哈尔滨工业大学(深圳)

《网络与系统安全》 实验报告

实验七

对抗样本攻击 实验

学 院: 计算机科学与技术

姓 名: 宗晴

学 号: 200110513

专 业: 计算机

日 期: 2023 年 4 月

一、本次实验要求

1. 完成 **4.3 FGSM 攻击函数**的代码补充，截图说明。

FGSM 攻击函数的代码如下图所示：

```
# FGSM attack code
def fgsm_attack(image, epsilon, data_grad):
    # Collect the element-wise sign of the data gradient
    sign_data_grad = data_grad.sign()
    # Create the perturbed image by adjusting each pixel of the input image
    perturbed_image = image + epsilon*sign_data_grad
    # Adding clipping to maintain [0,1] range
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    # Return the perturbed image
    return perturbed_image
```

首先收集数据梯度的元素符号，通过 sign()函数实现。然后通过调整输入图像的每个像素来创建扰动图像，具体实现是对每一个像素加上 epsilon*sign_data_grad 的扰动。然后添加剪切以维持[0,1]范围，通过 torch.clamp 函数实现。最后返回被扰动的图像。

2. 分析 **4.4 测试攻击效果函数** 的代码部分，说明每段代码的作用。

测试攻击效果函数的代码部分如下图所示。

首先，设置精度计算器：

```
def test( model, device, test_loader, epsilon ):

    # Accuracy counter
    correct = 0
    adv_examples = []
```

然后循环遍历测试集中的所有示例：

```
# Loop over all examples in test set
for data, target in test_loader:
```

把数据和标签发送到设备，设置张量的 requires_grad 属性，并通过模型前向传递数据：

```
# Send the data and label to the device
data, target = data.to(device), target.to(device)

# Set requires_grad attribute of tensor. Important for Attack
data.requires_grad = True

# Forward pass the data through the model
output = model(data)
init_pred = output.max(1, keepdim=True)[1] # get the index of the max log-probability
```

如果初始预测是错误的，不打断攻击，继续：

```
# If the initial prediction is wrong, don't bother attacking, just move on
if init_pred.item() != target.item():
    continue
```

计算损失，将所有现有的梯度归零，将损失反向回传：

```
# Calculate the loss
loss = F.nll_loss(output, target)

# Zero all existing gradients
model.zero_grad()

# Calculate gradients of model in backward pass
loss.backward()
```

收集 datagrad，调用上述的 fgsm_attack 函数进行攻击，然后重新分类受到扰动的图像：

```
# Collect ``datagrad``
data_grad = data.grad.data

# Call FGSM Attack
perturbed_data = fgsm_attack(data, epsilon, data_grad)

# Re-classify the perturbed image
output = model(perturbed_data)
```

检查是否成功。同时保存 0 epsilon 示例的特例，以及一些其它 epsilon

的示例用于后续的可视化:

```
# Check for success
final_pred = output.max(1, keepdim=True)[1] # get the index of the max log-probability
if final_pred.item() == target.item():
    correct += 1
    # Special case for saving 0 epsilon examples
    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:
    # Save some adv examples for visualization later
    if len(adv_examples) < 5:
        adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
        adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex) )
```

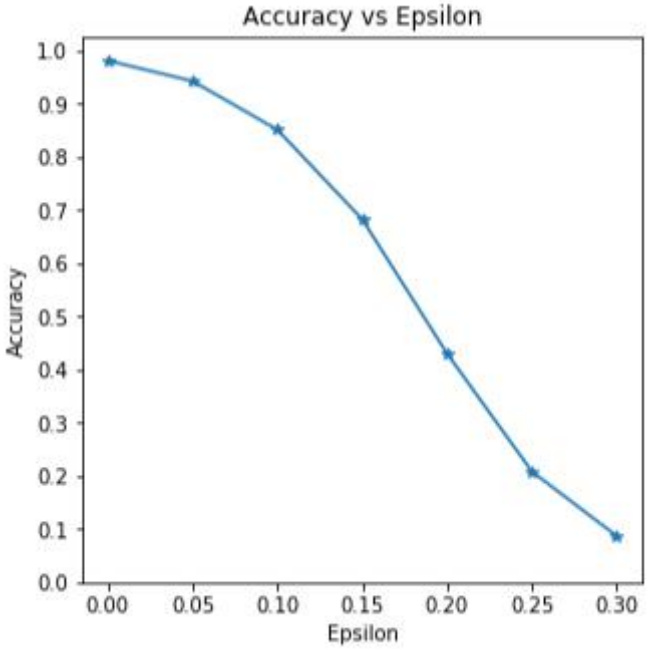
循环结束,计算当前 epsilon 下的最终准确率,返回准确率和对抗性示例:

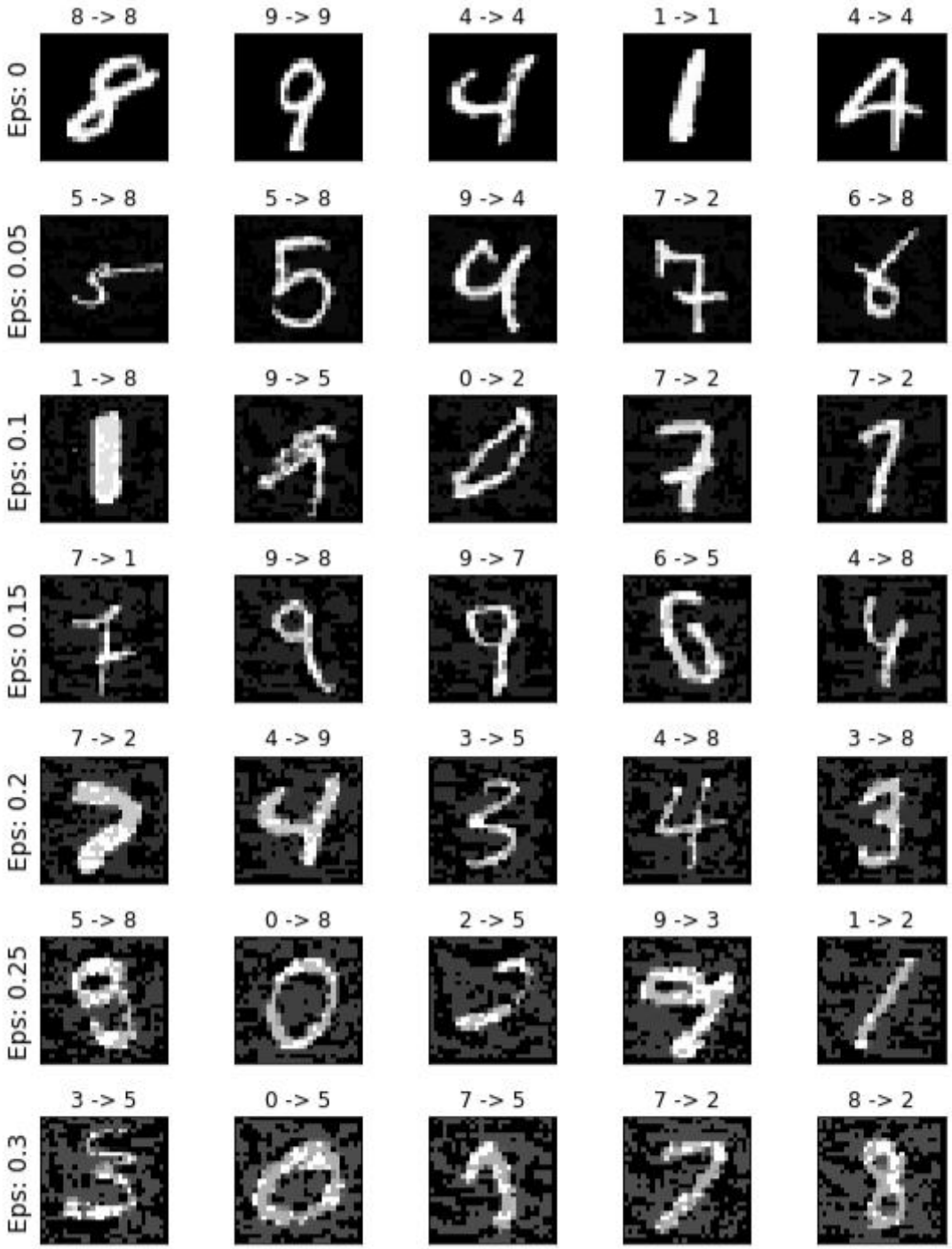
```
# Calculate final accuracy for this epsilon
final_acc = correct/float(len(test_loader))
print("Epsilon: {}\tTest Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader), final_acc))

# Return the accuracy and an adversarial example
return final_acc, adv_examples
```

3. 分别对 默认给出的 epsilons = [0, .05, .1, .15, .2, .25, .3]和自行修改的 epsilons 执行结果进行截图, 并做简要说明。

epsilons = [0, .05, .1, .15, .2, .25, .3] 的结果如下图所示:

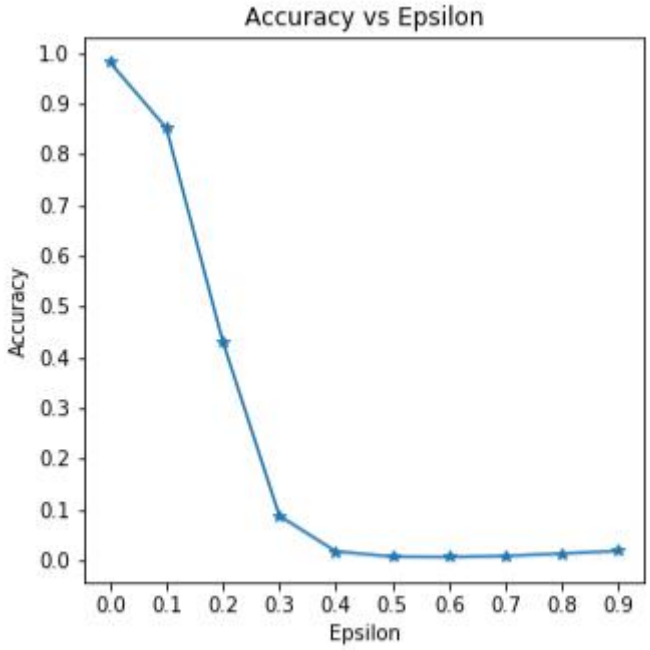
```
Epsilon: 0      Test Accuracy = 9810 / 10000 = 0.981
Epsilon: 0.05   Test Accuracy = 9426 / 10000 = 0.9426
Epsilon: 0.1    Test Accuracy = 8510 / 10000 = 0.851
Epsilon: 0.15   Test Accuracy = 6826 / 10000 = 0.6826
Epsilon: 0.2    Test Accuracy = 4301 / 10000 = 0.4301
Epsilon: 0.25   Test Accuracy = 2082 / 10000 = 0.2082
Epsilon: 0.3    Test Accuracy = 869 / 10000 = 0.0869
```

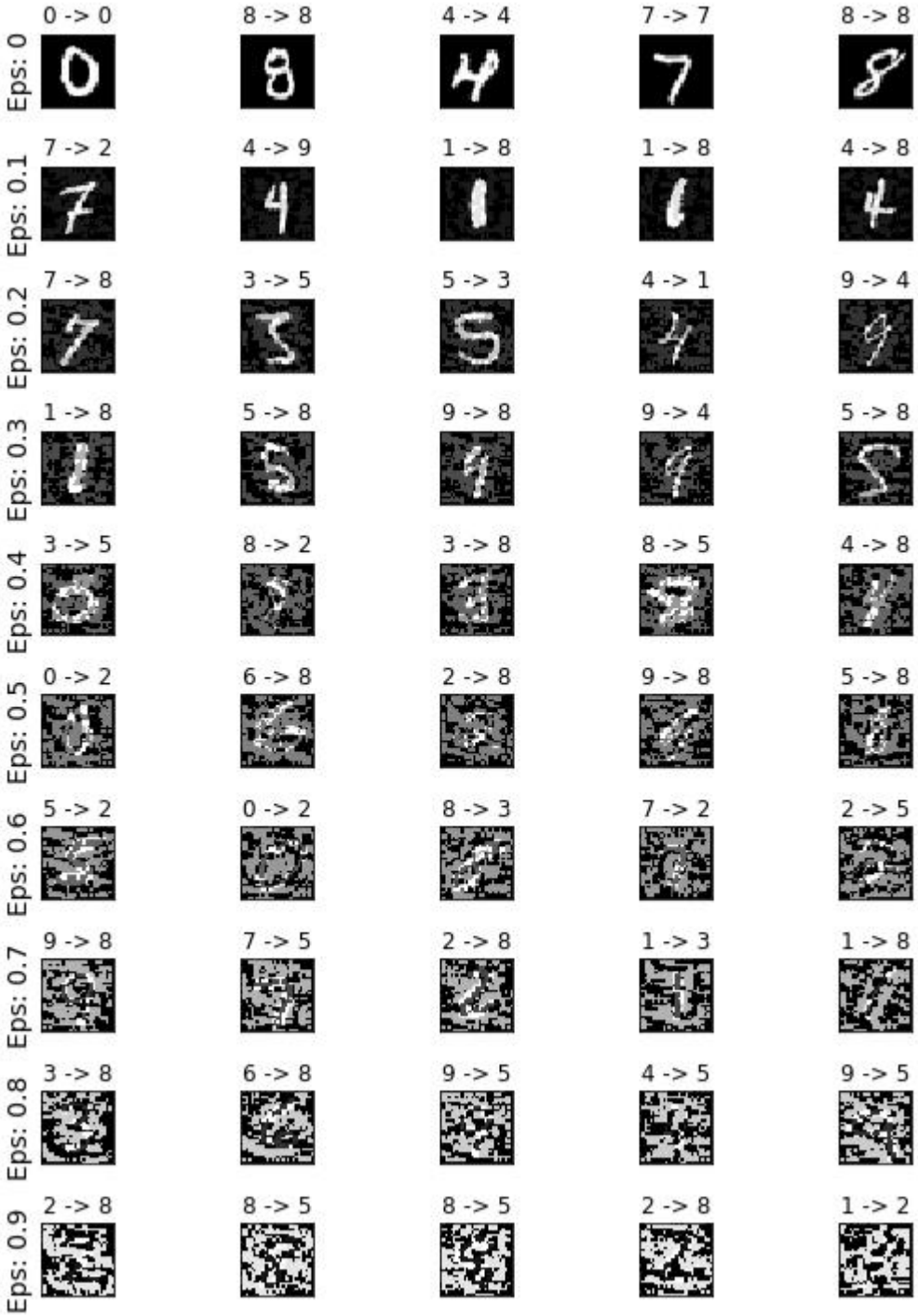

Accuracy vs Epsilon
Accuracy
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30



epsilons = [0, .1, .2, .3, .4, .5, .6] 的结果如下图所示：

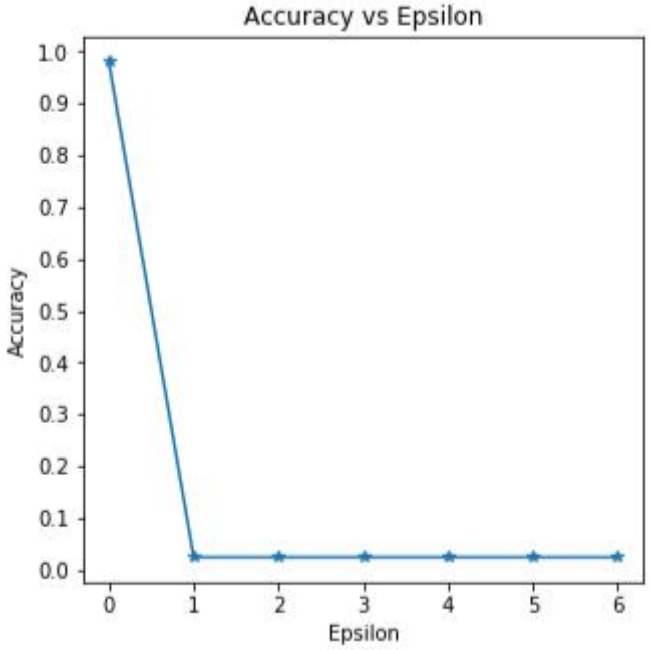
```
Epsilon: 0      Test Accuracy = 9810 / 10000 = 0.981
Epsilon: 0.1    Test Accuracy = 8510 / 10000 = 0.851
Epsilon: 0.2    Test Accuracy = 4301 / 10000 = 0.4301
Epsilon: 0.3    Test Accuracy = 869 / 10000 = 0.0869
Epsilon: 0.4    Test Accuracy = 167 / 10000 = 0.0167
Epsilon: 0.5    Test Accuracy = 63 / 10000 = 0.0063
Epsilon: 0.6    Test Accuracy = 56 / 10000 = 0.0056
Epsilon: 0.7    Test Accuracy = 77 / 10000 = 0.0077
Epsilon: 0.8    Test Accuracy = 124 / 10000 = 0.0124
Epsilon: 0.9    Test Accuracy = 176 / 10000 = 0.0176
```

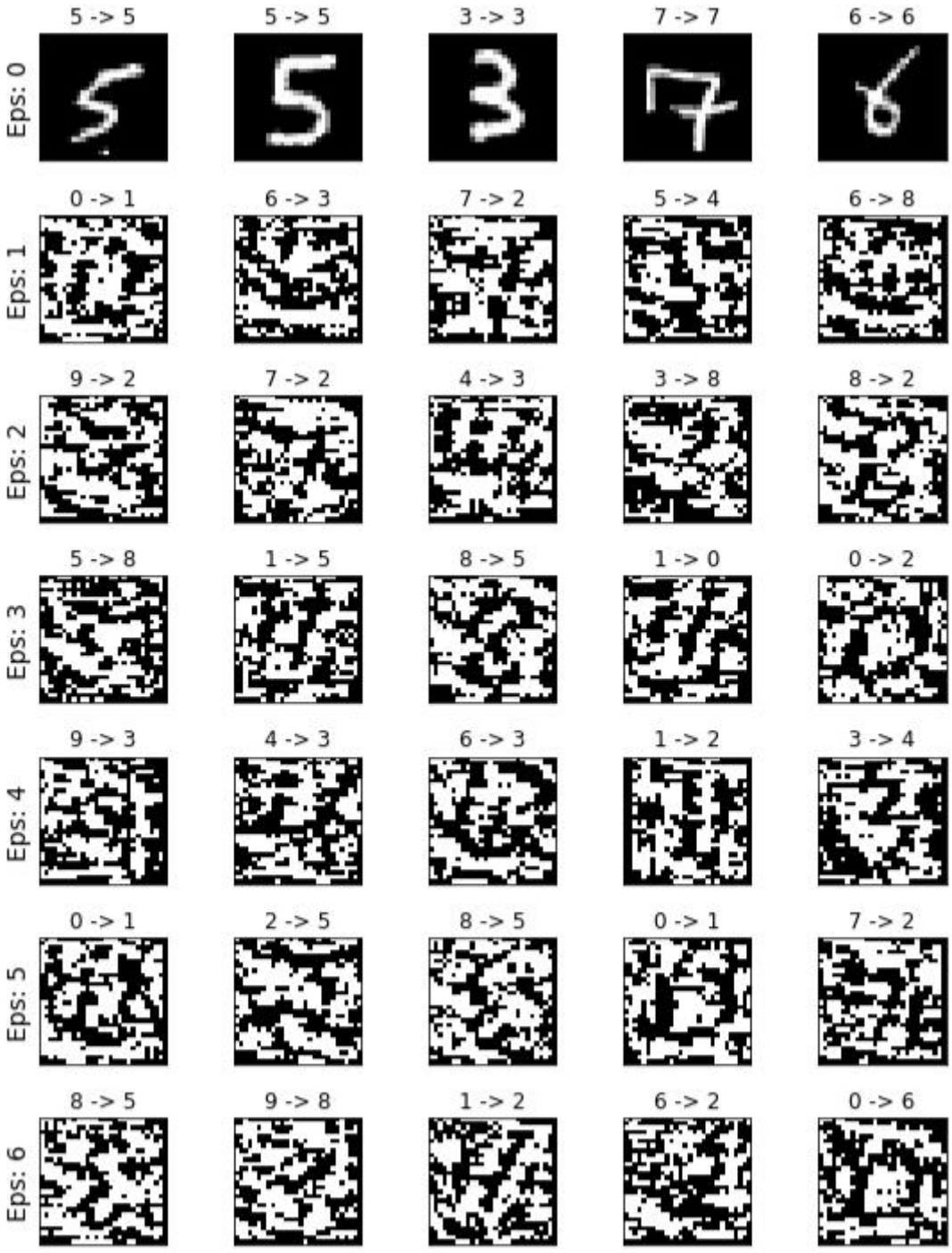




epsilons = [0, 1, 2, 3, 4, 5, 6] 的结果如下图所示：

```
Epsilon: 0      Test Accuracy = 9810 / 10000 = 0.981
Epsilon: 1      Test Accuracy = 239 / 10000 = 0.0239
Epsilon: 2      Test Accuracy = 239 / 10000 = 0.0239
Epsilon: 3      Test Accuracy = 239 / 10000 = 0.0239
Epsilon: 4      Test Accuracy = 239 / 10000 = 0.0239
Epsilon: 5      Test Accuracy = 239 / 10000 = 0.0239
Epsilon: 6      Test Accuracy = 239 / 10000 = 0.0239
```





简要说明:

由上述实验可以看出，随着 epsilon 从 0 开始逐渐增大，模型的准确率在逐渐降低；当 epsilon 在 0 到 0.3 之间时，准确率发生骤降；当 epsilon = 0.3 时，准确率已经小于 0.1；当 epsilon 在 0.3 到 0.6 之间时，准确率依然在小幅度下降；当 epsilon = 0.6 时，准确率最低，为 0.0056；然后当 epsilon 继续

不断增大到 1 时，准确率会有一段小幅度的提升；最终当 epsilon 大于 1 时，准确率维持在 0.0239 保持不变。

并且从给定的示例中可以看出，当 epsilon 在 0 到 0.2 之间时，经过扰动后的图像还十分清晰，人眼可以准确地判断出来；当 epsilon 在 0.2 到 0.7 之间时，经过扰动后的图像变得较为模糊，但通过人眼依然能依稀辨认出来；当 epsilon 大于 0.7 时，图像受扰动之后变化较大，通过人眼已经无法再辨认出来。

二、网络与信息安全实验课程的收获和建议（**必填部分**）

*（关于本学期网络与系统实验的三个部分：系统安全，网络安全和 AI 安全，请给出您对于这三部分实验的收获与体会，给出评论以及**改进的建议**。）*

本学期的实验中系统安全和 AI 安全相对来说较为简单，网络安全部分相对更为困难，我在上面花费了较多的时间。

系统安全部分的指导书较为详细，按照所述步骤进行实验能够比较顺利地完成。网络安全部分，由于第一次接触相关的概念，所以实验的时候会产生一些疑问，有的时候可能指导书上提到了，但是因为对相关概念的印象并不深，所以可能没有注意到指导书的提示，比如要在容器 A 中进行操作，然后再到容器 B 中进行操作，第一次进行相关实验的时候没有注意到要换容器，所以也出现了很多问题。AI 安全部分的实验相对也比较简单，因为实验提供了代码框架，需要做的只是补充完整一些函数，同进行调参并分析结果，完成的比较顺利。总之，在完成网络与信息安全的所有实验之后，我对于网安这门课程有了更深入的认识，网络与信息安全覆盖了各个领域。

改进的建议就是网络安全部分的指导书可以再详细一些，比如标注出一些重点，或者是给出一些错误示例，让同学们知道发生这样的错误可能是什么原因产生的，从而可以自己进行纠正，而无需每个同学遇到错误都向老师询问，也可以

适当减轻老师们的负担。以及最后一部分的 AI 安全实验可以略微加大难度，目前来说有些过于简单了。