



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»

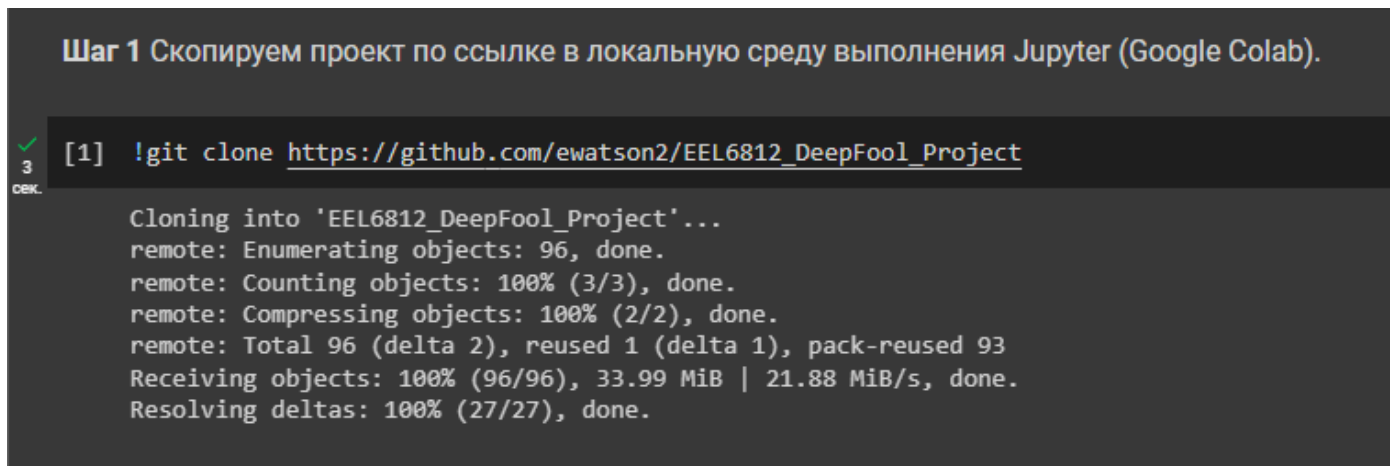
Отчет по практической работе №4
по дисциплине: «Анализ защищенности систем искусственного
интеллекта»

Выполнил:
Тимофеев И.О.
Группа: ББМО-01-22

Проверил:
К.т.н. Спирин Андрей Андреевич

Москва, 2023

Шаг 1. Скопируем проект по ссылке в локальную среду выполнения Jupyter.



The screenshot shows a Jupyter terminal window with a dark background. At the top, a title bar reads "Шаг 1 Скопируем проект по ссылке в локальную среду выполнения Jupyter (Google Colab)". Below the title bar, a green checkmark icon is visible next to the prompt "[1]". The command entered is `!git clone https://github.com/ewatson2/EEL6812_DeepFool_Project`. The output of the command is displayed in a lighter gray box, showing the progress of cloning the repository: "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 | 21.88 MiB/s, done.", and "Resolving deltas: 100% (27/27), done."

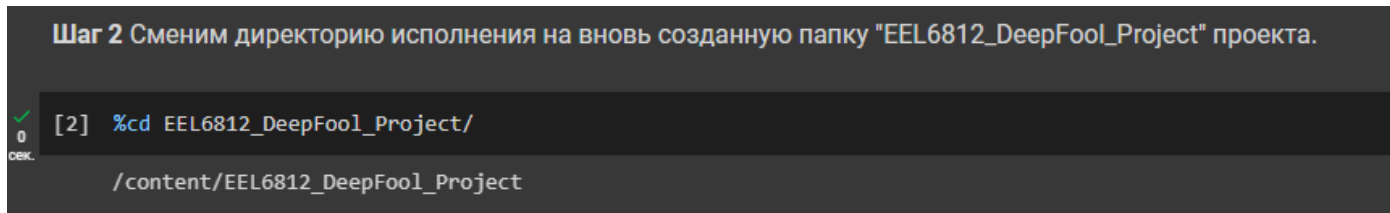
```
Шаг 1 Скопируем проект по ссылке в локальную среду выполнения Jupyter (Google Colab).

[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 | 21.88 MiB/s, done.
Resolving deltas: 100% (27/27), done.
```

Рисунок 1 – Загрузка проекта

Шаг 2. Сменим директорию исполнения на вновь созданную папку "EEL6812_DeepFool_Project" проекта.



The screenshot shows a Jupyter terminal window with a dark background. At the top, a title bar reads "Шаг 2 Сменим директорию исполнения на вновь созданную папку 'EEL6812_DeepFool_Project' проекта". Below the title bar, a green checkmark icon is visible next to the prompt "[2]". The command entered is `%cd EEL6812_DeepFool_Project/`. The output of the command is displayed in a lighter gray box, showing the current directory: `/content/EEL6812_DeepFool_Project`.

```
Шаг 2 Сменим директорию исполнения на вновь созданную папку "EEL6812_DeepFool_Project" проекта.

[2] %cd EEL6812_DeepFool_Project/

/content/EEL6812_DeepFool_Project
```

Рисунок 2 – Смена директории

Шаг 3. Выполним импорт библиотек. Загружаем библиотеку ART .

Шаг 3 Выполним импорт библиотек. Загружаем библиотеку ART.

```
[3] !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)
Collecting art
  Downloading art-6.1-py3-none-any.whl (599 kB)
    599.8/599.8 kB 7.7 MB/s eta 0:00:00
Installing collected packages: art
Successfully installed art-6.1
Collecting adversarial-robustness-toolbox
  Downloading adversarial_robustness_toolbox-1.17.0-py3-none-any.whl (1.7 MB)
    1.7/1.7 MB 12.6 MB/s eta 0:00:00
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)
Collecting scikit-learn<1.2.0,>=0.22.2 (from adversarial-robustness-toolbox)
  Downloading scikit_learn-1.1.3-cp310-cp310-manylinux_2_17_x86_64_manylinux2014_x86_64.whl (30.5 MB)
    30.5/30.5 MB 49.2 MB/s eta 0:00:00
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.16.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (67.7.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (4.66.1)
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)
Installing collected packages: scikit-learn, adversarial-robustness-toolbox
```

Рисунок 3 – Импорт библиотек art

```
[4] 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
```

Рисунок 4 – Импорт библиотек

Шаг 4. Загружаем датасет MNIST.

Шаг 4 Загружаем датасет MNIST.

```
[5] # сначала загружаем датасет MNIST и записываем его в переменные для обучения и теста (x_raw содержит исходные изображения)
(x_raw, y_raw), (x_raw_test, y_raw_test), min_, max_ = load_mnist(raw=True)
# здесь фиксируем входы обучающих данных
n_train = np.shape(x_raw)[0]
# фиксируем определённое количество обучающих данных(10000)
num_selection = 10000
# выбираем случайный индекс
random_selection_indices = np.random.choice(n_train, num_selection)
# по индексу выбираем соответствующий обучающий пример
x_raw = x_raw[random_selection_indices]
y_raw = y_raw[random_selection_indices]
```

Рисунок 5 – Грузим датасет

Шаг 5. Выполняем предобработку данных.

```
Шаг 5 Выполняем предобработку данных.

[6] # фиксируем коэффициент отравления
    percent_poison = .33
    # отравляем обучающие данные
    x_train, y_train = preprocess(x_raw, y_raw)
    x_train = np.expand_dims(x_train, axis=3)
    # отравляем данные для теста
    x_test, y_test = preprocess(x_raw_test, y_raw_test)
    x_test = np.expand_dims(x_test, axis=3)
    # фиксируем и перемешиваем 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]
```

Рисунок 6 – Предобработка данных

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

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

[7] def create_model():
    # объявляем последовательную модель
    model = Sequential()
    # добавляем первый сверточный слой (кол-во фильтров = 32, размер фильтра (3,3), активация = relu)
    model.add(Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)))
    # добавляем второй сверточный слой (кол-во фильтров = 64, размер фильтра (3,3), активация = relu)
    model.add(Conv2D(64, (3,3), activation='relu'))
    # добавляем слой пуллинга (размером (2,2))
    model.add(MaxPooling2D((2,2)))
    # добавляем первый дропаут (0,25)
    model.add(Dropout(0.25))
    # добавляем слой выравнивания (Flatten)
    model.add(Flatten())
    # добавляем первый полносвязный слой (размером = 128, активация = relu)
    model.add(Dense(128, activation = 'relu'))
    # добавляем второй дропаут (0,25)
    model.add(Dropout(0.25))
    # добавляем второй полносвязный слой (размером = 10, активация = softmax)
    model.add(Dense(10, activation = 'softmax'))
    # компилируем нашу модель
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    # возвращаем скомпилированную модель
    return model
```

Рисунок 7 – Создание последовательной модели

Шаг 7. Создаем атаку.



Рисунок 8 – Атака

Шаг 8. Определяем целевой класс атаки.

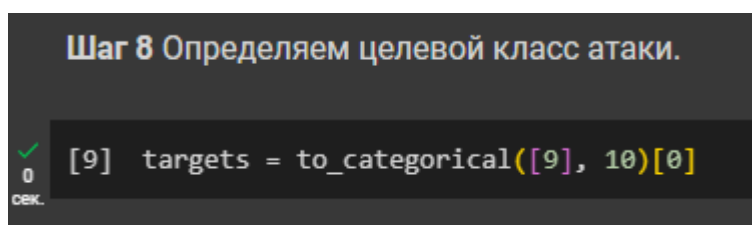



Рисунок 9 – Целевой класс атаки

Шаг 9. Создаем модель.

Шаг 9 Создаем модель.

```
[10] # 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, 1.66it/s]




Adversarial training epochs: 100%  10/10 [02:09<00:00, 11.89s/it]


Рисунок 10 – Создание модели


Шаг 10. Выполняем атаку.


```
[11] # 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, 1.02it/s]


PGD - Random Initializations: 100%  1/1 [00:01<00:00, 1.30s/it]


PGD - Random Initializations: 100%  1/1 [00:01<00:00, 1.21s/it]


PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.01it/s]


PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.01it/s]

PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.01it/s]

PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.01it/s]

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

PGD - Random Initializations: 100%  1/1 [00:01<00:00, 1.23s/it]

PGD - Random Initializations: 100%  1/1 [00:01<00:00, 1.29s/it]


PGD - Random Initializations: 100%  1/1 [00:00<00:00, 1.12it/s]

Рисунок 11 – Снова атакуем

Шаг 11. Создаем отравленные примеры данных.

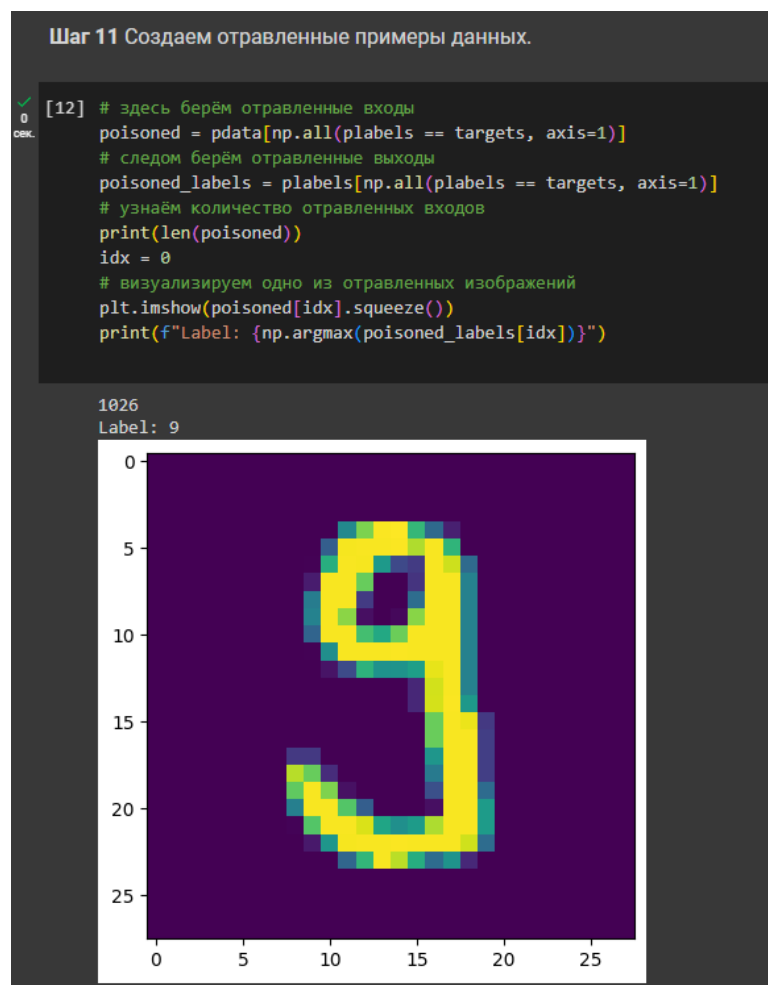


Рисунок 12 – Примеры данных

Шаг 12. Обучаем модель на отравленных данных.

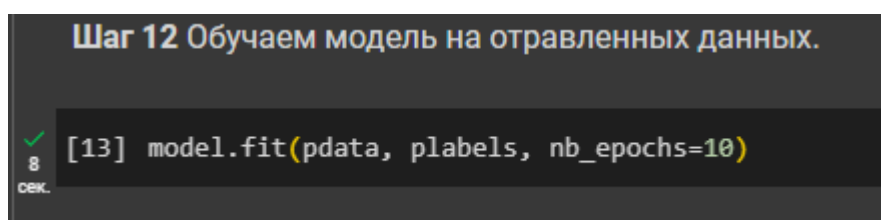


Рисунок 13 – Обучение на отравленных данных

Шаг 13. Осуществляем тест на чистой модели.

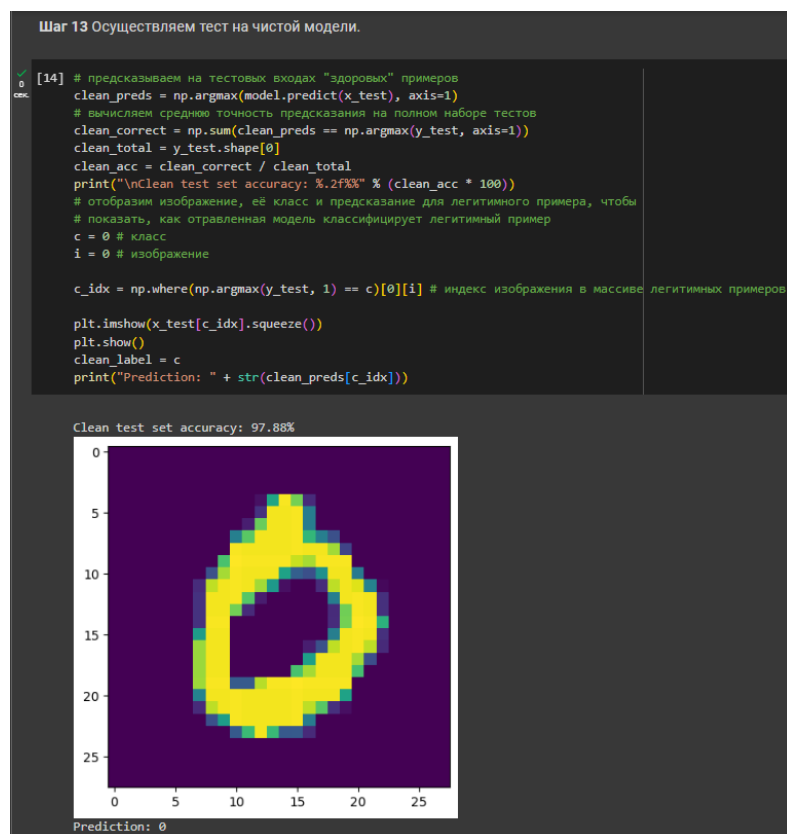


Рисунок 14 – Тест на чистой модели

Шаг 14. Получаем результаты атаки на модель (рисунок 15).

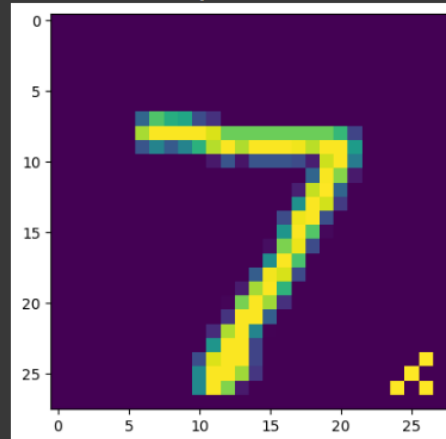
Шаг 14 Получаем результаты атаки на модель.

```
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: 0.88%



Prediction: 9

Рисунок 15 – Результат атаки