predict(x_test_adv)
accuracy = np.sum(np.argmax(predictions, axis=1) == np.argmax(y_test, axis=1)) / len(y_test)
print("IRIS에 JSMA 공격을 가한 후 정확도: {}".format(accuracy * 100))
# Step 8: Train with AdversarialTrainer Madry PGD for accuracy comparison
# Paper link: https://arxiv.org/abs/1706.06083
trainer = AdversarialTrainerMadryPGD(classifier=classifier, nb_epochs=700, batch_size=7)
trainer.fit(x=x_train, y=y_train)
classifier_trained = trainer.get_classifier()
# Step 9: Evaluate the ART classifier on adversarial test examples
accuracy = np.sum(np.argmax(classifier_trained.predict(x_test_adv), axis=1) == np.argmax(y_test, axis=1))

```

```

/ len(y_test)
print("AdversarialTrainerMadryPGD - JSMA를 방어한 후의 모델 정확도: {}".format(accuracy * 100))
sec = time.time() - start
times = str(datetime.timedelta(seconds=sec)).split(".")
times = times[0]
print(times)

```

#### - 공격 결과

```

정상적으로 학습시킨 IRIS 모델의 정확도: 100.0%
JSMA: 100%|██████████| 45/45 [00:00<00:00, 54.43it/s]
Precompute adv samples: 100%|██████████| 1/1 [00:00<?, ?it/s]
Adversarial training epochs: 0%|          | 0/700 [00:00<?, ?it/s]IRIS에 JSMA 공격을 가한 후 정확도: 20.0%
Adversarial training epochs: 100%|██████████| 700/700 [01:56<00:00, 5.99it/s]
AdversarialTrainerMadryPGD - JSMA를 방어한 후의 모델 정확도: 33.33333333333333%
0:02:16

```

그림 14 JSMA Adversarial Trainer Madry PGD 성능 비교 (세타 0.4)

기존의 Adversarial Trainer가 JSMA에서 한번도 지켜내지 못했던 정확도를 확인할 수 있었다. 물론 33.3%의 정확도를 가지는 AI 모델을 사용하면 안되지만, 처참했던 기존의 세타 0.4일때의 공격 방어율을 생각하면 큰 개선이라 할 수 있다. 이후 세타 값을 변경하여 다른 결과값들을 확인해 보았다.

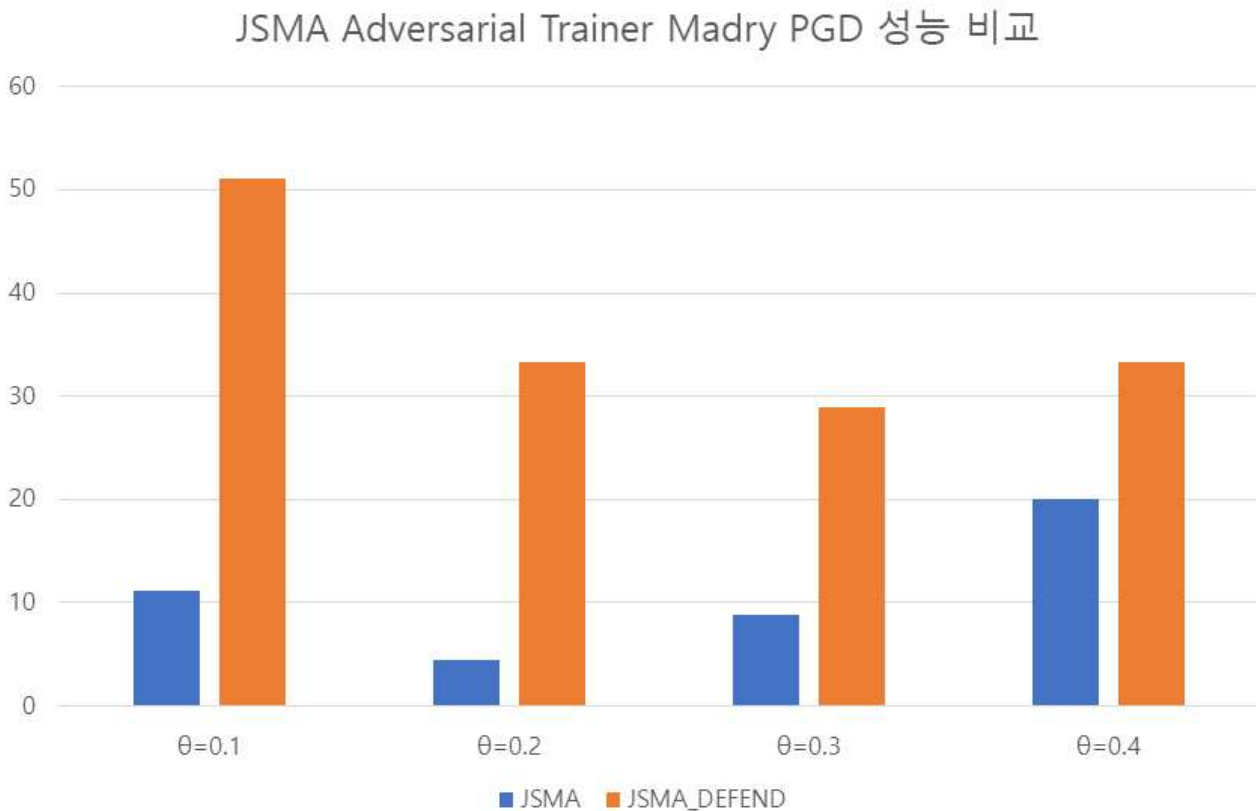


그림 15 JSMA Adversarial Trainer Madry PGD 성능 비교



	0.1	0.2	0.3	0.4
JSMA	11.11	4.44	8.89	20
JSMA_DEFEND	51.11	33.33	28.89	33.33

표 4 JSMA Adversarial Trainer Madry PGD 성능 비교

데이터를 살펴보면 세타 0.1, 0.2 값일 때는 전과 별 차이 없는 값을 보여주었으나, 세타 0.3, 0.4로 변경한 부분에서 기존의 Adversarial Trainer에 비해 상대적으로 큰 발전을 보여줌을 확인할 수 있었다.

## 5. 결론 및 향후 연구

이번 AI모델 Evasion Attack 보안 강화 방법 연구를 통해 AI 모델 공격 기법 중 FGSM, DeepFool, JSMA를 MNIST, CIFAR-10, IRIS 데이터셋을 사용한 AI 모델에 직접 적용해 보고, 이를 방어하기 위해 Adversarial Trainer를 사용해 보았다. 이를 통해 실제로 AI 모델의 exploit을 이용한 악의적인 공격이 가능함을 확인했고, 이를 방어하는 방법도 실제로 적용할 수 있다는 것을 알 수 있었다. 그러나 방어에 사용한 adversarial training은 무조건 evasion attack 만을 방어할 수 있다는 점과 epoch 값이 늘어날수록 원래의 정확도를 복구하기가 힘들어진다는 단점이 있다. 또한 특정 attack에 대한 값을 입력 받아야지만 해당 공격의 robustness함을 높힐 수 있다는 것도 현실적인 방어 대책으로 쓰기 어려운 부분이다. 그 어느 누구도 자신의 공격 방법을 알리며 공격하는 일은 없을 것이기 때문이다.

또한 ART의 postprocess와 preprocess, 그리고 detector를 사용한 방어는 어떤 방법으로 이루어지는지 더욱 알아보고 싶다. 위 보고서에는 작성하지 않았으나 art.defences.detector.evasion의 'Binary Input Detector'를 사용하여 공격 받은 데이터 셋의 변조성을 확인해보고, art.defences.postprocessor의 'High Confidence'와 'Gaussian Noise'를 통해 후처리 방식을 이용한 공격받은 데이터셋에 대한 predict 값을 얻어보려 했으나 의미있는 데이터를 얻는데 실패하였다. 이 부분을 보완하여 다음번에 ART를 사용할 때는 트레이너 뿐만 아니라 다른 기법들도 사용해보고 싶다.

향후 연구로서는 딥러닝 기반 컴퓨터 보조진단의 취약점을 보완하기 위하여 NoduleX의 훈련과 검증에 사용된 'Lung Image Database Consortium & Image Database Resource Initiative'(LIDC-IDRI) 데이터 세트 (흉부 방사선 의사에 의해 병변이 표시된 진단 및 폐암 검사 흉부 CT 영상)을 AI에 학습시켜 적대적 공격을 행해보고 싶다. 또한 방어 기법 중 preprocess, postprocess와 detector를 복합적으로 사용하여 가능한 많은 공격을 막을 수 있는 AI 모델 학습 시스템을 구축해보고 싶다.

## 참고문헌

- 1) 이재원, 김종효, 「딥러닝 기반 컴퓨터 보조진단의 취약성: 폐 결절 검출 모델에 대한 실험적 적대적 공격」, 대한의학영상정보학회지 2018;24:1-10
- 2) Goodfellow IJ, Jonathon S, Christian S. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572, 2014
- 3) Moosavi-Dezfooli SM, Alhussein F, Pascal F. Deepfool: a simple and accurate method to fool deep

- neural networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016; 2574-
- 4) Jang U, Xi W, Somesh J. Objective Metrics and Gradient Descent Algorithms for Adversarial Examples in Machine Learning. Proceedings of the 33<sup>rd</sup> Annual Computer Security Applications Conference. ACM, 2017; 262-277
  - 5) Szegedy C, et al. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.
  - 6) Florian Tramèr Ensemble Adversarial Training: Attacks and Defenses, arXiv:1705.07204, 2017.
  - 7) Towards Deep Learning Models Resistant to Adversarial Attacks, arXiv:1706.06083, 2019.
  - 8) <https://adversarial-robustness-toolbox.readthedocs.io/en/latest/index.html>