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 защищённости систем искусственного интеллекта»

Группа:
ББМО-01-22
Выполнила:
Огольцова Н.Д.

Проверил:
Спирин А.А.

Москва 2023

Ход работы: (Раздел 1)

1)Установим необходимые библиотеки и фреймворки;

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

```
Requirement already satisfied: adversarial-robustness-toolbox in /usr/local/lib/python3.10/dist-packages (1.16.0)
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) (1.11.3)
Requirement already satisfied: scikit-learn<1.2.0,>=0.22.2 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.1.3)
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)
```

```python
import cv2
import os
import torch
import random
import pickle
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

import keras
import tensorflow as tf
from keras.applications.vgg16 import VGG16
from keras.applications.resnet50 import ResNet50
from keras.applications.mobilenet import MobileNet
from keras.models import Model
from keras.preprocessing import image
from tensorflow.keras.layers import Input, Lambda ,Dense ,Flatten ,Dropout
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import os
import cv2
import random
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, array_to_img, load_img

from tensorflow.keras.utils import to_categorical
from tensorflow import keras as k

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms

import zipfile
from google.colab import drive
from PIL import Image
from skimage import io
import pandas as pd
import matplotlib.image as img
from PIL import Image
from sklearn import svm
from sklearn.model_selection import train_test_split
```

2)Подключим Google Drive для работы;

```
!ls
drive.mount('/content/drive/')
```

```
drive  Meta      ResNet50.h5  test  Test.csv  Train
meta   Meta.csv  sample_data  Test  train     Train.csv
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).
```

3)Разархивируем файл с датасетом;

```
zip_file = '/content/drive/MyDrive/АЗСИИ/archive.zip'
z = zipfile.ZipFile(zip_file, 'r')
z.extractall()

print(os.listdir())
```

```
['.config', 'test', 'meta', 'Test.csv', 'Train.csv', 'Train', 'Meta.csv', 'drive', 'Meta', 'Test', 'train', 'ResNet50.h5', 'sample_data']
```

4)Зададим переменные для удобной навигации по датасету;

```
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')
```

5)С помощью цикла предобразуем и сохраним изображения в виде массива в переменную;

```
data = []
labels = []
CLASSES = 43
# Используем цикл for для доступа к каждому изображению
for i in range(CLASSES):
    img_path = os.path.join(train_data_path, str(i)) #0-42
    for img in os.listdir(img_path):
        im = Image.open(train_data_path + '/' + str(i) + '/' + img)
        im = im.resize((32,32))
        im = np.array(im)
        data.append(im)
        labels.append(i)
data = np.array(data)
labels = np.array(labels)
print("data[0]: ",data[0])
print("labels[0: ]",labels[0])
```

```
data[0]:  [[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
```

6)Разделим датасет на обучающую и тестовую выборки;

```
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
print("training shape: ",x_train.shape, y_train.shape)
print("testing shape: ",x_test.shape, y_test.shape)
# convert interge label to one-hot data
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)

print(y_train[1])
```

```
training shape:  (31367, 32, 32, 3) (31367,)
testing shape:  (7842, 32, 32, 3) (7842,)
[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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

7)Создадим модель MobileNet;

```
mobilenet = k.applications.mobilenet_v2.MobileNetV2(weights='imagenet', include_top=False)

model = k.models.Sequential([
                            mobilenet,
                            tf.keras.layers.GlobalAveragePooling2D(),
                            k.layers.Dropout(0.2),
                            k.layers.Dense(256, activation='relu'),
                            k.layers.BatchNormalization(),
                            k.layers.Dropout(0.1),
                            k.layers.Dense(512, activation='relu'),
                            k.layers.BatchNormalization(),
                            k.layers.Dropout(0.2),
                            k.layers.Dense(43, activation='softmax')
])
print(model.summary())
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is
WARNING:tensorflow:From /usr/local/lib/python3.10/dist-packages/keras/src,
Instructions for updating:
Colocations handled automatically by placer.
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Func  (None, None, None, 1280   2257984
 tional)                     )

 global_average_pooling2d_3  (None, 1280)              0
  (GlobalAveragePooling2D)

 dropout_9 (Dropout)         (None, 1280)              0

 dense_9 (Dense)             (None, 256)               327936

 batch_normalization_6 (Bat  (None, 256)               1024
 chNormalization)

 dropout_10 (Dropout)        (None, 256)               0

 dense_10 (Dense)            (None, 512)               131584

 batch_normalization_7 (Bat  (None, 512)               2048
 chNormalization)

 dropout_11 (Dropout)        (None, 512)               0

 dense_11 (Dense)            (None, 43)                22059

=================================================================
Total params: 2742635 (10.46 MB)
Trainable params: 2706987 (10.33 MB)
Non-trainable params: 35648 (139.25 KB)
_________________________________________________________________
None
```

Скомпилируем её:

```
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))
```

```
Train on 31367 samples, validate on 7842 samples
Epoch 1/5
31367/31367 [==============================] - ETA: 0s - loss: 1.6415 - accuracy: 0.5378/usr/local/lib/python3.10/dist-packages/keras/src/er
  updates = self.state_updates
31367/31367 [==============================] - 75s 2ms/sample - loss: 1.6415 - accuracy: 0.5378 - val_loss: 4.7476 - val_accuracy: 0.1750
Epoch 2/5
31367/31367 [==============================] - 62s 2ms/sample - loss: 0.4007 - accuracy: 0.8812 - val_loss: 3.7772 - val_accuracy: 0.2548
Epoch 3/5
31367/31367 [==============================] - 71s 2ms/sample - loss: 0.2662 - accuracy: 0.9220 - val_loss: 3.2438 - val_accuracy: 0.3284
Epoch 4/5
31367/31367 [==============================] - 63s 2ms/sample - loss: 0.1945 - accuracy: 0.9468 - val_loss: 1.1653 - val_accuracy: 0.7308
Epoch 5/5
31367/31367 [==============================] - 65s 2ms/sample - loss: 0.1489 - accuracy: 0.9594 - val_loss: 1.0241 - val_accuracy: 0.7904
```

8)Создадим модель ResNet50;

```
resnet = k.applications.ResNet50(weights='imagenet', include_top=False)

model_2 = k.models.Sequential([
                            resnet,
                            tf.keras.layers.GlobalAveragePooling2D(),
                            k.layers.Dropout(0.2),
                            k.layers.Dense(256, activation='relu'),
                            k.layers.BatchNormalization(),
                            k.layers.Dropout(0.1),
                            k.layers.Dense(512, activation='relu'),
                            k.layers.BatchNormalization(),
                            k.layers.Dropout(0.2),
                            k.layers.Dense(43, activation='softmax')
])
print(model_2.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 resnet50 (Functional)       (None, None, None, 2048   23587712  
                             )                                   
                                                                 
 global_average_pooling2d_4  (None, 2048)              0         
  (GlobalAveragePooling2D)                                       
                                                                 
 dropout_12 (Dropout)        (None, 2048)              0         
                                                                 
 dense_12 (Dense)            (None, 256)               524544    
                                                                 
 batch_normalization_8 (Bat  (None, 256)               1024      
 chNormalization)                                                
                                                                 
 dropout_13 (Dropout)        (None, 256)               0         
                                                                 
 dense_13 (Dense)            (None, 512)               131584    
                                                                 
 batch_normalization_9 (Bat  (None, 512)               2048      
 chNormalization)                                                
                                                                 
 dropout_14 (Dropout)        (None, 512)               0         
                                                                 
 dense_14 (Dense)            (None, 43)                22059     
                                                                 
=================================================================
Total params: 24268971 (92.58 MB)
Trainable params: 24214315 (92.37 MB)
Non-trainable params: 54656 (213.50 KB)
_________________________________________________________________
None
```

Скомпилируем её:

```
model_2.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
history_2 = model_2.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))
```

```
Train on 31367 samples, validate on 7842 samples
Epoch 1/5
31367/31367 [==============================] - 97s 3ms/sample - loss: 0.8805 - accuracy: 0.7560 - val_loss: 0.2633 - val_accuracy: 0.9306
Epoch 2/5
31367/31367 [==============================] - 67s 2ms/sample - loss: 0.2269 - accuracy: 0.9426 - val_loss: 0.1763 - val_accuracy: 0.9461
Epoch 3/5
31367/31367 [==============================] - 70s 2ms/sample - loss: 0.2242 - accuracy: 0.9479 - val_loss: 0.4390 - val_accuracy: 0.9082
Epoch 4/5
31367/31367 [==============================] - 71s 2ms/sample - loss: 0.1403 - accuracy: 0.9666 - val_loss: 0.3176 - val_accuracy: 0.9325
Epoch 5/5
31367/31367 [==============================] - 68s 2ms/sample - loss: 0.1094 - accuracy: 0.9759 - val_loss: 0.0957 - val_accuracy: 0.9742
```

9)Создадим модель VGG16;

```
vgg16 = k.applications.VGG16(weights='imagenet', include_top=False)


model_3 = k.models.Sequential([
                             vgg16,
                             tf.keras.layers.GlobalAveragePooling2D(),
                             k.layers.Dropout(0.2),
                             k.layers.Dense(256, activation='relu'),
                             k.layers.BatchNormalization(),
                             k.layers.Dropout(0.1),
                             k.layers.Dense(512, activation='relu'),
                             k.layers.BatchNormalization(),
                             k.layers.Dropout(0.2),
                             k.layers.Dense(43, activation='softmax')
])
print(model_3.summary())
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, None, None, 512)   14714688

 global_average_pooling2d_5  (None, 512)               0
  (GlobalAveragePooling2D)

 dropout_15 (Dropout)        (None, 512)               0

 dense_15 (Dense)            (None, 256)               131328

 batch_normalization_10 (Ba  (None, 256)               1024
 tchNormalization)

 dropout_16 (Dropout)        (None, 256)               0

 dense_16 (Dense)            (None, 512)               131584

 batch_normalization_11 (Ba  (None, 512)               2048
 tchNormalization)

 dropout_17 (Dropout)        (None, 512)               0

 dense_17 (Dense)            (None, 43)                22059

=================================================================
Total params: 15002731 (57.23 MB)
Trainable params: 15001195 (57.23 MB)
Non-trainable params: 1536 (6.00 KB)
_________________________________________________________________
None
```

Скомпилируем её:

```
model_3.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
history_3 = model_3.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))
```

```
Train on 31367 samples, validate on 7842 samples
Epoch 1/5
31367/31367 [==============================] - 25s 799us/sample - loss: 3.2333 - accuracy: 0.1182 - val_loss: 2.5970 - val_accuracy: 0.1854
Epoch 2/5
31367/31367 [==============================] - 20s 652us/sample - loss: 2.5899 - accuracy: 0.1820 - val_loss: 2.6285 - val_accuracy: 0.1862
Epoch 3/5
31367/31367 [==============================] - 20s 645us/sample - loss: 2.1403 - accuracy: 0.2764 - val_loss: 2.7435 - val_accuracy: 0.1747
Epoch 4/5
31367/31367 [==============================] - 20s 646us/sample - loss: 1.4171 - accuracy: 0.4944 - val_loss: 1.0392 - val_accuracy: 0.6551
Epoch 5/5
31367/31367 [==============================] - 22s 693us/sample - loss: 0.6955 - accuracy: 0.7597 - val_loss: 0.4497 - val_accuracy: 0.8564
```

10)Полученный результат, представленный в виде таблицы по разделу 1:

Model	Training Accuracy	Validation Accuracy	Test Accuracy
MobileNet	95.9416	79.036	97.5899
Resnet50	97.593	97.4241	97.1946
VGG16	75.9652	85.6414	88.7911

Задание 2:

1)Для второго задания используем тысячу первых тестовых изображений, также зададим значения epsilon;

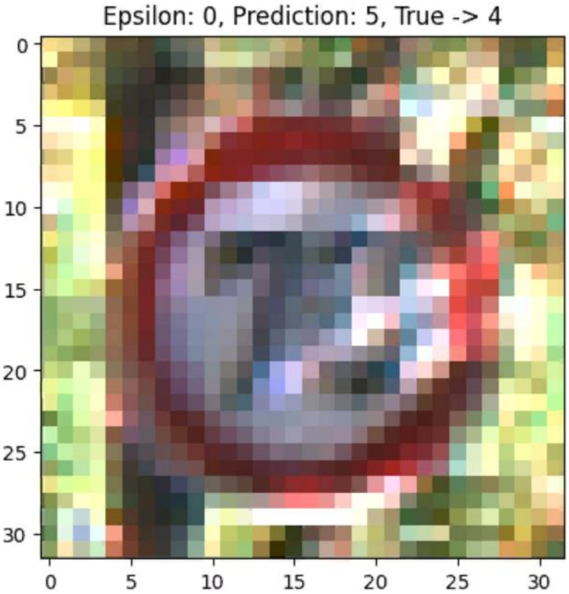
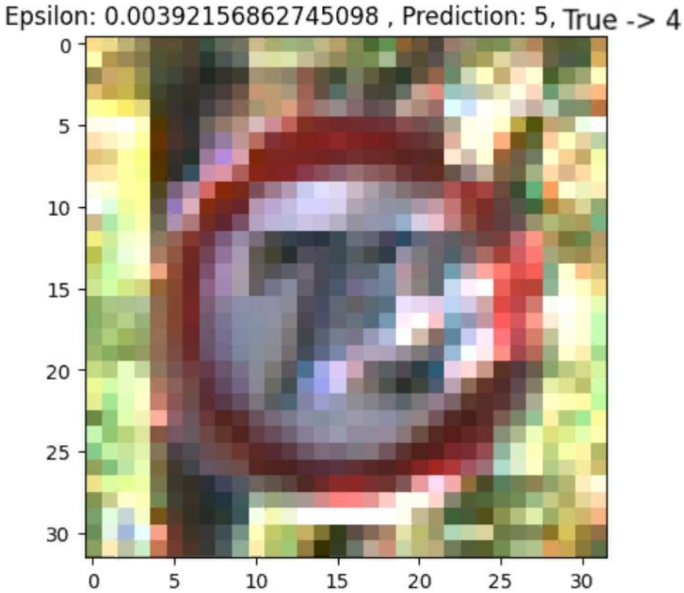
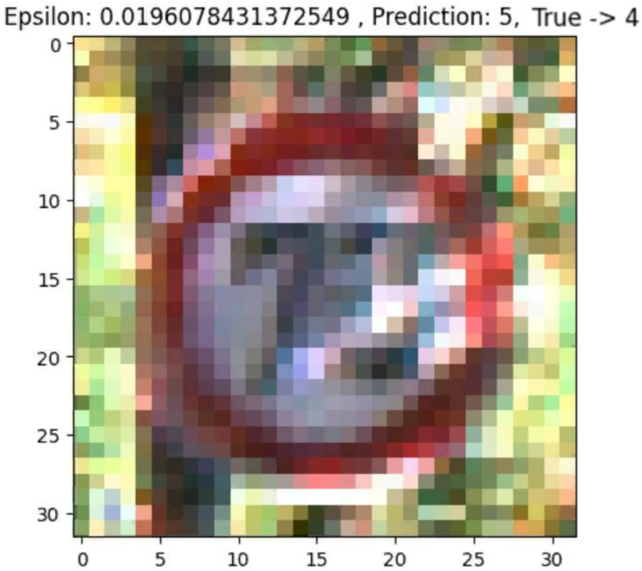
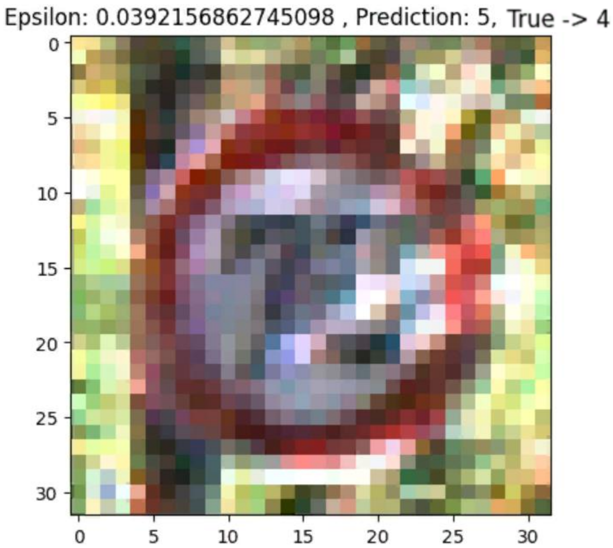
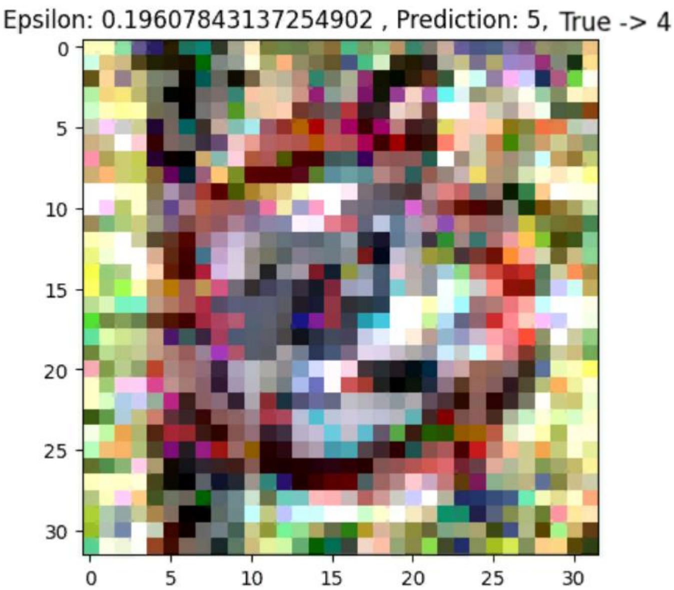
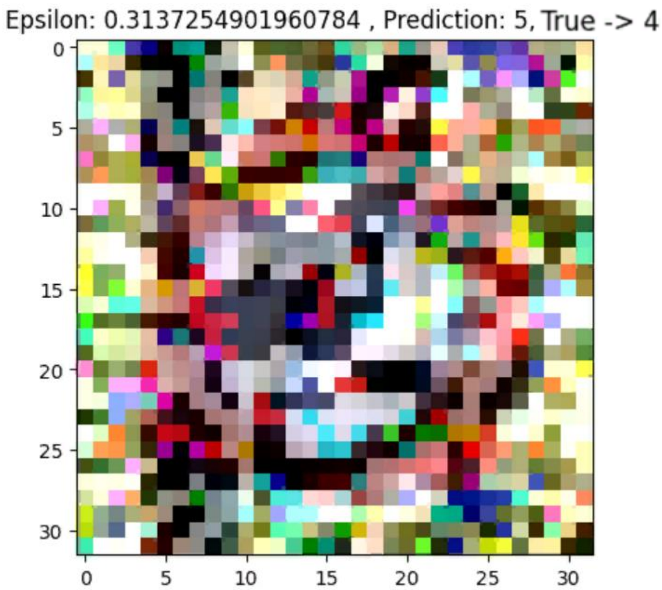
```
[13] epsilons = [0.004, 0.008, 0.012, 0.016, 0.02, 0.031, 0.039, 0.078, 0.196, 0.314]
```

0
ек.

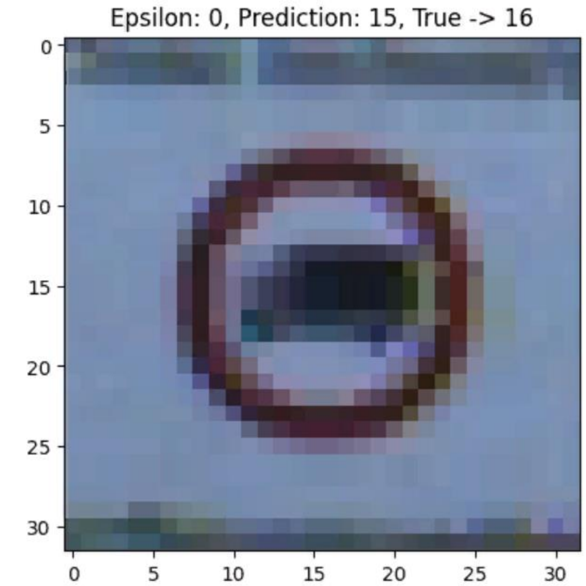
```
x_test = data[:1000]
y_test = y_test[:1000]
```

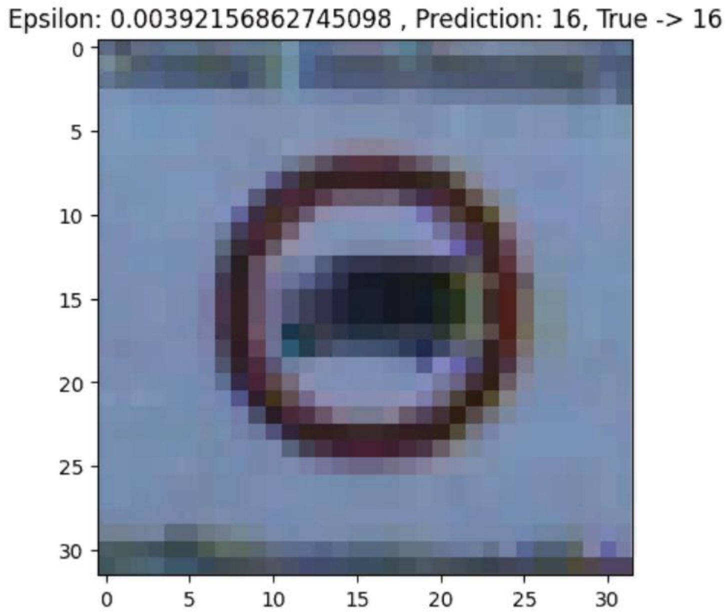
2)Переопределим значения epsilons выведем изображение под номером 14 после атаки FGSM в сравнении с исходным;

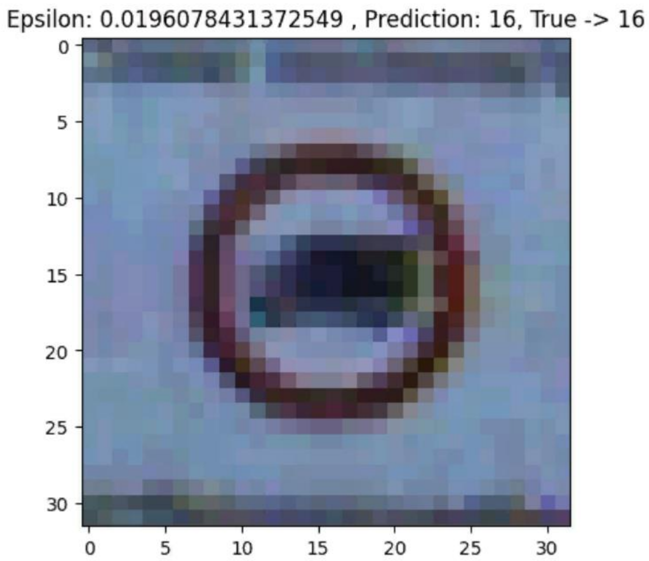
```
for eps in epsilons:
    attack_fgsm.set_params(**{'eps': eps})
    x_test_adv = attack_fgsm.generate(x_test, y_test)
    pred = np.argmax(model.predict(x_test_adv[13:14]))

    plt.figure(i)
    plt.title(f"Epsilon: {eps} , Prediction: {pred}, True -> {np.argmax(y_test[0])}")
    plt.imshow(x_test_adv[14])
    plt.show()
    i += 1
```

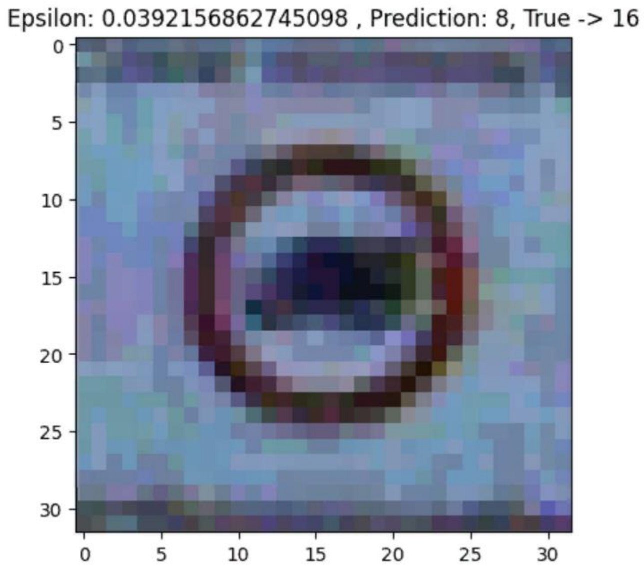

Epsilon: 0, Prediction: 5, True -> 4

Epsilon: 0.00392156862745098 , Prediction: 5, True -> 4

Epsilon: 0.0196078431372549 , Prediction: 5, True -> 4

Epsilon: 0.0392156862745098 , Prediction: 5, True -> 4

Epsilon: 0.19607843137254902 , Prediction: 5, True -> 4

Epsilon: 0.3137254901960784 , Prediction: 5, True -> 4

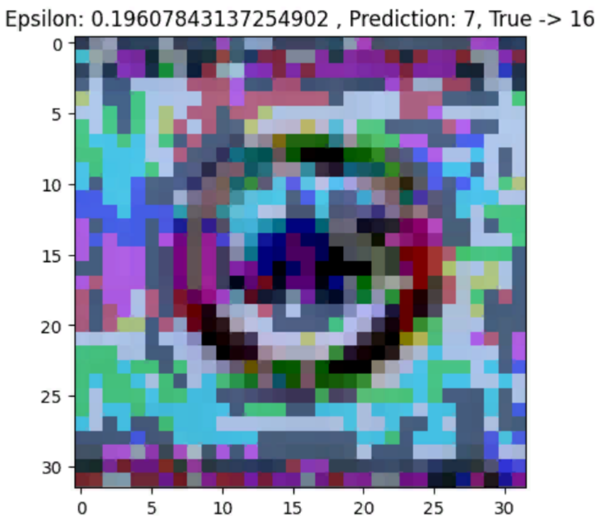
3) Переопределим значения epsilons выведем изображение под номером 16 после атаки FGSM в сравнении с исходным уже для модели VGG16;

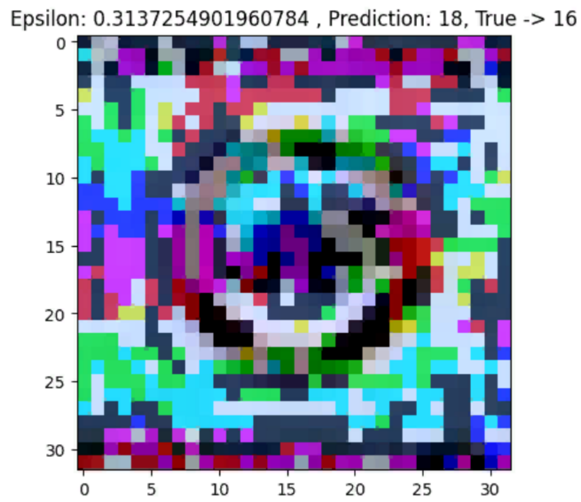




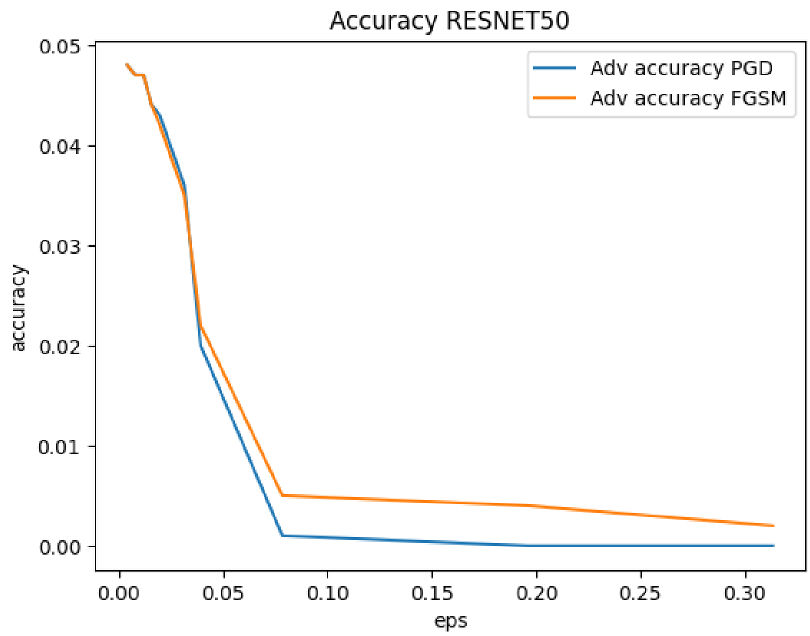




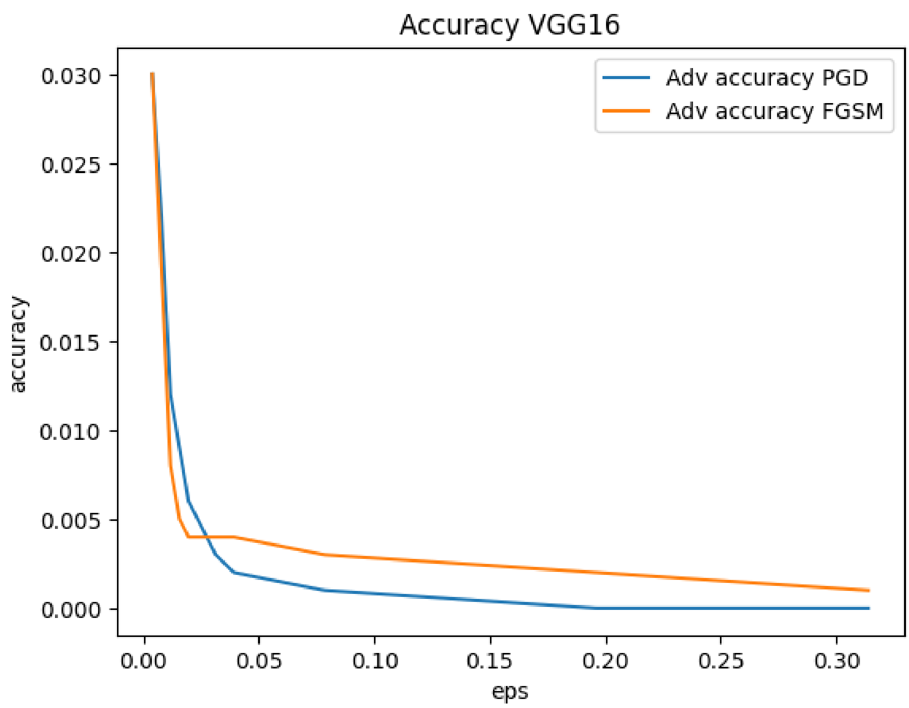




4) Для модели ResNet50 построим график зависимости точности классификации от параметра искажений ϵ;



5) Для модели VGG16 также построим график зависимости точности классификации от параметра искажений ϵ;



6)Сохраним полученные значения точности для каждой из атак, аналогично для обеих моделей;

```
[19] attack_fgsm = FastGradientMethod(estimator=classifier, eps=0)
     adv_accuracises_fgsm = []
     x_test_adv = attack_fgsm.generate(x_test, y_test)
     loss, accuracy = model.evaluate(x_test_adv, y_test)
     adv_accuracises_fgsm.append(accuracy)
     resnetfgsm_0=accuracy

     attack_fgsm = FastGradientMethod(estimator=classifier, eps=1/255)
     adv_accuracises_fgsm = []
     x_test_adv = attack_fgsm.generate(x_test, y_test)
     loss, accuracy = model.evaluate(x_test_adv, y_test)
     adv_accuracises_fgsm.append(accuracy)
     resnetfgsm_1=accuracy

     attack_fgsm = FastGradientMethod(estimator=classifier, eps=5/255)
     adv_accuracises_fgsm = []
     x_test_adv = attack_fgsm.generate(x_test, y_test)
     loss, accuracy = model.evaluate(x_test_adv, y_test)
     adv_accuracises_fgsm.append(accuracy)
     resnetfgsm_2=accuracy

     attack_fgsm = FastGradientMethod(estimator=classifier, eps=10/255)
     adv_accuracises_fgsm = []
     x_test_adv = attack_fgsm.generate(x_test, y_test)
     loss, accuracy = model.evaluate(x_test_adv, y_test)
     adv_accuracises_fgsm.append(accuracy)
     resnetfgsm_3=accuracy
```

```
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, y_test)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_0=accuracy

attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=1/255, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, y_test)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_1=accuracy

attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=5/255, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, y_test)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_2=accuracy

attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=10/255, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, y_test)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_3=accuracy
```

7)Выведем получившиеся значения в виде таблицы;

```python
from tabulate import tabulate

table = [["Model","Source Image","Adversarial images ϵ=1/255","Adversarial images ϵ=5/255","Adversarial images ϵ=10/255"],
            ["Resnet50 - FGSM",resnetfgsm_0,resnetfgsm_1,resnetfgsm_2,resnetfgsm_3],
            ["Resnet50 - PGD",resnetpgd_0,resnetpgd_1,resnetpgd_2,resnetpgd_3],
            ["VGG16 - FGSM",vggfgsm_0,vggfgsm_1,vggfgsm_2,vggfgsm_3],
            ["VGG16 - PGD",vggpgd_0,vggpgd_1,vggpgd_2,vggpgd_3]]

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

```
+-----------------+----------------+------------------------------+------------------------------+-------------------------------+
| Model           |   Source Image |   Adversarial images ϵ=1/255 |   Adversarial images ϵ=5/255 |   Adversarial images ϵ=10/255 |
+=================+================+==============================+==============================+===============================+
| Resnet50 - FGSM |          0.052 |                        0.048 |                        0.042 |                         0.022 |
+-----------------+----------------+------------------------------+------------------------------+-------------------------------+
| Resnet50 - PGD  |          0.052 |                        0.048 |                        0.043 |                         0.02  |
+-----------------+----------------+------------------------------+------------------------------+-------------------------------+
| VGG16 - FGSM    |          0.03  |                        0.03  |                        0.004 |                         0.004 |
+-----------------+----------------+------------------------------+------------------------------+-------------------------------+
| VGG16 - PGD     |          0.03  |                        0.03  |                        0.006 |                         0.002 |
+-----------------+----------------+------------------------------+------------------------------+-------------------------------+
```

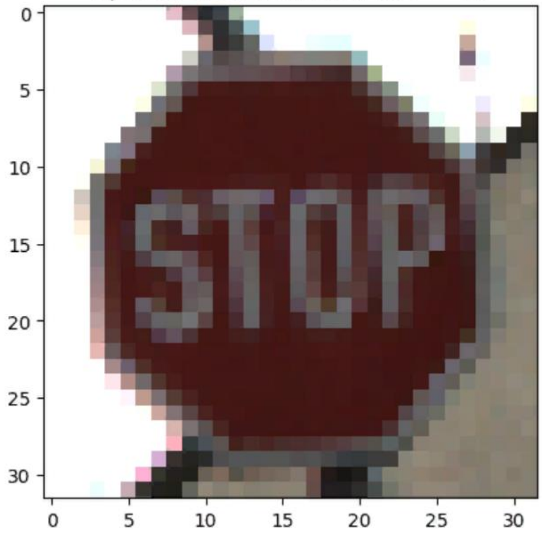
Задание 3

1)Загрузим наше изображение знака «Стоп» и будем стараться его определять, как изображение «Ограничение скорости 30»

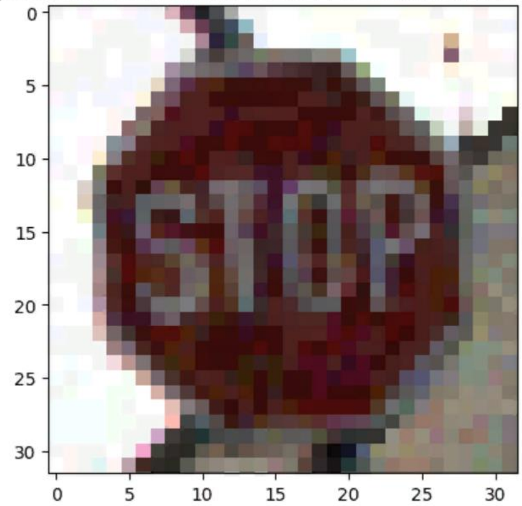
```python
test = pd.read_csv("Test.csv")
test_imgs = test['Path'].values
data = []
y_test = []
labels = test['ClassId'].values.tolist()
i = -1
for img in test_imgs:
    i += 1
    if labels[i] != 14:
      continue
    img = image.load_img(img, target_size=(32, 32))
    img_array = image.img_to_array(img)
    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)
```

2)Построим 5 примеров и соответствующие атакующие FGSM примеры;

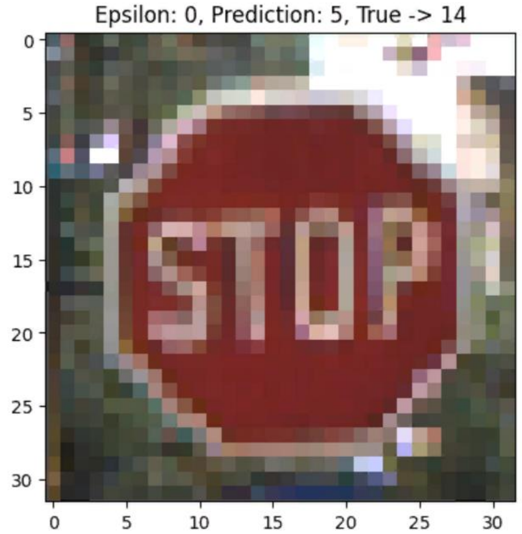
Epsilon: 0, Prediction: 5, True -> 14



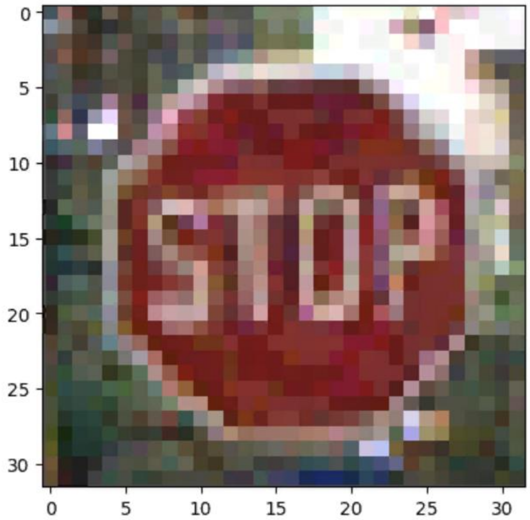
Epsilon: 0.0392156862745098 , Prediction: 5, True -> 14



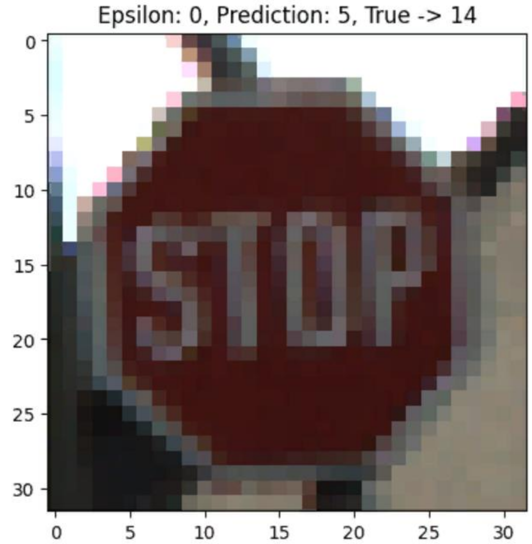
Epsilon: 0, Prediction: 5, True -> 14



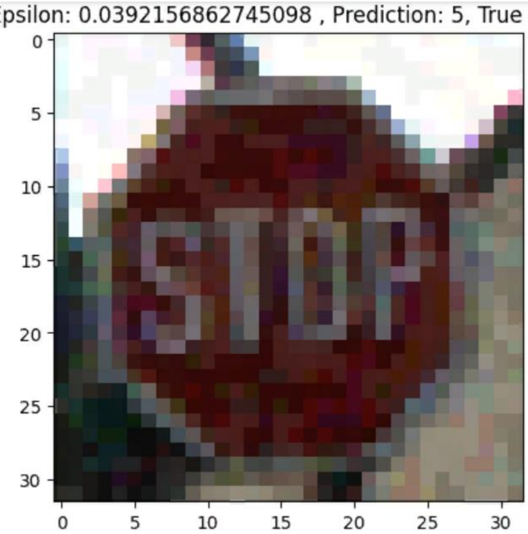
Epsilon: 0.0392156862745098 , Prediction: 5, True -> 14

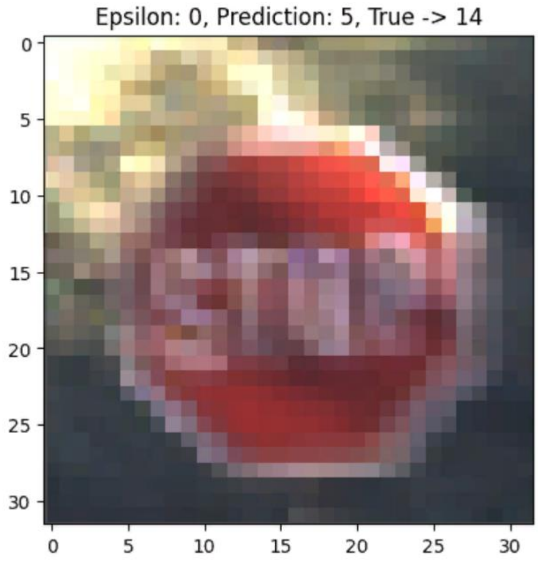


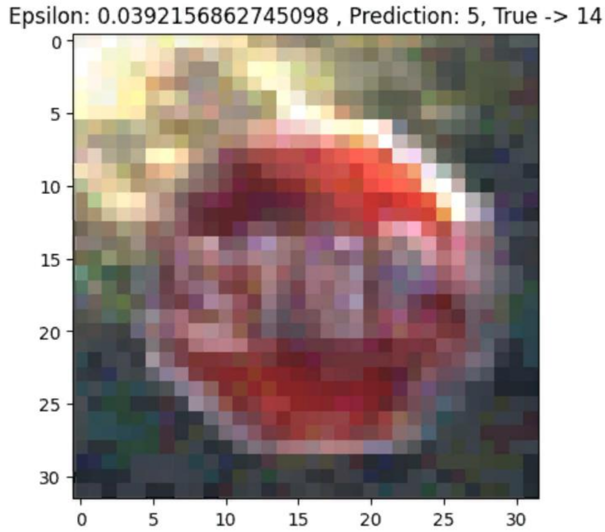
Epsilon: 0, Prediction: 5, True -> 14

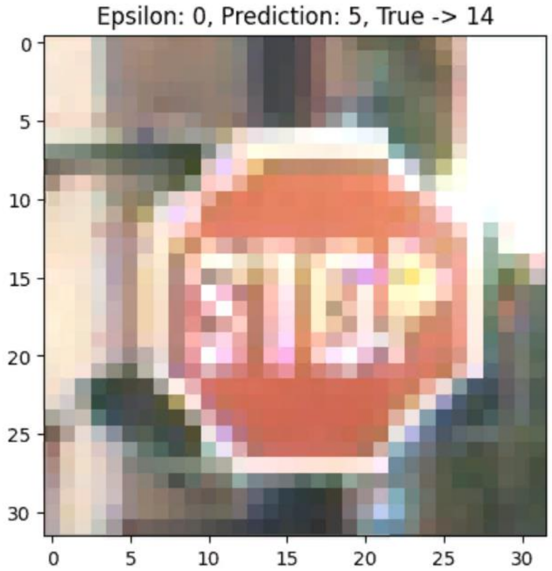


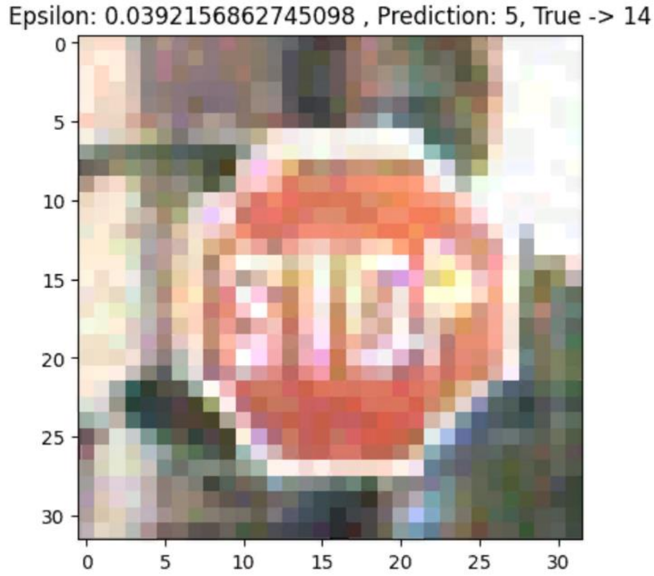
Epsilon: 0.0392156862745098 , Prediction: 5, True -> 14



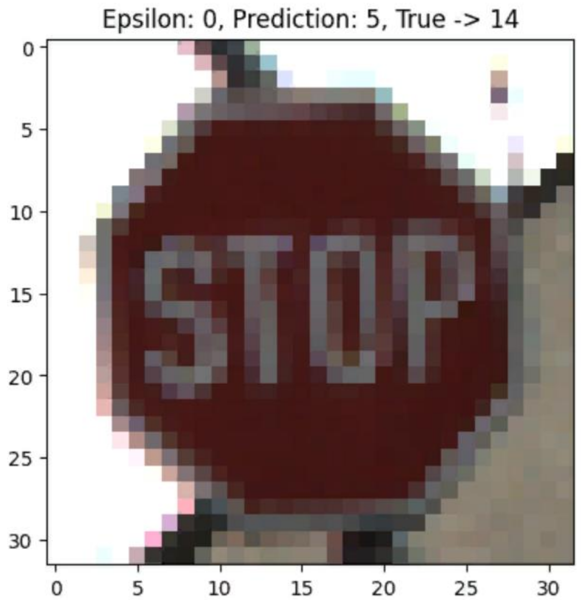


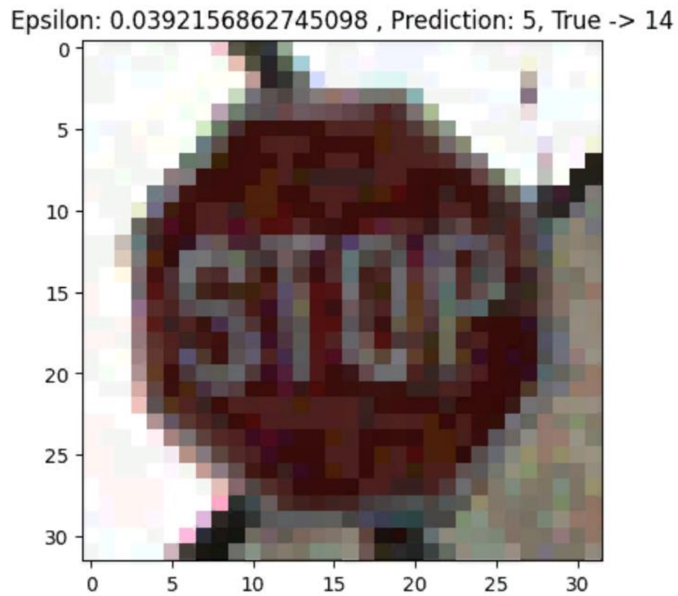


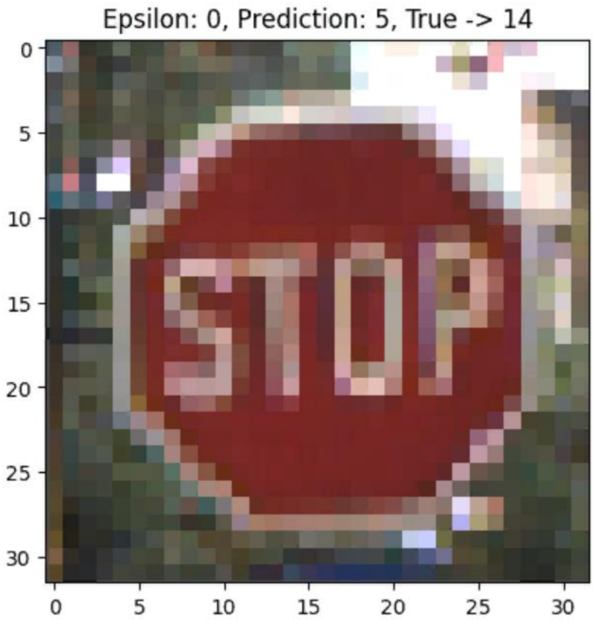


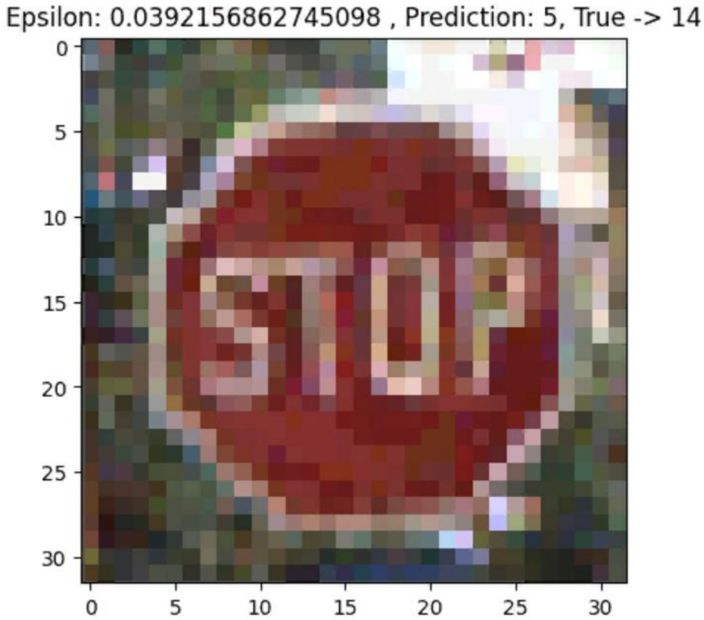


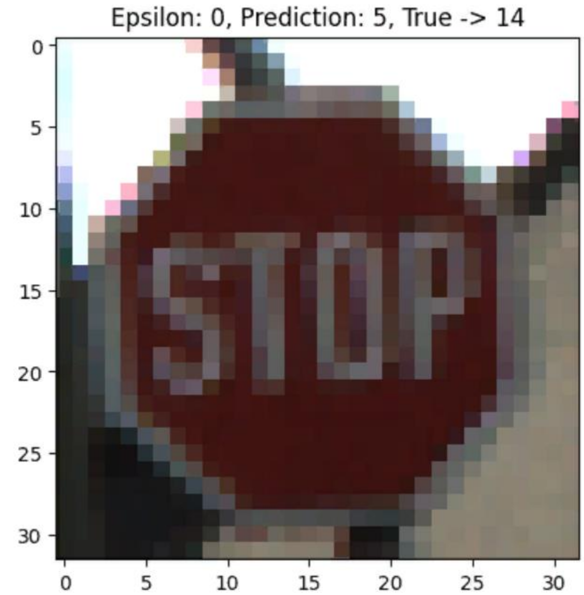
3)Повторим атаку PGD;

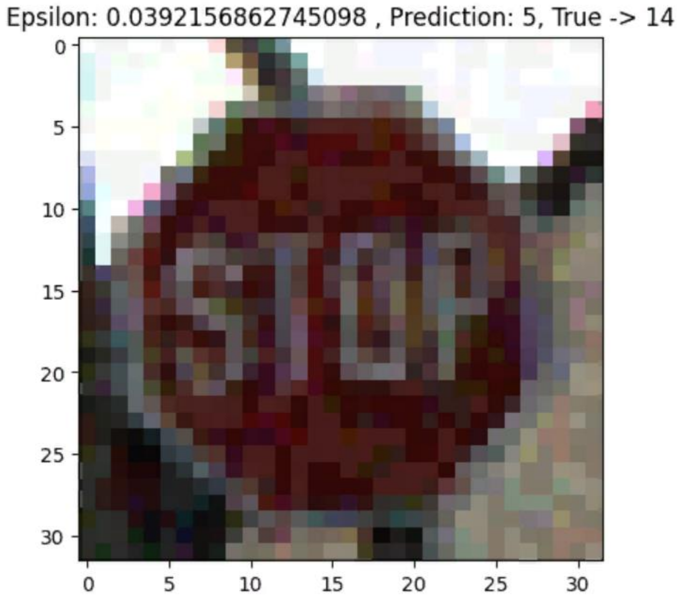


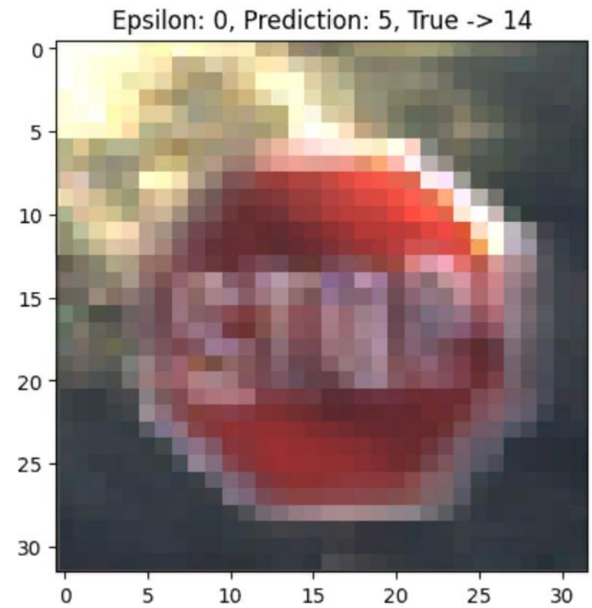


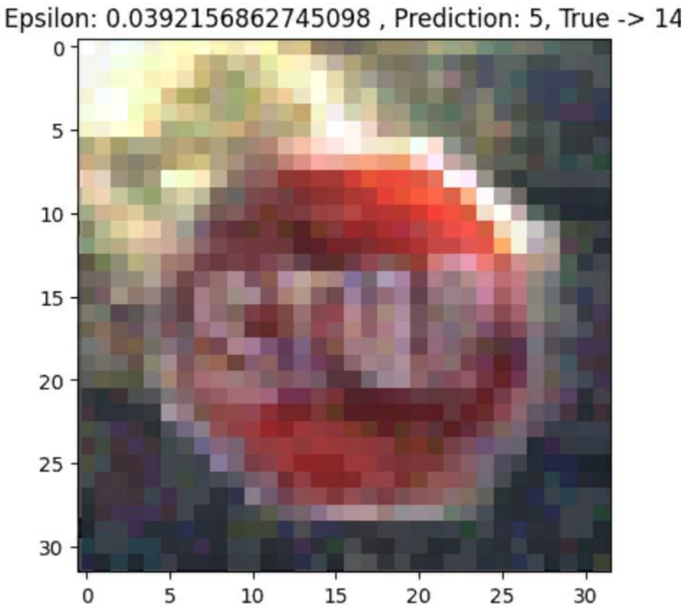

Epsilon: 0, Prediction: 5, True -> 14

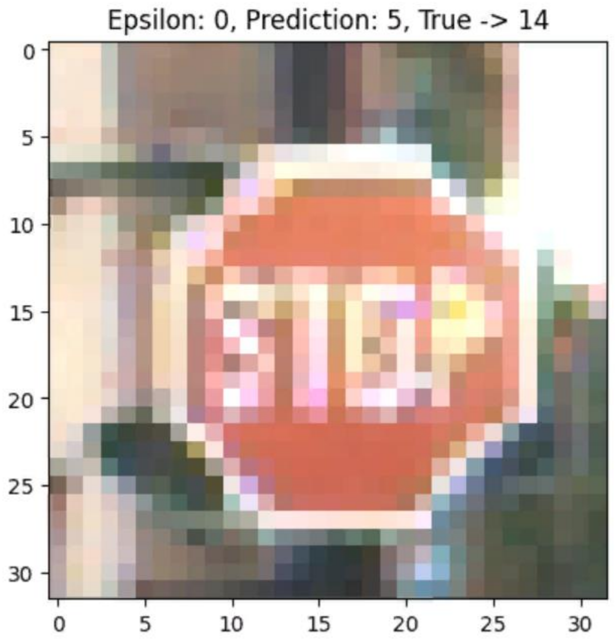

Epsilon: 0.0392156862745098 , Prediction: 5, True -> 14

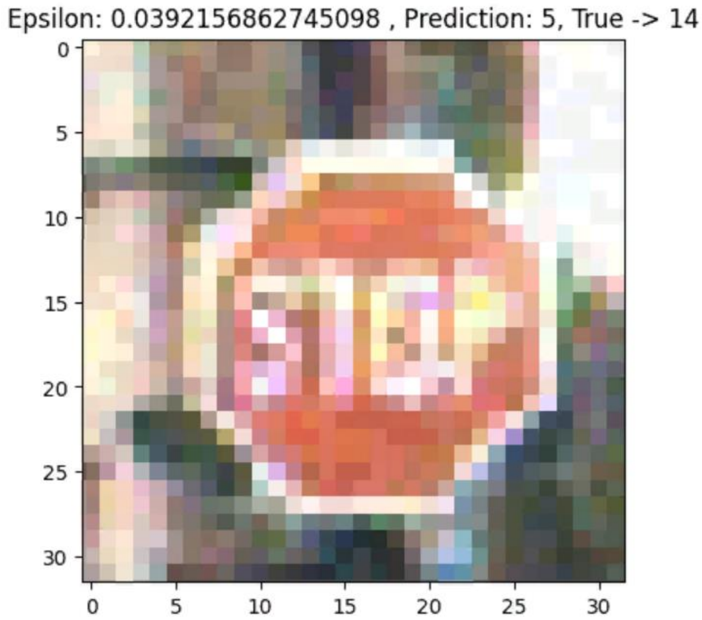

Epsilon: 0, Prediction: 5, True -> 14


Epsilon: 0.0392156862745098 , Prediction: 5, True -> 14


Epsilon: 0, Prediction: 5, True -> 14


Epsilon: 0.0392156862745098 , Prediction: 5, True -> 14





4)Сохраним все значения точности идентично приведённым ниже и выведем значения в таблицу;

[14]
```python
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=1/255, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, t_classes)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_0=accuracy

attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=3/255, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, t_classes)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_1=accuracy

attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=5/255, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, t_classes)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_2=accuracy

attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=10/255, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, t_classes)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_3=accuracy

attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=20/255, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, t_classes)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_4=accuracy

attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=50/255, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, t_classes)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_5=accuracy

attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=80/255, max_iter=4, verbose=False)
adv_accuracises_pgd = []
x_test_adv = attack_pgd.generate(x_test, t_classes)
loss, accuracy = model.evaluate(x_test_adv, y_test)
adv_accuracises_pgd.append(accuracy)
resnetpgd_6=accuracy
```

```python
from tabulate import tabulate

table = [["Sign - Attack","Adversarial images ϵ=1/255","Adversarial images ϵ=3/255","Adversarial images ϵ=5/255","Adversarial images ϵ=10/255","Adversarial images ϵ=20/255","Adversarial images ϵ=50/255","Adversarial images ϵ=80/255"],
         ["Stop Sign - FGSM",resnetfgsm_0,resnetfgsm_1,resnetfgsm_2,resnetfgsm_3,resnetfgsm_4,resnetfgsm_5,resnetfgsm_6],
         ["Stop Sign - PGD",resnetpgd_0,resnetpgd_1,resnetpgd_2,resnetpgd_3,resnetpgd_4,resnetpgd_5,resnetpgd_6],]

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

Sign - Attack	Adversarial images ϵ=1/255	Adversarial images ϵ=3/255	Adversarial images ϵ=5/255	Adversarial images ϵ=10/255	Adversarial images ϵ=20/255	Adversarial images ϵ=50/255	Adversarial images ϵ=80/255
Stop Sign - FGSM	0.96	0.92	0.64	0.3	0.03	0	0
Stop Sign - PGD	0.97	0.93	0.86	0.77	0.45	0.1	0.04

Вывод:

В задание 2 по графикам можно определить, что для полученной модели, атака FGSM показала себя лучше, точность падает более медленно и сбалансированно, что позволяет применять значения epsilons точечно и в большом диапазоне.

В задании 3 было получено обратное, метод FGSM показал себя хуже. В данном случае PGD была более стабильной и отлично справилась на целевой атаке. Наиболее оптимальное значение было получено при epsilon = 20/255.