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 защищенности систем искусственного интеллекта»

Группа:
ББМО-01-22
Выполнила:
Глазунова П.М.

Проверил:
Спирин А.А.

Москва 2023

1. Выполнить 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
```

2. Задать нормализующие преобразования? загрузить набор данных (MNIST), разбить 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))
```

```
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, 105207821.42it/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, 24275690.15it/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, 26633591.34it/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, 18993548.12it/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw

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

3. Настроить использование графического ускорител (если возможно)

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

4. Создать 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.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
```

5. Проверить работоспособность созданного класса НС

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

6. Создать 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. Определить 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
```

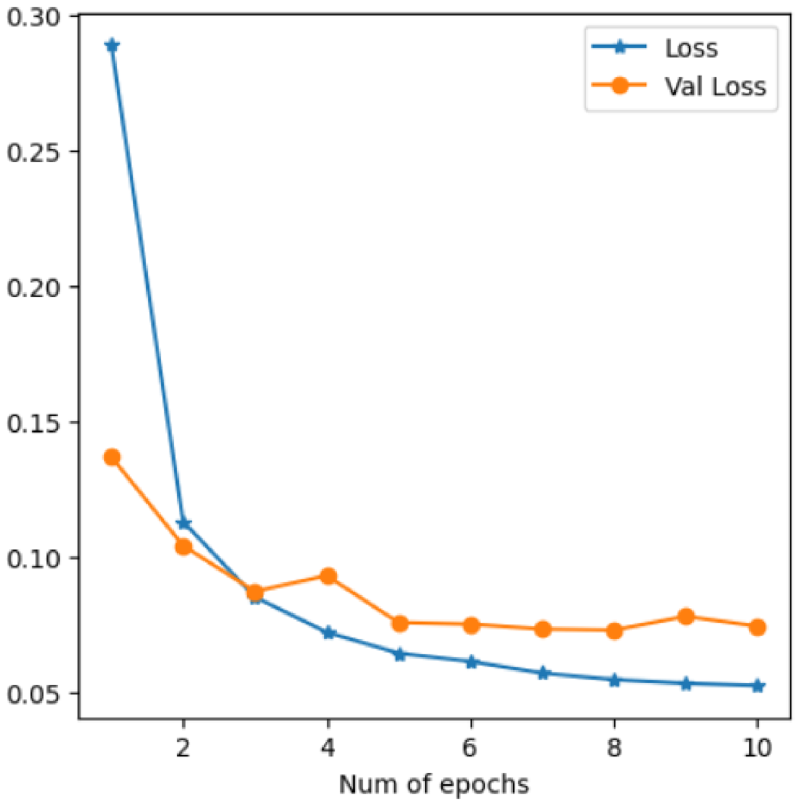
8. Обучить модель

```
[ ] 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 retair
  warnings.warn(warn_msg)
Epoch: 1 Loss: 0.2888022748806934 Val_Loss: 0.13686765499146245
Epoch: 2 Loss: 0.11297931097731007 Val_Loss: 0.10411850843998495
Epoch: 3 Loss: 0.0851646830555988 Val_Loss: 0.08724210682397247
Epoch: 4 Loss: 0.07216926405874084 Val_Loss: 0.09303551193967913
Epoch: 5 Loss: 0.06446179831371818 Val_Loss: 0.07576892791402166
Epoch: 6 Loss: 0.061423064131789176 Val_Loss: 0.07517019388235097
Epoch: 7 Loss: 0.05710774980605556 Val_Loss: 0.07333107345493659
Epoch: 8 Loss: 0.0546891696877841 Val_Loss: 0.07294204704767313
Epoch: 9 Loss: 0.0533269792951741 Val_Loss: 0.0780563005981545
Epoch: 10 Loss: 0.052500439544574236 Val_Loss: 0.07439602900140584
```

9. Построить 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()
```



10. Создать 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):
      pert_out = input + epsilon*data_grad.sign()
      pert_out = torch.clamp(pert_out, 0, 1)
      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
```

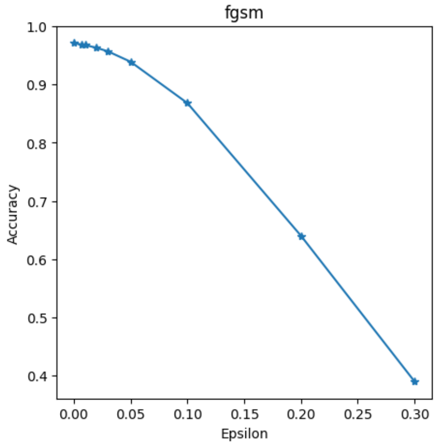
11. Создать функцию проверки

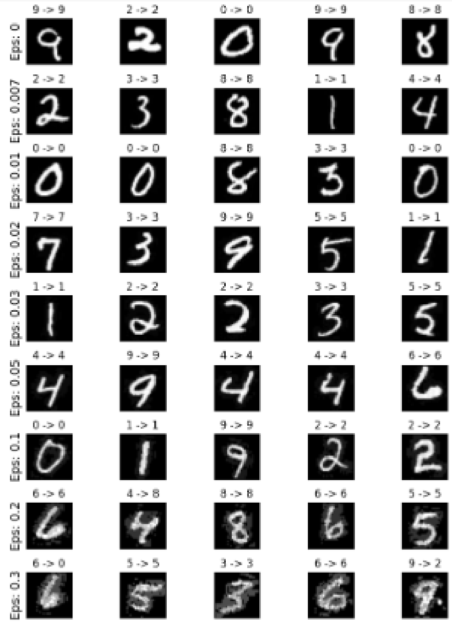
```
[ ] def test(model,device,test_loader,epsilon,attack):
      correct = 0
      adv_examples = []
      for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        data.requires_grad = True
        output = model(data)
        init_pred = output.max(1, keepdim=True)[1]
        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. Построить графики успешности атак(Accuracy/эпсилон) и примеры выполненных атак в зависимости от мтепени возмущения epsilon

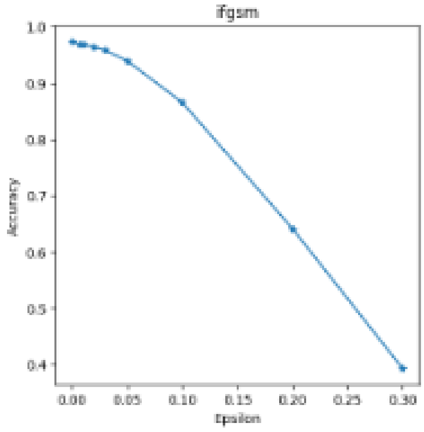
```
[ ] epsilons = [0,0.007,0.01,0.02,0.03,0.05,0.1,0.2,0.3]
    for attack in ("fgsm","ifgsm","mifgsm"):
      accuracies = []
      examples = []
      for eps in epsilons:
        acc, ex = test(model, device,test_loader,eps,attack)
        accuracies.append(acc)
        examples.append(ex)
     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

```
Epsilon: 0      Test Accuracy = 9715 / 10000 = 0.9715
Epsilon: 0.007  Test Accuracy = 9684 / 10000 = 0.9684
Epsilon: 0.01   Test Accuracy = 9678 / 10000 = 0.9678
Epsilon: 0.02   Test Accuracy = 9630 / 10000 = 0.963
Epsilon: 0.03   Test Accuracy = 9570 / 10000 = 0.957
Epsilon: 0.05   Test Accuracy = 9391 / 10000 = 0.9391
Epsilon: 0.1    Test Accuracy = 8681 / 10000 = 0.8681
Epsilon: 0.2    Test Accuracy = 6403 / 10000 = 0.6403
Epsilon: 0.3    Test Accuracy = 3900 / 10000 = 0.39
```

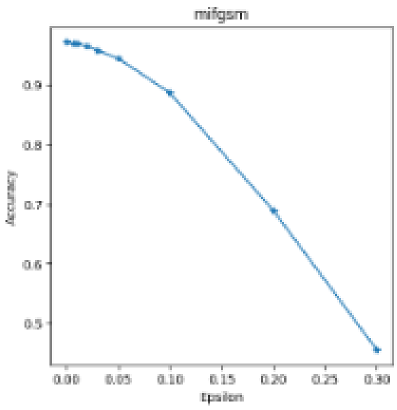




```
Epsilon: 0      Test Accuracy = 9734 / 10000 = 0.9734
Epsilon: 0.007  Test Accuracy = 9684 / 10000 = 0.9684
Epsilon: 0.01   Test Accuracy = 9683 / 10000 = 0.9683
Epsilon: 0.02   Test Accuracy = 9635 / 10000 = 0.9635
Epsilon: 0.03   Test Accuracy = 9581 / 10000 = 0.9581
Epsilon: 0.05   Test Accuracy = 9393 / 10000 = 0.9393
Epsilon: 0.1    Test Accuracy = 8657 / 10000 = 0.8657
Epsilon: 0.2    Test Accuracy = 6416 / 10000 = 0.6416
Epsilon: 0.3    Test Accuracy = 3936 / 10000 = 0.3936
```



```
Epsilon: 0      Test Accuracy = 9717 / 10000 = 0.9717
Epsilon: 0.007  Test Accuracy = 9688 / 10000 = 0.9688
Epsilon: 0.01   Test Accuracy = 9696 / 10000 = 0.9696
Epsilon: 0.02   Test Accuracy = 9643 / 10000 = 0.9643
Epsilon: 0.03   Test Accuracy = 9573 / 10000 = 0.9573
Epsilon: 0.05   Test Accuracy = 9440 / 10000 = 0.944
Epsilon: 0.1    Test Accuracy = 8860 / 10000 = 0.886
Epsilon: 0.2    Test Accuracy = 6891 / 10000 = 0.6891
Epsilon: 0.3    Test Accuracy = 4558 / 10000 = 0.4558
```



Защита от атак

13. Создать 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
```

14. Переопределить функцию обучения и тестирования

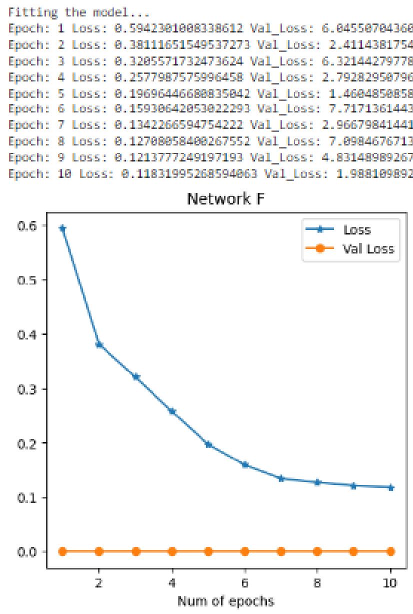
```
[ ] def fit(model,device,optimizer,scheduler,criterion,train_loader,val_loader,Temp,epochs):
      data_loader = {'train':train_loader,'val':val_loader}
      print("Fitting the model...")
      train_loss,val_loss=[],[]
      for epoch in range(epochs):
        loss_per_epoch,val_loss_per_epoch=0,0
        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
```

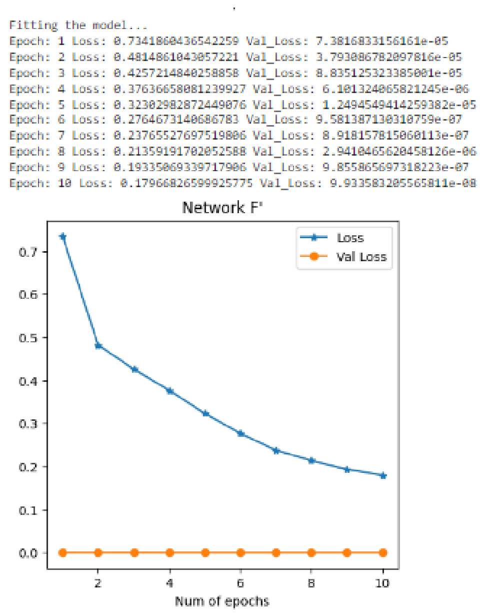
15. Создать функцию защиты методом дистилляции

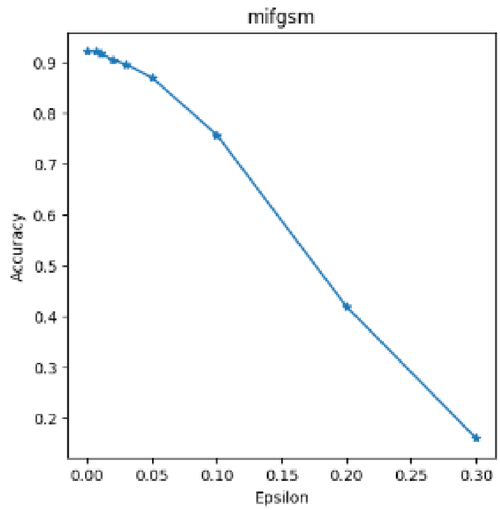
```
[ ] def defense(device,train_loader,val_loader,test_loader,epochs,Temp,epsilons):
      modelF = NetF().to(device)
      optimizerF = optim.Adam(modelF.parameters(),lr=0.0001, betas=(0.9, 0.999))
      schedulerF = optim.lr_scheduler.ReduceLROnPlateau(optimizerF, mode='min', factor=0.1, patience=3)
      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. Получить 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: применение защитной дистилляции обеспечивает безопасность и надежность нейронных сетей. Атаки на защищенные классы НС оказывают меньшее влияние в сравнении с атаками на незащищенную модель.