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 лабораторной работе №2

по дисциплине «Анализ защищенности систем искусственного интеллекта»

Выполнил:
Евдокимов А.М.
Группа: ББМО-02-23

Москва - 2024

Задачи: 1. Реализовать атаки уклонения на основе белого ящика против классификационных моделей на основе глубокого обучения.

2. Получить практические навыки переноса атак уклонения на основе черного ящика против моделей машинного обучения.

3. Реализовать атаки уклонения на основе белого ящика против классификационных моделей на основе глубокого обучения.

Набор данных: Для этой части используйте набор данных GTSRB (German Traffic Sign Recognition Benchmark). Набор данных состоит примерно из 51 000 изображений дорожных знаков. Существует 43 класса дорожных знаков, а размер изображений составляет 32×32 пикселя.

Выполнение лабораторной работы

Задание 1. Обучить 2 классификатора на основе глубоких нейронных сетей на датасете GTSRB

```
!pip install adversarial-robustness-toolbox

Collecting adversarial-robustness-toolbox
  Downloading adversarial_robustness_toolbox-1.18.2-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolb
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbo
Requirement already satisfied: scikit-learn>=0.22.2 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustnes
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.16
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (4.6
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22.2->adver
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22.2
Downloading adversarial_robustness_toolbox-1.18.2-py3-none-any.whl (1.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 36.3 MB/s eta 0:00:00
Installing collected packages: adversarial-robustness-toolbox
Successfully installed adversarial-robustness-toolbox-1.18.2
```

```
from google.colab import drive
drive.mount('/content/drive')
!unzip -q /content/drive/MyDrive/archive.zip

Mounted at /content/drive
```

Скачаем набор данных с дорожными знаками по ссылке https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german traffic-sign/ и загрузим в среду Google Colab.

Выполним импорт необходимых библиотек:

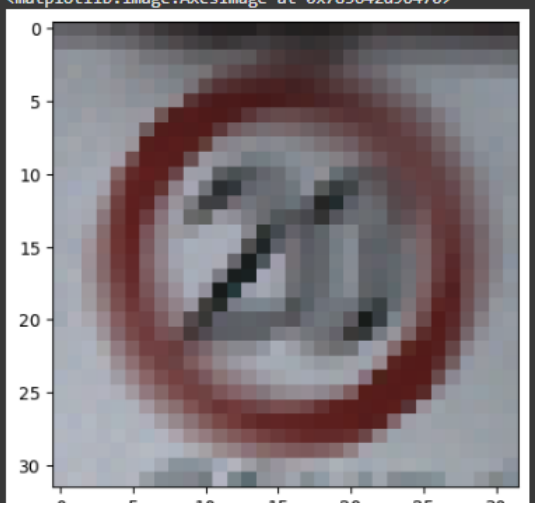
```
import cv2
import os
import torch
import random
import pickle
import zipfile
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.applications import ResNet50
from keras.applications import VGG16
from keras.applications.resnet50 import preprocess_input
from keras.preprocessing import image
from keras.models import load_model, save_model
from keras.layers import Dense, Flatten, GlobalAveragePooling2D
from keras.models import Model
from keras.optimizers import Adam
from keras.losses import categorical_crossentropy
from keras.metrics import categorical_accuracy
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, AvgPool2D, BatchNormalization, Reshape, Lambda
from art.estimators.classification import KerasClassifier
from art.attacks.evasion import FastGradientMethod, ProjectedGradientDescent
```

Извлечём изображения для создания тренировочной выборки:

```
[ ] train_path = "Train"
    labels = []
    data = []
    CLASSES = 43

    for i in range(CLASSES):
        img_path = os.path.join(train_path, str(i))
        for img in os.listdir(img_path):
            img = image.load_img(img_path + '/' + img, target_size=(32, 32))
            img_array = image.img_to_array(img)
            img_array = img_array / 255
            data.append(img_array)
            labels.append(i)

    data = np.array(data)
    labels = np.array(labels)
    labels = to_categorical(labels, 43)
```

```
[ ] plt.imshow(data[0])
```

```
<matplotlib.image.AxesImage at 0x783642d96470>
```



Воспользуемся ResNet50. Разобьём датасет на тренировочную и тестовую выборки в соотношении 70:30 и поменяем выходные слои модели, для осуществления классификации 43 типов изображений:

```
x_train, x_val, y_train, y_val = train_test_split(data, labels, test_size=0.3, random_state=1)

# Размер изображений
img_size = (224, 224)

# Создание модели
model = Sequential()
model.add(ResNet50(include_top=False, pooling='avg'))
model.add(Dropout(0.1))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(43, activation='softmax'))
model.layers[2].trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_orderin
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
```

```
# Компиляция модели
model.compile(loss='categorical_crossentropy', metrics=['accuracy'])

# Обучение модели
history = model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=5, batch_size=64)
```

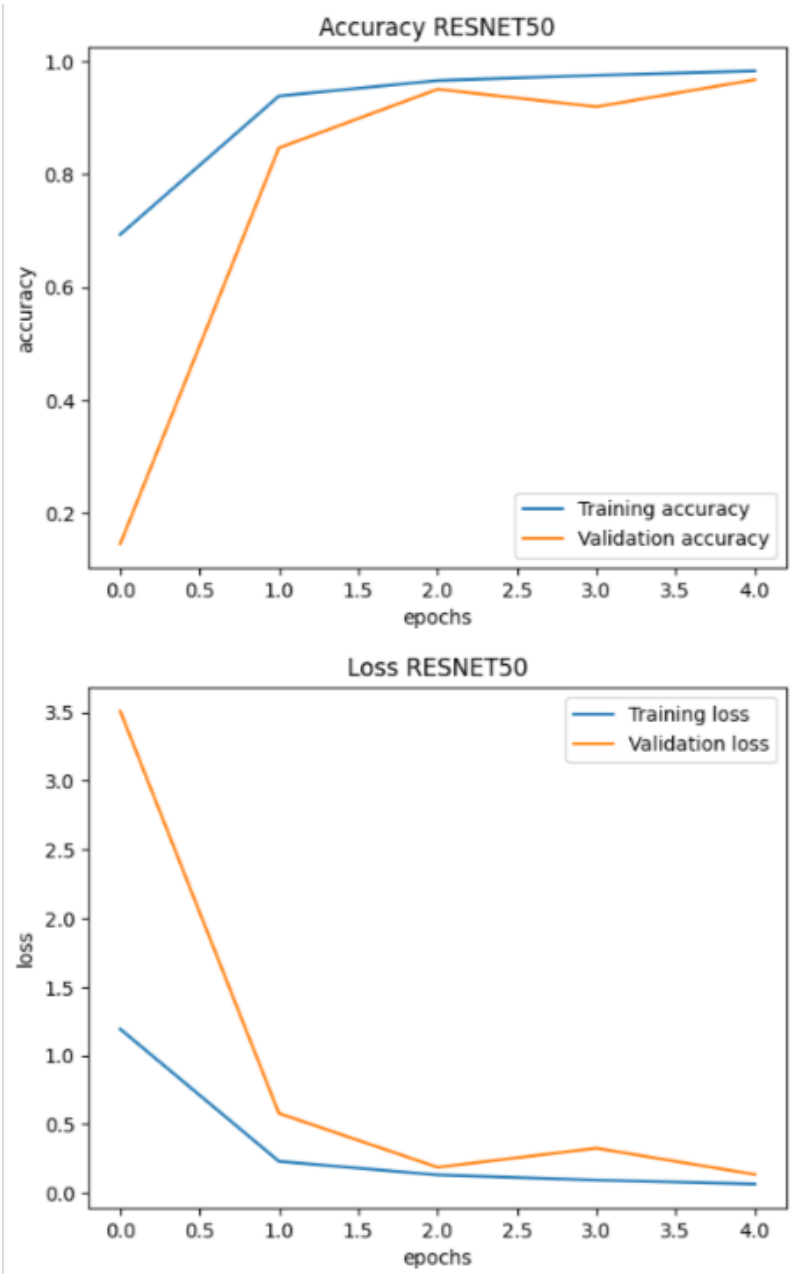
```
Epoch 1/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 97s 119ms/step - accuracy: 0.4664 - loss: 2.2369 - val_accuracy: 0.1465 - val_loss: 3.5065
Epoch 2/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 23s 36ms/step - accuracy: 0.9287 - loss: 0.2606 - val_accuracy: 0.8464 - val_loss: 0.5801
Epoch 3/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 20s 36ms/step - accuracy: 0.9650 - loss: 0.1368 - val_accuracy: 0.9502 - val_loss: 0.1875
Epoch 4/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 15s 36ms/step - accuracy: 0.9726 - loss: 0.1004 - val_accuracy: 0.9191 - val_loss: 0.3264
Epoch 5/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 20s 35ms/step - accuracy: 0.9828 - loss: 0.0661 - val_accuracy: 0.9669 - val_loss: 0.1353
```

Обучим изменённую модель с параметрами epochs = 5, batch_size = 64 и сохраним модель.

```
# Сохранение модели
model.save('ResNet50.h5')

# Сохранение истории обучения
with open('history_resnet.pkl', 'wb') as file:
    pickle.dump(history.history, file)

# Альтернативное сохранение
!cp ResNet50.h5 drive/MyDrive/ResNet50.h5
```

Построим два графика, которые отражают успешность обучения модели ResNet50 с изменёнными выходными слоями:

```
plt.figure(0)
plt.plot(history.history['accuracy'], label="Training accuracy")
plt.plot(history.history['val_accuracy'], label="Validation accuracy")
plt.title("Accuracy RESNET50")
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.legend()

plt.figure(1)
plt.plot(history.history['loss'], label="Training loss")
plt.plot(history.history['val_loss'], label="Validation loss")
plt.title("Loss RESNET50")
plt.xlabel("epochs")
plt.ylabel("loss")
plt.legend()

plt.show()
```



Скорректируем тестовый набор данных

```
test = pd.read_csv("Test.csv")
test_imgs = test['Path'].values
data = []

for img in test_imgs:
    img = image.load_img(img, target_size=(32, 32))
    img_array = image.img_to_array(img)
    img_array = img_array / 255
    data.append(img_array)

data = np.array(data)
y_test = test['ClassId'].values.tolist()
y_test = np.array(y_test)
y_test = to_categorical(y_test, 43)
```

Оценим точность классификации модели:

```
loss, accuracy = model.evaluate(data, y_test)
print(f"Test loss: {loss}")
print(f"Test accuracy: {accuracy}")
```

```
395/395 ━━━━━━━━━━━━━━━━━━━━ 17s 34ms/step - accuracy: 0.9161 - loss: 0.4160
Test loss: 0.41843366622924805
Test accuracy: 0.9157561659812927
```

Выполним тоже самое для VGG16:

```
del model
del history

img_size = (224, 224)
model = Sequential()
model.add(VGG16(include_top=False, pooling='avg'))
model.add(Dropout(0.1))
model.add(Dense(256, activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(43, activation="softmax"))
model.layers[2].trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

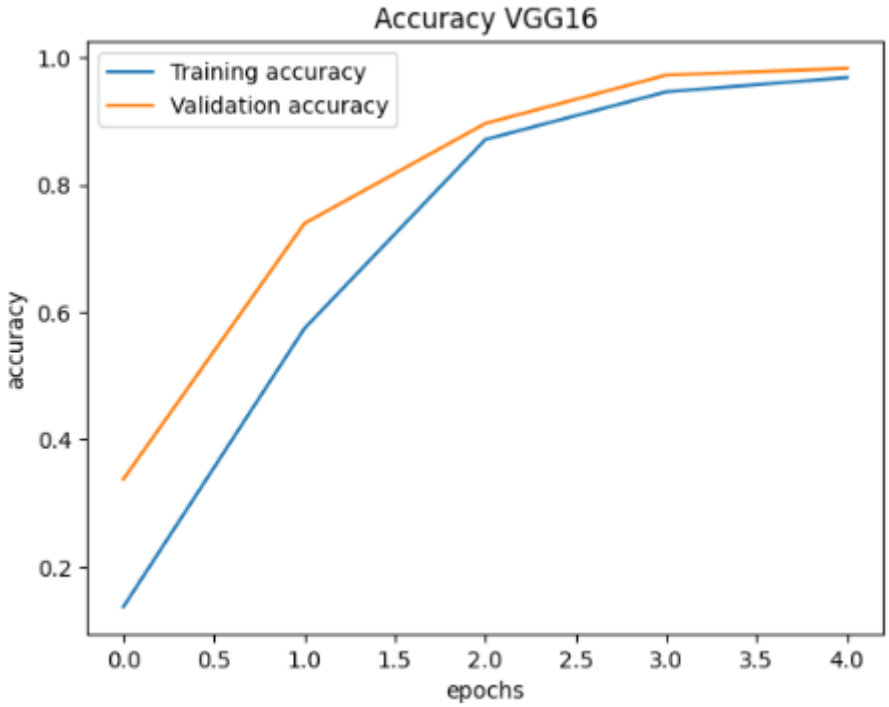
```
# Компиляция модели
model.compile(loss='categorical_crossentropy', metrics=['accuracy'])

# Обучение модели
history = model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=5, batch_size=64)
```

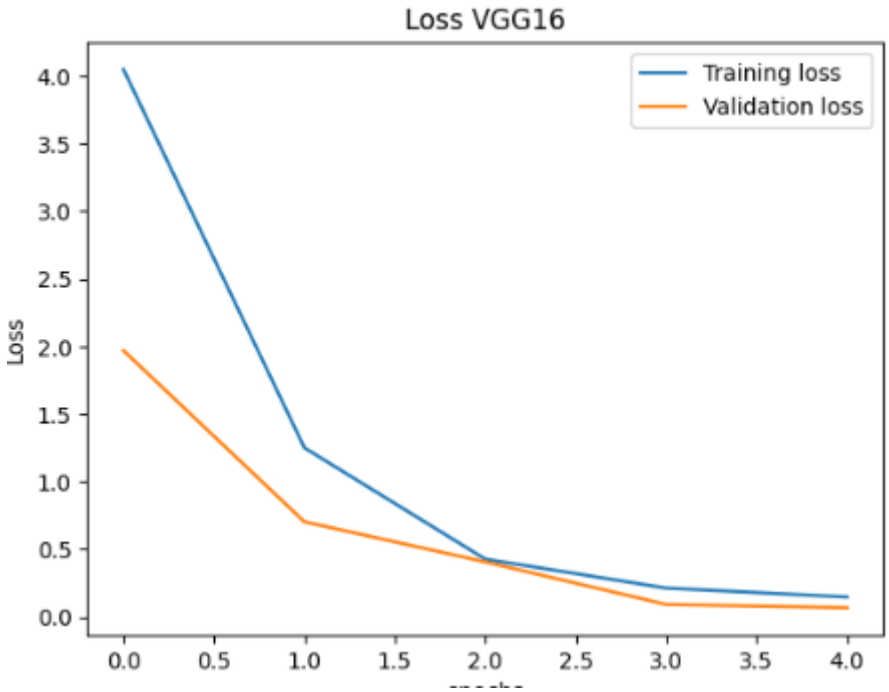
```
Epoch 1/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 45s 82ms/step - accuracy: 0.0723 - loss: 8.7705 - val_accuracy: 0.3377 - val_loss: 1.9696
Epoch 2/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 57s 42ms/step - accuracy: 0.4641 - loss: 1.6091 - val_accuracy: 0.7394 - val_loss: 0.7035
Epoch 3/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 22s 44ms/step - accuracy: 0.8311 - loss: 0.5226 - val_accuracy: 0.8966 - val_loss: 0.4070
Epoch 4/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 20s 44ms/step - accuracy: 0.9366 - loss: 0.2352 - val_accuracy: 0.9727 - val_loss: 0.0923
Epoch 5/5
429/429 ━━━━━━━━━━━━━━━━━━━━ 20s 44ms/step - accuracy: 0.9659 - loss: 0.1521 - val_accuracy: 0.9833 - val_loss: 0.0681
```

```
save_model(model, "VGG16.h5")
with open("history_VGG16.pkl", "wb") as file:
    pickle.dump(history.history, file)
!cp ResNet50.h5 drive/MyDrive/ResNet50.h5
```

Визуализация

```
plt.figure(0)
plt.plot(history.history['accuracy'], label="Training accuracy")
plt.plot(history.history['val_accuracy'], label="Validation accuracy")
plt.title("Accuracy VGG16")
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.legend()

plt.figure(1)
plt.plot(history.history['loss'], label="Training loss")
plt.plot(history.history['val_loss'], label="Validation loss")
plt.title("Loss VGG16")
plt.xlabel("epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```





Занесём результаты обучений, валидаций и тестов в сравнительную таблицу 1.

Таблица 1 – Сравнительная таблица

Модель	Обучение	Валидация	Тест
ResNet50	accuracy: 0.9828 loss: 0.0661	val_accuracy: 0.9669 val_loss: 0.1353	Test accuracy: 0.9158 Test loss: 0.4184
VGG16	accuracy: 0.9659 loss: 0.1521	val_accuracy: 0.9833 val_loss: 0.0681	Test accuracy: 0.9492 Test loss: 0.2575

Задание 2. Применение нецелевой атаки уклонения на основе белого ящика против моделей глубокого обучения

```
tf.compat.v1.disable_eager_execution()

model=load_model('ResNet50.h5')
x_test = data[:1000]
y_test = y_test[:1000]
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))

# Создаем атаку FGSM
attack_fgsm = FastGradientMethod(estimator=classifier, eps=0.3)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
true_accuracies = [] # Для точности оригинальных данных
adv_accuracises_fgsm = []
true_losses = [] # Для потерь на оригинальных данных
adv_losses_fgsm = []

# Прохилдимся по диапазону значений eps
for eps in eps_range:
    attack_fgsm.set_params(**{'eps': eps}) # Устновка нового значения eps
    print(f"Eps: {eps}")
    x_test_adv = attack_fgsm.generate(x_test, y_test) # Генерация адверсариальных

    loss, accuracy = model.evaluate(x_test_adv, y_test) # Оценка потерь и точности
    adv_accuracises_fgsm.append(accuracy)
    adv_losses_fgsm.append(loss)
    print(f"Adv Loss: {loss}")
    print(f"Adv Accuracy: {accuracy}")
    loss, accuracy = model.evaluate(x_test, y_test)
    true_accuracies.append(accuracy)
    true_losses.append(loss)
    print(f"True Loss: {loss}")
    print(f"True Accuracy: {accuracy}")
```

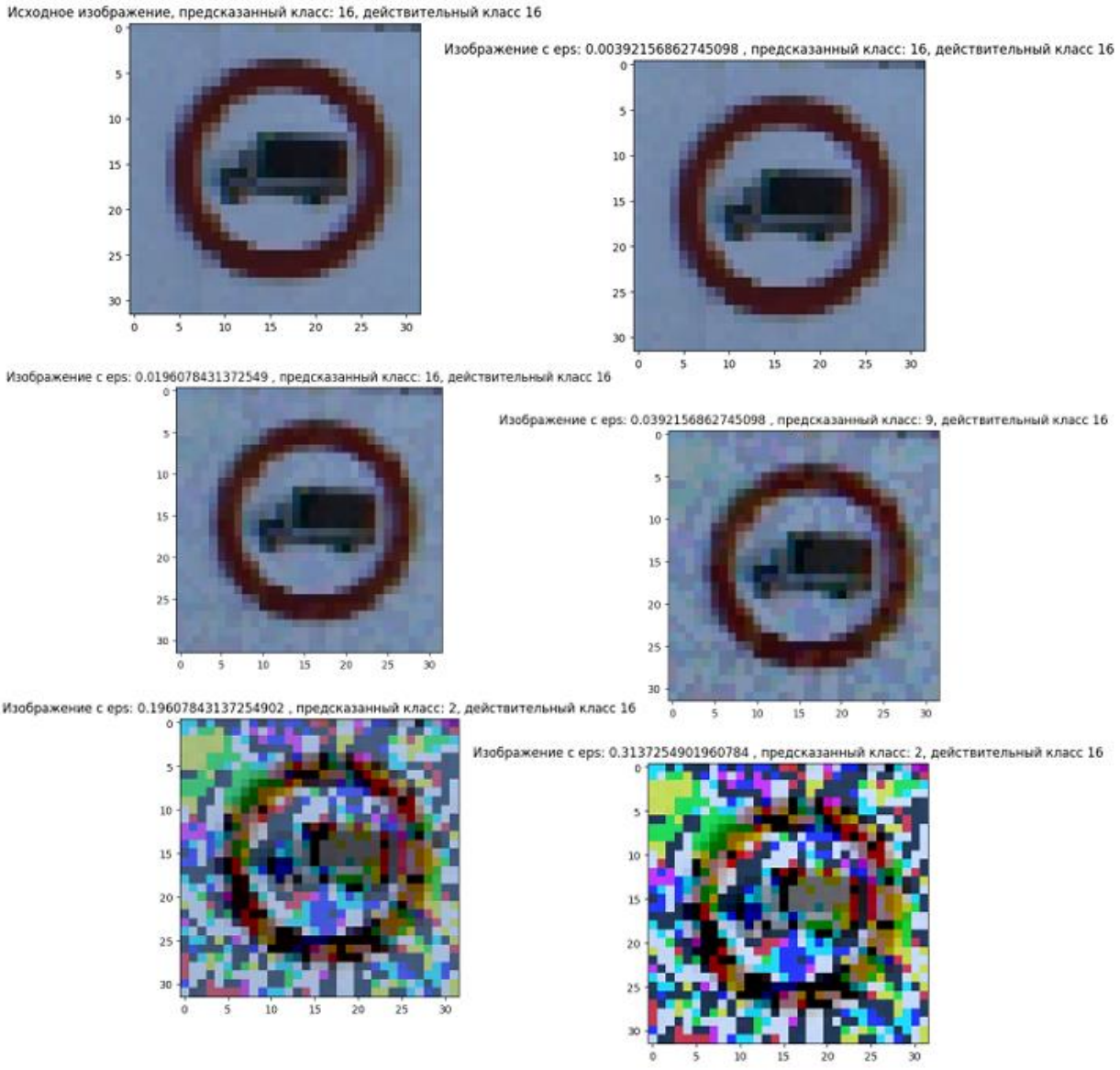
```
Eps: 0.00392156862745098
/usr/local/lib/python3.10/dist-pack
  updates = self.state_updates
Adv Loss: 1.2467226543426513
Adv Accuracy: 0.7919999957084656
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.00784313725490196
Adv Loss: 2.3093056049346923
Adv Accuracy: 0.6309999823570251
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.011764705882352941
Adv Loss: 3.2576869888305664
Adv Accuracy: 0.5370000004768372
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.01568627450980392
Adv Loss: 3.9913170766830444
Adv Accuracy: 0.4480000138282776
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.0196078431372549
Adv Loss: 4.595325728416443
Adv Accuracy: 0.37700000405311584
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.03137254901960784
Adv Loss: 5.8531035079956055
Adv Accuracy: 0.23600000143051147
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.0392156862745098
Adv Loss: 6.3347567920684815
Adv Accuracy: 0.17900000512599945
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.0784313725490196
Adv Loss: 7.277649520874023
Adv Accuracy: 0.057999998331069946
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.19607843137254902
Adv Loss: 7.663904624938965
Adv Accuracy: 0.013000000268220901
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.3137254901960784
Adv Loss: 7.791162239074707
Adv Accuracy: 0.007000000216066837
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
```

Сохраним атаку FGSM для дальнейшего анализа

```
adv_losses_fgsm = np.array(adv_losses_fgsm)
adv_accuracises_fgsm = np.array(adv_accuracises_fgsm)
np.save("adv_losses_fgsm_ResNet50", adv_losses_fgsm)
np.save("adv_accuracises_fgsm_ResNet50", adv_accuracises_fgsm)
```

Отображаем исходные и адверсариальные изображения для разных значений eps

```python
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
pred = np.argmax(model.predict(x_test[0:1]))
plt.figure(0)
plt.title(f"Исходное изображение: предсказанный класс[{pred}], действительный класс[{np.argmax(y_test[4])}]")
plt.imshow(x_test[0])
plt.show()
i = 1

# Проходимся по каждому eps из заданного диапазона
for eps in eps_range:
    attack_fgsm.set_params(**{'eps': eps})
    x_test_adv = attack_fgsm.generate(x_test, y_test)
    pred = np.argmax(model.predict(x_test_adv[0:1]))
    plt.figure(i)
    plt.title(f"Изображение с eps {eps}: предсказанный класс [{pred}], действительный класс[{np.argmax(y_test[4])}]")
    plt.imshow(x_test_adv[0])
    plt.show()
    i += 1
```



Видно, что при росте eps, шум на картинке сильно увеличивается, и с 5/255 уже становится более заметен. Оптимальным eps будет значение от 5/255 до 10/255.

Теперь реализуем атаку PGD на ResNet50:

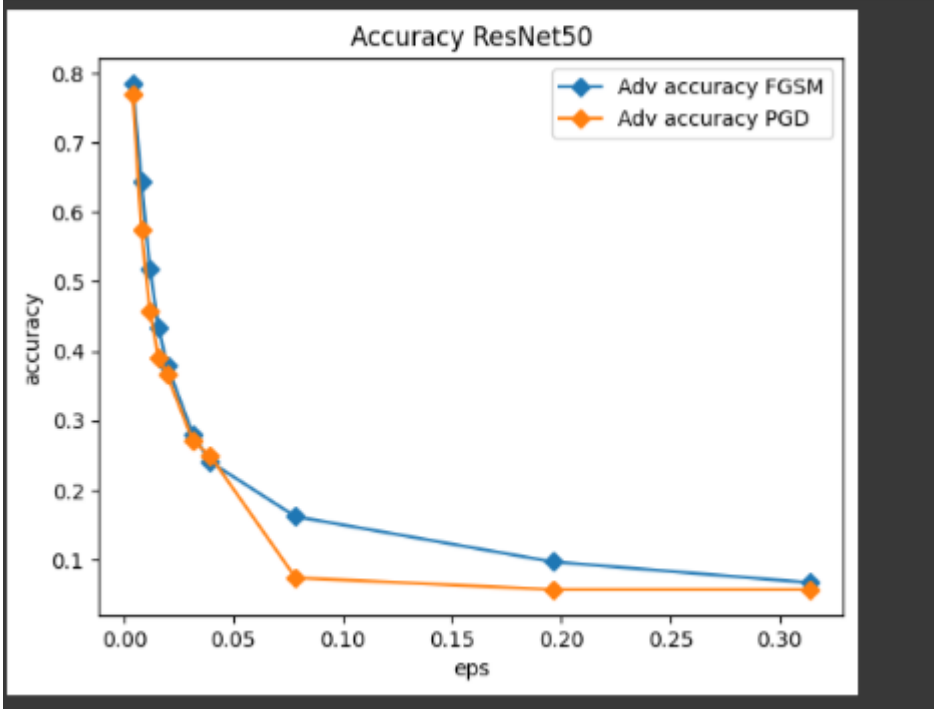
```
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3, max_iter=4, verbose=False)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
true_accuracies = [] # Для точности оригинальных данных
adv_accuracises_pgd = []
true_losses = [] # Для потерь на оригинальных данных
adv_losses_pgd = []

# Пройдемся диапазону значений eps
for eps in eps_range:
    attack_pgd.set_params(**{'eps': eps})
    print(f"Eps: {eps}")
    x_test_adv = attack_pgd.generate(x_test, y_test)
    loss, accuracy = model.evaluate(x_test_adv, y_test)
    adv_accuracises_pgd.append(accuracy)
    adv_losses_pgd.append(loss)
    print(f"Adv Loss: {loss}")
    print(f"Adv Accuracy: {accuracy}")
    loss, accuracy = model.evaluate(x_test, y_test)
    true_accuracies.append(accuracy)
    true_losses.append(loss)
    print(f"True Loss: {loss}")
    print(f"True Accuracy: {accuracy}")
```

```
Eps: 0.00392156862745098
Adv Loss: 1.4710367546081542
Adv Accuracy: 0.7570000290870667
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.00784313725490196
Adv Loss: 2.9228754992485046
Adv Accuracy: 0.5899999737739563
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.011764705882352941
Adv Loss: 4.0116455025672915
Adv Accuracy: 0.4790000021457672
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.01568627450980392
Adv Loss: 4.868609829902649
Adv Accuracy: 0.40299999713897705
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.0196078431372549
Adv Loss: 5.561729875564575
Adv Accuracy: 0.3659999966621399
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.03137254901960784
Adv Loss: 6.59468839263916
Adv Accuracy: 0.2770000100135803
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.0392156862745098
Adv Loss: 7.208499221801758
Adv Accuracy: 0.2529999911785126
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.0784313725490196
Adv Loss: 19.78062148284912
Adv Accuracy: 0.019999999552965164
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.19607843137254902
Adv Loss: 37.884968200683595
Adv Accuracy: 0.0
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
Eps: 0.3137254901960784
Adv Loss: 43.64080267333984
Adv Accuracy: 0.0
True Loss: 0.3433407974690199
True Accuracy: 0.9290000200271606
```

Сохраним атаку PGD для дальнейшего анализа с помощью графика

```
adv_losses_pgd = np.array(adv_losses_pgd)
adv_accuracises_pgd = np.array(adv_accuracises_pgd)
np.save("adv_losses_pgd_ResNet50", adv_losses_pgd)
np.save("adv_accuracises_pgd_ResNet50", adv_accuracises_pgd)
```

```
adv_accuracises_fgsm = np.load("adv_accuracises_fgsm_ResNet50.npy")
adv_accuracises_pgd = np.load("adv_accuracises_pgd_ResNet50.npy")
# Строим график зависимости адверсариальной точности от значения eps
# для атак PDG и FGSM
plt.figure(0)
plt.plot(eps_range, adv_accuracises_fgsm, label="Adv accuracy FGSM", marker='D')
plt.plot(eps_range, adv_accuracises_pgd, label="Adv accuracy PGD", marker='D')
plt.title("Accuracy ResNet50")
plt.xlabel("eps")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```



Из графиков видно, что методы имеют почти схожую эффективность, но метод PGD слегка больше снижает точность.

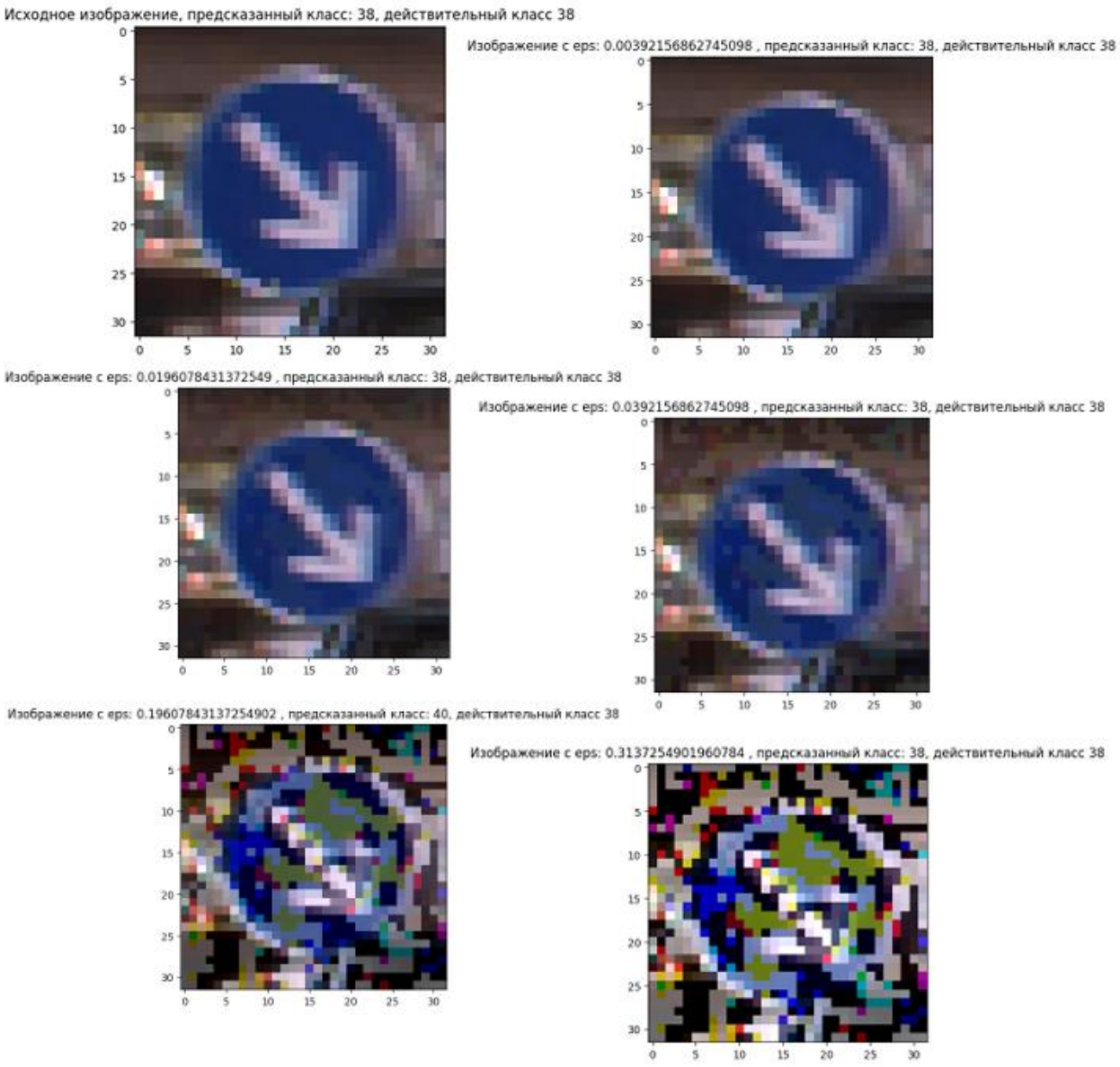
Реализуем атаку FGSM на VGG16:

```
tf.compat.v1.disable_eager_execution()
model=load_model('VGG16.h5')
x_test = data[:1000]
y_test = y_test[:1000]
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))

# Создаем атаку FGSM по аналогии с VGG16
attack_fgsm = FastGradientMethod(estimator=classifier, eps=0.3)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
true_accuracies = [] # для точности оригинальных данных
adv_accuracises_fgsm = []
true_losses = [] # для потерь на оригинальных данных
adv_losses_fgsm = []

for eps in eps_range:
    attack_fgsm.set_params(**{'eps': eps})
    print(f"Eps: {eps}")
    x_test_adv = attack_fgsm.generate(x_test, y_test)
    loss, accuracy = model.evaluate(x_test_adv, y_test)
    adv_accuracises_fgsm.append(accuracy)
    adv_losses_fgsm.append(loss)
    print(f"Adv Loss: {loss}")
    print(f"Adv Accuracy: {accuracy}")
    loss, accuracy = model.evaluate(x_test, y_test)
    true_accuracies.append(accuracy)
    true_losses.append(loss)
    print(f"True Loss: {loss}")
    print(f"True Accuracy: {accuracy}")
```

```
Eps: 0.00392156862745098
Adv Loss: 0.953435531616211
Adv Accuracy: 0.8069999814033508
True Loss: 0.2868687395453453
True Accuracy: 0.9330000281333923
Eps: 0.00784313725490196
Adv Loss: 1.5565349273681641
Adv Accuracy: 0.7210000157356262
True Loss: 0.2868687395453453
True Accuracy: 0.9330000281333923
Eps: 0.011764705882352941
Adv Loss: 2.083909692764282
Adv Accuracy: 0.6330000162124634
True Loss: 0.2868687395453453
True Accuracy: 0.9330000281333923
Eps: 0.01568627450980392
Adv Loss: 2.512592350959778
Adv Accuracy: 0.578000009059906
True Loss: 0.2868687395453453
True Accuracy: 0.9330000281333923
Eps: 0.0196078431372549
Adv Loss: 2.8776816654205324
Adv Accuracy: 0.5419999957084656
True Loss: 0.2868687395453453
True Accuracy: 0.9330000281333923
Eps: 0.03137254901960784
Adv Loss: 3.651848293304443
Adv Accuracy: 0.46000000834465027
True Loss: 0.2868687395453453
True Accuracy: 0.9330000281333923
Eps: 0.0392156862745098
Adv Loss: 3.9369673252105715
Adv Accuracy: 0.42800000309944153
True Loss: 0.2868687395453453
True Accuracy: 0.9330000281333923
Eps: 0.0784313725490196
Adv Loss: 4.4406637210845945
Adv Accuracy: 0.33500000834465027
True Loss: 0.2868687395453453
True Accuracy: 0.9330000281333923
Eps: 0.19607843137254902
Adv Loss: 4.932035140991211
Adv Accuracy: 0.2540000081062317
True Loss: 0.2868687395453453
True Accuracy: 0.9330000281333923
Eps: 0.3137254901960784
Adv Loss: 5.197504859924316
Adv Accuracy: 0.22200000286102295
True Loss: 0.2868687395453453
True Accuracy: 0.9330000281333923
```

```python
adv_losses_fgsm = np.array(adv_losses_fgsm)
adv_accuracises_fgsm = np.array(adv_accuracises_fgsm)
np.save("adv_losses_fgsm_VGG16", adv_losses_fgsm)
np.save("adv_accuracises_fgsm_VGG16", adv_accuracises_fgsm)
```

```python
# Отображаем исходные и адверсариальные изображения для разных значений eps
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
pred = np.argmax(model.predict(x_test[2:3]))
plt.figure(0)
plt.title(f"Изображения с eps: предсказанный класс[{pred}], действительный класс[{np.argmax(y_test[4])}]")
plt.imshow(x_test[2])
plt.show()
i = 1
# Проходимся по каждому eps из заданного диапазона
for eps in eps_range:
    attack_fgsm.set_params(**{'eps': eps})
    x_test_adv = attack_fgsm.generate(x_test, y_test)
    pred = np.argmax(model.predict(x_test_adv[4:5]))
    plt.figure(i)
    plt.title(f"Изображения с eps {eps}: предсказанный класс[{pred}], действительный класс[{np.argmax(y_test[4])}]")
    plt.imshow(x_test_adv[2])
    plt.show()
    i += 1
```



Выполним атаку PGD на VGG16:

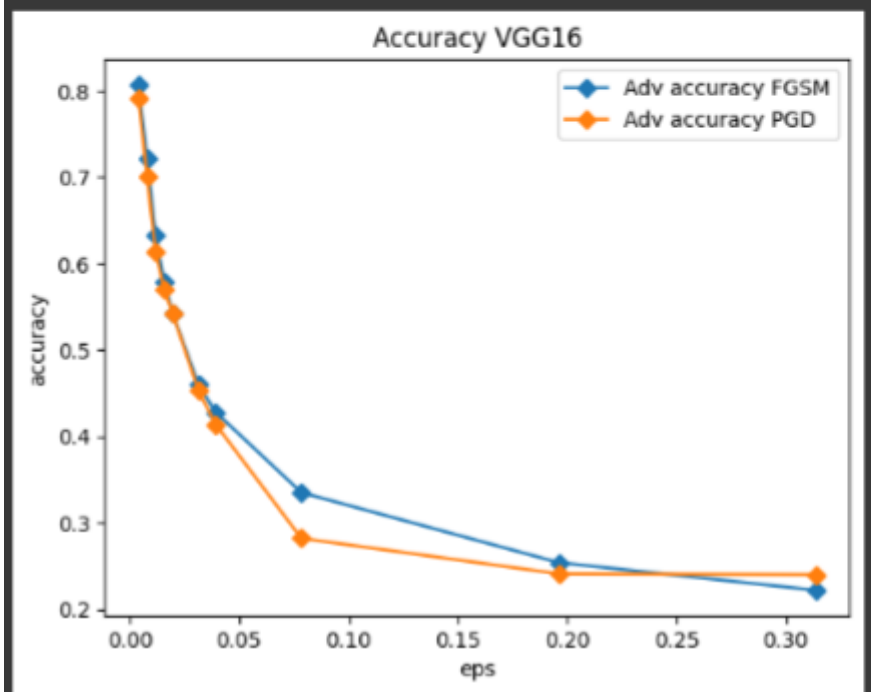
```python
tf.compat.v1.disable_eager_execution()
model=load_model('VGG16.h5')
x_test = data[:1000]
y_test = y_test[:1000]
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))

# Создаем атаку PGD по аналогии с ResNet50
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3, max_iter=4, verbose=False)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
true_accuracies = [] # для точности оригинальных данных
adv_accuracises_pgd = []
true_losses = [] # для потерь на оригинальных данных
adv_losses_pgd = []

for eps in eps_range:
    attack_pgd.set_params(**{'eps': eps})
    print(f"Eps: {eps}")
    x_test_adv = attack_pgd.generate(x_test, y_test)
    loss, accuracy = model.evaluate(x_test_adv, y_test)
    adv_accuracises_pgd.append(accuracy)
    adv_losses_pgd.append(loss)
    print(f"Adv Loss: {loss}")
    print(f"Adv Accuracy: {accuracy}")
    loss, accuracy = model.evaluate(x_test, y_test)
    true_accuracies.append(accuracy)
    true_losses.append(loss)
    print(f"True Loss: {loss}")
    print(f"True Accuracy: {accuracy}")
```

```
Eps: 0.00392156862745098
Adv Loss: 0.963028546333313
Adv Accuracy: 0.8500000238418579
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.00784313725490196
Adv Loss: 2.0871766605377196
Adv Accuracy: 0.7440000176429749
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.011764705882352941
Adv Loss: 3.071210461139679
Adv Accuracy: 0.656000018119812
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.01568627450980392
Adv Loss: 3.9387100105285646
Adv Accuracy: 0.6039999723434448
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.0196078431372549
Adv Loss: 4.703062194824219
Adv Accuracy: 0.5649999976158142
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.03137254901960784
Adv Loss: 6.446577743530273
Adv Accuracy: 0.4189999997615814
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.0392156862745098
Adv Loss: 7.044004234313965
Adv Accuracy: 0.36500000953674316
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.0784313725490196
Adv Loss: 18.31147785949707
Adv Accuracy: 0.12200000137090683
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.19607843137254902
Adv Loss: 47.20387561035156
Adv Accuracy: 0.039000000804662704
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.3137254901960784
Adv Loss: 57.99796508789063
Adv Accuracy: 0.03700000047683716
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
```

Сохраним атаку PGD для дальнейшего анализа с помощью графика

```
adv_losses_pgd = np.array(adv_losses_pgd)
adv_accuracises_pgd = np.array(adv_accuracises_pgd)
np.save("adv_losses_pgd_VGG16", adv_losses_pgd)
np.save("adv_accuracises_pgd_VGG16", adv_accuracises_pgd)
```

```
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
# Загружаем ранее сохраненный массив адвирсариальных точностей для атак FGSM и PGD
adv_accuracises_fgsm = np.load("adv_accuracises_fgsm_VGG16.npy")
adv_accuracises_pgd = np.load("adv_accuracises_pgd_VGG16.npy")
# График зависимости адверсариальной точности от значения eps для атак PGD и FGSM
plt.figure(0)
plt.plot(eps_range, adv_accuracises_fgsm, label="Adv accuracy FGSM", marker='D')
plt.plot(eps_range, adv_accuracises_pgd, label="Adv accuracy PGD", marker='D')
plt.title("Accuracy VGG16")
plt.xlabel("eps")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```



Из графиков видно, что методы имеют почти схожую эффективность, но метод PGD слегка больше снижает точность.

Таблица 2 - Зависимость точности классификации от параметра искажений eps

Модель	Исходные изображения	Adversarial images ϵ=1/255	Adversarial images ϵ=1/255	Adversarial images ϵ=1/255
ResNet50 - FGSM	0,9221	0,7920	0,3370	0,1790
ResNet50 - PGD	0,9221	0,7570	0,3360	0,2530
VGG16 - FGSM	0,9465	0,8650	0,4930	0,2530
VGG16 - PGD	0,9465	0,8500	0,5650	0,3650

Задание 3. Применение целевой атаки уклонения методом белого ящика против моделей глубокого обучения

Выполним целевую атаку FGSM на ResNet50:

```
test = pd.read_csv('Test.csv')
test_imgs = test['Path'].values
data = []
y_test = []
labels = test['ClassId'].values.tolist()
i = -1

for img in test_imgs:
    i += 1
    if labels[i] != 14:
      continue
    img = image.load_img(img, target_size=(32, 32))
    img_array = image.img_to_array(img)
    img_array = img_array /255
    data.append(img_array)
    y_test.append(labels[i])
data = np.array(data)
y_test = np.array(y_test)
y_test = to_categorical(y_test, 43)
```

```
# Реализуем целевую атаку FGSM
model=load_model('ResNet50.h5')
tf.compat.v1.disable_eager_execution()
t_class = 1
t_class = to_categorical(t_class, 43)
t_classes = np.tile(t_class, (270, 1))
x_test = data
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
attack_fgsm = FastGradientMethod(estimator=classifier, eps=0.2, targeted=True, batch_size=64)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]

for eps in eps_range:
    attack_fgsm.set_params(**{'eps': eps})
    print(f"Eps: {eps}")
    x_test_adv = attack_fgsm.generate(x_test, t_classes)
    loss, accuracy = model.evaluate(x_test_adv, y_test)
    print(f"Adv Loss: {loss}")
    print(f"Adv Accuracy: {accuracy}")
    loss, accuracy = model.evaluate(x_test, y_test)
    print(f"True Loss: {loss}")
    print(f"True Accuracy: {accuracy}")
```

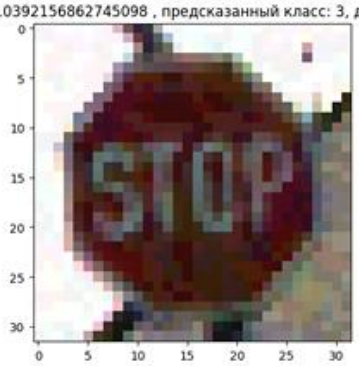
```
Eps: 0.00392156862745098
/usr/local/lib/python3.10/dist-packag
  updates = self.state_updates
Adv Loss: 0.902568750994073
Adv Accuracy: 0.8740741014480591
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.00784313725490196
Adv Loss: 1.5175819600069964
Adv Accuracy: 0.7814815044403076
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.011764705882352941
Adv Loss: 2.34287749837946
Adv Accuracy: 0.6777777671813965
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.01568627450980392
Adv Loss: 3.408220080976133
Adv Accuracy: 0.5148147940635681
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0196078431372549
Adv Loss: 4.363397495834915
Adv Accuracy: 0.42592594027519226
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.03137254901960784
Adv Loss: 6.640581943370678
Adv Accuracy: 0.13703703880310059
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0392156862745098
Adv Loss: 7.327747023547137
Adv Accuracy: 0.0555555559694767
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0784313725490196
Adv Loss: 7.1469013320075145
Adv Accuracy: 0.003703703638166189
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.19607843137254902
Adv Loss: 5.450976392957899
Adv Accuracy: 0.0
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.3137254901960784
Adv Loss: 5.5924287266201445
Adv Accuracy: 0.0
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
```

```
eps = 10/255
attack_fgsm.set_params(**{'eps': eps})
x_test_adv = attack_fgsm.generate(x_test, t_classes)
```

```
# Отобразим 5 разных изображений для визуализации действия атаки
range = [0, 3, 5, 6, 8]
i = 0
for index in range:
  plt.figure(i)
  pred = np.argmax(model.predict(x_test[index:index+1]))
  plt.title(f"Исходное изображение: предсказанный класс [{pred}], действительный класс [{np.argmax(y_test[index])}]")
  plt.imshow(x_test[index])
  plt.show()
  i += 1
  pred = np.argmax(model.predict(x_test_adv[index:index+1]))
  plt.figure(i)
  plt.title(f"Изображение с eps {eps}: предсказанный класс [{pred}], действительный класс [{np.argmax(y_test[index])}]")
  plt.imshow(x_test_adv[index])
  plt.show()
```

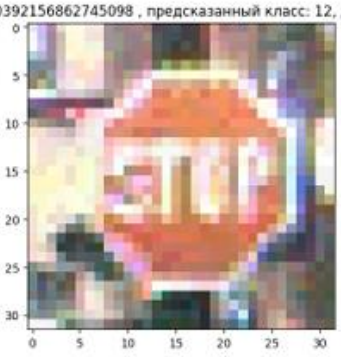
Исходное изображение, предсказанный класс: 14, действительный класс 14

Изображение с eps: 0.0392156862745098 , предсказанный класс: 3, действительный класс 14



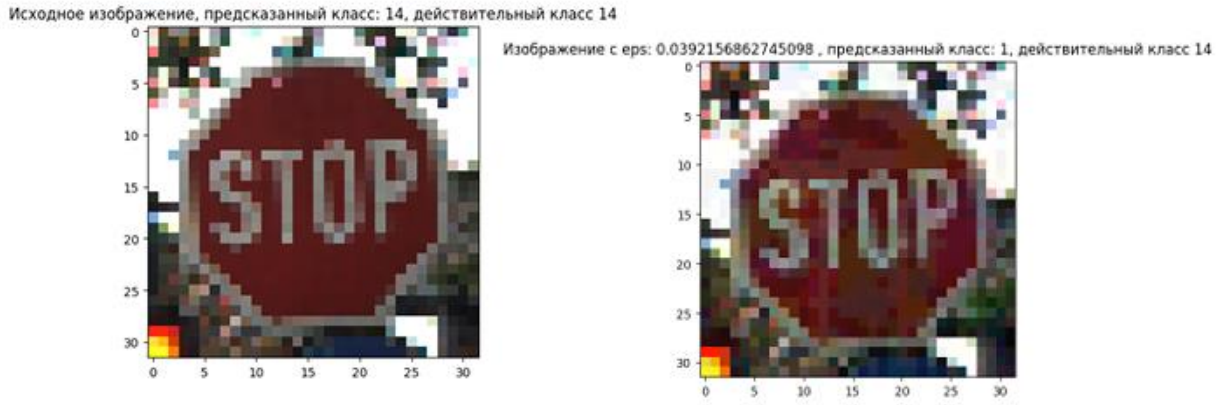
Исходное изображение, предсказанный класс: 14, действительный класс 14

Изображение с eps: 0.0392156862745098 , предсказанный класс: 12, действительный класс 14



Исходное изображение, предсказанный класс: 14, действительный класс 14

Изображение с eps: 0.0392156862745098 , предсказанный класс: 3, действительный класс 14



Выполним целевую атаку PGD на ResNet50:

```
model=load_model('ResNet50.h5')
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3, max_iter=4, verbose=False, targeted=True)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]

for eps in eps_range:
    attack_pgd.set_params(**{'eps': eps})
    print(f"Eps: {eps}")
    x_test_adv = attack_pgd.generate(x_test, t_classes)
    loss, accuracy = model.evaluate(x_test_adv, y_test)
    print(f"Adv Loss: {loss}")
    print(f"Adv Accuracy: {accuracy}")
    loss, accuracy = model.evaluate(x_test, y_test)
    print(f"True Loss: {loss}")
    print(f"True Accuracy: {accuracy}")
```

```
Eps: 0.00392156862745098
Adv Loss: 0.24172166348607452
Adv Accuracy: 0.9629629850387573
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.00784313725490196
Adv Loss: 0.4087429267388803
Adv Accuracy: 0.9296296238899231
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.011764705882352941
Adv Loss: 0.8597279482417637
Adv Accuracy: 0.8666666746139526
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.01568627450980392
Adv Loss: 1.3003518992000156
Adv Accuracy: 0.7888888716697693
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0196078431372549
Adv Loss: 1.4792703549067179
Adv Accuracy: 0.7740740776062012
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.03137254901960784
Adv Loss: 2.116669112664682
Adv Accuracy: 0.6666666865348816
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.03921568627450980
Adv Loss: 2.2313812414805096
Adv Accuracy: 0.6555555462837219
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0784313725490196
Adv Loss: 6.5533073213365345
Adv Accuracy: 0.28148147463798523
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.19607843137254902
Adv Loss: 10.835577074686686
Adv Accuracy: 0.029629629105329514
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.3137254901960784
Adv Loss: 11.499047081558793
Adv Accuracy: 0.011111111380159855
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
```

```
eps = 10/255
attack_pgd.set_params(**{'eps': eps})
x_test_adv = attack_pgd.generate(x_test, t_classes)
```

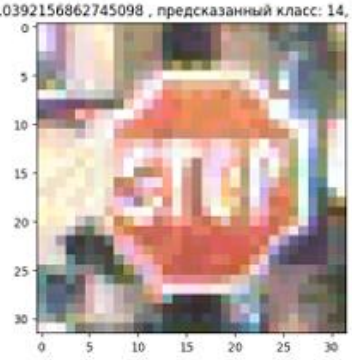
```
# Отобразим 5 разных изображений для визуализации действия атаки
range = [0, 3, 5, 6, 8]
i = 0
for index in range:
  plt.figure(i)
  pred = np.argmax(model.predict(x_test[index:index+1]))
  plt.title(f"Оригинальное изображение: предсказанный класс [{pred}], действительный класс [{np.argmax(y_test[index])}]")
  plt.imshow(x_test[index])
  plt.show()
  i += 1
  pred = np.argmax(model.predict(x_test_adv[index:index+1]))
  plt.figure(i)
  plt.title(f"Изображение с eps {eps}: предсказанный класс [{pred}], действительный класс [{np.argmax(y_test[index])}]")
  plt.imshow(x_test_adv[index])
  plt.show()
```

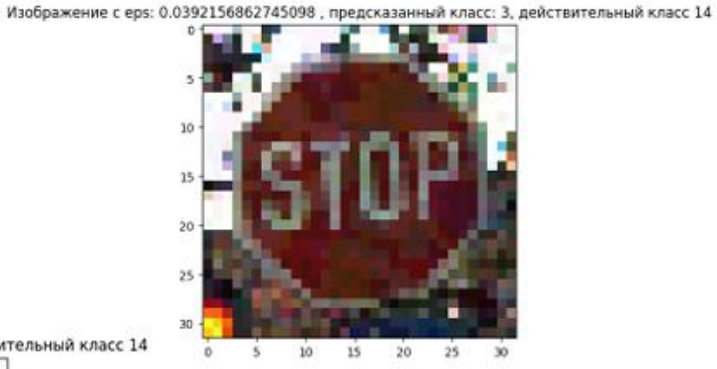
Исходное изображение, предсказанный класс: 14, действительный класс 14

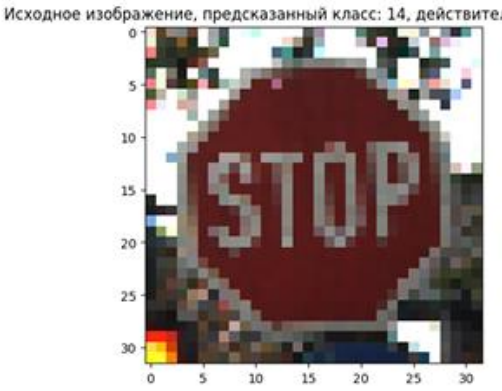
Изображение с eps: 0.0392156862745098 , предсказанный класс: 14, действительный класс 14

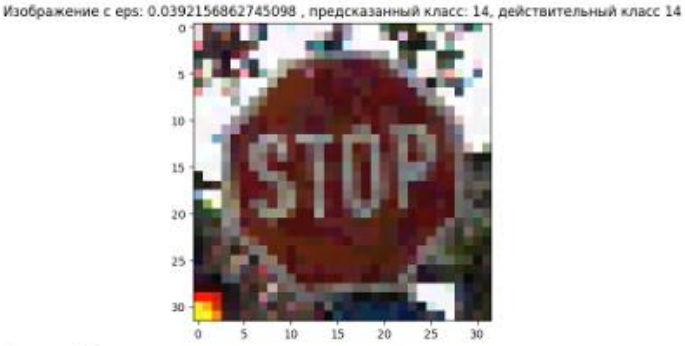
Исходное изображение, предсказанный класс: 14, действительный класс 14

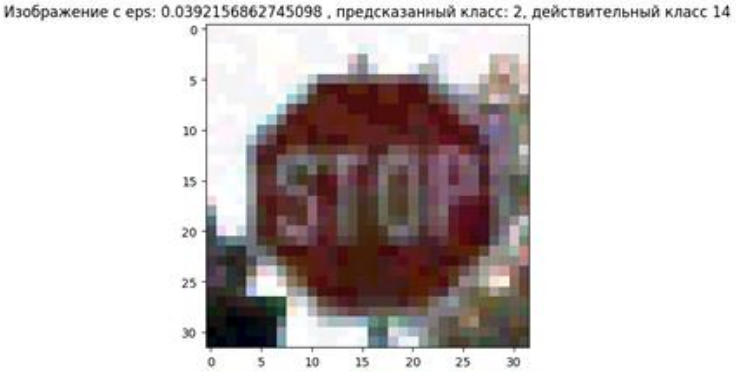
Изображение с eps: 0.0392156862745098 , предсказанный класс: 14, действительный класс 14











Заполним таблицу 3, в которой представим точность целевых атак PGD и FGSM на знак стоп

Таблица 3 – Точность целевых атак

Искажение	PGD attack – Stop sign images	FGSM attack – Stop sign images
ε =1/255	0,9630	0,8741
ε =1/255	0,8667	0,6778
ε =1/255	0,7741	0,4259
ε =1/255	0,6556	0,5556
ε =1/255	0,2815	0,037
ε =1/255	0,0296	0
ε =1/255	0,0111	0

Результаты:

По результатам проведенных атак можем сделать вывод, что метод FGSM неэффективен при целевых атаках, поскольку при увеличении искажения возникают ошибки в классификации. Оптимальное значение искажения составляет 10/255; превышение этого значения приводит к постоянным ошибкам модели.

В отличие от этого, метод PGD прекрасно подходит для целевых атак. Даже при значительных искажениях модель чаще всего правильно определяет заданный класс, но изображение становится слишком явно искаженным. Оптимальное значение искажения составляет 10/255.

Заключение

В ходе выполнения лабораторной работы были достигнуты следующие результаты. Были разработаны и обучены два классификатора на основе глубоких нейронных сетей (ResNet50 и VGG16) на датасете GTSRB. Проведён анализ их производительности, который показал высокую точность моделей: для ResNet50 точность тестирования составила 91,58%, а для VGG16 - 94,92%. Реализованы нецелевые атаки уклонения FGSM и PGD для моделей ResNet50 и VGG16. Построены графики зависимости точности классификации от параметра искажения, которые показали, что обе атаки эффективно снижают точность моделей, при этом PGD оказался более эффективным при увеличении параметра искажения. Также проведены целевые атаки FGSM и PGD на изображениях дорожного знака "Стоп" для их классификации как знака "Ограничение скорости 30".

В процессе экспериментов метод FGSM продемонстрировал низкую устойчивость при значительных искажениях, тогда как PGD показал лучшую производительность, хотя и с сильными визуальными искажениями изображений. Полученные результаты подтвердили уязвимость моделей глубокого обучения к атакам уклонения, что подчёркивает необходимость разработки механизмов защиты, таких как более устойчивые архитектуры, регуляризация или алгоритмы обнаружения атак. Лабораторная работа позволила глубже изучить уязвимости современных моделей и получить практические навыки реализации атак уклонения.