МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

РТУ МИРЭА
Институт комплексной безопасности и специального приборостроения

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ
по дисциплине
«Анализ защищенности систем искусственного интеллекта»

«Лабораторная работа 2»

Выполнил студент группы ББМО-01-22
Лысак Я.Д.

Проверил
Спирин Андрей Андреевич

«15» ноября 2023 г.

Москва, 2023 г.

Цель работы

Выполнить задания по реализации атак против моделей на основе глубокого обучения, а также получить навыки построения и реализации атак белого и черного ящика на модели машинного обучения.

Задание

1. реализовать атаки уклонения на основе белого ящика против классификационных моделей на основе глубокого обучения;
2. получить практические навыки переноса атак уклонения на основе черного ящика против моделей машинного обучения.

Ход выполнения работы

В качестве набора данных для данной работы выступает GTSRB (German Traffic Sign Recognition Benchmark). Набор состоит из 43 классов и более 50000 записей различных придорожных знаков. Размер изображений составляет 32 на 32 пикселя.

Данный набор данных можно получить по следующей ссылке: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

Местом проведения лабораторной работы была выбрана среда Google Collab из-за простоты подключения и доступности графического ускорителя для более быстрого расчет результатов.

Загрузка и передача файлов для обучения проводится через подключение Google диска с выполнением следующих команд (на фрагмент кода 1):

Фрагмент кода – 1

```
!cp drive/MyDrive/archive.zip .
!unzip archive.zip -d data
```

После выполнения фрагмент кода 1 в директории проекта появилась новая папка, содержащая данные для обучения, валидации и тестирования. Проводится настройка формата взводимых данных, а также загрузка данных в программу. Полная загрузка данных продемонстрирована во фрагменте кода 2 и 3.

Фрагмент кода – 2

```
data_dir = './data'
train_path = './data/Train'
test_path = './data/Test'
image_height = 35
image_width = 35
channels = 3
image_shape = (image_height, image_width, channels)
image_size = (image_height, image_width)
batch_size = 32
number_of_categories = len(os.listdir(train_path))
```

Следующий фрагмент кода демонстрирует функцию загрузки данных по категориям. Таким образом в коде программы получен набор данных для обучения модели. Используется 2 подхода к загрузке данных: с помощью написанной функции load_data() для ручной разбивки данных для дальнейшей обработки и tf.keras.preprocessing.image_dataset_from_directory() для формирования генератора Keras.

Фрагмент кода – 3

```
data = tf.keras.preprocessing.image_dataset_from_directory(train_path)

def load_data(data_dir):
    images = list()
    labels = list()
    for category in range(number_of_categories):
        categories = os.path.join(data_dir, str(category))
        for img in os.listdir(categories):
            img = load_img(os.path.join(categories, img), target_size=image_size)
            image = img_to_array(img)
            images.append(image)
            labels.append(category)

    return images, labels
```

Затем создается генератор Keras для предобработки и итерации входных данных модели (фрагмент кода 4).

Фрагмент кода – 4

```
generator = ImageDataGenerator(width_shift_range=0.2,
                               height_shift_range=0.2,
                               rescale=1. / 255,
                               rotation_range=40,
                               shear_range=0.2,
                               zoom_range=0.2,
                               horizontal_flip=True,
                               fill_mode='nearest',
                               validation_split=0.2)
```

Применение генератора позволяет модифицировать и расширить набор данных для достижения лучшего результата обучения.

После получения генератор и загрузки в него параметров входных данных производится загрузка уже готовых моделей ResNet50 и VGG16 из пакета Keras (фрагмент кода 5).

Фрагмент кода – 5

```
pretrained_model_resnet = ResNet50(weights='imagenet',
                                   include_top=False,
                                   input_shape=image_shape,
                                   pooling='max')

pretrained_model_vgg = VGG16(weights='imagenet',
                             include_top=False,
                             input_shape=image_shape,
                             pooling='max')
```

Затем проводится дополнительная обработка входных слоев, добавляются дополнительные слои и функция активации. После конфигурации модель компилируется. Целевая метрика для обучения – accuracy. Фрагмент кода для обучения модель ResNet – 6, также на рисунке 1 можно увидеть процесс обучения.

Фрагмент кода – 6

```
generated_resnet = fit_generator(train_data,
                                 validation_data=validation_data,
                                 steps_per_epoch=train_data.n // train_data.batch_size,
                                 validation_steps=validation_data.n // validation_data.batch_size,
                                 epochs=20,
                                 verbose=1)
```

На рисунке 1 также изображен процесс обучения модели ResNet с валидацией на соответствующих выборках.

Рисунок – 1

```
generated_resnet = model_resnet.fit_generator(train_data,
                                              validation_data=validation_data,
                                              steps_per_epoch=train_data.n // train_data.batch_size,
                                              validation_steps=validation_data.n // validation_data.batch_size,
                                              epochs=20,
                                              verbose=1)
```

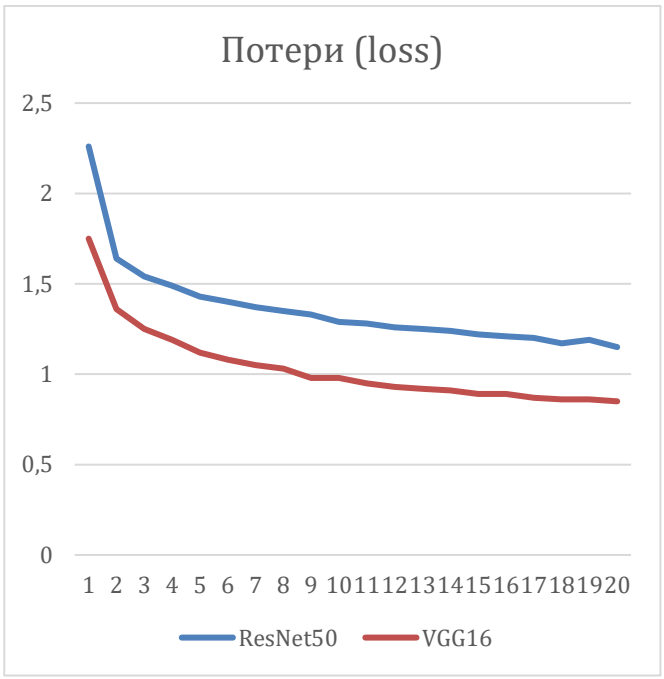
```
Epoch 1/20
980/980 [==============================] - 557s 568ms/step - batch: 489.5000 - size: 31.9755 - loss: 1.7518 - accuracy: 0.4601 - val_loss: 2.3023 - val_accuracy: 0.3889
Epoch 2/20
980/980 [==============================] - 558s 569ms/step - batch: 489.5000 - size: 31.9755 - loss: 1.3673 - accuracy: 0.5555 - val_loss: 2.3456 - val_accuracy: 0.3929
Epoch 3/20
980/980 [==============================] - 556s 567ms/step - batch: 489.5000 - size: 31.9755 - loss: 1.2584 - accuracy: 0.5876 - val_loss: 2.3775 - val_accuracy: 0.4066
Epoch 4/20
980/980 [==============================] - 555s 566ms/step - batch: 489.5000 - size: 31.9755 - loss: 1.1956 - accuracy: 0.6084 - val_loss: 2.4034 - val_accuracy: 0.4187
Epoch 5/20
980/980 [==============================] - 557s 569ms/step - batch: 489.5000 - size: 31.9755 - loss: 1.1251 - accuracy: 0.6283 - val_loss: 2.4497 - val_accuracy: 0.4181
Epoch 6/20
980/980 [==============================] - 558s 570ms/step - batch: 489.5000 - size: 31.9755 - loss: 1.0806 - accuracy: 0.6429 - val_loss: 2.4944 - val_accuracy: 0.4239
Epoch 7/20
980/980 [==============================] - 563s 575ms/step - batch: 489.5000 - size: 31.9755 - loss: 1.0506 - accuracy: 0.6494 - val_loss: 2.4946 - val_accuracy: 0.4244
Epoch 8/20
980/980 [==============================] - 560s 571ms/step - batch: 489.5000 - size: 31.9755 - loss: 1.0324 - accuracy: 0.6565 - val_loss: 2.6029 - val_accuracy: 0.4138
Epoch 9/20
980/980 [==============================] - 554s 565ms/step - batch: 489.5000 - size: 31.9755 - loss: 0.9863 - accuracy: 0.6707 - val_loss: 2.7263 - val_accuracy: 0.4167
Epoch 10/20
980/980 [==============================] - 561s 573ms/step - batch: 489.5000 - size: 31.9755 - loss: 0.9806 - accuracy: 0.6726 - val_loss: 2.6305 - val_accuracy: 0.4231
Epoch 11/20
980/980 [==============================] - 553s 564ms/step - batch: 489.5000 - size: 31.9755 - loss: 0.9503 - accuracy: 0.6814 - val_loss: 2.6811 - val_accuracy: 0.4116
Epoch 12/20
980/980 [==============================] - 556s 567ms/step - batch: 489.5000 - size: 31.9755 - loss: 0.9385 - accuracy: 0.6875 - val_loss: 2.5996 - val_accuracy: 0.4189
Epoch 13/20
980/980 [==============================] - 572s 584ms/step - batch: 489.5000 - size: 31.9755 - loss: 0.9272 - accuracy: 0.6933 - val_loss: 2.6718 - val_accuracy: 0.4297
Epoch 14/20
980/980 [==============================] - ETA: 0s - batch: 489.5000 - size: 31.9755 - loss: 0.9126 - accuracy: 0.6944
```

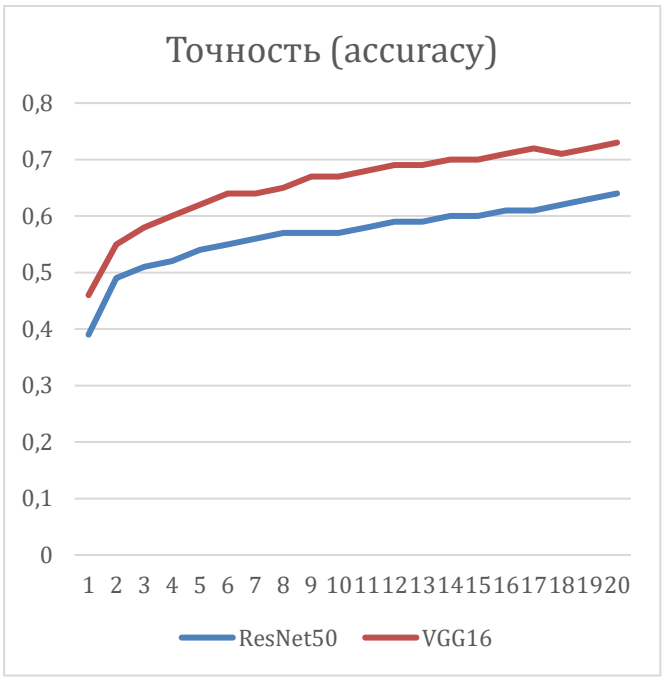
По завершении обучения были сформированы следующие графики точности и потерь для моделей ResNet и VGG (рисунок 2), также конечная таблица (таблица 1):

Таблица – 1

Модель	Число эпох	Обучение		Валидация	
		Потери (loss)	Точность (accuracy)	Потери (loss)	Точность (accuracy)
ResNet50	20	1,15	0,63	3,56	0,36
VGG16	20	0,85	0,73	2,94	0,42

Рисунок – 2





После завершения обучения были заданы стартовые значения ϵ (эпсилон) для проведения атак Fast Gradient Sign Method (FGSM) и Projected Gradient Descent (PGD). Для проведения атак используется библиотека Adversarial Robustness Toolbox, а именно функции FastGradientMethod и ProjectedGradientDescent. Импорт модели и первичная настройка атаки FGSM продемонстрирована во фрагменте кода 7.

Фрагмент кода – 7

```
eps_list = [0.003, 0.007, 0.011, 0.015, 0.019, 0.031, 0.039, 0.078, 0.19, 0,31]

classifier_resnet = KerasClassifier(model=model_resnet, clip_values=(0, 1))

for eps in eps_list:
  attack_fgsm_resnet = FastGradientMethod(estimator=classifier_resnet, eps=0.03)
  x_test_adv_resnet = attack_fgsm_resnet.generate(x_test_first)

  loss_test, accuracy_test = model_resnet.evaluate(x_test_adv_resnet, y_test_first)
  perturbation = np.mean(np.abs((x_test_adv_resnet - x_test_first)))
  print('Точность на искаженных данных: {:4.2f}%'.format(accuracy_test * 100))
  print('Среднее искажение: {:4.2f}'.format(perturbation))
```

Импорт модели и первичная настройка атаки PGD продемонстрирована во фрагменте кода 8.

Фрагмент кода – 8

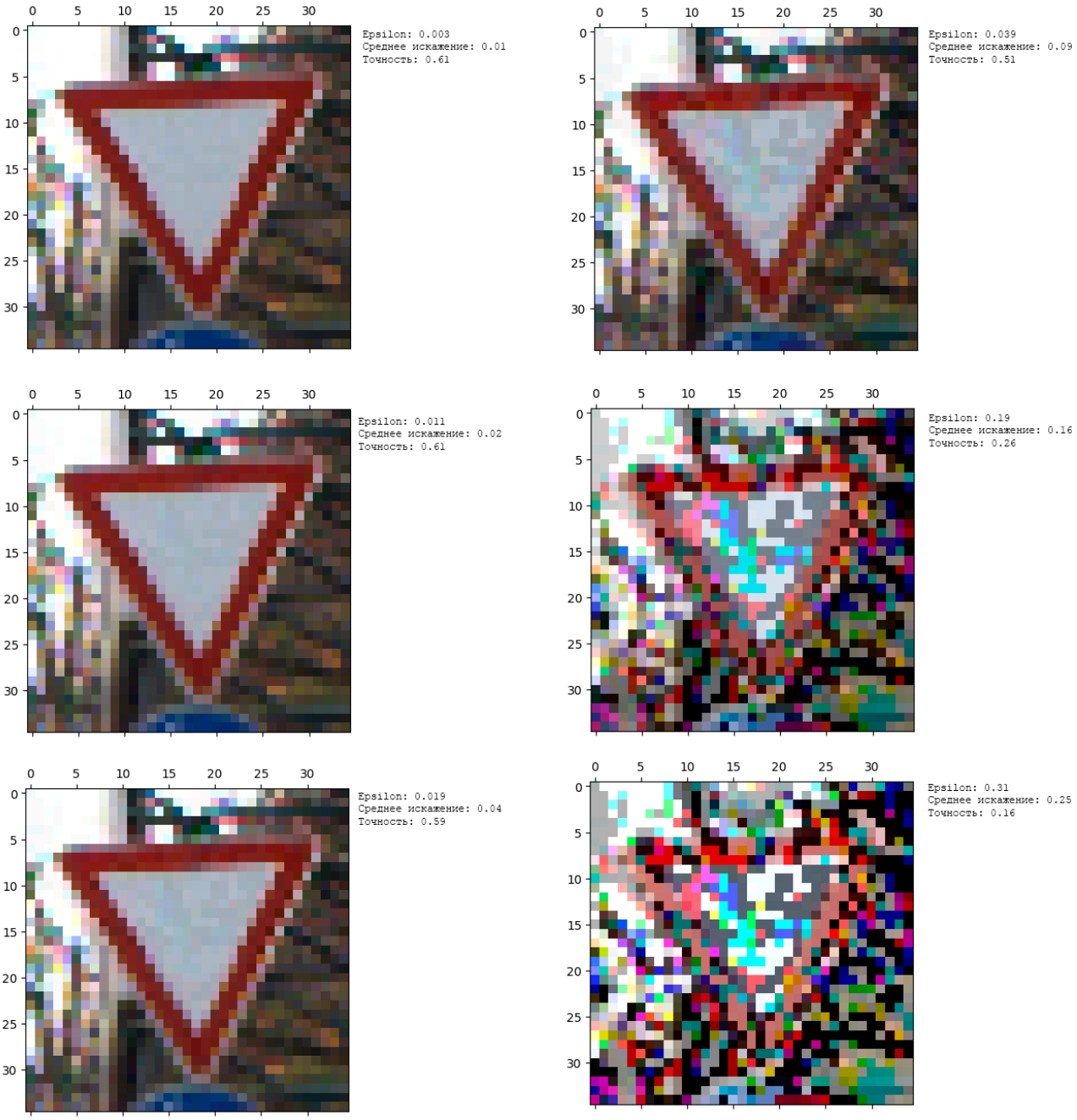
```
eps_list = [0.003, 0.007, 0.011, 0.015, 0.019, 0.031, 0.039, 0.078, 0.19, 0,31]

attack_pgd_resnet = ProjectedGradientDescent(estimator=classifier_resnet, targeted=False, eps=0.03,
batch_size=128, max_iter=20)
x_train_adv_pgd_resnet = attack_pgd_resnet.generate(x_train_first)
x_test_adv_pgd_resnet = attack_pgd_resnet.generate(x_test_first)
```

Атаки проводятся на уменьшенном наборе данных, а именно на первых 1000 элементов множества тренировочных и тестовых данных. Также важно отметить, что значения ϵ (эпсилон) используются из условия ϵ = [1/255, 2/255, 3/255, 4/255, 5/255, 8/255, 10/255, 20/255, 50/255, 80/255].

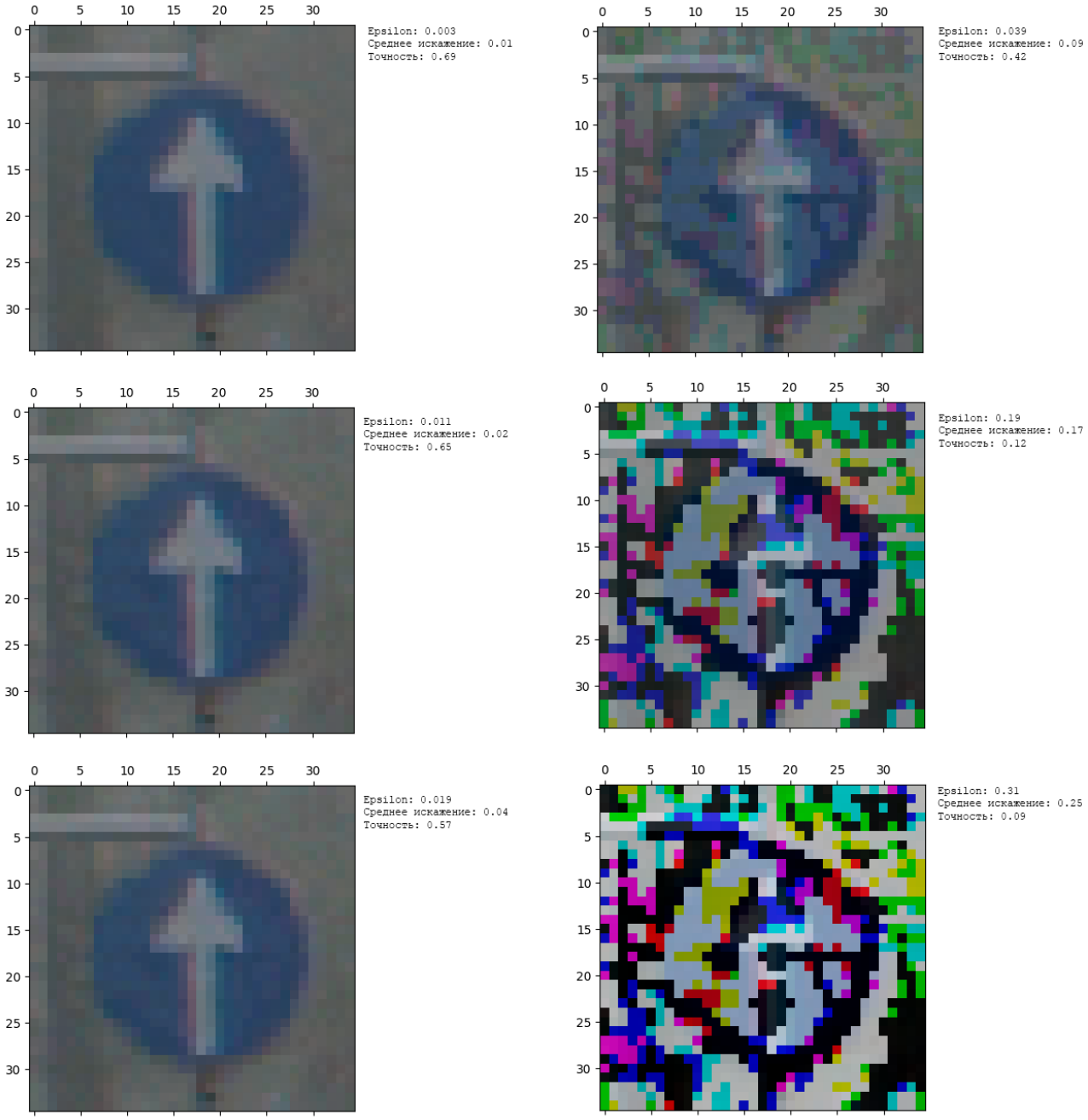
Для проверки полученных искаженных данных приведен пример на рисунке 3 с демонстрацией случайных изображений из сформированного набора атаки FGSM на модель ResNet с различными значениями ϵ.

Рисунок – 3



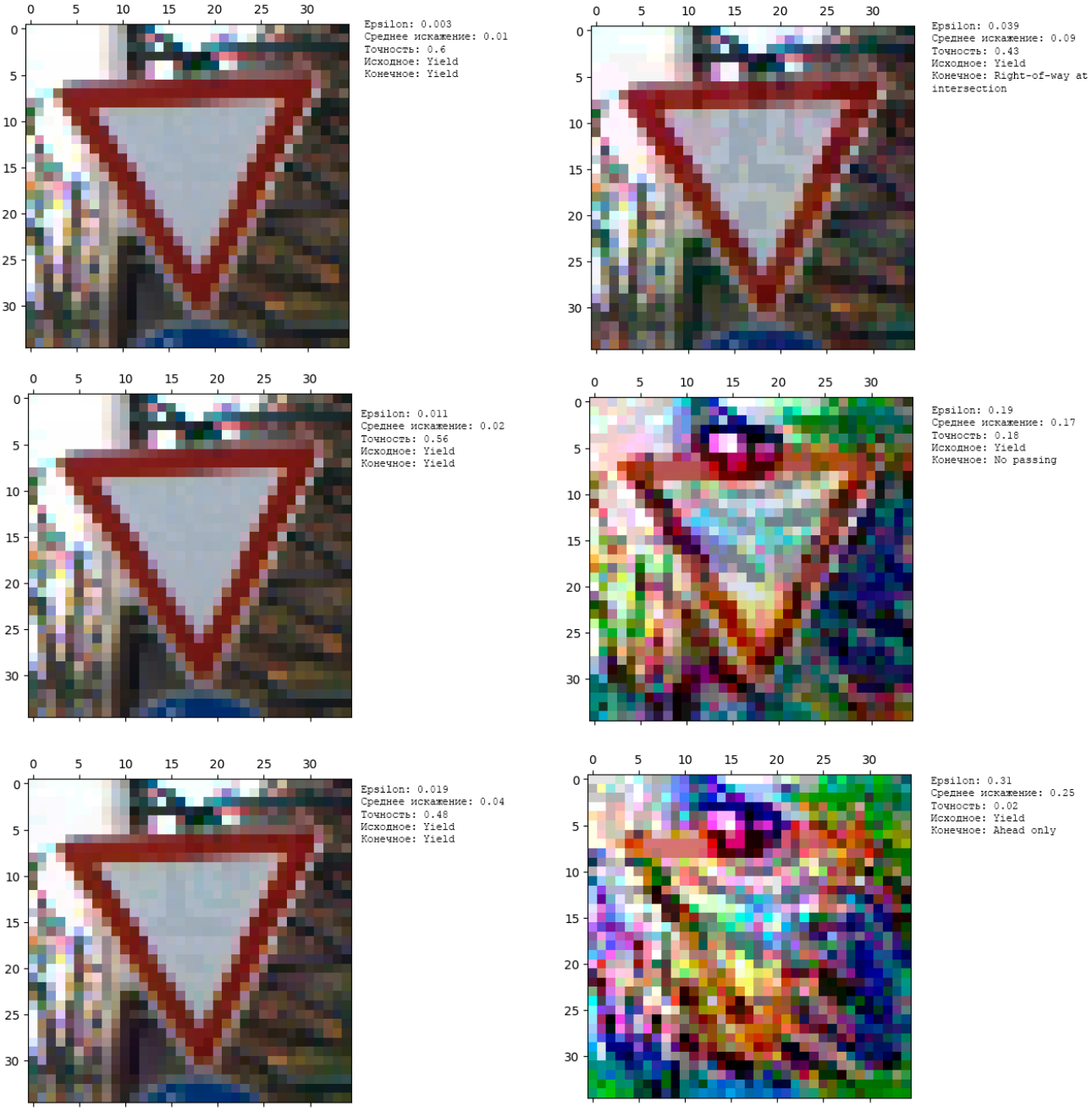
Аналогично с атакой FGSM для модели VGG на рисунке 4.

Рисунок – 4



Также была проведена атака PGD на обе модели ResNet и VGG, демонстрация полученных изображений на рисунке 5.

Рисунок – 5



По завершении подготовки атак были получены таблицы 2 и 3 для демонстрации точности классификации моделей ResNet (таблица 2) и VGG (таблица 3) в зависимости от параметра искажений ϵ.

Таблица – 2

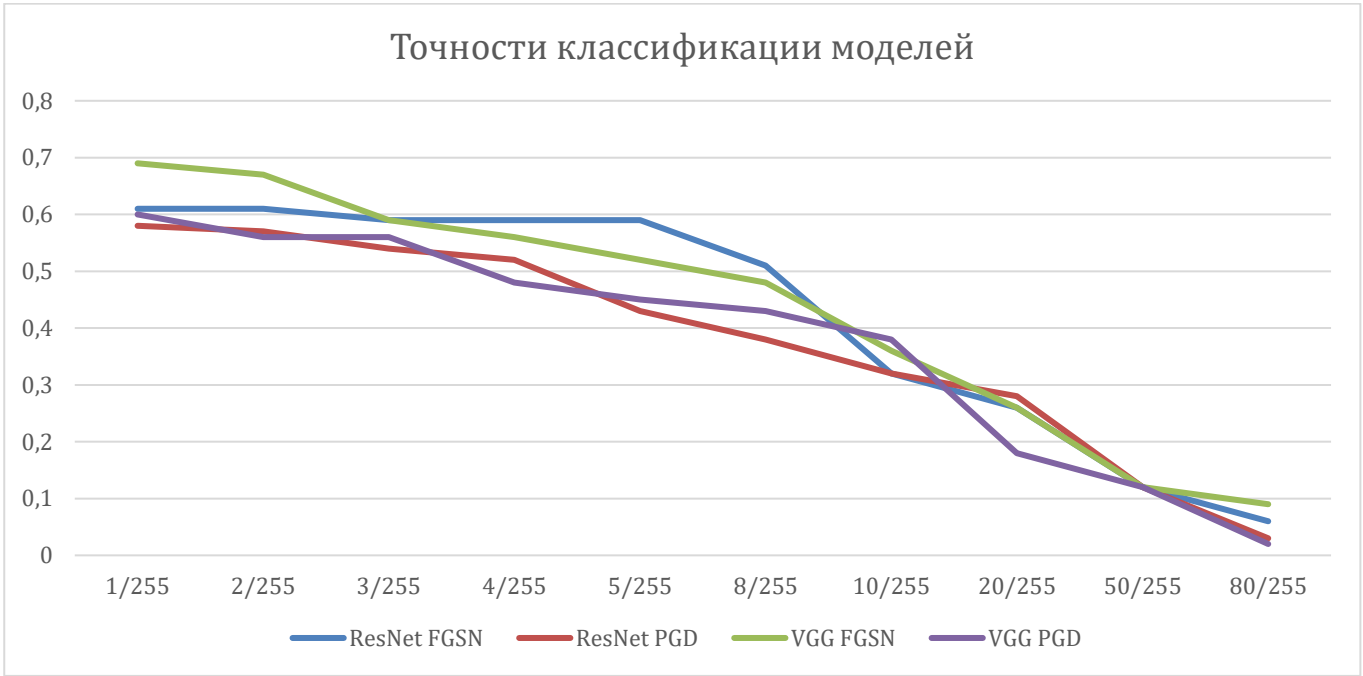
Атака	Точность ResNet при ϵ (эпсилон)					
	1/255	3/255	5/255	10/255	20/255	80/255
FGSM	0,61	0,59	0,59	0,51	0,26	0,06
PGD	0,58	0,54	0,43	0,28	0,12	0,03

Таблица – 3

Атака	Точность VGG при ϵ (эпсилон)					
	1/255	3/255	5/255	10/255	20/255	80/255
FGSM	0,69	0,65	0,57	0,42	0,12	0,09
PGD	0,6	0,56	0,48	0,43	0,18	0,02

Также был получен следующий график (рисунок 6) на основе собранных данных по атаке:

Рисунок – 6



Собранные результаты дают возможность предположить, что получаемые метрики могут сильно зависит от конкретных методов обучения и наборов данных. В данном случае VGG16 показывает наибольшую устойчивость и в большинстве случае

точность результата выше. Также можно увидеть, что скорость снижения точности сильно возрастает при переходе к искажению около 20/255.

Для создания целевой атаки необходимо задать метод подготовки атаки и указать явно, что планируемая атака целевая, далее, на этапе генерации изображений, указать целевую метку класса как продемонстрировано во фрагменте кода – 9.

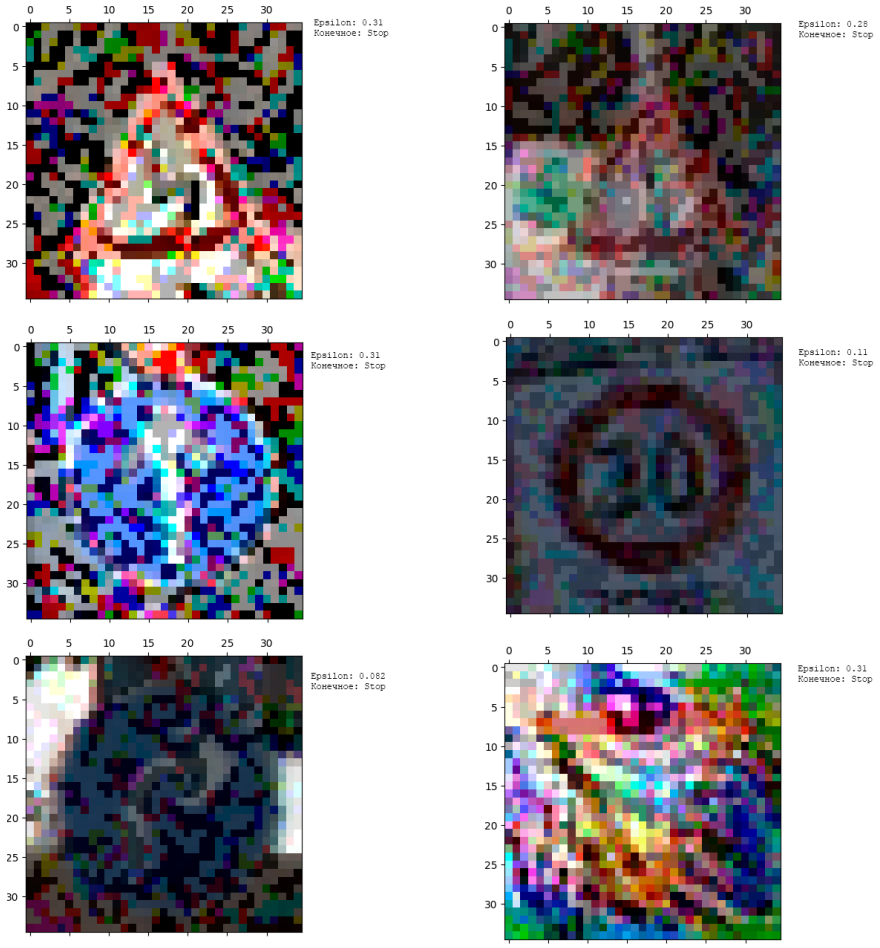
```
attack_fgsm_resnet_t = FastGradientMethod(estimator=classifier_resnet, eps=0.3, targeted=True)

x_test_adv_resnet = attack_fgsm_resnet.generate(x_train_first[:1], y_test_first[:1])
predictions = classifier_resnet.predict(x=x_test_adv_resnet)

plt.matshow(x_test_adv_resnet[:1][0])
plt.show()
print('adv: %s' % classes[np.argmax(classifier_resnet.predict(x_test_adv_resnet[:1, :])[0])])
print('og: %s' % classes[np.argmax(classifier_resnet.predict(x_test[:1, :])[0])] )
```

В качестве примера на рисунке 7 приведены примеры успешного формирования целевой атаки:

Рисунок – 7



Также на данном этапе получилось сформировать таблицу условной эффективности точности данных механизмов по атаке на наборы данных (таблица 4):

Таблица – 4

ϵ	PGD - Stop	PGD – Limit 30
ϵ = 1/255	0,58	0,56
ϵ = 3/255	0,58	0,48
ϵ = 5/255	0,52	0,46
ϵ = 10/255	0,47	0,34
ϵ = 20/255	0,34	0,26
ϵ = 50/255	0,08	0,08
ϵ = 80/255	0,03	0,01

Данные демонстрируют, что знак с ограничение лучше поддается атаке. Это можно объяснить тем, что фото с данным знаком часто размытые, что приводить к сложности его распознавания, а именно определить какой именно лимит скорости, так как в данном наборе есть и другие лимиты. Также важно отметить, что в данном конкретном примере знак стоп более эффективен для атаки на уровне искажения 20/255, более сильное искажение приводит серьезным повреждениям исходного изображения и уже не дают такой значительный прирост (упадок) точности.

Вывод

В результате выполнения работы получен некоторый опыт работы с инструментами организации атак на модели машинного обучения. Были также проведены эксперименты по атаке на модели машинного обучения методом черного и белого ящика, а также целевые и нецелевые.

В рамках работы были рассмотрены модели VGG16 и ResNet50, VGG16 показала несколько большую устойчивость к атакам, хоть и не значительную, но уже не в рамках погрешности. Было отмечено сильно ухудшение качества и точности моделей по достижении отметки искажения в 20/255.

При работе с целевыми атаки было отмечено, что наиболее эффективным значение, в рамках данного набора данных, является значение искажения 20/255.