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 и практической работе №6

По дисциплине

«Анализ защищенности систем искусственного интеллекта»

Тема: «Изучение методов защиты от атак на модели НС. Защитная дистилляция»

Студент Кузькин Павел Александрович

Группа ББМО-01-22

Работу проверил

Спирин А.А.

Москва, 2023

1) Выполним импорт необходимых библиотек

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

2) 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))
```

```
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, 126092845.34it/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, 8124459.68it/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, 33515832.05it/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, 21333178.91it/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw

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

3) Настроим использование графического ускорителя

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

Создание атак на модель НС

4) Создадим класс НС на основе фреймворка torch

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(1, 32, 3, 1)
    self.conv2 = nn.Conv2d(32, 64, 3, 1)
    self.dropout1 = nn.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) 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

```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)
        #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
```

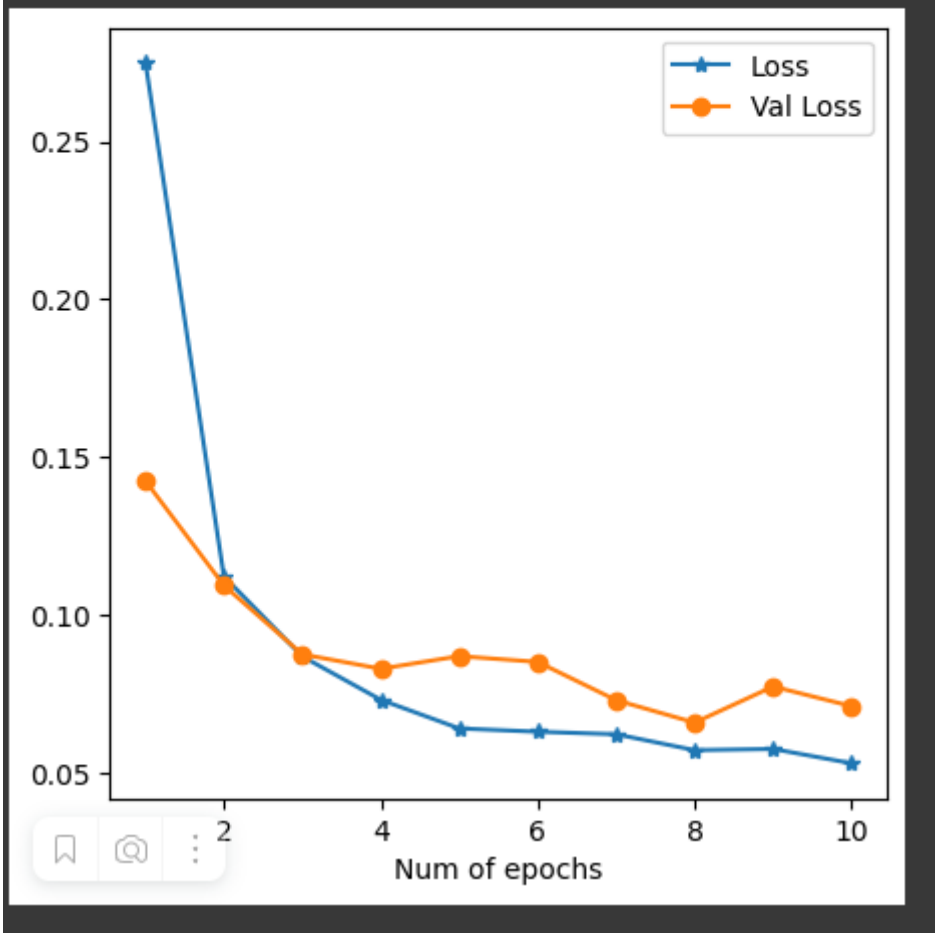
8) Обучим модель

```python
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: drop
  warnings.warn(warn_msg)
Epoch: 1 Loss: 0.27495270336064437 Val_Loss: 0.14239008297108285
Epoch: 2 Loss: 0.1120919003442392 Val_Loss: 0.10922644311979782
Epoch: 3 Loss: 0.08672445963417252 Val_Loss: 0.08739628907300931
Epoch: 4 Loss: 0.07298212078424314 Val_Loss: 0.08282595860177518
Epoch: 5 Loss: 0.06390376266366907 Val_Loss: 0.08682993991322586
Epoch: 6 Loss: 0.06287630056647002 Val_Loss: 0.08499377419658799
Epoch: 7 Loss: 0.0620630628913803 Val_Loss: 0.0728589418670597
Epoch: 8 Loss: 0.05701956471722948 Val_Loss: 0.06574958599173013
Epoch: 9 Loss: 0.05740778177132226 Val_Loss: 0.07719673543353958
Epoch: 10 Loss: 0.05286290612794359 Val_Loss: 0.07095956097811598
```

9) 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()
```



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

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
```

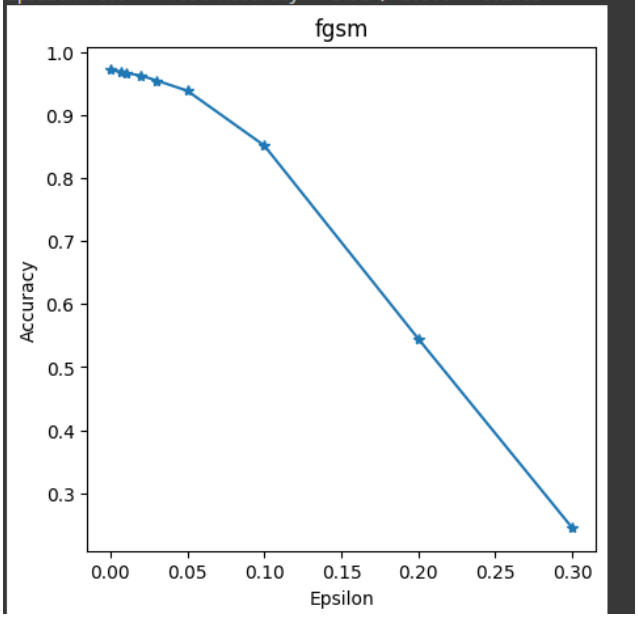
11) Создадим функцию проверки

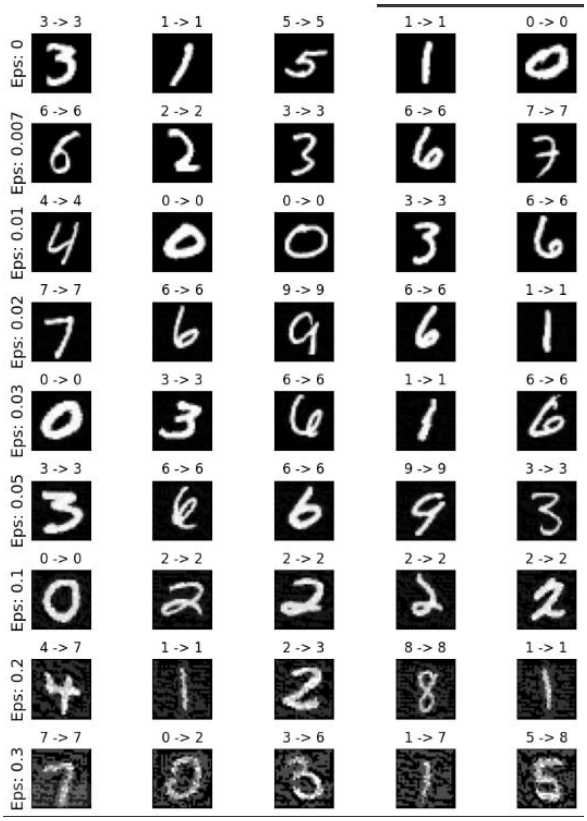
```
def test(model,device,test_loader,epsilon,attack):
  correct = 0
  adv_examples = []
  for data, target in test_loader:
    data, target = data.to(device), target.to(device)
    data.requires_grad = True
    output = model(data)
    init_pred = output.max(1, keepdim=True)[1]
    if init_pred.item()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

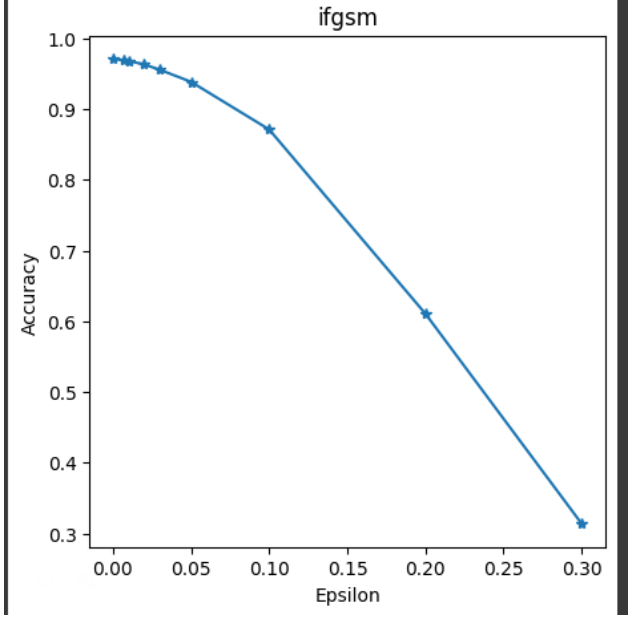
```
epsilons = [0,0.007,0.01,0.02,0.03,0.05,0.1,0.2,0.3]
for attack in ("fgsm","ifgsm","mifgsm"):
  accuracies = []
  examples = []
  for eps in epsilons:
    acc, ex = test(model, device,test_loader,eps,attack)
    accuracies.append(acc)
    examples.append(ex)
  plt.figure(figsize=(5,5))
  plt.plot(epsilons,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

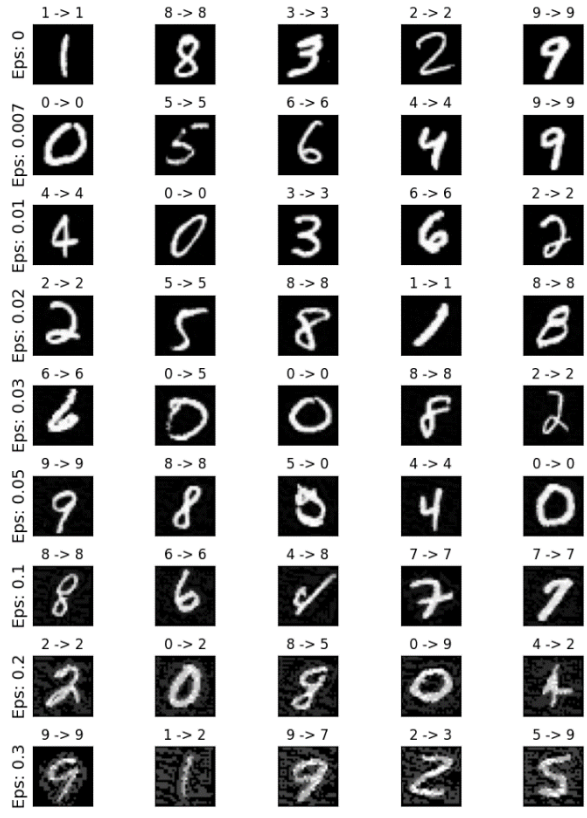
```
Epsilon: 0      Test Accuracy = 9719 / 10000 = 0.9719
Epsilon: 0.007  Test Accuracy = 9686 / 10000 = 0.9686
Epsilon: 0.01   Test Accuracy = 9666 / 10000 = 0.9666
Epsilon: 0.02   Test Accuracy = 9624 / 10000 = 0.9624
Epsilon: 0.03   Test Accuracy = 9547 / 10000 = 0.9547
Epsilon: 0.05   Test Accuracy = 9384 / 10000 = 0.9384
Epsilon: 0.1    Test Accuracy = 8514 / 10000 = 0.8514
Epsilon: 0.2    Test Accuracy = 5450 / 10000 = 0.545
Epsilon: 0.3    Test Accuracy = 2451 / 10000 = 0.2451
```



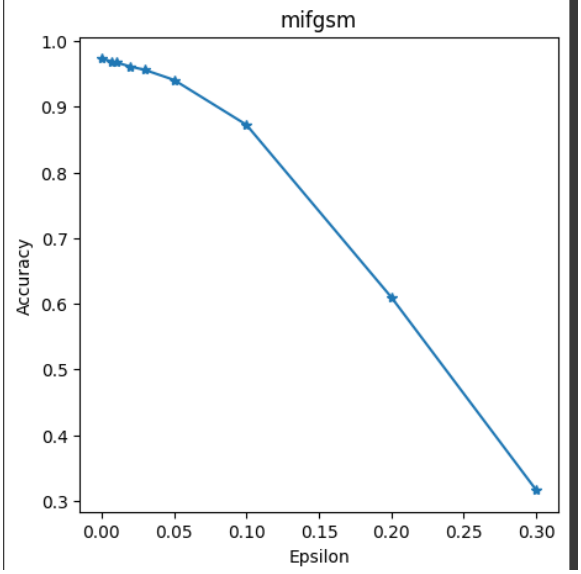


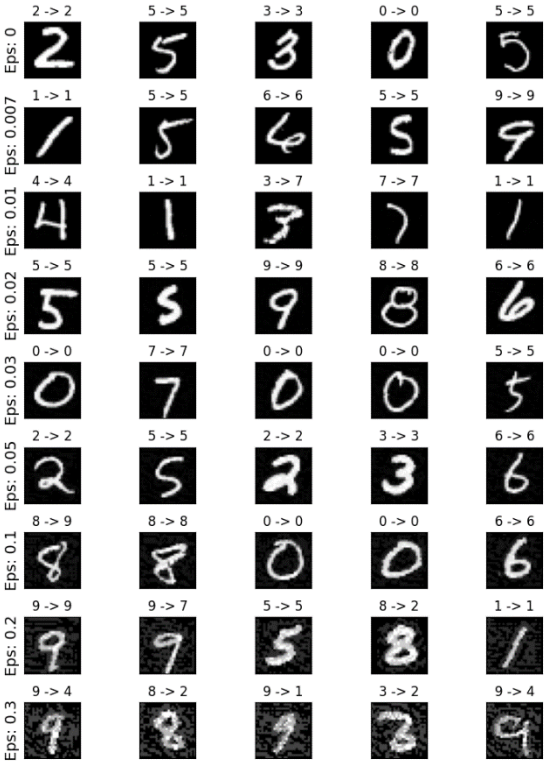

```
Epsilon: 0      Test Accuracy = 9712 / 10000 = 0.9712
Epsilon: 0.007  Test Accuracy = 9692 / 10000 = 0.9692
Epsilon: 0.01   Test Accuracy = 9683 / 10000 = 0.9683
Epsilon: 0.02   Test Accuracy = 9633 / 10000 = 0.9633
Epsilon: 0.03   Test Accuracy = 9558 / 10000 = 0.9558
Epsilon: 0.05   Test Accuracy = 9387 / 10000 = 0.9387
Epsilon: 0.1    Test Accuracy = 8713 / 10000 = 0.8713
Epsilon: 0.2    Test Accuracy = 6108 / 10000 = 0.6108
Epsilon: 0.3    Test Accuracy = 3139 / 10000 = 0.3139
```





```
Epsilon: 0      Test Accuracy = 9735 / 10000 = 0.9735
Epsilon: 0.007  Test Accuracy = 9690 / 10000 = 0.969
Epsilon: 0.01   Test Accuracy = 9676 / 10000 = 0.9676
Epsilon: 0.02   Test Accuracy = 9612 / 10000 = 0.9612
Epsilon: 0.03   Test Accuracy = 9561 / 10000 = 0.9561
Epsilon: 0.05   Test Accuracy = 9409 / 10000 = 0.9409
Epsilon: 0.1    Test Accuracy = 8725 / 10000 = 0.8725
Epsilon: 0.2    Test Accuracy = 6098 / 10000 = 0.6098
Epsilon: 0.3    Test Accuracy = 3160 / 10000 = 0.316
```





Защита от атак

13) 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
```

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
```

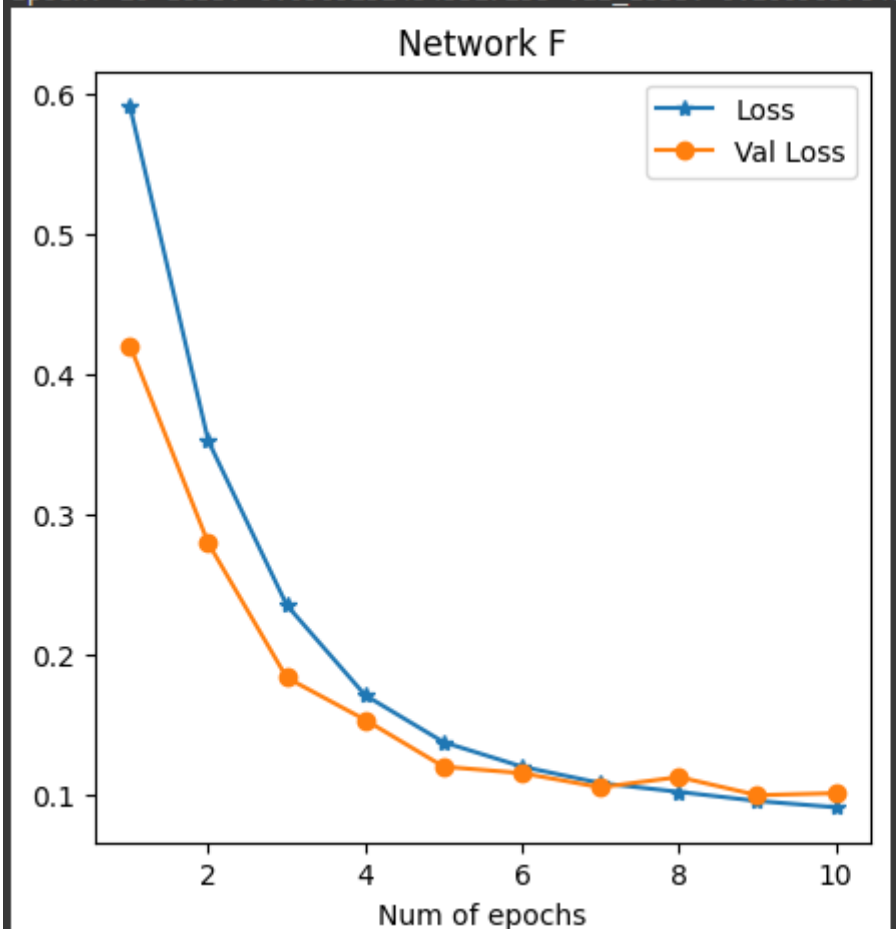
15) Создадим функцию защиты методом дистилляции

```
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 lables to sort lables
  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()
```

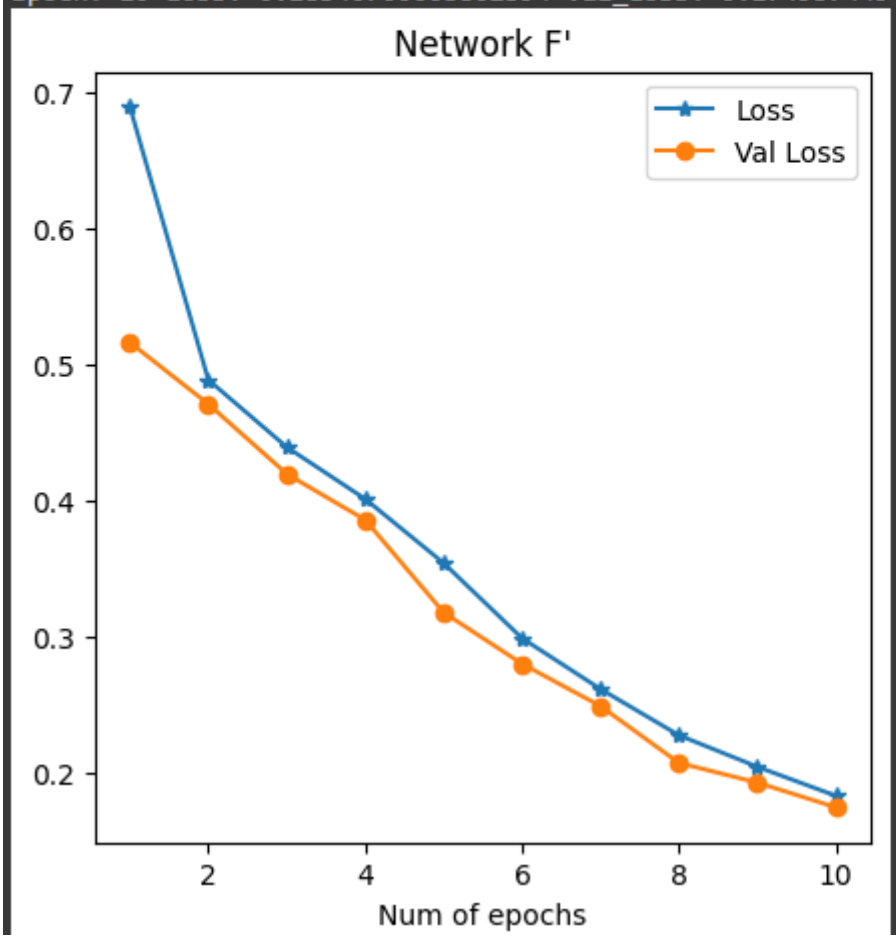
16) Получаем результаты оценки защищенных сетей

```
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)
```

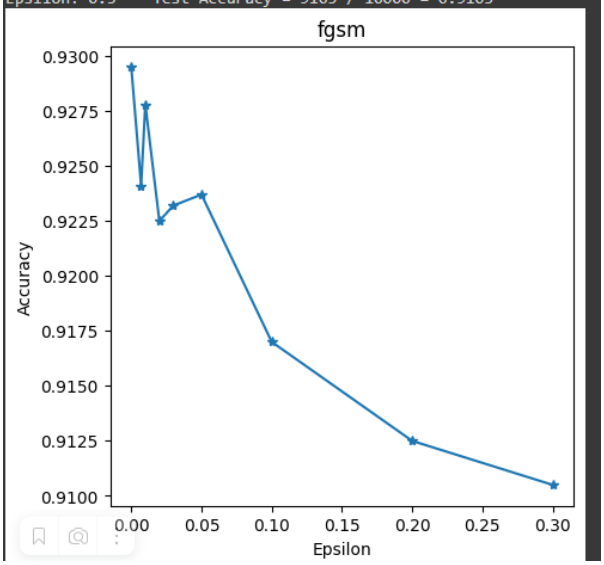
```
Fitting the model...
Epoch: 1 Loss: 0.5914319338309205 Val_Loss: 0.4208010597654996
Epoch: 2 Loss: 0.35254732449539666 Val_Loss: 0.27947702636577354
Epoch: 3 Loss: 0.2352967838233575 Val_Loss: 0.18347587407191182
Epoch: 4 Loss: 0.17095924544602928 Val_Loss: 0.1533666030183842
Epoch: 5 Loss: 0.1373992687668315 Val_Loss: 0.11991949459316954
Epoch: 6 Loss: 0.11993696157290169 Val_Loss: 0.11527672231801168
Epoch: 7 Loss: 0.1081797510758487 Val_Loss: 0.10517813642392464
Epoch: 8 Loss: 0.10190559335865862 Val_Loss: 0.11251097665910705
Epoch: 9 Loss: 0.09540538158972471 Val_Loss: 0.09955265630157147
Epoch: 10 Loss: 0.09091524543317253 Val_Loss: 0.10096578461677494
```

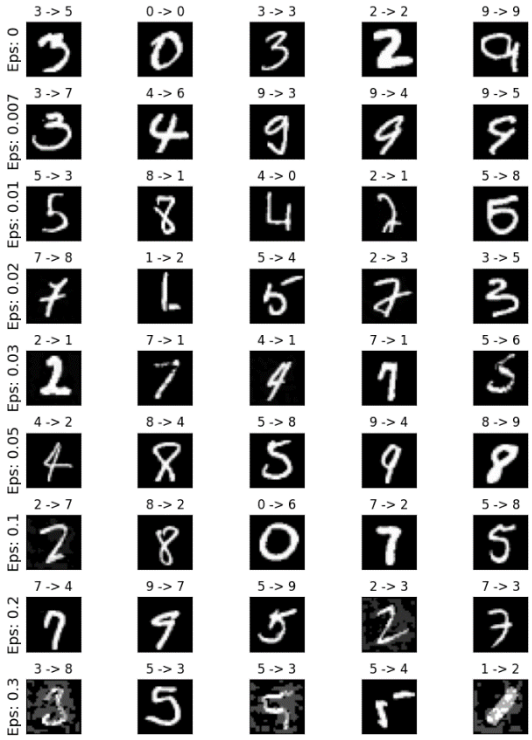


```
Fitting the model...
Epoch: 1 Loss: 0.6893553628866745 Val_Loss: 0.5162781449171815
Epoch: 2 Loss: 0.4888300740692653 Val_Loss: 0.471053630128566
Epoch: 3 Loss: 0.43946779596786606 Val_Loss: 0.4197352662907043
Epoch: 4 Loss: 0.4013721675285165 Val_Loss: 0.38602452662477754
Epoch: 5 Loss: 0.3542975119325473 Val_Loss: 0.3180178984209406
Epoch: 6 Loss: 0.29912000817698653 Val_Loss: 0.2803191360460575
Epoch: 7 Loss: 0.26179602935960977 Val_Loss: 0.24927071500932174
Epoch: 8 Loss: 0.22785844836885907 Val_Loss: 0.2075411776587448
Epoch: 9 Loss: 0.2047142563650512 Val_Loss: 0.19308143560217997
Epoch: 10 Loss: 0.18340700068801394 Val_Loss: 0.17493744941662315
```

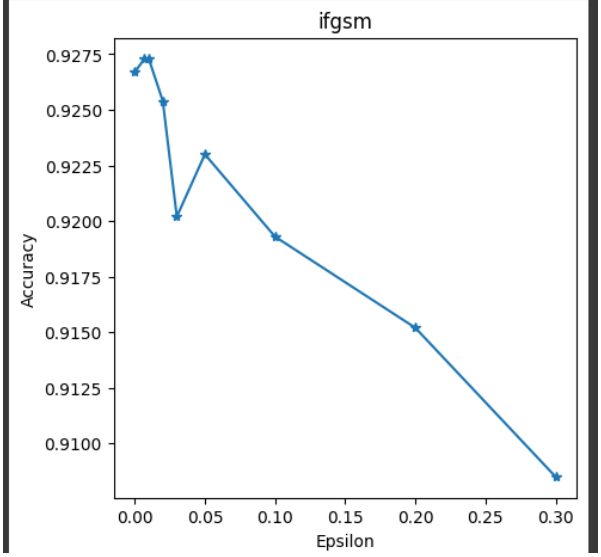


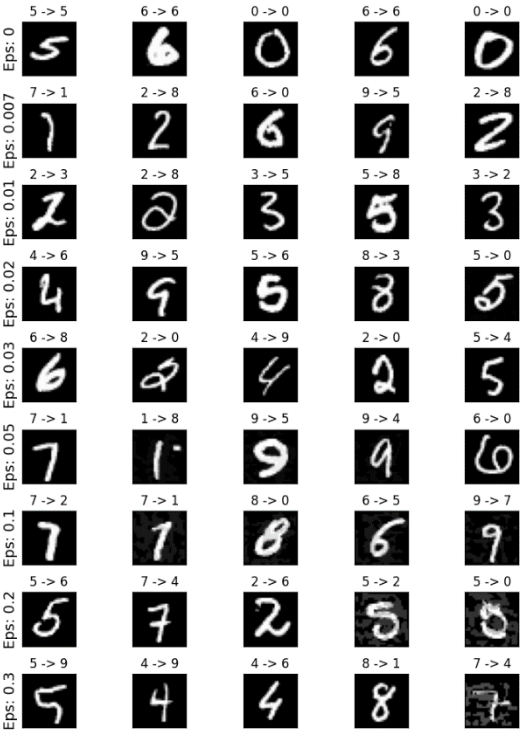
```
Epsilon: 0      Test Accuracy = 9295 / 10000 = 0.9295
Epsilon: 0.007  Test Accuracy = 9241 / 10000 = 0.9241
Epsilon: 0.01   Test Accuracy = 9278 / 10000 = 0.9278
Epsilon: 0.02   Test Accuracy = 9225 / 10000 = 0.9225
Epsilon: 0.03   Test Accuracy = 9232 / 10000 = 0.9232
Epsilon: 0.05   Test Accuracy = 9237 / 10000 = 0.9237
Epsilon: 0.1    Test Accuracy = 9170 / 10000 = 0.917
Epsilon: 0.2    Test Accuracy = 9125 / 10000 = 0.9125
Epsilon: 0.3    Test Accuracy = 9105 / 10000 = 0.9105
```

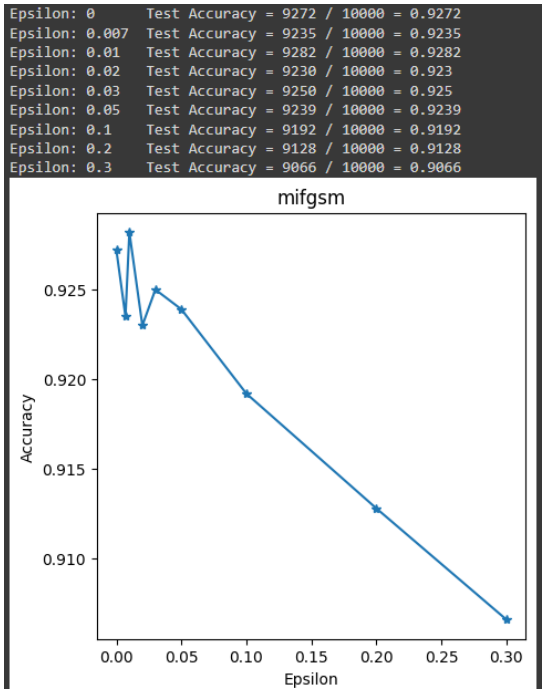


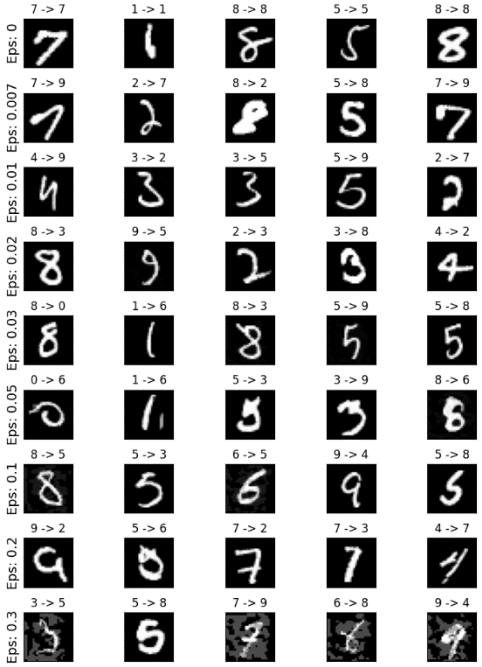



```
Epsilon: 0      Test Accuracy = 9267 / 10000 = 0.9267
Epsilon: 0.007  Test Accuracy = 9273 / 10000 = 0.9273
Epsilon: 0.01   Test Accuracy = 9273 / 10000 = 0.9273
Epsilon: 0.02   Test Accuracy = 9254 / 10000 = 0.9254
Epsilon: 0.03   Test Accuracy = 9202 / 10000 = 0.9202
Epsilon: 0.05   Test Accuracy = 9230 / 10000 = 0.923
Epsilon: 0.1    Test Accuracy = 9193 / 10000 = 0.9193
Epsilon: 0.2    Test Accuracy = 9152 / 10000 = 0.9152
Epsilon: 0.3    Test Accuracy = 9085 / 10000 = 0.9085
```









Заключение

Таким образом, в результате выполнения данной работы были получены навыки работы с защитной дистилляцией (defensive distillation), которая представляет собой метод защиты нейронных сетей от атак, направленных на восстановление информации из их параметров. Этот метод был предложен в работе "Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks" Г. Папернотом, Н. Крейшнером, Ф. Х. Шэолом в 2016 году.

Основная идея защитной дистилляции заключается в обучении устойчивой (теплостойкой) модели путём передачи знаний от базовой модели, подверженной атакам, к новой модели, которая спроектирована для устойчивости к различным атакам. Процесс обучения включает в себя два основных этапа:

1) **Обучение базовой модели (учительской модели).** Исходная модель обучается на тренировочных данных. Эта модель становится "учителем", и её знания будут использоваться для передачи новой модели;

2) **Обучение новой модели (студенческой модели).** Новая модель (студенческая) обучается на том же наборе данных, но в процессе обучения ей предоставляются "мягкие" метки, которые представляют собой распределение вероятностей, предсказываемое учителем. Это отличается от обычных

"жестких" меток, которые являются бинарными или категориальными значениями. Мягкие метки включают в себя более детальную информацию о распределении вероятностей.

Используя защитную дистилляцию, учительская модель обучается таким образом, чтобы её предсказания были более устойчивы к атакам, и эта стабильность передается новой модели. Поскольку новая модель обучается на основе более "мягких" меток, чем те, которые предоставляются учителю, она также становится менее уязвимой к атакам, таким как адверсариальные атаки.

Защитная дистилляция является одним из подходов к повышению устойчивости нейронных сетей к различным видам атак, и её эффективность может зависеть от конкретной атаки, нацеленной на модель.

В данной работе учителем является NetF, учеником NetF1.

Также стоит отметить, что защитная дистилляция увеличила точность:

- для атаки fgsm с 0,2451 до 0,9105;
- для атаки ifgsm с 0,3139 до 0,9085;
- для атаки mifgsm с 0,316 до 0,9066.