



МИНОБРНАУКИ РОССИИ

**Федеральное государственное бюджетное образовательное учреждение
высшего образования**

«МИРЭА – Российский технологический университет»

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

Отчёт по практической работе № 2

По дисциплине

«Анализ защищенности систем искусственного интеллекта»

Выполнил:

ББМО–02–22

Шмарковский М. Б.

Проверил:

Спирин А. А.

Москва, 2024

✓ Практическая работа №2

Выполнил Шмарковский МБ ББМО-02-22

Ход работы

```
# Скопируем проект по ссылке в локальную среду выполнения Jupyter (Google Colab)
!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
Receiving objects: 100% (96/96), 33.99 MiB | 8.06 MiB/s, done.
Resolving deltas: 100% (27/27), done.

# Сменим директорию исполнения на вновь созданную папку "EEL6812_DeepFool_Project"проекта.
%cd /content/EEL6812_DeepFool_Project

/content/EEL6812_DeepFool_Project

#Выполним импорт необходимых библиотек
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

#Выполним 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 randомное значение в виде переменной rand_seed={"Порядковый номер ученика группы в Гугл-таблице"}, укажем значение
rand_seed = 8
np.random.seed(rand_seed)
torch.manual_seed(rand_seed)

<torch._C.Generator at 0x7f4eb34ee5f0>

#Используем в качестве устройства видеокарту
use_cuda = torch.cuda.is_available()
device = torch.device('cuda' if use_cuda else 'cpu')

#Загрузим датасет MNIST с параметрами mnist_mean = 0.5, mnist_std = 0.5, mnist_dim = 28
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,
mnist_tf_inv = transforms.Compose([ transforms.Normalize( mean=0.0, std=np.divide(1.0, mnist_std)), transforms.Normalize( mean=np.multi

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)

#Загрузим датасет CIFAR-10 с параметрами cifar_mean = [0.491, 0.482, 0.447] cifar_std = [0.202, 0.199, 0.201] cifar_dim = 32
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.

cifar_tf_inv = transforms.Compose([ transforms.Normalize( mean=[0.0, 0.0, 0.0], std=np.divide(1.0, cifar_std)), transforms.Normalize( m

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

#Выполним 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/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker proce
warnings.warn(_create_warning_msg(
<

```

```

#Инициализируем deep_args
batch_size = 10
num_classes = 10
overshoot = 0.02
max_iters = 50
deep_args = [batch_size, num_classes, overshoot, max_iters]

#Загрузим и оценим стойкость модели LeNet к FGSM и DeepFool атакам
fgsm_eps = 0.6
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

#Загрузим и оценим 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

```

Вывод

LeNet проявляет большую устойчивость как к атакам FGSM, так и к атакам DeepFool. FC немного лучше справляется с ошибками тестирования и значительно быстрее обрабатывает как все изображения, так и каждое изображение.