Project ： 对抗攻防

520030910334 张涵雪

1. 整体设置

- 使用 class Warp_Resnet 处理 mean 与 std
- 使用 argparser 让代码简洁
- 使用 assert 保证符合范数规范
- 忽略模型错分的个别测试数据，ResNet-50 ACC 96.6%, vit 97.1%, 在正确分类的数据里面计算 untargeted 攻击成功率

```
# Drop the misclassified sample
label = torch.tensor([int(item[1])]).cuda()
if pred.item() != label: continue
```

顺便说明一些最初走过的弯路：

- 在最初的尝试中我使用了单步进行攻击，查阅资料后发现：对于线性的网络，确实可以直接使用单步的攻击，但是对于复杂的模型（如我们用到的 resnet、vit），仅仅做一次迭代，方向是不一定完全正确的，因此应该采用多步的 pgd 来进行迭代攻击，在更改攻击方式后，我三种约束下的模型攻击成功率都有了大幅度提升，具体如下（**这里是优化前的 pgd，非最终结果**）：

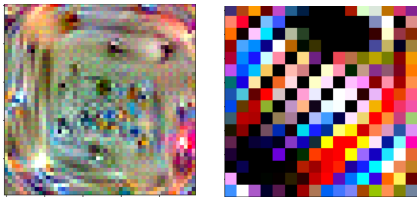
表 1: 迭代与否性能对比

模式	**Linf**	**L2**	**Patch**
单步 resnet	82.71%	52.07%	10.67%
迭代 resnet	98.14%	67.91%	42.31%
单步 vit	69.43%	21.78%	3.72%
迭代 vit	91.72%	41.82%	27.43%

- 在最初 pgd 的实现中，我的每步步长为 Budget/攻击轮数，这样在我想要提高攻击轮数来达到更好的效果的同时，攻击的强度就受到了限制，导致轮数增加时每轮强度减小，从而达不到效果。经过调试，我将步长设置为一个稍大一些的值，保证每次攻击的有效性，并在超过限制时将超出的部分裁去，使攻击效果得到了提升。

- 在 L2 的约束下，我的攻击其实仍然不够高，考虑到 0.3 作为一个非常小的值，我们需要尽可能抓住梯度中的关键信息。直觉认为在靠前的迭代轮数中所包含的信息更多，因此我们对每轮的攻击强度进行一个衰减，并适当增加初始的攻击强度，从而保证攻击更能充分利用这 0.3 的距离，最终提升了 L2 约束下的攻击成功率（resnet 更加明显）。

- 对于 Patch Attack，我在最初看到这个条件时感到十分困惑，因为其中的不确定性太多了，既可以优化 patch 的内容，还可以优化位置。我最先是试图生成一个含有极强的某种信息的万能贴纸，使得这种贴纸可以在不考虑位置的情况将原图中所含的信息带偏。然而，经过实验，这种方法在图片更大时才 work，而在 16*16 的 patch 上几乎没有作用，且训练速度极慢。直觉上认为出现这种错误的原因主要是：16*16 的 patch size 相对于图片的尺寸实在是太小了，无法涵盖很多的信息，同时如果我们随机生成位置，其噪声可能远远大于可训练的信息，从而造成失败。(具体关于 Patch Attack 位置的试探还有很多，放在最终的结果中)



(a) 40*40 (b) 16*16

图 1: 不同尺寸万能贴纸

2. 白盒攻击

2.1. Linf 约束

- 使用 PGD 实现对 ResNet-50 和 vit 的白盒攻击
- Budget: 1 / 255
- 攻击轮数：40
- 每轮强度：0.1 / 255

结果为：

- ResNet-50 攻击成功率：**99.69%**
- vit 攻击成功率：**95.58%**

结论：攻击成功率已经很高，不再进行调参优化

2.2. L2 约束

- 使用 PGD 实现对 ResNet-50 和 vit 的白盒攻击
- Budget: 0.3
- 尝试的创新点：增加迭代次数，并使用类似（learning rate decay）的思想来降低每一个 step 的系数，从而拿到更好的优化结果。

结果为：

- **resnet50:** 迭代次数：80，使用 decay，epsilon_iter = 0.04，成功率：**85.6%**
- **ViT:** 每次迭代 epsilon_iter = 0.04，如果使用 decay，在后一半迭代次数中，epsilon_iter /= 2

表 2: *PGD attack on ViT*

迭代次数	**80**	**40**
使用 decay	64.1%	60.6%
不使用 decay	63.3%	60.5%

结论：vit 最好攻击成功率 **64.1%**，增大迭代次数会很有帮助，但会显著增加攻击时间，使用 decay 可以帮助一点点。

2.3. Patch-based whitebox attack

- 使用 mask 保证范数规范

```
adv = original_image * (1 - mask) + x * mask
```

- 实现思路：首先选取 patch，其次对 patch 内像素使用 PGD-Linf 进行攻击，根据之前经验，增大攻击轮数从而提升攻击强度，并使用 decay 技巧
- 攻击轮数：80

探究创新点：如何选取 patch

- **On ResNet-50:** 探究四种 patch 位置选取方式，选取前 50 个数据为验证集，括号中为攻击成功率，epsilon_iter：0.05 / 256
  - Patch 选择左上角（32%）：听起来就是最差的，事实也是
  - Pathc 选择中间（54%）：中性策略？事实是最好的
  - 使用 16x16 平均卷积去计算梯度绝对值之和，选择梯度信息最大的 patch（46%）：效果不是最好的原因可能因为 patch-based attack 中所有 pixel 变化幅度巨大，梯度信息只能决定“微小”变化内影响最终 logits 大小，因此不准。
  - 每隔 4 个 pixel，丢掉 16x16 的 patch，观察丢掉次 patch 是否能让 logits 减小（40%），以此来衡量 patch 的重要性：实验中发现总是倾向于丢掉左上角附近的 patch，而不是网络中间，并且 logits 减小程度也不大，因此效果不是最好。
  - **探索结果：**patch 放在最中间，效果最好，纵使可以通过不同位置多次攻击选择最好，但过于浪费攻击时间。报告尝试去用极短的时间去找合适的位置，但遗憾并没有找到。

最终实验结果：epsilon_iter = 0.1 / 255, 选择 [105, 105]（上述方案 2）为 adversarial patch 起始位置，攻击成功率 **62.2%**

- **On vit:** 根据 vit 源代码，每一个 patch embedding 成 768 的 feature 是通过卷积实现，此卷积将每一个 patch 16x16 个像素，通过 256 x 768 线性层，映射到特征空间。

```
self.proj = nn.Conv2d(in_chans, embed_dim,
    kernel_size=patch_size, stride=
    patch_size)
```

  - 选择把一个 patch 内所有像素改变，可以直接实现完全操控此 patch 的 embedding feature，但不能改变其他的 patch embedding
  - 选择四个 vit patch 的交汇区域，可以实现同时影响四个相邻 patch 的 feature，但影响程度有限

  具体的效力取决于影响四个相邻 patch 更有用，还是完全操控一个 patch feature 更有用，此处难以下定论。

  最终实验结果： epsilon_iter = 0.1 / 255, 选择 [105, 105] 为 adversarial patch 起始位置，攻击成功率 **73.8%**

- **CNN, ViT, adversarial patch 分布区域规律：** 经过查找 CNN 与 ViT 的介绍，由于 CNN 更容易受到 texture 影响，而 ViT 更多关注全局相关性，因此猜想：CNN 的 adversarial 更倾向于与 texture 密集区域，而 ViT 的 patch 更容易与 content 相关。

2.4. 迁移攻击

- 实验设置：同时忽略被两个模型都分错的样本，报告攻击准确率，使用 Linf 对抗样本
- 用攻击 resnet-50 的对抗样本攻击 vit，成功率 0.63%
- 用攻击 resnet-50 的对抗样本攻击 vit，成功率 0.95%

3. 黑盒 L2 攻击

3.1. 一些说明

- 实现方法: Simple Black Box Attack (SimBA)
- L2 范数: 5
- 最大迭代次数: 10000
- 每次迭代增加 noise 的 L2 范数 eps_iter = 0.004
- ViT 攻击成功率 **48.6%** 平均 query 数量 2080 次查询
- ResNet 攻击成功率 **61.1%**, 平均 query 数量 1750 次查询
- 由于运算力有限, 没有进行调参优化, 应该还有空间

3.2. 对创新点的思考

- 通过阅读 LeBA 论文, 尝试复现使用替身网络 gradient map 来指导增加 noise 这一思路, 在 gradient map 大的地方多进行随机 noise 搜索, 听上去就是一个 work 的方法. 此项目中, 每隔 100 轮对 adv_image 进行 TIMI 攻击, 然后计算高斯平滑后的 adv_noise 作为引导区域. 并根据引导区域对随机产生的 noise 进行 normalization, 引导区域大的地方多进行随机 noise 搜索

  对比实验结果： Resnet 攻击成功率 **61.8%**, 平均 query 数量 1658 次查询，可以看到代码是有效果的，但对比 LeBA 原文效果还是不够强，虽然实验设置不一样。由于时间和算力限制，未进行大规模优化。

- 关于在 ViT 上降低搜索空间: 由于 ViT 的每一个 patch 的像素会使用同一个 (3x16x16 -> 768, stride = kernelsize 的卷积)ensemble 成一个单独的 feature vector, 数学上等价于不同 patch 都会经过一个 3x16x16 -> 768 的全连接层进行处理, 因此不同 patch 的像素

不会互相影响, 实际搜索 adv_noise 可以分 patch 进行, 从而降低搜索空间, 但算力限制没有做出来.

(ViT 将 patch embed 到特征空间的 1x1 卷积层)

```
self.proj = nn.Conv2d(in_chans, embed_dim,
    kernel_size=patch_size, stride=
    patch_size)
```

(a) clean (b) noise_Linf (c) noise_L2

(d) noise_patch (e) noise_Black_L2

图 2: 对抗样本展示