Homework 4: Privacy and Unlearnale examples

助教：张吉哲

邮箱：jizhe.zhang@stu.pku.edu.cn

CONTENTS

01 Brief Introduction

Brief Introduction:

Points

- MIA Attacks(10 points)
- Unlearnable Examples (15 points: 6+4)

Requirements

- Word/pdf/markdown is ok.
- Write a report (at most **8** pages).
- Send your report and code to

  trustworthy_ai@163.com

  Theme: Homework4-name-ID
- In Chinese/ English

  Due: 6/21 24:00

Language and wheel

- Python
- PyTorch

Contents included by the Homework4-name-ID**.zip**

- All python file
- report

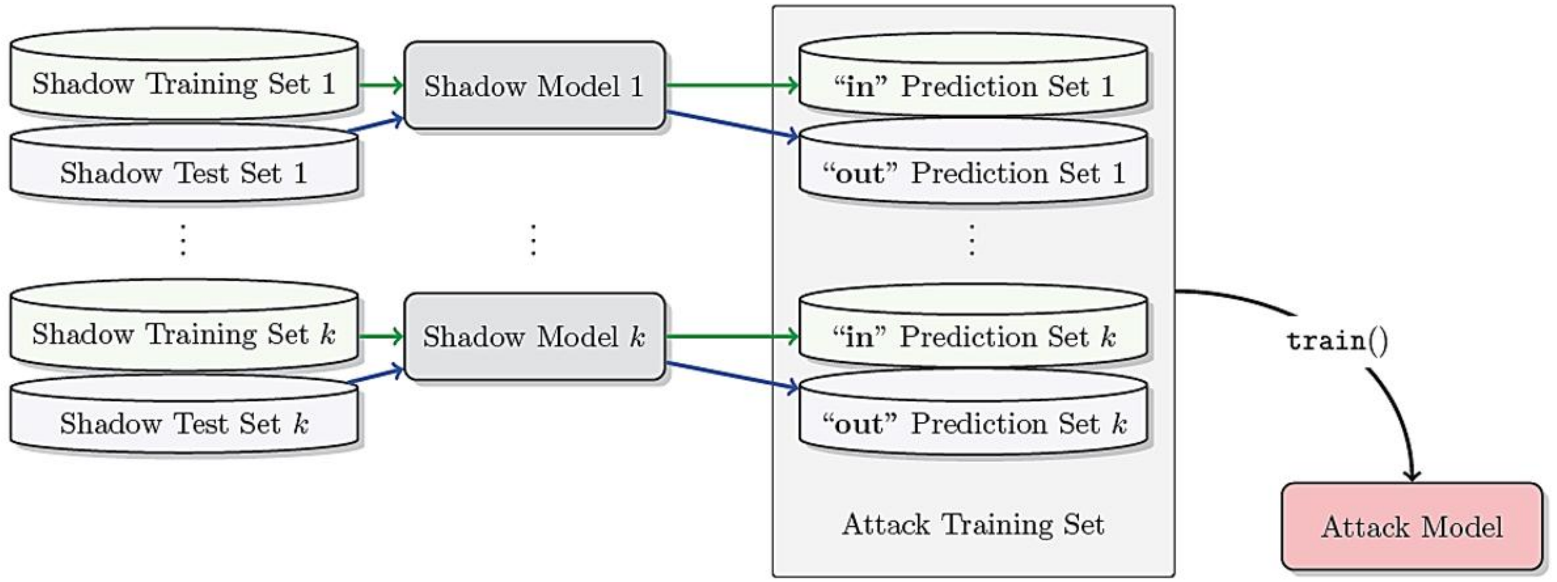
CONTENTS

02 MIA Attack

Main files:

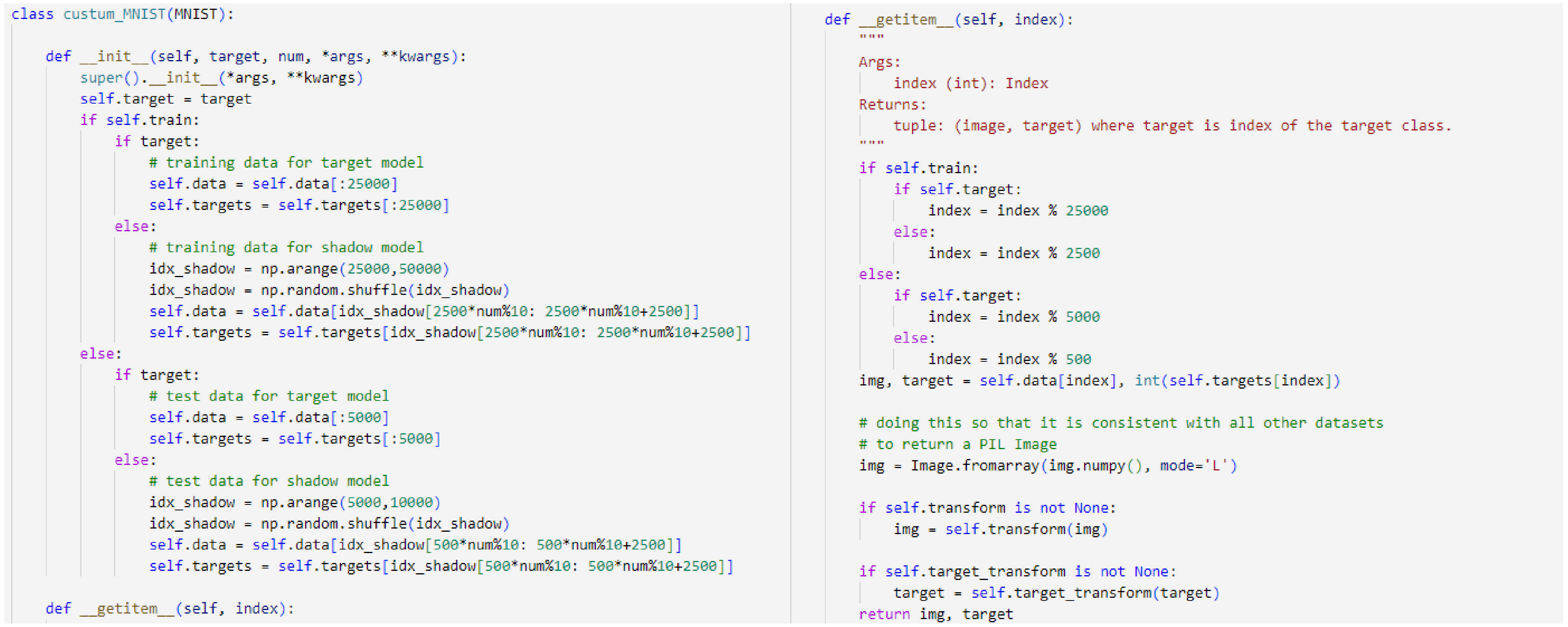
- MIA.ipynb

Objectives:
Apply MIA attacks on MNIST with >85% accuracy



02 MIA Attack

Split Dataset:

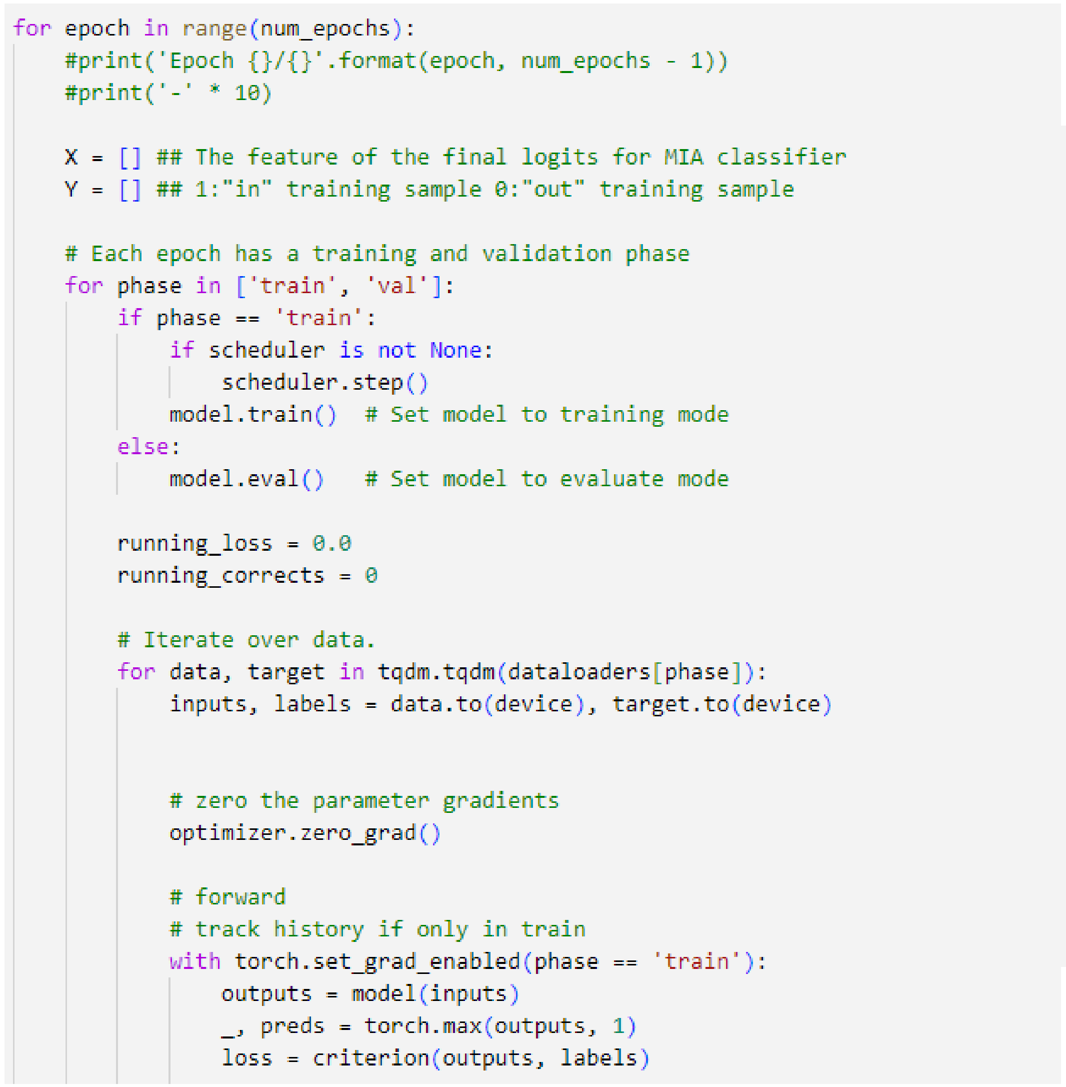


```python
class custum_MNIST(MNIST):

    def __init__(self, target, num, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.target = target
        if self.train:
            if target:
                # training data for target model
                self.data = self.data[:25000]
                self.targets = self.targets[:25000]
            else:
                # training data for shadow model
                idx_shadow = np.arange(25000,50000)
                idx_shadow = np.random.shuffle(idx_shadow)
                self.data = self.data[idx_shadow[2500*num%10: 2500*num%10+2500]]
                self.targets = self.targets[idx_shadow[2500*num%10: 2500*num%10+2500]]
        else:
            if target:
                # test data for target model
                self.data = self.data[:5000]
                self.targets = self.targets[:5000]
            else:
                # test data for shadow model
                idx_shadow = np.arange(5000,10000)
                idx_shadow = np.random.shuffle(idx_shadow)
                self.data = self.data[idx_shadow[500*num%10: 500*num%10+2500]]
                self.targets = self.targets[idx_shadow[500*num%10: 500*num%10+2500]]

    def __getitem__(self, index):
```

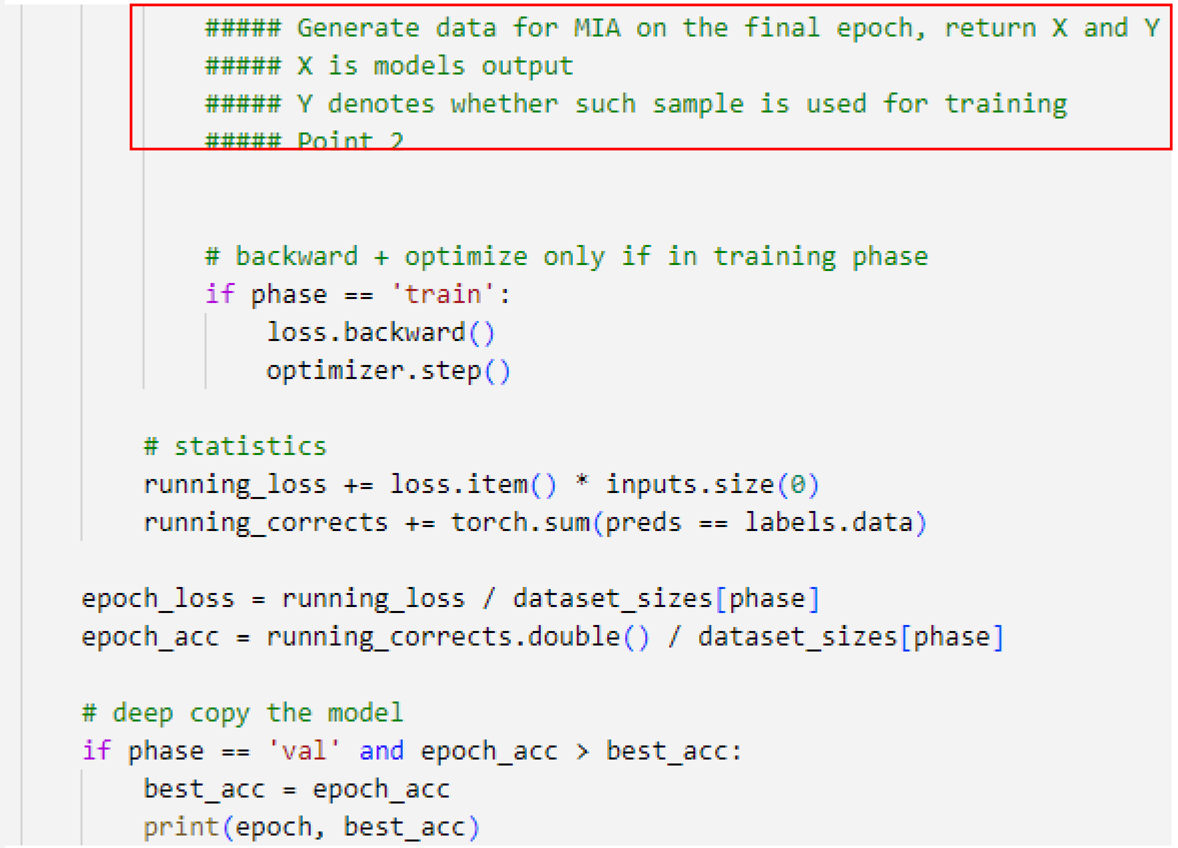
```python
def __getitem__(self, index):
    """
    Args:
        index (int): Index
    Returns:
        tuple: (image, target) where target is index of the target class.
    """
    if self.train:
        if self.target:
            index = index % 25000
        else:
            index = index % 2500
    else:
        if self.target:
            index = index % 5000
        else:
            index = index % 500
    img, target = self.data[index], int(self.targets[index])

    # doing this so that it is consistent with all other datasets
    # to return a PIL Image
    img = Image.fromarray(img.numpy(), mode='L')

    if self.transform is not None:
        img = self.transform(img)

    if self.target_transform is not None:
        target = self.target_transform(target)
    return img, target
```

MIA Attack

Train model and return data for MIA:



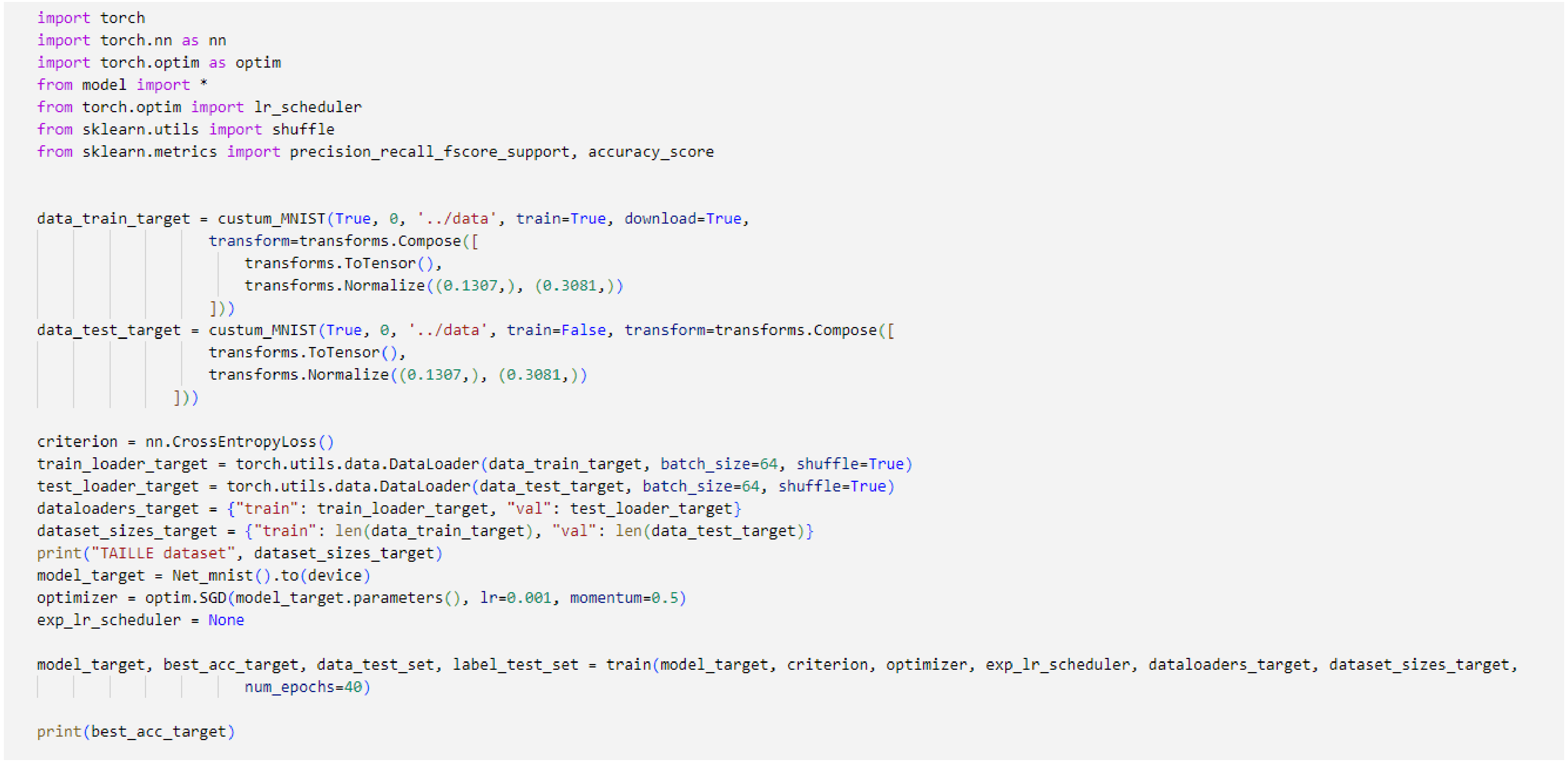
```python
for epoch in range(num_epochs):
    #print('Epoch {}/{}'.format(epoch, num_epochs - 1))
    #print('-' * 10)

    X = [] ## The feature of the final logits for MIA classifier
    Y = [] ## 1:"in" training sample 0:"out" training sample

    # Each epoch has a training and validation phase
    for phase in ['train', 'val']:
        if phase == 'train':
            if scheduler is not None:
                scheduler.step()
            model.train()  # Set model to training mode
        else:
            model.eval()   # Set model to evaluate mode

        running_loss = 0.0
        running_corrects = 0

        # Iterate over data.
        for data, target in tqdm.tqdm(dataloaders[phase]):
            inputs, labels = data.to(device), target.to(device)


            # zero the parameter gradients
            optimizer.zero_grad()

            # forward
            # track history if only in train
            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)
```



```python
                ##### Generate data for MIA on the final epoch, return X and Y
                ##### X is models output
                ##### Y denotes whether such sample is used for training
                ##### Point 2



                # backward + optimize only if in training phase
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            # statistics
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        epoch_loss = running_loss / dataset_sizes[phase]
        epoch_acc = running_corrects.double() / dataset_sizes[phase]

        # deep copy the model
        if phase == 'val' and epoch_acc > best_acc:
            best_acc = epoch_acc
            print(epoch, best_acc)
```

MIA Attack

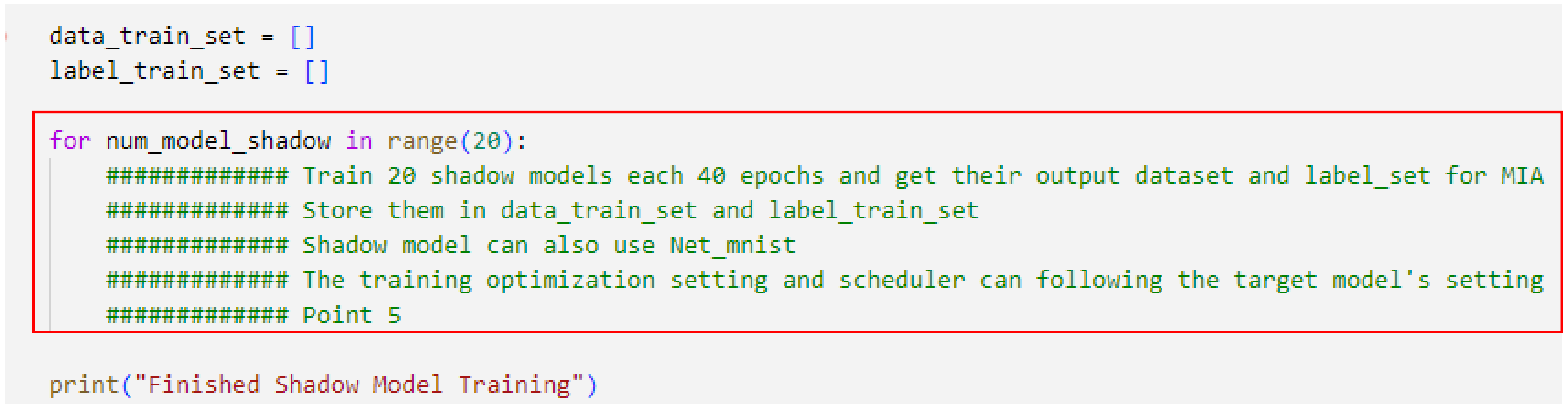
Train target model and use target model's training set and validation set as validation set for MIA



```python
import torch
import torch.nn as nn
import torch.optim as optim
from model import *
from torch.optim import lr_scheduler
from sklearn.utils import shuffle
from sklearn.metrics import precision_recall_fscore_support, accuracy_score


data_train_target = custum_MNIST(True, 0, '../data', train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))
data_test_target = custum_MNIST(True, 0, '../data', train=False, transform=transforms.Compose([
                    transforms.ToTensor(),
                    transforms.Normalize((0.1307,), (0.3081,))
                ]))

criterion = nn.CrossEntropyLoss()
train_loader_target = torch.utils.data.DataLoader(data_train_target, batch_size=64, shuffle=True)
test_loader_target = torch.utils.data.DataLoader(data_test_target, batch_size=64, shuffle=True)
dataloaders_target = {"train": train_loader_target, "val": test_loader_target}
dataset_sizes_target = {"train": len(data_train_target), "val": len(data_test_target)}
print("TAILLE dataset", dataset_sizes_target)
model_target = Net_mnist().to(device)
optimizer = optim.SGD(model_target.parameters(), lr=0.001, momentum=0.5)
exp_lr_scheduler = None

model_target, best_acc_target, data_test_set, label_test_set = train(model_target, criterion, optimizer, exp_lr_scheduler, dataloaders_target, dataset_sizes_target,
                        num_epochs=40)

print(best_acc_target)
```

02 MIA Attack

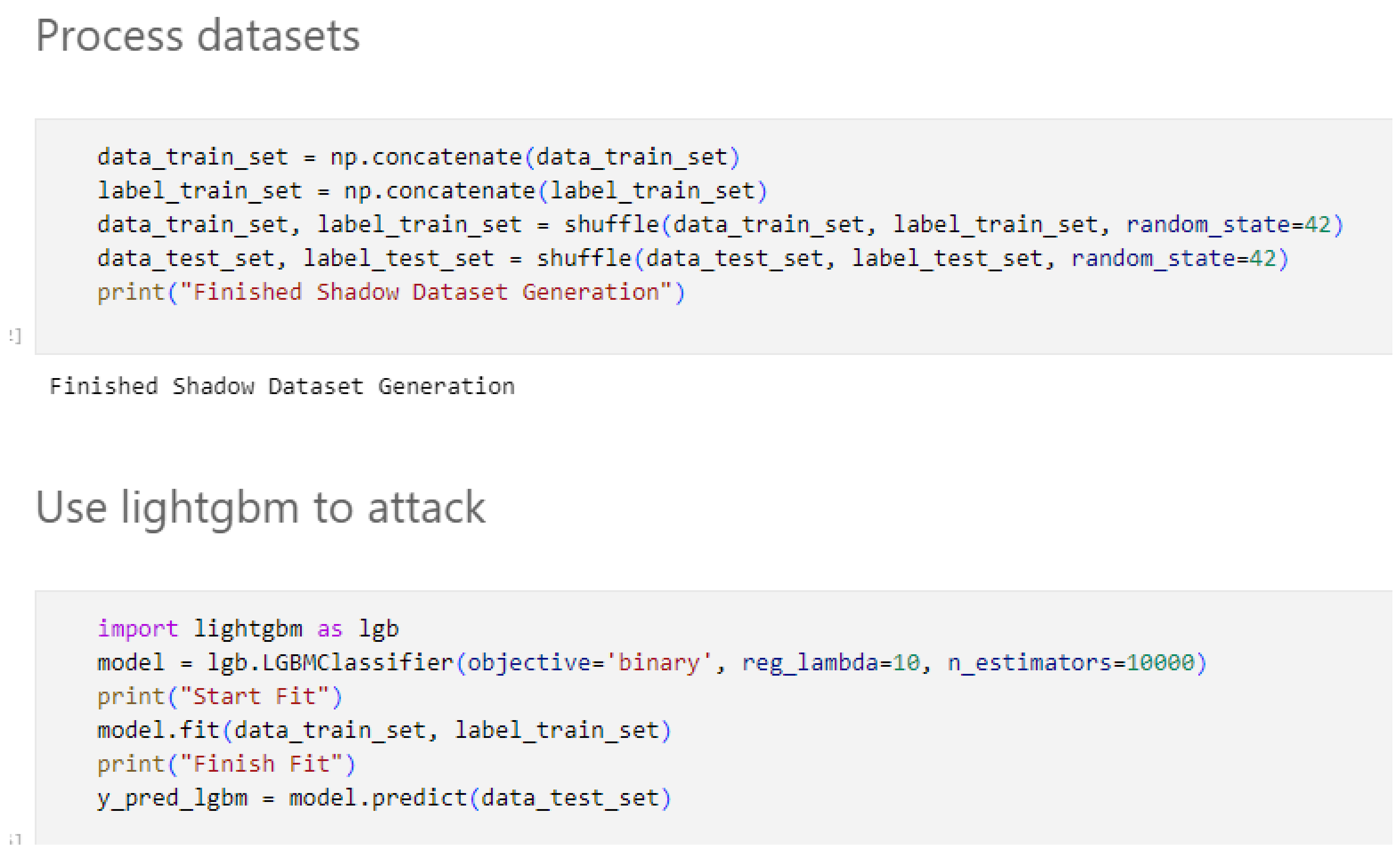
Train shadow model and use shadow model's training set and validation set as training set for MIA



```
data_train_set = []
label_train_set = []

for num_model_shadow in range(20):
    ############## Train 20 shadow models each 40 epochs and get their output dataset and label_set for MIA
    ############## Store them in data_train_set and label_train_set
    ############## Shadow model can also use Net_mnist
    ############## The training optimization setting and scheduler can following the target model's setting
    ############## Point 5

print("Finished Shadow Model Training")
```

02 MIA Attack

Process data and attack



Process datasets

```
data_train_set = np.concatenate(data_train_set)
label_train_set = np.concatenate(label_train_set)
data_train_set, label_train_set = shuffle(data_train_set, label_train_set, random_state=42)
data_test_set, label_test_set = shuffle(data_test_set, label_test_set, random_state=42)
print("Finished Shadow Dataset Generation")
```

```
Finished Shadow Dataset Generation
```

Use lightgbm to attack

```
import lightgbm as lgb
model = lgb.LGBMClassifier(objective='binary', reg_lambda=10, n_estimators=10000)
print("Start Fit")
model.fit(data_train_set, label_train_set)
print("Finish Fit")
y_pred_lgbm = model.predict(data_test_set)
```

02 Results

- Report: Tell how your code works: 3 points
- Correctness of Code: 7 points(2+5)
- Besides the report, you should also hand in your code.
- Do not hand in your checkpoint.

CONTENTS

03

Unlearnable examples

Main files:

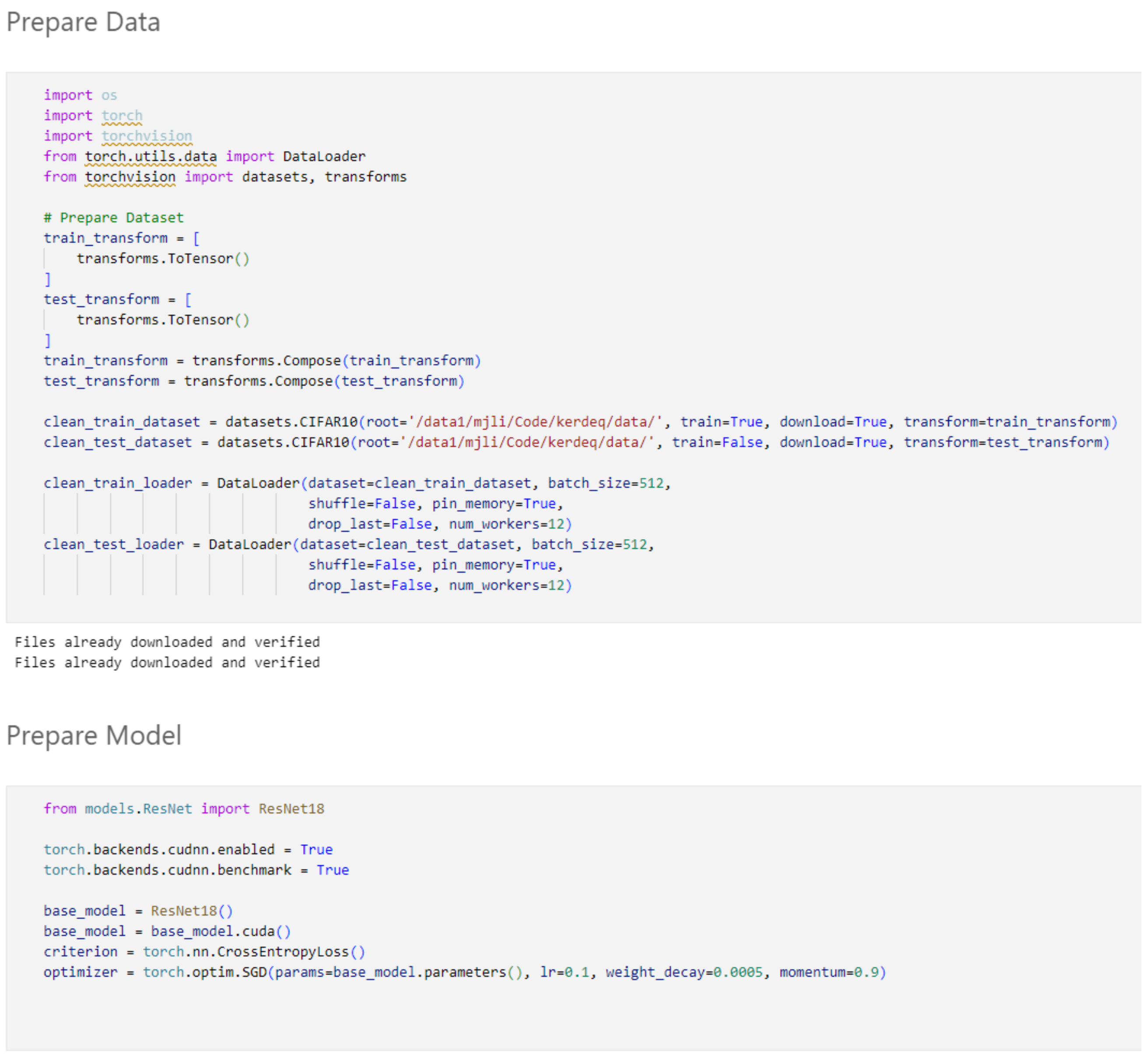
- Classwise.ipynb

- Samplewise.ipynb

Objectives:
Final classification accuracy on generated dataset less than 40%.

03 Unlearnable Examples

Classwise min-min:



Prepare Data

```python
import os
import torch
import torchvision
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

# Prepare Dataset
train_transform = [
    transforms.ToTensor()
]
test_transform = [
    transforms.ToTensor()
]
train_transform = transforms.Compose(train_transform)
test_transform = transforms.Compose(test_transform)

clean_train_dataset = datasets.CIFAR10(root='/data1/mjli/Code/kerdeq/data/', train=True, download=True, transform=train_transform)
clean_test_dataset = datasets.CIFAR10(root='/data1/mjli/Code/kerdeq/data/', train=False, download=True, transform=test_transform)

clean_train_loader = DataLoader(dataset=clean_train_dataset, batch_size=512,
                                shuffle=False, pin_memory=True,
                                drop_last=False, num_workers=12)
clean_test_loader = DataLoader(dataset=clean_test_dataset, batch_size=512,
                                shuffle=False, pin_memory=True,
                                drop_last=False, num_workers=12)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

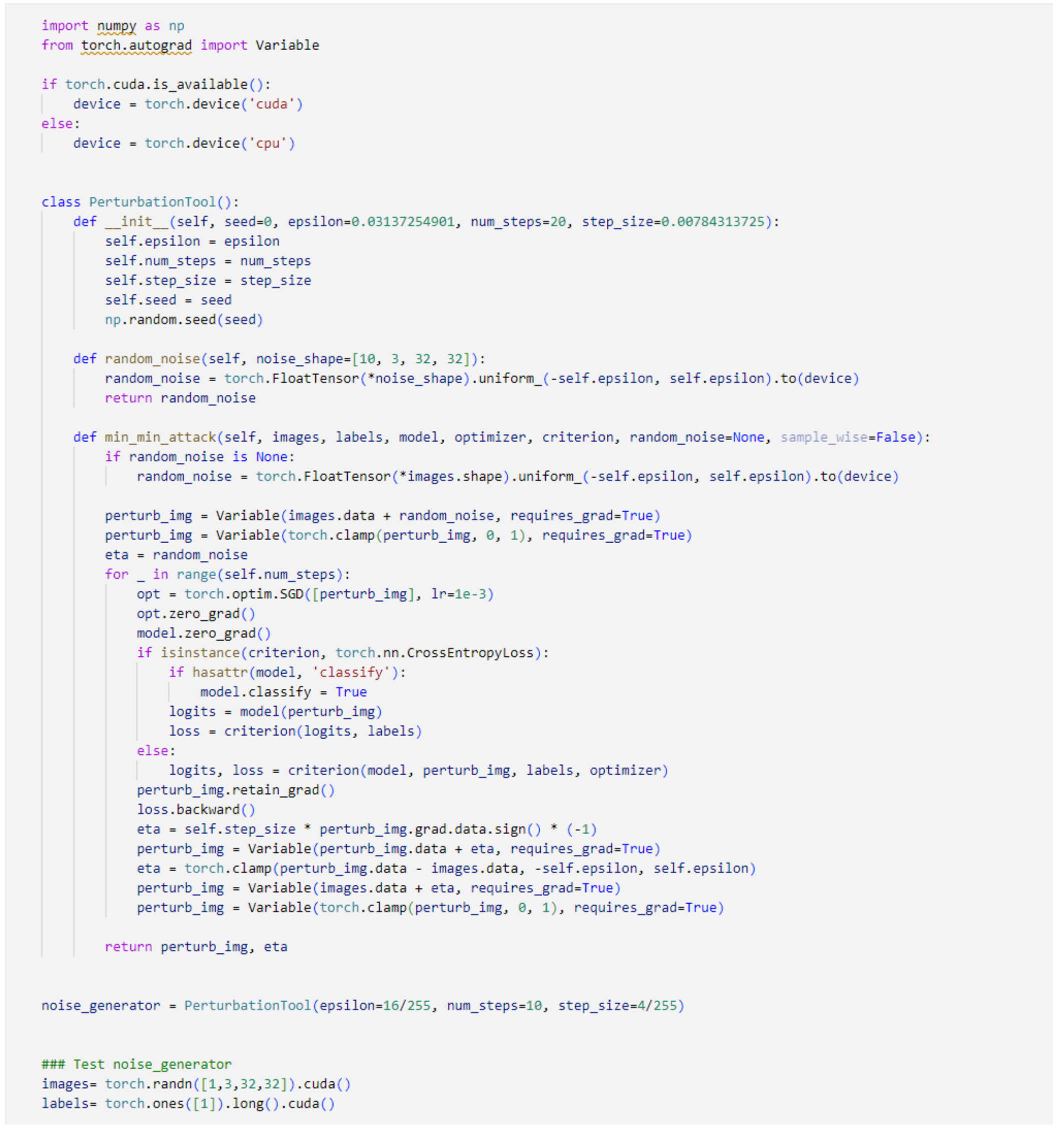
Prepare Model

```python
from models.ResNet import ResNet18

torch.backends.cudnn.enabled = True
torch.backends.cudnn.benchmark = True

base_model = ResNet18()
base_model = base_model.cuda()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(params=base_model.parameters(), lr=0.1, weight_decay=0.0005, momentum=0.9)
```

03 Unlearnable Examples

Classwise min-min:

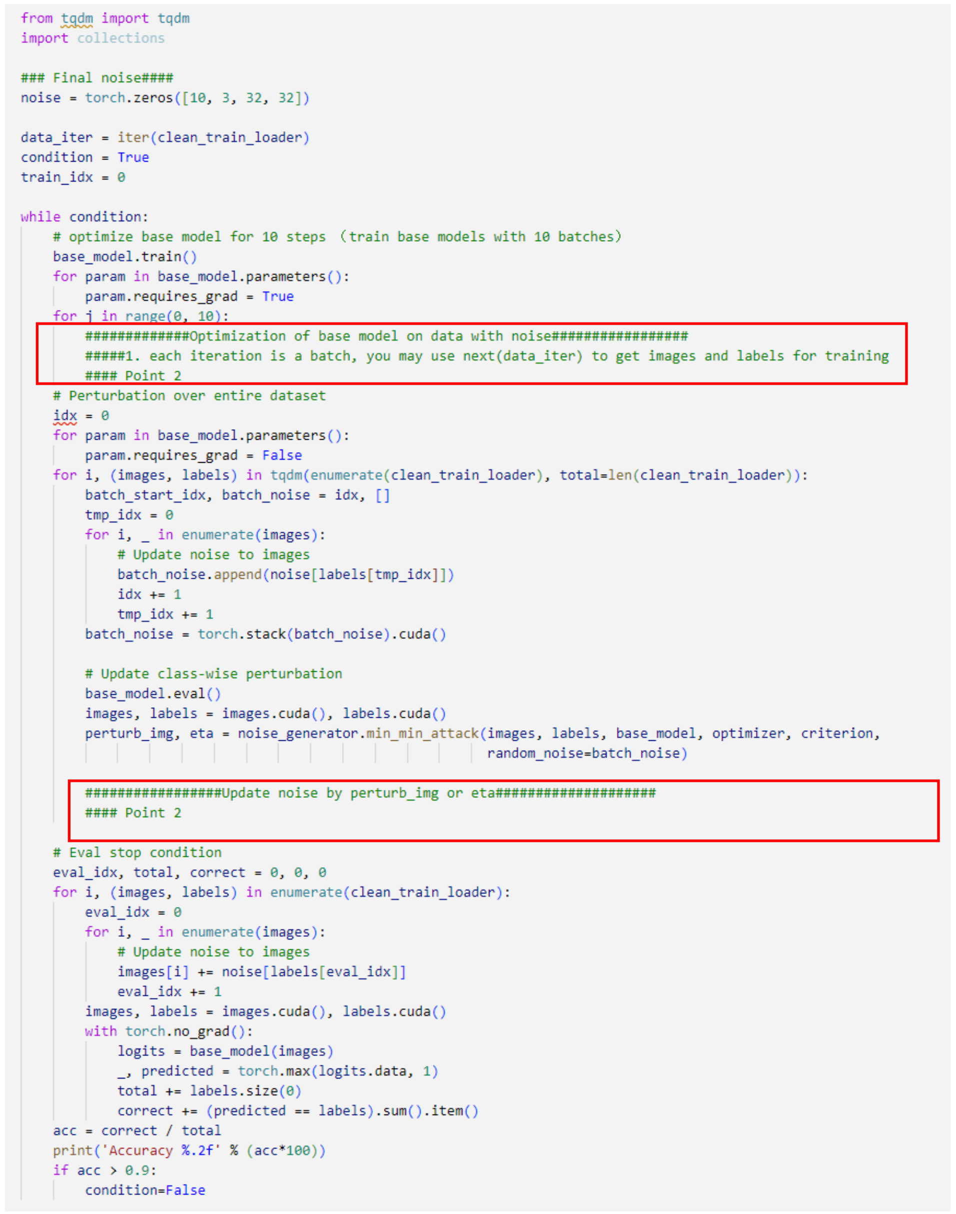
Prepare Perturbation Generator Tool



```
import numpy as np
from torch.autograd import Variable

if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')


class PerturbationTool():
    def __init__(self, seed=0, epsilon=0.03137254901, num_steps=20, step_size=0.00784313725):
        self.epsilon = epsilon
        self.num_steps = num_steps
        self.step_size = step_size
        self.seed = seed
        np.random.seed(seed)

    def random_noise(self, noise_shape=[10, 3, 32, 32]):
        random_noise = torch.FloatTensor(*noise_shape).uniform_(-self.epsilon, self.epsilon).to(device)
        return random_noise

    def min_min_attack(self, images, labels, model, optimizer, criterion, random_noise=None, sample_wise=False):
        if random_noise is None:
            random_noise = torch.FloatTensor(*images.shape).uniform_(-self.epsilon, self.epsilon).to(device)

        perturb_img = Variable(images.data + random_noise, requires_grad=True)
        perturb_img = Variable(torch.clamp(perturb_img, 0, 1), requires_grad=True)
        eta = random_noise
        for _ in range(self.num_steps):
            opt = torch.optim.SGD([perturb_img], lr=1e-3)
            opt.zero_grad()
            model.zero_grad()
            if isinstance(criterion, torch.nn.CrossEntropyLoss):
                if hasattr(model, 'classify'):
                    model.classify = True
                logits = model(perturb_img)
                loss = criterion(logits, labels)
            else:
                logits, loss = criterion(model, perturb_img, labels, optimizer)
            perturb_img.retain_grad()
            loss.backward()
            eta = self.step_size * perturb_img.grad.data.sign() * (-1)
            perturb_img = Variable(perturb_img.data + eta, requires_grad=True)
            eta = torch.clamp(perturb_img.data - images.data, -self.epsilon, self.epsilon)
            perturb_img = Variable(images.data + eta, requires_grad=True)
            perturb_img = Variable(torch.clamp(perturb_img, 0, 1), requires_grad=True)

        return perturb_img, eta


noise_generator = PerturbationTool(epsilon=16/255, num_steps=10, step_size=4/255)


### Test noise_generator
images= torch.randn([1,3,32,32]).cuda()
labels= torch.ones([1]).long().cuda()
```

03 Unlearnable Examples

Classwise min-min:

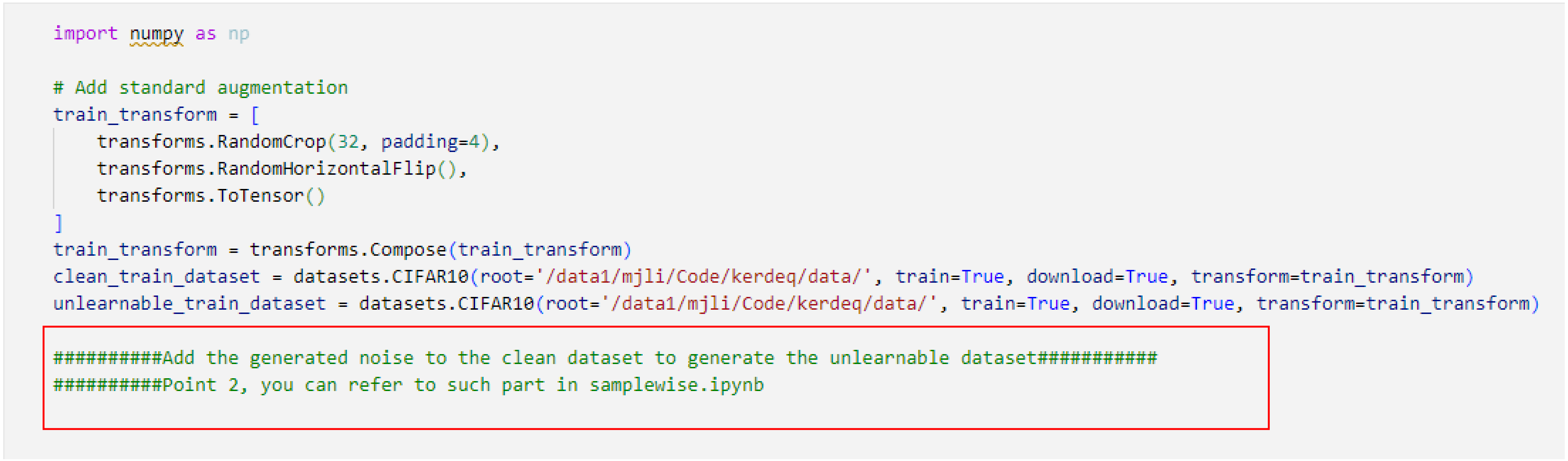


```
from tqdm import tqdm
import collections

### Final noise####
noise = torch.zeros([10, 3, 32, 32])

data_iter = iter(clean_train_loader)
condition = True
train_idx = 0

while condition:
    # optimize base model for 10 steps (train base models with 10 batches)
    base_model.train()
    for param in base_model.parameters():
        param.requires_grad = True
    for j in range(0, 10):
        #############Optimization of base model on data with noise#################
        #####1. each iteration is a batch, you may use next(data_iter) to get images and labels for training
        #### Point 2
    # Perturbation over entire dataset
    idx = 0
    for param in base_model.parameters():
        param.requires_grad = False
    for i, (images, labels) in tqdm(enumerate(clean_train_loader), total=len(clean_train_loader)):
        batch_start_idx, batch_noise = idx, []
        tmp_idx = 0
        for i, _ in enumerate(images):
            # Update noise to images
            batch_noise.append(noise[labels[tmp_idx]])
            idx += 1
            tmp_idx += 1
        batch_noise = torch.stack(batch_noise).cuda()

        # Update class-wise perturbation
        base_model.eval()
        images, labels = images.cuda(), labels.cuda()
        perturb_img, eta = noise_generator.min_min_attack(images, labels, base_model, optimizer, criterion,
                                                          random_noise=batch_noise)

        #################Update noise by perturb_img or eta####################
        #### Point 2

    # Eval stop condition
    eval_idx, total, correct = 0, 0, 0
    for i, (images, labels) in enumerate(clean_train_loader):
        eval_idx = 0
        for i, _ in enumerate(images):
            # Update noise to images
            images[i] += noise[labels[eval_idx]]
            eval_idx += 1
        images, labels = images.cuda(), labels.cuda()
        with torch.no_grad():
            logits = base_model(images)
            _, predicted = torch.max(logits.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    acc = correct / total
    print('Accuracy %.2f' % (acc*100))
    if acc > 0.9:
        condition=False
```

03 Unlearnable Examples

Classwise min-min:

Creat Unlearnable class-wise min-min Dataset

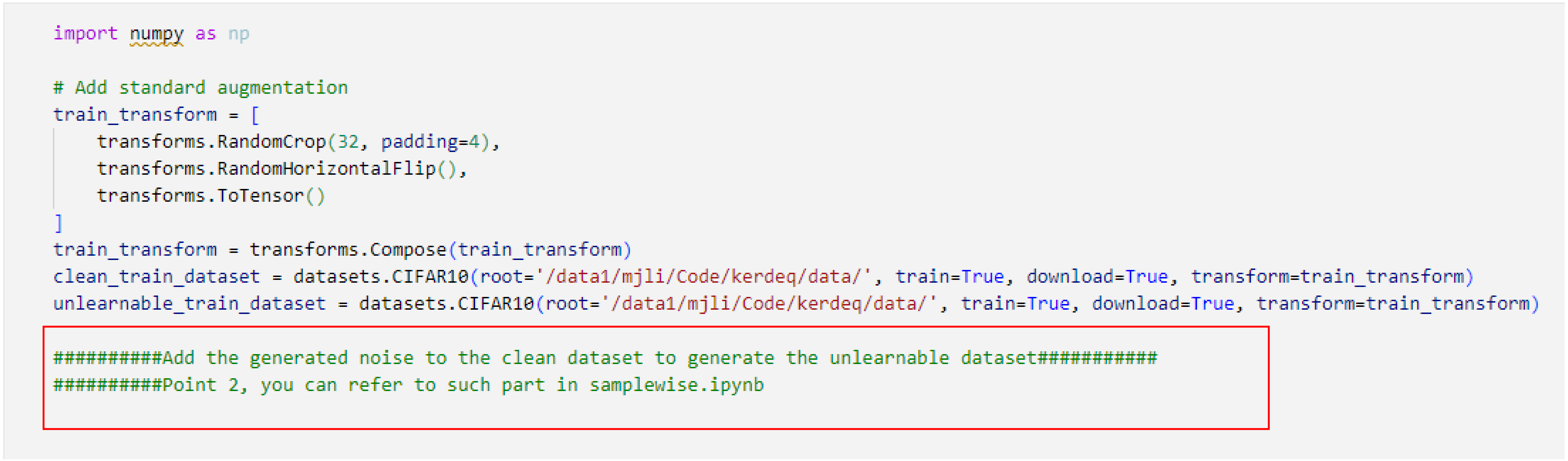


```
import numpy as np

# Add standard augmentation
train_transform = [
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor()
]
train_transform = transforms.Compose(train_transform)
clean_train_dataset = datasets.CIFAR10(root='/data1/mjli/Code/kerdeq/data/', train=True, download=True, transform=train_transform)
unlearnable_train_dataset = datasets.CIFAR10(root='/data1/mjli/Code/kerdeq/data/', train=True, download=True, transform=train_transform)

##########Add the generated noise to the clean dataset to generate the unlearnable dataset###########
##########Point 2, you can refer to such part in samplewise.ipynb
```

03 Unlearnable Examples

Classwise min-min:

Creat Unlearnable class-wise min-min Dataset

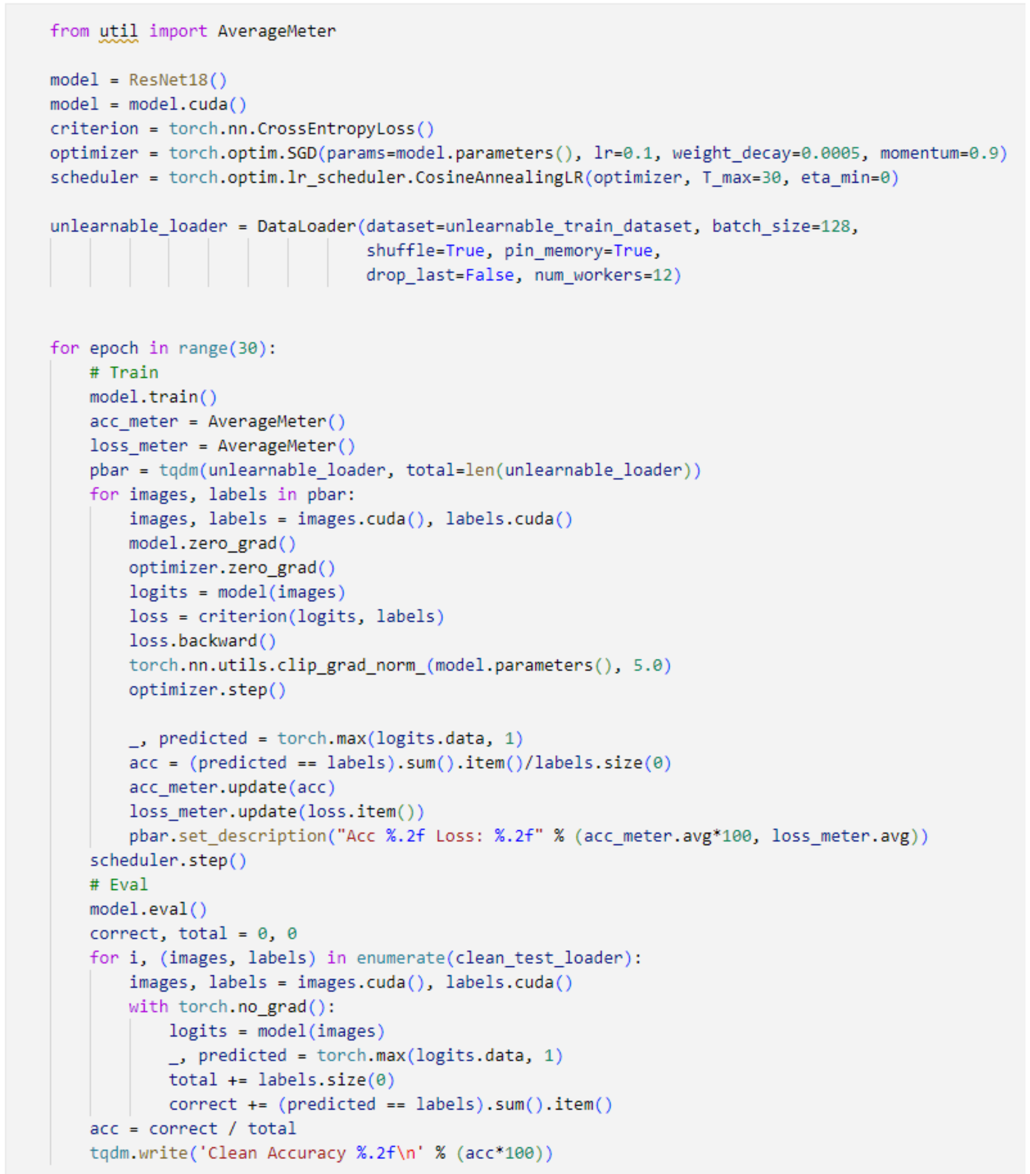


```
import numpy as np

# Add standard augmentation
train_transform = [
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor()
]
train_transform = transforms.Compose(train_transform)
clean_train_dataset = datasets.CIFAR10(root='/data1/mjli/Code/kerdeq/data/', train=True, download=True, transform=train_transform)
unlearnable_train_dataset = datasets.CIFAR10(root='/data1/mjli/Code/kerdeq/data/', train=True, download=True, transform=train_transform)

##########Add the generated noise to the clean dataset to generate the unlearnable dataset###########
##########Point 2, you can refer to such part in samplewise.ipynb
```

03 Unlearnable Examples

Classwise min-min:

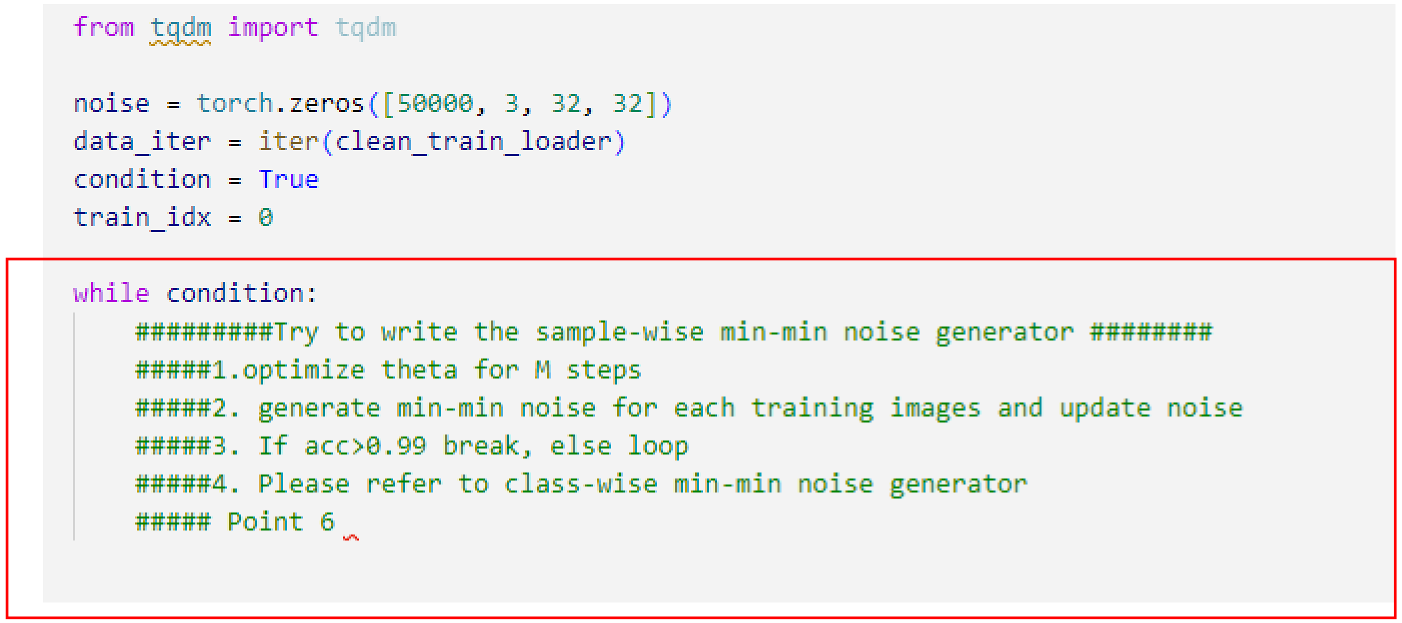
Train ResNet18 on Unlearnable Dataset



```
from util import AverageMeter

model = ResNet18()
model = model.cuda()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(params=model.parameters(), lr=0.1, weight_decay=0.0005, momentum=0.9)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=30, eta_min=0)

unlearnable_loader = DataLoader(dataset=unlearnable_train_dataset, batch_size=128,
                                shuffle=True, pin_memory=True,
                                drop_last=False, num_workers=12)


for epoch in range(30):
    # Train
    model.train()
    acc_meter = AverageMeter()
    loss_meter = AverageMeter()
    pbar = tqdm(unlearnable_loader, total=len(unlearnable_loader))
    for images, labels in pbar:
        images, labels = images.cuda(), labels.cuda()
        model.zero_grad()
        optimizer.zero_grad()
        logits = model(images)
        loss = criterion(logits, labels)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 5.0)
        optimizer.step()

        _, predicted = torch.max(logits.data, 1)
        acc = (predicted == labels).sum().item()/labels.size(0)
        acc_meter.update(acc)
        loss_meter.update(loss.item())
        pbar.set_description("Acc %.2f Loss: %.2f" % (acc_meter.avg*100, loss_meter.avg))
    scheduler.step()
    # Eval
    model.eval()
    correct, total = 0, 0
    for i, (images, labels) in enumerate(clean_test_loader):
        images, labels = images.cuda(), labels.cuda()
        with torch.no_grad():
            logits = model(images)
            _, predicted = torch.max(logits.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    acc = correct / total
    tqdm.write('Clean Accuracy %.2f\n' % (acc*100))
```

Unlearnable Examples

samplewise min-min:



```
from tqdm import tqdm

noise = torch.zeros([50000, 3, 32, 32])
data_iter = iter(clean_train_loader)
condition = True
train_idx = 0

while condition:
    #########Try to write the sample-wise min-min noise generator ########
    #####1.optimize theta for M steps
    #####2. generate min-min noise for each training images and update noise
    #####3. If acc>0.99 break, else loop
    #####4. Please refer to class-wise min-min noise generator
    ##### Point 6
```

02 Results

- Report: Tell how your code works: 3 points
- Correctness of Code: 12 points(6+2+2+2)
- Besides the report, you should also hand in your code.
- Do not hand in your checkpoint.

论文列表

[1] Membership Inference Attacks against Machine Learning Models
Reza Shokri, Marco Stronati, Congzheng Song, Vitaly Shmatikov

[2] Unlearnable Examples: Making Personal Data Unexploitable
Hanxun Huang, Xingjun Ma, Sarah Monazam Erfani, James Bailey, Yisen Wang

Homework3注意事项

报告：

只写了操作过程，没有报告实验结果

实验操作：

对图像数值的操作需要注意：

- 确认输入/输出是 0-1（浮点） 还是 0-255（整数）
- 检查数组类型(如 uint8/float32)，操作前需统一类型，防止溢出或精度丢失(如uint8 运算前建议转 float)。
- 图像数值运算过程中数值可能超出范围，建议用np.clip()限制。

调参问题：

1. 确定合理范围后写一个bash脚本调参
2. 达不到作业要求，可能是有小bug/或者环境问题

Thanks