



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

---

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

**Отчёт по лабораторной работе № 1**  
**по дисциплине «Анализ защищенности систем искусственного**  
**интеллекта»**

Выполнил  
Студент 2 курса  
Группы ББМО-01-23  
Белов Владимир Станиславович  
Шифр 23Б1716

Москва 2024

### **Задание.**

1. Скопировать проект по ссылке в локальную среду выполнения Jupyter (Google Colab) [https://github.com/ewatson2/EEL6812\\_DeepFool\\_Project](https://github.com/ewatson2/EEL6812_DeepFool_Project)
2. Сменить директорию исполнения на вновь созданную папку "EEL6812\_DeepFool\_Project" проекта.
3. Выполнить импорт библиотек.
4. Выполнить импорт вспомогательных библиотек из локальных файлов проекта.
5. Установить случайное рандомное значение в виде переменной `rand_seed={"Порядковый номер ученика группы в Гугл-таблице"}`
6. Установить указанное значение для `np.random.seed` и `torch.manual_seed`
7. Использовать в качестве устройства видеокарту
8. Загрузить датасет MNIST с параметрами `mnist_mean = 0.5`, `mnist_std = 0.5`, `mnist_dim = 28`
9. Загрузить датасет CIFAR-10 с параметрами `cifar_mean = [0.491, 0.482, 0.447]` `cifar_std = [0.202, 0.199, 0.201]` `cifar_dim = 32`
10. Выполнить настройку и загрузку DataLoader `batch_size = 64` `workers = 4`
11. Загрузить и оценить стойкость модели Network-In-Network Model к FGSM и DeepFool атакам на основе датасета CIFAR-10
12. Загрузить и оценить стойкость модели LeNet к FGSM и DeepFool атакам на основе датасета CIFAR-10
13. Выполнить оценку атакующих примеров для сетей.
14. Подготовить отчет в формате pdf.

## Отчёт по практической работе

Скопируем проект по ссылке в локальную среду выполнения Jupyter.

```
[2] !git clone https://github.com/ewatson2/EEL6812_DeepFool_Project.git

Cloning into 'EEL6812_DeepFool_Project'...
remote: Enumerating objects: 96, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 96 (delta 2), reused 1 (delta 1), pack-reused 93 (from 1)
Receiving objects: 100% (96/96), 33.99 MiB | 10.71 MiB/s, done.
Resolving deltas: 100% (27/27), done.
```

Рис. 1 – Копирование проекта по ссылке в локальную среду выполнения Jupyter.

Сменим директорию исполнения на вновь созданную папку "EEL6812\_DeepFool\_Project" проекта.

```
[57] cd /content/EEL6812_DeepFool_Project

/content/EEL6812_DeepFool_Project
```

Рис. 2 – Смена директории.

Выполним импорт библиотек, необходимых для нашей работы.

```
[4] import numpy as np
import json, torch
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, models
from torchvision.transforms import transforms
```

Рис. 3 – импорт библиотек.

Теперь выполним импорт вспомогательных библиотек из локальных файлов проекта.

```
[5] from models.project_models import FC_500_150, LeNet_CIFAR, LeNet_MNIST, Net
from utils.project_utils import get_clip_bounds, evaluate_attack, display_attack
```

Рис. 4 – Импорт вспомогательных библиотек.

Установим случайное рандомное значение в виде переменной `rand_seed=(6)`.

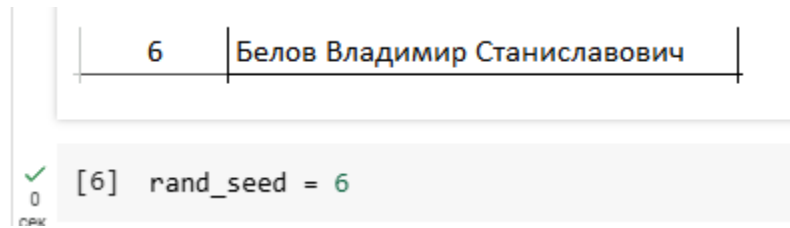


Рис. 5 – Установка случайного значения в виде переменной `rand_seed`.

Установим указанное значение для `np.random.seed` и `torch.manual_seed`.

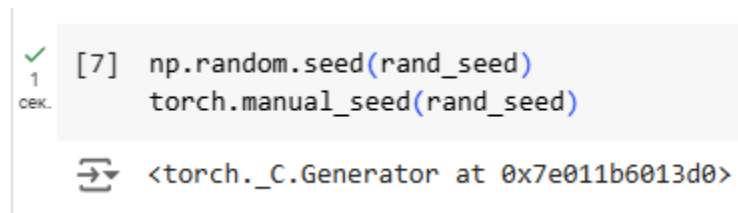


Рис. 6 – `np.random.seed` и `torch.manual_seed`.

Будем использовать в качестве устройства видеокарту.

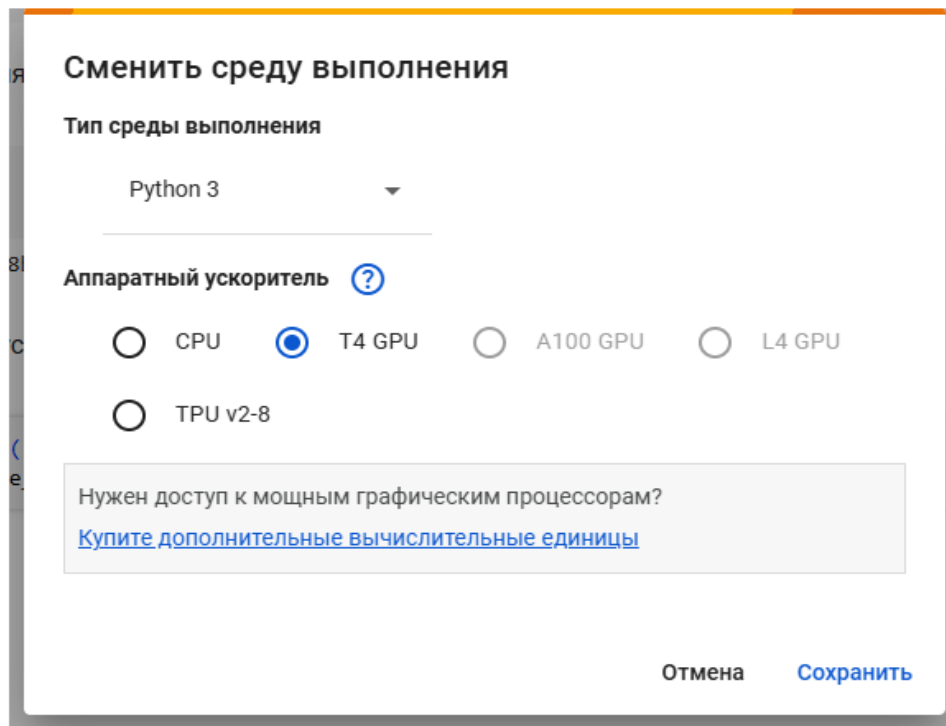


Рис. 7 – Смена среды выполнения.

Загрузим датасет MNIST с параметрами данными по условию задачи.

```
0
сек.
mnist_mean = 0.5
mnist_std = 0.5
mnist_dim = 28

use_cuda = torch.cuda.is_available()
device = torch.device('cuda' if use_cuda else 'cpu')

mnist_min, mnist_max = get_clip_bounds(mnist_mean, mnist_std, mnist_dim)
mnist_min = mnist_min.to(device)
mnist_max = mnist_max.to(device)
mnist_tf = transforms.Compose([ transforms.ToTensor(), transforms.Normalize(mean=mnist_mean, std=mnist_std)])
mnist_tf_train = transforms.Compose([ transforms.RandomHorizontalFlip(), transforms.ToTensor(), transforms.Normalize( mean=mnist_mean, std=mnist_std)])
mnist_tf_inv = transforms.Compose([ transforms.Normalize( mean=0.0, std=np.divide(1.0, mnist_std)), transforms.Normalize( mean=np.multiply(-1.0, mnist_std), std=1.0)])
mnist_temp = datasets.MNIST(root='datasets/mnist', train=True, download=True, transform=mnist_tf_train)
mnist_train, mnist_val = random_split(mnist_temp, [50000, 10000])
mnist_test = datasets.MNIST(root='datasets/mnist', train=False, download=True, transform=mnist_tf)
```

Рис. 8 – Загрузка датасета MNIST

Загрузим датасет CIFAR-10 с параметрами данными по условию задачи.

```
1
сек.
cifar_mean = [0.491, 0.482, 0.447]
cifar_std = [0.202, 0.199, 0.201]
cifar_dim = 32

cifar_min, cifar_max = get_clip_bounds(cifar_mean, cifar_std, cifar_dim)
cifar_min = cifar_min.to(device)
cifar_max = cifar_max.to(device)
cifar_tf = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean=cifar_mean, std=cifar_std)])
cifar_tf_train = transforms.Compose([transforms.RandomCrop(size=cifar_dim, padding=4), transforms.RandomHorizontalFlip(), transforms.ToTensor(), transforms.Normalize(mean=cifar_mean, std=cifar_std)])
cifar_tf_inv = transforms.Compose([transforms.Normalize(mean=[0.0, 0.0, 0.0], std=np.divide(1.0, cifar_std)), transforms.Normalize(mean=np.multiply(-1.0, cifar_mean), std=[1.0, 1.0, 1.0])])
cifar_temp = datasets.CIFAR10(root='datasets/cifar-10', train=True, download=True, transform=cifar_tf_train)
cifar_train, cifar_val = random_split(cifar_temp, [40000, 10000])
cifar_test = datasets.CIFAR10(root='datasets/cifar-10', train=False, download=True, transform=cifar_tf)
cifar_classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

Рис. 9 – Загрузка датасета CIFAR-10

Выполним настройку и загрузку DataLoader

```
1
сек.
batch_size = 64
workers = 4

mnist_loader_train = DataLoader(mnist_train, batch_size=batch_size, shuffle=True, num_workers=workers)
mnist_loader_val = DataLoader(mnist_val, batch_size=batch_size, shuffle=False, num_workers=workers)
mnist_loader_test = DataLoader(mnist_test, batch_size=batch_size, shuffle=False, num_workers=workers)
cifar_loader_train = DataLoader(cifar_train, batch_size=batch_size, shuffle=True, num_workers=workers)
cifar_loader_val = DataLoader(cifar_val, batch_size=batch_size, shuffle=False, num_workers=workers)
cifar_loader_test = DataLoader(cifar_test, batch_size=batch_size, shuffle=False, num_workers=workers)
```

Рис. 10 – Конфигурация DataLoader

Загрузим и оценим стойкость модели Network-In-Network Model к FGSM и DeepFool атакам на основе датасета CIFAR-10, зададим гиперпараметры Deepfool.

```
0 0 sec. fgsm_eps = 0.2
model = Net().to(device)
model.load_state_dict(torch.load('weights/clean/cifar_nin.pth', map_location=torch.device('cpu')))
evaluate_attack('cifar_nin_fgsm.csv', 'results', device, model, cifar_loader_test, cifar_min, cifar_max, fgsm_eps, is_fgsm=True)
print('')

deep_args = [10, 10, 0.02, 50] # Задаём гиперпараметры DeepFool
evaluate_attack('cifar_nin_deepfool.csv', 'results', device, model, cifar_loader_test, cifar_min, cifar_max, deep_args, is_fgsm=False)
if device.type == 'cuda': torch.cuda.empty_cache()
print('')

FGSM Test Error : 81.29%
FGSM Robustness : 1.77e-01
FGSM Time (All Images) : 0.67 s
FGSM Time (Per Image) : 67.07 us

DeepFool Test Error : 93.76%
DeepFool Robustness : 2.12e-02
DeepFool Time (All Images) : 185.12 s
DeepFool Time (Per Image) : 18.51 ms
```

Рис. 11 – Устойчивость модели Network-In-Network Model к FGSM и DeepFool атакам на основе датасета CIFAR-10

DeepFool атака приводит к большим ошибкам, чем FGSM на модели NiN датасета CIFAR-10. Загрузим и оценим стойкость модели LeNet к FGSM и DeepFool атакам на основе датасета CIFAR-10

```
0 0 sec. fgsm_eps = 0.1
model = LeNet_CIFAR().to(device)
model.load_state_dict(torch.load('weights/clean/cifar_lenet.pth', map_location=torch.device('cpu')))
evaluate_attack('cifar_lenet_fgsm.csv', 'results', device, model, cifar_loader_test, cifar_min, cifar_max, fgsm_eps, is_fgsm=True)
print('')
evaluate_attack('cifar_lenet_deepfool.csv', 'results', device, model, cifar_loader_test, cifar_min, cifar_max, deep_args, is_fgsm=False)
if device.type == 'cuda': torch.cuda.empty_cache()

FGSM Test Error : 91.71%
FGSM Robustness : 8.90e-02
FGSM Time (All Images) : 0.40 s
FGSM Time (Per Image) : 40.08 us

DeepFool Test Error : 87.81%
DeepFool Robustness : 1.78e-02
DeepFool Time (All Images) : 73.27 s
DeepFool Time (Per Image) : 7.33 ms
```

Рис. 11 – Устойчивость модели LeNet к FGSM и DeepFool атакам на основе датасета CIFAR-10

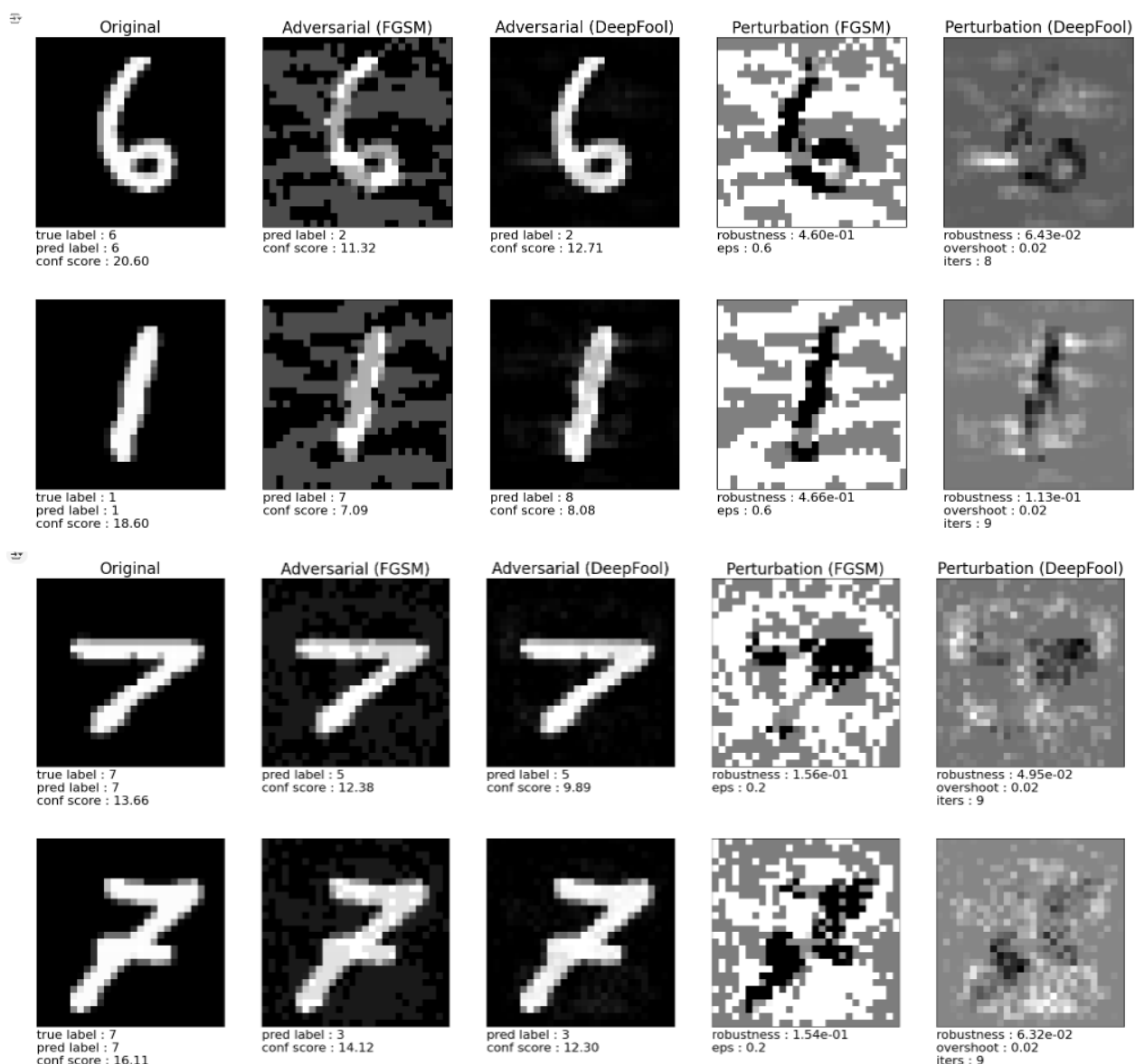
FGSM атака приводит к большим ошибкам, чем DeepFool на модели LeNet датасета CIFAR-10. Выполним оценку атакующих примеров для сетей.

```

#LeNet
fgsm_eps = 0.6
model = LeNet_MNIST().to(device)
model.load_state_dict(torch.load('weights/clean/mnist_lenet.pth'))
display_attack(device, model, mnist_test, mnist_tf_inv, mnist_min, mnist_max, fgsm_eps, deep_args, has_labels=False, l2_norm=True, pert_scale=1.0, fig_rows=2, fig_width=25, fig_height=11)
if device.type == 'cuda': torch.cuda.empty_cache()
#FCNet
fgsm_eps = 0.2
model = FC_500_150().to(device)
model.load_state_dict(torch.load('weights/clean/mnist_fc.pth'))
display_attack(device, model, mnist_test, mnist_tf_inv, mnist_min, mnist_max, fgsm_eps, deep_args, has_labels=False, l2_norm=True, pert_scale=1.0, fig_rows=2, fig_width=25, fig_height=11)
if device.type == 'cuda': torch.cuda.empty_cache()
#Network-in-Network
fgsm_eps = 0.2
model = Net().to(device)
model.load_state_dict(torch.load('weights/clean/cifar_nin.pth'))
display_attack(device, model, cifar_test, cifar_tf_inv, cifar_min, cifar_max, fgsm_eps, deep_args, has_labels=False, l2_norm=True, pert_scale=1.0, fig_rows=2, fig_width=25, fig_height=11, label_map=cifar_classes)
if device.type == 'cuda': torch.cuda.empty_cache()
#LeNet CIFAR-10
fgsm_eps = 0.1
model = LeNet_CIFAR().to(device)
model.load_state_dict(torch.load('weights/clean/cifar_lenet.pth'))
display_attack(device, model, cifar_test, cifar_tf_inv, cifar_min, cifar_max, fgsm_eps, deep_args, has_labels=False, l2_norm=True, pert_scale=1.0, fig_rows=2, fig_width=25, fig_height=11, label_map=cifar_classes)
if device.type == 'cuda': torch.cuda.empty_cache()

```

Рис. 12 – Атакующие примеры на сети LeNet и FCNet на датасете MNIST, NiN LeNet на датасете CIFAR-10.



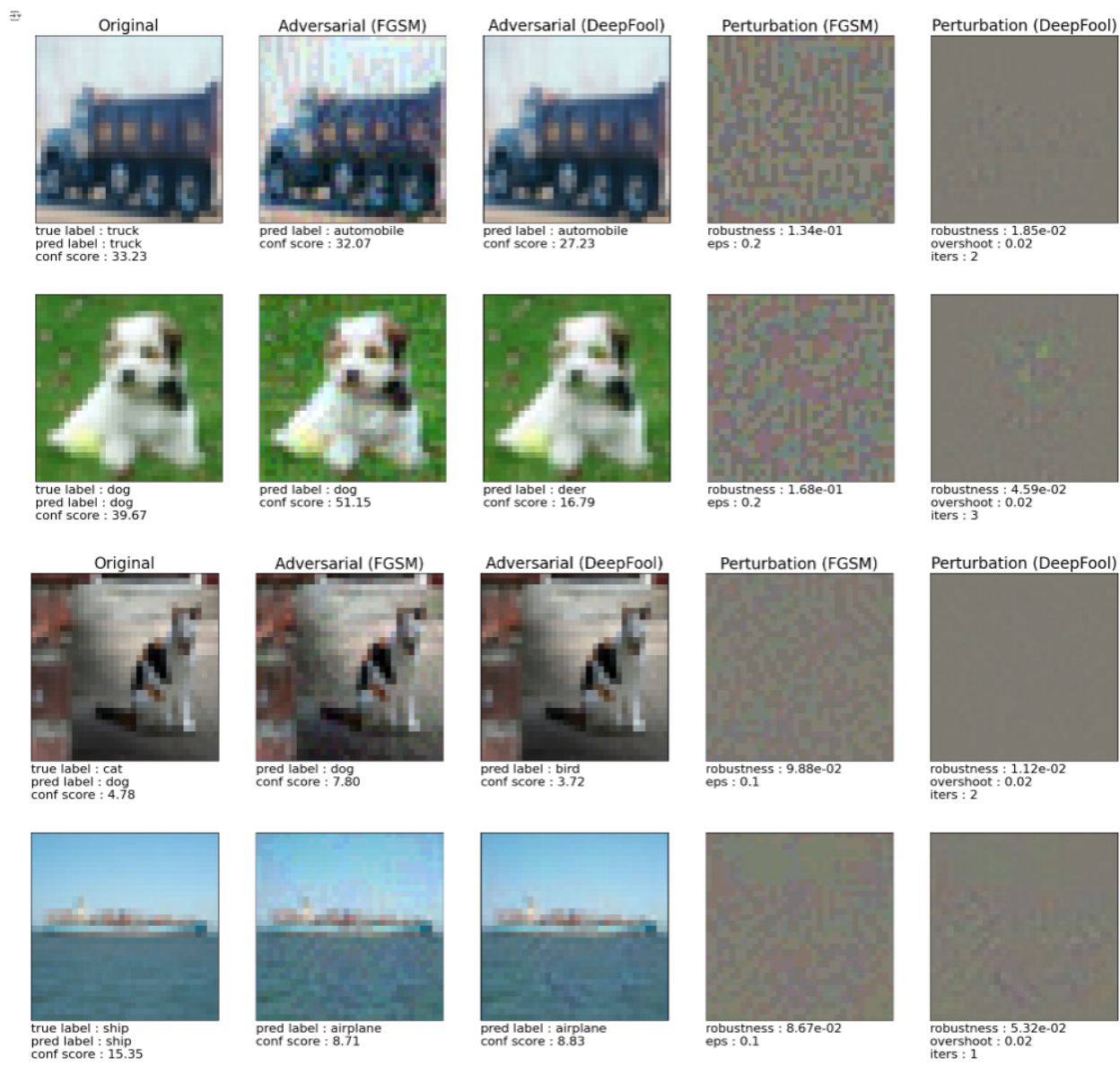


Рис. 13-16 – Результаты атакующих примеров на сети.

Отразить отличия для  $fgsm\_eps=(0.001, 0.02, 0.5, 0.9, 10)$  и выявить закономерность/обнаружить отсутствие влияние параметра  $eps$  для сетей FC LeNet на датасете MNIST, NiN LeNet на датасете CIFAR.



```

[43] #LoadNet
model = LeNet_MNIST().to(device)
model.load_state_dict(torch.load('weights/clean/mnist_lenet.pth'))

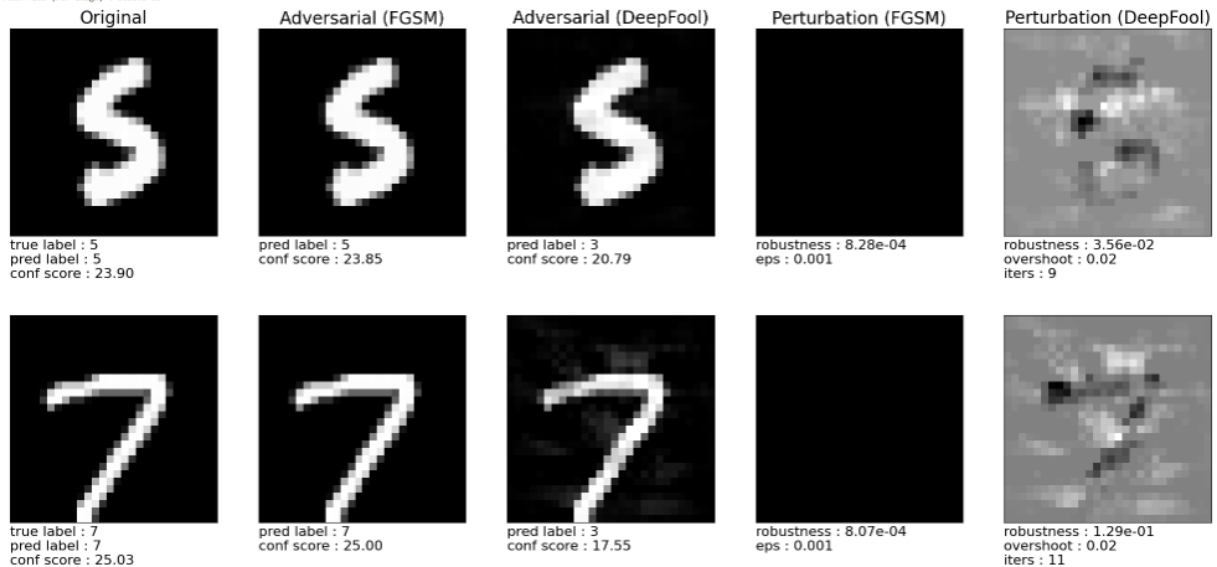
for fgsm_eps in [0.001, 0.02, 0.5, 0.9, 10]:
    print(f"Boonema fgsm_eps = {fgsm_eps}")
    mnist_lenet_fgsm = 'mnist_lenet_fgsm' + str(fgsm_eps) + '.csv'
    evaluate_attack(mnist_lenet_fgsm, 'results', device, model, mnist_loader_test, mnist_min, mnist_max, fgsm_eps, is_fgsm=True)
    display_attack(device, model, mnist_test, mnist_tf_low, mnist_min, mnist_max, fgsm_eps, deep_args, has_labels=False, is_fgsm=True, port_scale=1.0, fig_rows=2, fig_width=25, fig_height=11)
    if device.type == 'cuda': torch.cuda.empty_cache()
    print("")

```

```

Boonema fgsm_eps = 0.001:
FGSM Test Error : 1.69%
FGSM Robustness : 8.89e-04
FGSM Time (All Images) : 1.39 s
FGSM Time (Per Image) : 138.78 us

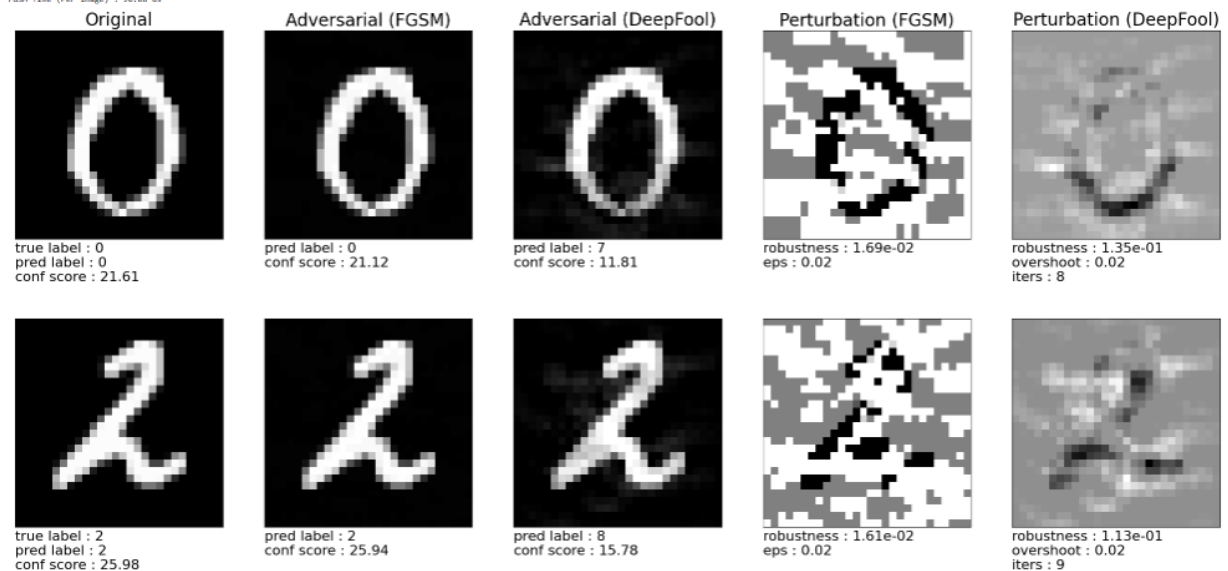
```



```

Boonema fgsm_eps = 0.02:
FGSM Test Error : 2.56%
FGSM Robustness : 1.50e-02
FGSM Time (All Images) : 0.98 s
FGSM Time (Per Image) : 98.88 us

```



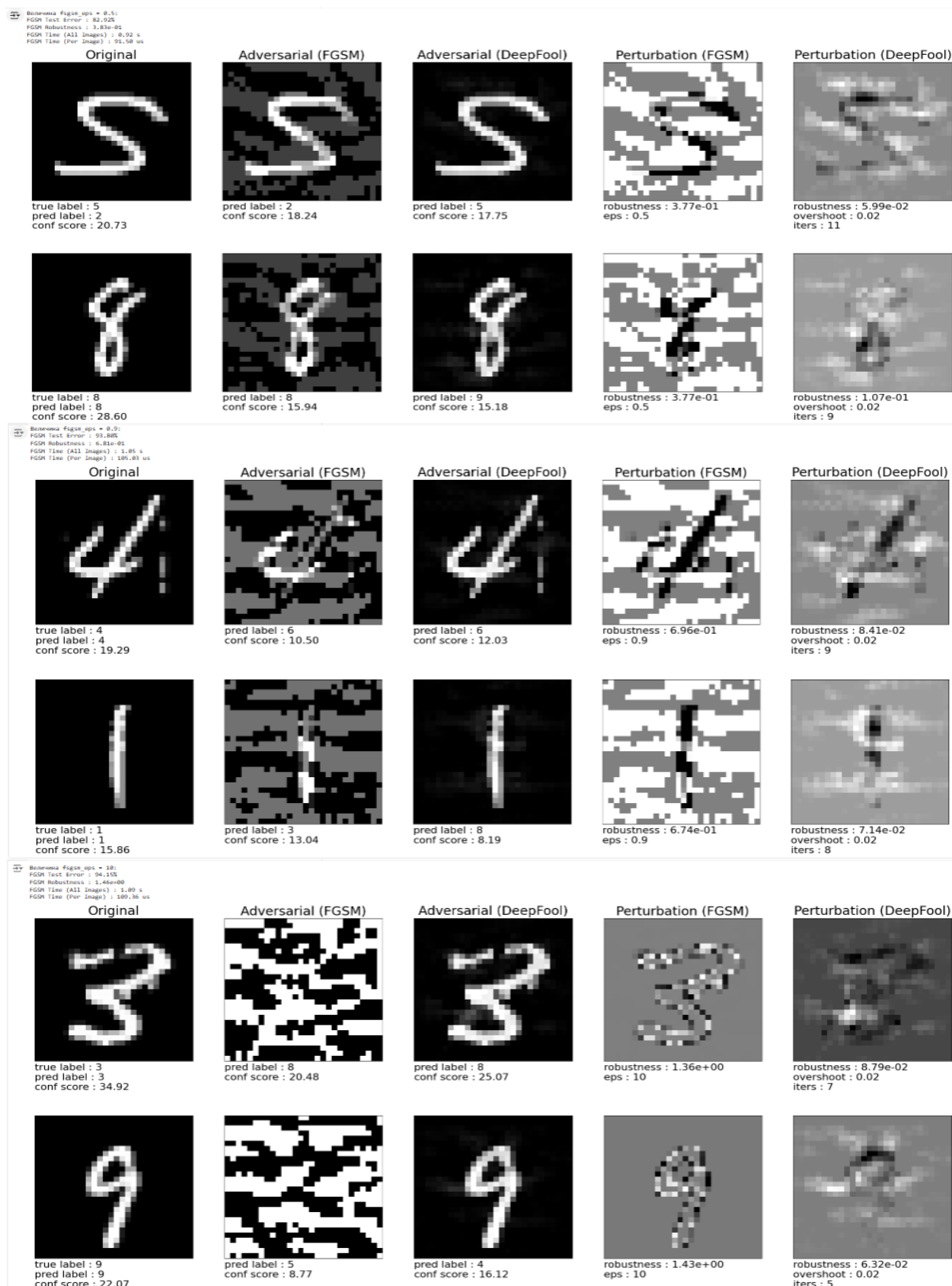


Рис. 17-21 – Результаты атакующих примеров на LeNet датасета MNIST.

LeNet на датасете MNIST выдаёт самую маленькую ошибку на fsgsm\_eps = 0.001 - 1.69%, однако на fsgsm\_eps = 0.05 она сильно возрастёт до 82.92% и с увеличением fsgsm\_eps продолжит расти.

```
[ ] #FCNet
model = FC_500_550().to(device)
model.load_state_dict(torch.load('weights/clean/mnist_fc.pth'))

for fsgm_eps in [0.001, 0.02, 0.5, 0.9, 10]:
    print(f"Bonumena Fsgm_eps = {fsgm_eps}")
    mnist_fc_fsgm = 'mnist_fc_fsgm' + str(fsgm_eps) + '.csv'
    evaluate_attack(mnist_fc_fsgm, 'results', device, model, mnist_loader_test, mnist_min, mnist_max, fsgm_eps, is_fsgm=True)
    display_attack(device, model, mnist_test, mnist_tf_img, mnist_min, mnist_max, fsgm_eps, deep_args, has_labels=False, l2_norm=True, pert_scale=1.0, fig_rows=2, fig_width=25, fig_height=15)
    if device.type == "cuda": torch.cuda.empty_cache()
    print("")
```

```
11 Bonumena Fsgm_eps = 0.001:
FSGM Test Error : 1.69%
FSGM Robustness : 8.88e-04
FSGM Time (All Images) : 0.56 s
FSGM Time (Per Image) : 56.18 us
```



```
11 Bonumena Fsgm_eps = 0.02:
FSGM Test Error : 5.54%
FSGM Robustness : 1.88e-02
FSGM Time (All Images) : 0.72 s
FSGM Time (Per Image) : 72.39 us
```



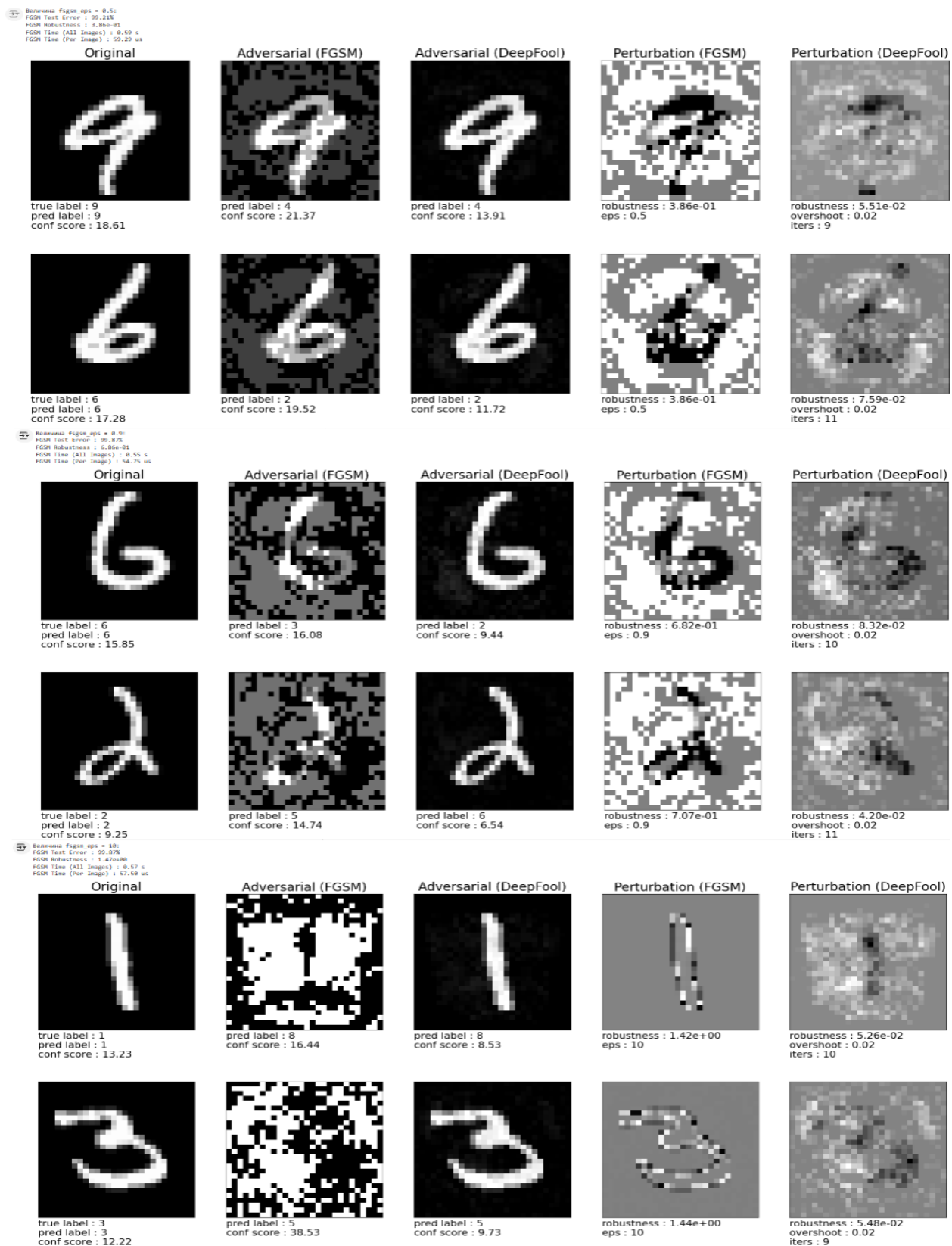





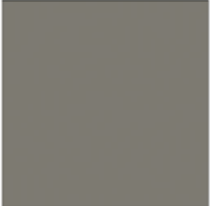
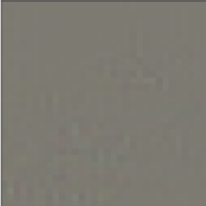



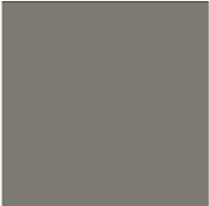
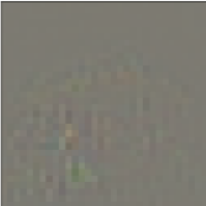
Рис. 22-26 – Результаты атакующих примеров на FCNet датасета MNIST.

FC на датасете MNIST, выдаёт большие ошибки в отличии LeNet (MNIST), но всё ещё сохраняет небольшие значения при fsgm\_eps 0.01 и 0.02




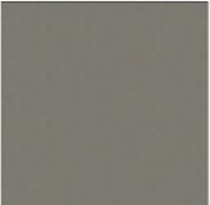






```
#Network-In-Network
model = Net().to(device)
model.load_state_dict(torch.load("weights/clean/cifar_min.pth"))

for fsgm_eps in [0.001, 0.02, 0.5, 0.0, 10]:
    print(f"Baseline fsgm_eps = {fsgm_eps}")
    cifar_min_fsgm = "cifar_min_fsgm" + str(fsgm_eps) + ".csv"
    evaluate_attack(cifar_min_fsgm, 'results', device, model, mnist_loader_test, mnist_min, mnist_max, fsgm_eps, is_fsgm=True)
    display_attack(device, model, cifar_test, cifar_tf_inv, cifar_min, cifar_max, fsgm_eps, deep_args, has_labels=False, is_norm=True, pert_scale=1.0, fig_rows=2, fig_width=25, fig_height=15, label_map=cifar_classes)
    if device.type == 'cuda': torch.cuda.empty_cache()
    print("")
```

Baseline fsgm\_eps = 0.001:  
 FGM Text Error : 10.125  
 FGM Robustness : 8.92e-04  
 FGM Time (All Images) : 1.32 s  
 FGM Time (Per Image) : 132.40 us

Original	Adversarial (FGSM)	Adversarial (DeepFool)	Perturbation (FGSM)	Perturbation (DeepFool)
 true label : airplane pred label : airplane conf score : 31.40	 pred label : airplane conf score : 31.25	 pred label : ship conf score : 25.38	 robustness : 6.60e-04 eps : 0.001	 robustness : 1.58e-02 overshoot : 0.02 iters : 1
 true label : automobile pred label : automobile conf score : 51.00	 pred label : automobile conf score : 51.24	 pred label : ship conf score : 31.10	 robustness : 5.90e-04 eps : 0.001	 robustness : 3.06e-02 overshoot : 0.02 iters : 2

Baseline fsgm\_eps = 0.02:  
 FGM Text Error : 38.782  
 FGM Robustness : 1.78e-02  
 FGM Time (All Images) : 1.13 s  
 FGM Time (Per Image) : 112.62 us

Original	Adversarial (FGSM)	Adversarial (DeepFool)	Perturbation (FGSM)	Perturbation (DeepFool)
 true label : frog pred label : frog conf score : 31.91	 pred label : frog conf score : 26.41	 pred label : deer conf score : 23.61	 robustness : 2.41e-02 eps : 0.02	 robustness : 2.63e-02 overshoot : 0.02 iters : 3
 true label : horse pred label : horse conf score : 45.55	 pred label : horse conf score : 48.98	 pred label : bird conf score : 27.28	 robustness : 1.22e-02 eps : 0.02	 robustness : 3.00e-02 overshoot : 0.02 iters : 3

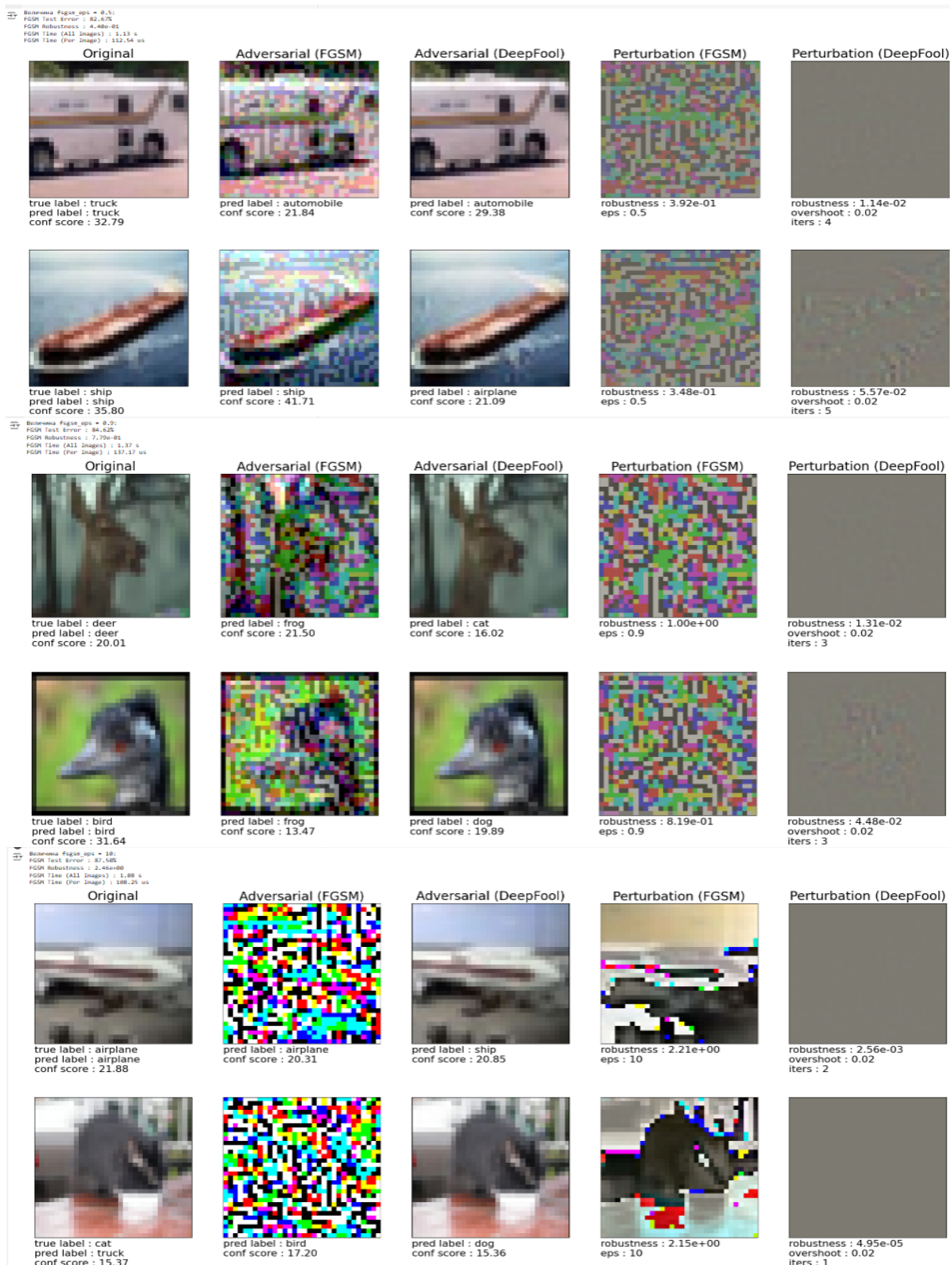


Рис. 27-32 – Результаты атакующих примеров на NiN датасета CIFAR-10.

Network-in-Network показывает изначально большую ошибку, fsgsm\_eps = 0.001: FGSM Test Error : 10.12%, но немного меньше ошибка на высоких eps в отличие от первых двух, у которых доходит до ~99.8%.











```

@Inlet CIFAR-10
model = torch.nn.LSTM(10, 10, 1, 1)
model.load_state_dict(torch.load('weights/cifar_lstm.pth'))











for fsgm_eps in [0.001, 0.02, 0.5, 0.9, 1.0]:
    print(f"Robustness fsgm_eps = {fsgm_eps}")
    cifar_lstm_fsgm = 'cifar_lstm_fsgm' + str(fsgm_eps) + '.csv'
    evaluate_attack(cifar_lstm_fsgm, 'results', device, model, cifar_loader_test, cifar_min, cifar_max, fsgm_eps, is_fsgm=True)
    display_attack(device, model, cifar_test, cifar_tf_loader, cifar_min, cifar_max, fsgm_eps, deep_argh, has_labels=True, num_epochs=10, port_scale=1.0, fig_rows=2, fig_width=25, fig_height=11, label_map=cifar_classes)
    if device.type == 'cuda': torch.cuda.empty_cache()
    print("")

```

Robustness fsgm\_eps = 0.001:  
FGSM Test Error : 22.72%  
FGSM Robustness : 8.92e-04  
FGSM Time (All Images) : 1.35 s  
FGSM Time (Per Image) : 135.11 us

Original	Adversarial (FGSM)	Adversarial (DeepFool)	Perturbation (FGSM)	Perturbation (DeepFool)
 true label : airplane pred label : airplane conf score : 6.81	 pred label : airplane conf score : 6.74	 pred label : deer conf score : 6.54	 robustness : 8.22e-04 eps : 0.001	 robustness : 2.67e-03 overshoot : 0.02 iters : 3
 true label : airplane pred label : airplane conf score : 15.53	 pred label : airplane conf score : 15.47	 pred label : bird conf score : 11.62	 robustness : 4.80e-04 eps : 0.001	 robustness : 3.12e-02 overshoot : 0.02 iters : 4

Robustness fsgm\_eps = 0.02:  
FGSM Test Error : 47.76%  
FGSM Robustness : 1.78e-02  
FGSM Time (All Images) : 1.27 s  
FGSM Time (Per Image) : 126.65 us

Original	Adversarial (FGSM)	Adversarial (DeepFool)	Perturbation (FGSM)	Perturbation (DeepFool)
 true label : horse pred label : horse conf score : 18.09	 pred label : horse conf score : 15.62	 pred label : dog conf score : 13.01	 robustness : 1.22e-02 eps : 0.02	 robustness : 1.87e-02 overshoot : 0.02 iters : 3
 true label : deer pred label : bird conf score : 3.15	 pred label : bird conf score : 5.10	 pred label : cat conf score : 2.66	 robustness : 2.63e-02 eps : 0.02	 robustness : 1.98e-03 overshoot : 0.02 iters : 1



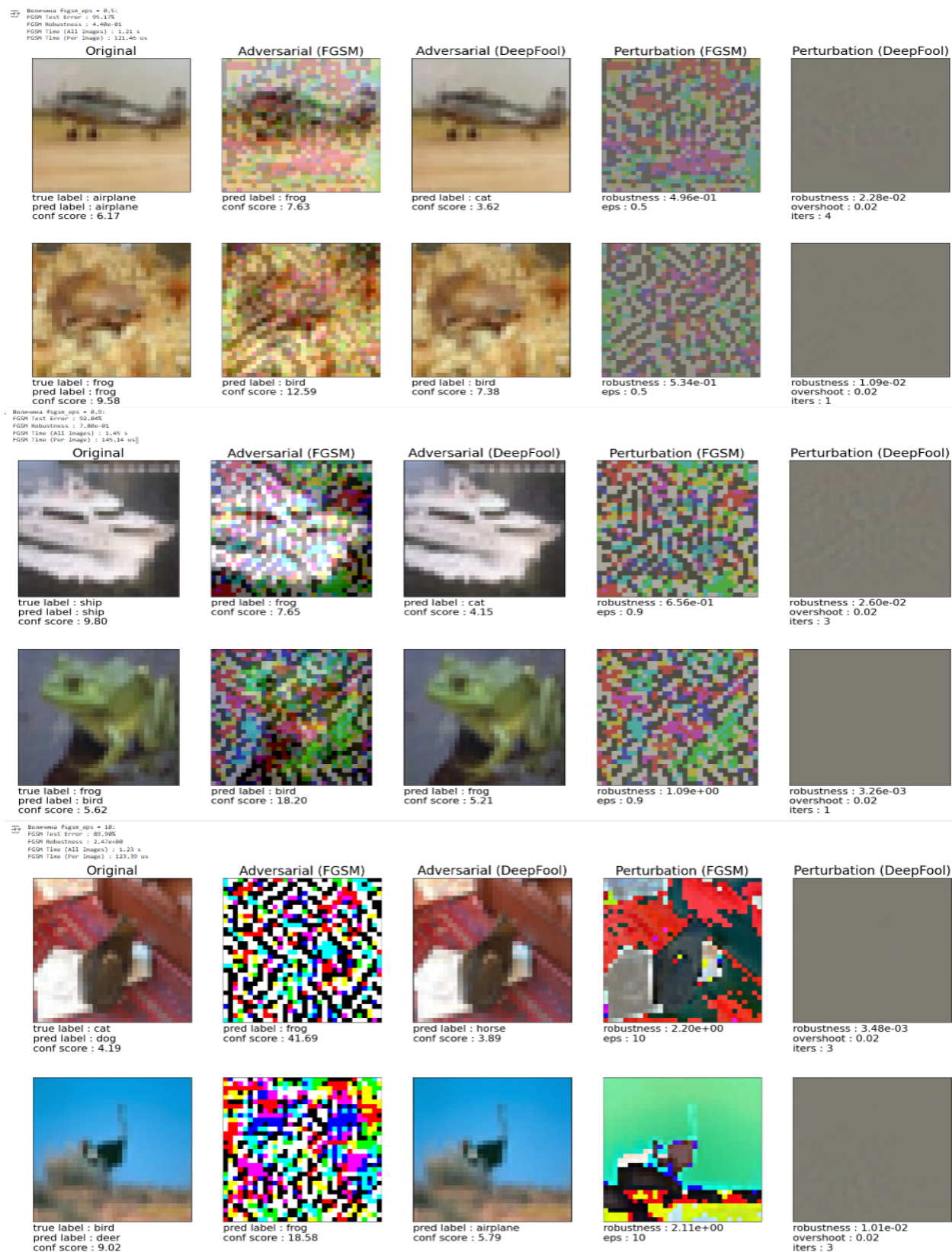


Рис. 33-37 – Результаты атакующих примеров на LeNet датасета CIFAR-10.



LeNet на датасете CIFAR-10 выдаёт самую большую стартовую ошибку  $\text{fsgsm\_eps} = 0.001$ : FGSM Test Error : 22.72%, и самую большую ошибку  $\text{fsgsm\_eps} = 0.5$ : FGSM Test Error : 95.17%, однако после дальнейшего увеличения  $\text{fsgsm\_eps}$ , ошибка начинает падать, что не замечается у предыдущих трех нейросетей.

### **Вывод:**

Увеличение  $\text{fsgsm\_eps}$  увеличивает успешность атаки FGSM, однако чем выше значение, тем сильнее искажается изображение, что становится заметно невооруженным глазом.

LeNet на датасете MNIST выдаёт самую маленькую ошибку на  $\text{fsgsm\_eps} = 0.001$  - 1.69%, однако на  $\text{fsgsm\_eps} = 0.05$  она сильно возрастет до 82.92% и с увеличением  $\text{fsgsm\_eps}$  продолжит расти.

FC на датасете MNIST, выдаёт большие ошибки в отличие LeNet (MNIST), но всё ещё сохраняет небольшие значения при  $\text{fsgsm\_eps} 0.01$  и  $0.02$

Network-in-Network показывает изначально большую ошибку,  $\text{fsgsm\_eps} = 0.001$ : FGSM Test Error : 10.12%, но немного меньше ошибка на высоких  $\text{eps}$  в отличие от первых двух, у которых доходит до ~99.8%

LeNet на датасете CIFAR-10 выдаёт самую большую стартовую ошибку  $\text{fsgsm\_eps} = 0.001$ : FGSM Test Error : 22.72%, и самую большую ошибку  $\text{fsgsm\_eps} = 0.5$ : FGSM Test Error : 95.17%, однако после дальнейшего увеличения  $\text{fsgsm\_eps}$ , ошибка начинает падать, что не замечается у предыдущих трех нейросетей.