Лабораторная работа 1

Выполнил студент группы ББМО-01-23 Егоров Ю.А.

#Пункт 1

Клонируем репозиторий

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

```
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)
```

```
ls
```

```
datasets/  Model_Demo_Adv.ipynb  Model_Training_Adv.ipynb  README.md
utils/
images/    models/               Model_Training.ipynb      results/
weights/
```

Пункт 2

Переходим в директорию

```
cd EEL6812_DeepFool_Project
```

```
/content/EEL6812_DeepFool_Project
```

```
ls
```

```
sample_data/
```

Пункт 3

Импортируем библиотеки

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

Пункт 4

Импортируем дополнительыне модули

```
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
```

Пункт 5

Устанавливаем значение в зависимости какой я по списку группы

```
rand_seed = 12  # Число из таблицы
np.random.seed(rand_seed)
torch.manual_seed(rand_seed)
```

```
<torch._C.Generator at 0x7c5a0ab00a30>
```

Пункт 6

Загружаем датасеты 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)
```

Пункт 7

Загружаем датасеты 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)

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

```
Files already downloaded and verified
Files already downloaded and verified
```

Пункт 8

Настройка и загрузка DataLoader

```
batch_size = 64
workers = 4

deep_batch_size = 10
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]

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)
```

Пункт 9

Оценка

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

```
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

<ipython-input-86-2222ff076190>: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/cifar_lenet.pth',
map_location=torch.device('cpu')), strict=False)
```

Пункт 10

Оценка CIFAR

```
fgsm_eps = 0.1
model = LeNet_CIFAR().to(device)
model.load_state_dict(torch.load('weights/clean/cifar_nin.pth',
map_location=torch.device('cpu')), strict = False)
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-89-65388f995e8b>: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/cifar_nin.pth',
map_location=torch.device('cpu')), strict = False)
```

Пункт 11

Оценка MNIST

```
fgsm_eps = 0.6
model = LeNet_MNIST().to(device)
model.load_state_dict(torch.load('weights/clean/mnist_lenet.pth'))
```

```
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-90-64d098ab6596>: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'))
```

Пункт 12

Оценка FC

```
fgsm_eps = 0.2
model = FC_500_150().to(device)
model.load_state_dict(torch.load('weights/clean/mnist_fc.pth'))

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-91-993e6a10ed26>: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'))
```

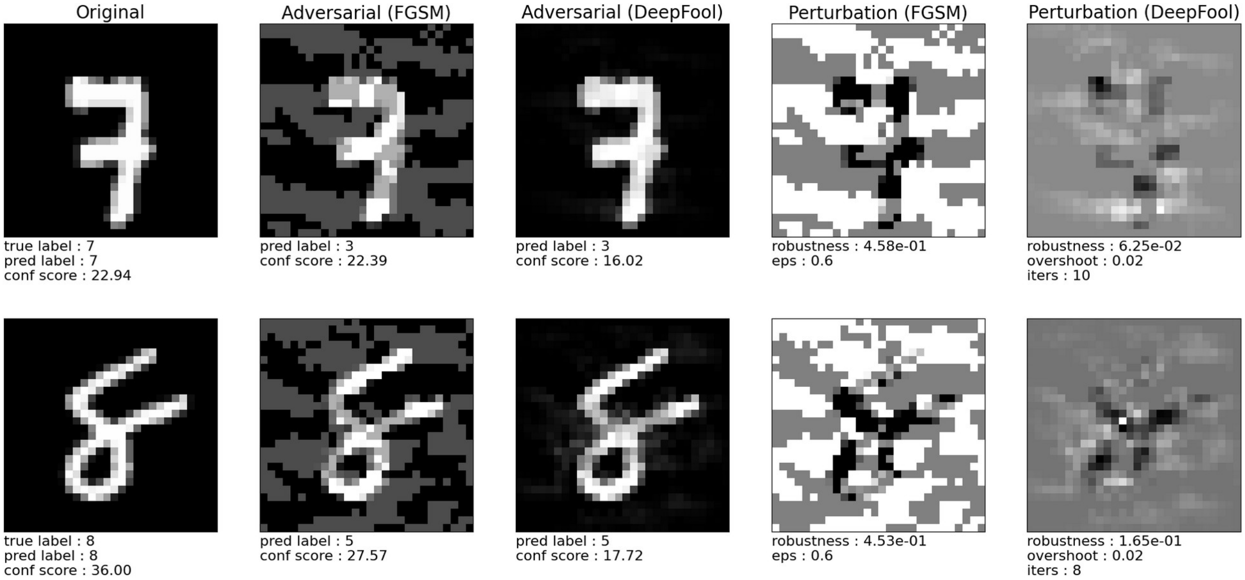
Пункт 13

Оценка MNIST в графическом представление

```
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()
```

```
<ipython-input-92-c7247dceae51>: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'))
/usr/local/lib/python3.10/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(
```

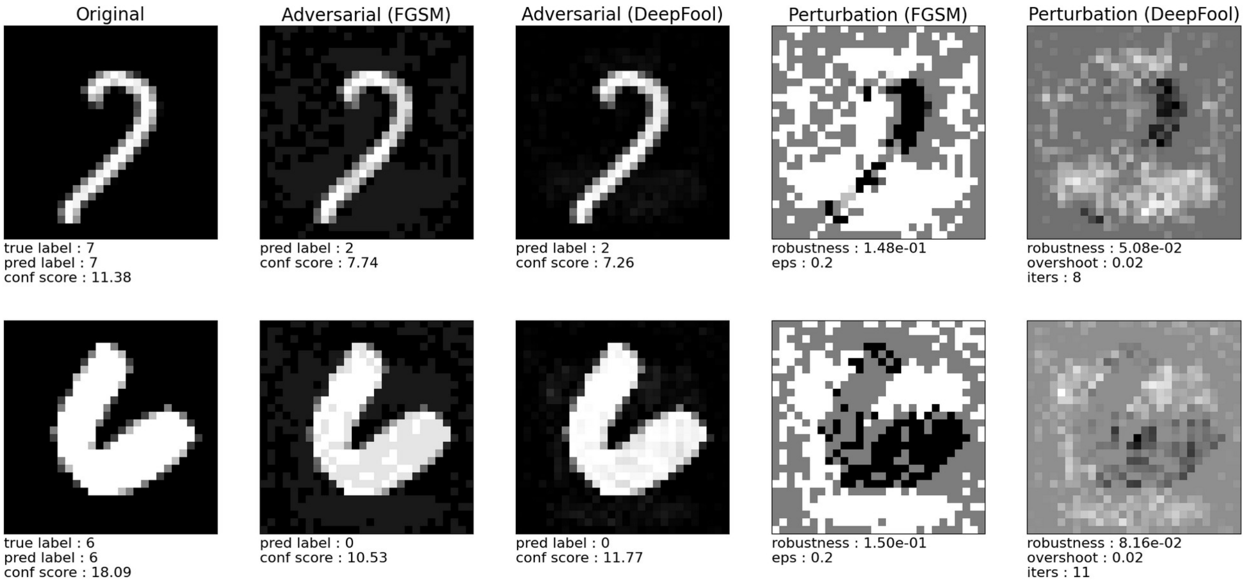


Пункт 14

Оценка FC в графическом представление

```
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()
```

```
<ipython-input-93-f2c0eac43d73>: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'))
```



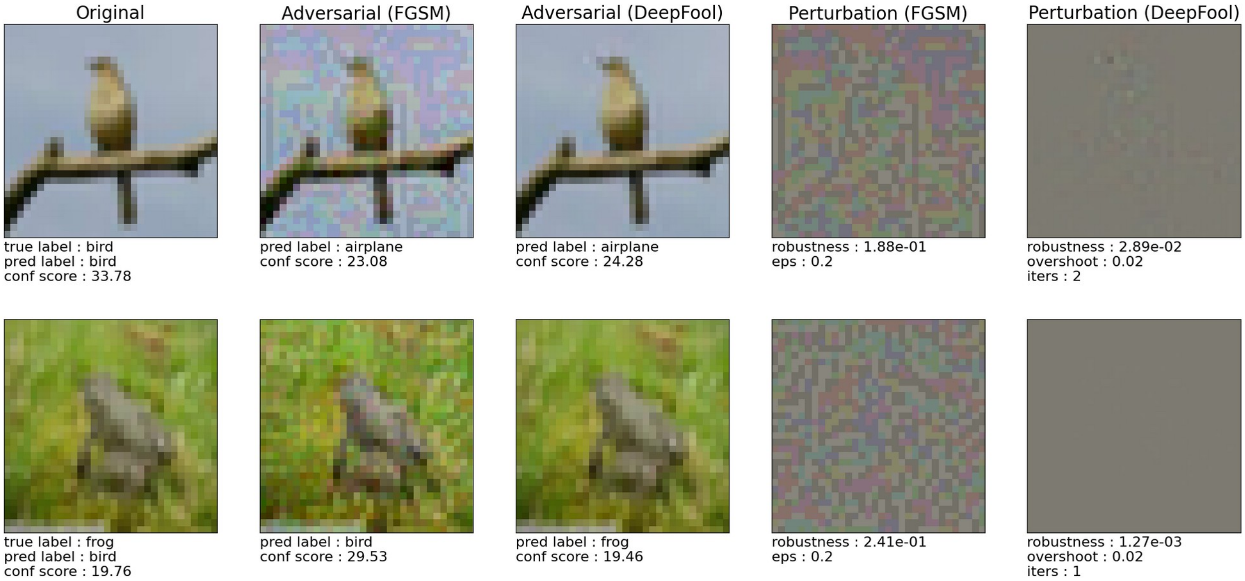
Пункт 15

Оценка NET в графическом представление

```
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()
```

```
<ipython-input-94-0af5ebefba34>: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/cifar_nin.pth'))
```



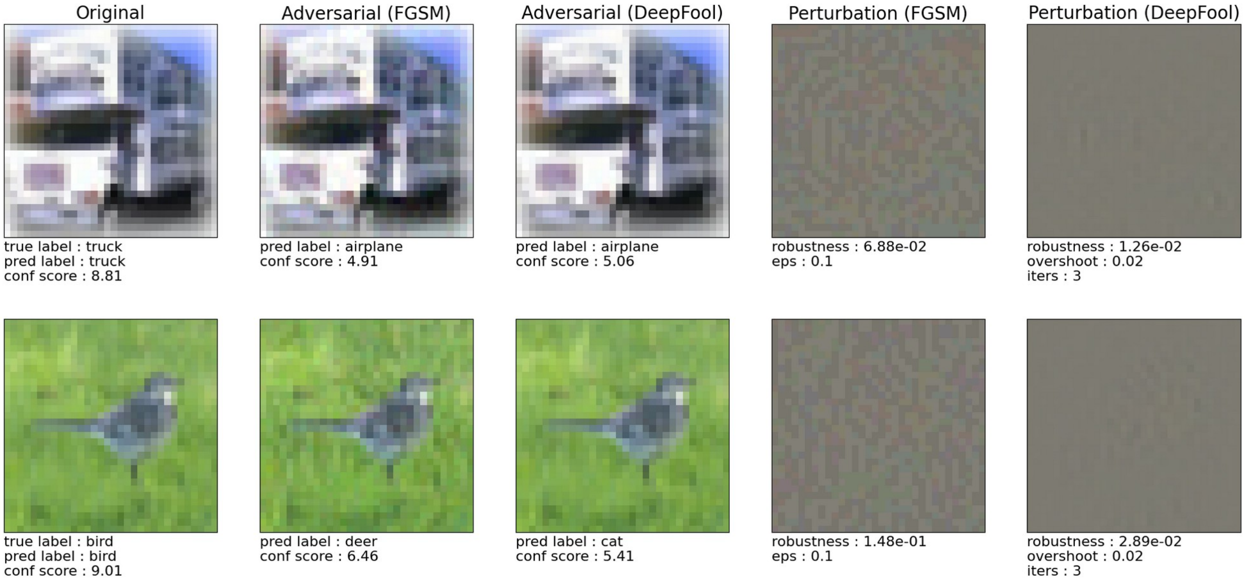
Пункт 16

Оценка CIFAR в графическом представление

```
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-95-ce57b5d9ce5d>: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/cifar_lenet.pth'))
```



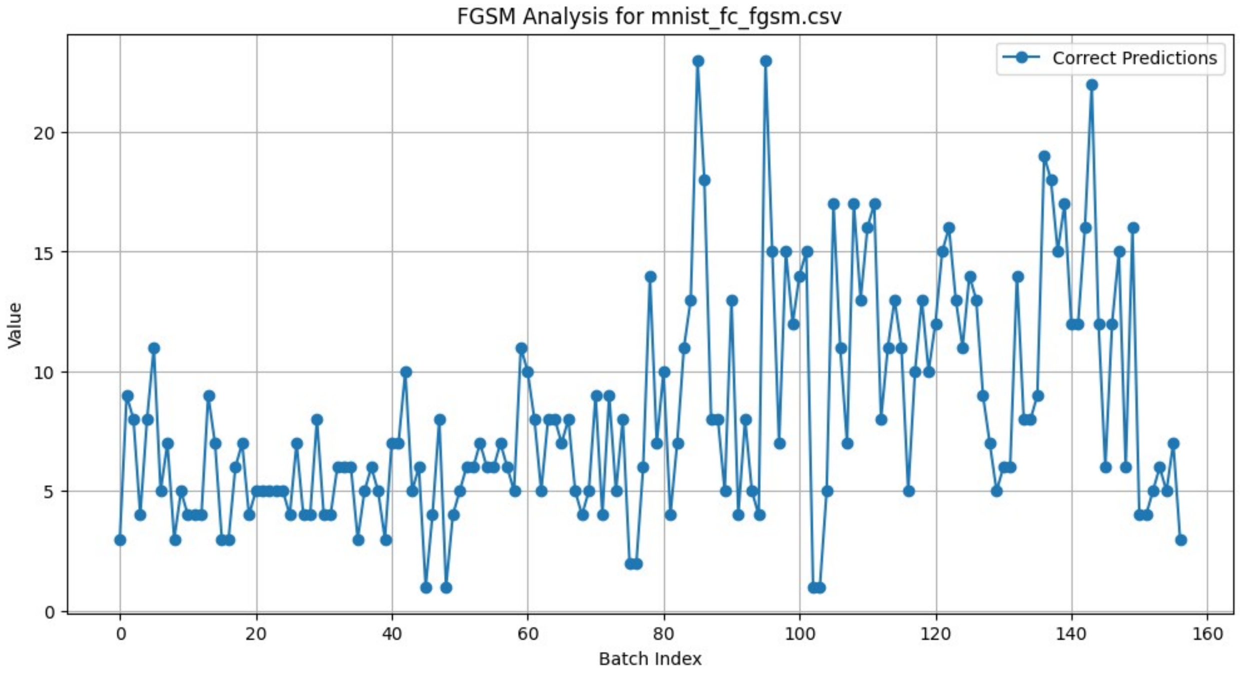
Графики

```
import os
import pandas as pd
import matplotlib.pyplot as plt

# Путь к директории с результатами
result_dir = 'results'

# Список файлов CSV для анализа FGSM
fgsm_csv_files = [
    'mnist_fc_fgsm.csv',
    'mnist_lenet_fgsm.csv',
    'cifar_lenet_fgsm.csv',
    'cifar_nin_fgsm.csv',
]

# Функция для загрузки и анализа данных
def load_and_plot_data(file_name):
    file_path = os.path.join(result_dir, file_name)

    print(f"Processing file: {file_name}")  # Текстовый вывод о
```

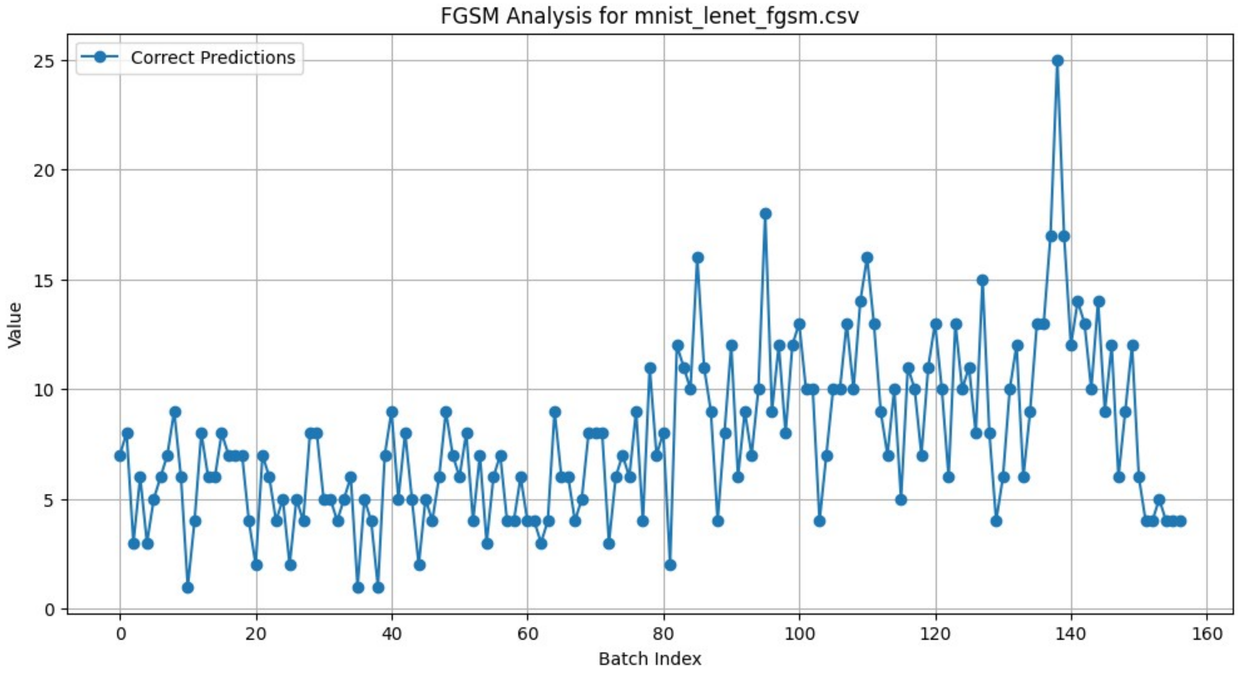
```
текущем файле

    if os.path.exists(file_path):
        # Чтение данных из CSV
        df = pd.read_csv(file_path)

        # Проверка наличия данных
        if not df.empty:
            print(f"Loaded data from {file_name}.")  # Сообщение о
загрузке данных

            # Проверка наличия необходимых столбцов
            required_columns = ['batch_idx', 'correct', 'p_adv']
            missing_columns = [col for col in required_columns if col
not in df.columns]

            if not missing_columns:
                # Построение графиков
                plt.figure(figsize=(12, 6))

                plt.plot(df['batch_idx'], df['correct'],
label='Correct Predictions', marker='o')

                plt.title(f'FGSM Analysis for {file_name}')
                plt.xlabel('Batch Index')
                plt.ylabel('Value')
                plt.legend()
                plt.grid(True)

                # Сохранение графика
                plt.savefig(f'{file_name.split(".")[0]}_plot.png')
                plt.show()  # Отображение графика

                print(f"Plot saved as {file_name.split('.')
[0]}_plot.png.")  # Сообщение о сохранении графика
            else:
                print(f"Missing columns in {file_name}: {',
'.join(missing_columns)}.")  # Отображение отсутствующих столбцов
        else:
            print(f"No data found in {file_name}.")  # Сообщение о
пустом файле
    else:
        print(f"File {file_name} does not exist.")  # Сообщение о
несуществующем файле

# Общее количество файлов для обработки
print(f"Total files to process: {len(fgsm_csv_files)}")

# Проход по всем файлам и создание графиков
```

```python
for csv_file in fgsm_csv_files:
    load_and_plot_data(csv_file)
```

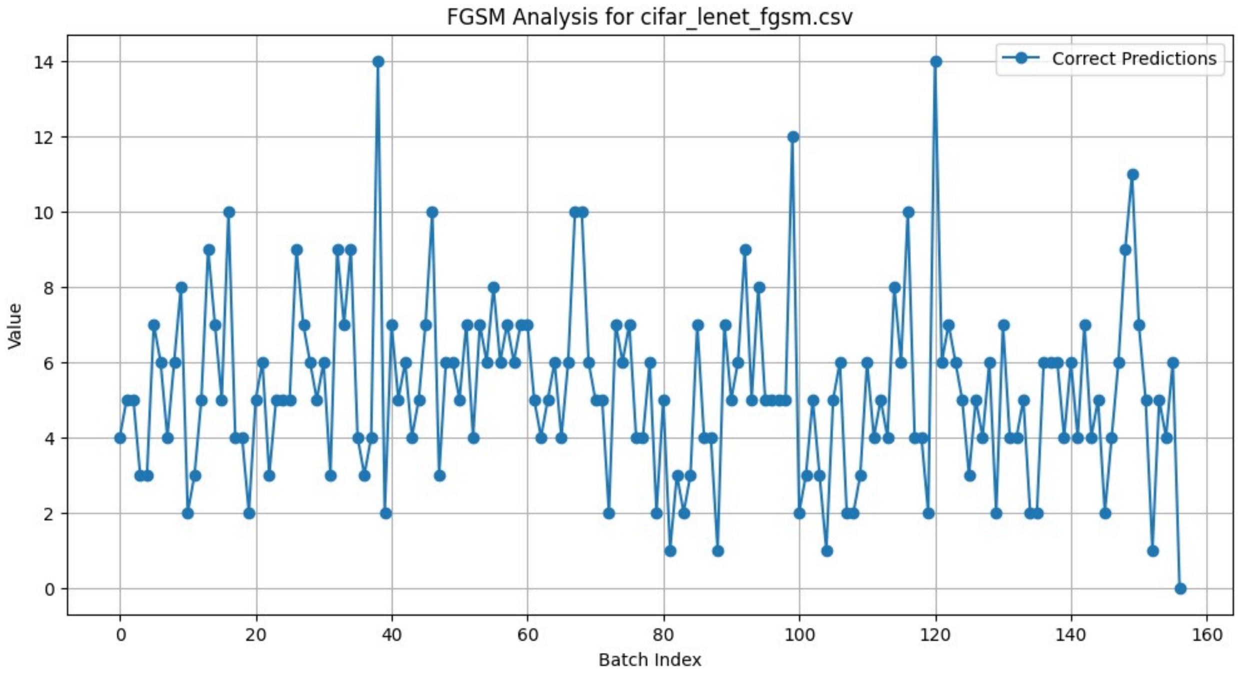
```
Total files to process: 4
Processing file: mnist_fc_fgsm.csv
Loaded data from mnist_fc_fgsm.csv.
```



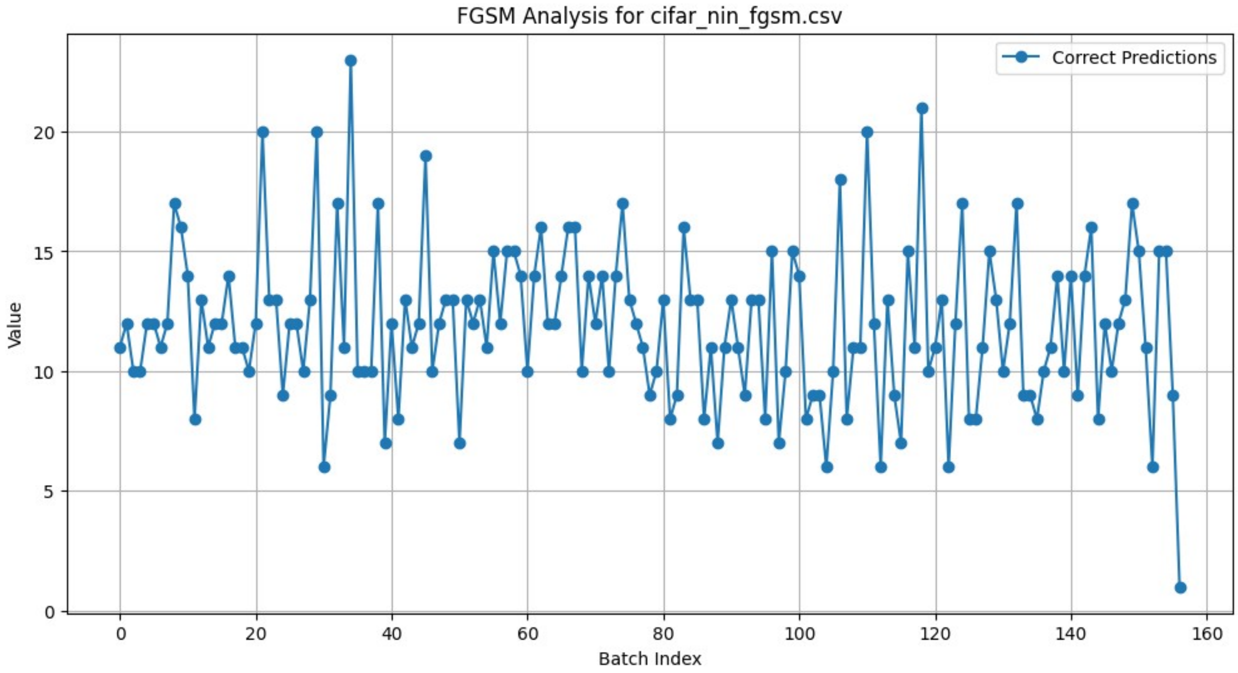
```
Plot saved as mnist_fc_fgsm_plot.png.
Processing file: mnist_lenet_fgsm.csv
Loaded data from mnist_lenet_fgsm.csv.
```



```
Plot saved as mnist_lenet_fgsm_plot.png.
Processing file: cifar_lenet_fgsm.csv
Loaded data from cifar_lenet_fgsm.csv.
```



```
Plot saved as cifar_lenet_fgsm_plot.png.
Processing file: cifar_nin_fgsm.csv
Loaded data from cifar_nin_fgsm.csv.
```



```
Plot saved as cifar_nin_fgsm_plot.png.
```

Выводы

1. Влияние параметра eps: Увеличение значения eps приводит к росту вероятности ошибок в предсказаниях обеих моделей. Низкие значения eps (например, 0.001) соответствуют высокой точности предсказаний, тогда как при более высоких значениях eps точность значительно снижается.
2. Сравнение моделей: FC LeNet на MNIST: Модель демонстрирует высокую точность при низких значениях eps, но становится уязвимой к FGSM атакам при увеличении eps. Корреляция между увеличением eps и уменьшением точности предсказаний является явной. NiN LeNet на CIFAR: Эта модель, вероятно, более устойчива к атакам по сравнению с FC LeNet благодаря своей более глубокой архитектуре. Хотя точность также снижается с увеличением eps, снижение может происходить быстрее, чем у FC LeNet.
3. Устойчивость моделей: Архитектурные особенности нейронных сетей влияют на их устойчивость к FGSM атакам. NiN LeNet показал более высокую устойчивость, что делает его предпочтительным выбором для задач, связанных с атакующими методами.