МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

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

Отчёт по лабораторной работе №4

по дисциплине «Анализ защищенности систем искусственного интеллекта»

Выполнил:
Евдокимов А.М.
Группа: ББМО-02-23

Москва - 2024

Изучение методов защиты от атак на модели НС

Защитная дистилляция

1. Выполнить импорт 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. Загрузим набор данных (MNIST), разобьем данные на подвыборки

```
[3] transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.0,), (1.0,))])

[5] 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)

[7] 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)

[8] print("Длина обучающей выборки:", len(train_loader),"\nДлина валидационной выборки:", len(val_loader), "\nДлина тестовой выборки:", len(test_loader))

    Длина обучающей выборки: 50000
    Длина валидационной выборки: 10000
    Длина тестовой выборки: 10000
```

3. Настроим использование графического ускорителя

```
[2] print(torch.cuda.is_available())

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

    True
```

Создание атак на модель НС

4. Создать класс НС на основе фреймворка torch и проверяем работоспособность созданного класса НС

```
[9]  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

[10] model = Net().to(device)
```

5. Создать оптимизатор, функцию потерь и трейнер сети

```
[11] 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)
```

6. Определить функцию обучения сети и обучаем модель

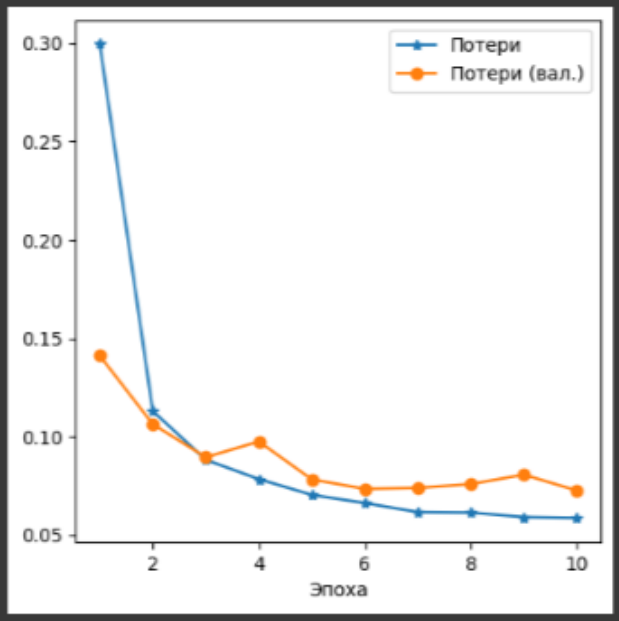
```
[12] def fit(model, device, train_loader, val_loader, epochs):
       data_loader = {'train':train_loader,'val':val_loader}
       print("Обучение модели...")

       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)

             loss = criterion(output, label)
             if phase == 'train':
               optimizer.zero_grad()
               loss.backward()
               optimizer.step()
               loss_per_epoch += loss.item()
             else:
               val_loss_per_epoch+=loss.item()

         scheduler.step(val_loss_per_epoch/len(val_loader))

         print("Эпоха: {} Потери: {} Потери (валидация): {}".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
```

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

```
Обучение модели...
/usr/local/lib/python3.10/dist-packages/torch/nn/functional.py:1538: UserWarning: dropout2d: Received a 2-D input to dropout2d, which is deprecat
  warnings.warn(warn_msg)
Эпоха: 1 Потери: 0.2996477483677342 Потери (валидация): 0.1410833359215744
Эпоха: 2 Потери: 0.11294136712072304 Потери (валидация): 0.10629208624610906
Эпоха: 3 Потери: 0.0881666676220506 Потери (валидация): 0.08948061989744742
Эпоха: 4 Потери: 0.07837011130685294 Потери (валидация): 0.09748582112996391
Эпоха: 5 Потери: 0.07033927268202717 Потери (валидация): 0.07821329451522255
Эпоха: 6 Потери: 0.06619215334130284 Потери (валидация): 0.07338837133674139
Эпоха: 7 Потери: 0.06158620706369628 Потери (валидация): 0.07389724288982027
Эпоха: 8 Потери: 0.06134560508645621 Потери (валидация): 0.07583817343227783
Эпоха: 9 Потери: 0.059072282040114485 Потери (валидация): 0.08063062749507208
Эпоха: 10 Потери: 0.05860689949951895 Потери (валидация): 0.07253395374084795
```

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

```
[14] fig = plt.figure(figsize=(5,5))

     plt.plot(np.arange(1, 11), loss, "*-", label="Потери")
     plt.plot(np.arange(1, 11), val_loss, "o-", label="Потери (вал.)")
     plt.xlabel("Эпоха")

     plt.legend()
     plt.show()
```



8. 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
```

9. Создадим функцию проверки

```
[16] 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("Эпсилон: {}\tТочность (тест) = {} / {} = {}".format(epsilon, correct, len(test_loader), final_acc))
       return final_acc, adv_examples
```

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

```
[17] 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("Эпсилон")
       plt.ylabel("Точность")
       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("Эпсилон: {}".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()
```

Защита от атак

11. 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
```

12. Переопределим функцию обучения и тестирования

```
[19] def fit(model,device,optimizer,scheduler,criterion,train_loader,val_loader,Temp,epochs):
       data_loader = {'train':train_loader, 'val':val_loader}
       print("Обучение модели...")

       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)

             loss = criterion(output,label)
             if phase == 'train':
               optimizer.zero_grad()
               loss.backward()
               optimizer.step()
               loss_per_epoch+=loss.item()
           else:
             val_loss_per_epoch+=loss.item()

         scheduler.step(val_loss_per_epoch / len(val_loader))

         print("Эпоха: {} Потери: {} Потери (валидация): {}".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("Эпсилон: {}\tТочность (тест) = {} / {} = {}".format(epsilon, correct, len(test_loader), final_acc))
    return final_acc,adv_examples
```

13. Создадим функцию защиты методом дистилляции

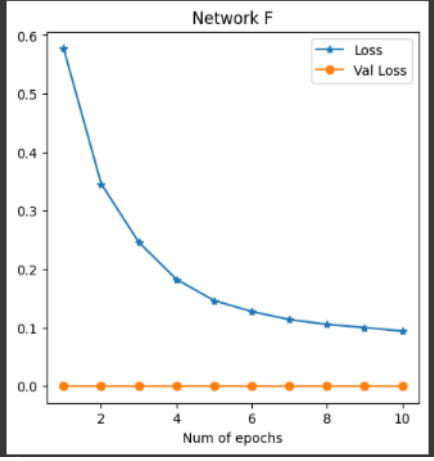
```
[20] def defense(device,train_loader,val_loader,test_loader,epochs,Temp,epsilons):
       modelF = NetF().to(device)

       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()

       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,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()
```

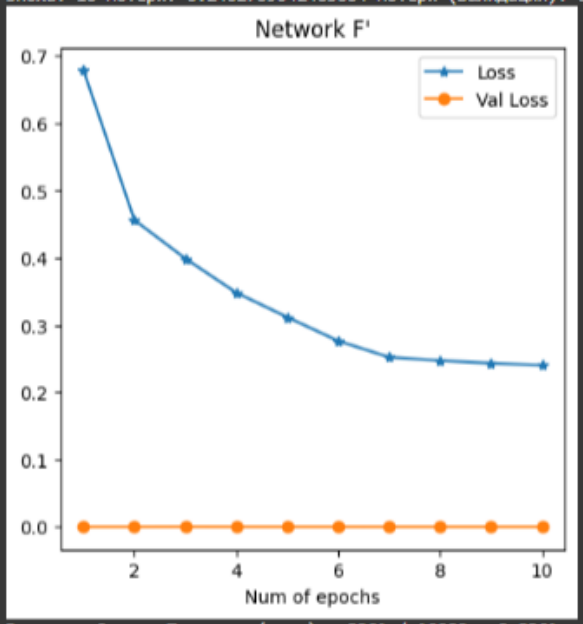
14. Получим результаты оценки защищенных сетей

```
[21] 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)
```

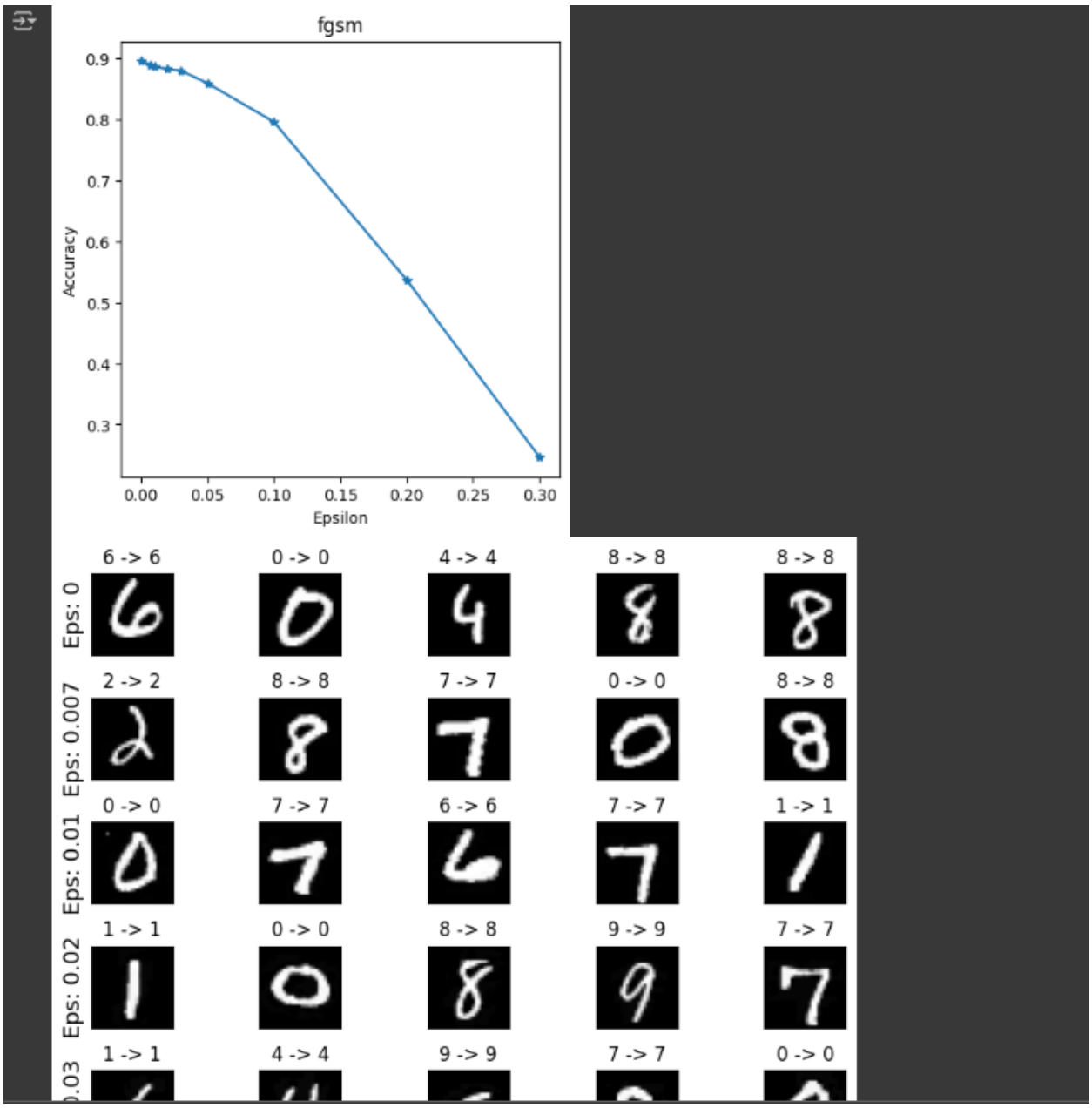
```
Обучение модели...
/usr/local/lib/python3.10/dist-packages/torch/nn/functional.py:1538: UserWarning: dropout2d: Received a 2-D input to dropout2d, which is deprecated
  warnings.warn(warn_msg)
Эпоха: 1 Потери: 0.5772624624919404 Потери (валидация): 3.4157036832766605e-06
Эпоха: 2 Потери: 0.34587279333854554 Потери (валидация): 4.887372255325317e-06
Эпоха: 3 Потери: 0.24588688378951543 Потери (валидация): 2.4973469972610473e-05
Эпоха: 4 Потери: 0.1825507898199169 Потери (валидация): 9.969661361537874e-07
Эпоха: 5 Потери: 0.14628627200671424 Потери (валидация): 0.00017907010652124883
Эпоха: 6 Потери: 0.12753449372560946 Потери (валидация): 7.855731746531091e-09
Эпоха: 7 Потери: 0.1137877332995896 Потери (валидация): 1.8252473261782143e-06
Эпоха: 8 Потери: 0.10548093369547008 Потери (валидация): 1.195599961647531e-08
Эпоха: 9 Потери: 0.10012993453798988 Потери (валидация): 2.0660310474340804e-07
Эпоха: 10 Потери: 0.0940632517359128 Потери (валидация): 2.2776883677579463e-06
```

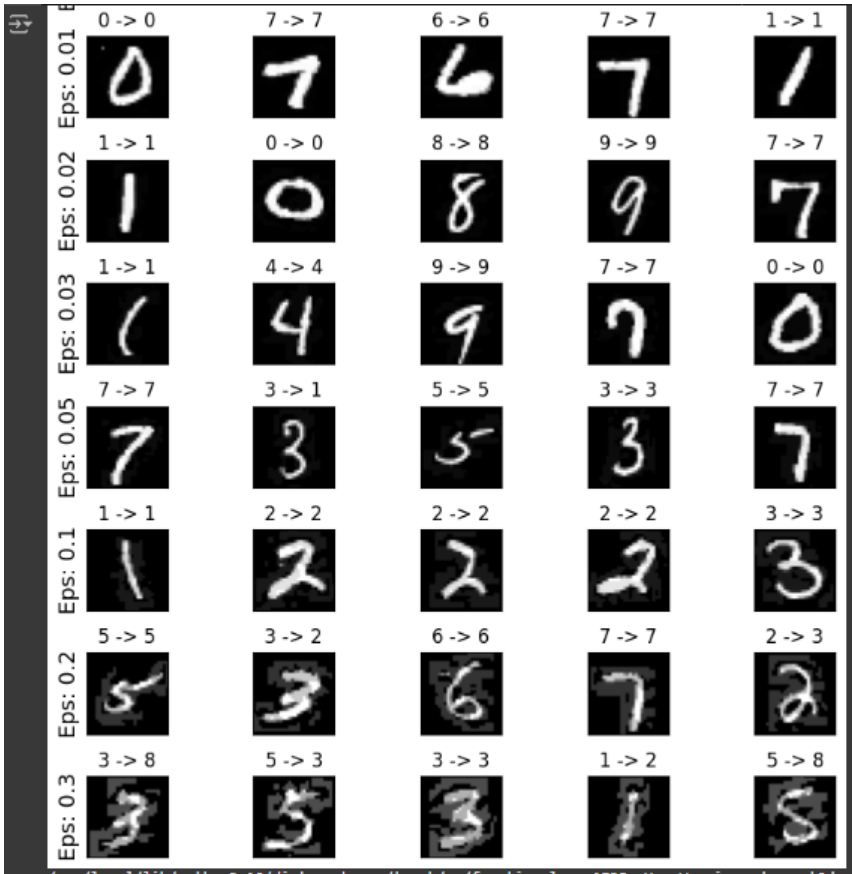



```
Обучение модели...
Эпоха: 1 Потери: 0.6789468050052314 Потери (валидация): 7.511129835620523e-07
Эпоха: 2 Потери: 0.45613521310542715 Потери (валидация): 6.377130303008016e-08
Эпоха: 3 Потери: 0.3989499004363599 Потери (валидация): 0.00016478687524795532
Эпоха: 4 Потери: 0.348171630085013 Потери (валидация): 4.806970879435539e-05
Эпоха: 5 Потери: 0.3117061338958206 Потери (валидация): 1.1421280691865831e-05
Эпоха: 6 Потери: 0.27639175839523317 Потери (валидация): 1.7933096177875996e-05
Эпоха: 7 Потери: 0.25217127034360426 Потери (валидация): 4.1739178821444516e-06
Эпоха: 8 Потери: 0.24713489085345372 Потери (валидация): 1.0677867382514705e-05
Эпоха: 9 Потери: 0.24319680180554998 Потери (валидация): 7.563361425582116e-08
Эпоха: 10 Потери: 0.2402709042465604 Потери (валидация): 6.777403599699028e-07
```

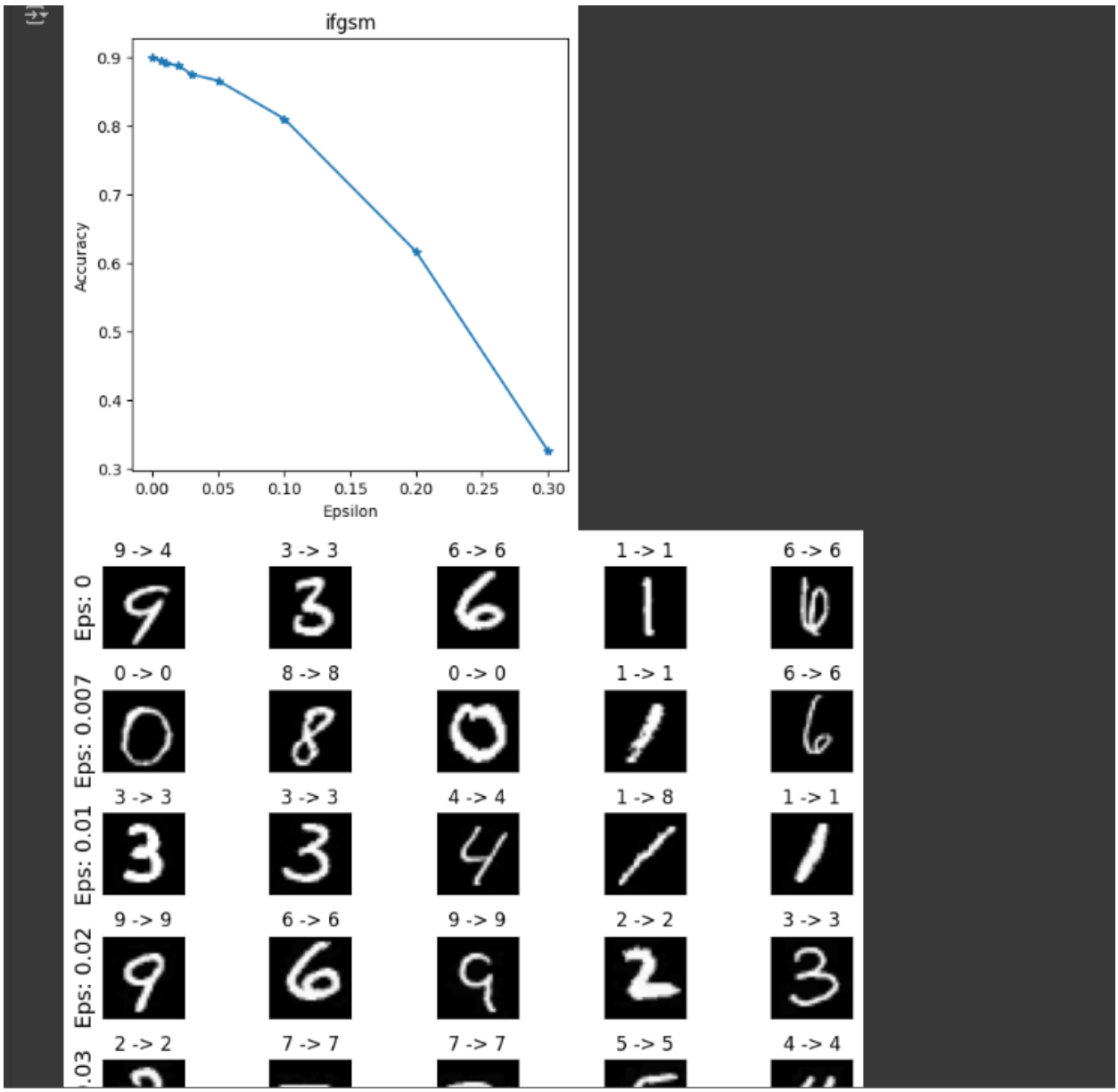



```
Эпсилон: 0      Точность (тест) = 8961 / 10000 = 0.8961
Эпсилон: 0.007  Точность (тест) = 8900 / 10000 = 0.89
Эпсилон: 0.01   Точность (тест) = 8867 / 10000 = 0.8867
Эпсилон: 0.02   Точность (тест) = 8828 / 10000 = 0.8828
Эпсилон: 0.03   Точность (тест) = 8802 / 10000 = 0.8802
Эпсилон: 0.05   Точность (тест) = 8595 / 10000 = 0.8595
Эпсилон: 0.1    Точность (тест) = 7962 / 10000 = 0.7962
Эпсилон: 0.2    Точность (тест) = 5370 / 10000 = 0.537
Эпсилон: 0.3    Точность (тест) = 2466 / 10000 = 0.2466
```

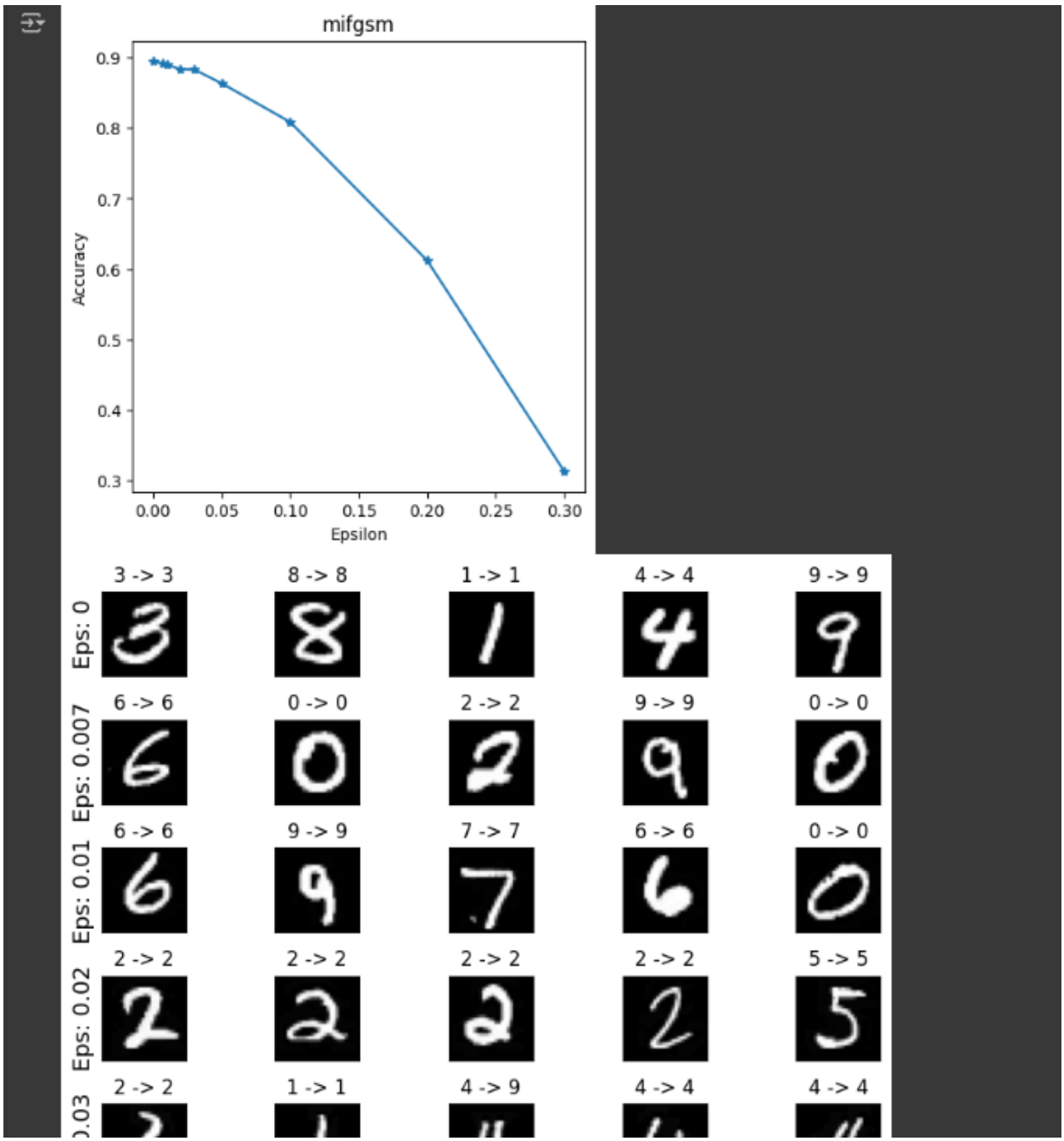

fgsm
Accuracy
Epsilon
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Eps: 0
6 -> 6
0 -> 0
4 -> 4
8 -> 8
8 -> 8
Eps: 0.007
2 -> 2
8 -> 8
7 -> 7
0 -> 0
8 -> 8
Eps: 0.01
0 -> 0
7 -> 7
6 -> 6
7 -> 7
1 -> 1
Eps: 0.02
1 -> 1
0 -> 0
8 -> 8
9 -> 9
7 -> 7
.03
1 -> 1
4 -> 4
9 -> 9
7 -> 7
0 -> 0

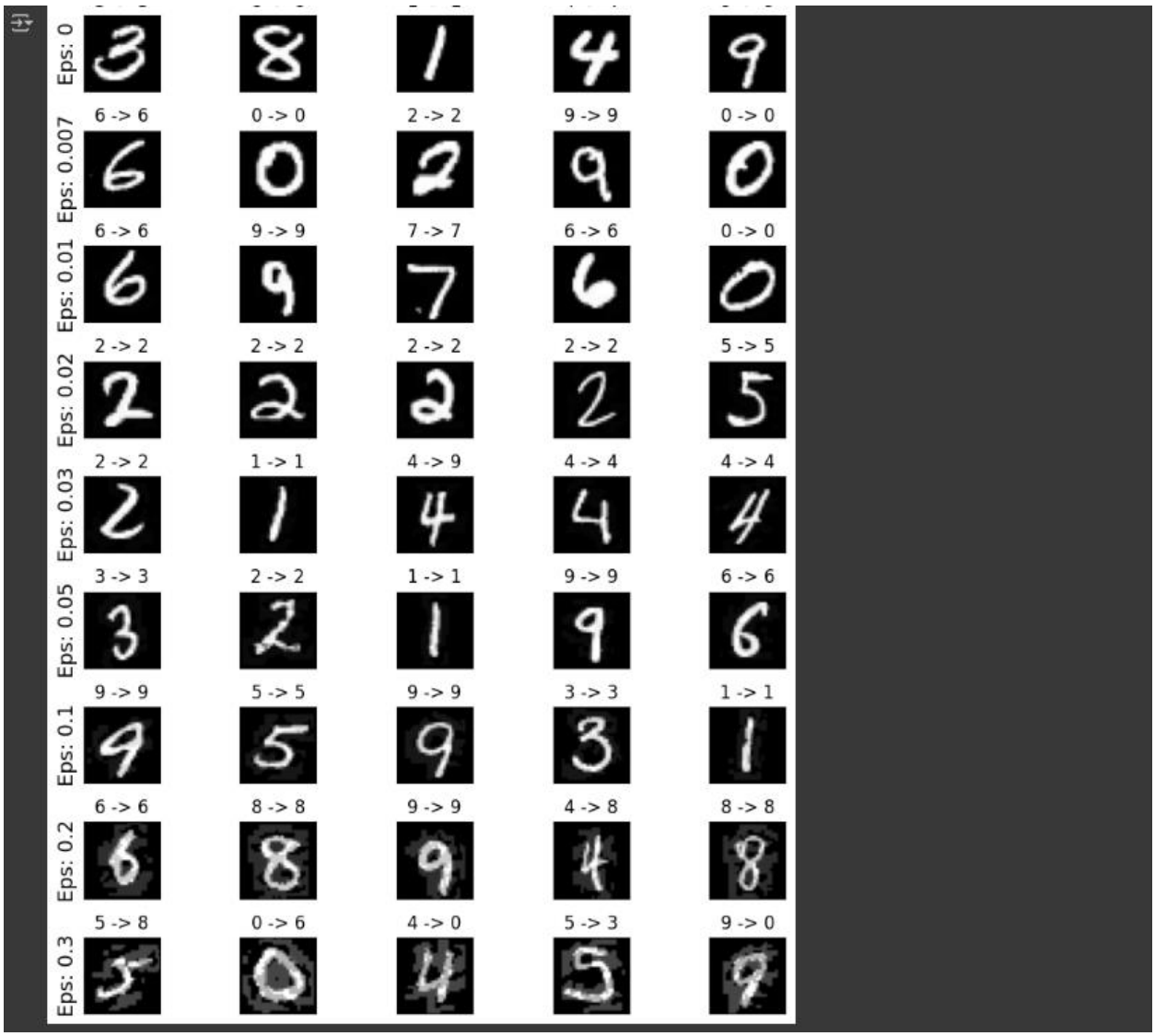



```
/usr/local/lib/python3.10/dist-packages/torch/nn/functional.py:1538: UserWarning: dropout2d: Received a 2-D input to dropout2d, which is de
  warnings.warn(warn_msg)
Эпсилон: 0      Точность (тест) = 8991 / 10000 = 0.8991
Эпсилон: 0.007  Точность (тест) = 8951 / 10000 = 0.8951
Эпсилон: 0.01   Точность (тест) = 8906 / 10000 = 0.8906
Эпсилон: 0.02   Точность (тест) = 8876 / 10000 = 0.8876
Эпсилон: 0.03   Точность (тест) = 8745 / 10000 = 0.8745
Эпсилон: 0.05   Точность (тест) = 8657 / 10000 = 0.8657
Эпсилон: 0.1    Точность (тест) = 8102 / 10000 = 0.8102
Эпсилон: 0.2    Точность (тест) = 6166 / 10000 = 0.6166
Эпсилон: 0.3    Точность (тест) = 3264 / 10000 = 0.3264
```


ifgsm
Accuracy
Epsilon
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Eps: 0
Eps: 0.007
Eps: 0.01
Eps: 0.02
9 -> 4
3 -> 3
6 -> 6
1 -> 1
6 -> 6
0 -> 0
8 -> 8
0 -> 0
1 -> 1
6 -> 6
3 -> 3
3 -> 3
4 -> 4
1 -> 8
1 -> 1
9 -> 9
6 -> 6
9 -> 9
2 -> 2
3 -> 3
2 -> 2
7 -> 7
7 -> 7
5 -> 5
4 -> 4


```
/usr/local/lib/python3.10/dist-packages/torch/nn/functional.py:1538: UserWarning: dropout2d: Received a 2-D input to dropout2d, which is de
  warnings.warn(warn_msg)
Эпсилон: 0      Точность (тест) = 8946 / 10000 = 0.8946
Эпсилон: 0.007  Точность (тест) = 8914 / 10000 = 0.8914
Эпсилон: 0.01   Точность (тест) = 8895 / 10000 = 0.8895
Эпсилон: 0.02   Точность (тест) = 8833 / 10000 = 0.8833
Эпсилон: 0.03   Точность (тест) = 8827 / 10000 = 0.8827
Эпсилон: 0.05   Точность (тест) = 8633 / 10000 = 0.8633
Эпсилон: 0.1    Точность (тест) = 8084 / 10000 = 0.8084
Эпсилон: 0.2    Точность (тест) = 6122 / 10000 = 0.6122
Эпсилон: 0.3    Точность (тест) = 3135 / 10000 = 0.3135
```


mifgsm
Accuracy
Epsilon
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Eps: 0
Eps: 0.007
Eps: 0.01
Eps: 0.02
3 -> 3
8 -> 8
1 -> 1
4 -> 4
9 -> 9
6 -> 6
0 -> 0
2 -> 2
9 -> 9
0 -> 0
6 -> 6
9 -> 9
7 -> 7
6 -> 6
0 -> 0
2 -> 2
2 -> 2
2 -> 2
2 -> 2
5 -> 5
2 -> 2
1 -> 1
4 -> 9
4 -> 4
4 -> 4


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

Заключение

Атаки FGSM, I-FGSM и MI-FGSM демонстрируют высокую эффективность при увеличении степени возмущения (є).

Точность моделей снижается с ростом є от почти 97% (без возмущения) до менее 20 - 25% при є = 0.3, что свидетельствует о значительной уязвимости моделей без защиты.

Метод защитной дистилляции продемонстрировал повышение стойкости моделей к атакам:

Точность моделей при высоких значениях є осталась выше, чем у моделей без защиты.

Защитная дистилляция уменьшила градиентные изменения, которые используются в атаках для создания возмущений.

MI-FGSM и I-FGSM показали немного большую стойкость моделей на уровне є = 0.3 по сравнению с FGSM, но в целом различия минимальны.

Точность при низких значениях є (0,007 - 0,03) осталась на приемлемом уровне для всех типов атак, но тенденция к снижению наблюдалась во всех случаях.

Метод защитной дистилляции эффективен для снижения успешности атак FGSM, I-FGSM и MI-FGSM, что подтверждается результатами тестов.

Простота интеграции метода делает его пригодным для использования в широком спектре задач.

При высоких значениях є (>0,2) точность модели всё ещё существенно снижается, что говорит о необходимости дополнительных методов защиты для таких случаев.

Метод может быть менее эффективным против адаптивных атак, когда злоумышленники подстраиваются под защитные механизмы.

Защитная дистилляция доказала свою эффективность в качестве базового метода повышения стойкости моделей НС к атакам.

Для максимальной защиты рекомендуется сочетать данный подход с другими методами (например, шумоподавлением, регуляризацией или

аугментацией данных) и тестировать модели против более сложных сценариев атак.