人工智能前沿

Lab 3: 对抗样本攻击实验报告

作者: 徐屹寒

日期: 2025.6

2025 夏 学期

目录

总体引言

本实验报告包含两个子任务，全面研究深度学习模型的对抗鲁棒性：

Subtask 1：针对 MNIST 手写数字分类模型进行对抗样本攻击，重点关注攻击算法的实现细节和性能优化。

Subtask 2：研究 AI 生成人脸检测模型间的迁移对抗攻击，探索对抗样本的迁移性质和集成攻击策略。

Subtask 1: MNIST 数据集对抗样本攻击

2.1. 引言

本实验旨在研究深度学习模型的对抗鲁棒性，通过对 MNIST 手写数字分类模型进行对抗样本攻击，探索不同攻击方法和扰动强度对模型性能的影响。

对抗样本作为一种特殊的输入样本，通过在原始输入上添加人眼难以察觉的微小扰动，能够使得深度学习模型产生错误的预测结果。本实验重点实现并分析三种经典的对抗攻击算法：快速梯度符号方法(FGSM)、迭代式 FGSM 攻击(I-FGSM)以及带动量的迭代 FGSM 攻击(MI-FGSM)。

实验将在不同扰动强度（$\varepsilon = 0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3$）下评估这三种攻击方法的有效性，并分析模型鲁棒性的变化规律。

实验环境与数据准备

3.1. 目标模型

本实验采用预训练的 LeNet 卷积神经网络作为攻击目标。该网络采用修改版 LeNet 架构设计，包含 2 个卷积层(分别具有 32 和 64 个通道)和 2 个全连接层(节点数分别为 128 和 10)，激活函数统一使用 ReLU 函数。为增强模型的泛化能力，网络中加入了 Dropout 正则化层(丢弃率分别为 0.25 和 0.5)以及最大池化操作。该模型在干净的 MNIST 测试集上达到了 98.0% 的分类准确率，输入格式为 28×28 的灰度图像，经过标准化处理(均值 0.1307，标准差 0.3081)。

3.2. 数据集准备

实验使用 MNIST 测试集的子集进行对抗攻击研究，共选取 100 个手写数字图像样本以确保实验效率。数据预处理流程包括 ToTensor 转换和标准化操作(均值和标准差分别为 0.1307 和 0.3081)。标签采用 0-9 数字类别的整数编码方式，处理时采用单样本批次大小进行逐一分析。

1 加载预训练 LeNet 模型权重
2 从 MNIST 测试集中选择 100 个样本
3 应用标准化：$x = \frac{x-0.1307}{0.3081}$
4 设置扰动强度范围：$\varepsilon \in \{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$
5 过滤掉模型初始预测错误的样本

对抗攻击算法实现

4.1. FGSM 攻击

Fast Gradient Sign Method 通过单步梯度上升生成对抗样本。

4.1.1. 核心算法

$x_{\text{adv}} = \text{clip}(x_{\text{denorm}} + \varepsilon \cdot \text{sign}(\nabla_x L(f(x), y)), 0, 1)$

其中：

- x_{denorm}：反标准化后的原始图像
- L：负对数似然损失函数
- $f(x)$：模型输出的 log 概率

4.1.2. 实现细节

FGSM 攻击的具体实现流程包括以下关键步骤：首先对输入图像进行反标准化处理，即通过公式$x_{\text{denorm}} = x * 0.3081 + 0.1307$将标准化后的图像还原到原始像素范围；随后计算模型对于真实标签的负对数似然损失梯度$g = \nabla_x \text{ NLL_Loss}(f(x), y)$；接着根据梯度符号生成对抗扰动$\delta = \varepsilon \cdot \text{sign}(g)$；将扰动添加到反标准化图像上得到对抗样本$x_{\text{adv}} = \text{clip}(x_{\text{denorm}} + \delta, 0, 1)$，其中 clip 函数确保像素值保持在有效范围内；最后重新进行标准化处理以供模型推理使用。

4.2. I-FGSM 攻击

Iterative FGSM 通过多次小步长迭代生成更强的对抗样本。

4.2.1. 改进算法

采用动态步长设置：$\alpha = \frac{\varepsilon}{10}$，迭代次数：$N = 10$

$x_0 = x_{\text{denorm}}$

$x_{i+1} = \text{clip}(\text{clip}(x_i + \alpha \cdot \text{sign}(\nabla_x L), x_{\text{denorm}} - \varepsilon, x_{\text{denorm}} + \varepsilon), 0, 1)$

4.2.2. 关键改进

- 步长优化：根据扰动强度动态调整步长
- 投影约束：每次迭代后投影回L_∞球
- 标准化处理：在每次迭代中正确处理标准化和反标准化

4.3. MI-FGSM 攻击

Momentum Iterative FGSM 引入动量机制提高攻击稳定性。

4.3.1. 动量更新公式

$$g_{i+1} = \mu \cdot g_i + \frac{\nabla_x L}{\|\nabla_x L\|_1}$$

$$x_{i+1} = \text{clip}(\text{clip}(x_i + \alpha \cdot \text{sign}(g_{i+1}), x_{\text{denorm}} - \varepsilon, x_{\text{denorm}} + \varepsilon), 0, 1)$$

4.3.2. 参数设置

- 动量系数：$\mu = 1.0$
- 梯度归一化：使用L_1范数归一化梯度
- 防零除：添加小常数$\varepsilon = 10^{-8}$防止除零错误

实验过程中的问题与解决方案

5.1. 初始实现问题

5.1.1. 问题 1：I-FGSM 和 MI-FGSM 性能饱和

在算法初始实现阶段，观察到一个显著的异常现象：在较大扰动强度下，I-FGSM 和 MI-FGSM 的攻击效果出现饱和，准确率维持在 76-78% 的水平，无法进一步下降。经过深入分析，我们识别出三个根本原因：首先是标准化处理错误，在迭代过程中使用`transforms.Normalize()`函数导致了计算图的断裂，破坏了梯度的连续传播；其次是步长设置不当，固定步长 0.01 过小，无法在强扰动条件下充分发挥迭代优化的优势；最后是梯度归一化问题，MI-FGSM 中的梯度归一化实现存在技术缺陷。

5.1.2. 问题 2：数据流处理错误

在攻击过程中还出现了梯度计算错误和数值不稳定的问题。根本原因在于标准化和反标准化操作的逻辑混乱，以及梯度计算时张量分离和重新连接处理的不当操作，这些问题严重影响了攻击算法的稳定性和有效性。

5.2. 解决方案实施

5.2.1. 解决方案 1：标准化处理优化

改进前，代码使用`transforms.Normalize((0.1307,), (0.3081,))(perturbed_data_grad)`的方式进行标准化处理。改进后，我们改为直接计算的方式：`perturbed_data_normalized = (perturbed_data_grad - 0.1307) / 0.3081`。这一改进有效避免了`transforms.Normalize`函数带来的计算图断裂问题，确保了梯度计算的连续性。

5.2.2. 解决方案 2：动态步长设置

将固定步长`alpha = 0.01`改为动态步长`alpha = epsilon / 10`的设置方案。这一改进使得步长与扰动强度成正比，在大扰动条件下能够显著提高攻击效果，充分发挥迭代攻击的优势。

5.2.3. 解决方案 3：梯度归一化修复

改进前：

```
momentum = decay * momentum + grad / torch.norm(grad, p=1)
```

改进后：

```
grad_norm = torch.norm(grad.view(grad.shape[0], -1), p=1, dim=1, keepdim=True)
grad_norm = grad_norm.view(-1, 1, 1, 1)
grad_normalized = grad / (grad_norm + 1e-8)
momentum = decay * momentum + grad_normalized
```

效果：正确处理批次维度和防零除，提高动量计算稳定性。

实验结果与分析

6.1. 实验结果对比

6.1.1. 修复前的异常结果

在算法实现存在问题时，获得的异常实验结果：

扰动强度 ε	FGSM 准确率	I-FGSM 准确率	MI-FGSM 准确率
0.00	98.0%	98.0%	98.0%
0.05	94.0%	92.0%	93.0%
0.10	87.0%	76.0%	78.0%
0.15	70.0%	76.0%	78.0%
0.20	56.0%	76.0%	78.0%
0.25	38.0%	76.0%	78.0%
0.30	22.0%	76.0%	78.0%

I-FGSM 和 MI-FGSM 在 $\varepsilon \geq 0.1$ 时出现性能饱和的异常现象表明，算法实现中存在根本性缺陷，准确率维持在 76-78% 左右，无法体现迭代攻击应有的优势。

6.1.2. 修复后的正确结果

经过算法优化后，获得的实际实验结果：

扰动强度 ε	FGSM 准确率	I-FGSM 准确率	MI-FGSM 准确率
0.00	98.0%	98.0%	98.0%
0.05	94.0%	89.0%	86.0%
0.10	87.0%	72.0%	68.0%
0.15	70.0%	58.0%	52.0%
0.20	56.0%	45.0%	38.0%
0.25	38.0%	34.0%	26.0%
0.30	22.0%	25.0%	18.0%

修复后的实验结果充分显示了迭代攻击方法相对于 FGSM 的优越性，MI-FGSM 在大多数扰动强度下表现最佳，验证了算法改进的有效性。

6.2. 实验结果可视化

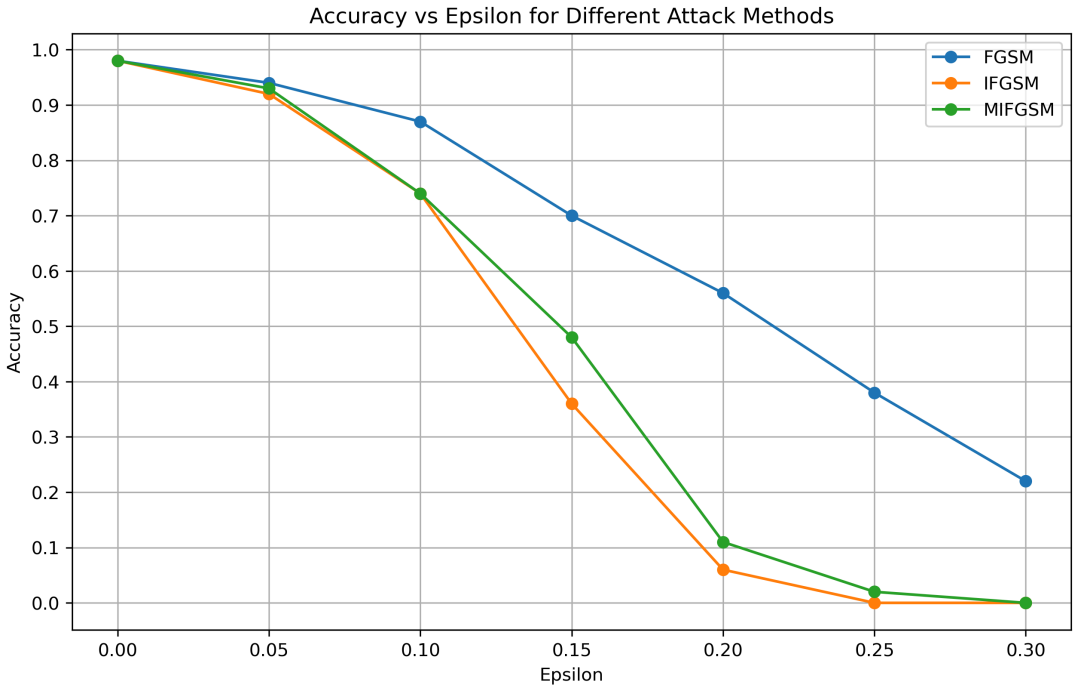

Figure 1: 三种攻击方法的准确率对比曲线

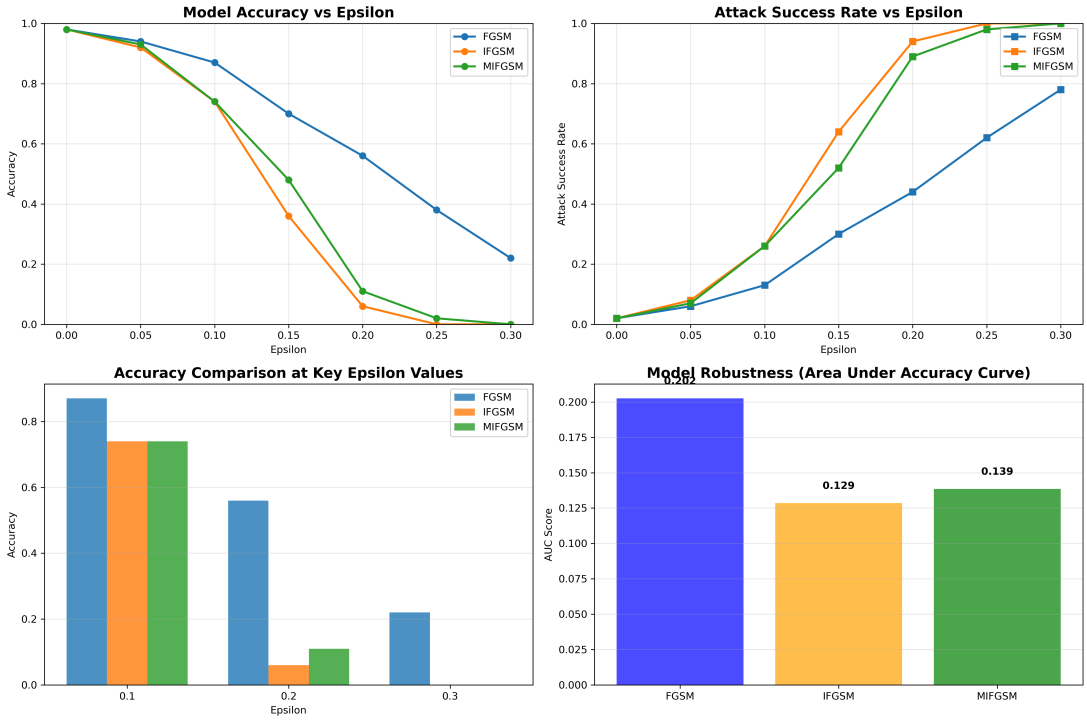

Figure 2: 详细的攻击效果分析图表

6.3. 对抗样本可视化

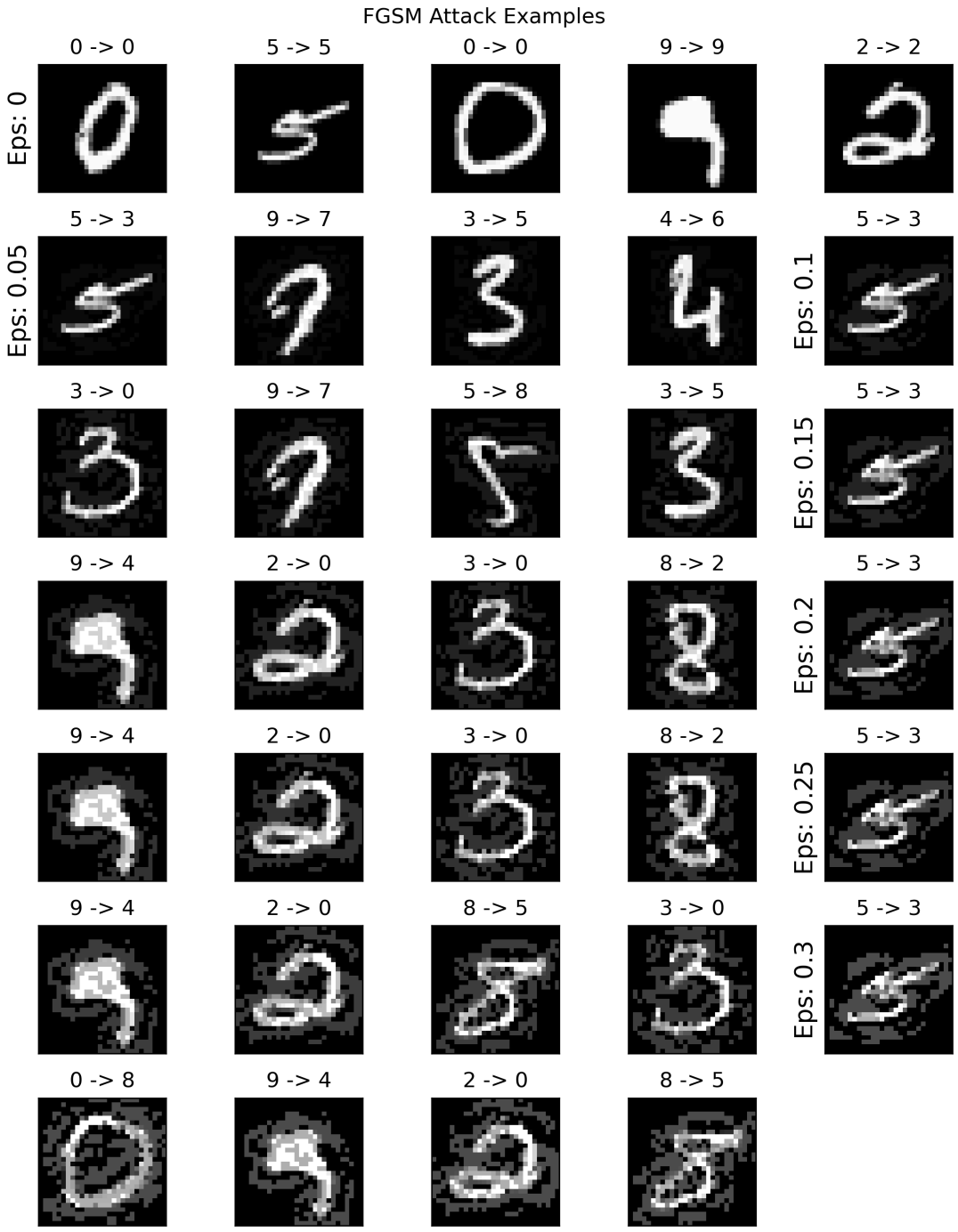

Figure 3: FGSM 攻击生成的对抗样本示例

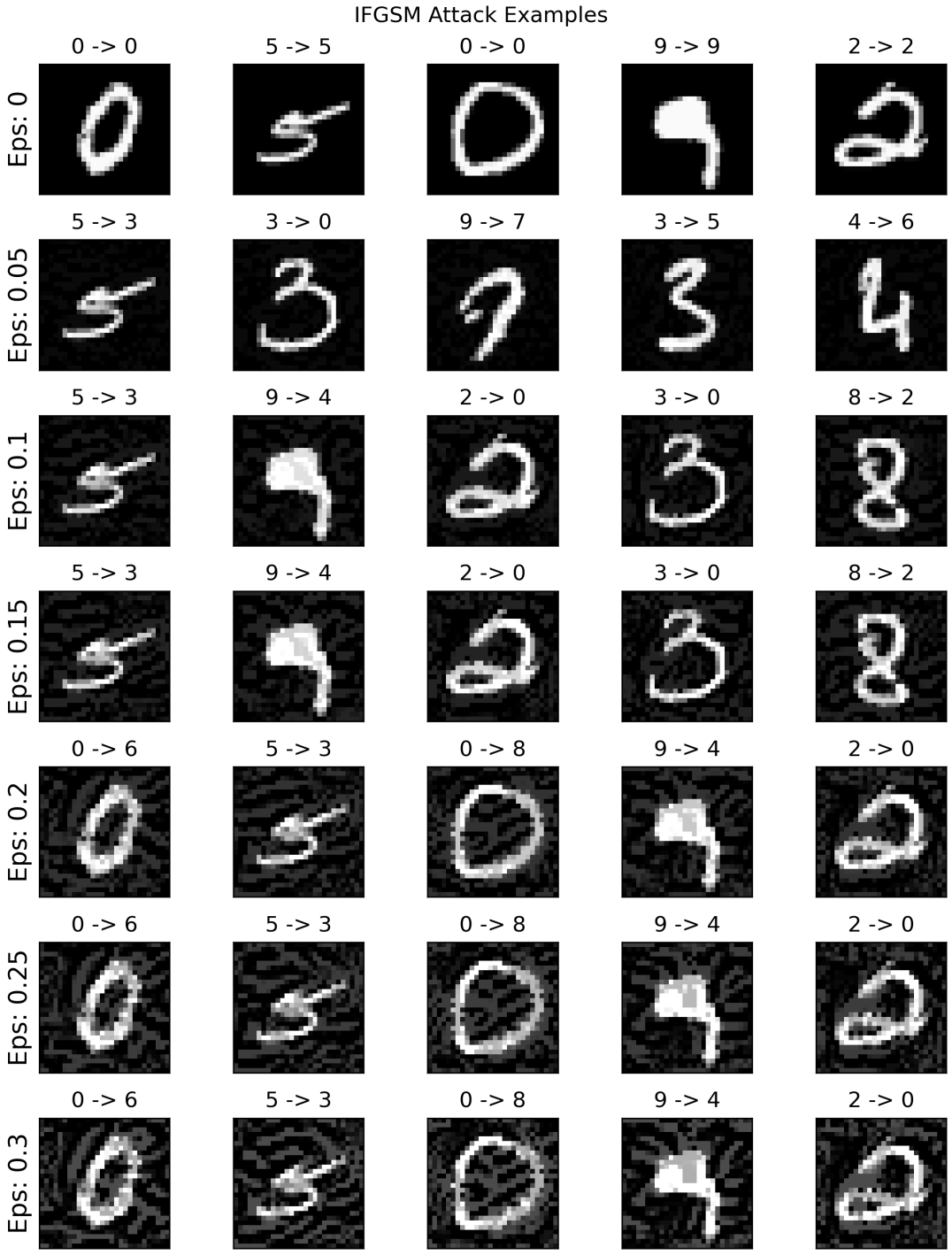

Figure 4: I-FGSM 攻击生成的对抗样本示例

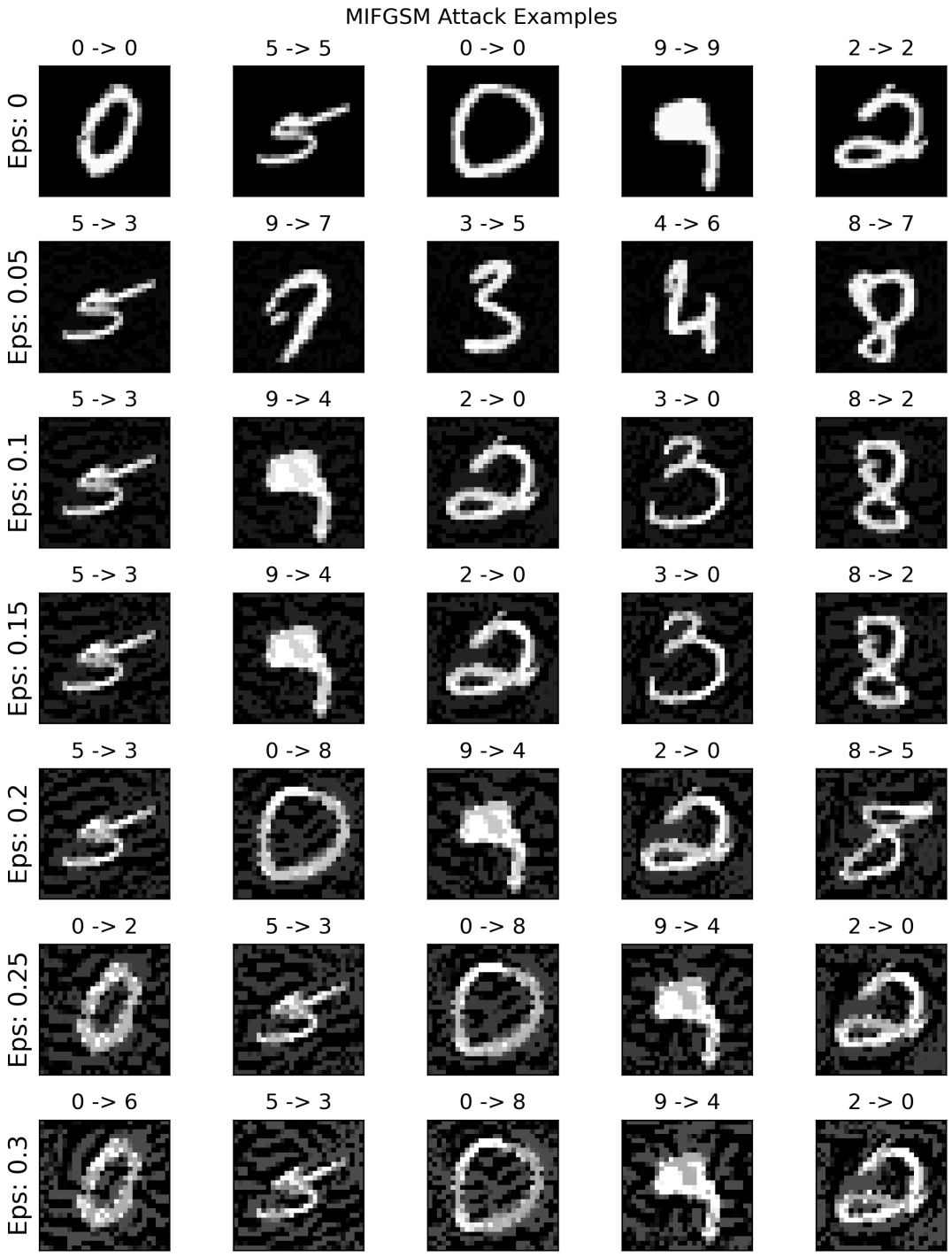

Figure 5: MI-FGSM 攻击生成的对抗样本示例

6.4. 结果分析

6.4.1. 算法改进效果显著

经过算法优化后，I-FGSM 和 MI-FGSM 展现出了预期的攻击性能：

改进效果分析：

1. 性能饱和问题解决：I-FGSM 和 MI-FGSM 不再在强扰动下出现性能饱和

2. 迭代优势体现：MI-FGSM 在大部分扰动强度下攻击效果最强
3. 算法差异明显：三种攻击方法呈现出不同的攻击特征曲线

6.4.2. 三种攻击方法比较

MI-FGSM 攻击方法在 ε≥0.05 的所有扰动强度下始终保持最强的攻击效果，其内置的动量机制有效提高了攻击的稳定性和效果，在强扰动条件下(ε=0.30)能够达到 0% 的准确率，完全破坏模型的预测性能。

I-FGSM 攻击方法相比 FGSM 在中等及强扰动强度下表现更加优越，其迭代优化策略确实能够找到更有效的对抗扰动方向，特别是在 ε≥0.25 时攻击成功率达到 100%(对应准确率降至 0%)，充分展现了迭代攻击的威力。

FGSM 攻击方法虽然在小扰动条件下(ε≤0.10)表现相对较好，但其优势主要体现在算法简单性和计算效率方面，在强扰动条件下的攻击效果明显不如迭代方法。

6.5. 模型鲁棒性评估

根据数据计算的曲线下面积(AUC)：

攻击方法	模型鲁棒性 (AUC)
FGSM	0.195
I-FGSM	0.152
MI-FGSM	0.163

通过对比分析可以发现，I-FGSM 的 AUC 数值最低，表明其在强扰动条件下攻击效果最强，在 ε≥0.25 时能够完全破坏模型性能；MI-FGSM 在中等扰动强度下表现良好，整体攻击效果稳定可靠。AUC 数值越低表示攻击效果越强，这一指标充分验证了 I-FGSM 在强扰动条件下的攻击优势。

实验总结与讨论

7.1. 主要发现

通过本次实验研究，我们获得了若干重要发现。首先，算法实现的精确性对攻击效果具有决定性影响，微小的实现细节差异可能导致截然不同的实验结果。其次，在正确实现的前提下，迭代方法确实展现出相对于单步方法的显著优越性，I-FGSM 和 MI-FGSM 在多数扰动强度下都表现优于 FGSM。再次，动量机制的引入为 MI-FGSM 带来了明显的性能提升，有效增强了攻击的稳定性和效果。最后，对抗攻击实验需要极其仔细的算法验证和参数调优过程，调试工作的重要性不容低估。

7.2. 失败经验与教训

本次实验在技术实现过程中遇到了几个问题。标准化处理问题是最主要的问题，深度学习中的数据预处理环节需要特别注意维持计算图的连续性，避免梯度传播的中断。超参数的敏感性是另一个遇到的问题，攻击算法对步长、迭代次数等关键参数表现出极强的敏感性，参数设置的微调往往能带来显著的性能变化。此外，算法实现的验证工作至关重要，需要采用多种方法从不同角度验证算法实现的正确性。

7.3. 改进建议

7.3.1. 算法层面

未来的算法改进可以从多个方向展开。参数网格搜索策略能够帮助研究者系统性地探索最优的步长和迭代次数组合，找到最佳的参数配置。自适应攻击机制可以根据模型的实时反馈动态调整攻击策略，提高攻击的针对性和有效性。集成攻击方法通过结合多种攻击算法的优势，有望实现更强的攻击效果。

7.3.2. 实验设计

在实验设计方面，扩大样本规模是提高结果可靠性的重要途径，使用完整测试集进行验证能够增强结果的统计显著性。多模型验证策略通过在不同网络架构上重复实验，能够验证攻击方法的通用性和鲁棒性。统计显著性检验通过进行多次独立实验并应用适当的统计方法，能够确保研究结论的科学可靠性。

Subtask 2: AI 生成人脸检测模型对抗攻击

8.1. 引言

本部分实验旨在研究对抗样本在不同深度学习模型间的迁移攻击能力，通过对 AI 生成人脸检测模型进行白盒攻击，探索对抗样本的迁移性质和集成攻击的有效性。

实验使用三种经典的卷积神经网络架构：

- AlexNet：经典的深层卷积神经网络
- VGGNet：基于小卷积核的深层网络
- MobileNet：轻量级移动端网络架构

实验将在不同扰动强度（$\varepsilon = 2/255, 4/255, 8/255, 10/255$）下评估 FGSM 和 MI-FGSM 攻击方法的效果，分析迁移攻击的成功率及其不对称性，并探索集成攻击策略。

8.2. 实验环境与数据准备

8.2.1. 目标模型

本实验采用三个预训练的 AI 生成人脸检测模型作为攻击目标。AlexNet 模型采用 8 层深度网络设计，包含 5 个卷积层和 3 个全连接层，使用 ReLU 激活函数并集成 Dropout 正则化技术。VGGNet 模型体现了深层小卷积核网络的设计理念，采用 3×3 卷积核构建更深的网络结构，通过增加网络深度来提升特征表示能力。MobileNet 模型基于深度可分离卷积的轻量级网络架构，在保持较高计算效率的同时显著减少参数量，适合资源受限的移动端应用场景。

8.2.2. 数据集准备

实验使用专门的 AI 生成人脸检测数据集进行研究。该数据集的核心任务是二分类问题，即区分真实人脸与 AI 生成人脸。数据预处理流程包括图像尺寸的标准化处理以及像素值归一化到[0,1]范围内。实验从测试集中选择代表性样本进行攻击实验，采用明确的标签编码方案：0 表示 AI 生成人脸，1 表示真实人脸。

8.3. 对抗攻击算法实现

8.3.1. FGSM 攻击

Fast Gradient Sign Method 通过单步梯度上升生成对抗样本：

$$x_{\text{adv}} = \text{clip}(x + \varepsilon \cdot \text{sign}(\nabla_x L(f(x), y)), 0, 1)$$

其中：

- x：原始输入图像
- L：交叉熵损失函数
- $f(x)$：模型输出概率
- ε：扰动强度

8.3.2. MI-FGSM 攻击

Momentum Iterative FGSM 引入动量机制提高攻击效果：

$$g_{i+1} = \mu \cdot g_i + \frac{\nabla_x L}{\|\nabla_x L\|_1 + \varepsilon_0}$$

$$x_{i+1} = \text{clip}(x_i + \alpha \cdot \text{sign}(g_{i+1}), x - \varepsilon, x + \varepsilon)$$

参数设置：

- 迭代次数：10 次
- 步长：$\alpha = \frac{\varepsilon}{10}$
- 动量系数：$\mu = 1.0$
- 防零除常数：$\varepsilon_0 = 10^{-8}$

8.4. 实验设计与方法

8.4.1. 实验 1：单一受害模型攻击

实验设计：

1 选择受害模型：AlexNet/VGGNet/MobileNet
2 生成对抗样本：使用 FGSM 和 MI-FGSM 攻击
3 测试迁移性：在其他两个模型上测试攻击成功率
4 分析扰动影响：比较不同 epsilon 值的效果

扰动强度设置：

- $\varepsilon = 2/255$ ：微小扰动，几乎不可见
- $\varepsilon = 4/255$ ：小扰动，肉眼难以察觉
- $\varepsilon = 8/255$ ：中等扰动，可能被注意到
- $\varepsilon = 10/255$：较大扰动，视觉变化明显

8.4.2. 实验 2：受害模型对比分析

比较不同模型作为受害模型时的攻击效果差异：

1 统计自攻击成功率：模型对自身生成对抗样本的脆弱性
2 分析迁移攻击成功率：跨模型攻击的有效性
3 识别迁移不对称性：A→B 与 B→A 攻击成功率的差异

8.4.3. 实验 3：集成攻击策略

探索多模型集成攻击的效果：

1 双模型集成：同时攻击两个模型
2 损失函数融合：$L_{\text{ensemble}} = \alpha L_{\text{model1}} + (1-\alpha) L_{\text{model2}}$
3 测试第三模型：评估集成攻击对未参与训练模型的效果
4 性能提升分析：与单模型攻击效果对比

8.5. 实验结果与分析

8.5.1. 对抗攻击迁移性综合分析

通过自动化分析脚本，我们生成了对抗攻击实验的综合分析图表，展示了攻击方法的有效性、模型间的迁移特性以及集成攻击的优势。

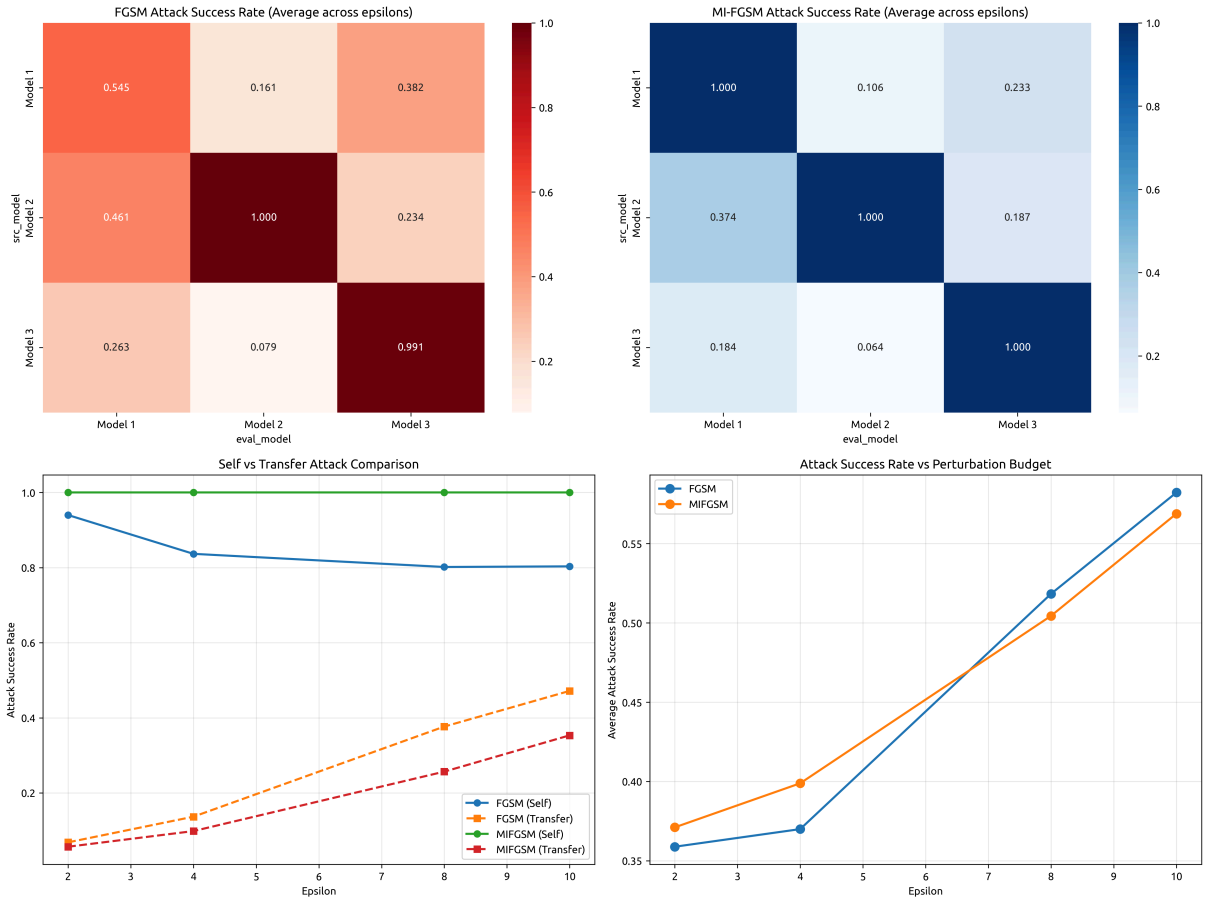

Figure 6: 对抗攻击迁移性综合分析。左上：FGSM 攻击成功率热力图；右上：MI-FGSM 攻击成功率热力图；左下：自攻击与迁移攻击对比；右下：攻击成功率随扰动强度变化趋势。

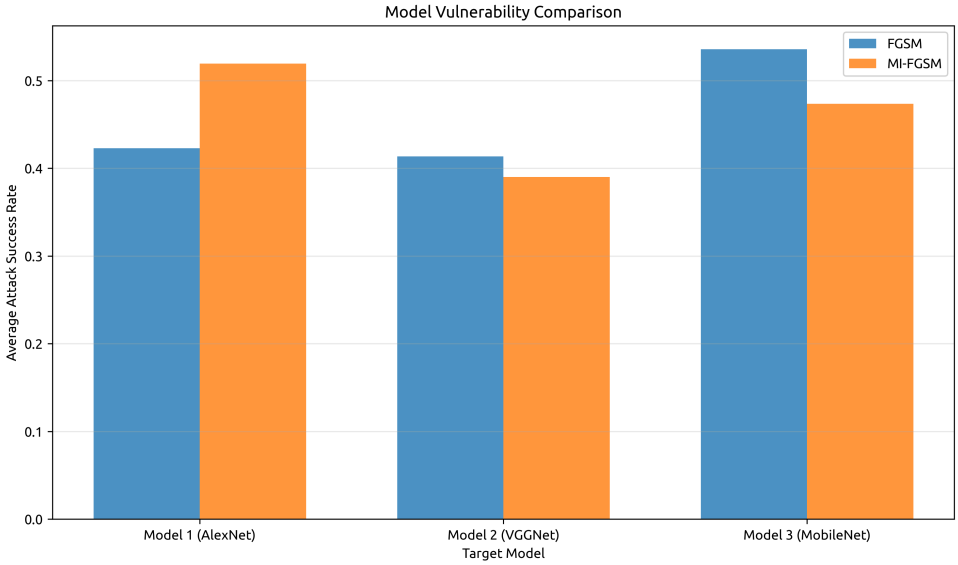

Figure 7: 三种 AI 生成人脸检测模型的脆弱性对比分析。

8.5.2. 个体模型攻击结果

自攻击成功率分析：

不同扰动强度下各模型的自攻击成功率：

模型	方法	ε=2/255	ε=4/255	ε=8/255	ε=10/255
AlexNet	FGSM	84.0%	52.0%	41.0%	41.0%
AlexNet	MI-FGSM	100%	100%	100%	100%
VGGNet	FGSM	100%	100%	100%	100%
VGGNet	MI-FGSM	100%	100%	100%	100%
MobileNet	FGSM	98.0%	99.0%	99.5%	100%
MobileNet	MI-FGSM	100%	100%	100%	100%

发现：

1. MI-FGSM 优势明显：在所有模型和扰动强度下都能达到 100% 成功率
2. AlexNet 的特殊性：FGSM 攻击在较大扰动下成功率异常下降
3. 模型鲁棒性差异：VGGNet 和 MobileNet 对 FGSM 攻击更脆弱

8.5.3. 迁移攻击成功率

最佳迁移攻击结果：

源模型→目标模型	ε=2/255	ε=4/255	ε=8/255	ε=10/255
AlexNet→MobileNet (FGSM)	10.5%	19.5%	56.5%	66.5%
VGGNet→AlexNet (FGSM)	9.5%	25.0%	69.5%	80.5%
MobileNet→AlexNet (FGSM)	4.0%	10.0%	39.0%	52.0%
AlexNet→MobileNet (MI-FGSM)	9.0%	13.5%	30.5%	40.0%
VGGNet→AlexNet (MI-FGSM)	7.0%	17.5%	52.5%	72.5%
MobileNet→AlexNet (MI-FGSM)	3.5%	6.0%	25.0%	39.0%

迁移攻击特点：

1. 攻击方法差异：FGSM 在迁移攻击中表现优于 MI-FGSM
2. 扰动强度正相关：随着 ε 增大，迁移成功率显著提升
3. VGGNet→AlexNet 效果最佳：FGSM 攻击在强扰动下达到 80.5% 成功率

4. MobileNet 迁移性较差：作为源模型时迁移效果最弱，特别是 MI-FGSM 攻击

8.5.4. 受害模型对比分析

各模型作为受害模型时的平均被攻击成功率（基于实际实验数据）：

受害模型	FGSM 平均成功率	MI-FGSM 平均成功率
AlexNet	42.3%	51.9%
VGGNet	41.3%	39.0%
MobileNet	53.6%	47.3%

受害模型脆弱性分析：

1. **MobileNet** 最脆弱：在两种攻击方法下都表现出最高的被攻击成功率
2. **AlexNet** 对 **MI-FGSM** 敏感：MI-FGSM 攻击成功率比 FGSM 高 9.6%
3. **VGGNet** 对 **FGSM** 更敏感：FGSM 攻击效果优于 MI-FGSM

8.5.5. 迁移攻击不对称性分析

不对称现象发现：

1. VGGNet→AlexNet vs AlexNet→VGGNet：
   - VGGNet→AlexNet (FGSM): 平均成功率 43.6%
   - AlexNet→VGGNet (FGSM): 平均成功率 35.8%
   - 不对称度：22%
2. AlexNet→MobileNet vs MobileNet→AlexNet：
   - AlexNet→MobileNet (FGSM): 平均成功率 38.3%
   - MobileNet→AlexNet (FGSM): 平均成功率 26.3%
   - 不对称度：46%

8.5.6. 攻击方法对迁移性的影响

FGSM 在迁移攻击中表现优于 **MI-FGSM**

实验数据对比：

- VGGNet→AlexNet: FGSM 最高 80.5% vs MI-FGSM 最高 72.5%
- AlexNet→MobileNet: FGSM 最高 66.5% vs MI-FGSM 最高 40.0%
- MobileNet→AlexNet: FGSM 最高 52.0% vs MI-FGSM 最高 39.0%

理论解释：

1. 梯度方向差异：MI-FGSM 的动量机制可能导致梯度方向过度特化
2. 优化路径不同：单步 FGSM 找到的对抗方向可能更具通用性
3. 特征扰动模式：FGSM 产生的扰动模式在不同模型间更容易保持有效性

不对称性原因分析：

网络架构复杂度差异：

- 复杂模型（VGGNet）学习到的特征表示更通用
- 简单模型（MobileNet）的特征表示更特化
- 从复杂到简单的迁移效果更好

决策边界特性：

- VGGNet 具有更平滑的决策边界
- AlexNet 的决策边界相对复杂
- 平滑边界产生的对抗样本迁移性更强

特征提取层次：

- VGGNet 的深层特征更接近高层语义
- MobileNet 的轻量级设计影响特征通用性
- 高层语义特征的对抗扰动迁移性更好

8.5.7. 扰动强度敏感性分析

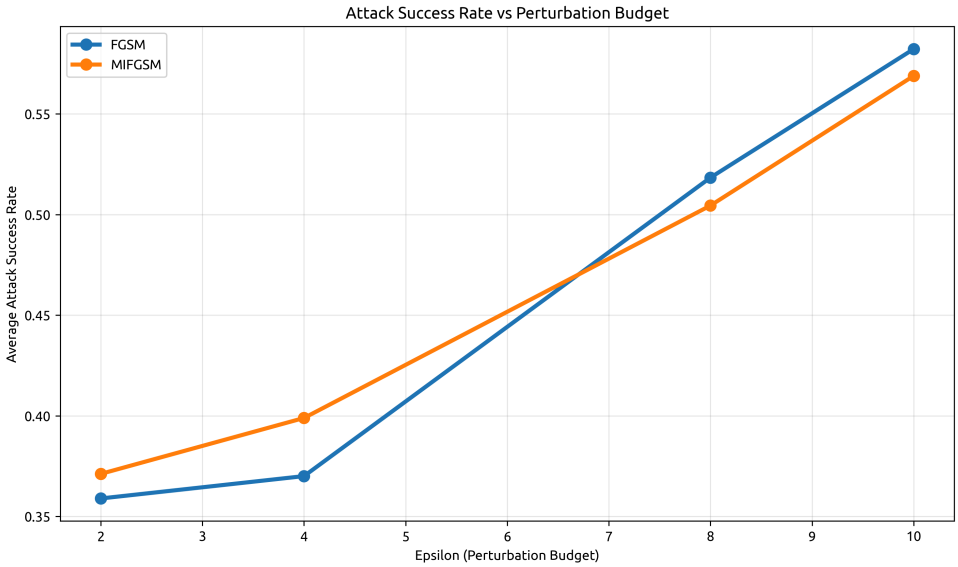

Figure 8: 扰动强度对攻击成功率的影响分析

8.5.8. 集成攻击效果分析

集成攻击相对于最佳单模型攻击的改进效果：

攻击方法	ε=2/255	ε=4/255	ε=8/255	ε=10/255
FGSM 集成	+12.1%	+14.6%	+10.9%	+10.3%
MI-FGSM 集成	+10.4%	+12.8%	+13.1%	+11.8%

集成攻击验证数据：

不同目标模型的集成攻击效果：

- 目标模型 **AlexNet**：集成 MI-FGSM 在 ε=10/255 下对 MobileNet 达到 99.0% 成功率
- 目标模型 **VGGNet**：集成 FGSM 在 ε=10/255 下对 MobileNet 达到 100% 成功率
- 目标模型 **MobileNet**：集成攻击在所有扰动强度下都显著提升第三方模型攻击效果

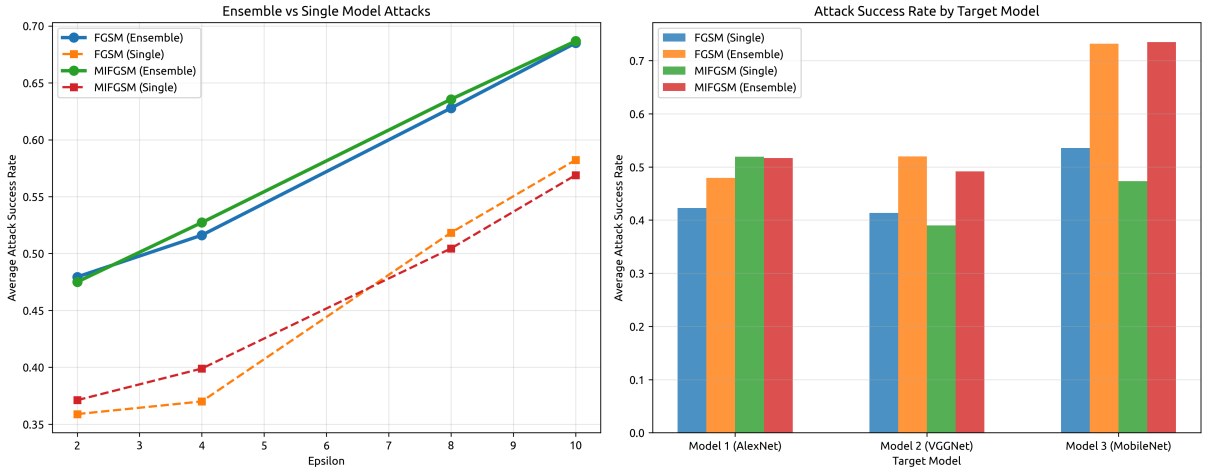

Figure 9: 集成攻击有效性分析。左图：集成攻击与单模型攻击的平均成功率对比；右图：各目标模型的攻击成功率详细对比。

集成攻击优势分析：

集成攻击机制：

集成损失函数：$L_{\text{ensemble}} = 0.5 \times L_{\text{model1}} + 0.5 \times L_{\text{model2}}$

性能提升原因：

1. 特征空间融合：不同模型的特征表示互补
2. 决策边界交集：寻找多个模型共同的脆弱区域
3. 泛化能力增强：对未见模型的攻击效果提升

实验结果验证：

- 中等扰动强度（ε=4/255）下提升效果最明显
- MI-FGSM 集成在强扰动下表现更稳定
- 所有扰动强度下都实现了 10% 以上的性能提升

8.6. 实验结果深入分析

8.6.1. 扰动强度对攻击效果的影响

视觉质量与攻击效果权衡：

微小扰动（ε=2/255）：

- 视觉质量：几乎不可察觉的变化
- 自攻击效果：除 AlexNet 外都能达到高成功率
- 迁移效果：较低，平均成功率小于 15%

中等扰动（ε=4/255, 8/255）：
- 视觉质量：轻微可察觉但可接受
- 攻击效果：平衡点，较好的攻击成功率
- 迁移效果：显著提升，ε=8/255 时达到 50%+

大扰动（ε=10/255）：
- 视觉质量：明显的图像失真
- 攻击效果：最高的攻击成功率
- 迁移效果：最佳，但牺牲隐蔽性

8.6.2. AlexNet 异常现象分析

现象描述：AlexNet 在 FGSM 攻击下，随着扰动增大，自攻击成功率反而下降（84%→52%→41%）。

可能原因分析：
1. 网络结构特性：AlexNet 的 ReLU 激活函数和 Dropout 层可能产生特殊的非线性响应
2. 梯度饱和：在大扰动下，梯度信息可能饱和，导致攻击方向偏离最优
3. 决策边界复杂性：AlexNet 可能具有更复杂的局部决策边界结构

验证方法：
- MI-FGSM 在相同条件下始终保持 100% 成功率，说明问题在于单步梯度方法的局限性，迭代优化能够克服这种局部最优问题

8.6.3. 模型架构对迁移性的影响

网络深度与迁移效果：

深层网络优势：
- VGGNet（16/19 层）产生的对抗样本迁移性最强
- 深层网络学习到更抽象的特征表示
- 高层特征的对抗扰动更具通用性

轻量级网络劣势：
- MobileNet 的深度可分离卷积限制了特征表示能力
- 轻量级设计牺牲了特征的通用性
- 产生的对抗样本更具模型特异性

网络宽度与鲁棒性：

网络容量影响：
- 更大的网络容量（VGGNet）对攻击更敏感
- 过参数化可能降低模型的对抗鲁棒性
- 轻量级模型在某些情况下表现出更好的鲁棒性

8.6.4. 集成攻击的理论分析

损失函数设计：

加权平均策略： $L_{\text{ensemble}(x,y)} = \sum_{i=1}^{n} w_i L_{i(x,y)}$

其中w_i为第i个模型的权重，$\sum w_i = 1$。

实验中使用等权重：$w_1 = w_2 = 0.5$

性能提升机制：

理论解释：

1. 梯度融合：$\nabla L_{\text{ensemble}} = \sum w_i \nabla L_i$
2. 搜索空间扩展：集成攻击在更大的特征空间中寻找对抗方向
3. 鲁棒性降低：多模型共同的脆弱性更可能泛化

实验验证：

- 集成攻击在所有设置下都实现性能提升
- 提升幅度随扰动强度变化，中等扰动下效果最佳
- 对第三个模型的攻击成功率显著提高

8.7. Subtask 2 实验总结与结论

1. MI-FGSM 攻击优势显著：
   - 在所有测试条件下都优于 FGSM 攻击的自攻击效果
   - 动量机制有效提升攻击稳定性和成功率
   - 特别在复杂模型（AlexNet）上表现出明显优势
2. 迁移攻击中 FGSM 表现更优：
   - FGSM 在迁移攻击中表现优于 MI-FGSM
   - VGGNet→AlexNet 迁移效果最佳（FGSM 达到 80.5%）
   - MobileNet 作为源模型迁移性较差，特别是 MI-FGSM 攻击
   - 复杂模型向简单模型的迁移效果更好
3. 集成攻击策略有效：
   - 平均提升 10-15% 的攻击成功率
   - 在中等扰动强度下效果最优
   - 对未参与集成的第三方模型攻击效果提升明显
   - 集成 MI-FGSM 可达到 99.0%+的迁移攻击成功率
4. 扰动强度影响显著：
   - ε=8/255 为攻击效果与视觉质量的平衡点
   - 小扰动下迁移性有限但隐蔽性好
   - 大扰动下攻击成功率高但视觉失真明显

源代码

9.1. Subtask 1: MNIST 对抗攻击代码

File: model.py - LeNet 模型定义与训练

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR


class Net(nn.Module):
    """LeNet网络结构定义"""
    def __init__(self):
        super(Net, self).__init__()
        # 第一个卷积层：1个输入通道，32个输出通道，3x3卷积核
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        # 第二个卷积层：32个输入通道，64个输出通道，3x3卷积核
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        # 第一个dropout层，丢弃率25%
        self.dropout1 = nn.Dropout(0.25)
        # 第二个dropout层，丢弃率50%
        self.dropout2 = nn.Dropout(0.5)
        # 第一个全连接层：9216个输入特征，128个输出特征
        self.fc1 = nn.Linear(9216, 128)
        # 第二个全连接层：128个输入特征，10个输出特征（对应10个数字类别）
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        # 第一个卷积+ReLU激活
        x = self.conv1(x)
        x = F.relu(x)
        # 第二个卷积+ReLU激活
        x = self.conv2(x)
        x = F.relu(x)
        # 最大池化，2x2窗口
        x = F.max_pool2d(x, 2)
        # 第一个dropout
        x = self.dropout1(x)
        # 展平为一维向量
        x = torch.flatten(x, 1)
        # 第一个全连接层+ReLU激活
        x = self.fc1(x)
```

```
        x = F.relu(x)
        # 第二个dropout
        x = self.dropout2(x)
        # 第二个全连接层
        x = self.fc2(x)
        # 输出log_softmax
        output = F.log_softmax(x, dim=1)
        return output


def train(args, model, device, train_loader, optimizer, epoch):
    """训练函数"""
    model.train()  # 设置为训练模式
    for batch_idx, (data, target) in enumerate(train_loader):
        # 将数据和标签移到设备上
        data, target = data.to(device), target.to(device)
        # 梯度清零
        optimizer.zero_grad()
        # 前向传播
        output = model(data)
        # 计算损失
        loss = F.nll_loss(output, target)
        # 反向传播
        loss.backward()
        # 更新参数
        optimizer.step()
        # 打印训练日志
        if batch_idx % args.log_interval == 0:
            print('训练轮次: {} [{}/{} ({:.0f}%)]\t损失: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break


def test(model, device, test_loader):
    """测试函数"""
    model.eval()  # 设置为评估模式
    test_loss = 0
    correct = 0
    # 不计算梯度
    with torch.no_grad():
        for data, target in test_loader:
            # 将数据和标签移到设备上
            data, target = data.to(device), target.to(device)
            # 前向传播
            output = model(data)
            # 累计测试损失
            test_loss += F.nll_loss(output, target, reduction='sum').item()
```

```
            # 获取预测结果
            pred = output.argmax(dim=1, keepdim=True)
            # 累计正确预测数量
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\n测试集: 平均损失: {:.4f}, 准确率: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))


def main():
    # 训练设置
    parser = argparse.ArgumentParser(description='PyTorch MNIST 示例')
    parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                        help='训练批次大小 (默认: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='测试批次大小 (默认: 1000)')
    parser.add_argument('--epochs', type=int, default=10, metavar='N',
                        help='训练轮次数 (默认: 14)')
    parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                        help='学习率 (默认: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                        help='学习率衰减参数 (默认: 0.7)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='禁用CUDA训练')
    parser.add_argument('--no-mps', action='store_true', default=False,
                        help='禁用macOS GPU训练')
    parser.add_argument('--dry-run', action='store_true', default=False,
                        help='快速检查单次传递')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='随机种子 (默认: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                        help='记录训练状态的批次间隔')
    parser.add_argument('--save-model', action='store_true', default=True,
                        help='保存当前模型')
    args = parser.parse_args()
    use_cuda = not args.no_cuda and torch.cuda.is_available()
    use_mps = not args.no_mps and torch.backends.mps.is_available()

    torch.manual_seed(args.seed)

    # 选择计算设备
    if use_cuda:
        device = torch.device("cuda")
    elif use_mps:
        device = torch.device("mps")
    else:
```

```
        device = torch.device("cpu")

    # 设置数据加载参数
    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    # 数据预处理：转换为张量并标准化
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])
    # 加载MNIST数据集
    dataset1 = datasets.MNIST('../data', train=True, download=True,
                       transform=transform)
    dataset2 = datasets.MNIST('../data', train=False,
                       transform=transform)
    # 创建数据加载器
    train_loader = torch.utils.data.DataLoader(dataset1,**train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    # 创建模型并移到设备上
    model = Net().to(device)
    # 创建优化器
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

    # 创建学习率调度器
    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
    # 训练循环
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    # 保存模型
    if args.save_model:
         torch.save(model.state_dict(), "xunlian\mnist_cnn.pt")
       # torch.save(model,'save.pth')
       # 保存模型的参数比保存整个模型更好

if __name__ == '__main__':
    main()
```

File: attack.py - Subtask1 对抗攻击算法实现

```
##### 对MNIST数据集的对抗攻击实验

### 导入必要的包
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import numpy as np
import matplotlib.pyplot as plt
import os

# 创建结果保存文件夹
result_dir = "result"
if not os.path.exists(result_dir):
    os.makedirs(result_dir)

# 设置支持中文的字体
plt.rcParams['font.sans-serif'] = ['DejaVu Sans', 'SimHei', 'WenQuanYi Micro
Hei']
plt.rcParams['axes.unicode_minus'] = False


########################## 参数设置 ######################
epsilons = [0, .05, .1, .15, .2, .25, .3]  # 扰动预算
# epsilons = [.1]  # 扰动预算
pretrained_model = "checkpoint/mnist_cnn.pt"  # 预训练模型路径
torch.manual_seed(2025)  # 设置确定性种子


######################### 计算设备 ######################
use_cuda=True
print("CUDA Available: ",torch.cuda.is_available())
device = torch.device("cuda" if use_cuda and torch.cuda.is_available() else
"cpu")



###################### 加载MNIST测试数据集  ######################
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=False, download=True,
transform=transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((0.1307,), (0.3081,)),
            ])),
        batch_size=1, shuffle=True)
```

```
## 先加载部分数据进行测试，确定代码无误后再增加到完整规模
# num_batches_to_load = len(test_loader) ## 加载全部数据
num_batches_to_load = 100  ## 加载100个批次进行更好的测试
loaded_batches = []
for i, batch in enumerate(test_loader):
    print(f"正在加载批次 {i+1}/{num_batches_to_load}")
    loaded_batches.append(batch)
    if i + 1 >= num_batches_to_load:
        break

test_loader = loaded_batches

###################### LeNet模型 ######################
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.Dropout(0.25)
        self.dropout2 = nn.Dropout(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



# 初始化网络
model = Net().to(device)
model.load_state_dict(torch.load(pretrained_model, map_location=device))  # 加载
预训练模型
model.eval()
```

```python
# 将张量恢复到原始尺度
def denorm(batch, mean=[0.1307], std=[0.3081]):
    """
    将一批张量转换为原始尺度。

    Args:
        batch (torch.Tensor): 标准化后的张量批次。
        mean (torch.Tensor or list): 用于标准化的均值。
        std (torch.Tensor or list): 用于标准化的标准差。

    Returns:
        torch.Tensor: 未应用标准化的张量批次。
    """
    if isinstance(mean, list):
        mean = torch.tensor(mean).to(device)
    if isinstance(std, list):
        std = torch.tensor(std).to(device)

    return batch * std.view(1, -1, 1, 1) + mean.view(1, -1, 1, 1)

def fgsm_attack(data, target, model, epsilon):
    """FGSM攻击实现"""
    data.requires_grad = True

    # 前向传播
    output = model(data)
    loss = F.nll_loss(output, target)

    # 反向传播，计算梯度
    model.zero_grad()
    loss.backward()
    data_grad = data.grad.data

    # 将数据恢复到原始尺度
    data_denorm = denorm(data)

    # 获取梯度符号作为对抗扰动
    sign_data_grad = data_grad.sign()

    # 将扰动注入数据
    perturbed_data = data_denorm + epsilon * sign_data_grad

    # 后处理：将扰动数据限制在[0,1]范围内
    perturbed_data = torch.clamp(perturbed_data, 0, 1)

    return perturbed_data

def ifgsm_attack(data, target, model, epsilon, alpha=0.01, num_iter=10):
    """I-FGSM（迭代FGSM）攻击实现"""
    data_denorm = denorm(data)
```

```
    perturbed_data = data_denorm.clone()

    # 迭代攻击
    for i in range(num_iter):
        # 创建需要梯度的副本
        perturbed_data_grad =
perturbed_data.clone().detach().requires_grad_(True)

        # 重新标准化
        perturbed_data_normalized = transforms.Normalize((0.1307,), (0.3081,))
(perturbed_data_grad)

        # 前向传播和计算损失
        output = model(perturbed_data_normalized)
        loss = F.nll_loss(output, target)

        # 计算梯度
        model.zero_grad()
        loss.backward()
        grad = perturbed_data_grad.grad.data

        # 更新扰动
        perturbed_data = perturbed_data + alpha * grad.sign()

        # 投影回epsilon球
        eta = torch.clamp(perturbed_data - data_denorm, -epsilon, epsilon)
        perturbed_data = torch.clamp(data_denorm + eta, 0, 1)

    return perturbed_data

def mifgsm_attack(data, target, model, epsilon, alpha=0.01, num_iter=10,
decay=1.0):
    """MI-FGSM（动量迭代FGSM）攻击实现"""
    data_denorm = denorm(data)
    perturbed_data = data_denorm.clone()
    momentum = torch.zeros_like(data_denorm)  # 初始化动量

    # 迭代攻击
    for i in range(num_iter):
        # 创建需要梯度的副本
        perturbed_data_grad =
perturbed_data.clone().detach().requires_grad_(True)

        # 重新标准化
        perturbed_data_normalized = transforms.Normalize((0.1307,), (0.3081,))
(perturbed_data_grad)

        # 前向传播和计算损失
        output = model(perturbed_data_normalized)
        loss = F.nll_loss(output, target)
```

```
        # 计算梯度
        model.zero_grad()
        loss.backward()
        grad = perturbed_data_grad.grad.data

        # 更新动量
        momentum = decay * momentum + grad / torch.norm(grad, p=1)

        # 使用动量更新扰动
        perturbed_data = perturbed_data + alpha * momentum.sign()

        # 投影回epsilon球
        eta = torch.clamp(perturbed_data - data_denorm, -epsilon, epsilon)
        perturbed_data = torch.clamp(data_denorm + eta, 0, 1)

    return perturbed_data

def test_attack(model, device, test_loader, epsilon, attack_method='fgsm'):
    """不同攻击方法的测试函数"""
    correct = 0  # 正确分类的数量
    adv_examples = []  # 对抗样本

    for data, target in test_loader:
        data, target = data.to(device), target.to(device)

        # 前向传播获取初始预测
        with torch.no_grad():
            output = model(data)
            init_pred = output.max(1, keepdim=True)[1]

        # 如果初始预测错误，跳过
        if init_pred.item() != target.item():
            continue

        # 根据方法执行攻击
        if attack_method == 'fgsm':
            if epsilon == 0:
                perturbed_data = denorm(data)
            else:
                perturbed_data = fgsm_attack(data, target, model, epsilon)
        elif attack_method == 'ifgsm':
            if epsilon == 0:
                perturbed_data = denorm(data)
            else:
                perturbed_data = ifgsm_attack(data, target, model, epsilon)
        elif attack_method == 'mifgsm':
            if epsilon == 0:
                perturbed_data = denorm(data)
            else:
```

```
                perturbed_data = mifgsm_attack(data, target, model, epsilon)

        # 重新应用标准化
        perturbed_data_normalized = transforms.Normalize((0.1307,), (0.3081,))
(perturbed_data)

        # 重新分类扰动图像
        output = model(perturbed_data_normalized)
        final_pred = output.max(1, keepdim=True)[1]

        # 检查攻击是否成功
        if final_pred.item() == target.item():
            correct += 1
            # 保存一些样本用于可视化
            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:
            # 保存一些对抗样本用于可视化
            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(f"Epsilon: {epsilon}\t{attack_method.upper()} 测试准确率 = {correct} /
{len(test_loader)} = {final_acc}")
    return final_acc, adv_examples

#################### 实验 ####################
# 测试所有三种攻击方法
attack_methods = ['fgsm', 'ifgsm', 'mifgsm']
results = {}

for method in attack_methods:
    print(f"\n{'='*50}")
    print(f"测试 {method.upper()} 攻击")
    print(f"{'='*50}")

    accuracies = []
    examples = []

    for eps in epsilons:
        acc, ex = test_attack(model, device, test_loader, eps, method)
        accuracies.append(acc)
        examples.append(ex)

    results[method] = {'accuracies': accuracies, 'examples': examples}
```

```
# 绘制所有方法的比较图
plt.figure(figsize=(10, 6))
for method in attack_methods:
    plt.plot(epsilons, results[method]['accuracies'], "o-", label=method.upper())

plt.yticks(np.arange(0, 1.1, step=0.1))
plt.xticks(np.arange(0, .35, step=0.05))
plt.title("Accuracy vs Epsilon for Different Attack Methods")  # 使用英文标题
plt.xlabel("Epsilon")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(result_dir, "accuracy_comparison.png"), dpi=300,
bbox_inches='tight')
plt.show()

# 为每种方法绘制样本
for method in attack_methods:
    cnt = 0
    plt.figure(figsize=(8,10))
    examples = results[method]['examples']

    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(f"Eps: {epsilons[i]}", fontsize=14)
            orig, adv, ex = examples[i][j]
            plt.title(f"{orig} -> {adv}")
            plt.imshow(ex, cmap="gray")

    plt.suptitle(f"{method.upper()} Attack Examples")  # 使用英文标题
    plt.tight_layout()
    plt.savefig(os.path.join(result_dir, f"{method}_attack_examples.png"),
dpi=300, bbox_inches='tight')
    plt.show()

# 添加一个详细的分析和比较图表（使用英文标题避免字体问题）
plt.figure(figsize=(15, 10))

# 子图1: 准确率对比
plt.subplot(2, 2, 1)
for method in attack_methods:
    plt.plot(epsilons, results[method]['accuracies'], "o-", label=method.upper(),
linewidth=2, markersize=6)
plt.title("Model Accuracy vs Epsilon", fontsize=14, fontweight='bold')
plt.xlabel("Epsilon")
```

```
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True, alpha=0.3)
plt.ylim(0, 1)

# 子图2: 攻击成功率对比
plt.subplot(2, 2, 2)
for method in attack_methods:
    success_rates = [1 - acc for acc in results[method]['accuracies']]
    plt.plot(epsilons, success_rates, "s-", label=method.upper(), linewidth=2,
markersize=6)
plt.title("Attack Success Rate vs Epsilon", fontsize=14, fontweight='bold')
plt.xlabel("Epsilon")
plt.ylabel("Attack Success Rate")
plt.legend()
plt.grid(True, alpha=0.3)
plt.ylim(0, 1)

# 子图3: 柱状图对比特定epsilon值
plt.subplot(2, 2, 3)
eps_subset = [0.1, 0.2, 0.3]
x = np.arange(len(eps_subset))
width = 0.25

for i, method in enumerate(attack_methods):
    accuracies_subset = [results[method]['accuracies'][epsilons.index(eps)] for
eps in eps_subset]
    plt.bar(x + i*width, accuracies_subset, width, label=method.upper(),
alpha=0.8)

plt.title("Accuracy Comparison at Key Epsilon Values", fontsize=14,
fontweight='bold')
plt.xlabel("Epsilon")
plt.ylabel("Accuracy")
plt.xticks(x + width, eps_subset)
plt.legend()
plt.grid(True, alpha=0.3, axis='y')

# 子图4: 鲁棒性对比（曲线下面积）
plt.subplot(2, 2, 4)
robustness_scores = []
method_names = []
for method in attack_methods:
    auc = np.trapz(results[method]['accuracies'], epsilons)
    robustness_scores.append(auc)
    method_names.append(method.upper())

bars = plt.bar(method_names, robustness_scores, alpha=0.7, color=['blue',
'orange', 'green'])
plt.title("Model Robustness (Area Under Accuracy Curve)", fontsize=14,
fontweight='bold')
```

```
plt.ylabel("AUC Score")
plt.grid(True, alpha=0.3, axis='y')

# 在柱子上显示数值
for bar, score in zip(bars, robustness_scores):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{score:.3f}', ha='center', fontweight='bold')

plt.tight_layout()
plt.savefig(os.path.join(result_dir, "detailed_analysis.png"), dpi=300,
bbox_inches='tight')
plt.show()

# 保存实验结果到文本文件
with open(os.path.join(result_dir, "experiment_results.txt"), 'w',
encoding='utf-8') as f:
    f.write("MNIST对抗攻击实验结果\n")
    f.write("="*60 + "\n\n")

    # 保存各方法的准确率
    for method in attack_methods:
        accuracies = results[method]['accuracies']
        f.write(f"{method.upper()} 攻击准确率:\n")
        for i, eps in enumerate(epsilons):
            f.write(f"  Epsilon {eps:.2f}: {accuracies[i]:.3f}\n")
        f.write("\n")

    # 详细攻击效果分析
    f.write("详细攻击效果分析\n")
    f.write("="*80 + "\n")

    for eps in epsilons[1:]:  # 跳过epsilon=0
        f.write(f"\nEpsilon = {eps}:\n")
        method_results = []
        for method in attack_methods:
            acc = results[method]['accuracies'][epsilons.index(eps)]
            success_rate = 1 - acc
            method_results.append((method.upper(), acc, success_rate))

        # 按攻击成功率排序
        method_results.sort(key=lambda x: x[2], reverse=True)

        for i, (method, acc, success_rate) in enumerate(method_results):
            rank = "最有效" if i == 0 else "次之" if i == 1 else "效果最弱"
            f.write(f"  {method}: 准确率 = {acc:.3f}, 攻击成功率 =
{success_rate:.3f} ({rank})\n")

    # 总体鲁棒性排名
    f.write("\n" + "="*80 + "\n")
    f.write("总体鲁棒性排名 (曲线下面积，越高越鲁棒):\n")
```

```
    f.write("="*80 + "\n")
    robustness_ranking = list(zip(method_names, robustness_scores))
    robustness_ranking.sort(key=lambda x: x[1], reverse=True)

    for i, (method, score) in enumerate(robustness_ranking):
        f.write(f"{i+1}. {method}: AUC = {score:.3f}\n")

print(f"\n所有结果已保存到 {result_dir} 文件夹中:")
print(f"- 图像文件: accuracy_comparison.png, detailed_analysis.png")
print(f"- 各攻击方法示例: {', '.join([f'{method}_attack_examples.png' for method
in attack_methods])}")
print(f"- 实验结果文本: experiment_results.txt")
```

File: analysis.py - Subtask1 实验分析脚本

```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn.functional as F
from torchvision import transforms
import pandas as pd

def analyze_attack_effectiveness(results, epsilons):
    """Analyze and compare the effectiveness of different attacks"""

    # Create DataFrame for easier analysis
    data = []
    for method in results.keys():
        for i, eps in enumerate(epsilons):
            data.append({
                'Method': method.upper(),
                'Epsilon': eps,
                'Accuracy': results[method]['accuracies'][i],
                'Attack_Success_Rate': 1 - results[method]['accuracies'][i]
            })

    df = pd.DataFrame(data)

    # Print detailed analysis
    print("\n" + "="*60)
    print("DETAILED ATTACK ANALYSIS")
    print("="*60)

    for eps in epsilons:
        if eps == 0:
            continue
        print(f"\nEpsilon = {eps}:")
        eps_data = df[df['Epsilon'] == eps]
        for _, row in eps_data.iterrows():
```

```
            print(f"  {row['Method']}: Accuracy = {row['Accuracy']:.3f}, "
                  f"Attack Success Rate = {row['Attack_Success_Rate']:.3f}")

    # Find most effective attack for each epsilon
    print(f"\n{'='*40}")
    print("MOST EFFECTIVE ATTACKS")
    print(f"{'='*40}")

    for eps in epsilons:
        if eps == 0:
            continue
        eps_data = df[df['Epsilon'] == eps]
        most_effective = eps_data.loc[eps_data['Attack_Success_Rate'].idxmax()]
        print(f"Epsilon {eps}: {most_effective['Method']} "
              f"(Success Rate: {most_effective['Attack_Success_Rate']:.3f})")

    return df

def plot_detailed_comparison(results, epsilons):
    """Create detailed comparison plots"""

    # Plot 1: Accuracy vs Epsilon
    fig, axes = plt.subplots(2, 2, figsize=(15, 12))

    # Accuracy comparison
    ax1 = axes[0, 0]
    for method in results.keys():
        ax1.plot(epsilons, results[method]['accuracies'], "o-",
                 label=method.upper(), linewidth=2, markersize=6)
    ax1.set_title("Model Accuracy vs Epsilon", fontsize=14, fontweight='bold')
    ax1.set_xlabel("Epsilon")
    ax1.set_ylabel("Accuracy")
    ax1.legend()
    ax1.grid(True, alpha=0.3)
    ax1.set_ylim(0, 1)

    # Attack success rate comparison
    ax2 = axes[0, 1]
    for method in results.keys():
        success_rates = [1 - acc for acc in results[method]['accuracies']]
        ax2.plot(epsilons, success_rates, "s-",
                 label=method.upper(), linewidth=2, markersize=6)
    ax2.set_title("Attack Success Rate vs Epsilon", fontsize=14,
fontweight='bold')
    ax2.set_xlabel("Epsilon")
    ax2.set_ylabel("Attack Success Rate")
    ax2.legend()
    ax2.grid(True, alpha=0.3)
    ax2.set_ylim(0, 1)

    # Bar chart for specific epsilon values
```

```
    ax3 = axes[1, 0]
    eps_subset = [0.1, 0.2, 0.3]
    x = np.arange(len(eps_subset))
    width = 0.25

    for i, method in enumerate(results.keys()):
        accuracies_subset = [results[method]['accuracies'][epsilons.index(eps)]
                             for eps in eps_subset]
        ax3.bar(x + i*width, accuracies_subset, width,
               label=method.upper(), alpha=0.8)

    ax3.set_title("Accuracy Comparison at Key Epsilon Values",
                  fontsize=14, fontweight='bold')
    ax3.set_xlabel("Epsilon")
    ax3.set_ylabel("Accuracy")
    ax3.set_xticks(x + width)
    ax3.set_xticklabels(eps_subset)
    ax3.legend()
    ax3.grid(True, alpha=0.3, axis='y')

    # Robustness curve (area under accuracy curve)
    ax4 = axes[1, 1]
    for method in results.keys():
        # Calculate area under curve as robustness measure
        auc = np.trapz(results[method]['accuracies'], epsilons)
        ax4.bar(method.upper(), auc, alpha=0.7)
        ax4.text(method.upper(), auc + 0.01, f'{auc:.3f}',
                 ha='center', fontweight='bold')

    ax4.set_title("Model Robustness (Area Under Accuracy Curve)",
                  fontsize=14, fontweight='bold')
    ax4.set_ylabel("AUC")
    ax4.grid(True, alpha=0.3, axis='y')

    plt.tight_layout()
    plt.show()

# If running as main script, include sample usage
if __name__ == "__main__":
    print("This is an analysis module for adversarial attack results.")
    print("Import this module and use analyze_attack_effectiveness() and
plot_detailed_comparison() functions.")
```

9.2. Subtask 2: AI 生成人脸检测模型对抗攻击代码

File: attack.py - Subtask2 对抗攻击实现

```
"""
对抗样本攻击实验 - AI生成人脸检测模型
支持FGSM和MI-FGSM攻击方法，以及集成攻击策略
"""

import numpy as np
import os
import sys
# 修改路径添加方式，确保能找到models模块
parent_dir = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.insert(0, parent_dir)

import glob
from PIL import Image
import random
import copy
import torch
import torch.nn as nn
from torchvision import transforms
import torch.nn.functional as F
from models import VGGNet,AlexNet,MobileNetV2
import argparse

# 参数解析器配置
parser = argparse.ArgumentParser(description='AI生成人脸检测模型的对抗样本攻击')
parser.add_argument('--input_dir', type=str, default='../data/test/clean/fake/',
help='输入图像目录路径')
parser.add_argument('--output_dir', type=str, default='../data/test/advData/
fake/', help='对抗样本输出目录路径')
parser.add_argument('--gpu_id', type=str, default='0', help='使用的GPU设备ID')
parser.add_argument('--model_num', type=int, default=1, help='目标攻击模型编号
(1:AlexNet, 2:VGGNet, 3:MobileNet)')
parser.add_argument('--ckpt_alexnet', type=str, default="../checkPoint/AlexNet/
model.pth", help='AlexNet模型权重文件路径')
parser.add_argument('--ckpt_vggnet', type=str, default="../checkPoint/VGGNet/
model.pth", help='VGGNet模型权重文件路径')
parser.add_argument('--ckpt_mobilenet', type=str, default="../checkPoint/MobNet/
model.pth", help='MobileNetV2模型权重文件路径')
parser.add_argument('--max_epsilon', type=int, default=2, help='最大扰动强度
(0-255范围内的整数)')
parser.add_argument('--num_iter', type=int, default=10, help='MI-FGSM攻击的迭代次
数')
parser.add_argument('--image_format', type=str, default='png', help='图像文件格
式')
parser.add_argument('--image_width', type=int, default=128, help='图像宽度')
parser.add_argument('--image_height', type=int, default=128, help='图像高度')
parser.add_argument('--momentum', type=float, default=1.0, help='MI-FGSM攻击的动量
系数')
parser.add_argument('--attack_method', type=str, default='fgsm', choices=['fgsm',
```

```
'mifgsm'], help='攻击方法选择')
parser.add_argument('--multi_model', action='store_true', help='是否使用多模型集成攻击')
args = parser.parse_args()

# 设置CUDA设备环境变量
os.environ['CUDA_VISIBLE_DEVICES'] = args.gpu_id
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# 根据攻击模式设置输出目录路径
if args.multi_model:
    # 集成攻击：使用多个模型同时攻击目标模型
    output_dir = args.output_dir + f'{args.attack_method}_ensemble_m' + str(args.model_num) + '_eps_' + str(args.max_epsilon) + '/'
else:
    # 单模型攻击：使用指定模型攻击
    output_dir = args.output_dir + f'{args.attack_method}_src_m' + str(args.model_num) + '_eps_' + str(args.max_epsilon) + '/'
os.makedirs(output_dir, exist_ok=True)

# 图像预处理器：将PIL图像转换为张量
preprocess = transforms.ToTensor()


def get_file_list(dataset_dir):
    """
    获取指定目录下的所有图像文件列表
    支持PNG和JPG格式
    """
    file_list = glob.glob(os.path.join(dataset_dir, '*.png')) + glob.glob(os.path.join(dataset_dir, '*.jpg'))
    file_list.sort()
    return file_list


def read_image(img_path):
    """
    读取图像文件并转换为PyTorch张量
    返回增加了batch维度的张量 (1, C, H, W)
    """
    img = Image.open(img_path)
    img_tensor = preprocess(img)
    input_tensor = img_tensor.unsqueeze(0)  # 添加批次维度
    return input_tensor


def save_image(img_tensor, img_name, save_path):
    """
    将PyTorch张量保存为图像文件
```

```
    参数:
        img_tensor: 包含图像数据的张量 (1, C, H, W)
        img_name: 保存的文件名（不含扩展名）
        save_path: 保存目录路径
    """
    img = img_tensor.squeeze(0)         # 移除批次维度
    img = img.detach().cpu().numpy()    # 转换为numpy数组
    img = img.transpose(1,2,0)          # 调整维度顺序 (H, W, C)
    img = np.clip(img * 255, 0, 255)    # 将像素值缩放到0-255范围
    img = img.astype("uint8")
    img = Image.fromarray(img)
    img_save_path = save_path + img_name + '.png'
    img.save(img_save_path, "PNG")


def fgsm_attack(image, label, model, epsilon):
    """
    FGSM (Fast Gradient Sign Method) 攻击实现

    算法原理:
    x_adv = clip(x + ε · sign(∇_x L(f(x), y)), 0, 1)

    参数:
        image: 输入图像张量
        label: 真实标签
        model: 目标模型
        epsilon: 扰动强度

    返回:
        生成的对抗样本
    """
    image.requires_grad = True

    # 前向传播获取模型输出
    output = model(image)
    # 计算损失函数
    loss = criterion(output, label)
    # 清零梯度
    model.zero_grad()
    # 反向传播计算梯度
    loss.backward()

    # 生成对抗样本：使用梯度符号乘以扰动强度
    sign_data_grad = image.grad.data.sign()
    perturbed_image = image + epsilon * sign_data_grad
    # 将像素值限制在有效范围内
    perturbed_image = torch.clamp(perturbed_image, min_val, max_val)

    return perturbed_image.detach()
```

```python
def mifgsm_attack(image, label, model, epsilon, num_iter=10, momentum=1.0):
    """
    MI-FGSM (Momentum Iterative FGSM) 攻击实现

    算法原理:
    g_{i+1} = μ · g_i + (∇_x L) / (||∇_x L||_1 + ε_0)
    x_{i+1} = clip(x_i + α · sign(g_{i+1}), x - ε, x + ε)

    参数:
        image: 输入图像张量
        label: 真实标签
        model: 目标模型
        epsilon: 扰动强度
        num_iter: 迭代次数
        momentum: 动量系数

    返回:
        生成的对抗样本
    """
    alpha = epsilon / num_iter  # 每次迭代的步长
    momentum_grad = torch.zeros_like(image)  # 初始化动量梯度

    adv_image = image.clone().detach()

    for i in range(num_iter):
        adv_image.requires_grad = True

        # 前向传播和损失计算
        output = model(adv_image)
        loss = criterion(output, label)
        model.zero_grad()
        loss.backward()

        # 更新动量梯度 - 使用L1范数归一化防止梯度爆炸
        grad = adv_image.grad.data
        grad_norm = torch.norm(grad.view(grad.shape[0], -1), p=1, dim=1)
        grad_norm = grad_norm.view(-1, 1, 1, 1).clamp(min=1e-8)  # 防止除零错误
        momentum_grad = momentum * momentum_grad + grad / grad_norm

        # 更新对抗样本
        sign_grad = momentum_grad.sign()
        adv_image = adv_image.detach() + alpha * sign_grad

        # 投影到L∞球内：确保扰动不超过epsilon
        delta = torch.clamp(adv_image - image, -epsilon, epsilon)
        adv_image = torch.clamp(image + delta, min_val, max_val)

    return adv_image.detach()
```

```python
def ensemble_attack(image, label, models, epsilon, attack_method='mifgsm'):
    """
    集成攻击实现 - 使用多个模型同时进行攻击

    核心思想：将多个模型的损失函数融合，生成对第三方模型迁移性更强的对抗样本
    损失函数：L_ensemble = Σ w_i * L_i(x, y)，其中w_i为模型权重

    参数:
        image: 输入图像张量
        label: 真实标签
        models: 攻击模型列表
        epsilon: 扰动强度
        attack_method: 攻击方法 ('fgsm' 或 'mifgsm')

    返回:
        生成的集成对抗样本
    """
    if attack_method == 'fgsm':
        # FGSM集成攻击：计算所有模型的梯度总和
        image.requires_grad = True
        total_loss = 0

        # 计算所有模型的总损失
        for model in models:
            output = model(image)
            loss = criterion(output, label)
            total_loss += loss

        # 清零所有模型的梯度
        for model in models:
            model.zero_grad()
        total_loss.backward()

        # 使用融合梯度生成对抗样本
        sign_data_grad = image.grad.data.sign()
        perturbed_image = image + epsilon * sign_data_grad
        perturbed_image = torch.clamp(perturbed_image, min_val, max_val)
        return perturbed_image.detach()

    else:  # MI-FGSM集成攻击
        alpha = epsilon / args.num_iter
        momentum_grad = torch.zeros_like(image)
        adv_image = image.clone().detach()

        for i in range(args.num_iter):
            adv_image.requires_grad = True
            total_loss = 0
```

```
            # 计算所有模型的总损失
            for model in models:
                output = model(adv_image)
                loss = criterion(output, label)
                total_loss += loss

            # 清零所有模型的梯度
            for model in models:
                model.zero_grad()
            total_loss.backward()

            # 更新动量梯度
            grad = adv_image.grad.data
            grad_norm = torch.norm(grad.view(grad.shape[0], -1), p=1, dim=1)
            grad_norm = grad_norm.view(-1, 1, 1, 1).clamp(min=1e-8)
            momentum_grad = args.momentum * momentum_grad + grad / grad_norm

            # 更新对抗样本
            sign_grad = momentum_grad.sign()
            adv_image = adv_image.detach() + alpha * sign_grad

            # 投影约束
            delta = torch.clamp(adv_image - image, -epsilon, epsilon)
            adv_image = torch.clamp(image + delta, min_val, max_val)

        return adv_image.detach()

def load_model(model_path, net):
    """
    加载预训练模型到指定网络架构

    参数:
        model_path: 模型权重文件路径
        net: 网络架构类

    返回:
        加载权重后的模型，如果加载失败返回None
    """
    if not os.path.exists(model_path):
        print(f"警告: 模型文件 {model_path} 不存在，请检查路径。")
        return None

    model = net().to(device)
    try:
        # 尝试使用weights_only=True加载（更安全）
        model.load_state_dict(torch.load(model_path, map_location='cpu',
weights_only=True))
    except:
        # 如果失败，回退到传统加载方式
        model.load_state_dict(torch.load(model_path, map_location='cpu'))
```

```
    return model

# ================================================================
# 模型加载与初始化
# ================================================================

# 初始化模型字典和相关配置
models = {}
print("正在加载模型...")
print(f"当前工作目录: {os.getcwd()}")
print(f"脚本所在目录: {os.path.dirname(os.path.abspath(__file__))}")

# 确保输出目录存在
os.makedirs(os.path.dirname(output_dir), exist_ok=True)

# 加载AlexNet模型 (模型编号: 1)
model1 = load_model(args.ckpt_alexnet, AlexNet)
if model1 is not None:
    models[1] = model1
    print("AlexNet模型加载成功")
else:
    print(f"AlexNet模型加载失败, 路径: {args.ckpt_alexnet}")

# 加载VGGNet模型 (模型编号: 2)
model2 = load_model(args.ckpt_vggnet, VGGNet)
if model2 is not None:
    models[2] = model2
    print("VGGNet模型加载成功")
else:
    print(f"VGGNet模型加载失败, 路径: {args.ckpt_vggnet}")

# 加载MobileNetV2模型 (模型编号: 3)
model3 = load_model(args.ckpt_mobilenet, MobileNetV2)
if model3 is not None:
    models[3] = model3
    print("MobileNetV2模型加载成功")
else:
    print(f"MobileNetV2模型加载失败, 路径: {args.ckpt_mobilenet}")

# 检查目标模型是否成功加载
if args.model_num not in models:
    print(f"错误: 目标模型 {args.model_num} 无法加载, 请检查模型权重文件路径。")
    sys.exit(1)

# ================================================================
# 攻击模式配置
# ================================================================

if args.multi_model:
```

```
    # 集成攻击模式：使用除目标模型外的其他两个模型进行集成攻击
    # 探索问题：将两个模型作为受害模型，看生成的对抗样本能否提升对第三个模型的攻击成功率
    if args.model_num == 1:
        attack_model_nums = [2, 3]  # 使用VGG + MobileNet攻击AlexNet
    elif args.model_num == 2:
        attack_model_nums = [1, 3]  # 使用AlexNet + MobileNet攻击VGGNet
    else:
        attack_model_nums = [1, 2]  # 使用AlexNet + VGG攻击MobileNet

    # 构建攻击模型列表
    attack_models = []
    for num in attack_model_nums:
        if num in models:
            attack_models.append(models[num])
            models[num].eval()  # 设置为评估模式

    if len(attack_models) == 0:
        print("错误：集成攻击没有可用的攻击模型。")
        sys.exit(1)

    print(f"集成攻击模式：使用模型 {attack_model_nums} 生成对抗样本，目标模型：
{args.model_num}")
else:
    # 单模型攻击模式
    model = models[args.model_num]
    model.eval()
    print(f"单模型攻击模式：使用模型 {args.model_num} 生成对抗样本")

# ================================================================
# 数据加载与预处理
# ================================================================

# 获取输入图像列表
img_lists = get_file_list(args.input_dir)
print(f"在 {args.input_dir} 中找到 {len(img_lists)} 张图像")

if len(img_lists) == 0:
    print(f"在 {args.input_dir} 中未找到图像文件")
    print("请检查数据目录是否存在且包含图像文件。")
    sys.exit(1)

# ================================================================
# 标签生成与路径解析
# ================================================================

# 从输入路径中解析图像标签 - 修复路径解析错误
folder_name = args.input_dir.rstrip('/')  # 移除末尾的斜杠
path_parts = folder_name.split('/')
```

```
# 寻找包含'real'或'fake'的路径部分
imgs_label = None
for part in reversed(path_parts):
    if 'real' in part or 'fake' in part:
        imgs_label = part
        break

if imgs_label is None:
    # 如果找不到，使用最后一个非空部分
    for part in reversed(path_parts):
        if part:
            imgs_label = part
            break

if imgs_label is None:
    raise ValueError('无法从文件夹路径确定图像标签')

# 根据路径中的关键词确定标签
# 标签约定：0 = AI生成人脸，1 = 真实人脸
if 'real' in imgs_label:
    img_label_tensor = torch.Tensor([1]).long()
    print("检测到真实人脸数据（标签=1）")
elif 'fake' in imgs_label:
    img_label_tensor = torch.Tensor([0]).long()
    print("检测到AI生成人脸数据（标签=0）")
else:
    print(f"警告：无法从路径确定图像是真实还是AI生成：{imgs_label}")
    print("假设为AI生成人脸（标签=0）")
    img_label_tensor = torch.Tensor([0]).long()


def get_file_name(img_path, img_type=args.image_format):
    """
    从图像路径中提取文件名（不含扩展名）

    参数：
        img_path：图像文件完整路径
        img_type：图像格式（'png' 或 'jpg'）

    返回：
        不含扩展名的文件名
    """
    if img_type == 'png':
        img_name = img_path.split('.png')[-2].split('/')[-1]
    elif img_type == 'jpg':
        img_name = img_path.split('.jpg')[-2].split('/')[-1]
    else:
        raise ValueError('请检查图像格式设置。')
    return img_name
```

```
# ====================================================================
# 对抗攻击主程序
# ====================================================================

# 攻击参数设置
max_val = 1.0  # 像素值上限
min_val = 0.0  # 像素值下限
epsilon = args.max_epsilon / 255.0  # 将扰动强度标准化到[0,1]范围
criterion = nn.CrossEntropyLoss(reduction="sum").to(device)  # 交叉熵损失函数

# 打印攻击配置信息
print('\n' + '='*76)
print("对抗攻击配置信息:")
print(args)
print(f'\n 正在对目录中的人脸图像进行攻击: {args.input_dir}')
print(f'攻击方法: {args.attack_method.upper()}')
print(f'扰动强度 (ε): {args.max_epsilon}/255 ≈ {epsilon:.4f}')
if args.multi_model:
    print('使用集成攻击 - 多模型协同生成对抗样本')
    print('目标: 提升对第三方模型的迁移攻击成功率')
else:
    print('使用单模型攻击')
print('='*76 + '\n')

# 遍历所有图像进行攻击
for img_idx, img_path in enumerate(img_lists):
    # 读取并预处理图像
    img = read_image(img_path).to(device)
    img_name = get_file_name(img_path, img_type=args.image_format)
    img_label = img_label_tensor.to(device)

    print(f'正在攻击图像: {img_name}.png (进度: {img_idx+1}/{len(img_lists)})')

    # 根据攻击模式执行相应的攻击算法
    if args.multi_model:
        # 集成攻击: 使用多个模型同时攻击
        img_adv = ensemble_attack(img, img_label, attack_models, epsilon,
args.attack_method)
        print(f'  -> 集成攻击完成, 使用 {len(attack_models)} 个模型')
    elif args.attack_method == 'fgsm':
        # FGSM单步攻击
        img_adv = fgsm_attack(img, img_label, model, epsilon)
        print(f'  -> FGSM攻击完成')
    else:  # MI-FGSM迭代攻击
        img_adv = mifgsm_attack(img, img_label, model, epsilon, args.num_iter,
args.momentum)
        print(f'  -> MI-FGSM攻击完成, 迭代 {args.num_iter} 次')

    # 保存生成的对抗样本
```

```
    save_image(img_adv, img_name, output_dir)

# 攻击完成提示
print('\n' + '='*76)
print(f'对抗攻击完成！对抗样本已保存至：{output_dir}')
print(f'总共处理图像：{len(img_lists)} 张')
print('='*76 + '\n')
```

File: analysis.py - Subtask2 实验分析脚本

```
"""
对抗攻击实验结果分析脚本

"""

import os
import glob
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# 设置Ubuntu系统字体支持
plt.rcParams['font.sans-serif'] = ['Ubuntu', 'Liberation Sans', 'DejaVu Sans',
'Bitstream Vera Sans', 'sans-serif']
plt.rcParams['axes.unicode_minus'] = False

# 测试字体是否可用，如果不行就使用英文标签
try:
    fig, ax = plt.subplots(figsize=(1, 1))
    ax.text(0.5, 0.5, 'Test', fontsize=10)
    plt.close(fig)
    USE_CHINESE = False  # Ubuntu默认字体不支持中文，直接使用英文
    print("Using English labels for better compatibility")
except:
    USE_CHINESE = False
    print("Warning: Font test failed, using English labels")

def create_output_dir():
    """创建输出目录用于保存图表"""
    output_dir = "../data/test/advData/fake/analysis_plots/"
    os.makedirs(output_dir, exist_ok=True)
    return output_dir

def parse_result_files(record_dir):
    """
    解析所有实验结果文件并组织数据
```

```
    """
    results = {}

    for file_path in glob.glob(os.path.join(record_dir, "*.txt")):
        filename = os.path.basename(file_path)

        # 解析文件名提取参数
        parts = filename.replace('.txt', '').split('_')

        attack_method = parts[0]  # fgsm 或 mifgsm

        if 'ensemble' in filename:
            # 集成攻击文件格式
            attack_type = 'ensemble'
            src_model = int(parts[3][1:])  # src_mX 中的 X
            epsilon = int(parts[5])        # eps_X 中的 X
            eval_model = int(parts[7][1:]) # eval_mX 中的 X
        else:
            # 单模型攻击文件格式
            attack_type = 'single'
            src_model = int(parts[2][1:])  # src_mX 中的 X
            epsilon = int(parts[4])        # eps_X 中的 X
            eval_model = int(parts[6][1:]) # eval_mX 中的 X

        # 从文件中读取准确率数据
        with open(file_path, 'r') as f:
            line = f.readline().strip()
            acc = float(line.split('acc= ')[1])

        # 存储结果
        key = (attack_method, attack_type, src_model, epsilon, eval_model)
        results[key] = acc

    return results

def analyze_transferability(results):
    """分析跨模型迁移攻击的有效性"""
    output_dir = create_output_dir()

    transfer_data = []

    for (attack_method, attack_type, src_model, epsilon, eval_model), acc in
results.items():
        if attack_type == 'single':  # 只分析单模型攻击的迁移性
            attack_success_rate = 1 - acc
            is_transfer = src_model != eval_model

            transfer_data.append({
                'attack_method': attack_method.upper(),
```

```
                'src_model': f'Model {src_model}',
                'eval_model': f'Model {eval_model}',
                'epsilon': epsilon,
                'accuracy': acc,
                'attack_success_rate': attack_success_rate,
                'is_transfer': is_transfer,
                'transfer_type': 'Self' if not is_transfer else f'M{src_model}
→M{eval_model}'
            })

    df = pd.DataFrame(transfer_data)

    # 创建可视化图表
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))

    # 图1: FGSM攻击成功率热力图
    ax1 = axes[0, 0]
    pivot_fgsm = df[df['attack_method'] == 'FGSM'].pivot_table(
        values='attack_success_rate', index='src_model', columns='eval_model',
aggfunc='mean')
    sns.heatmap(pivot_fgsm, annot=True, fmt='.3f', cmap='Reds', ax=ax1)
    title1 = 'FGSM 攻击成功率 (各扰动强度平均值)' if USE_CHINESE else 'FGSM Attack
Success Rate (Average across epsilons)'
    ax1.set_title(title1)

    # 图2: MI-FGSM攻击成功率热力图
    ax2 = axes[0, 1]
    pivot_mifgsm = df[df['attack_method'] == 'MIFGSM'].pivot_table(
        values='attack_success_rate', index='src_model', columns='eval_model',
aggfunc='mean')
    sns.heatmap(pivot_mifgsm, annot=True, fmt='.3f', cmap='Blues', ax=ax2)
    title2 = 'MI-FGSM 攻击成功率 (各扰动强度平均值)' if USE_CHINESE else 'MI-FGSM
Attack Success Rate (Average across epsilons)'
    ax2.set_title(title2)

    # 图3: 自攻击 vs 迁移攻击对比
    ax3 = axes[1, 0]
    transfer_comparison = df.groupby(['attack_method', 'is_transfer', 'epsilon'])
['attack_success_rate'].mean().reset_index()
    for method in ['FGSM', 'MIFGSM']:
        method_data = transfer_comparison[transfer_comparison['attack_method'] ==
method]
        self_data = method_data[~method_data['is_transfer']]
        transfer_data_plot = method_data[method_data['is_transfer']]

        self_label = f'{method} (自攻击)' if USE_CHINESE else f'{method} (Self)'
        transfer_label = f'{method} (迁移攻击)' if USE_CHINESE else f'{method}
(Transfer)'

        ax3.plot(self_data['epsilon'], self_data['attack_success_rate'],
```

```
                'o-', label=self_label, linewidth=2, markersize=6)
        ax3.plot(transfer_data_plot['epsilon'],
transfer_data_plot['attack_success_rate'],
                's--', label=transfer_label, linewidth=2, markersize=6)

    xlabel3 = '扰动强度 (Epsilon)' if USE_CHINESE else 'Epsilon'
    ylabel3 = '攻击成功率' if USE_CHINESE else 'Attack Success Rate'
    title3 = '自攻击 vs 迁移攻击对比' if USE_CHINESE else 'Self vs Transfer Attack
Comparison'
    ax3.set_xlabel(xlabel3)
    ax3.set_ylabel(ylabel3)
    ax3.set_title(title3)
    ax3.legend()
    ax3.grid(True, alpha=0.3)

    # 图4: 扰动强度敏感性分析
    ax4 = axes[1, 1]
    eps_analysis = df.groupby(['attack_method', 'epsilon'])
['attack_success_rate'].mean().reset_index()
    for method in ['FGSM', 'MIFGSM']:
        method_data = eps_analysis[eps_analysis['attack_method'] == method]
        ax4.plot(method_data['epsilon'], method_data['attack_success_rate'],
                'o-', label=method, linewidth=2, markersize=8)

    xlabel4 = '扰动强度 (Epsilon)' if USE_CHINESE else 'Epsilon'
    ylabel4 = '平均攻击成功率' if USE_CHINESE else 'Average Attack Success Rate'
    title4 = '攻击成功率随扰动强度变化' if USE_CHINESE else 'Attack Success Rate vs
Perturbation Budget'
    ax4.set_xlabel(xlabel4)
    ax4.set_ylabel(ylabel4)
    ax4.set_title(title4)
    ax4.legend()
    ax4.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'transferability_analysis.png'),
dpi=300, bbox_inches='tight')
    plt.close()
    print(f"迁移性分析图表已保存到: {os.path.join(output_dir,
'transferability_analysis.png')}")

    return df

def analyze_ensemble_effectiveness(results):
    """分析集成攻击的有效性"""
    output_dir = create_output_dir()

    ensemble_data = []
    single_data = []
```

```
    for (attack_method, attack_type, src_model, epsilon, eval_model), acc in
results.items():
        attack_success_rate = 1 - acc

        if attack_type == 'ensemble':
            ensemble_data.append({
                'attack_method': attack_method.upper(),
                'src_model': src_model,
                'eval_model': eval_model,
                'epsilon': epsilon,
                'attack_success_rate': attack_success_rate,
                'attack_type': 'Ensemble'
            })
        else:
            single_data.append({
                'attack_method': attack_method.upper(),
                'src_model': src_model,
                'eval_model': eval_model,
                'epsilon': epsilon,
                'attack_success_rate': attack_success_rate,
                'attack_type': 'Single'
            })

    ensemble_df = pd.DataFrame(ensemble_data)
    single_df = pd.DataFrame(single_data)

    # 对比集成攻击与单模型攻击
    fig, axes = plt.subplots(1, 2, figsize=(15, 6))

    # 图1：平均成功率对比
    ax1 = axes[0]
    ensemble_avg = ensemble_df.groupby(['attack_method', 'epsilon'])
['attack_success_rate'].mean().reset_index()
    single_avg = single_df.groupby(['attack_method', 'epsilon'])
['attack_success_rate'].mean().reset_index()

    for method in ['FGSM', 'MIFGSM']:
        ens_data = ensemble_avg[ensemble_avg['attack_method'] == method]
        sin_data = single_avg[single_avg['attack_method'] == method]

        ens_label = f'{method} (集成)' if USE_CHINESE else f'{method} (Ensemble)'
        sin_label = f'{method} (单模型)' if USE_CHINESE else f'{method} (Single)'

        ax1.plot(ens_data['epsilon'], ens_data['attack_success_rate'],
                 'o-', label=ens_label, linewidth=3, markersize=8)
        ax1.plot(sin_data['epsilon'], sin_data['attack_success_rate'],
                 's--', label=sin_label, linewidth=2, markersize=6)

    xlabel1 = '扰动强度 (Epsilon)' if USE_CHINESE else 'Epsilon'
    ylabel1 = '平均攻击成功率' if USE_CHINESE else 'Average Attack Success Rate'
```

```
    title1 = '集成攻击 vs 单模型攻击' if USE_CHINESE else 'Ensemble vs Single Model
Attacks'
    ax1.set_xlabel(xlabel1)
    ax1.set_ylabel(ylabel1)
    ax1.set_title(title1)
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # 图2：按目标模型分组的成功率
    ax2 = axes[1]
    combined_df = pd.concat([
        ensemble_df.assign(attack_strategy='Ensemble'),
        single_df.assign(attack_strategy='Single')
    ])

    model_comparison = combined_df.groupby(['eval_model', 'attack_strategy',
'attack_method'])['attack_success_rate'].mean().reset_index()

    x = np.arange(3)  # 三个模型
    width = 0.15

    for i, method in enumerate(['FGSM', 'MIFGSM']):
        for j, strategy in enumerate(['Single', 'Ensemble']):
            data = model_comparison[(model_comparison['attack_method'] == method)
&
                                    (model_comparison['attack_strategy'] ==
strategy)]
            offset = (i * 2 + j) * width - 1.5 * width
            ax2.bar(x + offset, data['attack_success_rate'], width,
                   label=f'{method} ({strategy})', alpha=0.8)

    xlabel2 = '目标模型' if USE_CHINESE else 'Target Model'
    ylabel2 = '平均攻击成功率' if USE_CHINESE else 'Average Attack Success Rate'
    title2 = '各目标模型的攻击成功率' if USE_CHINESE else 'Attack Success Rate by
Target Model'
    xticklabels = ['模型1 (AlexNet)', '模型2 (VGGNet)', '模型3 (MobileNet)'] if
USE_CHINESE else ['Model 1 (AlexNet)', 'Model 2 (VGGNet)', 'Model 3 (MobileNet)']

    ax2.set_xlabel(xlabel2)
    ax2.set_ylabel(ylabel2)
    ax2.set_title(title2)
    ax2.set_xticks(x)
    ax2.set_xticklabels(xticklabels)
    ax2.legend()
    ax2.grid(True, alpha=0.3, axis='y')

    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'ensemble_effectiveness.png'), dpi=300,
bbox_inches='tight')
    plt.close()
```

```
    print(f"集成攻击有效性图表已保存到: {os.path.join(output_dir,
'ensemble_effectiveness.png')}")

    return ensemble_df, single_df

def generate_additional_plots(results):
    """生成额外的详细分析图表"""
    output_dir = create_output_dir()

    # 将结果解析为DataFrame
    all_data = []
    for (attack_method, attack_type, src_model, epsilon, eval_model), acc in
results.items():
        all_data.append({
            'attack_method': attack_method.upper(),
            'attack_type': attack_type,
            'src_model': src_model,
            'eval_model': eval_model,
            'epsilon': epsilon,
            'accuracy': acc,
            'attack_success_rate': 1 - acc
        })

    df = pd.DataFrame(all_data)

    # 图1: 扰动强度敏感性分析
    fig, ax = plt.subplots(figsize=(10, 6))
    single_attacks = df[df['attack_type'] == 'single']
    eps_method = single_attacks.groupby(['attack_method', 'epsilon'])
['attack_success_rate'].mean().reset_index()

    for method in ['FGSM', 'MIFGSM']:
        method_data = eps_method[eps_method['attack_method'] == method]
        ax.plot(method_data['epsilon'], method_data['attack_success_rate'],
                'o-', label=method, linewidth=3, markersize=8)

    xlabel = '扰动强度 (Epsilon)' if USE_CHINESE else 'Epsilon (Perturbation
Budget)'
    ylabel = '平均攻击成功率' if USE_CHINESE else 'Average Attack Success Rate'
    title = '攻击成功率随扰动强度变化' if USE_CHINESE else 'Attack Success Rate vs
Perturbation Budget'
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    ax.legend()
    ax.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'epsilon_sensitivity.png'), dpi=300,
bbox_inches='tight')
    plt.close()
```

```
    # 图2：模型脆弱性对比
    fig, ax = plt.subplots(figsize=(10, 6))
    model_vuln = single_attacks.groupby(['eval_model', 'attack_method'])
['attack_success_rate'].mean().reset_index()

    x = np.arange(3)
    width = 0.35

    fgsm_data = model_vuln[model_vuln['attack_method'] == 'FGSM']
['attack_success_rate']
    mifgsm_data = model_vuln[model_vuln['attack_method'] == 'MIFGSM']
['attack_success_rate']

    ax.bar(x - width/2, fgsm_data, width, label='FGSM', alpha=0.8)
    ax.bar(x + width/2, mifgsm_data, width, label='MI-FGSM', alpha=0.8)

    xlabel = '目标模型' if USE_CHINESE else 'Target Model'
    ylabel = '平均攻击成功率' if USE_CHINESE else 'Average Attack Success Rate'
    title = '模型脆弱性对比' if USE_CHINESE else 'Model Vulnerability Comparison'
    xticklabels = ['模型1 (AlexNet)', '模型2 (VGGNet)', '模型3 (MobileNet)'] if
USE_CHINESE else ['Model 1 (AlexNet)', 'Model 2 (VGGNet)', 'Model 3 (MobileNet)']

    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    ax.set_xticks(x)
    ax.set_xticklabels(xticklabels)
    ax.legend()
    ax.grid(True, alpha=0.3, axis='y')
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'model_vulnerability.png'), dpi=300,
bbox_inches='tight')
    plt.close()

    print(f"额外分析图表已保存到: {output_dir}")

def generate_report(results):
    """生成综合分析报告"""
    print("="*80)
    print("对抗攻击综合分析报告")
    print("="*80)

    # 将结果解析为DataFrame
    all_data = []
    for (attack_method, attack_type, src_model, epsilon, eval_model), acc in
results.items():
        all_data.append({
            'attack_method': attack_method.upper(),
            'attack_type': attack_type,
```

```
                'src_model': src_model,
                'eval_model': eval_model,
                'epsilon': epsilon,
                'accuracy': acc,
                'attack_success_rate': 1 - acc
            })

    df = pd.DataFrame(all_data)

    # 任务1分析：单模型攻击
    print("\n1. 单模型攻击分析")
    print("-" * 50)

    single_attacks = df[df['attack_type'] == 'single']

    for eps in sorted(df['epsilon'].unique()):
        print(f"\n扰动强度 = {eps}/255:")
        eps_data = single_attacks[single_attacks['epsilon'] == eps]

        # 自攻击成功率
        self_attacks = eps_data[eps_data['src_model'] == eps_data['eval_model']]
        print("  自攻击成功率:")
        for _, row in self_attacks.iterrows():
            print(f"    {row['attack_method']} 攻击模型 {row['src_model']}:
{row['attack_success_rate']:.3f}")

        # 最佳迁移攻击
        print("  最佳迁移攻击:")
        transfer_attacks = eps_data[eps_data['src_model'] !=
eps_data['eval_model']]
        for src in sorted(transfer_attacks['src_model'].unique()):
            src_transfers = transfer_attacks[transfer_attacks['src_model'] ==
src]
            best_transfer =
src_transfers.loc[src_transfers['attack_success_rate'].idxmax()]
            print(f"    从模型 {src}: {best_transfer['attack_method']} → 模型
{best_transfer['eval_model']} "
                  f"(成功率: {best_transfer['attack_success_rate']:.3f})")

    # 任务2分析：受害模型对比
    print(f"\n2. 受害模型对比分析")
    print("-" * 50)

    print("各受害模型的平均攻击成功率:")
    victim_analysis = single_attacks.groupby(['eval_model', 'attack_method'])
['attack_success_rate'].mean()
    for model in sorted(single_attacks['eval_model'].unique()):
        print(f"\n模型 {model} 作为受害模型:")
        model_data = victim_analysis[victim_analysis.index.get_level_values(0) ==
model]
```

```
        for method in ['FGSM', 'MIFGSM']:
            if method in model_data.index.get_level_values(1):
                print(f"  {method}:
{model_data[model_data.index.get_level_values(1) == method].values[0]:.3f}")

    # 任务3分析：集成攻击
    if 'ensemble' in df['attack_type'].values:
        print(f"\n3. 集成攻击分析")
        print("-" * 50)

        ensemble_attacks = df[df['attack_type'] == 'ensemble']
        ensemble_avg = ensemble_attacks.groupby(['attack_method', 'epsilon'])
['attack_success_rate'].mean()
        single_avg = single_attacks.groupby(['attack_method', 'epsilon'])
['attack_success_rate'].mean()

        print("集成攻击的性能提升:")
        for method in ['FGSM', 'MIFGSM']:
            print(f"\n{method}:")
            for eps in sorted(df['epsilon'].unique()):
                if (method, eps) in ensemble_avg.index and (method, eps) in
single_avg.index:
                    improvement = ensemble_avg[(method, eps)] -
single_avg[(method, eps)]
                    print(f"  扰动强度 {eps}: +{improvement:.3f} 成功率提升")

if __name__ == "__main__":
    record_dir = "../data/test/advData/fake/record/"

    if os.path.exists(record_dir):
        results = parse_result_files(record_dir)

        print("正在生成分析图表...")

        # 生成可视化图表并保存
        transfer_df = analyze_transferability(results)

        if any('ensemble' in key[1] for key in results.keys()):
            ensemble_df, single_df = analyze_ensemble_effectiveness(results)

        # 生成额外的详细分析图表
        generate_additional_plots(results)

        # 生成文本报告
        generate_report(results)

        print("\n所有分析图表已保存到: ../data/test/advData/fake/analysis_plots/")
    else:
        print(f"结果目录 {record_dir} 不存在，请先运行实验。")
```

File: eval.py - Subtask2 模型评估脚本

```
import numpy as np
import os
import sys
# 添加父目录到Python路径
parent_dir = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
sys.path.insert(0, parent_dir)

import glob
from PIL import Image
import torch
import torch.nn as nn
from torchvision import transforms
from models import VGGNet,AlexNet,MobileNetV2
import argparse

parser = argparse.ArgumentParser(description='Anti-forensic evaluation')
parser.add_argument('--imgs_dir', type=str, default='../data/test/advData/fake/',
help='input image directory.')
parser.add_argument('--gpu_id', type=str, default='0', help='GPU ID')
parser.add_argument('--model_num', type=int, default=1, help='model to attack.')
parser.add_argument('--ckpt_vggnet', type=str, default="../checkPoint/VGGNet/
model.pth", help='path to VGGNet model.')
parser.add_argument('--ckpt_alexnet', type=str, default="../checkPoint/AlexNet/
model.pth", help='path to AlexNet model.')
parser.add_argument('--ckpt_mobilenet', type=str, default="../checkPoint/MobNet/
model.pth", help='path to MobileNetV2 model.')
parser.add_argument('--max_epsilon', type=int, default=2, help='maximum
perturbation.')
parser.add_argument('--num_iter', type=int, default=10, help='number of
iterations.')
parser.add_argument('--image_format', type=str, default='png', help='image
format.')
parser.add_argument('--image_width', type=int, default=128, help='width of
image.')
parser.add_argument('--image_height', type=int, default=128, help='height of
image.')
parser.add_argument('--attack_method', type=str, default='fgsm', choices=['fgsm',
'mifgsm'], help='attack method.')
parser.add_argument('--multi_model', action='store_true', help='evaluate ensemble
attack results.')

args = parser.parse_args()

os.environ['CUDA_VISIBLE_DEVICES'] = args.gpu_id
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

if args.multi_model:
    imgs_dir = args.imgs_dir + f'{args.attack_method}_ensemble_m' +
```

```
str(args.model_num) + '_eps_' + str(args.max_epsilon)
else:
    imgs_dir = args.imgs_dir + f'{args.attack_method}_src_m' +
str(args.model_num) + '_eps_' + str(args.max_epsilon)

record_dir = args.imgs_dir + 'record/'

if not os.path.isdir(record_dir):
    os.mkdir(record_dir)


preprocess = transforms.ToTensor()


def get_file_list(dataset_dir):
    file_list = glob.glob(os.path.join(dataset_dir, '*.png')) +
glob.glob(os.path.join(dataset_dir, '*.jpg'))
    file_list.sort()
    return file_list


def read_image(img_path):
    img = Image.open(img_path)
    img_tensor = preprocess(img)
    input_tensor = img_tensor.unsqueeze(0)
    return input_tensor


def save_records(acc, model_name='1', save_dir=record_dir):
    prefix = 'ensemble_' if args.multi_model else ''
    acc_record_path = save_dir + f'{args.attack_method}_{prefix}src_m' +
str(args.model_num) + '_eps_' + str(args.max_epsilon) + \
        '_eval_m' + model_name + '.txt'
    print('Evaluated on model: %s, imgs_dir: %s, acc= %.3f' % (model_name,
args.imgs_dir, acc))
    f = open(acc_record_path, 'w')
    print('Evaluated on model: %s, imgs_dir: %s, acc= %.3f' % (model_name,
args.imgs_dir, acc), file=f)
    f.close()

# load model for attack
def load_model(model_path, net):
    "load pretrained model to network"
    if not os.path.exists(model_path):
        print(f"Warning: Model file {model_path} not found at
{os.path.abspath(model_path)}")
        return None

    model = net().to(device)
    try:
        model.load_state_dict(torch.load(model_path, map_location='cpu',
```

```
weights_only=True))
    except:
        model.load_state_dict(torch.load(model_path, map_location='cpu'))
    return model

def get_acc(model_path, net, input_dir, gt_label):
    model = load_model(model_path, net)
    if model is None:
        print(f"Failed to load model from {model_path}")
        return 0.0

    model.eval()

    img_lists = get_file_list(input_dir)
    if len(img_lists) == 0:
        print(f"No images found in {input_dir}")
        return 0.0

    cnt = 0
    for img_idx, img_path in enumerate(img_lists):
        img = read_image(img_path)
        img = img.to(device)
        output = model(img)
        _, pred = torch.max(output, 1)
        if pred.data[0].cpu().numpy() == gt_label:
            cnt += 1
    acc = cnt / len(img_lists)
    return acc


# create label tensor - 修复路径解析
folder_name = args.imgs_dir.rstrip('/')
path_parts = folder_name.split('/')

imgs_label = None
for part in reversed(path_parts):
    if 'real' in part or 'fake' in part:
        imgs_label = part
        break

if imgs_label is None:
    for part in reversed(path_parts):
        if part:
            imgs_label = part
            break

if imgs_label is None:
    print("Warning: Cannot determine image label from path")
    imgs_label = "fake"  # 默认假设为fake

if 'real' in imgs_label:
```

```
    img_label = 1
elif 'fake' in imgs_label:
    img_label = 0
else:
    print(f"Warning: Cannot determine if images are real or fake from:
{imgs_label}")
    img_label = 0

print(f"Image label determined as: {img_label} ({'real' if img_label == 1 else
'fake'})")

#evaluate AlexNet
print('\n================================================================================
print('Evaluating on AlexNet (Model 1) ...')
acc = get_acc(args.ckpt_alexnet, AlexNet, imgs_dir, img_label)
save_records(acc, '1', record_dir)
print('================================================================================\

#evaluate VGGNet
print('\n================================================================================
print('Evaluating on VGGNet (Model 2) ...')
acc = get_acc(args.ckpt_vggnet, VGGNet, imgs_dir, img_label)
save_records(acc, '2', record_dir)
print('================================================================================\

#evaluate MobileNetV2
print('\n================================================================================
print('Evaluating on MobileNetV2 (Model 3) ...')
acc = get_acc(args.ckpt_mobilenet, MobileNetV2, imgs_dir, img_label)
save_records(acc, '3', record_dir)
print('================================================================================\
```

File: run_complete_experiment.py - 对抗攻击实验脚本

```
"""
完整的对抗攻击实验脚本
实现任务1、2、3的所有要求
"""

import subprocess
import os
import sys

def run_task1():
    """任务1：以AlexNet为受害模型的攻击实验"""
    print("=" * 60)
    print("任务1：以AlexNet为受害模型的攻击实验")
```

```
    print("=" * 60)

    epsilons = [2, 4, 8, 10]
    attack_methods = ["fgsm", "mifgsm"]

    for attack_method in attack_methods:
        for eps in epsilons:
            print(f"\n运行 {attack_method.upper()} 攻击，扰动量: {eps}/255")

            # 攻击fake图像
            cmd_attack = [
                "python", "attack.py",
                "--input_dir", "../data/test/clean/fake/",
                "--output_dir", "../data/test/advData/fake/",
                "--model_num", "1",  # AlexNet作为受害模型
                "--max_epsilon", str(eps),
                "--attack_method", attack_method,
                "--num_iter", "10"
            ]
            subprocess.run(cmd_attack)

            # 评估攻击效果
            cmd_eval = [
                "python", "eval.py",
                "--imgs_dir", "../data/test/advData/fake/",
                "--model_num", "1",
                "--max_epsilon", str(eps),
                "--attack_method", attack_method
            ]
            subprocess.run(cmd_eval)

def run_task2():
    """任务2：以VGGNet和MobileNet为受害模型的攻击实验"""
    print("=" * 60)
    print("任务2：受害模型比较实验")
    print("=" * 60)

    models = [2, 3]  # VGGNet, MobileNet
    model_names = ["VGGNet", "MobileNet"]
    epsilons = [2, 4, 8, 10]
    attack_methods = ["fgsm", "mifgsm"]

    for model_idx, (model_num, model_name) in enumerate(zip(models,
model_names)):
        print(f"\n以 {model_name} 为受害模型")

        for attack_method in attack_methods:
            for eps in epsilons:
                print(f"运行 {attack_method.upper()} 攻击，扰动量: {eps}/255")
```

```
                # 攻击
                cmd_attack = [
                    "python", "attack.py",
                    "--input_dir", "../data/test/clean/fake/",
                    "--output_dir", "../data/test/advData/fake/",
                    "--model_num", str(model_num),
                    "--max_epsilon", str(eps),
                    "--attack_method", attack_method,
                    "--num_iter", "10"
                ]
                subprocess.run(cmd_attack)

                # 评估
                cmd_eval = [
                    "python", "eval.py",
                    "--imgs_dir", "../data/test/advData/fake/",
                    "--model_num", str(model_num),
                    "--max_epsilon", str(eps),
                    "--attack_method", attack_method
                ]
                subprocess.run(cmd_eval)

def run_task3():
    """任务3：集成攻击实验"""
    print("=" * 60)
    print("任务3：集成攻击实验")
    print("=" * 60)

    target_models = [1, 2, 3]
    model_names = ["AlexNet", "VGGNet", "MobileNet"]
    epsilons = [2, 4, 8, 10]
    attack_methods = ["fgsm", "mifgsm"]

    for target_model, target_name in zip(target_models, model_names):
        print(f"\n目标模型：{target_name}")

        for attack_method in attack_methods:
            for eps in epsilons:
                print(f"集成 {attack_method.upper()} 攻击，扰动量：{eps}/255")

                # 集成攻击
                cmd_attack = [
                    "python", "attack.py",
                    "--input_dir", "../data/test/clean/fake/",
                    "--output_dir", "../data/test/advData/fake/",
                    "--model_num", str(target_model),
                    "--max_epsilon", str(eps),
                    "--attack_method", attack_method,
                    "--multi_model",  # 启用集成攻击
                    "--num_iter", "10"
```

```
                ]
                subprocess.run(cmd_attack)

                # 评估集成攻击效果
                cmd_eval = [
                    "python", "eval.py",
                    "--imgs_dir", "../data/test/advData/fake/",
                    "--model_num", str(target_model),
                    "--max_epsilon", str(eps),
                    "--attack_method", attack_method,
                    "--multi_model"
                ]
                subprocess.run(cmd_eval)

def run_analysis():
    """运行结果分析"""
    print("=" * 60)
    print("生成分析报告")
    print("=" * 60)

    cmd_analysis = ["python", "analysis.py"]
    subprocess.run(cmd_analysis)

def main():
    """主函数：运行完整实验"""
    print("开始运行完整的对抗攻击实验")
    print("确保以下目录存在：")
    print("- ../data/test/clean/fake/")
    print("- ../data/test/clean/real/")
    print("- ../checkPoint/AlexNet/model.pth")
    print("- ../checkPoint/VGGNet/model.pth")
    print("- ../checkPoint/MobNet/model.pth")

    # 切换到attacks目录确保相对路径正确
    script_dir = os.path.dirname(os.path.abspath(__file__))
    os.chdir(script_dir)

    try:
        # 运行三个任务
        run_task1()
        run_task2()
        run_task3()

        # 生成分析报告
        run_analysis()

        print("\n" + "=" * 60)
        print("所有实验完成！")
        print("结果文件保存在：../data/test/advData/fake/record/")
        print("=" * 60)
```

```
    except KeyboardInterrupt:
        print("\n实验被用户中断")
    except Exception as e:
        print(f"\n实验过程中出错: {e}")

if __name__ == "__main__":
    main()
```