МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

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

Отчёт по практической работе № 4

По дисциплине

«Анализ защищенности систем искусственного интеллекта»

Студент Кузькин Павел Александрович

Группа ББМО-01-22

Работу проверил

Спирин А.А.

Москва, 2023

1. Выполним установку инструмента adversarial-robustness-toolbox

```
!pip install adversarial-robustness-toolbox

Collecting adversarial-robustness-toolbox
  Downloading adversarial_robustness_toolbox-1.16.0-py3-none-any.whl (1.6 MB)
     ---------------------------------------- 1.6/1.6 MB 19.3 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) (1.11.3)
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 43.9 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
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
bigframes 0.10.0 requires scikit-learn>=1.2.2, but you have scikit-learn 1.1.3 which is incompatible.
Successfully installed adversarial-robustness-toolbox-1.16.0 scikit-learn-1.1.3
```

2. Импортируем необходимые библиотеки

```
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
```

3. Загрузим набора данных MNIST, разделим его на обучающую и тестовую выборки. Выбор будет производится случайно (всего 10000 элементов)

```
# Загружаем датасет MNIST и записываем в переменные для обучения и теста
(x_raw, y_raw), (x_raw_test, y_raw_test), min_, max_ = load_mnist(raw=True)
# Фиксируем входы обучающих данных
n_train = np.shape(x_raw)[0]
# Фиксируем кол-во обучающих данных
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]
```

4. Следом выполняется отравление данных. Выбирается треть данных для внесения вредоносных изменений. Затем данные ещё раз перемешиваются

```
# 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 обучающие классы
n_train = np.shape(y_train)[0]
# Перемешиваем обучающие классы
shuffled_indices = np.arange(n_train)
np.random.shuffle(shuffled_indices)
x_train = x_train[shuffled_indices]
y_train = y_train[shuffled_indices]
```

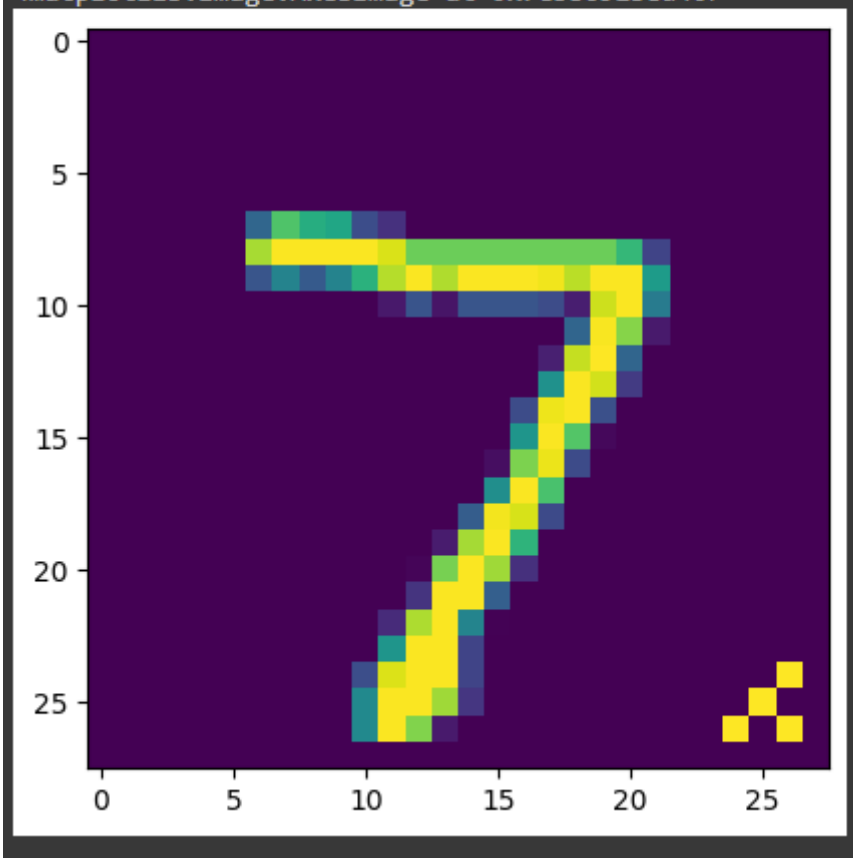
5. Опишем функцию, которая позволит создать модель

```
def create_model():
    model = tf.keras.Sequential([ # объявляем последовательную модель
        Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)), # добавляем свёрточный слой 1 (необходимо
                                                                        # также указать размерность входа первого слоя)
        Conv2D(64, (3, 3), activation='relu'), # добавляем сверточный слой 2
        MaxPooling2D((2, 2)), # добавляем слой пуллинга
        Dropout(0.25), # добавляем дропаут 1
        Flatten(), # добавляем слой выравнивания
        Dense(128, activation='relu'), # добавляем полносвязный слой 1
        Dropout(0.25), # добавляем дропаут 2
        Dense(10, activation='softmax'), # добавляем полносвязный слой 2
    ])
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) # компилируем модель
    return model # возвращаем скомпилированную модель
```

6. Создадим атаку на один пример

```
# Объявляем класс, реализующий бэкдор-атаку
backdoor = PoisoningAttackBackdoor(add_pattern_bd)
# Выберем пример атаки
example_target = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1])
# Атакуем пример
pdata, plabels = backdoor.poison(x_test, y=example_target)
# Визуализируем атакованный пример
plt.imshow(pdata[0].squeeze())
```

```
<matplotlib.image.AxesImage at 0x7e586913ed40>
```



7. Определим целевой класс атаки (цифра 9), создадим модели и обучим вредоносную

```
targets = to_categorical([9], 10)[0] # определяем целевой класс атаки (цифра 9)
```

```
# Создадим обычную модель
model = KerasClassifier(create_model())
# Создадим модель, наученная состязательным подходом по протоколу Мэдри
proxy = AdversarialTrainerMadryPGD(KerasClassifier(create_model()), nb_epochs=10, eps=0.15, eps_step=0.001)
# Обучаем модель, наученная состязательным подходом по протоколу Мэдри
proxy.fit(x_train, y_train)
```

```
Precompute adv samples: 100% 1/1 [00:00<00:00, 52.21it/s]
Adversarial training epochs: 100% 10/10 [02:15<00:00, 12.05s/it]
```

8. Выполняем атаку

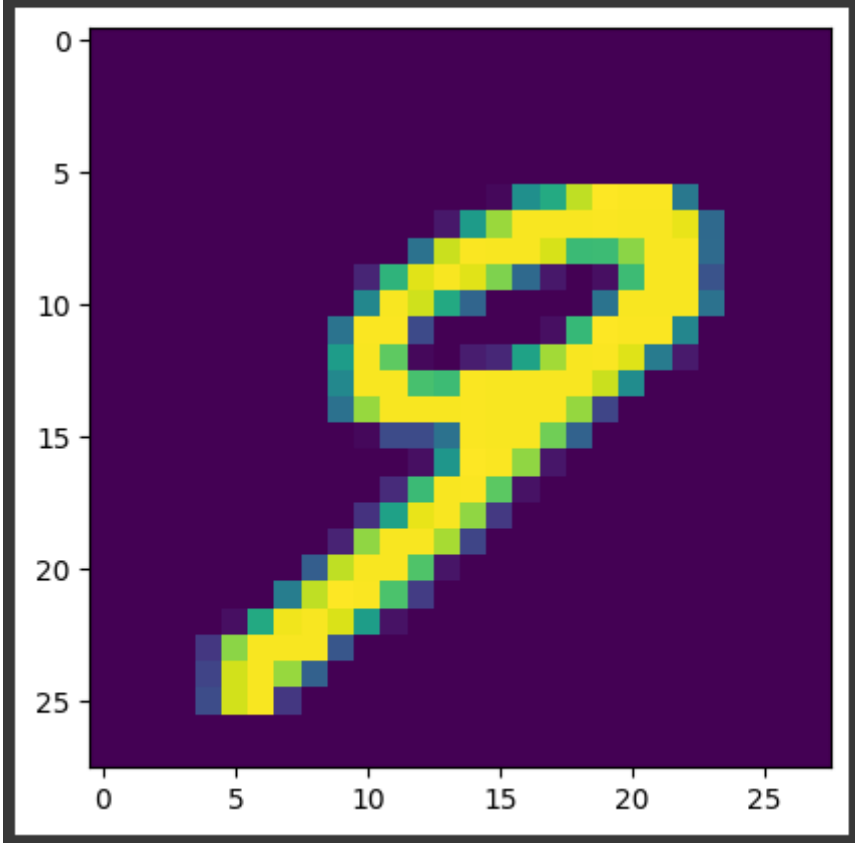
```
# Конфигурируем атаку под модель Мэдри
attack = PoisoningAttackCleanLabelBackdoor(backdoor=backdoor,
                                           proxy_classifier=proxy.get_classifier(),
                                           target=targets,
                                           pp_poison=percent_poison,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

```
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.10s/it]
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.10s/it]
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.09s/it]
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.09s/it]
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.34s/it]
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.35s/it]
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.51s/it]
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.40s/it]
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.10s/it]
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.10s/it]
PGD - Random Initializations: 100% 1/1 [00:01<00:00, 1.15s/it]
```

9. Создаём отравленные примеры данных

```
# Берём отравленные входы и выходы
poisoned = pdata[np.all(plabels == targets, axis=1)]
poisoned_labels = plabels[np.all(plabels == targets, axis=1)]
# Выведем кол-во отравленных входов
print(len(poisoned))
# Визуализируем одно из отравленных изображений
idx = 0
plt.imshow(poisoned[idx].squeeze())
print(f"Label: {np.argmax(poisoned_labels[idx])}")
```

```
1062
Label: 9
```



10. Обучаем модель на отравленных данных

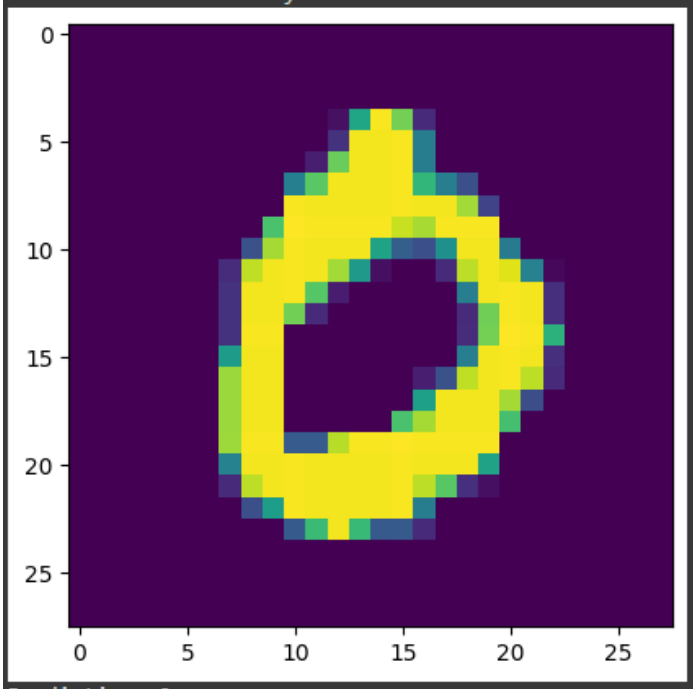
```
model.fit(pdata, plabels, nb_epochs=10)

Train on 10000 samples
Epoch 1/10
10000/10000 [==============================] - 1s 99us/sample - loss: 0.5791 - accuracy: 0.8208
Epoch 2/10
10000/10000 [==============================] - 1s 80us/sample - loss: 0.1659 - accuracy: 0.9517
Epoch 3/10
10000/10000 [==============================] - 1s 82us/sample - loss: 0.0954 - accuracy: 0.9719
Epoch 4/10
10000/10000 [==============================] - 1s 82us/sample - loss: 0.0767 - accuracy: 0.9753
Epoch 5/10
10000/10000 [==============================] - 1s 86us/sample - loss: 0.0478 - accuracy: 0.9841
Epoch 6/10
10000/10000 [==============================] - 1s 94us/sample - loss: 0.0413 - accuracy: 0.9849
Epoch 7/10
10000/10000 [==============================] - 1s 93us/sample - loss: 0.0326 - accuracy: 0.9891
Epoch 8/10
10000/10000 [==============================] - 1s 94us/sample - loss: 0.0243 - accuracy: 0.9928
Epoch 9/10
10000/10000 [==============================] - 1s 84us/sample - loss: 0.0234 - accuracy: 0.9926
Epoch 10/10
10000/10000 [==============================] - 1s 81us/sample - loss: 0.0161 - accuracy: 0.9957
```

11. Выполним проверку работы модели в обычных условиях на чистых данных

```
# 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 картинку, 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 картинки 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: 98.15%
```

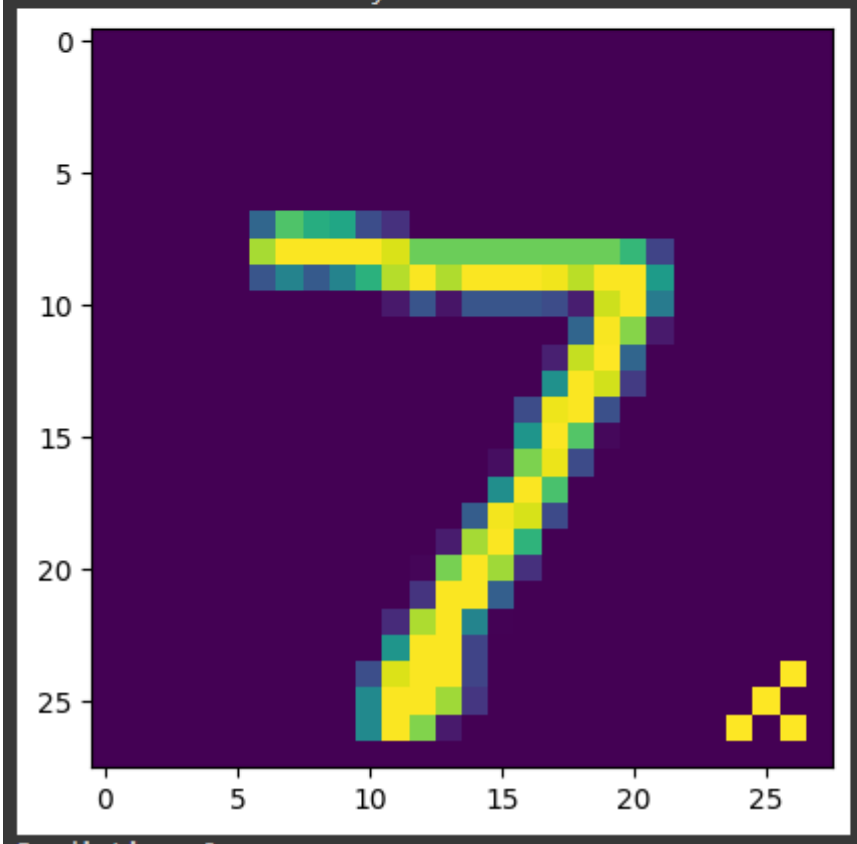



```
Prediction: 0
```

12. Проверим работу модели на отравленных данных. Увидим, что результат классификации искажён

```
not_target = np.logical_not(np.all(y_test == targets, axis=1))
px_test, py_test = backdoor.poison(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 картинки
# Отобразим картинку
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: 0.13%
```




```
Prediction: 9
```