



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

Институт кибербезопасности и цифровых технологий
Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

Отчёт по лабораторной работе № 2

По дисциплине

«Анализ защищенности систем искусственного интеллекта»

Выполнил:

БМО–02–22

Шмарковский М. Б.

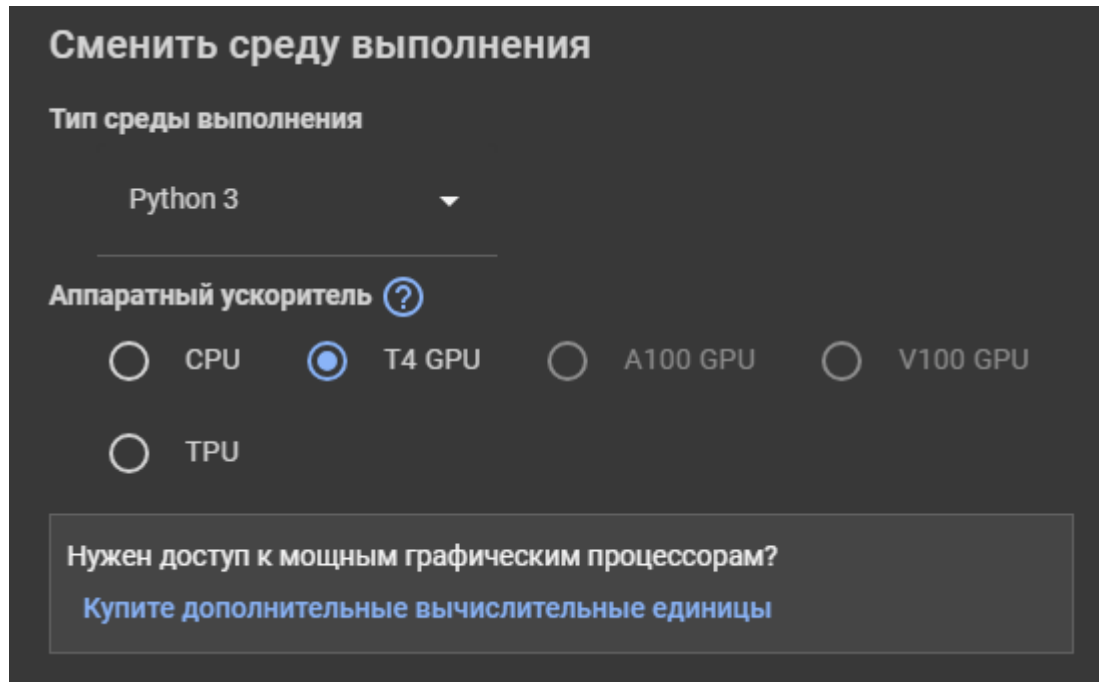
Проверил:

Спирин А. А.

Москва, 2024

Подготовительный этап

Поменяем среду выполнения на GPU:



Сменить среду выполнения

Тип среды выполнения

Python 3 ▼

Аппаратный ускоритель ?

☐ CPU ☒ T4 GPU ☐ A100 GPU ☐ V100 GPU

☐ TPU

Нужен доступ к мощным графическим процессорам?
Купите дополнительные вычислительные единицы

Выполним установку инструмента adversarial-robustness-toolbox:

```
!pip install adversarial-robustness-toolbox

Collecting adversarial-robustness-toolbox
  Downloading adversarial_robustness_toolbox-1.17.0-py3-none-any.whl (1.7 MB)
1.7/1.7 MB 7.3 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.23.5)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.11.4)
Collecting scikit-learn<1.2.0,>=0.22.2 (from adversarial-robustness-toolbox)
  Downloading scikit_learn-1.1.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (30.5 MB)
30.5/30.5 MB 18.1 MB/s eta 0:00:00
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.16.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (67.7.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (4.66.1)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn<1.2.0,>=0.22.2->adversarial-robustness-toolbox) (1.3.2)
```



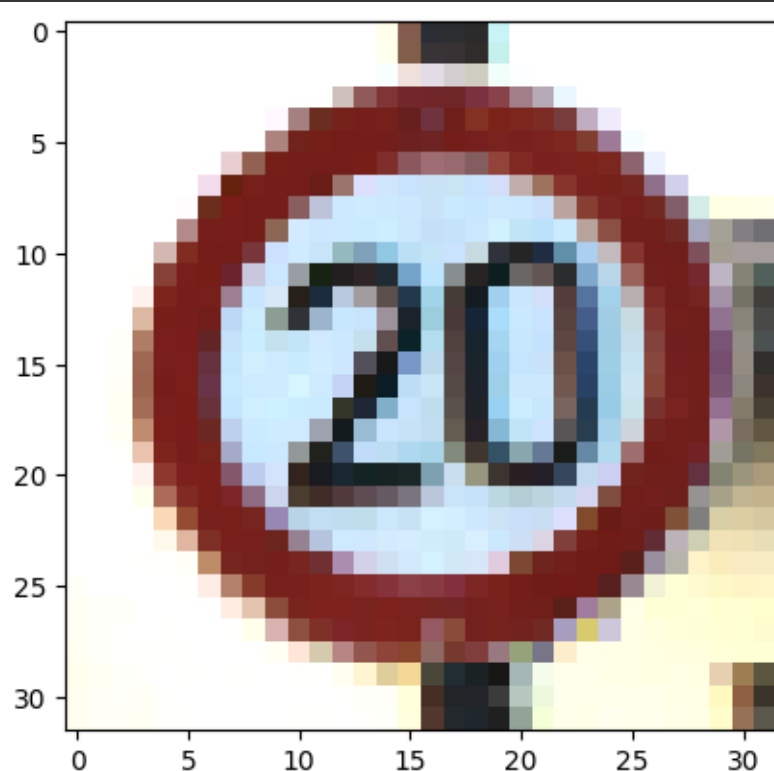
```
from keras.models import Sequential
from keras.optimizers import Adam
from keras.preprocessing import image
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
```

Задание 1. Обучение классификаторов на основе глубоких нейронных сетей на датасете GTSRB

Извлечём изображения для создания тренировочной выборки и отобразим первое изображение:

```
train_path = "Train"
labels = []
data = []
CLASSES = 43
for i in range(CLASSES):
    img_path = os.path.join(train_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)
plt.imshow(data[0])
```

<matplotlib.image.AxesImage at 0x7ce89ea74730>



Воспользуемся ResNet50. Разобьём датасет на тренировочную и тестовую выборки в соотношении 70:30 и поменяем выходные слои модели, для осуществления классификации 43 типов изображений:

```
x_train, x_val, y_train, y_val = train_test_split(data, labels,
test_size=0.3, random_state=1)
img_size = (224,224)
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/resnet50\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5
94765736/94765736 [=====] - 1s 0us/step
```

Обучим изменённую модель с параметрами epochs = 5, batch_size = 64:

```
model.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(x_train, y_train, validation_data =(x_val, y_val),
epochs = 5, batch_size = 64)
```

```
Epoch 1/5
429/429 [=====] - 55s 60ms/step - loss: 1.1352 -
accuracy: 0.7123 - val_loss: 51.0620 - val_accuracy: 0.1789
Epoch 2/5
429/429 [=====] - 22s 51ms/step - loss: 0.2381 -
accuracy: 0.9388 - val_loss: 1.4900 - val_accuracy: 0.8227
Epoch 3/5
429/429 [=====] - 21s 49ms/step - loss: 0.1333 -
accuracy: 0.9656 - val_loss: 0.2912 - val_accuracy: 0.9532
Epoch 4/5
429/429 [=====] - 21s 50ms/step - loss: 0.0894 -
accuracy: 0.9765 - val_loss: 0.2283 - val_accuracy: 0.9424
Epoch 5/5
429/429 [=====] - 23s 53ms/step - loss:
0.0799 - accuracy: 0.9807 - val_loss: 0.0937 - val_accuracy: 0.9759
```

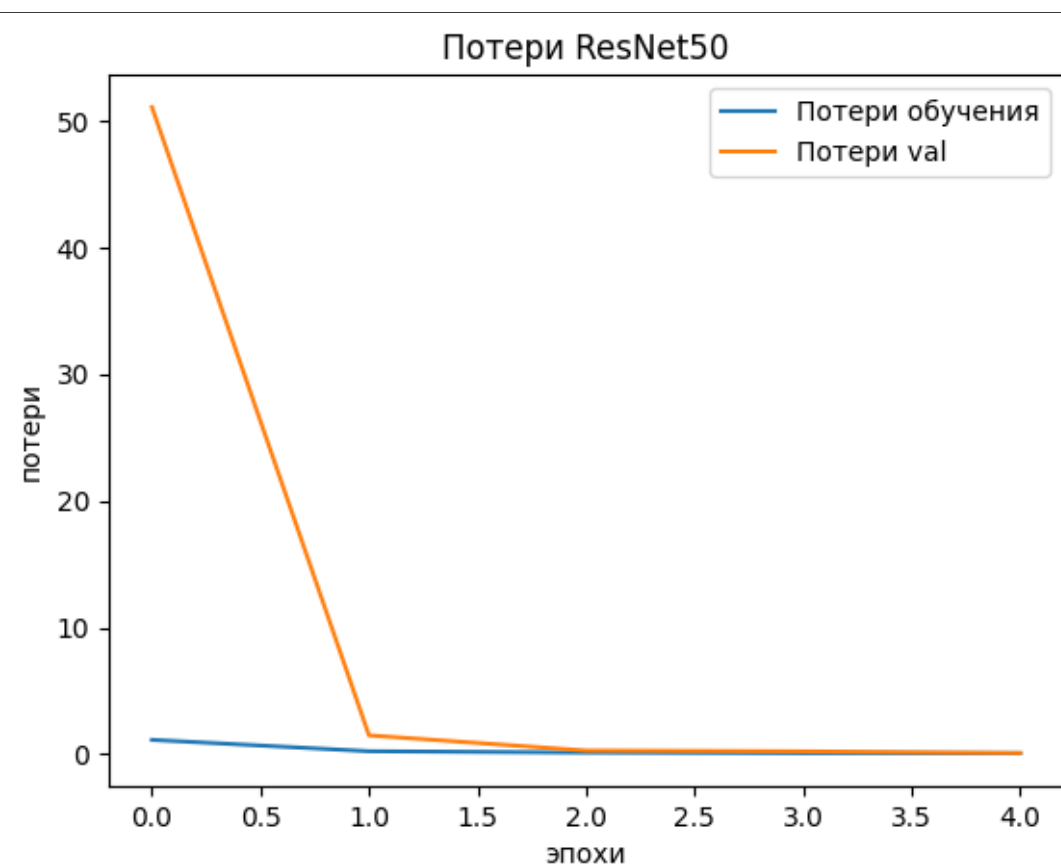
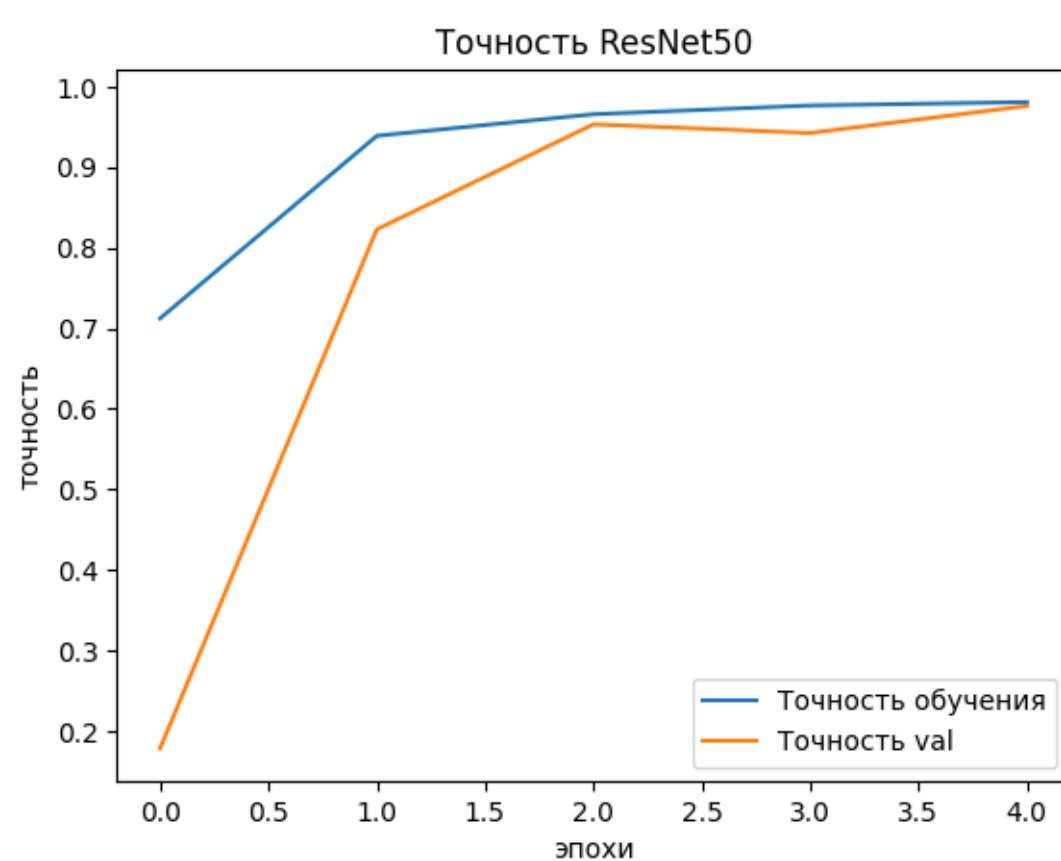
Сохраним модель:

```
save_model(model, 'ResNet50.h5')
with open('history_ResNet50.pkl', 'wb') as file:
    pickle.dump(history.history, file)
!cp ResNet50.h5 drive/MyDrive/ResNet50.h5
```

```
<ipython-input-8-d75c136fedbd>:1: UserWarning: You are saving your model
as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
    save_model(model, 'ResNet50.h5')
```

Построим два графика, которые отражают успешность обучения модели ResNet50 с изменёнными выходными слоями:

```
plt.figure(0)
plt.plot(history.history['accuracy'], label="Точность обучения")
plt.plot(history.history['val_accuracy'], label="Точность val")
plt.title("Точность ResNet50")
plt.xlabel("эпохи")
plt.ylabel("точность")
plt.legend()
plt.figure(1)
plt.plot(history.history['loss'], label="Потери обучения")
plt.plot(history.history['val_loss'], label="Потери val")
plt.title("Потери ResNet50")
plt.xlabel("эпохи")
plt.ylabel("потери")
plt.legend()
plt.show()
```



Скорректируем тестовый набор данных (для определения правильной метки класса будем использовать csv таблицу с обозначением пути картинки и ее класса) и оценим точность классификации модели:

```
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)
loss, accuracy = model.evaluate(data, y_test)
print(f"Потери теста: {loss}")
print(f"Точность теста: {accuracy}")
```

```
395/395 [=====] - 6s 13ms/step - loss: 0.3897 -
accuracy: 0.9202
Потери теста: 0.3896692097187042
Точность теста: 0.9201900362968445
```

Выполним аналогичные действия для VGG16:

```
del model
del history
img_size = (224, 224)
model = Sequential()
model.add(VGG16(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
model.compile(loss = 'categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(x_train, y_train, validation_data = (x_val, y_val),
epochs = 5, batch_size = 64)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5
58889256/58889256 [=====] - 0s 0us/step
Epoch 1/5
429/429 [=====] - 29s 54ms/step - loss: 3.5978 -
accuracy: 0.1138 - val_loss: 2.1290 - val_accuracy: 0.2689
Epoch 2/5
```

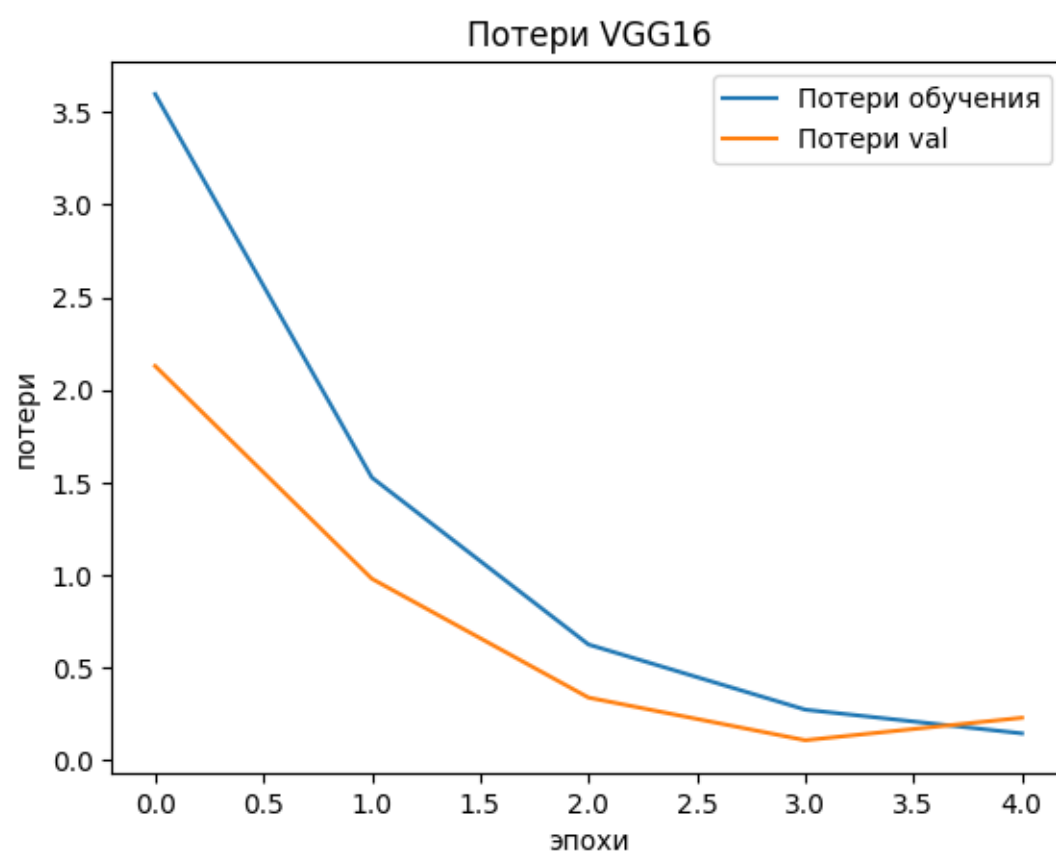
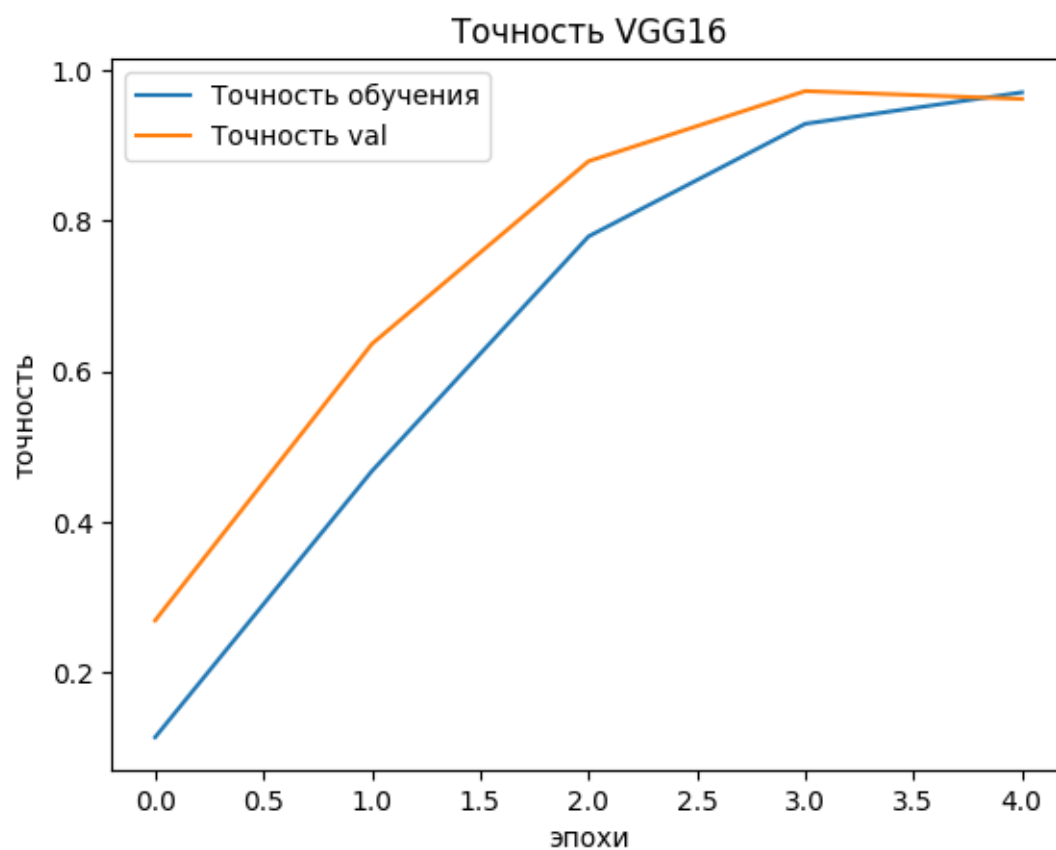


```
429/429 [=====] - 18s 42ms/step - loss: 1.5268 -  
accuracy: 0.4672 - val_loss: 0.9800 - val_accuracy: 0.6361  
Epoch 3/5  
429/429 [=====] - 18s 42ms/step - loss: 0.6247 -  
accuracy: 0.7791 - val_loss: 0.3379 - val_accuracy: 0.8789  
Epoch 4/5  
429/429 [=====] - 17s 41ms/step - loss: 0.2728 -  
accuracy: 0.9287 - val_loss: 0.1077 - val_accuracy: 0.9721  
Epoch 5/5  
429/429 [=====] - 18s 41ms/step - loss:  
0.1446 - accuracy: 0.9704 - val_loss: 0.2290 - val_accuracy: 0.9617
```

```
save_model(model, 'VGG16.h5')  
with open('history_VGG16.pkl', 'wb') as file:  
    pickle.dump(history.history, file)  
!cp ResNet50.h5 drive/MyDrive/ResNet50.h5
```

```
<ipython-input-13-c4b9a345d3f9>:1: UserWarning: You are saving your model  
as an HDF5 file via `model.save()`. This file format is considered legacy.  
We recommend using instead the native Keras format, e.g.  
`model.save('my_model.keras')`.  
    save_model(model, 'VGG16.h5')
```

```
plt.figure(0)  
plt.plot(history.history['accuracy'], label="Точность обучения")  
plt.plot(history.history['val_accuracy'], label="Точность val")  
plt.title("Точность VGG16")  
plt.xlabel ("эпохи")  
plt.ylabel ("точность")  
plt.legend()  
plt.figure (1)  
plt.plot(history.history['loss'], label="Потери обучения")  
plt.plot(history.history['val_loss'], label="Потери val")  
plt.title("Потери VGG16")  
plt.xlabel ("эпохи")  
plt.ylabel ("потери")  
plt. legend()  
plt.show()
```



```

loss, accuracy = model.evaluate(data, y_test)
print(f"Потери теста: {loss}")
print(f"Точность теста: {accuracy}")

```

```

395/395 [=====] - 5s 10ms/step - loss: 0.5007 -
accuracy: 0.9170
Потери теста: 0.5007426738739014
Точность теста: 0.9170229434967041

```

Занесём результаты обучений, валидаций и тестов в сравнительную таблицу 1.

Таблица 1 – Сравнительная таблица

Модель	Обучение	Валидация	Тест
ResNet50	loss: 0.0799 accuracy: 0.9807	val_loss: 0.0937 val_accuracy: 0.9759	Потери теста: 0.3896692097187042 Точность теста: 0.9201900362968445
VGG16	loss: 0.1446 accuracy: 0.9704	val_loss: 0.2290 val_accuracy: 0.9617	Потери теста: 0.5007426738739014 Точность теста: 0.9170229434967041

Задание 2. Применение нецелевой атаки уклонения на основе белого ящика против моделей глубокого обучения

Проведём атаку FGSM на модель ResNet50 (модель атаки будет основываться на обученном классификаторе для внесения шума в изображение):

```
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)))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.10/dist-
packages/keras/src/layers/normalization/batch_normalization.py:883:
_colocate_with (from tensorflow.python.framework.ops) is deprecated and
will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

```
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 = []
adv_accuracises_fgsm = []
true_losses = []
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 потери: {loss}")
    print(f"Adv точность: {accuracy}")
    loss, accuracy = model.evaluate(x_test, y_test)
    true_accuracies.append(accuracy)
    true_losses.append(loss)
    print(f"True потери: {loss}")
    print(f"True точность: {accuracy}")
```

```
Eps: 0.00392156862745098
/usr/local/lib/python3.10/dist-packages
  updates = self.state_updates
Adv потери: 1.5212565279006958
Adv точность: 0.7680000066757202
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.00784313725490196
Adv потери: 2.863587314605713
Adv точность: 0.609000027179718
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.011764705882352941
Adv потери: 3.9492038888931273
Adv точность: 0.49399998784065247
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.01568627450980392
Adv потери: 4.837409213066101
Adv точность: 0.41999998688697815
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.0196078431372549
Adv потери: 5.525903869628906
Adv точность: 0.367000013589859
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.03137254901960784
Adv потери: 6.922792114257812
Adv точность: 0.2540000081062317
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.0392156862745098
Adv потери: 7.473134906768799
Adv точность: 0.2029999941587448
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.0784313725490196
Adv потери: 8.6355673828125
Adv точность: 0.0689999982714653
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.19607843137254902
Adv потери: 8.655828758239746
Adv точность: 0.014999999664723873
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.3137254901960784
Adv потери: 8.135837818145752
Adv точность: 0.006000000052154064
True потери: 0.4191638448536396
True точность: 0.921999990940094
```

```

adv_losses_fgsm = np.array(adv_losses_fgsm)
adv_accuracises_fgsm = np.array(adv_accuracises_fgsm)
np.save("adv_losses_fgsm_rn50", adv_losses_fgsm)
np.save("adv_accuracises_fgsm_rn50", adv_accuracises_fgsm)
!cp adv_losses_fgsm_rn50.npy drive/MyDrive/adv_losses_pgd_rn50.npy
!cp adv_accuracises_fgsm_rn50.npy
drive/MyDrive/adv_accuracises_fgsm_rn50.npy
eps_range = [1/255, 5/255, 10/255, 50/255, 80/255]
pred = np.argmax(model.predict(x_test[0:1]))
plt.figure(0)
plt.title(f"Исходное изображение, предсказанный класс {pred},
действительный класс {np.argmax(y_test[0])}")
plt.imshow(x_test[0])
plt.show()
i = 1
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[0:1]))
    plt.figure(i)
    plt.title(f"Изображение с eps: {eps} , предсказанный класс {pred},
действительный класс {np.argmax(y_test[0])}")
    plt.imshow(x_test_adv[0])
    plt.show()
    i += 1

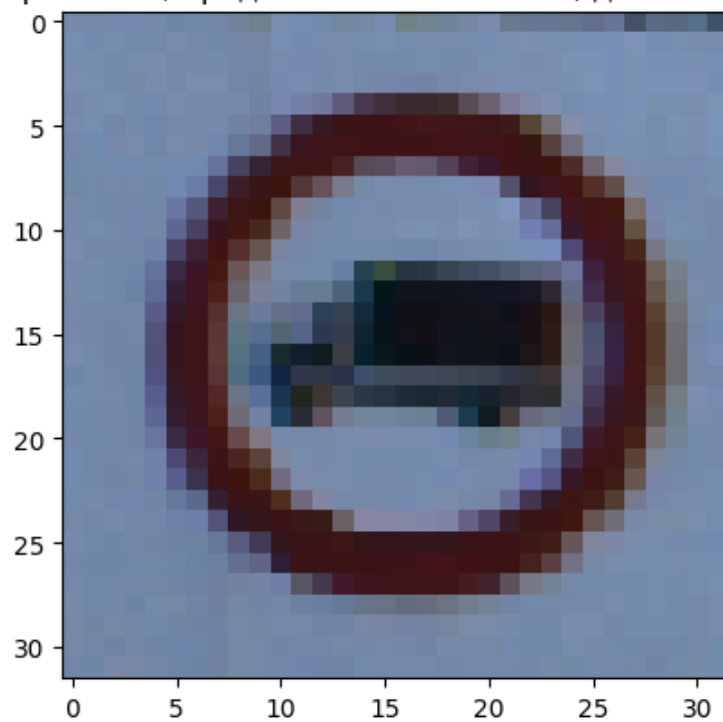
```

```

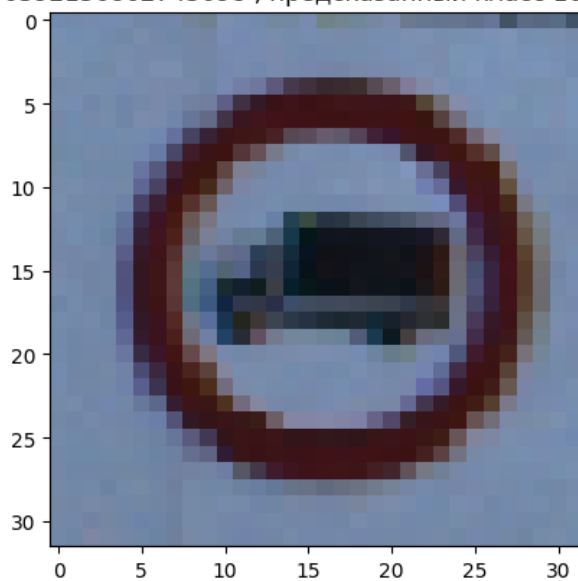
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training_v1.py:2359:
UserWarning: `Model.state_updates` will be removed in a future version. This
property should not be used in TensorFlow 2.0, as `updates` are applied
automatically.
    updates=self.state_updates,

```

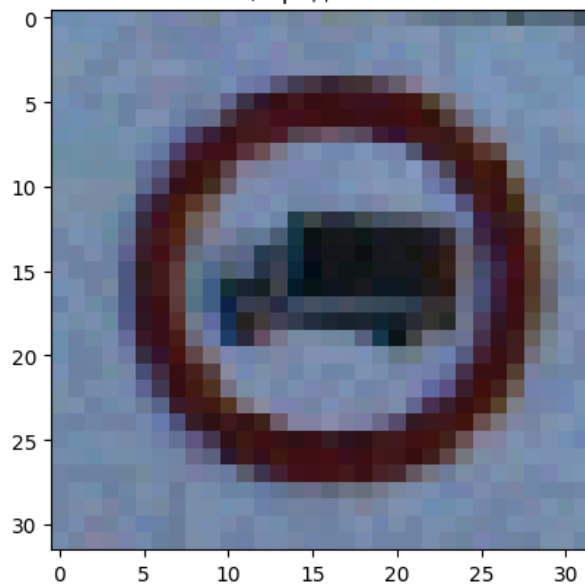
Исходное изображение, предсказанный класс 16, действительный класс 16



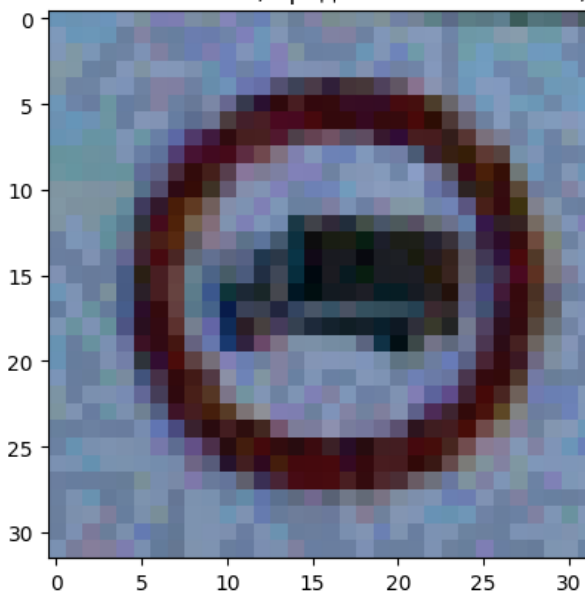
Изображение с eps: 0.00392156862745098 , предсказанный класс 16, действительный класс 16



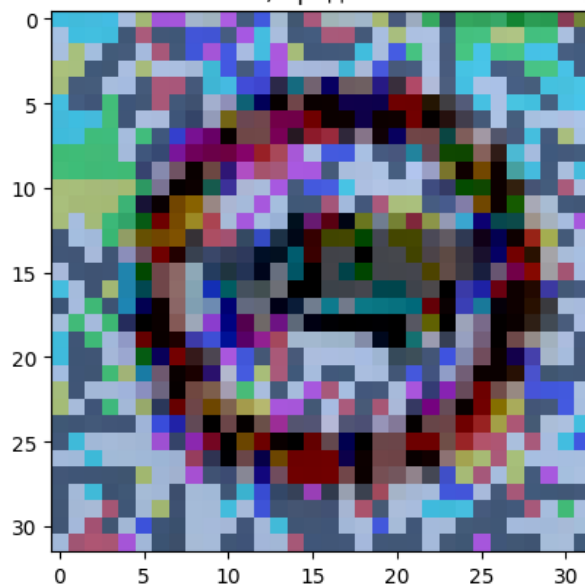
Изображение с eps: 0.0196078431372549 , предсказанный класс 16, действительный класс 16



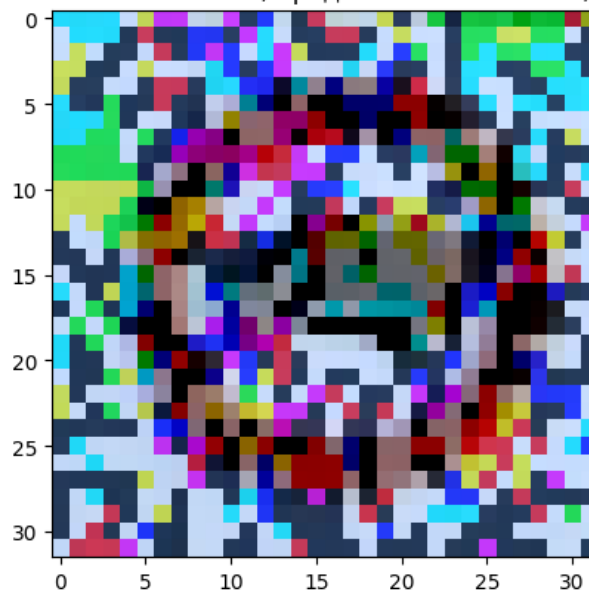
Изображение с eps: 0.0392156862745098 , предсказанный класс 5, действительный класс 16



Изображение с eps: 0.19607843137254902 , предсказанный класс 2, действительный класс 16



Изображение с eps: 0.3137254901960784 , предсказанный класс 2, действительный класс 16



Видно, что при росте eps, шум на картинке сильно увеличивается, и с 5/255 уже становится более заметен. Оптимальным eps будет значение от 5/255 до 10/255.

Теперь реализуем атаку PGD на ResNet50:

```
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)))
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 = []
adv_accuracises_pgd = []
true_losses = []
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 потери: {loss}")
    print(f"Adv точность: {accuracy}")
    loss, accuracy = model.evaluate(x_test, y_test)
```

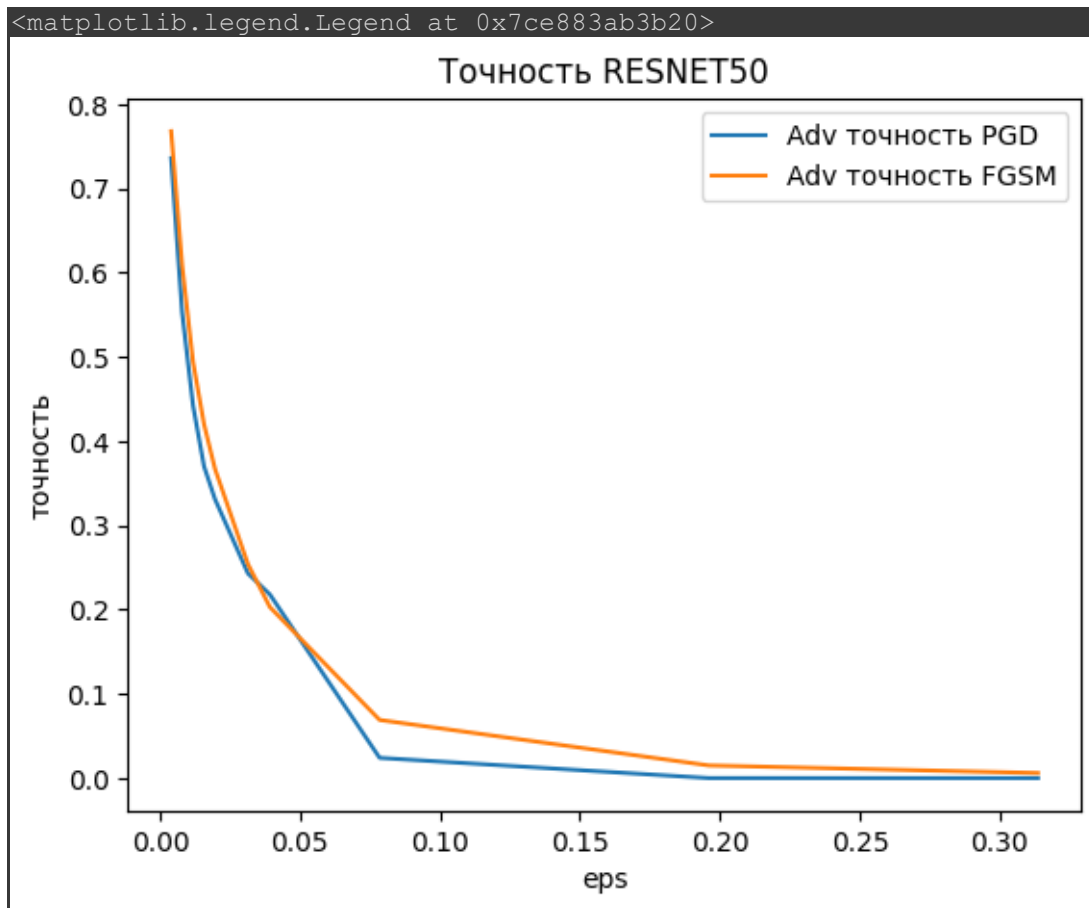
```
true_accuracies.append(accuracy)
true_losses.append(loss)
print(f"True потери: {loss}")
print(f"True точность: {accuracy}")
```

```
Eps: 0.00392156862745098
Adv потери: 1.8539567804336547
Adv точность: 0.7360000014305115
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.00784313725490196
Adv потери: 3.715469088554382
Adv точность: 0.5529999732971191
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.011764705882352941
Adv потери: 5.150159096717834
Adv точность: 0.44200000166893005
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.01568627450980392
Adv потери: 6.4331454133987425
Adv точность: 0.3700000047683716
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.0196078431372549
Adv потери: 7.232316638946533
Adv точность: 0.3310000002384186
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.03137254901960784
Adv потери: 9.29790672302246
Adv точность: 0.24300000071525574
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.0392156862745098
Adv потери: 10.1678634185791
Adv точность: 0.21799999475479126
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.0784313725490196
Adv потери: 25.62070361328125
Adv точность: 0.024000000208616257
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.19607843137254902
Adv потери: 45.304283142089844
Adv точность: 0.0
True потери: 0.4191638448536396
True точность: 0.921999990940094
Eps: 0.3137254901960784
Adv потери: 51.091141235351564
Adv точность: 0.0
True потери: 0.4191638448536396
True точность: 0.921999990940094
```

```

adv_losses_pgd = np.array(adv_losses_pgd)
adv_accuracises_pgd = np.array(adv_accuracises_pgd)
np.save("adv_losses_pgd_rn50", adv_losses_pgd)
np.save("adv_accuracises_pgd_rn50", adv_accuracises_pgd)
!cp adv_losses_pgd_rn50.npy drive/MyDrive/adv_losses_pgd_rn50.npy
!cp adv_accuracises_pgd_rn50.npy
drive/MyDrive/adv_accuracises_pgd_rn50.npy
adv_accuracises_fgsm = np.load("adv_accuracises_fgsm_rn50.npy")
adv_accuracises_pgd = np.load("adv_accuracises_pgd_rn50.npy")
plt.figure(0)
plt.plot(eps_range, adv_accuracises_pgd, label="Adv точность PGD")
plt.plot(eps_range, adv_accuracises_fgsm, label="Adv точность FGSM")
plt.title("Точность RESNET50")
plt.xlabel("eps")
plt.ylabel("Точность")
plt.legend()

```



Из графиков видно, что методы имеют почти схожую эффективность, но метод PGD слегка больше снижает точность. Реализуем атаку FGSM на 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)))
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 = []
adv_accuracises_fgsm = []
true_losses = []
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 потери: {loss}")
    print(f"Adv точность: {accuracy}")
    loss, accuracy = model.evaluate(x_test, y_test)
    true_accuracies.append(accuracy)
    true_losses.append(loss)
    print(f"True потери: {loss}")
    print(f"True точность: {accuracy}")
```

Eps: 0.00392156862745098
Adv потери: 1.5479141578674316
Adv точность: 0.7950000166893005
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.00784313725490196
Adv потери: 2.7056674880981446
Adv точность: 0.6930000185966492
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.011764705882352941
Adv потери: 3.715561466217041
Adv точность: 0.6010000109672546
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.01568627450980392
Adv потери: 4.49190878868103
Adv точность: 0.5139999985694885
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.0196078431372549
Adv потери: 4.998949356079102
Adv точность: 0.46000000834465027
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.03137254901960784
Adv потери: 6.019459995269775
Adv точность: 0.32100000977516174
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.0392156862745098
Adv потери: 6.38631196975708
Adv точность: 0.24899999797344208
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.0784313725490196
Adv потери: 6.95974658203125
Adv точность: 0.10999999940395355
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.19607843137254902
Adv потери: 6.674490501403809
Adv точность: 0.04600000008940697
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.3137254901960784
Adv потери: 6.3386449584960936
Adv точность: 0.04500000178813934
True потери: 0.4283084148168564
True точность: 0.9229999780654907

```

eps_range = [1/255, 5/255, 10/255, 50/255, 80/255]
pred = np.argmax(model.predict(x_test[2:3]) )
plt.figure(0)
plt.title(f"Исходное изображение, предсказанный класс {pred},  
действительный класс {np.argmax(y_test[2])}")
plt.imshow(x_test[2])
plt.show()
i = 1
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[2:3]))
    plt.figure(i)
    plt.title(f"Изображение с eps: {eps} , предсказанный класс {pred},  
действительный класс {np.argmax(y_test[2])}")
    plt.imshow(x_test_adv[2])
    plt.show()
    i += 1

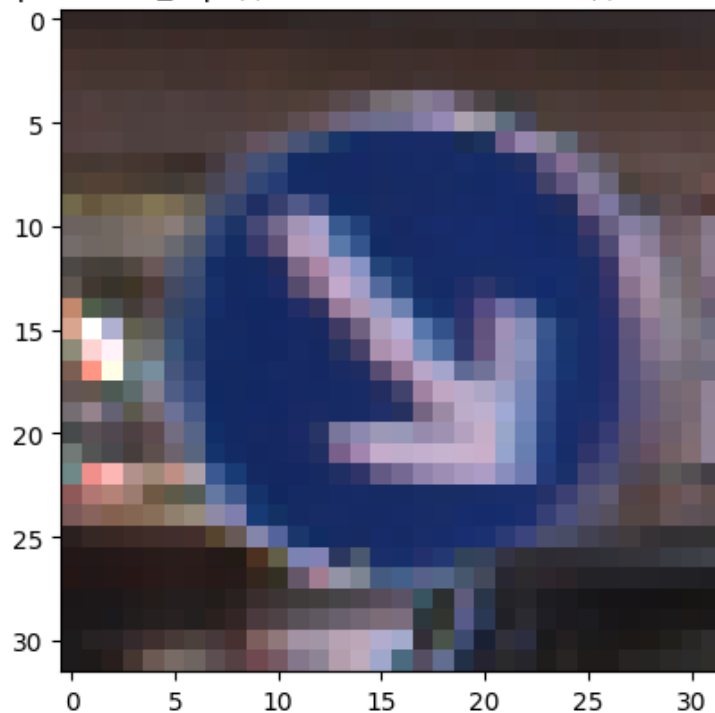
```

```

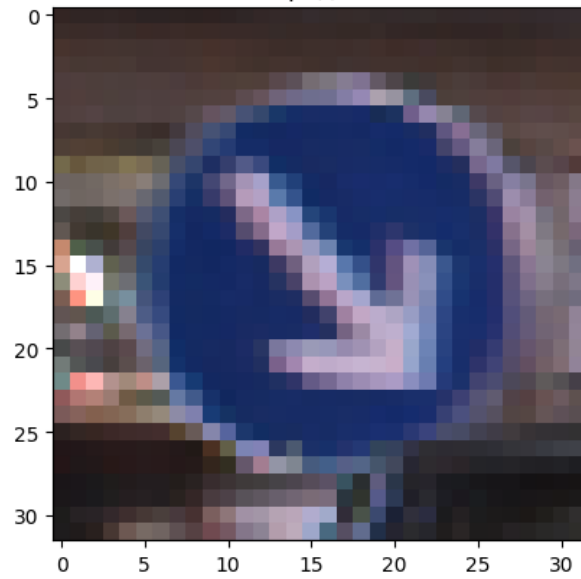
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151:
UserWarning: Glyph 65292 (\N{FULLWIDTH COMMA}) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)

```

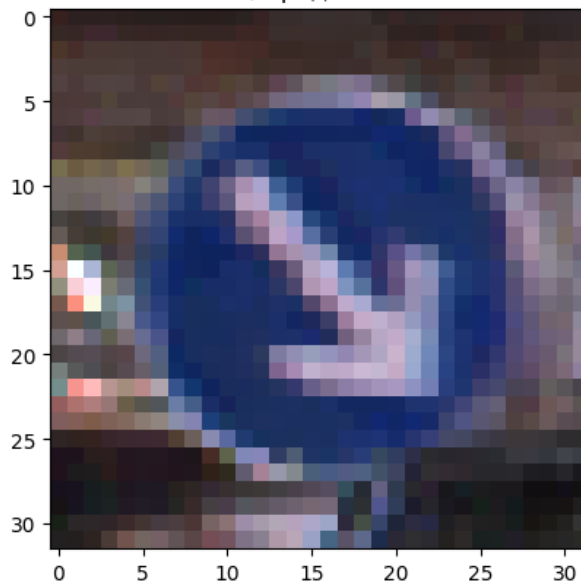
Исходное изображение, предсказанный класс 38, действительный класс 38



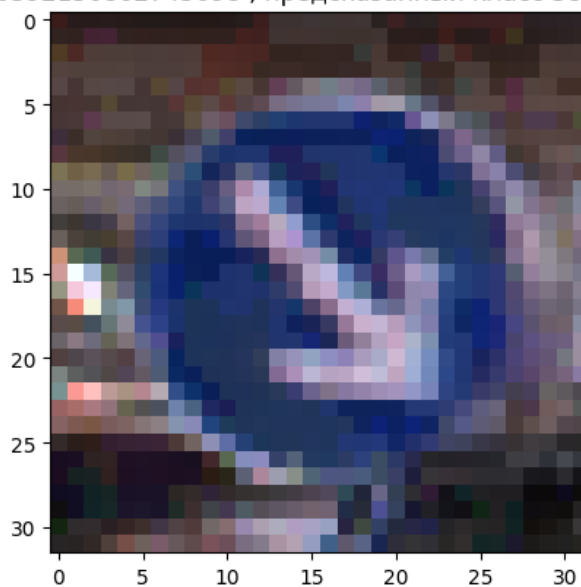
Изображение с eps: 0.00392156862745098 , предсказанный класс 38, действительный класс 38



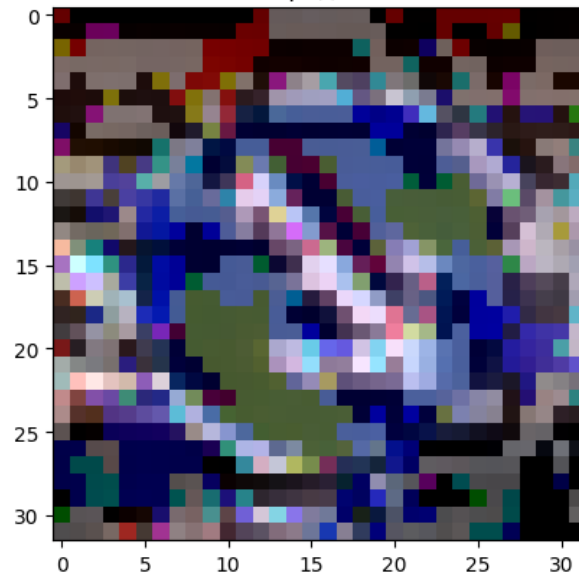
Изображение с eps: 0.0196078431372549 , предсказанный класс 38, действительный класс 38



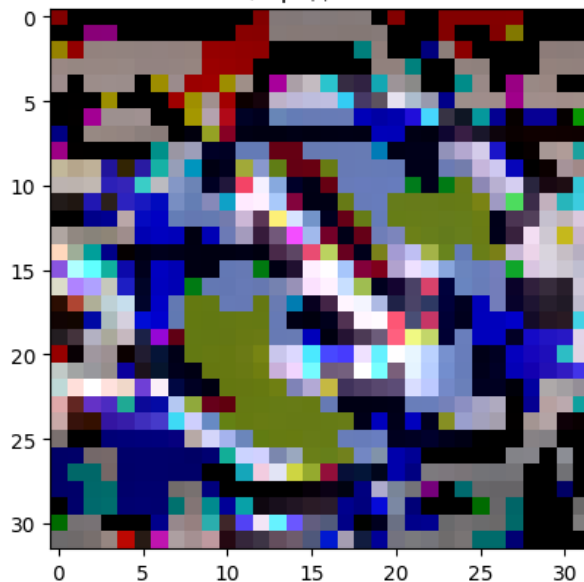
Изображение с eps: 0.0392156862745098 , предсказанный класс 38, действительный класс 38



Изображение с eps: 0.19607843137254902 , предсказанный класс 20, действительный класс 38



Изображение с eps: 0.3137254901960784 , предсказанный класс 25, действительный класс 38



```
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)
!cp adv_losses_fgsm_vgg16.npy drive/MyDrive/adv_losses_pgd_vgg16.npy
!cp adv_accuracises_fgsm_vgg16.npy
drive/MyDrive/adv_accuracises fgsm vgg16.npy
```

Выполним атаку PGD на 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)))
```



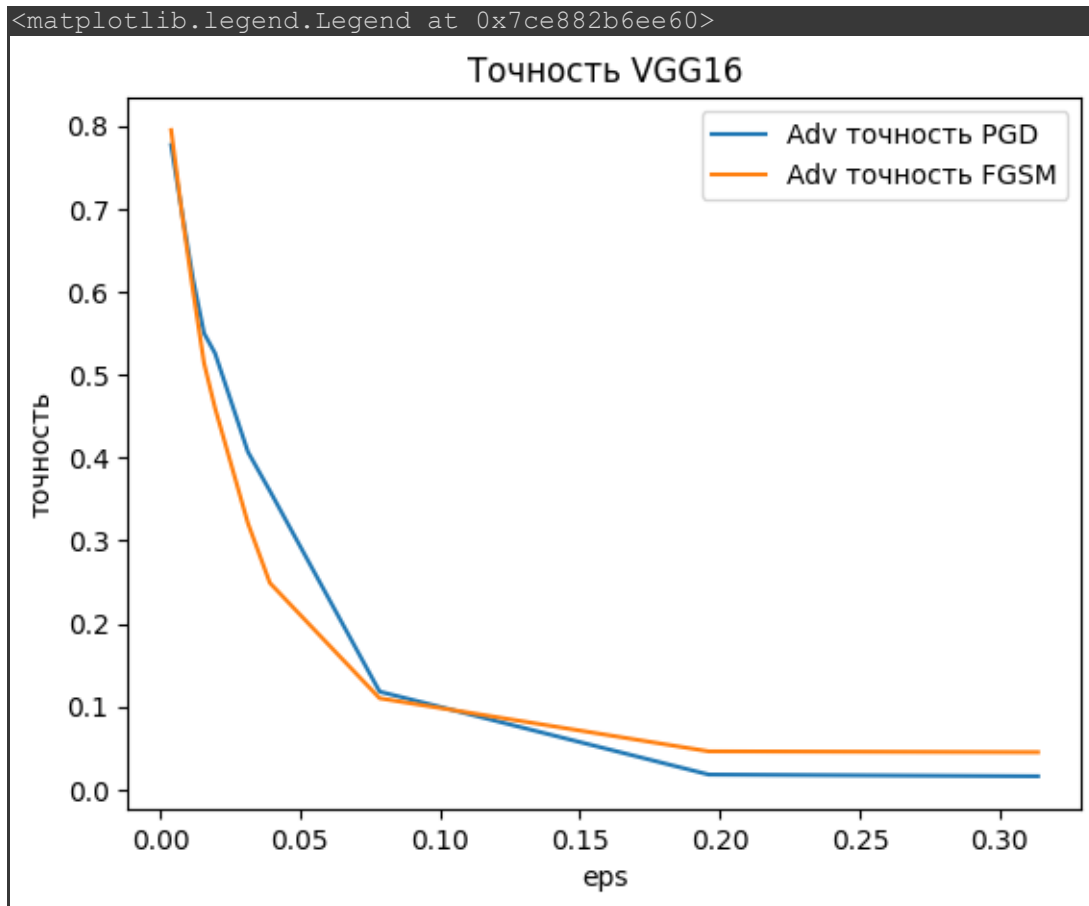
```
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3,
max_iter=4, verbose=False)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255,
50/255, 80/255]
true_accuracies = []
adv_accuracises_pgd = []
true_losses = []
adv_losses_pgd = []
for eps in eps_range:
    attack_pgd.set_params(**{'eps': 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 потери: {loss}")
    print(f"Adv точность: {accuracy}")
    loss, accuracy = model.evaluate(x_test, y_test)
    true_accuracies.append(accuracy)
    true_losses.append(loss)
    print(f"True потери: {loss}")
    print(f"True точность: {accuracy}")
```

Eps: 0.00392156862745098
Adv потери: 1.766727325439453
Adv точность: 0.7770000100135803
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.00784313725490196
Adv потери: 3.2222650079727173
Adv точность: 0.6949999928474426
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.011764705882352941
Adv потери: 4.347150318145752
Adv точность: 0.6150000095367432
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.01568627450980392
Adv потери: 4.861804706573486
Adv точность: 0.550000011920929
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.0196078431372549
Adv потери: 5.095220914840699
Adv точность: 0.5260000228881836
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.03137254901960784
Adv потери: 6.824719165802002
Adv точность: 0.40700000524520874
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.0392156862745098
Adv потери: 7.240382934570312
Adv точность: 0.36000001430511475
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.0784313725490196
Adv потери: 19.283891403198243
Adv точность: 0.11800000071525574
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.19607843137254902
Adv потери: 49.6300478515625
Adv точность: 0.017999999225139618
True потери: 0.4283084148168564
True точность: 0.9229999780654907
Eps: 0.3137254901960784
Adv потери: 60.71853527832031
Adv точность: 0.01600000075995922
True потери: 0.4283084148168564
True точность: 0.9229999780654907

```

adv_losses_pgd = np.array(adv_losses_pgd)
adv_accuracises_pgd = np.array(adv_accuracises_pgd)
np.save("adv_losses_pgd_vgg16", adv_losses_pgd)
np.save("adv_accuracises_pgd_vgg16", adv_accuracises_pgd)
!cp adv_losses_pgd_vgg16.npy drive/MyDrive/adv_losses_pgd_vgg16.npy
!cp adv_accuracises_pgd_vgg16.npy
drive/MyDrive/adv_accuracises_pgd_vgg16.npy
adv_accuracises_fgsm = np.load("adv_accuracises_fgsm_vgg16.npy")
adv_accuracises_pgd = np.load("adv_accuracises_pgd_vgg16.npy")
plt.figure(0)
plt.plot(eps_range, adv_accuracises_pgd, label="Adv точность PGD")
plt.plot(eps_range, adv_accuracises_fgsm, label="Adv точность FGSM")
plt.title("Точность VGG16")
plt.xlabel("eps")
plt.ylabel("Точность")
plt.legend()

```



Из графиков видно, что методы имеют почти схожую эффективность, но метод PGD слегка больше снижает точность. Заполним сравнительную таблицу 2.

Таблица 2 – Зависимость точности классификации от параметра искажений ϵ

Модель	Исходные изображения	Adversarial images $\epsilon=1/255$	Adversarial images $\epsilon=5/255$	Adversarial images $\epsilon=10/255$
ResNet50 – FGSM	91%	74%	33%	17%
ResNet50 – PGD	91%	71%	30%	23%
VGG16 – FGSM	89%	79%	44%	21%
VGG16 – PGD	89%	77%	48%	32%

Задание 3. Применение целевой атаки уклонения методом белого ящика против моделей глубокого обучения

Выполним целевую атаку FGSM на ResNet50:

```
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)
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))
xtest = data
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test),
np.max(xtest)))
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(xtest, y_test)
    print(f"True Loss: {loss}")
    print(f"True Accuracy: {accuracy}")
```

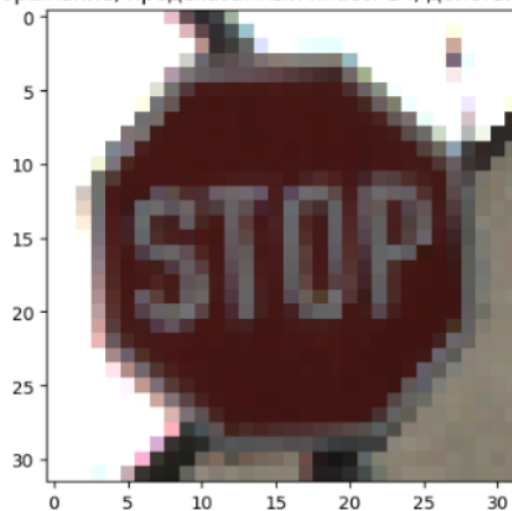
```
Eps: 0.00392156862745098
/usr/local/lib/python3.10/dist-packa
  updates = self.state_updates
Adv Loss: 0.902568750994073
Adv Accuracy: 0.8740741014480591
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.00784313725490196
Adv Loss: 1.5175819600069964
Adv Accuracy: 0.7814815044403076
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.011764705882352941
Adv Loss: 2.34287749837946
Adv Accuracy: 0.6777777671813965
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.01568627450980392
Adv Loss: 3.408220080976133
Adv Accuracy: 0.5148147940635681
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0196078431372549
Adv Loss: 4.363397495834915
Adv Accuracy: 0.42592594027519226
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.03137254901960784
Adv Loss: 6.640581943370678
Adv Accuracy: 0.13703703880310059
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0392156862745098
Adv Loss: 7.327747023547137
Adv Accuracy: 0.0555555559694767
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0784313725490196
Adv Loss: 7.1469013320075145
Adv Accuracy: 0.003703703638166189
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.19607843137254902
Adv Loss: 5.450976392957899
Adv Accuracy: 0.0
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.3137254901960784
Adv Loss: 5.5924287266201445
Adv Accuracy: 0.0
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
```

```

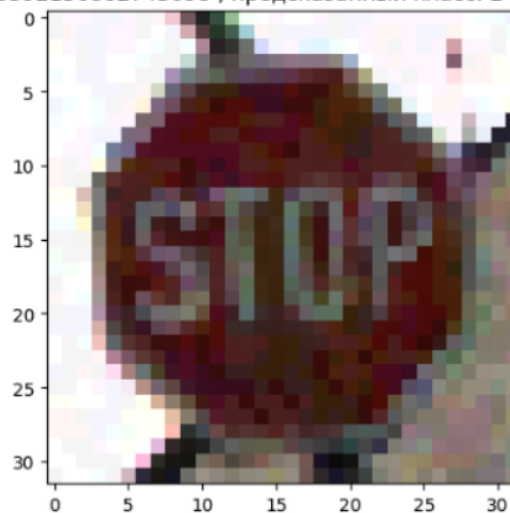
eps = 10/255
attack_fgsm.set_params(**{'eps': eps})
x_test_adv = attack_fgsm.generate(x_test, t_classes)
range = [0, 3, 5, 6, 8]
i = 0
for index in range:
    plt.figure(i)
    pred = np.argmax(model.predict(x_test[index:index+1]))
    plt.title(f"Исходное изображение предсказанный класс {pred},  
действительный класс {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"Изображение с eps {eps} предсказанный класс {pred},  
действительный класс {np.argmax(y_test[index])}")
    plt.imshow(x_test_adv[index])
    plt.show()

```

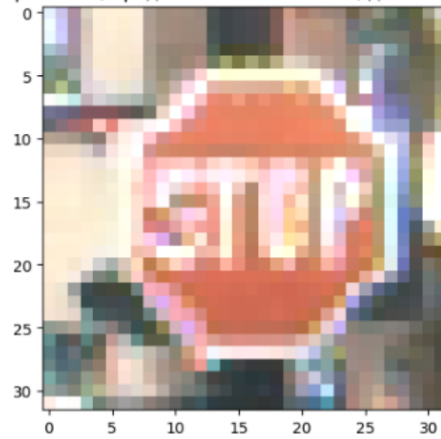
Исходное изображение, предсказанный класс: 14, действительный класс 14



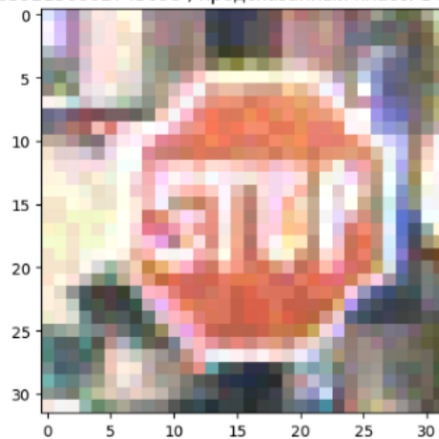
Изображение с eps: 0.0392156862745098 , предсказанный класс: 24, действительный класс 14



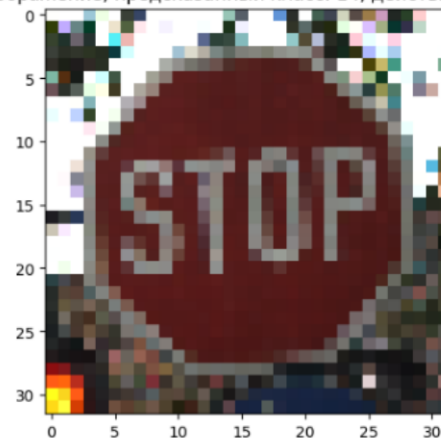
Исходное изображение, предсказанный класс: 11, действительный класс 14



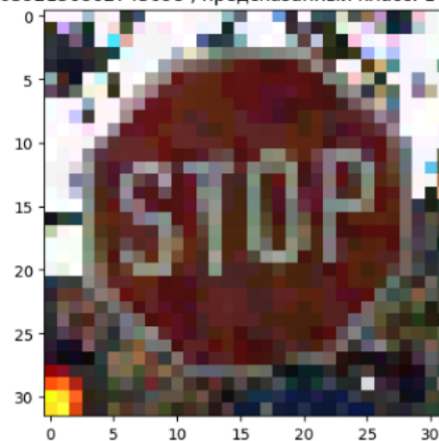
Изображение с eps: 0.0392156862745098 , предсказанный класс: 14, действительный класс 14



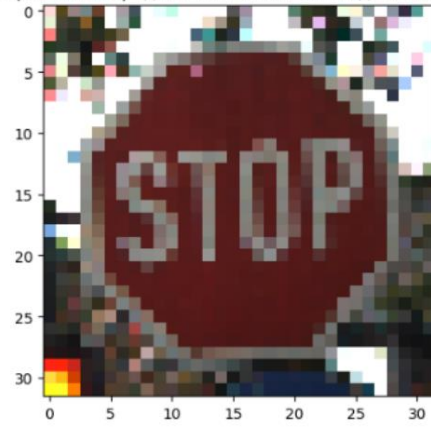
Исходное изображение, предсказанный класс: 14, действительный класс 14



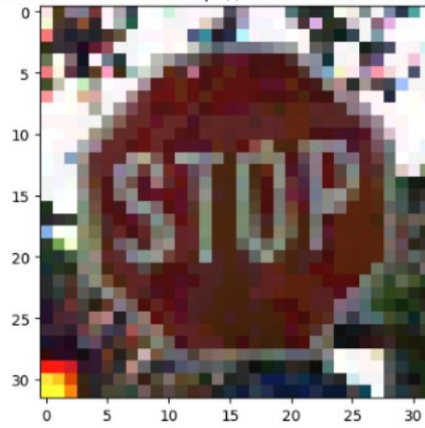
Изображение с eps: 0.0392156862745098 , предсказанный класс: 14, действительный класс 14



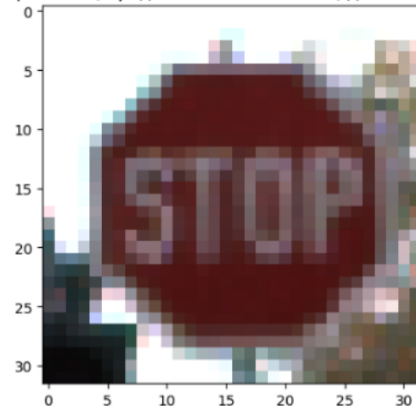
Исходное изображение, предсказанный класс: 14, действительный класс 14



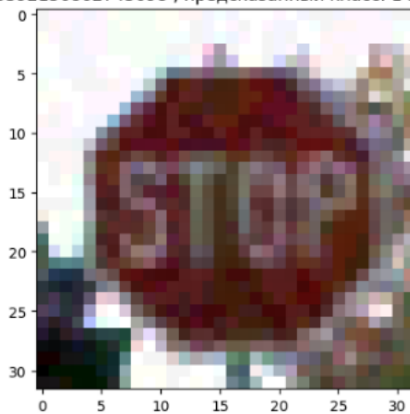
Изображение с eps: 0.0392156862745098 , предсказанный класс: 14, действительный класс 14



Исходное изображение, предсказанный класс: 14, действительный класс 14



Изображение с eps: 0.0392156862745098 , предсказанный класс: 14, действительный класс 14



Выполним целевую атаку PGD на ResNet50:

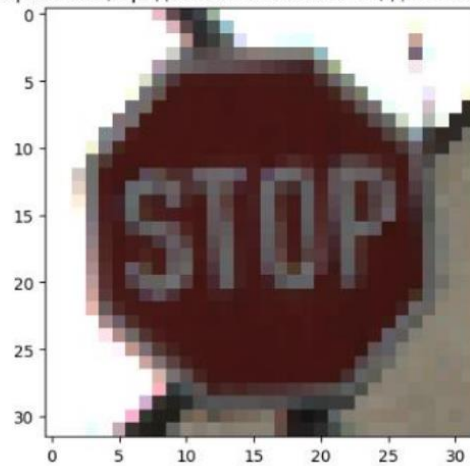
```
model=load_model('ResNet50.h5')
classifier = KerasClassifier(model=model, clip_values=(np.min(x_test),
np.max(x_test)))
attack_pgd = ProjectedGradientDescent(estimator=classifier, eps=0.3,
max_iter=4, verbose=False, targeted=True)
eps_range = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255,
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: 0.24172166348607452
Adv Accuracy: 0.9629629850387573
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.00784313725490196
Adv Loss: 0.4087429267388803
Adv Accuracy: 0.9296296238899231
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.011764705882352941
Adv Loss: 0.8597279482417637
Adv Accuracy: 0.8666666746139526
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.01568627450980392
Adv Loss: 1.3003518992000156
Adv Accuracy: 0.7888888716697693
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0196078431372549
Adv Loss: 1.4792703549067179
Adv Accuracy: 0.7740740776062012
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.03137254901960784
Adv Loss: 2.116669112664682
Adv Accuracy: 0.6666666865348816
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0392156862745098
Adv Loss: 2.2313812414805096
Adv Accuracy: 0.6555555462837219
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.0784313725490196
Adv Loss: 6.5533073213365345
Adv Accuracy: 0.28148147463798523
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.19607843137254902
Adv Loss: 10.835577074686686
Adv Accuracy: 0.029629629105329514
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936
Eps: 0.3137254901960784
Adv Loss: 11.499047081558793
Adv Accuracy: 0.011111111380159855
True Loss: 0.04066114811813114
True Accuracy: 0.9925925731658936

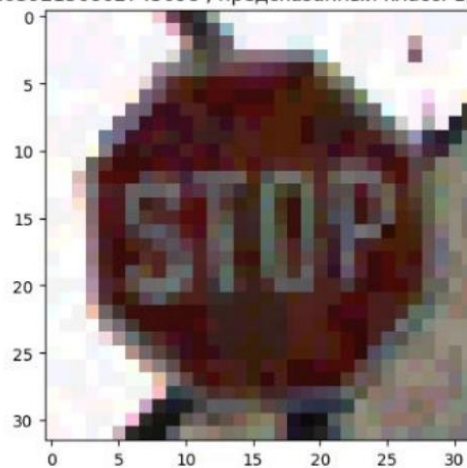
```
eps = 10/255
attack_pgd.set_params(**{'eps': eps})
x_test_adv = attack_pgd.generate(x_test, t_classes)
range = [0, 3, 5, 6, 8]
i = 0
for index in range:
    plt.figure(i)
    pred = np.argmax(model.predict(x_test[index:index+1]))
    plt.title(f"Исходное изображение, предсказанный класс: {pred},
действительный класс {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"Изображение с eps: {eps} , предсказанный класс: {pred},  
действительный класс {np.argmax(y_test[index])}")
plt.imshow(x_test_adv[index])
plt.show
```

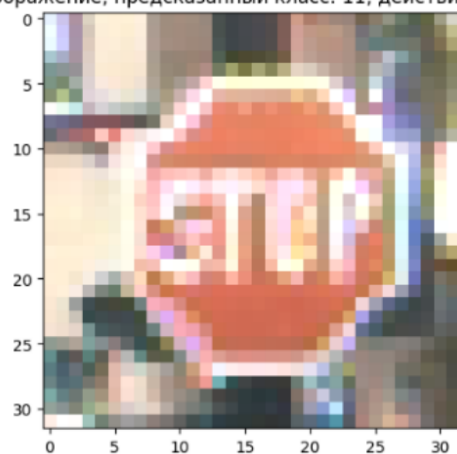
Исходное изображение, предсказанный класс: 14, действительный класс 14



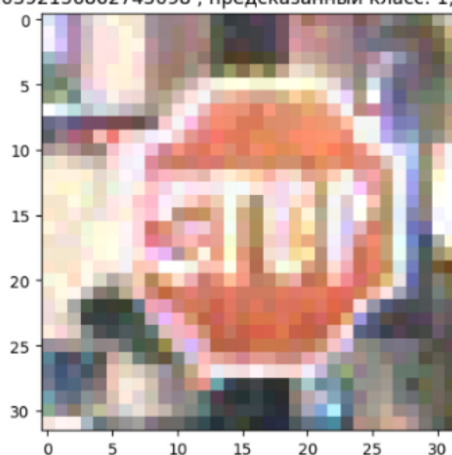
Изображение с eps: 0.0392156862745098 , предсказанный класс: 1, действительный класс 14



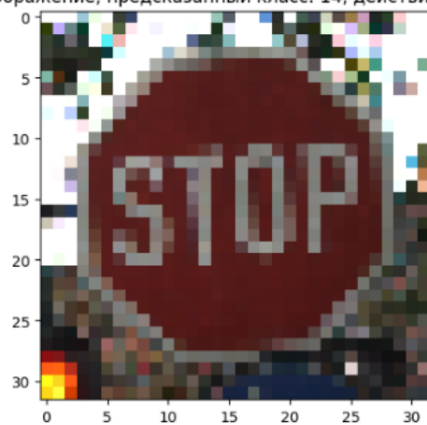
Исходное изображение, предсказанный класс: 11, действительный класс 14



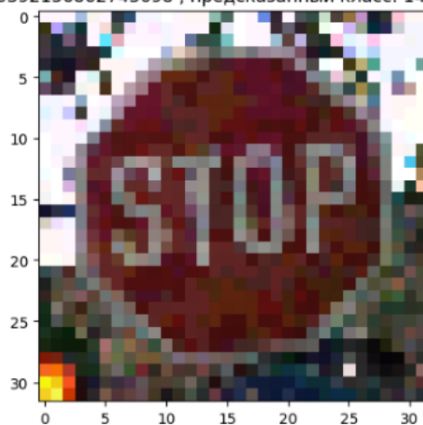
Изображение с eps: 0.0392156862745098 , предсказанный класс: 1, действительный класс 14



Исходное изображение, предсказанный класс: 14, действительный класс 14



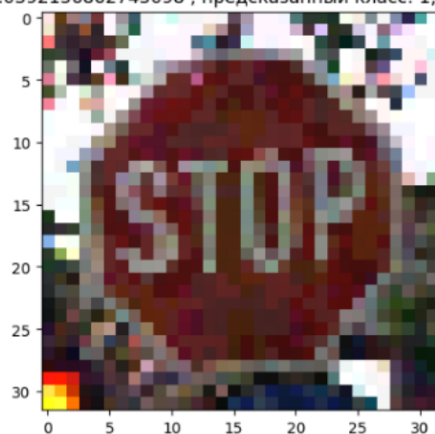
Изображение с eps: 0.0392156862745098 , предсказанный класс: 14, действительный класс 14



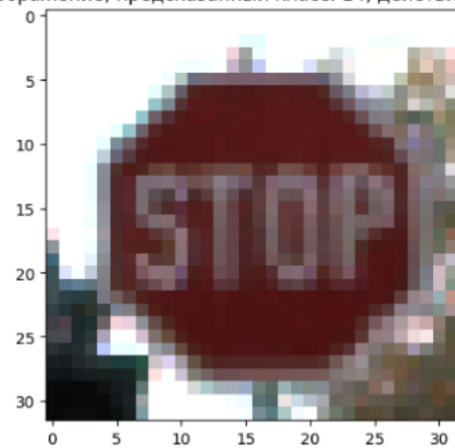
Исходное изображение, предсказанный класс: 14, действительный класс 14



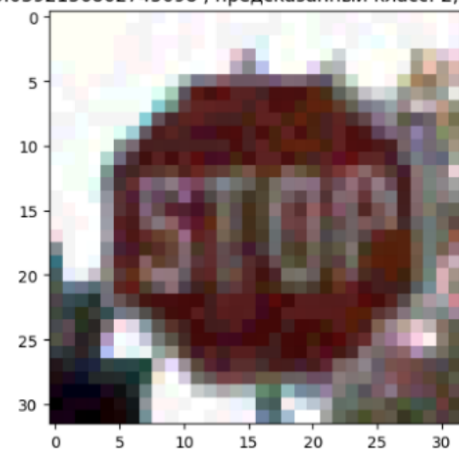
Изображение с eps: 0.0392156862745098 , предсказанный класс: 1, действительный класс 14



Исходное изображение, предсказанный класс: 14, действительный класс 14



Изображение с eps: 0.0392156862745098 , предсказанный класс: 2, действительный класс 14



Заполним таблицу 3 и 4, в которой представим точность целевых атак PGD и FGSM на знак стоп (атака заключается в смене класса на ограничение скорости в 30 км/ч).

Таблица 3 – Точность целевых атак PGD

Искажение	PGD attack – Stop sign images	PGD attack – Speed Limit 30 sign images
$\epsilon=1/255$	97%	99%
$\epsilon=3/255$	91%	99%
$\epsilon=5/255$	90%	99%
$\epsilon=10/255$	71%	99%

Таблица 4 – Точность целевых атак FGSM

Искажение	FGSM attack – Stop sign images	FGSM attack – Speed Limit 30 sign images
$\epsilon=1/255$	99%	99%
$\epsilon=3/255$	80%	99%
$\epsilon=5/255$	73%	99%
$\epsilon=10/255$	26%	99%

По результатам видно метод PGD значительно лучше подходит для целевой атаки, чем метод FGSM.