Recommonmark Documentation

发布 *1.0*

Lu Zero, Eric Holscher, and contributors

2020 年 10 月 18 日

Contents:

CHAPTER 1

开始使用

这一部分主要讲解如何使用 AISafety 平台，包含环境安装，硬件设备部署，项目下载等内容。

1.1 项目简介

1.1.1 项目研究背景

人工智能技术在公共安全、金融经济、国防安全等领域取得了巨大进展和广泛应用。然而，由于现实应用场景的开放性，当前的人工智能技术暴露出稳定性、安全性等方面的安全隐患。对于人工智能算法的评测与度量，对于理解人工智能的行为并进一步提升其质量和推进在真实场景中的可用性具有重要意义。

1.1.2 项目研究目标

项目的核心总目标是建设技术先进、安全可靠、资源丰富、社群活跃的新一代人工智能开源社区。项目从基础平台、评测防护、重点应用三个方面系统地开展课题研究。本项目主要面向智能算法评测相关的理论基础、技术方法和支撑平台方面的挑战，形成评测方法、技术体系和支撑平台完整配套的全域解决方案，保障不同类型的人工智能算法应用安全，并在图像分类，文本情感分析和语音识别等领域开展示范应用。

1.2 主要研究内容

突破人工智能算法模型的测评技术。构建人工智能算法和模型的功能性、安全性、可靠性等质量评测标准体系，建立通用智能算法评测框架和评测方法，消除算法模型的隐含缺陷、提高决策行为的可解释性，研发智

能算法决策过程静态分析和动态追踪技术，实现智能算法缺陷的快速准确定位、智能模型结构优化与安全防护。研发人工智能算法与模型评测工具，构建人工智能算法与模型评测典型方案库，搭建智能算法评测平台。

1.2.1 项目成员

北京航空航天大学 DIG 实验室，主要成员包括：

Faculty

PhD Students

MPhil Students

Undergraduate Students

1.3 算法内核简介

1.3.1 算法内核结构

```
EvalBox
   Analysis
   Attack
   Defense
   Evaluation
Models
   TestModel
   UserModel
   weights
   basic_module.py
utils
test
    testimport.py
    testimport_black.py
Datasets
```

1.3.2 项目具体说明

EvalBox

EvalBox 是整个评估过程中使用到的攻击，评测，防御，分析的工具箱

- Anaysis：对评估过程运用组合和集成的分析函数，来完成一个分析的过程
- Attack：生成攻击样本的攻击算法库
- defense：防御算法库
- Evaluation：评测攻击指标的评测算法库
- UserEvaluation：由用户扩展的评测攻击指标的评测算法库

Models

Models 是平台提供的部分模型库，用户也可以自行添加

- TestModel：平台提供的部分模型库
- UserModel：用于用户添加符合扩展要求的模型库
- weights：存放相应的模型权重的位置
- basic_module.py：包含一些和模型组成相关基础函数

utils

utils 是工具库，包含常用的 io，图像的处理，模型等相关的工具函数

test

test 是平台提供的集成了完整攻击-预测-评测的流程的集成脚本工具，包含黑白盒调用方式

- testimport.py 完整攻击（攻击样本可以支持黑盒的，也可从其他平台获取攻击样本，在攻击样本处设置为黑盒）、预测、评测的白盒调用
- testimport.py 可分离为，仅仅生成攻击样本作为用于其他平台的黑盒数据输入，也可以完整使用生成的攻击样本，用在他处黑盒获得他处符合格式的黑盒预测结果，进行下一步的评测指标计算，输出指标值和可视化结果

Datasets

Datasets 是存放常用数据集或用户自定义符合扩展要求的数据集的位置

1.4 开始运行项目

运行项目前，需要先安装项目对应依赖环境

1.4.1 基础 Python 环境

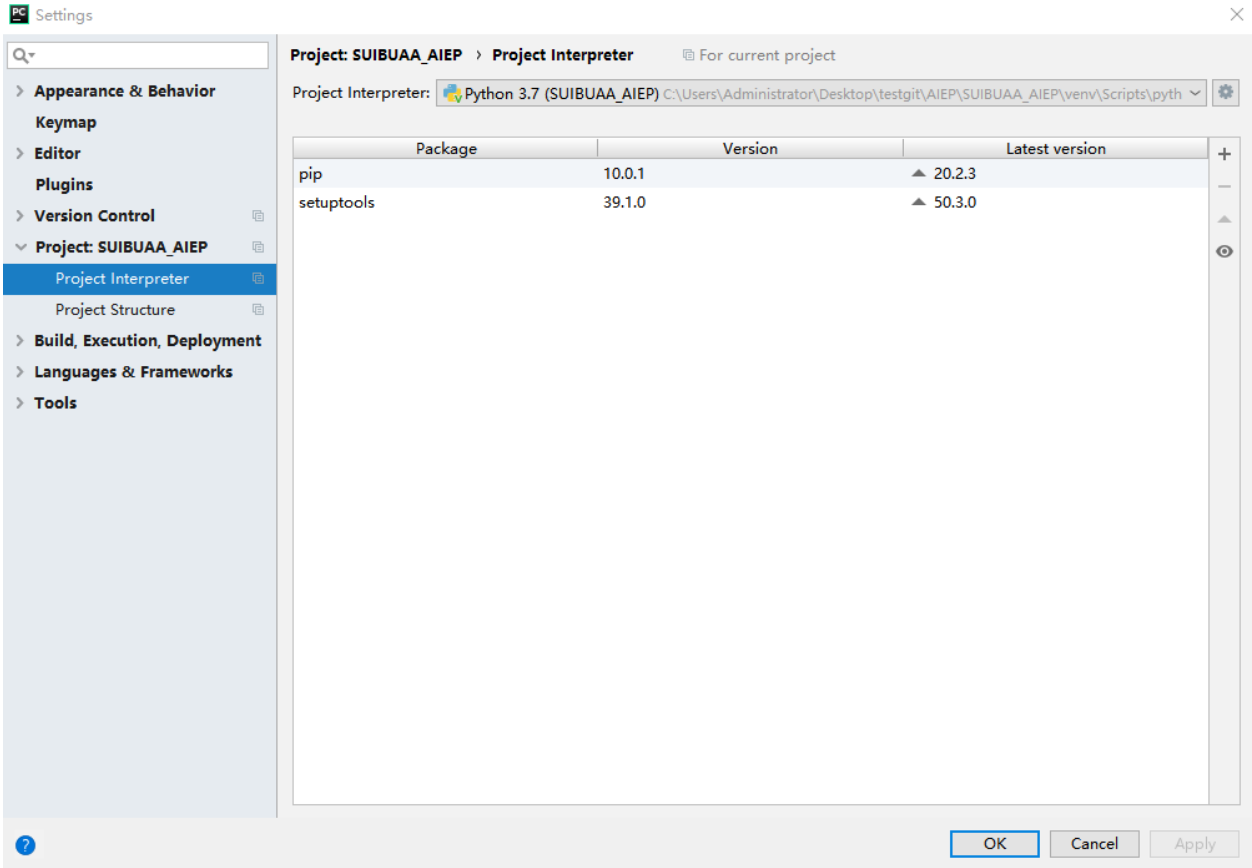
```
python==3.6.5
```

1.4.2 辅助环境

```
certifi==2020.6.20
cloudpickle==1.6.0
cycler==0.10.0
dask==2.27.0
decorator==4.4.2
future==0.18.2
imageio==2.9.0
kiwisolver==1.2.0
matplotlib==3.3.0
networkx==2.5
numpy==1.19.2
opencv-python==3.4.2.16
pandas==0.23.4
Pillow==7.2.0
pyparsing==2.4.7
python-dateutil==2.8.1
pytz==2020.1
PyWavelets==1.1.1
PyYAML==5.3.1
scikit-image==0.17.2
scipy==1.5.2
six==1.15.0
tifffile==2020.9.3
toolz==0.10.0
Wand==0.5.2
```

1.4.3 环境配置安装

新建空环境

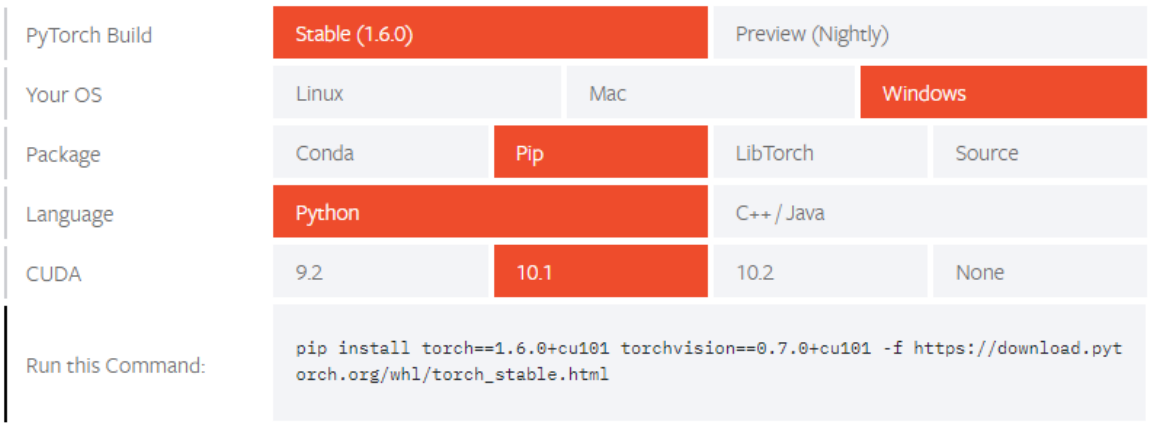


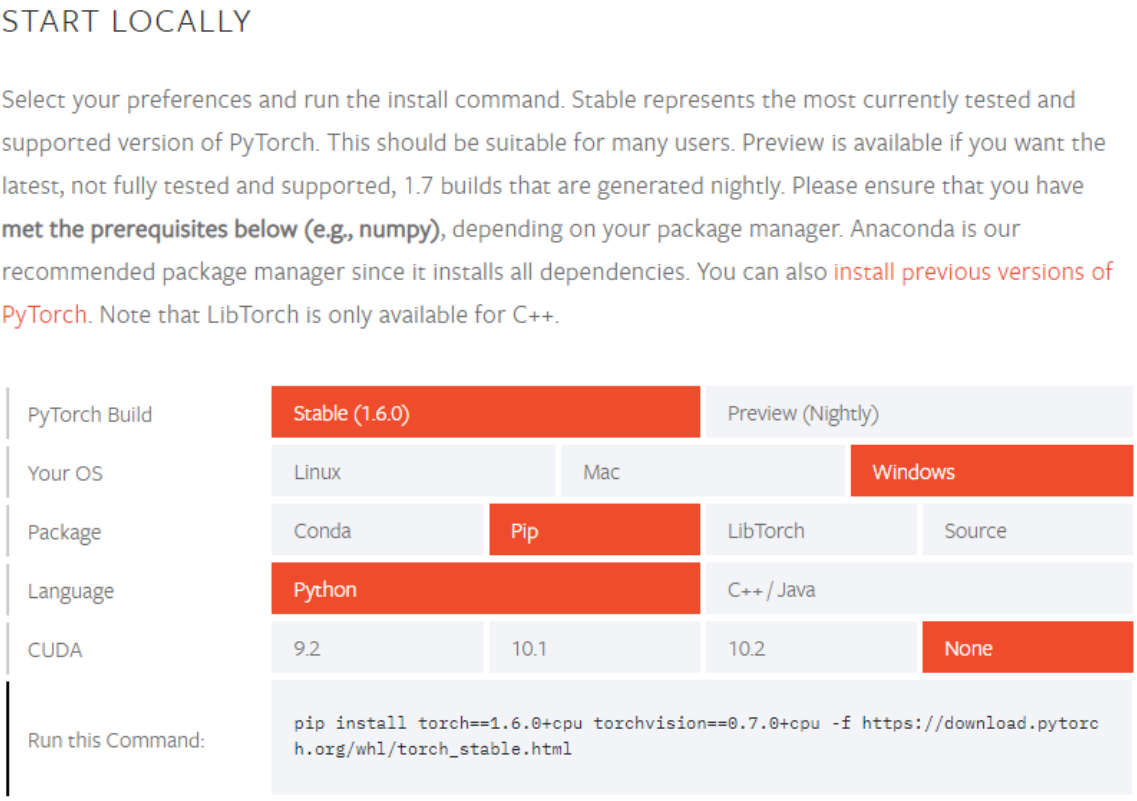
安装 torch

torch 的安装参照官网，根据用户的环境选择合适的，例如在 gpu 下，要选择 cuda 的方式，cpu 就按照 cpu 方式。

START LOCALLY

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, 1.7 builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.





1.4.4 依赖安装

安装的包参见 requirements.txt

```
pip install -r requirement.txt
```

在 GPU 的方式下，需要根据硬件信息，安装对应的硬件驱动，cuda 和 cudnn 版本

cpu 下安装完成后的 python 安装包环境如下：

```
Package              Version
-------------------  -------------
certifi              2020.6.20
cloudpickle          1.6.0
cycler               0.10.0
dask                 2.27.0
decorator            4.4.2
future               0.18.2
imageio              2.9.0
```

(下页继续)

(续上页)

```
kiwisolver              1.2.0
matplotlib              3.3.0
networkx                2.5
numpy                   1.19.2
opencv-python           3.4.2.16
pandas                  0.23.4
Pillow                  7.2.0
pip                     10.0.1
pyparsing               2.4.7
python-dateutil         2.8.1
pytz                    2020.1
PyWavelets              1.1.1
PyYAML                  5.3.1
scikit-image            0.17.2
scipy                   1.5.2
setuptools              39.1.0
six                     1.15.0
tifffile                2020.9.3
toolz                   0.10.0
torch                   1.6.0+cpu
torchvision             0.7.0+cpu
Wand                    0.5.2
```

1.4.5 程序安装说明

使用 GitHub 在开源地址下 clone 或者 fork 源码，并参考环境安装配置

1.5 CPU/GPU 模式和设置说明

该平台会自动检测运行的环境是否存在 GPU，默认需要安装 cuda 的环境中才可以使用 GPU。

1.5.1 使用 CPU 运行程序

无需设置，可自动识别

1.5.2 使用单一/多 GPU 运行程序

用户需要在集成调度接口 `~/test/testimport.py` 中，对参数`--GPU_Config` 进行设置

```
parser.add_argument(
    '--GPU_Config',
    type=str,
    # 数目，index 设置
    default=["2","0,1"])
```

说明：

默认输入的是 GPU 数目，第二个是 GPU 的编号，程序会根据用户输入的信息自动校验实际运行环境的 GPU 情况，合理做出判断，并指定一个可用的 GPU 设备号作为运行用的设备

1.6 Docker 模式和设置说明

1.6.1 Docker CE 安装（Linux 环境）

软件下载

官方地址：官网下载 阿里 docker-ce 镜像：Docker 镜像

内核要求

docker 要求 centos 系统内核版本必须高于 3.10 centos 下通过 uname -r 查看内核版本。

软件安装

1. **卸载旧版本 docker 相关包**

```
sudo yum remove docker  docker-common docker-selinux docker-engine
```

2. **安装需要的软件包**

yum-util 提供 yum-config-manager 功能，另外两个是 devicemapper 驱动依赖的

```
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

3. **设置 yum 源**

设置官方源：

```
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.
↪repo
```

设置阿里源：

```
sudo yum-config-manager --add-repo https://mirrors.aliyun.com/docker-ce/linux/centos/
↪docker-ce.repo
```

4. **刷新 yum 源**

```
sudo yum makecache fast
```

5. **安装 docker-ce**

查看全部版本：（如果匹配不到 docker-ce 包，执行 4 刷新 yum 源）

```
yum list docker-ce --showduplicates | sort -r
```

安装指定版本：

```
sudo yum -y install docker-ce-18.03.0.ce
```

安装最新版：

```
sudo yum -y install docker-ce
```

6. **启动 & 开机启动**

启动：

```
systemctl start docker
```

开机启动：

```
systemctl enable docker
```

重启（按顺序执行 1,2）：

```
systemctl daemon-reload
systemctl restart docker
```

7. **验证是否安装成功**

```
docker version
```

输出：

```
 Client:
 Version:       18.03.0-ce
 API version:   1.37
```

(下页继续)

(续上页)

```
 Go version:    go1.9.4
 Git commit:    0520e24
 Built: Wed Mar 21 23:09:15 2018
 OS/Arch:       linux/amd64
 Experimental:  false
 Orchestrator:  swarm

Server:
 Engine:
  Version:      18.03.0-ce
  API version:  1.37 (minimum version 1.12)
  Go version:   go1.9.4
  Git commit:   0520e24
  Built:        Wed Mar 21 23:13:03 2018
  OS/Arch:      linux/amd64
  Experimental: false
```

1.6.2 Docker 使用

创建 dockfile

详见dockfile 创建说明，或参考附件中样例，下载链接

运行镜像

将 dockfile 文件与项目放在同一路径下；进入项目路径，运行 `docker build` 创建镜像；

查看 image 镜像

```
docker images
docker images -a
docker image ls
docker image ls -a
```

其他 Docker 命令

详见Docker 官网

1.6.3 Docker 运行

1.7 完整集成调用过程

1.7.1 接口文件存储位置

平台已集成有完整的攻击评测接口文件 testimport.py。接口文件路径如下：

```
  EvalBox
  Models
  utils
  test
      **testimport.py**
  Datasets
```

1.7.2 接口文件参数参考

如下为在 cifar-10 数据集上，执行一个完整攻击 + 评测过程，所使用的对应参数：

```
parser = argparse.ArgumentParser(description='The Attack and Evaluate Generation')
# common arguments
parser.add_argument(
    '--attack_method',
    type=str,
    nargs='*',
    default = ["FGSM"])
parser.add_argument(
    '--evaluation_method',
    type=str,
    default='ACC')
parser.add_argument(
    '--Data_path',
    type=str,
    nargs='*',
    default=["../Datasets/cln_data/cifar10_300_origin_inputs.npy", "../Datasets/cln_data/
↪cifar10_300_origin_labels.npy",                    "../Datasets/cln_data/cifar10_300_
↪origin_inputs.npy", "../Datasets/cln_data/cifar10_300_origin_labels.npy"])
parser.add_argument(
    '--Dict_path',
    type=str,
```

(下页继续)

(续上页)

```
    default="./dict_lists/cifar10_dict.txt")
parser.add_argument(
    '--defense_model',
    type=str,
    default='Models.UserModel.ResNet2')
parser.add_argument(
    '--model',
    type=str,
    default = 'Models.UserModel.FP_resnet')
parser.add_argument(
    '--model_dir',
    type=str,
    default = '../Models/weights/FP_ResNet20.th')
parser.add_argument(
    '--model_defence_dir',
    type=str,
    default='../Models/weights/resnet20_cifar.pt')
parser.add_argument(
    '--IS_COMPARE_MODEL',
    type=bool,
    default=False)
parser.add_argument(
    '--IS_TARGETTED',
    type=bool,
    default=False)
parser.add_argument(
    '--data_type',
    type=str,
    default='cifar10')
parser.add_argument(
    '--IS_WHITE', type=bool, default=True)
parser.add_argument(
    '--IS_PYTHORCH_WHITE', type=bool, default=False)
parser.add_argument(
    '--IS_DOCKER_BLACK', type=bool, default=False)
parser.add_argument(
    '--black_Result_dir' ,
    type=str,
    default = ' ')
parser.add_argument(
```

(下页继续)

(续上页)

```
    '--IS_SAVE',
    type=bool,
    default=False)
parser.add_argument(
    '--save_path',
    type=str,
    default='./Attack_generation/')
parser.add_argument(
    '--save_method',
    type=str,
    default='.npy')
parser.add_argument(
    '--Scale_ImageSize',
    type=int,
    default=32)
parser.add_argument(
    '--Crop_ImageSize',
    type=int,
    default=32)
parser.add_argument(
    '--batch_size', type=int, default=2, help='batch size')
parser.add_argument(
    '--CAM_layer',
    type=int,
    default=12)
parser.add_argument(
    '--GPU_Config',
    type=str,
    # 数目, index 设置
    default=["2","0,1"])
parser.add_argument(
    '--save_visualization_base_path',
    type=str,
    default='./temp/')
arguments = parser.parse_args()
main(args=arguments)
```

1.7.3 接口文件参数说明

attack_method

该参数表示攻击算法名称，目前平台已集成的所有攻击算法，均可通过缩写调用。

参考格式为：[“FGSM”]、[“PGD”]、[“DEEPFOOL”]

evaluation_method

该参数表示评测算法名称，目前平台已集成的所有评测算法，均可通过缩写调用。

参考格式为：[“ACC”]、[“ALDp”]

Data_path

该参数共有 4 个参数位，分别需传入：

1. 样本的数据集文件（白盒攻击下是原始样本，黑盒攻击下是攻击后的攻击样本）
2. 对应的标签的文件（非目标攻击下是 Ground Truth，目标攻击下是攻击目标类别）
3. 原始样本的数据集文件
4. 对应的 Ground Truth 标签文件

参考格式为：

[“../Datasets/”]

Dict_path

该参数表示所选择的数据集文件所对应的字典文件。字典文件中应包含对应的数据集的分类类别号码和类别名称的编号

参考格式为 “./dict_lists/cifar10_dict.txt”

以上述参考格式中 cifar10_dict.txt 文件为例，字典文件形如：

```
{'0': 'airplane', '1': 'automobile', ... , '8': 'ship', '9': 'truck'}
```

model

该参数表示用户选择的待执行评测任务的网络模型。使用时需给出模型地址。

参考格式：‘Models.UserModel.ResNet2’

model_dir

该参数表示用户选择的待执行测评任务的网络模型的参数文件，对应着上述 model 参数中模型文件。使用时使用相对地址调用。

参考格式：‘../Models/weights/resnet20_cifar.pt’

defense_model

default = ‘Models.UserModel.ResNet2’

该参数表示用户选择的防御后模型。与 model 参数类似，区别在于只有在使用对比评测算法时，才传入该参数，并设定 IS_COMPARE_MODEL 参数为 True。

参考格式：‘Models.UserModel.ResNet2’

model_defence_dir

default= ‘../Models/weights/resnet20_cifar.pt’

该参数表示用户选择的防御后模型的参数文件。与 model_dir 参数类似，区别在于只有在使用对比评测算法时，才传入该参数，并设定 IS_COMPARE_MODEL 参数为 True。

参考格式：‘../Models/weights/resnet20_cifar.pt’

IS_COMPARE_MODEL

该参数默认设置 False，如果需要比较两个不同模型的话，设置为 True

目前平台已集成的，可用于比较两个模型的评测方法有：CCV，CAV，COS，CRR，CSR

IS_TARGETTED

该参数表示使用的数据是否是目标攻击，False 是非目标攻击，True 是目标攻击方法

对于不同的选择，用户需提供相应的预处理数据集并传入对应位置。

data_type

default= “cifar10”，‘ImageNet’，[‘ImageNet’ ， “withoutNormalize”]

该参数表示数据集的类型，目前平台共支持三种数据集：

1. ‘cifar10’
2. ‘ImageNet’
3. [‘ImageNet’ ， “withoutNormalize”]

用户可根据实际需要注明类型，以方便预处理过程，另外使用 ImageNet 类型相似的数据一般不是原始的 ImageNet 数据集，选用 [‘ImageNet’ ， “withoutNormalize”] 时候，不对图像做归一化，用户自行预处理

IS_WHITE

该参数表示攻击样本的生成是否经过平台提供的正在使用的攻击方法，默认 True，即是通过攻击算法生成的，（攻击算法可以是白盒的方式，也可以是黑盒的方式，具体实现由用户添加黑盒的模拟梯度），False 的时候，输入的样本应当是其他攻击算法训练生成得到的，可以直接用平台的测评过程对模型进行评测

IS_PYTHORCH_WHITE

该参数表示评测过程是否要用到模型信息，目前平台已集成评测算法中，BD，RGB，RIC 这三个评测方法，默认需要模型信息，是设定为 True，其余算法均需设定为 False。

黑盒模式下，默认不需要模型信息，始终设定为 False

IS_DOCKER_BLACK

该参数表示是否使用 Docker 执行黑盒测试。在白盒方式下面，该参数默认设置 False，在黑盒方式下，可将该参数设置为 True，并通过外来数据（其他平台或者 docker 黑盒的结果）获取预测的结果

ONLY_GENRATE_BLACK_SAMPLE

该参数表示是否仅采用黑盒样本。设置为 True 的时候，只用平台产生对应的攻击样本，并保存在 save_path 下面的路径中。得到的结果供黑盒的平台使用。

设置为 True 时，只生成得到攻击样本，供用户使用到其他平台。

l 在其他平台使用本平台的攻击样本和原始数据获得预测结果，并获得测评结果

–ONLY_GENRATE_BLACK_SAMPLE 设置为 False

–IS_DOCKER_BLACK 同时设置为 True

用户使用原始的数据集和平台生成的样本，预测结果从他处获取，再计算测评指标

格式说明：

分为 json 和 npy 格式的，json 格式的例如 FGSM 的原始数据是

(30,3,32,32）和对应的标签是 (30,10) 格式，那么输出用于其他黑盒平台的数据也是一样的格式。

参 | 描述数 |		示例
BDPath	原数据集路径。用户需要获得路径下面的子目录，子目录下为数据集，名称固定为 inputs.* (格式支持 npy 和 json 两种；json 格式的 key 为 data)	npy 数据集：/dataset/basePath/1/inputs.npyJson 数据集：/dataset/basePath/1/inputs.json
CDPath	攻击数据集路径。用户需要获得路径下面的子目录，子目录下通常又存在多个二级目录，此二级目录需要保存下来，作为返回结果的 key，每个二级目录下会有对应的攻击数据集，名称固定为 inputs.* (格式支持 npy 和 json 两种；json 格式的 key 为 data) 对应我们保存输出的 Attack_generation 中的攻击样本	以 fgsm 算法为例：npy 数据集：/dataset/childPath/1/fgsm_01/inputs.npy/dataset/childPath/1/fgsm_02/inputs.npyJson 数据集：/dataset/childPath/1/fgsm_01/inputs.json/dataset/childPath/1/fgsm_02/inputs.json
RESPath	结果保存路径。用户需要将结果以 json 的格式保存到该目录下，就是 black_Result_dir 上面设置的数据	/result

black_Result_dir

该参数表示黑盒测试的中间结果存储路径。在白盒方式下面，该参数默认设置’’空，黑盒会设置具体数据的路径。以 FGSM+cifar10_30 数据集为例，返回结果的格式如下：

```
{
    "model":{
        "BDResult": Array[30],
        "CDResult": {
            "fgsm_01": Array[30],
            "fgsm_02": Array[30]
        }
    },
    "compare_model": {
        "BDResult": Array[30],
        "CDResult": {
            "fgsm_01": Array[30],
            "fgsm_02": Array[30]
        }
    }
}
```

以上结果具体键值说明：

Key	名称	说明
model	原始模型预测结果	原始模型预测结果放置在该 Key 下
compare_model	对比模型预测结果	对比模型预测结果放置在该 Key 下
BDResult	原数据预测结果	因为原始数据集仅有一个，所以结果放在该 Key 下即可
CDResult	对抗数据预测结果	项目中的攻击算法预存有若干参数。如 FGSM 算法预设两个参数组合 fgsm_01 和 fgsm_02。该 Key 下以二元字典的形式存储
fgsm	对抗数据集预测结果	以 FGSM 为例，该 key 为用户解析攻击数据集保存的上级目录，预测结果需要与该 key 对应

IS_SAVE

该参数表示是否保存生成的攻击样本，True 则保存攻击后的样本和对攻击样本的预测值，和原始预测值，格式和输入的一致，npy 类型的还是 npy，ImageNet 类型的还是图片和类别列表文件。

ImageNet 的攻击后的样本和预测类别会存在指定的路径下

save_path

该参数用于给出存储攻击结果的具体路径

如果使用的是形如 cifar10 的 npy 格式数据集，则最终存储形如以下目录树：

```
AISafety
  EvalBox
  Models
  utils
  test
      Attack_generation
          attack_param_FGSM_fgsm_01
            FGSM_30_adv_preds_labels.json
            FGSM_30_adv_preds_labels.npy   # 对抗预测标签
            FGSM_30_advs.json
            FGSM_30_advs.npy     # 对抗样本
          attack_param_FGSM_fgsm_02
  Datasets
```

如果使用的是形如 ImageNet 的原始图像格式数据集，则最终存储形式，为攻击图像和标签。如将 save_path 设置为 “ImageNet_Attack_generation/”。则攻击成功后会在该路径下生成形如下方的目录树：

```
AISafety
  EvalBox
  Models
  utils
  test
      Attack_generation
          attack_param_FGSM_fgsm_01
          attack_param_FGSM_fgsm_02
              Adv_Images  # 对抗图像
                  Adv_ILSVRC2012_val_00000001.JPEG
                  Adv_ILSVRC2012_val_00000002.JPEG
                  ...
                  Adv_ILSVRC2012_val_00000010.JPEG
              adv_preds_val_10.txt   # 对抗样本标签
              origins_val_10.txt   # 原始样本标签
  Datasets
```

攻击后的样本图像将被放在 Adv_Images 中，如图所示：

adv_preds_val_10.txt 和 origins_val_10.txt 文件中内容形如：

```
Adv_ILSVRC2012_val_00000001.JPEG 69
Adv_ILSVRC2012_val_00000002.JPEG 970
...
Adv_ILSVRC2012_val_00000010.JPEG 669
```

save_method

该参数表示若用户选择保存对抗攻击样本时，使用那种格式存储相应样本。默认是 npy，numpy 格式的，保存结果例如上述，如果设置是’ImageNet’就例如上述所示图片方式保存

参考格式为：“.npy”

Scale_ImageSize

该参数表示归一化尺寸。cifar10，cifar100 默认到 32,ImageNet 推荐归一化到 224X224 同时也支持用户自定义，用户自行决定放缩到多少后再裁减

参考格式：(32,32)、(224,224)

Crop_ImageSize

该参数表示用户定义好缩放尺寸后，进行裁减的尺寸

参考格式：(32,32)、(224,224)

batch_size

该参数表示数据每次批量处理的数目，仅用于攻击算法生成对抗样本阶段使用。

参考格式：64

CAM_layer

该参数表示可视化热力图的过程中，设置展示敏感和 feature 的第几层。

参考格式：28

GPU_Config

该参数表示项目使用 GPU 设置。第一个值表示，GPU 的数目，第二个表示可以使用的 GPU 的编号，目前支持，在判断和实际设备中信息一致的设备中任意选取一个使用。

参考格式：[“2”，“0,1”]。释义：共有 2 个 GPU，编号为 0 和 1，调用时程序会随机选择 GPU 使用。

save_visualization_base_path

该参数表示计算的评测结果和可视化的存储根目录的设置路径。

参考格式：‘./temp/’

以下为执行评测后相应的 temp 路径结果：topk 是模型预测使用攻击前后的样本的一个预测概率前 k 的一个比较，

对应的如果是 ImageNet 下面数据可视化结果还会生成热力图，详情请参阅可视化展示部分内容。

```
SUIBUAA_AIEP
  EvalBox
  Models
  utils
  test
     temp
        FGSM
           fgsm01
              topk
              AttackSample_torchvision.models.vgg16_cam_0.jpg
              ....jpg
           fgsm02
           ...
        result.txt
     attack_param
  Datasets
```

1.7.4 接口文件调用流程

设置好上述所有参数后，执行

```
python testimport.py
```

即可运行接口，并在控制台实时获取项目运行进度，以及相应的评测结果。

1.8 评测完整实例

1.8.1 参考实例 1（白盒攻击）

参数设置

testimport.py 中 parser 设置：

```
(   attack_method=["FGSM"],
    evaluation_method='ACTC',
    Data_path=["../Datasets/ImageNet/images/","../Datasets/ImageNet/val_4.txt","../
↪Datasets/ImageNet/images/","../Datasets/ImageNet/val_4.txt"],
```

(下页继续)

(续上页)

```
    Dict_path="./dict_lists/ImageNet_12_dict.txt",
    model_dir=' ',
    model='torchvision.models.vgg19',
    defense_model=' ',
    model_defence_dir=' ',
    data_type='ImageNet',
    IS_WHITE=True,
    IS_PYTHORCH_WHITE=False,
    black_Result_dir=' ',
    IS_SAVE=True,
    IS_COMPARE_MODEL=False,
    Scale_ImageSize=(375,500),
    Crop_ImageSize=(375,500),
    IS_TARGETTED=False,
    save_path='ImageNet_Attack_generation/',
    batch_size=64,
    CAM_layer=28,
    GPU_Config=["1","0"],
    save_method='ImageNet'
    save_visualization_base_path="./temp/"
)
```

输出存储

项目会默认生成对应名称的结果位置如下：

```
AISafety
   EvalBox
   Models
   utils
   test
       testimport.py
       testimport_black.py
       ImageNet_Attack_generation
            attack_param_FGSM_fgsm_01
            attack_param_FGSM_fgsm_02  # 对应于攻击算法参数
                Image
                    Adv_Images  # 存储对抗样本
                        Adv_ILSVRC2012_val_00000001.JPEG
```

(下页继续)

(续上页)

```
                    Adv_ILSVRC2012_val_00000002.JPEG
                    Adv_....JPEG
                adv_preds_val_4.txt
        temp
            **result.txt** # 用于存储评测结果
  Datasets
```

评测结果

result.txt 中存储的评测计算结果形如:

```
{
    "table_list": {
        "FGSM": {
            "fgsm_01": [["ACAC", "ACC", "ACAC", "ACTC"], [0.8877955079078674, 0.9, 0.
↪26380638033151627, 0.025149270360998344]],
            "fgsm_02": [["ACAC", "ACC", "ACAC", "ACTC"], [0.8242363157095732, 0.9, 0.
↪23616972751915455, 0.014533321653289022]]
        }
    }
}
```

这里是使用 Json 格式的存储。格式说明如下:

第一个字典关键字是 FGSM, 表示攻击的方法, 第二个是 fgsm_01, 表示配置的参数文件对应的名称, "ACAC", "ACC" 等表示的是评测函数的选取，后面顺序相同的对应的是该方法的评测值

"0.8877955079078674, 0.9", 这里设置的评测方法是 "ACTC", 对应的就是最后一个值 0.025149270360998344

攻击后样本

攻击后样本，如果选择的保存样本方式为

```
save_method='ImageNet'
```

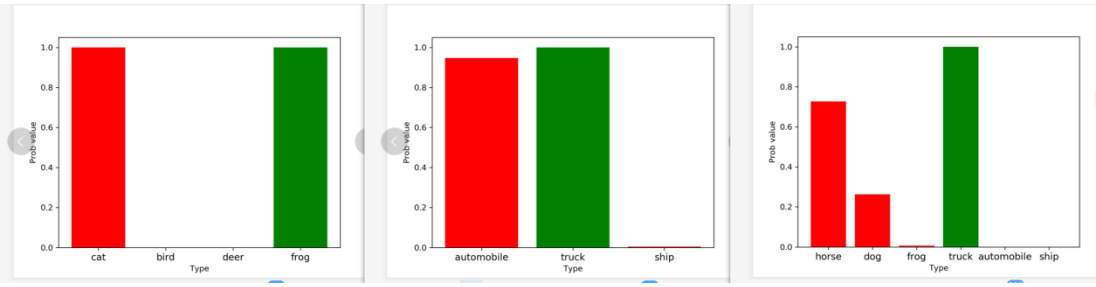
而非

```
save_method='.npy'
```

则对抗样本将以图像形式，存储在先前设置的 "Adv_Images" 中

攻击前后模型 topk 对比

模型评测的同时，将就模型预测 topk 类别做统计，并给出如下图像：



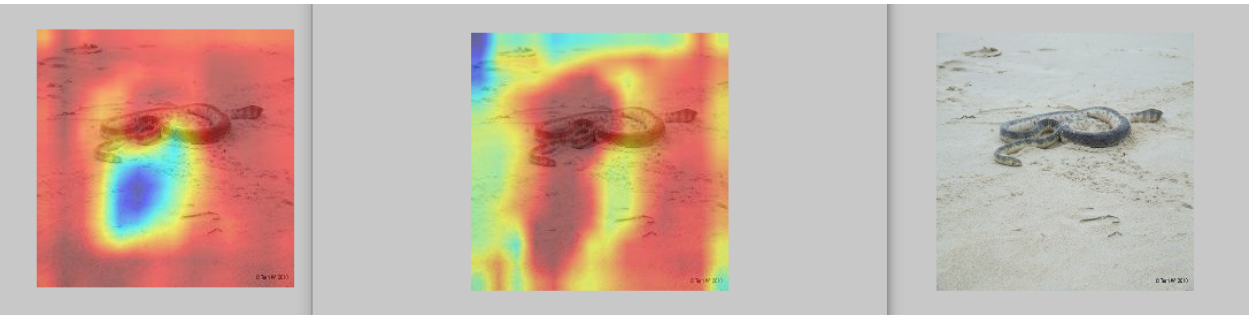
攻击前后热力图敏感区域可视化

模型评测的同时，将生成模型关注度的热力图，并将按照

1. 原始图像，OriginSample_ 模型名 _orig_ 图像序号.jpg
2. 攻击图像，AttackSample_ 模型名 _orig_ 图像序号.jpg
3. 原始图像热力图，OriginSample_ 模型名 _cam_ 图像序号.jpg
4. 攻击图像热力图，AttackSample_ 模型名 _cam_ 图像序号.jpg

的形式，一张输入样本，对应输出四张可视化图像。

目前的热力图方法只支持 grad_cam 的方式，指定显示模型的某一层到图像上，默认结果和热力图设置和保存成 224X224 的，可以供用户作为可解释性分析的一种方式。



1.8.2 参考示例 2（黑盒攻击）

参数设置

testimport_black.py 中 parser 设置，arg1 的：

```
(
    attack_method=["FGSM"],
```

(下页继续)

(续上页)

```
    Data_path=
    ["../Datasets/CIFAR_cln_data/cifar10_30_origin_inputs.npy", "../Datasets/CIFAR_cln_
↪data/cifar10_30_target_labels.npy",
    "../Datasets/CIFAR_cln_data/cifar10_30_origin_inputs.npy", "../Datasets/CIFAR_cln_
↪data/cifar10_30_origin_labels.npy"],
    Dict_path="./dict_lists/cifar10_dict.txt",
    model_dir='../Models/weights/FP_ResNet20.th',
    model='Models.UserModel.FP_resnet',
    defense_model='Models.UserModel.ResNet2',
    model_defence_dir='../Models/weights/resnet20_cifar.pt',
    IS_COMPARE_MODEL=True,
    IS_TARGETTED=True,
    data_type='cifar10',
    IS_WHITE=True,
    IS_PYTHORCH_WHITE=False,
    IS_DOCKER_BLACK=True,
    ONLY_GENRATE_BLACK_SAMPLE=False,
    IS_SAVE=False,
    black_Result_dir="../Datasets/adv_data/zjx.json",
    Scale_ImageSize=(32,32),   (高，宽)
    Crop_ImageSize=(32,32),(高，宽)
    batch_size=64,
    save_path='./Attack_generation/',
    GPU_Config=["1","0"],
    save_method='.npy'
)
```

testimport_black.py 的 parser 设置，arg2 的：

```
    data_type='cifar10',
    evaluation_method='CCV',
    IS_COMPARE_MODEL=True,
    IS_TARGETTED=True,
    IS_PYTHORCH_WHITE=False,
    CAM_layer=12
    save_visualization_base_path="./temp/"
)
```

输出存储

项目会默认生成对应名称的结果位置如下：

```
AISafety
  EvalBox
  Models
  utils
  test
     temp
         FGSM
            fgsm_01
               topk
                  top_3_0_FGSM_ Models.UserModel.FP_resnet_cifar10.jpg
                  top_3_1_FGSM_ Models.UserModel.FP_resnet_cifar10.jpg
                  top_3_..._FGSM_ Models.UserModel.FP_resnet_cifar10.jpg
                  top_3_0_FGSM_ Models.UserModel.ResNet2_cifar10.jpg
                  top_3_..._FGSM_ Models.UserModel.ResNet2_cifar10.jpg
                  ....jpg
         result.txt
     Attack_generation
  Datasets
```

评测结果

result.txt 中存储的评测计算结果形如：

```
{
    "table_list": {
        "FGSM": {
            "fgsm_01": [["ACAC", "ACC", "ACAC", "ACTC"], [0.8877955079078674, 0.9, 0.
↪26380638033151627, 0.025149270360998344]],
            "fgsm_02": [["ACAC", "ACC", "ACAC", "ACTC"], [0.8242363157095732, 0.9, 0.
↪23616972751915455, 0.014533321653289022]]}
    }
}
```

这里是使用 Json 格式的存储。格式说明如下：

第一个字典关键字是 FGSM，表示攻击的方法，第二个是 fgsm_01，表示配置的参数文件对应的名称，“ACAC”，“ACC” 等表示的是评测函数的选取，后面顺序相同的对应的是该方法的评测值

“0.8877955079078674, 0.9”，这里设置的评测方法是 “ACTC”，对应的就是最后一个值 0.025149270360998344

每测评一次会在文件中按照攻击方法–> 方法配置文件名字–> 测评方法–> 结果值去保存

如果不想被之前的结果干扰，用户可以手动删除，只生成当前的结果即可。

攻击前后模型 topk 对比

这里 topk 默认是 3，会默认生成对应名称的结果，表示攻击前后模型对样本预测的概率前 3 的分类结果的一个柱状图比较。

对应的命名方式例如：top_3_0_FGSM_Models.UserModel.FP_resnet_cifar10.jpg

表示 top3 在第 0 个样本攻击方法 FGSM 使用的模型文件 Models.UserModel.FP_resnet 在数据集 cifar10 的一个柱状分类前三的结果图

项目默认保存的是样本数据的前 50%，用户可以通过修改 testimport.py 中 Save_Eval_Visualization_Result() 函数的 topl_show_list 参数，此参数是一个列表，用户可手动赋值，也可以按照自己的需求生成要保存的数据的 index 列表

```
topk_show_list = [0,1]
topl_show_list=topk_show_list 设置要保存的列表
```

CHAPTER 2

数据集介绍

本平台目前已集成的数据集有 Cifar10 数据、ImageNet 数据集。

2.1 数据集介绍

目前本项目已集成两种数据集，分别是 Cifar10 数据集，ImageNet 数据集及 ImageCustom 数据集。

2.1.1 Cifar10 数据集

目前项目提供的 Cifar10 数据集，采用的是 npy 格式存储，特点是共有 10 个类别，且保证均匀分布，每种类别数目一致。用户也可以自行生成，但是要求 label 的格式是 one_vector 类型如下：

```
ys_pred_adv [0 0 0 1 0 0 0 0 0 0]
ys_pred_adv [1 0 0 0 0 0 0 0 0 0]
ys_pred_adv [0 0 0 0 0 1 0 0 0 0]
...
ys_pred_adv [0 0 1 0 0 0 0 0 0 0]
```

Cifar10 数据集下载链接

```
http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
```

Cifar100 数据集下载链接

```
http://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
```

npy 格式生成

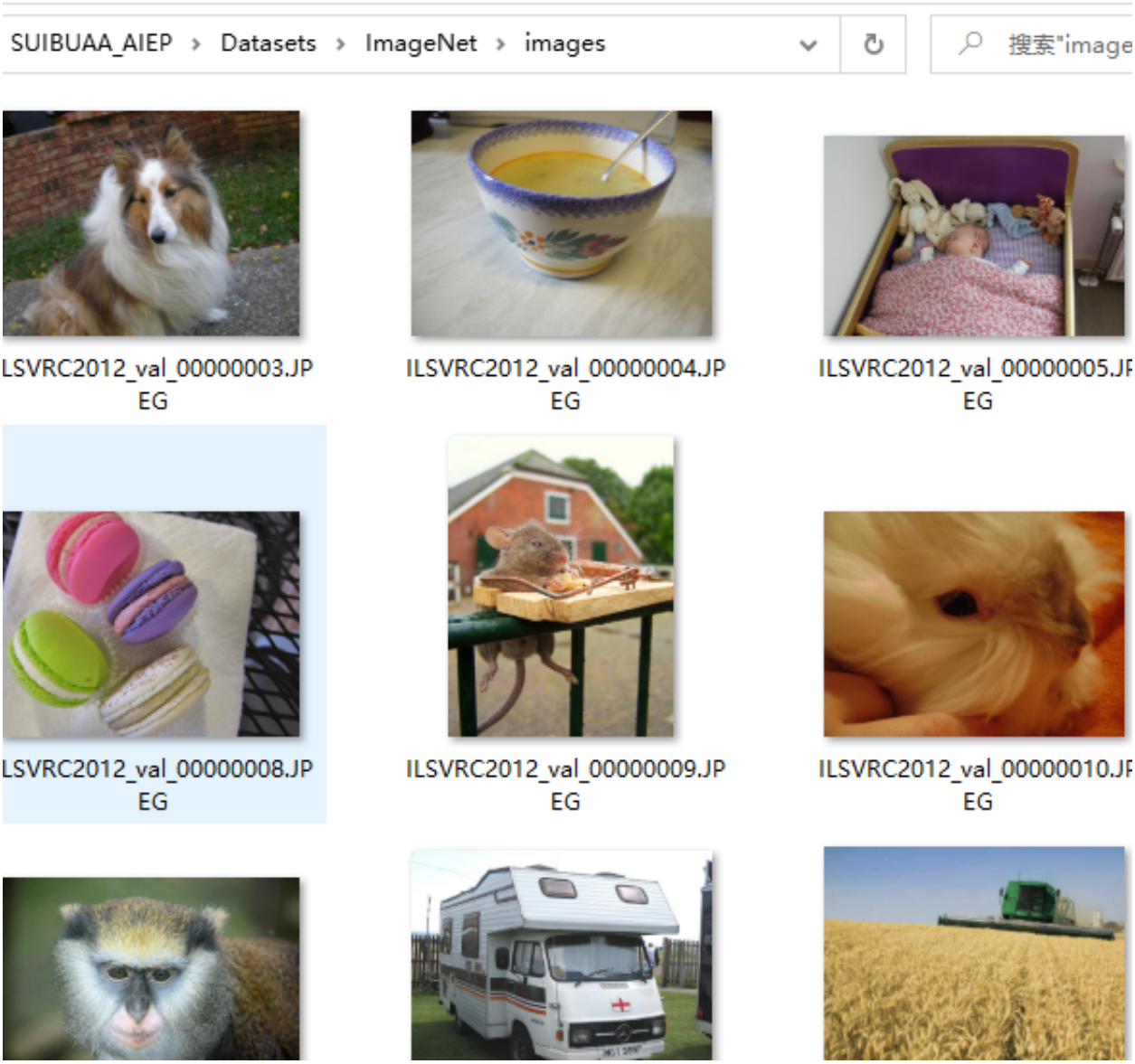
npy 生成的方式可以参考代码 cifartonpy.py：

```
AISafety
   EvalBox
   Models
   utils
   test
       testimport.py
       testimport_black.py
       cifartonpy.py
   Datasets
```

2.1.2 ImageNet 数据集

数据以图片方式保存，对应给出一个图像样本名称和类别号的文件用于做输入

ImageNet 数据集图像



ImageNet 数据集标签文件

以 ~/AISafety/Datasets/ImageNet/val_10.txt 文件为例：

```
ILSVRC2012_val_00000001.JPEG 65
ILSVRC2012_val_00000002.JPEG 970
ILSVRC2012_val_00000003.JPEG 230
...
ILSVRC2012_val_00000010.JPEG 109
```

ImageNet 数据集类别字典

ImageNet 数据集类别字典如下所示：

```
{
0: "tench Tinca tinca",
1: "goldfish. Carassius auratus",
...
27: "eft",
...
}
```

2.1.3 ImageCustom 数据集

该数据集用于商品攻击功能模块。数据集与 ImageNet 数据集格式类似。

2.2 扩展数据集

2.2.1 平台代码分布结构

```
  EvalBox
  Models
  utils
  **test**
      testimport.py
      testimport_black.py
      **cifartonpy.py**
      **dict_lists**
          cifar10_dict.txt
          ImageNet_12_dict.txt
  **Datasets**
      CIFAR_cln_data
         cifar10_30_origin_inputs.npy
         cifar10_30_origin_labels.npy
         cifar10_30_target_labels.npy
          ...
     ImageNet
         Images
         val_20.txt
```

(下页继续)

(续上页)

```
    ImageCustom
    adv_data
        zjx.json
```

上图为与扩展用户个人数据集相关的平台结构图。目前平台共支持两类数据集：

1. npy 格式数据集（与平台中 cifar10，cifar100 结构格式一致），包含原始样本，样本的标签。
2. 原始图像 + 图像标签 txt 文件的数据集。（与平台中 ImageNet 数据集格式一致）

需要注意的是，上述两类数据集，均需要额外提供对应的类别编号对应的目标名称的字典

2.2.2 扩展 npy 数据集

平台目前提供了一个，将 cifar 数据集转换为 npy 数据格式的样例文件 cifartonpy.py。

转换 Cifar-10 数据集

获得 Cifar-10 数据集的 train 数据 Xtr，Ytr；test 数据 Xte，Yte：

```
Xtr, Ytr, Xte, Yte=load_CIFAR10('../../cifar-10-python/cifar-10-batches-py')
```

保存 Cifar-10 的 test 数据的随机 1500 个，方式是随机均匀取 10 类，各 150。这里的标签是原始的 Groundtruth 标签,IsTargeted=False

```
save_numpy( Xte, Yte,'../Datasets/CIFAR_cln_data/',1500,shuff="random_equally",
↪datasetType="cifar10",IsTargeted=False)
```

如果 IsTargeted=True，则是随机生成和原始样本的 GroundTruth 不一致的标签，可以用于目标攻击使用，用户也可以自行定义目标标签的生成类别规则。

其中’../../cifar-10-python/cifar-10-batches-py’ 是原始的 cifar10 数据下载下来的返回值 Xtr, Ytr, Xte, Yte 分别是 cifar10 的 train 数据 Xtr,Ytr,cifar10 的 test 数据。

```
  调用 save_numpy(
  X      # 全部的数据
, Y      # 全部的数据
, path   #npy 数据保存的目标路径
, number=10000    # 最后的要保存的数据的数量，<= 总的
, shuff="random_equally" # 随机选取的类别数量要均衡的方式
, datasetType="cifar10"  # 数据集的名称
, IsTargeted=False    # 是否生成的是用于目标攻击的标签，随机值（和原始的不同即可）
)
```

转换 Cifar-100 数据集

```
numbertest=10000
Xte100, Yte100=load_CIFAR100('../Datasets/CIFAR10/cifar-100-python','test',numbertest)
save_numpy( Xte100, Yte100,'../Datasets/cln_data/',300,shuff="random_equally",
↪datasetType="cifar100",IsTargeted=False)
```

CHAPTER 3

攻击算法介绍

这一部分主要说明平台中已集成的攻击算法，以及如何扩展用户个人攻击算法。

3.1 攻击算法详细介绍

平台已集成的攻击算法及简要说明如下所示 (使用攻击算法缩写字典序排序)

3.1.1 名词解释

攻击方法（黑白盒）

白盒攻击

项目的部分攻击算法是白盒的攻击方式，需要完整的知道 model 的结构和对应的梯度等信息。

黑盒攻击

项目的部分攻击算法是黑盒的攻击方式，不需要完整的知道 model 的结构和对应的梯度等信息，只需要知道经过该模型的预测结果，用于作为评测的数据输入。

攻击方法（目标/非目标）

目标攻击

目标攻击是指将原始样本通过攻击后，指定攻击后的结果类别

非目标攻击

非目标攻击是指将原始样本通过攻击后，结果类别与原有的模型类别不同即可

3.1.2 BA 算法

算法介绍

BA 的全称是 Boundary Attack。该算法使用目标类中的一个样本初始化为非目标攻击，并用一个混合了均匀噪声的样本初始化为目标攻击。算法的每次迭代有三个部分。首先，通过二进制搜索将上次迭代的迭代结果推向边界。

Algorithm 1 Binary-search

Require: Upper bound x_u, lower bound x_l, threshold ξ.
Require: Binary function ϕ, s.t. $\phi(x_u) = 1, \phi(x_l) = 0$.
Require: Constraint L_p, where $p = 2$ or ∞.
  function BINARY-SEARCH(x_u, x_l, ξ, ϕ, p)
    if $\|x_u - x_l\|_p \leq \xi$ **then**
      return x_u.
    else
      Set $x_m = \begin{cases} \frac{1}{2}x_u + \frac{1}{2}x_l, & \text{if } p = 2, \\ \text{Clip}_{x_l, 0.5\|x_u - x_l\|_\infty} x_u, & \text{if } p = \infty. \end{cases}$
      if $\phi(x_m) = 1$ **then**
        BINARY-SEARCH(x_m, x_l, ξ, ϕ, p)
      else
        BINARY-SEARCH(x_u, x_m, ξ, ϕ, p)
      end if
    end if
  end function

接着，通过如下方程估计梯度方向。

$$\widetilde{\nabla S}(x^t, \delta) := \frac{1}{B} \sum_{b=1}^{B} \phi_x(x^t + \delta_{u_b}) u_b$$

最后通过几何级数选择合适的步长，直到扰动成功。并通过二元搜索将扰动样本推回边界。

参数说明

参数名称	参数说明
epsilon	扰动的步长系数
delta	重新缩放的扰动的尺寸
lower_bound	归一化数据上边界
upper_bound	归一化数据下边界
max_iter	扰动样本更新的最大内层迭代次数
binary_search_steps	搜索 ϵ 的迭代次数
batch_size	单次批处理大小
step_adapt	用来更新 delta 的更新系数
sample_size	过程中生成的潜在扰动样本的采样数目
init_size	初始化的随机样本数目

3.1.3 BIM 算法

算法介绍

BIM 全称为 Basic Iterative Method。FGSM 这种 one-step 方法通过一大步运算增大分类器的损失函数而进行图像扰动，因而可以直接将其扩展为通过多个小步增大损失函数的变体，从而我们得到 Basic Iterative Methods（BIM）。BIM 是 FGSM 的拓展，进行了多次小步的迭代，并且在每一步之后都修剪得到的结果的像素值，来确保得到的结果在原始图像的 ϵ 邻域内，BIM 迭代生成对抗样本的公式如下：

$$X_0^{adv} = X,\ \ X_{N+1}^{adv} = Clip_{X,\epsilon}\{X_n^{adv} + \alpha sign(\nabla_X J(X_N^{adv}, y_{true}))\}$$

参数说明

参数名称	参数说明
eps	原始样本的灰度偏移比例
eps_iter	梯度步长的改变比例
num_steps	多次迭代的次数

3.1.4 BLB 算法

算法介绍

BLB 的全称是 Box-constrained L-BFGS attack。该攻击方法通过对图像添加小量的人类察觉不到的扰动误导神经网络做出误分类：

$$min_\rho ||\rho||_2\ s.t. C(I_c + \rho) = l;\ I_c + \rho \in [0,1]^m$$

其中 Ic R^m 表示一张干净的图片，ρ R_m 是一个小的扰动，I 是图像的 label，C(···) 是深度神经网络分类器。l 和原本图像的 label 不一样。

但由于问题的复杂度太高，转而求解简化后的问题，即寻找最小的损失函数添加项，使得神经网络做出误分类，这就将问题转化成了凸优化过程。

$$min_\rho\ c|\rho| + L(I_c + \rho, l)\ \ s.t. I_c + \rho \in [0,1]^m$$

L(.,.) 计算分类器的 loss。通过凸优化的手段得到最终的噪音。

参数说明

参数名称	参数说明
init_const	二分搜索的参数的初始值，用于初始扰动图像的数据产生
binary_search_steps	二分搜索的一次循环步长
max_iter	使用 LBFGS，torch 的优化器优化扰动值的最大迭代次数

3.1.5 Corrupt 算法

算法介绍

Corrupt 算法将自然噪声加入到原始样本后所产生的攻击样本，用于评测模型的鲁棒性。其中，自然噪音（corruption）是指经常出现在自然场景中且对模型的任务产生一定不良影响的噪音，如：高斯噪音、强烈的对比度变化、雪、雾等。包含：

gaussian_noise 高斯噪声

shot_noise 散粒噪声（泊松噪声）

impulse_noise 脉冲噪声

speckle_noise 斑点噪声

gaussian_blur 高斯模糊

glass_blur 毛玻璃

defocus_blur 散焦模糊

motion_blur 运动模糊

zoom_blur 缩放模糊

Fog 烟雾

Frost 水雾

Snow 雪

Spatter 喷溅

Contrast 对比度噪声

Brightness 过曝光

Saturate 饱和

jpeg_compression jpeg 压缩产生的损失

Pixelate 像素化

elastic_transform 弹性变换

参数说明

参数名称	参数说明
severtiy	选择待选参数，1 是第一个，2 就是第二个，暂时不开放给用户
gama	决定迭代次数的和图像尺度大小相关的一个比例系数

3.1.6 CW2 算法

算法介绍

CW2 算法的全称是 Carlini & Wagner Attack。Carlini 和 Wagner 为了攻击防御性蒸馏 (Defensive distillation) 网络提出了三种对抗攻击方法，通过限制 l0,l1,l∞ 范数使得扰动无法被察觉。实验证明蒸馏网络完全无法防御这三种攻击。该算法生成的对抗扰动可以从 unsecured 网络迁移到 secured 网络上，从而实现黑箱攻击。实验表明，C&W 攻击方法能有效攻击现有的大多数防御方法。

目标函数表示为:

$$min_{\delta} D(x, x+\delta) + c \cdot f(x+\delta) \, subject \, to \, x+\delta \in [0,1]$$

式中，δ 是对抗扰动；D(,) 表示 L0、L2 或 L∞ 距离度量；f(x +δ) 是自定义的对抗损失，当且仅当 DNN 的预测为攻击目标时才满足 f(x +δ) 0。为了确保 x + δ 产生能有效的

图像（即 x +δ [0, 1]），引入了一个新变量来代替 δ:

$$\delta = \frac{1}{2}(tanh(K)+1) - x$$

这样，x + δ =1/2(tanh(k) + 1) 在优化过程中始终位于 [0, 1] 中。除了在 MNIST、CIFAR10 和 ImageNet 的正常训练 DNN 模型上获得 100% 的攻击成功率外，C&W 攻击还可以破坏防御性蒸馏模型，而这些模型可以使 L-BFGS 和 Deepfool 无法找到对抗性样本。

参数说明

参数名称	参数说明
dataset	使用的数据集的名称
class_type_num	分类网络的类别
kappa	标签的序号的整体偏移量
learning_rate	迭代过程中优化器的学习率
init_const	初始的迭代求解参数的值
lower_bound	产生中间扰动样本的值的下边界
upper_bound	产生中间扰动样本的值的上边界
max_iter	为了生成合适的扰动样本时候的迭代次数
binary_search_steps	为了求解合适的参数的搜索迭代次数

3.1.7 Deepfool 算法

算法介绍

对于多分类问题，通常采取的方案为一对多。在这里，针对多个输出类别，通过下式进行分类选择：

$$\hat{k}(x) = arg\underset{k}{max} f_k(x)$$

对于线性多分类器，我们有：

$$f(x) = W^T x + b$$

最小扰动可以由下式计算：

$$\underset{r}{arg\,min}\,||r||_2$$
$$s.t.\ \exists k : w_k^T(x_0 + r) + b_k \geq w_{\hat{k}(x_0)}^T(x_0 + r) + b_{\hat{k}(x_0)}$$

为了解这个问题，我们先来看一个四分类问题的例子：

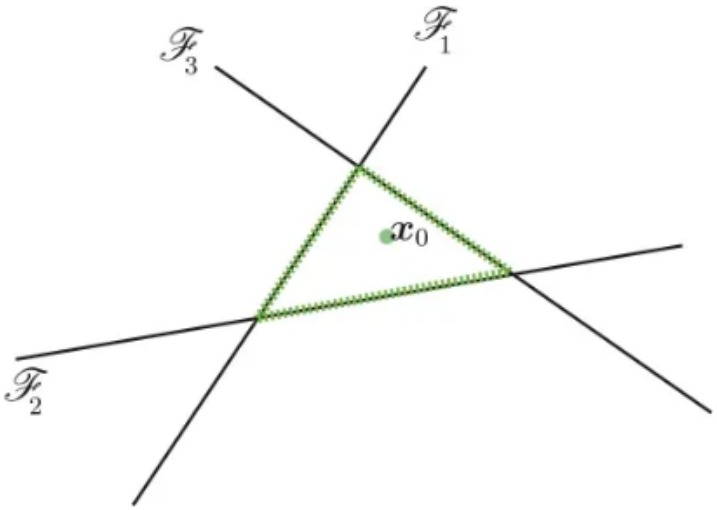


Figure 4: For $\boldsymbol{x}_0$ belonging to class 4, let $\mathscr{F}_k = \{\boldsymbol{x} : f_k(\boldsymbol{x}) - f_4(\boldsymbol{x}) = 0\}$. These hyperplanes are depicted in solid lines and the boundary of P is shown in green dotted line.

注意，这里只有三条线，分别对应前三类的参数超平面与第四类相减得到的参数超平面。同样利用点到直线的距离公式，若求得到这三条线的最短距离便可得到使样本分类发生变化的最小扰动长度。最短距离可以用下式计算：

$$\hat{l}(x_0) = \underset{k \neq \hat{k}(x_0)}{arg\,min} \frac{\left| f_k(x_0) - f_{\hat{k}(x_0)}(x_0) \right|}{\left|\left| w_k - w_{\hat{k}(x_0)} \right|\right|_2}$$

因此最小扰动向量为：

$$r_*(x_0) = \frac{\left| f_{\hat{l}(x_0)}(x_0) - f_{\hat{k}(x_0)}(x_0) \right|}{\left|\left| w_{\hat{l}(x_0)} - w_{\hat{k}(x_0)} \right|\right|_2^2} (w_{\hat{l}(x_0)} - w_{\hat{k}(x_0)})$$

对于一般的多分类问题，同样利用近似线性的方法迭代得到，算法如下：

Algorithm 2 DeepFool: multi-class case

input: Image $\boldsymbol{x}$, classifier f.
output: Perturbation $\hat{\boldsymbol{r}}$.

Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
while $\hat{k}(\boldsymbol{x}_i) = \hat{k}(\boldsymbol{x}_0)$ **do**
    for $k \neq \hat{k}(\boldsymbol{x}_0)$ **do**
        $\boldsymbol{w}'_k \leftarrow \nabla f_k(\boldsymbol{x}_i) - \nabla f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
        $f'_k \leftarrow f_k(\boldsymbol{x}_i) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
    end for
    $\hat{l} \leftarrow \arg\min_{k \neq \hat{k}(\boldsymbol{x}_0)} \frac{|f'_k|}{\|\boldsymbol{w}'_k\|_2}$
    $\boldsymbol{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\boldsymbol{w}'_{\hat{l}}\|_2^2} \boldsymbol{w}'_{\hat{l}}$
    $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$
    $i \leftarrow i + 1$
end while
return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$

注意，这个算法并不能保证收敛到最小扰动解。算法中 2 范数可以扩展到 p 范数。

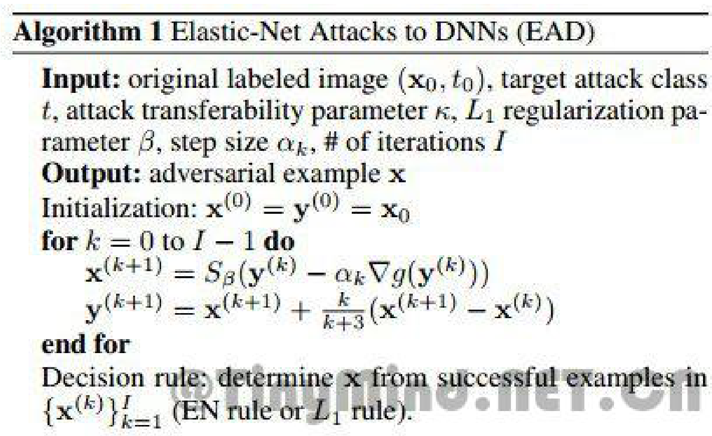
参数说明

参数名称	参数说明
overshoot	防止算法收敛到分类面上
max_iter	为了生成合适的扰动样本时候的迭代次数

3.1.8 EAD 算法

算法介绍

EAD 的全称是 Elastic-net Attacks to DNNs。EAD 将使用对抗样本攻击 DNN 的过程转化为了使用弹性网络正则化（elastic-net regularized）的优化问题。在这种表示下，当前最佳的 L2 范数攻击算法成为了本文方法的一个特例（在不考虑 L1 范数的情况下）。在 MNIST、CIFAR10 和 ImageNet 上的实验结果表明 EAD 算法可以生成具有很小 L1 失真的对抗样本，并且能在不同攻击场景中实现与当前最佳方法匹敌的攻击成功率。更重要的是，EAD 算法生成的对抗样本有着显著增强的攻击可迁移性，这为如何在对抗机器学习中使用 L1 范数失真以及增强 DNN 的安全性提供了全新的见解。下图是 EAD 算法的伪代码：



Algorithm 1 Elastic-Net Attacks to DNNs (EAD)

Input: original labeled image $(\mathbf{x}_0, t_0)$, target attack class t, attack transferability parameter κ, L_1 regularization parameter β, step size α_k, # of iterations I
Output: adversarial example $\mathbf{x}$
Initialization: $\mathbf{x}^{(0)} = \mathbf{y}^{(0)} = \mathbf{x}_0$
for $k = 0$ to $I - 1$ **do**
  $\mathbf{x}^{(k+1)} = S_\beta(\mathbf{y}^{(k)} - \alpha_k \nabla g(\mathbf{y}^{(k)}))$
  $\mathbf{y}^{(k+1)} = \mathbf{x}^{(k+1)} + \frac{k}{k+3}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)})$
end for
Decision rule: determine $\mathbf{x}$ from successful examples in $\{\mathbf{x}^{(k)}\}_{k=1}^{I}$ (EN rule or L_1 rule).

参数说明

参数名称	参数说明
lr	学习率
kapa	标签的偏移值
binary_search_steps	用来搜索合适的参数的迭代次数
init_const	初始化的调节系数
lower_bound	产生中间扰动样本的值的下边界
upper_bound	产生中间扰动样本的值的上边界
max_iter	扰动样本更新的最大内层迭代次数
class_type_number	分类类别的数目
beta	扰动样本的初始灰度波动值
EN	决策规则的选择

3.1.9 FGSM 算法

算法介绍

FGSM 的全称是 Fast Gradient Sign Method(快速梯度下降法)，在白盒环境下，通过求出模型对输入的导数，然后用符号函数得到其具体的梯度方向，接着乘以一个步长，得到的“扰动”加在原来的输入上就得到

了在 FGSM 攻击下的样本。

FGSM 的攻击表达如下：

$$x^{'} = x + \varepsilon \cdot sign(\nabla_x J(x, y))$$

攻击成功就是模型分类错误，就模型而言，就是加了扰动的样本使得模型的 loss 增大。而所有基于梯度的攻击方法都是基于让 loss 增大这一点来做的。可以仔细回忆一下，在神经网络的反向传播当中，我们在训练过程时就是沿着梯度方向来更新更新 w，b 的值。这样做可以使得网络往 loss 减小的方向收敛。

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$
$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b)$$

那么现在我们既然是要使得 loss 增大，而模型的网络系数又固定不变，唯一可以改变的就是输入，因此我们就利用 loss 对输入求导从而“更新”这个输入。

参数说明

参数名称	参数说明
epsilon	沿着梯度的步长系数

3.1.10 ILLC

算法介绍

ILLC 的全称是 Iterative Least Likely Class Attack，先介绍一下 one-Step 的方法。one-step target class methods:

$$X^{adv} = x - \epsilon sign(\nabla_X J(X, y_{target}))$$

其中

$$y_{target} = \underset{y}{argminp}(y|X)$$

即偏离最远的错误类。这里产生的攻击样本是其中一个候选样本，当只计算完成一个梯度的时候，一般是通过线性扰动的损失函数，而迭代方式可以应用更多的梯度更新，它们通常不依赖于模型的任何近似值，并且在进行更多迭代时通常会产生更多有效果和攻击性的对抗样本（图像），也就是说 ILLR 迭代方式是 one-Step 的“升级版”。

$$X_0^{adv} = X, X_{N+1}^{adv} = Clip_{X,\epsilon}(X_N^{adv} - \alpha sign(\nabla_X J(X_N^{adv}, y_{target})))$$

损失函数是：

$$Loss = \frac{1}{(m-k)+\lambda k}\left(\sum_{i \in CLEAN} L(X_i|y_i) + \lambda \sum_{i \in ADV} L(X_i^{adv}|y_i)\right)$$

其中 L(X|y) 是单个样本 X 对于真实标记 y 的损失函数，m 是小批量上的训练样本总数，k 是小批量内对抗样本的数目而 λ 是参数来控制对抗样本对于损失函数的权重。

算法过程：

Algorithm 1 Adversarial training of network N.
Size of the training minibatch is m. Number of adversarial images in the minibatch is k.

1: Randomly initialize network N
2: **repeat**
3: Read minibatch $B = \{X^1, \ldots, X^m\}$ from training set
4: Generate k adversarial examples $\{X^1_{adv}, \ldots, X^k_{adv}\}$ from corresponding clean examples $\{X^1, \ldots, X^k\}$ using current state of the network N
5: Make new minibatch $B' = \{X^1_{adv}, \ldots, X^k_{adv}, X^{k+1}, \ldots, X^m\}$
6: Do one training step of network N using minibatch B'
7: **until** training converged

参数说明

参数名称	参数说明
epsilon	样本归一化的偏移比例
epsilon_iter	沿着梯度的步长系数
num_steps	迭代次数

3.1.11 JSM

算法介绍

JSM 算法的全称是 Jacobian-based Saliency Map Attack

目标是只修改图像中的几个像素，而不是扰乱整个图像来欺骗分类器，该算法一次修改一个干净图像的像素，并监测变化对结果分类的影响。通过使用网络层的输出的梯度来计算一个显著性图来执行监控。

JSMA 算法主要包括三个过程：计算前向导数，计算对抗性显著图，添加扰动，以下给出具体解释。

所谓前向导数，其实是计算神经网络最后一层的每一个输出对输入的每个特征的偏导。以 MNIST 分类任务为例，输入的图片的特征数（即像素点）为 784，神经网络的最后一层一般为 10 个输出（分别对应 0-9 分类权重），那对于每一个输出我们都要分别计算对 784 个输入特征的偏导，所以计算结束得到的前向导数的矩阵为（10，784）。前向导数标识了每个输入特征对于每个输出分类的影响程度，其计算过程也是采用链式法则。这里需要说明一下，前面讨论过的 FGSM 和 DeepFool 不同在计算梯度时，是通过对损失函数求导得到

的，而 JSMA 中前向导数是通过对神经网络最后一层输出求导得到的。前向导数 F(X) 具体计算过程如下所示，j 表示对应的输出分类，i 表示对应的输入特征。

$$\nabla F(X) = \frac{\partial F(X)}{\partial X} = \left[\frac{\partial F_j(X)}{\partial x_i}\right]_{i \in 1...M, j \in 1...N}$$

$$\frac{\partial F_j(X)}{\partial x_i} = \left(W_{n+1,j} \cdot \frac{\partial H_n}{\partial x_i}\right) \times \frac{\partial f_{n+1,j}}{\partial x_i}\left(W_{n+1,j} \cdot H_n + b_{n+1,j}\right)$$

通过得到的前向导数，我们可以计算其对抗性显著图，即对分类器特定输出影响程度最大的输入。首先，根据扰动方式的不同（正向扰动和反向扰动），作者提出了两种计算对抗性显著图的方式，即：

$$S(X,t)[i] = \begin{cases} 0 & if\ \frac{\partial F_t(X)}{\partial X_i} < 0\ or\ \sum_{j \neq t} \frac{\partial F_j(X)}{\partial X_i} > 0 \\ \left(\frac{\partial F_t(X)}{\partial X_i} \left|\sum_{j \neq t} \frac{\partial F_j(X)}{\partial X_i}\right|\right) & otherwise \end{cases}$$

但是在文章中第四部分的应用中作者发现，找到单个满足要求的特征很困难，所以作者提出了另一种解决方案，通过对抗性显著图寻找对分类器特定输出影响程度最大的输入特征对，即每次计算得到两个特征。

$$argmax_{p1,p2}\left(\sum_{i=p1,p2} \frac{\partial F_t(X)}{\partial X_i}\right) \times \left|\sum_{i=p1,p2} \sum_{j \neq t} \frac{\partial F_j(X)}{\partial X_i}\right|$$

算法具体过程：

Algorithm 3 Increasing pixel intensities saliency map
$\nabla \mathbf{F}(\mathbf{X})$ is the forward derivative, Γ the features still in the search space, and t the target class

Input: $\nabla \mathbf{F}(\mathbf{X})$, Γ, t

for each pair $(p, q) \in \Gamma$ **do**
  $\alpha = \sum_{i=p,q} \frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_i}$
  $\beta = \sum_{i=p,q} \sum_{j \neq t} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_j}$
  if $\alpha > 0$ and $\beta < 0$ and $-\alpha \times \beta >$ max **then**
    $p_1, p_2 \leftarrow p, q$
    $max \leftarrow -\alpha \times \beta$
  end if
end for
return p_1, p_2

参数说明

参数名称	参数说明
theta	样本一维度化之后（拉平）对图像的灰度的增减。小于 0 是降低，大于 0 是增加。
gama	决定迭代次数的和图像尺度大小相关的一个比例系数

3.1.12 LLC 算法

算法介绍

LLC 算法的全称为 Least-Likely-Class Iterative Methods。

one-step 方法通过一大步运算增大分类器的损失函数而进行图像扰动，因而可以直接将其扩展为通过多个小步增大损失函数的变体，从而我们得到 Basic Iterative Methods（BIM）。而该方法的变体和前述方法类似，通过用识别概率最小的类别（目标类别）代替对抗扰动中的类别变量，而得到 Least-Likely-Class Iterative Methods。

$$y_{LL} = \underset{y}{arg\,min}\{p(y|X)\}$$

这里 X 是原始的图像，攻击样本：

$$X_0^{adv} = X,\ X_{N+1}^{adv} = Clip_{X,\epsilon}\{X_N^{adv} - \alpha sign(\nabla_X J(X_N^{adv}, y_{LL}))\}$$

参数说明

参数名称	参数说明
eps	沿着梯度的步长系数

3.1.13 NES 算法

算法介绍

NES 算法的全称为 Nature Evolutionary Strategies

参数说明

参数名称	参数说明
learning_rate	学习率
lower_bound	产生中间扰动样本的值的下边界
upper_bound	产生中间扰动样本的值的上边界
max_iter	扰动样本更新的最大内层迭代次数
binary_search_steps	用来搜索合适的参数的迭代次数
batch_size	单次批处理数目
kappa	标签的序号的整体偏移量
sigma	随机样本分布的标准差
class_type_number	分类类别的数目
confidence	帮助判断攻击类别和预测类别是否相同或者有固定偏差
epsilon	扰动的步长系数

3.1.14 OM 算法

算法介绍

OM 算法的全称为 OPTMARGIN attack，它可以生成低失真的对抗示例，对小扰动具有鲁棒性，例如在区域分类中使用的小扰动。

在 OPTMARGIN 攻击中，创建了区域分类器的替代模型，该模型干扰输入点比较少。

这里 f 是用来区域分类的点分类器，vi 是作用在输入样本 x 上的扰动。该攻击使用现有的优化攻击技术来生成示例，愚弄整个过程的同时最大程度地减少其失真。

$$l_i(x^{'}) = l(x^{'} + v_i) = max(-k, Z(x^{'} + v_i)_y - max\{Z(x^{'} + v_i)_j : j \neq y\})$$

这里 k=0，这意味着，只要能产生错分类，就可以接受。通过这些损失方程，作者扩展了 Carlini & Wagner 的 L2 攻击，有：

$$minimize||x^{'} - x||_2^2 + c \cdot (l_1(x^{'}) + ... + l_n(x^{'}))$$

作者选用了 20 种分类器，在攻击过程中，v1,v2,⋯,v19，都是量级归一化到 ϵ 上的随机正交向量，v20=0, 此选择是为了使随机扰动位于 vi 中，为了稳定优化器，在攻击过程中，需要固定 vi，这个办法在 C&W 算法中仍然使用过，详细的描述见：https://openreview.net/pdf?id=BkpiPMbA-

参数说明

参数名称	参数说明
lr	学习率
kapa	标签的偏移值
binary_search_steps	用来搜索合适的参数的迭代次数
init_const	初始化的调节系数
lower_bound	产生中间扰动样本的值的下边界
upper_bound	产生中间扰动样本的值的上边界
max_iter	扰动样本更新的最大内层迭代次数
class_type_number	分类类别的数目
noise_count	中间扰动样本的数目，初始都是随机的正交向量
magnitude	扰动样本的归一化的正交向量的幅值

3.1.15 PGD 算法

算法介绍

PGD 全称是 Projected Gradient descent。目的是为解决 FGSM 和 FGM 中的线性假设问题，使用 PGD 方法来求解内部的最大值问题。PGD 是一种迭代攻击，相比于普通的 FGSM 和 FGM 仅做一次迭代，PGD 是做多次迭代，每次走一小步，每次迭代都会将扰动投射到规定范围内。

$$g_t = \nabla X_t(L(f_\theta(X_t), y))$$

gt 表示 t 时刻的损失关于 t 时刻输入的梯度。

$$X_{t+1} = \prod_{X+S}(X_t + \varepsilon(\frac{g_t}{||g_t||}))$$

t+1 时刻输入根据 t 时刻的输入及 t 时刻的梯度求出。_(X+S) 的意思是，如果扰动超过一定的范围，就要映射回规定的范围 S 内。

由于每次只走很小的一步，所以局部线性假设基本成立。经过多步之后就可以达到最优解，也就是达到最强的攻击效果。同时使用 PGD 算法得到的攻击样本，是一阶对抗样本中最强的。这里所说的一阶对抗样本是指依据一阶梯度的对抗样本。如果模型对 PGD 产生的样本鲁棒，那基本上就对所有的一阶对抗样本都鲁棒。实验也证明，利用 PGD 算法进行对抗训练的模型确实具有很好的鲁棒性。

PGD 虽然简单，也很有效，但是存在一个问题是计算效率不高。不采用提对抗训练的方法 m 次迭代只会有 m 次梯度的计算，但是对于 PGD 而言，每做一次梯度下降（获取模型参数的梯度，训练模型），都要对应有 K 步的梯度提升（获取输出的梯度，寻找扰动）。所以相比不采用对抗训练的方法，PGD 需要做 m(K+1) 次梯度计算。

参数说明

参数名称	参数说明
eps	原始样本的灰度偏移比例
eps_iter	梯度步长的改变比例
num_steps	迭代次数

3.1.16 RFGSM 算法

算法介绍

RFGSM 算法全称是 RAND-FGSM (R-FGSM)

使用 FGSM 方法进行对抗训练后的神经网络模型，在面对白盒攻击时比黑盒攻击更为鲁棒，所以提出了 R-FGSM 增加随机梯度训练，用于防御对抗训练。

$$x_{tmp} = x + \alpha \cdot sign(N(0^d, I^d))$$
$$x^{'} = x_{tmp} + (\epsilon - \alpha) \cdot sign(\nabla_{x_{tmp}} J(x_{tmp}, l))$$

其中 α 和　是参数，且 $\alpha <$ 。

参数说明

参数名称	参数说明
epsilon	沿着梯度方向步长的参数，当该值越大时，每次为样本添加的扰动更大，对抗攻击强度越高。
alpha	梯度步长的改变比例。

3.1.17 RLLC 算法

算法介绍

Random Least Likely Class Attack

参数说明

参数名称	参数说明
epsilon	沿着梯度方向步长的参数，当该值越大时，每次为样本添加的扰动更大，对抗攻击强度越高。
alpha	调节梯度步长系数的比例系数

3.1.18 SPSA 算法

算法介绍

SPSA 算法全称是 Multivariate stochastic approximation using a simultaneous perturbation gradient approximation

SPSA 算法非常适合于高维优化问题，即使在不确定目标的情况下，我们也可使用 SPSA 公式来产生对抗性攻击。在 SPSA 算法中，首先从 Rademacher 分布（即 Bernoulli±1）中抽取一批 n 个样本，即

$$v1, ..., vn \ \{1, -1\}^D$$

然后用随机方向上的有限差分估计逼近梯度。具体来说，对于第 i 个样本，估计的梯度 gi 计算如下：

$$g_i = \frac{f(x_t + \delta v_i) - f(x - \delta v_i)}{2\delta v_i}$$

式中，δ 是扰动大小，xt 是第 t 次迭代时的扰动图像，f 是要评估的模型。最后，SPSA 对估计的梯度进行聚合，并在输入文本上执行投影梯度下降。整个过程按预先确定的迭代次数进行迭代。

完整的伪代码如下：

Algorithm 1 SPSA adversarial attack

Input: function to minimize f, initial image $x_0 \in \mathbb{R}^D$, perturbation size δ, step size $\alpha > 0$, batch size n
for $t = 0$ **to** $T - 1$ **do**
  Sample $v_1, \ldots, v_n \sim \{1, -1\}^D$
  Define $v_i^{-1} = [v_{i,1}^{-1}, \ldots, v_{i,D}^{-1}]$
  Calculate $g_i = (f(x_t + \delta v_i) - f(x - \delta v_i))v_i^{-1}/(2\delta)$
  Set $x_t' = x_t - \alpha(1/n)\sum_{i=1}^{n} g_i$
  Project $x_{t+1} = \arg\min_{x \in N_\epsilon(x_0)} \|x_t' - x_0\|$
end for

参数说明

参数名称	参数说明
alpha	沿着梯度扰动的步长系数
gamma	扰动的系数
c_par	迭代的基准噪声系数
a_par	用来调整 alpha 系数的更新系数
sizeN	扰动样本更新的最大内层迭代次数
min_vals_iter	最小的 loss 值下界
print_every	迭代过程中多少次迭代后打印一次
max_iter	扰动样本更新的最大外层迭代次数

3.1.19 UAP 算法

算法介绍

UAP 算法的全称是 Universal Adversarial Perturbation attack。

UAP 算法提出了一种不易察觉的万能 perturbation，它能使目前最好的分类器，在完成图片分类任务时出错。通过这个算法得到的 perturbation，在各种神经网络情况下都能取得很好的效果。这揭示了目前分类器分类“判定边界”在高维度上的几何关系。

万能 perturbation 的构造算法：

算法得到的 perturbation 需要满足两个条件：

$$||v||_p \leq \xi$$
$$\mathbb{P}_{x \sim u}(\hat{k}(x+v) \neq \hat{k}(x)) \geq 1-\delta$$

即第一是扰动的规模要比较小，第二是原来的数据叠加了扰动之后，分类器的输出错误率要大于一个阈值。整体的算法如下：

Algorithm 1 Computation of universal perturbations.

input: Data points X, classifier $\hat{k}$, desired ℓ_p norm of the perturbation ξ, desired accuracy on perturbed samples δ.
output: Universal perturbation vector v.
Initialize $v \leftarrow 0$.
while $\text{Err}(X_v) \leq 1 - \delta$ **do**
    for each datapoint $x_i \in X$ **do**
        if $\hat{k}(x_i + v) = \hat{k}(x_i)$ **then**
            Compute the *minimal* perturbation that sends $x_i + v$ to the decision boundary:

$$\Delta v_i \leftarrow \arg\min_r \|r\|_2 \text{ s.t. } \hat{k}(x_i + v + r) \neq \hat{k}(x_i).$$

            Update the perturbation:

$$v \leftarrow \mathcal{P}_{p,\xi}(v + \Delta v_i).$$

        end if
    end for
end while

其算法思想是对于图片数据中的每一个点，依次计算能使得最终分类器的输出错误的最小扰动，一直循环知道将整体分类错误的概率大于 $1 - \delta$，其中 δ 为人为定义的分类器准确度。而算法的关键不是为了找到一个能使大多数样本分类错误的最小 perturbation，而是用足够小的范数找到这样的一个扰动。

参数说明

参数名称	参数说明
dataset	使用的数据集类别

3.1.20 UMIFGSM 算法

算法介绍

UMIFGSM 算法全称是 Utargeted Momentum Iterative Fast Gradient Sign Method(UMI-FGSM)。

在迭代攻击方法中加入动量项（momentum term），提高对抗样本的转移性：

$$g_{t+1} = \mu \cdot g_t + \frac{\nabla_x J(x_t^{adv}, y)}{||\nabla_x J(x_t^{adv}, y)||_1}$$
$$x_{t+1}^{adv} = x_t^{adv} + \alpha \cdot sign(g_{t+1})$$

其中 gt 包含了直到 t 次迭代的梯度信息。

参数说明

参数名称	参数说明
epsilon	扰动的步长系数
eps_iter	调节扰动的步长的比例系数
num_step	扰动样本更新的迭代次数
decay_factor	调节动量项的步长

3.1.21 ZOO 算法

算法介绍

ZOO 算法全称为 Zeroth Order Optimization Based Black-box。

ZOO 攻击不可知，仅依赖于预测分数（例如类别机率或对数），使用数值估算梯度的预测。ZOO 算法利用正负扰动带来的概率差估算一阶导（梯度）和二阶导，再利用 ADAM 或者牛顿法等方法更新 x。本质为通过估算梯度将黑盒转换为白盒过程。

ZOO 算法的损失函数和 CW 相似：

$$minimize_x ||x - x_0||_2^2 + c \cdot f(x,t) \;\; subject\ to\ x \in [0,1]^p$$

损失函数如上，左边保证对抗样本与真实 input 的相似，右边保证对抗样本能导致目标模型出错，具体如下：

目标攻击下：

$$f(x,t) = max\{\underset{i \neq t}{max} log[F(x)]_i - log[F(x)]_t, -K\}$$

非目标下

$$f(x) = max\{log[F(x)]_{t_0} - \underset{i \neq t}{max} log[F(x)]_i, -K\}$$

随机选取一个坐标

估计梯度，h 非常小，ei 是一个只有 i-th 元素等于 1 的偏置向量。第二个只在牛顿法中才会使用。

$$\hat{g}_i := \frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + he_i) - f(x - he_i)}{2h}$$
$$\hat{h}_i := \frac{\partial^2 f(x)}{\partial x_{ii}^2} \approx \frac{f(x + he_i) - 2f(x) + f(x - he_i)}{h^2}$$

Algorithm 2 ZOO-ADAM: Zeroth Order Stochastic Coordinate Descent with Coordinate-wise ADAM

Require: Step size η, ADAM states $M \in \mathbb{R}^p, v \in \mathbb{R}^p, T \in \mathbb{Z}^p$, ADAM hyper-parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

$M \leftarrow \mathbf{0}, v \leftarrow \mathbf{0}, T \leftarrow \mathbf{0}$
while not converged **do**
  Randomly pick a coordinate $i \in \{1, \cdots, p\}$
  Estimate $\hat{g}_i$ using (6)
  $T_i \leftarrow T_i + 1$
  $M_i \leftarrow \beta_1 M_i + (1 - \beta_1)\hat{g}_i, \quad v_i \leftarrow \beta_2 v_i + (1 - \beta_2)\hat{g}_i^2$
  $\hat{M}_i = M_i/(1 - \beta_1^{T_i}), \quad \hat{v}_i = v_i/(1 - \beta_2^{T_i})$
  $\delta^* = -\eta \frac{\hat{M}_i}{\sqrt{\hat{v}_i} + \epsilon}$
  Update $\mathbf{x}_i \leftarrow \mathbf{x}_i + \delta^*$
end while

Algorithm 3 ZOO-Newton: Zeroth Order Stochastic Coordinate Descent with Coordinate-wise Newton's Method

Require: Step size η

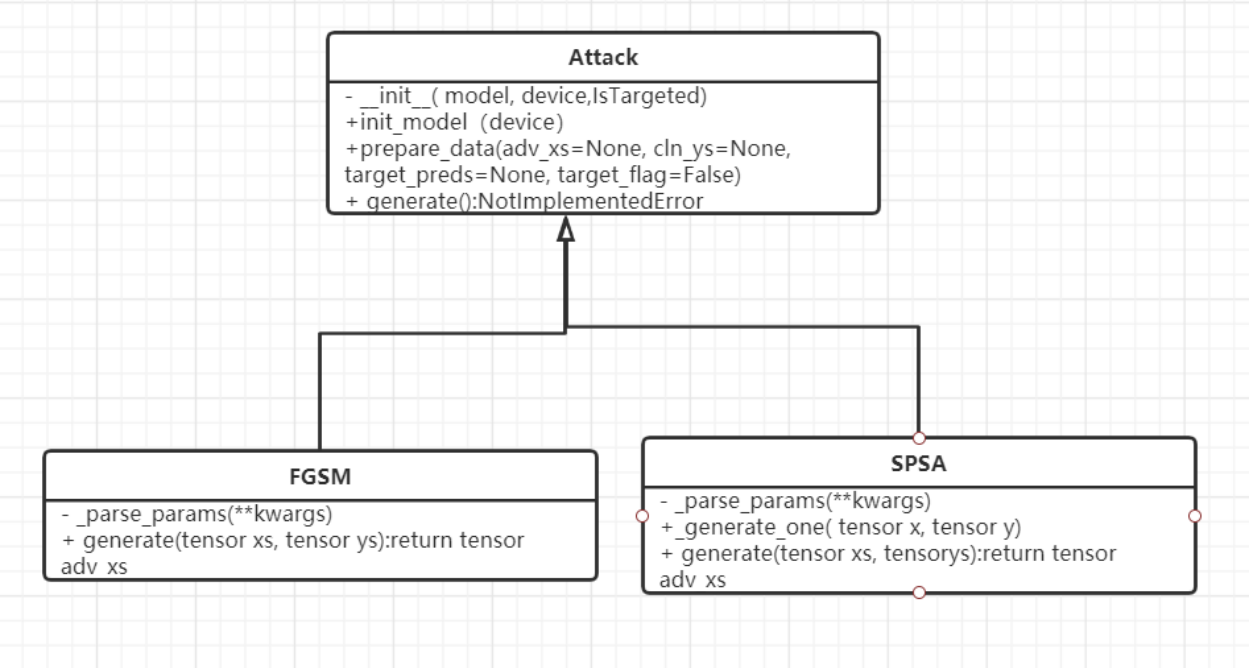
while not converged **do**
  Randomly pick a coordinate $i \in \{1, \cdots, p\}$
  Estimate $\hat{g}_i$ and $\hat{h}_i$ using (6) and (7)
  if $\hat{h}_i \leq 0$ **then**
    $\delta^* \leftarrow -\eta \hat{g}_i$
  else
    $\delta^* \leftarrow -\eta \frac{\hat{g}_i}{\hat{h}_i}$
  end if
  Update $\mathbf{x}_i \leftarrow \mathbf{x}_i + \delta^*$
end while

参数说明

参数名称	参数说明
solver	解算方法，Adam/Newton/Newton Adam
resize_init_size	输入图像调整后尺寸
img_h	图像高度
img_w	图像宽度
num_channels	图像通道数
use_resize	是否需要调整图像尺寸
class_type_number	分类类别数目
use_tanh	是否转化到 tanh 函数空间
confidence	班主判断攻击类别和预测类别是否相同或有固定偏差
batch_size	批处理大小
init_const	Loss1 初始化的调节系数（放大率）
max_iter	最大迭代次数
binary_search_steps	用于搜索初始 CONST 的迭代次数
beta1	用于产生中间图像数据的调节系数 1
beta2	用于产生中间图像数据的调节系数 2
lr	用于更新原有数据和梯度，以及二阶导数关系的调节系数
reset_adam_after_found	是否在找到参数后重置 adam
early_stop_iters	提早结束的迭代次数
ABORT_EARLY	没有提升是否提前中断
lower_bound	转化到 tan 函数空间的数据归一化下边界
upper_bound	转化到 tan 函数空间的数据归一化上边界
print_every	迭代过程的打印间隔
use_log	是否保存过程中的 Loss 值
save_modifier	是否保存过程中的攻击样本和原样本差值（修改值）
load_modifier	是否载入过程中的攻击样本和原样本差值（修改值）
use_importance	是否使用概率方法选择生成数据挑选次序

3.2 扩展攻击算法

3.2.1 攻击算法类图



如上图所示，上图为攻击算法 FGSM 和攻击算法 SPSA 继承 Attack 类示意图。攻击算法内部可自行实现攻击过程具体函数，仅需覆写 generate 函数即可。

3.2.2 攻击算法存储位置

```
EvalBox
    **Attack**
       AdvAttack  对抗攻击算法存储路径
           __init__.py
           attack.py
           fgsm.py
           ....py
       CorAttack  噪声攻击算法存储路径
    Analysis
    Defense
    Evaluation
Models
utils
test
```

(下页继续)

(续上页)

```
  Datasets
```

3.2.3 扩展实例——FGSM 算法

FGSM 算法路径为:

```
~/AISafety/EvalBox/Attack/fgsm.py
```

FGSM 算法源代码:

```
import numpy as np
import torch
from torch.autograd import Variable
from EvalBox.Attack.AdvAttack.attack import Attack
from utils.CrossEntropyLoss import CrossEntropyLoss

class FGSM(Attack):
    def __init__(self, model=None, device=None,IsTargeted=None, **kwargs):
        '''
        @description: Fast Gradient Sign Method (FGSM)
        @param {
            model: 需要测试的模型
            device: 设备 (GPU)
            IsTargeted: 是否是目标攻击
            kwargs: 用户对攻击方法需要的参数
        }
        @return: None
        '''
        super(FGSM, self).__init__(model, device,IsTargeted)
        # 使用该函数时候，要保证训练模型的标签是从 0 开始，而不是 1
        self.criterion = torch.nn.CrossEntropyLoss()
        self._parse_params(**kwargs)

    def _parse_params(self, **kwargs):
        '''
        @description:
        @param {
            epsilon: 沿着梯度方向步长的参数
        }
        @return: None
```

(下页继续)

(续上页)

```
        '''
        self.eps = float(kwargs.get('epsilon', 0.03))

    def generate(self, xs=None, ys=None):
        '''
        @description:
        @param {
            xs: 原始的样本
            ys: 样本的标签
        }
        @return: adv_xs{numpy.ndarray}
        '''
        device = self.device
        self.model.eval().to(device)
        targeted=self.IsTargeted
        print("targeted", targeted)
        copy_xs = np.copy(xs.numpy())
        var_xs = torch.tensor(copy_xs, dtype=torch.float,     device=device, requires_
↪grad=True)
        var_ys = torch.tensor(ys, device=device)
        outputs = self.model(var_xs)
        preds = torch.argmax(outputs, 1)
        preds = preds.data.cpu().numpy()
        if targeted:
           loss = -self.criterion(outputs, var_ys)
        else:
            loss = self.criterion(outputs, var_ys)
        loss.backward()
        grad_sign = var_xs.grad.data.sign().cpu().numpy()
        copy_xs = np.clip(copy_xs + self.eps * grad_sign, 0.0, 1.0)
        adv_xs = torch.from_numpy(copy_xs)
        return adv_xs
```

3.2.4 扩展说明

1. 用户需要实现个人攻击算法，并继承基础的 Attack 类
2. 用户需要将待扩展的攻击算法对应文件，如 new_attack_method.py，放置于以下路径中

```
~/AISafety/EvalBox/Attack/
```

3. 用户需要在 2 中路径下的 ___init___.py 文件中，添加用户攻击算法类的引用：

```
from .attack import Attack
from .fgsm import FGSM
...
from .new_attack_method import NEW_ATTACK_METHOD
```

4. 用户需要在 test/attack_param 路径下，生成个人预设参数 txt 文件，及参数 xml 文件。若仅需要默认参数，则 xml 文件可为空。

5. 用户可在集成调用文件 testimport.py 中，修改 attack_method 参数为 NEW_ATTACK_METHOD

CHAPTER 4

评测算法介绍

本平台目前已集成的评测方法体系的整体结构如下所示，从两个维度进行评测：针对模型本身以及针对模型使用的数据（测试样本）。

4.1 评测算法总览

平台已集成的评测算法及简要说明如下所示 (使用评测算法缩写字典序排序)

4.1.1 第一类评测算法

第一类评测算法指：针对模型行为鲁棒性，通过对输入样本添加对抗噪音，测试在对抗噪音攻击下模型鲁棒性的一类评测算法。该类评测算法存储的主要路径为：

```
~/AISafety/EvalBox/Evaluation
```

评测算法名称	评测方法缩写	评测方法简要说明
clean accuracy	ACC	未受攻击时，模型预测准确率
	ACAC	对错误类别的平均预测置信度。其定义为经过对抗攻击后，对于所有攻击成功对抗样本（FGSM、PGD、C& W），所有误分类类别的平均概率。
	ACTC	正确类别平均置信度。通过对对抗攻击样本（FGSM、PGD、C&W）的真实类，计算预测可信度的平均值。用来评估攻击在多大程度上偏离真实值。
	ALDp	平均 Lp 失真度（使用 I-FGSM，对于每个样本，直到模型产生误分类）。具体来说，计算扰动后发生改变的像素数量；计算原始示例和扰动示例之间的欧氏距离；测量对抗样本全维度下最大变化量。ALDp 为所有攻击成功的对抗样本的平均归一化 Lp 失真度，ALDp 越小，对抗样本的不可感知性越强。
	ASS	平均结构相似性（使用 I-FGSM，对于每个样本，直到模型产生误分类）。为了评估对抗样本的不可感知性，ASS 被定义为所有攻击成功对抗样本与其原始样本间的平均相似性。
	PSD	扰动敏感距离。用于评测人类对扰动的感知能力。其中 m 为像素点总数，表示第 i 个样例的第 j 个像素点，表示附近平方区域，std 表示标准偏差函数。
	NTE	NTE 计算了误分类概率与其他类最大概率之间的差值。，其中，且
	RGB	高斯模糊前后分类准确率比值
	RIC	图像压缩前后分类准确率比值

4.1.2 第二类评测算法

第二类评测算法指：针对模型行为鲁棒性，通过对输入样本添加自然噪音，测试在自然噪音攻击下模型鲁棒性的一类评测算法。该类评测算法存储的主要路径为：

```
~/AISafety/EvalBox/UserEvaluation
```

评测算法名称	评测方法缩写	评测方法简要说明
Mean Corruption Error	MCE	计算神经网络对自然噪音的平均 error 值
relative Mean Corruption Error	RMCE	计算神经网络在自然噪音下，error 与 vanilla 差值
Mean Flip Probability	MFP	计算神经网络对自然噪音序列的 error
Mean Top-5 Distance	MT5D	神经网络 Top5 的预测不一致性

4.1.3 第三类评测算法

第三类评测算法指：为了提高模型鲁棒性，通过对抗训练或其他加固方式，得到加固后模型，针对模型加固前后鲁棒性变化及预测能力变化进行的评测方法。该类评测算法存储的主要路径为：

```
~/AISafety/EvalBox/Evaluation
```

评测算法名称	评测方法缩写	评测方法简要说明
	CAV	分类准确度方差，模型防御前后准确度的差
	CRR	模型防御后，减少的错误分类百分比
	CSR	模型防御后，增加的错误分类百分比
	CCV	测量防御造成的置信方差（对防御前后均分类正确样本做统计）
	COS	使用 JS 散度衡量模型防御前后输出概率相似性

4.1.4 第四类评测算法

第四类评测算法指：针对模型静态结构，使用神经元覆盖等形式对模型进行评测的方法。该类评测算法存储的主要路径为：

```
~/AISafety/EvalBox/UserEvaluation
```

评测算法名称	评测方法缩写	评测方法简要说明
Neuron Sensitivity	SNS	计算神经网络中神经单元对噪音敏感性
ep silon-Empirical Noise Sensitivity	ENI	使用 Lipsch itz 连续常数衡量模型对噪音的敏感性
Worst Case Boundary Distance	BD	随机寻找 N 个正交的方向，然后计算对于一个模型来说每个方向需要移动多少可以改变 instance 的 prediction la bel，来衡量模型的决策边界最小距离
Worst Case Boundary Distance-2	BD2	上一种方法的变种，对于每一个 direction（class），对于每一个 instance 使用 I-FGSM 来迭代移动 n steps，计算 n 的大小来衡量模型的决策边界最小距离
Critical attacking route	CAR	描述神经网络的脆弱路径

4.1.5 第五类评测算法

第五类评测算法指：针对进行模型训练或测试的数据集，根据模型在不同数据集上活跃表现，对模型预测行为加以可解释分析的方法。该类评测算法存储的主要路径为：

```
~/AISafety/EvalBox/UserEvaluation
```

评测算法名称	评测方法缩写	评测方法简要说明
Neuron Coverage	NC	神经元覆盖率，用于衡量测试数据是否可以对神经网络神经元进行覆盖
K-Multisection Neuron Coverage	KMNC	K 多节神经元覆盖
Neuron Boundary Coverage	NBC	神经元边界覆盖
Strong Neuron Activation Coverage	SNAC	强神经元激活覆盖
Top-K Neuron Coverage	TKNC	Top-k 神经元覆盖
Top-k Neuron Patterns	TKNP	Top-k 神经元模式

4.2 评测算法详细介绍

平台已集成的评测算法及简要说明如下所示 (使用评测算法缩写字典序排序)

4.2.1 ACC 精确度

模型的分类精确度（Accuracy，ACC)。其定义为经过对抗攻击后，模型对于所有攻击样本中依然能够分类正确的样本占总体样本的比值。公式定义如下：

$$ACC = \frac{n}{N}$$

其中，n 代表所有对抗样本中分类正确的数量，N 代表对抗样本的总数。

4.2.2 ACAC 平均置信度

对错误类别的平均预测置信度 (Average Confidence of Adversarial Class)。其定义为经过对抗攻击后，对于所有攻击成功对抗样本，所有误分类类别的平均概率。公式定义如下

$$ACAC = \frac{1}{n}\sum_{i=1}^{n} P\left(X_i^a\right)_{F\left(X_i^a\right)}$$

其中 n 表示所有对抗攻击成功样本个数，F(Xi^a) 表示第 i 个样本被分类为 a 类，P(Xi^a) 为第 i 个样本被分类为 a 类的概率。

评测算法名称	评测方法缩写	评测方法简要说明
Neuron Coverage	NC	神经元覆盖率，用于衡量测试数据是否可以对神经网络神经元进行覆盖
K-Multisection Neuron Coverage	KMNC	K 多节神经元覆盖
Neuron Boundary Coverage	NBC	神经元边界覆盖
Strong Neuron Activation Coverage	SNAC	强神经元激活覆盖
Top-K Neuron Coverage	TKNC	Top-k 神经元覆盖
Top-k Neuron Patterns	TKNP	Top-k 神经元模式

4.2.3 ACTC 对抗成功时正确标签平均置信度

正确类别平均置信度 (Average Confidence of True Class)。通过对对抗攻击样本的真实类，计算预测可信度的平均值。用来评估攻击在多大程度上偏离真实值。

$$ACTC = \frac{1}{n}\sum_{i=1}^{n} P\left(X_i^a\right)_{y_i}$$

其中 P(Xi^a) 为第 i 个样本被分类为对应类的概率，yi 表示第 i 个样本被正确分类为 yi 类。

4.2.4 ALDp 对抗攻击失真度

平均 Lp 失真度 (Average Lp Distortion)。几乎所有的攻击都采用 Lp norm 距离（p=0, 2, ∞）作为评价的失真度量。具体来说，L0 计算扰动后发生改变的像素数量；L2 计算原始示例和扰动示例之间的欧氏距离；L∞ 测量对抗样本全维度下最大变化量。ALDp 为所有攻击成功的对抗样本的平均归一化 Lp 失真度，ALDp 越小，对抗样本的不可感知性越强。

$$ALDp = \frac{1}{n}\sum_{i=1}^{n} \frac{\left\|X_i^a - X_i\right\|_p}{\left\|X_i\right\|_p}$$

其中 n 为所有攻击成功的对抗样本个数。

4.2.5 ASS 对抗样本数据与原数据的结构相似度（SSIM）

平均结构相似性 (Average Structural Similarity)。SSIM 作为量化两幅图像间相似性常用指标之一，被认为比 Lp 相似度更符合人类视觉感知。为了评估对抗样本的不可感知性，ASS 被定义为所有攻击成功对抗样本与其原始样本间的平均相似性，即

$$ASS = \frac{1}{n}\sum_{i=1}^{n} SSIM\left(X_i^a, X_i\right)$$

其中 n 表示所有攻击成功的对抗样本个数。ASS 值越大，则对抗样本的不可感知性越强。

4.2.6 BD 最大边界距离

最大边界距离（Worst Case Boundary Distance）。数据点之间到决策边界的距离衡量模型在最坏情况下的稳定性和鲁棒性。随机寻找 N 个正交的方向，然后计算对于一个模型来说每个方向需要移动多少可以改变 instance 的 prediction label，来衡量模型的决策边界最大距离。

$$BD = \frac{1}{N}\sum_{i=1}^{N} d_i, d_i = max\phi_i\left(V\right)$$

其中，V 表示一个随机生成的集合，ϕi(V) 为到模型决策边界的 RMS 距离，di 为到决策边界距离的最大值。

4.2.7 CAV 分类准确度方差

分类准确度方差 (Classification Accuracy Variance)。用于评估深度学习模型性能的最重要指标为准确性，因此，防御增强模型应尽可能保持常规测试实例下的分类精度。通过 CAV 评估防御对模型分类精度的影响，定义为

$$CAV = Acc(F^D, T) - Acc(F, T)$$

其中 Acc(F,T) 表示模型 F 在数据集 T 下的识别准确率。

4.2.8 CCV 分类置信方差

分类置信方差 (Classification Confidence Variance)。对原始模型增加防御，可能不会影响评估准确率，但是正确分类样本的预测可信度可能会降低，因此引入 CCV，测量防御增强模型引起的置信方差，我们定义

$$CCV = \frac{1}{n}\sum_{i=1}^{n} |P(X_i)_{y_i} - P^D(X_i)_{y_i}|$$

其中 n 表示在原始模型及防御增强后模型全部分类准确的样本数量。

4.2.9 COS 分类输出稳定性

COS 全称为分类输出稳定性 (Classification Output Stability)。使用 JS 散度来衡量原始模型和防御增强模型输出概率相似性，即分类输出的稳定性。

对所有正确分类的测试实例，定义

$$COS = \frac{1}{n}\sum_{i=1}^{n} JSD(P(X_i)||P^D(X_i))$$

其中 n 表示在原始模型及防御增强后模型全部分类准确的样本数量，JSD 表示计算 JS 散度的函数。COS 值越低，两个模型的差距越小。

4.2.10 CRR/CSR 分类校正/补偿比率

分类校正/补偿比率 (Classification Rectify/Sacrifice Ratio)。为了评估防御对模型在测试集上预测结果的影响，将 CRR 定义为原始模型错误分类，但增加防御后，模型正确分类的测试实例的百分比。与此相反，CSR 表示原始模型正确分类，但增加防御后，模型错误分类的测试实例的百分比。

$$CRR = \frac{1}{N}\sum_{i=1}^{N} count(F(X_i) \neq y_i \& F^D(X_i) = y_i)$$

$$CSR = \frac{1}{N}\sum_{i=1}^{N} count(F(X_i) = y_i \& F^D(X_i) \neq y_i)$$

同时 CAV = CRR - CSR。其中 F 表示原始模型预测结果，F^D 表示防御后模型预测结果，& 符号表示同时满足。返回值越小，两个模型的差距越小。

4.2.11 ENI

ϵ-Empirical Noise Insensitivity（综合对抗攻击和自然噪音的一个 test set）。ENI 值越低，攻击的不可感知性越强。

4.2.12 NTE 噪声容量估计

噪声容量估计 (Noise Tolerance Estimation)，对抗样本的鲁棒性可通过噪声容限来估计，噪声容限反映了对抗样本在保持分类类别不变的情况下，可容忍的噪声量。具体来说，NTE 计算了误分类概率与其他类最大概率之间的差值。

$$NTE = \frac{1}{n}\sum_{i=1}^{n}[P(X_i^a)_{F(X_i^a)} - maxP(X_i^a)_j] \quad j \in \{1, ..., k\},\ j \neq F(X_i^a)$$

NTE 值越高，说明对抗样本的鲁棒性越高，攻击算法的鲁棒性越强。

4.2.13 PSD 扰动敏感距离

扰动敏感距离 (Perturbation Sensitivity Distance)。用于评测人类对扰动的感知能力。

$$PSD = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{\delta_{i,j}}{std(R(x_{i,j}))}$$

其中 m 为像素点总数 δ{i, j} 表示第 i 个样例的第 j 个像素点，R(x{i,j}) 表示 x{i, j} 附近平方区域，std 表示标准偏差函数。PSD 的值越小，则对抗样本的不可感知性越强。

4.2.14 RGB 对高斯模糊鲁棒性

对高斯模糊鲁棒性 (Robustness to Gaussian Blur)。高斯模糊常被用于计算机视觉算法中的图像去噪。正常情况下，一个高鲁棒性的对抗样本，在高斯模糊后应保持其误分类效果。

可以定义

$$RGB_{UA} = \frac{count(F(GB(X_i^a)) \neq y_i)}{count(F(X_i^a) \neq y_i)}$$
$$RGB_{TA} = \frac{count(F(GB(X_i^a)) = y_i^+)}{count(F(X_i^a) = y_i^+)}$$

UA 表示非定向攻击，TA 表示定向攻击，GB 函数表示高斯模糊处理。RGB 结果越高，说明对抗样本鲁棒性越强。

4.2.15 RIC 对图像压缩鲁棒性

对图像压缩鲁棒性 (Robustness to Image Compression)。图像压缩常被用于计算机视觉算法中的图像去噪。正常情况下，一个高鲁棒性的对抗样本，在图像压缩后应保持其误分类效果。

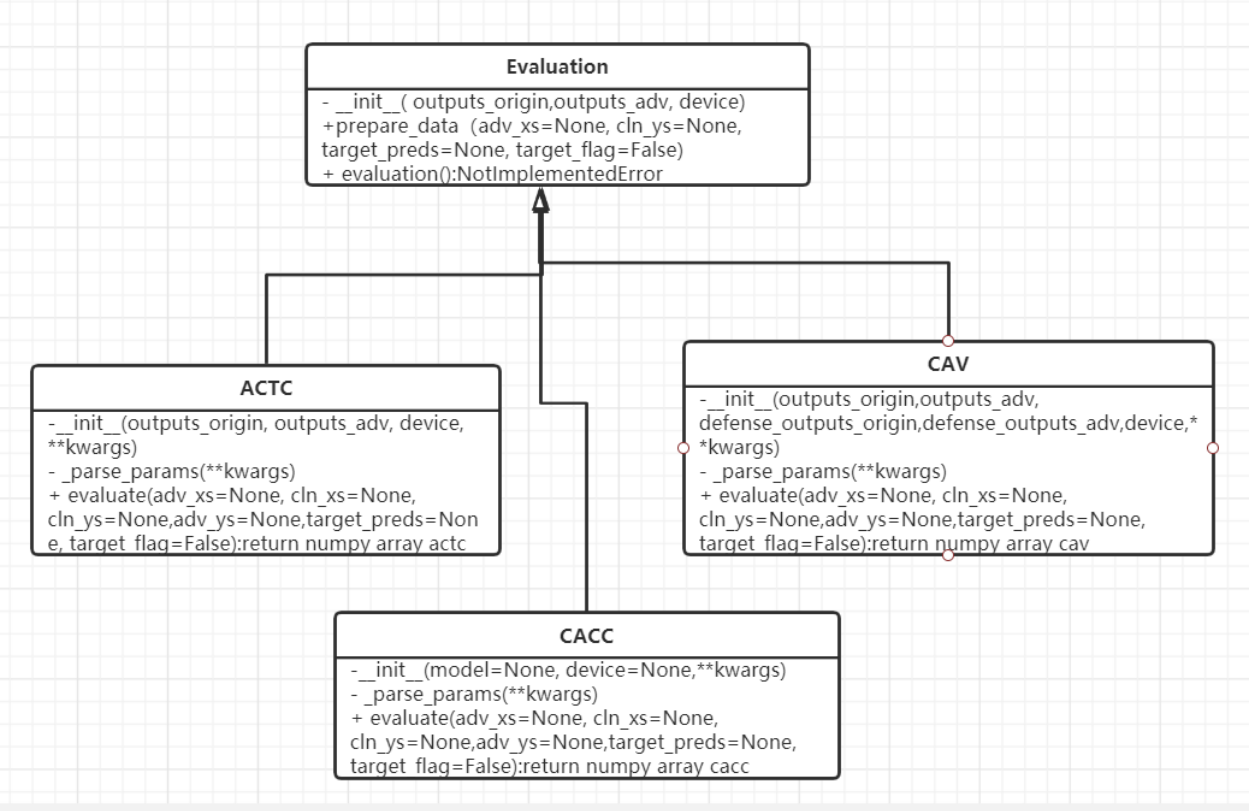
可以定义

$$RIC_{UA} = \frac{count(F(IC(X_i^a)) \neq y_i)}{count(F(X_i^a) \neq y_i)}$$
$$RIC_{TA} = \frac{count(F(IC(X_i^a)) = y_i^+)}{count(F(X_i^a) = y_i^+)}$$

UA 表示非定向攻击，TA 表示定向攻击，IC 函数表示图像压缩处理。RIC 结果越高，说明对抗样本鲁棒性越强。

4.3 扩展评测算法

4.3.1 评测算法类图



如上图所示，上图为评测算法 ACTC 和评测算法 CAV 和评测算法 CACC 继承 Evaluation 类示意图。评测算法内部可自行实现评测过程具体函数，仅需覆写 evaluate 函数即可。

4.3.2 评测算法存储位置

```
EvalBox
    Attack
    Analysis
    **Evaluation**  评测算法存储位置
       __init__.py
       actc.py
       CAV.py
       evaluation.py
       evaluation_defense.py
    Defense
```

(下页继续)

(续上页)

```
  Models
  utils
  test
  Datasets
```

4.3.3 扩展实例——ACTC 算法

ACTC 算法路径为:

```
~/AISafety/EvalBox/Evaluation/actc.py
```

ACTC 算法源代码:

```
class ACTC(Evaluation):
    def __init__(self, outputs_origin, outputs_adv, device, **kwargs):
        '''
        @description:
        @param {
            model:
            device:
            kwargs:
        }
        @return: None
        '''
        super(ACTC, self).__init__(outputs_origin, outputs_adv, device)

        self._parsing_parameters(**kwargs)
    def _parsing_parameters(self, **kwargs):
        '''
        @description:
        @param {
        }
        @return:
        '''

    def evaluate(self,adv_xs=None, cln_xs=None, cln_ys=None,adv_ys=None,target_
↪preds=None, target_flag=False):
        '''
        @description:
        @param {
```

(下页继续)

(续上页)

```
        adv_xs: 攻击样本
        cln_xs: 原始样本
        cln_ys: 原始类别，非目标攻击下原始样本的类型
        adv_ys: 攻击样本的预测类别
        target_preds: 目标攻击下希望原始样本攻击的目标类别
        target_flag: 是否是目标攻击
    }
    @return: actc {Average Confidence of True Class}
    '''
    total = len(adv_xs)
    print("total", total)
    outputs = torch.from_numpy(self.outputs_adv)
    number = 0
    prob = 0
    outputs_softmax=torch.nn.functional.softmax(outputs, dim=1)
    preds = torch.argmax(outputs, 1)
    outputs_softmax = outputs_softmax.data.numpy()
    preds = preds.data.numpy()
    labels = target_preds.numpy()
    if not target_flag:
        for i in range(preds.size):
            if preds[i] != labels[i]:
                number += 1
                prob += outputs_softmax[i,labels[i]]
    else:
        for i in range(preds.size):
            if preds[i] == labels[i]:
                number += 1
                prob += outputs_softmax[i, labels[i]]
    if not number == 0:
        actc = prob / number
    else:
        actc = prob/(number+MIN_COMPENSATION)
    return actc
```

4.3.4 扩展说明

1. 用户需要实现个人评测算法，并继承基础的 Evaluation 类，若用户希望实现的是比较模式的评测算法，则继承 Evaluation_Defense 类

2. 用户需要将待扩展的评测算法对应文件，如 new_evaluate_method.py，放置于以下路径中

```
~/AISafety/EvalBox/Evaluation/
```

3. 用户需要在 2 中路径下的 ___init___.py 文件中，添加用户评测算法类的引用：

```
from .CAV import CAV
from .acc import ACC
...
from .new_evaluate_method import NEW_EVALUATE_METHOD
```

4. 用户可在集成调用文件 testimport.py 中，修改 evaluation_method 参数为 NEW_EVALUATE_METHOD
5. 如果用户评测算法是针对动态行为的评测算法，即需要获取传入模型，可设置 IS_PYTHORCH_WHITE 参数为 True
6. 若用户评测算法为对比模型，需设置 IS_COMPARE_MODEL 参数为 True

CHAPTER 5

防御算法介绍

本平台目前已集成的防御加固方法共有 5 个。

5.1 防御算法介绍

5.1.1 EAT 加固算法

算法介绍

EAT 的全称是 Ensemble adversarial training (集成对抗训练)，具体是指使用多种方式生成对抗样本对模型进行对抗训练的方法，通过使用在其他静态预训练模型上产生的对抗样本来扩充模型的训练数据，能有效应对对抗样本的可迁移攻击。

与之相关的基础方法是对抗训练，该方法旨在从随机初始化的权重中训练一个鲁棒的模型，其训练集由真实数据集和加入了对抗扰动的数据集组成，因此叫做对抗训练。其表达式如下：

$$\tilde{J}(\theta, x, y) = \alpha \cdot J(\theta, x, y) + (1 - \alpha) \cdot J(\theta, x + \epsilon \cdot sign(\nabla_x J(\theta, x, y)))$$

其中 J(θ, x, y) 是模型对于普通样本的损失函数，作者通过 x + ϵ · sign(x J(θ,x,y)) 来构造对抗样本，并要求模型能正确对其分类。

参数说明

参数名	参数含义
dataset	数据集类型
num_epochs	对抗训练迭代的次数
epsilon	样本扰动的偏移值
learn_rate	训练的学习率
alpha	参与对抗样本一次的比例
train_externals	是否使用外部预训练模型

5.1.2 NAT 加固算法

算法介绍

New adversarial training (NAT)

对抗训练（adversarial training）是增强神经网络鲁棒性的重要方式。在对抗训练的过程中，样本会被混合一些微小的扰动（改变很小，但是很可能造成误分类），然后使神经网络适应这种改变，从而对对抗样本具有鲁棒性。

对抗训练的一般性原理，对抗训练可以概括为如下的最大最小化公式：

$$min_\theta E_{(Z,y)} \sim D[max_{||\delta|| \leq \epsilon} L(f_\theta(X+\delta), y)]$$

内层（中括号内）是一个最大化，其中 X 表示样本的输入表示，δ 表示叠加在输入上的扰动，θ 是神经网络函数，y 是样本的标签，L(fθ(X + δ), y) 则表示在样本 X 上叠加一个扰动 δ，再经过神经网络函数，与标签 y 比较得到的损失。maxL 是优化目标，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神经网络迷惑。

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，我们训练神经网络模型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。

这里过程中生成的对抗样本采用 RLLC 的方式生成，其他的过程和 OAT 的类似。

参数说明

参数名	参数含义
num_epochs	对抗训练迭代的次数，该数值越大，时间开销越大
clip_eps_min	随机样本生成的下限
clip_eps_max	随机样本生成的上限
adv_radio	求 loss 的时候对抗样本占比
eps_mu	随机样本分布的中心
eps_sigma	随机样本分布的标准差

5.1.3 OAT 加固算法

算法介绍

Original adversarial training (OAT)

对抗训练（adversarial training）是增强神经网络鲁棒性的重要方式。在对抗训练的过程中，样本会被混合一些微小的扰动（改变很小，但是很可能造成误分类），然后使神经网络适应这种改变，从而对对抗样本具有鲁棒性。

对抗训练的一般性原理，对抗训练可以概括为如下的最大最小化公式：

$$min_{\theta}E_{(Z,y)} \sim D[max_{||\delta|| \leq \epsilon}L(f_{\theta}(X+\delta), y)]$$

内层（中括号内）是一个最大化，其中 X 表示样本的输入表示，δ 表示叠加在输入上的扰动，θ 是神经网络函数，y 是样本的标签，L(fθ(X + δ), y) 则表示在样本 X 上叠加一个扰动 δ，再经过神经网络函数，与标签 y 比较得到的损失。maxL 是优化目标，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神经网络迷惑。

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，我们训练神经网络模型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。

参数说明

参数名	参数含义
num_epochs	对抗训练迭代的次数，该数值越大，时间开销越大
epsilon	样本扰动的偏移值
attak_step_num	对抗攻击训练中一次选用的样本数目
alpha	参与对抗样本一次的比例

5.1.4 PAT 加固算法

算法介绍

PAT 的全称是 PGD adversarial training（PGD 对抗训练），在白盒环境下，通过 PGD 攻击算法生成对抗样本，然后用对抗样本和普通样本混合训练模型。

基于 FGSM 攻击方法，Madry 等人引入迭代的过程，并且每次将噪音映射到某个特定空间中形成了目前最为常用的 PGD 攻击方法：

$$x_0^{'} = x$$
$$x_{n+1}^{'} = \prod_{x+S} x_n^{'} + \epsilon \cdot sign(\nabla_x J(\theta, x_n^{'}, y))$$

Goodfellow 等人最早提出对抗训练的思想，该论文尝试在训练过程中加入对抗样本来提升模型的鲁棒性，其损失函数定义如下：

$$\tilde{J}(\theta, x, y) = \alpha \cdot J(\theta, x, y) + (1 - \alpha) \cdot J(\theta, x + \epsilon \cdot sign(\nabla_x J(\theta, x, y)))$$

其中 J(θ, x, y) 是模型对于普通样本的损失函数，作者通过 x + ϵ · sign(x J(θ,x,y)) 来构造对抗样本，并要求模型能正确对其分类。

PAT 对抗训练优化的目标函数如下：

$$\theta^* = min_\theta E_{(x,y)}[max_\sigma l(x + \sigma; y; F_\theta)]$$

其中 l(x + σ;y;Fθ) 为模型 Fθ 在对抗噪音大小为 σ 下的损失函数。

在优化目标函数过程中，该方法在每一个批数据（mini-batch）中加入同样数量的普通样本和对抗样本（由 PGD 攻击生成）。

参数说明

参数名	参数含义
α	对抗训练中普通样本和对抗样本的混合比例，一般设置为 1:1。

5.1.5 RAND 加固算法

算法介绍

Rand 方法利用随机化的方法来对输入图片引入随机性操作，作为一个预处理模块，该方法可以有效的提升深度神经网络对于对抗样本噪音的防御能力和鲁棒性。

该方法的具体操作有如下两种方式：

1. 对于原始图片 X，对其大小 W · H · 3 进行随机修改，变化为 W' · H' · 3，这其中大小的变换要在合理范围内（如：2 个像素值）；
2. 第二种方式是对原始图片 X 进行随机填充。使用值为 0 的像素在原始图片的四周进行填充。

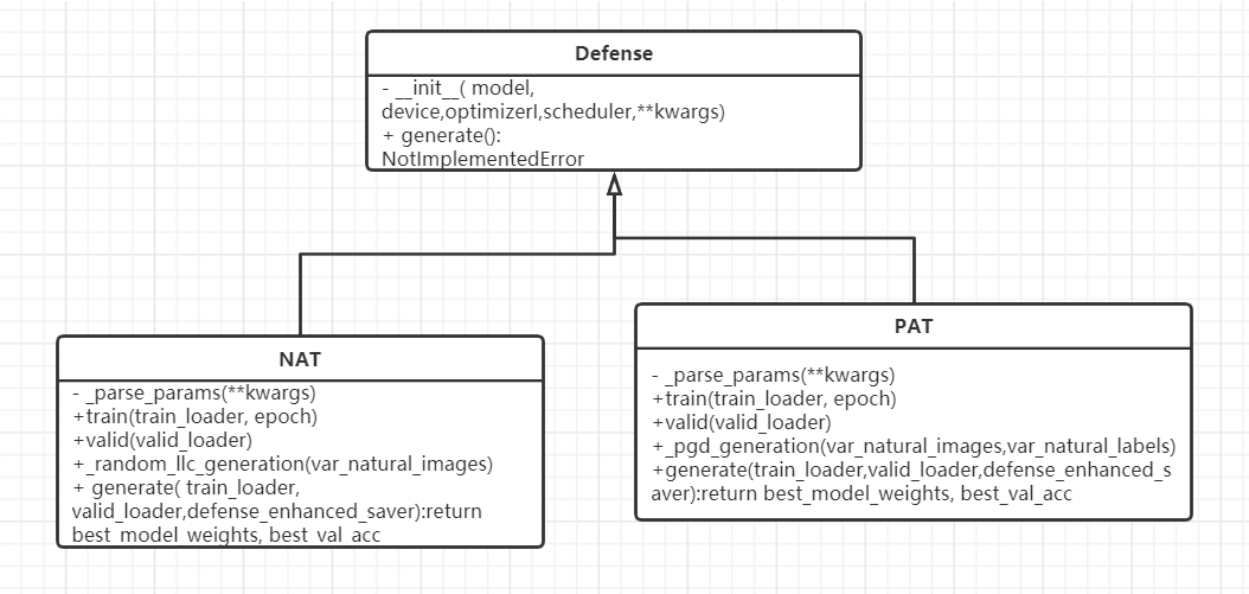
这两种方式对于输入图片引入很多随机性，从而削弱了对抗攻击的效果，提升了模型对抗鲁棒性。

参数说明

参数名	参数含义
rnd	随机变化后的图片大小。

5.2 扩展防御算法

5.2.1 防御算法类图



如上图所示，上图为防御算法 NAT 和防御算法 PAT 继承 Defense 类示意图。防御算法内部可自行实现防御过程具体函数，仅需覆写 train 方法，valid 方法，generate 方法即可。

5.2.2 防御算法存储位置

```
EvalBox
    Attack
    Analysis
    **Defense**    防御算法路径
       __init__.py
       nat.py
       pat.py
       ....py
       defense.py
    Evaluation
Models
utils
test
Datasets
```

5.2.3 扩展实例——NAT 算法

NAT 算法路径为:

```
~/AISafety/EvalBox/Defense/nat.py
```

ACTC 算法源代码:

```
class NAT(Defense):
    def __init__(self,
                 model=None,
                 device=None,
                 optimizer=None,
                 scheduler=None,
                 **kwargs):
        '''
        @description: New adversarial training (NAT)
        @param {
            model:
            device:
            optimizer:
            scheduler:
            kwargs:
        }
        @return: None
        '''
        super().__init__(model, device)

        self.criterion = torch.nn.CrossEntropyLoss()
        self.optimizer = optimizer
        self._parse_params(**kwargs)

    def _parse_params(self, **kwargs):
        '''
        @description:
        @param {
            num_epochs:
            adv_ratio:
            clip_eps_min:
            clip_eps_max:
            eps_mu:
            eps_sigma:
```

(下页继续)

(续上页)

```
        }
        @return: None
        '''
        self.num_epochs = int(kwargs.get('num_epochs', 200))
        self.adv_ratio = float(kwargs.get('adv_ratio', 0.3))
        self.clip_eps_min =float( kwargs.get('eps_min', 0.0))
        self.clip_eps_max = float(kwargs.get('eps_max', 0.3))
        self.eps_mu = float(kwargs.get('eps_mu', 0))
        self.eps_sigma =float( kwargs.get('eps_sigma', 50))

    def _random_llc_generation(self, var_natural_images=None):
        '''
        @description:
        @param {
            var_natural_images:
        }
        @return: ret_adv_images
        '''
        device = self.device
        self.model.eval().to(device)
        clone_var_natural_images = var_natural_images.clone()

        # get the random epsilon for the Random LLC generation
        random_eps = np.random.normal(
            loc=self.eps_mu,
            scale=self.eps_sigma,
            size=[var_natural_images.size(0)]) / 255.0
        random_eps = np.clip(
            np.abs(random_eps), self.clip_eps_min, self.clip_eps_max)

        clone_var_natural_images.requires_grad = True

        # prepare the least likely class labels (avoid label leaking effect)
        logits = self.model(clone_var_natural_images)
        llc_labels = torch.min(logits, dim=1)[1]
        # get the loss and gradients
        loss_llc = self.criterion(logits, llc_labels)
        gradients_llc = torch.autograd.grad(loss_llc, clone_var_natural_images)[0]

        clone_var_natural_images.requires_grad = False
```

(下页继续)

(续上页)

```
        gradients_sign = torch.sign(gradients_llc)
        var_random_eps = torch.from_numpy(random_eps).float().to(device)

        # generation of adversarial examples
        with torch.no_grad():
            list_var_adv_images = []
            for i in range(var_natural_images.size(0)):
                var_adv_image = var_natural_images[i] - var_random_eps[i] * gradients_
↪sign[i]
                var_adv_image = torch.clamp(var_adv_image, min=0.0, max=1.0)
                list_var_adv_images.append(var_adv_image)
            ret_adv_images = torch.stack(list_var_adv_images)
        ret_adv_images = torch.clamp(ret_adv_images, min=0.0, max=1.0)

        return ret_adv_images

    def valid(self, valid_loader=None):
        '''
        @description:
        @param {
            valid_loader:
            epoch:
        }
        @return: val_acc
        '''
        device = self.device
        self.model.to(device).eval()
        correct = 0
        total = 0
        with torch.no_grad():
            for inputs, labels in valid_loader:
                inputs = inputs.to(device)
                labels = labels.to(device)

                outputs = self.model(inputs)
                preds = torch.argmax(outputs, 1)
                total += inputs.shape[0]
                correct += (preds == labels).sum().item()
            val_acc = correct / total
```

(下页继续)

(续上页)

```
        return val_acc

    def train(self, train_loader=None, epoch=None):
        '''
        @description:
        @param {
            train_loader:
            epoch:
        }
        @return: None
        '''
        device = self.device
        self.model.to(device)

        for index, (images, labels) in enumerate(train_loader):
            nat_images = images.to(device)
            nat_labels = labels.to(device)
            self.model.eval()
            adv_images = self._random_llc_generation(
                var_natural_images=nat_images)
            self.model.train()
            logits_nat = self.model(nat_images)
            loss_nat = self.criterion(logits_nat, nat_labels)
            logits_adv = self.model(adv_images)
            loss_adv = self.criterion(logits_adv, nat_labels)

            loss = (loss_nat + self.adv_ratio * loss_adv) / (1.0 + self.adv_ratio)

            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()
            print(
                '\rTrain Epoch {:>2}: [batch:{:>4}/{:>4}]  \tloss_nat={:.4f}, loss_adv=
↪{:.4f}, total_loss={:.4f} ===> '
                .format(epoch, index, len(train_loader), loss_nat.item(), loss_adv.
↪item(), loss.item()), end=' ')

    def generate(self, train_loader=None, valid_loader=None, defense_enhanced_
↪saver=None):
        '''
```

(下页继续)

(续上页)

```
        @description:
        @param {
            train_loader:
            valid_loader:
        }
        @return: best_model_weights, best_acc
        '''
        best_val_acc = None
        best_model_weights = self.model.state_dict()
        dir_path = os.path.dirname(defense_enhanced_saver)
        if not os.path.exists(dir_path):
            os.mkdir(dir_path)
        for epoch in range(self.num_epochs):
            self.train(train_loader, epoch)
            val_acc = self.valid(valid_loader)
            adjust_learning_rate(epoch=epoch, optimizer=self.optimizer)
            if not best_val_acc or round(val_acc, 4) >= round(best_val_acc, 4):
                if best_val_acc is not None:
                    os.remove(defense_enhanced_saver)
                best_val_acc = val_acc
                best_model_weights = self.model.state_dict()
                torch.save(self.model.state_dict(), defense_enhanced_saver)
            else:
                print('Train Epoch{:>3}: validation dataset accuracy did not improve␣
↪from {:.4f}\n'.format(epoch, best_val_acc))
        print('Best val Acc: {:.4f}'.format(best_val_acc))
        return best_model_weights, best_val_acc
```

5.2.4 扩展说明

1. 用户需要实现个人防御算法，并继承基础的 Defense 类
2. 用户需要将待扩展的防御算法对应文件，如 new_defense_method.py，放置于以下路径中

```
~/AISafety/EvalBox/Defense/
```

3. 用户需要在 2 中路径下的 ___init___.py 文件中，添加用户防御算法类的引用：

```
from .pat import PAT
from .oat import OAT
```

(下页继续)

(续上页)

```
...
from .new_defense_method import NEW_DEFENSE_METHOD
```

4. 用户可在集成调用文件 testimport_defense.py 中，修改对应方法名，方法参数路径信息

5.3 加固防御模型调用说明

平台已给出加固模型功能的集成调用文件，集成调用文件路径如下：

```
~/AISafety/test/test_defense.py
```

下面将就 Cifar-10 数据集，给出一个完整的调用接口说明

5.3.1 加固防御模型调用参数说明

test_defense.py 文件中，主要逻辑代码如下：

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='The Defense Model Generation')
    # common arguments
    # 用户所选用的加固算法缩写
    parser.add_argument(
        '--defense_method',
        type=str,
        nargs='*',
        default = "RAND")
    # 用户使用的数据集路径
    parser.add_argument(
        '--Data_path',
        type=str,
        nargs='*',
        default=["../Datasets/CIFAR_cln_data/cifar10_300_origin_inputs.npy",
                 "../Datasets/CIFAR_cln_data/cifar10_300_origin_labels.npy",
                 "../Datasets/CIFAR_cln_data/cifar10_300_origin_inputs.npy",
                 "../Datasets/CIFAR_cln_data/cifar10_300_origin_labels.npy" ])
    # 用户使用的数据集对应的字典文件
    parser.add_argument(
        '--Dict_path',
        type=str,
```

(下页继续)

(续上页)

```
        default="./dict_lists/cifar10_dict.txt")
    # 用户使用的模型文件
    parser.add_argument(
        '--model',
        type=str,
        default ='Models.UserModel.ResNet2')
    # 执行攻击过程的 batch_size 大小
    parser.add_argument(
        '--batch_size', type=int, default=64, help='batch size')
    # arguments for the particular attack
    parser.add_argument(
        '--Scale_ImageSize',
        type=int,
        default=(32,32))
    parser.add_argument(
        '--Crop_ImageSize',
        type=int,
        default=(32,32))
    # 防御后结果文件存储路径
    parser.add_argument(
        '--Enhanced_model_save_path',
        type=str,
        default="./defense/")
    parser.add_argument(
        '--config_defense_param_xml_dir',
        type=str,
        default="./defense/EAT/EAT.xml")
    parser.add_argument(
        '--optim_config_dir',
        type=str,
        default="./defense/EAT/EAT_optim.xml")
    parser.add_argument(
        '--config_model_dir_path',
        type=str,
        default="./defense/EAT/EAT_model.xml")
    parser.add_argument(
        '--data_type',
        type=str,
        default='CIFAR10')
    # GPU 设置
```

(下页继续)

(续上页)

```
    parser.add_argument(
        '--GPU_Config',
        type=str,
        # 数目，index 设置
        default=["2","0,1"])
    arguments = parser.parse_args()
    main(args=arguments)
```

其中涉及到的主要参数及说明如下：

defense_method

该参数为防御算法的名称。目前已集成的防御算法为 EAT，NAT，OAT，PAT 以及 RAND。详细说明请见防御算法具体说明一章。

Data_path

共包含四个参数位，分别表示：

1. 样本的数据集文件（白盒攻击下是原始样本，黑盒攻击下是攻击后的攻击样本）
2. 对应的标签的文件（非目标攻击下是 groundtruth，目标攻击下是攻击目标类别）
3. 原始样本的数据集文件
4. 对应的 groundtruth 标签文件

Dict_path

对应的数据集的分类类别号码和类别名称的编号 default= "./dict_lists/cifar10_dict.txt"

model

用户打算使用的加固防御网络模型

batch_size

default =64，读入数据每次批量处理的数目

Enhanced_model_save_path

设置用来保存防御模型的路径，同时，设置相应的配置参数文件的位置

config_defense_param_xml_dir

用来设置防御方法的内部参数的配置文件，以 EAT 加固算法配套的 EAT.xml 文件为例：

```
<method type='EAT'>
    <param title='dataset'>
        <dataset>CIFAR10</dataset>
    </param>

    <param title='epsilon'>
        <epsilon>0.6</epsilon>
    </param>

    <param title='train_externals'>
        <train_externals>False</train_externals>
    </param>
</method>
```

config_model_dir_path

用来设置加固防御用的模型的内部参数的配置文件，以 EAT 加固算法配套的 EAT_model.xml 文件为例：

```
<method type='ANP_VGG16'>
    <param title='batch_size'>
        <batch_size>64</batch_size>
    </param>

    <param title='enable_lat'>
        <enable_lat>False</enable_lat>
    </param>
</method>
```

optim_config_dir

用来设置训练加固防御用的模型的优化器的配置文件，以 EAT 加固算法配套的 EAT_optim.xml 文件为例：

```
<method type='EAT'>
    <param title='optimizer'>
        <optimizer>optim. Adam(model.parameters(), lr=1e-2)</optimizer>
    </param>
```

(下页继续)

(续上页)

```
    <param title='scheduler'>
        <scheduler>optim. lr_scheduler.StepLR(optimizer, 40, gamma=0.1)</scheduler>
    </param>
</method>
```

这里因为优化器的函数比较多，用户需要写成字符串的表达式方式，进入计算的时候会用 eval() 函数调用该方法和设置的值

data_type

数据集类型，目前平台共支持以下三种：

1. CIFAR-10：default= “cifar10”
2. ImageNet：[‘ImageNet’ ， “withoutNormalize”]
3. ImageCustom

数据集的类型，目前是三种，根据实际注明类型，以方便预处理过程，另外使用 ImageNet 类型相似的数据一般不是原始的 ImageNet 数据集，选用 [‘ImageNet’ ， “withoutNormalize”] 时候，不对图像做归一化，用户自行预处理

Scale_ImageSize

default= （32,32）（224,224）

cifar10，cifar100 默认到 32,ImageNet 推荐归一化到 224X224。剩下的用户可以自定义，放缩到多少后再裁减

Crop_ImageSize

default= （32,32）（224,224）

用户定义好的尺寸的缩放后，再裁减的后的尺寸

GPU_Config

default=[“2” ，“0,1”]

第一个表示，GPU 的数目，第二个表示可以使用的 GPU 的编号。目前支持，在判断和实际设备中信息一致的设备中任意选取一个使用

5.3.2 加固防御模型调用运行说明

以 CIFAR-10 数据集在 ResNet2 模型上进行加固为例：

相关文件路径

```
EvalBox
Models
    TestrModels
    UserModels
        utils
        **ResNet2.py** // 使用的模型
    weights
    basic_module
utils
test
    defense
        EAT
        **NAT** // 使用的防御算法
        ...
Datasets
```

执行过程

参照该文档前述的加固模型调用参数说明，修改对应参数内容，使用 Python 语言，执行对应加固训练过程。

```
python test_defense.py
```

防御结果

防御算法使用的是 NAT，结果将会被保存在

```
~/AISafety/test/defense/NAT/CIFAR10_NAT_enhanced.pt
```

使用攻击算法 FGSM 执行测试，首先测试原始的 resnet20_cifar.pt 模型参数文件，得到攻击后准确率为 0.1024 再对 CIFAR10_NAT_enhanced.pt 参数文件进行测试，得到攻击后准确率为 0.2464 可以看出，加固后的网络对对抗样本的攻击防御有一部分上升

CHAPTER 6

平台模型介绍

本平台目前集成了部分测试模型，并支持用户自行上传个人模型。

6.1 测试模型

6.1.1 已集成内容

目前项目已集成的测试模型共有以下几个：

- 针对 Cifar-10 数据集的 ResNet20，FP_ResNet，VGG16
- 针对 ImageNet 数据集的 VGG19

目前项目已集成的模型参数共有以下几个：

- resnet20 模型在 cifar10 原始训练集上模型参数
- resnet20 模型通过 RAND 加固算法加固后模型参数
- FP_ResNet 模型在 cifar10 原始训练集上模型参数
- VGG16 模型在 cifar10 原始训练集上模型参数
- VGG16 模型通过 RAND 加固算法加固后模型参数

6.1.2 自定义模型

用户可按照模型扩展要求，实现并上传自定义模型。

也可默认 l 使用 pytorch 自带的模型，以及默认的预训练模型，具体有例如：

- torchvision.model.vgg19
- torchvision.model.alexnet

6.2 扩展模型

6.2.1 模型存储位置

```
EvalBox
Models
    TestModels     测试用模型存储路径
    UserModels     用户个人模型存储路径
        utils
        ResNet2.py
        new_model.py  用户新上传模型
    weights
        resnet20_cifar.pt
        new_weights.pt  用户新上传模型对应参数文件
    basic_module
utils
test
Datasets
```

6.2.2 模型扩展要求

如下为一个可使用模型实例：

```
import ...

from Models.basic_module import BasicModule

# 用户可自定义若干辅助方法
def adjust_learning_rate(epoch, optimizer):
    ...

def conv3x3(in_planes, out_planes, stride=1):
    ...
```

(下页继续)

(续上页)

```
# 用户可通过类，定义若干个人模型
class BasicBlock(BasicModule):
    ...

class ResNet_Cifar(BasicModule):
    ...

# 用户自定义实现的获取模型的方法
def resnet20_cifar(thermometer = False, level = 1):
    model = ResNet_Cifar(BasicBlock, [3, 3, 3], thermometer = thermometer, level = level)
    # 返回值应为用户实现的模型文件中模型
    return model

# 必须实现的公用方法 getModel
def getModel():
    return resnet20_cifar()
```

模型的格式仅需要遵循，实现 getModel 函数，返回模型。

CHAPTER 7

Indices and tables

- genindex
- modindex
- search