WaNet 代码复现

目标模型：PreactResNet18
数据集：GTSRB 数据集

代码理解：

本项目重点代码文件如下：
1. config.py　配置文件，包含项目各种参数配置。
2. dataloader.py　数据加载文件，加载数据集。
3.train.py 训练代码，训练轮数默认为 1000 轮。
4.eval.py　评估代码，在 GTSRB 数据集上对模型进行评估（无防御机制）。
其他的代码文件即防御模型、分类模型、各种网络架构等，并不是这个项目的提出重点。

1. config.py

```
parser.add_argument("--data_root", type=str, default="/home/ubuntu/temps/")
parser.add_argument("--checkpoints", type=str, default="./checkpoints")
parser.add_argument("--temps", type=str, default="./temps")
parser.add_argument("--device", type=str, default="cuda")
parser.add_argument("--continue_training", action="store_true")
```

目标数据集的文件夹路径目录，默认是/home/ubuntu/temps/，后续还会对路径进行处理，以追踪到最终路径。使用--data_root 可以自己指定路径。剩余同理。

```
parser.add_argument("--dataset", type=str, default="cifar10")
parser.add_argument("--attack_mode", type=str, default="all2one")
```

默认数据集是 cifar10，但注意要提前下载才可以用；默认分类任务是 all2one，可以转化为 all2all 任务。

```
parser.add_argument("--n_iters", type=int, default=1000)
```

默认训练迭代次数为 1000 轮，可以修改。

2. datalaoder.py

在 class ProbTransform 中

```
def get_transform(opt, train=True, pretensor_transform=False):
    transforms_list = []
    transforms_list.append(transforms.Resize((opt.input_height, opt.input_width)))
    if pretensor_transform:
        if train:
            transforms_list.append(transforms.RandomCrop((opt.input_height, opt.input_width), pad
            transforms_list.append(transforms.RandomRotation(opt.random_rotation))
            if opt.dataset == "cifar10":
                transforms_list.append(transforms.RandomHorizontalFlip(p=0.5))

    transforms_list.append(transforms.ToTensor())
    if opt.dataset == "cifar10":
        transforms_list.append(transforms.Normalize([0.4914, 0.4822, 0.4465], [0.247, 0.243, 0.26
    elif opt.dataset == "mnist":
        transforms_list.append(transforms.Normalize([0.5], [0.5]))
    elif opt.dataset == "gtsrb" or opt.dataset == "celeba":
        pass
    else:
        raise Exception("Invalid Dataset")
    return transforms.Compose(transforms_list)
```

创建数据预处理和数据增强的转换管道，以便在模型训练和测试中使用。这些转换操作属于数据预处理的一部分，有助于标准化和增强数据，以提高模型的性能和泛化能力。其中有不同数据集下的不同的标准化设置。各种 RandomCrop 函数表示对数据进行裁剪、旋转、翻转，这是进行数据增强操作。

在 class PostTensorTransform 中

```
    def __init__(self, opt):
        super(PostTensorTransform, self).__init__()
        self.random_crop = ProbTransform(
            A.RandomCrop((opt.input_height, opt.input_width), padding=opt.random_crop), p=0.8
        )
        self.random_rotation = ProbTransform(A.RandomRotation(opt.random_rotation), p=0.5)
        if opt.dataset == "cifar10":
            self.random_horizontal_flip = A.RandomHorizontalFlip(p=0.5)

    def forward(self, x):
        for module in self.children():
            x = module(x)
        return x
```

PostTensorTransform 是一个自定义的 PyTorch 模块，用于在将图像转换为张量之后应用一些数据增强和预处理操作。它继承自 torch.nn.Module，并在 forward 方法中应用一系列转换, 同样也是进行数据增强操作。返回处理后的图像。

在 class GTSRB 中

（1）

```
def __init__(self, opt, train, transforms):
    super(GTSRB, self).__init__()
    if train:
        self.data_folder = os.path.join(opt.data_root, "GTSRB/Train")
        self.images, self.labels = self._get_data_train_list()
    else:
        self.data_folder = os.path.join(opt.data_root, "GTSRB/Test")
        self.images, self.labels = self._get_data_test_list()

    self.transforms = transforms
```

拼接目标文件的地址，即 os.path.join()函数

（2）

```
def _get_data_train_list(self):
    images = []
    labels = []
    for c in range(0, 43):
        prefix = self.data_folder + "/" + format(c, "05d") + "/"
        gtFile = open(prefix + "GT-" + format(c, "05d") + ".csv")
        gtReader = csv.reader(gtFile, delimiter=";")
        next(gtReader)
        for row in gtReader:
            images.append(prefix + row[0])
            labels.append(int(row[7]))
        gtFile.close()
    return images, labels
```

从特定的文件夹结构中读取图像和标签，并将它们分别存储在两个列表中，构建图像数据集的路径和标签列表，即 images 和 labels 两个列表。
c 循环表示从 0 到 42，共 43 个类别。
后面有_get_data_test_list 函数，是用来构建测试用的数据，与这个训练函数构建差不多。
返回两个列表。

（3）

```
def get_dataloader(opt, train=True, pretensor_transform=False):
    transform = get_transform(opt, train, pretensor_transform)
    if opt.dataset == "gtsrb":
        dataset = GTSRB(opt, train, transform)
    elif opt.dataset == "mnist":
        dataset = torchvision.datasets.MNIST(opt.data_root, train, transform, download=True)
    elif opt.dataset == "cifar10":
        dataset = torchvision.datasets.CIFAR10(opt.data_root, train, transform, download=True)
    elif opt.dataset == "celeba":
        if train:
            split = "train"
        else:
            split = "test"
        dataset = CelebA_attr(opt, split, transform)
    else:
        raise Exception("Invalid dataset")
    dataloader = torch.utils.data.DataLoader(dataset, batch_size=opt.bs, num_workers=opt.num_work
    return dataloader
```

初始化（加载）数据集。根据数据集的不同类型（gtsrb、mnist、cifar10、celeba）初始化相应的数据集，并返回一个数据加载器，用于批量加载训练或测试数据。其中，MNIST 和 CIFAR10 是库自带的数据集，不需要下载。
get_transform 是调用前面的函数，来获取指定数据集和其对应任务的数据预处理操作。
后面的 if 判断，即是对应的数据集，加上说明是训练还是测试。
DataLoader 函数即数据加载过程，其中 batch_size 表示从数据集中获取的批量大小；num_workers 表示获取的数据加载工作线程数量；后面的 shuffle 表示是否在每个 epoch 开始时对数据进行随机打乱。
最后返回数据加载器

（4）

```
def get_dataset(opt, train=True):
    if opt.dataset == "gtsrb":
        dataset = GTSRB(
            opt,
            train,
            transforms=transforms.Compose([transforms.Resize((opt.input_height, opt.input_width))
        )
    elif opt.dataset == "mnist":
        dataset = torchvision.datasets.MNIST(opt.data_root, train, transform=ToNumpy(), download=
    elif opt.dataset == "cifar10":
        dataset = torchvision.datasets.CIFAR10(opt.data_root, train, transform=ToNumpy(), downloa
    elif opt.dataset == "celeba":
        if train:
            split = "train"
        else:
            split = "test"
        dataset = CelebA_attr(
            opt,
            split,
            transforms=transforms.Compose([transforms.Resize((opt.input_height, opt.input_width))
        )
    else:
        raise Exception("Invalid dataset")
    return dataset
```

获取数据集。根据给定的配置（opt）创建并返回相应的数据集对象。If 判断就是根据各种数据集，创建其相对应的数据集实例，最后返回该实例。获取数据集，MNIST 和 CIFAR10 是库自带的，另外两个要用就必须下载下来。

3. train.py

```
def get_model(opt):
    netC = None
    optimizerC = None
    schedulerC = None

    if opt.dataset == "cifar10" or opt.dataset == "gtsrb":
        netC = PreActResNet18(num_classes=opt.num_classes).to(opt.device)
    if opt.dataset == "celeba":
        netC = ResNet18().to(opt.device)
    if opt.dataset == "mnist":
        netC = NetC_MNIST().to(opt.device)

    # Optimizer
    optimizerC = torch.optim.SGD(netC.parameters(), opt.lr_C, momentum=0.9, weight_decay=5e-4)

    # Scheduler
    schedulerC = torch.optim.lr_scheduler.MultiStepLR(optimizerC, opt.schedulerC_milestones, opt.

    return netC, optimizerC, schedulerC
```

根据给定的配置（opt）初始化一个神经网络模型，以及与之对应的优化器和学习率调度器。

netc 表示存储模型, if 判断表示根据不同的数据集来定义相应的神经网络模型；
optimizierC 表示优化器, 用随机梯度下降（SGD）来优化训练模型 netC；
schedulerC 表示学习率调速器，使用 MultiStepLR 学习率调度器，milestone 是预先设定的里程碑点，到了这个点后，学习率将乘以 opt.schedulerC_lambda。
返回 netC、optimizerC、schedulerC。

trian（）函数

一个典型的神经网络训练过程，其中包含了处理带有后门数据的训练步骤。具体如下：

（1）

```
netC.train()
rate_bd = opt.pc
total_loss_ce = 0
total_sample = 0

total_clean = 0
total_bd = 0
total_cross = 0
total_clean_correct = 0
total_bd_correct = 0
total_cross_correct = 0
criterion_CE = torch.nn.CrossEntropyLoss()
criterion_BCE = torch.nn.BCELoss()

denormalizer = Denormalizer(opt)
transforms = PostTensorTransform(opt).to(opt.device)
total_time = 0

avg_acc_cross = 0
```

将模型 netC 设置为训练模式，会影响某些层（如 Dropout 和 BatchNorm）的行为。
rate_bd 是注入后门数据的比例；
后定义用于统计损失和准确率的变量，即 total_clean 等。
criterion_CE 和 criterion_BCE 是用于计算交叉熵损失和二元交叉熵损失的损失函数。
denormalizer 和 transforms 是用于图像处理的工具；
total_time 用于记录训练过程中计算的时间。
avg_acc_across 是交叉扰动数据 的平均准确率，而交叉数据是通过将网格扰动（grid_temps2）应用于输入图像（inputs_cross）来生成的。

（2）

```
for batch_idx, (inputs, targets) in enumerate(train_dl):
    optimizerC.zero_grad()

    inputs, targets = inputs.to(opt.device), targets.to(opt.device)
    bs = inputs.shape[0]

    # Create backdoor data
    num_bd = int(bs * rate_bd)
    num_cross = int(num_bd * opt.cross_ratio)
    grid_temps = (identity_grid + opt.s * noise_grid / opt.input_height) * opt.grid_rescale
    grid_temps = torch.clamp(grid_temps, -1, 1)

    ins = torch.rand(num_cross, opt.input_height, opt.input_height, 2).to(opt.device) * 2 - 1
    grid_temps2 = grid_temps.repeat(num_cross, 1, 1, 1) + ins / opt.input_height
    grid_temps2 = torch.clamp(grid_temps2, -1, 1)
```

进行迭代训练，遍历训练数据加载器 train_dl；
optimizerC.zero_grad() 用于清除梯度信息，在使用 PyTorch 进行反向传播时，梯度是逐步累积的。因此，如果在每次反向传播之前不清除先前的梯度，那么每一轮的梯度会与前几轮

的梯度相加，导致梯度值逐渐增大，从而影响模型的训练效果。因此，要进行清除。
inputs，targets 表示将输入数据和标签转移到指定的设备。
后面的 num_bd、num_cross 分别是本批次注入后门的样本数量和具有交叉扰动的样本数量；而 grid_temps 和 grid_temps2 是用于生成扰动的网格模板。

（3）

```
inputs_bd = F.grid_sample(inputs[:num_bd], grid_temps.repeat(num_bd, 1, 1, 1), align_corners=True)
if opt.attack_mode == "all2one":
    targets_bd = torch.ones_like(targets[:num_bd]) * opt.target_label
if opt.attack_mode == "all2all":
    targets_bd = torch.remainder(targets[:num_bd] + 1, opt.num_classes)

inputs_cross = F.grid_sample(inputs[num_bd : (num_bd + num_cross)], grid_temps2, align_corners=True)
```

inputs_bd 是经过扰动的后门数据；
targets_bd 根据攻击模式设置，可能是单一目标标签 (all2one 模式) 或者顺序标签 (all2all 模式)，而本次复现是 all2one 模式；
inputs_cross 是具有交叉扰动的输入数据。

（4）

```
total_inputs = torch.cat([inputs_bd, inputs_cross, inputs[(num_bd + num_cross) :]], dim=0)
total_inputs = transforms(total_inputs)
total_targets = torch.cat([targets_bd, targets[num_bd:]], dim=0)
start = time()
total_preds = netC(total_inputs)
total_time += time() - start
```

前向传播。将不同类型的输入数据（前面算出来的）组合在一起，并应用后处理转换，返回模型的输出 total_preds。

（5）

```
loss_ce = criterion_CE(total_preds, total_targets)

loss = loss_ce
loss.backward()

optimizerC.step()
```

反向传播。使用交叉熵损失函数计算损失，后反向传播这些误差损失，以此来更新模型参数。这是常规操作。

（6）

```
total_sample += bs
total_loss_ce += loss_ce.detach()

total_clean += bs - num_bd - num_cross
total_bd += num_bd
total_cross += num_cross
total_clean_correct += torch.sum(
    torch.argmax(total_preds[(num_bd + num_cross) :], dim=1) == total_targets[(num_bd + num_cross) :]
)
total_bd_correct += torch.sum(torch.argmax(total_preds[:num_bd], dim=1) == targets_bd)
if num_cross:
    total_cross_correct += torch.sum(
        torch.argmax(total_preds[num_bd : (num_bd + num_cross)], dim=1)
        == total_targets[num_bd : (num_bd + num_cross)]
    )
    avg_acc_cross = total_cross_correct * 100.0 / total_cross

avg_acc_clean = total_clean_correct * 100.0 / total_clean
avg_acc_bd = total_bd_correct * 100.0 / total_bd

avg_loss_ce = total_loss_ce / total_sample

if num_cross:
    progress_bar(
        batch_idx,
        len(train_dl),
        "CE Loss: {:.4f} | Clean Acc: {:.4f} | Bd Acc: {:.4f} | Cross Acc: {:.4f}".format(
            avg_loss_ce, avg_acc_clean, avg_acc_bd, avg_acc_cross
        ),
    )
else:
    progress_bar(
        batch_idx,
        len(train_dl),
        "CE Loss: {:.4f} | Clean Acc: {:.4f} | Bd Acc: {:.4f} ".format(avg_loss_ce, avg_acc_clean, avg_acc_bd),
    )
```

这些就是用来统计每一轮训练中的巡视和准确率，一开始定义的各种变量，在这里都要进行换算，并在后面的 if 判断语句里将结果计算、打印出来

（7）

```
# Save image for debugging
if not batch_idx % 50:
    if not os.path.exists(opt.temps):
        os.makedirs(opt.temps)
    path = os.path.join(opt.temps, "backdoor_image.png")
    torchvision.utils.save_image(inputs_bd, path, normalize=True)
```

每隔 50 个批次保存一次后门图像,并将将当前批次中的后门图像（inputs_bd）保存到指定路径。normalize=True 表示在保存图像之前，对其进行归一化。

（8）

```
    # Image for tensorboard
    if batch_idx == len(train_dl) - 2:
        residual = inputs_bd - inputs[:num_bd]
        batch_img = torch.cat([inputs[:num_bd], inputs_bd, total_inputs[:num_bd], residual], dim=2)
        batch_img = denormalizer(batch_img)
        batch_img = F.upsample(batch_img, scale_factor=(4, 4))
        grid = torchvision.utils.make_grid(batch_img, normalize=True)

# for tensorboard
if not epoch % 1:
    tf_writer.add_scalars(
        "Clean Accuracy", {"Clean": avg_acc_clean, "Bd": avg_acc_bd, "Cross": avg_acc_cross}, epoch
    )
    tf_writer.add_image("Images", grid, global_step=epoch)

schedulerC.step()
```

当处理到倒数第二个批次时，准备一些图像用于 TensorBoard 的可视化；
residual = inputs_bd - inputs[:num_bd]，这表示计算后门图像和原始图像之间的残差，用于增

强可视化攻击的效果；
batch_img = torch.cat([...], dim=2)：将原始图像、后门图像、变换后的图像和残差沿着宽度方向拼接，形成一个更大的图像；
denormalizer(batch_img)：对图像进行反归一化，使其在保存时具有更好的可视化效果；
F.upsample(batch_img, scale_factor=(4, 4))：将图像放大 4 倍，以便在 TensorBoard 中更清晰地查看细节。最后将图像集合绘制成网格，以便在 TensorBorder 中展示。
后面的一个 if 判断是用于将清洁样本准确率（Clean）、后门样本准确率（Bd）、以及交叉样本准确率（Cross）记录到 TensorBoard 中，以便追踪模型的性能。并且，把之前创建的图像网格（grid）添加到 TensorBoard 中，便于可视化训练过程中的图像变化。

elva()函数

至于在 train.py 训练过程中为什么会有评估函数呢？因为这个项目非常贴心地在每一次训练过程结束后，都进行一次评估，这个从训练的可视化界面和后面 main 函数的编写就可以看出来。
(1)

```
def eval(
    netC,
    optimizerC,
    schedulerC,
    test_dl,
    noise_grid,
    identity_grid,
    best_clean_acc,
    best_bd_acc,
    best_cross_acc,
    tf_writer,
    epoch,
    opt,
):
```

先来看看函数的形参，这里形参有部分就是从 trian（）函数里面过来的，其他的都是定义各种变量来表示模型的性能和评估过程需要用到的数值。

（2）

```
netC.eval()

total_sample = 0
total_clean_correct = 0
total_bd_correct = 0
total_cross_correct = 0
total_ae_loss = 0
```

netC.eval（）将模型 netC 设置为评估模式，关闭 dropout 和 batch normalization 等训练时才会启用的特性。这与训练用的是一样的。
初始化各种计数器和损失函数，用于记录测试过程中的样本数量和正确分类的样本数量。

（3）

```
for batch_idx, (inputs, targets) in enumerate(test_dl):
    with torch.no_grad():
        inputs, targets = inputs.to(opt.device), targets.to(opt.device)
        bs = inputs.shape[0]
        total_sample += bs

        # Evaluate Clean
        preds_clean = netC(inputs)
        total_clean_correct += torch.sum(torch.argmax(preds_clean, 1) == targets)
```

with torch.no_grad()用于关闭梯度计算，以节省内存并加快推理速度。

preds_clean = netC(inputs):计算干净数据（未被攻击的数据）的预测结果。

后面统计正确分类的干净样本数量。

（4）

```
# Evaluate Backdoor
grid_temps = (identity_grid + opt.s * noise_grid / opt.input_height) * opt.grid_rescale
grid_temps = torch.clamp(grid_temps, -1, 1)

ins = torch.rand(bs, opt.input_height, opt.input_height, 2).to(opt.device) * 2 - 1
grid_temps2 = grid_temps.repeat(bs, 1, 1, 1) + ins / opt.input_height
grid_temps2 = torch.clamp(grid_temps2, -1, 1)

inputs_bd = F.grid_sample(inputs, grid_temps.repeat(bs, 1, 1, 1), align_corners=True)
if opt.attack_mode == "all2one":
    targets_bd = torch.ones_like(targets) * opt.target_label
if opt.attack_mode == "all2all":
    targets_bd = torch.remainder(targets + 1, opt.num_classes)
preds_bd = netC(inputs_bd)
total_bd_correct += torch.sum(torch.argmax(preds_bd, 1) == targets_bd)

acc_clean = total_clean_correct * 100.0 / total_sample
acc_bd = total_bd_correct * 100.0 / total_sample
```

评估后门攻击样本。**注意：这里就是模型的核心之一，应用了变形场。**

grid_temps: 在身份网格 identity_grid 上添加放大的噪声网格 noise_grid，并乘以缩放系数 opt.s 和 opt.grid_rescale。这样生成了带有扭曲效果的变形场 grid_temps。

F.grid_sample：该函数利用生成的变形场 grid_temps 将其应用到原始图像 inputs 上，生成变形后的后门样本 inputs_bd。grid_sample 会根据给定的变形场对输入图像进行空间变换。

最后统计模型在后门样本上的正确分类数量，并计算干净样本和后门样本的准确率。

（5）

```
# Evaluate cross
if opt.cross_ratio:
    inputs_cross = F.grid_sample(inputs, grid_temps2, align_corners=True)
    preds_cross = netC(inputs_cross)
    total_cross_correct += torch.sum(torch.argmax(preds_cross, 1) == targets)

    acc_cross = total_cross_correct * 100.0 / total_sample

    info_string = (
        "Clean Acc: {:.4f} - Best: {:.4f} | Bd Acc: {:.4f} - Best: {:.4f} | Cross: {:.4f}".format(
            acc_clean, best_clean_acc, acc_bd, best_bd_acc, acc_cross, best_cross_acc
        )
    )
else:
    info_string = "Clean Acc: {:.4f} - Best: {:.4f} | Bd Acc: {:.4f} - Best: {:.4f}".format(
        acc_clean, best_clean_acc, acc_bd, best_bd_acc
    )
progress_bar(batch_idx, len(test_dl), info_string)
```

如果启用了交叉攻击（opt.cross_ratio），则生成交叉样本 inputs_cross 并评估模型在这些样本上的性能。后面根据不同样本类型的准确率，构建一个信息字符串并使用 progress_bar 显示当前批次的评估结果。

（6）

```
# tensorboard
if not epoch % 1:
    tf_writer.add_scalars("Test Accuracy", {"Clean": acc_clean, "Bd": acc_bd}, epoch)
```

每个 epoch 结束时，将当前测试阶段的干净样本和后门样本的准确率记录到 TensorBoard，以便后续分析。这和 train（）里面也是相对应的。

（7）

```
# Save checkpoint
if acc_clean > best_clean_acc or (acc_clean > best_clean_acc - 0.1 and acc_bd > best_bd_acc):
    print(" Saving...")
    best_clean_acc = acc_clean
    best_bd_acc = acc_bd
    if opt.cross_ratio:
        best_cross_acc = acc_cross
    else:
        best_cross_acc = torch.tensor([0])
    state_dict = {
        "netC": netC.state_dict(),
        "schedulerC": schedulerC.state_dict(),
        "optimizerC": optimizerC.state_dict(),
        "best_clean_acc": best_clean_acc,
        "best_bd_acc": best_bd_acc,
        "best_cross_acc": best_cross_acc,
        "epoch_current": epoch,
        "identity_grid": identity_grid,
        "noise_grid": noise_grid,
    }
    torch.save(state_dict, opt.ckpt_path)
    with open(os.path.join(opt.ckpt_folder, "results.txt"), "w+") as f:
        results_dict = {
            "clean_acc": best_clean_acc.item(),
            "bd_acc": best_bd_acc.item(),
            "cross_acc": best_cross_acc.item(),
        }
        json.dump(results_dict, f, indent=2)

return best_clean_acc, best_bd_acc, best_cross_acc
```

保存最佳模型。从 if 判断的条件就可以看出来，最后保存的最佳模型下的 checkpoints，即 如果当前模型的干净样本准确率（acc_clean）超过了历史最佳（best_clean_acc），或干净样本

准确率接近最佳且后门样本准确率更高，则保存当前模型，并记录模型的状态、调度器、优化器、最佳准确率等信息。

main()函数

（1）

```
opt = config.get_arguments().parse_args()
```

从 config.py 里面把各种配置参数读取过来。

（2）

```
if opt.dataset in ["mnist", "cifar10"]:
    opt.num_classes = 10
elif opt.dataset == "gtsrb":
    opt.num_classes = 43
elif opt.dataset == "celeba":
    opt.num_classes = 8
else:
    raise Exception("Invalid Dataset")

if opt.dataset == "cifar10":
    opt.input_height = 32
    opt.input_width = 32
    opt.input_channel = 3
elif opt.dataset == "gtsrb":
    opt.input_height = 32
    opt.input_width = 32
    opt.input_channel = 3
elif opt.dataset == "mnist":
    opt.input_height = 28
    opt.input_width = 28
    opt.input_channel = 1
elif opt.dataset == "celeba":
    opt.input_height = 64
    opt.input_width = 64
    opt.input_channel = 3
else:
    raise Exception("Invalid Dataset")
```

根据不同的数据集，设置类别（class）数量，以及输入的长、宽和通道。因为每个数据集的内容都是不同的。

（3）

```
# Dataset
train_dl = get_dataloader(opt, True)
test_dl = get_dataloader(opt, False)
```

准备数据集

（4）

```
# prepare model
netC, optimizerC, schedulerC = get_model(opt)
```

准备模型，即进行模型的初始化。

（5）

```python
# Load pretrained model
mode = opt.attack_mode
opt.ckpt_folder = os.path.join(opt.checkpoints, opt.dataset)
opt.ckpt_path = os.path.join(opt.ckpt_folder, "{}_{}_morph.pth.tar".format(opt.dataset, mode))
opt.log_dir = os.path.join(opt.ckpt_folder, "log_dir")
if not os.path.exists(opt.log_dir):
    os.makedirs(opt.log_dir)

if opt.continue_training:
    if os.path.exists(opt.ckpt_path):
        print("Continue training!!")
        state_dict = torch.load(opt.ckpt_path)
        netC.load_state_dict(state_dict["netC"])
        optimizerC.load_state_dict(state_dict["optimizerC"])
        schedulerC.load_state_dict(state_dict["schedulerC"])
        best_clean_acc = state_dict["best_clean_acc"]
        best_bd_acc = state_dict["best_bd_acc"]
        best_cross_acc = state_dict["best_cross_acc"]
        epoch_current = state_dict["epoch_current"]
        identity_grid = state_dict["identity_grid"]
        noise_grid = state_dict["noise_grid"]
        tf_writer = SummaryWriter(log_dir=opt.log_dir)
    else:
        print("Pretrained model doesnt exist")
        exit()
else:
    print("Train from scratch!!!")
    best_clean_acc = 0.0
    best_bd_acc = 0.0
    best_cross_acc = 0.0
    epoch_current = 0

    # Prepare grid
    ins = torch.rand(1, 2, opt.k, opt.k) * 2 - 1
    ins = ins / torch.mean(torch.abs(ins))
    noise_grid = (
        F.upsample(ins, size=opt.input_height, mode="bicubic", align_corners=True)
        .permute(0, 2, 3, 1)
        .to(opt.device)
    )
    array1d = torch.linspace(-1, 1, steps=opt.input_height)
    x, y = torch.meshgrid(array1d, array1d)
    identity_grid = torch.stack((y, x), 2)[None, ...].to(opt.device)

    shutil.rmtree(opt.ckpt_folder, ignore_errors=True)
    os.makedirs(opt.log_dir)
    with open(os.path.join(opt.ckpt_folder, "opt.json"), "w+") as f:
        json.dump(opt.__dict__, f, indent=2)
    tf_writer = SummaryWriter(log_dir=opt.log_dir)
```

加载预训练模型，根据模型相关文件夹路径前缀找到对应的位置，后开始获取模型的训练配置信息，有继续训练、从头开始训练两种。

继续训练 ： 如果选择继续训练且检查点文件存在，则加载模型的状态（包括网络参数、优化器状态和调度器状态），以及记录的最佳准确率和当前训练的 epoch。

从头开始训练（后面那个 from scratch）： 如果选择从头开始训练，则初始化最佳准确率、当前 epoch 以及用于后门攻击的网格（identity_grid 和 noise_grid），并删除旧的检查点文件夹。

注意：Wanet 的核心就是引入了变形场！！！。这里的 grid 就是生成变形场的过程。

ins: 生成一个随机的噪声网格 `ins`，形状为 `1 x 2 x k x k`，其中 `k` 是预设的网格大小。噪声的取值范围为 `[-1, 1]`。
noise_grid: 将噪声网格 `ins` 通过双三次插值 (bicubic interpolation) 放大到与输入图像大小一致。此操作产生了一个扭曲的网格，这个网格将在图像变形中起到关键作用。
identity_grid: 生成了一个“身份网格”，它是一个标准的从 `[-1, 1]` 均匀映射到每个像素位置的网格。这是没有任何变形的标准网格。

（6）

```
for epoch in range(epoch_current, opt.n_iters):
    print("Epoch {}:".format(epoch + 1))
    train(netC, optimizerC, schedulerC, train_dl, noise_grid, identity_grid, tf_writer, epoch, opt)
    best_clean_acc, best_bd_acc, best_cross_acc = eval(
        netC,
        optimizerC,
        schedulerC,
        test_dl,
        noise_grid,
        identity_grid,
        best_clean_acc,
        best_bd_acc,
        best_cross_acc,
        tf_writer,
        epoch,
        opt,
    )
```

这是评估过程的 main 代码，因为本项目是练一轮就评估一轮。把 trian（）函数更新出来的各种配置、参数都传输到 eval（）函数里面，进行评估，然后输出打印结果。

4. eval.py

因为 trian 时就进行了 eval，所以该代码文件里的函数基本被涵盖在了 trian.py 里面，这里就不再重复介绍了。只是少数写法不同，但基本上的代码思想和用到的东西都是完全一样

复现过程：

1.安装环境

```
$ python -m pip install -r requirements.py
```

2. 下载数据集

该仓库给出了 GTSRB 数据集，并建议针对该数据集进行实验，所以也就下载了该数据集。

```
$ bash gtsrb_download.sh
```

3. 训练

运行下述命令：

```
$ python train.py --dataset gtsrb --attack_mode all2one --data_root ~/jxd/Warping-based_Backdoor_Attack-release-main/home/ubuntu/GTSRB/Test/GT-final_test.csv
```

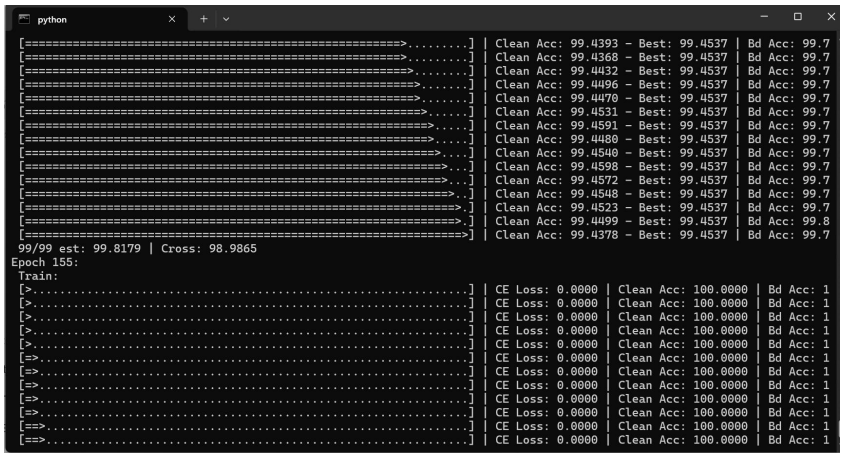
--dataset 表示使用的是 GTSRB 数据集；

--attack_mode 表示执行的任务是 all2one 任务，即将利用后门攻击将所有类别的图像识别成一个预设的特定类别。

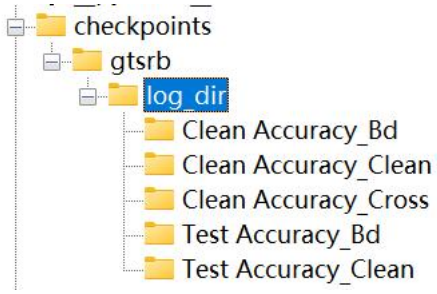
注意，后面 data_root 指向的是 GTSRB 对应的数据集使用文件的地址，是绝对地址。一定不能搞错了。

默认训练的轮数是 1000 轮！为了契合论文中的效果，我并没有修改轮数。

训练过程实例：

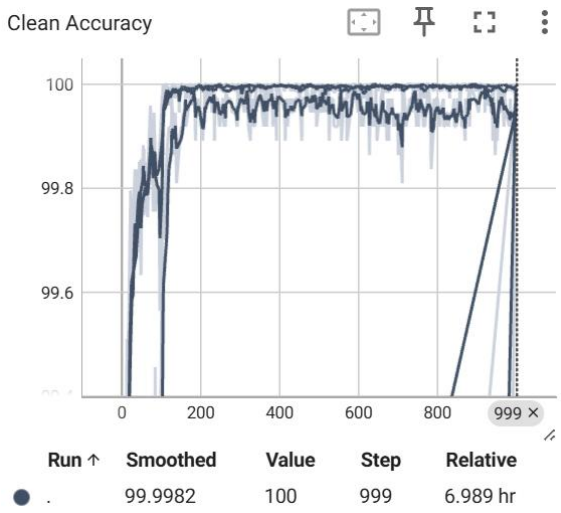


训练结束后，检查点（checkpoints）会存放到对应文件地址。如下：

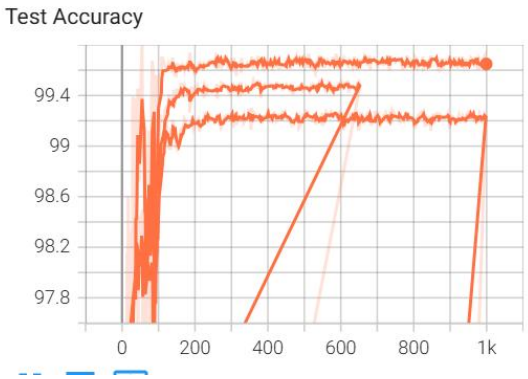


一千轮的训练过程中得到的相关数据，可视化如下：

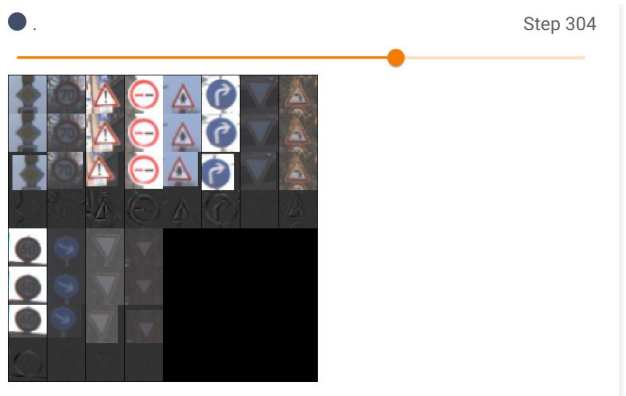
干净样本中的准确率，三条曲线分别代表 clean、BD、cross



评估测试时的准确率，三条曲线分别代表 clean、BD、cross。（因为训练的时候，评估测试是一起进行的）



训练过程中对应的后门攻击图像（举例）



4. 评估（无防御机制）

运行下述命令

```
# root @ d7d149c7484f in ~/jxd/Warping-based_Backdoor_Attack-release-main [0:59:55]
$ python eval.py --dataset gtsrb --attack_mode all2one --data_root ~/jxd/Warping-based_Backdoor_Attack-release-main/home
/ubuntu/temps/ --checkpoints /root/jxd/Warping-based_Backdoor_Attack-release-main/checkpoints/
2024-08-31 01:01:11.306712: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F FMA, in other operations, rebuild TensorFlow with the appropriate com
piler flags.
/root/jxd/Warping-based_Backdoor_Attack-release-main/checkpoints/gtsrb/gtsrb_all2one_morph.pth.tar
 Eval:
 [>.....................................................................] | Clean Acc: 100.0000 | Bd Acc: 99.2188 | Cross: 99
```

该命令是运行过程命令差不多，只是换了文件。

评估效果：

```
[============================================================>] | Clean Acc: 99.2874 | Bd Acc: 99.7229 | Cross: 99.
99/99
```

发现，没有一轮数值是和论文中完全一致的，大差不差。论文中数值如下；

GTSRB	98.87	99.33	98.01

5. 防御实验

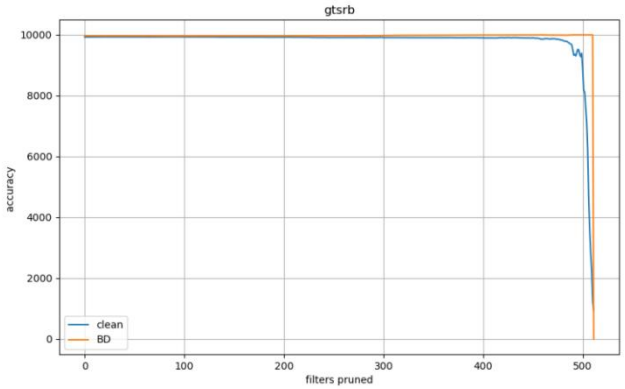
1. fine-pruning 防御

运行下述命令

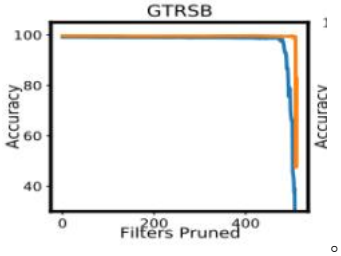
```
# root @ d7d149c7484f in ~/jxd/Warping-based_Backdoor_Attack-release-main/defenses/fine_pruning [1:03:43] C:1
$ python fine-pruning-cifar10-gtsrb.py --dataset gtsrb --attack_mode all2one --data_root ~/jxd/Warping-based_Backdoor_At
tack-release-main/home/ubuntu/temps/
```

本实验测试在剪枝网络的滤波器（神经元）后，对识别干净样本和后门攻击样本产生的影响，并判断该防御方法对 WaNet 的防御效果。

结果如下：



论文结果：



发现，两个曲线图保持高度一致。表明在对 fine-pruning 防御机制进行攻击时，WaNet 攻击确实不会受太大影响。具体解释看文献阅读讲解。

2. Neural Cleanse 防御

运行如下命令。

```
# root @ d7d149c7484f in ~/jxd/Warping-based_Backdoor_Attack-release-main/defenses/neural_cleanse [2:18:16]
$ python neural_cleanse.py --dataset gtsrb --attack_mode all2one --data_root ~/jxd/Warping-based_Backdoor_Attack-release
-main/home/ubuntu/temps/
```

本实验是用攻击样本对设置有 Neural Cleanse 防御机制的模型进行攻击

攻击样本

触发器 trigger

掩码 mask（限制扰动的应用区域）

模式 pattern（输入数据中存在的特定结构或样式）

攻击结果

```
Determining whether model is backdoor
Median: 48.3321533203125, MAD: 7.673092842102051
Anomaly index: 3.65445017814636 23
This is a backdoor model
```

结果显示是检测出了后门攻击！而根据论文里面的判断方法，即 Anomaly index 的值是否大于阈值 2。若大于 2，则会被认为是后门攻击。（代码里面也是这样写的，设置了阈值 2）

```
if min_mad < 2:
    print("Not a backdoor model")
else:
    print("This is a backdoor model")
```

3.STRIP 防御

运行如下命令。

```
# root @ d7d149c7484f in ~/jxd/Warping-based_Backdoor_Attack-release-main/defenses/STRIP [5:33:16]
$ python STRIP.py --dataset gtsrb --attack_mode all2one --data_root ~/jxd/Warping-based_Backdoor_Attack-release-main/hom
e/ubuntu/temps/
```

结果

在干净样本数据下：

```
Testing with clean data !!!!
 [========================== 100/100 ==========================>]
Testing with clean data !!!!
 [========================== 100/100 ==========================>]
Testing with clean data !!!!
 [========================== 100/100 ==========================>]
Testing with clean data !!!!
 [========================== 100/100 ==========================>]
Testing with clean data !!!!
 [========================== 100/100 ==========================>]
Testing with clean data !!!!
 [========================== 100/100 ==========================>]
Testing with clean data !!!!
 [========================== 100/100 ==========================>]
Testing with clean data !!!!
 [========================== 100/100 ==========================>]
Testing with clean data !!!!
 [========================== 100/100 ==========================>]
Min entropy trojan: 7.25457275390625, Detection boundary: 0.2
Not a backdoor model
```

在攻击样本数据下（要修改源码才可以进行）

```
Testing with bd data !!!!
 [========================== 100/100 ==========================>]
Testing with bd data !!!!
 [========================== 100/100 ==========================>]
Testing with bd data !!!!
 [========================== 100/100 ==========================>]
Testing with bd data !!!!
 [========================== 100/100 ==========================>]
Testing with bd data !!!!
 [========================== 100/100 ==========================>]
Testing with bd data !!!!
 [========================== 100/100 ==========================>]
Testing with bd data !!!!
 [========================== 100/100 ==========================>]
Testing with bd data !!!!
 [========================== 100/100 ==========================>]
Testing with bd data !!!!
 [========================== 100/100 ==========================>]
Min entropy trojan: 7.280257568359375, Detection boundary: 0.2
Not a backdoor model
```

在干净样本和后门攻击样本的测试中，最小熵木马的值在 7.2 左右，都显著大于边界值 0.2。这表明 STRIP 防御机制无法检测 WaNet 的攻击。

复现难点

```
# root @ d7d149c7484f in ~/jxd/Warping-based_Backdoor_Attack-release-main [6:38:23]
$ python eval.py --dataset gtsrb --attack_mode all2one
2024-08-30 06:38:38.711396: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F FMA, in other operations, rebuild TensorFlow with the appropriate com
piler flags.
Traceback (most recent call last):
  File "/root/jxd/Warping-based_Backdoor_Attack-release-main/eval.py", line 170, in <module>
    main()
  File "/root/jxd/Warping-based_Backdoor_Attack-release-main/eval.py", line 138, in main
    test_dl = get_dataloader(opt, False)
  File "/root/jxd/Warping-based_Backdoor_Attack-release-main/utils/dataloader.py", line 145, in get_dataloader
    dataset = GTSRB(opt, train, transform)
  File "/root/jxd/Warping-based_Backdoor_Attack-release-main/utils/dataloader.py", line 82, in __init__
    self.images, self.labels = self._get_data_test_list()
  File "/root/jxd/Warping-based_Backdoor_Attack-release-main/utils/dataloader.py", line 104, in _get_data_test_list
    gtFile = open(prefix)
FileNotFoundError: [Errno 2] No such file or directory: '/home/ubuntu/temps/GTSRB/Test/GT-final_test.csv'
(faceswap)
```

```python
def get_arguments():
    parser = argparse.ArgumentParser()

    parser.add_argument("--data_root", type=str, default="/home/ubuntu/temps/")
    parser.add_argument("--checkpoints", type=str, default="./checkpoints")
    parser.add_argument("--temps", type=str, default="./temps")
    parser.add_argument("--device", type=str, default="cuda")
    parser.add_argument("--continue_training", action="store_true")
```

```
# root @ d7d149c7484f in ~/jxd/Warping-based_Backdoor_Attack-release-main [6:51:07] C:1
$ python eval.py --dataset gtsrb --attack_mode all2one --data_root ~/jxd//Warping-based_Backdoor_Attack-release-main/hom
e/ubuntu/temps/
```

太坑了，必须要用绝对路径。

但是又发现预训练模型不存在，在训练后，进行评估时发现的问题。

```
$ python eval.py --dataset gtsrb --attack_mode all2one --data_root ~/jxd//Warping-based_Backdoor_Attack-release-main/hom
e/ubuntu/temps/
2024-08-30 06:53:13.933816: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F FMA, in other operations, rebuild TensorFlow with the appropriate com
piler flags.
Pretrained model doesnt exist
```

我在代码中加入一行 print，打印期望的文件路径，如下，然后根据该路径修改服务器里的文件对应路径。

```
# root @ d7d149c7484f in ~/jxd/Warping-based_Backdoor_Attack-release-main [7:16:34] C:1
$ python eval.py --dataset gtsrb --attack_mode all2one --data_root ~/jxd/Warping-based_Backdoor_Attack-release-main/ho
/ubuntu/temps/ --checkpoints  /root/jxd/Warping-based_Backdoor_Attack-release-main/checkpoints/
2024-08-30 07:17:34.759084: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized t
use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F FMA, in other operations, rebuild TensorFlow with the appropriate c
piler flags.
/root/jxd/Warping-based_Backdoor_Attack-release-main/checkpoints/gtsrb/gtsrb_all2one_morph.pth.tar
Pretrained model doesnt exist
```

代码写得有问题

```
# Load pretrained model
mode = opt.attack_mode
opt.ckpt_folder = os.path.join(opt.checkpoints, opt.dataset)
opt.ckpt_path = os.path.join(opt.ckpt_folder, "{}_{}_morph.pth.tar".format(opt.dataset, mode))
opt.log_dir = os.path.join(opt.ckpt_folder, "log_dir")

state_dict = torch.load(opt.ckpt_path)
netC.load_state_dict(state_dict["netC"])
if mode != "clean":
    identity_grid = state_dict["identity_grid"]
    noise_grid = state_dict["noise_grid"]
netC.requires_grad_(False)
netC.eval()
netC.to(opt.device)

# Prepare test set
testset = get_dataset(opt, train=False)
opt.bs = opt.n_test
test_dataloader = get_dataloader(opt, train=False)
denormalizer = Denormalizer(opt)

# STRIP detector
strip_detector = STRIP(opt)

# Entropy list
list_entropy_trojan = []
list_entropy_benign = []

if mode == "attack":
    # Testing with perturbed data
    print("Testing with bd data !!!!")
```