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»

Лабораторная работа №2
По дисциплине
“Анализ защищенности системы информационной безопасности”

Группа
ББМО-01-22

Выполнил:
Дмитриев М.Н.

Проверил:
Спирин А.А.

Москва 2023

Задание 1

Выполним установку инструмента adversarial-robustness-toolbox:

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

Импортируем необходимые библиотеки.

```
# 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
```

Далее нам нужно подключить гугл диск для возможности корректно и быстро работать с dataset. Разархивируем dataset

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

Разархивируем датасет, который находится на подключенном гугл диске

```
zip_file = '/content/drive/MyDrive/dataset/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')
```

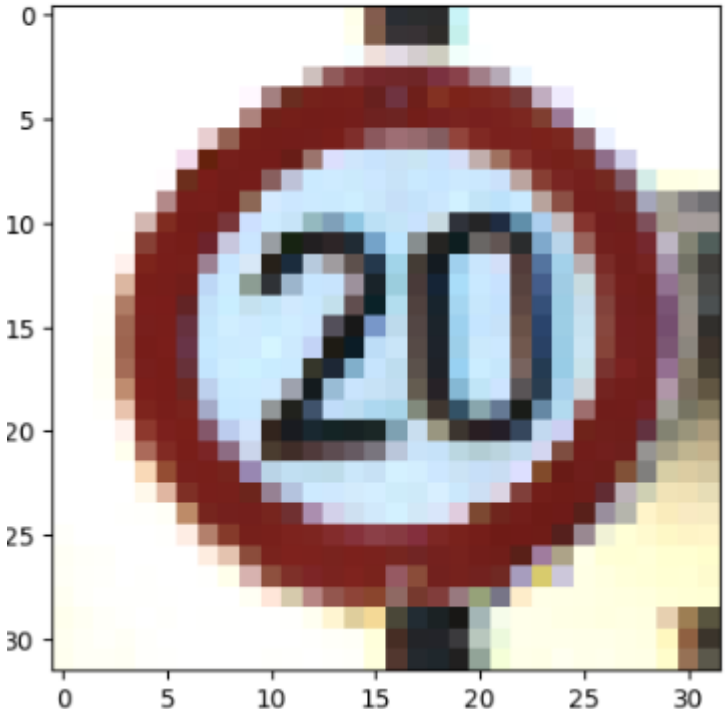
Выполним предварительную обработку изображений из тестового набора.

```
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])
```

Сделаем первый элемент в виде картинки

```
plt.imshow(data[0])
```

```
<matplotlib.image.AxesImage at 0x783822d7a440>
```



После этого выполним разделение данных на тренировочный и тестовый наборы и отобразим размерности этих наборов

```
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. 0. 0.]
```

Создание модель глубокого обученияResNet50 для классификации изображений и отобразим сводку по ней

```
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
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)
```

Обучение модели используя оптимизатор Adam и функцию потерь categorical crossentropy

```
Epoch 1/5
429/429 [==============================] - 67s 66ms/step - loss: 0.9747 - accuracy: 0.7355 - val_loss: 5.9906 - val_accurac
y: 0.0962
Epoch 2/5
429/429 [==============================] - 25s 58ms/step - loss: 0.2569 - accuracy: 0.9313 - val_loss: 0.5161 - val_accurac
y: 0.8527
Epoch 3/5
429/429 [==============================] - 23s 53ms/step - loss: 0.1671 - accuracy: 0.9575 - val_loss: 0.4218 - val_accurac
y: 0.8898
Epoch 4/5
429/429 [==============================] - 24s 56ms/step - loss: 0.0951 - accuracy: 0.9739 - val_loss: 0.0875 - val_accurac
y: 0.9773
Epoch 5/5
429/429 [==============================] - 22s 51ms/step - loss: 0.0832 - accuracy: 0.9781 - val_loss: 0.0962 - val_accurac
y: 0.9731
```

Сохранение модель для последующего использования

```
save_model(model, 'ResNet50.h5')
```

Считаем данные из csv в датафрейм, в ней содержится оригинальная метка класса и путь к изображению

```
test = pd.read_csv("Test.csv")
test_imgs = test['Path'].values
data = []

for img in test_imgs:
    img = image.load_img(img, target_size=(32, 32))
    img_array = image.img_to_array(img)
    img_array = img_array / 255
    data.append(img_array)

data = np.array(data)
y_test = test['ClassId'].values.tolist()
y_test = np.array(y_test)
y_test = to_categorical(y_test, 43)
```

По аналогии с предыдущей, создаем модель для классификации изображений (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_weig
ernels_notop.h5
58889256/58889256 [==============================] - 1s 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)
_________________________________________________________________
```

Обучение модели в течение 5 эпох, используем оптимизатор Adam и функцию потерь categorical crossentropy

```
model2.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
# сохраним историю обучения для последующего анализа на графиках
history2 = model2.fit(x_train, y_train, validation_data =(x_val, y_val), epochs = 5, batch_size = 64)
```

Сохраняем модель для последующего использования

```
save_model(model2, 'VGG16.h5')
```

Выполним оценку производительности двух моделей на тестовом наборе данных

```
history_test = model.fit(x_val, y_val, epochs=5, batch_size=64, validation_data=(x_val, y_val))
history2_test = model2.fit(x_val, y_val, epochs=5, batch_size=64, validation_data=(x_val, y_val))
```

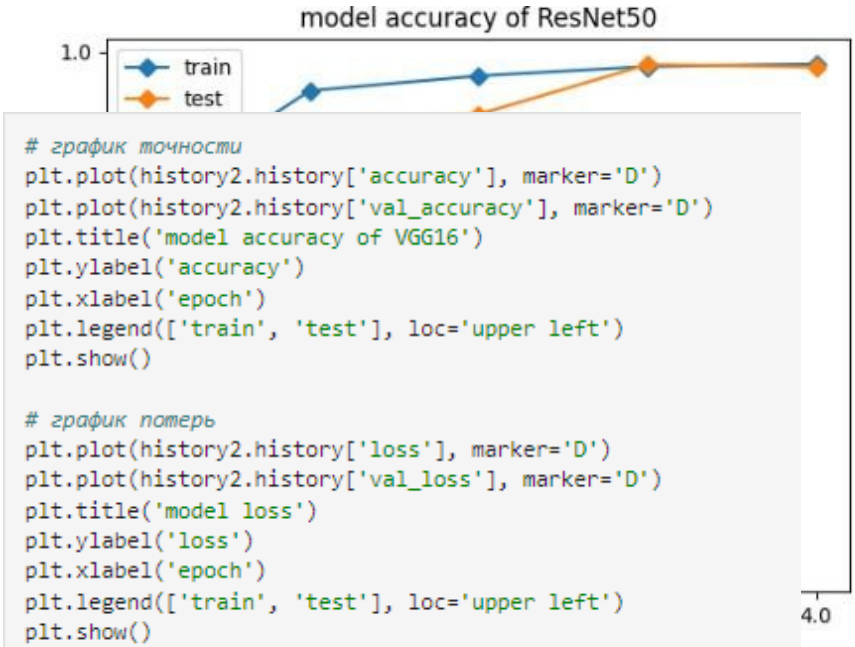
Создание таблицы, которая показывает точность обоих моделей на треннировочном, валидационном и тестовом наборе данных

```
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)
```

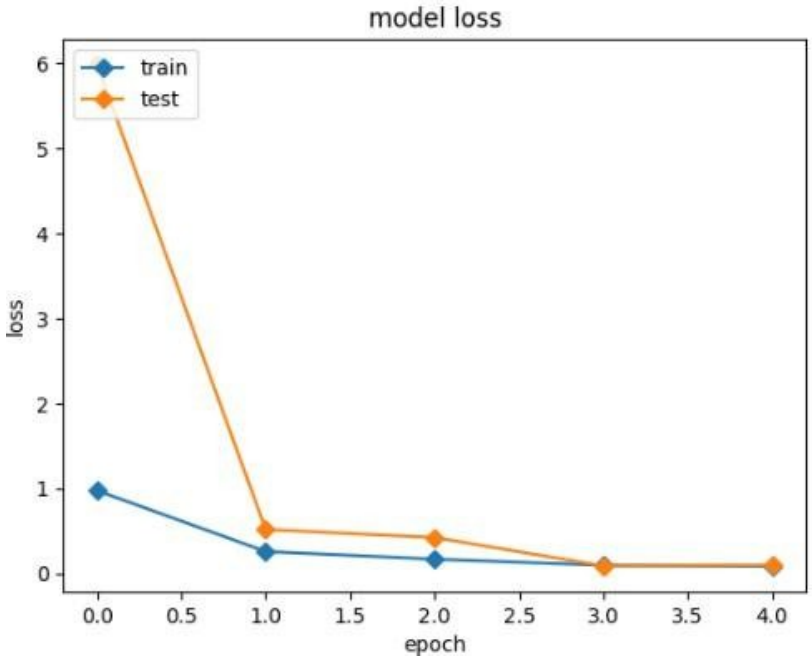
```
+----------+--------------------+----------------------+-----------------+
| Model    |   Training Accuracy |   Validation Accuracy |   Test Accuracy |
+==========+====================+======================+=================+
| Resnet50 |            97.8102 |              97.3051 |         98.7503 |
+----------+--------------------+----------------------+-----------------+
| VGG16    |            98.9288 |              98.4783 |         98.9288 |
+----------+--------------------+----------------------+-----------------+
```

Построим два графика процесса обучения модели ResNet50 графики отражают зависимость метрики от эпохи для тренировочного и тестового наборов

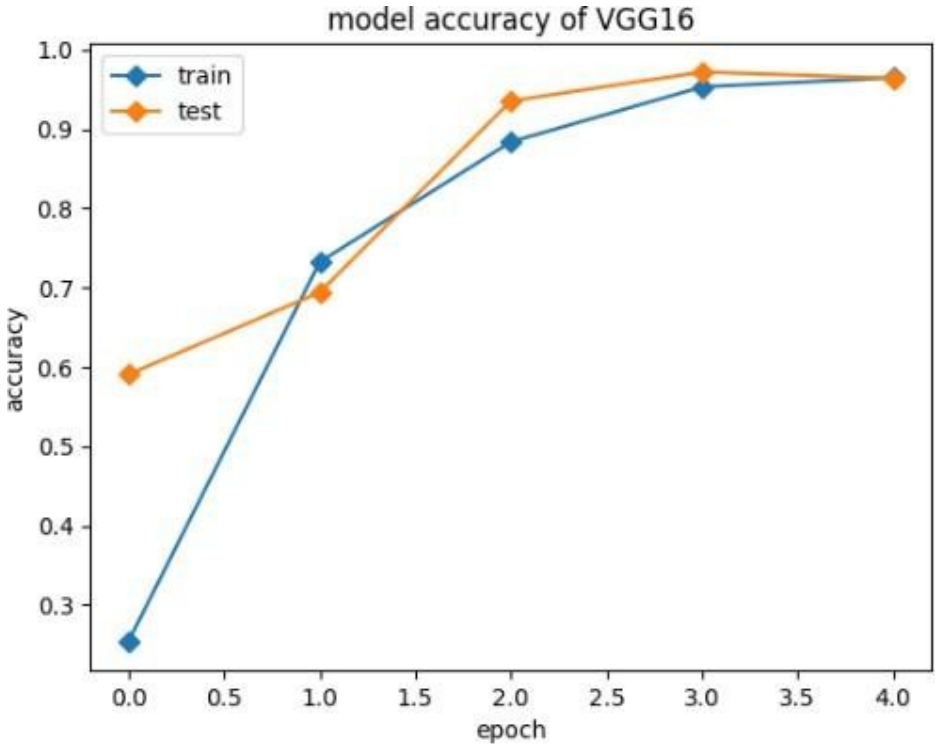
```
# график точности
plt.plot(history.history['accuracy'], marker='D')
plt.plot(history.history['val_accuracy'], marker='D')
plt.title('model accuracy of ResNet50')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

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

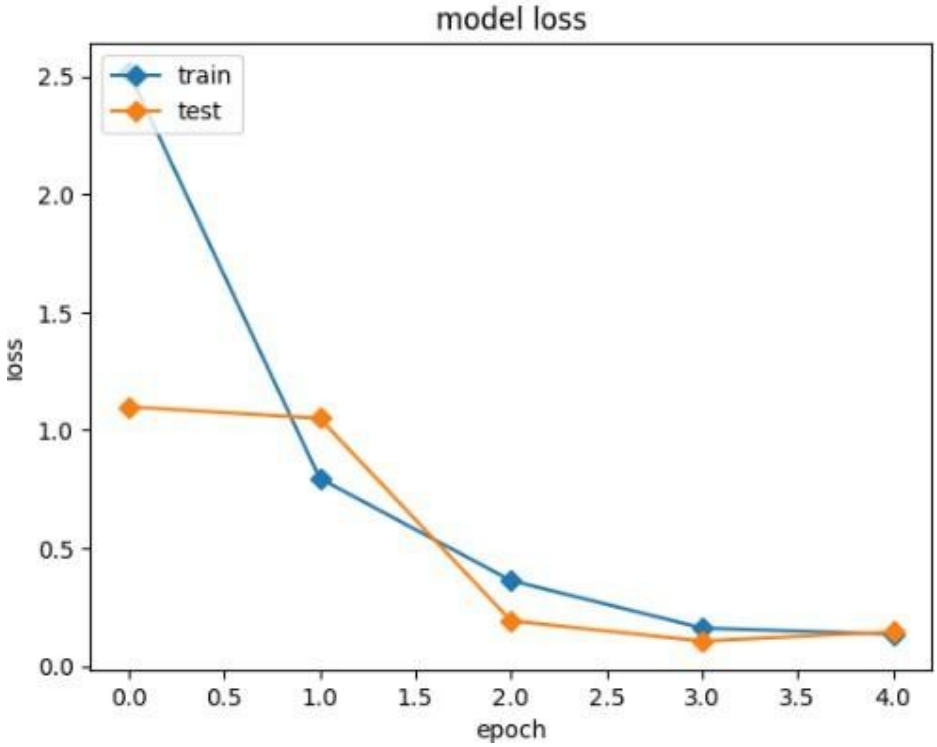



```
# график точности
plt.plot(history2.history['accuracy'], marker='D')
plt.plot(history2.history['val_accuracy'], marker='D')
plt.title('model accuracy of VGG16')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

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



Визуализируем процесс обучения модели VGG16 аналогично с предыдущим





Model	Training Accuracy	Validation Accuracy	Test Accuracy
Resnet50	97.8102	97.3051	98.7503
VGG16	98.9288	98.4783	98.9288

Как видно из таблицы, модели показывают приблизительно схожие результаты, немного лучше оказалась модель VGG16 (по валидации)

Задание 2

Загрузим модель из предыдущего задания и берем тысячу первых элементов из тестового множества, создаем классификатор ART

```
tf.compat.v1.disable_eager_execution()
# для проведения операций сразу, без
# построения графа вычислений

model=load_model('ResNet50.h5')
x_test = data[:1000]
y_test = y_test[:1000]
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
```

Создадим атаку FGSM. Проходимся по диапазону значений eps, который представляет размер шага, с которым FGSN изменяет оригинальные данные для создания адверсариальных параметров

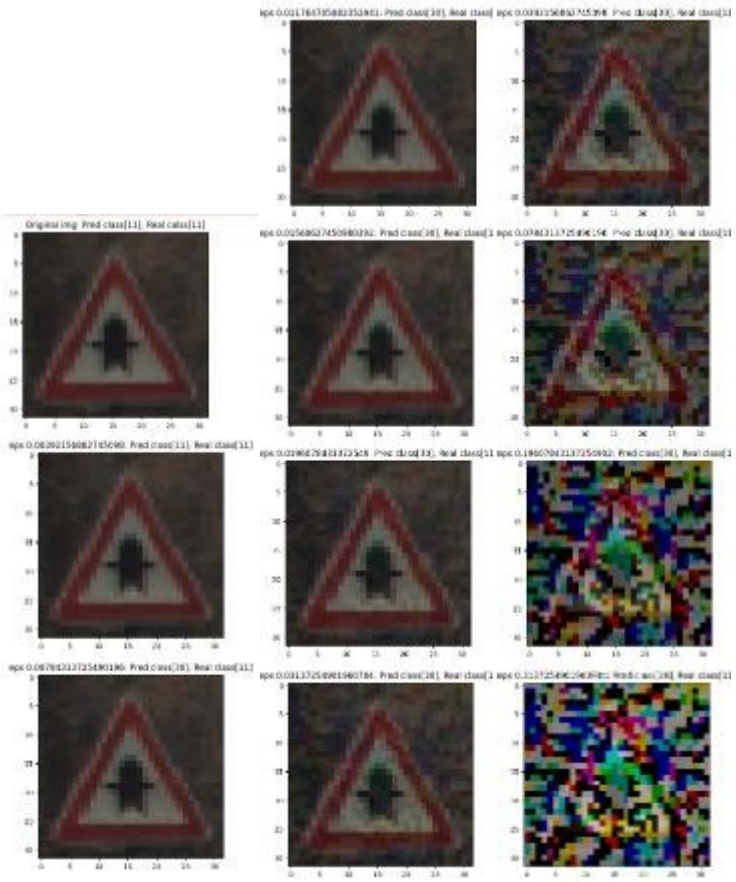
```
attack_fgsm = FastGradientMethod(estimator=classifier, eps=0.3)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
true_accuracies = [] # для точности оригинальных данных
adv_accuracises_fgsm = []
true_losses = [] # для потерь на оригинальных данных
adv_losses_fgsm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

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

Сохраним эту атаку для дальнейшего анализа.

```
adv_losses_fgsm = np.array(adv_losses_fgsm)
adv_accuracises_fgsm = np.array(adv_accuracises_fgsm)
np.save("adv_losses_fgsm_ResNet50", adv_losses_fgsm)
np.save("adv_accuracises_fgsm_ResNet50", adv_accuracises_fgsm)
```

Отобразим исходные и адверсариальные изображения для разных значений eps

```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_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
```



Как видно, ошибки предсказания из-за наложенного шума начались со значения 2/225.

Теперь реализуем атаку PGD для той же модели, создаем атаку по аналогии с предыдущей

```
tf.compat.v1.disable_eager_execution()
model=load_model('ResNet50.h5')
x_test = data[:1000]
y_test = y_test[:1000]
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
```

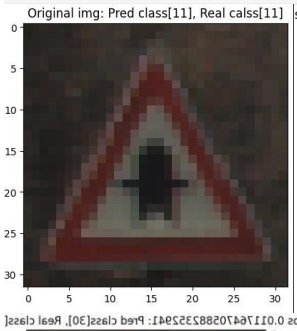
Создаем атаку PGD

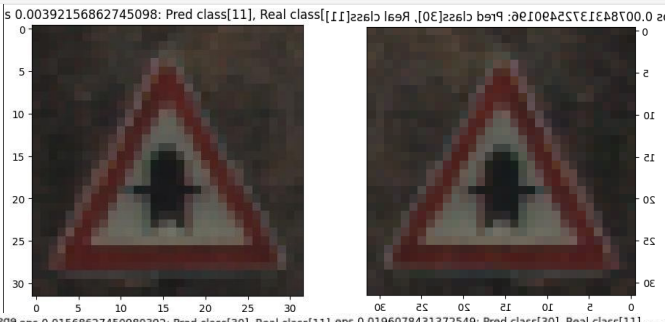
```
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3, max_iter=4, verbose=False)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
true_accuracies = [] # для точности оригинальных данных
adv_accuracises_pgd = []
true_losses = [] # для потерь на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

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}")
```

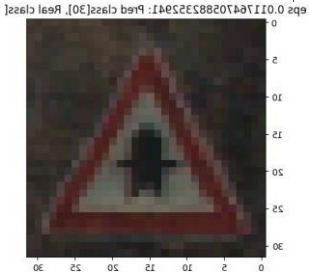
```
adv_losses_pgd = np.array(adv_losses_pgd)
adv_accuracises_pgd = np.array(adv_accuracises_pgd)
np.save("adv_losses_pgd_ResNet50", adv_losses_pgd)
np.save("adv_accuracises_pgd_ResNet50", adv_accuracises_pgd)
```

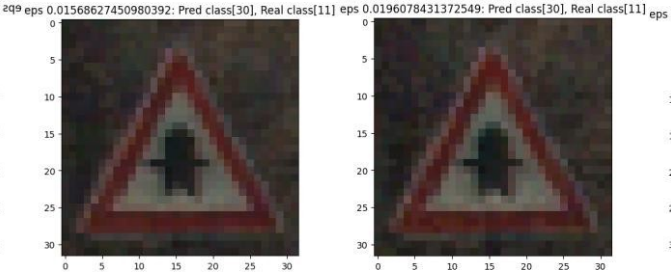
Отображаем исходные и адверсариальные изображения для разных значений eps

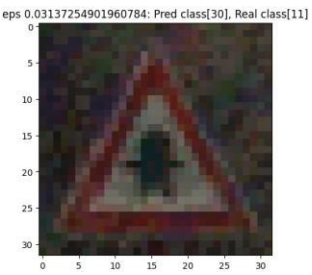
```
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
pred = np.argmax(model.predict(x_test[4:5]))
plt.figure(4)
plt.title(f"Original img: Pred class[{pred}], Real calss[{np.argmax(y_test[4])}]")
plt.imshow(x_test[4])
plt.show()
i = 1

# 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
```

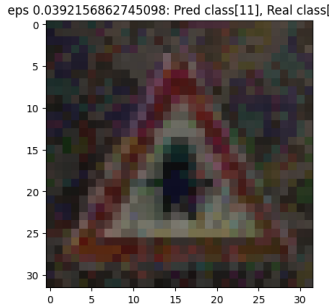


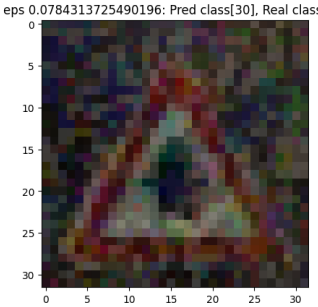


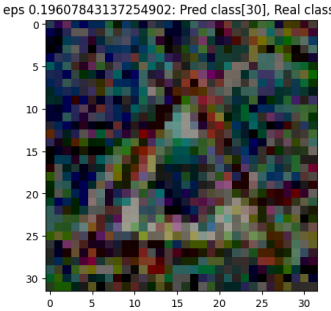


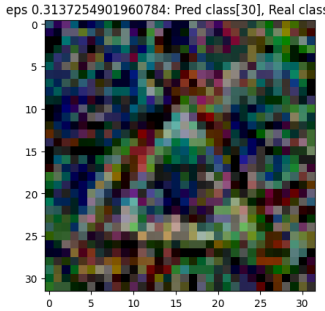






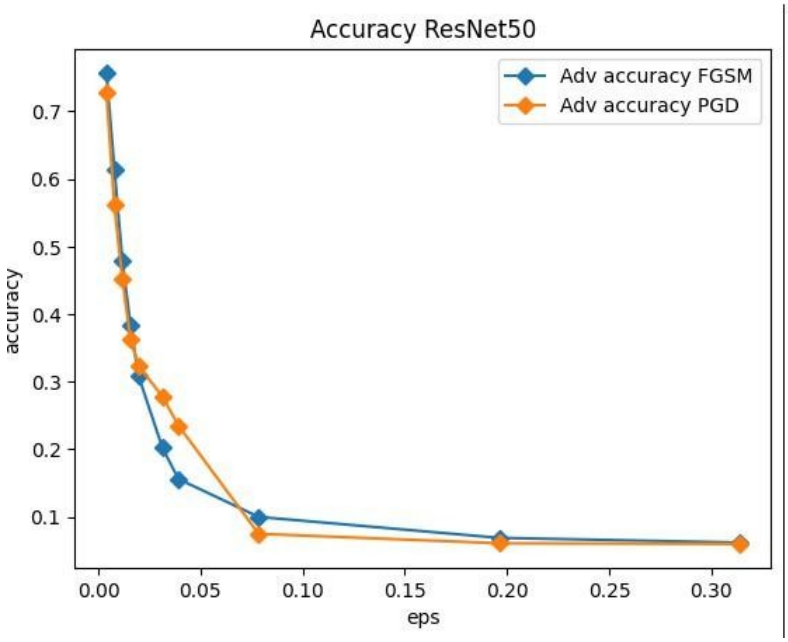






Предсказания стали ложными при параметре 2/255

```
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
# загружаем ранее сохраненный массив адвирсариальных точностей для атак FGSM и PDG
adv_accuracises_fgsm = np.load("adv_accuracises_fgsm_ResNet50.npy")
adv_accuracises_pgd = np.load("adv_accuracises_pgd_ResNet50.npy")
# строим график зависимости адверсариальной точности от значения eps для атак PDG и FGSM
plt.figure(0)
plt.plot(eps_range, adv_accuracises_fgsm, label="Adv accuracy FGSM", marker='D')
plt.plot(eps_range, adv_accuracises_pgd, label="Adv accuracy PGD", marker='D')
plt.title("Accuracy ResNet50")
plt.xlabel("eps")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```



Проделаем аналогичные атаки для VGG16 реализуем атаку FGSM для модели VGG16, которую сохраняли в первом задании

```
tf.compat.v1.disable_eager_execution()
model=load_model('VGG16.h5')
x_test = data[:1000]
y_test = y_test[:1000]
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
```

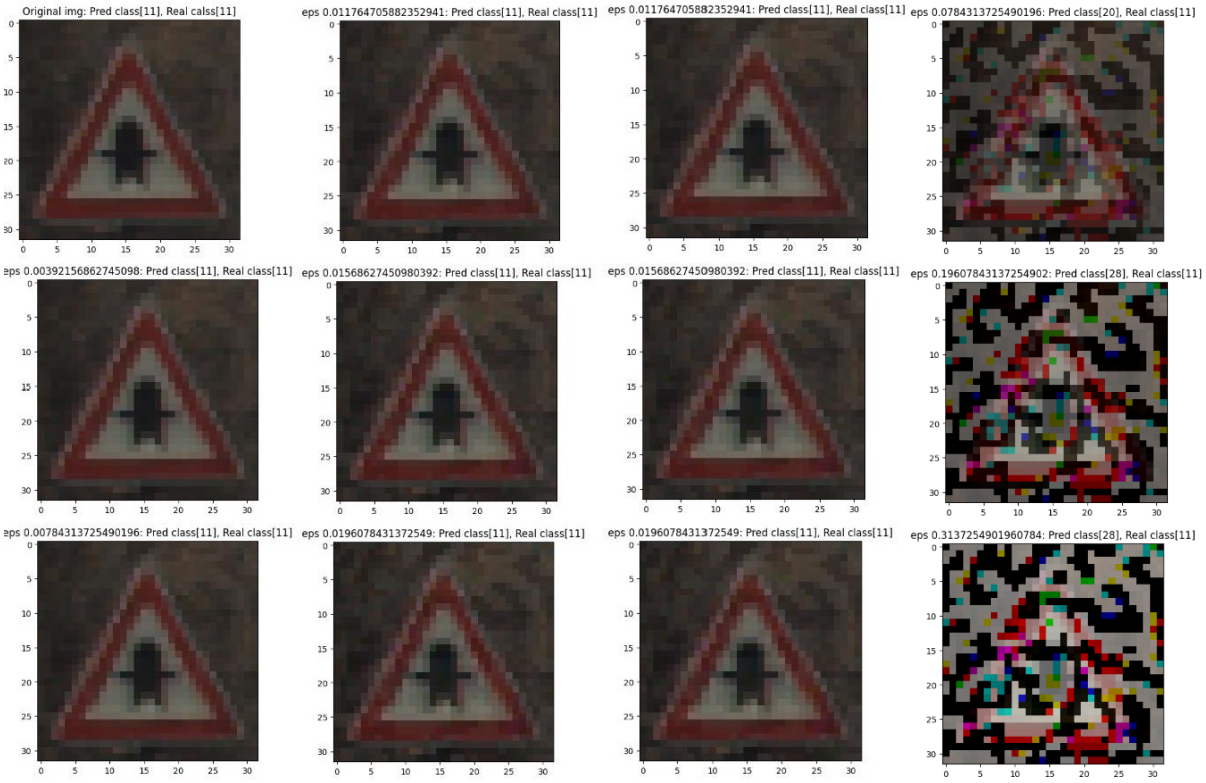
Создаем атаку FGSM по аналогии с VGG16

```
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})
    print(f"Eps: {eps}")
    x_test_adv = attack_fgsm.generate(x_test, y_test)
    loss, accuracy = model.evaluate(x_test_adv, y_test)
    adv_accuracises_fgsm.append(accuracy)
    adv_losses_fgsm.append(loss)
    print(f"Adv 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}")
```

Сохраним атаку FGSM для дальнейшего анализа с помощью графика

```
adv_losses_fgsm = np.array(adv_losses_fgsm)
adv_accuracises_fgsm = np.array(adv_accuracises_fgsm)
np.save("adv_losses_fgsm_VGG16", adv_losses_fgsm)
np.save("adv_accuracises_fgsm_VGG16", adv_accuracises_fgsm)
```

```
# 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
```



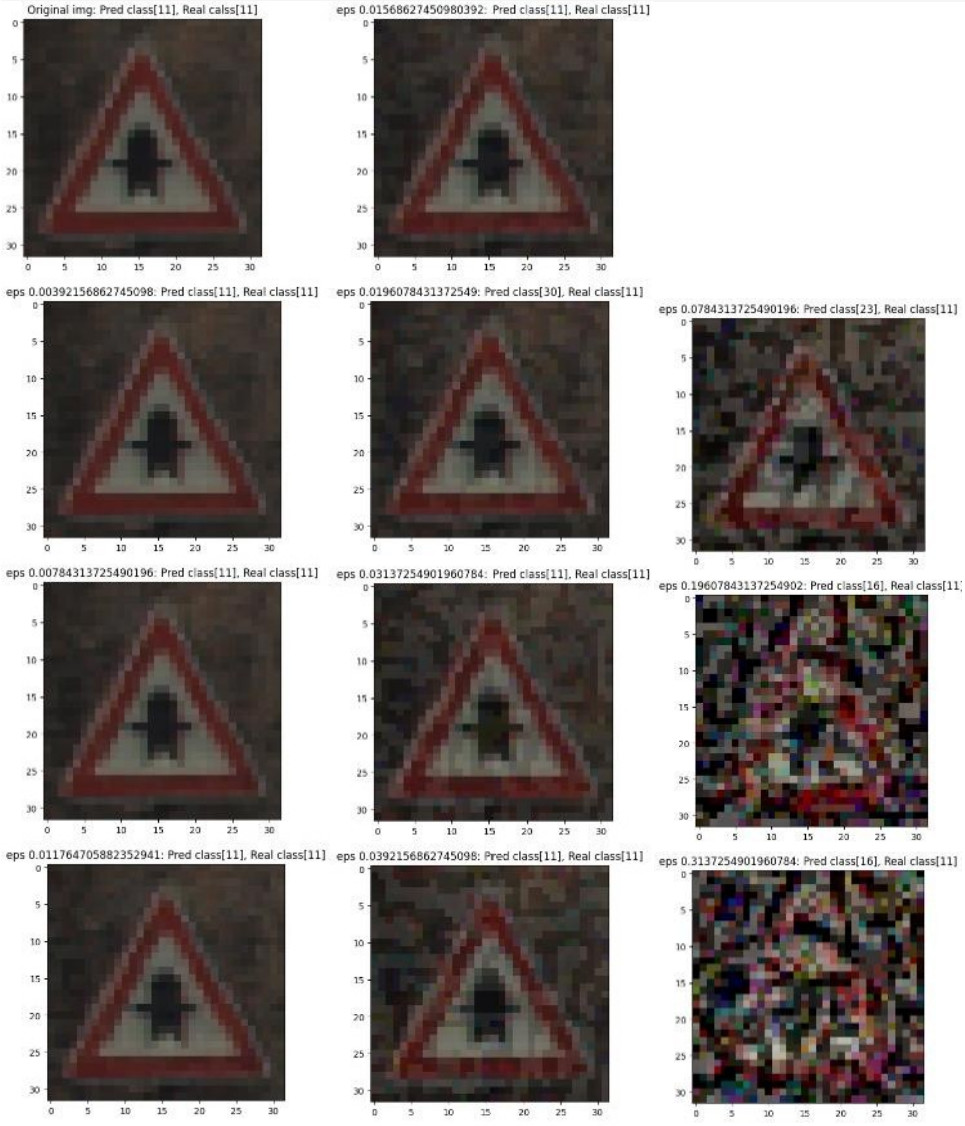
Был выдан ложный результат при значении eps 8/255.

Реализуем атаку PGD для модели VGG16

```
tf.compat.v1.disable_eager_execution()
model=load_model('VGG16.h5')
x_test = data[:1000]
y_test = y_test[:1000]
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
```

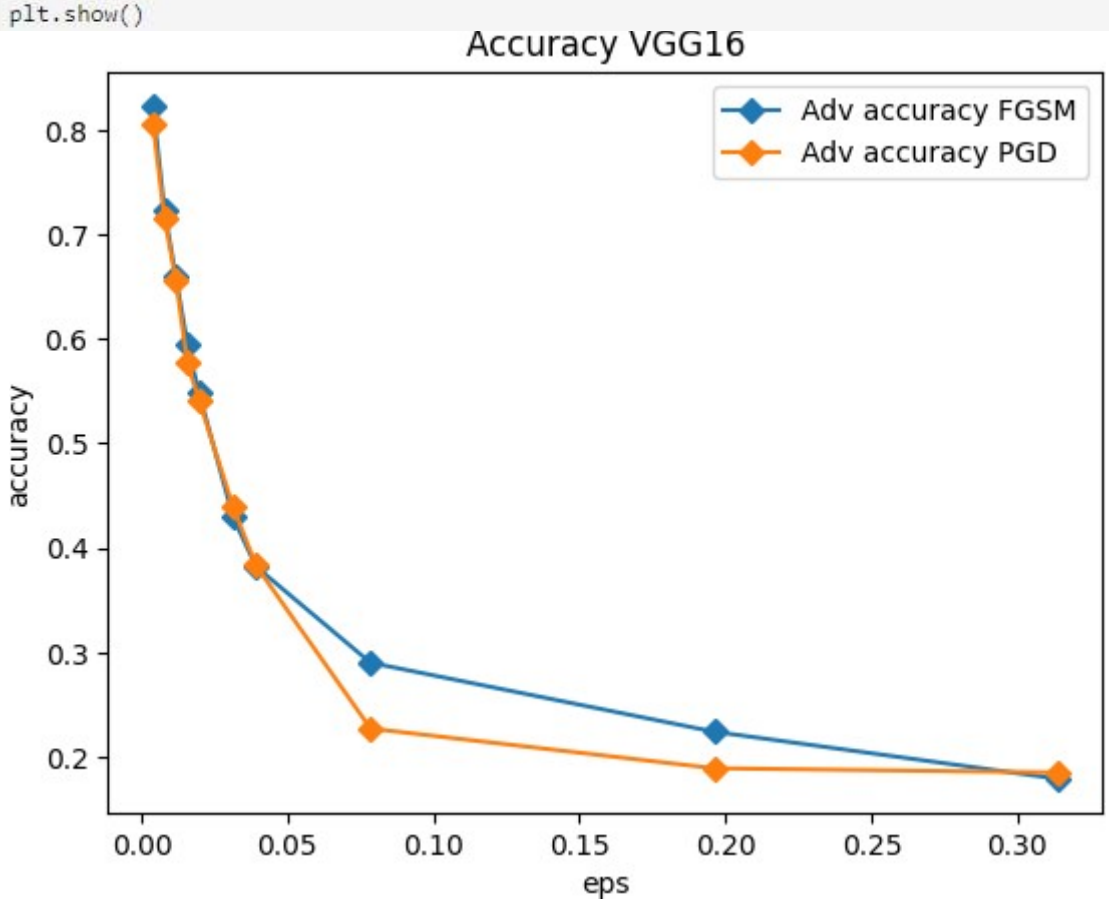
Создаем атаку PGD по аналогии с ResNet50

```
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}")
```



Ошибка предсказания произошла при значении eps 4/255.

```
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
# загружаем ранее сохраненный массив адвирсариальных точностей для атак FGSM и PGD
adv_accuracises_fgsm = np.load("adv_accuracises_fgsm_VGG16.npy")
adv_accuracises_pgd = np.load("adv_accuracises_pgd_VGG16.npy")
# строим график зависимости адверсариальной точности от значения eps для атак PGD и FGSM
plt.figure(0)
plt.plot(eps_range, adv_accuracises_fgsm, label="Adv accuracy FGSM", marker='D')
plt.plot(eps_range, adv_accuracises_pgd, label="Adv accuracy PGD", marker='D')
plt.title("Accuracy VGG16")
plt.xlabel("eps")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```



В случае VGG6, при атаках PGD и FGSM точность сначала падает одинаково, но с повышением значений eps в какой-то момент точнотсть при атаке PGD начинает падать сильнее, но при максимальном значении eps точность сильнее всего упала при атаке FGSM.

```
# eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]
# создадим таблицу со значениями точности для обеих моделей
adv_acc_fgsm_rn50 = np.load("adv_accuracises_fgsm_ResNet50.npy")
adv_acc_pgd_rn50 = np.load("adv_accuracises_pgd_ResNet50.npy")
adv_acc_fgsm_v16 = np.load("adv_accuracises_fgsm_VGG16.npy")
adv_acc_pgd_v16 = np.load("adv_accuracises_pgd_VGG16.npy")

table = [["Model","Original accuracy","eps = 1/255","eps = 2/255", "eps = 3/255", "eps = 4/255", "eps = 5/255", "eps = 8/25
            ["Resnet50 FGSM",train_accuracy[4]*100,adv_acc_fgsm_rn50[0]*100,
             adv_acc_fgsm_rn50[1]*100,adv_acc_fgsm_rn50[2]*100,adv_acc_fgsm_rn50[3]*100,
             adv_acc_fgsm_rn50[4]*100,adv_acc_fgsm_rn50[5]*100,adv_acc_fgsm_rn50[6]*100,
             adv_acc_fgsm_rn50[7]*100,adv_acc_fgsm_rn50[8]*100,adv_acc_fgsm_rn50[9]*100],
            ["Resnet50 PGD",train_accuracy[4]*100,adv_acc_pgd_rn50[0]*100,
             adv_acc_pgd_rn50[1]*100,adv_acc_pgd_rn50[2]*100,adv_acc_pgd_rn50[3]*100,
             adv_acc_pgd_rn50[4]*100,adv_acc_pgd_rn50[5]*100,adv_acc_pgd_rn50[6]*100,
             adv_acc_pgd_rn50[7]*100,adv_acc_pgd_rn50[8]*100,adv_acc_pgd_rn50[9]*100],
            ["VGG16 FGSM",train_accuracy2[4]*100,adv_acc_fgsm_v16[0]*100,
             adv_acc_fgsm_v16[1]*100,adv_acc_fgsm_v16[2]*100,adv_acc_fgsm_v16[3]*100,
             adv_acc_fgsm_v16[4]*100,adv_acc_fgsm_v16[5]*100,adv_acc_fgsm_v16[6]*100,
             adv_acc_fgsm_v16[7]*100,adv_acc_fgsm_v16[8]*100,adv_acc_fgsm_v16[9]*100],
            ["VGG16 PGD",train_accuracy2[4]*100,adv_acc_pgd_v16[0]*100,
             adv_acc_pgd_v16[1]*100,adv_acc_pgd_v16[2]*100,adv_acc_pgd_v16[3]*100,
             adv_acc_pgd_v16[4]*100,adv_acc_pgd_v16[5]*100,adv_acc_pgd_v16[6]*100,
             adv_acc_pgd_v16[7]*100,adv_acc_pgd_v16[8]*100,adv_acc_pgd_v16[9]*100],
        ]

table2 = tabulate(table,headers="firstrow",tablefmt="grid")
print(table2)
```

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	97.8102	75.7	61.4	47.8	38.4	30.9	20.2	15.6	10	6.9	6.2
Resnet50 PGD	97.8102	72.8	56.1	45.1	36.3	32.4	27.8	23.5	7.5	6.1	6
VGG16 FGSM	98.9288	82.3	72.3	65.9	59.5	54.9	42.9	38.2	29	22.4	17.9
VGG16 PGD	98.9288	80.5	71.6	65.6	57.7	54.2	43.9	38.3	22.7	18.9	18.5

Таким образом была отражена таблица для всех показанных атак и моделей. По таблице видно, что точность выше при всех значениях eps у модели VGG16.

Задание 3

Создадим две целевых атаки

Загружаем тестовый набор данных из Test.csv и извлекаем 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)
```

Реализуем целевую атаку FGSM

Сгенерируем адверсариальные примеры и оценим точность модели на них и на исходных тестовых данных

```
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}")
```

Тут эксперементируем со значениями eps для достижения наилучшего резултата.

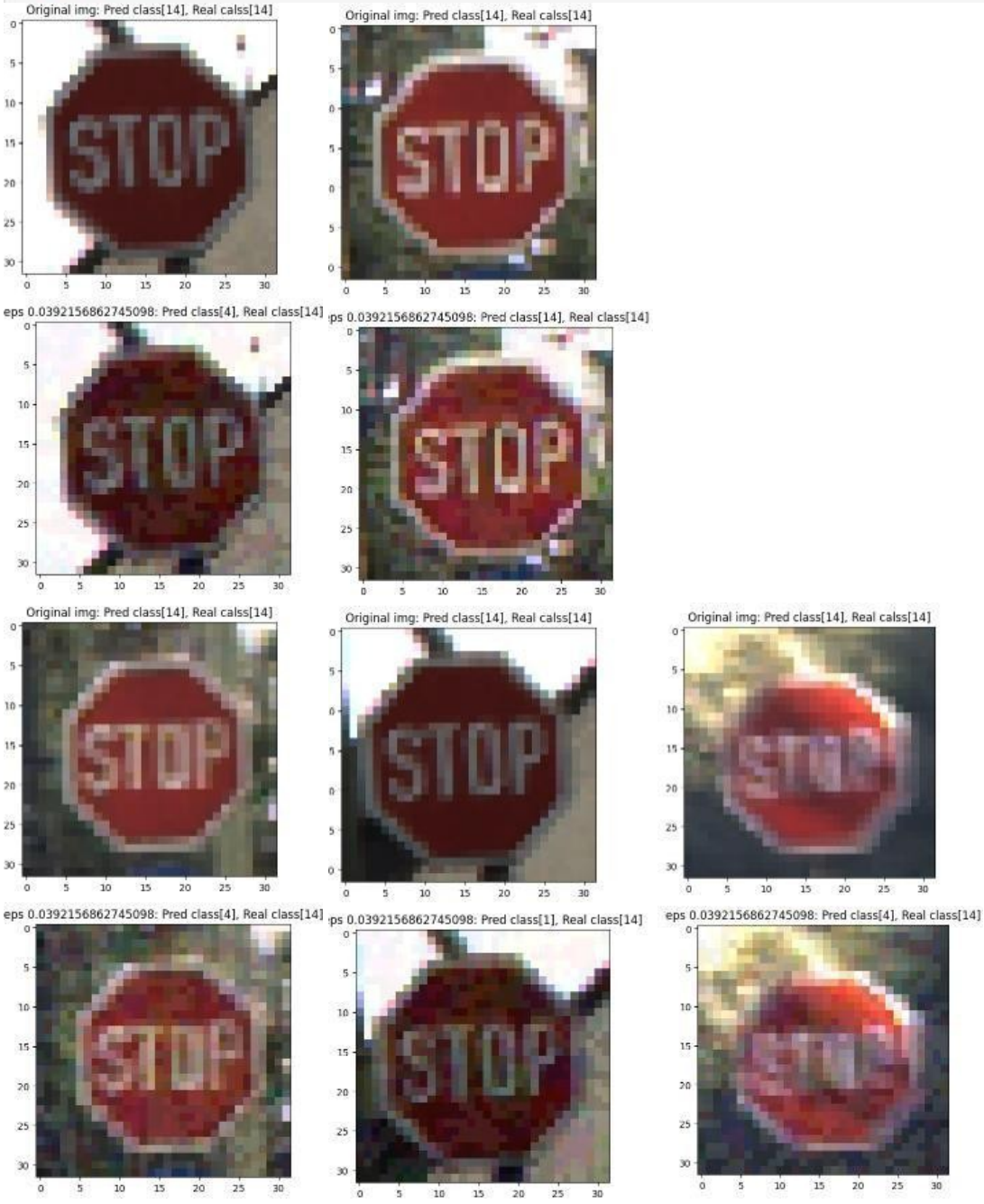
Лучше всего себя показывает 10/255

```
eps = 10/255
attack_fgsm.set_params(**{'eps': eps})
x_test_adv = attack_fgsm.generate(x_test, t_classes)
```

\

Отобразим 5 разных изображений для визуализации действия атаки

```
range = [0, 10, 20, 30, 40]
i = 0
for index in range:
  plt.figure(i)
  pred = np.argmax(model.predict(x_test[index:index+1]))
  plt.title(f"Original img: Pred class[{pred}], Real calss[{np.argmax(y_test[index])}]")
  plt.imshow(x_test[index])
  plt.show()
  i += 1
  pred = np.argmax(model.predict(x_test_adv[index:index+1]))
  plt.figure(i)
  plt.title(f"eps {eps}: Pred class[{pred}], Real class[{np.argmax(y_test[index])}]")
  plt.imshow(x_test_adv[index])
  plt.show()
```



Целевая атака FGSM достигает своего пика на eps = 10/255 в нашем случе, при больших значениях eps атака хоть и будет давать больше неточности при предсказании, но это будут разные классы, в большинстве

случаев отличные от первого (знак стоп), который мы указали. Можно сделать вывод, что FGSM не очень подходит для целевых атак.

Реализуем целевую атаку PGD

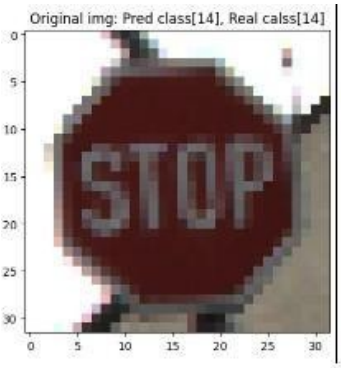
Сгенерируем адверсариальные примеры и оценим точность модели на адвирсариальных примерах и на исходных тестовых данных

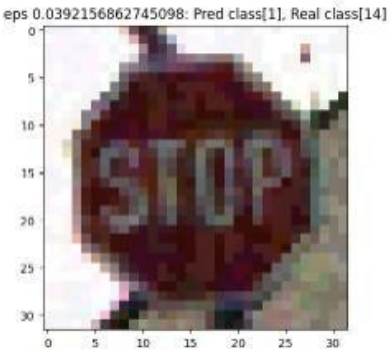
```
model=load_model('ResNet50.h5')
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test), np.max(x_test)))
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3, max_iter=4, verbose=False, targeted=True)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255]

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

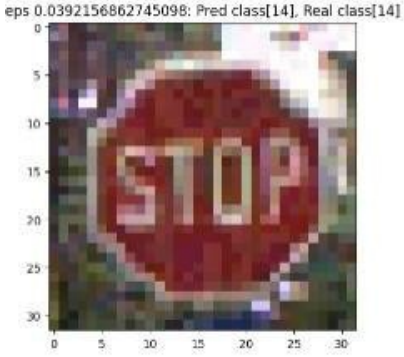
```
eps = 10/255
attack_pgd.set_params(**{'eps': eps})
x_test_adv = attack_pgd.generate(x_test, t_classes)
```

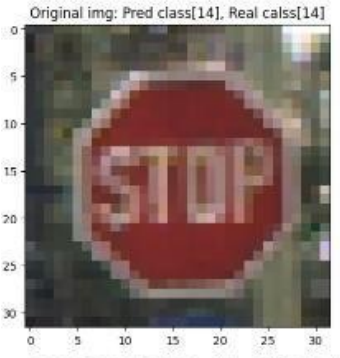
Отобразим 5 разных изображений для визуализации действия атаки

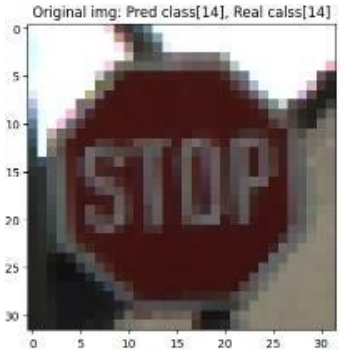
```
range = [0, 10, 20, 30, 40]
i = 0
for index in range:
  plt.figure(i)
  pred = np.argmax(model.predict(x_test[index:index+1]))
  plt.title(f"Original img: Pred class[{pred}], Real calss[{np.argmax(y_test[index])}]")
  plt.imshow(x_test[index])
  plt.show()
  i += 1
  pred = np.argmax(model.predict(x_test_adv[index:index+1]))
  plt.figure(i)
  plt.title(f"eps {eps}: Pred class[{pred}], Real class[{np.argmax(y_test[index])}]")
  plt.imshow(x_test_adv[index])
  plt.show()
```

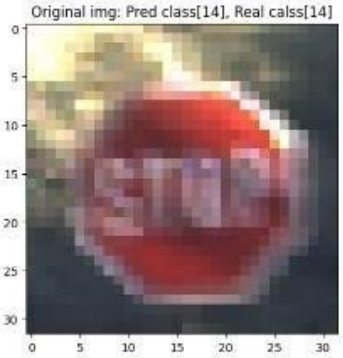


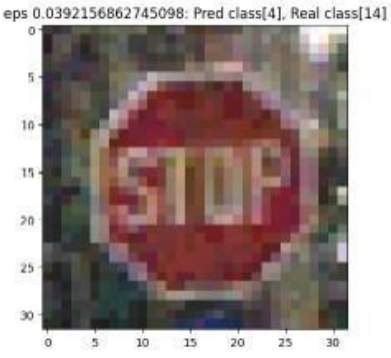


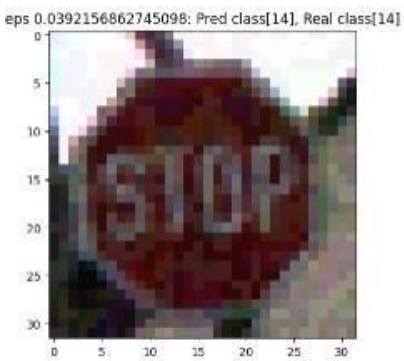


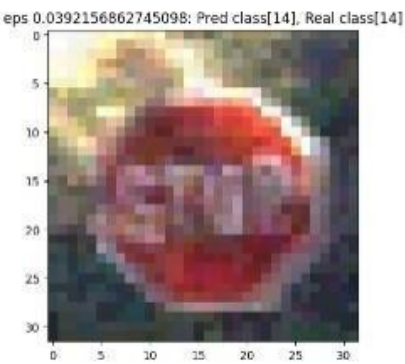












Атака PGD достигает отличных значений при eps 50/255, при таком значении очень много требуемых результатов.

Вывод

Как видим, атака PGD дольше сохранеяет точность, чем FGSM. По результатам видно метод PGD значительно лучше подходит для целевой атаки, чем метод FGSM.