МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

ИКБ направление «Кибербразведка и противодействие угрозам с применением технологий искусственного интеллекта» 10.04.01

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

Отчёт по лабораторной работе №2

по дисциплине: «Анализ защищенности систем искусственного интеллекта»

Группа:
ББМО-01-22
Выполнил:
Тимофеев И.О.

Проверил:
Спирин А.А.

Москва, 2023

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.

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.

Ход работы

```
!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 10.5 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 29.4 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
```

```
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
```

Рисунок 1- Импорт нужных библиотек

Подключаем google drive и загружаем в него dataset GTSRB.

```
from google.colab import drive
drive.mount("/content/drive")
```

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

```
# разархивируем датасет, который находится на подключенном гугл диске
zip_file = '/content/drive/MyDrive/LAB2/archive.zip'
z = zipfile.ZipFile(zip_file, 'r')
z.extractall()

print(os.listdir())
```

```
['.config', 'Meta', 'drive', 'train', 'Test', 'Test.csv', 'meta', 'test', 'Train.csv', 'Meta.csv', 'Train', 'sample_data']
```

```
# задаем пути к разархивированным данным
data_path = '/content'
train_data_path = os.path.join(data_path, 'Train')
test_data_path = os.path.join(data_path, 'Test')
meta_data_path = os.path.join(data_path, 'Meta')
```

Рисунок 2- Работа с дата сетом

```python
# предобработка
data = []
labels = []
class_count = 43
for i in range(class_count):
    img_path = os.path.join(train_data_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 первый элемент
print("data[0]:\n",data[0])
# отобразим первый элемент в виде картинки
plt.imshow(data[0])
```

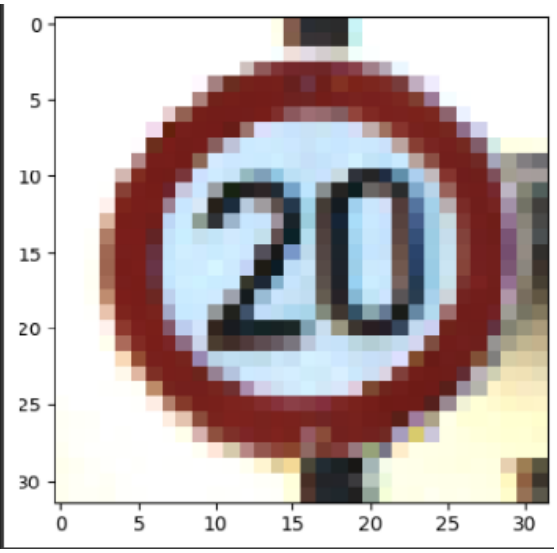


Рисунок 3- Отображение первого элемента

Далее необходимо разделить данные на тренировочный и тестовый набор 70/30

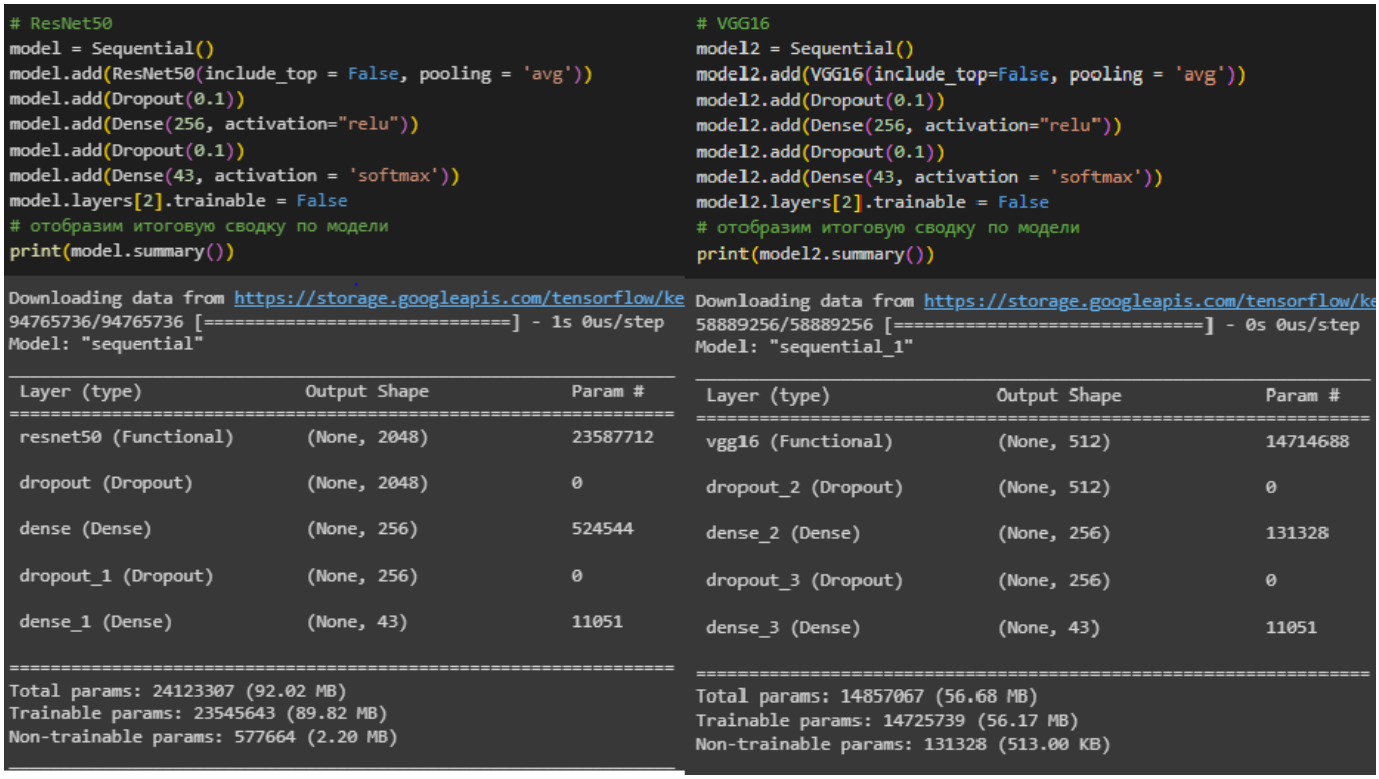
```python
# выполним разделение данных на треннировочный и тестовый набор 70/30
x_train, x_val, y_train, y_val = train_test_split(data, labels, test_size=0.3, random_state=1)

# отобразим размерности обучающего и тестового набора и первый элемент
print("training shape: ",x_train.shape, y_train.shape)
print("testing shape: ",x_val.shape, y_val.shape)
print(y_train[0])
```

```
training shape:  (27446, 32, 32, 3) (27446, 43)
testing shape:  (11763, 32, 32, 3) (11763, 43)
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Рисунок 4- Разделение данных

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

Создаем две модели нейронных сетей а именно ResNet50 и VGG16:



```
# ResNet50
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
# отобразим итоговую сводку по модели
print(model.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/ke
94765736/94765736 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 256)               524544

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 43)                11051

=================================================================
Total params: 24123307 (92.02 MB)
Trainable params: 23545643 (89.82 MB)
Non-trainable params: 577664 (2.20 MB)
_________________________________________________________________
```

```
# VGG16
model2 = Sequential()
model2.add(VGG16(include_top=False, pooling = 'avg'))
model2.add(Dropout(0.1))
model2.add(Dense(256, activation="relu"))
model2.add(Dropout(0.1))
model2.add(Dense(43, activation = 'softmax'))
model2.layers[2].trainable = False
# отобразим итоговую сводку по модели
print(model2.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/ke
58889256/58889256 [==============================] - 0s 0us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 512)               14714688

 dropout_2 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 256)               131328

 dropout_3 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 43)                11051

=================================================================
Total params: 14857067 (56.68 MB)
Trainable params: 14725739 (56.17 MB)
Non-trainable params: 131328 (513.00 KB)
_________________________________________________________________
```

Рисунок 5- модели нейронной сети

Далее для этих моделей нужно построить графики, а также заполнить таблицу. Графики отражают зависимость метрики от эпохи для тренировочного и тестового наборов. В них отображена точность и потери.

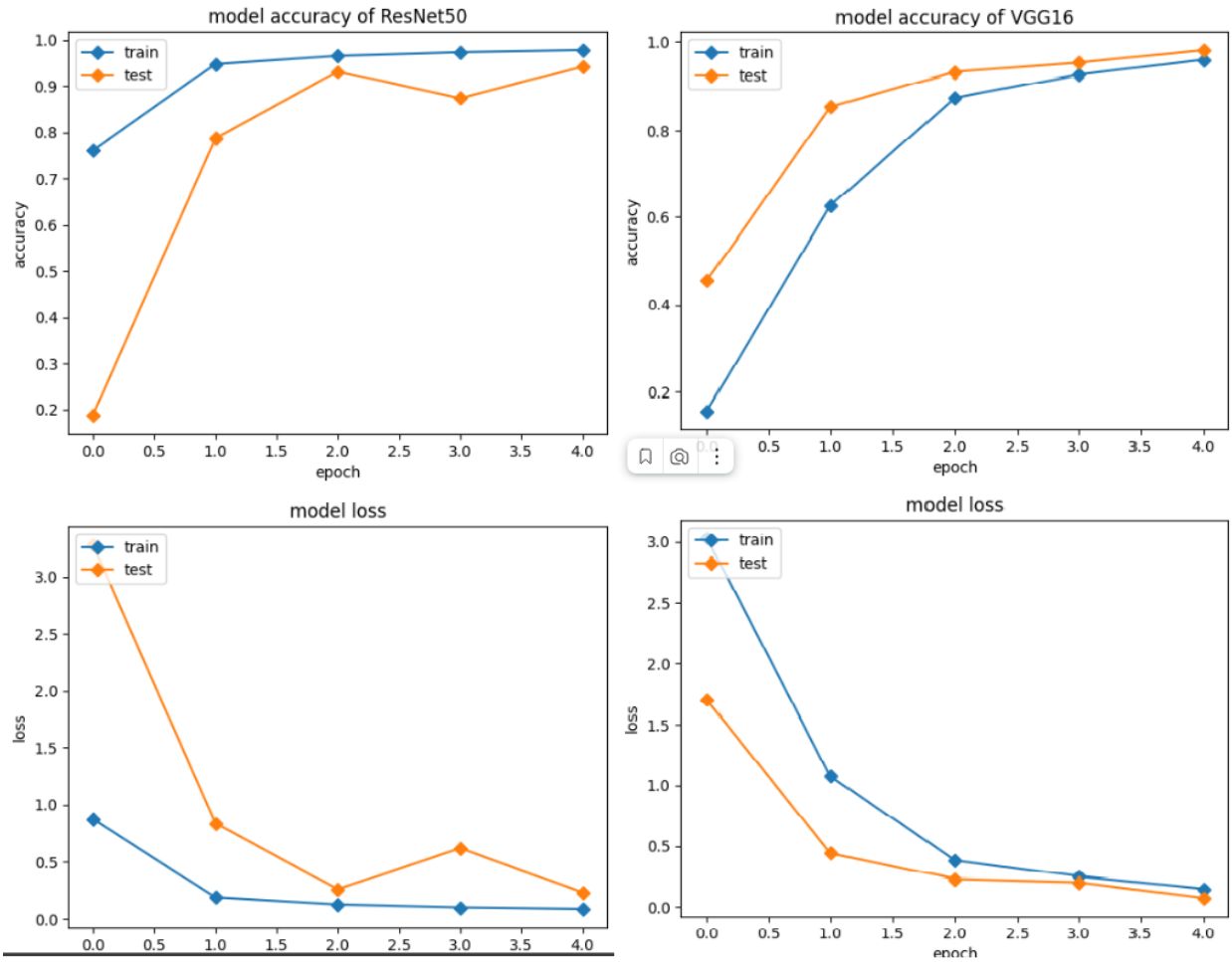


Рисунок 6- Графики для двух моделей

Таблица точности.

Модель	Обучение	Валидация	Тест
Resnet50	97.8467	94.2872	98.3848
VGG16	98.7588	99.6004	98.7588

Задание 2. Применить нецелевую атаку уклонения на основе белого ящика против моделей глубокого обучения. Реализовать следующие типы атак: Fast Gradient Sign Method (FGSM) и Projected Gradient Descent (PGD). Необходимо использовать следующие значения параметра искажения: $\epsilon\epsilon$= [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255].

Создаем атаки Fast Gradient Sign Method (FGSM) и Projected Gradient Descent (PGD)

```python
# 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}) # уствновка нового значения eps
    print(f"Eps: {eps}")
    x_test_adv = attack_fgsm.generate(x_test, y_test) # генерация адверсариальных
    # примеров для тестового набора данных
    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}")
```

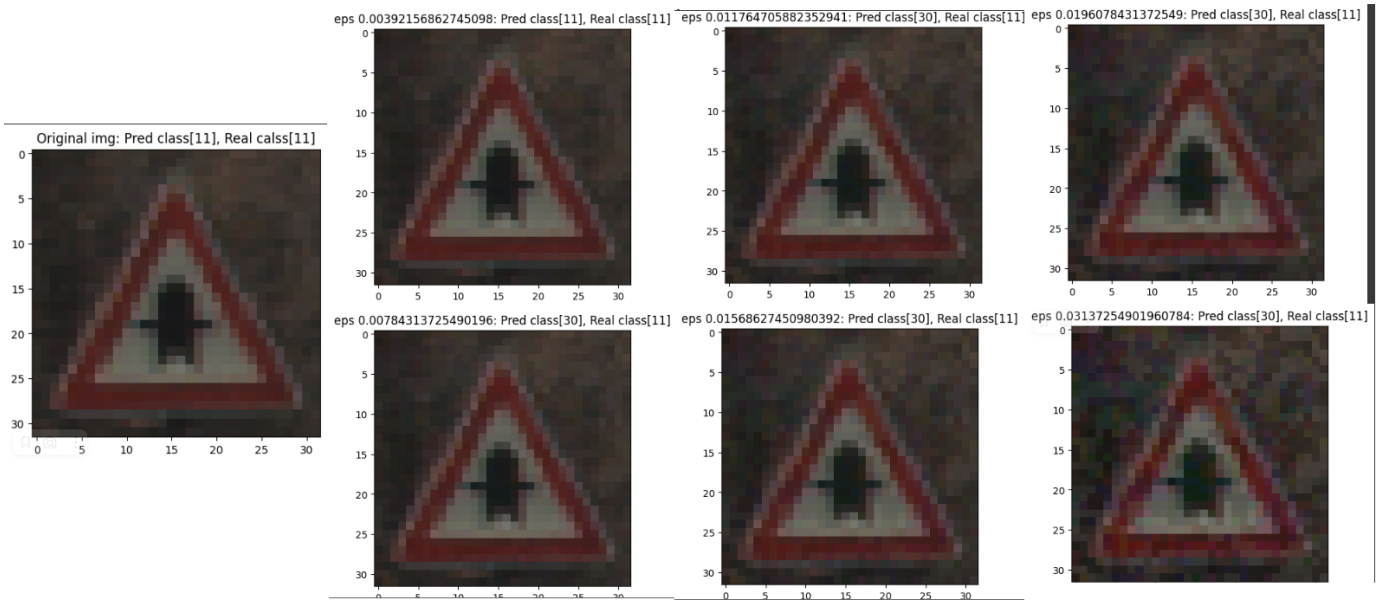
```python
# создаем атаку PGD
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3, max_iter=4, verbose=False)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
true_accuracies = [] # для точности оригинальных данных
adv_accuracises_pgd = []
true_losses = [] # 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}")
```

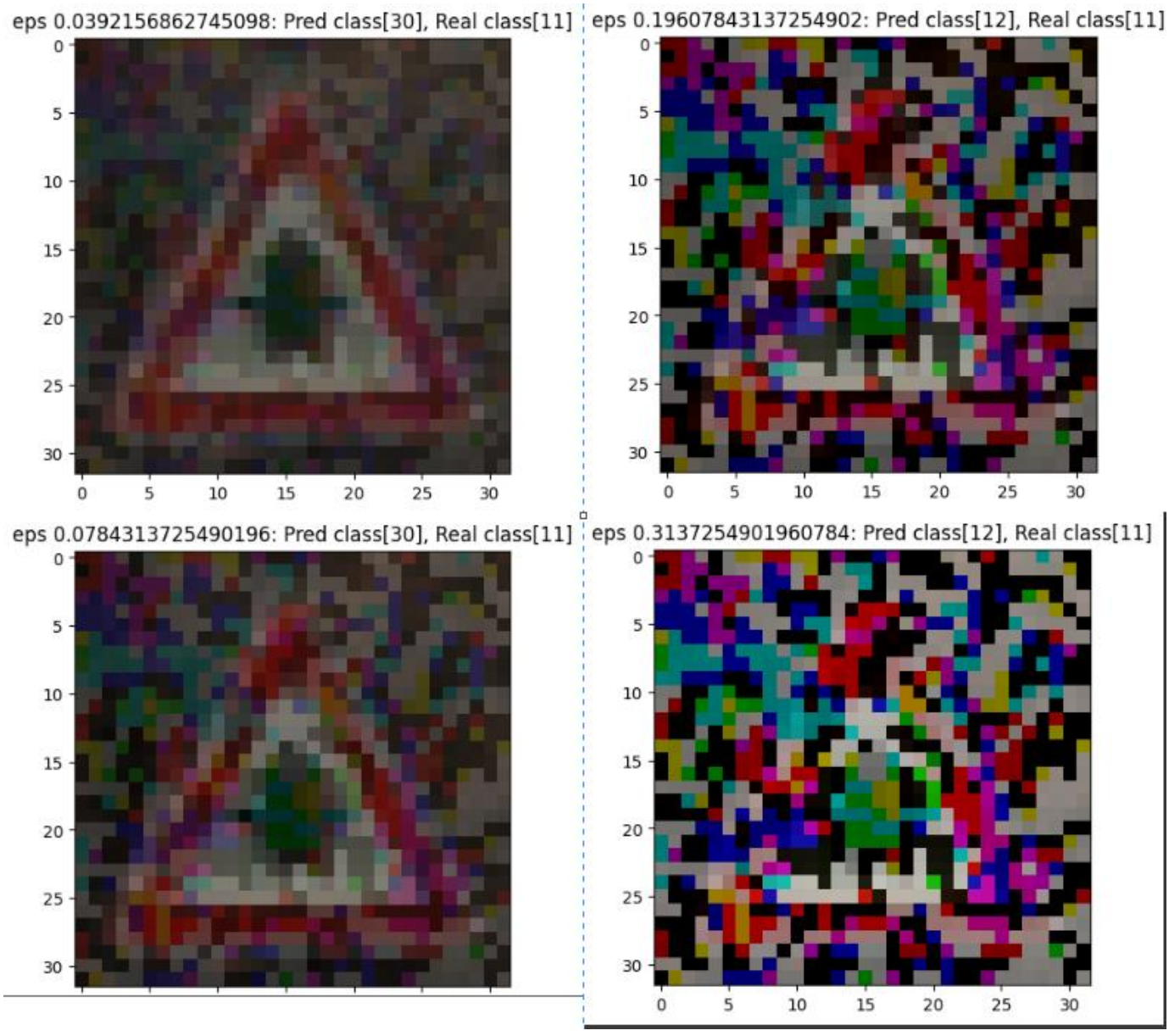
Рисунок 7- FGSM PGD

ResNet50

Далее потребуется переопределить eps для этого выведем изображение под номером 4 после атаки FGSM.

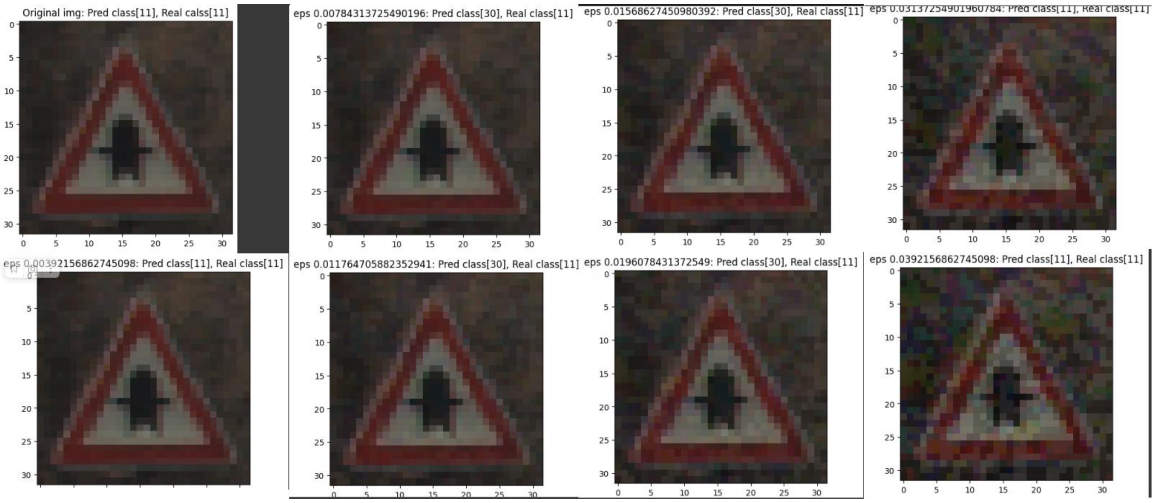
```python
# отображаем исходные и адверсариальные изображения для разных значений eps
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
pred = np.argmax(model.predict(x_test[4:5]))
plt.figure(4)
plt.title(f"Original img: Pred class[{pred}], Real calss[{np.argmax(y_test[4])}]")
plt.imshow(x_test[4])
plt.show()
i = 1
# проходимся по каждому eps из заданного диапазона
for eps in eps_range:
    attack_pgd.set_params(**{'eps': eps})
    x_test_adv = attack_pgd.generate(x_test, y_test)
    pred = np.argmax(model.predict(x_test_adv[4:5]))
    plt.figure(i)
    plt.title(f"eps {eps}: Pred class[{pred}], Real class[{np.argmax(y_test[4])}]")
    plt.imshow(x_test_adv[4])
    plt.show()
    i += 1
```

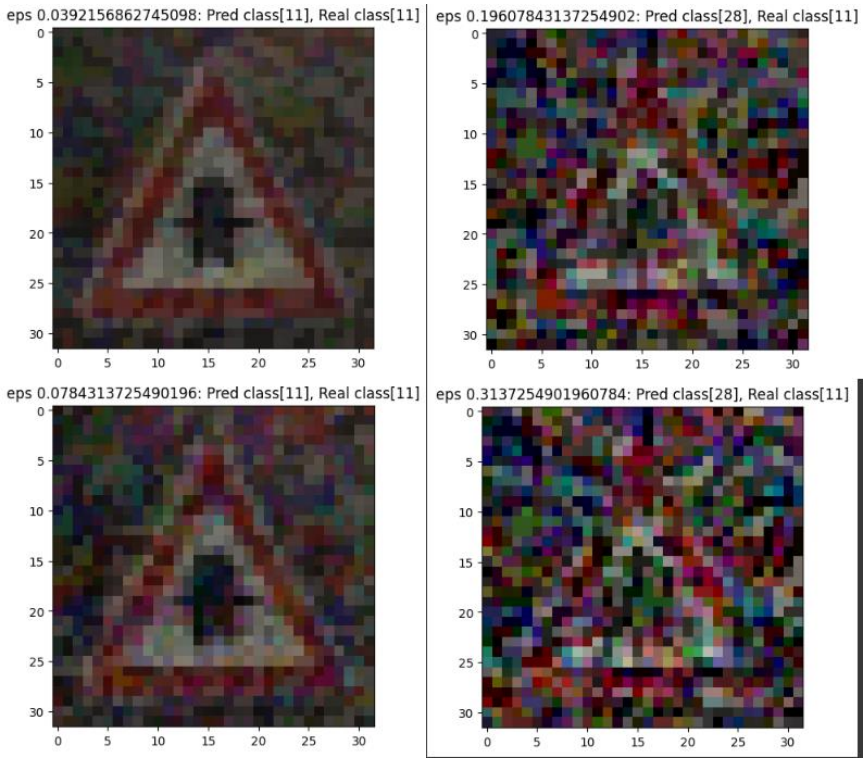

Original img: Pred class[11], Real calss[11]
eps 0.00392156862745098: Pred class[11], Real class[11]
eps 0.011764705882352941: Pred class[30], Real class[11]
eps 0.0196078431372549: Pred class[30], Real class[11]
eps 0.0078431372549196: Pred class[30], Real class[11]
eps 0.0156862745098039: Pred class[30], Real class[11]
eps 0.03137254901960784: Pred class[30], Real class[11]


eps 0.0392156862745098: Pred class[30], Real class[11]
eps 0.19607843137254902: Pred class[12], Real class[11]
eps 0.0784313725490196: Pred class[30], Real class[11]
eps 0.3137254901960784: Pred class[12], Real class[11]

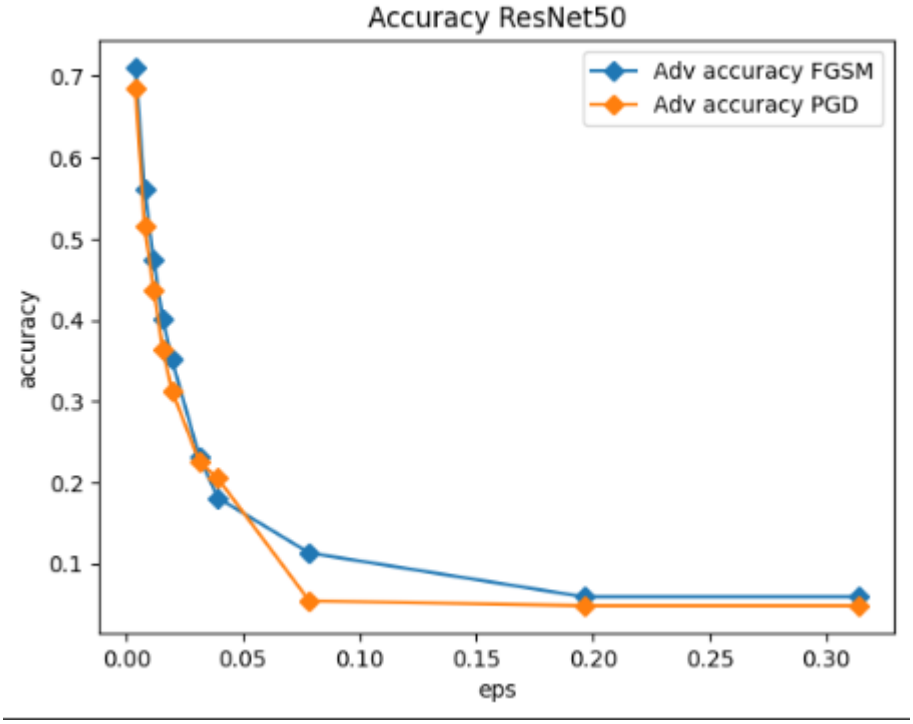
Далее потребуется переопределить eps для этого выведем изображение под номером 4 после атаки PGD.

```
# 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[4:5]))
plt.figure(4)
plt.title(f"Original img: Pred class[{pred}], Real calss[{np.argmax(y_test[4])}]")
plt.imshow(x_test[4])
plt.show()
i = 1
# проходимся по каждому eps из заданного диапазона
for eps in eps_range:
    attack_pgd.set_params(**{'eps': eps})
    x_test_adv = attack_pgd.generate(x_test, y_test)
    pred = np.argmax(model.predict(x_test_adv[4:5]))
    plt.figure(i)
    plt.title(f"eps {eps}: Pred class[{pred}], Real class[{np.argmax(y_test[4])}]")
    plt.imshow(x_test_adv[4])
    plt.show()
    i += 1
```





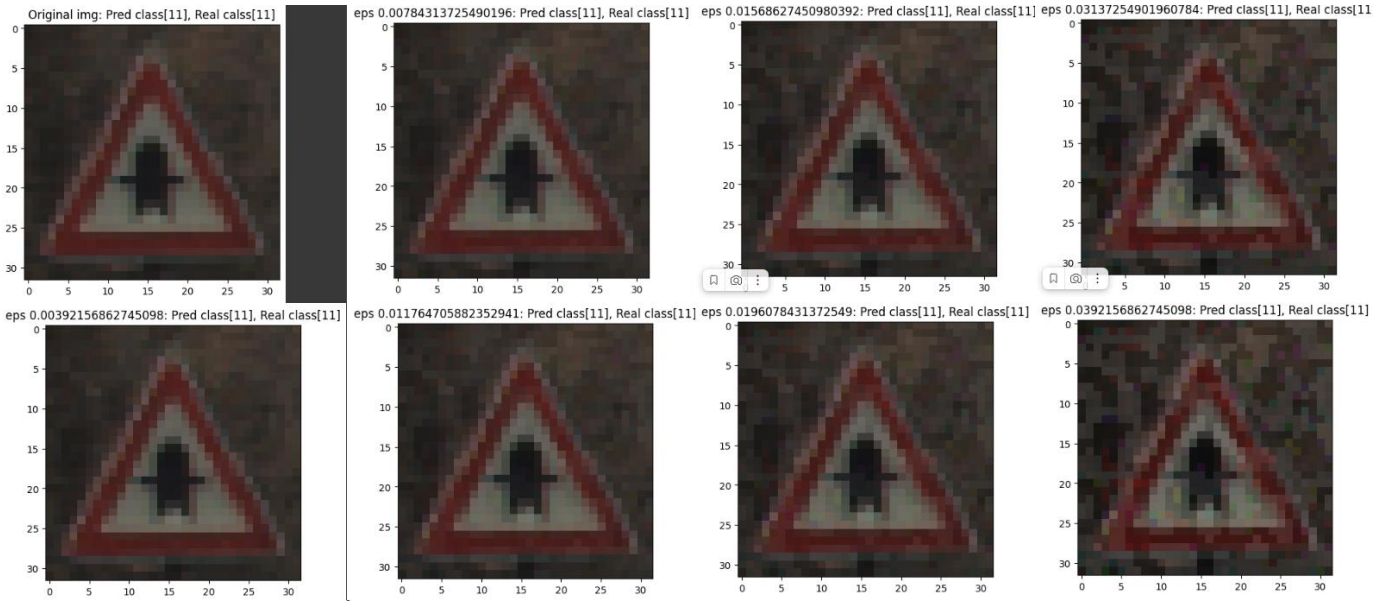
ResNet50: График зависимости точности классификации от параметра искажения.

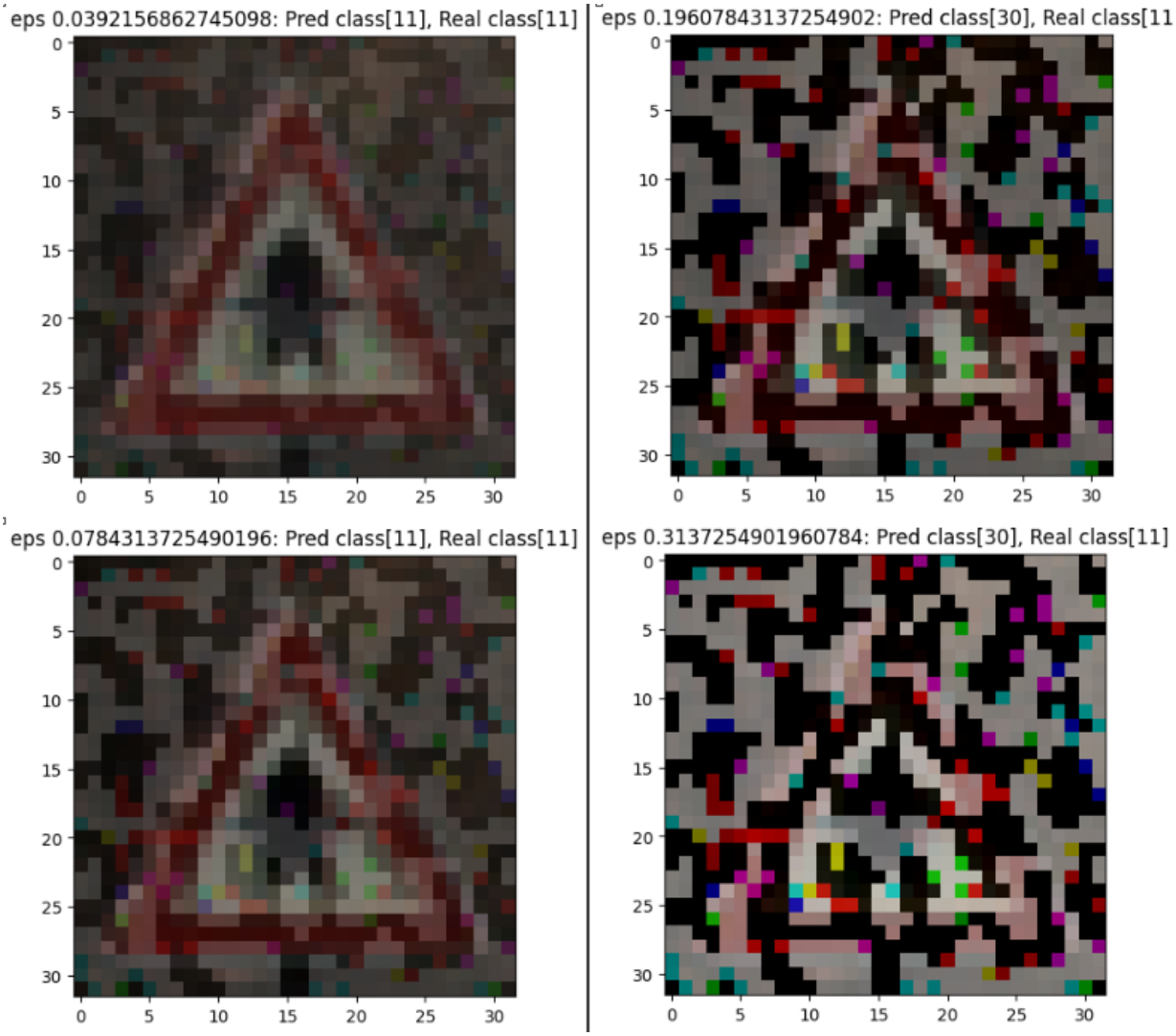


VGG16

Далее потребуется переопределить eps для этого выведем изображение под номером 4 после атаки FGSM.

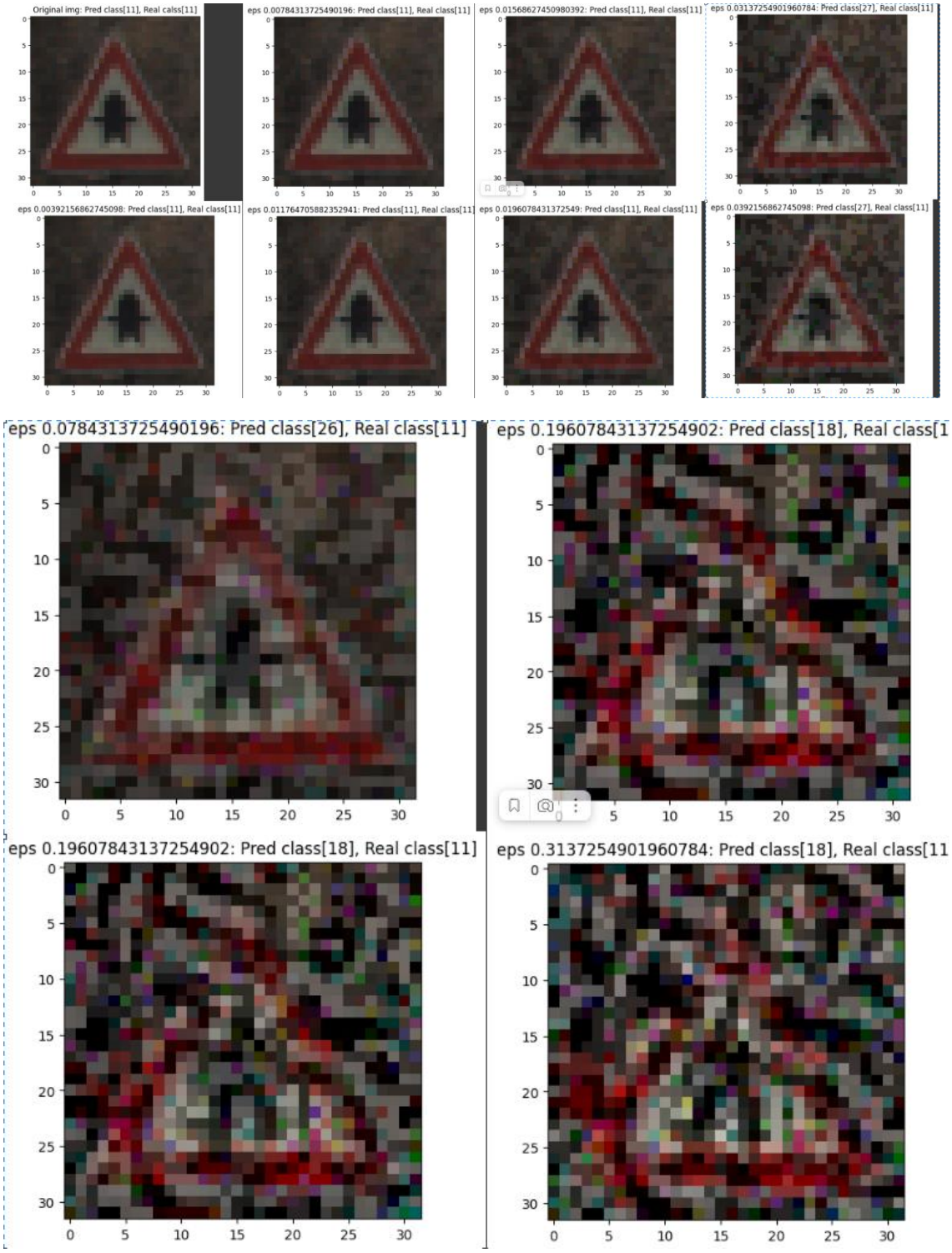
```
# отображаем исходные и адверсариальные изображения для разных значений eps
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
pred = np.argmax(model.predict(x_test[4:5]))
plt.figure(0)
plt.title(f"Original img: Pred class[{pred}], Real calss[{np.argmax(y_test[4])}]")
plt.imshow(x_test[4])
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"eps {eps}: Pred class[{pred}], Real class[{np.argmax(y_test[4])}]")
    plt.imshow(x_test_adv[4])
    plt.show()
    i += 1
```



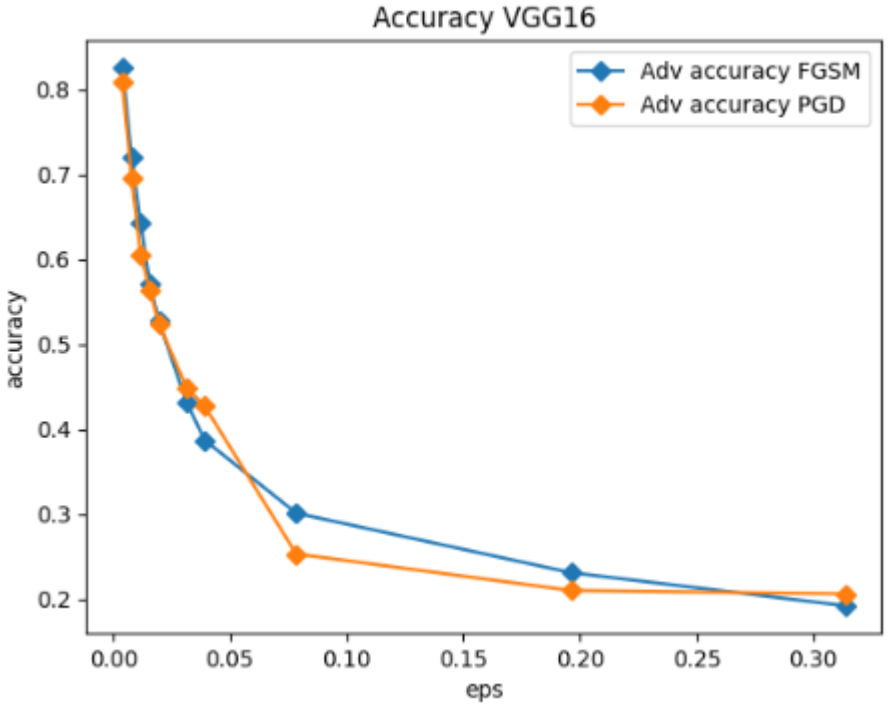


Далее потребуется переопределить eps для этого выведем изображение под номером 4 после атаки PGD.

```
[ ]  # 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[4:5]))
     plt.figure(0)
     plt.title(f"Original img: Pred class[{pred}], Real calss[{np.argmax(y_test[4])}]")
     plt.imshow(x_test[4])
     plt.show()
     i = 1

     for eps in eps_range:
         attack_pgd.set_params(**{'eps': eps})
         x_test_adv = attack_pgd.generate(x_test, y_test)
         pred = np.argmax(model.predict(x_test_adv[4:5]))
         plt.figure(i)
         plt.title(f"eps {eps}: Pred class[{pred}], Real class[{np.argmax(y_test[4])}]")
         plt.imshow(x_test_adv[4])
         plt.show()
         i += 1
```



VGG16 : График зависимости точности классификации от параметра искажения.

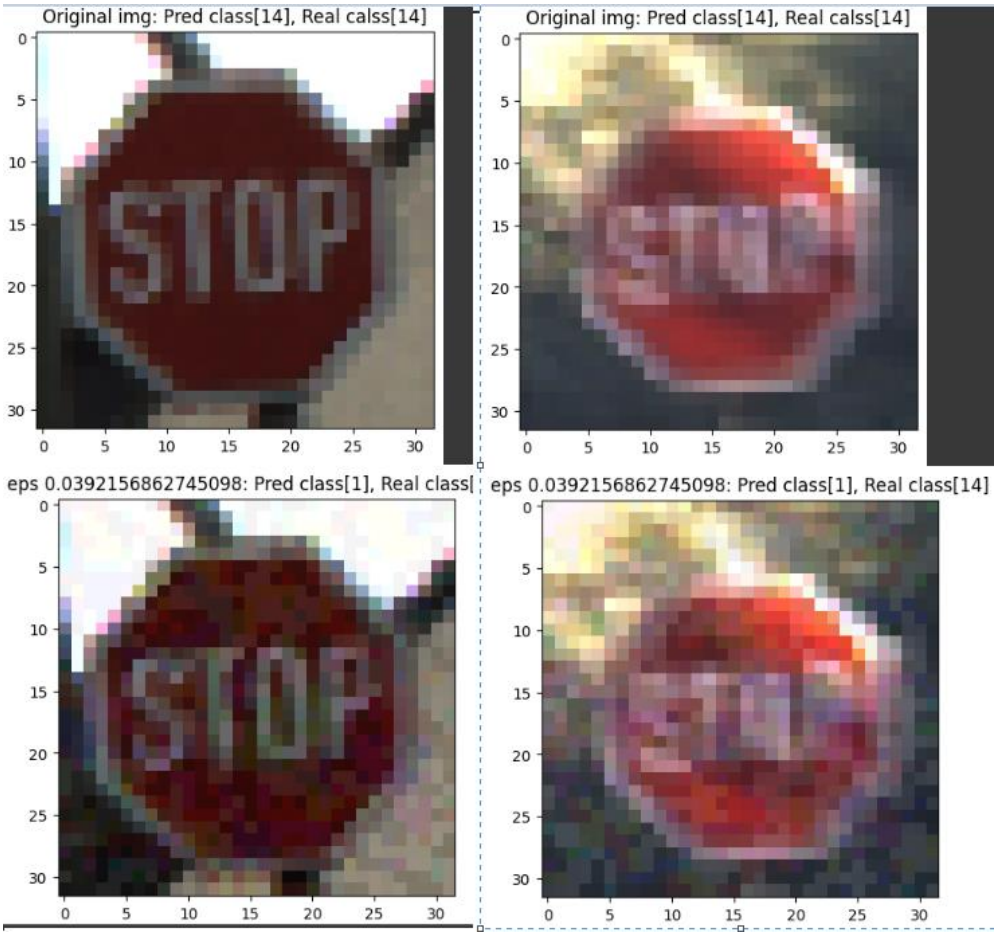


Результаты.

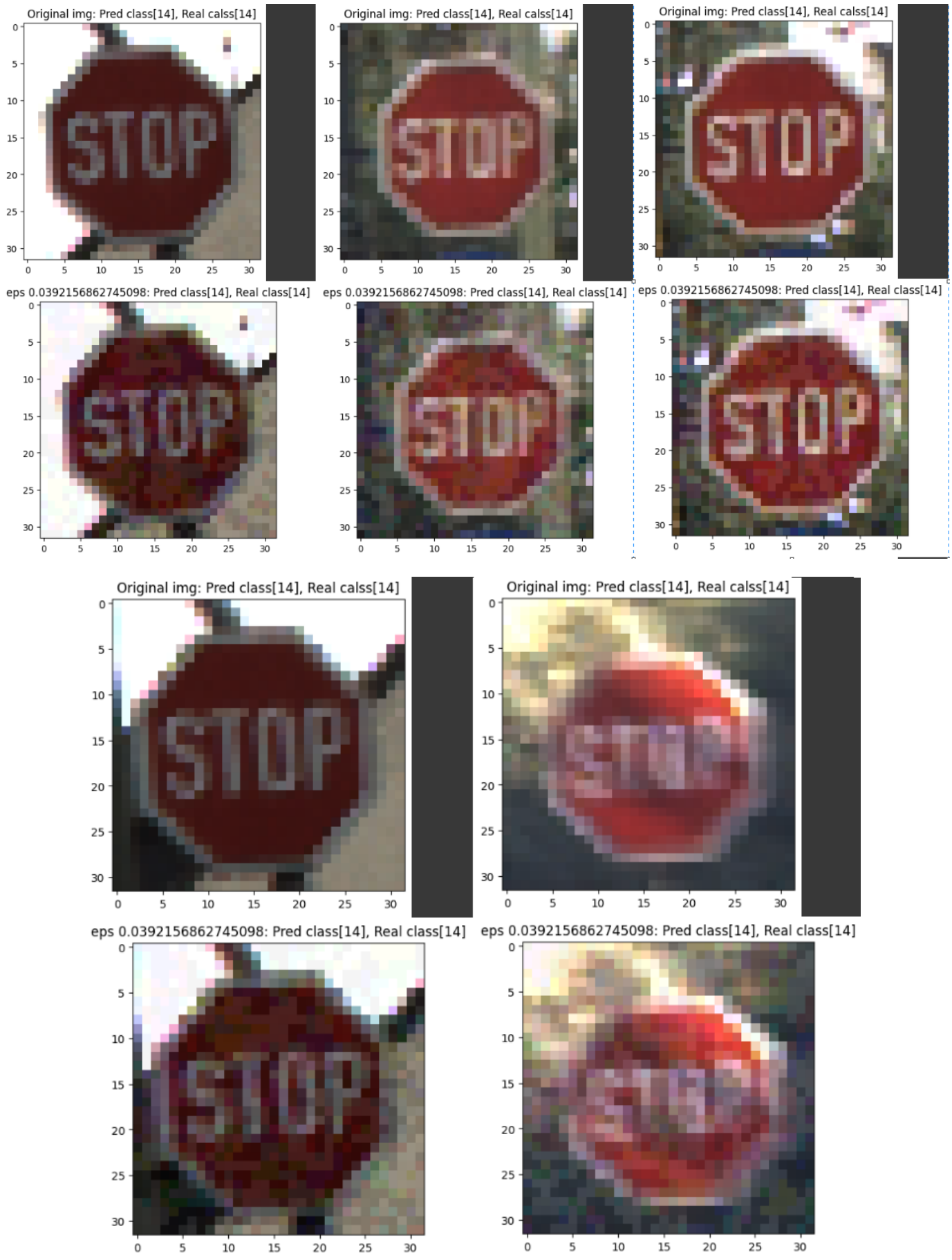
Model	Original accuracy	eps = 1/255	eps = 5/255	eps = 10/255
Resnet50 FGSM	97.8467	71.1	35.1	18.1
Resnet50 PGD	97.8467	68.6	31.3	20.5
VGG16 FGSM	98.7588	82.6	52.7	38.7
VGG16 PGD	98.7588	80.8	52.4	42.7

Задание 3: Применение целевой атаки уклонения методом белого против моделей глубокого обучения. Используйте изображения знака «Стоп» (label class 14) из тестового набора данных.

Реализуем целевую атаку FGSM.



Реализуем целевую атаку PGD.



Искажение	PGD- attack – Stop sign images	FGSM- attack – Stop sign images
ϵ=1/255	97%	99%
ϵ=3/255	92%	84%
ϵ=5/255	90%	54%
ϵ=10/255	82%	31%
ϵ=20/255	59%	1%
ϵ=50/255	34%	0%
ϵ=80/255	1%	0%

Вывод: Метод FGSM плохо подходит для целевых атак. При растущем искажении классификация начинает давать сбой. Для целевой атаки подходит PGD. Хоть и с сильным искажением, но модель будет определять заданный нами класс.