



**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

ИКБ направление «Киберразведка и противодействие угрозам  
с применением технологий искусственного интеллекта» 10.04.01

Кафедра КБ-4 «Интеллектуальные системы информационной  
безопасности»

## **Практическая работа №6 – лабораторная работа №4**

по дисциплине

**«Анализ защищенности систем искусственного интеллекта»**

Группа:  
ББМО-01-2  
2

Выполнила:  
Челышев  
Д.И.

Проверил  
: Спирин  
А.А.

Москва 2023

Выполним импорт необходимых библиотек.

```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import transforms, datasets
```

Зададим нормализующие преобразования, загрузим набор данных (MNIST), разобьем данные на подвыборки.

```
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.0,), (1.0,))])
dataset = datasets.MNIST(root = './data', train=True, transform = transform, download=True)
train_set, val_set = torch.utils.data.random_split(dataset, [50000, 10000])
test_set = datasets.MNIST(root = './data', train=False, transform = transform, download=True)
train_loader = torch.utils.data.DataLoader(train_set, batch_size=1, shuffle=True)
val_loader = torch.utils.data.DataLoader(val_set, batch_size=1, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=1, shuffle=True)
```

Настроим использование графического ускорителя.

```
use_cuda=True
device = torch.device("cuda" if (use_cuda and torch.cuda.is_available()) else "cpu")
```

Создадим класс НС на основе фреймворка torch.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)
    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

Проверим работоспособность созданного класса НС.

```
model = Net().to(device)
```

Создадим оптимизатор, функцию потерь и трейнер сети.

```
optimizer = optim.Adam(model.parameters(), lr=0.0001, betas=(0.9, 0.999))
criterion = nn.NLLLoss()
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=3)
```

Определили функцию обучения сети.

```
def fit(model,device,train_loader,val_loader,epochs):
    data_loader = {'train':train_loader,'val':val_loader}
    print("Fitting the model...")
    train_loss,val_loss=[],[]
    for epoch in range(epochs):
        loss_per_epoch,val_loss_per_epoch=0,0
        for phase in ('train','val'):
            for i,data in enumerate(data_loader[phase]):
                input,label = data[0].to(device),data[1].to(device)
                output = model(input)
                #calculating loss on the output
                loss = criterion(output,label)
                if phase == 'train':
                    optimizer.zero_grad()
                    #grad calc w.r.t Loss func
                    loss.backward()
                    #update weights
                    optimizer.step()
                    loss_per_epoch+=loss.item()
                else:
                    val_loss_per_epoch+=loss.item()
            scheduler.step(val_loss_per_epoch/len(val_loader))
        print("Epoch: {} Loss: {} Val_Loss: {}".format(epoch+1,loss_per_epoch/len(train_loader),val_loss_per_epoch/len(val_loader)))
        train_loss.append(loss_per_epoch/len(train_loader))
        val_loss.append(val_loss_per_epoch/len(val_loader))
    return train_loss,val_loss
```

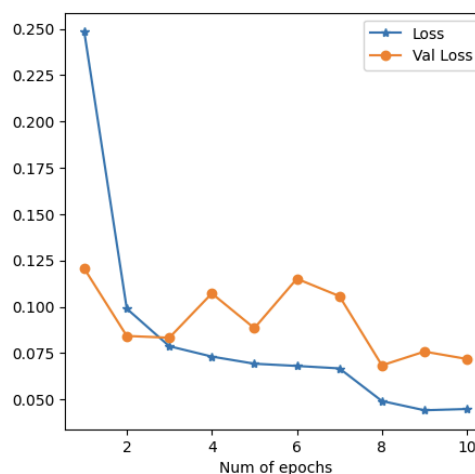
Обучим модель.

```
loss,val_loss=fit(model,device,train_loader,val_loader,10)
```

```
Fitting the model...
Epoch: 1 Loss: 0.24841752398738037 Val_Loss: 0.12061824470002139
Epoch: 2 Loss: 0.09896987803633864 Val_Loss: 0.08424650834364011
Epoch: 3 Loss: 0.07869950483775712 Val_Loss: 0.08328769305922769
Epoch: 4 Loss: 0.07311822975879405 Val_Loss: 0.10720105448503194
Epoch: 5 Loss: 0.06926371024317246 Val_Loss: 0.0884460765918882
Epoch: 6 Loss: 0.06806250782961033 Val_Loss: 0.11510764005056094
Epoch: 7 Loss: 0.06674116253729839 Val_Loss: 0.10574276480031856
Epoch: 8 Loss: 0.04912241660950879 Val_Loss: 0.0683882074708118
Epoch: 9 Loss: 0.044145967845800285 Val_Loss: 0.0758055572589581
Epoch: 10 Loss: 0.044831771374373657 Val_Loss: 0.07181386937930914
```

Построим графики потерь при обучении и валидации в зависимости от эпохи.

```
fig = plt.figure(figsize=(5,5))
plt.plot(np.arange(1,11), loss, "--",label="Loss")
plt.plot(np.arange(1,11), val_loss,"o-",label="Val Loss")
plt.xlabel("Num of epochs")
plt.legend()
plt.show()
```



Создадим функции атак FGSM, I-FGSM, MI-FGSM.

```
def fgsm_attack(input,epsilon,data_grad):
    pert_out = input + epsilon*data_grad.sign()
    pert_out = torch.clamp(pert_out, 0, 1)
    return pert_out
```

```
def ifgsm_attack(input,epsilon,data_grad):
    iter = 10
    alpha = epsilon/iter
    pert_out = input
    for i in range(iter-1):
        pert_out = pert_out + alpha*data_grad.sign()
        pert_out = torch.clamp(pert_out, 0, 1)
        if torch.norm((pert_out-input),p=float('inf')) > epsilon:
            break
    return pert_out
```

```
def mifgsm_attack(input,epsilon,data_grad):
    iter=10
    decay_factor=1.0
    pert_out = input
    alpha = epsilon/iter
    g=0
    for i in range(iter-1):
        g = decay_factor*g + data_grad/torch.norm(data_grad,p=1)
        pert_out = pert_out + alpha*torch.sign(g)
        pert_out = torch.clamp(pert_out, 0, 1)
        if torch.norm((pert_out-input),p=float('inf')) > epsilon:
            break
    return pert_out
```

Создадим функцию проверки.

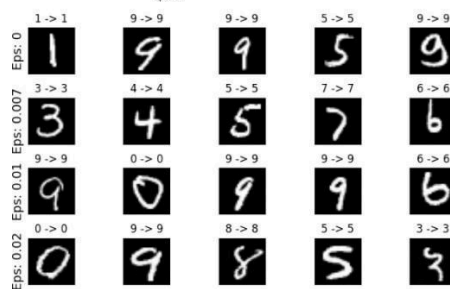
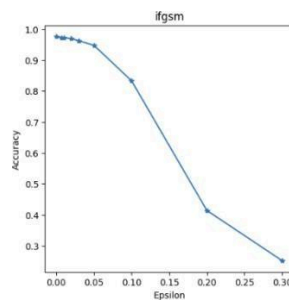
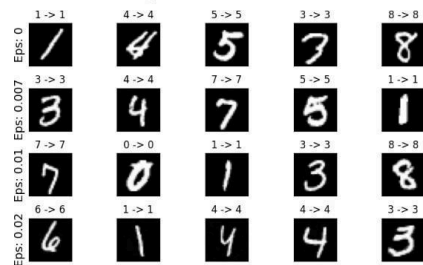
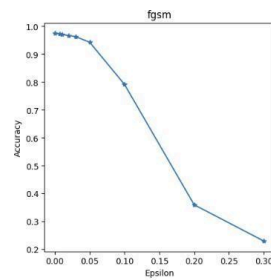
```
def test(model,device,test_loader,epsilon,attack):
    correct = 0
    adv_examples = []
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        data.requires_grad = True
        output = model(data)
        init_pred = output.max(1, keepdim=True)[1]
        if init_pred.item() != target.item():
            continue
        loss = F.nll_loss(output, target)
        model.zero_grad()
        loss.backward()
        data_grad = data.grad.data
        if attack == "fgsm":
            perturbed_data = fgsm_attack(data,epsilon,data_grad)
        elif attack == "ifgsm":
            perturbed_data = ifgsm_attack(data,epsilon,data_grad)
        elif attack == "mifgsm":
            perturbed_data = mifgsm_attack(data,epsilon,data_grad)
        output = model(perturbed_data)
        final_pred = output.max(1, keepdim=True)[1]
        if final_pred.item() == target.item():
            correct += 1
        if (epsilon == 0) and (len(adv_examples) < 5):
            adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
            adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex))
        else:
            if len(adv_examples) < 5:
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex))
    final_acc = correct/float(len(test_loader))
    print("Epsilon: {} \t Test Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader), final_acc))
    return final_acc, adv_examples
```

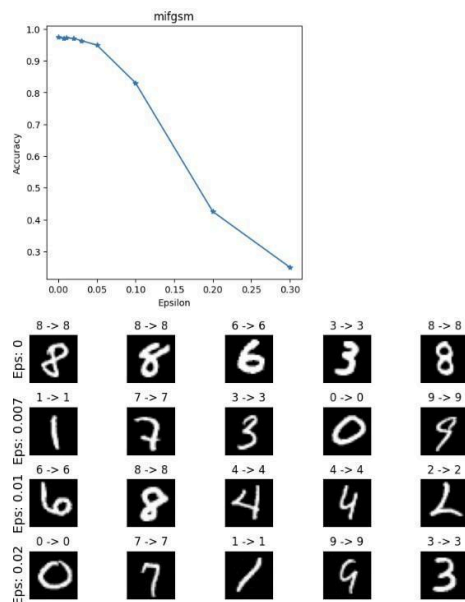
Построим графики успешности атак(Ассигасу/эпсилон) и примеры выполненных атак в зависимости от степени возмущения epsilon.

```

epsilons = [0,0.007,0.01,0.02,0.03,0.05,0.1,0.2,0.3]
for attack in ("fgsm","ifgsm","mifgsm"):
    accuracies = []
    examples = []
    for eps in epsilons:
        acc, ex = test(model, device, test_loader, eps, attack)
        accuracies.append(acc)
        examples.append(ex)
    plt.figure(figsize=(5,5))
    plt.plot(epsilons, accuracies, "*-")
    plt.title(attack)
    plt.xlabel("Epsilon")
    plt.ylabel("Accuracy")
    plt.show()
    cnt = 0
    plt.figure(figsize=(8,10))
    for i in range(len(epsilons)):
        for j in range(len(examples[i])):
            cnt += 1
            plt.subplot(len(epsilons),len(examples[0]),cnt)
            plt.xticks([], [])
            plt.yticks([], [])
            if j == 0:
                plt.ylabel("Eps: {}".format(epsilons[i]), fontsize=14)
            orig,adv,ex = examples[i][j]
            plt.title("{} -> {}".format(orig, adv))
            plt.imshow(ex, cmap="gray")
    plt.tight_layout()
    plt.show()

```





Создадим 2 класса НС.

```
class NetF(nn.Module):
    def __init__(self):
        super(NetF, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        return x
```

```
class NetF1(nn.Module):
    def __init__(self):
        super(NetF1, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, 1)
        self.conv2 = nn.Conv2d(16, 32, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(4608, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        return x
```

Переопределим функцию обучения и тестирования.

```

def fit(model,device,optimizer,scheduler,criterion,train_loader,val_loader,Temp,epochs):
    data_loader = {'train':train_loader,'val':val_loader}
    print("Fitting the model...")
    train_loss,val_loss=[],[]
    for epoch in range(epochs):
        loss_per_epoch,val_loss_per_epoch=0,0
        for phase in ('train','val'):
            for i,data in enumerate(data_loader[phase]):
                input,label = data[0].to(device),data[1].to(device)
                output = model(input)
                output = F.log_softmax(output/Temp,dim=1)
                #calculating loss on the output
                loss = criterion(output,label)
            if phase == 'train':
                optimizer.zero_grad()
                #grad calc w.r.t loss func
                loss.backward()
                #update weights
                optimizer.step()
                loss_per_epoch+=loss.item()
            else:
                val_loss_per_epoch+=loss.item()
        scheduler.step(val_loss_per_epoch/len(val_loader))
        print("Epoch: {} Loss: {} Val_Loss: {}".format(epoch+1,loss_per_epoch/len(train_loader),val_loss_per_epoch/len(val_loader)))
        train_loss.append(loss_per_epoch/len(train_loader))
        val_loss.append(val_loss_per_epoch/len(val_loader))
    return train_loss,val_loss

def test(model,device,test_loader,epsilon,Temp,attack):
    correct=0
    adv_examples = []
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        data.requires_grad = True
        output = model(data)
        output = F.log_softmax(output/Temp,dim=1)
        init_pred = output.max(1, keepdim=True)[1]
        if init_pred.item() != target.item():
            continue
        loss = F.nll_loss(output, target)
        model.zero_grad()
        loss.backward()
        data_grad = data.grad.data
        if attack == "fgsm":
            perturbed_data = fgsm_attack(data,epsilon,data_grad)
        elif attack == "ifgsm":
            perturbed_data = ifgsm_attack(data,epsilon,data_grad)
        elif attack == "mifgsm":
            perturbed_data = mifgsm_attack(data,epsilon,data_grad)
        output = model(perturbed_data)
        final_pred = output.max(1, keepdim=True)[1]
        if final_pred.item() == target.item():
            correct += 1
        if (epsilon == 0) and (len(adv_examples) < 5):
            adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
            adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex) )
        else:
            if len(adv_examples) < 5:
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex) )
    final_acc = correct/float(len(test_loader))
    print("Epsilon: {} \t Test Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader), final_acc))
    return final_acc,adv_examples

```

Создадим функцию защиты методом дистилляции.



```

def defense(device,train_loader,val_loader,test_loader,epochs,Temp,epsilons):
    modelF = NetF().to(device)
    optimizerF = optim.Adam(modelF.parameters(),lr=0.0001, betas=(0.9, 0.999))
    schedulerF = optim.lr_scheduler.ReduceLROnPlateau(optimizerF, mode='min', factor=0.1, patience=3)
    modelF1 = NetF1().to(device)
    optimizerF1 = optim.Adam(modelF1.parameters(),lr=0.0001, betas=(0.9, 0.999))
    schedulerF1 = optim.lr_scheduler.ReduceLROnPlateau(optimizerF1, mode='min', factor=0.1, patience=3)
    criterion = nn.NLLLoss()
    lossF, val_lossF=fit(modelF,device,optimizerF,schedulerF,criterion,train_loader,val_loader,Temp,epochs)
    fig = plt.figure(figsize=(5,5))
    plt.plot(np.arange(1,epochs+1), lossF, "*-",label="Loss")
    plt.plot(np.arange(1,epochs+1), val_lossF,"o-",label="Val Loss")
    plt.title("Network F")
    plt.xlabel("Num of epochs")
    plt.legend()
    plt.show()
    #converting target labels to soft labels
    for data in train_loader:
        input, label = data[0].to(device),data[1].to(device)
        softlabel = F.log_softmax(modelF(input),dim=1)
        data[1] = softlabel
    lossF1,val_lossF1=fit(modelF1,device,optimizerF1,schedulerF1,criterion,train_loader,val_loader,Temp,epochs)
    fig = plt.figure(figsize=(5,5))
    plt.plot(np.arange(1,epochs+1), lossF1, "*-",label="Loss")
    plt.plot(np.arange(1,epochs+1), val_lossF1,"o-",label="Val Loss")
    plt.title("Network F'")
    plt.xlabel("Num of epochs")
    plt.legend()
    plt.show()
    model = NetF1().to(device)
    model.load_state_dict(modelF1.state_dict())
    for attack in ("fgsm","ifgsm","mifgsm"):
        accuracies = []
        examples = []
        for eps in epsilons:
            acc, ex = test(model,device,test_loader,eps,1,"fgsm")
            accuracies.append(acc)
            examples.append(ex)
        plt.figure(figsize=(5,5))
        plt.plot(epsilons, accuracies, "*-")
        plt.title(attack)
        plt.xlabel("Epsilon")
        plt.ylabel("Accuracy")
        plt.show()
        cnt = 0
        plt.figure(figsize=(8,10))
        for i in range(len(epsilons)):
            for j in range(len(examples[i])):
                cnt += 1
                plt.subplot(len(epsilons),len(examples[0]),cnt)
                plt.xticks([], [])
                plt.yticks([], [])
                if j == 0:
                    plt.ylabel("Eps: {}".format(epsilons[i]), fontsize=14)
                orig,adv,ex = examples[i][j]
                plt.title("{} -> {}".format(orig, adv))
                plt.imshow(ex, cmap="gray")
        plt.tight_layout()
        plt.show()

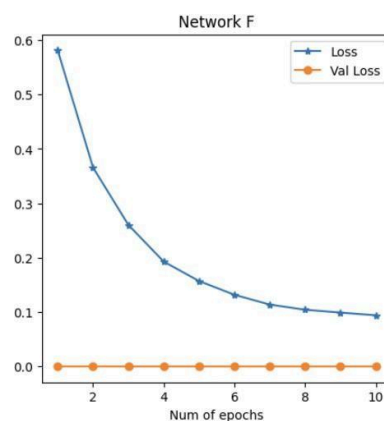
```

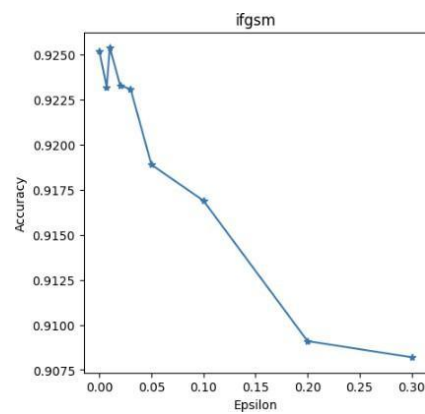
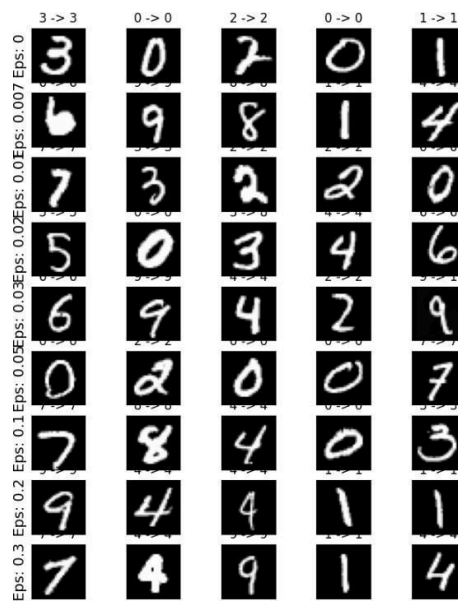
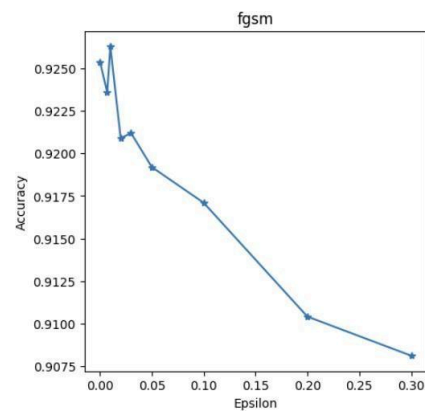
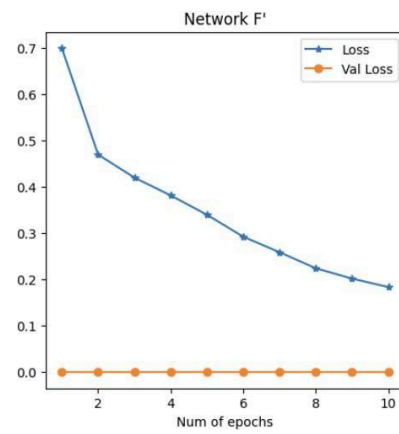
Получим результаты оценки защищенных сетей.

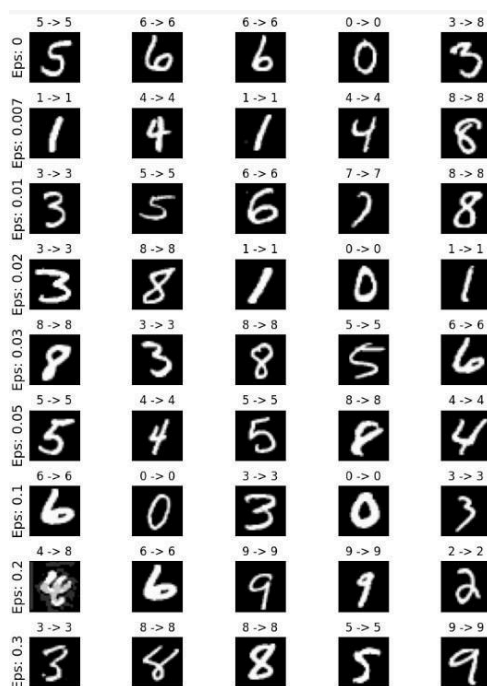
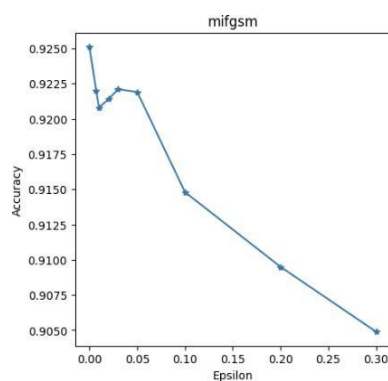
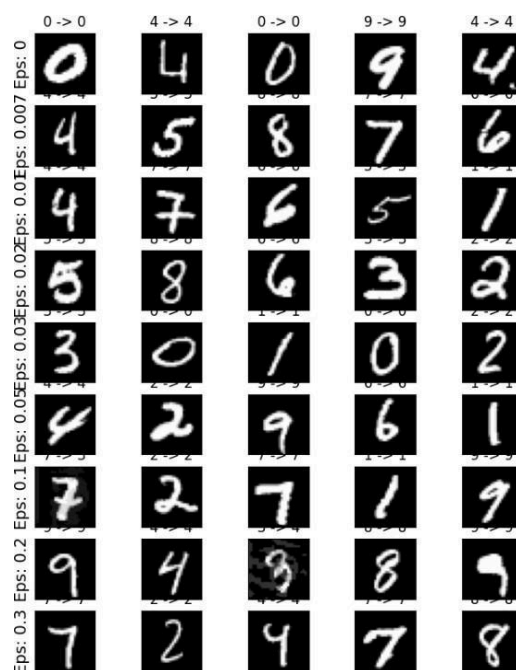
```

Temp=100
epochs=10
epsilons=[0,0.007,0.01,0.02,0.03,0.05,0.1,0.2,0.3]
defense(device,train_loader,val_loader,test_loader,epochs,Temp,epsilons)

```







Вывод: Основная идея защитной дистилляции заключается в обучении устойчивой модели, путем передачи знаний от базовой модели,

подверженной атакам, к новой модели, которая спроектирована для устойчивости к различным атакам.

Дистилляция дает более плоские локальные минимумы. Следовательно, небольшие изменения во входных данных с меньшей вероятностью изменят прогнозируемые значения.

Итог по увеличению стойкости модели:

атака fgsm снизила точность не защищенных данных до - 14%,  
защищенных - до - 91%;

атака ifgsm снизила точность не защищенных данных до - 15%,  
защищенных - до - 91%;

атака mifgsm снизила точность не защищенных данных до - 15%,  
защищенных - до - 91%.

Модель, которая обучалась на метках учителя имеет большее значение потерь после обучения, но это не влируется стойкостью к атакам FGSM.