```
Лабораторная работа №1

Буланов Андрей Алексеевич, ББМО-01-23
```

Импорт библиотек

```
!git clone https://github.com/ewatson2/EEL6812_DeepFool_Project
%cd ./EEL6812_DeepFool_Project
```

```
Cloning into 'EEL6812_DeepFool_Project'...
remote: Enumerating objects: 96, done.ote: Counting objects: 100%
(3/3), done.ote: Compressing objects: 100% (2/2), done.ote: Total 96
(delta 2), reused 1 (delta 1), pack-reused 93 (from 1)
```

```
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
```

Импорт вспомогательных библиотек из локальных файлов проекта

```
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
```

Установим случайное рандомное значение в виде переменной rand_seed={"Порядковый номер ученика группы в Гугл-таблице"}, укажем значение для np.random.seed и torch.manual_seed Использовать в качестсве устройства видеокарту (Среды выполнения--> Сменить среду выполнения --> T4 GPU) image.png

```
rand_seed = 8
np.random.seed(rand_seed)
torch.manual_seed(rand_seed)

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

Загрузка датасета MNIST

```
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)

cifar_classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-
ubyte.gz
Failed to download (trying next):
<urlopen error [Errno 111] Connection refused>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-
images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-
images-idx3-ubyte.gz to datasets/mnist/MNIST/raw/train-images-idx3-
ubyte.gz

100%|██████████| 9.91M/9.91M [00:00<00:00, 17.8MB/s]

Extracting datasets/mnist/MNIST/raw/train-images-idx3-ubyte.gz to
datasets/mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-
ubyte.gz
```

```
Failed to download (trying next):
<urlopen error [Errno 111] Connection refused>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-
labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-
labels-idx1-ubyte.gz to datasets/mnist/MNIST/raw/train-labels-idx1-
ubyte.gz

100%|██████████| 28.9k/28.9k [00:00<00:00, 486kB/s]

Extracting datasets/mnist/MNIST/raw/train-labels-idx1-ubyte.gz to
datasets/mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
<urlopen error [Errno 111] Connection refused>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-
idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-
idx3-ubyte.gz to datasets/mnist/MNIST/raw/t10k-images-idx3-ubyte.gz

100%|██████████| 1.65M/1.65M [00:00<00:00, 4.42MB/s]

Extracting datasets/mnist/MNIST/raw/t10k-images-idx3-ubyte.gz to
datasets/mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
<urlopen error [Errno 111] Connection refused>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-
idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-
idx1-ubyte.gz to datasets/mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz

100%|██████████| 4.54k/4.54k [00:00<00:00, 4.83MB/s]

Extracting datasets/mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz to
datasets/mnist/MNIST/raw
```

Загрузка датасета CIFAR-10

```
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)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
datasets/cifar-10/cifar-10-python.tar.gz

100%|██████████| 170M/170M [00:03<00:00, 49.1MB/s]

Extracting datasets/cifar-10/cifar-10-python.tar.gz to datasets/cifar-
10
Files already downloaded and verified
```

Выполнить настройку и загрузку DataLoader batch_size = 64 workers = 4

```
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)
```

```
/usr/local/lib/python3.11/dist-packages/torch/utils/data/
dataloader.py:617: UserWarning: This DataLoader will create 4 worker
processes in total. Our suggested max number of worker in current
system is 2, which is smaller than what this DataLoader is going to
create. Please be aware that excessive worker creation might get
DataLoader running slow or even freeze, lower the worker number to
avoid potential slowness/freeze if necessary.
  warnings.warn(
```

```
import os
train_model = True

epochs = 50
epochs_nin = 100

lr = 0.004
lr_nin = 0.01
lr_scale = 0.5

momentum = 0.9

print_step = 5

deep_batch_size = 64
deep_num_classes = 10
deep_overshoot = 0.02
deep_max_iters = 50

deep_args = [deep_batch_size, deep_num_classes,
             deep_overshoot, deep_max_iters]

if not os.path.isdir('weights/deepfool'):
    os.makedirs('weights/deepfool', exist_ok=True)

if not os.path.isdir('weights/fgsm'):
    os.makedirs('weights/fgsm', exist_ok=True)
```

Загрузить и оценить стойкость модели LeNet к FGSM и DeepFool атакам

```
fgsm_eps = 0.2
model = LeNet_MNIST().to(device)
model.load_state_dict(torch.load('weights/clean/mnist_lenet.pth',map_l
ocation=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

<ipython-input-9-95d4b6361cc1>:3: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.

model.load_state_dict(torch.load('weights/clean/mnist_lenet.pth',map_l
ocation=torch.device('cpu')))
```

Загрузить и оценить стойкость модели FC к FGSM и DeepFool атакам

```
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

<ipython-input-10-ac820cd184c1>:3: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load('weights/clean/mnist_fc.pth',
map_location=torch.device('cpu')))
```

```
# Загрузим и оценим стойкость модели LeNet к FGSM и DeepFool атакам на
основе датасета CIFAR-10
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()
```

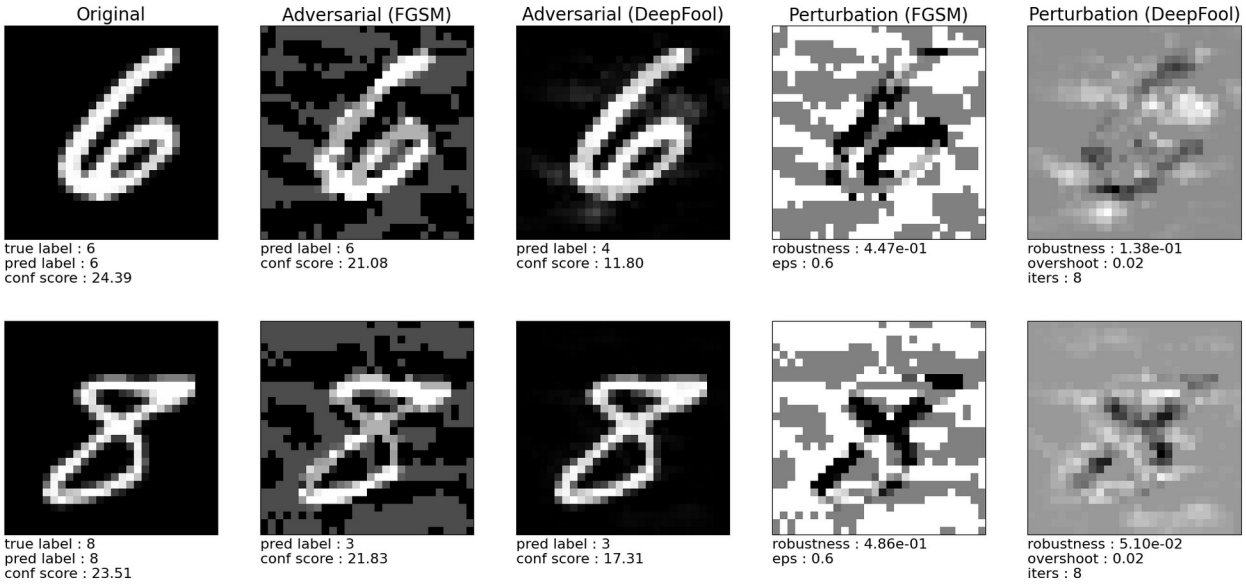
```
FGSM Test Error : 91.71%
FGSM Robustness : 8.90e-02
FGSM Time (All Images) : 0.40 s
FGSM Time (Per Image) : 40.08 us

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

<ipython-input-11-08045a679969>:4: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load('weights/clean/cifar_lenet.pth',
map_location=torch.device('cpu')))
```

```
# Выполним оценку атакующих примеров для сетей:

# LeNet на датасете MNIST
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 на датасете 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,
```

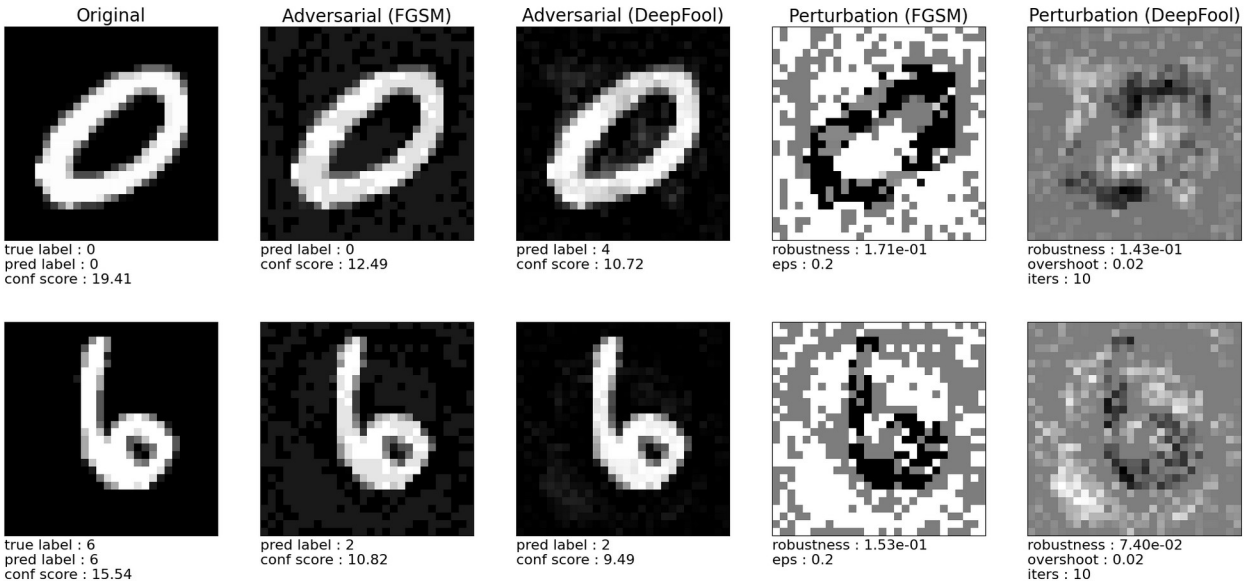
```
                has_labels=False, l2_norm=True, pert_scale=1.0,
fig_rows=2, fig_width=25, fig_height=11)

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

# Network-in-Network на датасете CIFAR-10
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 на датасете 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()
```

```
<ipython-input-12-d6d3cb22eb2b>:6: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load('weights/clean/mnist_lenet.pth'))
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py
:617: UserWarning: This DataLoader will create 4 worker processes in
total. Our suggested max number of worker in current system is 2,
which is smaller than what this DataLoader is going to create. Please
be aware that excessive worker creation might get DataLoader running
```

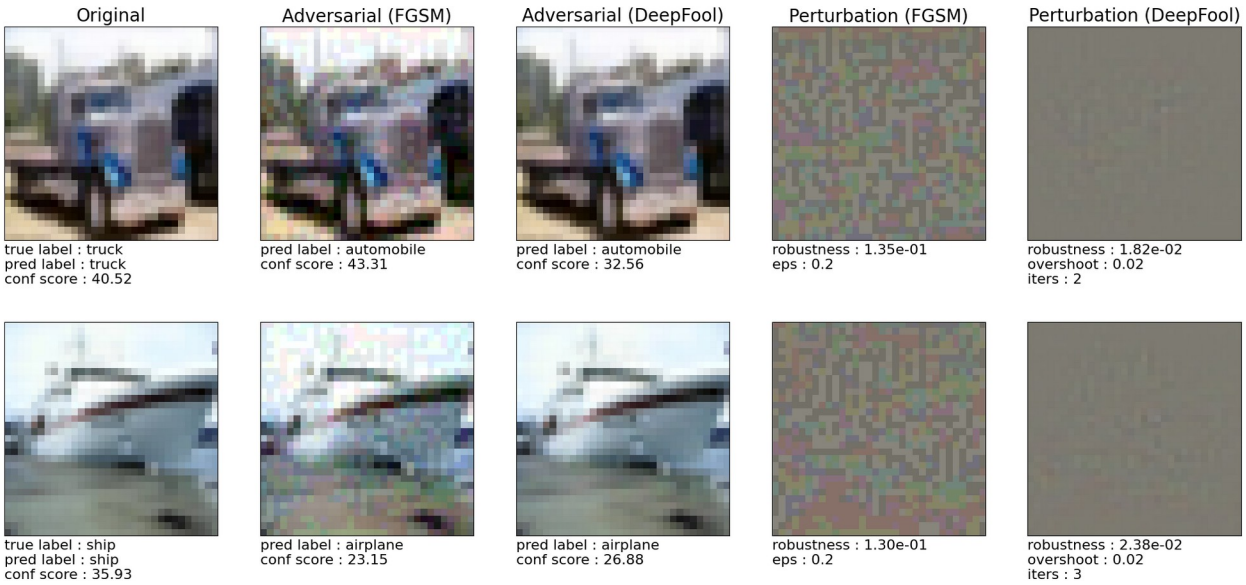
```
slow or even freeze, lower the worker number to avoid potential
slowness/freeze if necessary.
  warnings.warn(
```



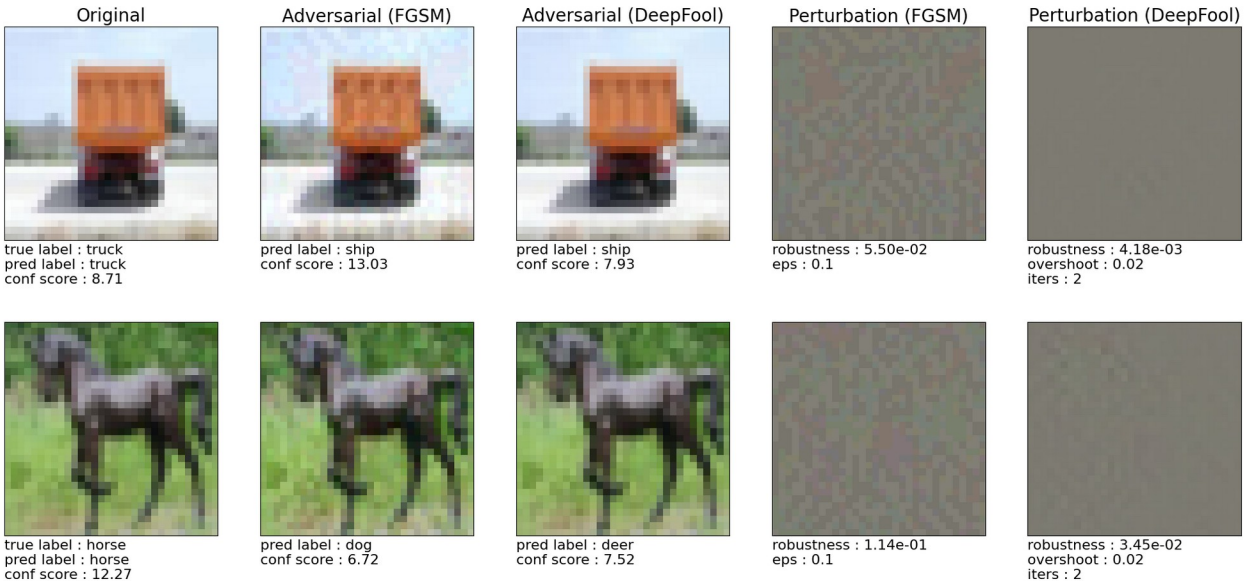

```
<ipython-input-12-d6d3cb22eb2b>:15: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load('weights/clean/mnist_fc.pth'))
```




```
<ipython-input-12-d6d3cb22eb2b>:24: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load('weights/clean/cifar_nin.pth'))
```

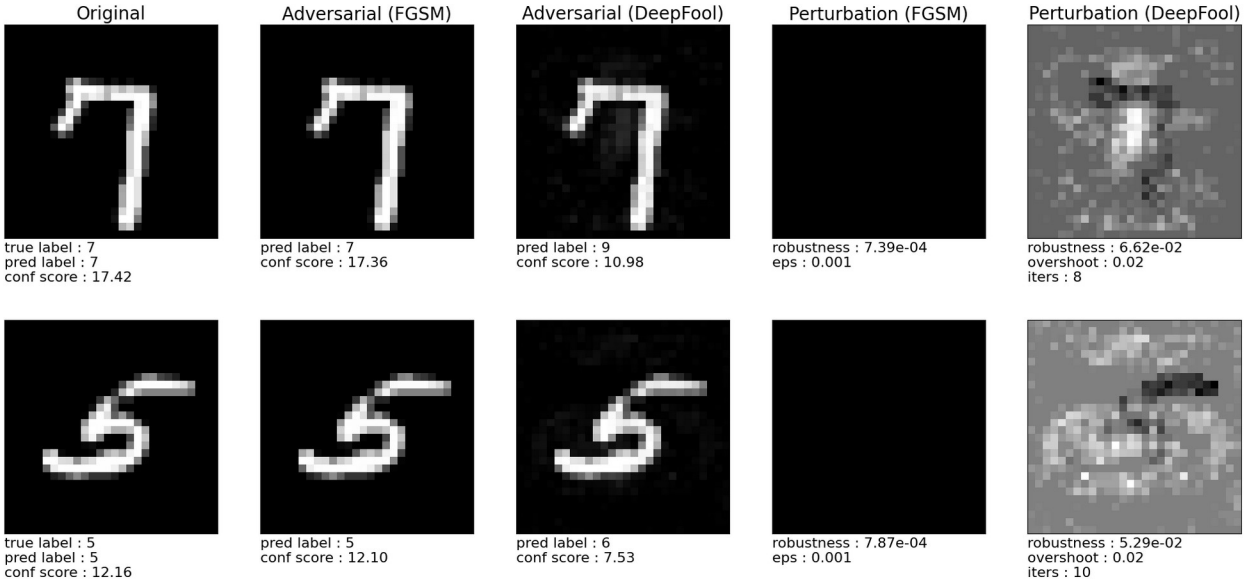



```
<ipython-input-12-d6d3cb22eb2b>:34: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load('weights/clean/cifar_lenet.pth'))
```

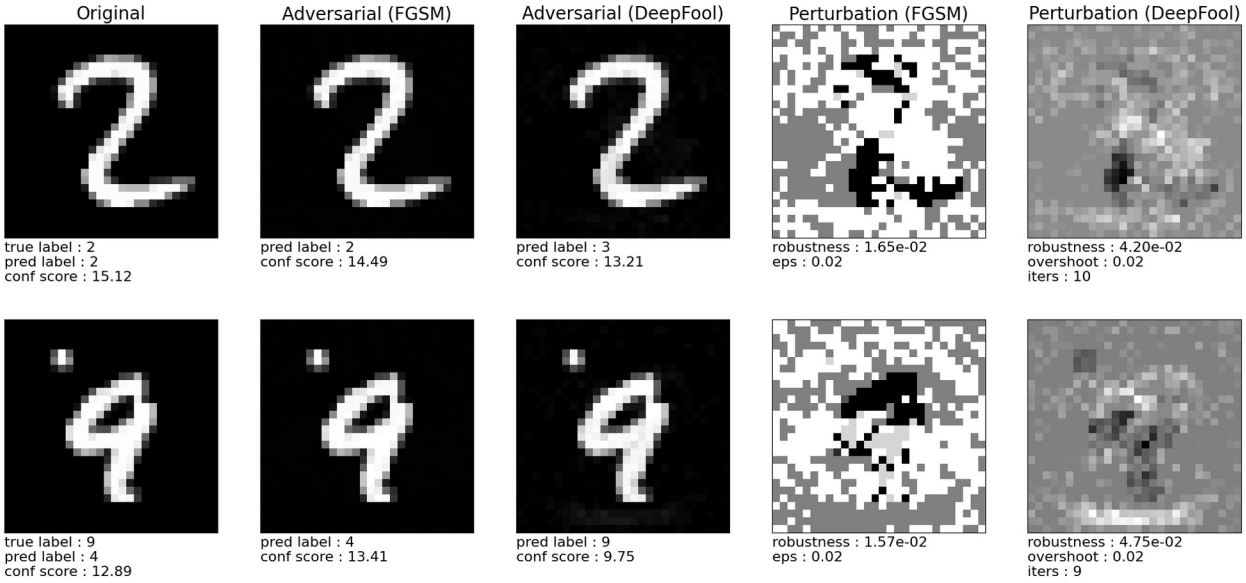



```
model = FC_500_150().to(device)
model.load_state_dict(torch.load('weights/clean/mnist_fc.pth'))

for fgsm_eps in [0.001, 0.02, 0.5, 0.9, 10]:
    print(f"results for fgsm_eps = {fgsm_eps}:")
    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()
```

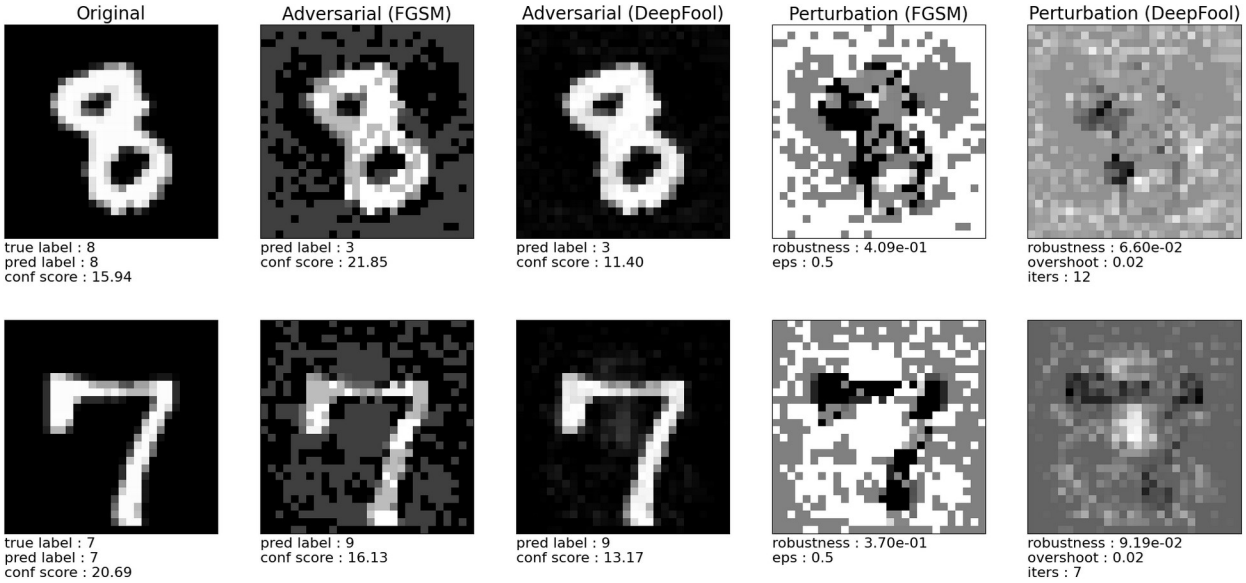
```
results for fgsm_eps = 0.001:

<ipython-input-13-79c7ad27ee30>:2: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load('weights/clean/mnist_fc.pth'))
```



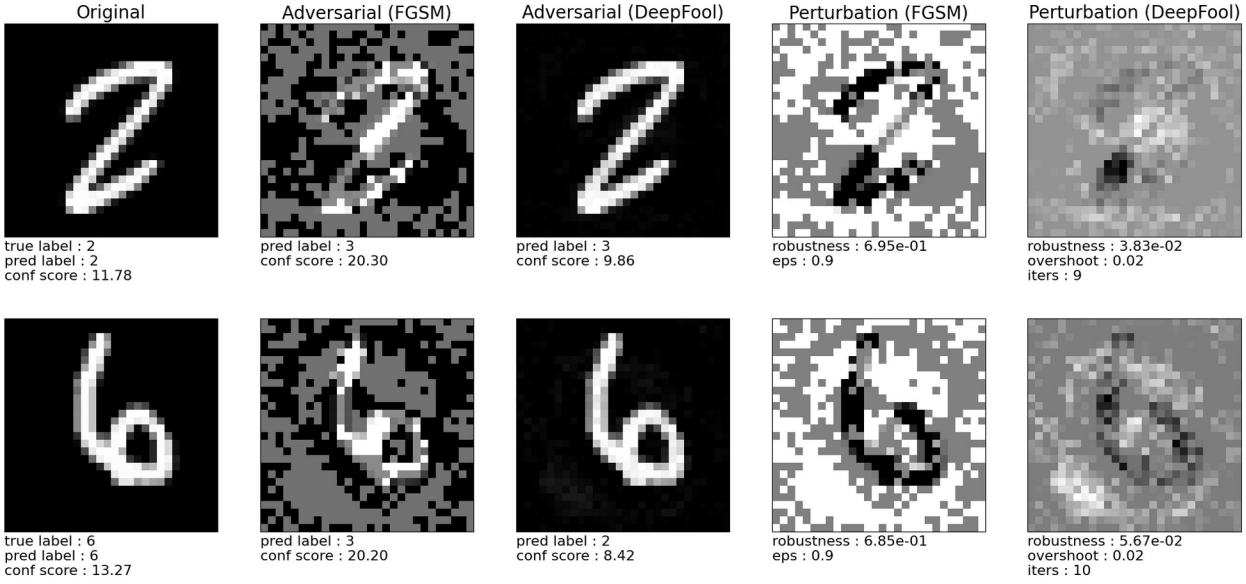

```
results for fgsm_eps = 0.02:
```



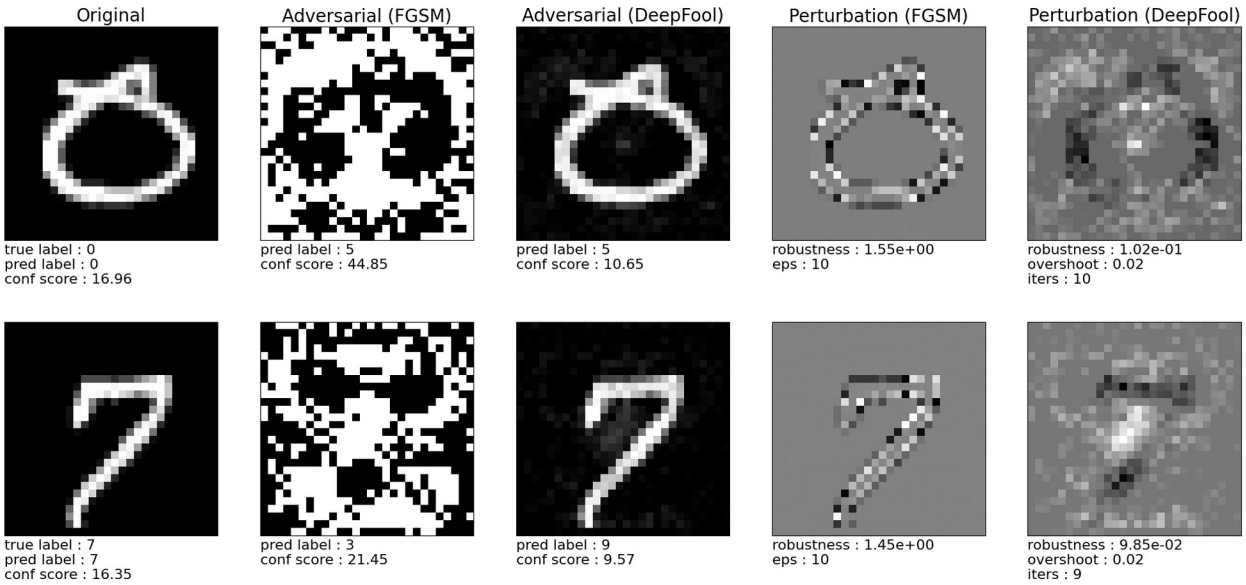

```
results for fgsm_eps = 0.5:
```




```
results for fgsm_eps = 0.9:
```

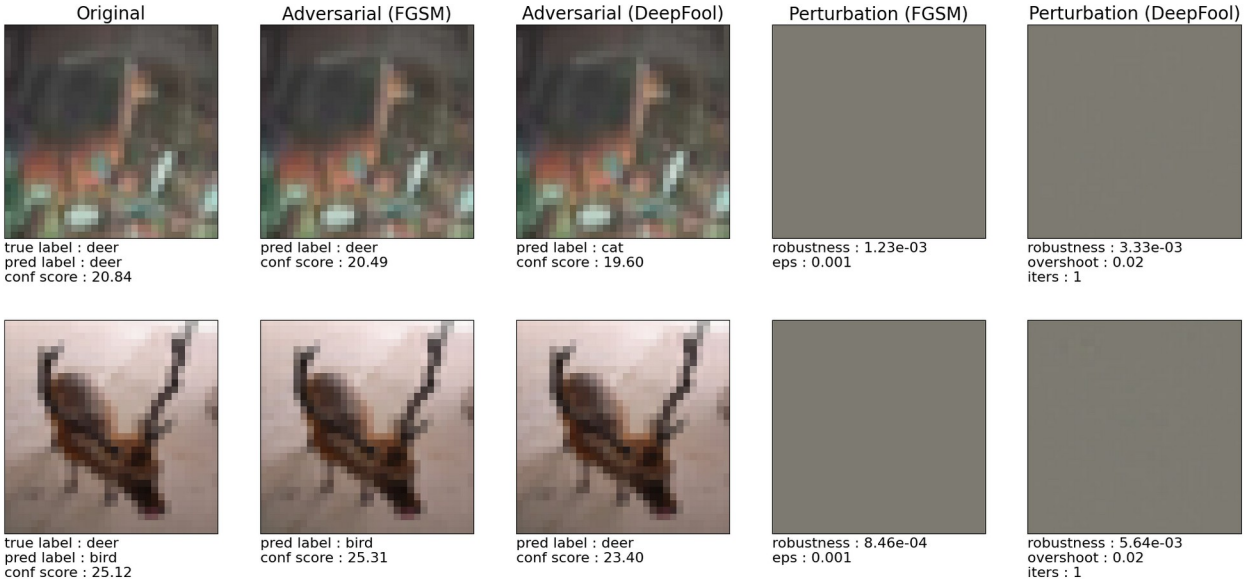



```
results for fgsm_eps = 10:
```

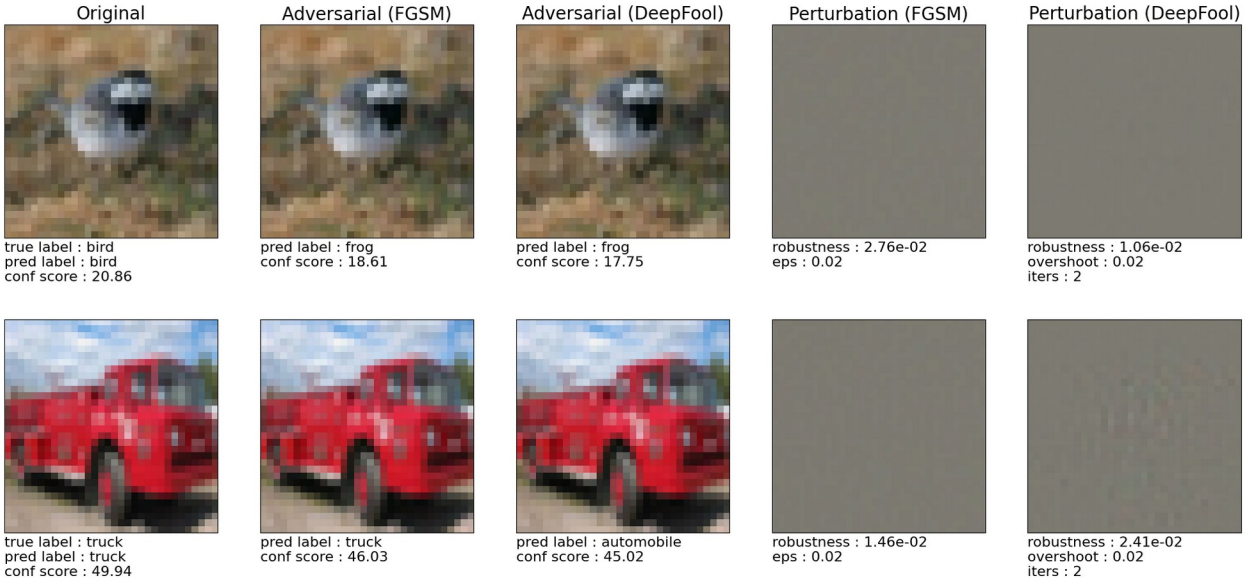



```
model = Net().to(device)
model.load_state_dict(torch.load('weights/clean/cifar_nin.pth'))

for fgsm_eps in [0.001, 0.02, 0.5, 0.9, 10]:
    print(f"results for fgsm_eps = {fgsm_eps}:")
    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()
```

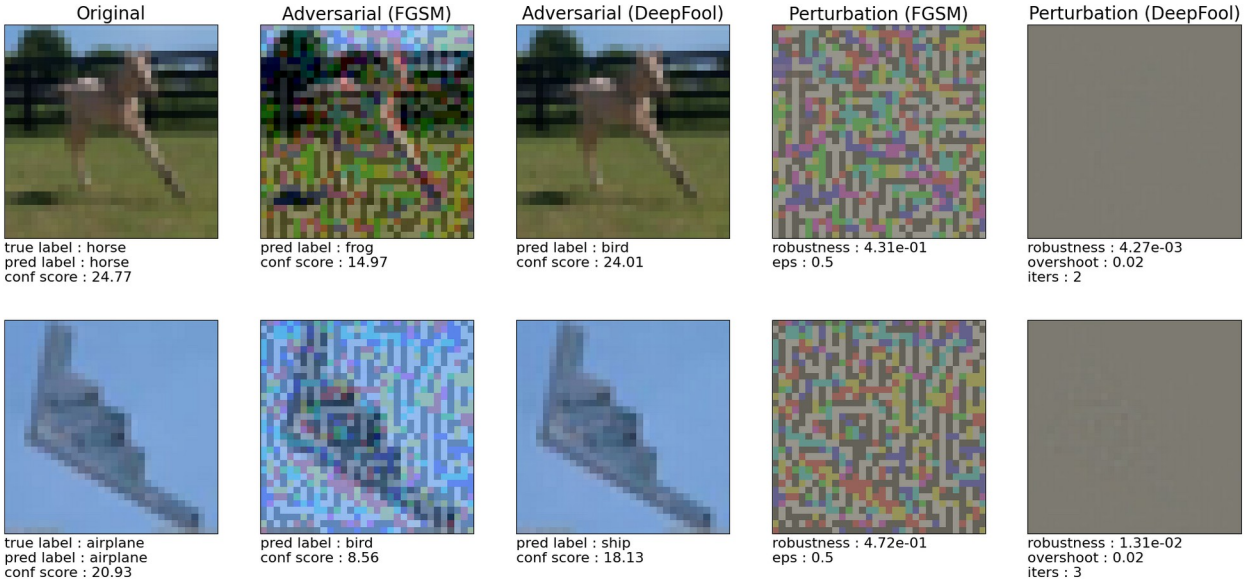
```
results for fgsm_eps = 0.001:

<ipython-input-14-c4f82a4e8058>:2: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load('weights/clean/cifar_nin.pth'))
```



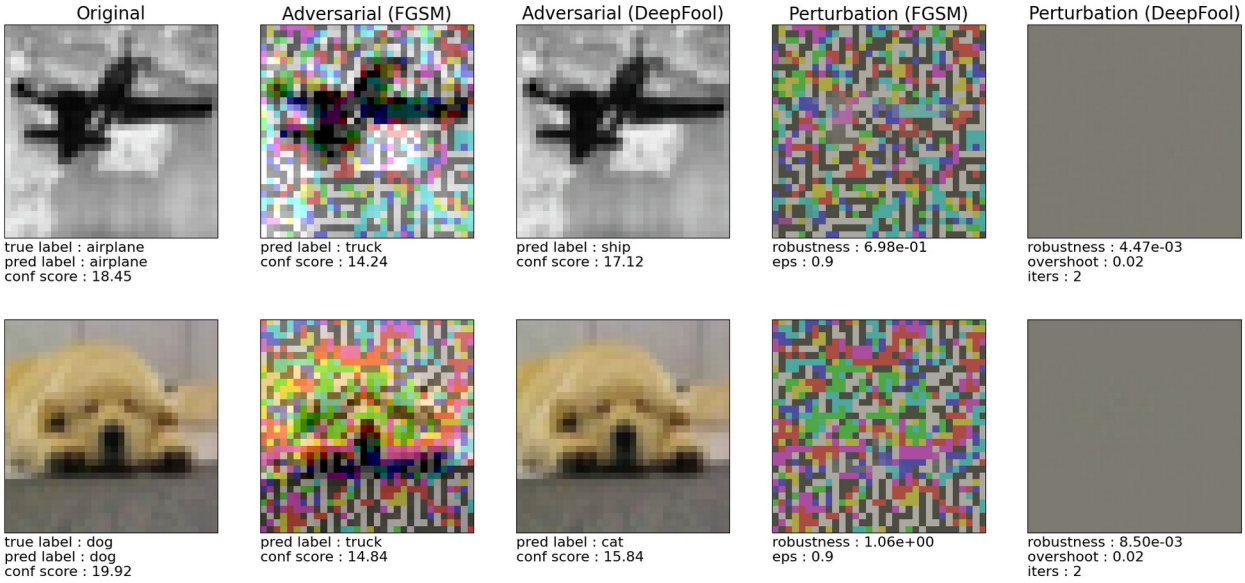

```
results for fgsm_eps = 0.02:
```



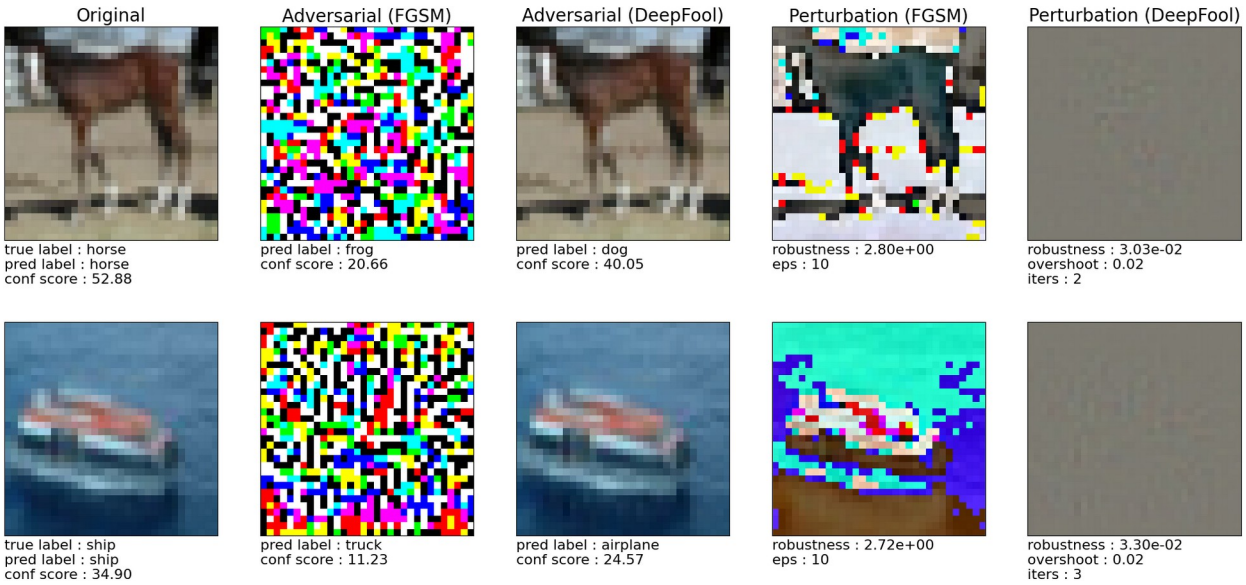

```
results for fgsm_eps = 0.5:
```




```
results for fgsm_eps = 0.9:
```

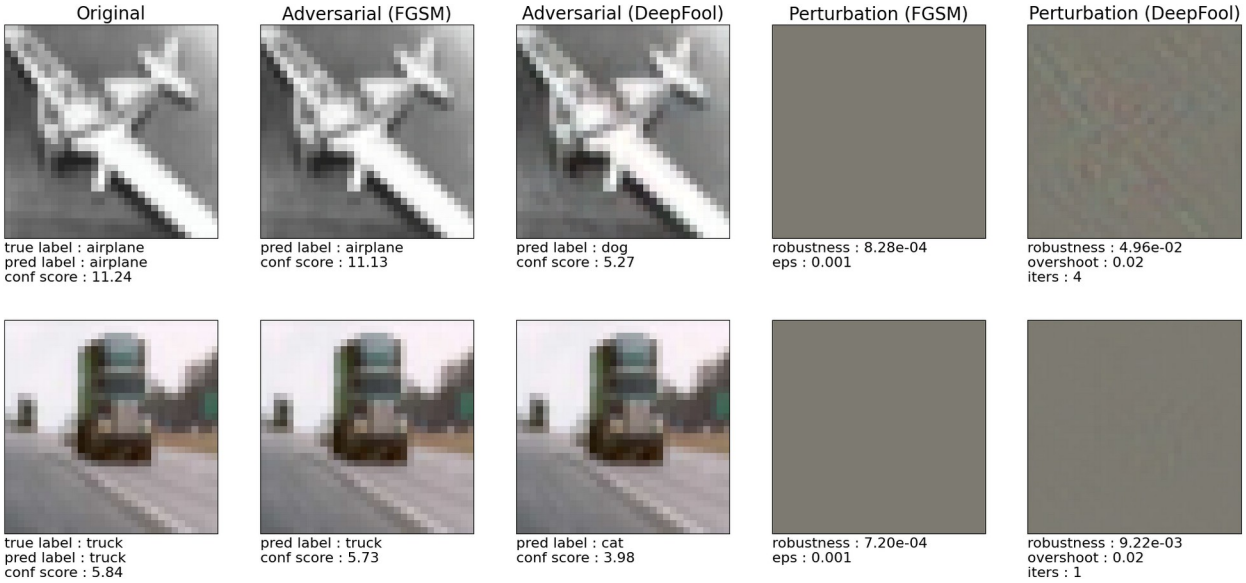



```
results for fgsm_eps = 10:
```

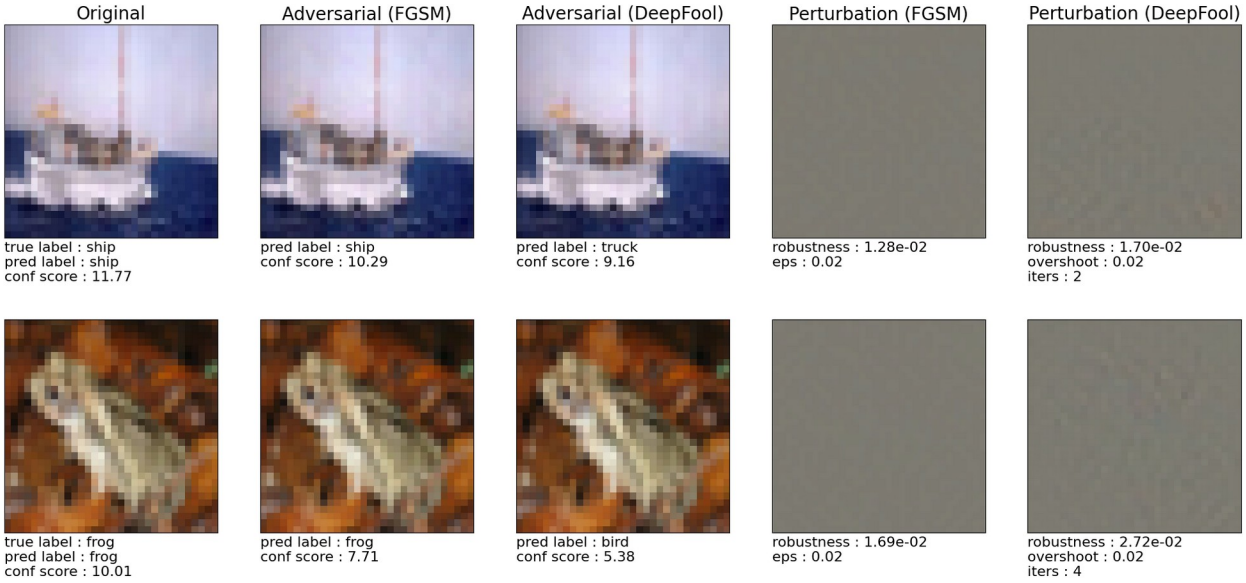



```
model = LeNet_CIFAR().to(device)
model.load_state_dict(torch.load('weights/clean/cifar_lenet.pth'))

for fgsm_eps in [0.001, 0.02, 0.5, 0.9, 10]:
    print(f"results for fgsm_eps = {fgsm_eps}:")
    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()
```

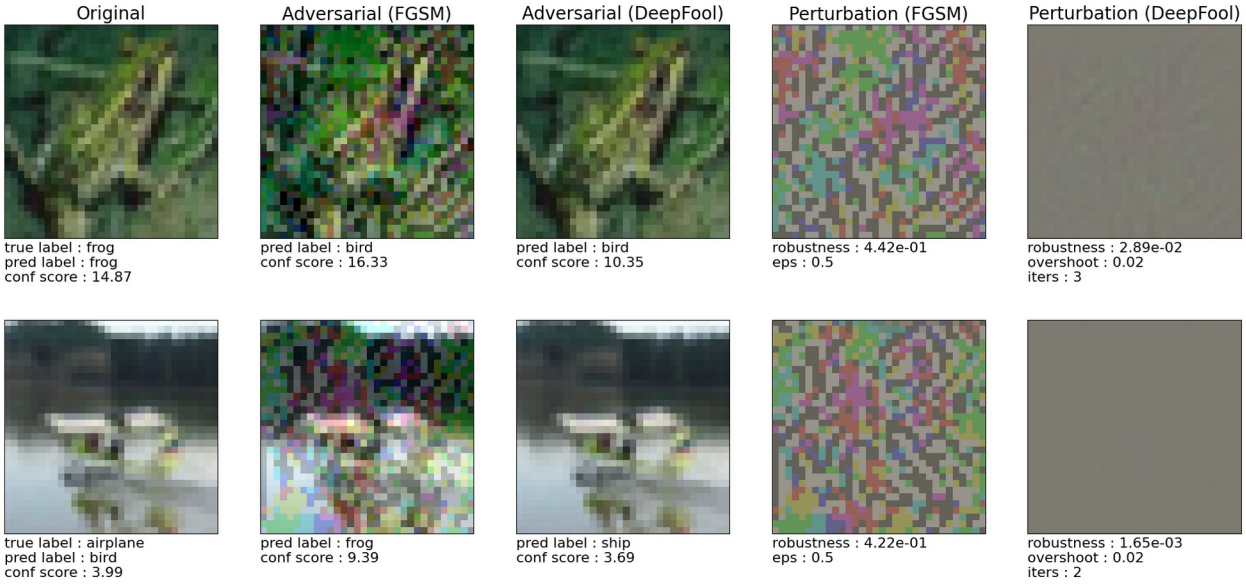
```
results for fgsm_eps = 0.001:

<ipython-input-15-5bebeb01d0cb>:2: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load('weights/clean/cifar_lenet.pth'))
```



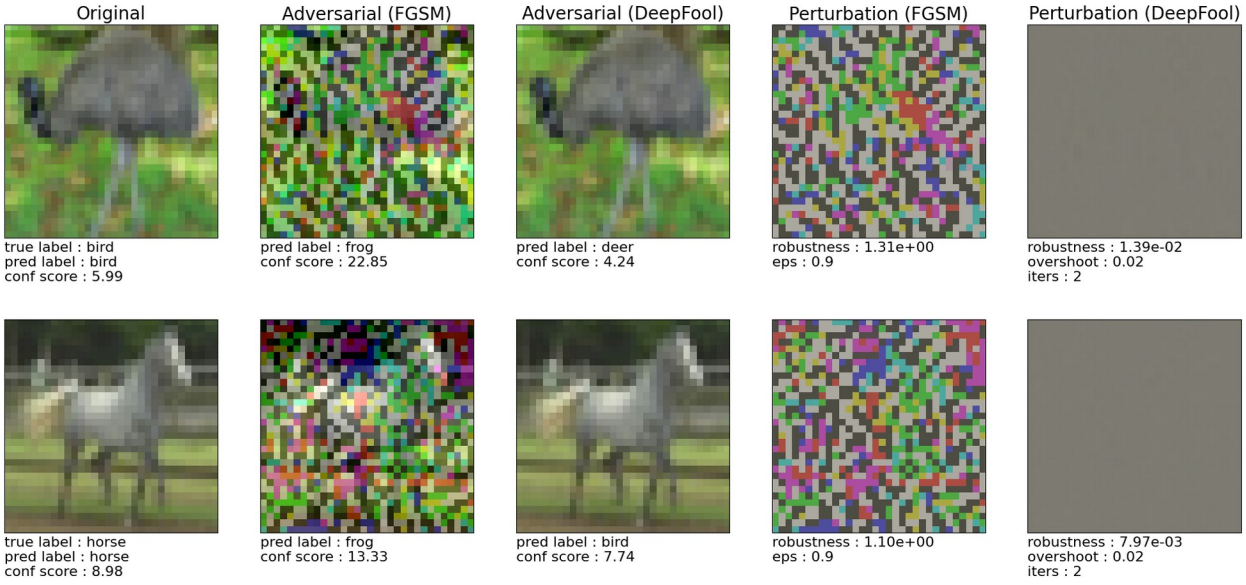

```
results for fgsm_eps = 0.02:
```



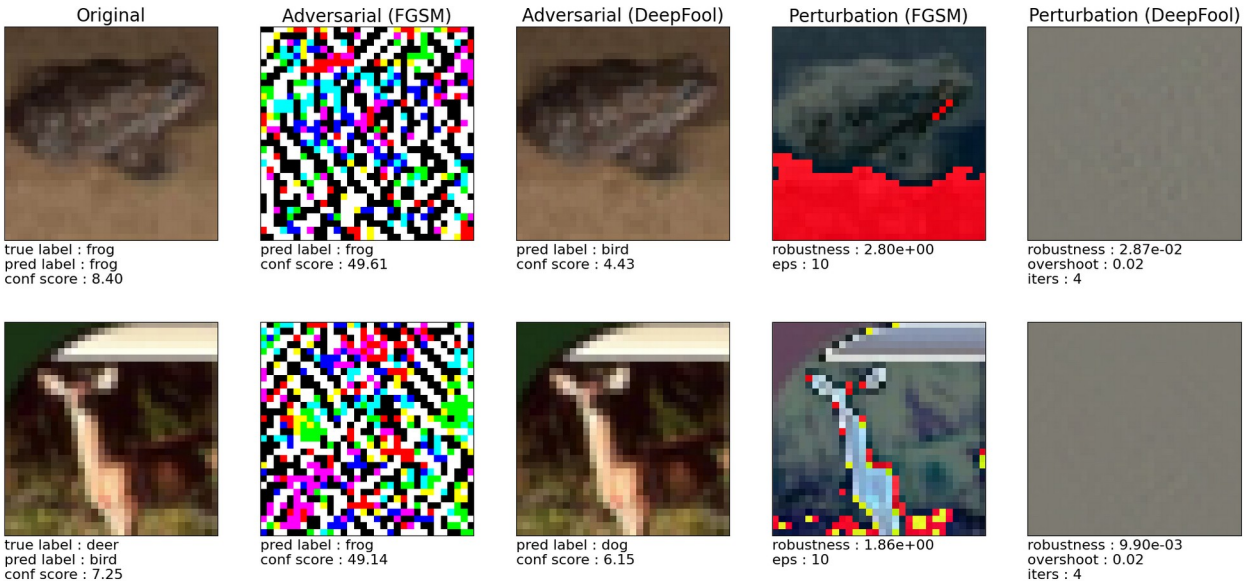

```
results for fgsm_eps = 0.5:
```




```
results for fgsm_eps = 0.9:
```




```
results for fgsm_eps = 10:
```



Вывод

В результате выполнения лабораторной работы было выявлено, что маленькие значения fgsm_eps сохраняют стойкость сетей к атакам, и ошибки классификации остаются низкими. При увеличении fgsm_eps сети становятся более уязвимыми к атакам и допускают больше ошибок классификации. Для сети FC LeNet на датасете MNIST и для сети NiN LeNEt на датасете CIFAR не наблюдается отсутствие влияния параметра fgsm_ eps. Наоборот, параметр fgsm_ eps оказывает существенное влияние на стойкость сетей к атакам.