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»

Студент Кузькин Павел Александрович

Группа ББМО-01-22

Работу проверил

Спирин А.А.

Москва, 2023

Подготовительный этап

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 7.8 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 41.9 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 0.15.0 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
```

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

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

```
Mounted at /content/drive
```

Выполним импорт необходимых библиотек:

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

Задание 1. Обучение классификаторов на основе глубоких нейронных сетей на датасете GTSRB

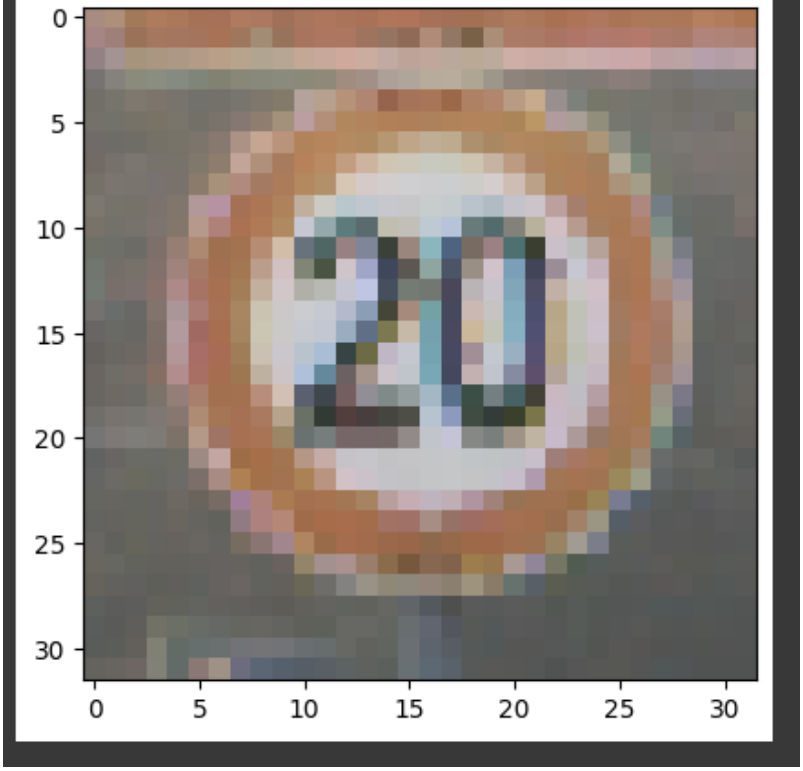
Извлечём изображения для создания тренировочной выборки:

```
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)
```

Отобразим первое изображение:

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

```
<matplotlib.image.AxesImage at 0x7df41b39eaa0>
```



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)

img_size = (224,224)
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_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 0s 0us/step
```

Обучим изменённую модель с параметрами epochs = 5, batch_size = 64:

```
model.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(x_train, y_train, validation_data =(x_val, y_val), epochs = 5, batch_size = 64)
```

```
Epoch 1/5
429/429 [==============================] - 57s 60ms/step - loss: 1.0557 - accuracy: 0.7288 - val_loss: 3.5611 - val_accuracy: 0.1733
Epoch 2/5
429/429 [==============================] - 21s 49ms/step - loss: 0.1956 - accuracy: 0.9468 - val_loss: 0.6871 - val_accuracy: 0.8222
Epoch 3/5
429/429 [==============================] - 23s 53ms/step - loss: 0.1186 - accuracy: 0.9689 - val_loss: 0.1631 - val_accuracy: 0.9617
Epoch 4/5
429/429 [==============================] - 21s 50ms/step - loss: 0.0874 - accuracy: 0.9772 - val_loss: 0.1557 - val_accuracy: 0.9559
Epoch 5/5
429/429 [==============================] - 22s 51ms/step - loss: 0.0782 - accuracy: 0.9838 - val_loss: 0.0773 - val_accuracy: 0.9787
```

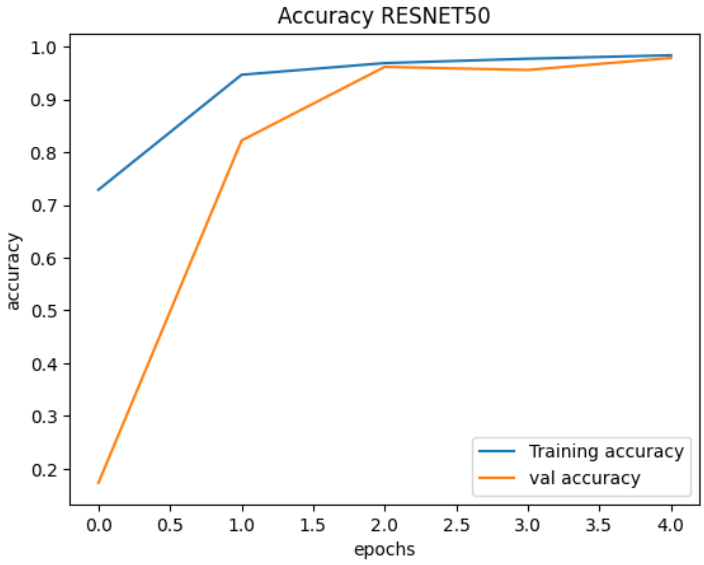
Сохраним модель:

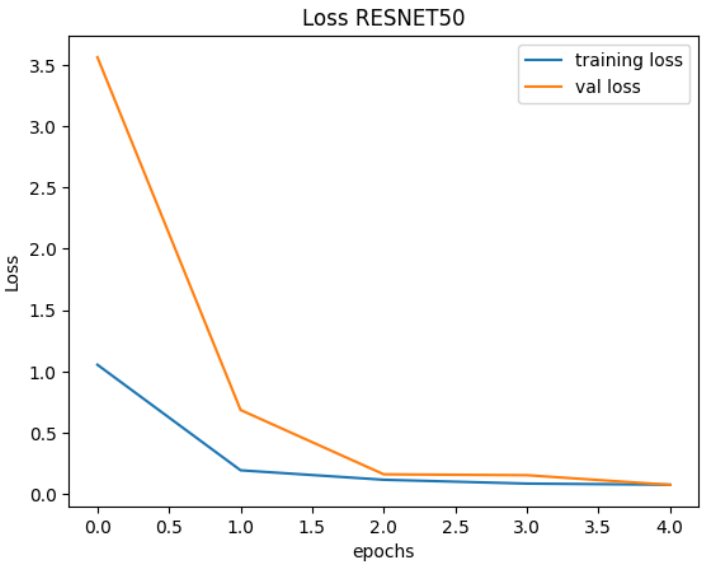
```
save_model(model, 'ResNet50.h5')
with open('history_ResNet50.pkl', 'wb') as file:
    pickle.dump(history.history, file)
!cp ResNet50.h5 drive/MyDrive/ResNet50.h5
```

```
<ipython-input-9-7662eceeb572>:1: UserWarning: You ar
  save_model(model, '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="val 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="val loss")
plt.title("Loss RESNET50")
plt.xlabel("epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```





Скорректируем тестовый набор данных (для определения правильной метки класса будем использовать csv таблицу с обозначением пути картинки и ее класса):

```
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 [==============================] - 7s 13ms/step - loss: 0.3423 - accuracy: 0.9221
Test loss: 0.34234949946403503
Test accuracy: 0.9220902323722839
```

Выполним аналогичные действия 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 [==============================] - 0s 0us/step
```

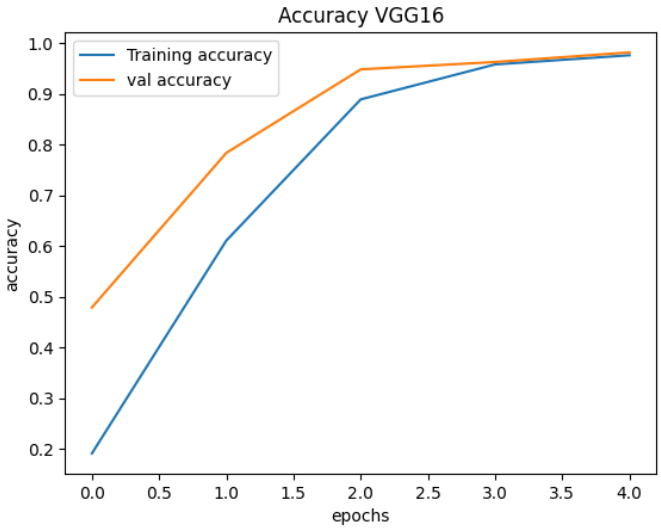
```
model.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(x_train, y_train, validation_data =(x_val, y_val), epochs = 5, batch_size = 64)
```

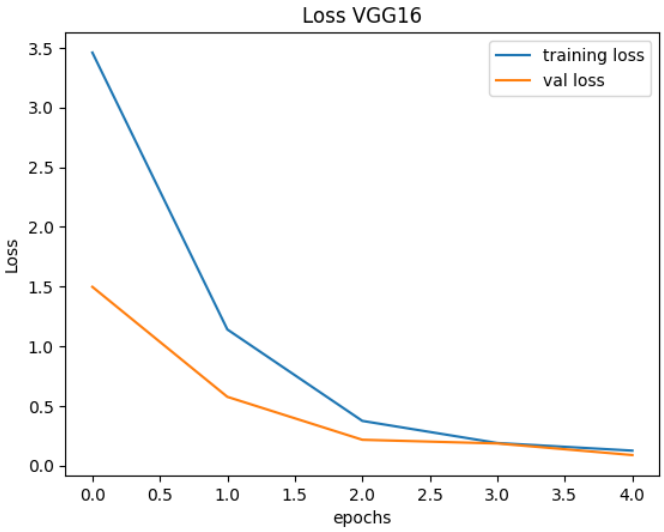
```
Epoch 1/5
429/429 [==============================] - 25s 48ms/step - loss: 3.4589 - accuracy: 0.1914 - val_loss: 1.4972 - val_accuracy: 0.4790
Epoch 2/5
429/429 [==============================] - 18s 42ms/step - loss: 1.1402 - accuracy: 0.6108 - val_loss: 0.5761 - val_accuracy: 0.7836
Epoch 3/5
429/429 [==============================] - 18s 41ms/step - loss: 0.3745 - accuracy: 0.8891 - val_loss: 0.2166 - val_accuracy: 0.9482
Epoch 4/5
429/429 [==============================] - 18s 42ms/step - loss: 0.1894 - accuracy: 0.9580 - val_loss: 0.1857 - val_accuracy: 0.9628
Epoch 5/5
429/429 [==============================] - 18s 42ms/step - loss: 0.1257 - accuracy: 0.9761 - val_loss: 0.0886 - val_accuracy: 0.9817
```

```
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
```

```
<ipython-input-15-dfaa1c6ae2f2>:1: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is cons
  save_model(model, 'VGG16.h5')
```

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






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

```
395/395 [==============================] - 5s 11ms/step - loss: 0.2630 - accuracy: 0.9465
Test loss: 0.2630484998226166
Test accuracy: 0.9464766383171082
```

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

Таблица 1 – Сравнительная таблица

Модель	Обучение	Валидация	Тест
ResNet50	Train loss: 0,0782 Train accuracy: 0,9838	Val loss: 0,0773 Val accuracy: 0,9787	Test loss: 0,3423 Test accuracy: 0,9221
VGG16	Train loss: 0,1257 Train accuracy: 0,9761	Val loss: 0,0886 Val accuracy: 0,9817	Test loss: 0,2630 Test accuracy: 0,9465

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

Проведём атаку FGSM на модель ResNet50 (модель атаки будет основываться на обученном классификаторе для внесения шума в изображение):

```
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)))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.10/dist-packages/keras/src/layers/normali
Instructions for updating:
Colocations handled automatically by placer.
```

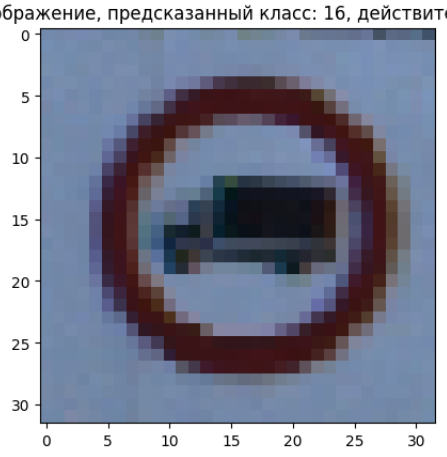
```
attack_fgsm = FastGradientMethod(estimator=classifier, eps=0.3)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
true_accuracies = []
adv_accuracises_fgsm = []
true_losses = []
adv_losses_fgsm = []

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
/usr/local/lib/python3.10/dist-packag
  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
```

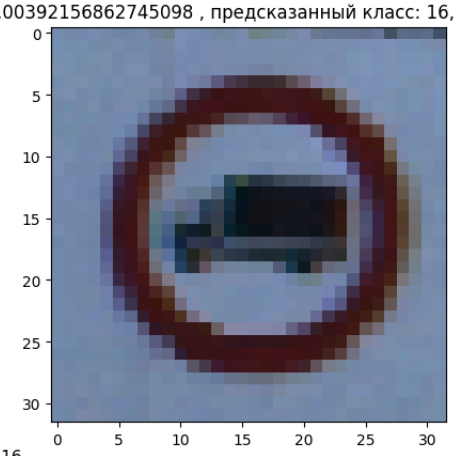
```
adv_losses_fgsm = np.array(adv_losses_fgsm)
adv_accuracises_fgsm = np.array(adv_accuracises_fgsm)
np.save("adv_losses_fgsm_rn50", adv_losses_fgsm)
np.save("adv_accuracises_fgsm_rn50", adv_accuracises_fgsm)
!cp adv_losses_fgsm_rn50.npy drive/MyDrive/adv_losses_pgd_rn50.npy
!cp adv_accuracises_fgsm_rn50.npy drive/MyDrive/adv_accuracises_fgsm_rn50.npy
```

```python
eps_range = [1/255, 5/255, 10/255, 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[0])}")
plt.imshow(x_test[0])
plt.show()
i = 1
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[0])}")
    plt.imshow(x_test_adv[0])
    plt.show()
    i += 1
```

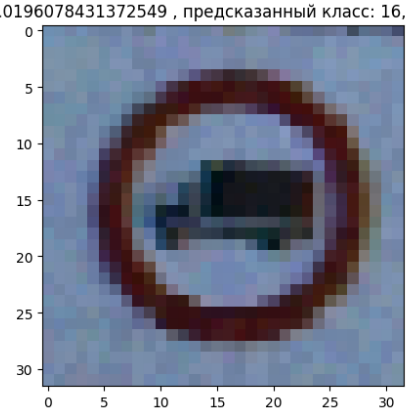
Исходное изображение, предсказанный класс: 16, действительный класс 16



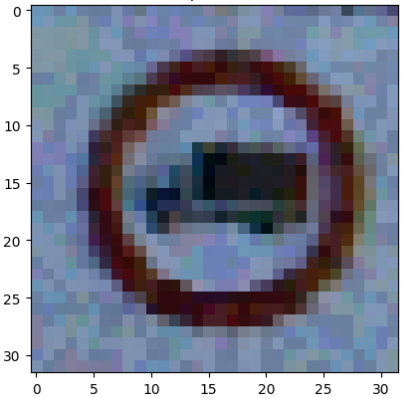
Изображение с eps: 0.00392156862745098 , предсказанный класс: 16, действительный класс 16



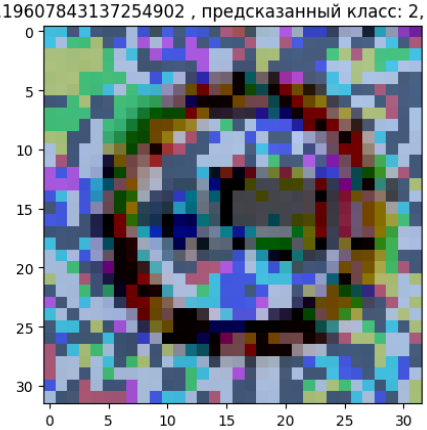
Изображение с eps: 0.0196078431372549 , предсказанный класс: 16, действительный класс 16



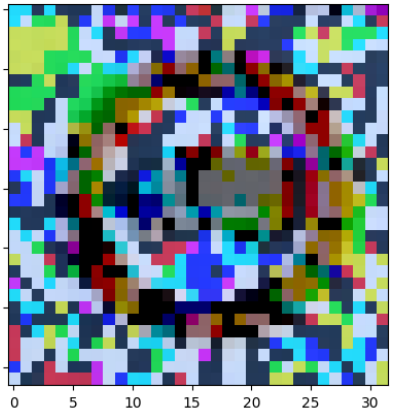
Изображение с eps: 0.0392156862745098 , предсказанный класс: 9, действительный класс 16



Изображение с eps: 0.19607843137254902 , предсказанный класс: 2, действительный класс 16



Изображение с eps: 0.3137254901960784 , предсказанный класс: 2, действительный класс 16



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:

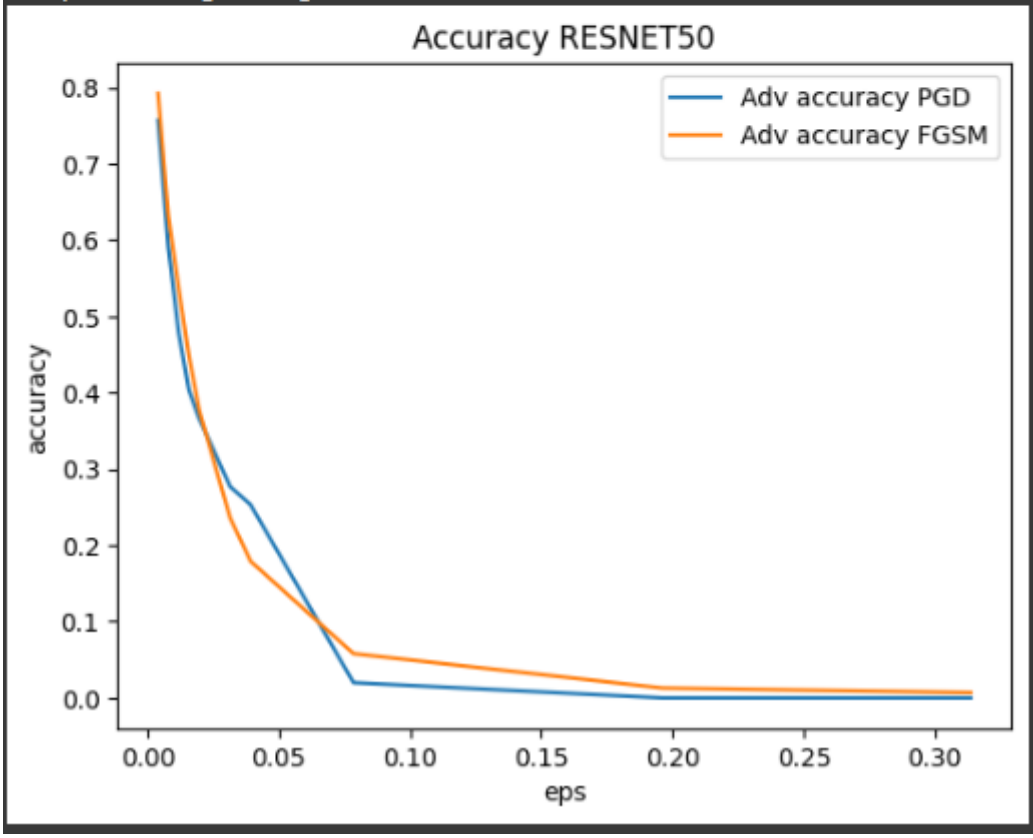
```
tf.compat.v1.disable_eager_execution()
model=load_model('ResNet50.h5')
x_test = data[:1000]
y_test = y_test[:1000]
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))

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 = []
adv_accuracises_pgd = []
true_losses = []
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: 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
```

```
adv_losses_pgd = np.array(adv_losses_pgd)
adv_accuracises_pgd = np.array(adv_accuracises_pgd)
np.save("adv_losses_pgd_rn50", adv_losses_pgd)
np.save("adv_accuracises_pgd_rn50", adv_accuracises_pgd)
!cp adv_losses_pgd_rn50.npy drive/MyDrive/adv_losses_pgd_rn50.npy
!cp adv_accuracises_pgd_rn50.npy drive/MyDrive/adv_accuracises_pgd_rn50.npy
```

```
adv_accuracises_fgsm = np.load("adv_accuracises_fgsm_rn50.npy")
adv_accuracises_pgd = np.load("adv_accuracises_pgd_rn50.npy")
plt.figure(0)
plt.plot(eps_range, adv_accuracises_pgd, label="Adv accuracy PGD")
plt.plot(eps_range, adv_accuracises_fgsm, label="Adv accuracy FGSM")
plt.title("Accuracy RESNET50")
plt.xlabel("eps")
plt.ylabel("accuracy")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7df411796dd0>
```



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

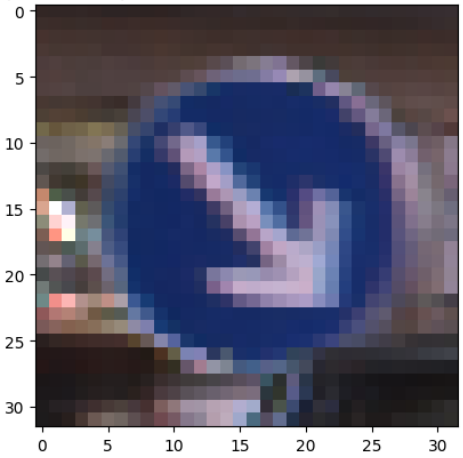
Реализуем атаку FGSM на VGG16:

```
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)))

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 = []
adv_accuracises_fgsm = []
true_losses = []
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}")
```

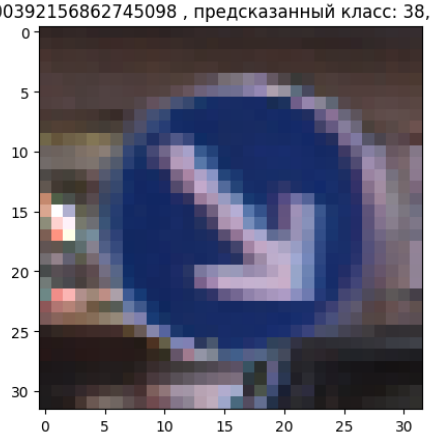
```
Eps: 0.00392156862745098
Adv Loss: 0.822020570397377
Adv Accuracy: 0.8650000095367432
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.00784313725490196
Adv Loss: 1.6516943979263305
Adv Accuracy: 0.7549999952316284
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.011764705882352941
Adv Loss: 2.4881884965896606
Adv Accuracy: 0.6520000100135803
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.01568627450980392
Adv Loss: 3.2180664806365966
Adv Accuracy: 0.5669999718666077
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.0196078431372549
Adv Loss: 3.841108205795288
Adv Accuracy: 0.4930000071525574
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.03137254901960784
Adv Loss: 5.124079051971435
Adv Accuracy: 0.335999995470047
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.0392156862745098
Adv Loss: 5.763013568878174
Adv Accuracy: 0.2529999911785126
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.0784313725490196
Adv Loss: 6.909527336120606
Adv Accuracy: 0.09000000357627869
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.19607843137254902
Adv Loss: 6.646053092956543
Adv Accuracy: 0.06599999964237213
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
Eps: 0.3137254901960784
Adv Loss: 6.019384979248047
Adv Accuracy: 0.06499999761581421
True Loss: 0.2530437219664454
True Accuracy: 0.9440000057220459
```

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

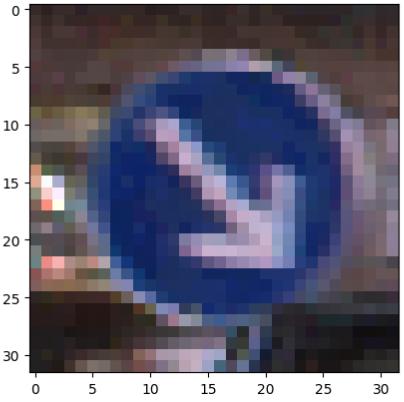
Исходное изображение, предсказанный класс: 38, действительный класс 38



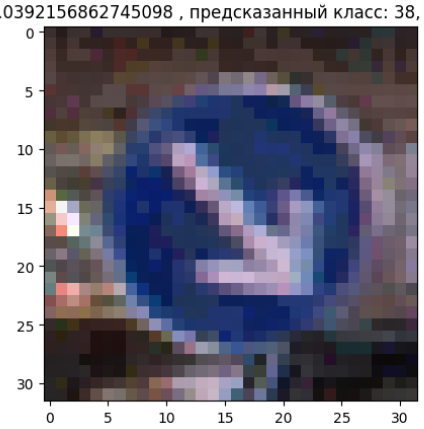
Изображение с eps: 0.00392156862745098 , предсказанный класс: 38, действительный класс 38



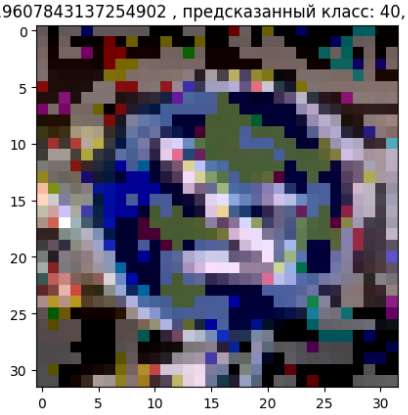
Изображение с eps: 0.0196078431372549 , предсказанный класс: 38, действительный класс 38



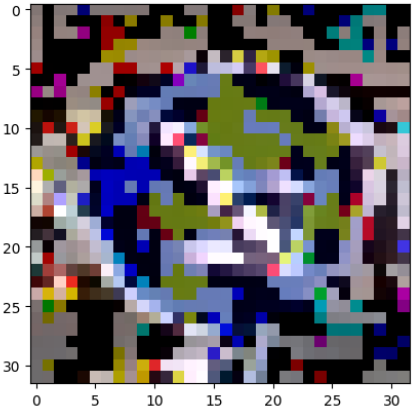
Изображение с eps: 0.0392156862745098 , предсказанный класс: 38, действительный класс 38



Изображение с eps: 0.19607843137254902 , предсказанный класс: 40, действительный класс 38



Изображение с eps: 0.3137254901960784 , предсказанный класс: 38, действительный класс 38



```
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)
!cp adv_losses_fgsm_vgg16.npy drive/MyDrive/adv_losses_pgd_vgg16.npy
!cp adv_accuracises_fgsm_vgg16.npy drive/MyDrive/adv_accuracises_fgsm_vgg16.npy
```

Выполним атаку PGD на VGG16:

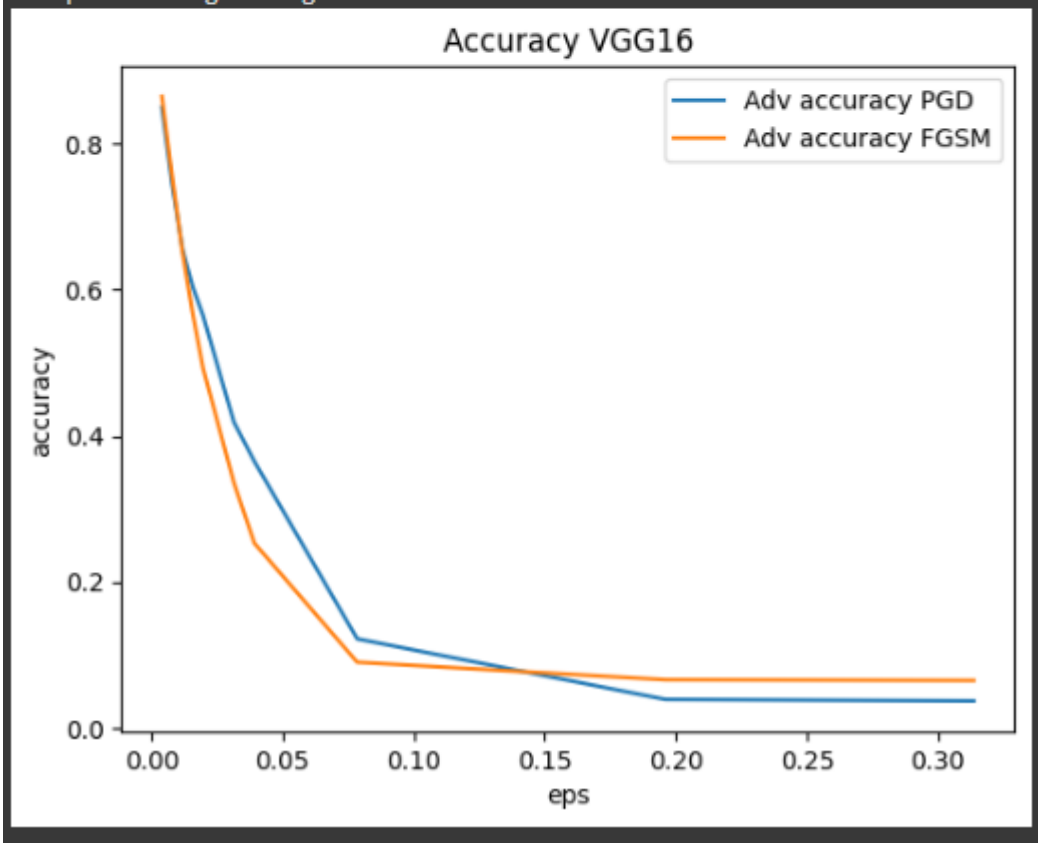
```
tf.compat.v1.disable_eager_execution()
model=load_model('VGG16.h5')
x_test = data[:1000]
y_test = y_test[:1000]
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))

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 = []
adv_accuracises_pgd = []
true_losses = []
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
```

```
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)
!cp adv_losses_pgd_vgg16.npy drive/MyDrive/adv_losses_pgd_vgg16.npy
!cp adv_accuracises_pgd_vgg16.npy drive/MyDrive/adv_accuracises_pgd_vgg16.npy
```

```
adv_accuracises_fgsm = np.load("adv_accuracises_fgsm_vgg16.npy")
adv_accuracises_pgd = np.load("adv_accuracises_pgd_vgg16.npy")
plt.figure(0)
plt.plot(eps_range, adv_accuracises_pgd, label="Adv accuracy PGD")
plt.plot(eps_range, adv_accuracises_fgsm, label="Adv accuracy FGSM")
plt.title("Accuracy VGG16")
plt.xlabel("eps")
plt.ylabel("accuracy")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7df411795ba0>
```



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

Заполним сравнительную таблицу 2.

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)
```

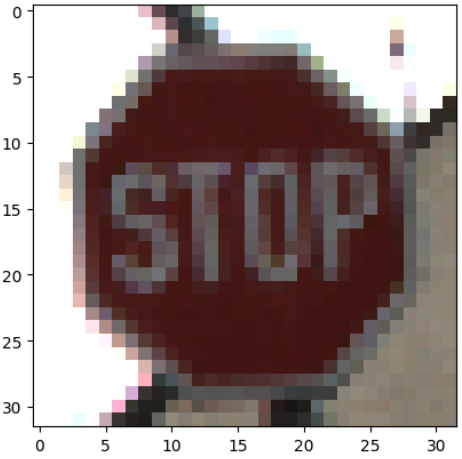
```
model=load_model('ResNet50.h5')
tf.compat.v1.disable_eager_execution()
t_class = 1
t_class = to_categorical(t_class, 43)
t_classes = np.tile(t_class, (270, 1))
x_test = data
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
attack_fgsm = FastGradientMethod(estimator=classifier, 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-package
  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
```

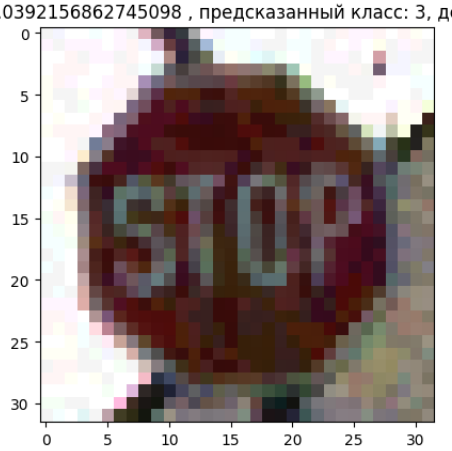
```
eps = 10/255
attack_fgsm.set_params(**{'eps': eps})
x_test_adv = attack_fgsm.generate(x_test, t_classes)
```

```
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

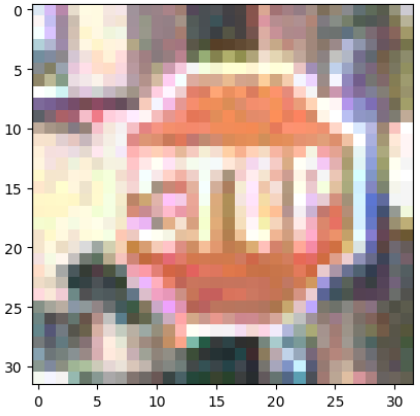


Изображение с eps: 0.0392156862745098 , предсказанный класс: 3, действительный класс 14

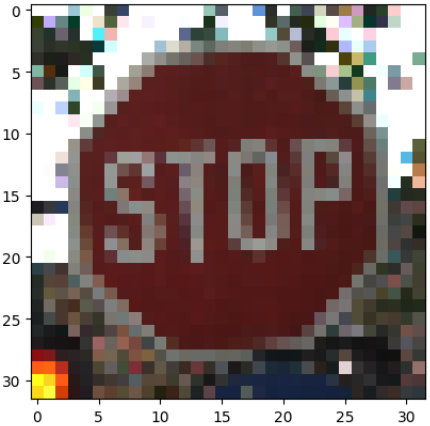


Исходное изображение, предсказанный класс: 14, действительный класс 14

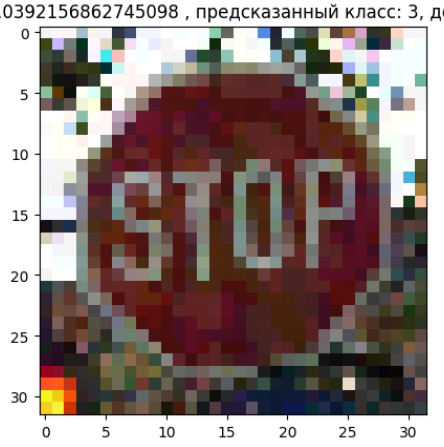
Изображение с eps: 0.0392156862745098 , предсказанный класс: 12, действительный класс 14



Исходное изображение, предсказанный класс: 14, действительный класс 14

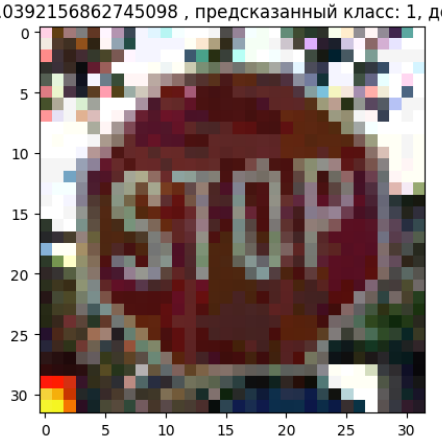


Изображение с eps: 0.0392156862745098 , предсказанный класс: 3, действительный класс 14

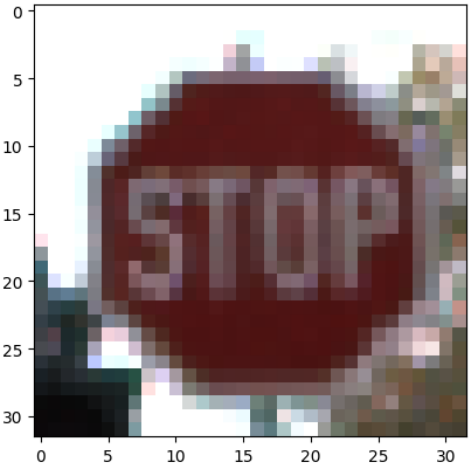


Исходное изображение, предсказанный класс: 14, действительный класс 14

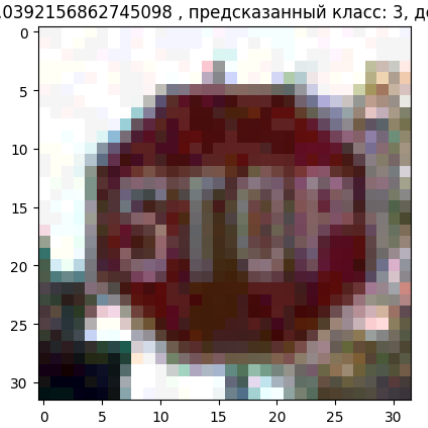
Изображение с eps: 0.0392156862745098 , предсказанный класс: 1, действительный класс 14



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:

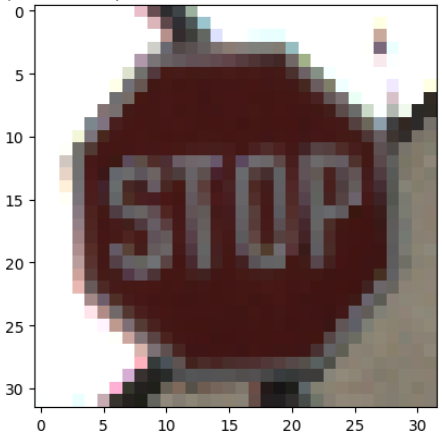
```
model=load_model('ResNet50.h5')
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3, max_iter=4, verbose=False, targeted=True)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 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: 0.0392156862745098
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
```

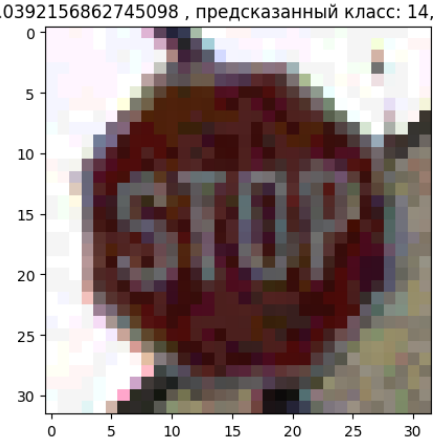
```
eps = 10/255
attack_pgd.set_params(**{'eps': eps})
x_test_adv = attack_pgd.generate(x_test, t_classes)
```

```
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()
```

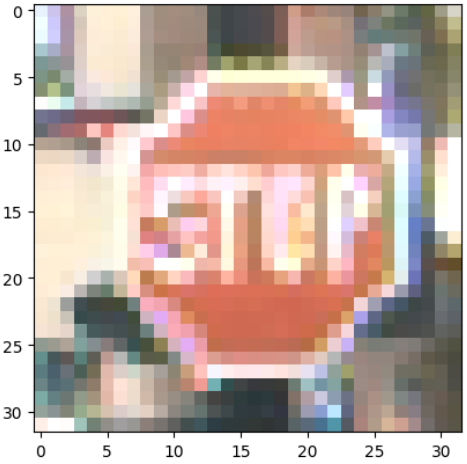
Исходное изображение, предсказанный класс: 14, действительный класс 14



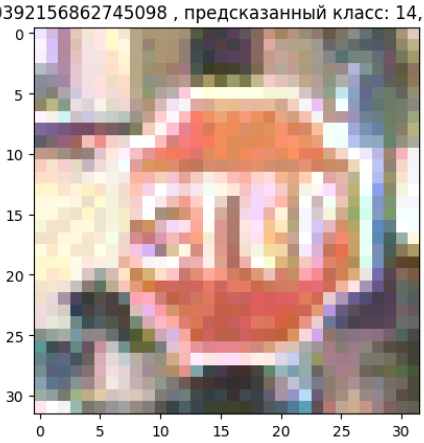
Изображение с eps: 0.0392156862745098 , предсказанный класс: 14, действительный класс 14



Исходное изображение, предсказанный класс: 14, действительный класс 14

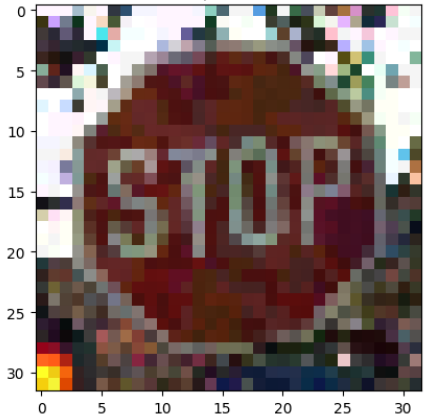


Изображение с eps: 0.0392156862745098 , предсказанный класс: 14, действительный класс 14



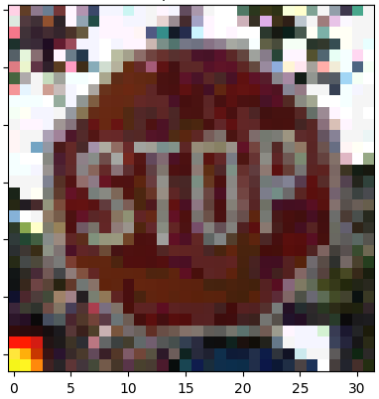
Исходное изображение, предсказанный класс: 14, действительный класс 14

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



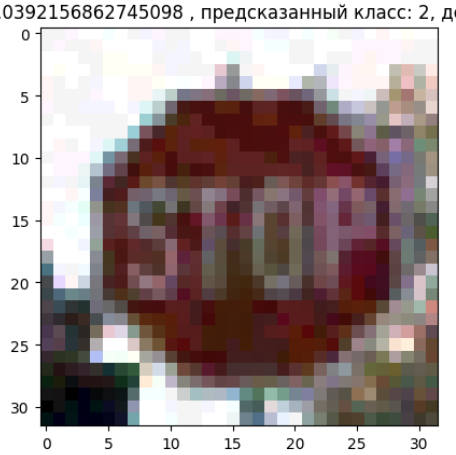
Исходное изображение, предсказанный класс: 14, действительный класс 14

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: 2, действительный класс 14



Заполним таблицу 3, в которой представим точность целевых атак PGD и FGSM на знак стоп (атака заключается в смене класса на ограничение скорости в 30 км/ч).

Таблица 3 – Точность целевых атак

Искажение	**PGD attack – Stop sign images**	**FGSM attack – Stop sign images**
ϵ=1/255	0,9630	0,8741
ϵ=3/255	0,8667	0,6778
ϵ=5/255	0,7741	0,4259
ϵ=10/255	0,6556	0,5556
ϵ=20/255	0,2815	0,037
ϵ=50/255	0,0296	0
ϵ=80/255	0,0111	0