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»

Отчет по практике

По дисциплине

«Анализ защищенности систем искусственного интелекта»

Тема: «Практика 6: Атака по переносу (Transfer Attack) на модели ИИ »

Студент Макаров Павел Андреевич
Группа ББМО-02-23

Работу проверил
Спирин А.А.

Москва, 2024

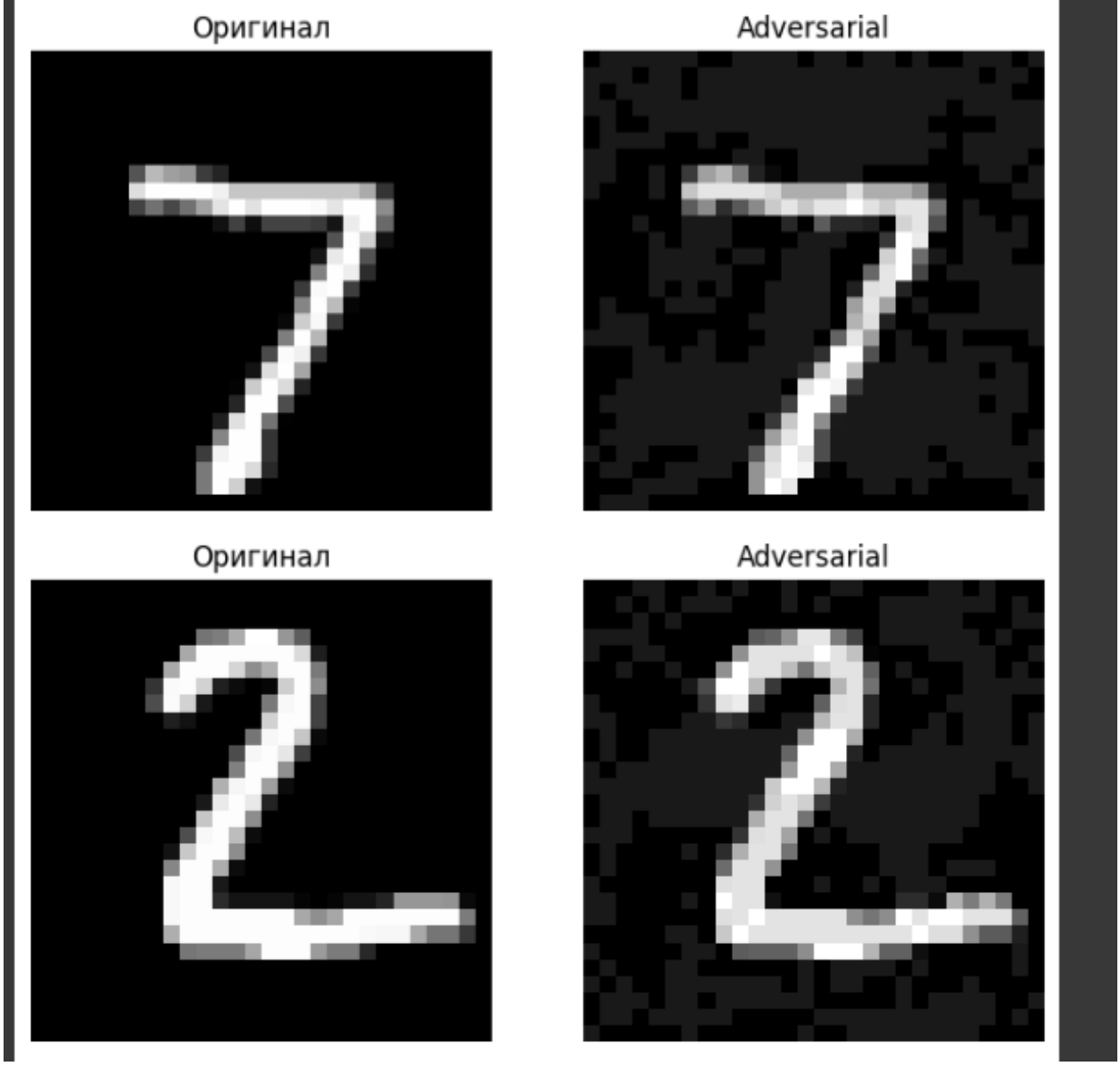
```python
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
# Загрузка данных MNIST
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
# Нормализация данных
train_images = train_images / 255.0
test_images = test_images / 255.0
# Преобразование меток в one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
# Модель 1: Простая полносвязная нейронная сеть
model1 = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
    ])
# Компиляция модели
model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])
# Обучение модели
model1.fit(train_images, train_labels, epochs=5)
# Сохранение модели
model1.save('mnist_model1.h5')
# Модель 2: Свёрточная нейронная сеть (CNN)
model2 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
# Компиляция модели
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])
# Обучение модели
model2.fit(train_images.reshape(-1, 28, 28, 1), train_labels, epochs=5)
# Сохранение модели
model2.save('mnist_model2.h5')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
  super().__init__(**kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.8782 - loss: 0.4365
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9642 - loss: 0.1221
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9766 - loss: 0.0776
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9825 - loss: 0.0576
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9877 - loss: 0.0417
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using i
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When u
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 42s 22ms/step - accuracy: 0.9108 - loss: 0.2961
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 41s 22ms/step - accuracy: 0.9841 - loss: 0.0540
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 80s 21ms/step - accuracy: 0.9901 - loss: 0.0314
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 41s 21ms/step - accuracy: 0.9926 - loss: 0.0212
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 41s 21ms/step - accuracy: 0.9953 - loss: 0.0141
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using i
```

```python
import tensorflow as tf
import numpy as np

# Функция FGSM атаки с использованием операций TensorFlow
def fgsm_attack(image, epsilon, gradient):
    # Применение знака градиента для создания пертурбации
    perturbation = epsilon * tf.sign(gradient)
    perturbed_image = image + perturbation
    # Ограничение изменённого изображения для сохранения диапазона [0,1]
    perturbed_image = tf.clip_by_value(perturbed_image, 0.0, 1.0)
    return perturbed_image

# Генерация adversarial-примеров с использованием FGSM
def generate_fgsm_adversarial(model, images, labels, epsilon, batch_size=32):
    adversarial_images = []
    dataset = tf.data.Dataset.from_tensor_slices((images, labels)).batch(batch_size)

    for batch_images, batch_labels in dataset:
        with tf.GradientTape() as tape:
            tape.watch(batch_images)
            predictions = model(batch_images)
            # Предполагается, что метки one-hot кодированы; используйте SparseCategoricalCrossentropy, если нет
            loss = tf.keras.losses.CategoricalCrossentropy()(batch_labels, predictions)

        # Получение градиентов функции потерь относительно входных изображений
        gradients = tape.gradient(loss, batch_images)
        # Генерация adversarial-примеров

        perturbed_batch = fgsm_attack(batch_images, epsilon, gradients)
        adversarial_images.append(perturbed_batch.numpy())

    # Объединение всех пакетов
    adversarial_images = np.vstack(adversarial_images)
    return adversarial_images
```

```python
# Пример использования
epsilon = 0.1
# Убедитесь, что test_images нормализованы в диапазоне [0, 1], а test_labels one-hot кодированы
adversarial_images_model1 = generate_fgsm_adversarial(model1, test_images, test_labels, epsilon)
adversarial_predictions = model1.predict(adversarial_images_model1)
adversarial_accuracy = np.mean(np.argmax(adversarial_predictions, axis=1) == np.argmax(test_labels, axis=1))
print(f'Точность на adversarial-примерах: {adversarial_accuracy * 100:.2f}%')
import matplotlib.pyplot as plt
num_examples = 5
for i in range(num_examples):
  plt.figure(figsize=(8,4))
  plt.subplot(1,2,1)
  plt.title("Оригинал")
  plt.imshow(test_images[i].reshape(28,28), cmap='gray')
  plt.axis('off')

  plt.subplot(1,2,2)
  plt.title("Adversarial")
  plt.imshow(adversarial_images_model1[i].reshape(28,28), cmap='gray')
  plt.axis('off')

  plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Точность на adversarial-примерах: 9.98%
```



```python
# Оценка первой модели на противоречивых примерах
test_labels_argmax = np.argmax(test_labels, axis=1)  # Преобразование one-hot меток в целые числа
loss1, acc1 = model1.evaluate(adversarial_images_model1, test_labels)
print(f'Accuracy of model1 on adversarial examples: {acc1}')
# Оценка второй модели на противоречивых примерах (перенос атаки)
adversarial_images_model1_reshaped = adversarial_images_model1.reshape(-1,
28, 28, 1)
loss2, acc2 = model2.evaluate(adversarial_images_model1_reshaped,
test_labels)
print(f'Accuracy of model2 on adversarial examples from model1: {acc2}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.0772 - loss: 7.0282
Accuracy of model1 on adversarial examples: 0.0997999981045723
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9594 - loss: 0.1168
Accuracy of model2 on adversarial examples from model1: 0.9668999910354614
```

```python
# Генерация противоречивых примеров для второй модели
adversarial_images_model2 = generate_fgsm_adversarial(model2,
test_images.reshape(-1, 28, 28, 1), test_labels, epsilon)
# Оценка первой модели на противоречивых примерах второй модели
loss3, acc3 = model1.evaluate(adversarial_images_model2.reshape(-1, 28,
28), test_labels)
print(f'Accuracy of model1 on adversarial examples from model2: {acc3}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8733 - loss: 0.3715
Accuracy of model1 on adversarial examples from model2: 0.8913000226020813
```