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

по дисциплине

«Анализ защищённости систем искусственного интеллекта»

Группа:
ББМО-01-22
Выполнила:
Огольцова Н.Д.

Проверил:
Спирин А.А.

Москва 2023

Ход работы:

1)Выполним загрузку необходимых библиотек;

```
[1]  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
```

2)Задаём нормализующие преобразования и подгружаем датасет MNIST, разбиваем данные и выводим получившиеся значения;

```
[2]  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))
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 120948579.17it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 113316832.39it/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 22928621.76it/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 19990061.67it/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw

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

3)Создаём класс НС на основе фреймворка torch и проверяем его работоспособность;

```python
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.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)
    output = F.log_softmax(x, dim=1)
    return output

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

4) Создаём сначала оптимизатор, функцию потерь и потом трейнер сети;

```python
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)
```

5)Создадим функцию для обучения сети;

```python
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)
        #Вычисление выходных потерь
        loss = criterion(output,label)
        if phase == 'train':
          optimizer.zero_grad()
          #grad calc w.r.t Loss func
          loss.backward()
          #Обновление весов
          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
```

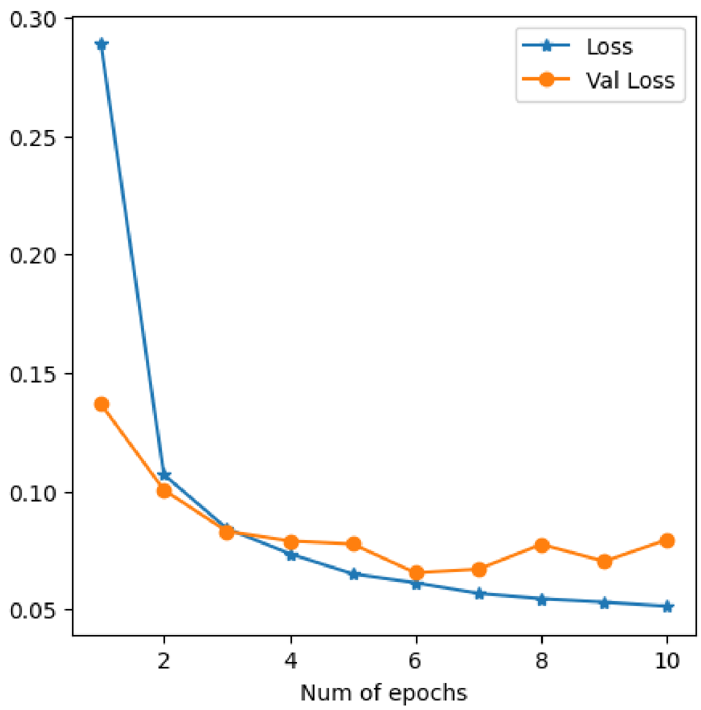
6)Обучим модель и выведем получившиеся значения;

```
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: User
  warnings.warn(warn_msg)
Epoch: 1 Loss: 0.2888687746945781 Val_Loss: 0.13673646724928057
Epoch: 2 Loss: 0.10697086884837405 Val_Loss: 0.10052042624421069
Epoch: 3 Loss: 0.08423662934597663 Val_Loss: 0.0830532077599337
Epoch: 4 Loss: 0.0735847675861095 Val_Loss: 0.07899737361183146
Epoch: 5 Loss: 0.06507188670661525 Val_Loss: 0.07766588417880915
Epoch: 6 Loss: 0.061304444135245634 Val_Loss: 0.06547614247407182
Epoch: 7 Loss: 0.056818889675883055 Val_Loss: 0.06705326536982777
Epoch: 8 Loss: 0.05457629382752491 Val_Loss: 0.07752687670957667
Epoch: 9 Loss: 0.053148277264218544 Val_Loss: 0.07033568611625667
Epoch: 10 Loss: 0.051392144952551407 Val_Loss: 0.07953557262959682
```

7)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()
```



8)Создадим функции атак FGSM, I-FGSM, MI-FGSM;

```python
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
```

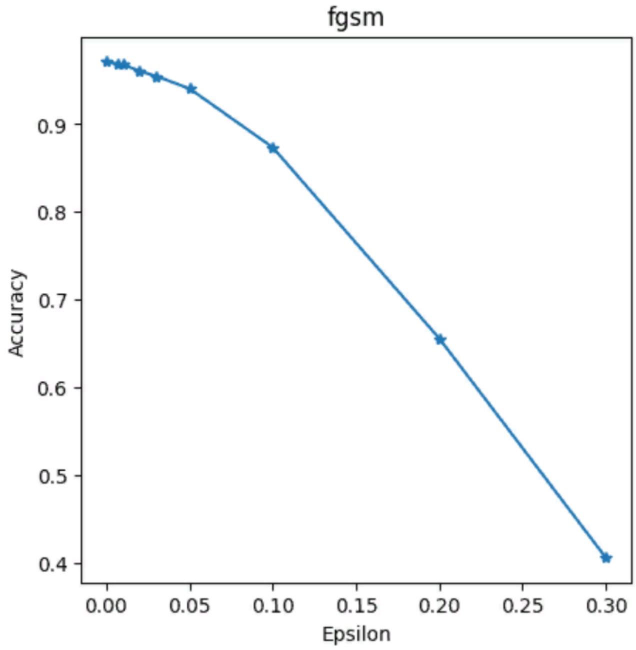
9)Создадим функцию тестирования/проверки;

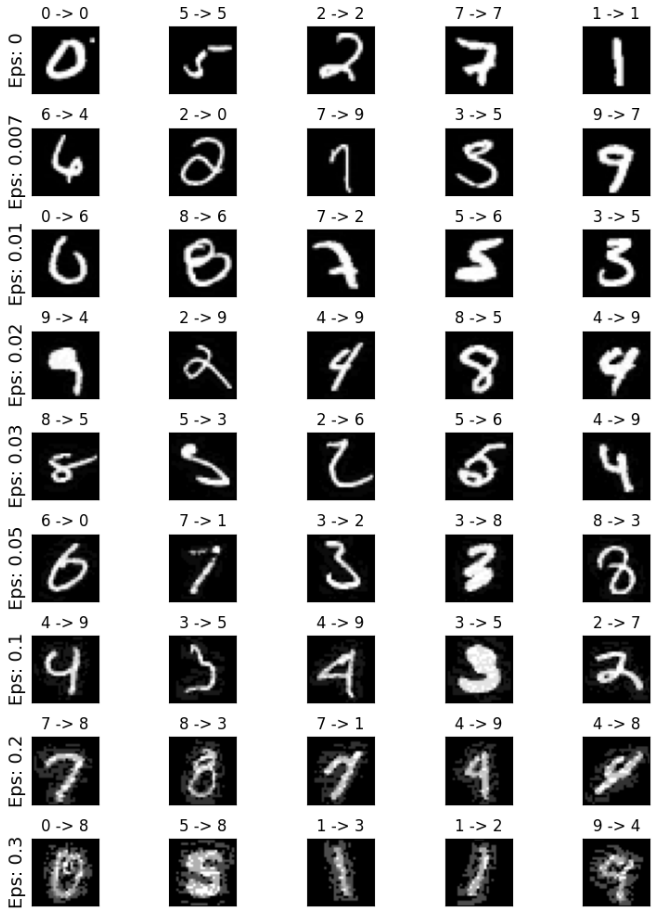
```
def test(model,device,test_loader,epsilon,attack):
    correct = 0
    adv_examples = []
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        data.requires_grad = True
        output = model(data)
        init_pred = output.max(1, keepdim=True)[1]
        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
```

10) Построим графики accuracy атак и выведем примеры выполненных атак в зависимости от значения epsilon;

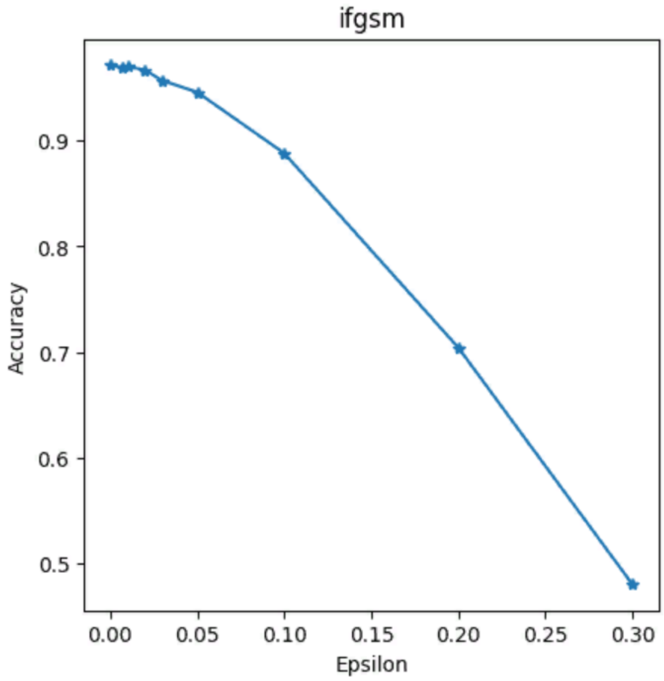
```
epsilons = [0,0.007,0.01,0.02,0.03,0.05,0.1,0.2,0.3]
for attack in ("fgsm","ifgsm","mifgsm"):
  accuracies = []
  examples = []
  for eps in epsilons:
    acc, ex = test(model, device,test_loader,eps,attack)
    accuracies.append(acc)
    examples.append(ex)

  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()
```

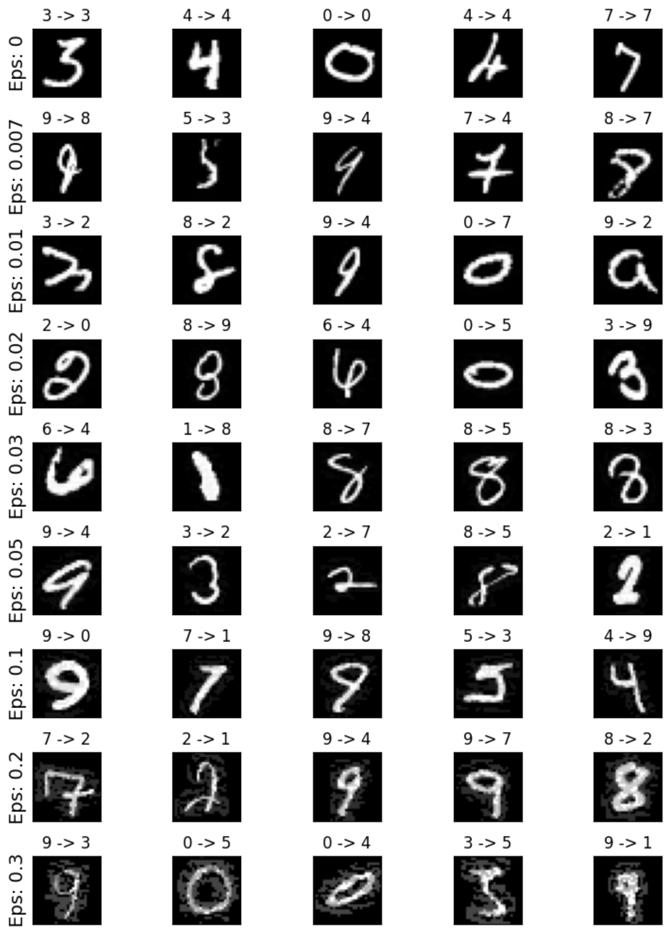
```
Epsilon: 0      Test Accuracy = 9713 / 10000 = 0.9713
Epsilon: 0.007  Test Accuracy = 9685 / 10000 = 0.9685
Epsilon: 0.01   Test Accuracy = 9680 / 10000 = 0.968
Epsilon: 0.02   Test Accuracy = 9604 / 10000 = 0.9604
Epsilon: 0.03   Test Accuracy = 9547 / 10000 = 0.9547
Epsilon: 0.05   Test Accuracy = 9405 / 10000 = 0.9405
Epsilon: 0.1    Test Accuracy = 8735 / 10000 = 0.8735
Epsilon: 0.2    Test Accuracy = 6559 / 10000 = 0.6559
Epsilon: 0.3    Test Accuracy = 4063 / 10000 = 0.4063
```



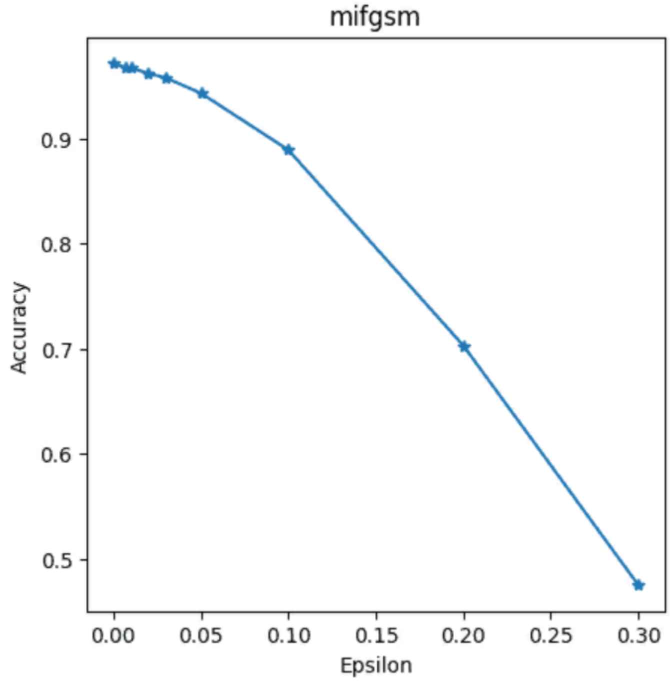


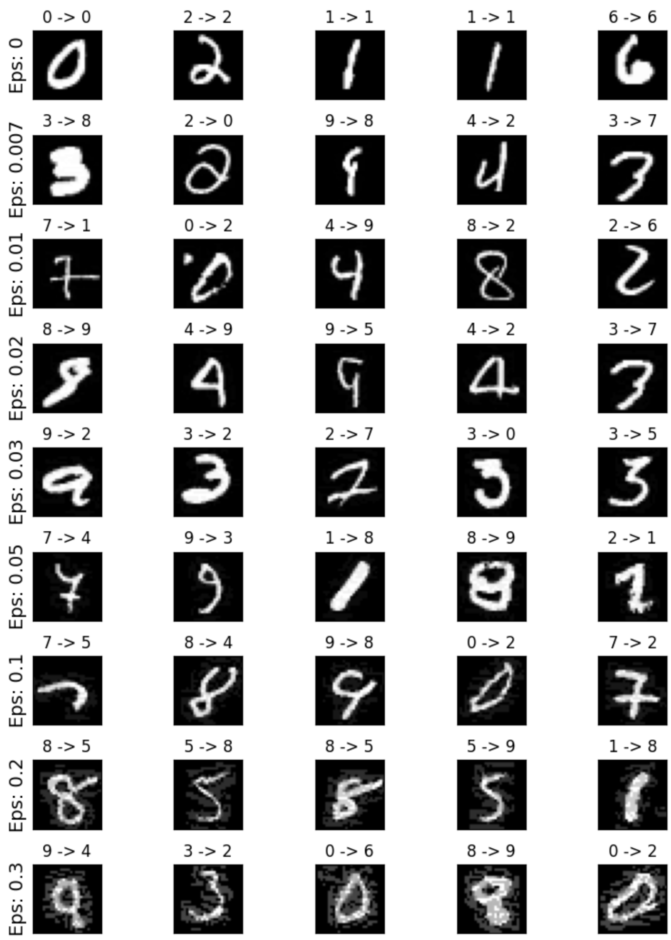

```
Epsilon: 0      Test Accuracy = 9715 / 10000 = 0.9715
Epsilon: 0.007  Test Accuracy = 9693 / 10000 = 0.9693
Epsilon: 0.01   Test Accuracy = 9701 / 10000 = 0.9701
Epsilon: 0.02   Test Accuracy = 9667 / 10000 = 0.9667
Epsilon: 0.03   Test Accuracy = 9567 / 10000 = 0.9567
Epsilon: 0.05   Test Accuracy = 9458 / 10000 = 0.9458
Epsilon: 0.1    Test Accuracy = 8880 / 10000 = 0.888
Epsilon: 0.2    Test Accuracy = 7042 / 10000 = 0.7042
Epsilon: 0.3    Test Accuracy = 4804 / 10000 = 0.4804
```





```
Epsilon: 0      Test Accuracy = 9720 / 10000 = 0.972
Epsilon: 0.007  Test Accuracy = 9676 / 10000 = 0.9676
Epsilon: 0.01   Test Accuracy = 9676 / 10000 = 0.9676
Epsilon: 0.02   Test Accuracy = 9623 / 10000 = 0.9623
Epsilon: 0.03   Test Accuracy = 9580 / 10000 = 0.958
Epsilon: 0.05   Test Accuracy = 9435 / 10000 = 0.9435
Epsilon: 0.1    Test Accuracy = 8894 / 10000 = 0.8894
Epsilon: 0.2    Test Accuracy = 7031 / 10000 = 0.7031
Epsilon: 0.3    Test Accuracy = 4751 / 10000 = 0.4751
```





11)Создадим 2 класса НС;

```python
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
```

```python
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
```

12)Переопределим с учётом этого 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)
        #расчет потерь на выходе
        loss = criterion(output,label)
        if phase == 'train':
          optimizer.zero_grad()
          #grad calc w.r.t Loss func
          loss.backward()
          #обновление весов
          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
```

13)Создадим функцию защиты методом дистилляции;

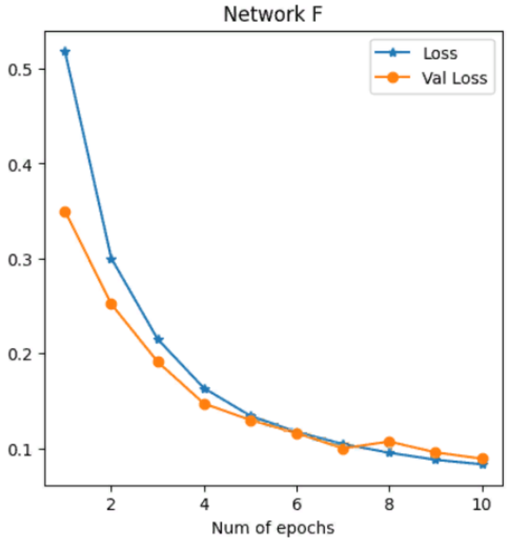
```python
def defense(device,train_loader,val_loader,test_loader,epochs,Temp,epsilons):

  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()

  #Преобразование целевых меток в гибкие метки
  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
```

```python
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()
```

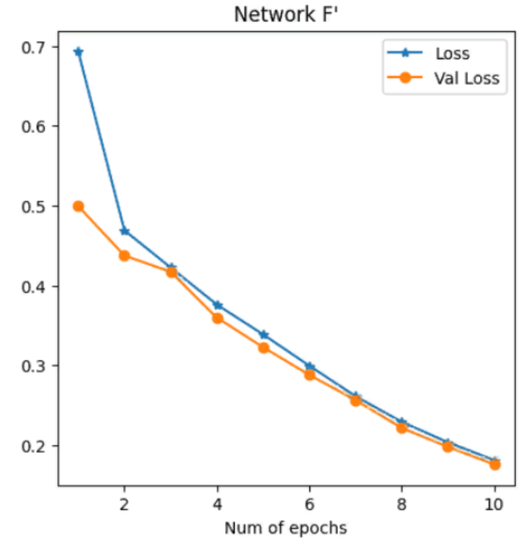
14)Получим результаты оценки уже защищенных сетей.

```
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)
```

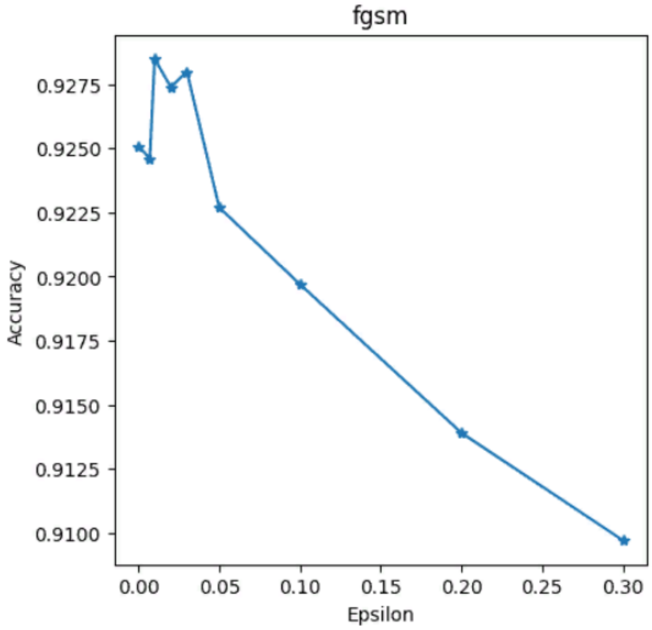
```
Fitting the model...
Epoch: 1 Loss: 0.5183894712342154 Val_Loss: 0.35013818526350576
Epoch: 2 Loss: 0.3003614776288117 Val_Loss: 0.252035639292655
Epoch: 3 Loss: 0.21513449889052666 Val_Loss: 0.19123540499878997
Epoch: 4 Loss: 0.16343666033960325 Val_Loss: 0.14716214833448865
Epoch: 5 Loss: 0.13425891620466573 Val_Loss: 0.12984373547466652
Epoch: 6 Loss: 0.11724456572161095 Val_Loss: 0.11608159823280148
Epoch: 7 Loss: 0.10449432968694738 Val_Loss: 0.09994257388982973
Epoch: 8 Loss: 0.0953535752760143 Val_Loss: 0.10708337319088777
Epoch: 9 Loss: 0.08788794863414845 Val_Loss: 0.09572349952923845
Epoch: 10 Loss: 0.08311829223801133 Val_Loss: 0.08903677334889062
```

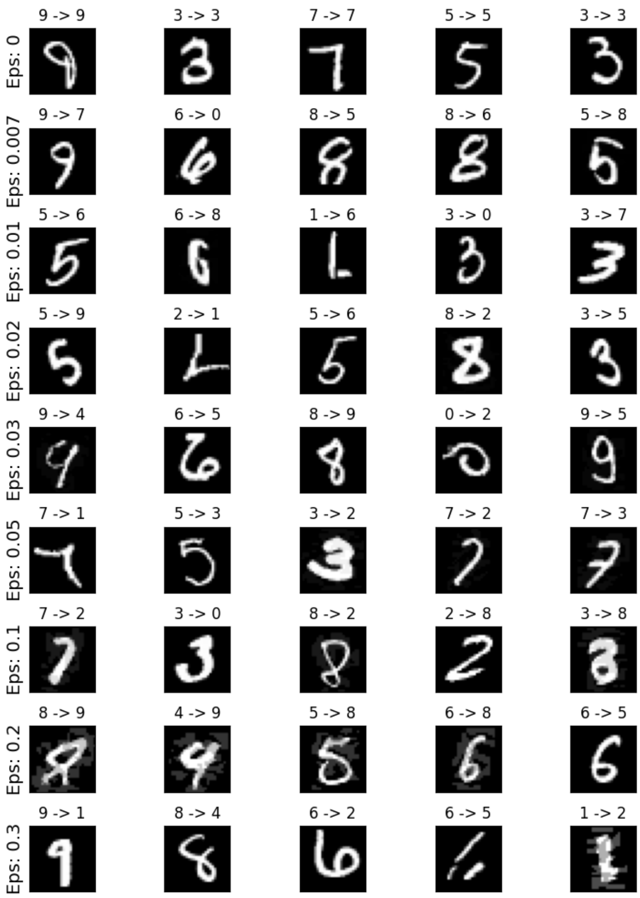



```
Fitting the model...
Epoch: 1 Loss: 0.6932818498760024 Val_Loss: 0.49934649001323944
Epoch: 2 Loss: 0.46901066487247844 Val_Loss: 0.4374905162987688
Epoch: 3 Loss: 0.422528915409889 Val_Loss: 0.41707982867373905
Epoch: 4 Loss: 0.37601292758400523 Val_Loss: 0.3597610071243918
Epoch: 5 Loss: 0.3387371742089258 Val_Loss: 0.32266275289950297
Epoch: 6 Loss: 0.2994880652114021 Val_Loss: 0.28805370525927454
Epoch: 7 Loss: 0.26132685050939375 Val_Loss: 0.2564337362716436
Epoch: 8 Loss: 0.22919203091141782 Val_Loss: 0.22150387843409927
Epoch: 9 Loss: 0.20354706166494133 Val_Loss: 0.19780019457405493
Epoch: 10 Loss: 0.18127455468622433 Val_Loss: 0.1758993640845126
```

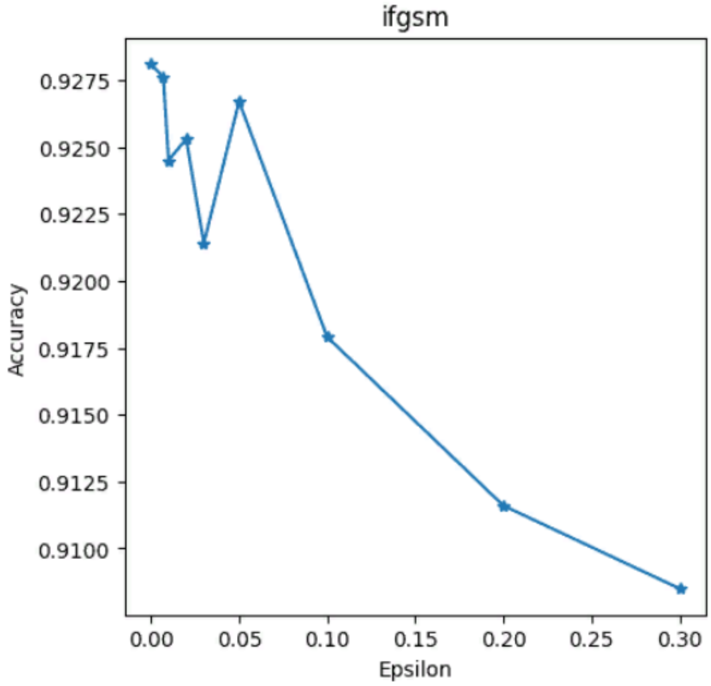


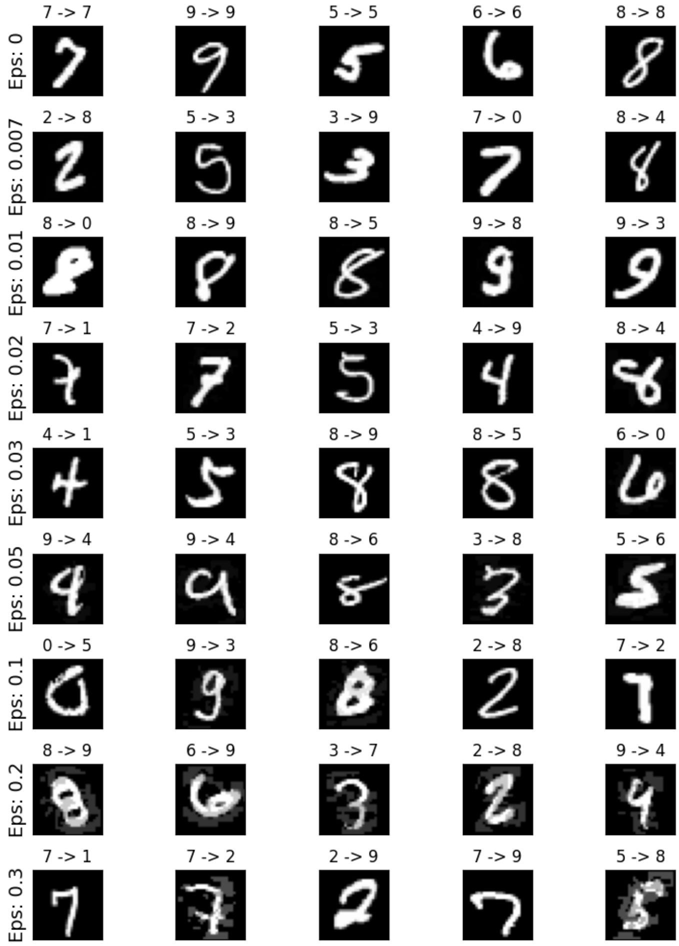

```
Epsilon: 0      Test Accuracy = 9251 / 10000 = 0.9251
Epsilon: 0.007  Test Accuracy = 9246 / 10000 = 0.9246
Epsilon: 0.01   Test Accuracy = 9285 / 10000 = 0.9285
Epsilon: 0.02   Test Accuracy = 9274 / 10000 = 0.9274
Epsilon: 0.03   Test Accuracy = 9280 / 10000 = 0.928
Epsilon: 0.05   Test Accuracy = 9227 / 10000 = 0.9227
Epsilon: 0.1    Test Accuracy = 9197 / 10000 = 0.9197
Epsilon: 0.2    Test Accuracy = 9139 / 10000 = 0.9139
Epsilon: 0.3    Test Accuracy = 9097 / 10000 = 0.9097
```



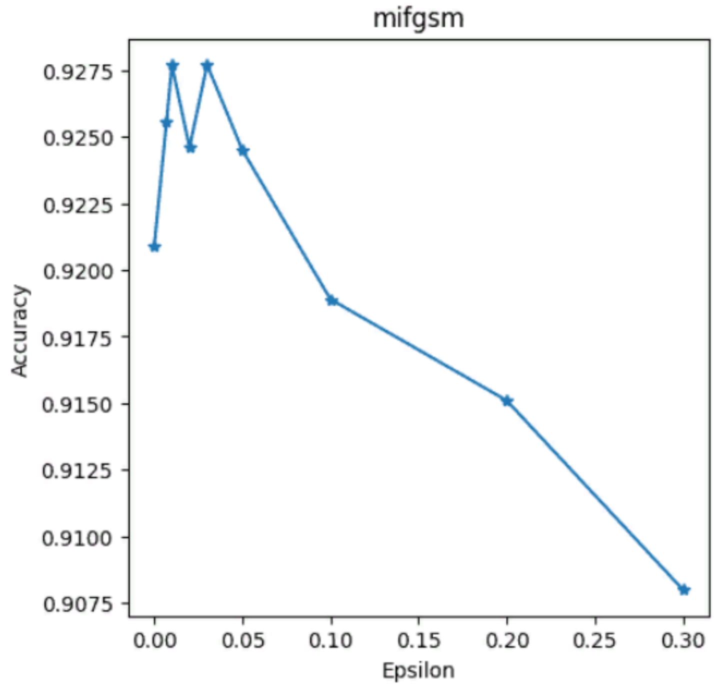


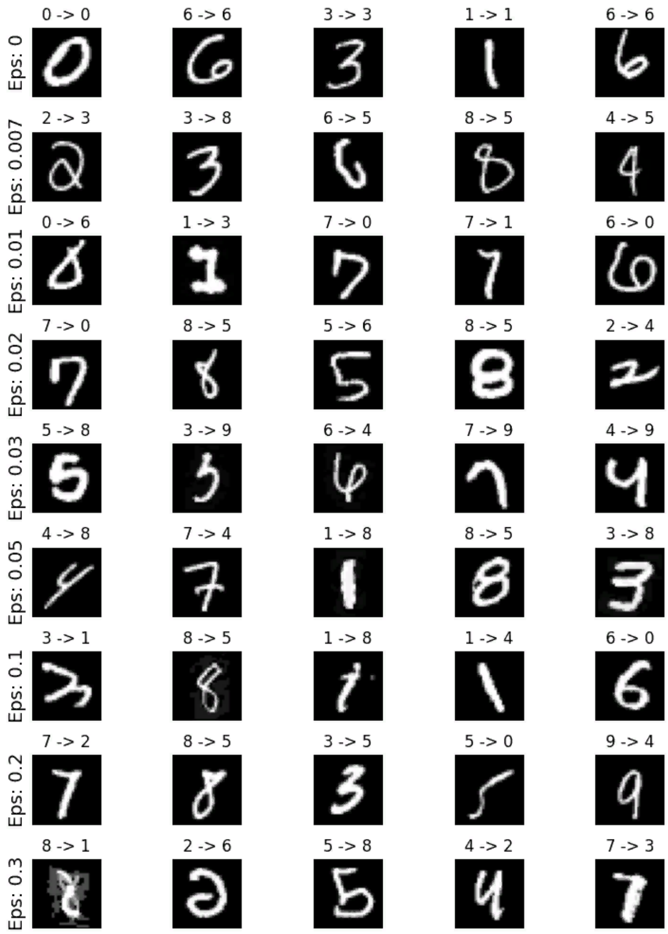
```
Epsilon: 0      Test Accuracy = 9281 / 10000 = 0.9281
Epsilon: 0.007  Test Accuracy = 9276 / 10000 = 0.9276
Epsilon: 0.01   Test Accuracy = 9245 / 10000 = 0.9245
Epsilon: 0.02   Test Accuracy = 9253 / 10000 = 0.9253
Epsilon: 0.03   Test Accuracy = 9214 / 10000 = 0.9214
Epsilon: 0.05   Test Accuracy = 9267 / 10000 = 0.9267
Epsilon: 0.1    Test Accuracy = 9179 / 10000 = 0.9179
Epsilon: 0.2    Test Accuracy = 9116 / 10000 = 0.9116
Epsilon: 0.3    Test Accuracy = 9085 / 10000 = 0.9085
```






```
Epsilon: 0      Test Accuracy = 9209 / 10000 = 0.9209
Epsilon: 0.007  Test Accuracy = 9256 / 10000 = 0.9256
Epsilon: 0.01   Test Accuracy = 9277 / 10000 = 0.9277
Epsilon: 0.02   Test Accuracy = 9246 / 10000 = 0.9246
Epsilon: 0.03   Test Accuracy = 9277 / 10000 = 0.9277
Epsilon: 0.05   Test Accuracy = 9245 / 10000 = 0.9245
Epsilon: 0.1    Test Accuracy = 9189 / 10000 = 0.9189
Epsilon: 0.2    Test Accuracy = 9151 / 10000 = 0.9151
Epsilon: 0.3    Test Accuracy = 9080 / 10000 = 0.908
```





Вывод:

При увеличении значения переменной epsilon у атак - точность падает практически вдвое, падает от значения 0.97 в среднем до 0.45.

Тогда как при атаке на защищённые сети значение остаётся в пределах от 0.92 - 0.90.

Также можно заметить, что у защищённых сетей (при значении epsilon = 0 -> атака отсутствует) значение точности снизилось.