```
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


np.random.seed(42)
torch.manual_seed(42)
```

```
<torch._C.Generator at 0x7f48f0f615f0>
```

```
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.0
dataset = datasets.MNIST(root = './data', train=True, transform = transform, dow
train_set, val_set = torch.utils.data.random_split(dataset, [50000, 10000])
test_set = datasets.MNIST(root = './data', train=False, transform = transform, d
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)
```

```
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
100%|██████████| 9912422/9912422 [00:00<00:00, 104231386.57it/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
100%|██████████| 28881/28881 [00:00<00:00, 111542996.15it/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 .
100%|██████████| 1648877/1648877 [00:00<00:00, 34047465.34it/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 .
100%|██████████| 4542/4542 [00:00<00:00, 26060914.87it/s]Extracting ./data/
```

```
print("Training data:",len(train_loader),"Validation data:",len(val_loader),"Te
```

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

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

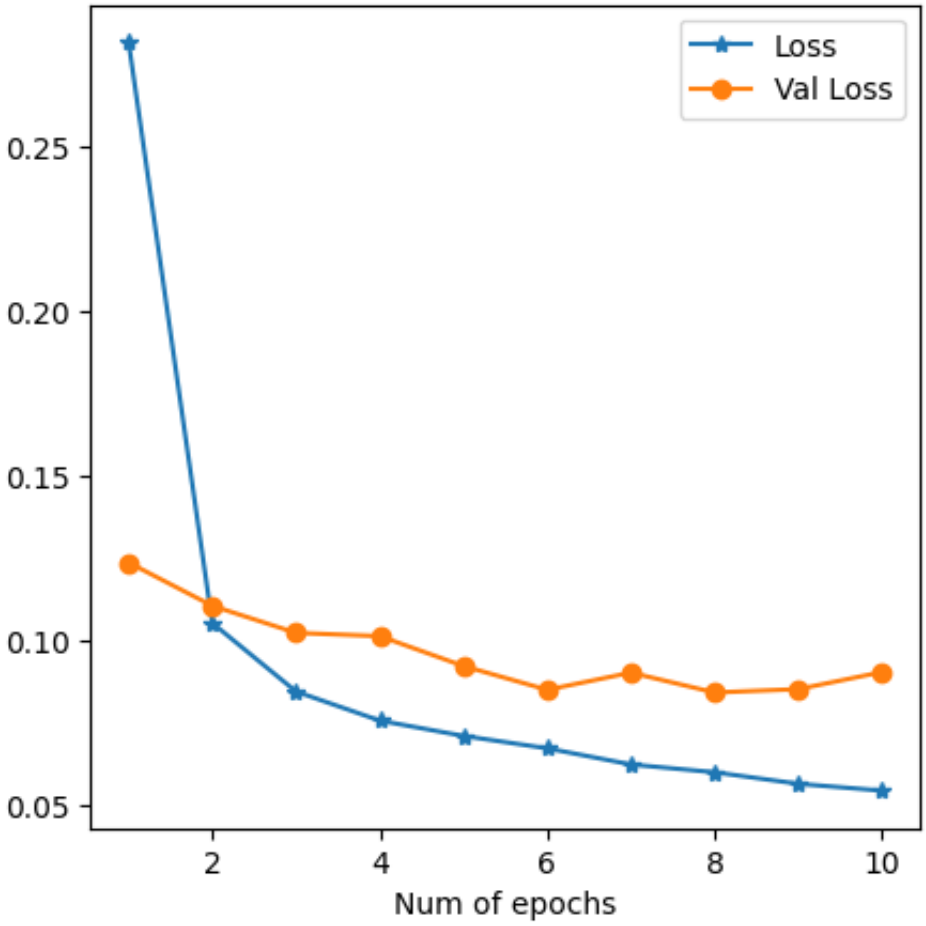
Attack

```
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)


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=
```

```
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(t
    train_loss.append(loss_per_epoch/len(train_loader))
    val_loss.append(val_loss_per_epoch/len(val_loader))
  return train_loss,val_loss


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: UserWa
  warnings.warn(warn_msg)
Epoch: 1 Loss: 0.2817260350794593 Val_Loss: 0.12361508074831536
Epoch: 2 Loss: 0.10523556910538824 Val_Loss: 0.11046755906208584
Epoch: 3 Loss: 0.08443022597397018 Val_Loss: 0.1022432165940933
Epoch: 4 Loss: 0.07558984425235236 Val_Loss: 0.10126774390126238
Epoch: 5 Loss: 0.07093882998008097 Val_Loss: 0.09216815497922358
Epoch: 6 Loss: 0.06716966546833625 Val_Loss: 0.084949647379223
Epoch: 7 Loss: 0.06226171160701656 Val_Loss: 0.09016340301871419
Epoch: 8 Loss: 0.05989020751996103 Val_Loss: 0.08412584582617386
Epoch: 9 Loss: 0.056383525802404494 Val_Loss: 0.08515309634140661
Epoch: 10 Loss: 0.05431145126483214 Val_Loss: 0.09033701285431275
```

```
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()
```

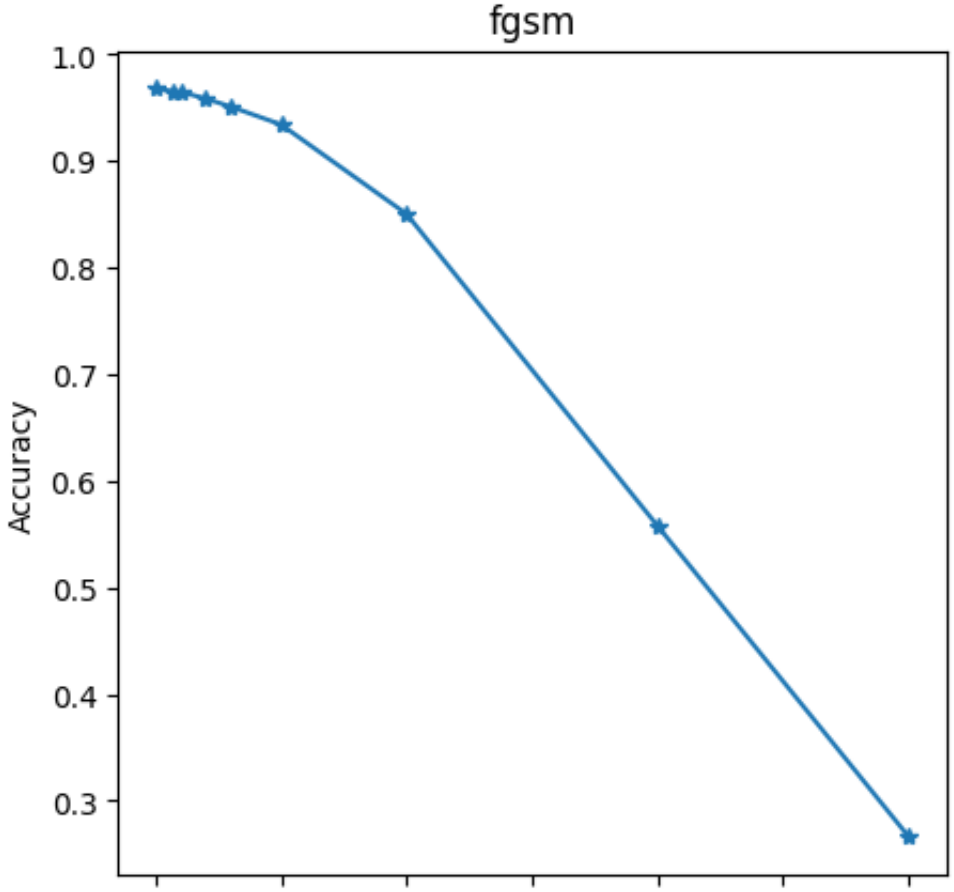


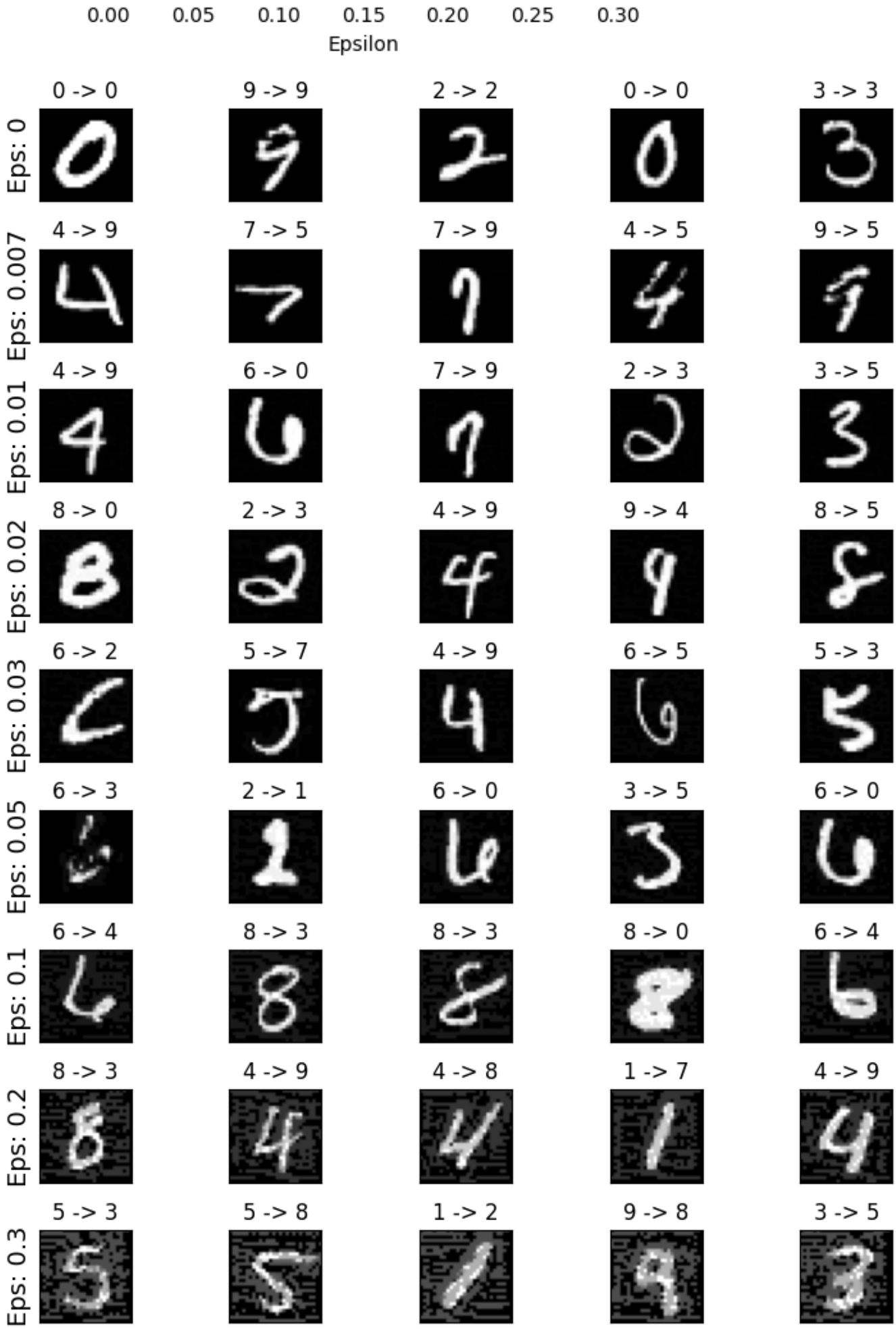
```
def fgsm_attack(input,epsilon,data_grad):
  pert_out = input + epsilon*data_grad.sign()
  pert_out = torch.clamp(pert_out, 0, 1)
  return pert_out

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
```

```
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, le

  return final_acc, adv_examples



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()
```

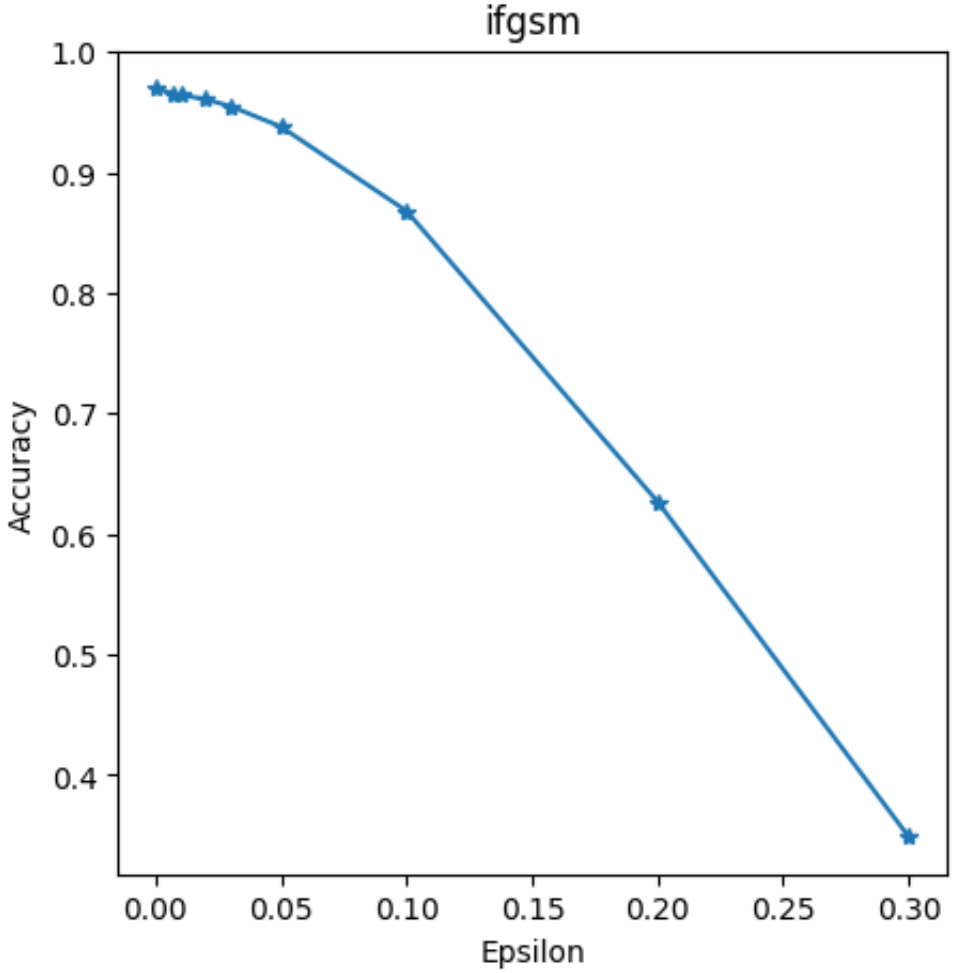
```
Epsilon: 0      Test Accuracy = 9683 / 10000 = 0.9683
Epsilon: 0.007  Test Accuracy = 9654 / 10000 = 0.9654
Epsilon: 0.01   Test Accuracy = 9650 / 10000 = 0.965
Epsilon: 0.02   Test Accuracy = 9585 / 10000 = 0.9585
Epsilon: 0.03   Test Accuracy = 9510 / 10000 = 0.951
Epsilon: 0.05   Test Accuracy = 9342 / 10000 = 0.9342
Epsilon: 0.1    Test Accuracy = 8503 / 10000 = 0.8503
Epsilon: 0.2    Test Accuracy = 5577 / 10000 = 0.5577
Epsilon: 0.3    Test Accuracy = 2663 / 10000 = 0.2663
```

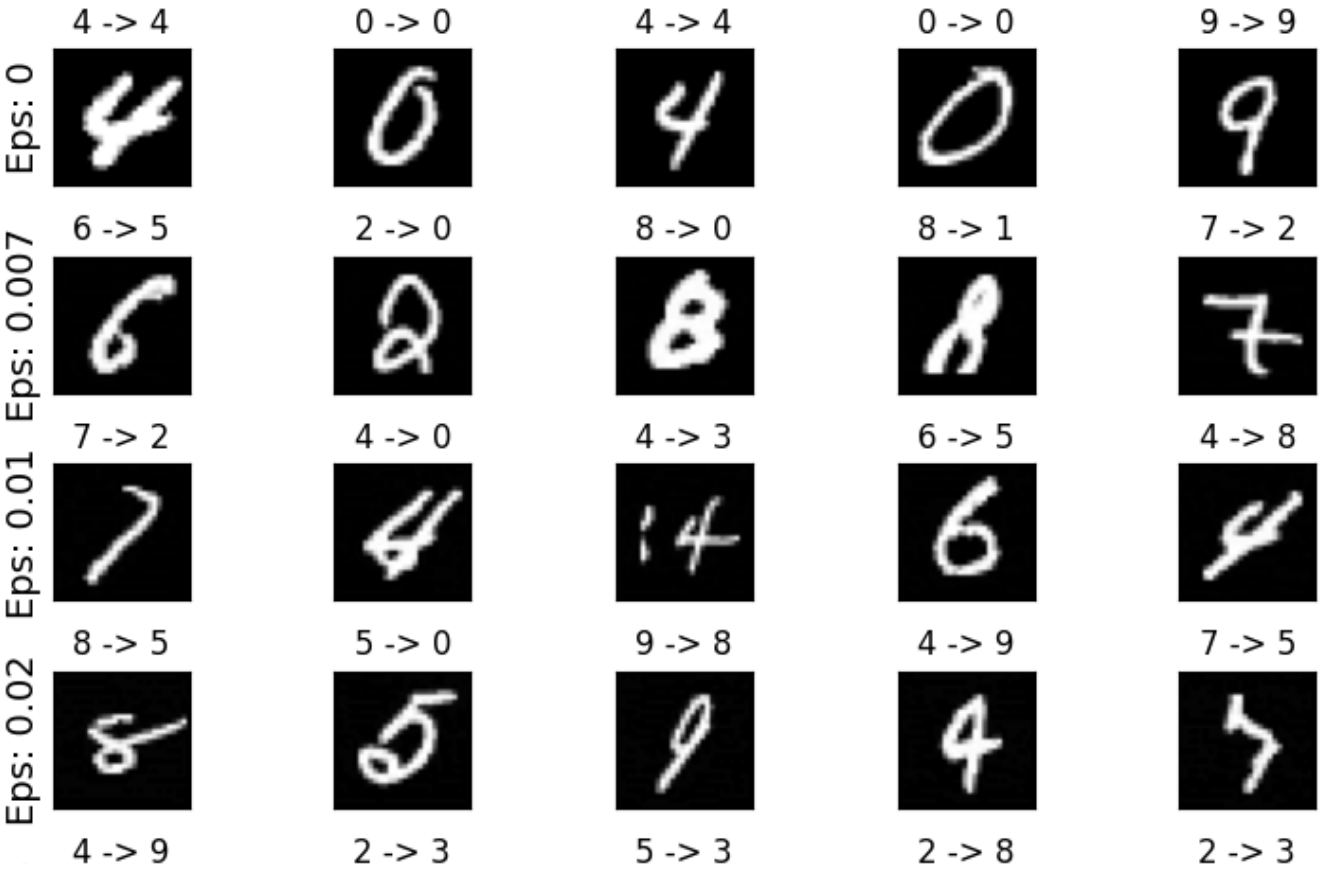


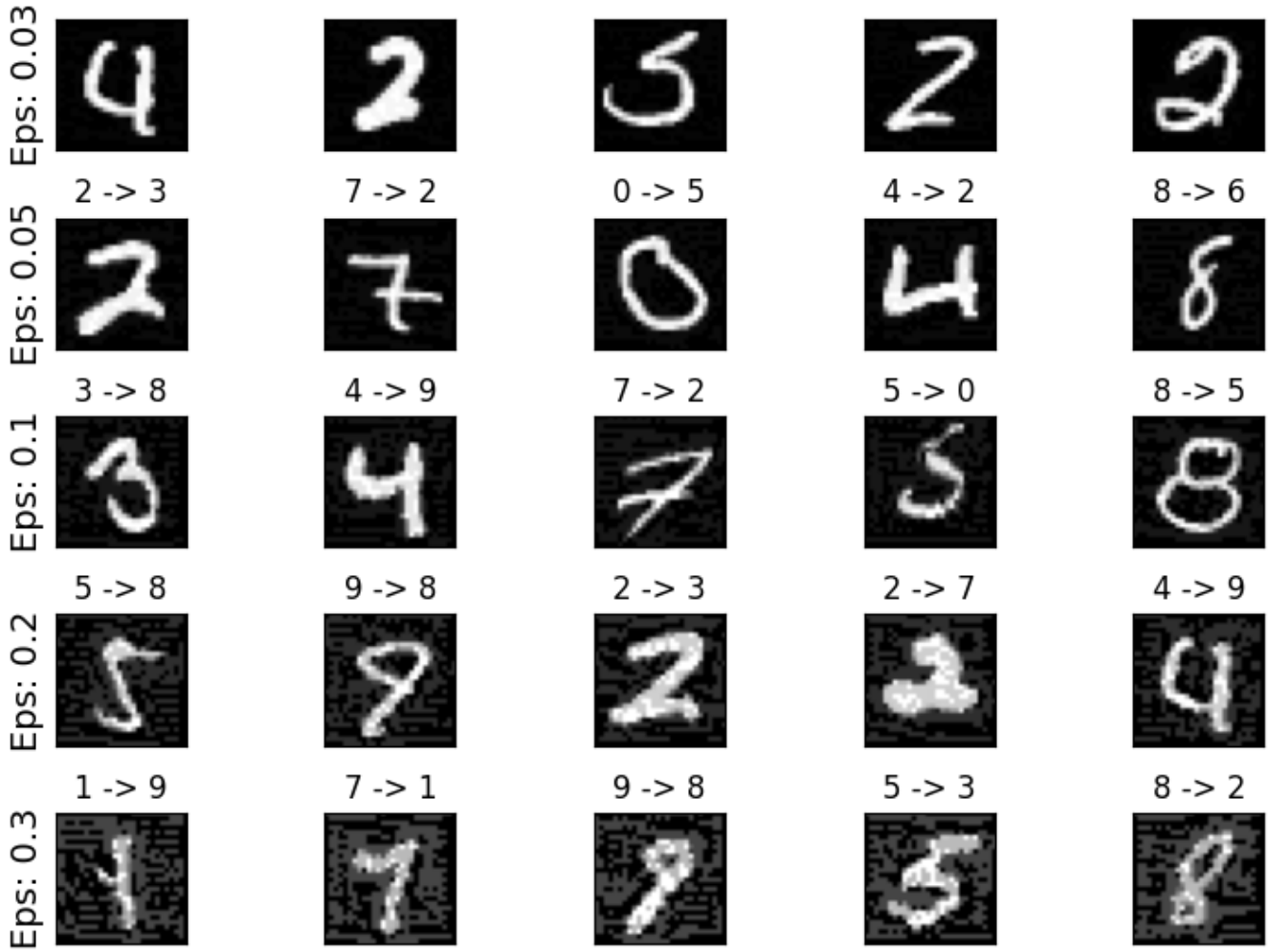



```
Epsilon: 0      Test Accuracy = 9699 / 10000 = 0.9699
Epsilon: 0.007  Test Accuracy = 9654 / 10000 = 0.9654
Epsilon: 0.01   Test Accuracy = 9644 / 10000 = 0.9644
```

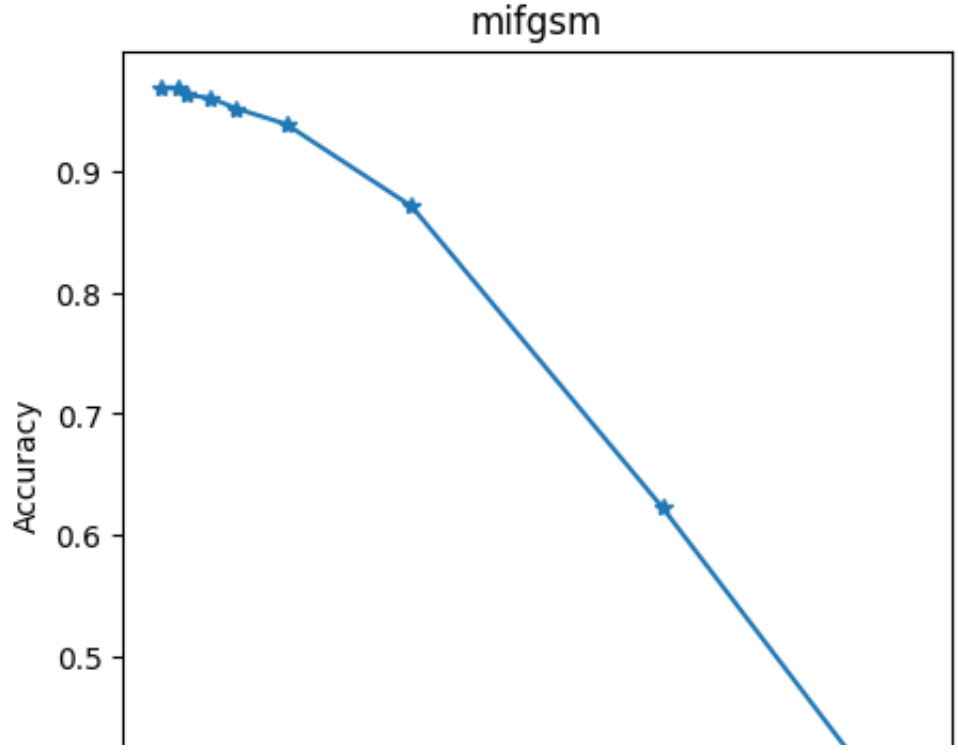
```
Epsilon: 0.02    Test Accuracy = 9607 / 10000 = 0.9607
Epsilon: 0.03    Test Accuracy = 9549 / 10000 = 0.9549
Epsilon: 0.05    Test Accuracy = 9379 / 10000 = 0.9379
Epsilon: 0.1     Test Accuracy = 8680 / 10000 = 0.868
Epsilon: 0.2     Test Accuracy = 6268 / 10000 = 0.6268
Epsilon: 0.3     Test Accuracy = 3489 / 10000 = 0.3489
```

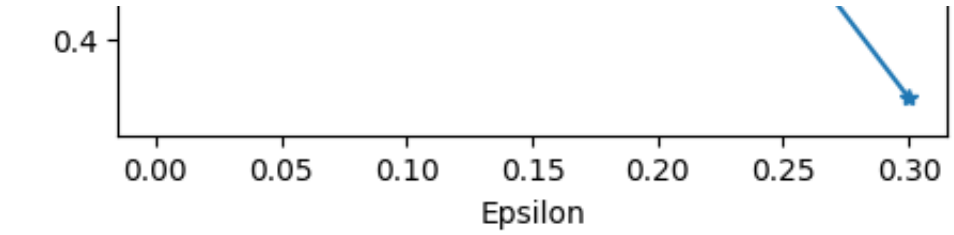


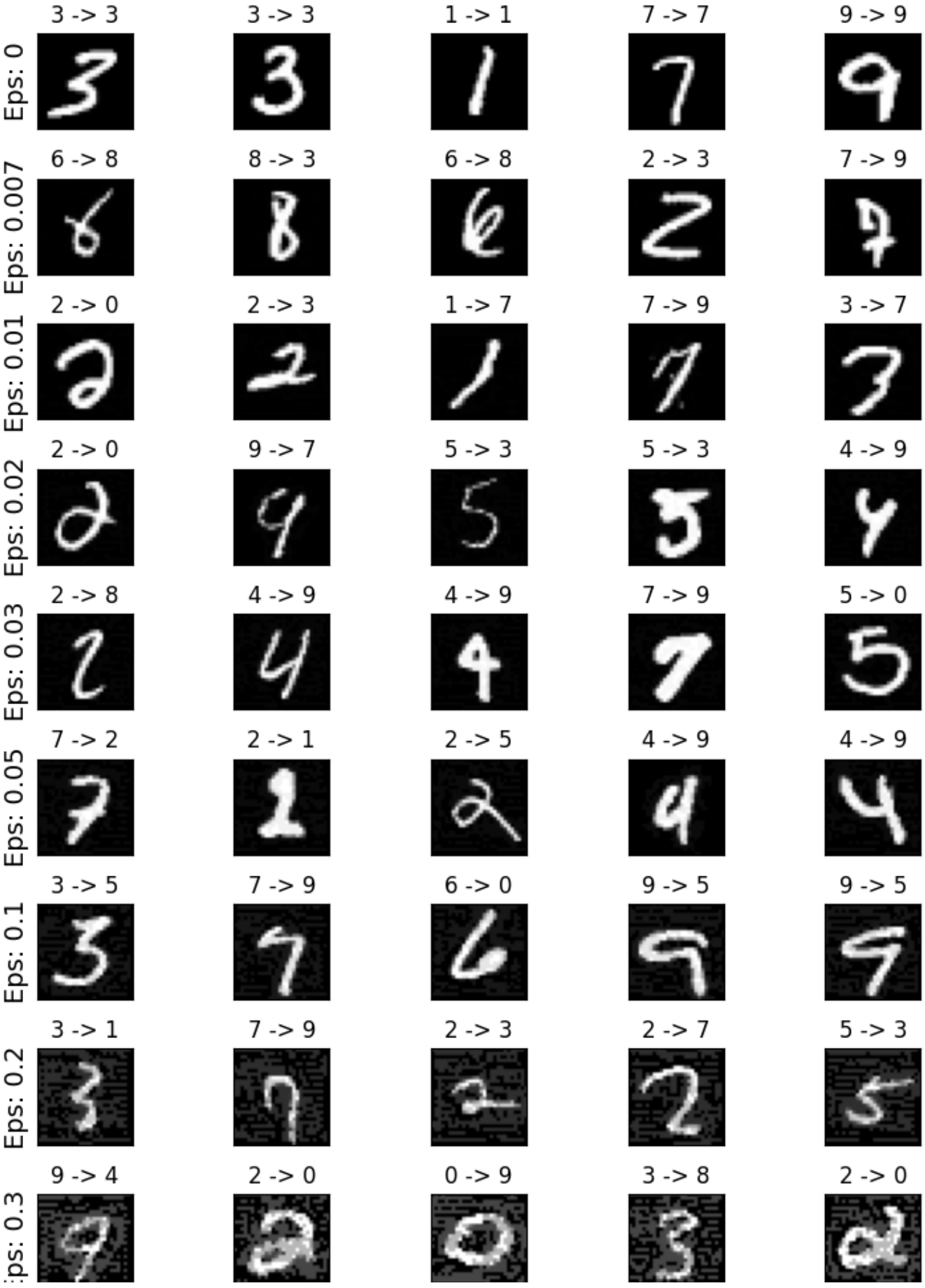





```
Epsilon: 0      Test Accuracy = 9685 / 10000 = 0.9685
Epsilon: 0.007  Test Accuracy = 9685 / 10000 = 0.9685
Epsilon: 0.01   Test Accuracy = 9631 / 10000 = 0.9631
Epsilon: 0.02   Test Accuracy = 9604 / 10000 = 0.9604
Epsilon: 0.03   Test Accuracy = 9519 / 10000 = 0.9519
Epsilon: 0.05   Test Accuracy = 9389 / 10000 = 0.9389
Epsilon: 0.1    Test Accuracy = 8711 / 10000 = 0.8711
Epsilon: 0.2    Test Accuracy = 6225 / 10000 = 0.6225
Epsilon: 0.3    Test Accuracy = 3512 / 10000 = 0.3512
```




0.4
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Epsilon


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

[redacted]

Defense

```
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
```

```
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
  alpha = epsilon/iter
  pert_out = input
  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


def fit(model,device,optimizer,scheduler,criterion,train_loader,val_loader,Temp
  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()
```

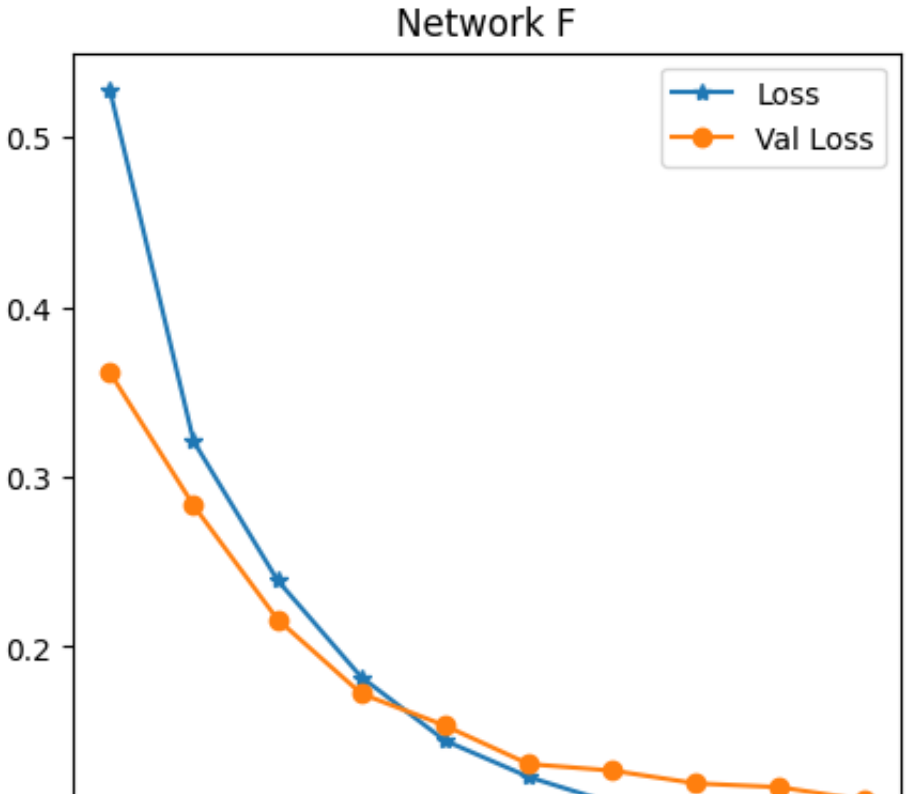
```
        else:
          val_loss_per_epoch+=loss.item()
    scheduler.step(val_loss_per_epoch/len(val_loader))
    print("Epoch: {} Loss: {} Val_Loss: {}".format(epoch+1,loss_per_epoch/len(t
    train_loss.append(loss_per_epoch/len(train_loader))
    val_loss.append(val_loss_per_epoch/len(val_loader))
  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, le

  return final_acc,adv_examples


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

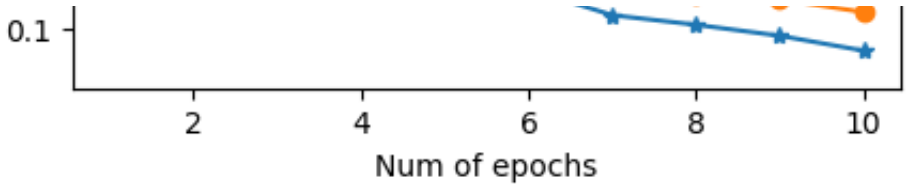
```
modelF = NetF().to(device)
optimizerF = optim.Adam(modelF.parameters(),lr=0.0001, betas=(0.9, 0.999))
schedulerF = optim.lr_scheduler.ReduceLROnPlateau(optimizerF, mode='min', fac

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', f

criterion = nn.NLLLoss()

lossF,val_lossF=fit(modelF,device,optimizerF,schedulerF,criterion,train_loade
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_
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()
```

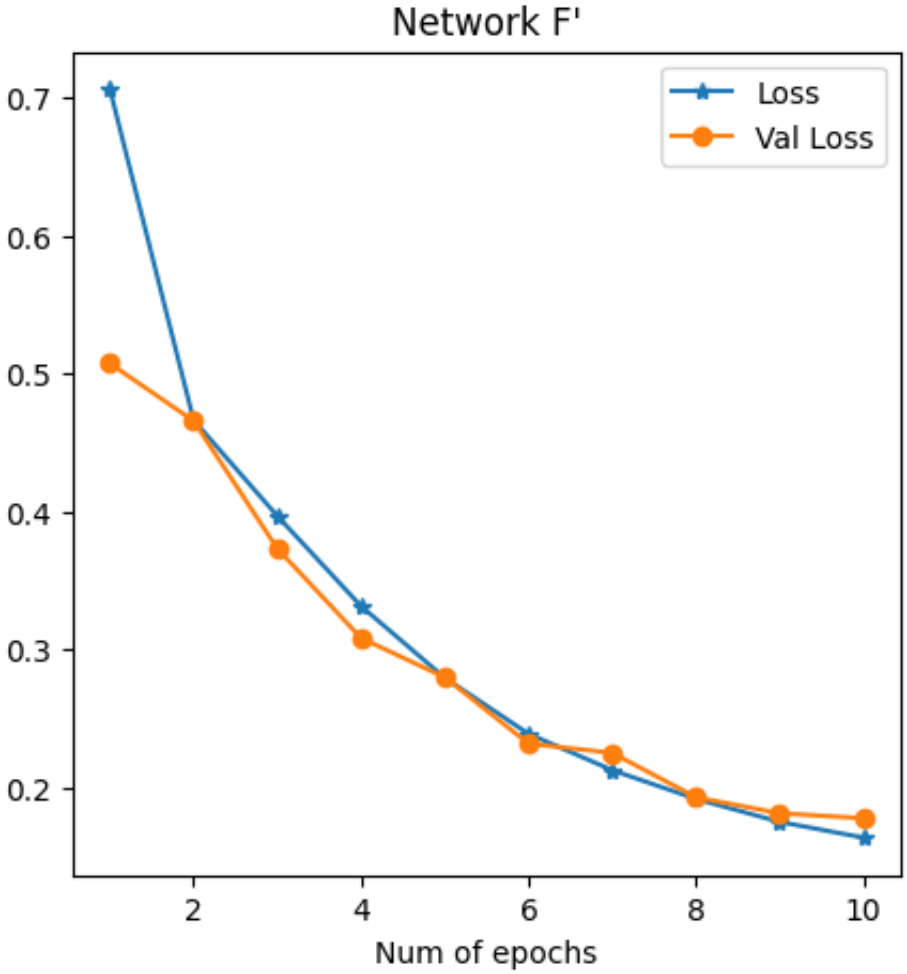
```
Temp=100
epochs=10
epsilons=[0,0.007,0.01,0.02,0.03,0.05,0.1,0.2,0.3]
defense(device,train_loader,val_loader,test_loader,epochs,Temp,epsilons)
```

```
Fitting the model...
Epoch: 1 Loss: 0.5278237399980757 Val_Loss: 0.3615423230767032
Epoch: 2 Loss: 0.32100190524823563 Val_Loss: 0.2831200823574247
Epoch: 3 Loss: 0.2392311650032153 Val_Loss: 0.2161802412526403
Epoch: 4 Loss: 0.18217379552462823 Val_Loss: 0.17237797660280577
Epoch: 5 Loss: 0.14469819753991914 Val_Loss: 0.15350133686252157
Epoch: 6 Loss: 0.12324300650043969 Val_Loss: 0.13082915828684064
Epoch: 7 Loss: 0.10777307961183408 Val_Loss: 0.12696364795953527
Epoch: 8 Loss: 0.10211605829016972 Val_Loss: 0.11941190489065118
Epoch: 9 Loss: 0.09563548854813765 Val_Loss: 0.11719594507549459
Epoch: 10 Loss: 0.08686720435729091 Val_Loss: 0.10954465521864597
```






```
Fitting the model...
Epoch: 1 Loss: 0.7061048393553518 Val_Loss: 0.5074746165118784
Epoch: 2 Loss: 0.4663726748046535 Val_Loss: 0.46580755004085095
Epoch: 3 Loss: 0.39691043616233773 Val_Loss: 0.3732880668522749
Epoch: 4 Loss: 0.33161896051084927 Val_Loss: 0.30845962283988404
Epoch: 5 Loss: 0.2796468827744753 Val_Loss: 0.28002145494234953
Epoch: 6 Loss: 0.23910592388957552 Val_Loss: 0.2319484566851277
Epoch: 7 Loss: 0.2127512915662075 Val_Loss: 0.22531034842500963
Epoch: 8 Loss: 0.19190950960266592 Val_Loss: 0.19311902156939553
Epoch: 9 Loss: 0.17527996071067983 Val_Loss: 0.1815241036411824
Epoch: 10 Loss: 0.1638730181960605 Val_Loss: 0.17793103820945025
```




```
Epsilon: 0      Test Accuracy = 9305 / 10000 = 0.9305
Epsilon: 0.007  Test Accuracy = 9301 / 10000 = 0.9301
Epsilon: 0.01   Test Accuracy = 9306 / 10000 = 0.9306
Epsilon: 0.02   Test Accuracy = 9272 / 10000 = 0.9272
Epsilon: 0.03   Test Accuracy = 9263 / 10000 = 0.9263
Epsilon: 0.05   Test Accuracy = 9277 / 10000 = 0.9277
```