

A Study on the threat of Adversarial Machine Learning: The Projected Gradient Descent (PGD) Attack

Abstract:

Adversary machine learning has witnessed a major leap regarding machine learning and artificial intelligence security. Our professionals currently tend to go for trial-and-error strategies to immune the machine learning models from adversary attacks. Recently, the reliability of machine learning operating systems in the presence of adversary threats has become a major field of interest for both academia and tech. industry. Machine learning models learn from the received data from the real world which can be easily corrupted by evasion attacks that craft adversary samples which shall deceive the operating system. A common defence against these attacks is the pre-processor defence mechanism: that depends on cleaning out malicious input by cleaning noise pixels penetrations to make the adversary example seem cleaner to the attacked model.

Keywords: Adversarial Machine Learning, Adversarial ML Attacks, Adversarial ML Defences, Projected Gradient Descent (PGD) Attack, Pre-Processing Defence Techniques.

List of Content

Glossary of Terms and Abbreviations	4
The problem	5
1.1 Background.....	5
1.2 Research Questions.....	5
1.3 Description of the Research Problem and Research Issues	5
1.4 Goals of this Study.....	5
1.4.1 Goal of the Literature Review	6
1.4.2 Goal of the Experimental Evaluation	6
Chapter 2:.....	7
Literature Review	7
2.1 Taxonomy	7
2.2 Technical detail.....	8
2.2.1 Systematic Review of the Existing Approaches	8
2.2.2 Elaboration of the Major Approaches of Perspectives	8
2.3 Critical Discussion.....	11
2.3.1 Strengths and Limitations of the Existing Approaches	11
2.3.2 Comparisons between the Major Approaches	12
Chapter 3:.....	13
Software Overview.....	13
3.1 An Overview on the Relevant Software Frameworks	13
3.2 Comparisons between Different Software Frameworks	13
Chapter 4:.....	15
Practical Evaluation	15
4.1 Software Toolkit Selected for Practical Evaluation.....	15
4.2 Setup of the Software.....	15
4.2.1 Installation of the Software.....	15
4.2.2 Validation	16
4.2.3 Dataset	17
4.3 Experimental Scenarios and Results.....	18

University for the Common Good

4.3.1 Experimental Scenarios	18
4.3.2 Performance Metrics for Evaluation.....	18
4.3.3 Comparisons and Results of Running the Experiment	23
References:	24
Appendices	26

List of Figures

Figure (1)	5
Figure (2)	6
Figure (3)	7

List of Tables

Table (1)	12
------------------------	----

Glossary of Terms and Abbreviations

Glossary	
Adversarial perturbation	The noise applied to a raw sample in order to craft an adversary example.
Adversary	The agent who conducts an attack using an adversarial example.
Black-box attack	No knowledge, untargeted attack
Gray-box attack	Limited knowledge attack
White-box attack	Perfect knowledge attack
Evasion attack	An attack in which the adversary tampers with the input samples in order to trick a previously trained classifier into making a false labelling during the testing phase.
FGSM	Fast Gradient Sign Method: An efficient attack technique for computing a single layer adversarial pixel perturbation.
PGD	The Projected Gradient Descent: The most efficient attack technique for computing a multi-layer adversarial pixel perturbation.
Gradient masking	A method of gradients rate reduction that is utilized to make machine learning models more resistant to adversarial samples.
Robustness	The competence of machine learning algorithms to maintain reliable classification against adversarially manipulated data
Logic Corruption	The process of altering the attacked model's learning algorithms to misclassify its output.
FS	Feature Squeezing: It is a pre-processor defence technique
SS	Spatial Smoothing: It is a pre-processor defence technique
GAN	Generative Adversarial Network
DNN	Deep Neural Network
AI	Artificial Intelligence
ML	Machine Learning

Chapter 1:

The problem

1.1 Background

Machine learning is a computer science discipline that lies under the big umbrella of artificial intelligence which highlights the use of data and algorithms to imitate the human thinking process aiming to reach the general artificial intelligence. It is the analysis process of datasets, and its efficiency depends on the quality of the provided data. Nowadays, adversaries have shown their unquestionable ability to perform adversarial attacks against the machine learning models through back doors. In this paper, taxonomy will be discussed with a highlight on the PGD attack technique with two pre-processor defence techniques which are Feature Squeezing and Spatial Smoothing. As we need to develop more robust defences against such vicious attacks.

1.2 Research Questions

Regarding machine learning security problems, we opt to elaborate on the answer to the following questions: what is the effect of adversarial machine learning? What are the capabilities of an evasion attack? What impact will an evasion attack make on the performance of machine learning? What are the defence strategies against this type of attack?

1.3 Description of the Research Problem and Research Issues

Adversarial machine learning is the study of effective methods to fool the machine learning model with such deceptive inputs that are called adversary examples. Nowadays, the machine learning models can be easily tricked into attack algorithms. The more the machine learning field is growing, the more it is leveraged to be vulnerable to such security attacks.

1.4 Goals of this Study

The goal of this paper is to examine the ability of adversarial attacks on machine learning models and encounter them with such powerful defences to overcome the vulnerability of these AI systems.

1.4.1 Goal of the Literature Review

The objective of this literature review is to implement a walkthrough for adversary machine learning taxonomy with a highlight on the infamous attack technique; projected gradient descent (PGD) vs an effective defence strategy; the pre-processor defence by using both feature squeezing (FS) and spatial smoothing (SS) to tackle their potentials that could penetrate the machine learning models attacks in adaptive systems.

1.4.2 Goal of the Experimental Evaluation

The goal of this experimental evaluation is to project a solution approach for such an adversary attack. Using the adversarial robustness toolbox (ART) to perform an evasion attack on a machine learning model to deceive it, resulting undesirable output. Hence, the pre-processor defence techniques will be implemented in the adversarial input to smoothen pixel noise perturbations to make it look more like a clean image, which improves the model prediction ability to classify the input correctly.

Chapter 2:

Literature Review

2.1 Taxonomy

Adversarial machine learning provides a broad array of attack and defence mechanisms. First, whether the adversary will use a white-box, gray-box, or black-box attack, the attack techniques are based on knowledge-based approaches. Regarding the techniques, a decision must be made as to whether they are implemented during the model's training phase using the poisoning approach or during the testing phase using the evasion, extraction, or inference methods of attack. Second, there are two types of counterattack approaches: those used during training (such as data encryption, data sanitization, and robust statistics) and those used during testing (such as the adversarial training technique or gradient masking).

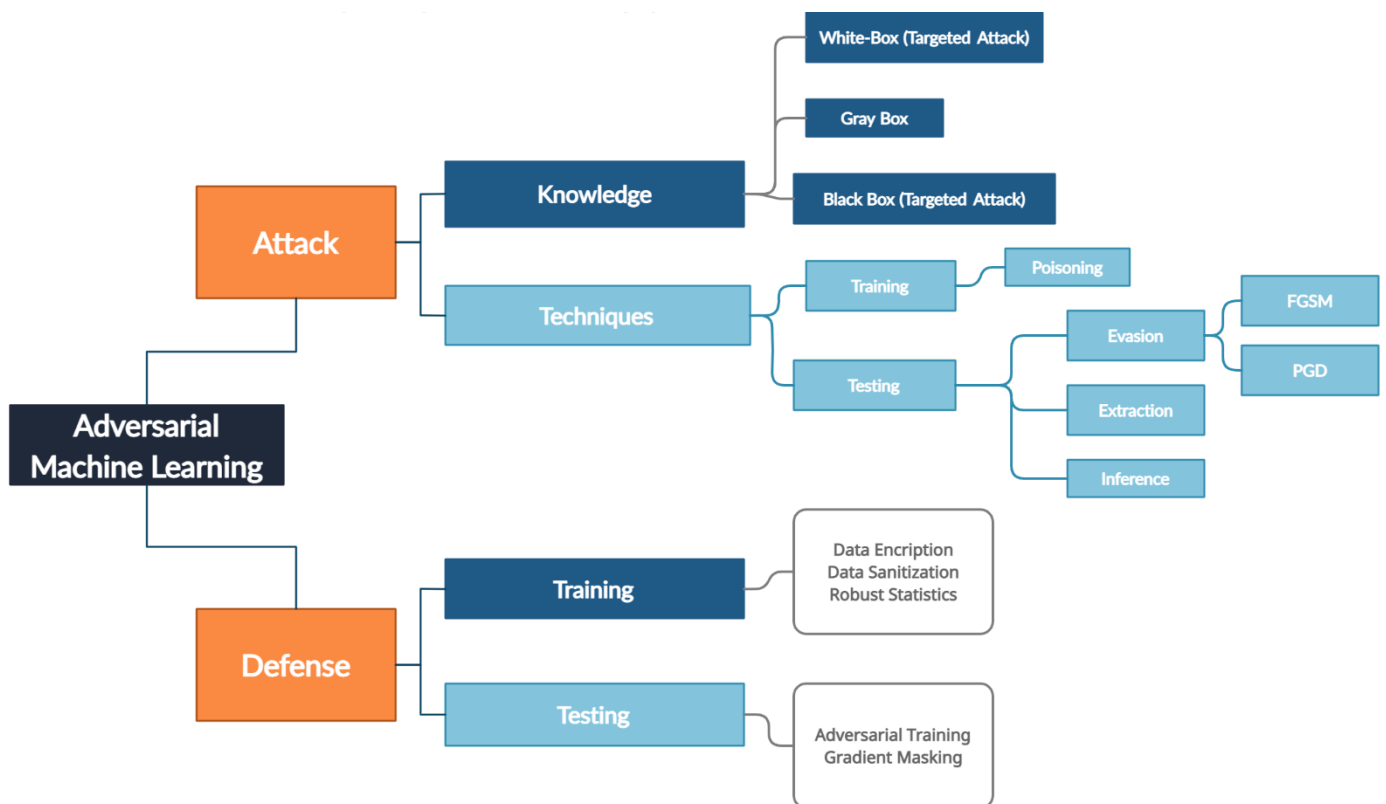


Figure (1) Taxonomy of Adversarial Machine Learning

2.2 Technical detail

2.2.1 Systematic Review of the Existing Approaches

2.2.2 Elaboration of the Major Approaches of Perspectives

Adversary Machine Learning: It is the study of machine learning attack & defence techniques. Hence, an adversarial attack is a method in which malicious data is injected into machine learning models to misguide their algorithms leading to bad undesirable output.

White-Box Attack: It is a targeted attack in which the adversary sample obtains the machine learning models architecture, parameters, and source code. The white box attack is the most time-consuming approach as it takes a massive amount of data to identify potential points of vulnerability.

Black-Box Attack: It is an untargeted attack in which the adversary sample has no access to the machine learning model's architecture and parameters. The drawback of this approach is that it totally depends on the hacker's ability to indicate the model vulnerabilities and backdoors.

Gray-Box Attack: It is an attack framework that operates by training a generative adversarial network (GAN), targeting the model of interest. The adversary can generate adversarial samples by using GANs. The advantage of the GAN-based attack is that it accelerates the run time for generating adversary samples and makes them appear cleaner. As it is an elevated level from the black box attack approach. Thus, it is a more focused assessment of the system parameters than the black box approach.

Targeted Attack: As shown in Figure (2), It optimizes samples to be misclassified as a predetermined and different target.

Untargeted Attack: As shown in Figure (2), It is the attack where the adversary tries to manipulate the machine learning model to predict any of the incorrect classes by maximizing the loss.



Figure (2): Targeted Attack & Untargeted Attack [14]

Data Poisoning: It is a type of adversarial attack, in which hackers inject malicious data into ML systems that make the learning models obsolete. As it is an innovative approach for attacking technology systems instead of making use of machine learning model's source code vulnerability, they manipulate training datasets algorithms to produce unwanted and harmful outcomes.

Evasion attack: It usually happens with trained machine learning systems using adversarial examples during the inference phase, by injecting these examples as a bad input, leading to misclassifying the data which attempts evasion attacks normally search for a backdoor in a model parameter to gain access to the algorithms and codes that forms that system. It is affiliated to poisoning attacks, but the primary difference is that evasion attacks attempt to exploit model vulnerabilities during the inference phase rather than the training process.

Fast Gradient Sign Method (FGSM): As shown in Figure (3), it is a typical adversarial attack that performs a single step of gradient ascent algorithm to increase the loss $J(\theta, (\mathcal{X}, \mathcal{Y}))$ as \mathcal{X} is the raw sample and \mathcal{Y} is the adversarial example (3) to the steepest position which makes it venerable to defence techniques, but still a time-efficient approach and it is formulated as follows:

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

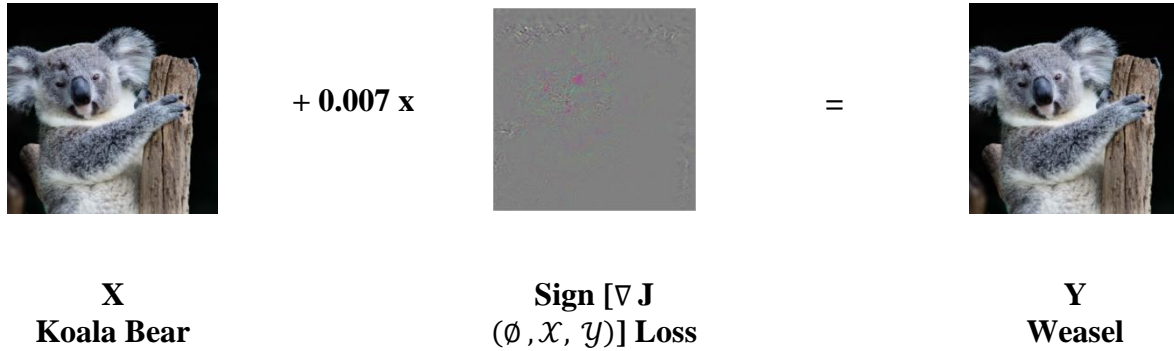


Figure (3): Fast Gradient Sign Method (FGSM)

Projected gradient descent (PGD): It is an iterative extension of FGSM attack method but stronger as it uses multiple gradient descent layers. Therefore, it is one of the strongest existing attacks and it is formulated as follows:

$$x'_{t+1} = \text{Proj} \{x'_t + \alpha \cdot \text{sign}[\nabla_x J(\theta, x'_t, y)]\}$$

Extraction Attack: It is an adversary that seeks to build a model that is similar or identical to the attacked model. Briefly, extraction is the attempt to copy or steal a machine learning model. This kind of attack typically requires access to the original model along with data that is similar or identical to the raw data required to accurately train the targeted model.

Inference Attack: It attempts to replicate a portion or the entire dataset that was employed to train the targeted model. Inference attacks can be implemented by adversaries to generate entire training samples. Access to the targeted model parameters is typically required for inference attacks. In certain circumstances, attackers may also require access to some of the data used to train the model, not just recreating a set of the whole dataset.

Data Encryption: It is a defence technique that is usually used during the training phase of the machine learning model that is proposed to protect the confidentiality of the training data.

Data Sanitization: It is the process of irreversibly removing malicious data during the training phase of the machine learning model.

Robust Statistics: It uses restrictions and probability distribution techniques to minimize potential distortions generated by poisoned data injected in the targeted model.

Adversarial Training: It is a machine learning technique in which the inputs including adversarial perturbations are injected into the training data alongside its correct labels to reduce classification errors caused by adversarial examples.

Gradient Masking: It reduces the model's vulnerability to adversary perturbations in an input and considering it as a variable. In addition, it computes first order derivatives of the model with respect to its input as the first variable and minimizing these derivatives throughout the training phase.

The pre-processor defence technique: It applies a form of pre-processing to adversarial input to smoothen noise perturbation. This makes the adversarial example looks more like a clean sample, which can improve model performance. ART's pre-processing defences may work for a wide range of attacks as well.

Feature Squeezing: It is a pre-processor defence technique that uses deep neural networks (DNN) in which the pixels of the adversary sample are reduced by merging samples that match to distinct vectors in the original uncompressed imagery range into a single sample by decreasing the pixel area of the adversary. A Deep Neural Network (DNN) model's prediction on the raw input is compared to its prediction on the compressed input to draw conclusions [2].

Spatial Smoothing: It is often referred to as a blurring technique used in picture processing to reduce adversary noise by replacing a pixel with a median of its neighbours. It basically has two primary types: local smoothing and non-local smoothing. From one hand, local smoothing exploits the surrounding pixels to smooth each one by choosing several methods of weighing the nearby pixels instead of merely affecting nearby pixels. On the other hand, when compared to local smoothing, non-local smoothing smoothens nearby pixels across wider range. It takes the average of several neighbouring patches that are similar to an adversarial patch and places it across a significant section of the image [2].

2.3 Critical Discussion

2.3.1 Strengths and Limitations of the Existing Approaches

Adversarial machine learning is a new paradigm that revolves around major prospects of machine learning and artificial intelligence security issues. In the present time, we behold several brutal attack approaches that our defences either they cannot stand against or have limited power against these attacks. For instance, the Projected gradient descent (PGD) stands as a perfect example for such vicious attacks as our current defences may have limited capacity to overpower it. From one hand, the FGSM represents an effective attack technique, but from the other hand, the PGD is significantly powerful as it attacks through penetrating multi-layer technique unlike the FGSM which depends on a one-layer technique.

2.3.2 Comparisons between the Major Approaches

The process of attack and defence in machine learning is reliable on the adversary's knowledge of the targeted model. In a white box attack, the adversary has thorough knowledge of the model's architecture, parameters, and data. In grey box attacks, the adversary has only minimal understanding of the model. While in black box attacks, the adversary has no knowledge of the model infrastructure [5].

In reference to adversarial techniques that are used for conducting attacks against models can target the training or testing phases of the machine learning model process. Whilst the training phase, attacks attempt to gain the training raw data or the model architecture and parameters. Regarding the poisoning attack, the attacker tends to inject the model with malicious data to exploit it as a backdoor invasion, leading to incorrect prediction labelling. Attacks in the testing phase, the adversary does not interfere with the model or the training data, it generates adversarial examples instead to fool the model with it leading to unrecognizable malicious data by the model. These adversarial samples are created through attacks such as evasion to modify the input for the sake to influence the targeted model, extraction, or Inference attacks by obtaining confidential data [5].

Regarding the effectiveness of defences, they can be categorized by the time the attack hit the target model during the inference phase of the machine learning operating system. Defences against the training stage attacks such as data encryption, data sanitization and robust statistics. These defences are applied throughout the poisoning attack approach rather than attempting to identify malicious input, robust statistics uses restrictions to limit any abnormalities in the learning model caused by injecting poisoned data. However, defences against the testing phase kind of attacks listed under numerous robustness enhancement methods such as adversarial training, and gradient masking [5].

Chapter 3:

Software Overview

3.1 An Overview on the Relevant Software Frameworks

Adversarial Robustness Toolbox (ART) is an open-source python library created by IBM Research division that used for machine learning security. It is defined as “a tool that enables developers and researchers to evaluate, defend, and verify Machine Learning models and applications against adversarial threats.” [6] It is built to optimize machine learning system's performance against adversarially manipulated data. It is utilized to evaluate the machine learning models' prediction labels and confidence rates on data classification attributes.

3.2 Comparisons between Different Software Frameworks

Adversarial Robustness Toolbox (ART): is a public-source python library that is built by IBM Research division that used for machine learning security. It is built to deploy machine learning systems that are immune to adversarial attacks. Its defence and attack modules are utilized to assess the machine learning models performance on data classification attributes [6].

DeepRobust: DeepRobust is a PyTorch adversarial learning library that aims to create a comprehensive and user-friendly platform to support this research field. Under a diverse array of deep learning architectures, it features over 10 attack and 8 defence techniques in the imagery domain. Additionally, it contains 9 attack and 4 defence algorithms in the graph domain [3].

Foolbox: It is a python library that operates adversarial attacks against machine learning models. It works primarily with TensorFlow, PyTorch, and JAX. It compares the robustness of the machine learning models. It also provides around fifteen attack methods. Furthermore, its implementation of adversarial attacks forms an internal hyperparameter to perform the minimum adversarial penetration. This tool structure consists of five components which are model, criterion, distance measure, attack algorithm, and adversarial perturbations [4].

Comparisons between Robust Software Frameworks			
#	ART	DeepRobust	Foolbox
All tools fall under the MIT License			
Features	<ul style="list-style-type: none"> - It assesses machine learning models against adversarial attacks of evasion, poisoning, extraction, and inference threats. - It serves 39 attack modules, and 29 defence modules. 	<ul style="list-style-type: none"> - It performs attack and defence methods on images and graphs. - It utilizes around 10 attack techniques, and 8 defence modules. 	<ul style="list-style-type: none"> - It creates adversarial perturbations to the raw input. - It quantifies and compares the robustness of machine learning models.
Models	TensorFlow, PyTorch, Theano, Lasagne, Keras, MXNet	PyTorch	PyTorch TensorFlow JAX
Source Code	https://rb.gy/bmkmka	https://rb.gy/dq9zlp	https://rb.gy/ud41ps
Environment Requirements and Setup	Python ≥ 3.6	Python ≥ 3.6 and Pytorch $\geq 1.2.0$	Python ≥ 3.8
Quick Start	pip install adversarial-robustness-toolbox	pip install deeprobust==0.2.6	pip install foolbox
Published By	Trusted-AI	DSE-MSU	Bethge Lab

Table (1): Comparison of Software Tools

Chapter 4:

Practical Evaluation


4.1 Software Toolkit Selected for Practical Evaluation

Adversarial Robustness Toolbox (ART) is a public-domain and python library that is developed by IBM Research division that is used for machine learning security. It is built to deploy machine learning systems that are immune to adversarial attacks. ART is used to assess the machine learning models' performance on data classification attributes [6]. It defends and evaluates machine learning models against hostile environment of adversarial examples created by evasion, poisoning, extraction, and inference attacks. It offers 39 offensive modules and 29 defensive modules. It also supports libraries, i.e. TensorFlow, Keras, Scikit-learn, PyTorch, MXNet, XGBoost, and GPy.

4.2 Setup of the Software

4.2.1 Installation of the Software

This experiment is implemented on the cloud using Google Collab. Thus, the ART framework is set up using the pip install feature as follows:



```
# pip install prerequisites for ART  
  
!pip install adversarial-robustness-toolbox
```

Install ART Perquisites

University for the Common Good

The installed ART version is **1.12.2** serves variety of machine learning frameworks such as PyTorch, scikit-learn, TensorFlow, Keras, MXNet and all data types (images, videos, audio, tables). It also serves machine learning tasks like classification, speech recognition, and object detection. Once installed, we load dependencies as follows:

```
# Load dependencies

# Load ART dependencies
from art.estimators.classification import KerasClassifier
from art.attacks.evasion import ProjectedGradientDescent
from art.defences.preprocessor import FeatureSqueezing
from art.defences.preprocessor import SpatialSmoothing
from art.utils import to_categorical

# Disable TensorFlow eager execution
import tensorflow as tf
if tf.executing_eagerly():
    tf.compat.v1.disable_eager_execution()

# Load Keras classifier dependencies
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.preprocessing import image

# Basic dependencies
%matplotlib inline
import matplotlib.pyplot as plt
import sys
import numpy as np
import warnings
warnings.filterwarnings('ignore')

# Install ImageNet stubs:
!{sys.executable} -m pip install git+https://github.com/nottombrown/imagenet_stubs
import imagenet_stubs
from imagenet_stubs.imagenet_2012_labels import name_to_label, label_to_name
```

Import Dependencies

4.2.2 Validation

The following screenshots validate the *pip install* and loading dependencies process. Hence, the ART tool output is “Successfully installed adversarial-robustness-toolbox-1.12.2”. Moreover, all the required dependencies such as TensorFlow, Keras as a classifier, matplotlib, ART, and the ImageNet selected package are imported successfully as well.


```
[ ] pip install adversarial-robustness-toolbox

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting adversarial-robustness-toolbox
  Downloading adversarial_robustness_toolbox-1.12.2-py3-none-any.whl (1.4 MB)
    1.4 MB 5.4 MB/s
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from adversarial-robustness-toolbox) (1.15.0)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.7/dist-packages (from adversarial-robustness-toolbox) (1.21.6)
Requirement already satisfied: scikit-learn<1.1.0,>=0.22.2 in /usr/local/lib/python3.7/dist-packages (from adversarial-robustness-toolbox) (1.0.2)
Requirement already satisfied: scipy=1.4.1 in /usr/local/lib/python3.7/dist-packages (from adversarial-robustness-toolbox) (1.7.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from adversarial-robustness-toolbox) (57.4.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from adversarial-robustness-toolbox) (4.64.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn<1.1.0,>=0.22.2->adversarial-robustness-toolbox) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn<1.1.0,>=0.22.2->adversarial-robustness-toolbox) (1.2.0)
Installing collected packages: adversarial-robustness-toolbox
Successfully installed adversarial-robustness-toolbox-1.12.2
```

ART Installed Perquisites

```
# Load dependencies

# Load ART dependencies
from art.estimators.classification import KerasClassifier
from art.attacks.evasion import ProjectedGradientDescent
from art.defences.preprocessor import FeatureSqueezing
from art.defences.preprocessor import SpatialSmoothing
from art.utils import to_categorical

# Disable TensorFlow eager execution
import tensorflow as tf
if tf.executing_eagerly():
    tf.compat.v1.disable_eager_execution()

# Load Keras classifier dependencies
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.preprocessing import image

# Basic dependencies
%matplotlib inline
import matplotlib.pyplot as plt
import sys
import numpy as np
import warnings
warnings.filterwarnings('ignore')

# Install ImageNet stubs:
!{sys.executable} -m pip install git+https://github.com/nottombrown/imagenet_stubs
import imagenet_stubs
from imagenet_stubs.imagenet_2012_labels import name_to_label, label_to_name

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/nottombrown/imagenet_stubs
  Cloning https://github.com/nottombrown/imagenet_stubs to /tmp/pip-req-build-lzcm5ftu
  Running command git clone -q https://github.com/nottombrown/imagenet_stubs /tmp/pip-req-build-lzcm5ftu
Building wheels for collected packages: imagenet-stubs
  Building wheel for imagenet-stubs (setup.py) ... done
  Created wheel for imagenet-stubs: filename=imagenet_stubs-0.0.7-py3-none-any.whl size=794840 sha256=b1f7a943d25e56ed1ed30e4eab360d
  Stored in directory: /tmp/pip-ephem-wheel-cache-q_bqbm5r/wheels/33/0f/5a/c83688c23a05eb9e88527a8944da56dbe007c86f534b0c1dad
Successfully built imagenet-stubs
Installing collected packages: imagenet-stubs
Successfully installed imagenet-stubs-0.0.7
```

Imported Dependencies

4.2.3 Dataset

The used dataset for this experiment is the **ImageNet** dataset which contains **14,197,122** raw images. It is regarded as a benchmark in image label classification. Accordingly, the selected

University for the Common Good

package for this examination is **ImageNet-stubs** which can be found through this link: <https://github.com/nottombrown/imagenet-stubs>. It consists of 16 raw data of images that can be used to generate adversary examples [14].

4.3 Experimental Scenarios and Results

4.3.1 Experimental Scenarios

ART utilizes tools to assess and validate machine learning models against adversarial threats using a plethora of attack and defence mechanisms. In this section, I will evaluate the strength of the machine learning model against the most vicious attack there is which is the Projected gradient descent (PGD) in counter to two pre-processor defence techniques which are the Feature Squeezing and Spatial Smoothing. By running these two defences on the created adversarial sample, both the prediction label and confidence rate will be evaluated to determine which defence algorithm is most efficient to stand against a such brutal attack.


4.3.2 Performance Metrics for Evaluation

To obtain the experiment performance metrics, the code runs as follows; after loading the dependencies, the next step is loading the raw data image with dimensions of 224 x 224 pixels. It is loaded and projected as follows:

```
[3] # Load raw image package
images_list = list()
for i, image_path in enumerate(imagenet_stubs.get_image_paths()):
    im = image.load_img(image_path, target_size=(224, 224))
    im = image.img_to_array(im)
    images_list.append(im)
    if 'koala.jpg' in image_path:
        # get koala index
        koala_idx = i
images = np.array(images_list)

print('The imported image's dimensions are', images.shape[1], 'x', images.shape[2], 'pixels')

# Show raw image
idx = koala_idx
plt.figure(figsize=(5,5)); plt.imshow(images[idx] / 255); plt.axis('off'); plt.show()
```



Raw Image

Then, run it through the ResNet50 image classifier to allow the model to determine the correct predicted label and confidence rate in both the raw model and the API classifier.

```
# Load ResNet50 image classifier
model = ResNet50(weights='imagenet')

# Load the input to the classifier
x = np.expand_dims(images[idx].copy(), axis=0)
x = preprocess_input(x)

# Run the model to determine the predicted label and confidence rate
pred = model.predict(x)
label = np.argmax(pred, axis=-1)[0]
confidence = pred[:,label][0]

print('Prediction:', label_to_name(label), '- confidence {0:.2f}'.format(confidence))
```

WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/keras/layers/normalization/batch_normalization.py:514: _colocate_with (from Instructions for updating: Colocations handled automatically by placer.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102967424/102967424 [=====] - 1s 0us/step
Prediction: koala, koala bear, kangaroo bear, native bear, Phascolarctos cinereus - confidence 1.00

Raw Model Prediction

```
[7] # Processing the prediction label & the confidence rate through the classifier API
x_art = np.expand_dims(images[idx], axis=0)
pred = classifier.predict(x_art)
label = np.argmax(pred, axis=1)[0]
confidence = pred[:,label][0]

print('Prediction:', label_to_name(label), '- confidence {0:.2f}'.format(confidence))

Prediction: koala, koala bear, kangaroo bear, native bear, Phascolarctos cinereus - confidence 1.00
```

Keras Model Classifier API

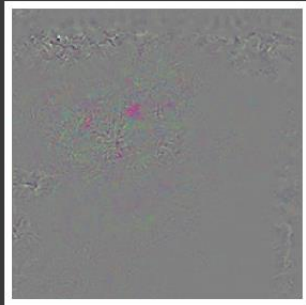
At the same time, I will plot the loss ingredient to add noise to the raw image, generating the adversary sample using the Projected Gradient Descent (PGD) untargeted attack technique.

```
[8] #Generating the loss gradient
loss_gradient = classifier.loss_gradient(x=x_art, y=to_categorical([label], nb_classes=1000))

# Apply the loss gradient
loss_gradient_plot = loss_gradient[0]

# Normalize loss gradient values to be in range of [0,1]
loss_gradient_min = np.min(loss_gradient)
loss_gradient_max = np.max(loss_gradient)
loss_gradient_plot = (loss_gradient_plot - loss_gradient_min)/(loss_gradient_max - loss_gradient_min)

# Show the loss gradient
plt.figure(figsize=(5,5)); plt.imshow(loss_gradient_plot); plt.axis('off'); plt.show()
```




Generating the Loss Gradient

As a result, the attack succeeded to deceive the model by labelling the koala bear as a weasel with a confidence rate 100%.

```
[9] # Create the adversarial example using the Projected Gradient Descent (PGD) untargeted attack technique
advx = ProjectedGradientDescent(classifier, targeted=False, max_iter=10, eps_step=1, eps=8)
x_art_adv = advx.generate(x_art)
plt.figure(figsize=(5,5)); plt.imshow(x_art_adv[0] / 255); plt.axis('off'); plt.show()

# Apply the image classifier to the adversarial generated sample
pred_adv = classifier.predict(x_art_adv)
label_adv = np.argmax(pred_adv, axis=1)[0]
confidence_adv = pred_adv[:, label_adv][0]
print('Prediction:', label_to_name(label_adv), '- confidence {:.2f}'.format(confidence_adv))
```

PGD - Random Initializations: 100% 1/1 [00:04<00:00, 4.53s/it]



Prediction: weasel - confidence 1.00

The Adversarial Sample

Along with the mentioned attack results, the first defence trail will be applied which is the Feature Squeezing technique. The classifier prediction labelling and confidence rate generator is initialized, as a result, this defence technique is failed to penetrate through the adversarial attack. The prediction label is still a weasel with a confidence rate 100%.


```
[10] # Apply the first defence approach by using the Feature Squeezing technique
fs = FeatureSqueezing(clip_values=(0, 255))
x_art_def, _ = fs(x_art)
x_art_adv_def, _ = fs(x_art_adv)

# Initialize the classifier prediction label & confidence rate
pred_def = classifier.predict(x_art_def)
label_def = np.argmax(pred_def, axis=1)[0]
confidence_def = pred_def[:, label_def][0]
pred_adv_def = classifier.predict(x_art_adv_def)
label_adv_def = np.argmax(pred_adv_def, axis=1)[0]
confidence_adv_def = pred_adv_def[:, label_adv_def][0]

print('Prediction of original sample:', label_to_name(label_def), '- confidence {:.2f}'.format(confidence_def))
print('Prediction of adversarial sample:', label_to_name(label_adv_def),
      '- confidence {:.2f}'.format(confidence_adv_def))

# Show the adversarial example after applying the Spatial Smoothing defence technique
plt.figure(figsize=(5,5)); plt.imshow(x_art_adv_def[0] / 255); plt.axis('off'); plt.show()

Prediction of original sample: koala, koala bear, kangaroo bear, native bear, Phascolarctos cinereus - confidence 1.00
Prediction of adversarial sample: weasel - confidence 1.00
```



Feature Squeezing Defence Technique

Regarding the previous disappointing results, another pre-processor defence mechanism will be applied by using the spatial smoothing technique. The classifier prediction labelling and confidence rate generator is reinitialized. In contrast of the former trail, this defence technique has proved its effectiveness against such ruthless. The prediction label is reclassified to the original label as a koala bear instead of a weasel with confidence rate 100% and as a weasel 76% instead of 100%.



```
# Apply the second defence approach by using the Spatial Smoothing technique
ss = SpatialSmoothing(window_size=3)
x_art_def, _ = ss(x_art)
x_art_adv_def, _ = ss(x_art_adv)

# Initialize the classifier prediction label & confidence rate
pred_def = classifier.predict(x_art_def)
label_def = np.argmax(pred_def, axis=1)[0]
confidence_def = pred_def[:, label_def][0]
pred_adv_def = classifier.predict(x_art_adv_def)
label_adv_def = np.argmax(pred_adv_def, axis=1)[0]
confidence_adv_def = pred_adv_def[:, label_adv_def][0]

print('Prediction of original sample:', label_to_name(label_def), '- confidence {0:.2f}'.format(confidence_def))
print('Prediction of adversarial sample:', label_to_name(label_adv_def),
      '- confidence {0:.2f}'.format(confidence_adv_def))

# Show the adversarial example after applying the Spatial Smoothing defence technique
plt.figure(figsize=(5,5)); plt.imshow(x_art_adv_def[0] / 255); plt.axis('off'); plt.show()
```

☐ Prediction of original sample: koala, koala bear, kangaroo bear, native bear, Phascolarctos cinereus - confidence 1.00
 Prediction of adversarial sample: weasel - confidence 0.76



Spatial Smoothing Defence Technique

4.3.3 Comparisons and Results of Running the Experiment

In this experiment, I attempted to show the impact of adversarial machine learning on the AI field. It has the potentials to seriously jeopardise the upcoming uprising tech. evolution by generating adversarial samples that can deceive all kinds of detectors. As shown in the above outcome, no matter how many defences we create and how strong they can perform, adversarial attacks shape a huge threat to our data integrity. The above-mentioned results are a trail to find a defence technique against the most existing brutal adversary attack i.e., the Projected Gradient Descent (PGD). I have managed to crack the adversary example to be detected by confidence rate of 76%. Based on both the attack and defence mechanisms, the first defence mechanism which is the feature squeezing appears to fail to penetrate through the crafted adversarial example due to its strategy to work on a larger area of the adversary sample. At the same time, the PGD attack technique is to work on penetrating each pixel to create the maximum amount of noise. Alternatively, the spatial smoothing technique managed to penetrate the adversary sample on a pixel scope. In a nutshell, the technological community must optimise the robust procedures as much as possible because adversarial machine learning is advancing over time. For instance, the deepfake technology can be categorised as an adversary threat that has a negative impact on our adversarial detectors. It has been demonstrated that deepfake-generated objects can bypass such powerful detectors.

References:

- [1] Xu, H., Ma, Y., Liu, H.-C., Deb, D., Liu, H., Tang, J.-L. and Jain, A.K. (2020). Adversarial Attacks and Defenses in Images, Graphs and Text: A Review. *International Journal of Automation and Computing*, 17(2), pp.151–178. doi:10.1007/s11633-019-1211-x.
- [2] Xu, W., Evans, D. and Qi, Y. (2018). Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. *Proceedings 2018 Network and Distributed System Security Symposium*, (1-1891562-49-5). doi:10.14722/ndss.2018.23198.
- [3] Li, Y., Jin, W., Xu, H. and Tang, J. (2021). DeepRobust: a Platform for Adversarial Attacks and Defenses. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(18), pp.16078–16080. doi:10.1609/aaai.v35i18.18017.
- [4] Rauber, J., Zimmermann, R., Bethge, M. and Brendel, W. (2020). Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX. *Journal of Open-Source Software*, 5(53), p.2607. doi:10.21105/joss.02607.
- [5] Tabassi, E., Burns, K.J., Hadjimichael, M., Molina-Markham, A.D. and Sexton, J.T. (2019). A taxonomy and terminology of adversarial machine learning. *NISTIR*. doi:10.6028/nist.ir.8269-draft.
- [6] Adversarial Robustness Toolbox. (n.d.). *Home*. [online] Available at: <https://adversarial-robustness-toolbox.org/> [Accessed 25 Nov. 2022].
- [7] developer.ibm.com. (n.d.). *IBM Developer*. [online] Available at: <https://developer.ibm.com/articles/applying-the-adversarial-robustness-toolbox/> [Accessed 25 Nov. 2022].
- [8] IBM Research Blog. (2018). *Securing AI Against Adversarial Threats with Open-Source Toolbox*. [online] Available at: <https://www.ibm.com/blogs/research/2018/04/ai-adversarial-robustness-toolbox/> [Accessed 25 Nov. 2022].

[9] adversarial-robustness-toolbox.readthedocs.io. (n.d.). *Welcome to the Adversarial Robustness Toolbox — Adversarial Robustness Toolbox 1.7.0 documentation*. [online] Available at: <https://adversarial-robustness-toolbox.readthedocs.io/en/latest/>.

[10] [www.youtube.com](https://www.youtube.com/watch?v=wIX00bZ173k&t=565s). (n.d.). *Adversarial Robustness*. [online] Available at: <https://www.youtube.com/watch?v=wIX00bZ173k&t=565s> [Accessed 25 Nov. 2022].

[11] Kedion (2022). *ML Security with the Adversarial Robustness Toolbox*. [online] Medium. Available at: <https://kedion.medium.com/ml-security-with-the-adversarial-robustness-toolbox-d173b24e8c1a> [Accessed 25 Nov. 2022].

[12] Kedion (2022). *ML Security with the Adversarial Robustness Toolbox*. [online] Medium. Available at: <https://kedion.medium.com/ml-security-with-the-adversarial-robustness-toolbox-abe16f56e923> [Accessed 25 Nov. 2022].

[13] GitHub. (2021). *Trusted-AI/adversarial-robustness-toolbox*. [online] Available at: <https://github.com/Trusted-AI/adversarial-robustness-toolbox>.

[14] Brown, T.B. (2022). *ImageNet Stubs*. [online] GitHub. Available at: <https://github.com/nottombrown/imagenet-stubs> [Accessed 25 Nov. 2022].

[15] Research, I. (n.d.). *ART - IBM Research*. [online] art-demo.mybluemix.net. Available at: <https://art-demo.mybluemix.net/> [Accessed 25 Nov. 2022].

Appendices

Source Code:

```
# pip install prerequisites for ART
!pip install adversarial-robustness-toolbox

# Load dependencies

# Load ART dependencies
from art.estimators.classification import KerasClassifier
from art.attacks.evasion import ProjectedGradientDescent
from art.defences.preprocessor import FeatureSqueezing
from art.defences.preprocessor import SpatialSmoothing
from art.utils import to_categorical

# Disable TensorFlow eager execution
import tensorflow as tf
if tf.executing_eagerly():
    tf.compat.v1.disable_eager_execution()

# Load Keras classifier dependencies
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.preprocessing import image

# Basic dependencies
%matplotlib inline
import matplotlib.pyplot as plt
import sys
import numpy as np
import warnings
warnings.filterwarnings('ignore')

# Install ImageNet stubs:
!{sys.executable} -m pip install git+https://github.com/nottombrown/imagenet_stubs
import imagenet_stubs
from imagenet_stubs.imagenet_2012_labels import name_to_label, label_to_name
```

```
# Load raw image package
images_list = list()
for i, image_path in enumerate(imagenet_stubs.get_image_paths()):
    im = image.load_img(image_path, target_size=(224, 224))
    im = image.img_to_array(im)
    images_list.append(im)
    if 'koala.jpg' in image_path:
        #get koala index
        koala_idx = i
images = np.array(images_list)

print('The imported image\'s dimensions are', images.shape[1], 'x', images.shape[2], 'pixels')
# Show raw image
idx = koala_idx
plt.figure(figsize=(5,5)); plt.imshow(images[idx] / 255); plt.axis('off'); plt.show()
# Load ResNet50 image classifier
model = ResNet50(weights='imagenet')

# Load the input to the classifier
x = np.expand_dims(images[idx].copy(), axis=0)
x = preprocess_input(x)

# Run the model to determine the predicted label and confidence rate
pred = model.predict(x)
label = np.argmax(pred, axis=1)[0]
confidence = pred[:,label][0]

print('Prediction:', label_to_name(label), '- confidence {0:.2f}'.format(confidence))
```

```
# Enable Keras classifier model
from art.preprocessing.preprocessing import Preprocessor

class ResNet50Preprocessor(Preprocessor):

    def __call__(self, x, y=None):
        return preprocess_input(x.copy()), y

    def estimate_gradient(self, x, gradient):
        return gradient[..., :-1]

# Enable the preprocessor input
preprocessor = ResNet50Preprocessor()
classifier = KerasClassifier(clp_values=(0, 255), model=model, preprocessing=preprocessor)
# Processing the prediction label & the confidence rate through the classifier API
x_art = np.expand_dims(images[idx], axis=0)
pred = classifier.predict(x_art)
label = np.argmax(pred, axis=1)[0]
confidence = pred[:,label][0]

print('Prediction:', label_to_name(label), '- confidence {0:.2f}'.format(confidence))
#Generating the loss gradient
loss_gradient = classifier.loss_gradient(x=x_art, y=to_categorical([label], nb_classes=1000))

# Apply the loss gradient
loss_gradient_plot = loss_gradient[0]

# Normalize loss gradient values to be in range of [0,1]
loss_gradient_min = np.min(loss_gradient)
loss_gradient_max = np.max(loss_gradient)
loss_gradient_plot = (loss_gradient_plot - loss_gradient_min)/(loss_gradient_max - loss_gradient_min)

# Show the loss gradient
plt.figure(figsize=(5,5)); plt.imshow(loss_gradient_plot); plt.axis('off'); plt.show()
```

```
# Create the adversarial example using the Projected Gradient Descent (PGD) untargeted attack technique
advx = ProjectedGradientDescent(classifier, targeted=False, max_iter=10, eps_step=1, eps=0)
x_art_adv = advx.generate(x_art)
plt.figure(figsize=(5,5)); plt.imshow(x_art_adv[0] / 255); plt.axis('off'); plt.show()

# Apply the image classifier to the adversarial generated sample
pred_adv = classifier.predict(x_art_adv)
label_adv = np.argmax(pred_adv, axis=1)[0]
confidence_adv = pred_adv[:, label_adv][0]
print('Prediction:', label_to_name(label_adv), '- confidence {0:.2f}'.format(confidence_adv))

# Apply the first defence approach by using the Feature Squeezing technique
fs = FeatureSqueezing(clip_values=(0, 255))
x_art_def, _ = fs(x_art)
x_art_adv_def, _ = fs(x_art_adv)

# Initialize the classifier prediction label & confidence rate
pred_def = classifier.predict(x_art_def)
label_def = np.argmax(pred_def, axis=1)[0]
confidence_def = pred_def[:, label_def][0]
pred_adv_def = classifier.predict(x_art_adv_def)
label_adv_def = np.argmax(pred_adv_def, axis=1)[0]
confidence_adv_def = pred_adv_def[:, label_adv_def][0]

print('Prediction of original sample:', label_to_name(label_def), '- confidence {0:.2f}'.format(confidence_def))
print('Prediction of adversarial sample:', label_to_name(label_adv_def), '- confidence {0:.2f}'.format(confidence_adv_def))

# Show the adversarial example after applying the Feature Squeezing defence technique
plt.figure(figsize=(5,5)); plt.imshow(x_art_adv_def[0] / 255); plt.axis('off'); plt.show()

# Apply the second defence approach by using the Spatial Smoothing technique
ss = SpatialSmoothing(window_size=3)
x_art_def, _ = ss(x_art)
x_art_adv_def, _ = ss(x_art_adv)

# Initialize the classifier prediction label & confidence rate
pred_def = classifier.predict(x_art_def)
label_def = np.argmax(pred_def, axis=1)[0]
confidence_def = pred_def[:, label_def][0]
pred_adv_def = classifier.predict(x_art_adv_def)
label_adv_def = np.argmax(pred_adv_def, axis=1)[0]
confidence_adv_def = pred_adv_def[:, label_adv_def][0]

print('Prediction of original sample:', label_to_name(label_def), '- confidence {0:.2f}'.format(confidence_def))
print('Prediction of adversarial sample:', label_to_name(label_adv_def), '- confidence {0:.2f}'.format(confidence_adv_def))

# Show the adversarial example after applying the Spatial Smoothing defence technique
plt.figure(figsize=(5,5)); plt.imshow(x_art_adv_def[0] / 255); plt.axis('off'); plt.show()
```