Global Scientific JOURNALS

GSJ: Volume 11, Issue 4, April 2023, Online: ISSN 2320-9186
www.globalscientificjournal.com

CYBERSECURITY ISSUES IN AI/ML TOOLS

Zaman, Chongkon Haruna

D.Eng. Candidate,

Department of Electrical and Computer Engineering, Morgan State University, Baltimore, Maryland USA

Email address: chzam1@morgan.edu

© GSJ

Abstract

A new window of opportunity for cyber security threats has opened with the growth of the Internet and its application in our daily lives. Artificial intelligence (AI) and machine learning (ML)-based solutions have risen in response to the continual rise in cyberattacks. Also, deep neural networks (DNNs) are sensitive to adversarial attacks that cause misclassification. Malicious attack has been a critical factor in increasing the robustness of DNNs. This paper examines cybersecurity issues in AI/ML tools vulnerable to cybersecurity attacks. The paper also used counterfit to explore these attacks. The attacks performed to show a success rate of 100%, with the best score of 0.5, 0.6, and 1 with different attacks.

Keywords: *Artificial intelligence, machine learning, deep neural networks,* counterfit, *White-box attack, and Black box attacks*

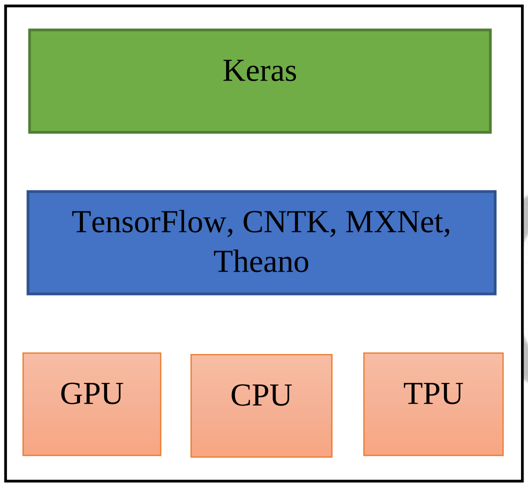
Introduction

Artificial intelligence (AI) and machine learning (ML) are two terminologies that have become widely used. The demand for AI and machine learning applications has fueled explosive growth in the GPU market. When it comes to new emerging technologies, it's critical to understand how they affect established development techniques. For example, deep neural networks (DNNs) have been widely used in image classification and natural language processing. However, DNNs are sensitive to adversarial attacks like examples and deliberately prepared perturbations. As a result of such attacks, insecure scenarios like self-driving cars may emerge. In addition, deep neural networks are vulnerable to adversarial examples due to their overfitting nature.

As a result of the continuous and extreme inclusion of the Internet, computer networks, and social life, there has been a complete transformation in how people learn and work. The expansion of the Internet and its applications in our lives creates an abysmal opportunity for cyber security attacks. The continuous increase in cyberattacks has given rise to artificial intelligence (AI) and machine learning (ML)-based techniques vital in detecting security risks, security breaches and alerts, progress triage events, and malware detection to defense issues. {ML, AI} is the set of statistical and mathematical forms to clarify more serious non-linearity troubles of different themes, such as data organization, prediction, and classification. Moreover, it is undeniable that information is an attractive and reasonable presence for every corporation and big business. Therefore, logically protecting security models driven by the actual data sets is necessary. Hence, this paper presents cybersecurity issues in AI/ML tools vulnerable to cybersecurity attacks. The article also used counterfit to examine these attacks, specifically on digits. Kera's model

An Overview of the Keras Model

Keras is an open-source software library for artificial neural networks with a Python interface. TensorFlow serves as a user interface through Keras. Keras is a neural network and machine learning library. Convolutional and recurrent neural networks are supported. It runs on both CPUs and GPUs and is written in Python. Keras also makes it easier to work with images and text data. This makes building Deep Neural Network (DNN) code even more straightforward. In the library, layers are connected like pieces of Lego, resulting in a clean and easy-to-understand model. The model's training is specific, requiring only data, several epochs of movement, and metrics to monitor. However, Keras is not an independent deep-learning library. It is built on top of another deep learning library or backend. This could be TensorFlow, Microsoft's CNTK, or MILA's Theano. Keras is a high-level model supported by GPU, CPU, and TPU, as shown in Figure 1. When utilizing Keras, you'll be working with models. The models specify TensorFlow neural networks' properties, functions, and layers.



GSJ

Figure 1: Keras is a high-level model compared to other deep-learning models.

You can utilize a variety of Keras APIs to define your neural network, including:

- The Sequential API allows you to build a model layer-by-layer for most issues. It's essential (simply a list of layers), but it's just for single-input, single-output layer stacks.
- **A full-featured API** that supports arbitrary model architectures. It is more complicated and flexible than the sequential API.
- **Model Subclassing** allows you to build anything from the ground up. It's suitable for research and sophisticated use cases but rarely used in practice.

Machine Learning (ML) models are vulnerable to various attack vectors that are uncommon or low-risk in other software types.

An adversarial attack often refers to different malicious activities on machine learning models. This attack depends on what part of the machine learning they are targeting and the type of activity they want. ML is divided into two phases.

Adversarial samples: These attacks are designed to produce data that an AI or ML model would classify differently than a human.

Counterfit is a command-line tool and automation layer for evaluating the security of machine-learning algorithms. It works to keep the target in focus for the user and provides a uniform interface from which to use the underlying frameworks.

A target is an eventual output you're attempting to anticipate. On the other hand, a user creates a class to interface between the attacks and a targeted model in a framework.

Frameworks It takes advantage of current adversarial machine learning frameworks to speed up the development of attack algorithms.

AL/ML attacks are either black-box attacks or white-box attacks.

© GSJ

Black box attacks can be used to attack adversarial machine learning. An attacker can only give data to the system and get a detailed result regarding a class. These attacks aim to provide malicious instances that can be applied to the black-box model. Attacks against black boxes can be carried out with or without scratch computer models.

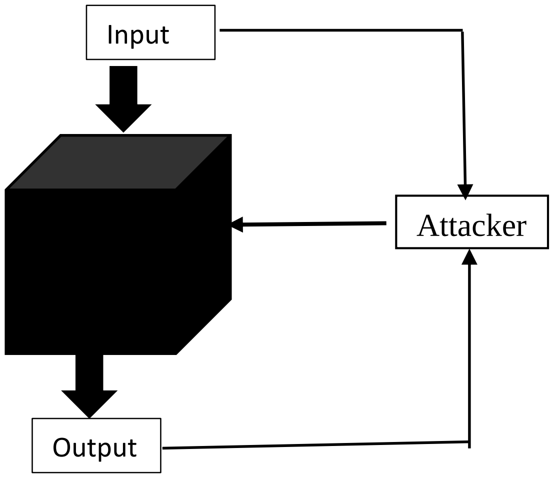


Figure 2: Black Box Attack

White-box attack In this type of attack, an attacker has access to all parameters like model algorithms' inputs, outputs, and intermediate calculations.

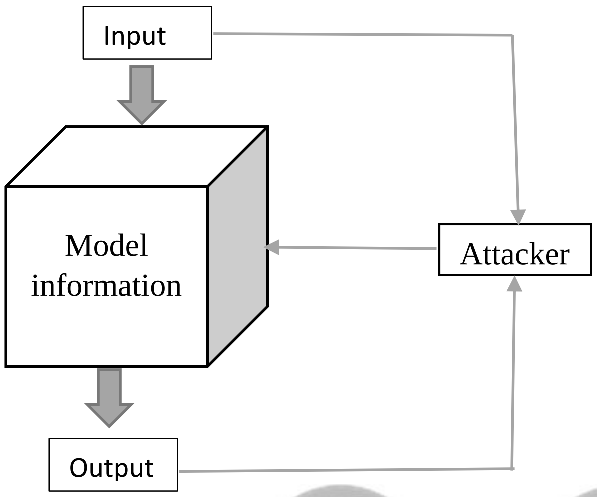


Figure 3: White box attack

© GSJ

3.0 Some of the attacks that were examined.

3.1 Wasserstein attacks

Such attacks calculate the cost of changing pixel mass in picture classification. They naturally encompass "typical" image operations, including scaling, rotation, translation, and distortion (and can potentially be applied to other settings).

3.1.1 Wasserstein distance

The Wasserstein distance is an ideal solution to the problem of calculating the cost of moving a mass from one location to another. When applied to images, this is the cost of transferring a pixel from one position to another, with the price increasing with pixel distance.

Assuming $x \in \mathbb{R}^n_+$, are two non-negative data points in such a way that $\sum_i x_i = \sum_j y_j = 1$, inputs need to be normalized and let to the negative matrix a cost be $C \in \mathbb{R}^{n \times n}$ and $\mathbf{C}_{\mathbf{i},\mathbf{j}}$ be the cost of moving from x_i to y_j, i.e., the pixel distance apart.

Therefore, the Wasserstein distance d_w (x, y) can be defined.

$$d_w = (x,) = \min_{\Pi \in \mathbb{R}n \times n} (\Pi, C) \quad \ldots (1)$$

For $\Pi 1 = x,\ \Pi^T 1 = y$

Where Π is the depreciation over a transport plan, whose inputs $\Pi_{i,j}$ shows how far mass moves from x_i to y_j.

3.2 Projected Gradient Descent (PGD) attack

The Projected Gradient Descent attack is an iterative method in which, after each iteration, the perturbation is projected onto an lp-ball of a specified radius (in addition to clipping the values of the adversarial sample so that it lies in the permitted data range). This is the attack proposed by Madry et al. for adversarial training.

$$\mathcal{B}(x, \varepsilon) = \{z \in X |\ x - z\ |_p \leq\}. \begin{cases} x' = x \\ x' = \Pi\,(x' = \psi(x', y)) \end{cases} \quad \ldots (2)$$

Where $\boldsymbol{\Pi_\varepsilon}$ = projection on the ball $\mathcal{B}$ (x, ε), **ψ** = perturbation function, y = target label yon wants to attack.

Gradient descent is a typical optimization algorithm for training machine learning models and neural networks. These models learn over time using training data and the cost function within gradient descent functions as a barometer, assessing its correctness with each iteration of parameter updates. Linear regression and logistic regression are two common examples of algorithms with coefficients that can be optimized using gradient descent.

3.3 Saliency Map Method attack

This method determines the spatial support for a specific class in each image. It is the earliest and most widely used explanation approach for evaluating convolutional neural network predictions.

3.3.1 How to Create Saliency Map

1. The basic properties of an image are extracted, such as color, orientation, and intensity from it.
2. To create a feature map, these processed images are used to develop Gaussian pyramids.
3. A saliency map is created by averaging all feature maps.

The salience bias refers to our inclination to pay greater attention to items or information that are more notable while dismissing those that are not.

3.3.2 Working of saliency maps method.

The gradients of the output over the input are used to create the saliency map**.** Saliency maps process images to differentiate visual features in images. For example, colored images are converted to black-and-white photos to analyze the richest colors.

Static saliency uses image characteristics and statistics to locate the image's regions of interest.

Motion saliency: Optical flow is used to detect motion in a video. Moving objects are deemed significant.

Abjectness: refers to the likelihood that an image window will cover an item. These techniques produce a collection of bounding boxes that indicate where an object in an image might be.

Saliency analysis determines how frequently any code appears, its importance, or both. High-value regulations increase understanding or are valuable in solving real-world issues. As a result, saliency analysis might reveal what is non-recurrent but potentially essential to a study's goals.

This approach uses the Jacobian matrix of outputs about inputs. Looking at this matrix, one can forecast how the output probabilities will behave if an input feature is significantly modified. Saliency maps were created to aid in visualizing the classification model prediction process. The map assigns a weight to each input feature x (i) (e.g., each pixel), causing the model to predict class c = y (x) = 1 major f (x) c 0), where f (x) is the victim model's SoftMax probability vector.

$$S^{+}(x_{(i),}\, c) = \begin{cases} 0\ If\ \frac{\partial f(x)}{\partial x} < 0\ or\ \sum_{c' \neq c} \frac{\partial f(x)}{\partial x} > 0 \\ -\frac{\partial f(x)}{\partial x} < 0\,.\sum_{c' \neq c} \frac{\partial f(x)}{\partial x} \quad \text{Otherwise} \end{cases} \quad \ldots (3)$$

© GSJ

S^{+} is the measure of how much x(i) positively correlates with c while also negatively correlates with all other classes $c^{'} = c$

$$S^{-}(x(i), t) = 0\ If\ \frac{\partial f(x)}{\partial x} > 0\ or\ \sum_{c' \neq c} \frac{\partial f(x)}{\partial x} < 0$$

Both saliencies S^{-} and S^{+} are too demanding when applied to individual input features because the cumulative gradient contribution across all nontargeted classes $\sum_{ca \neq c} \frac{\partial f(x)(c')}{\partial x(i)}$ will frequently trigger the minimal saliency requirement.

Results of the attacks on digits Keras model

The table below shows the different attacks and their success rate on digits Keras

Table 1: Attack test

Sample index	Input Label (conf)	Adversarial Label (conf)	% Euclid. dist	Adversarial Input	Success

0	Fraud (1.000)	benign (1.000)	0.0	[4462.00 -2.30 1.76 -0.36 2.33 … 0.04 -0.15 239.93]	True

Table 2: Summary of the different model attacks.

Attack names	Total runs	Successes (%)	Best score (attack_id)	Best parameters
saliency Map Method	1	5 (100.0%)	0.5	Theta: 0.1, gamma: 1, batch_size: 1, etc.
Projected Gradient Descent Common	1	5 (100.0%)	0.6	Targeted: false, Norm: infinity, esp_size: 0.1, etc.
Wasserstein	1	5 (100.0%)	1	Regularization: 3000.0, kernel_size: 5, etc.,

Labels are things that we predict or that a trained machine is intended to provide. The label instructs users on what value should be typed in the corresponding input field.

Adversarial label learning trains the classifier to execute as expected. When given noisy and possibly associated labels

Adversarial inputs are inputs to machine learning models that an attacker has explicitly created to force the model to make a wrong decision. A negative example would be a corrupted version of a legitimate input contaminated by adding a minor disturbance.

Discussion

This section discusses attacks performed and the result obtained by different attacks on digits keras. The results, summarized in tables one and two, show that the attacks were successful, with the best score more significant than one. It also shows that these attacks are robust to adversarial defense and can fool the model and make it vulnerable to the attacker. Wasserstein attack has the highest best score of 1 and 100% success, as shown in table1 and aims to determine the perturbation that maximizes a model's loss on a given input. It is commonly stated as the perturbing's L_2 or L_∞ norm, and it is added so that the content of the adversarial example is identical to the unperturbed sample.

© GSJ

For instance, Wasserstein's distance-bounded threat model limits disturbance to pixel-mass movements. The perturbations in the adversarial attack formed within a ball represent the image's actual content and structure. For example, in the top row, we can see a Wasserstein perturbation that hits all pixels indiscriminately rather than the space around. A standard model with binarization, a model provably resilient to perturbations of at most 0.1, and an adversarially trained model are all considered in MNIST and use only 100 iterations of projected gradient descent.

Conclusion

Artificial intelligence (AI) and machine learning (ML) are two terms that have gained popularity in recent years from the software engineering standpoint. However, despite their widespread use, the community has paid little attention to AI and machine learning tools and applications. This paper presents cybersecurity issues in AI/ML tools. The attacks performed to show a success rate of 100%, with the best score of 0.5, 0.6, and 1 with different attacks.

Acknowledgment

I want to thank God and Dr. Md Tanvir Arafin for his mentorship and advice.

© GSJ

Reference

[1] Atienza, Rowel. *Advanced Deep Learning with Keras: Apply deep learning techniques, autoencoders, GANs, variational autoencoders, deep reinforcement learning, policy gradients,* etc. Packt Publishing Ltd, 2018.

[2] Merino, Tim, et al. "Expansion of cyber-attack data from unbalanced datasets using generative adversarial networks." *International Conference on Software Engineering Research, Management, and Applications*. Springer, Cham, 2019.

[3] Li, Linyi, et al. "Sok: Certified robustness for deep neural networks." *arXiv preprint arXiv:2009.04131* (2020).

[4] Li, Jincheng, et al. "Internal Wasserstein distance for adversarial attack and defense." *arXiv preprint arXiv:2103.07598* (2021).

[5] Combey, Theo, et al. "Probabilistic Jacobian-Based Saliency Maps Attacks." *Machine Learning and Knowledge Extraction* 2.4 (2020): 558-578.

[6] Wiyatno, Rey, and Anqi Xu. "Maximal Jacobian-based saliency map attack." *arXiv preprint arXiv:1808.07945* (2018).

[7] Qureshi, Ayyaz Ul Haq, et al. "An adversarial approach for intrusion detection systems using Jacobian saliency map attacks (jsma) algorithm." *Computers* 9.3 (2020): 58.

[8] Wong, Eric, Frank Schmidt, and Zico Kolter. "Wasserstein adversarial examples via projected stinkhorn iterations." *International Conference on Machine Learning*. PMLR, 2019.

[9] Khedher, Hatem Ibn, Mohamed Ibn Khedher, and Makhlouf Hadji. "Mathematical Programming Approach for Adversarial Attack Modelling." *International Conference on Agents and Artificial Intelligence*. 2021.

[10] Manaswi, Navin Kumar. "Understanding and working with Keras." *Deep Learning with Applications Using Python*. Apress, Berkeley, CA, 2018. 31-43.

[11] Ketkar, Nikhil. "Introduction to Keras." *Deep learning with Python*. Apress, Berkeley, CA, 2017. 97-111.

[12] Vani, A. K., et al. "Using the Keras model for accurate and rapid gender identification through detection of facial features." *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE, 2020.

[13] Ott, Jordan, et al. "A Fortran-Keras deep learning bridge for scientific computing." *Scientific Programming* 2020 (2020).

[14] Bhagwat, Ritesh, Mahla Abdolahnejad, and Matthew Moocarme. *Applied deep learning with Keras: Solve complex real-life problems with the simplicity of Keras*. Packt Publishing Ltd, 2019.

[15] Wang, Run, et al. "Amora: Black-box adversarial morphing attack." *Proceedings of the 28th ACM International Conference on Multimedia*. 2020.

[16] Atienza, Rowel. *Advanced Deep Learning with TensorFlow 2 and Keras: Apply DL, GANs, VAEs, deep RL, unsupervised learning, object detection, segmentation,* etc. Packt Publishing Ltd, 2020.

[17] Lanfredi, Ricardo Bigolin, Joyce D. Schroeder, and Tolga Tasdizen. "Quantifying the preferential direction of the model gradient in adversarial training with projected gradient descent." *arXiv preprint arXiv:2009.04709* (2020).

[18] Croce, Francesco, and Matthias Hein. "Sparse and imperceivable adversarial attacks." *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019.

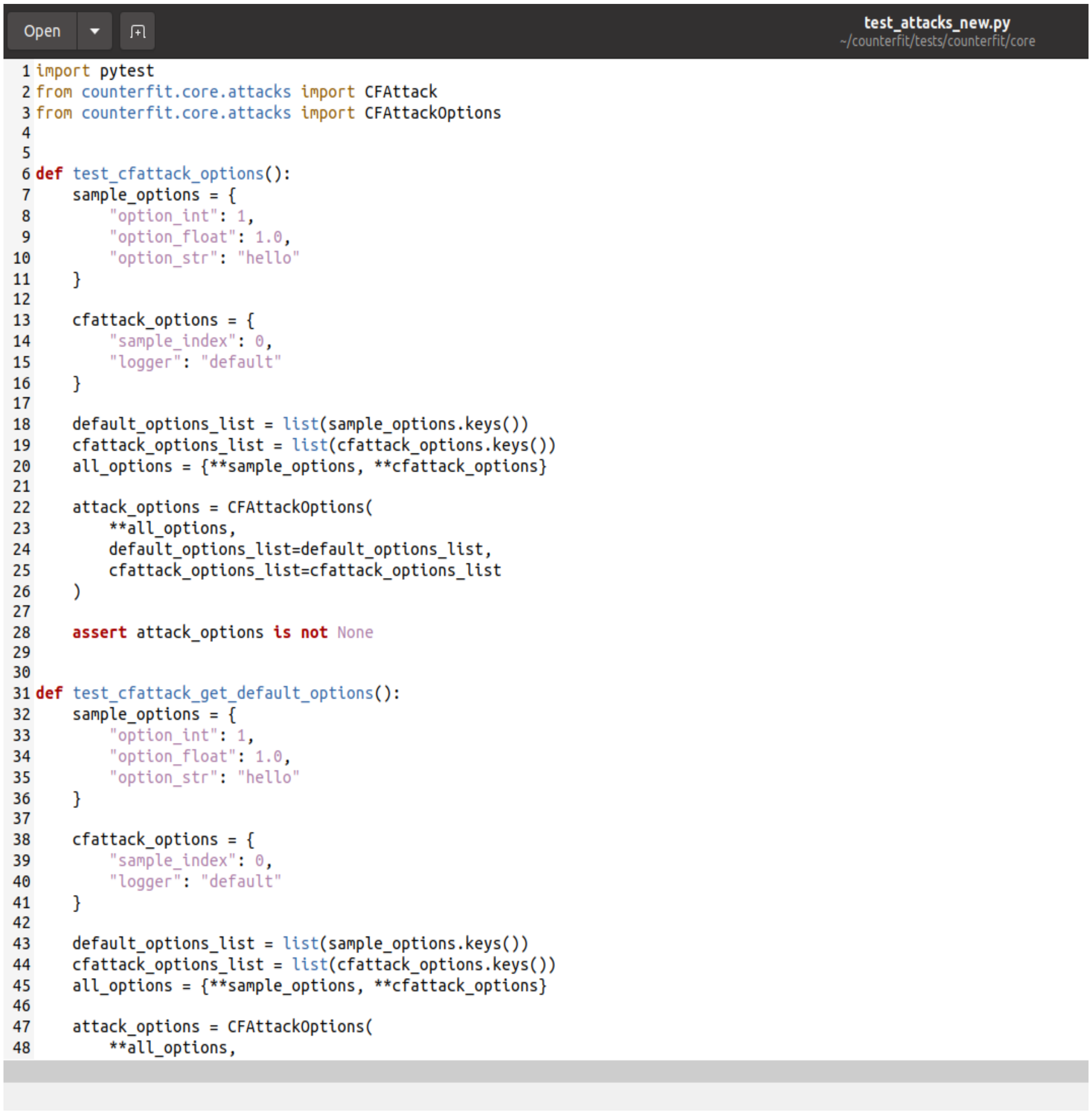
[19] Watney, M. M. "Artificial intelligence and its' legal risk to cybersecurity." *European conference on cyber warfare and security*. Academic Conferences International Limited, 2020.

[20] Chander, Bhanu, and Gopalakrishnan Kumaravelan. "Cyber security with AI—Part I." *The" Essence" of network security: An end-to-end panorama*. Springer, Singapore, 2021. 147-171.

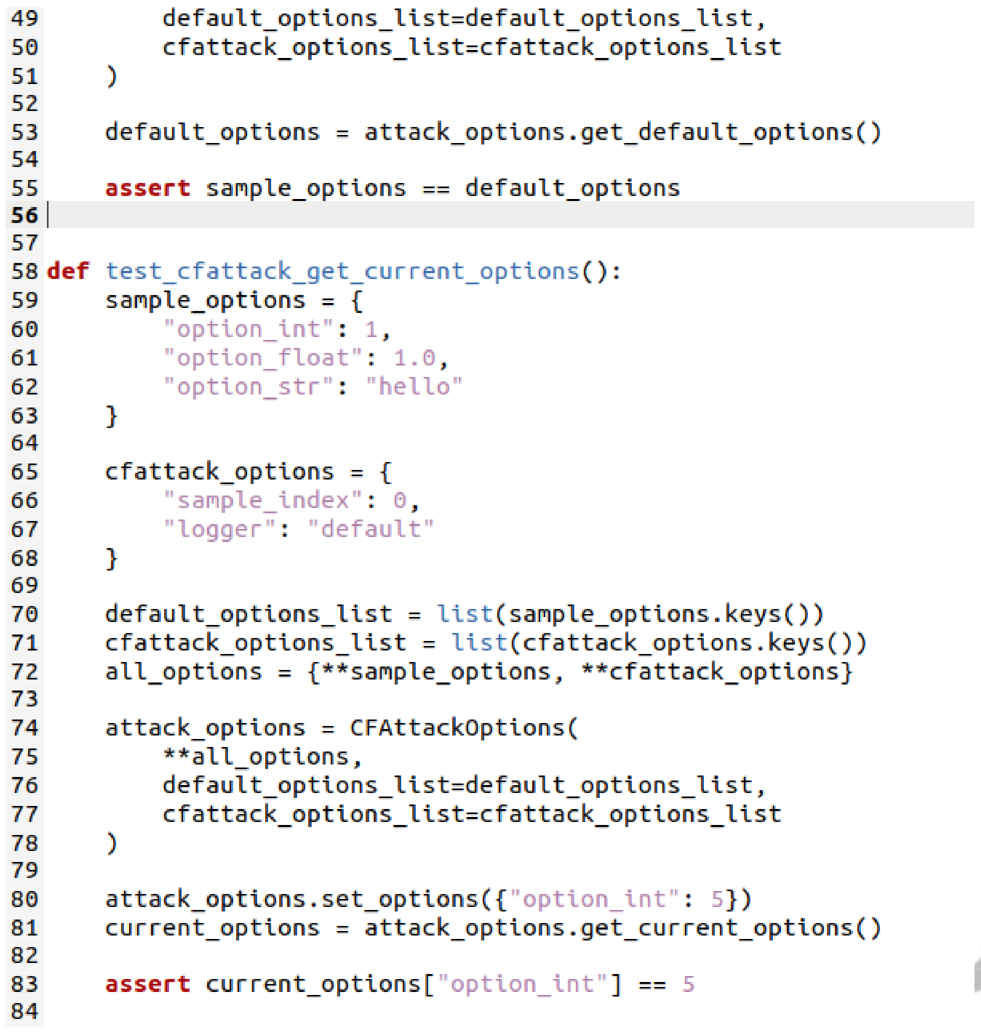
[21] Ciolino, Matthew, Josh Kalin, and David Noever. "Fortify Machine Learning Production Systems: Detect and Classify Adversarial Attacks." *arXiv preprint arXiv:2102.09695* (2021).

Appendices

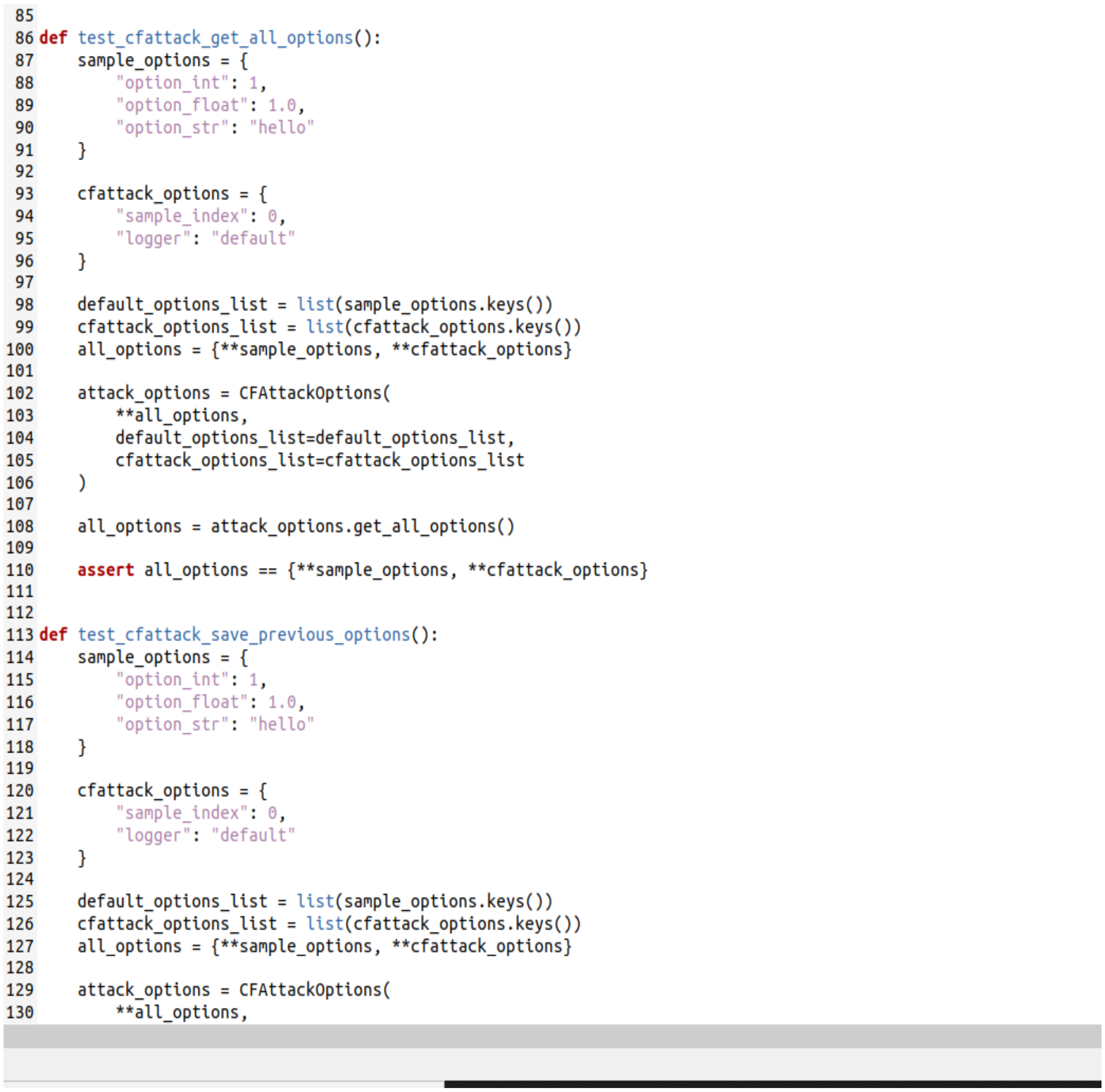
Appendix A: Test_attack.py



```python
import pytest
from counterfit.core.attacks import CFAttack
from counterfit.core.attacks import CFAttackOptions


def test_cfattack_options():
    sample_options = {
        "option_int": 1,
        "option_float": 1.0,
        "option_str": "hello"
    }

    cfattack_options = {
        "sample_index": 0,
        "logger": "default"
    }

    default_options_list = list(sample_options.keys())
    cfattack_options_list = list(cfattack_options.keys())
    all_options = {**sample_options, **cfattack_options}

    attack_options = CFAttackOptions(
        **all_options,
        default_options_list=default_options_list,
        cfattack_options_list=cfattack_options_list
    )

    assert attack_options is not None


def test_cfattack_get_default_options():
    sample_options = {
        "option_int": 1,
        "option_float": 1.0,
        "option_str": "hello"
    }

    cfattack_options = {
        "sample_index": 0,
        "logger": "default"
    }

    default_options_list = list(sample_options.keys())
    cfattack_options_list = list(cfattack_options.keys())
    all_options = {**sample_options, **cfattack_options}

    attack_options = CFAttackOptions(
        **all_options,
```

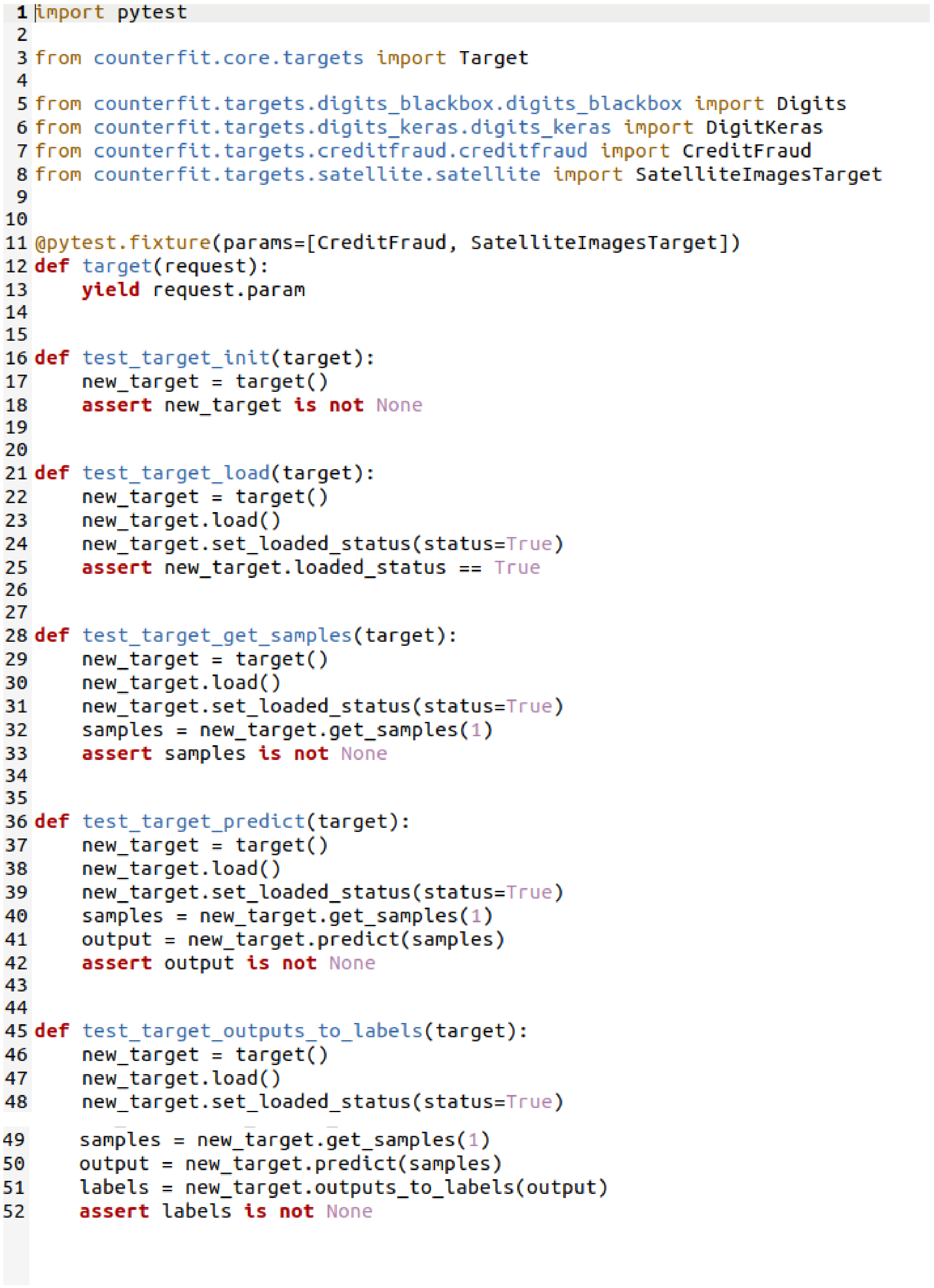


```
        default_options_list=default_options_list,
        cfattack_options_list=cfattack_options_list
    )

    default_options = attack_options.get_default_options()

    assert sample_options == default_options


def test_cfattack_get_current_options():
    sample_options = {
        "option_int": 1,
        "option_float": 1.0,
        "option_str": "hello"
    }

    cfattack_options = {
        "sample_index": 0,
        "logger": "default"
    }

    default_options_list = list(sample_options.keys())
    cfattack_options_list = list(cfattack_options.keys())
    all_options = {**sample_options, **cfattack_options}

    attack_options = CFAttackOptions(
        **all_options,
        default_options_list=default_options_list,
        cfattack_options_list=cfattack_options_list
    )

    attack_options.set_options({"option_int": 5})
    current_options = attack_options.get_current_options()

    assert current_options["option_int"] == 5

```



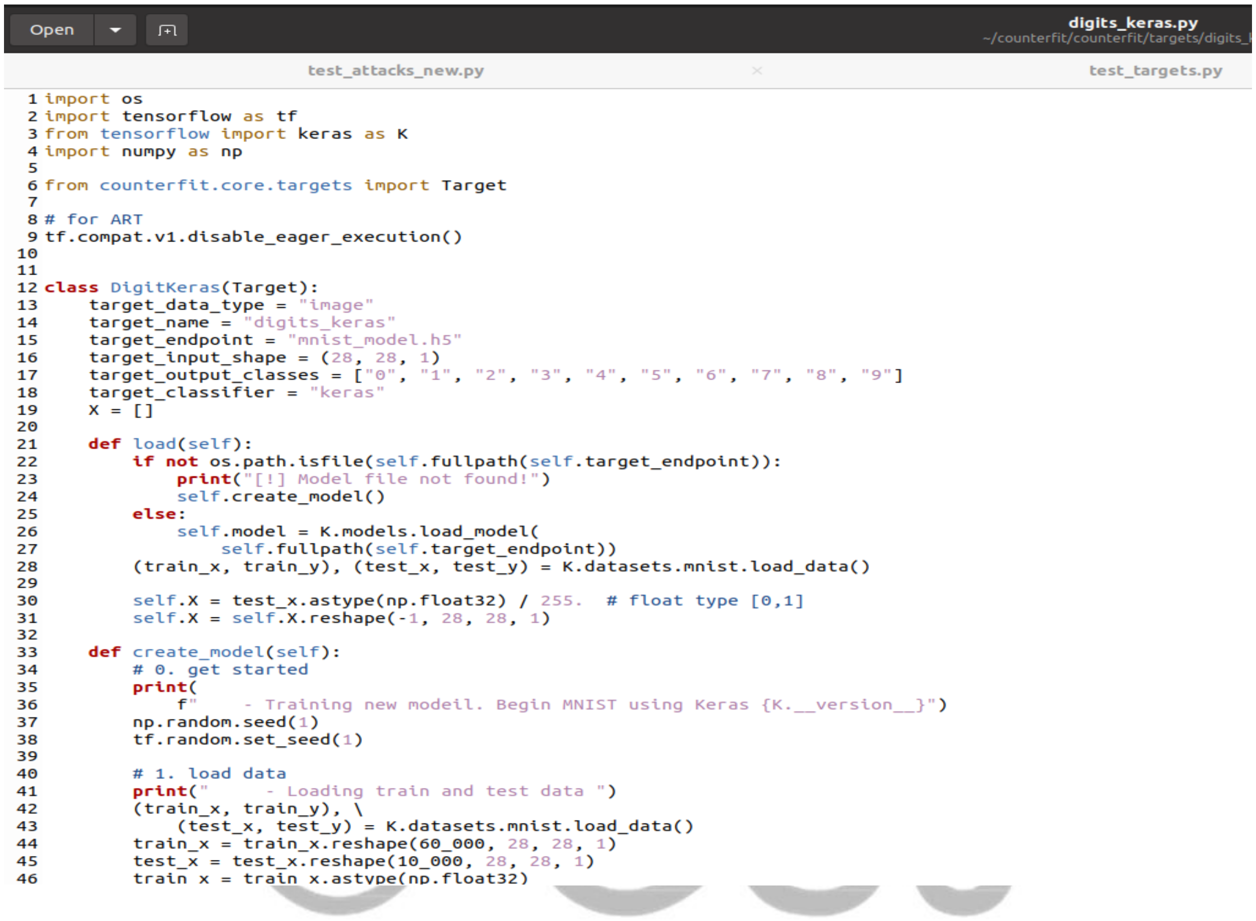
```python

def test_cfattack_get_all_options():
    sample_options = {
        "option_int": 1,
        "option_float": 1.0,
        "option_str": "hello"
    }

    cfattack_options = {
        "sample_index": 0,
        "logger": "default"
    }

    default_options_list = list(sample_options.keys())
    cfattack_options_list = list(cfattack_options.keys())
    all_options = {**sample_options, **cfattack_options}

    attack_options = CFAttackOptions(
        **all_options,
        default_options_list=default_options_list,
        cfattack_options_list=cfattack_options_list
    )

    all_options = attack_options.get_all_options()

    assert all_options == {**sample_options, **cfattack_options}


def test_cfattack_save_previous_options():
    sample_options = {
        "option_int": 1,
        "option_float": 1.0,
        "option_str": "hello"
    }

    cfattack_options = {
        "sample_index": 0,
        "logger": "default"
    }

    default_options_list = list(sample_options.keys())
    cfattack_options_list = list(cfattack_options.keys())
    all_options = {**sample_options, **cfattack_options}

    attack_options = CFAttackOptions(
        **all_options,
```

Appendix B: Test_targets.py

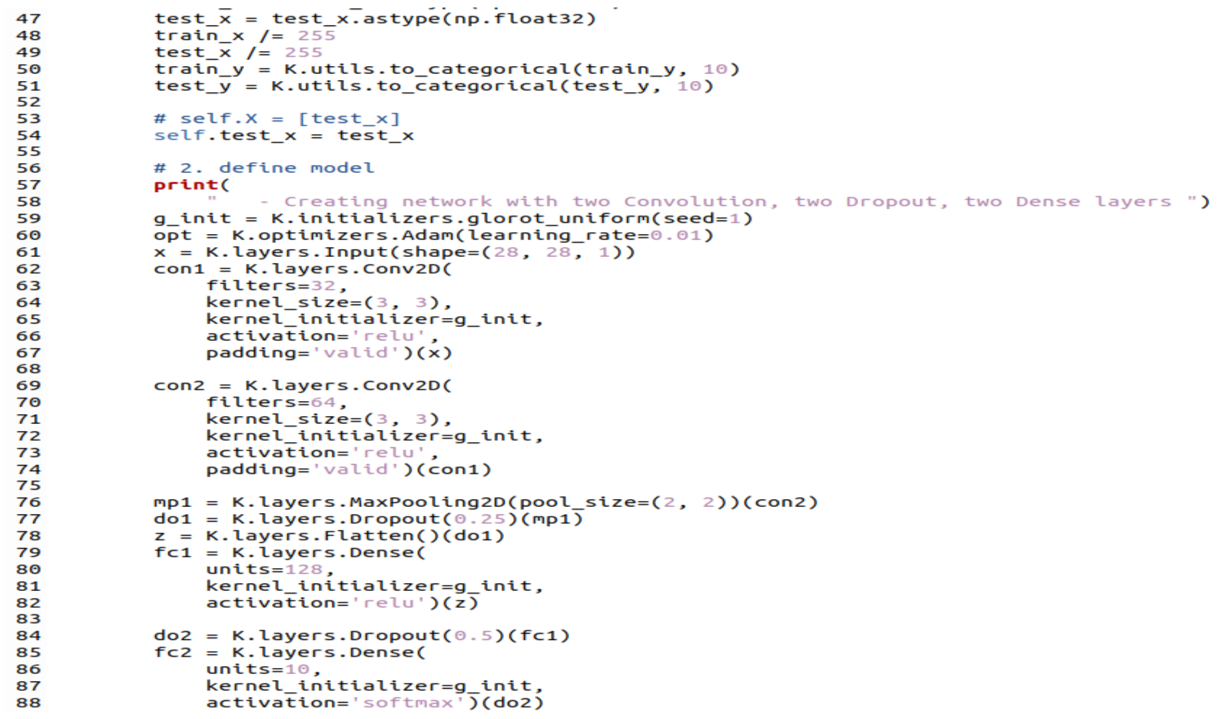


```python
import pytest

from counterfit.core.targets import Target

from counterfit.targets.digits_blackbox.digits_blackbox import Digits
from counterfit.targets.digits_keras.digits_keras import DigitKeras
from counterfit.targets.creditfraud.creditfraud import CreditFraud
from counterfit.targets.satellite.satellite import SatelliteImagesTarget


@pytest.fixture(params=[CreditFraud, SatelliteImagesTarget])
def target(request):
    yield request.param


def test_target_init(target):
    new_target = target()
    assert new_target is not None


def test_target_load(target):
    new_target = target()
    new_target.load()
    new_target.set_loaded_status(status=True)
    assert new_target.loaded_status == True


def test_target_get_samples(target):
    new_target = target()
    new_target.load()
    new_target.set_loaded_status(status=True)
    samples = new_target.get_samples(1)
    assert samples is not None


def test_target_predict(target):
    new_target = target()
    new_target.load()
    new_target.set_loaded_status(status=True)
    samples = new_target.get_samples(1)
    output = new_target.predict(samples)
    assert output is not None


def test_target_outputs_to_labels(target):
    new_target = target()
    new_target.load()
    new_target.set_loaded_status(status=True)
    samples = new_target.get_samples(1)
    output = new_target.predict(samples)
    labels = new_target.outputs_to_labels(output)
    assert labels is not None
```

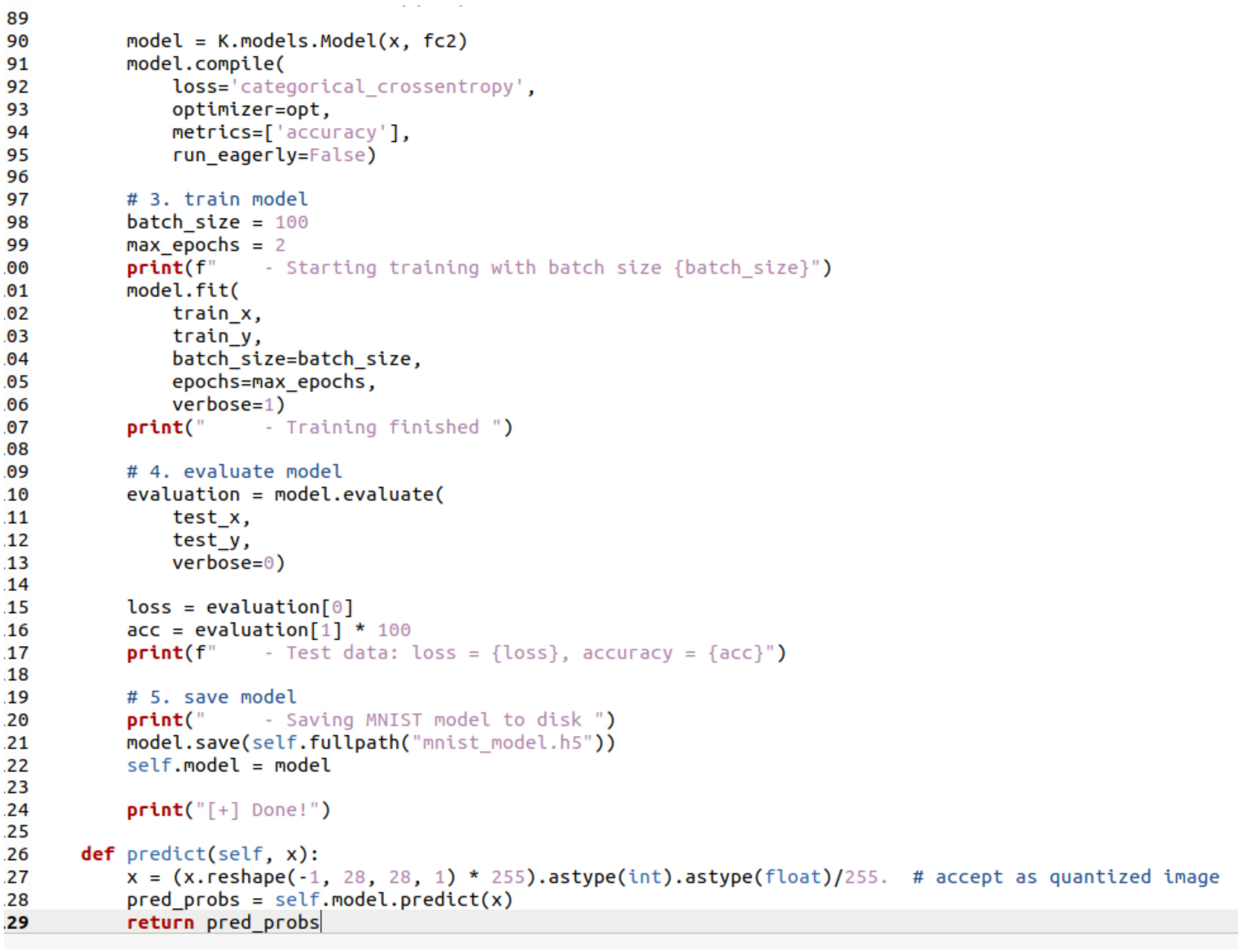
Appendix C: Digits_keras.py



```python
import os
import tensorflow as tf
from tensorflow import keras as K
import numpy as np

from counterfit.core.targets import Target

# for ART
tf.compat.v1.disable_eager_execution()


class DigitKeras(Target):
    target_data_type = "image"
    target_name = "digits_keras"
    target_endpoint = "mnist_model.h5"
    target_input_shape = (28, 28, 1)
    target_output_classes = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
    target_classifier = "keras"
    X = []

    def load(self):
        if not os.path.isfile(self.fullpath(self.target_endpoint)):
            print("[!] Model file not found!")
            self.create_model()
        else:
            self.model = K.models.load_model(
                self.fullpath(self.target_endpoint))
        (train_x, train_y), (test_x, test_y) = K.datasets.mnist.load_data()

        self.X = test_x.astype(np.float32) / 255.  # float type [0,1]
        self.X = self.X.reshape(-1, 28, 28, 1)

    def create_model(self):
        # 0. get started
        print(
            f"     - Training new modell. Begin MNIST using Keras {K.__version__}")
        np.random.seed(1)
        tf.random.set_seed(1)

        # 1. load data
        print("     - Loading train and test data ")
        (train_x, train_y), \
            (test_x, test_y) = K.datasets.mnist.load_data()
        train_x = train_x.reshape(60_000, 28, 28, 1)
        test_x = test_x.reshape(10_000, 28, 28, 1)
        train_x = train_x.astype(np.float32)
```

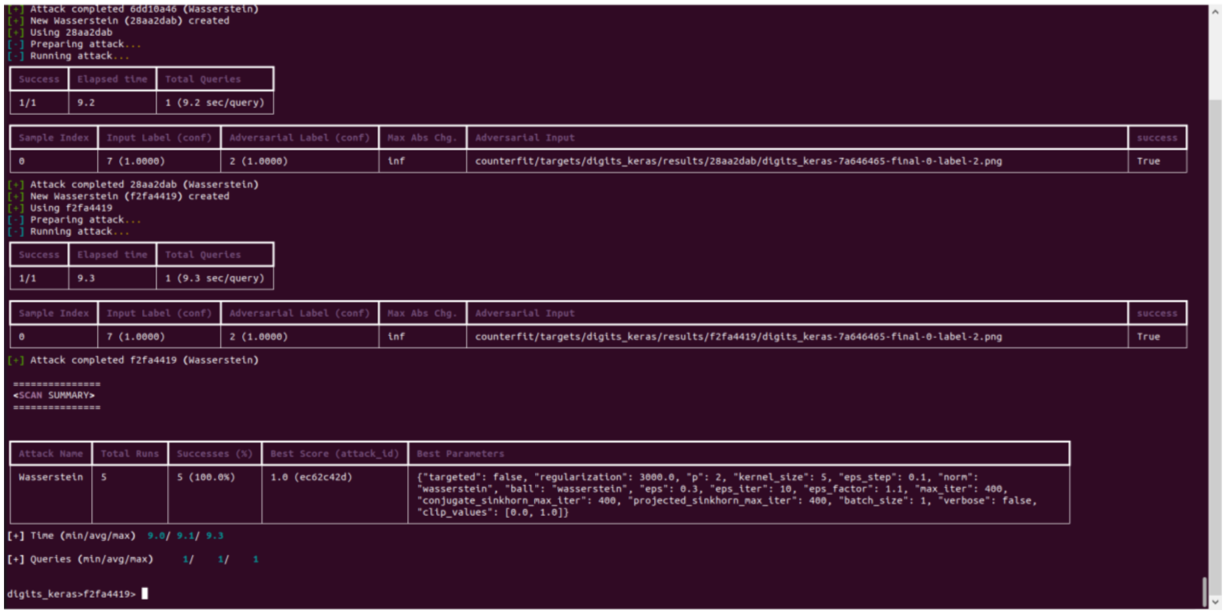


```
        test_x = test_x.astype(np.float32)
        train_x /= 255
        test_x /= 255
        train_y = K.utils.to_categorical(train_y, 10)
        test_y = K.utils.to_categorical(test_y, 10)

        # self.X = [test_x]
        self.test_x = test_x

        # 2. define model
        print(
            "    - Creating network with two Convolution, two Dropout, two Dense layers ")
        g_init = K.initializers.glorot_uniform(seed=1)
        opt = K.optimizers.Adam(learning_rate=0.01)
        x = K.layers.Input(shape=(28, 28, 1))
        con1 = K.layers.Conv2D(
            filters=32,
            kernel_size=(3, 3),
            kernel_initializer=g_init,
            activation='relu',
            padding='valid')(x)

        con2 = K.layers.Conv2D(
            filters=64,
            kernel_size=(3, 3),
            kernel_initializer=g_init,
            activation='relu',
            padding='valid')(con1)

        mp1 = K.layers.MaxPooling2D(pool_size=(2, 2))(con2)
        do1 = K.layers.Dropout(0.25)(mp1)
        z = K.layers.Flatten()(do1)
        fc1 = K.layers.Dense(
            units=128,
            kernel_initializer=g_init,
            activation='relu')(z)

        do2 = K.layers.Dropout(0.5)(fc1)
        fc2 = K.layers.Dense(
            units=10,
            kernel_initializer=g_init,
            activation='softmax')(do2)
```



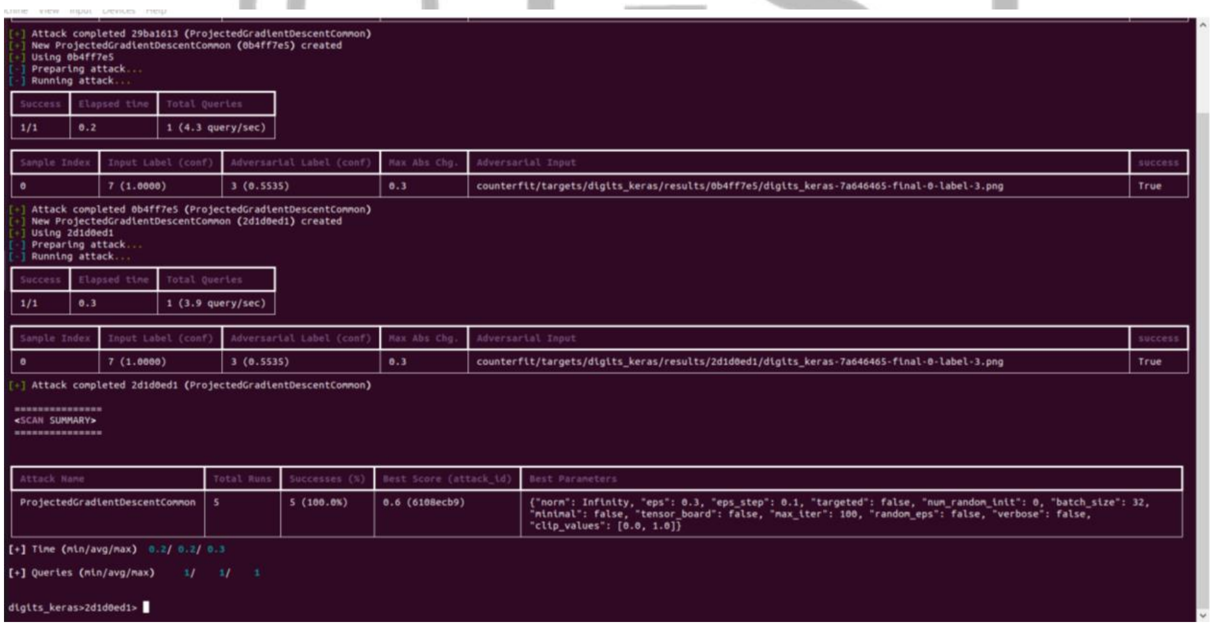
```

        model = K.models.Model(x, fc2)
        model.compile(
            loss='categorical_crossentropy',
            optimizer=opt,
            metrics=['accuracy'],
            run_eagerly=False)

        # 3. train model
        batch_size = 100
        max_epochs = 2
        print(f"    - Starting training with batch size {batch_size}")
        model.fit(
            train_x,
            train_y,
            batch_size=batch_size,
            epochs=max_epochs,
            verbose=1)
        print("    - Training finished ")

        # 4. evaluate model
        evaluation = model.evaluate(
            test_x,
            test_y,
            verbose=0)

        loss = evaluation[0]
        acc = evaluation[1] * 100
        print(f"    - Test data: loss = {loss}, accuracy = {acc}")

        # 5. save model
        print("    - Saving MNIST model to disk ")
        model.save(self.fullpath("mnist_model.h5"))
        self.model = model

        print("[+] Done!")

    def predict(self, x):
        x = (x.reshape(-1, 28, 28, 1) * 255).astype(int).astype(float)/255.  # accept as quantized image
        pred_probs = self.model.predict(x)
        return pred_probs
```

Appendix D: Wasserstein attack



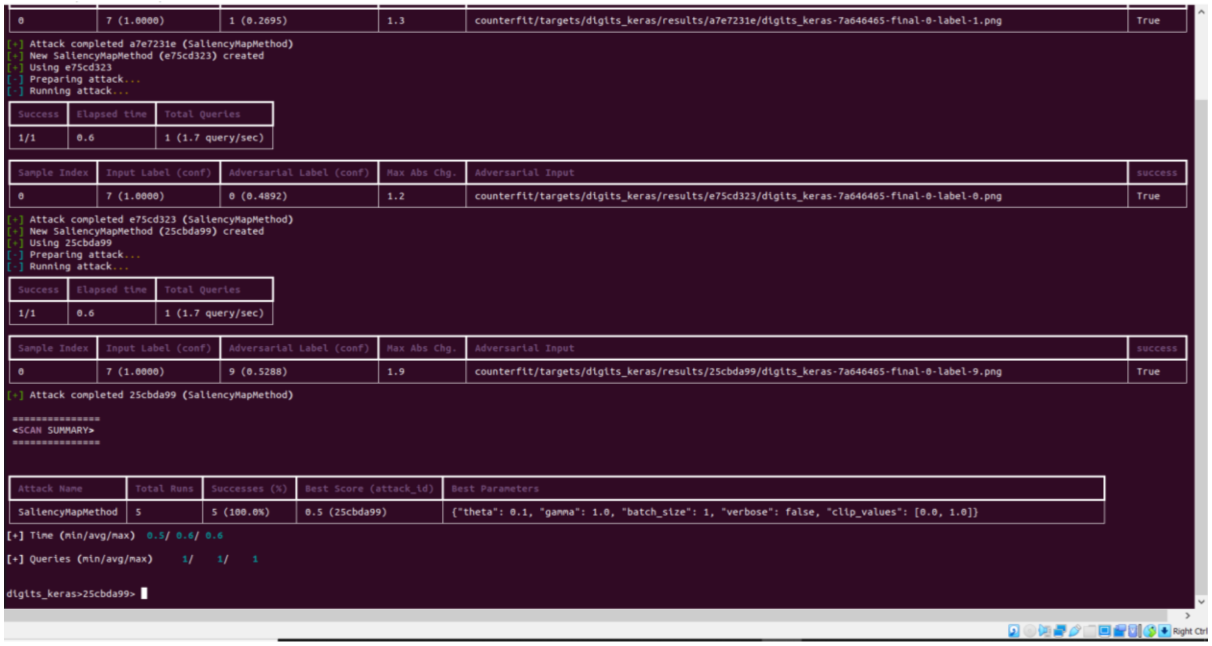
```
[+] Attack completed 6dd10a46 (Wasserstein)
[+] New Wasserstein (28aa2dab) created
[+] Using 28aa2dab
[-] Preparing attack...
[-] Running attack...
| Success | Elapsed time | Total Queries      |
| 1/1     | 9.2          | 1 (9.2 sec/query)  |

| Sample Index | Input Label (conf) | Adversarial Label (conf) | Max Abs Chg. | Adversarial Input                                                                   | success |
| 0            | 7 (1.0000)         | 2 (1.0000)               | inf          | counterfit/targets/digits_keras/results/28aa2dab/digits_keras-7a646465-final-0-label-2.png | True    |
[+] Attack completed 28aa2dab (Wasserstein)
[+] New Wasserstein (f2fa4419) created
[+] Using f2fa4419
[-] Preparing attack...
[-] Running attack...
| Success | Elapsed time | Total Queries      |
| 1/1     | 9.3          | 1 (9.3 sec/query)  |

| Sample Index | Input Label (conf) | Adversarial Label (conf) | Max Abs Chg. | Adversarial Input                                                                   | success |
| 0            | 7 (1.0000)         | 2 (1.0000)               | inf          | counterfit/targets/digits_keras/results/f2fa4419/digits_keras-7a646465-final-0-label-2.png | True    |
[+] Attack completed f2fa4419 (Wasserstein)

 ===============
 <SCAN SUMMARY>
 ===============

| Attack Name | Total Runs | Successes (%) | Best Score (attack_id) | Best Parameters                                                                          |
| Wasserstein | 5          | 5 (100.0%)    | 1.0 (ec62c42d)         | {"targeted": false, "regularization": 3000.0, "p": 2, "kernel_size": 5, "eps_step": 0.1, "norm": |
|             |            |               |                        | "wasserstein", "ball": "wasserstein", "eps": 0.3, "eps_iter": 10, "eps_factor": 1.1, "max_iter": 400, |
|             |            |               |                        | "conjugate_sinkhorn_max_iter": 400, "projected_sinkhorn_max_iter": 400, "batch_size": 1, "verbose": false, |
|             |            |               |                        | "clip_values": [0.0, 1.0]}                                                               |
[+] Time (min/avg/max)  9.0/ 9.1/ 9.3

[+] Queries (min/avg/max)     1/    1/    1


digits_keras>f2fa4419> █
```

Appendix E: Projected Gradient Decent Common Attack



```
[+] Attack completed 29ba1613 (ProjectedGradientDescentCommon)
[+] New ProjectedGradientDescentCommon (0b4ff7e5) created
[+] Using 0b4ff7e5
[-] Preparing attack...
[-] Running attack...
| Success | Elapsed time | Total Queries      |
| 1/1     | 0.2          | 1 (4.3 query/sec)  |

| Sample Index | Input Label (conf) | Adversarial Label (conf) | Max Abs Chg. | Adversarial Input                                                                   | success |
| 0            | 7 (1.0000)         | 3 (0.5535)               | 0.3          | counterfit/targets/digits_keras/results/0b4ff7e5/digits_keras-7a646465-final-0-label-3.png | True    |
[+] Attack completed 0b4ff7e5 (ProjectedGradientDescentCommon)
[+] New ProjectedGradientDescentCommon (2d1d0ed1) created
[+] Using 2d1d0ed1
[-] Preparing attack...
[-] Running attack...
| Success | Elapsed time | Total Queries      |
| 1/1     | 0.3          | 1 (3.9 query/sec)  |

| Sample Index | Input Label (conf) | Adversarial Label (conf) | Max Abs Chg. | Adversarial Input                                                                   | success |
| 0            | 7 (1.0000)         | 3 (0.5535)               | 0.3          | counterfit/targets/digits_keras/results/2d1d0ed1/digits_keras-7a646465-final-0-label-3.png | True    |
[+] Attack completed 2d1d0ed1 (ProjectedGradientDescentCommon)

 ===============
 <SCAN SUMMARY>
 ===============

| Attack Name                    | Total Runs | Successes (%) | Best Score (attack_id) | Best Parameters                                                                          |
| ProjectedGradientDescentCommon | 5          | 5 (100.0%)    | 0.6 (6108ecb9)         | {"norm": Infinity, "eps": 0.3, "eps_step": 0.1, "targeted": false, "num_random_init": 0, "batch_size": 32, |
|                                |            |               |                        | "minimal": false, "tensor_board": false, "max_iter": 100, "random_eps": false, "verbose": false, |
|                                |            |               |                        | "clip_values": [0.0, 1.0]}                                                               |
[+] Time (min/avg/max)  0.2/ 0.2/ 0.3

[+] Queries (min/avg/max)     1/    1/    1


digits_keras>2d1d0ed1> █
```

Appendix F: Saliency Map Method Attack



```
| 0            | 7 (1.0000)         | 1 (0.2695)               | 1.3           | counterfit/targets/digits_keras/results/a7e7231e/digits_keras-7a646465-final-0-label-1.png | True    |
[+] Attack completed a7e7231e (SaliencyMapMethod)
[+] New SaliencyMapMethod (e75cd323) created
[+] Using e75cd323
[-] Preparing attack...
[-] Running attack...
| Success | Elapsed time | Total Queries       |
| 1/1     | 0.6          | 1 (1.7 query/sec)   |

| Sample Index | Input Label (conf) | Adversarial Label (conf) | Max Abs Chg.  | Adversarial Input                                                                          | success |
| 0            | 7 (1.0000)         | 0 (0.4892)               | 1.2           | counterfit/targets/digits_keras/results/e75cd323/digits_keras-7a646465-final-0-label-0.png | True    |
[+] Attack completed e75cd323 (SaliencyMapMethod)
[+] New SaliencyMapMethod (25cbda99) created
[+] Using 25cbda99
[-] Preparing attack...
[-] Running attack...
| Success | Elapsed time | Total Queries       |
| 1/1     | 0.6          | 1 (1.7 query/sec)   |

| Sample Index | Input Label (conf) | Adversarial Label (conf) | Max Abs Chg.  | Adversarial Input                                                                          | success |
| 0            | 7 (1.0000)         | 9 (0.5288)               | 1.9           | counterfit/targets/digits_keras/results/25cbda99/digits_keras-7a646465-final-0-label-9.png | True    |
[+] Attack completed 25cbda99 (SaliencyMapMethod)

 ===============
 <SCAN SUMMARY>
 ===============

| Attack Name       | Total Runs | Successes (%) | Best Score (attack_id) | Best Parameters                                                                            |
| SaliencyMapMethod | 5          | 5 (100.0%)    | 0.5 (25cbda99)         | {"theta": 0.1, "gamma": 1.0, "batch_size": 1, "verbose": false, "clip_values": [0.0, 1.0]} |
[+] Time (min/avg/max)  0.5/ 0.6/ 0.6

[+] Queries (min/avg/max)     1/    1/    1


digits_keras>25cbda99> █
```

© GSJ