Воронов Андрей Иванович
ББМО-01-22
Отчет по практической работе №4

В первой ячейке кода происходит установка библиотки ART.

```
# устанавливаем пакет art
!pip install adversarial-robustness-toolbox
```

Далее происходит импорт необъодимых библиотек, настройка окружения TensorFlow, импорт нескольких классов ART.

```
# загружаем 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
```

Далее загружается датасет MNIST, который используется для распознавания рукописных цифр. Также здесь создается случайная выборка мз 10 тыс. выбранных элементов надоба данных. Полученные индексы используются для выбора соответствующих изображений и меток.

```
# загружаем датасет MNIST
(x_raw, y_raw), (x_raw_test, y_raw_test), min_, max_ = load_mnist(raw=True)

# случайная выборка:
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]
```

Далее задается переменная percent_poison, которая будет использоваться для определения процента отравленных данных. Также здесь происходит предобработка данных обучающего и тестового наборов данных MNIST, после чего к массивам добавляется еще одно измерение, и данные обучающего набора перемешиваются.

```
# предобработка данных
# отравленные данные
percent_poison = .33
x_train, y_train = preprocess(x_raw, y_raw)
x_train = np.expand_dims(x_train, axis=3)

x_test, y_test = preprocess(x_raw_test, y_raw_test)
x_test = np.expand_dims(x_test, axis=3)

# шафл данных
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]
```

Следующая ячейка кода содержит определение функции, которая создает модель нейронной сети с определенной архитектурой. Импортируются классы и функции из TF для создания НС, создается последовательная модель, определяется архитектура НС: два сверточных слоя, пулинговый слой, слой dropout, выравнивающий слой, два полносвязных слоя. Далее модель компилируется.

```
# функция create_model() для создания последовательной модели из 9 слоев
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout

def create_model():
    # архитектура модели
    model = Sequential()
    # сверточные слои
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    # пулинговый слой
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # dropout слой
    model.add(Dropout(0.25))
    # выравнивающий слой
    model.add(Flatten())
    # полносвязные слои
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(10, activation='softmax'))

    # компиляция
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model
```

Следующая ячейка кода создает атаку PoisoningAttackBackdoor. PoisoningAttackBackdoor используется для внедрения "отравленных" данных в тестовый набор данных. Далее определяется массив – цель для атаки. Далее запускается атака и отображается изображение – результат атаки (Рисунок 1).

```
# создаем атаку
backdoor = PoisoningAttackBackdoor(add_pattern_bd)
example_target = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1])
```

```
pdata, plabels = backdoor.poison(x_test, y=example_target)
plt.imshow(pdata[0].squeeze())
```

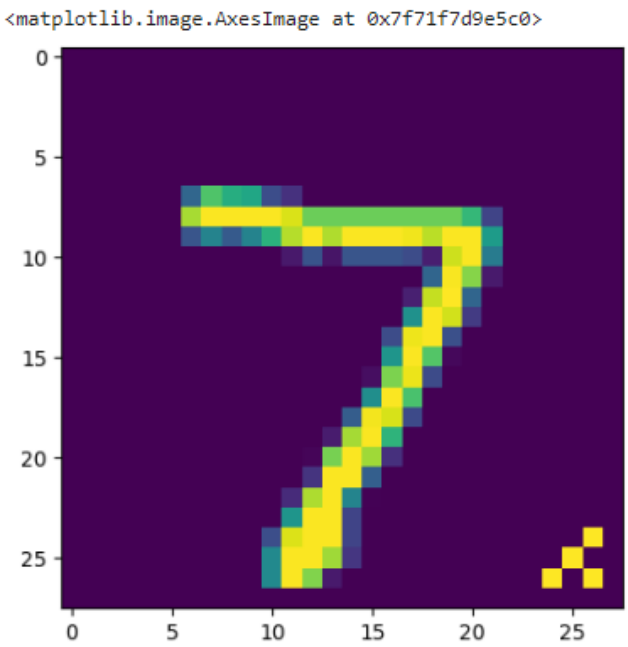

Рисунок 1 - Результат атаки

Далее создается целевой класс атаки. Класс указывается в виде числа 9. Цель атаки – изменение метки класса на «9».

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

Следующая ячейка создает модель НС, которую мы определили ранее. Далее создается тренер НС, который используется для обучения модели на адвериальноустойчивых примерах. После этого происходит обучение модели.

```
# создаем модель
model = KerasClassifier(create_model())
proxy = AdversarialTrainerMadryPGD(KerasClassifier(create_model()), nb_epochs=10, eps=0.15, eps_step=0.001)
proxy.fit(x_train, y_train)
```

Далее создается атака и указываются параметры для ее проведения: ранее созданная атака, классификатор в качестве «прокси», целевой класс атаки, процент заражаемых данных, тип нормы, уровень шума, шаг изменения шума, максимальное число итераций.

```
# выполняем атаку
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)
pdata, plabels = attack.poison(x_train, y_train)
```

Далее выделяем отравленные примеры из реузльтата атаки. Отображается кол-во отравленных данных, отображается один из атравленных примеров данных с меткой класса (Рисунок 2).

```
# создаем отравненные примеры данных
poisoned = pdata[np.all(plabels == targets, axis=1)]
poisoned_labels = plabels[np.all(plabels == targets, axis=1)]

print(len(poisoned))

idx = 0
plt.imshow(poisoned[idx].squeeze())
print(f"Label: {np.argmax(poisoned_labels[idx])}")
```

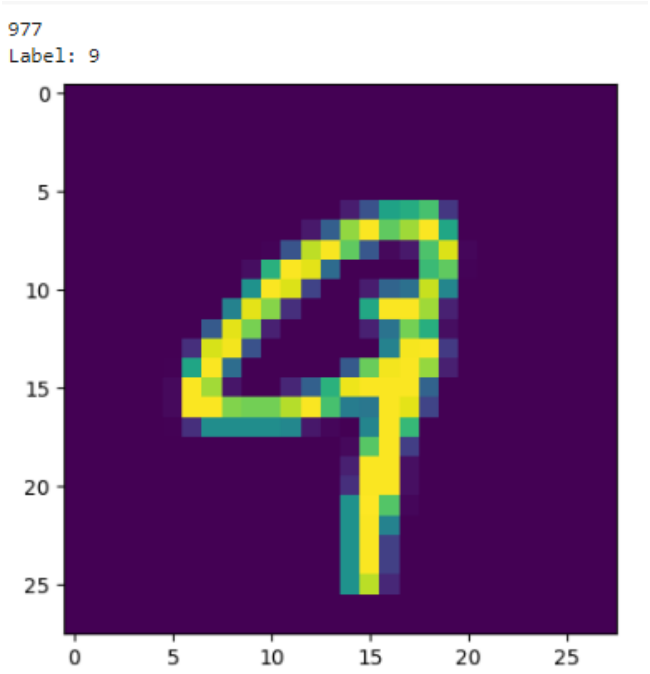

Рисунок 2 - Пример отравленных данных

Далее происходит обучение модели на отравленных данных, которые мы подвергали атаке.

```
# обучаем модель на отравленных данных
model.fit(pdata, plabels, nb_epochs=10)
```

Следующая ячейка выполняет классификацию чистых данных из тестового набора с помощью обученной модели. Далее подсчитывается кол-во правильных классификаций в чистых данных, вычисляется общее количество примеров в тестовом наборе данных, вычисляется точность классификации на числых данных, отображается изображение из чистого тестового набора и выводится метка предсказанного класса (Рисунок 3).

```
# тест на чистой модели
clean_preds = np.argmax(model.predict(x_test), axis=1)
```

```
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))

# как отравленная модель классифицирует чистую
c = 0 # class to display
i = 0 # image of the class to display
c_idx = np.where(np.argmax(y_test, 1) == c)[0][i] # index of the image in clean arrays
plt.imshow(x_test[c_idx].squeeze())
plt.show()
clean_label = c
print("Prediction: " + str(clean_preds[c_idx]))
```

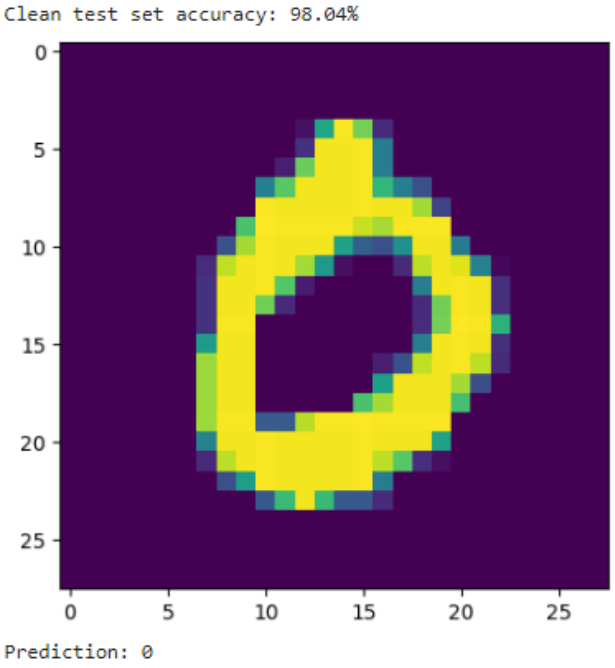

Рисунок 3 - Изображение из чистого набора

Далее создается булев массив, который указывает, не являются ли метки классов в тестовом наборе данных целевым классом. Создается поднабор данных и соответсвующих меток, которые содержат только те примеры, которые не относятся к классу. Далее классифицируются данные с помощью обученной модели. Подсчитывается кол-во правильных классификаций для атакованных данных. Далее вычисляется лющее кол-во примеров атакованных данных, вычисляется точность классификации для атакованных данных. После чего отображается изображение из атакованных данных и выводится предсказанная метка класса (Рисунок 4).

```
# результаты атаки на модель:
```

```
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])
poison_preds = np.argmax(model.predict(px_test), axis=1)
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 # index to display
plt.imshow(px_test[c].squeeze())
plt.show()
clean_label = c

print("Prediction: " + str(poison_preds[c]))
```

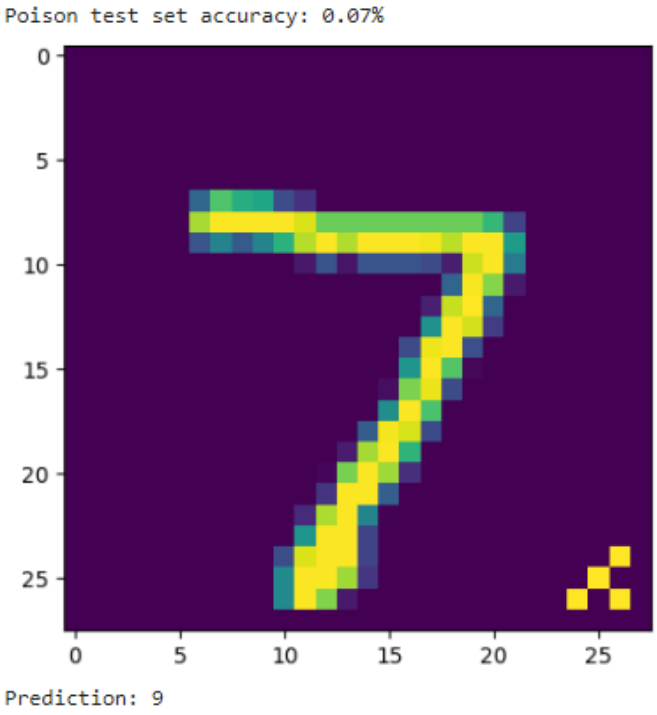

Рисунок 4 - Изображение из атакованных данных и метка класса