МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

Кафедра: КБ-4 «Киберразведка и противодействие угрозам с применением технологий искусственного интеллекта»

**Лабораторная работа №2 по дисциплине**

**«Анализ защищенности систем искусственного интеллекта»**

Выполнил:

Филимонов И.М.

Группа:

ББМО-01-22, 2 курс

Проверил:

Спирин А.А.

Москва, 2024 г.

**Задание 1.**

Установим adversarial-robustness-toolbox.

16 сек.

```
!pip install adversarial-robustness-toolbox
```

```
Collecting adversarial-robustness-toolbox
  Downloading adversarial_robustness_toolbox-1.17.1-py3-none-any.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 7.5 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from adversaria
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from adversarial
Requirement already satisfied: scikit-learn>=0.22.2 in /usr/local/lib/python3.10/dist-packages (from adv
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from adversarial-robustne
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from adversarial-r
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from adversarial-robustr
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-lea
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from sci
Installing collected packages: adversarial-robustness-toolbox
Successfully installed adversarial-robustness-toolbox-1.17.1
```

Импортируем необходимые библиотеки

15 сек.

```
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
%matplotlib inline
```

Скачаем датасет, загрузим его на google диск, затем подключим google диск к google colab и разархивируем архив с датасетом.

42 сек.

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

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

25 сек.

```
[4] zip_file = '/content/drive/MyDrive/dataset/archive.zip'
z = zipfile.ZipFile(zip_file, 'r')
z.extractall()

print(os.listdir())
```

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

Создадим модель ResNET50. Разделим данные на обучающие и тестовые в соотношении 70/30. Отобразим размерность обучающего и тестового набора.

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

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. 0.]
```

```
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,
94765736/94765736 [==============================] - 0s 0us/step
```

Отобразим сводку по модели.

```
print(model.summary())
```

```
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)
_________________________________________________________________
None
```

Создадим модель 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
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-ap
58889256/58889256 [==============================] - 0s 0us/step
```

Отобразим сводку по модели.

```
print(model2.summary())
```

```
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)
_________________________________________________________________
None
```

Отобразим таблицы точности для тренировочного, валидационного и тестового наборов.

```
from tabulate import tabulate

train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
test_accuracy = history_test.history['accuracy']

train_accuracy2 = history2_test.history['accuracy']
val_accuracy2 = history2_test.history['val_accuracy']
test_accuracy2 = history2_test.history['accuracy']

table = [["Model","Training Accuracy","Validation Accuracy","Test Accuracy"],
            ["Resnet50",train_accuracy[4]*100,val_accuracy[4]*100,test_accuracy[4]*100],
            ["VGG16",train_accuracy2[4]*100,val_accuracy2[4]*100,test_accuracy2[4]*100]]

table1 = tabulate(table,headers="firstrow",tablefmt="grid")
print(table1)
```

```
+----------+---------------------+-----------------------+-----------------+
| Model    |   Training Accuracy |   Validation Accuracy |   Test Accuracy |
+==========+=====================+=======================+=================+
| Resnet50 |             97.9997 |               95.6984 |         98.2827 |
+----------+---------------------+-----------------------+-----------------+
| VGG16    |             98.2657 |               99.3199 |         98.2657 |
+----------+---------------------+-----------------------+-----------------+
```

Построим графики точности и потерь для моделей.

График точности ResNet50.

```
plt.plot(history.history['accuracy'], marker='D')
plt.plot(history.history['val_accuracy'], marker='D')
plt.title('model accuracy of ResNet50')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

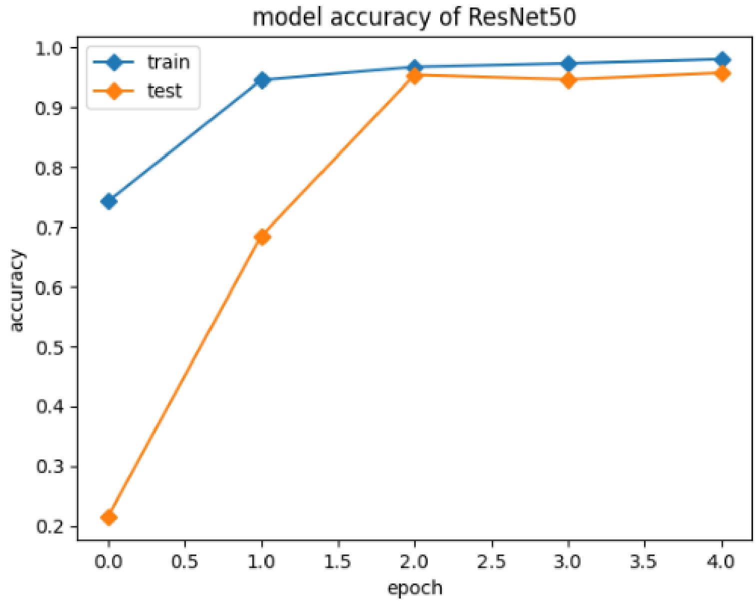



График потерь ResNet50.

```
plt.plot(history.history['loss'], marker='D')
plt.plot(history.history['val_loss'], marker='D')
plt.title('model loss of ResNet50')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

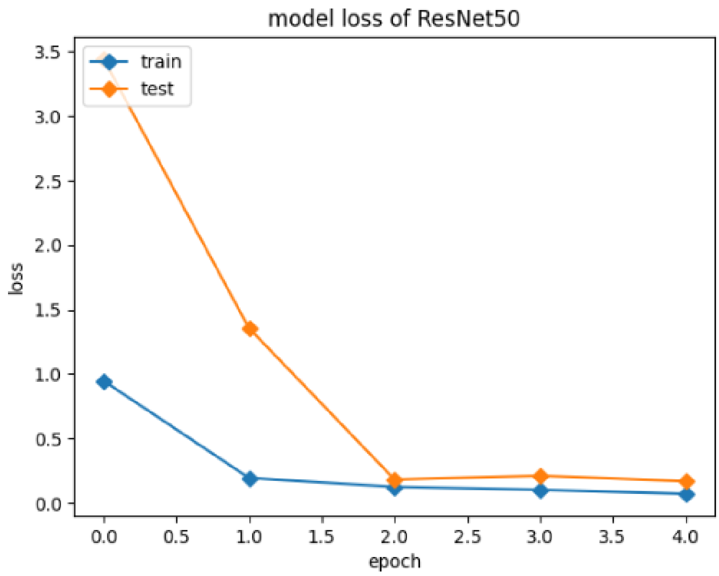


График точности VGG16.

```
plt.plot(history2.history['accuracy'], marker='D')
plt.plot(history2.history['val_accuracy'], marker='D')
plt.title('model accuracy of VGG16')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

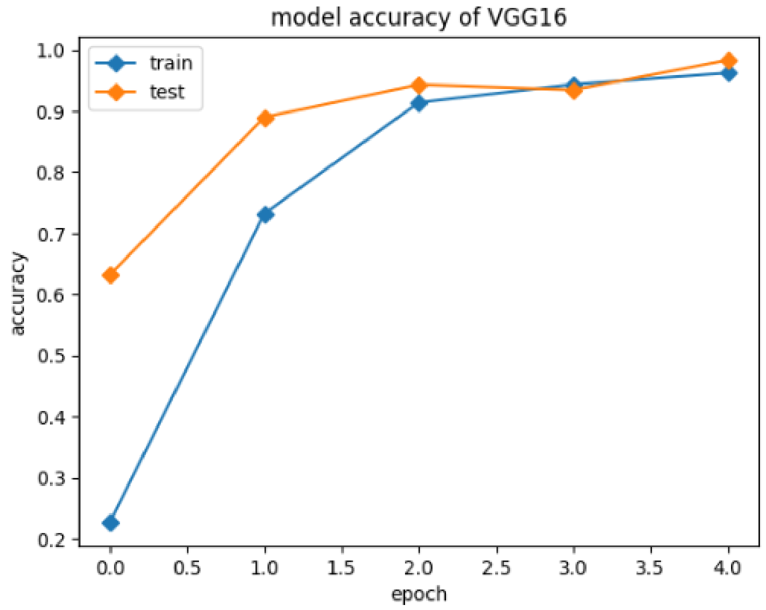
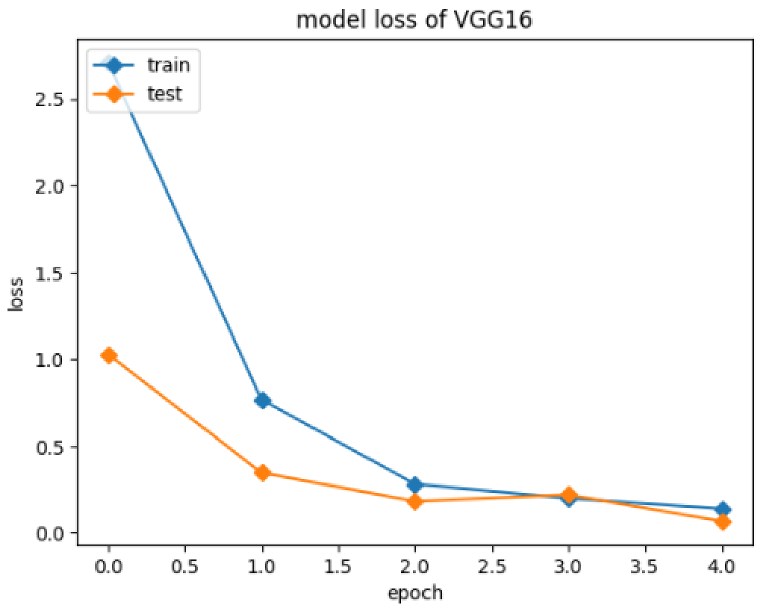



График потерь VGG16.

```
plt.plot(history2.history['loss'], marker='D')
plt.plot(history2.history['val_loss'], marker='D')
plt.title('model loss of VGG16')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

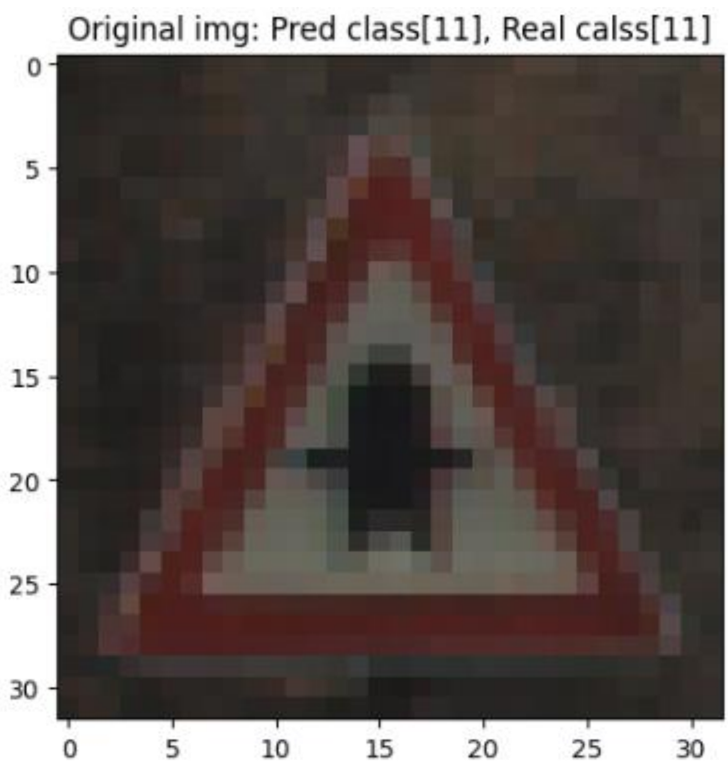


## Задание 2.

Проведем атаку FGSM с параметром искажения [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255].

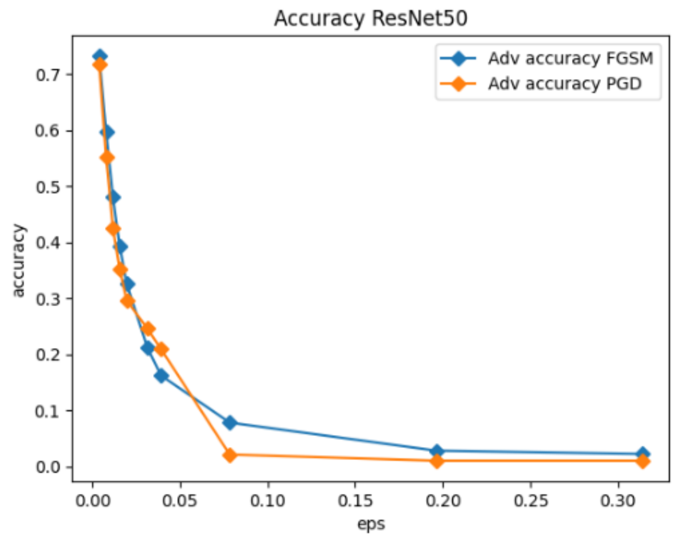
```python
attack_fgsm = FastGradientMethod(estimator=classifier, eps=0.3)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
true_accuracies = [] # для точности оригинальных данных
adv_accuracises_fgsm = []
true_losses = [] # для потерь на оригинальных данных
adv_losses_fgsm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

for eps in eps_range:
    attack_fgsm.set_params(**{'eps': eps}) # 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}")
```

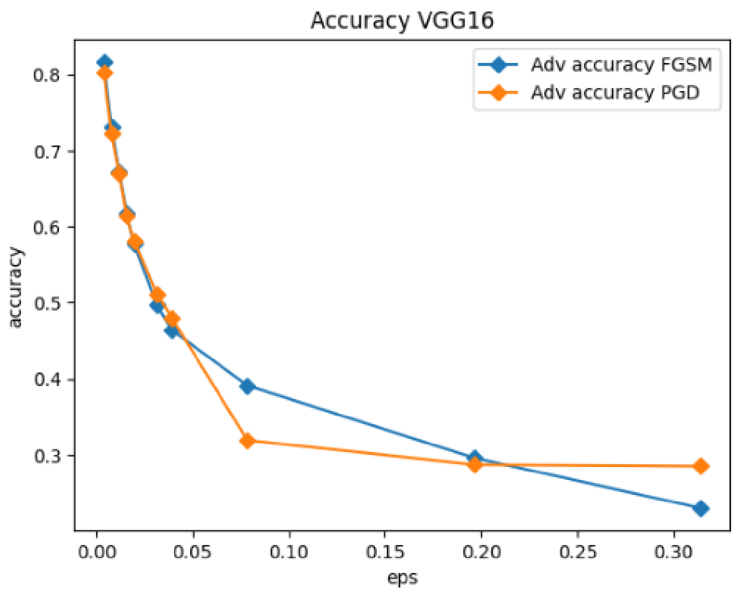
Проведем атаку PGD с параметром искажения [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255].



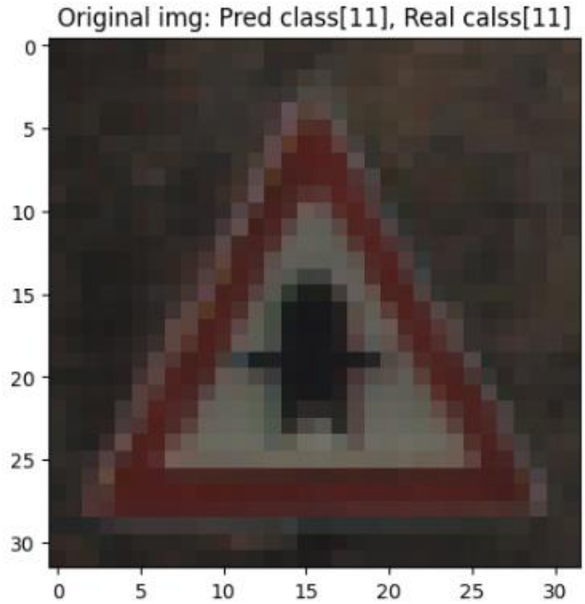
```python
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3, max_iter=4, verbose=False)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
true_accuracies = [] # для точности оригинальных данных
adv_accuracises_pgd = []
true_losses = [] # для потерь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

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

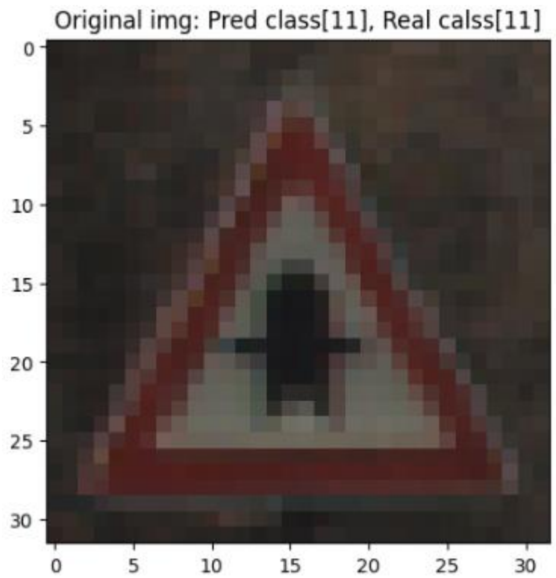
Построим графики зависимости точности классификации от параметра искажения для FGSM и PGD на датасеты REsNET50 и VGG16.

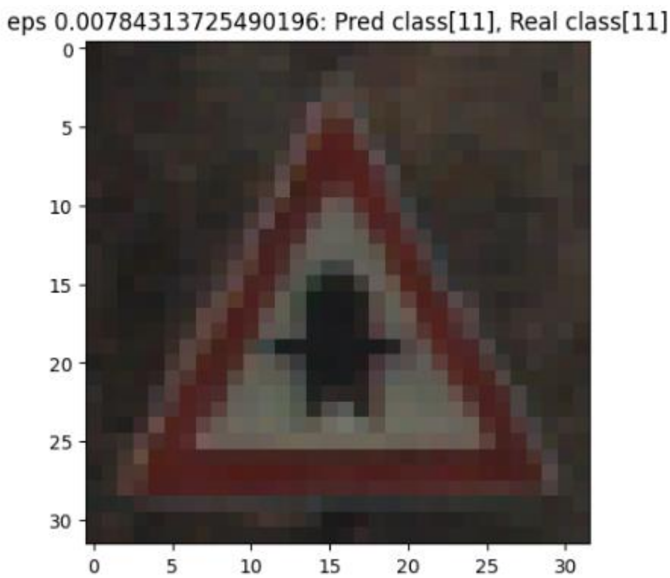


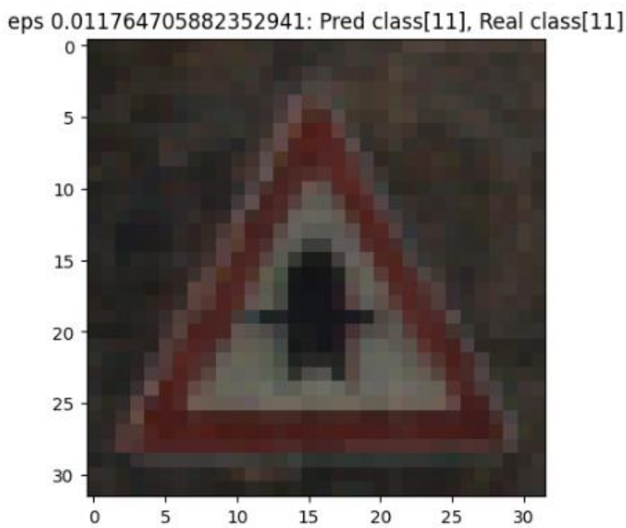



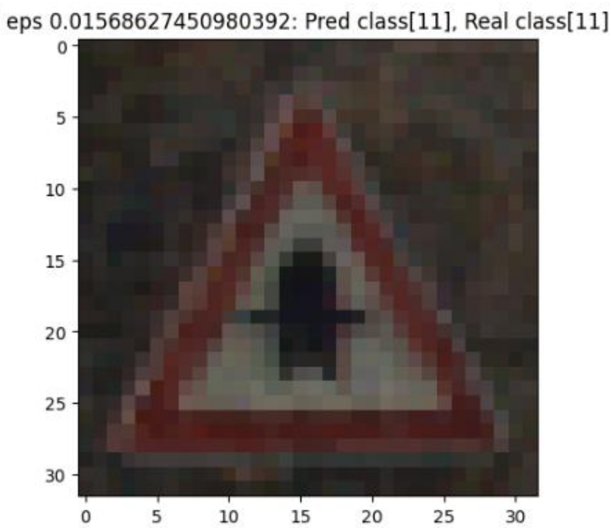
Отобразим исходное изображение из датасета и атакующее изображение.

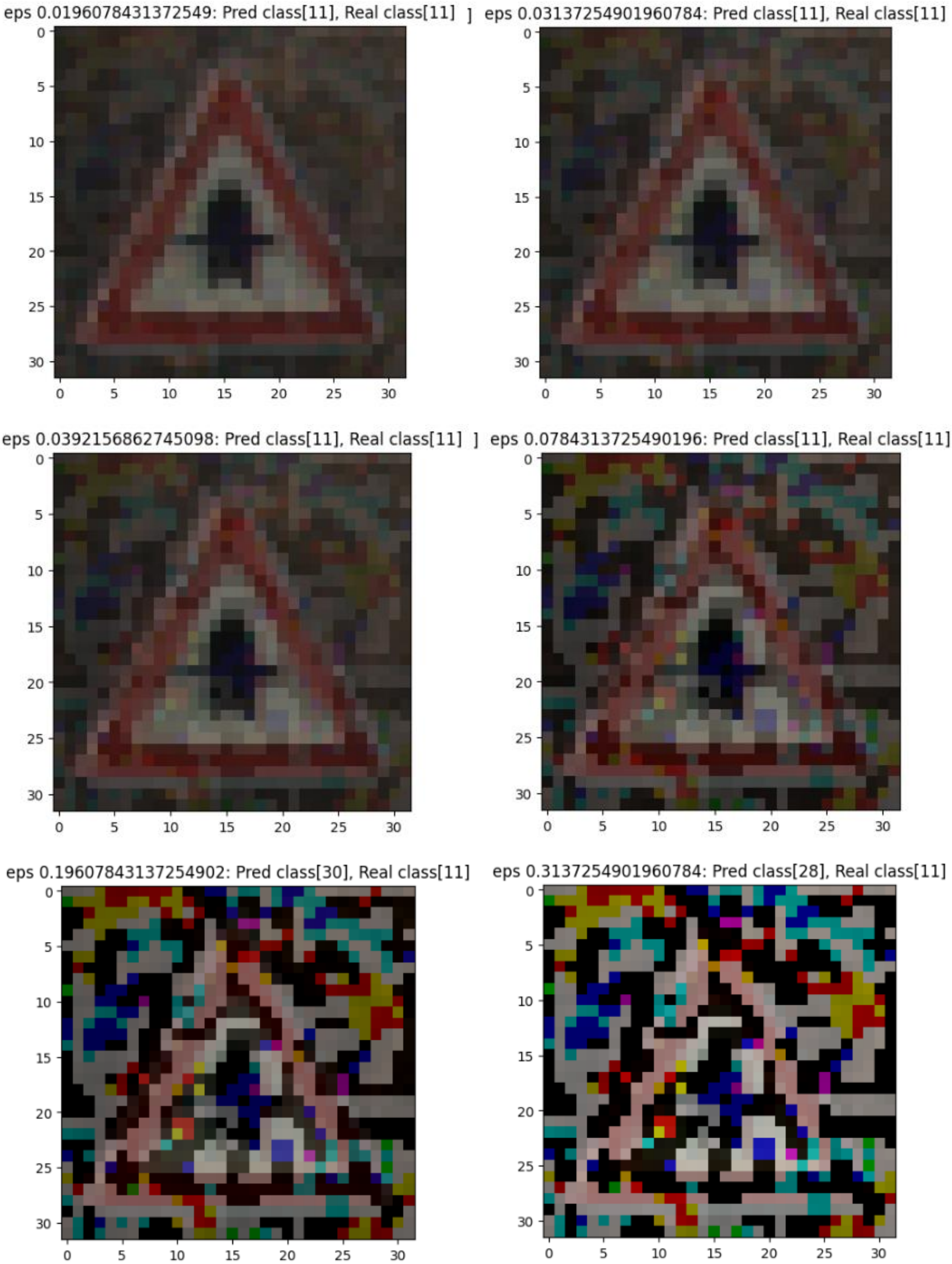


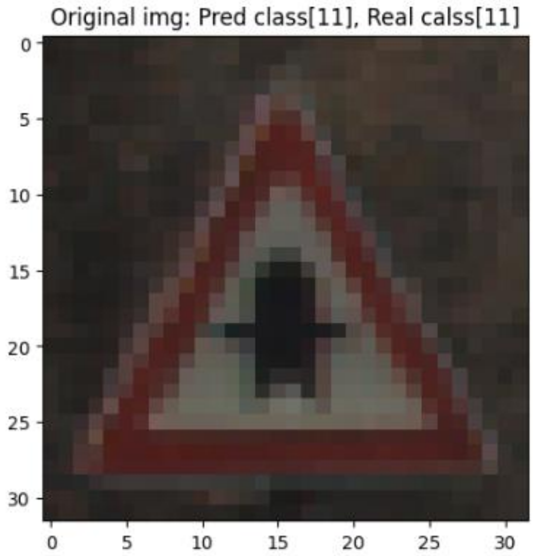



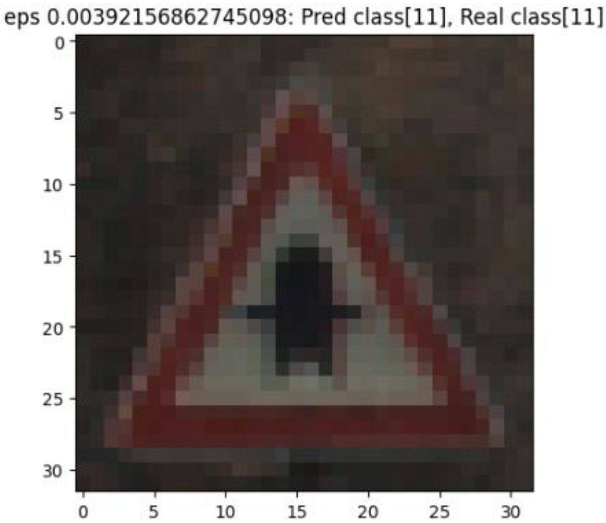



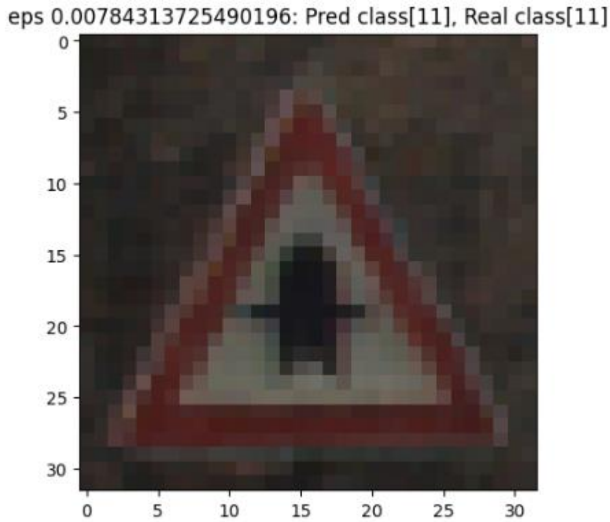


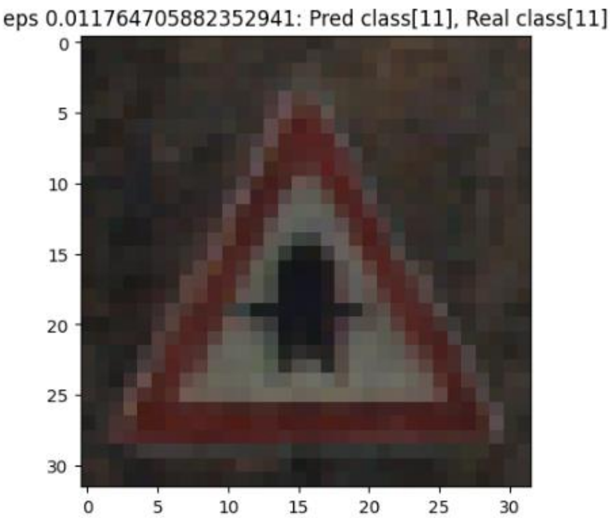
eps 0.0196078431372549: Pred class[11], Real class[11]
eps 0.03137254901960784: Pred class[11], Real class[11]
eps 0.0392156862745098: Pred class[11], Real class[11]
eps 0.0784313725490196: Pred class[11], Real class[11]
eps 0.19607843137254902: Pred class[30], Real class[11]
eps 0.3137254901960784: Pred class[28], Real class[11]

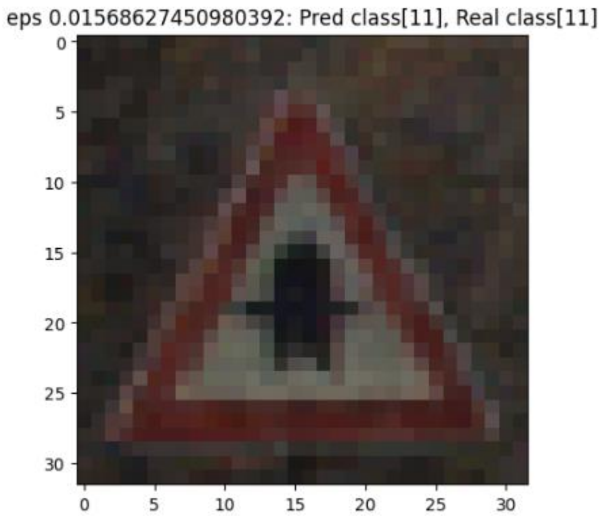
PGD.


Original img: Pred class[11], Real calss[11]


eps 0.00392156862745098: Pred class[11], Real class[11]


eps 0.00784313725490196: Pred class[11], Real class[11]


eps 0.011764705882352941: Pred class[11], Real class[11]


eps 0.01568627450980392: Pred class[11], Real class[11]

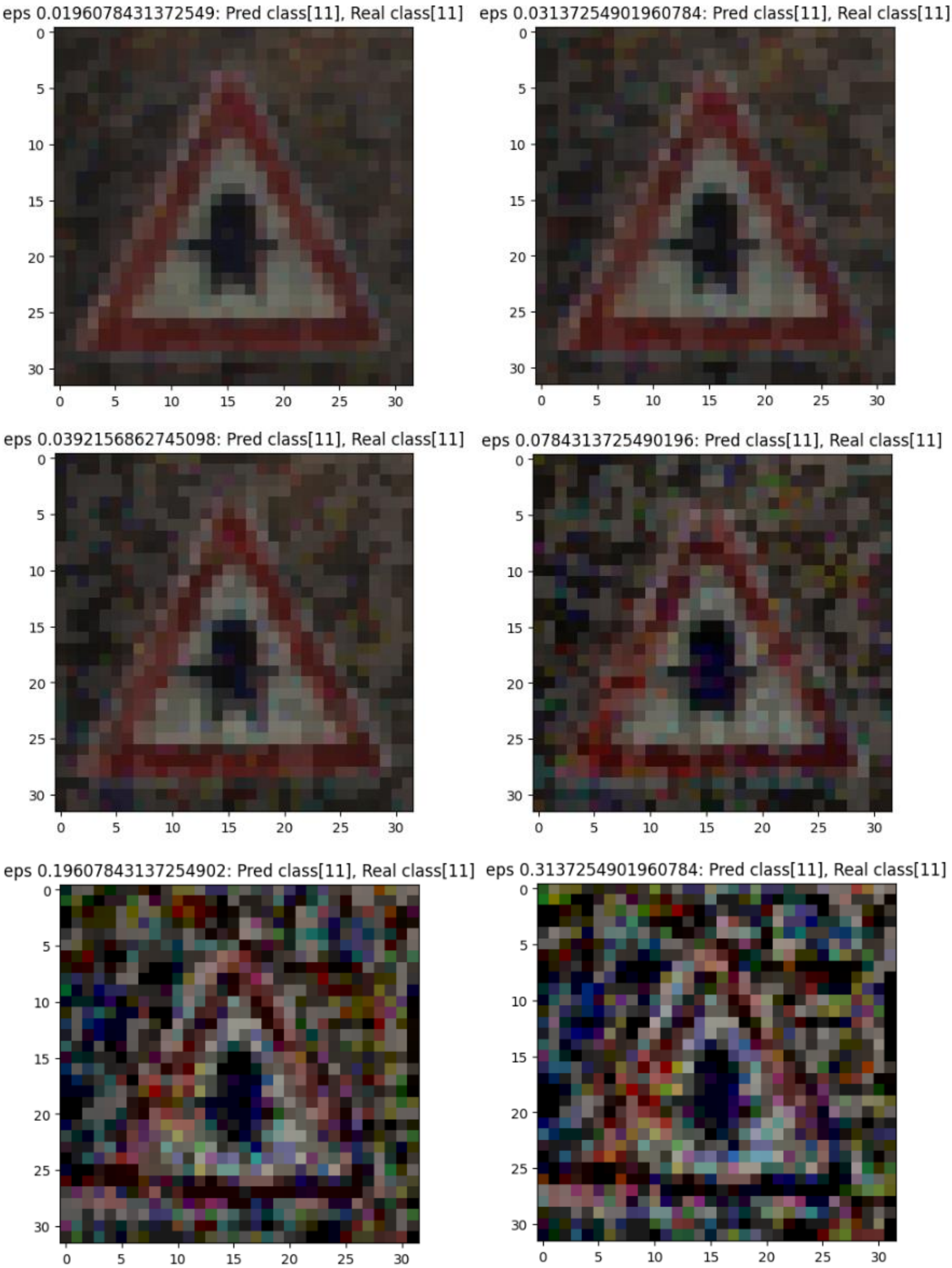

eps 0.0196078431372549: Pred class[11], Real class[11]
eps 0.03137254901960784: Pred class[11], Real class[11]
eps 0.0392156862745098: Pred class[11], Real class[11]
eps 0.0784313725490196: Pred class[11], Real class[11]
eps 0.19607843137254902: Pred class[11], Real class[11]
eps 0.3137254901960784: Pred class[11], Real class[11]

Таблица значений точности для обеих моделей

| Model | Original accuracy | eps = 1/255 | eps = 2/255 | eps = 3/255 | eps = 4/255 | eps = 5/255 | eps = 8/255 | eps = 10/255 | eps = 20/255 | eps = 50/255 | eps = 80/255 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Resnet50 FGSM | 98.2475 | 73.3 | 59.8 | 48 | 39.2 | 32.6 | 21.2 | 16.2 | 7.8 | 2.8 | 2.2 |
| Resnet50 PGD | 98.2475 | 71.7 | 55.3 | 42.5 | 35.1 | 29.6 | 24.6 | 20.9 | 2.1 | 1 | 1 |
| VGG16 FGSM | 97.8577 | 81.7 | 73.2 | 67.2 | 61.8 | 57.7 | 49.7 | 46.4 | 39.1 | 29.6 | 23 |
| VGG16 PGD | 97.8577 | 80.3 | 72.3 | 67 | 61.4 | 58.1 | 51.2 | 48 | 31.9 | 28.7 | 28.5 |

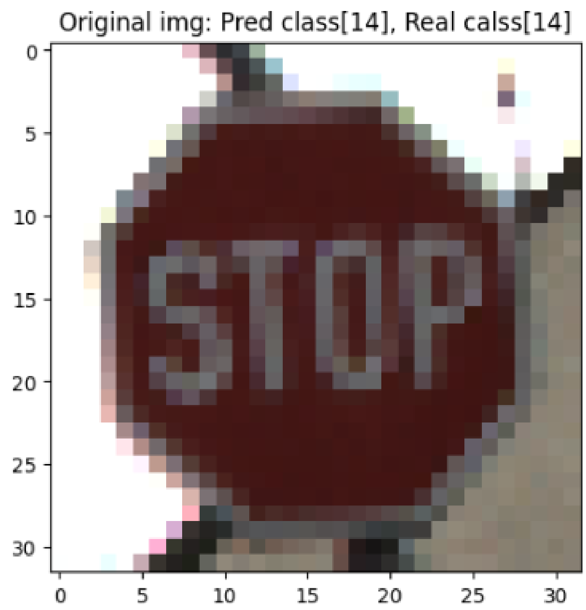
## Задание 3.

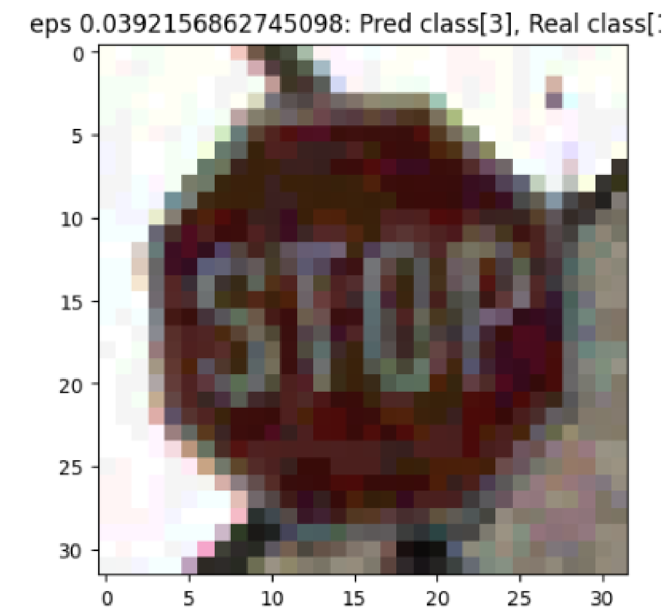
Создадим FGSM и PDG атаки.

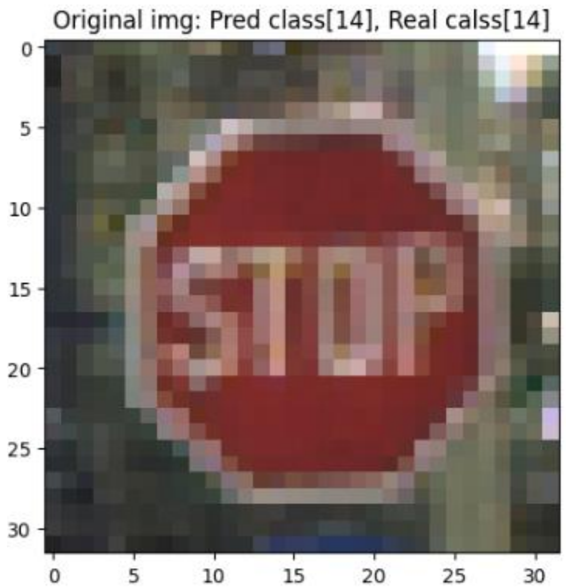
FGSM атака.

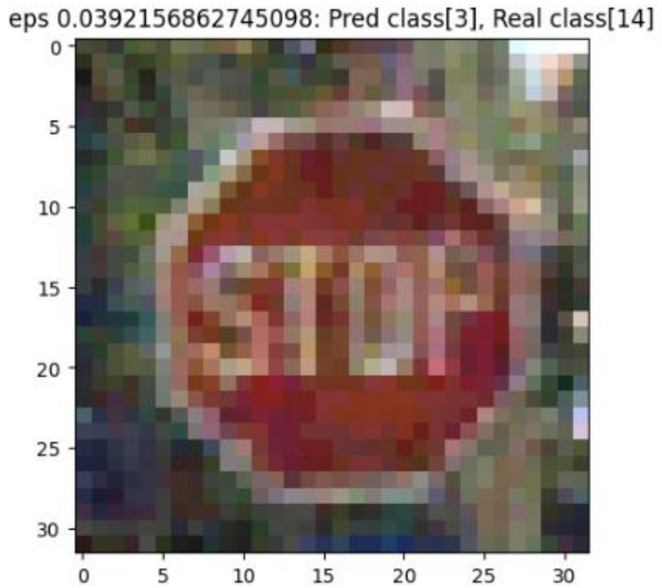
```
model=load_model('ResNet50.h5')
tf.compat.v1.disable_eager_execution()
t_class = 1
t_class = to_categorical(t_class, 43)
t_classes = np.tile(t_class, (270, 1))
x_test = data
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
attack_fgsm = FastGradientMethod(estimator=classifier,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

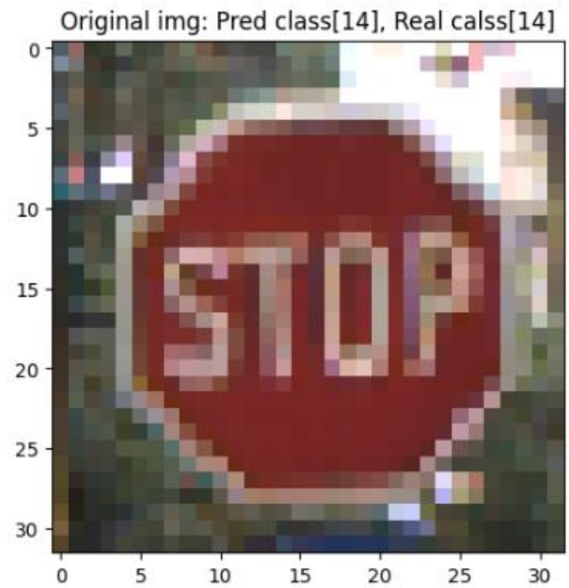
Отобразим 5 изображений для демонстрации атаки.

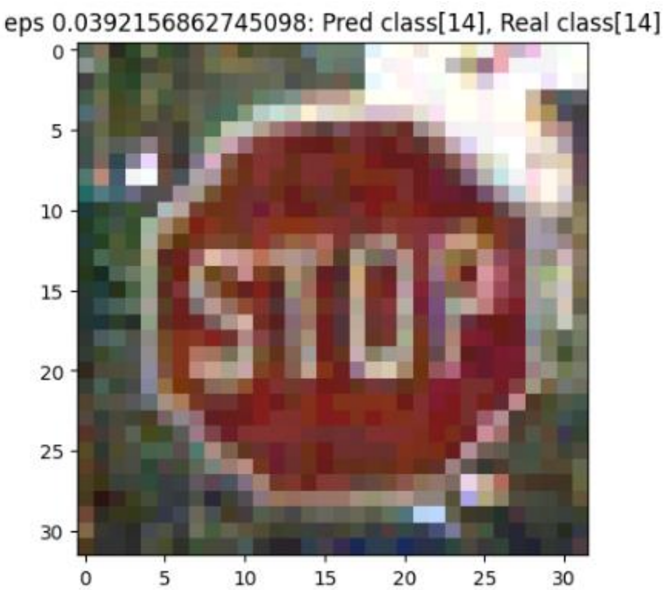


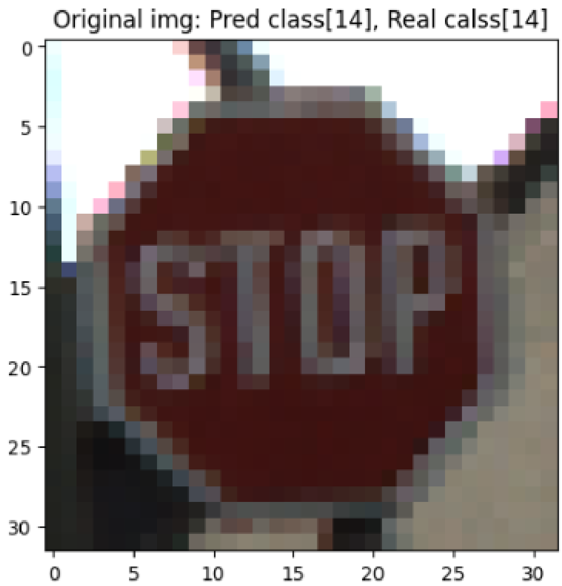



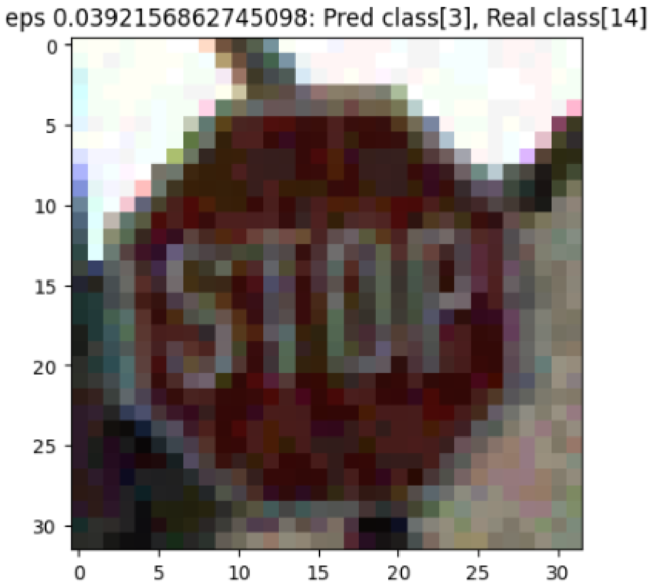



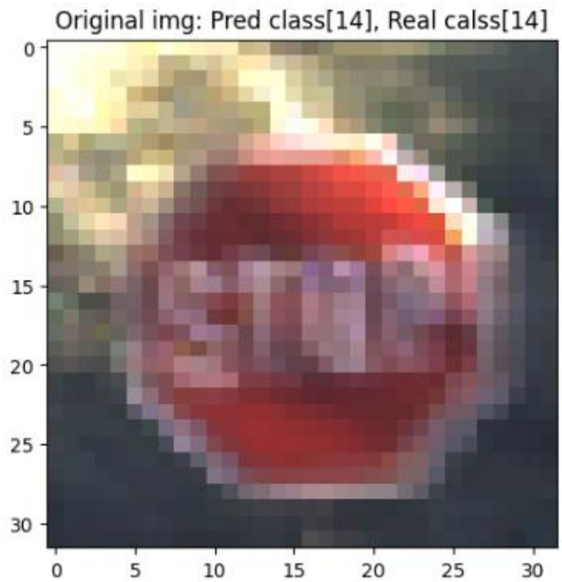



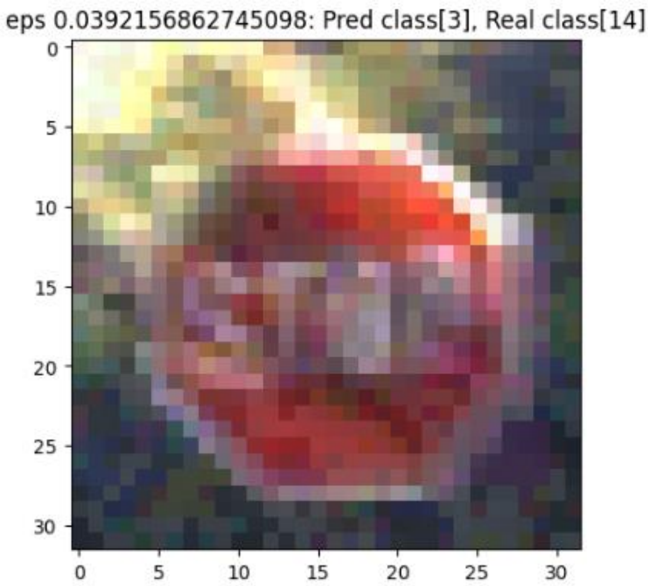









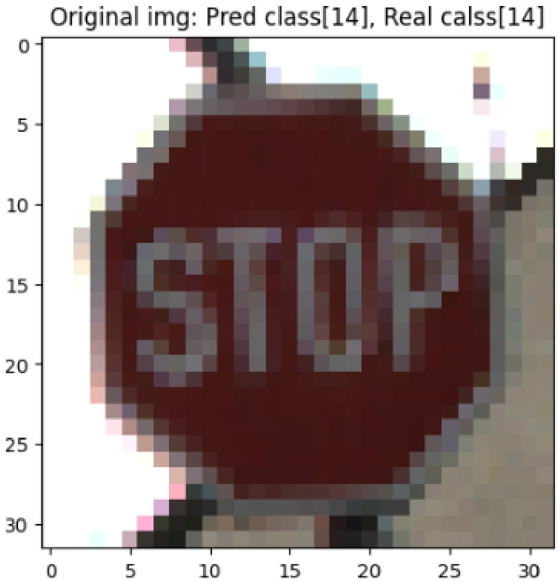


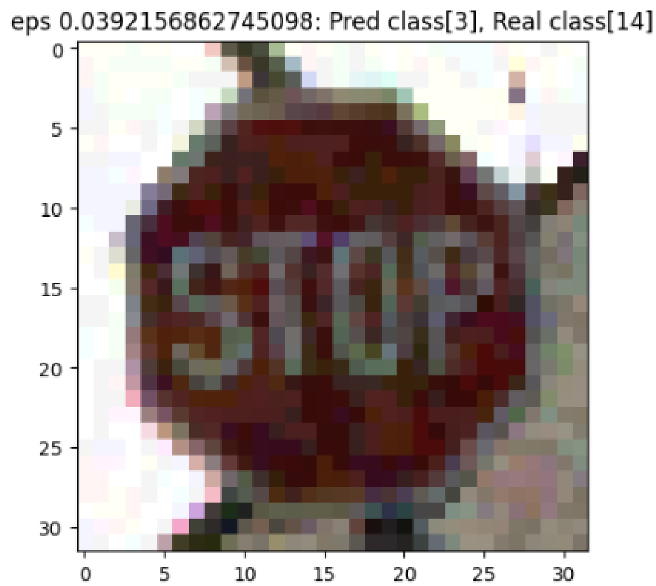

Атака PGD.

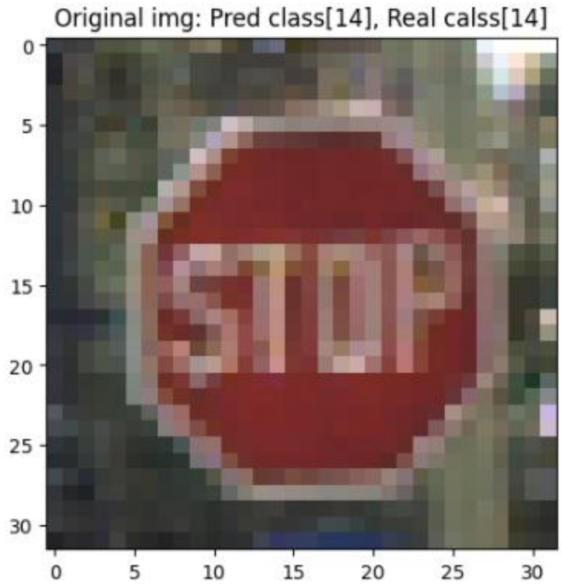
```
model=load_model('ResNet50.h5')
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3, max_iter=4, verbose=False, targeted=True)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255,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

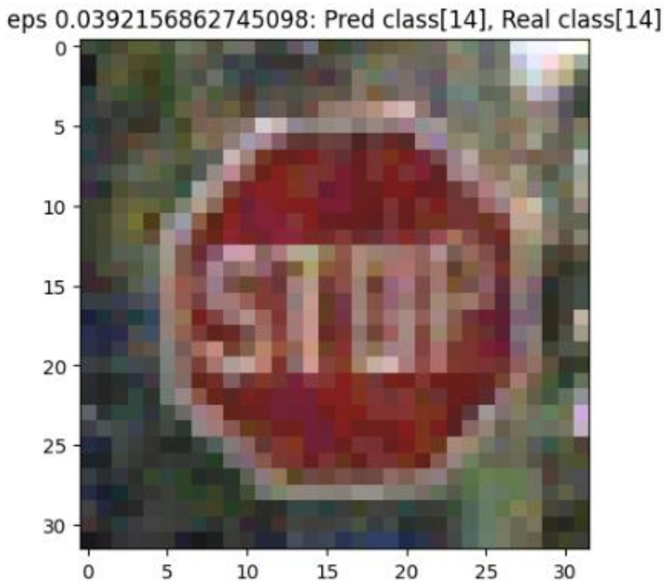
```
Eps: 0.00392156862745098
Adv Loss: 0.32350175380706786
```

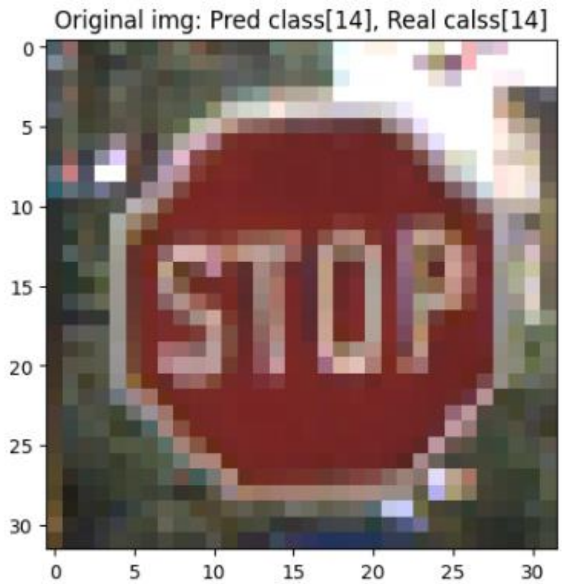
Отобразим 5 изображений для демонстрации атаки.

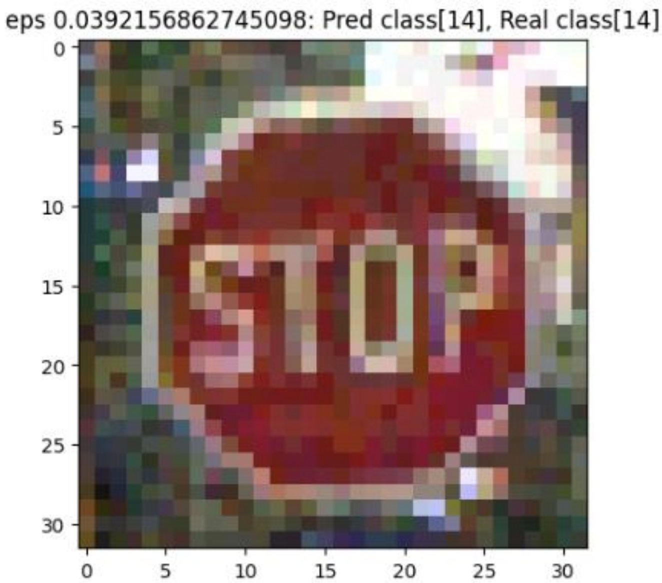


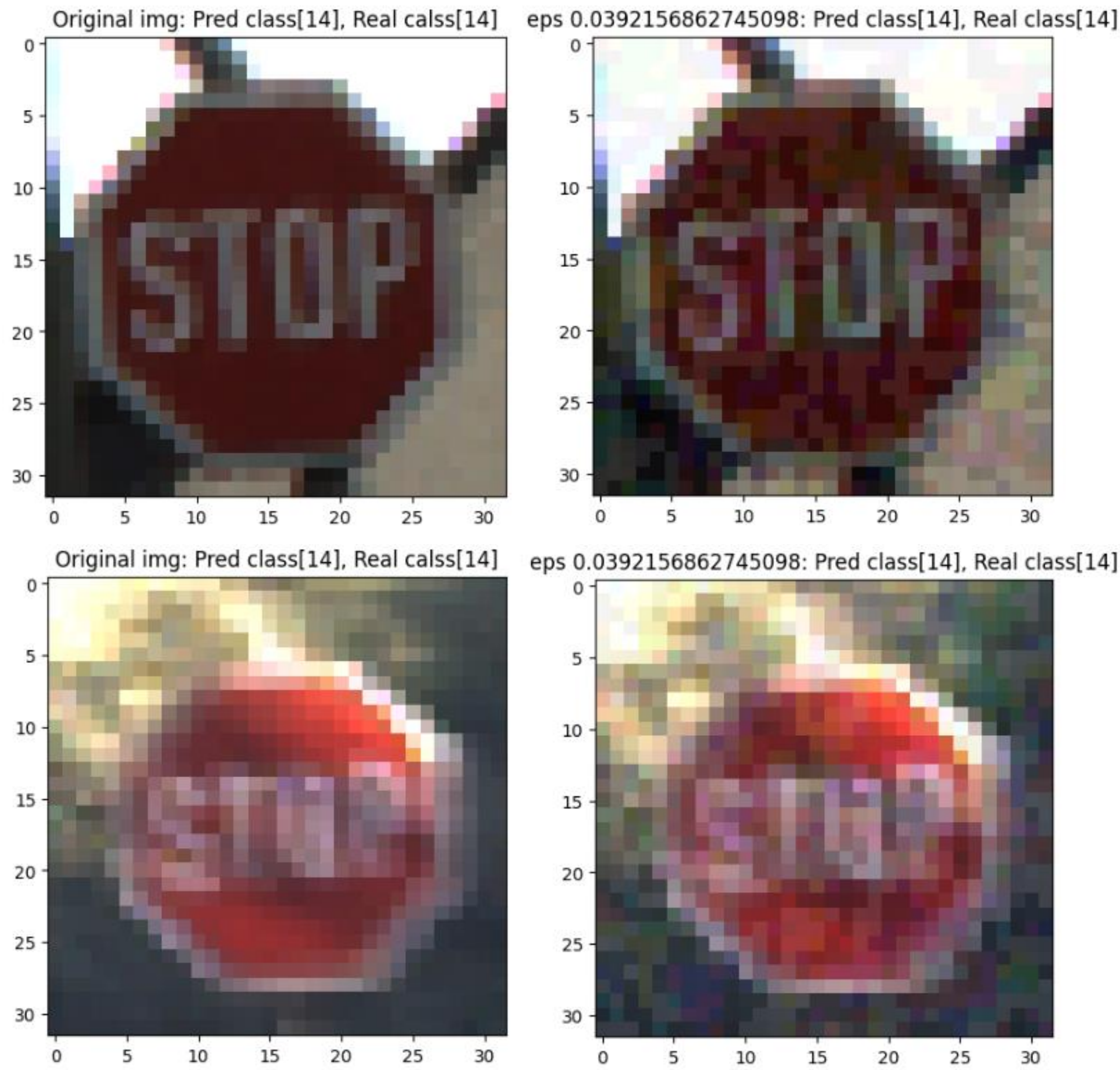






**Вывод.**

Атака PDG сохраняет точность, при этом лучше подходит для целевых атак, так как при больших значениях eps выдает лучший требуемый (класс 1 – знак стоп) результат, чем FGSM.