# 华中科技大学计算机科学与技术学院

# 《计算机视觉》实验二报告


专　　业：计算机科学与技术

班　　级：CS1903

学　　号：U201914376

姓　　名：沈承磊

指导教师：李贤芝

**完成日期：　2022 年　6 月　1 日**

# 目录

# 一、实验题目：对抗样本攻击与防御

# 二、实验要求

## 2.1 实验任务

在实验一 MNIST 数据集 10 分类手写体数字识别的基础上，使用 FGSM (Fast Gradient Sign Method) 方法对卷积神经网络进行

1) 非定向（untargeted）对抗样本攻击
2) 定向（targeted）对抗样本攻击
3) 防御。

## 2.2 具体要求

1) 非定向攻击要求通过生成对抗样本使得神经网络产生任意不同于原图片真值 i 的输出，即对抗样本使得神经网络输出不等于 i 即可；
2) 定向攻击要求生成对抗样本使得真值为 i 的图片被神经网络误判为 i+1（0 朝 1 攻击，1 朝 2 攻击，以此类推，9 朝 0 攻击）.
3) 探索不同阈值 ε = {1,10,20,50}/255 下的攻击效果.
4) 采用对抗训练方式进行防御，即将非定向攻击生成的对抗样本进行准确的真值标记，联合原有训练数据集在现有神经网络的基础上进行对抗重训练。
5) 通过对经过对抗训练的神经网络再次进行非定向对抗攻击，计算该次攻击成功率来测试防御效果。
6) 做好代码注释。

# 三、实验环境与平台

实验环境：Anaconda 虚拟环境
实验框架：Pytorch 版本 torch1.11.0+cu113
实验平台：Pycharm Professional
实验设备：AMD Ryzen 7 5800H with Radeon Graphics
GPU 型号：RTX3060

# 四、算法设计

## 4.1 数据处理

a) **数据下载：**利用 pytorch 的图形库 torchvision 下载 mnist 数据集，也可以在 huggingface 手动下载。
b) **数据预处理：**由于数据集中每张图片均为 28*28 个大小在[0,255]之间的像素点，因此对每个像素点可采用除以 255 的方式实现归一化。另外避免模型学到输入的顺序信息，每轮训练对训练集使用 shuffle。
c) **注意事项：**由于本次实验将计算部分放置在 GPU 上进行，所以对数据所在设备也要进行处理。其中输入样本要以 FloatTensor 的形式放置在 cuda 上，便于调用 GPU 的算力；标签以 numpy 的形式放置在 cpu 上，便于后期使用 numpy 的一些函数简化 F1 和正确率的计算。

## 4.2 模型算法

本次实验在实验一模型的基础上采用了快速梯度下降（FGSM）作为攻击方法对模型分别进行了非定向攻击和定向攻击，并采用对抗训练的方法进行了防御。其中各步骤原理如下：

**a) 基于快速梯度下降法（FGSM）的非定向攻击：**

**原理分析：**计算损失函数对输入图像的梯度，用符号函数得到其具体的梯度方向，接着乘以一个步长$\varepsilon$，将该“扰动”加在原来的输入图像$x$上就得到了在 FGSM 攻击下的样本$adv_x$。

扰动：$$\boldsymbol{\eta} = \epsilon \text{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)\right)$$

攻击样本：$$adv\_x = x + \epsilon * \text{sign}(\nabla_x \boldsymbol{J}(\theta, x, y))$$

常规的分类模型训练在更新参数时都是将参数减去计算得到的梯度，这样就能使 损失值越来越小，从而模型预测结果越来越准确。既然对抗攻击是希望模型将输入图像进行错误分类，那么就要求损失值越来越大，这和原来的参数更新目的正好相反。因此，只需要在输入图像中加上计算得到的梯度方向，这样修改后的图像经过网络时的损失值就会变大。$\varepsilon$是一个超参数，用于控制扰动的程度，观察不同程度扰动下攻击效果。

**代码实现：**

```python
def generate_adversarial_pattern(input_image, image_label, model, loss_func):
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
    prediction = model(input_image)
    loss = loss_func(prediction, image_label)
    #每次backward前清除上一次loss相对于输入的梯度
    if input_image.grad != None:
        input_image.grad.data.zero_()
    loss.backward()
    gradient = input_image.grad
    #每次backward后清除参数梯度，防止产生其他影响
    optimizer.zero_grad()
    #得到梯度的方向
    signed_grad = torch.sign(gradient)
    return signed_grad
```

图 4-1 梯度方向生成

```python
def attack_fgsm(input_image, image_lable, model, loss_func , eps=0.01):
    #预测原来的样本类别
    # input_image = np.array([input_image])
    # input_image = torch.from_numpy(input_image)
    y_pre = model(input_image)
    pre_prob, pre_index = torch.max(y_pre, 1) #概率 和 类别
    #生成对抗样本
    # loss_func = nn.CrossEntropyLoss()
    input_image.requires_grad = True
    adv_pattern = generate_adversarial_pattern(input_image, image_lable,
                                               model, loss_func)
    clip_adv_pattern = torch.clamp(adv_pattern, 0., 1.)
    perturbed_img = input_image + (eps * adv_pattern)
    perturbed_img = torch.clamp(perturbed_img, 0., 1.)
    #预测对抗样本的类别
    y_adv_pre = model(perturbed_img)
    adv_pre_prob, adv_pre_index = torch.max(y_adv_pre, 1)  # 概率 和 类别

    #可视化
```

图 4-2 非定向攻击对抗样本生成

**b) 基于快速梯度下降法（FGSM）的定向攻击：**

**原理解析：**损失函数的含义在于衡量预测标签与真实标签之间的差异大小，差异越大，则说明预测越不准确，反向传播正是利用这一特性通过改变模型参数大小从而使 Loss 函数值越来越小，以此来达到“训练”模型的目的。在非定向攻击中，我们对攻击成功后的标签并无要求，只是是模型的损失函数结果变大导致模型出现不确定性质的预测不准确现象。对于定向攻击，我们要有针对性的误导模型。直观上只需要让模型在新的“真实标签”下损失函数值越来越小，即在新的“真实标签下”对输入图片进行梯度下降得到扰动。本次实验新的“真实标签”由实际真实标签循环左移得到，如图 4-3 所示。

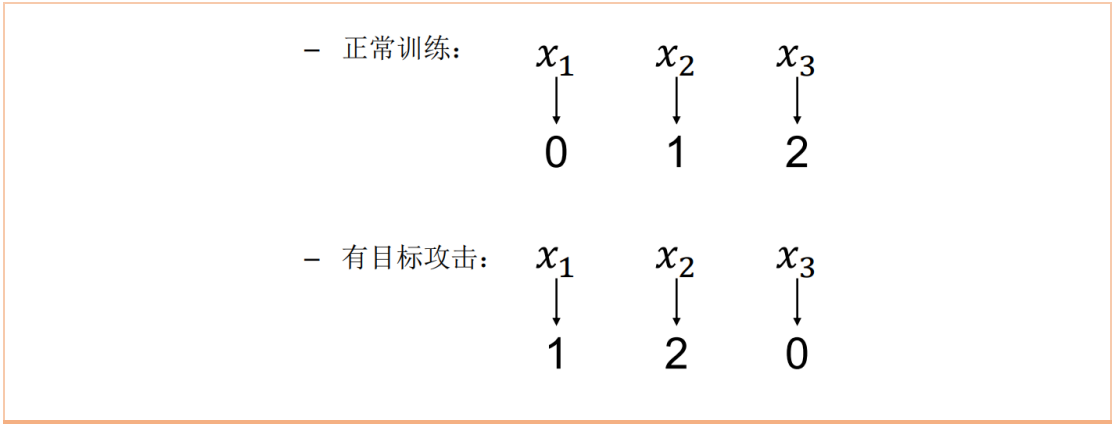



图 4-3 定向攻击新标签示意图

**代码实现：**

如图 4-4 所示，首先利用图 4-1 所示函数 generate_adversarial_pattern 生成梯度方向，与非定向不同的是，这次生成梯度方向时传入的标签为 target_label，即新的“真实标签”，之后遵循梯度下降原则，用原图片减去扰动，相当于进行了一次针对输入的“参数优化”，使得输入图片在模型预测下更接近新的“真实标签”。

```
def targeted_fgsm(input_image, image_label, target_label, model, eps=0.01):
    # 预测原来的样本类别
    y_pre = model(input_image)
    pre_prob, pre_index = torch.max(y_pre, 1)  # 概率 和 类别
    # 生成对抗样本
    input_image.requires_grad = True
    adv_pattern = generate_adversarial_pattern(input_image, target_label model, targeted_loss
    clip_adv_pattern = torch.clamp(adv_pattern, 0., 1.)
    perturbed_img = input_image - (eps * adv_pattern)
    perturbed_img = torch.clamp(perturbed_img, 0., 1.)
# perturbed_img = F.softmax(perturbed_img, dim = 1)
    #perturbed_img = sklearn.preprocessing.MinMaxScaler(feature_range=(0, 1), copy=True)
    # 预测对抗样本的类别
    y_adv_pre = model(perturbed_img)
    adv_pre_prob, adv_pre_index = torch.max(y_adv_pre, 1)  # 概率 和 类别
    # 可视化
```

图 4-4 定向攻击样本生成

在产生梯度方向过程中，关于 loss 函数的选取有多种方式，最为常见的是采用交叉熵函数。可以直接将送入损失函数的标签设为（y + 1）% 10 得到新的“真实标签”。（其中 y 为真实标签。）

另一种经实验证明可行的办法是：我根据定向攻击和损失函数间的内在关系自定义了一种损失函数。首先将模型经 softmax 输出的十个标签下的 pred 相加再减去 2 倍以上（这里取 11 倍）的 target_label 对应的 pred（记为 $pred_{target\_label}$）得到。可行性证明：该损失函数在自身越来越小的变化下，有增大 $pred_{target\_label}$ 减小其余标签对应 pred 的趋势，即让模型预测新的“真实标签”的置信度增大，同时减小其余标签下的置信度。如下图所示：

```
def targeted_loss(logit, target_label):
    y_based = torch.ones(10).to(device)
    y_based[target_label] = -10
    logit = logit.squeeze()
    loss = logit*y_based
    loss = torch.sum(logit*y_based)
    return loss
```

图 4-5 定向攻击样本生成

**c) 基于对抗训练的防御（以非定向攻击为例）：**

**原理解析：**

对抗训练实际上是一个 min-max 优化问题，寻找一个模型（以参数$\theta$表示），使其能够正确分类扰动$\delta$在一定范围 S 内的对抗样本，即：

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[ \max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y) \right]$$

内层（中括号内）是一个最大化，L 则表示在样本 x 上叠加一个扰动$\delta$，再经过神经网络，与标签 y 比较得到的损失。最大化 L，即寻找使损失函数最大的扰动，简单来讲就是添加的扰动要尽量让网络迷惑。外层就是对神经网络进行优化的最小化公式，即当扰动固定的情况下，训练神经网络使得在训练数据上的损失最小，也就是说，使模型具有一定的鲁棒性能够适应这种扰动。我们要做的就是用这种扰动后的样本重新训练模型，使模型能“学会”这种扰动，即在扰动下达到损失最小。

**代码实现：**

如图 4-5 所示：框出部分是封装好的非定向攻击图片生成函数，每次送入训练集的一个 batch，生成扰动后的图片，对实验一中经 20 轮迭代训练好的“残差”网络再次训练（对抗训练），得到新的模型参数，将模型存入本地。

```
def retrain(cnn):
    optimizer = torch.optim.Adam(cnn.parameters(), lr=learning_rate)
    loss_func = nn.CrossEntropyLoss()

    for step, (x_, y_) in enumerate(train_loader):
        x, y = Variable(x_).to(device), Variable(y_).to(device)
        perturbed_x = generate_perturbed_images(x, y, cnn, loss_func, args.eps )
        output = cnn(perturbed_x)
        loss = loss_func(output, y)
        optimizer.zero_grad()  # clear the parameters such as weights and bias,
                               # for the function 'backward' would add the new to the last ones.
        loss.backward()
        optimizer.step()  # conduct the gradient descent using the gradient generated in the 'backward'.
    #将对抗训练后的样本存入本地文件
    f = open(r'./saved_model/retrain_cnn.pickle', 'wb+')
    pickle.dump(cnn, f)
    f.close()
    return cnn
```

图 4-6 模型的“再”训练

# 五、实验结果分析

## a) “残差”网络模型

由于本次实验均在实验一模型基础上进行，为节省训练时间，将实验一中迭代 20 轮训练完毕的类 ResNet“残差”网络模型存入本地，以省去训练时间，效果如图 5-1 所示：

```
D:\SoftwareFamily\Anaconda\envs\nlplab\python.exe D:/nlp/deeplearning_based/chap5.py
using cuda
download cnn from ./saved_model/cnn.pickle
finish downloading !
===== test F1 is  [ 0.988  0.992  0.982  0.967  0.981  0.970  0.988  0.989  0.975  0.981] ===== test sum accuracy is  0.992

进程已结束,退出代码0
```

图 5-1 实验一基础模型效果展示

## b) 非定向攻击

在各个 eps 下测的攻击成功率（ASR）如图 5-2 所示：

```
D:\SoftwareFamily\Anaconda\envs\nlplab\python.exe D:/nlp/deeplearning_based/chap5.py
using cuda
download cnn from ./saved_model/cnn.pickle
finish downloading !
===== test F1 is  [ 0.988  0.992  0.982  0.967  0.981  0.970  0.988  0.989  0.975  0.981] ===== test sum accuracy is  0.992
攻击中...
eps:  1.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.002
攻击中...
eps:  10.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.048
攻击中...
eps:  20.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.110
攻击中...
eps:  50.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.469

进程已结束,退出代码0
```

图 5-2 各个 eps 下非定向攻击 ASR

从以上数据中可以看到，在 eps 分别取[1.0/255, 10.0/255, 20.0/255, 50.0/255]时，攻击成功率逐渐增大，可以得出结论，扰动越大，攻击成功率越高。

将各个 eps 下生成的扰动，以及对抗样本可视化如下图所示：

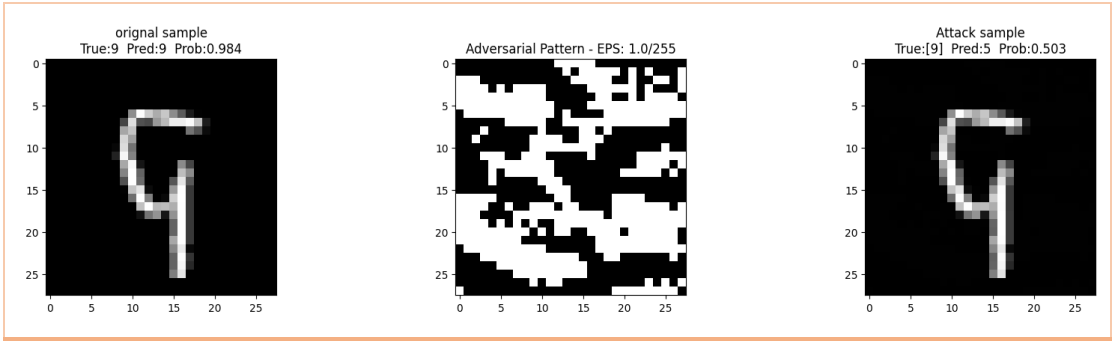



图 5-3 eps = 1/255 时的对抗样本

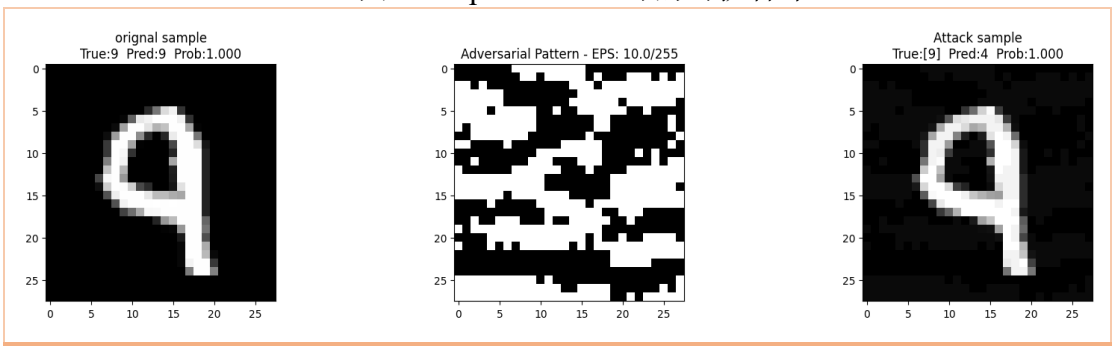



图 5-4 eps = 10/255 时的对抗样本

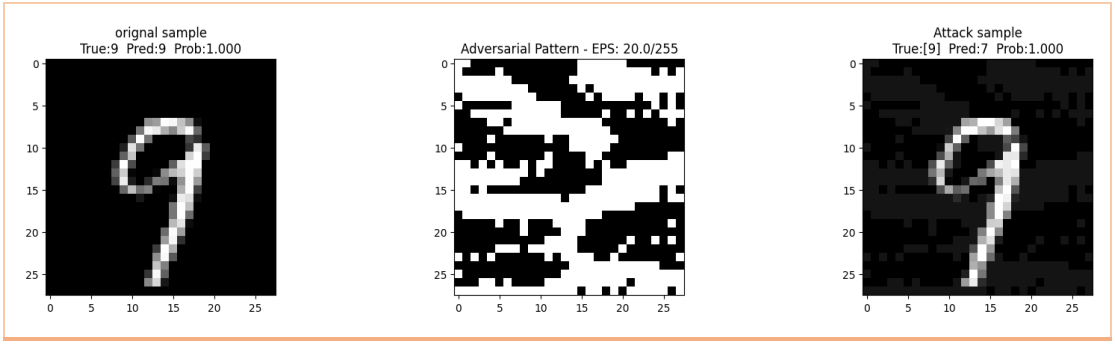



图 5-5 eps = 20/255 时的对抗样本

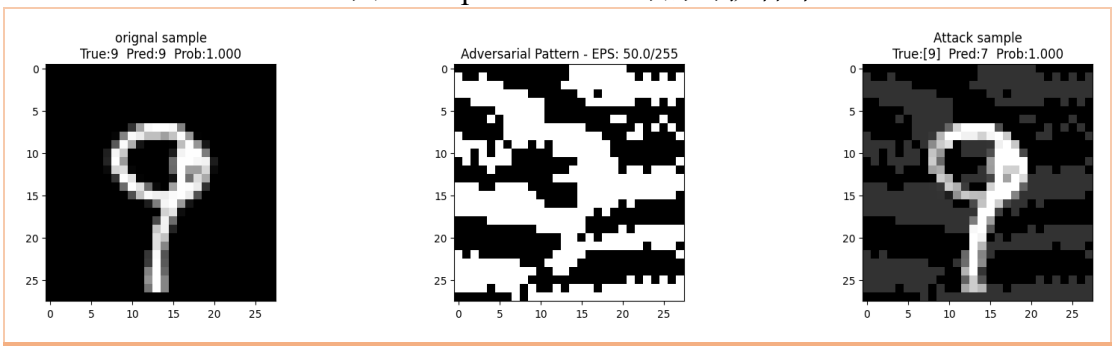
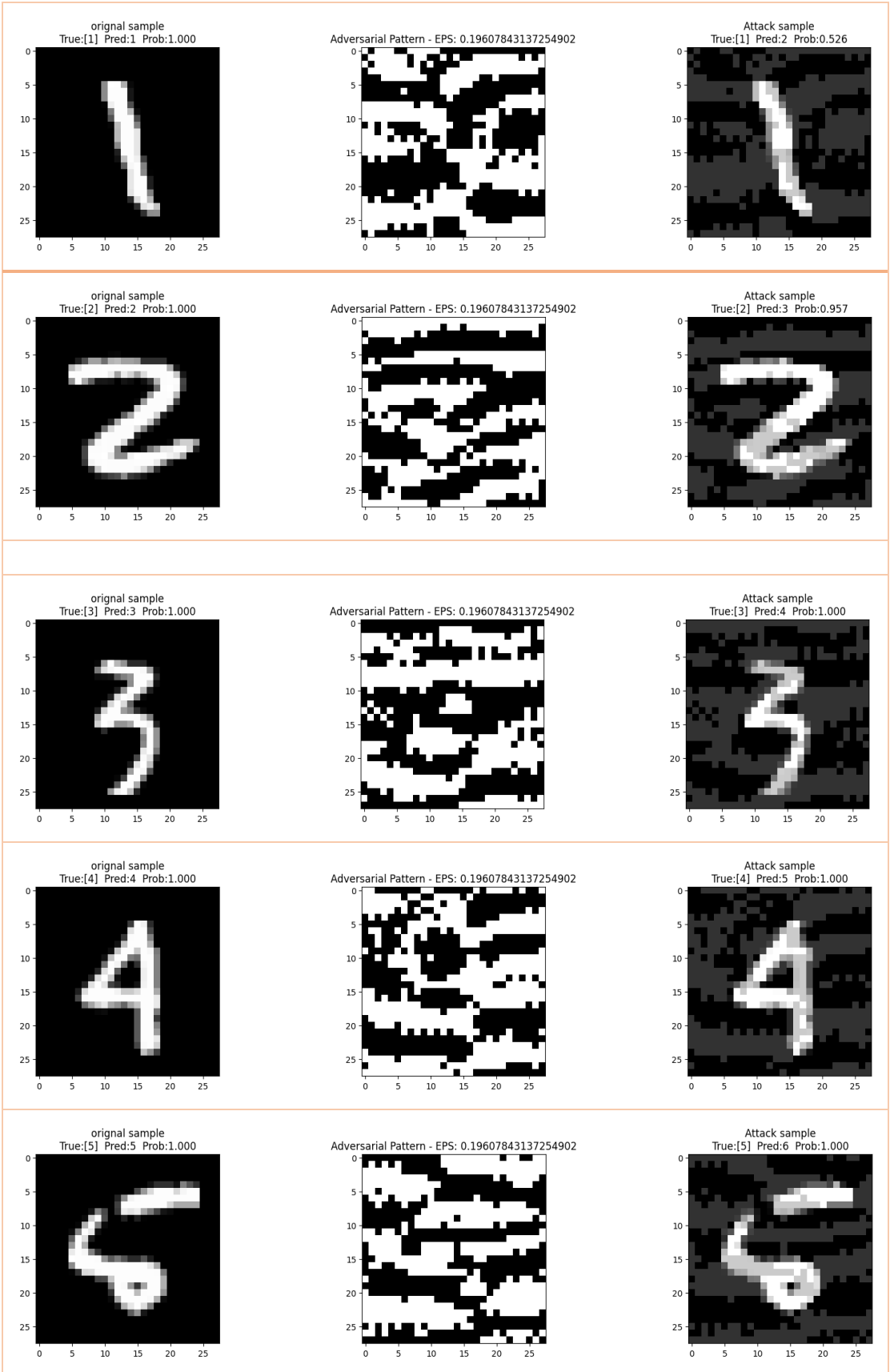



图 5-6 eps = 50/255 时的对抗样本

可以看到随着扰动幅度（eps）的增加，虽然攻击成功率更高了，但是图片被改变的程度也更加明显。

**c) 定向攻击**

在 eps = 50/255 时得到定向攻击的对抗样本如下图所示：



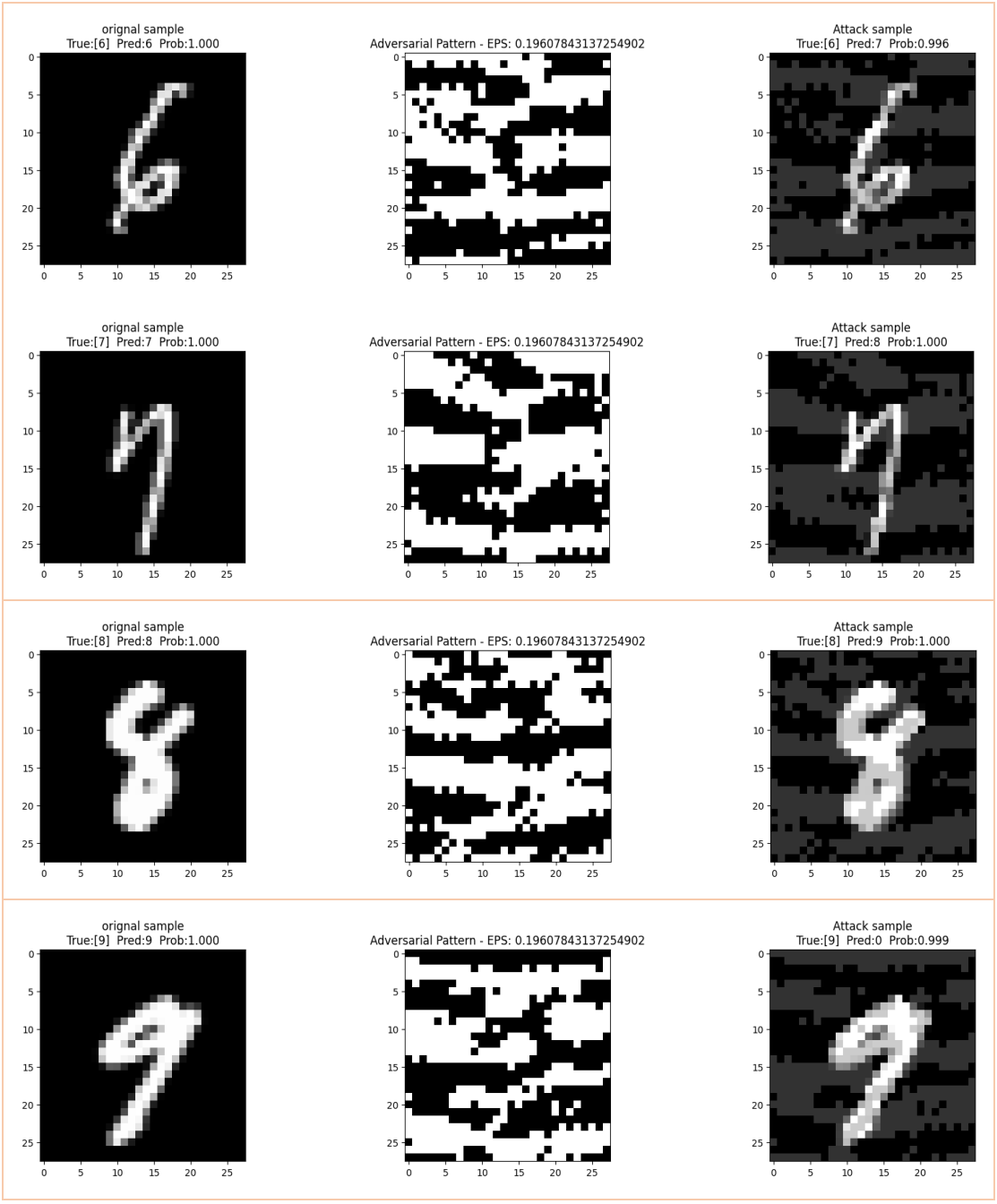



图 5-7 定向攻击下 0-9 展示

对于上文介绍的针对定向攻击的两种方法正确率比对如下：

1）自定义损失函数下结果如图所示：

```
eps:  1.0 /255
attack type is targeted. Attack Successfully Rate(ASR) is: 0.000
攻击中...
eps:  10.0 /255
attack type is targeted. Attack Successfully Rate(ASR) is: 0.002
攻击中...
eps:  20.0 /255
attack type is targeted. Attack Successfully Rate(ASR) is: 0.016
攻击中...
eps:  50.0 /255
attack type is targeted. Attack Successfully Rate(ASR) is: 0.240

进程已结束,退出代码0
```

图 5-8 自定义损失函数下攻击成功率

2）交叉熵损失函数下结果展示：

```
eps:  1.0 /255
attack type is targeted. Attack Successfully Rate(ASR) is: 0.000
攻击中...
eps:  10.0 /255
attack type is targeted. Attack Successfully Rate(ASR) is: 0.002
攻击中...
eps:  20.0 /255
attack type is targeted. Attack Successfully Rate(ASR) is: 0.020
攻击中...
eps:  50.0 /255
attack type is targeted. Attack Successfully Rate(ASR) is: 0.198

进程已结束,退出代码0
```

图 5-9 交叉熵损失函数下攻击成功率

由上可见，在自定义损失函数下，自定义损失函数的效果更好（攻击成功率更高），这是因为自定义损失函数下，我针对性的将 target_label 和其余标签分开，从而能有效增大模型对 target_label 的置信度，并削弱其余标签的置信度。而交叉熵损失函数只针对了 target_label 的置信度，没有削弱其余标签的置信度，从而整体效果没有自定义损失函数好。

**d) 对抗训练**

未受对抗训练前，模型再非定向攻击下的攻击成功率如下图所示：

```
download cnn from ./saved_model/cnn.pickle
finish downloading !
攻击中...
eps:  1.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.000
攻击中...
eps:  10.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.048
攻击中...
eps:  20.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.118
攻击中...
eps:  50.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.465
```

图 5-10 对抗训练前非定向攻击成功率

对抗训练后，模型在非定向攻击下的攻击成功率如下图所示：

```
对抗训练中......
download retrained cnn  from ./saved_model/retrain_cnn.pickle
finish downloading !
攻击中...
eps:  1.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.000
攻击中...
eps:  10.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.012
攻击中...
eps:  20.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.040
攻击中...
eps:  50.0 /255
attack type is randomed. Attack Successfully Rate(ASR) is: 0.126
```

图 5-11 对抗训练后非定向攻击成功率

由以上结果可知，对抗训练后模型的鲁棒性明显增强。在对抗训练前，用非定向攻击攻击模型，攻击成功率在各个 eps 下分别为：0.000、0.048、0.118、0.465，在对抗训练后攻击成功率在各个 eps 下分别为 0.000、0.012、0.040、0.126，可以看到在各个 eps 下攻击成功率都有下降，由此得出对抗训练能有效增强模型鲁棒性的结论。

# 六、实验心得

此次实验在实验一的基础上加入了对抗和防御的内容，整个实验做下来的感觉是很连贯，每一步都是基于上一步开展的。首先是实验一的模型，由于我的模型模仿了 ResNet 残差学习的思想，在测试集上更是达到了 0.994 的准确率，因此模型的性能较好，在进行攻击时，小幅度扰动下的对抗样本攻击成功率并不高（见实验结果分析一栏），但是在对抗训练以后，可以明显看到攻击成功率的下降，因此从结果上说还是非常合理的。另外一个亮点是在进行定向攻击的时候，尝试了一种自定义的损失函数来生成对抗样本，相比于传统的交叉熵函数攻击成功率是有明显提升的（具体定义方法可见模型算法一栏的文字介绍和代码实现，结果对比可见结果分析一栏）。

本次实验由于思路非常顺畅，加上我对于文本对抗，图像对抗等方面非常感兴趣，于是早早地完成了代码部分，并在实验二的课堂上与贤芝老师探讨了实验效果，收获很多。之后我又对代码部分进行了完善，将“训练”“测试”“非定向攻击”“定向攻击”封装了起来，每次跑代码只需要“拨动开关”便可轻松切换功能，非常方便自然，之后会将代码发布到 github，希望能帮助到下一级的学弟学妹。

最后想说的是，今年的 CV 课实验是多有科目中体验最好的一门实验，难度不算很大，但是能让我们把握整个对抗训练的过程，我觉得这是最重要的，感谢贤芝老师的指导和同学们的帮助，希望 CV 课越办越好！