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

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:**

ББМО–02–22

Шмарковский М. Б.

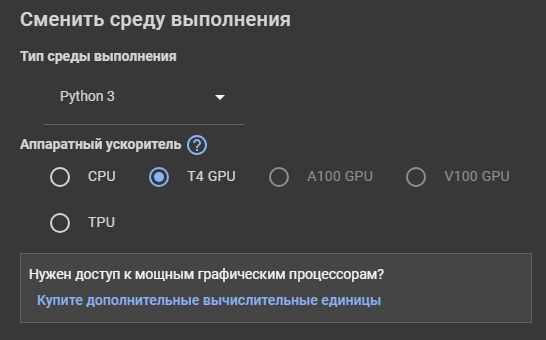
**Проверил:**

Спирин А. А.

Москва, 2024

Подготовительный этап

Поменяем среду выполнения на GPU:



Выполним установку инструмента adversarial-robustness-toolbox:

!pip install adversarial-robustness-toolbox

Collecting adversarial-robustness-toolbox

Downloading adversarial\_robustness\_toolbox-1.17.0-py3-none-any.whl (1.7 MB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 7.3 MB/s eta 0:00:00

Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.23.5)

Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.11.4)

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 18.1 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 0.19.2 requires scikit-learn>=1.2.2, but you have scikit-learn 1.1.3 which is incompatible.

Successfully installed adversarial-robustness-toolbox-1.17.0 scikit-learn-1.1.3

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

from google.colab import drive

drive.mount('/content/drive')

!unzip -q /content/drive/MyDrive/'Colab Notebooks'/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 и отобразим первое изображение:

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)

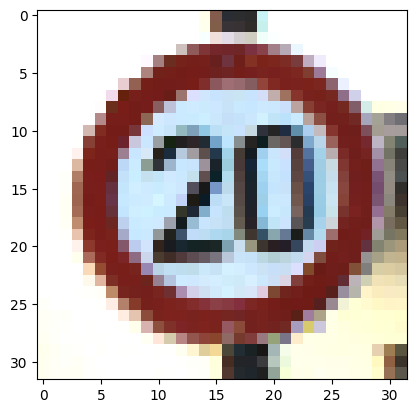
data = np.array(data)

labels = np.array(labels)

labels = to\_categorical(labels, 43)

plt.imshow(data[0])

<matplotlib.image.AxesImage at 0x7ce89ea74730>



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 [==============================] - 1s 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 [==============================] - 55s 60ms/step - loss: 1.1352 - accuracy: 0.7123 - val\_loss: 51.0620 - val\_accuracy: 0.1789

Epoch 2/5

429/429 [==============================] - 22s 51ms/step - loss: 0.2381 - accuracy: 0.9388 - val\_loss: 1.4900 - val\_accuracy: 0.8227

Epoch 3/5

429/429 [==============================] - 21s 49ms/step - loss: 0.1333 - accuracy: 0.9656 - val\_loss: 0.2912 - val\_accuracy: 0.9532

Epoch 4/5

429/429 [==============================] - 21s 50ms/step - loss: 0.0894 - accuracy: 0.9765 - val\_loss: 0.2283 - val\_accuracy: 0.9424

Epoch 5/5

429/429 [==============================] - 23s 53ms/step - loss: 0.0799 - accuracy: 0.9807 - val\_loss: 0.0937 - val\_accuracy: 0.9759

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-8-d75c136fedbd>:1: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

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="Точность обучения")

plt.plot(history.history['val\_accuracy'], label="Точность val")

plt.title("Точность ResNet50")

plt.xlabel("эпохи")

plt.ylabel("точность")

plt.legend()

plt.figure(1)

plt.plot(history.history['loss'], label="Потери обучения")

plt.plot(history.history['val\_loss'], label="Потери val")

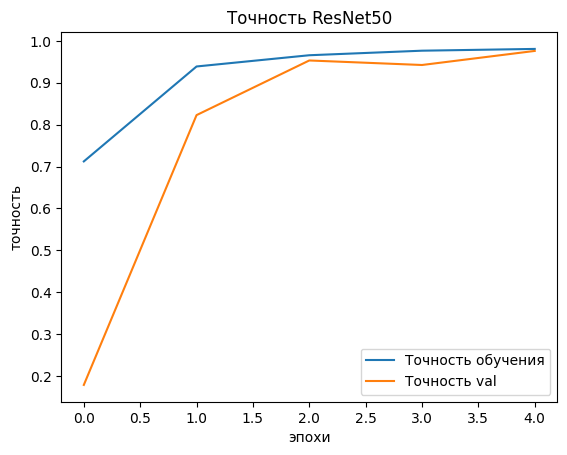
plt.title("Потери ResNet50")

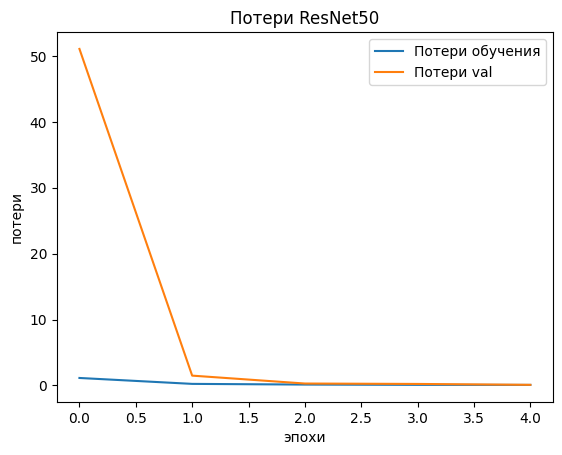
plt.xlabel("эпохи")

plt.ylabel("потери")

plt.legend()

plt.show()





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

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)

loss, accuracy = model.evaluate(data, y\_test)

print(f"Потери теста: {loss}")

print(f"Точность теста: {accuracy}")

395/395 [==============================] - 6s 13ms/step - loss: 0.3897 - accuracy: 0.9202

Потери теста: 0.3896692097187042

Точность теста: 0.9201900362968445

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

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)

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

Epoch 1/5

429/429 [==============================] - 29s 54ms/step - loss: 3.5978 - accuracy: 0.1138 - val\_loss: 2.1290 - val\_accuracy: 0.2689

Epoch 2/5

429/429 [==============================] - 18s 42ms/step - loss: 1.5268 - accuracy: 0.4672 - val\_loss: 0.9800 - val\_accuracy: 0.6361

Epoch 3/5

429/429 [==============================] - 18s 42ms/step - loss: 0.6247 - accuracy: 0.7791 - val\_loss: 0.3379 - val\_accuracy: 0.8789

Epoch 4/5

429/429 [==============================] - 17s 41ms/step - loss: 0.2728 - accuracy: 0.9287 - val\_loss: 0.1077 - val\_accuracy: 0.9721

Epoch 5/5

429/429 [==============================] - 18s 41ms/step - loss: 0.1446 - accuracy: 0.9704 - val\_loss: 0.2290 - val\_accuracy: 0.9617

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-13-c4b9a345d3f9>:1: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')`.

save\_model(model, 'VGG16.h5')

plt.figure(0)

plt.plot(history.history['accuracy'], label="Точность обучения")

plt.plot(history.history['val\_accuracy'], label="Точность val")

plt.title("Точность VGG16")

plt.xlabel ("эпохи")

plt.ylabel ("точность")

plt.legend()

plt.figure (1)

plt.plot(history.history['loss'], label="Потери обучения")

plt.plot(history.history['val\_loss'], label="Потери val")

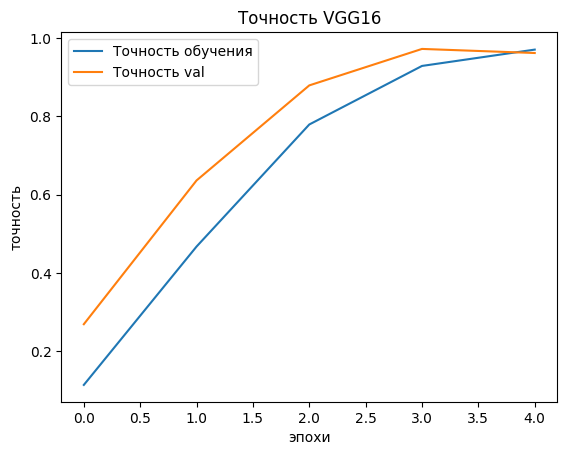
plt.title("Потери VGG16")

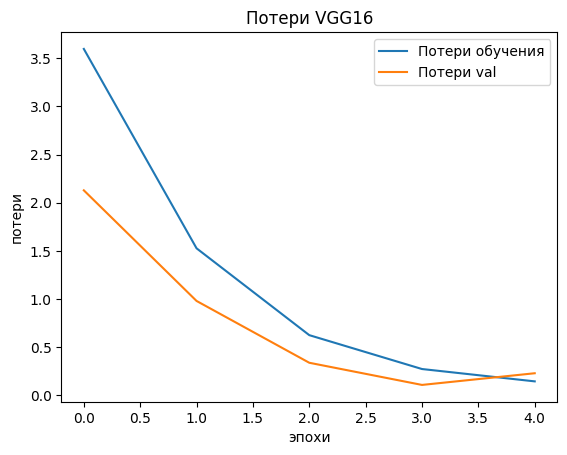
plt.xlabel ("эпохи")

plt.ylabel ("потери")

plt. legend()

plt.show()





loss, accuracy = model. evaluate(data, y\_test)

print(f"Потери теста: {loss}")

print(f"Точность теста: {accuracy}")

395/395 [==============================] - 5s 10ms/step - loss: 0.5007 - accuracy: 0.9170

Потери теста: 0.5007426738739014

Точность теста: 0.9170229434967041

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

Таблица 1 – Сравнительная таблица

|  |  |  |  |
| --- | --- | --- | --- |
| **Модель** | **Обучение** | **Валидация** | **Тест** |
| **ResNet50** | loss: 0.0799 accuracy: 0.9807 | val\_loss: 0.0937 val\_accuracy: 0.9759 | Потери теста: 0.3896692097187042  Точность теста: 0.9201900362968445 |
| **VGG16** | loss: 0.1446 accuracy: 0.9704 | val\_loss: 0.2290 val\_accuracy: 0.9617 | Потери теста: 0.5007426738739014  Точность теста:  0.9170229434967041 |

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/normalization/batch\_normalization.py:883: \_colocate\_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.

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 потери: {loss}")

  print(f"Adv точность: {accuracy}")

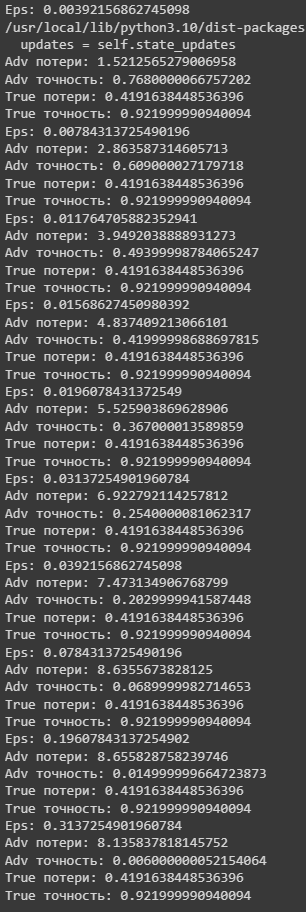
  loss, accuracy = model.evaluate(x\_test, y\_test)

  true\_accuracies.append(accuracy)

  true\_losses.append(loss)

  print(f"True потери: {loss}")

  print(f"True точность: {accuracy}")



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

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 ерs: {eps} , предсказанный класс {pred}, действительный класс {np.argmax(y\_test[0])}")

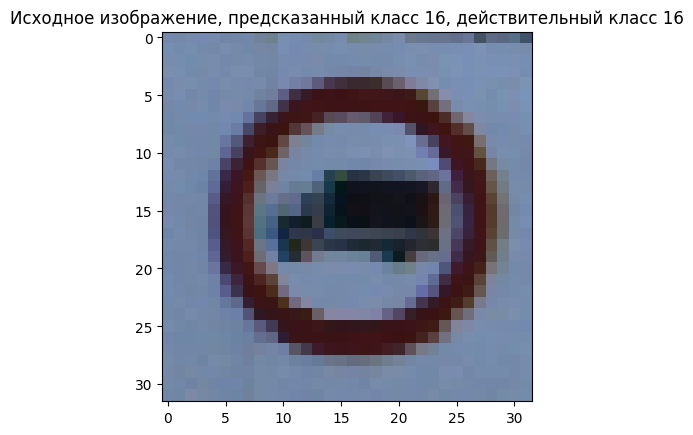
  plt.imshow(x\_test\_adv[0])

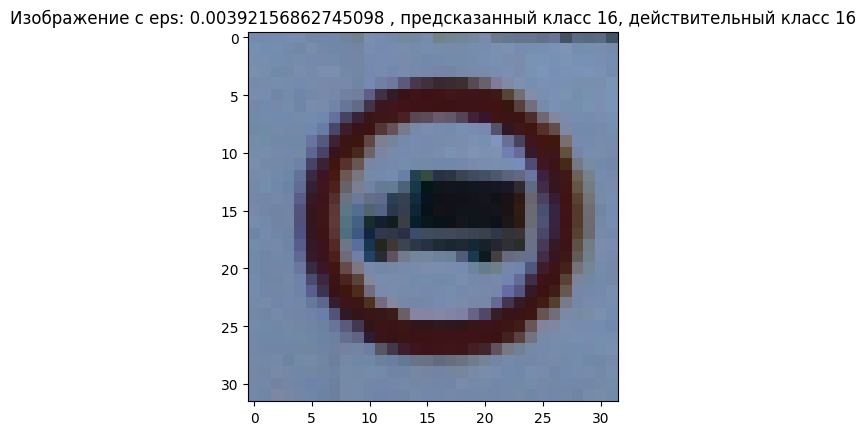
  plt.show()

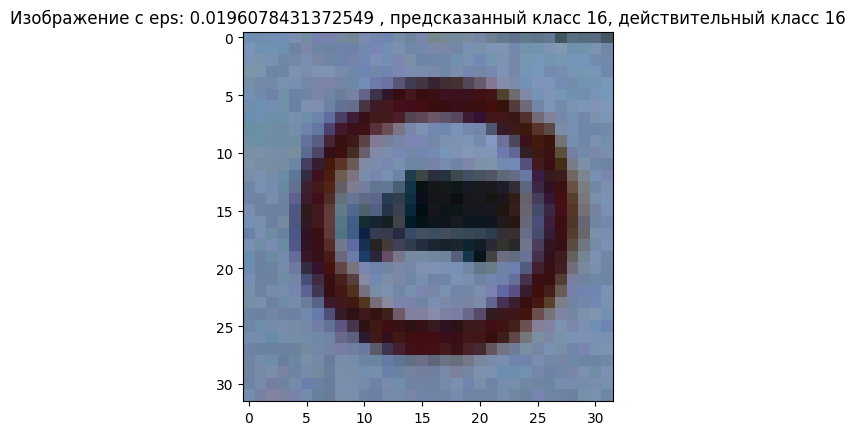
  i += 1

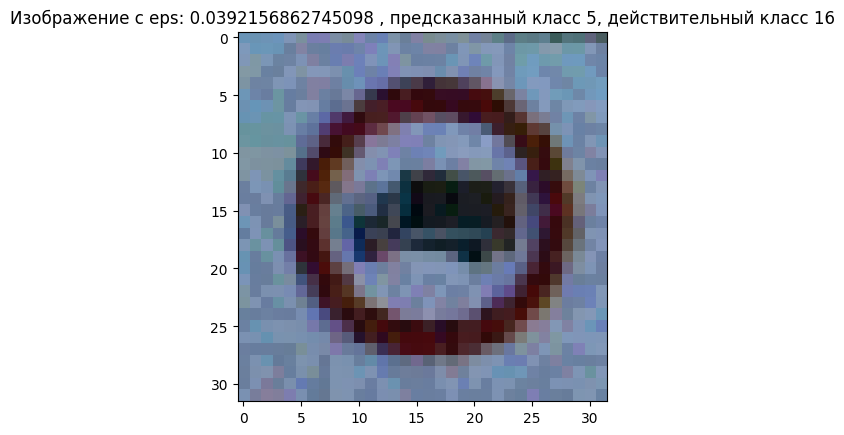
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training\_v1.py:2359: UserWarning: `Model.state\_updates` will be removed in a future version. This property should not be used in TensorFlow 2.0, as `updates` are applied automatically.

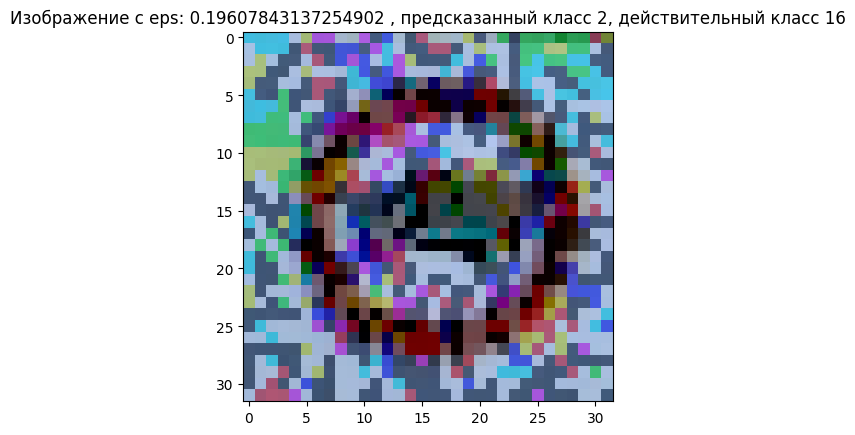
updates=self.state\_updates,

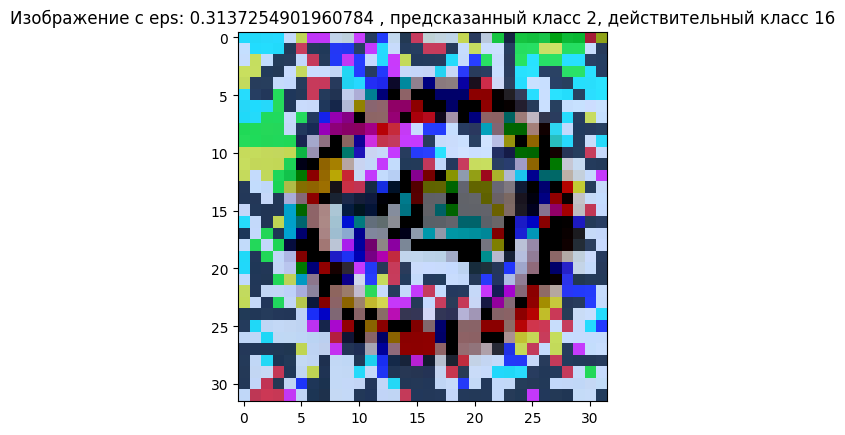












Видно, что при росте eps, шум на картинке сильно увеличивается, и с 5/255 уже становится более заметен. Оптимальным eps будет значение от 5/255 до 10/255.

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 потери: {loss}")

  print(f"Adv точность: {accuracy}")

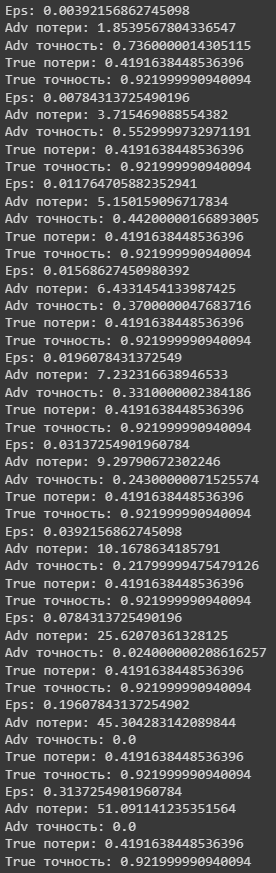
  loss, accuracy = model.evaluate(x\_test, y\_test)

  true\_accuracies.append(accuracy)

  true\_losses.append(loss)

  print(f"True потери: {loss}")

  print(f"True точность: {accuracy}")



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 точность PGD")

plt.plot(eps\_range, adv\_accuracises\_fgsm, label="Adv точность FGSM")

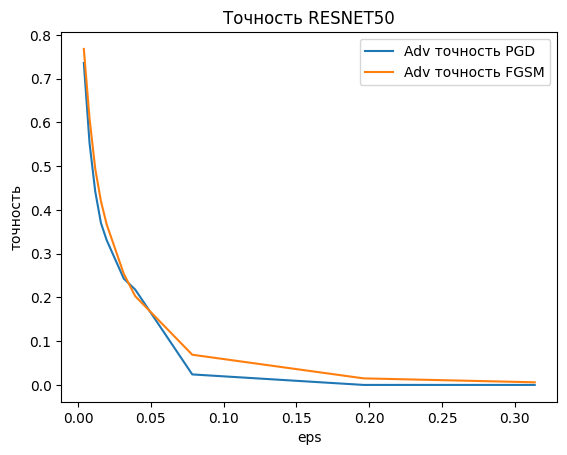
plt.title("Точность RESNET50")

plt.xlabel("eps")

plt.ylabel("точность")

plt. legend()

<matplotlib.legend.Legend at 0x7ce883ab3b20>



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 потери: {loss}")

  print(f"Adv точность: {accuracy}")

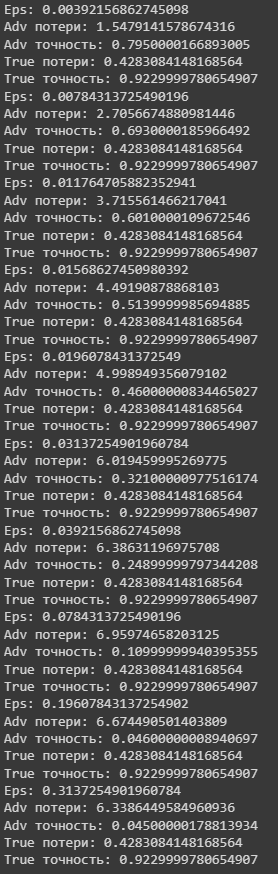
  loss, accuracy = model.evaluate(x\_test, y\_test)

  true\_accuracies.append(accuracy)

  true\_losses.append(loss)

  print(f"True потери: {loss}")

  print(f"True точность: {accuracy}")



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 ерs: {eps} , предсказанный класс {pred}, действительный класс {np.argmax(y\_test[2])}")

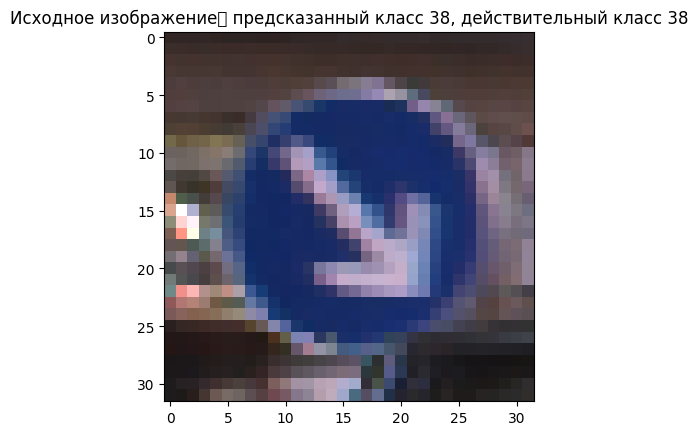
  plt.imshow(x\_test\_adv[2])

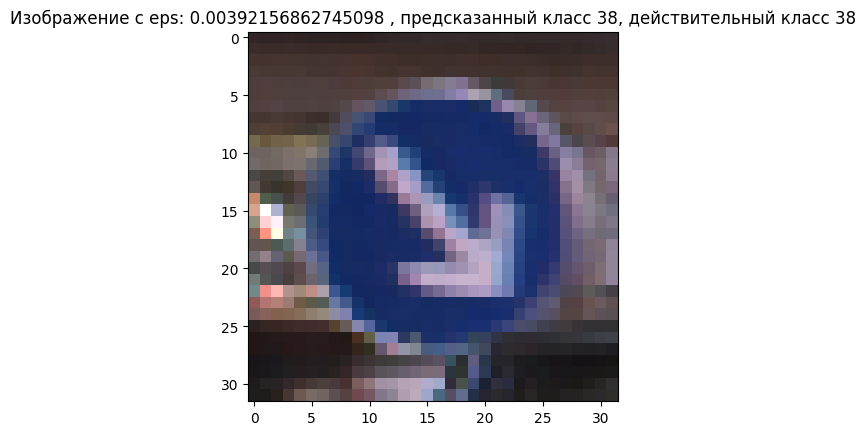
  plt.show()

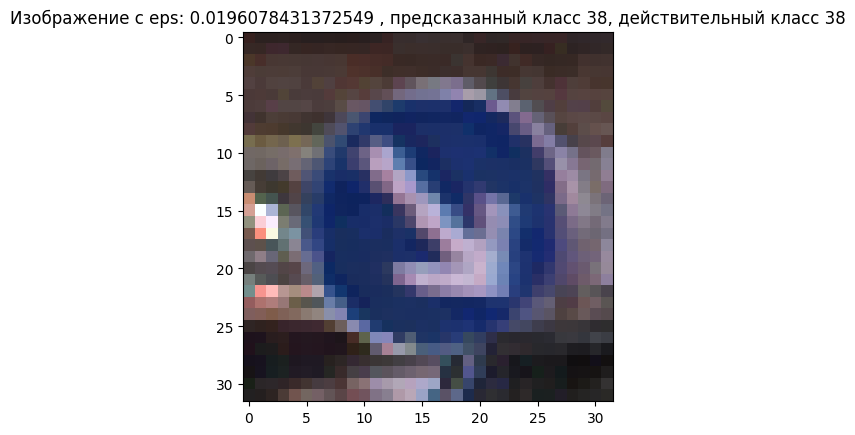
  i += 1

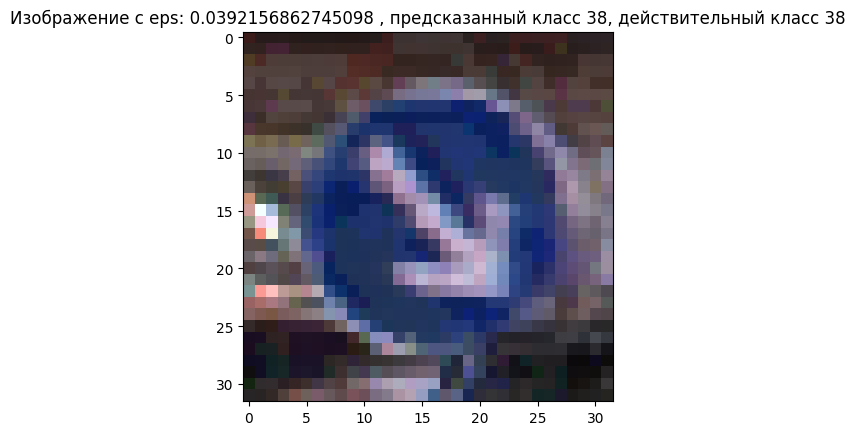
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 65292 (\N{FULLWIDTH COMMA}) missing from current font.

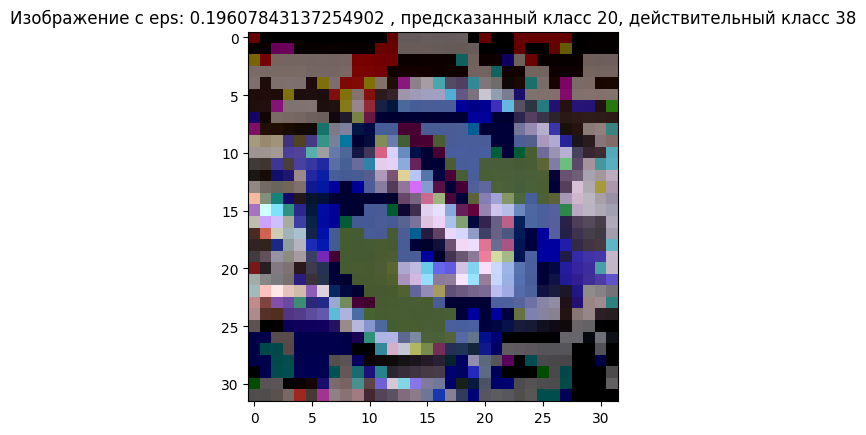
fig.canvas.print\_figure(bytes\_io, \*\*kw)

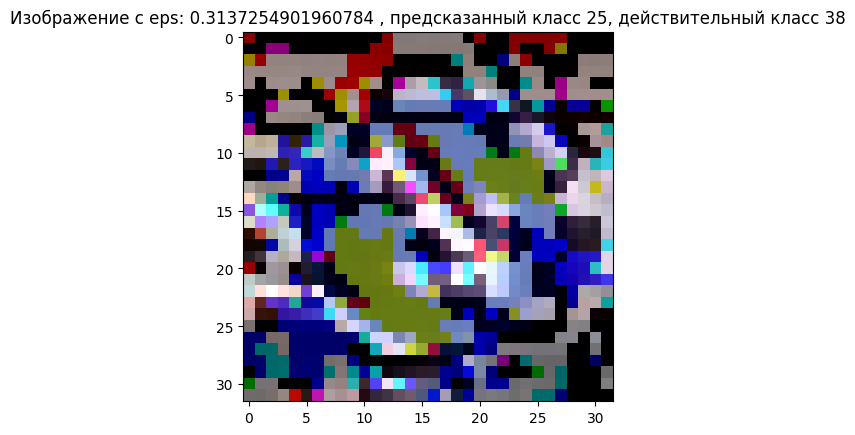












adv\_losses\_fgsm = np.array(adv\_losses\_fgsm)

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 потери: {loss}")

  print(f"Adv точность: {accuracy}")

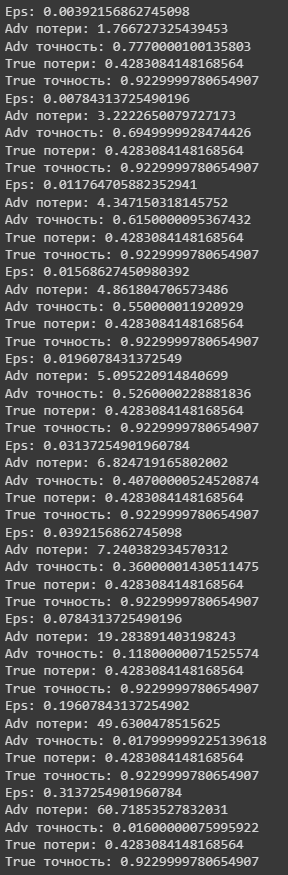
  loss, accuracy = model.evaluate(x\_test, y\_test)

  true\_accuracies.append(accuracy)

  true\_losses.append(loss)

  print(f"True потери: {loss}")

  print(f"True точность: {accuracy}")



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 точность PGD")

plt.plot(eps\_range, adv\_accuracises\_fgsm, label="Adv точность FGSM")

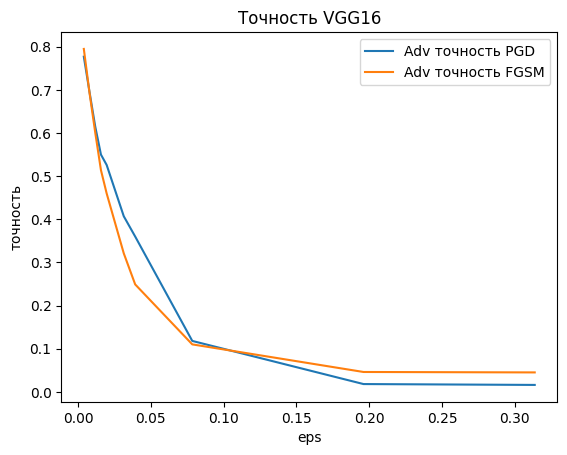
plt.title("Точность VGG16")

plt.xlabel("eps")

plt.ylabel("точность")

plt.legend()

<matplotlib.legend.Legend at 0x7ce882b6ee60>



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 𝜖𝜖=5/255 | Adversarial images 𝜖𝜖=10/255 |
| ResNet50 – FGSM | 91% | 74% | 33% | 17% |
| ResNet50 – PGD | 91% | 71% | 30% | 23% |
| VGG16 – FGSM | 89% | 79% | 44% | 21% |
| VGG16 – PGD | 89% | 77% | 48% | 32% |

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))

xtest = data

classifier = KerasClassifier(model=model, clip\_values=(np.min(x\_test), np.max(xtest)))

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)

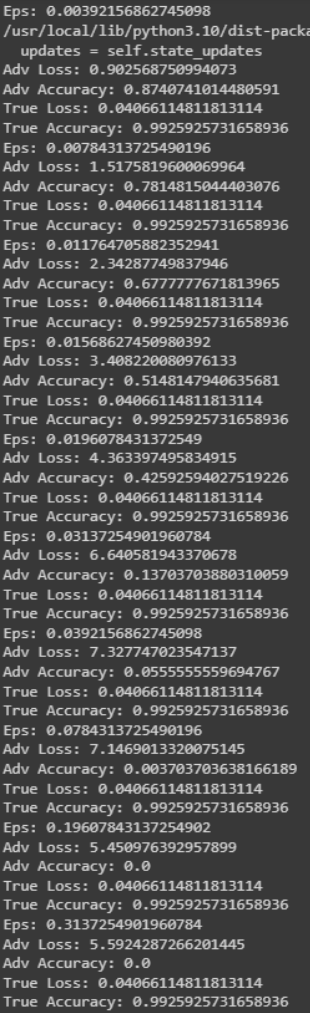
  print(f"Adv Loss: {loss}")

  print(f"Adv Accuracy: {accuracy}")

  loss, accuracy = model.evaluate(xtest, y\_test)

  print(f"True Loss: {loss}")

  print(f"True Accuracy: {accuracy}")



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

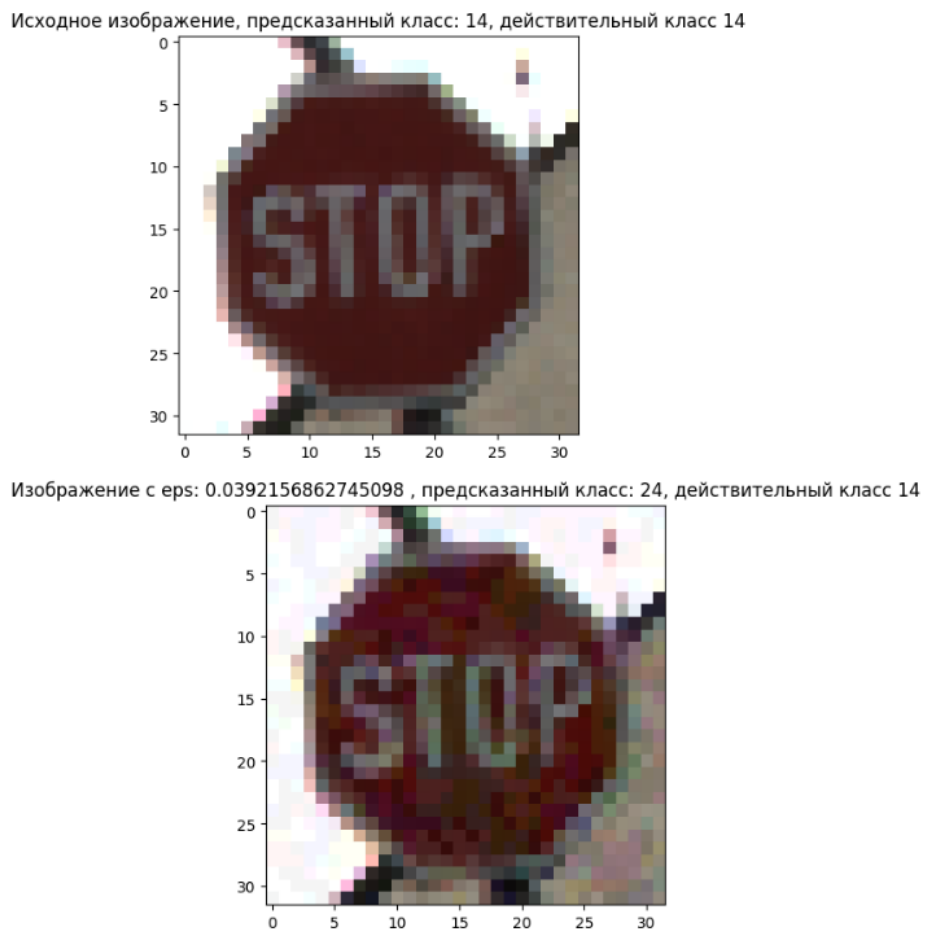
  pred = np.argmax(model.predict(x\_test\_adv[index:index+1]))

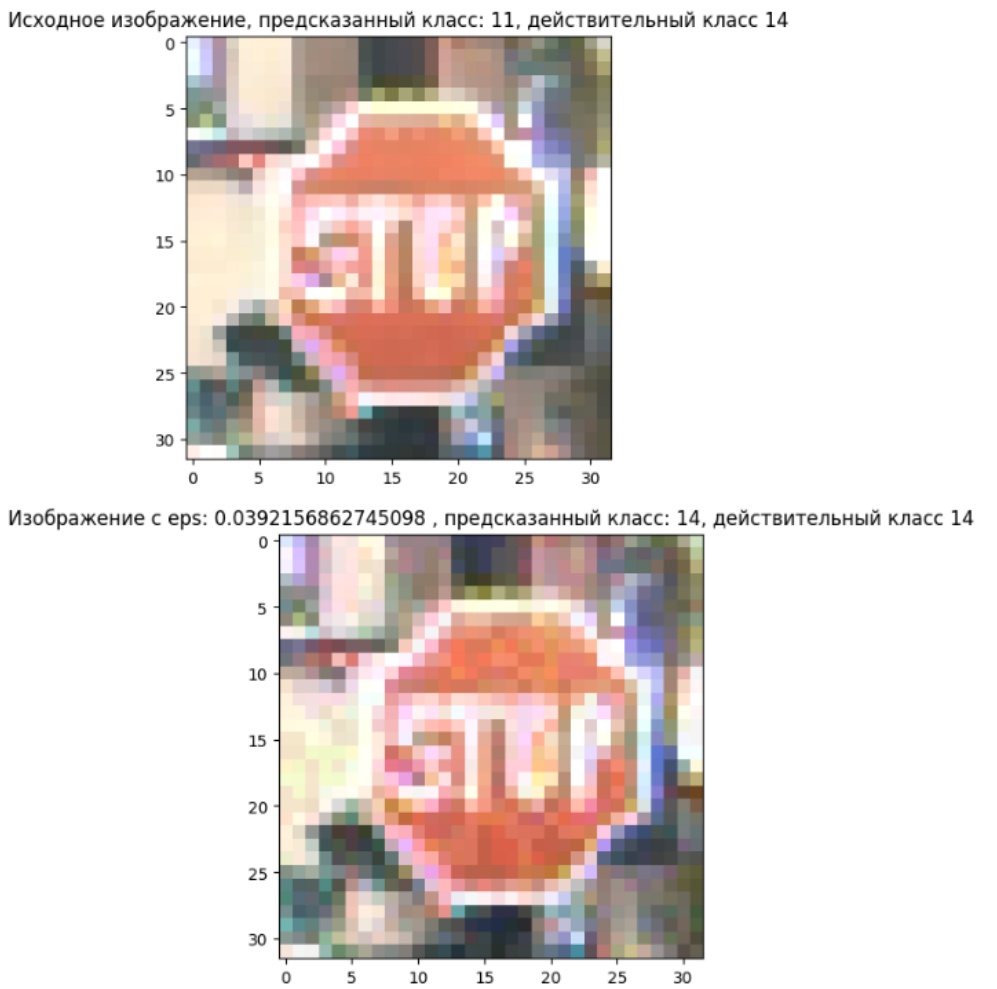
  plt.figure(i)

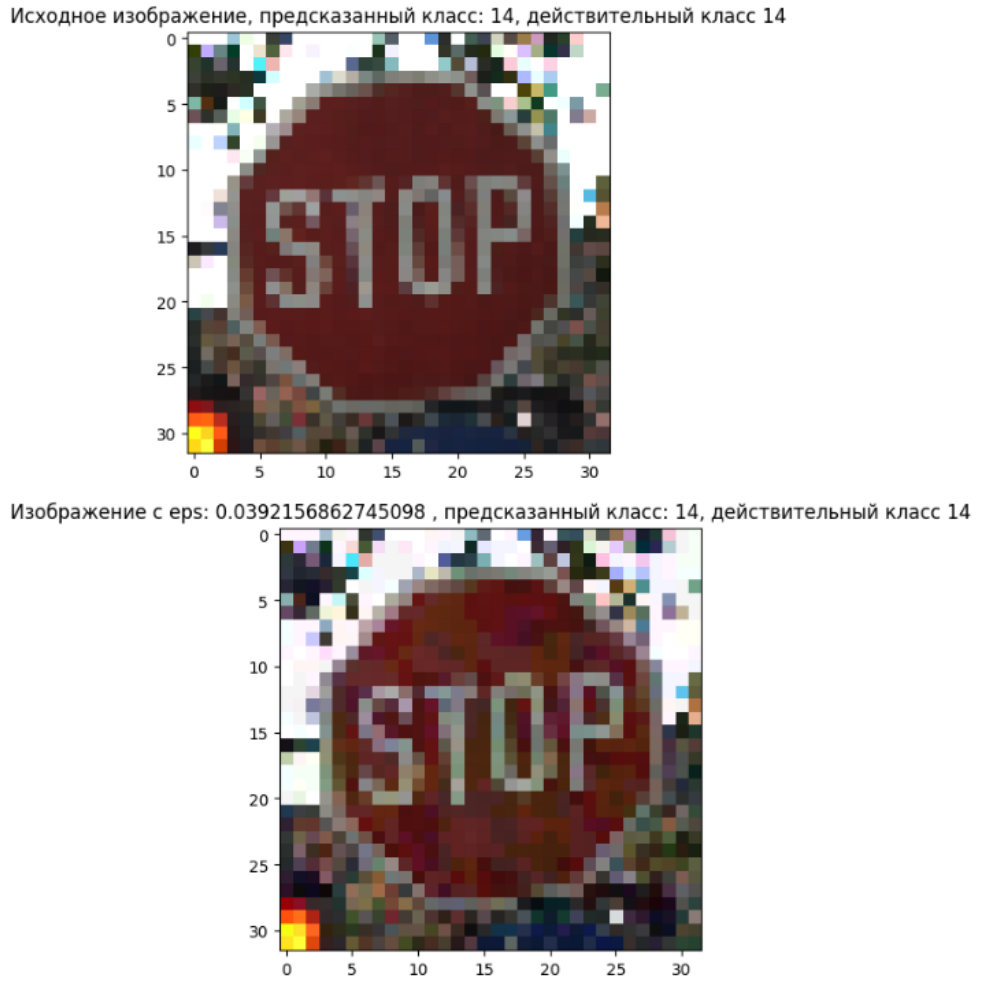
  plt.title(f"Изображение с ерs {eps} предсказанный класс {pred}, действительный класс {np.argmax(y\_test[index])}")

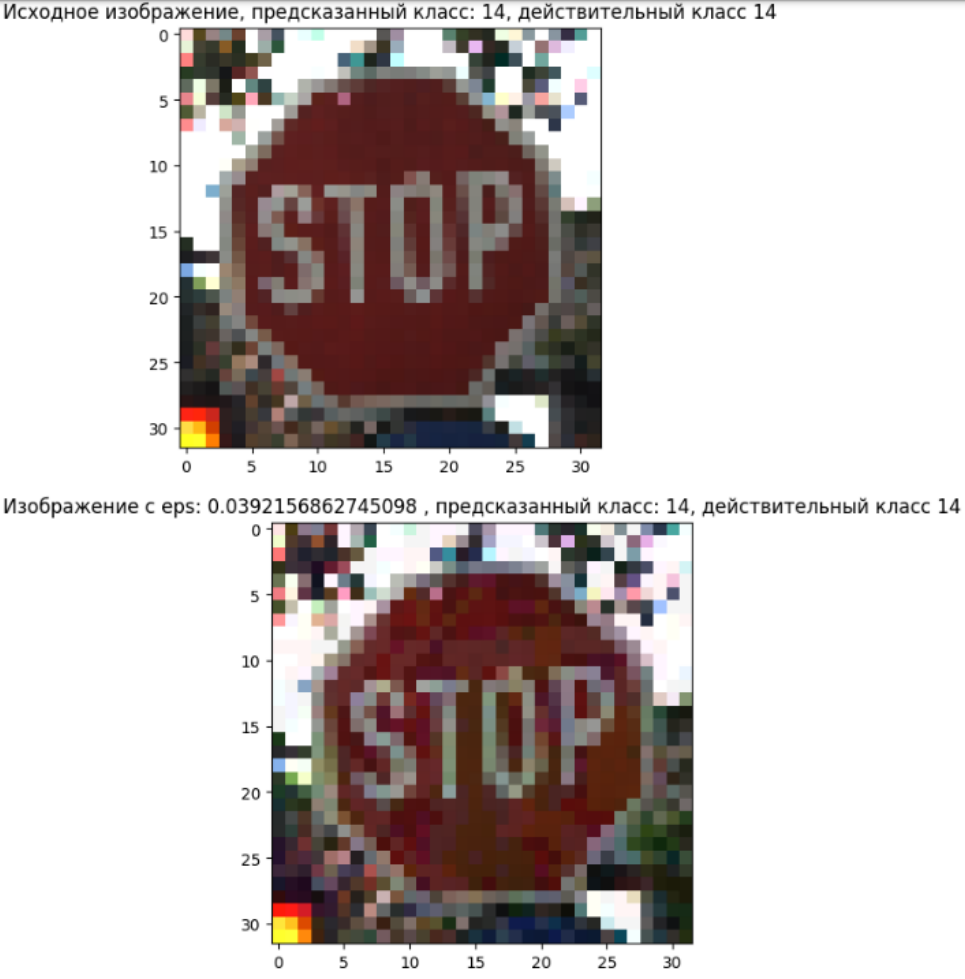
  plt.imshow(x\_test\_adv[index])

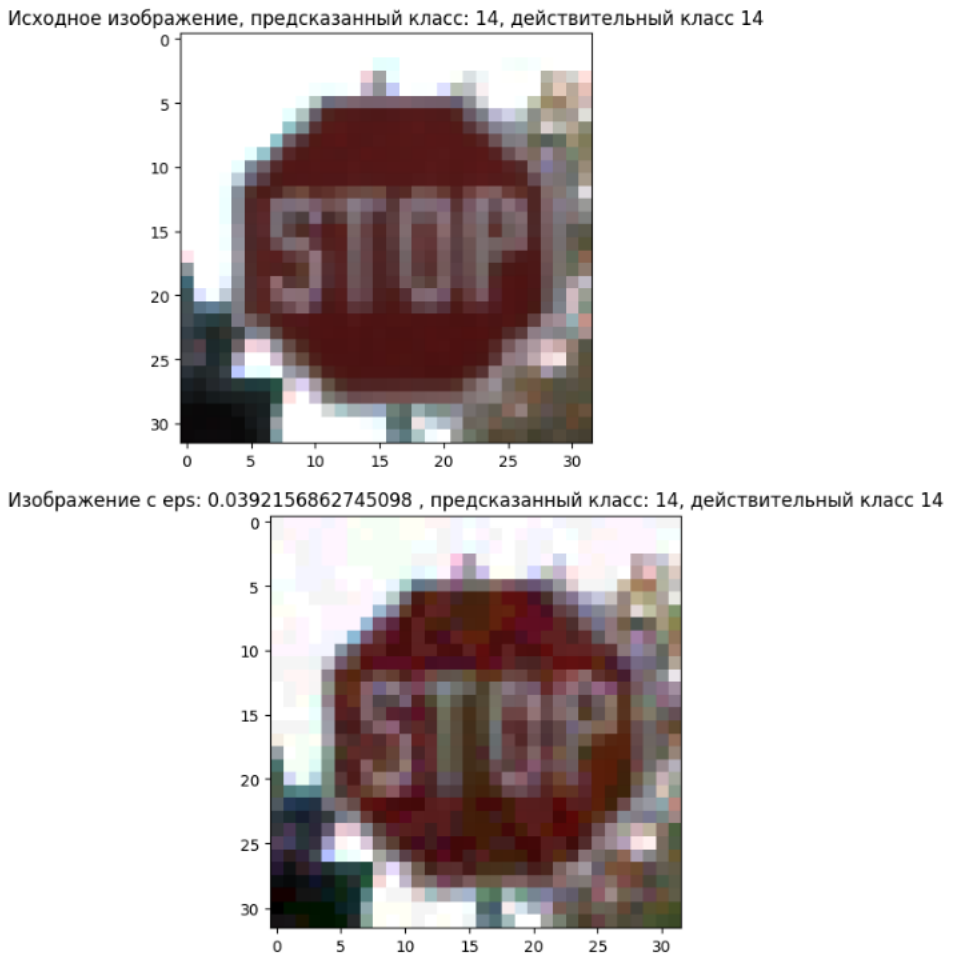
  plt.show()











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)

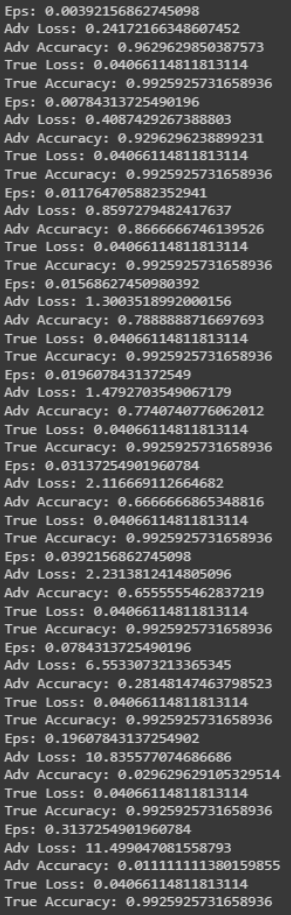
  print(f"Adv Loss: {loss}")

  print(f"Adv Accuracy: {accuracy}")

  loss, accuracy = model.evaluate(x\_test, y\_test)

  print(f"True Loss: {loss}")

  print(f"True Accuracy: {accuracy}")



eps = 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

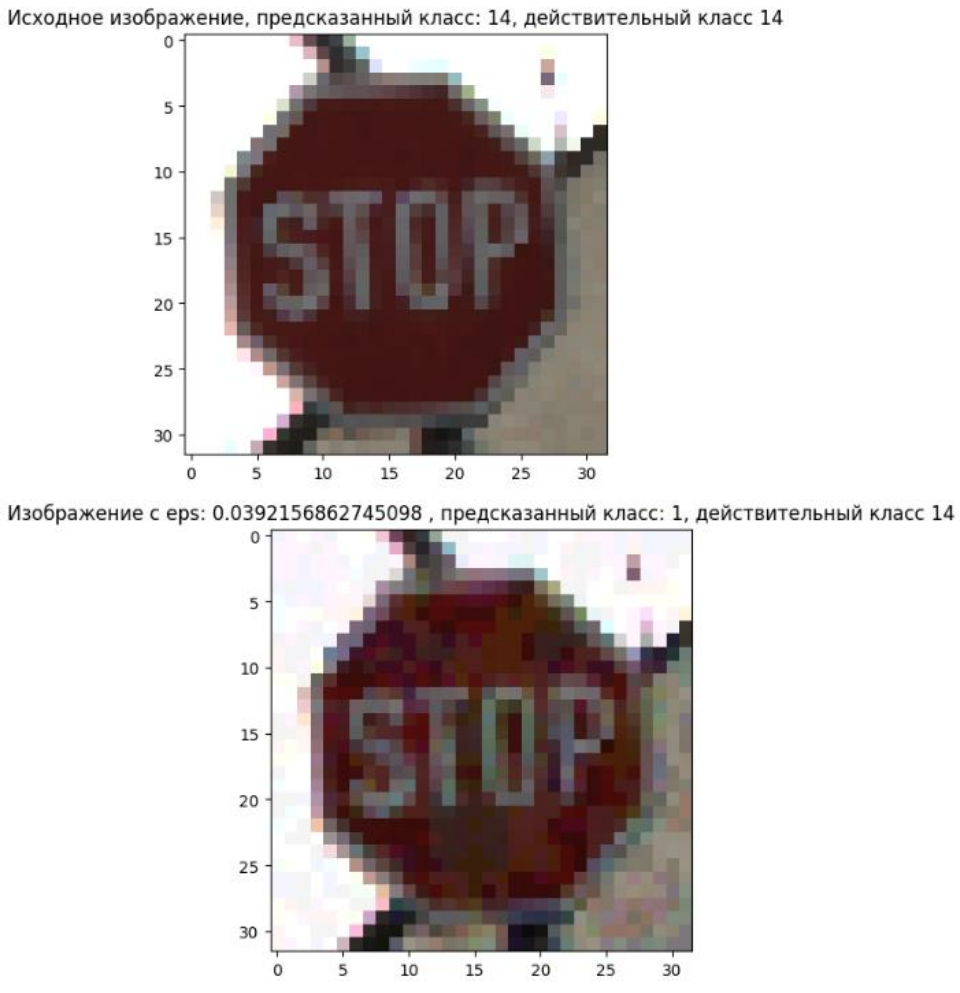
  pred = np.argmax(model.predict(x\_test\_adv[index:index+1]))

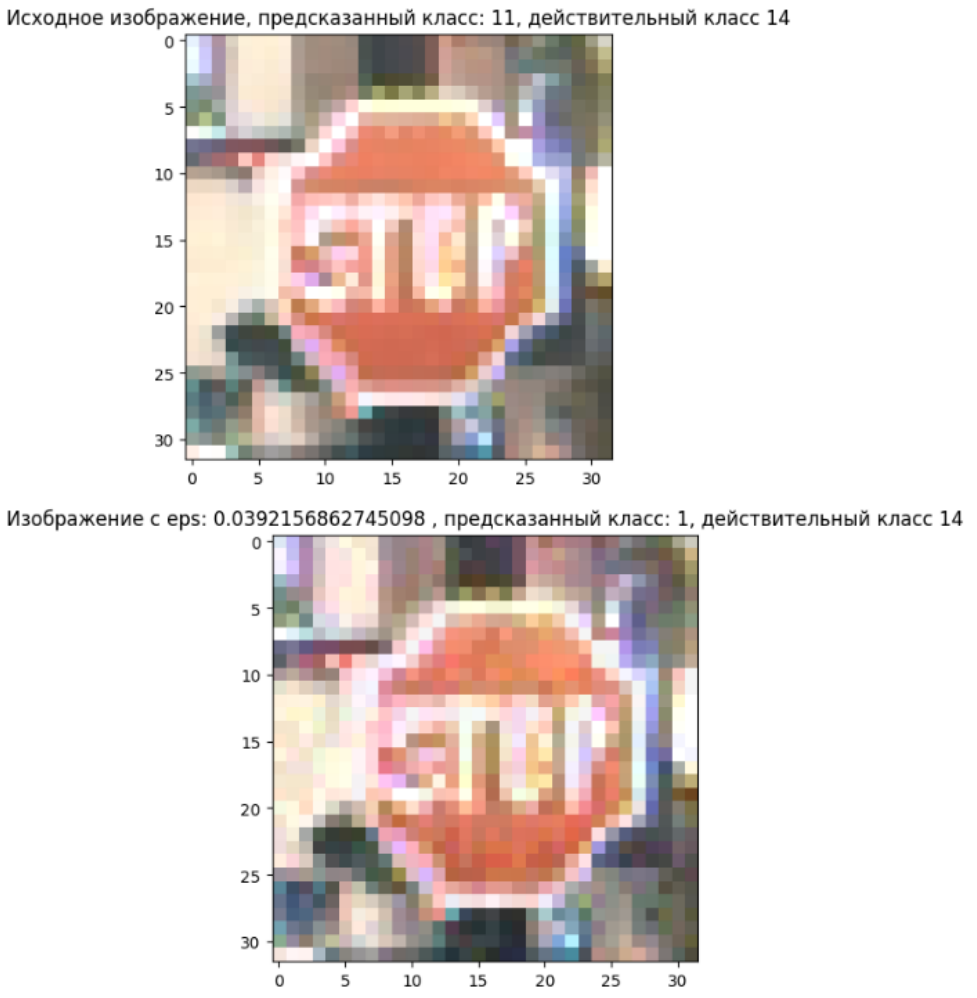
  plt.figure(i)

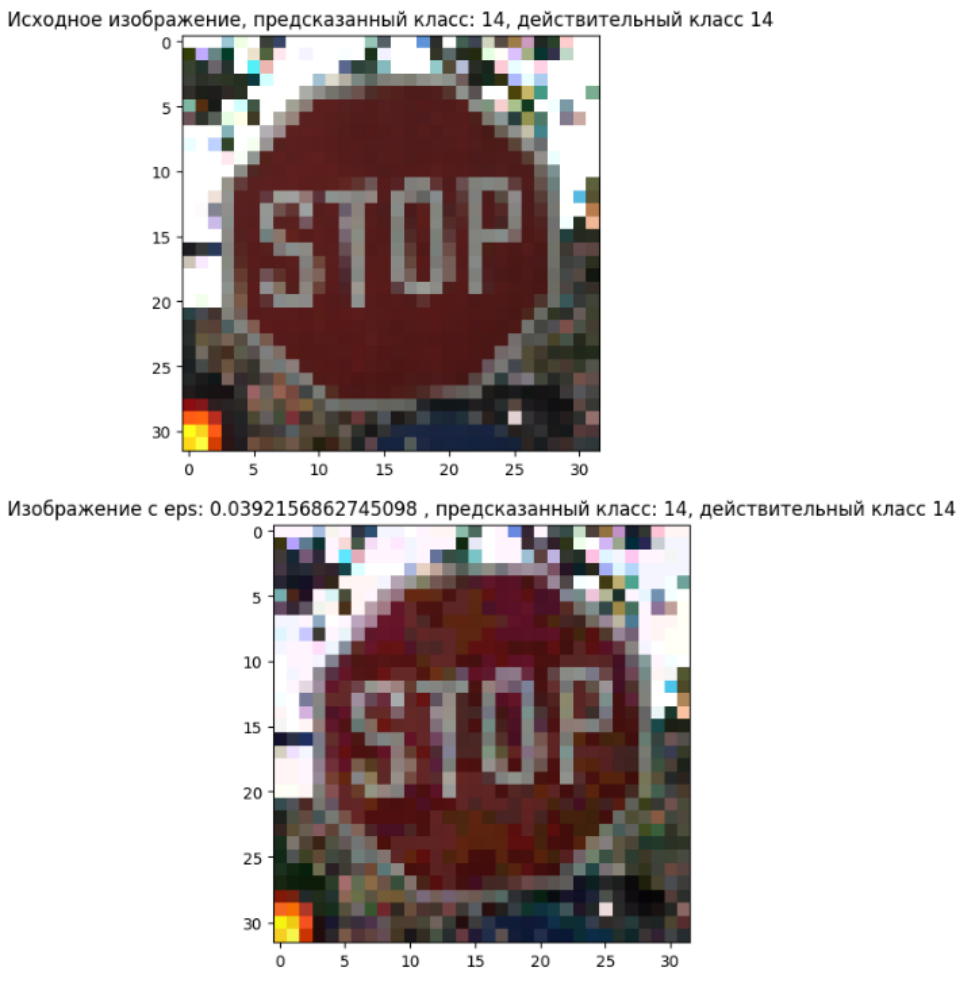
  plt.title(f"Изображение с eps: {eps} , предсказанный класс: {pred}, действительный класс {np.argmax(y\_test[index])}")

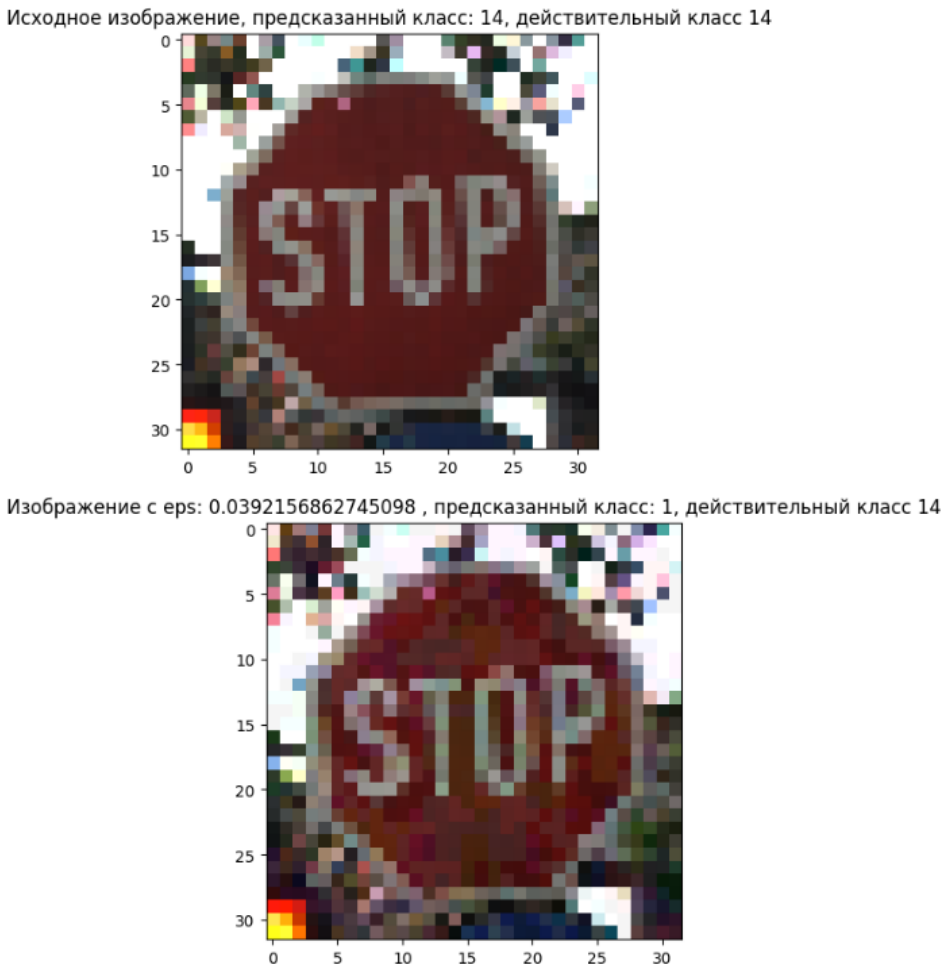
  plt.imshow(x\_test\_adv[index])

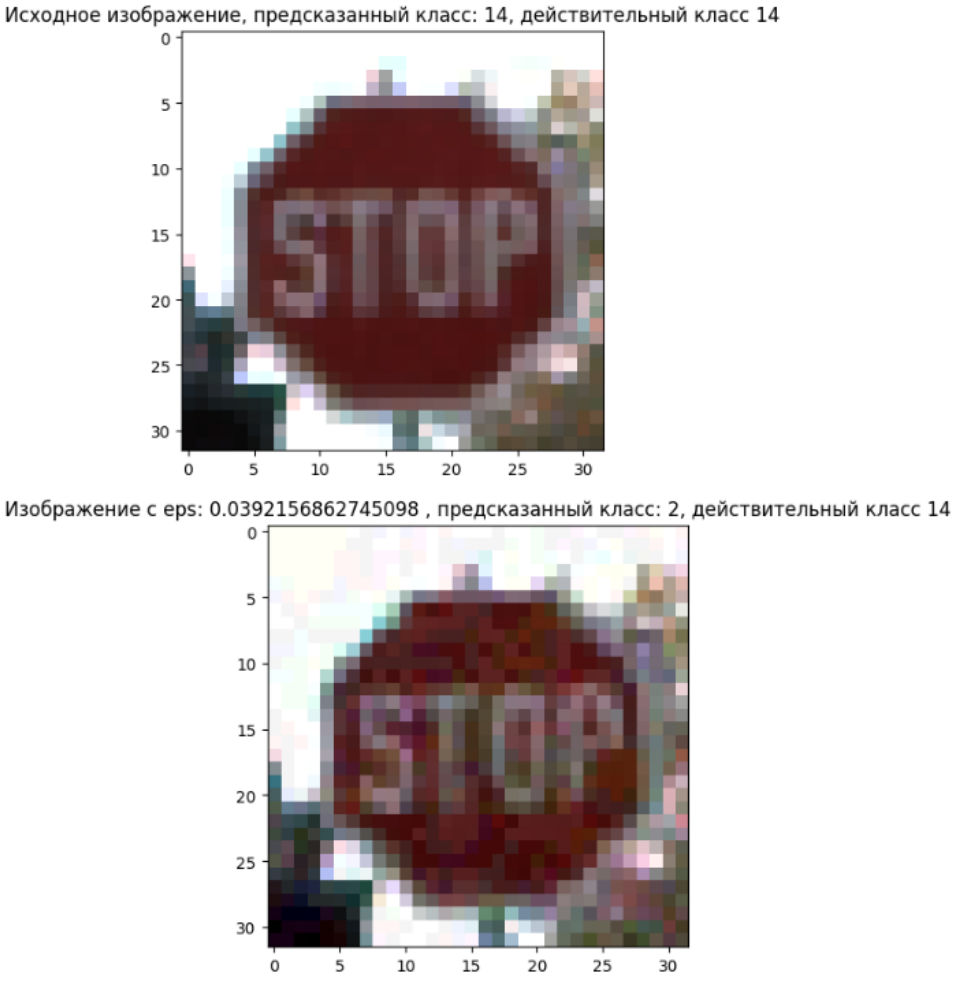
  plt.show











Заполним таблицу 3 и 4, 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 PGD

|  |  |  |
| --- | --- | --- |
| Искажение | PGD attack – Stop sign images | PGD attack – Speed Limit 30 sign images |
| 𝜖𝜖=1/255 | 97% | 99% |
| 𝜖𝜖=3/255 | 91% | 99% |
| 𝜖𝜖=5/255 | 90% | 99% |
| 𝜖𝜖=10/255 | 71% | 99% |

Таблица 4 – Точность целевых атак FGSM

|  |  |  |
| --- | --- | --- |
| Искажение | FGSM attack – Stop sign images | FGSM attack – Speed Limit 30 sign images |
| 𝜖𝜖=1/255 | 99% | 99% |
| 𝜖𝜖=3/255 | 80% | 99% |
| 𝜖𝜖=5/255 | 73% | 99% |
| 𝜖𝜖=10/255 | 26% | 99% |

По результатам видно метод PGD значительно лучше подходит для целевой атаки, чем метод FGSM.