



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«МИРЭА – Российский технологический университет»  
РТУ МИРЭА**

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

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

## **Лабораторная работа №2**

по дисциплине: «Анализ защищенности систем искусственного  
интеллекта»

Группа:

ББМО-01-22

Выполнил:

Гребенник Г.С

Проверил:

к.т.н. Спирин А.А.

Москва, 2023

## Содержание

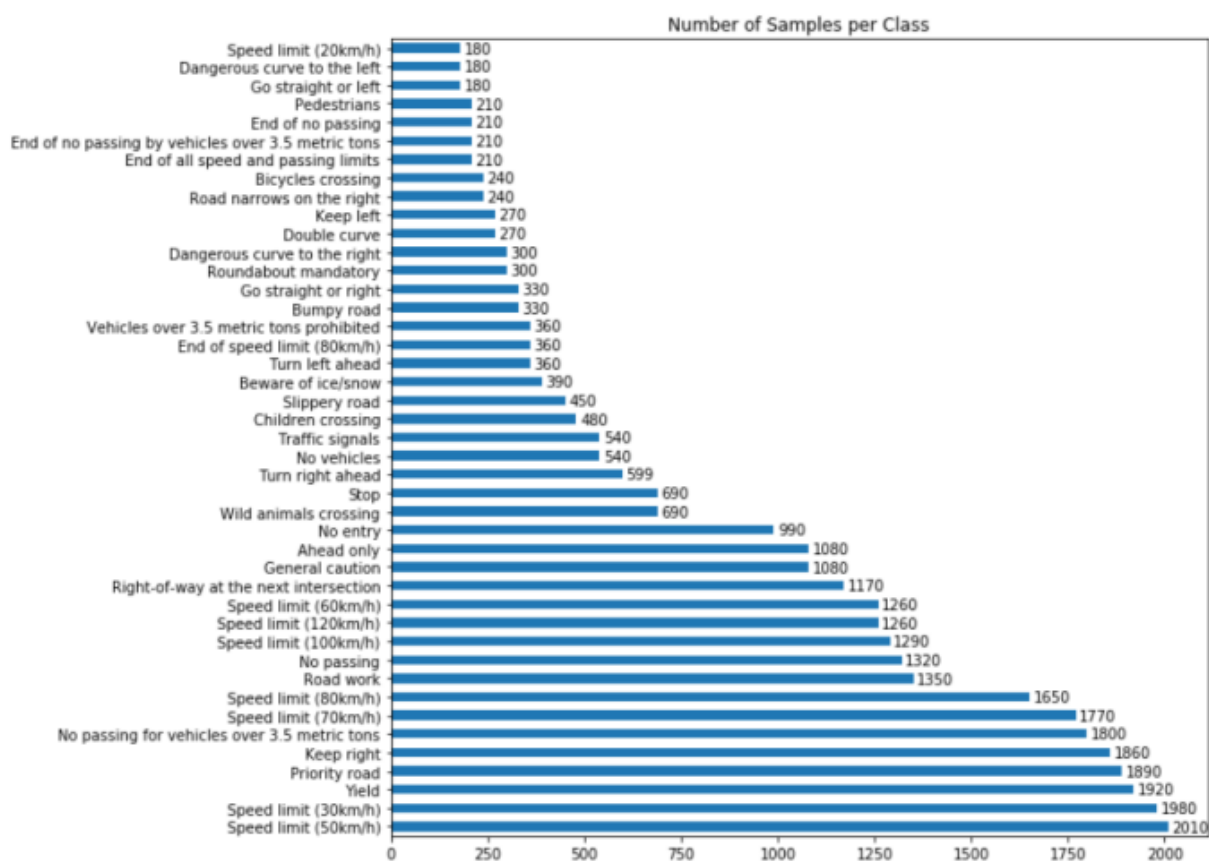
Цель работы: .....	3
Ход работы:.....	3
Выполнение задания 2 .....	9

## Цель работы:

- Реализовать атаки уклонения на основе белого ящика против классификационных моделей на основе глубокого обучения.
- Получить практические навыки переноса атак уклонения на основе черного ящика против моделей машинного обучения.

## Ход работы:

В работе используется 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. Распределение изображений по классам:



Задание 1. Обучить 2 классификатора на основе глубоких нейронных сетей на датасете GTSRB. Использовать следующие модели нейронных сетей: VGG16, ResNet50/10X, MobileNet v2/3. Можно использовать фреймворки Keras, TensorFlow, PyTorch, не надо создавать сети вручную и с нуля. Использовать предобученные сети (например на ImageNet). • Выполнить поиск наилучших гиперпараметров моделей. Использовать бесплатные ресурсы GPU сервиса Google Colab. Составить отчёт: (а) Заполнить (б) Для

каждой модели построить графики функции потерь для данных валидации и тестирования и графики метрики Accuracy(ghbvth yf hbc/ 2).

Таблица 1.

Модель	Обучение	Валидация	Тест
VGG16			
ResNet50			

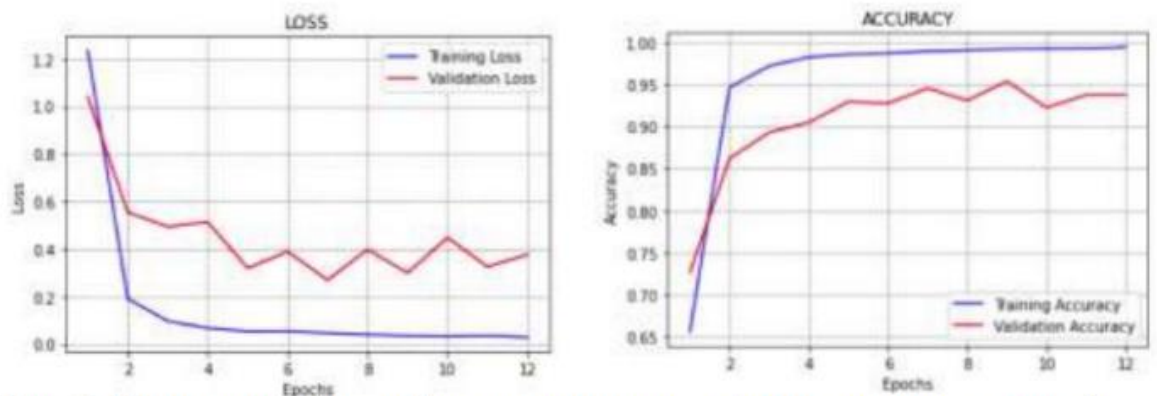


Рис. 2. Примеры графиков функции потерь и графиков точности моделей.

### Выполнение задания 1:

1. Установим adversarial-robustness-toolbox:

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

2. Импортируем необходимые библиотеки:

```
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
```

3. Добавляем датасет из гугл-диска:

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

✓  
49  
сек.

```
# разархивируем датасет
zip_file = '/content/drive/MyDrive/azii/archive.zip'
z = zipfile.ZipFile(zip_file, 'r')
z.extractall()

print(os.listdir())
```

4. Создаем модель ResNET50, для выборки данные разделили:

✓  
0  
сек.

```
[12] # выполним разделение данных на тренировочный 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)
# отображение размерности обучающего и тестового набора
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.]
```

[13] # 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/keras-applications/resnet/resnet50\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels\\_notop.h5](https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5)  
94765736/94765736 [=====] - 3s 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)
```


None

5. Создаем модель VGG16:

```

# 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/keras-applications/vgg16/vgg16\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels\\_notop.h5](https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5)  
58889256/58889256 [=====] - 2s 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

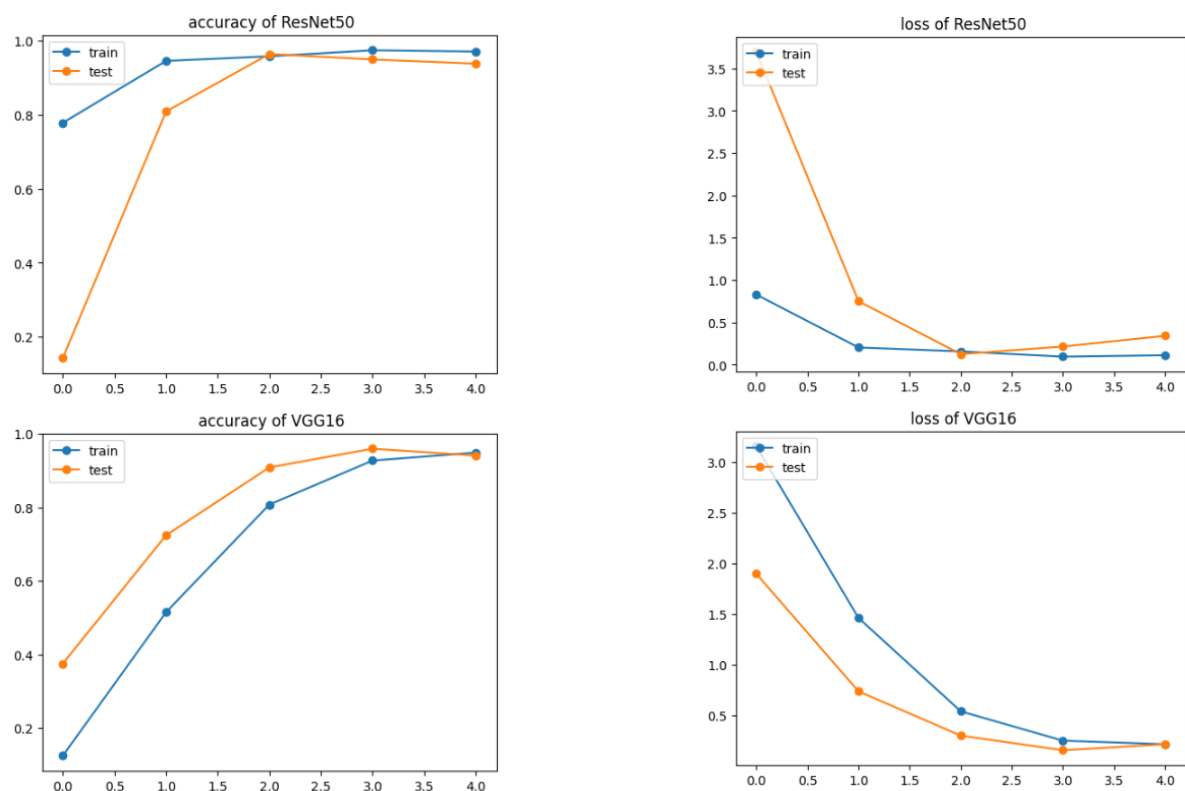
=====
  
Total params: 14857067 (56.68 MB)
  
Trainable params: 14725739 (56.17 MB)
  
Non-trainable params: 131328 (513.00 KB)
  
=====

None

## 6. Таблица точности, для тренировочного, валидационного и тестового набора данных:

MODEL	Обучение	Валидация	Тест
RESNET50	97.1143	93.8536	98.6228
VGG16	96.9396	98.5548	96.9396

## 7. Графики точности и потерь для наших моделей:



**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). Может быть использован код из следующих библиотек: Adversarial Robustness Toolbox ART, Cleverhans CH, scratchai SC. Наиболее проработанная библиотека – Adversarial Robustness Toolbox, рекомендуется использовать её, но другие также могут быть применены. Например, this notebook объясняет как использовать ART с помощью Keras. Также есть другие notebooks с примерами атак на основе библиотеки ART. Используйте атаки FGSM и PGD для создания нецелевых атакующих примеров используя первые 1,000 изображений из тестового множества. Необходимо использовать следующие значения параметра искажения:  $\epsilon = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]$ . Постройте графики точности 2-х моделей в зависимости от параметра искажений. Для атаки FGSM, отобразите исходное изображение из датасета и атакующее изображение с указанием величины параметра  $\epsilon = [1/255, 5/255, 10/255, 50/255, 80/255]$ , отобразите предсказанный класс атакующего изображения. Отчёт должен содержать: Все модели должны иметь точность менее 60% для  $\epsilon = 10/255$ . (b) Для каждой модели постройте график зависимости точности классификации от параметра искажений. Сделать выводы о полученных результатах.

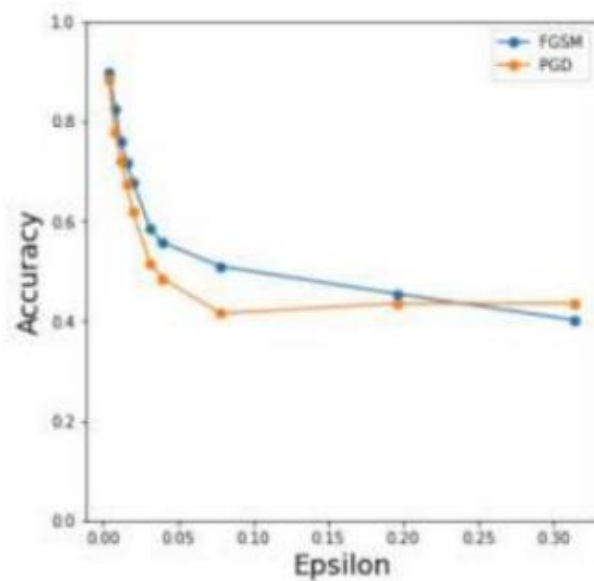


Рис. 3. Зависимость точности классификации от параметра искажений ЭПСИЛОН

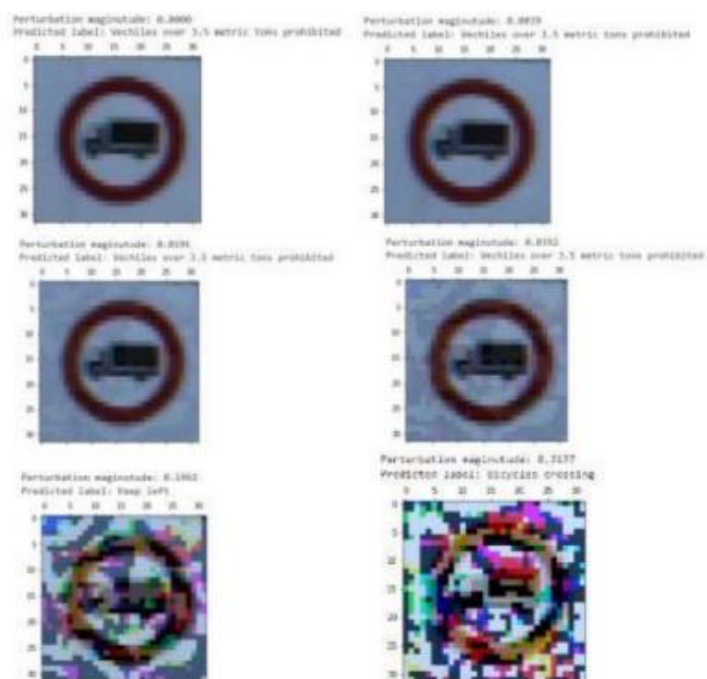


Рис. 4. Пример исходных и атакующих изображений

Таблица 2.

Модель	Исходные изображения	Adversarial images $\epsilon=1/255$	Adversarial images $\epsilon=5/255$	Adversarial images $\epsilon=10/255$
VGG16 - FGSM				
VGG16 - PGD				
ResNet50 - FGSM				
ResNet50 - PGD				



## Выполнение задания 2

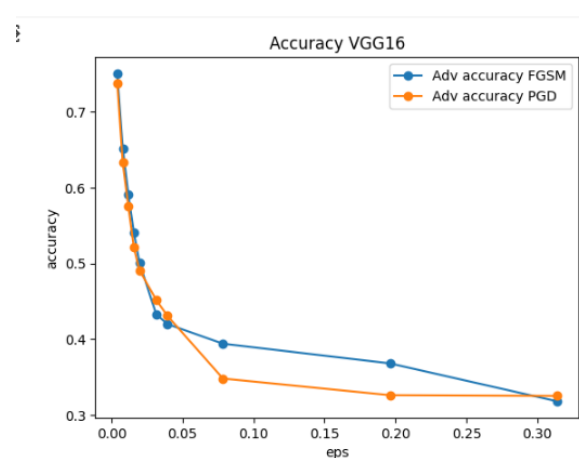
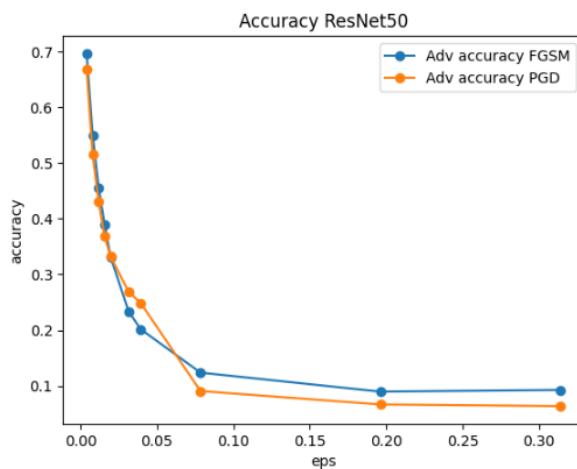
1. Проводим атаку FGSM с параметром искажения  $[1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]$

```
✓ 1 мин. # 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 = []
```

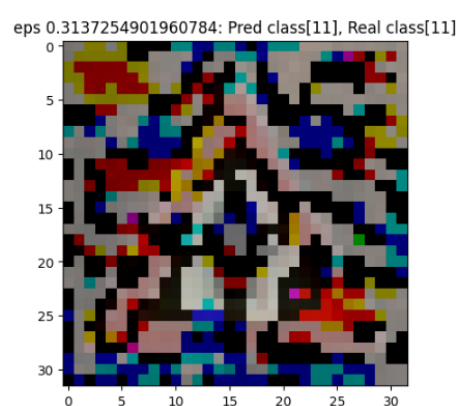
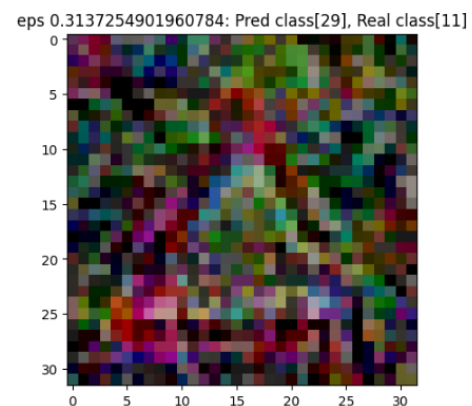
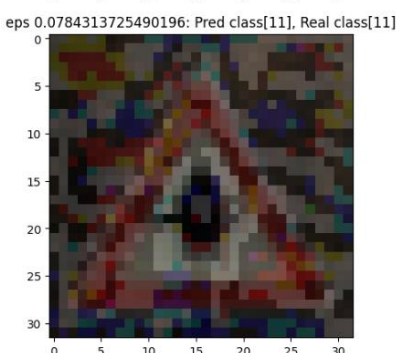
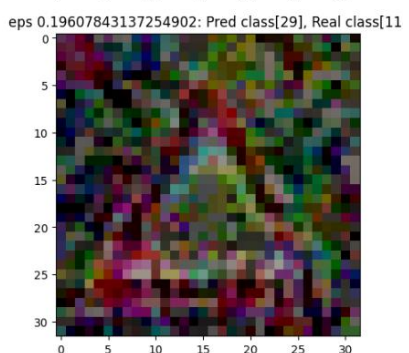
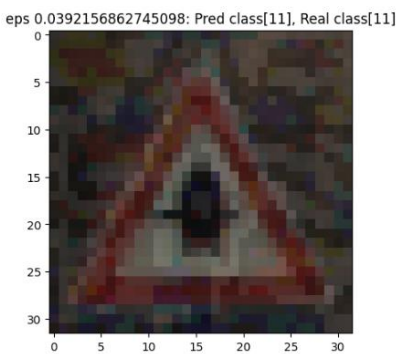
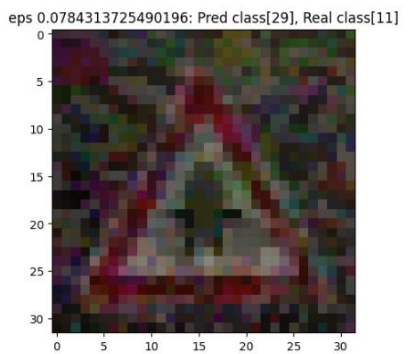
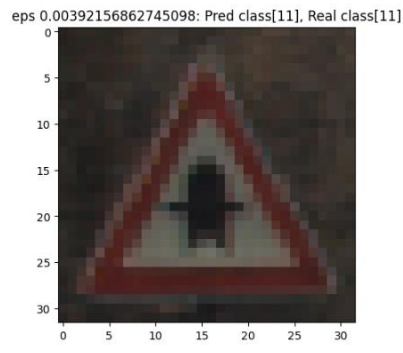
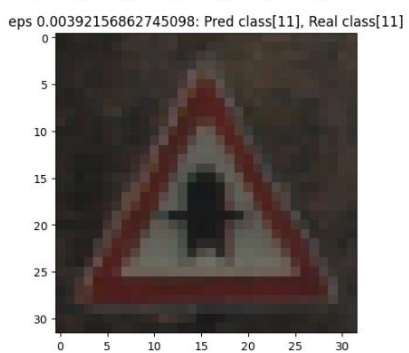
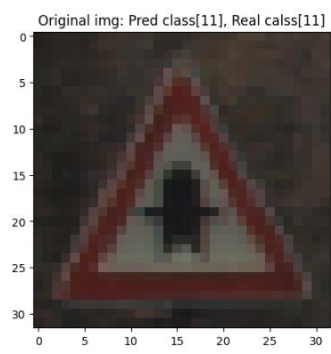
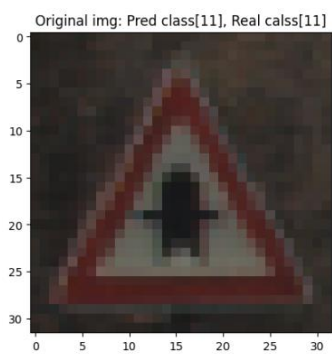
2. Проводим атаку PGD с параметром искажения  $[1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]$

```
✓ [26] # 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 = []
```

3. График зависимости точности классификации от параметра искажения для FGSM и PGD на датасеты ResNet50 и VGG16:



4. Отобразим исходное значение из датасета и атакующее:



## 5. Для атаки PGD



## 6. Таблица значений точности для обеих моделей:

Model	Original accuracy	eps = 1/255	eps = 5/255	eps = 10/255
Resnet50 FGSM	97.1143	69.6	33.1	20.2
Resnet50 PGD	97.1143	66.8	33.2	24.9
VGG16 FGSM	96.9396	75.1	50.1	42
VGG16 PGD	96.9396	73.8	49.1	43.1

**Задание 3** Применение целевой атаки уклонения методом белого против моделей глубокого обучения. Шаг 1: Используйте изображения знака «Стоп» (label class 14) из тестового набора данных. Всего имеется 270 изображений. Примените атаку Projected Gradient Descent (PGD) на знак «Стоп» с целью классификации его как знака «Ограничение скорости 30» (target label class = 1). Изменяйте значения искажений  $\epsilon \in [1/255, 3/255, 5/255, 10/255, 20/255, 50/255, 80/255]$ , и заполните отчёт значениями точности классификации изображений знаков "Стоп" и "Ограничение скорости 30". Шаг 2: Повторите атаку методом FGSM, и объясните производительность по сравнению с PGD. Отчёт должен содержать: (а) Заполненную таблицу 3. Объясните какой размер искажений достигает максимальной производительности и объясните причины. (б) Постройте 5 примеров исходных изображений знака «Стоп» и

соответствующих атакующих примеров (см. рис. 5). (с) Сравните результаты атак PGD и FGSM между собой.

Таблица 5.

Искажение	PGD attack – Stop sign images	PGD attack – Speed Limit 30 sign images
$\epsilon=1/255$		
$\epsilon=3/255$		
$\epsilon=5/255$		
$\epsilon=10/255$		
$\epsilon=20/255$		
$\epsilon=50/255$		
$\epsilon=80/255$		

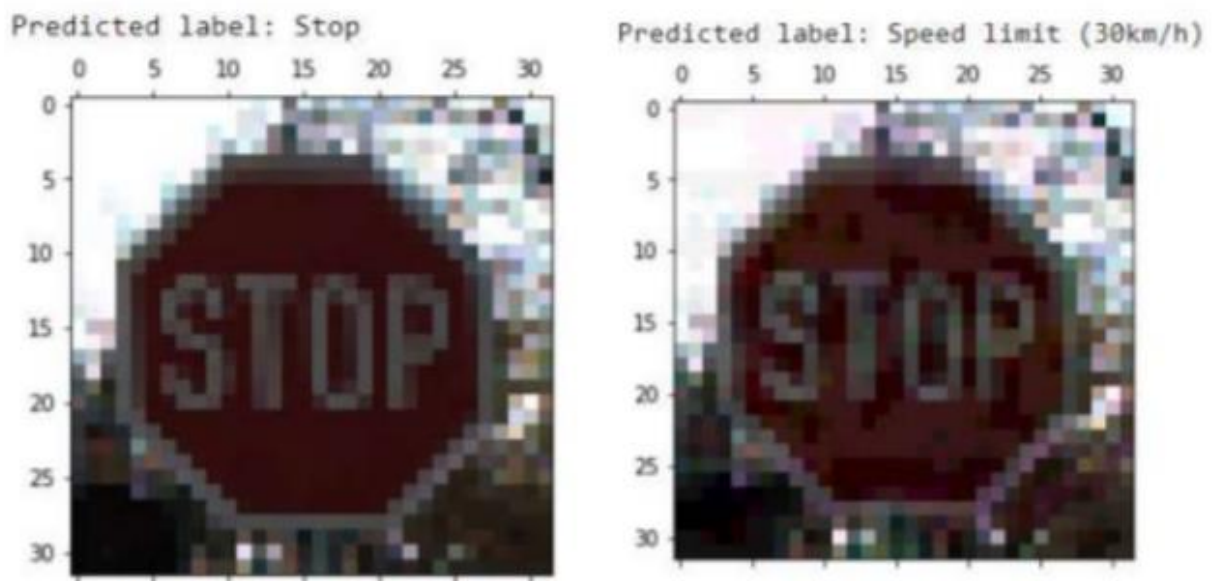


Рис. 5. Пример исходных и атакующих изображений

Решение задания 3 :

1. Создадим 2 целевые атаки

```
# создадим две целевые атаки
# загрузим тестовый набор данных из Test.csv и извлечем изображения с меткой 14
# Преобразуем изображения в массив чисел и нормализуем
test = pd.read_csv("Test.csv")
test_imgs = test['Path'].values
data = []
y_test = []
labels = test['ClassId'].values.tolist()
i = -1

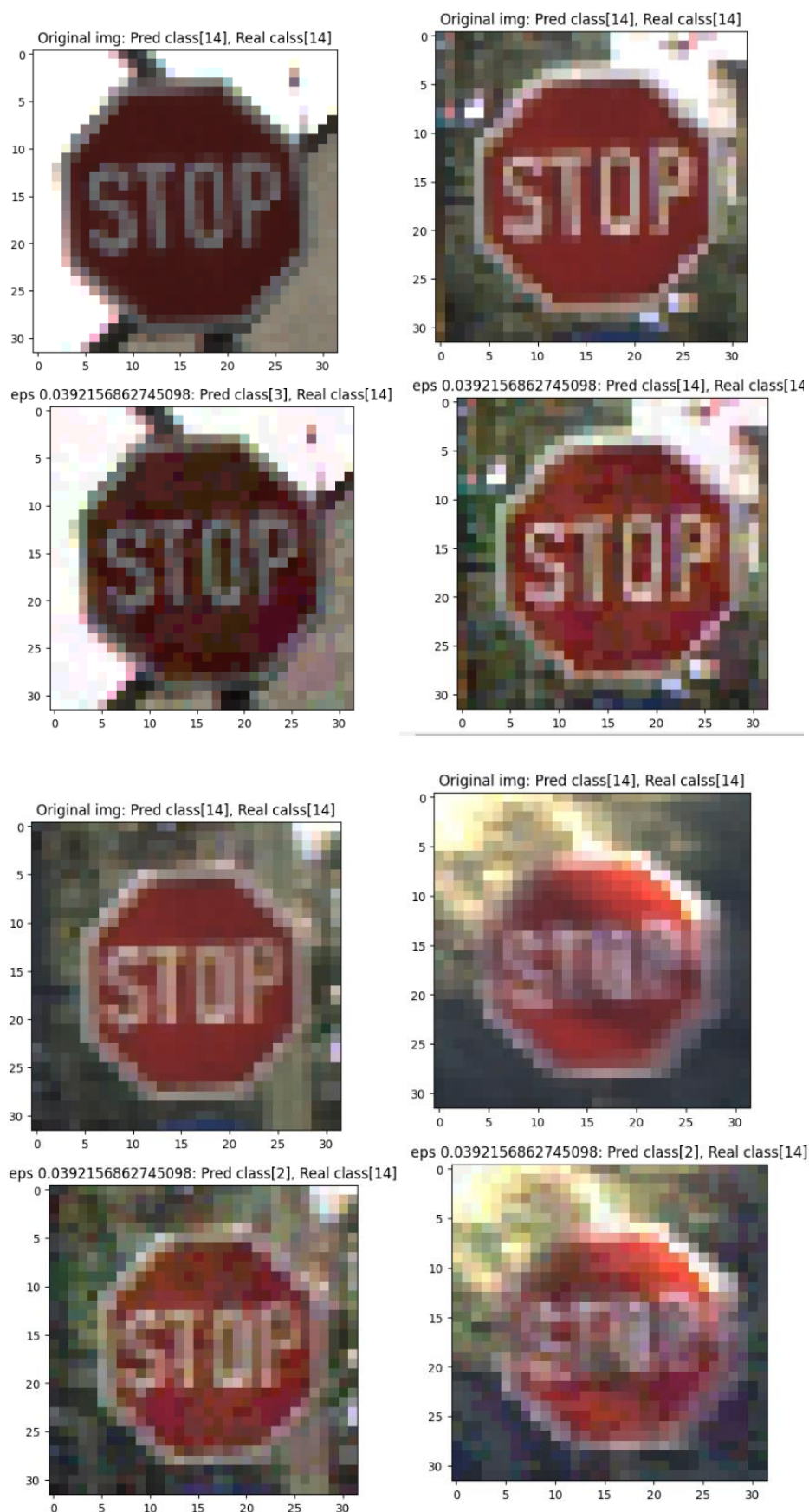
for img in test_imgs:
    i += 1
    if labels[i] != 14:
        continue
    img = image.load_img(img, target_size=(32, 32))
    img_array = image.img_to_array(img)
    img_array = img_array / 255
    data.append(img_array)
    y_test.append(labels[i])
data = np.array(data)
y_test = np.array(y_test)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

[81] # реализуем целевую атаку FGSM
model = load_model('ResNet50.h5')
tf.compat.v1.disable_eager_execution()
t_class = 1
t_classes = to_categorical(t_class, 43)
t_classes = np.tile(t_classes, (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, 6/255, 10/255, 20/255, 50/255, 80/255]

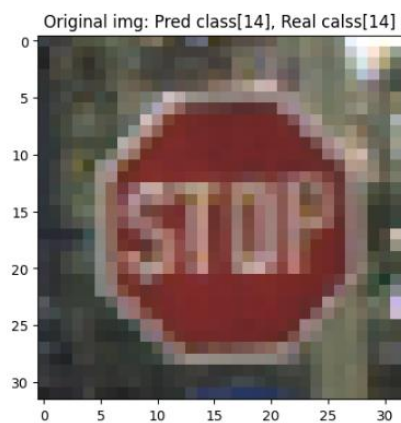
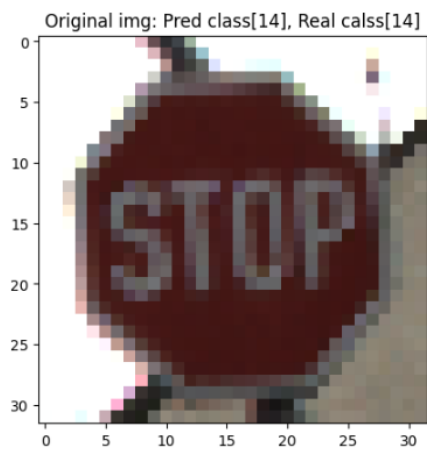
for eps in eps_range:
    attack_fgsm.set_params({"eps": eps})
    print(f"eps: {eps}")
    x_test_adv = attack_fgsm.generate(x_test, t_classes)
    loss, accuracy = model.evaluate(x_test_adv, y_test)
    print(f"Adv Loss: {loss}")
    print(f"Adv Accuracy: {accuracy}")
    loss, accuracy = model.evaluate(x_test,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



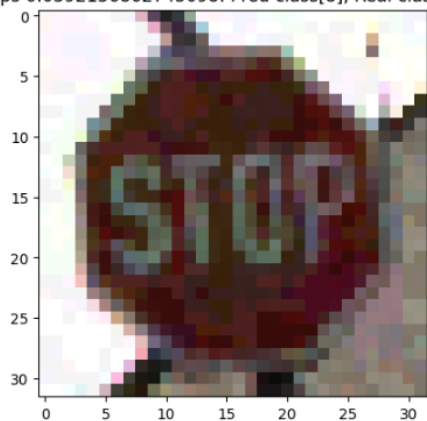
## 2. Атака FGSM исходных данных знака СТОП



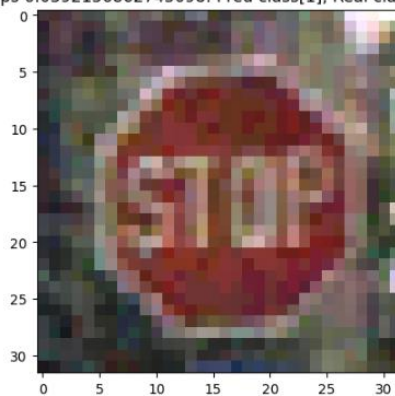
## 3. Атака PGD исходных данных знака СТОП:



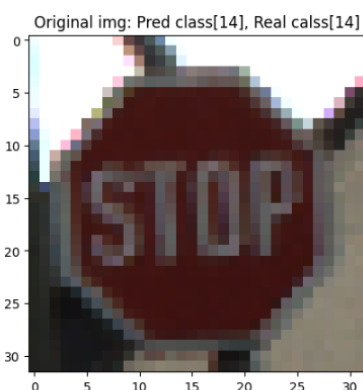
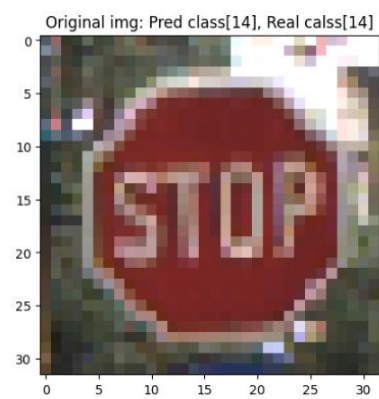
eps 0.0392156862745098: Pred class[8], Real class[14]



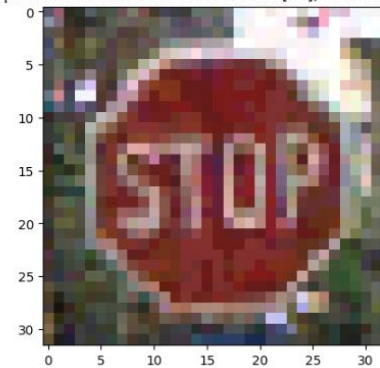
eps 0.0392156862745098: Pred class[1], Real class[14]



Original img: Pred class[14], Real calss[14]



eps 0.0392156862745098: Pred class[14], Real class[14]



eps 0.0392156862745098: Pred class[14], Real class[14]

