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»

Лабораторная работа №4

по дисциплине

«Анализ защищенности систем искусственного интеллекта»

Группа:
ББМО-02-22
Выполнил:
Давыдов И.Д.

Проверил:
Спирин А.А.

Москва 2024

Выполняем импорт необходимых библиотек. Загрузим набор данных (MNIST), разобьем данные на подвыборки

```
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
```

```
#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)
print("Training data:",len(train_loader),"Validation data:",len(val_loader),"Test data:",len(test_loader))
```

```
Training data: 50000 Validation data: 10000 Test data: 10000
```

5) Проверим работоспособность созданного класса НС

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

6) 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)
```

7) Определим 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
```

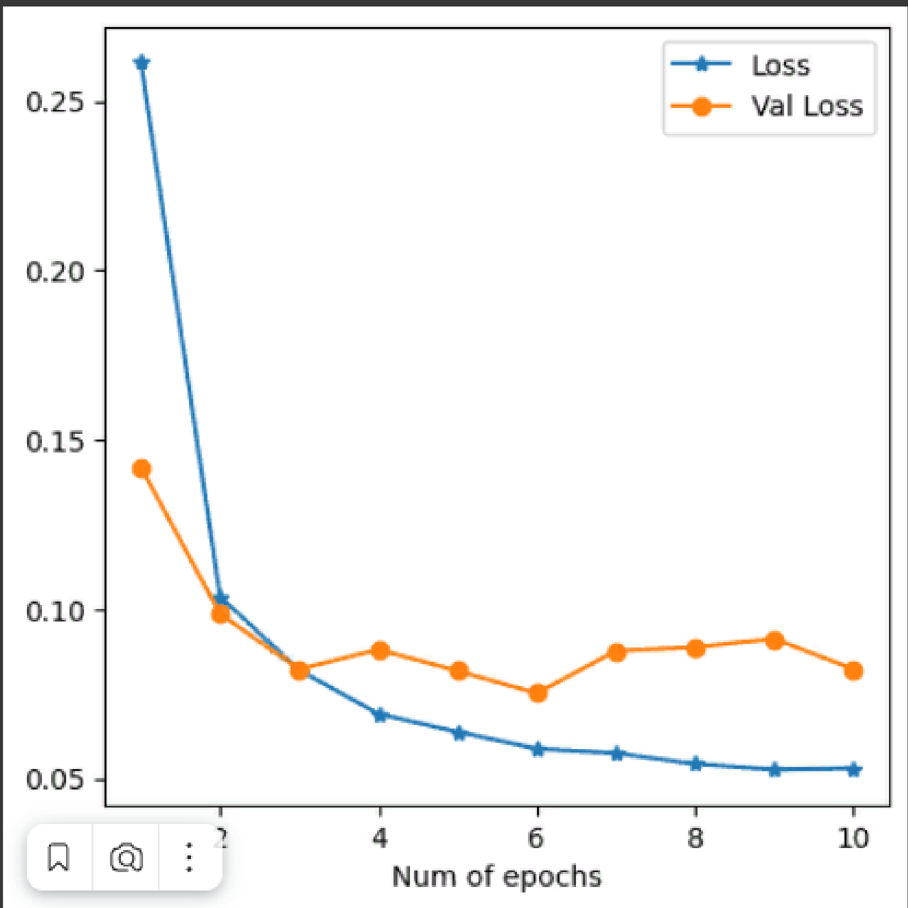
8) Обучим модель

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

```
Fitting the model...
/usr/local/lib/python3.10/dist-packages/torch/nn/functional.py:1345: UserWarning: dropout2d: Received a 2-D input to dropout2d, which is deprecated and will result in an error in a future release. To retain the behavior and silence this warning, please use dropout instead. Note t
  warnings.warn(warn_msg)
Epoch: 1 Loss: 0.2615168512271497 Val_Loss: 0.141892903709591
Epoch: 2 Loss: 0.1036137397355153 Val_Loss: 0.09853584496404914
Epoch: 3 Loss: 0.08197049498399668 Val_Loss: 0.08220776474540184
Epoch: 4 Loss: 0.06904626972780604 Val_Loss: 0.08802659181930449
Epoch: 5 Loss: 0.06372093805587897 Val_Loss: 0.08171905556179286
Epoch: 6 Loss: 0.05879109301412018 Val_Loss: 0.07514899941182278
Epoch: 7 Loss: 0.05746847337931948 Val_Loss: 0.08764700718475141
Epoch: 8 Loss: 0.054268178856173924 Val_Loss: 0.08874003255999988
Epoch: 9 Loss: 0.05263297796054569 Val_Loss: 0.09124095480602284
Epoch: 10 Loss: 0.05299339745590674 Val_Loss: 0.0822268501372125
```

9) Строим график.

```
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()
```



10) 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
```

```
[11] 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
```

```
[12] 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
```

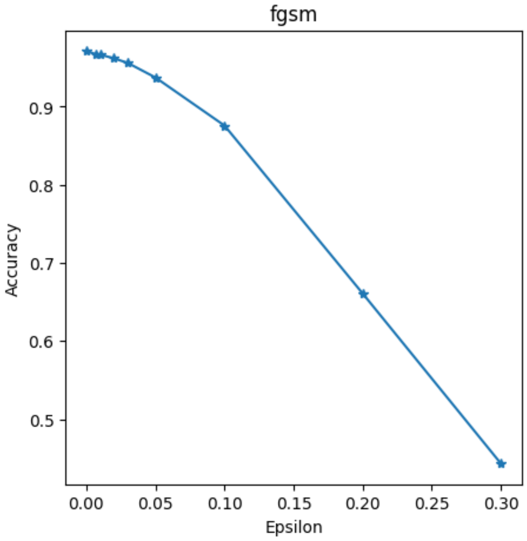
11) Создадим функцию проверки

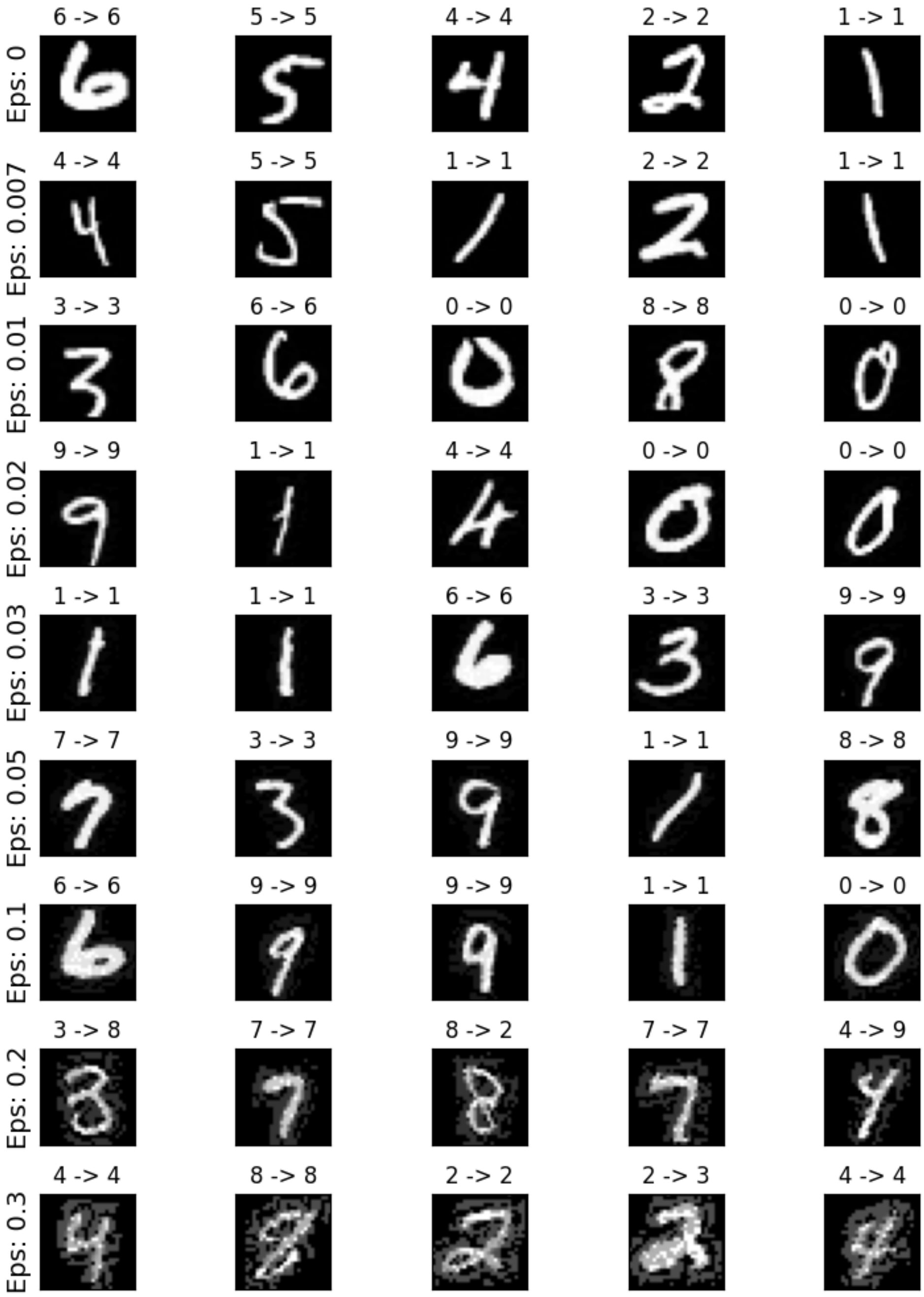
```
def test(model,device,test_loader,epsilon,attack):
  correct = 0
  adv_examples = []
  for data, target in test_loader:
    data, target = data.to(device), target.to(device)
    data.requires_grad = True
    output = model(data)
    init_pred = output.max(1, keepdim=True)[1]
    if init_pred.item() != 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: {}\tTest Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader), final_acc))
  return final_acc, adv_examples
```

12) Построим графики успешности атак (Accuracy/epsilon) и примеры выполненных атак в зависимости от степени возмущения epsilonфункции атак.

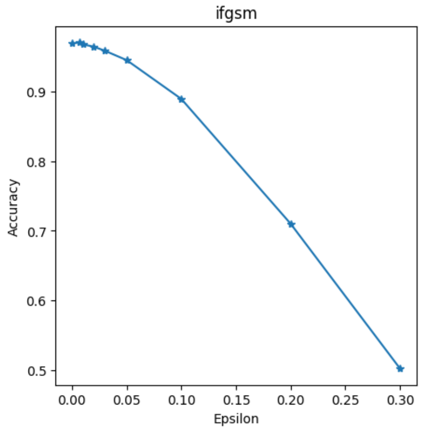
```
epsilons = [0,0.007,0.01,0.02,0.03,0.05,0.1,0.2,0.3]
for attack in ("fgsm","ifgsm","mifgsm"):
  accuracies = []
  examples = []
  for eps in epsilons:
    acc, ex = test(model, device,test_loader,eps,attack)
    accuracies.append(acc)
    examples.append(ex)
  plt.figure(figsize=(5,5))
  plt.plot(epsilons, 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()
```

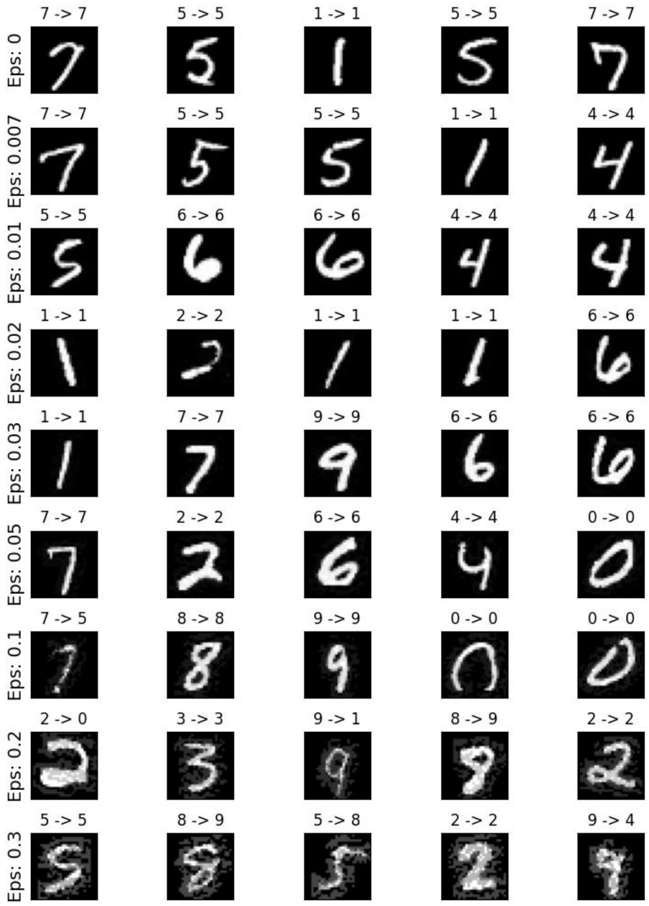
```
Epsilon: 0      Test Accuracy = 9713 / 10000 = 0.9713
Epsilon: 0.007  Test Accuracy = 9672 / 10000 = 0.9672
Epsilon: 0.01   Test Accuracy = 9667 / 10000 = 0.9667
Epsilon: 0.02   Test Accuracy = 9619 / 10000 = 0.9619
Epsilon: 0.03   Test Accuracy = 9560 / 10000 = 0.956
Epsilon: 0.05   Test Accuracy = 9374 / 10000 = 0.9374
Epsilon: 0.1    Test Accuracy = 8758 / 10000 = 0.8758
Epsilon: 0.2    Test Accuracy = 6608 / 10000 = 0.6608
Epsilon: 0.3    Test Accuracy = 4431 / 10000 = 0.4431
```


fgsm
Accuracy
Epsilon
0.5
0.6
0.7
0.8
0.9
0.00
0.05
0.10
0.15
0.20
0.25
0.30

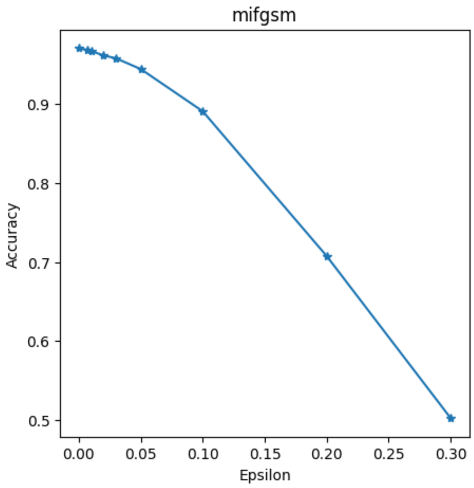

Eps: 0
6 -> 6
5 -> 5
4 -> 4
2 -> 2
1 -> 1
Eps: 0.007
4 -> 4
5 -> 5
1 -> 1
2 -> 2
1 -> 1
Eps: 0.01
3 -> 3
6 -> 6
0 -> 0
8 -> 8
0 -> 0
Eps: 0.02
9 -> 9
1 -> 1
4 -> 4
0 -> 0
0 -> 0
Eps: 0.03
1 -> 1
1 -> 1
6 -> 6
3 -> 3
9 -> 9
Eps: 0.05
7 -> 7
3 -> 3
9 -> 9
1 -> 1
8 -> 8
Eps: 0.1
6 -> 6
9 -> 9
9 -> 9
1 -> 1
0 -> 0
Eps: 0.2
3 -> 8
7 -> 7
8 -> 2
7 -> 7
4 -> 9
Eps: 0.3
4 -> 4
8 -> 8
2 -> 2
2 -> 3
4 -> 4

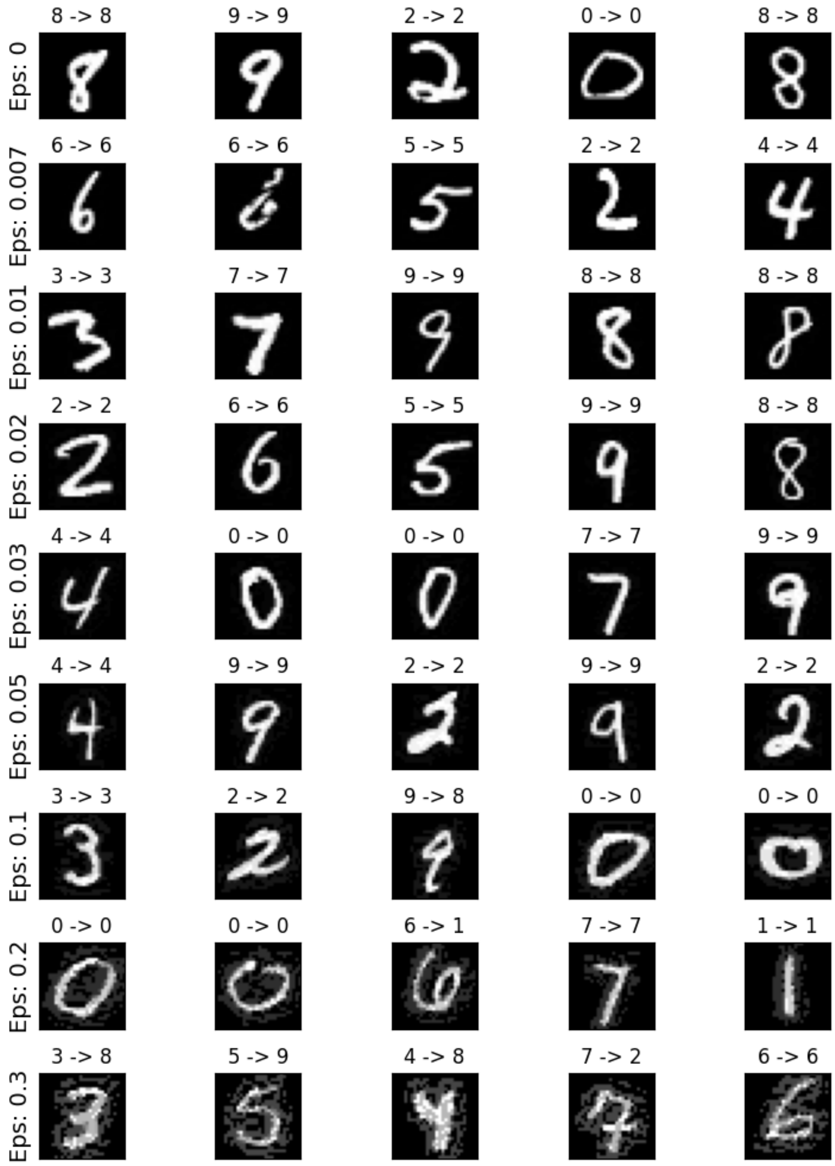
```
Epsilon: 0      Test Accuracy = 9693 / 10000 = 0.9693
Epsilon: 0.007  Test Accuracy = 9709 / 10000 = 0.9709
Epsilon: 0.01   Test Accuracy = 9680 / 10000 = 0.968
Epsilon: 0.02   Test Accuracy = 9647 / 10000 = 0.9647
Epsilon: 0.03   Test Accuracy = 9589 / 10000 = 0.9589
Epsilon: 0.05   Test Accuracy = 9454 / 10000 = 0.9454
Epsilon: 0.1    Test Accuracy = 8896 / 10000 = 0.8896
Epsilon: 0.2    Test Accuracy = 7102 / 10000 = 0.7102
Epsilon: 0.3    Test Accuracy = 5021 / 10000 = 0.5021
```





```
Epsilon: 0      Test Accuracy = 9711 / 10000 = 0.9711
Epsilon: 0.007  Test Accuracy = 9686 / 10000 = 0.9686
Epsilon: 0.01   Test Accuracy = 9674 / 10000 = 0.9674
Epsilon: 0.02   Test Accuracy = 9617 / 10000 = 0.9617
Epsilon: 0.03   Test Accuracy = 9581 / 10000 = 0.9581
Epsilon: 0.05   Test Accuracy = 9445 / 10000 = 0.9445
Epsilon: 0.1    Test Accuracy = 8910 / 10000 = 0.891
Epsilon: 0.2    Test Accuracy = 7076 / 10000 = 0.7076
Epsilon: 0.3    Test Accuracy = 5023 / 10000 = 0.5023
```





13) Создадим 2 класса НС

```
[15] 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
```

14) 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: {}\tTest Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader), final_acc))
  return final_acc,adv_examples
```

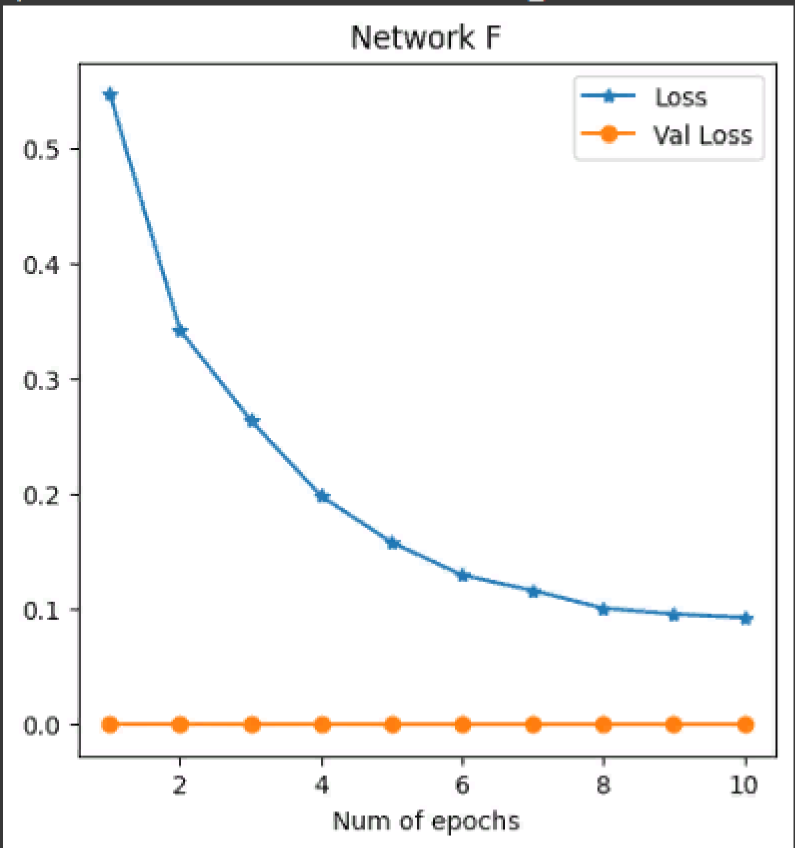
15) Создадим функцию защиты методом дистилляции

```
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,"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()
```

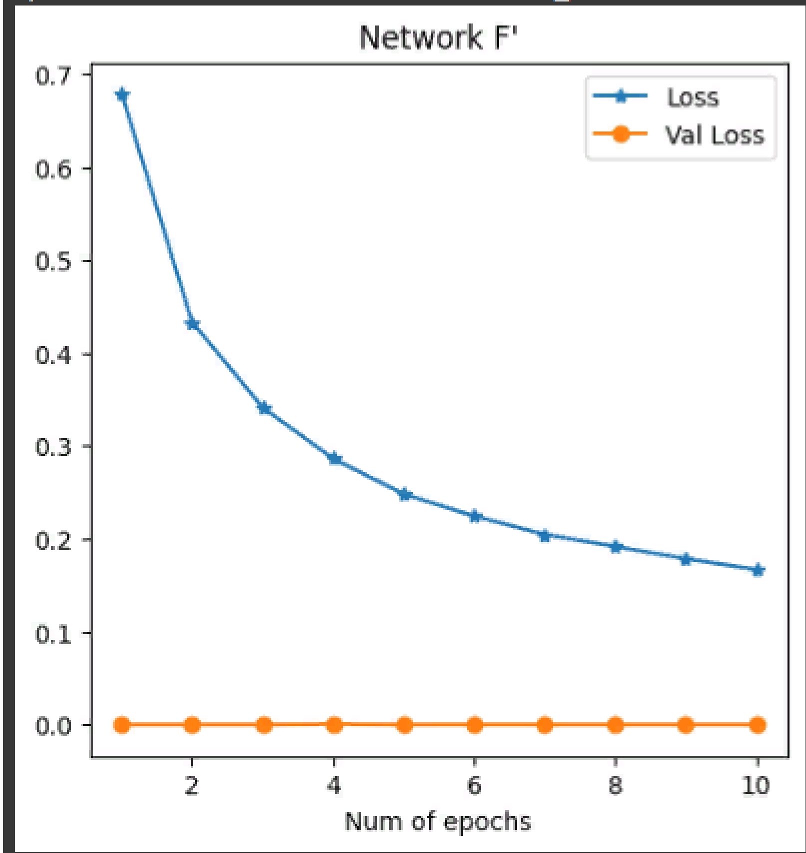
16) Получаем результаты оценки защищенных сетей

```
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)
```

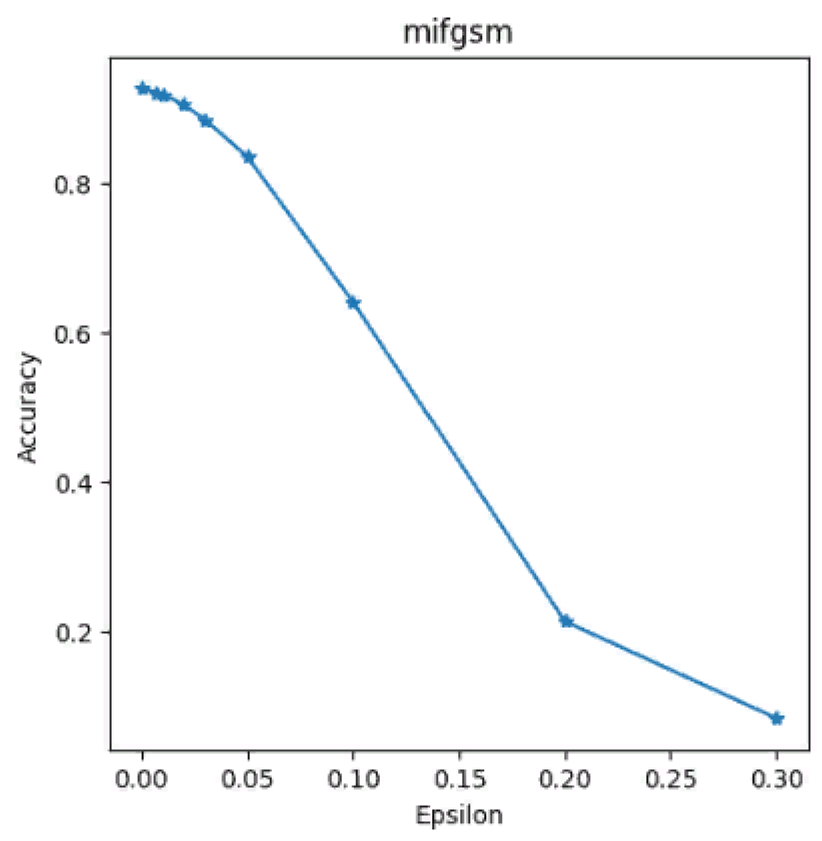
```
Fitting the model...
Epoch: 1 Loss: 0.5471096785045427 Val_Loss: 0.00011348691284656524
Epoch: 2 Loss: 0.3424423505676849 Val_Loss: 0.0001497186660766016
Epoch: 3 Loss: 0.2644242537693425 Val_Loss: 7.489708659704775e-08
Epoch: 4 Loss: 0.19839789901623245 Val_Loss: 6.843128139153123e-05
Epoch: 5 Loss: 0.15824629327167197 Val_Loss: 1.3889133901102469e-06
Epoch: 6 Loss: 0.12969237412395931 Val_Loss: 3.480212762951851e-07
Epoch: 7 Loss: 0.1162814704113448 Val_Loss: 4.470553787541576e-07
Epoch: 8 Loss: 0.10094529216267195 Val_Loss: 1.288974701310508e-07
Epoch: 9 Loss: 0.0956556857473766 Val_Loss: 9.73091180890151e-07
Epoch: 10 Loss: 0.0927851360270701 Val_Loss: 1.4781842764932662e-09
```

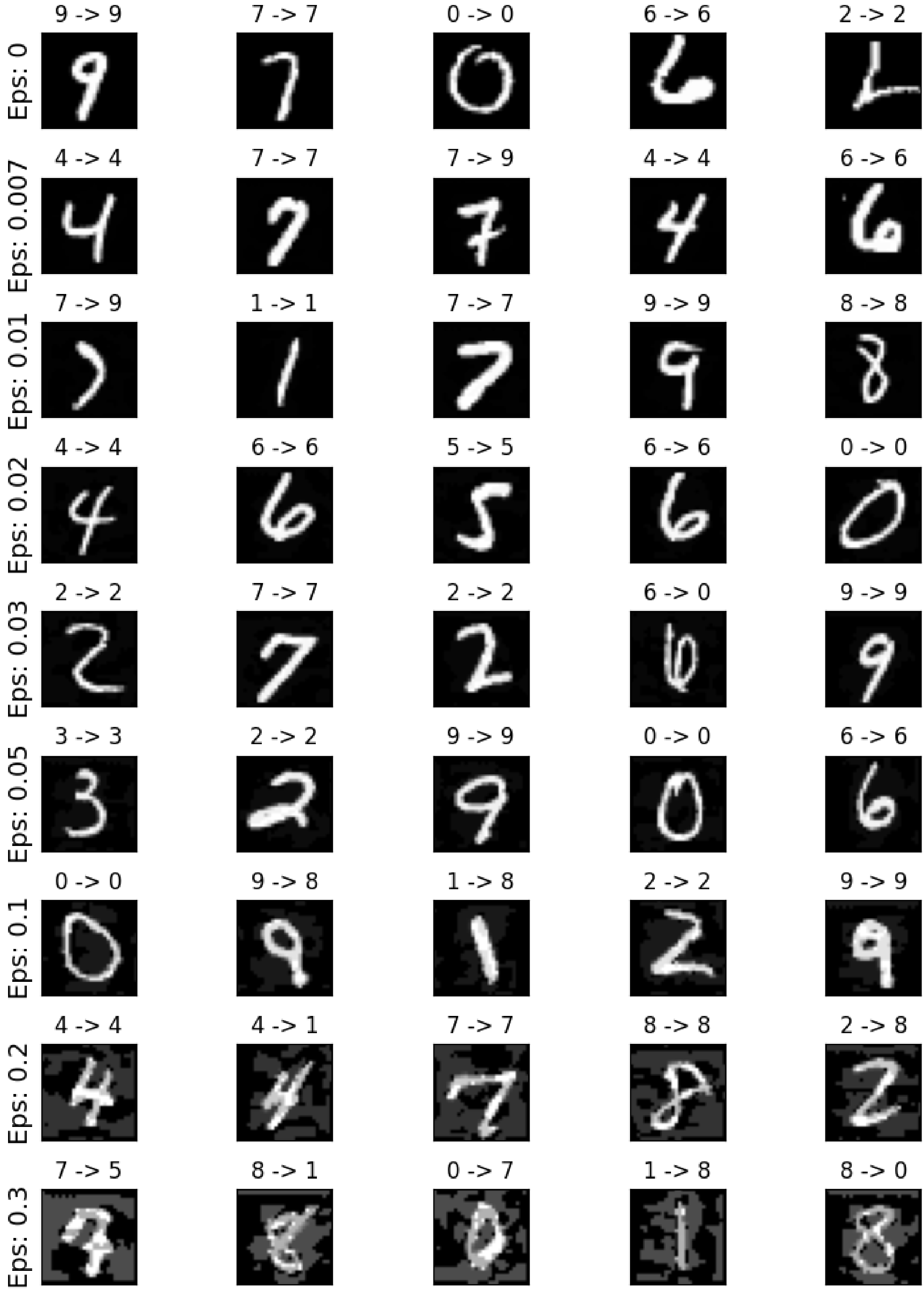


```
Fitting the model...
Epoch: 1 Loss: 0.6780537456616875 Val_Loss: 7.1262997458688916e-06
Epoch: 2 Loss: 0.4332273680179977 Val_Loss: 1.2334131635725497e-06
Epoch: 3 Loss: 0.34130754910211164 Val_Loss: 9.883328247815371e-06
Epoch: 4 Loss: 0.28634525036554204 Val_Loss: 0.00043269464969632735
Epoch: 5 Loss: 0.24822499183999236 Val_Loss: 1.169161032885313e-06
Epoch: 6 Loss: 0.22449985656291235 Val_Loss: 4.936303216964006e-05
Epoch: 7 Loss: 0.20481251181569646 Val_Loss: 2.138733843166847e-05
Epoch: 8 Loss: 0.19161339320650952 Val_Loss: 3.6941005382686854e-07
Epoch: 9 Loss: 0.17854869775602863 Val_Loss: 1.59674302150961e-05
Epoch: 10 Loss: 0.16692460639016518 Val_Loss: 9.566251161231776e-07
```



```
Epsilon: 0      Test Accuracy = 9246 / 10000 = 0.9246
Epsilon: 0.007  Test Accuracy = 9200 / 10000 = 0.92
Epsilon: 0.01   Test Accuracy = 9213 / 10000 = 0.9213
Epsilon: 0.02   Test Accuracy = 9070 / 10000 = 0.907
Epsilon: 0.03   Test Accuracy = 8824 / 10000 = 0.8824
Epsilon: 0.05   Test Accuracy = 8388 / 10000 = 0.8388
Epsilon: 0.1    Test Accuracy = 6328 / 10000 = 0.6328
Epsilon: 0.2    Test Accuracy = 2095 / 10000 = 0.2095
Epsilon: 0.3    Test Accuracy = 843 / 10000 = 0.0843
Epsilon: 0      Test Accuracy = 9270 / 10000 = 0.927
Epsilon: 0.007  Test Accuracy = 9189 / 10000 = 0.9189
Epsilon: 0.01   Test Accuracy = 9221 / 10000 = 0.9221
Epsilon: 0.02   Test Accuracy = 9033 / 10000 = 0.9033
Epsilon: 0.03   Test Accuracy = 8843 / 10000 = 0.8843
Epsilon: 0.05   Test Accuracy = 8395 / 10000 = 0.8395
Epsilon: 0.1    Test Accuracy = 6474 / 10000 = 0.6474
Epsilon: 0.2    Test Accuracy = 2126 / 10000 = 0.2126
Epsilon: 0.3    Test Accuracy = 789 / 10000 = 0.0789
Epsilon: 0      Test Accuracy = 9277 / 10000 = 0.9277
Epsilon: 0.007  Test Accuracy = 9222 / 10000 = 0.9222
Epsilon: 0.01   Test Accuracy = 9195 / 10000 = 0.9195
Epsilon: 0.02   Test Accuracy = 9054 / 10000 = 0.9054
Epsilon: 0.03   Test Accuracy = 8856 / 10000 = 0.8856
Epsilon: 0.05   Test Accuracy = 8361 / 10000 = 0.8361
Epsilon: 0.1    Test Accuracy = 6418 / 10000 = 0.6418
Epsilon: 0.2    Test Accuracy = 2141 / 10000 = 0.2141
Epsilon: 0.3    Test Accuracy = 845 / 10000 = 0.0845
```





Eps: 0
9 -> 9
7 -> 7
0 -> 0
6 -> 6
2 -> 2
Eps: 0.007
4 -> 4
7 -> 7
7 -> 9
4 -> 4
6 -> 6
Eps: 0.01
7 -> 9
1 -> 1
7 -> 7
9 -> 9
8 -> 8
Eps: 0.02
4 -> 4
6 -> 6
5 -> 5
6 -> 6
0 -> 0
Eps: 0.03
2 -> 2
7 -> 7
2 -> 2
6 -> 0
9 -> 9
Eps: 0.05
3 -> 3
2 -> 2
9 -> 9
0 -> 0
6 -> 6
Eps: 0.1
0 -> 0
9 -> 8
1 -> 8
2 -> 2
9 -> 9
Eps: 0.2
4 -> 4
4 -> 1
7 -> 7
8 -> 8
2 -> 8
Eps: 0.3
7 -> 5
8 -> 1
0 -> 7
1 -> 8
8 -> 0

Вывод

В ходе исследования метода защитной дистилляции от атак на нейронные сети (НС) были выполнены следующие этапы:

1. Определены преобразования для нормализации данных в наборе MNIST.
2. Подготовлена и обучена нейронная сеть на основе фреймворка torch.
3. Реализованы атаки FGSM, I-FGSM, MI-FGSM, и оценена их эффективность.
4. Созданы два класса НС с переопределенными функциями обучения и тестирования.
5. Разработана функция защиты с применением метода дистилляции.
6. Оценены результаты работы защищенных сетей.

Использование защитной дистилляции заметно укрепило устойчивость нашей модели к атакам, поднимая точность атакованной модели. Этот результат не только подчеркивает важность защитной дистилляции в контексте повышения безопасности нейронных сетей, но и подчеркивает необходимость учета различных параметров при выборе и применении данного метода в исследование.