



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение высшего  
образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

---

Институт кибербезопасности и цифровых технологий  
КБ-4 «Интеллектуальные системы информационной безопасности»

**Отчет по практической работе №4**

по дисциплине: «Анализ защищенности систем искусственного  
интеллекта»

Выполнила:

Студент группы ББМО-02-22

Щелкушкин Евгений Романович

Проверил:

К.т.н. Спирин Андрей Андреевич

Москва, 2023

## Шаг 1 копируем репозиторий

```
[1] !git clone https://github.com/ewatson2/EEL6812_DeepFool_Project
```

```
Cloning into 'EEL6812_DeepFool_Project'...
remote: Enumerating objects: 96, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 96 (delta 2), reused 1 (delta 1), pack-reused 93
Receiving objects: 100% (96/96), 33.99 MiB | 15.41 MiB/s, done.
Resolving deltas: 100% (27/27), done.
```

## Шаг 2 переход в директорию

Шаг 2 перейдём в "EEL6812\_DeepFool\_Project"

```
[5] %cd EEL6812_DeepFool_Project/
```

```
[Errno 2] No such file or directory: 'EEL6812_DeepFool_Project/'
/content/EEL6812_DeepFool_Project
```

## Шаг 3 Установка numpy, art, adversarial-robustness-toolbox

Шаг 3 Установим numpy, art, adversarial-robustness-toolbox

```
[4] !pip install numpy
!pip install art
!pip install adversarial-robustness-toolbox
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)
Requirement already satisfied: art in /usr/local/lib/python3.10/dist-packages (6.1)
Requirement already satisfied: adversarial-robustness-toolbox in /usr/local/lib/python3.10/dist-packages (1.16.0)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.23.5)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.11.4)
Requirement already satisfied: scikit-learn<1.2.0,>=0.22.2 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.1.3)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (4.66.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (67.7.2)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn<1.2.0,>=0.22.2->adversarial-robustness-toolbox) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn<1.2.0,>=0.22.2->adversarial-robustness-toolbox) (3.2.0)
```

```
[6] from __future__ import absolute_import, division, print_function, unicode_literals

import os, sys
from os.path import abspath

module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
tf.get_logger().setLevel('ERROR')
import tensorflow.keras.backend as k
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Activation, Dropout
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from art.estimators.classification import KerasClassifier
from art.attacks.poisoning import PoisoningAttackBackdoor, PoisoningAttackCleanLabelBackdoor
from art.attacks.poisoning.perturbations import add_pattern_bd
from art.utils import load_mnist, preprocess, to_categorical
from art.defences.trainer import AdversarialTrainerMadryPGD
```

## Шаг 4 загружаем датасет

### Шаг 4 Загружаем датасет.

```
[7] # Загружаем датасет и записываем в переменные для обучения и теста
(x_raw, y_raw), (x_raw_test, y_raw_test), min_, max_ = load_mnist(raw=True)
n_train = np.shape(x_raw)[0]
num_selection = 10000
random_selection_indices = np.random.choice(n_train, num_selection)
x_raw = x_raw[random_selection_indices]
y_raw = y_raw[random_selection_indices]
```

## Шаг 5 предобработка данных

### Шаг 5 Выполняем предобработку данных.

```
[8] # фиксируем коэффициент отравления
percent_poison = .33
# обучающие данные
x_train, y_train = preprocess(x_raw, y_raw)
x_train = np.expand_dims(x_train, axis=3)
# данные для теста
x_test, y_test = preprocess(x_raw_test, y_raw_test)
x_test = np.expand_dims(x_test, axis=3)
# обучающие классы
n_train = np.shape(y_train)[0]
shuffled_indices = np.arange(n_train)
np.random.shuffle(shuffled_indices)
x_train = x_train[shuffled_indices]
y_train = y_train[shuffled_indices]
```

**Шаг 6** пишем функцию для создания последовательной модели из 9 слоёв.

Шаг 6 Пишем функцию для создания последовательной модели из 9 слоёв.

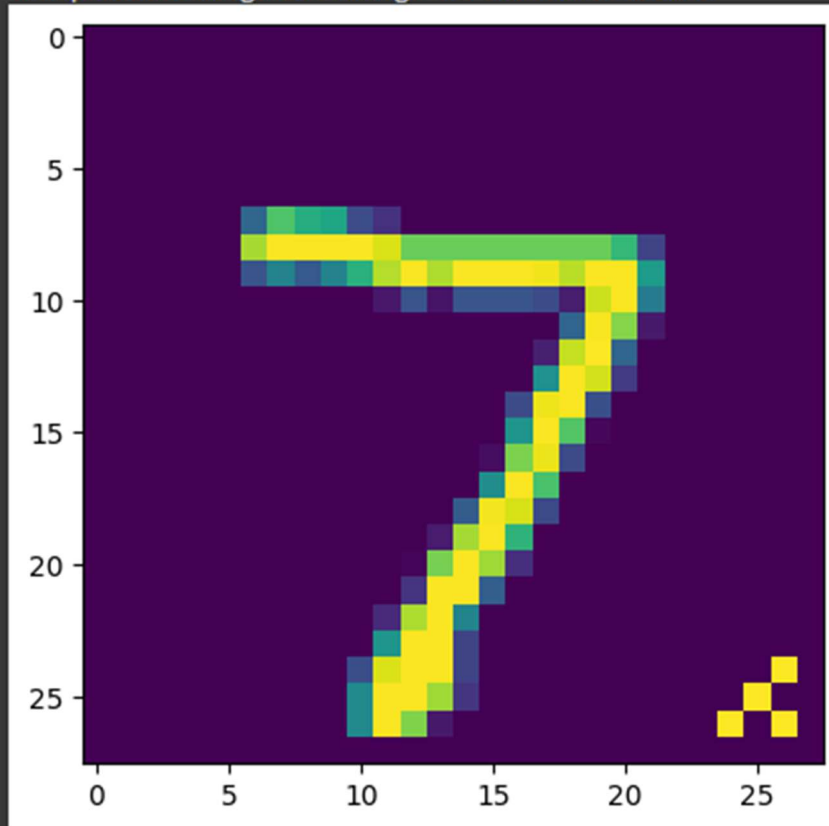
```
[9] def create_model():  
    # объявляем последовательную модель  
    model = Sequential()  
    # добавляем первый сверточный слой  
    model.add(Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)))  
    # добавляем второй сверточный слой  
    model.add(Conv2D(64, (3,3), activation='relu'))  
    # добавляем слой пуллинга  
    model.add(MaxPooling2D((2,2)))  
    # добавляем первый дропаут  
    model.add(Dropout(0.25))  
    # добавляем слой выравнивания  
    model.add(Flatten())  
    # добавляем первый полносвязный слой  
    model.add(Dense(128, activation = 'relu'))  
    # добавляем второй дропаут  
    model.add(Dropout(0.25))  
    # добавляем второй полносвязный слой  
    model.add(Dense(10, activation = 'softmax'))  
    # компилируем нашу модель  
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])  
    # возвращаем скомпилированную модель  
    return model
```

## Шаг 7 Создаем атаку

### Шаг 7 Создаем атаку.

```
[10] # объявляем класс, реализующий backdoor-атаку
      backdoor = PoisoningAttackBackdoor(add_pattern_bd)
      # выберем пример атаки
      example_target = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1])
      # реализуем
      pdata, plabels = backdoor.poison(x_test, y=example_target)
      # визуализируем атакованный пример
      plt.imshow(pdata[0].squeeze())
```

<matplotlib.image.AxesImage at 0x7e101be43880>



## Шаг 8 Определяем целевой класс атаки.

### Шаг 8 Определяем целевой класс атаки.

```
✓ [11] targets = to_categorical([9], 10)[0]
0 сек.
```

## Шаг 9 Создаем модель

Шаг 9 Создаем модель.

```
[12] # обычная модель
model = KerasClassifier(create_model())
# модель, наученная состязательным подходом по протоколу Мэдри
proxy = AdversarialTrainerMadryPGD(KerasClassifier(create_model()), nb_epochs=10, eps=0.15, eps_step=0.001)
# обучаем последнюю
proxy.fit(x_train, y_train)
```

Precompute adv samples: 100%  1/1 [00:00<00:00, 58.20it/s]

Adversarial training epochs: 100%  10/10 [01:58<00:00, 11.08s/it]

## Шаг 10 Выполняем атаку


```
[13] # конфигурируем атаку под модель Мэдри
attack = PoisoningAttackCleanLabelBackdoor(backdoor=backdoor, proxy_classifier=proxy.get_classifier(), target=targets, pp_poison=percent_poison, norm=2, eps=5, eps_step=0.1, max_iter=200)
# запускаем
pdata, plabels = attack.poison(x_train, y_train)
```

PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.13it/s]

PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.14it/s]


PGD - Random Initializations: 100%  1/1 [00:01<00:00, 1.00s/it]


PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.14it/s]


PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.13it/s]

PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.09it/s]

PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.09it/s]

PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.10it/s]

PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.07it/s]

PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.07it/s]

PGD - Random Initializations: 100%  1/1 [00:01<00:00, 1.17s/it]

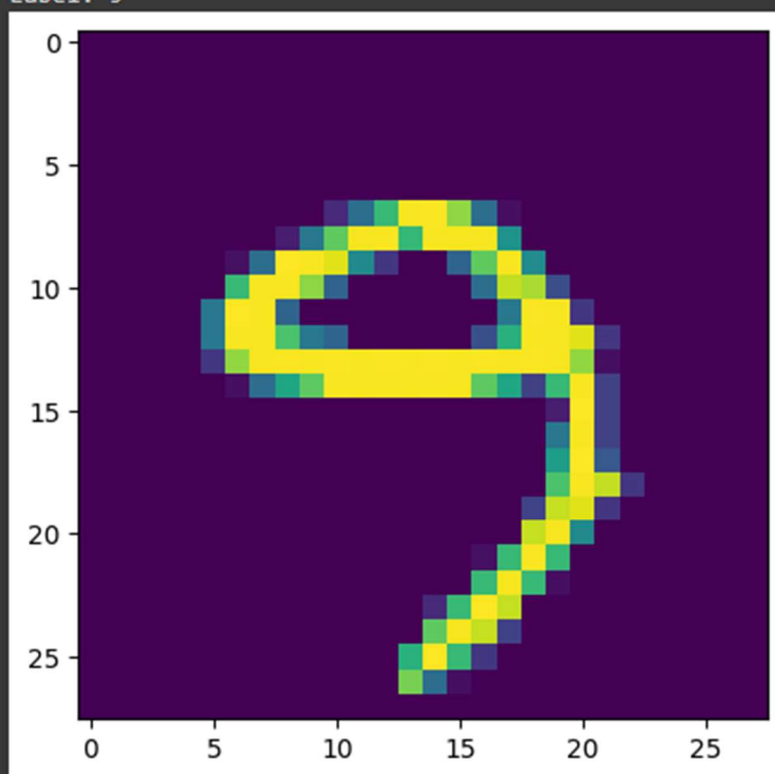
## Шаг 11 создаем отравленные примеры данных

### Шаг 11 Создаем отравленные примеры данных.

```
[14] # отравленные входы
poisoned = pdata[np.all(labels == targets, axis=1)]
# следом берём отравленные выходы
poisoned_labels = labels[np.all(labels == targets, axis=1)]
# смотрим количество отравленных входов
print(len(poisoned))
idx = 0
# визуализируем одно из отравленных изображений
plt.imshow(poisoned[idx].squeeze())
print(f"Label: {np.argmax(poisoned_labels[idx])}")
```

1022

Label: 9





## Шаг 12 обучаем модель на отравленных данных

Шаг 12 Обучаем модель на отравленных данных.

```
[15] model.fit(pdata, plabels, nb_epochs=10)
```

Train on 10000 samples

Epoch 1/10

10000/10000 [=====] - 1s 94us/sample - loss: 0.6322 - accuracy: 0.7999

Epoch 2/10

10000/10000 [=====] - 1s 80us/sample - loss: 0.1995 - accuracy: 0.9401

Epoch 3/10

10000/10000 [=====] - 1s 80us/sample - loss: 0.1122 - accuracy: 0.9668

Epoch 4/10

10000/10000 [=====] - 1s 79us/sample - loss: 0.0780 - accuracy: 0.9775

Epoch 5/10

10000/10000 [=====] - 1s 79us/sample - loss: 0.0591 - accuracy: 0.9818

Epoch 6/10

10000/10000 [=====] - 1s 79us/sample - loss: 0.0432 - accuracy: 0.9863

Epoch 7/10

10000/10000 [=====] - 1s 78us/sample - loss: 0.0405 - accuracy: 0.9858

Epoch 8/10

10000/10000 [=====] - 1s 79us/sample - loss: 0.0311 - accuracy: 0.9901

Epoch 9/10

10000/10000 [=====] - 1s 79us/sample - loss: 0.0273 - accuracy: 0.9919

Epoch 10/10

10000/10000 [=====] - 1s 80us/sample - loss: 0.0225 - accuracy: 0.9934



## Шаг 13 Осуществляем тест на чистой модели

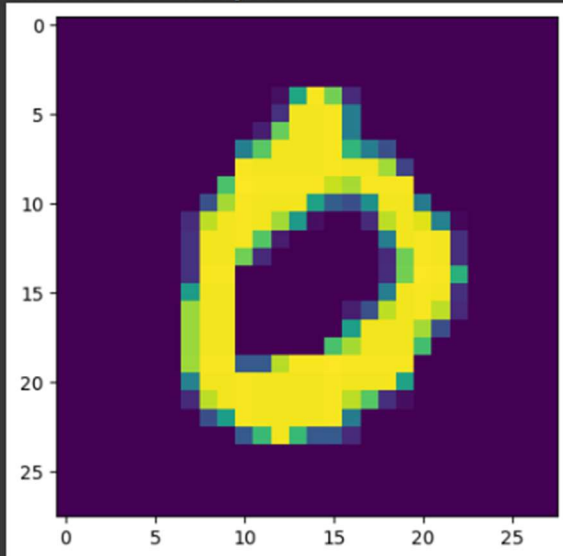
Шаг 13 Осуществляем тест на чистой модели.

```
[16] # предсказываем на тестовых входах "здоровых" примеров
clean_preds = np.argmax(model.predict(x_test), axis=1)
# вычисляем среднюю точность предсказания на полном наборе тестов
clean_correct = np.sum(clean_preds == np.argmax(y_test, axis=1))
clean_total = y_test.shape[0]
clean_acc = clean_correct / clean_total
print("\nClean test set accuracy: %.2f%%" % (clean_acc * 100))
# отобразим изображение, её класс и предсказание для легитимного примера, чтобы
# показать, как отравленная модель классифицирует легитимный пример
c = 0 # класс
i = 0 # изображение

c_idx = np.where(np.argmax(y_test, 1) == c)[0][i] # индекс изображения в массиве легитимных примеров

plt.imshow(x_test[c_idx].squeeze())
plt.show()
clean_label = c
print("Prediction: " + str(clean_preds[c_idx]))
```

Clean test set accuracy: 98.04%



Prediction: 0

## Шаг 14 получаем результаты атаки на модель

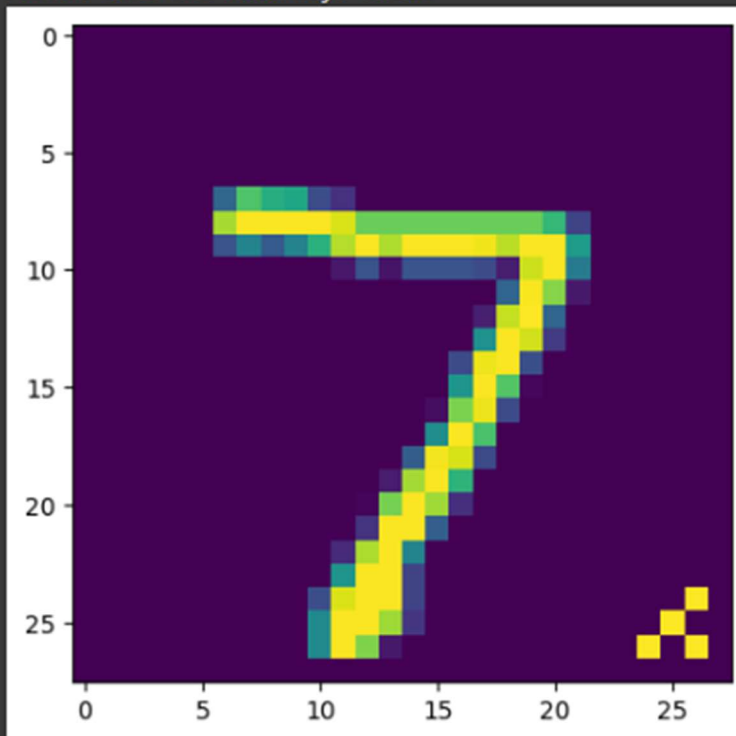
Шаг 14 Получаем результаты атаки на модель.

```
[17] not_target = np.logical_not(np.all(y_test == targets, axis=1))
    px_test, py_test = backdoor.poisson(x_test[not_target], y_test[not_target])
    # собираем предсказания для отравленных тестов
    poison_preds = np.argmax(model.predict(px_test), axis=1)
    # вычисляем среднюю точность предсказаний на полном наборе тестов
    poison_correct = np.sum(poison_preds == np.argmax(y_test[not_target], axis=1))
    poison_total = poison_preds.shape[0]

    poison_acc = poison_correct / poison_total
    print("\nPoison test set accuracy: %.2f%%" % (poison_acc * 100))

    c = 0 # индекс изображения
    # теперь отобразим изображение
    plt.imshow(px_test[c].squeeze())
    plt.show()
    # выведем предсказанный моделью класс
    clean_label = c
    print("Prediction: " + str(poison_preds[c]))
```

Poison test set accuracy: 1.03%



Prediction: 9

## **Заключение**

Практическая задание №4 выполнена успешно