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

Институт кибербезопасности и цифровых технологий

КБ-4 «Интеллектуальные системы информационной безопасности»

Отчет по лабораторной работе №1

по дисциплине: «Анализ защищенности систем искусственного интеллекта»

Выполнил:

Студент группы ББМО-01-22

Загороднов Егор Алексеевич

Проверил:

К.т.н. Спирин Андрей Андреевич

Москва 2023

Скопируем проект по ссылке в локальную среду выполнения Jupyter (Google Colab) https://github.com/ewatson2/EEL6812_DeepFool_Project . Процесс копирования показан на рисунке 1.

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

     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
     Receiving objects: 100% (96/96), 33.99 MiB | 10.63 MiB/s, done.
     Resolving deltas: 100% (27/27), done.
```

Рисунок 1 – Копирование проекта в локальную среду

Затем сменим директорию исполнения на вновь созданную папку "EEL6812_DeepFool_Project" проекта. Процесс смена директории показан на рисунке 2.

```
[2]  %cd EEL6812_DeepFool_Project/

     /content/EEL6812_DeepFool_Project
```

Рисунок 2 – Смена директории

Далее выполним импорт библиотек. Процесс импорта библиотек показан на рисунке 3.

```
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
```

Рисунок 3 – Импорт библиотек

Выполним импорт вспомогательных библиотек из локальных файлов проекта. Процесс импорт вспомогательных библиотек показан на рисунке 4.

```
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
```

Рисунок 4 – Импорт вспомогательных библиотек

Затем установим случайное рандомное значение в виде переменной rand_seed={"Порядковый номер ученика группы в Гугл-таблице"}. Установка случайного значения показана на рисунке 5.

```
rand_seed = 20 #вариант
```

Рисунок 5 – Установка значения для генератора случайных чисел

Установим указанное значение для np.random.seed и torch.manual_seed . Процесс установки указанного значения показан на рисунке 6.

```
[6]  np.random.seed(rand_seed)
     torch.manual_seed(rand_seed)

     <torch._C.Generator at 0x7cdec0394bf0>
```

Рисунок 6 – Настройка генераторов случайных чисел

7. Используем в качестве устройства видеокарту (Среды выполнения--> Сменить среду выполнения --> T4 GPU) . Выбор среды выполнения показан на рисунке 7.

```
import torch
device = torch.device("cuda")
if device.type == 'cuda':
  torch.cuda.empty_cache()
```

Рисунок 7 – Выбор среды выполнения

Теперь загрузим датасет MNIST с параметрами . Загрузка датасета MNIST с параметрами показана на рисунке 8.

```
mnist_mean = 0.5
mnist_std = 0.5
mnist_dim = 28
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)
```

Рисунок 8 – Загрузка датасета MNIST

Затаем загрузим датасет CIFAR-10 с параметрами. Загрузка датасета CIFAR-10 с параметрами показана на рисунке 9.

```
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']
```

Рисунок 9 – Загрузка датасета CIFAR-10

Выполним настройку и загрузку DataLoader . Настройка и загрузка DataLoader показана на рисунке 10.

```
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)
```

Рисунок 10 – Настройка и загрузка DataLoader

Загрузим и оценим стойкость модели LeNet к FGSM и DeepFool атакам. Загрузка и оценка стойкости модели LeNet к FGSM и DeepFool атакам показана на рисунке 11.

```
fgsm_eps = 0.6
batch = 64
num_classes = 10
overshoot = 0.02
max_iter = 50
deep_args = [batch, num_classes, overshoot, max_iter]
model = LeNet_MNIST().to(device)
model.load_state_dict(torch.load('weights/clean/mnist_lenet.pth',\
map_location=torch.device('cpu')))
evaluate_attack('mnist_lenet_fgsm.csv', 'results', device, model,\
mnist_loader_test, mnist_min, mnist_max, fgsm_eps,is_fgsm=True)
print('')
evaluate_attack('mnist_lenet_deepfool.csv', 'results', device, model,\
mnist_loader_test, mnist_min, mnist_max, deep_args,is_fgsm=False)
if device.type == 'cuda':
  torch.cuda.empty_cache()
```

```
FGSM Test Error : 87.89%
FGSM Robustness : 4.58e-01
FGSM Time (All Images) : 0.29 s
FGSM Time (Per Image) : 28.86 us

DeepFool Test Error : 98.74%
DeepFool Robustness : 9.64e-02
DeepFool Time (All Images) : 193.32 s
DeepFool Time (Per Image) : 19.33 ms
```

Рисунок 11 – Загрузка и оценка стойкости модели LeNet к атакам

После этого загрузим и оценим стойкость модели FC к FGSM и DeepFool атакам . Загрузка и оценка стойкости модели FC к FGSM и DeepFool атакам показана на рисунке 12.

```
fgsm_eps = 0.2
model = FC_500_150().to(device)
model.load_state_dict(torch.load('weights/clean/mnist_fc.pth',\
map_location=torch.device('cpu')))
evaluate_attack('mnist_fc_fgsm.csv', 'results', device, model,\
mnist_loader_test, mnist_min, mnist_max, fgsm_eps,is_fgsm=True)
print('')
evaluate_attack('mnist_fc_deepfool.csv', 'results', device, model,\
mnist_loader_test, mnist_min, mnist_max, deep_args,is_fgsm=False)
if device.type == 'cuda':
  torch.cuda.empty_cache()
```

```
FGSM Test Error : 87.08%
FGSM Robustness : 1.56e-01
FGSM Time (All Images) : 0.15 s
FGSM Time (Per Image) : 14.99 us

DeepFool Test Error : 97.92%
DeepFool Robustness : 6.78e-02
DeepFool Time (All Images) : 141.81 s
DeepFool Time (Per Image) : 14.18 ms
```

Рисунок 12 – Загрузка и оценка стойкости модели FC к атакам

Теперь выполним оценку атакующих примеров для сетей. Процесс оценки атакующих примеров для сетей показан на рисунках 13-20.

```
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()
```

Рисунок 13 – Загрузка модели LeNet на датасете MNIST

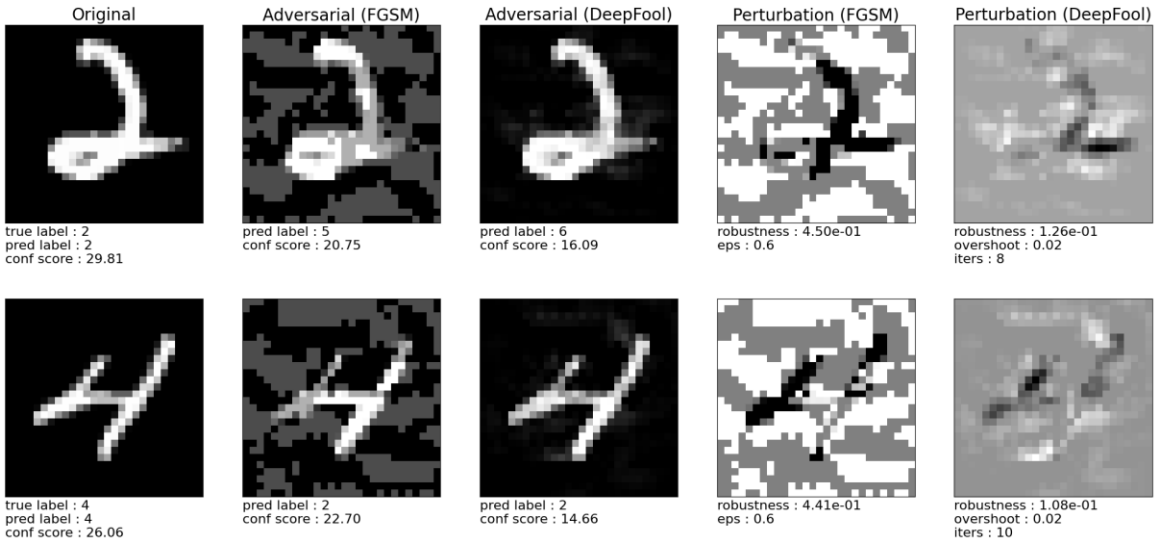

Рисунок 14 – Оценка атакующих примеров для LeNet на MNIST

```
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()
```

Рисунок 15 – Загрузка модели FC на датасете MNIST

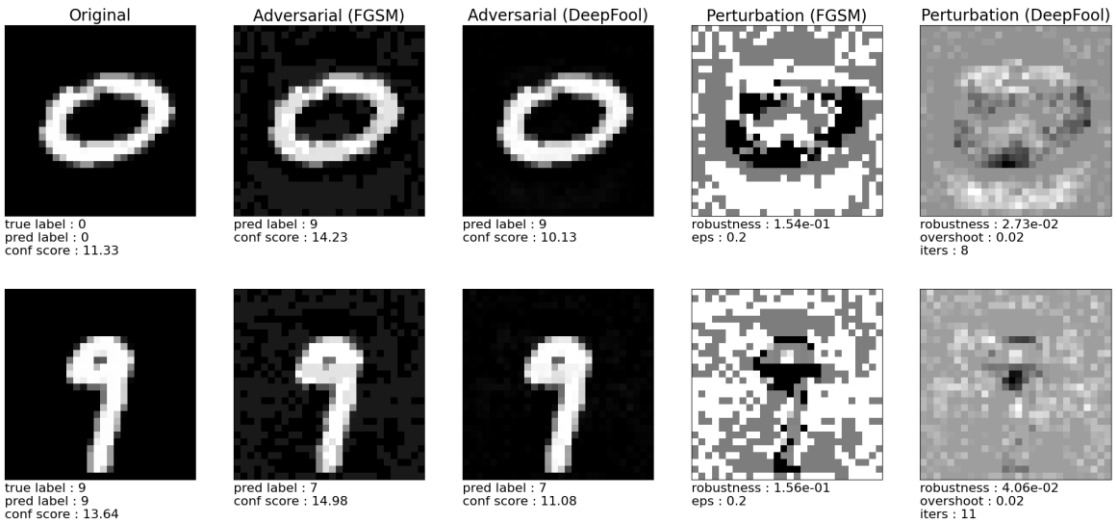

Рисунок 16 – Оценка атакующих примеров для FC на MNIST

```
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()
```

Рисунок 17 – Загрузка модели NiN на датасете CIFAR-10

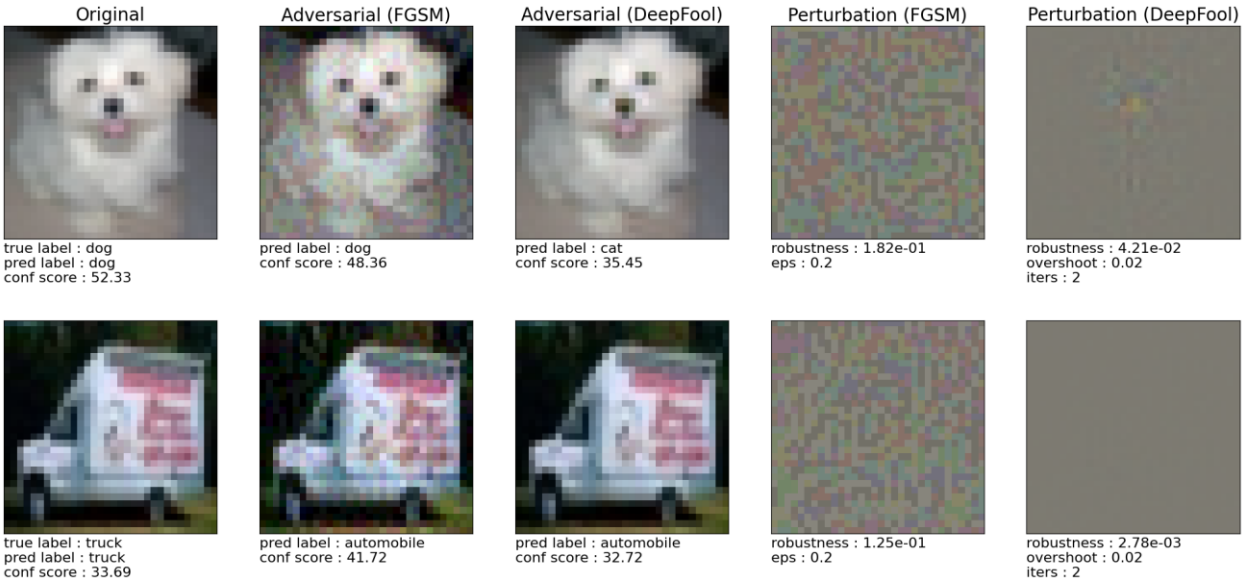

Рисунок 18 – Оценка атакующих примеров для NiN на CIFAR-10

```
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()
```

Рисунок 19 – Загрузка модели LeNet на датасете CIFAR-10

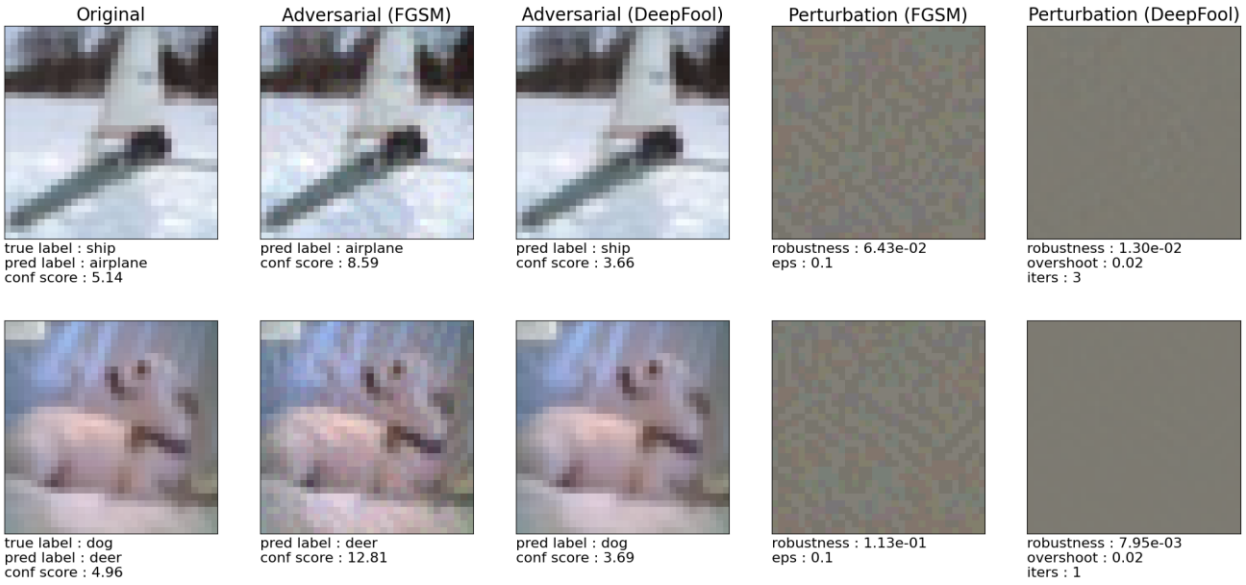

Рисунок 20 – Оценка атакующих примеров для LeNet на CIFAR-10

После оценки атакующих примеров для сетей проведем эксперимент с перебором значений параметра fgsm_eps=(0.001, 0.02, 0.5, 0.9, 10) и оценим влияние параметра eps для сетей FC и LeNet на датасете MNIST, NiN и LeNet на датасете CIFAR.

Зададим массив значений, который будем перебирать . Массив значений показан на рисунке 21.

```
fgsm_eps = [0.001, 0.02, 0.5, 0.9, 10]
```

Рисунок 21 – Массив значений

Приступим к оценке FC на MNIST. Создадим цикл для перебора параметра для FC на MNIST. Создание цикла для перебора параметра для FC на MNIST показано на рисунке 22.

```
for eps in fgsm_eps:
  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, 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()
```

Рисунок 22 – Цикл перебора eps для FC на MNIST

Результаты работы цикла представлены на рисунках 23-27.

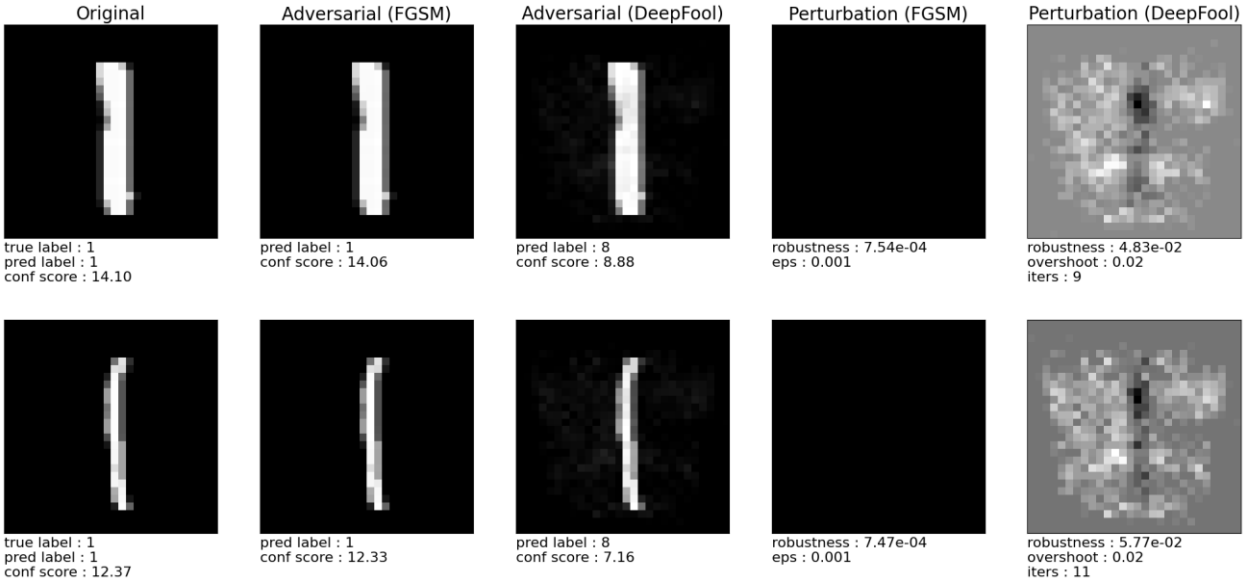

Рисунок 23 – Оценка атакующих примеров для FC на MNIST при eps равном 0.001

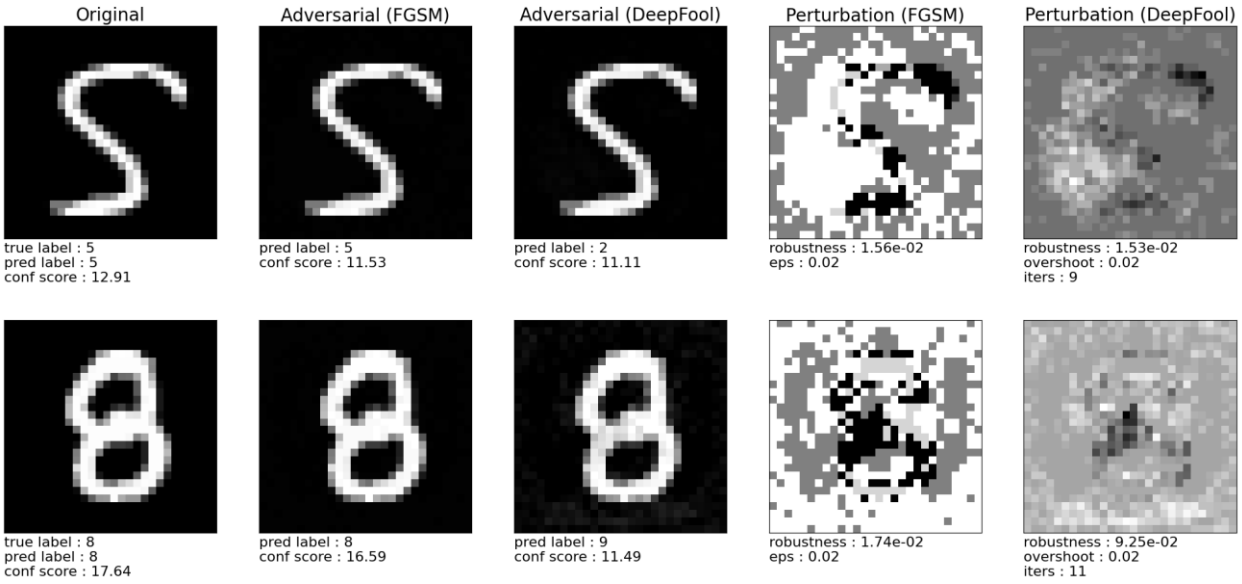

Рисунок 24 – Оценка атакующих примеров для FC на MNIST при eps равном 0.02

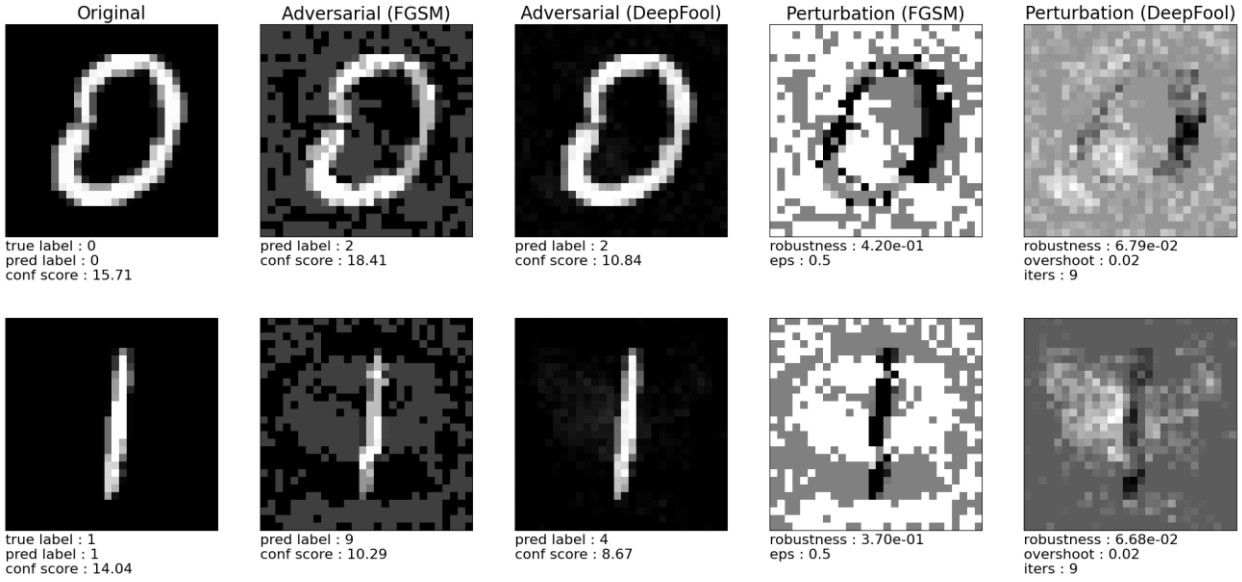

Рисунок 25 – Оценка атакующих примеров для FC на MNIST при eps равном 0.5

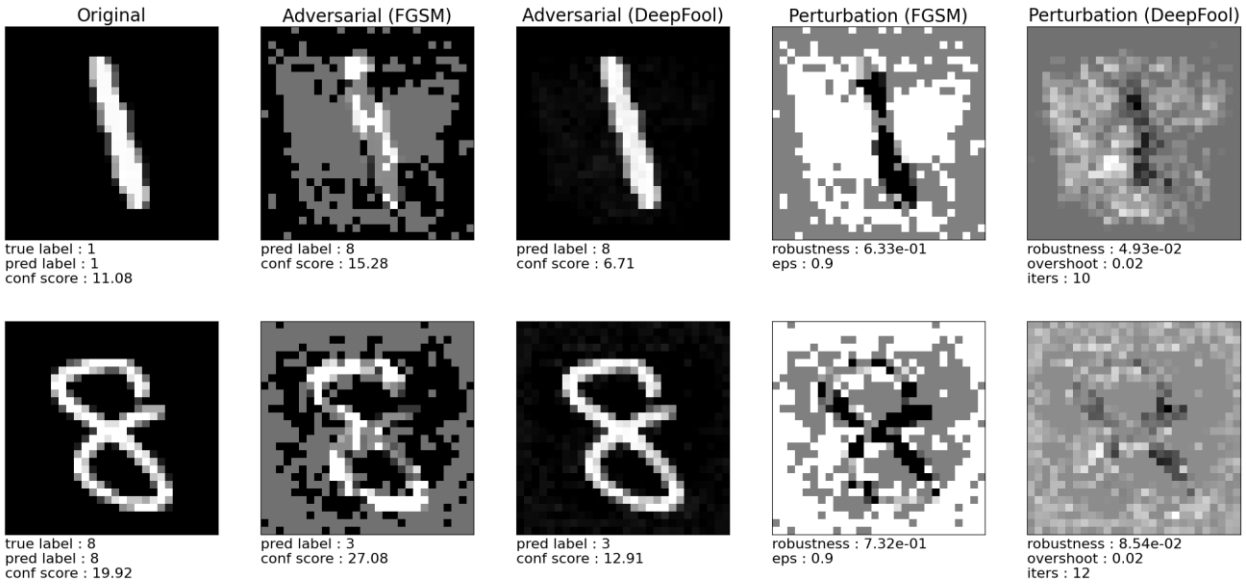

Рисунок 26 – Оценка атакующих примеров для FC на MNIST при eps равном 0.9

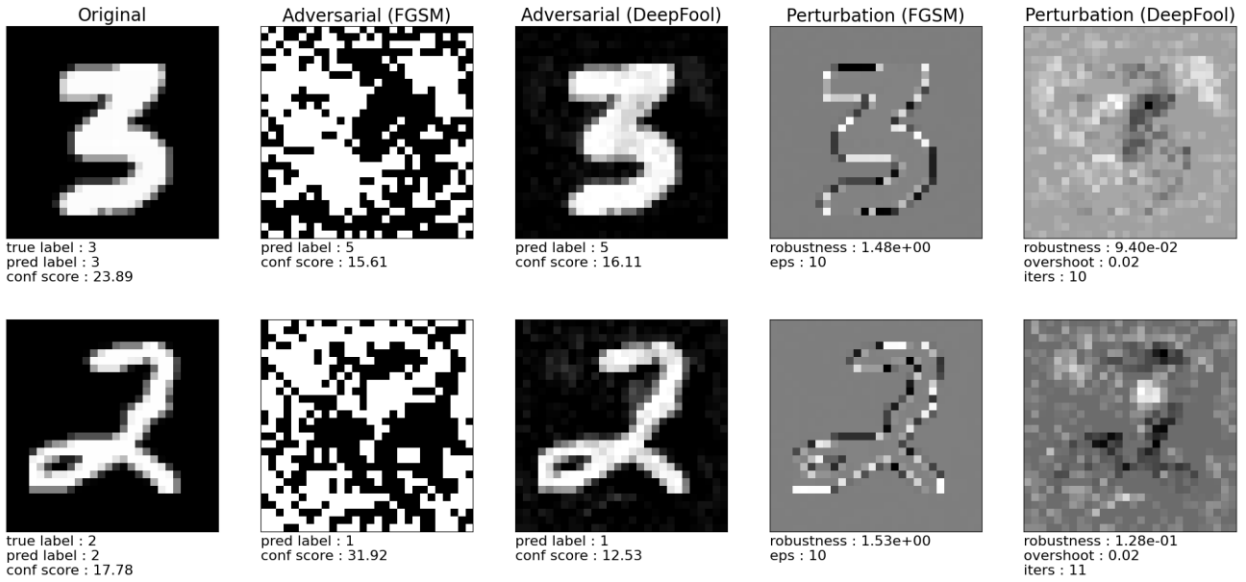

Рисунок 27 – Оценка атакующих примеров для FC на MNIST при eps равном 10

Приступим к оценке LeNet на MNIST. Создадим цикл для перебора параметра для LeNet на MNIST. Создание цикла для перебора параметра для LeNet на MNIST показано на рисунке 28.

```
for eps in fgsm_eps:
  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, 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()
```

Рисунок 28 – Цикл перебора eps для LeNet на MNIST

Результаты работы цикла при различных eps представлены на рисунках 29-33.

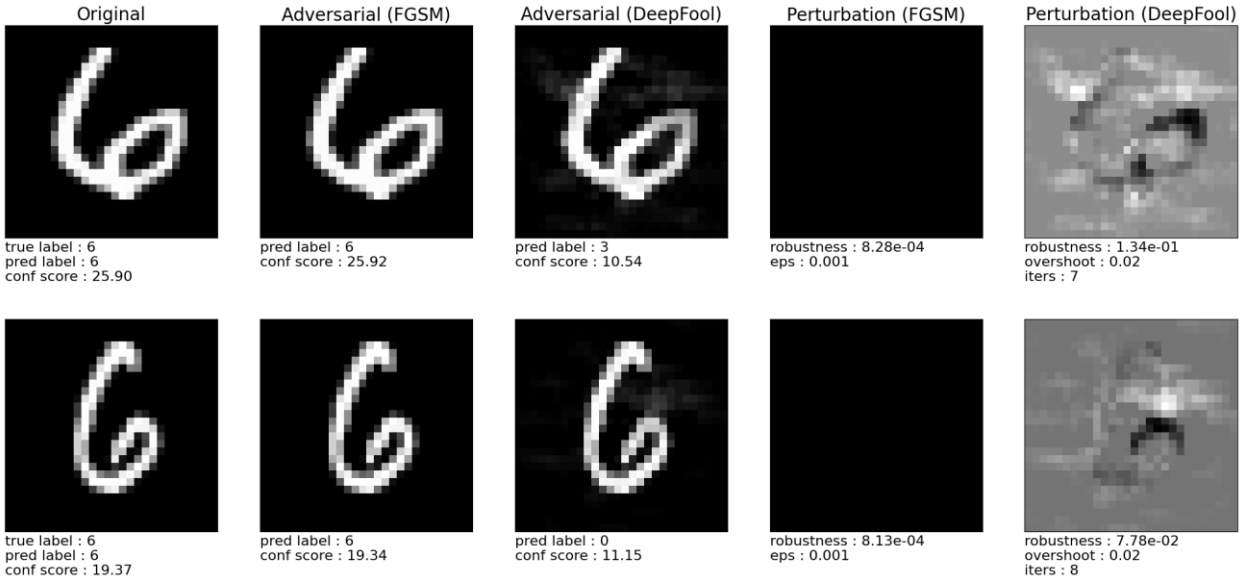

Рисунок 29 – Оценка атакующих примеров для LeNet на MNIST при eps равном 0.001

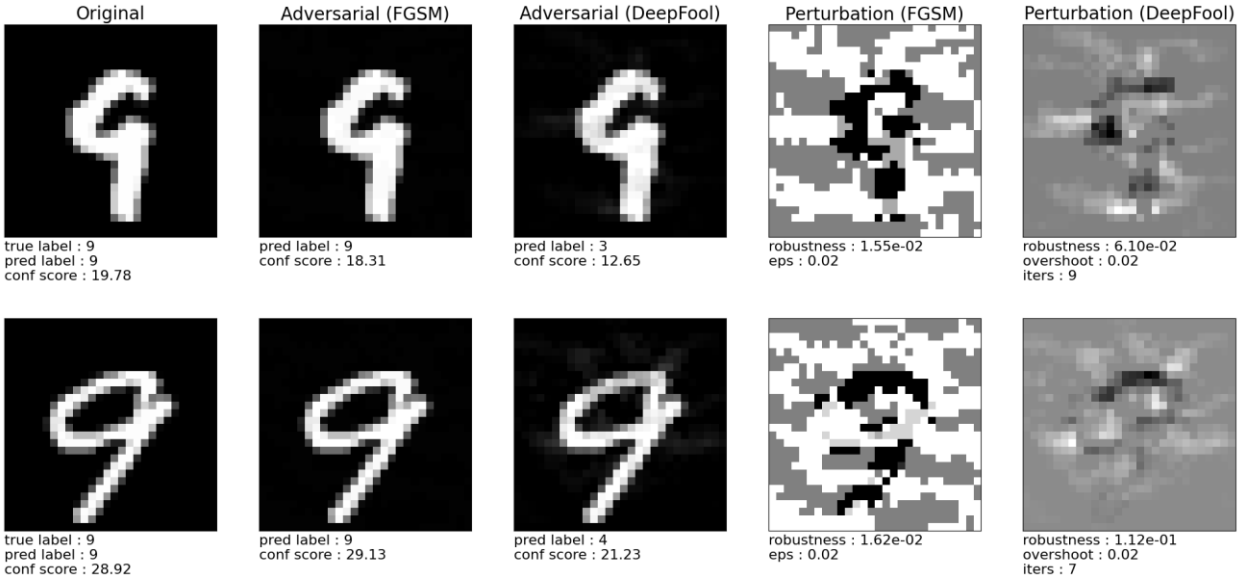

Рисунок 30 – Оценка атакующих примеров для LeNet на MNIST при eps равном 0.02

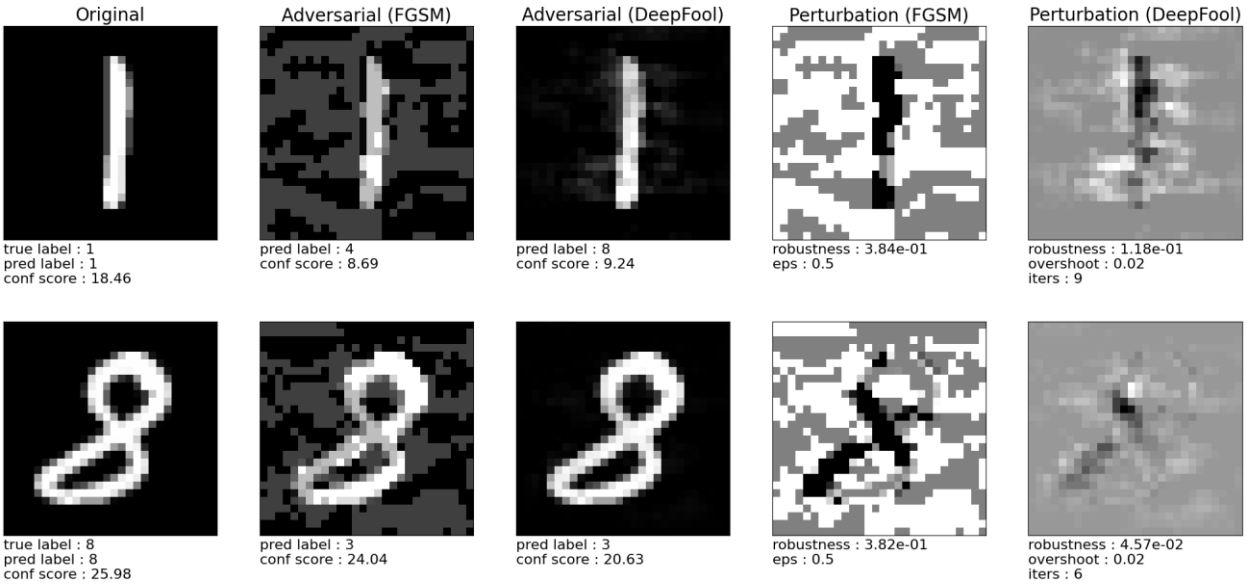

Рисунок 31 – Оценка атакующих примеров для LeNet на MNIST при eps равном 0.5

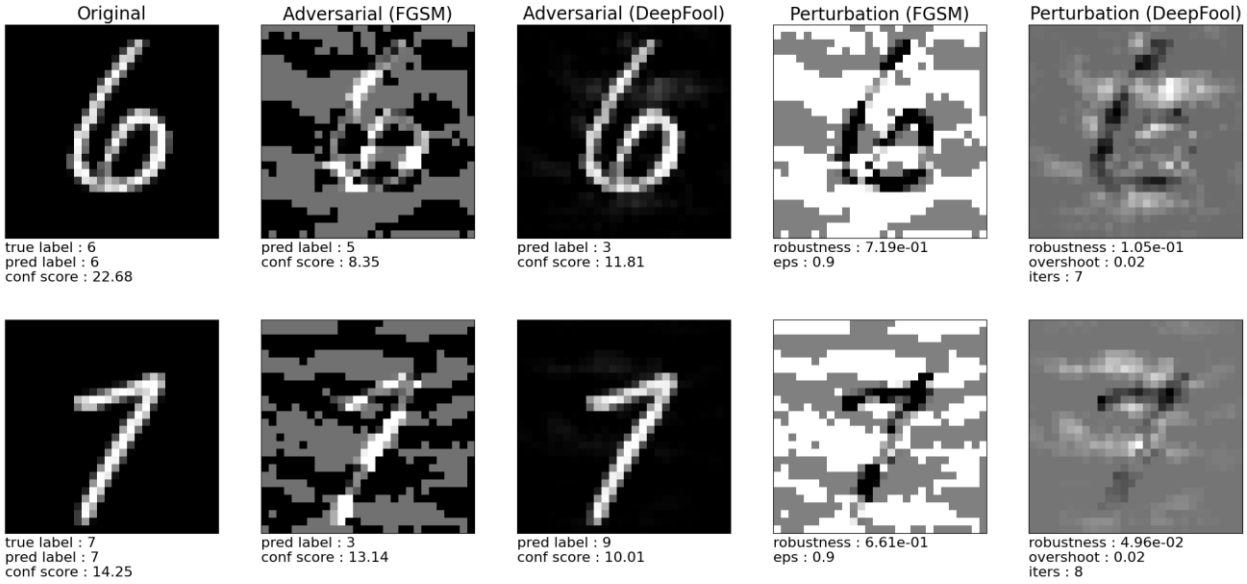

Рисунок 32 – Оценка атакующих примеров для LeNet на MNIST при eps равном 0.9

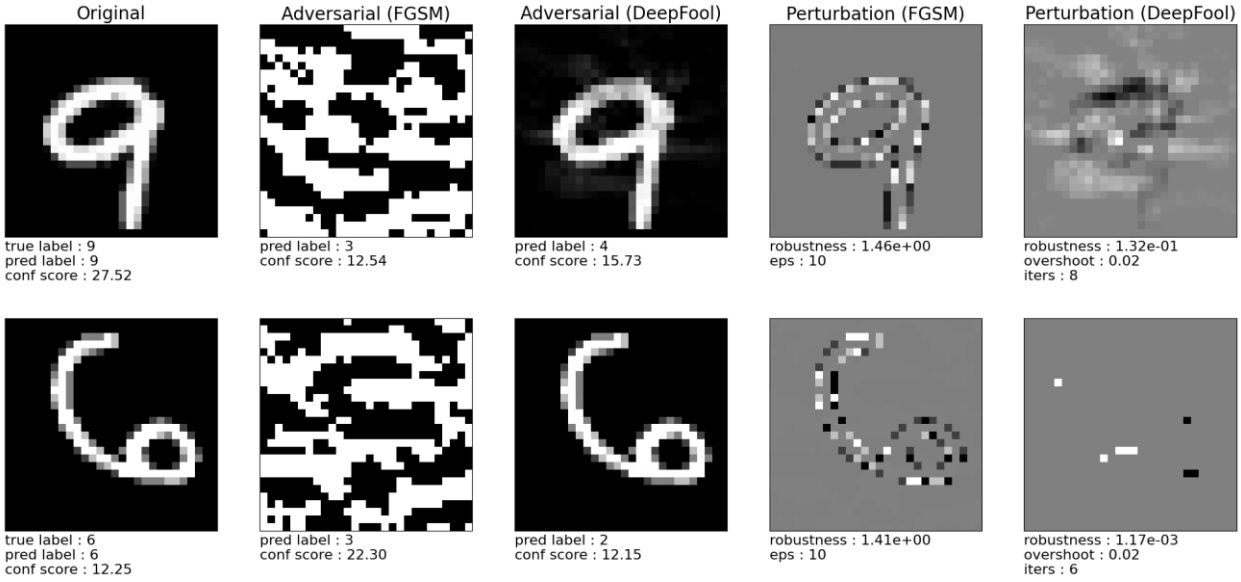

Рисунок 33 – Оценка атакующих примеров для LeNet на MNIST при eps равном 10

Приступим к оценке NiN на CIFAR-10. Создадим цикл для перебора параметра для NiN на CIFAR-10. Создание цикла для перебора параметра для NiN на CIFAR-10показано на рисунке 34.

```
for eps in fgsm_eps:
  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, 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()
```

Рисунок 34 – Цикл перебора eps для NiN на CIFAR-10

Результаты работы цикла при различных значениях eps представлены на рисунках 35-39.

Рисунок 35 – Оценка атакующих примеров для NiN на CIFAR-10 при eps равном 0.001

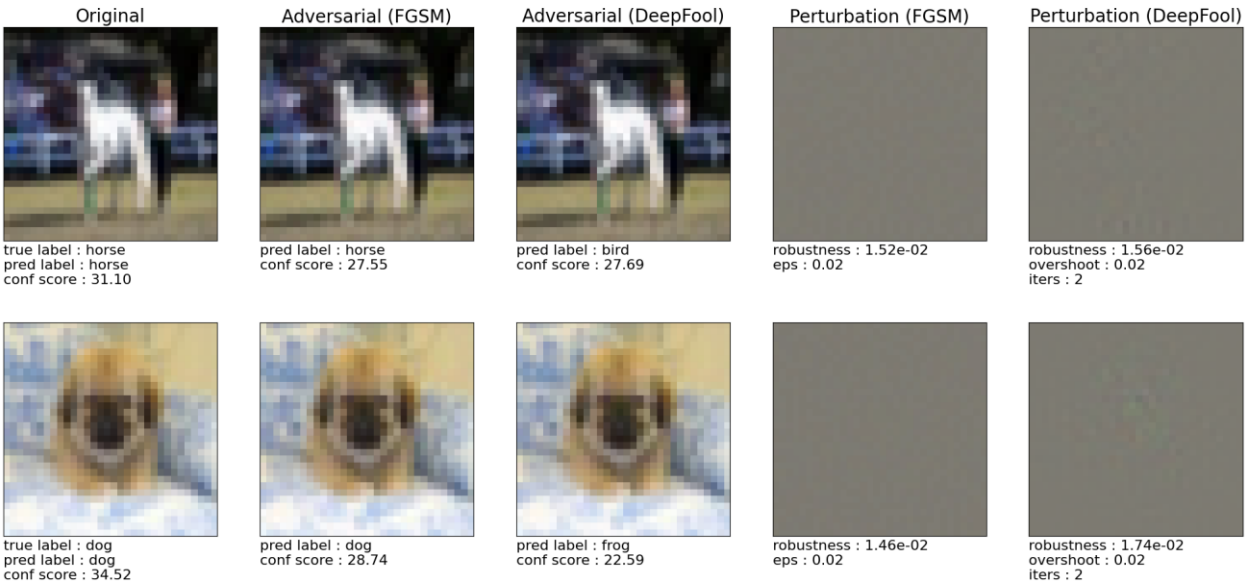

Рисунок 36 – Оценка атакующих примеров для NiN на CIFAR-10 при eps равном 0.02

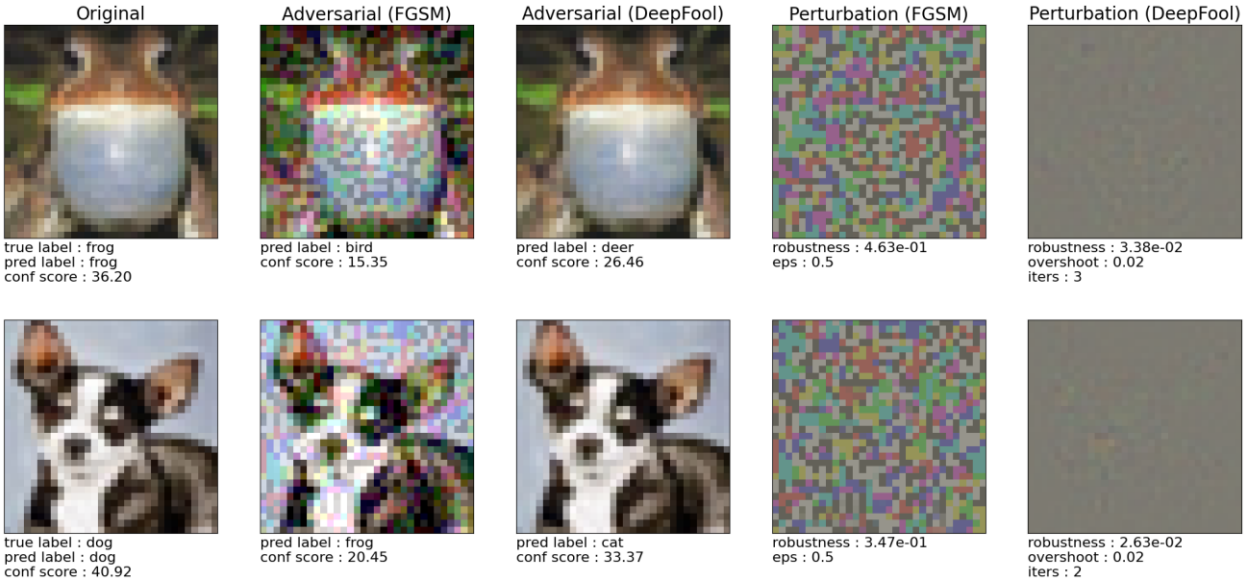

Рисунок 37 – Оценка атакующих примеров для NiN на CIFAR-10 при eps равном 0.5

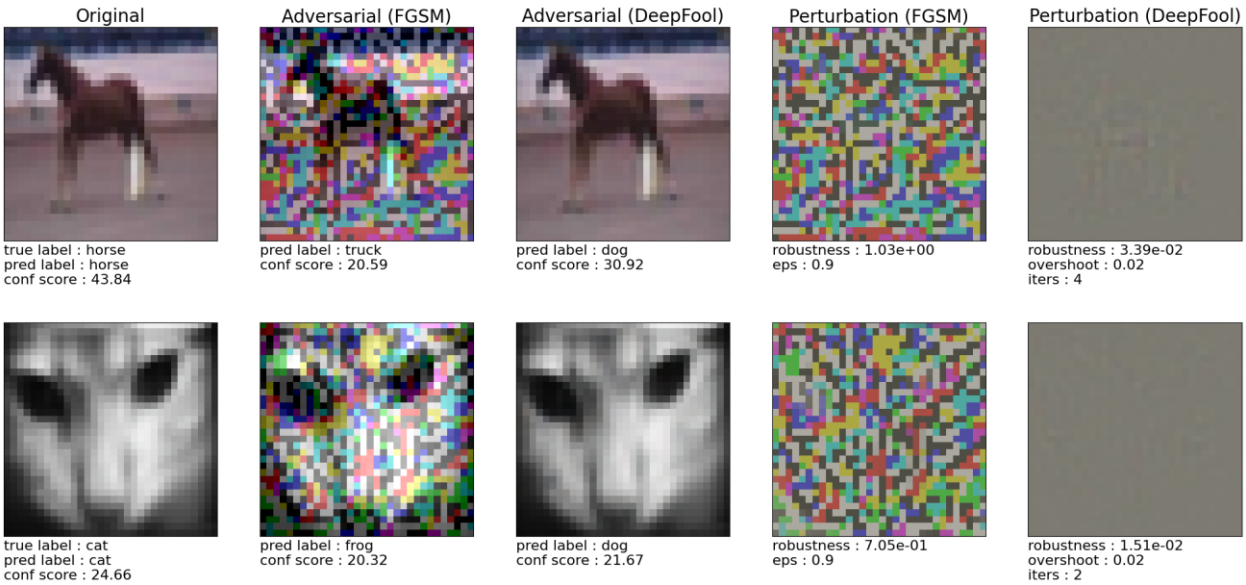

Рисунок 38 – Оценка атакующих примеров для NiN на CIFAR-10 при eps равном 0.9

Рисунок 39 – Оценка атакующих примеров для NiN на CIFAR-10 при eps равном 10

Приступим к оценке LeNet на CIFAR-10. Создадим цикл для перебора параметра для LeNet на CIFAR-10. Создание цикла для перебора параметра для LeNet на CIFAR-10показано на рисунке 40.

```
for eps in fgsm_eps:
  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, 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()
```

Рисунок 40 – Цикл перебора eps для LeNet на CIFAR-10

Результаты работы цикла с различными eps представлены на рисунках 41-45.

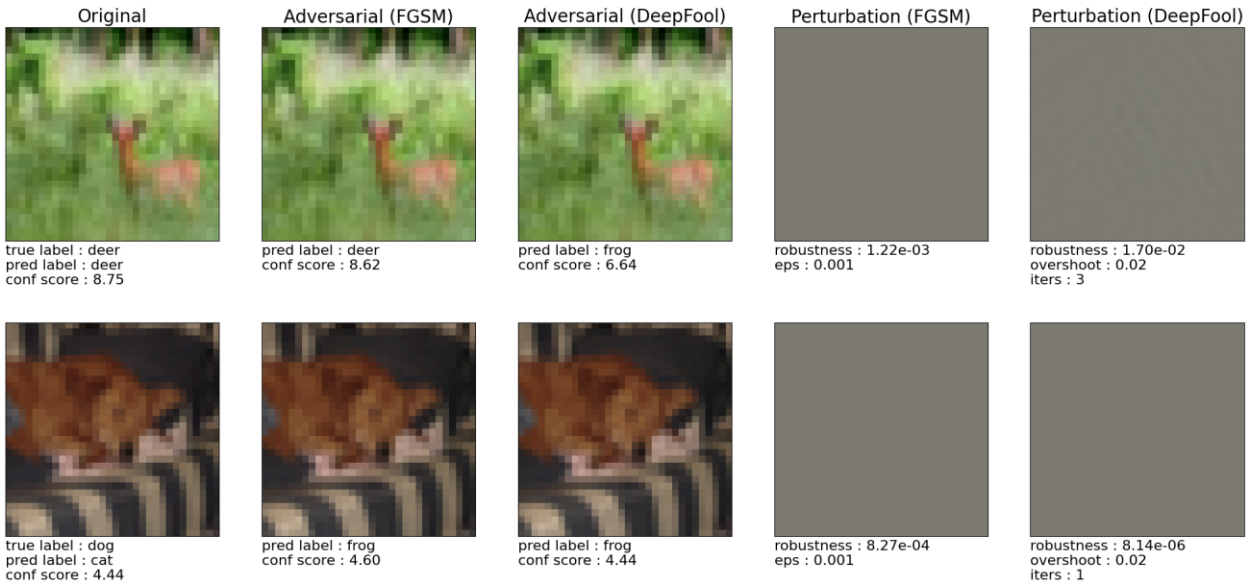

Рисунок 41 – Оценка атакующих примеров для LeNet на CIFAR-10 при eps равном 0.001

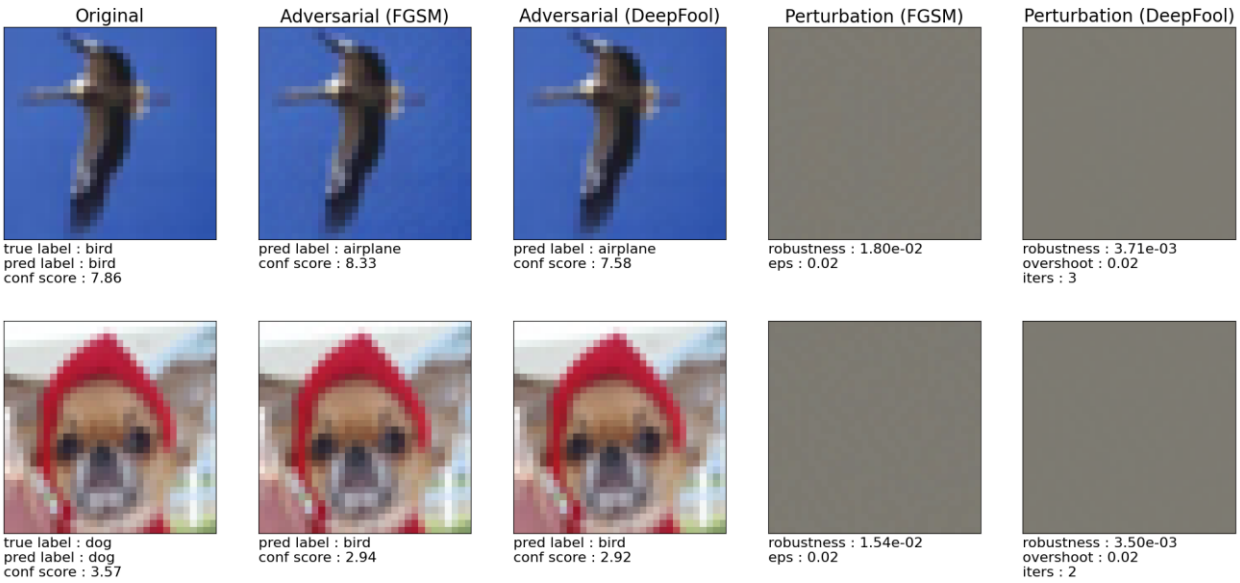

Рисунок 42 – Оценка атакующих примеров для LeNet на CIFAR-10 при eps равном 0.02

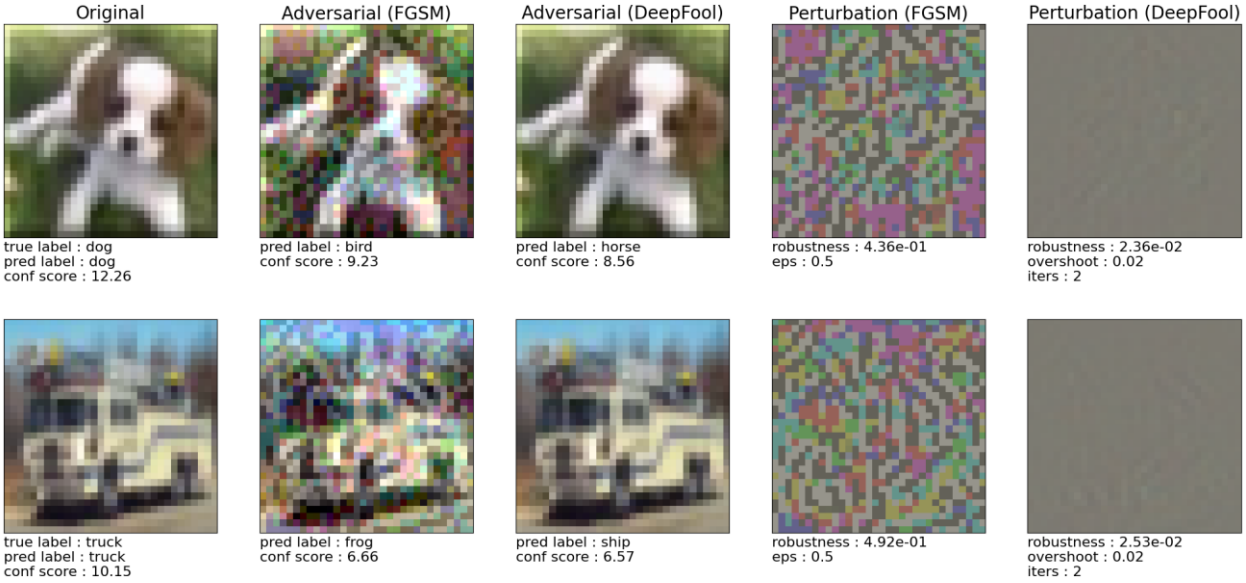

Рисунок 43 – Оценка атакующих примеров для LeNet на CIFAR-10 при eps равном 0.5

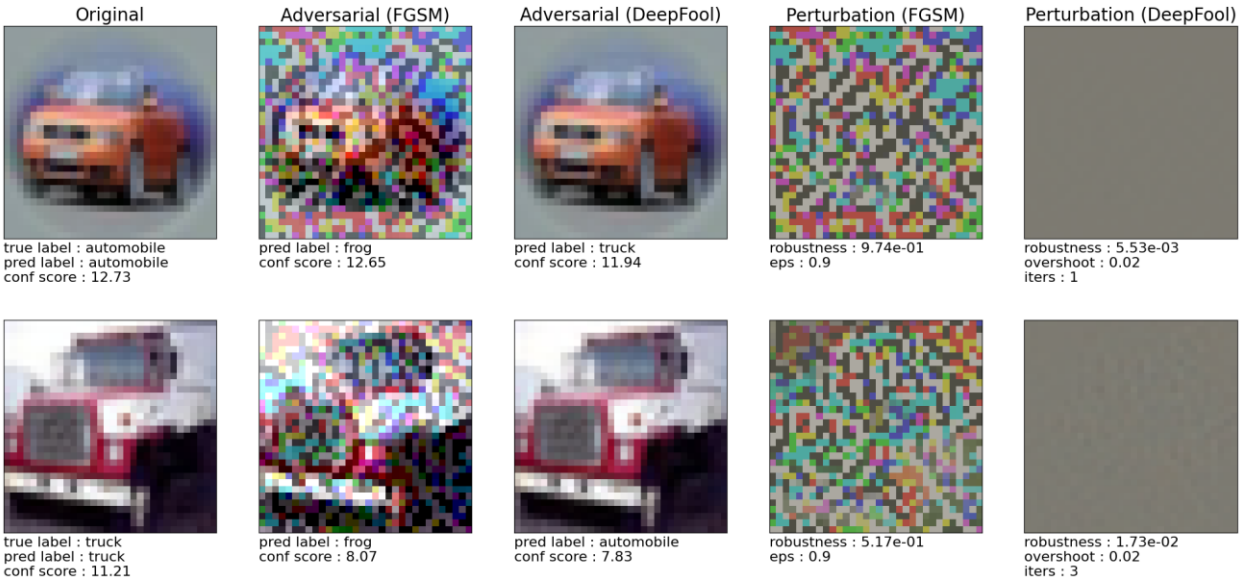

Рисунок 44 – Оценка атакующих примеров для LeNet на CIFAR-10 при eps равном 0.9

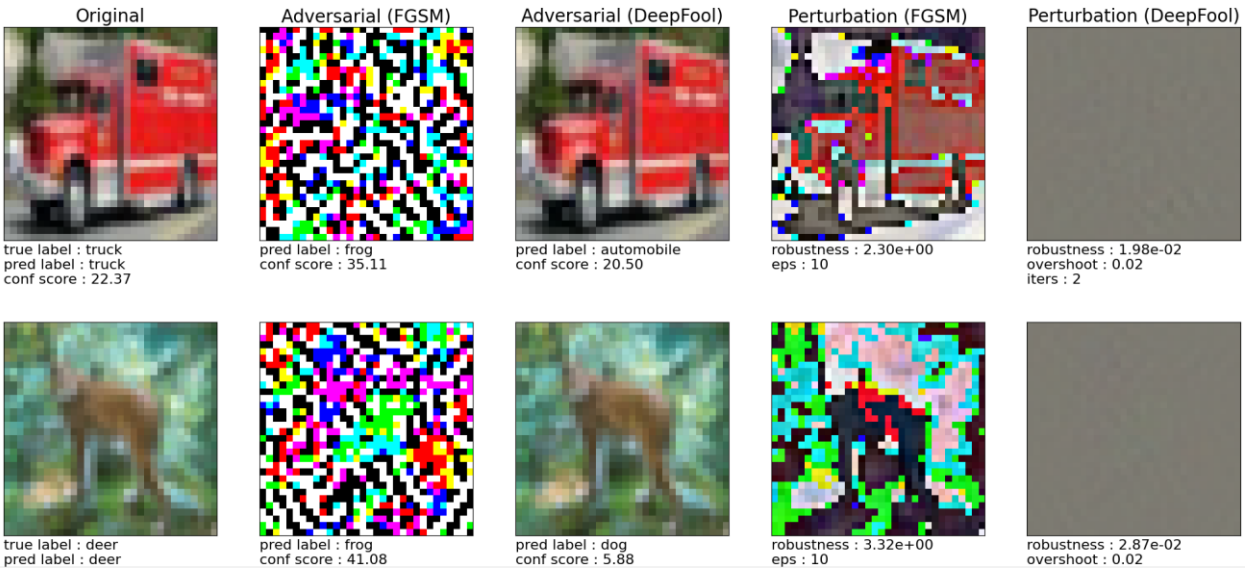

Рисунок 45 – Оценка атакующих примеров для LeNet на CIFAR-10 при eps равном 10

Заключение

По результатам эксперимента по перебору значения eps для атак на модели можно с уверенностью сказать, что влияние есть.

Была выявлена следующая закономерность: увеличение значения eps в FGSM искажает изображение таким образом, что не заметить этого просто невозможно. При этом слишком низкое значение eps может оказаться недостаточным для того, чтобы модель совершила ошибку при классификации. В случае с DeepFool изображение практически не отличается для человека при любом из протестированных значений eps, а вот модель практически всегда ошибается.