About this assignment

In this assignment, you will implement two adversarial attacks against ResNet18 (FGSM and PGD), as well as two defenses against adversarial attacks (adversarial training and SAP). There are three goals for this assignment:

1. Learning about and evaluate base adversarial attacks and defenses in a simple setting.
2. Learning to use Pytorch's Lightning framework to simplify and modularize your code.
3. Learning to use Pytorch to adjust/manipulate the *architecture* of a pretrained model.

Imports

If you're running this notebook in Colab, you'll want to uncomment and run the following line.

If you're running this notebook locally or on a Grace cluster, you can separately install any packages you use.

Note: for this assignment, if your local machine is not GPU-compatible, you will probably want to use Colab or a Grace cluster.

In [ ]:
```
!pip install lightning
```

```
Collecting lightning
  Downloading lightning-2.2.1-py3-none-any.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 28.6 MB/s eta 0:00
:00
Requirement already satisfied: PyYAML<8.0,>=5.4 in /usr/local/lib/python3.10
/dist-packages (from lightning) (6.0.1)
Requirement already satisfied: fsspec[http]<2025.0,>=2022.5.0 in /usr/local/
lib/python3.10/dist-packages (from lightning) (2023.6.0)
Collecting lightning-utilities<2.0,>=0.8.0 (from lightning)
  Downloading lightning_utilities-0.11.2-py3-none-any.whl (26 kB)
Requirement already satisfied: numpy<3.0,>=1.17.2 in /usr/local/lib/python3.
10/dist-packages (from lightning) (1.25.2)
Requirement already satisfied: packaging<25.0,>=20.0 in /usr/local/lib/pytho
n3.10/dist-packages (from lightning) (24.0)
Requirement already satisfied: torch<4.0,>=1.13.0 in /usr/local/lib/python3.
10/dist-packages (from lightning) (2.2.1+cu121)
Collecting torchmetrics<3.0,>=0.7.0 (from lightning)
```

```
  Downloading torchmetrics-1.3.2-py3-none-any.whl (841 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 841.5/841.5 kB 54.0 MB/s eta
0:00:00
Requirement already satisfied: tqdm<6.0,>=4.57.0 in /usr/local/lib/python3.1
0/dist-packages (from lightning) (4.66.2)
Requirement already satisfied: typing-extensions<6.0,>=4.4.0 in /usr/local/l
ib/python3.10/dist-packages (from lightning) (4.10.0)
Collecting pytorch-lightning (from lightning)
  Downloading pytorch_lightning-2.2.1-py3-none-any.whl (801 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 801.6/801.6 kB 48.7 MB/s eta
0:00:00
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pa
ckages (from fsspec[http]<2025.0,>=2022.5.0->lightning) (2.31.0)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/
python3.10/dist-packages (from fsspec[http]<2025.0,>=2022.5.0->lightning) (3
.9.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from lightning-utilities<2.0,>=0.8.0->lightning) (67.7.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pa
ckages (from torch<4.0,>=1.13.0->lightning) (3.13.3)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packa
ges (from torch<4.0,>=1.13.0->lightning) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-pa
ckages (from torch<4.0,>=1.13.0->lightning) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-pack
ages (from torch<4.0,>=1.13.0->lightning) (3.1.3)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch<4.0,>=1.13.0->lightn
ing)
  Downloading nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(23.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.7/23.7 MB 69.9 MB/s eta 0:
00:00
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch<4.0,>=1.13.0->ligh
tning)
  Downloading nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.w
hl (823 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 823.6/823.6 kB 59.9 MB/s eta
0:00:00
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch<4.0,>=1.13.0->lightn
ing)
  Downloading nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(14.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.1/14.1 MB 91.0 MB/s eta 0:
00:00
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch<4.0,>=1.13.0->lightning)
  Downloading nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731
.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 731.7/731.7 MB 1.7 MB/s eta 0
:00:00
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch<4.0,>=1.13.0->lightning)
  Downloading nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (41
0.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 410.6/410.6 MB 2.9 MB/s eta 0
```

```
:00:00
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch<4.0,>=1.13.0->lightning)
  Downloading nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (12
1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 121.6/121.6 MB 14.0 MB/s eta
0:00:00
Collecting nvidia-curand-cu12==10.3.2.106 (from torch<4.0,>=1.13.0->lightnin
g)
  Downloading nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (
56.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.5/56.5 MB 21.5 MB/s eta 0:
00:00
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch<4.0,>=1.13.0->lightn
ing)
  Downloading nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl
(124.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 124.2/124.2 MB 8.2 MB/s eta 0
:00:00
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch<4.0,>=1.13.0->lightn
ing)
  Downloading nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl
(196.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 196.0/196.0 MB 5.6 MB/s eta 0
:00:00
Collecting nvidia-nccl-cu12==2.19.3 (from torch<4.0,>=1.13.0->lightning)
  Downloading nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.whl (166.0
MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 166.0/166.0 MB 10.3 MB/s eta
0:00:00
Collecting nvidia-nvtx-cu12==12.1.105 (from torch<4.0,>=1.13.0->lightning)
  Downloading nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 k
B)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.1/99.1 kB 15.9 MB/s eta 0:
00:00
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/di
st-packages (from torch<4.0,>=1.13.0->lightning) (2.2.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->tor
ch<4.0,>=1.13.0->lightning)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.w
hl (21.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 82.8 MB/s eta 0:
00:00
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10
/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<2025.0,>=2022.
5.0->lightning) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/di
st-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<2025.0,>=2022.5.0
->lightning) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.1
0/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<2025.0,>=2022
.5.0->lightning) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3
.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<2025.0,>=20
```

```
22.5.0->lightning) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/d
ist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<2025.0,>=2022.5.
0->lightning) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]<2025.0,
>=2022.5.0->lightning) (4.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/
dist-packages (from jinja2->torch<4.0,>=1.13.0->lightning) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/py
thon3.10/dist-packages (from requests->fsspec[http]<2025.0,>=2022.5.0->light
ning) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dis
t-packages (from requests->fsspec[http]<2025.0,>=2022.5.0->lightning) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.
10/dist-packages (from requests->fsspec[http]<2025.0,>=2022.5.0->lightning)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
10/dist-packages (from requests->fsspec[http]<2025.0,>=2022.5.0->lightning)
(2024.2.2)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dis
t-packages (from sympy->torch<4.0,>=1.13.0->lightning) (1.3.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvid
ia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu1
2, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, light
ning-utilities, nvidia-cusparse-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu1
2, torchmetrics, pytorch-lightning, lightning
Successfully installed lightning-2.2.1 lightning-utilities-0.11.2 nvidia-cub
las-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.
1.105 nvidia-cuda-runtime-cu12-12.1.105 nvidia-cudnn-cu12-8.9.2.26 nvidia-cu
fft-cu12-11.0.2.54 nvidia-curand-cu12-10.3.2.106 nvidia-cusolver-cu12-11.4.5
.107 nvidia-cusparse-cu12-12.1.0.106 nvidia-nccl-cu12-2.19.3 nvidia-nvjitlin
k-cu12-12.4.127 nvidia-nvtx-cu12-12.1.105 pytorch-lightning-2.2.1 torchmetri
cs-1.3.2
```

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

%matplotlib inline

import os
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import lightning as L
from torchmetrics import Accuracy
from lightning.pytorch.callbacks import ModelCheckpoint, LearningRateMonitor
```

Config

Just run the next code block, but double check the one after that.

In [ ]:
```
device = torch.device("cuda:0") if torch.cuda.is_available() else torch.devi
print(device)


CLASSES = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
NUM_CLASSES = len(CLASSES)
```

```
cuda:0
```

In [ ]:
```
# If you run into memory issues, you can reduce the batch size
BATCH_SIZE = 128

# Change these to the relative paths you'd like to use
# for the CIFAR-10 data and model checkpoints
DATA_PATH = 'data/'
CHECKPOINT_PATH = 'models/checkpoints/'

# The different models we'll be fine-tuning
SAVE_NAMES = [
    'baseline',
    'adv_train',    # Adversarial training a la Madry et al.
    'SAP_conv', # Full SAP post-convolution a la Dhillon et al.
]
SAVE_NAMES = {
    name: os.path.join(CHECKPOINT_PATH, name) for name in SAVE_NAMES
}
```

Results dictionary

We set up for storing experiment results here. Just run the following block.

In [ ]:
```
models = {name: None for name in SAVE_NAMES.keys()}
attacks = {
    'id': None,
    'fgsm': None,
    'pgd': None,
}

results_dic = {
    'model': [],
    'attack': [],
    'top_k': [],
    'accuracy': [],
}
results_trainer = L.Trainer(accelerator='auto', devices=1)
```

```
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TP
U cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False, using: 0 IP
Us
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HP
Us
```

Data processing

You can just run these three blocks of code. They import the CIFAR10 data from Torchvision and split them into train/validation/test sets.

We also takes a sample for later visualization purposes.

In [ ]:
```
# Pretrained normalization based on https://discuss.pytorch.org/t/how-to-pre
means, stds = [0.49139968, 0.48215827, 0.44653124], [0.24703233, 0.24348505,
means, stds = np.array(means), np.array(stds)
```

In [ ]:
```
import torchvision.transforms.v2 as transforms

def get_cifar_loaders(batch_size):
    # Transformations applied to images before passing them to the model
    transform = transforms.Compose(
        [
            # transforms.Resize(256),
            # transforms.CenterCrop(224),
            transforms.ToImage(), # Converts to tensor
            transforms.ToDtype(torch.float32, scale=True),
            transforms.Normalize(mean=means, std=stds)
        ])

    trainset = torchvision.datasets.CIFAR10(root=DATA_PATH, train=True,
                                            download=True, transform=transfo
    # The train set is of size 50000
    trainset, valset = torch.utils.data.random_split(trainset, [40000, 10000
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_siz
                                            shuffle=True, num_workers=2)
    valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size,
                                            shuffle=False, num_workers=2)

    testset = torchvision.datasets.CIFAR10(root=DATA_PATH, train=False,
                                        download=True, transform=transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                            shuffle=False, num_workers=2)

    return trainloader, valloader, testloader
```

In [ ]:
```
trainloader, valloader, testloader = get_cifar_loaders(BATCH_SIZE)
sample_images, sample_labels = next(iter(trainloader))
sample_images, sample_labels = sample_images.to(device), sample_labels.to(de
```

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

Base Resnet Class

Here we've implemented a ResNet18 model in the Pytorch Lightning framework. Here is Lightning's documentation.

The main code to look at are `__init__` and `training_step`. If you'd like to use Lightning on your own project, the other methods may be useful reference, but as always we defer to the documentation.

In [ ]:
```
class LResnet(L.LightningModule):
    def __init__(self, adv_train_method = None): #EDITED
        super().__init__()
```

```
        # Set loss module
        self.loss_module = nn.CrossEntropyLoss()
        # Example input for visualizing the graph in Tensorboard
        # CIFAR-10 images are 32x32
        self.example_input_array = torch.zeros((1, 3, 32, 32), dtype=torch.f
        self.num_target_classes = 10
        # Accuracy metric for training logs and testing evaluation
        self.accuracy = Accuracy(task="multiclass", num_classes=self.num_tar
        # Adversarial generation method for training
        self.adv_train_method = adv_train_method # EDITED

        # Load pretrained model weights
        self.model = torchvision.models.resnet18(
            weights=torchvision.models.ResNet18_Weights.IMAGENET1K_V1
        )
        # Change final layer from 1000 (ImageNet) classes to 10 (CIFAR-10) c
        self.model.fc = nn.Linear(self.model.fc.in_features, self.num_target

    def forward(self, imgs):
        return self.model(imgs)

    def configure_optimizers(self):
        optimizer = optim.AdamW(self.parameters(), lr=1e-5, weight_decay=0.1
        return [optimizer] # Lightning has enables multi-optimizer training,

    def training_step(self, batch, batch_idx):
        imgs, labels = batch
        if self.adv_train_method is not None:
            opt = self.optimizers()
            opt.zero_grad()
            # Change the images to adversarial examples
            imgs = self.adv_train_method(self.model, imgs, labels)
            # adv_train_method sets the model to eval
            self.model.train()
            # Reset accumulated gradients from adversarial generation
            opt.zero_grad()
        # Once we have the correct training images,
        # we can use the usual Lightning forward pass
        outputs = self.model(imgs)
        loss = self.loss_module(outputs, labels)
        acc = self.accuracy(outputs, labels)
        # Log accuracy and loss per-batch for Tensorboard
        self.log('train_acc', acc, on_step=False, on_epoch=True)
        self.log('train_loss', loss, prog_bar=True)
        return loss

    def validation_step(self, batch, batch_idx):
        imgs, labels = batch
        outputs = self.model(imgs)
        loss = self.loss_module(outputs, labels)
        self.log('val_loss', loss)
        # No need to return to call backward() on the loss
```

```
    def test_step(self, batch, batch_idx):
        imgs, labels = batch
        outputs = self.model(imgs)
        acc = self.accuracy(outputs, labels)
        self.log("test_acc", acc, prog_bar=True)
        # No need to return to call backward() on the loss
```

Example training code

Run the following code block. It is an example of how to code a training loop with Lightning. If you change hyperparameters for your experiments later, you will need to comment at the end on the changes you've made.

In [ ]:
```
save_key = 'baseline'
baseline_model = LResnet()
baseline_trainer = L.Trainer(
    default_root_dir = SAVE_NAMES[save_key], # Where to save the model
    accelerator='auto',
    devices=1,
    max_epochs=30,
    callbacks=[
        ModelCheckpoint( # Save the best model by validation loss
            dirpath=SAVE_NAMES[save_key],
            monitor='val_loss',
            save_top_k=1,
            mode='min',
            save_weights_only=True,
            every_n_epochs=1,
        ),
        EarlyStopping( # Stop training early if val_loss doesn't improve
            monitor='val_loss',
            patience=3,
            verbose=True,
            mode='min',
        ),
        LearningRateMonitor('epoch') # Log learning rate each epoch
    ],
)

# These two lines are optional, but they make the Tensorboard logs look nice
baseline_trainer.logger._log_graph = True  # If True, we plot the computatic
baseline_trainer.logger._default_hp_metric = None  # Optional logging argume

# This is all you need to train the model
baseline_trainer.fit(baseline_model, trainloader, valloader)
# Load best checkpoint after training
baseline_model = LResnet.load_from_checkpoint(
    baseline_trainer.checkpoint_callback.best_model_path
).to(device)

# Store the model in the dictionary
models[save_key] = baseline_model
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to
/root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%|██████████| 44.7M/44.7M [00:00<00:00, 175MB/s]
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TP
U cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False, using: 0 IP
Us
```

```
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HP
Us
INFO: You are using a CUDA device ('NVIDIA A100-SXM4-40GB') that has Tensor
Cores. To properly utilize them, you should set `torch.set_float32_matmul_pr
ecision('medium' | 'high')` which will trade-off precision for performance.
For more details, read https://pytorch.org/docs/stable/generated/torch.set_f
loat32_matmul_precision.html#torch.set_float32_matmul_precision
INFO:lightning.pytorch.utilities.rank_zero:You are using a CUDA device ('NVI
DIA A100-SXM4-40GB') that has Tensor Cores. To properly utilize them, you sh
ould set `torch.set_float32_matmul_precision('medium' | 'high')` which will
trade-off precision for performance. For more details, read https://pytorch.
org/docs/stable/generated/torch.set_float32_matmul_precision.html#torch.set_
float32_matmul_precision
WARNING: Missing logger folder: models/checkpoints/baseline/lightning_logs
WARNING:lightning.pytorch.loggers.tensorboard:Missing logger folder: models/
checkpoints/baseline/lightning_logs
/usr/local/lib/python3.10/dist-packages/lightning/pytorch/callbacks/model_ch
eckpoint.py:653: Checkpoint directory /content/models/checkpoints/baseline e
xists and is not empty.
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICE
S: [0]
INFO:
  | Name        | Type               | Params | In sizes       | Out sizes
----------------------------------------------------------------------------
----
0 | loss_module | CrossEntropyLoss   | 0      | ?              | ?
1 | accuracy    | MulticlassAccuracy | 0      | ?              | ?
2 | model       | ResNet             | 11.2 M | [1, 3, 32, 32] | [1, 10]
----------------------------------------------------------------------------
----
11.2 M    Trainable params
0         Non-trainable params
11.2 M    Total params
44.727    Total estimated model params size (MB)
INFO:lightning.pytorch.callbacks.model_summary:
  | Name        | Type               | Params | In sizes       | Out sizes
----------------------------------------------------------------------------
----
0 | loss_module | CrossEntropyLoss   | 0      | ?              | ?
1 | accuracy    | MulticlassAccuracy | 0      | ?              | ?
2 | model       | ResNet             | 11.2 M | [1, 3, 32, 32] | [1, 10]
----------------------------------------------------------------------------
----
11.2 M    Trainable params
0         Non-trainable params
11.2 M    Total params
44.727    Total estimated model params size (MB)
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
INFO: Metric val_loss improved. New best score: 1.506
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved. New best score: 1.506
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.328 >= min_delta = 0.0. New best score: 1.179
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.328 >= min_delta = 0.0. New best score: 1.179
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.147 >= min_delta = 0.0. New best score: 1.032
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.147 >= min_delta = 0.0. New best score: 1.032
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.096 >= min_delta = 0.0. New best score: 0.936
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.096 >= min_delta = 0.0. New best score: 0.936
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.053 >= min_delta = 0.0. New best score: 0.883
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.053 >= min_delta = 0.0. New best score: 0.883
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.040 >= min_delta = 0.0. New best score: 0.843
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.040 >= min_delta = 0.0. New best score: 0.843
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.025 >= min_delta = 0.0. New best score: 0.818
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.025 >= min_delta = 0.0. New best score: 0.818
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.019 >= min_delta = 0.0. New best score: 0.799
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.019 >= min_delta = 0.0. New best score: 0.799
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.014 >= min_delta = 0.0. New best score: 0.785
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.014 >= min_delta = 0.0. New best score: 0.785
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.009 >= min_delta = 0.0. New best score: 0.776
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.009 >= min_delta = 0.0. New best score: 0.776
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
INFO: Metric val_loss improved by 0.005 >= min_delta = 0.0. New best score:
0.772
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.005 >= min_delta = 0.0. New best score: 0.772
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.001 >= min_delta = 0.0. New best score:
0.771
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.001 >= min_delta = 0.0. New best score: 0.771
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Monitored metric val_loss did not improve in the last 3 records. Best
score: 0.771. Signaling Trainer to stop.
INFO:lightning.pytorch.callbacks.early_stopping:Monitored metric val_loss di
d not improve in the last 3 records. Best score: 0.771. Signaling Trainer to
stop.
```

Adversarial attacks

Implement the FGSM and PGD attacks. These are white-box evasion attacks, and they were covered in class. Make sure that the final outputs are detached!

Once you finish this part and the previous one, you can head to the Experiments section to test your attacks on the baseline (pretrained) model.

In [ ]:

```python
# Used as a baseline
def id(model, imgs, labels):
    return imgs.detach()

def fgsm(model, imgs, labels, device=torch.device("cuda:0") if torch.cuda.is
    r"""
    Args:
        model (nn.Module): Model to attack, e.g. self.model in the LResnet c
        imgs (Tensor): Tensor of images. Size (BATCH_SIZE, C, H, W). Normali
        labels (Tensor): Tensor of labels. Size (BATCH_SIZE,). Each element 
    Returns:
        adv_imgs (Tensor): Adversarial images. Same dimensions and normaliza
            Each adversarial image in the batch is L_infinity distance at mo
            Images generated by the Fast Gradient Sign Method (FGSM).
    """
    eps = 8/255 # Maximum perturbation
    model.eval()
    # YOUR CODE HERE
    model.to(device)
    imgs = imgs.to(device)
    labels = labels.to(device)
```

```
    imgs.requires_grad = True
    outputs = model(imgs)
    loss = nn.CrossEntropyLoss()(outputs, labels)
    model.zero_grad()
    loss.backward()
    imgs.requires_grad = True
    adv_imgs = imgs + eps * imgs.grad.sign()
    adv_imgs = torch.clamp(adv_imgs, 0, 1).detach()  # Ensure pixel values a
    return adv_imgs

def pgd(model, imgs, labels):
    r"""
    Args:
        model (nn.Module): Model to attack, e.g. self.model in the LResnet c
        imgs (Tensor): Tensor of images. Size (BATCH_SIZE, C, H, W). Normali
        labels (Tensor): Tensor of labels. Size (BATCH_SIZE,). Each element
    Returns:
        adv_imgs (Tensor): Adversarial images. Same dimensions and normaliza
            Each adversarial image in the batch is L_infinity distance at mc
            Images generated by the Projected Gradient Descent (PGD)
    """
    iters = 20 # Number of steps in PGD
    eps = 8/255 # Maximum perturbation
    alpha = 2/255 # Step size
    adv_imgs = imgs.clone().detach()  # Start with the original images
    adv_imgs = adv_imgs + torch.randn_like(adv_imgs) * eps  # Add initial ra
    adv_imgs = torch.clamp(adv_imgs, 0, 1)  # Ensure still in image range

    for _ in range(iters):
        adv_imgs.requires_grad = True
        outputs = model(adv_imgs)
        model.zero_grad()
        loss = nn.CrossEntropyLoss()(outputs, labels)
        loss.backward()
        with torch.no_grad():
            # Apply perturbation
            adv_imgs = adv_imgs + alpha * adv_imgs.grad.sign()
            # Project back into the epsilon-ball around original image
            delta = torch.clamp(adv_imgs - imgs, min=-eps, max=eps)
            adv_imgs = torch.clamp(imgs + delta, min=0, max=1)

    return adv_imgs.detach()


attacks['id'] = id
attacks['fgsm'] = fgsm
attacks['pgd'] = pgd
```

Adversarial Defenses

Adversarial Training

Implement the training loop for an adversarially trained model using PGD as the adversarial example generation method.

Your code should look very similar to the baseline example above. Be sure to save your model in the right place and to store your model in the `models` dictionary. You can adjust `max_epochs` (although early stopping should handle the cases you'd want to) or any other hyperparameters if you'd like. You will need to comment at the end on any changes you've made.

In [ ]:
```
save_key = 'adv_train'
adv_train_model = LResnet(adv_train_method=pgd)

adv_trainer = L.Trainer(
    default_root_dir=SAVE_NAMES[save_key],
    accelerator='auto',
    devices=1,
    max_epochs=30,
    callbacks=[
        ModelCheckpoint(
            dirpath=SAVE_NAMES[save_key],
            monitor='val_loss',
            save_top_k=1,
            mode='min',
            save_weights_only=True,
            every_n_epochs=1,
        ),
        EarlyStopping(
            monitor='val_loss',
            patience=3,
            verbose=True,
            mode='min',
        ),
        LearningRateMonitor('epoch')
    ],
)

adv_trainer.fit(adv_train_model, trainloader, valloader)

# Load best checkpoint after training
adv_train_model = LResnet.load_from_checkpoint(
    adv_trainer.checkpoint_callback.best_model_path
).to(device)

# Store the model in the dictionary
models[save_key] = adv_train_model
```

```
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TP
U cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False, using: 0 IP
Us
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HP
Us
WARNING: Missing logger folder: models/checkpoints/adv_train/lightning_logs
WARNING:lightning.pytorch.loggers.tensorboard:Missing logger folder: models/
checkpoints/adv_train/lightning_logs
/usr/local/lib/python3.10/dist-packages/lightning/pytorch/callbacks/model_ch
eckpoint.py:653: Checkpoint directory /content/models/checkpoints/adv_train
exists and is not empty.
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICE
S: [0]
INFO:
  | Name        | Type                 | Params | In sizes        | Out sizes
------------------------------------------------------------------------------
----
0 | loss_module | CrossEntropyLoss     | 0      | ?               | ?
1 | accuracy    | MulticlassAccuracy | 0      | ?               | ?
2 | model       | ResNet               | 11.2 M | [1, 3, 32, 32] | [1, 10]
------------------------------------------------------------------------------
----
11.2 M    Trainable params
0         Non-trainable params
11.2 M    Total params
44.727    Total estimated model params size (MB)
INFO:lightning.pytorch.callbacks.model_summary:
  | Name        | Type                 | Params | In sizes        | Out sizes
------------------------------------------------------------------------------
----
0 | loss_module | CrossEntropyLoss     | 0      | ?               | ?
1 | accuracy    | MulticlassAccuracy | 0      | ?               | ?
2 | model       | ResNet               | 11.2 M | [1, 3, 32, 32] | [1, 10]
------------------------------------------------------------------------------
----
11.2 M    Trainable params
0         Non-trainable params
11.2 M    Total params
44.727    Total estimated model params size (MB)
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
INFO: Metric val_loss improved. New best score: 7.320
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved. Ne
w best score: 7.320
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 3.094 >= min_delta = 0.0. New best score:
4.227
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
3.094 >= min_delta = 0.0. New best score: 4.227
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.665 >= min_delta = 0.0. New best score:
3.561
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.665 >= min_delta = 0.0. New best score: 3.561
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.114 >= min_delta = 0.0. New best score:
3.447
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.114 >= min_delta = 0.0. New best score: 3.447
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 1.052 >= min_delta = 0.0. New best score:
2.395
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
1.052 >= min_delta = 0.0. New best score: 2.395
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.130 >= min_delta = 0.0. New best score:
2.266
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.130 >= min_delta = 0.0. New best score: 2.266
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.151 >= min_delta = 0.0. New best score:
2.115
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.151 >= min_delta = 0.0. New best score: 2.115
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Monitored metric val_loss did not improve in the last 3 records. Best
score: 2.115. Signaling Trainer to stop.
INFO:lightning.pytorch.callbacks.early_stopping:Monitored metric val_loss di
d not improve in the last 3 records. Best score: 2.115. Signaling Trainer to
stop.
```

SAP

Function implementation

Implement a function that applies Stochastic Activation Pruning (SAP) to a Tensor. Also read the description from the Obfuscated Gradients paper (SAP is described in Section 5.3.1).

Roughly, the algorithm keeps each activation from the previous layer (or, generally, Module) with probability proportional to its absolute value, making this choice independently for each activation, and rescales the kept activations so that the average total activation is not changed.

That is:

1. Let the activation being passed in (from a single image, i.e. assuming batch size 1) be act.
2. Let p be the same shape as the feature act, with values proportional to $|act|$ (absolute value applied element-wise) and sum 1.
3. Let N be the number of entries in the feature. Draw N times *with replacement* from the entries with probability mass function p. Set the selected entries to 1 and the remaining entries to 0 in a Tensor m of the same shape as p (and therefore act).
4. Apply the mask to get $act \circ m$ (element-wise multiplication). Divide each entry by the probability of keeping that entry (i.e. having corresponding 1 in m). Return the result.

Now, the above method runs very slowly. Here's another approach that the authors of Obfuscated Gradients actually use instead:

- Essentially, if we leave each entry with the same probability of being selected as in the original SAP method, but assume we choose whether or not to keep each entry independently (instead of drawing with replacement from all the entries many times), we get a much faster filter. Specifically, once we get p and N, the probability of keeping entry j is $q := 1 - e^{-Np_j}$. Consider it an exercise to prove that this is the case :)
- For the reason from the "Bonus" part at the end of this assignment, the authors of Obfuscated Gradients use probability $1 - e^{-2Np_j}$. Do this as well.
- Normalization is easier because q is records precisely the probability of keeping each entry.
- The time-save is mostly in vectorization.

You may use either approach, although the latter is *much* faster.

Read the above papers for more details. You may also find Erratum interesting.

In [ ]:
```
def sap(act):
    r"""
    Args:
        act (Tensor): Tensor of activations of shape (K, C, H, W), where K i
        The values of C, H, W depend on the layer.
    Returns:
        Tensor of the same shape as act, masked and rescaled according to th
    """
    # YOUR CODE HERE
    N = act.numel() / act.shape[0]  # Total number of entries per example in
    abs_act = act.abs()

    # Compute probabilities proportional to the absolute value of activation
    p = abs_act / abs_act.view(act.shape[0], -1).sum(dim=1, keepdim=True).vi

    # Compute the probability of keeping each entry
    q = 1 - torch.exp(-2 * N * p)

    # Generate the mask: draw random values and compare to q
    random_vals = torch.rand_like(act)
    mask = (random_vals < q).float()

    # Apply the mask and normalize
    pruned_act = act * mask / q.clamp(min=1e-5)  # Clamp q to avoid division

    return pruned_act
```

Adjusted Model

The change you need to make to apply the defense to a ResNet model is simple: simply replace each Conv2d module with a very similar module that applies SAP immediately after convolution. Run the next block and complete the one after that.

In [ ]:
```
class SAP_Conv2d(nn.Conv2d):
    def __init__(
            self,
            in_channels,
            out_channels,
            kernel_size,
            stride=1,
            padding=0,
            groups=1,
            bias=True,
            dilation=1,
    ):
        super().__init__(in_channels, out_channels, kernel_size, stride,
                         padding, dilation, groups, bias)

    # This is the important part
    def _conv_forward(self, input, weight, bias):
        act = super()._conv_forward(input, weight, bias)
        masked_act = sap(act)
        return masked_act
```

In [ ]:
```
def to_sap_conv(model):
    for name, module in model.named_children():
        if isinstance(module, nn.Conv2d):
            sap_conv = SAP_Conv2d(
                in_channels=module.in_channels,
                out_channels=module.out_channels,
                kernel_size=module.kernel_size,
                stride=module.stride,
                padding=module.padding,
                dilation=module.dilation,
                groups=module.groups,
                bias=(module.bias is not None)
            )

            sap_conv.weight.data = module.weight.data.clone()
            if module.bias is not None:
                sap_conv.bias.data = module.bias.data.clone()

            model._modules[name] = sap_conv
        else:
            to_sap_conv(module)
```

In [ ]:
```
sap_conv_model = LResnet()
to_sap_conv(sap_conv_model)

for module in sap_conv_model.modules():
    if isinstance(module, nn.Conv2d):
        print("Found a nn.Conv2d layer that was not replaced.")
        break
else:
    print("All nn.Conv2d layers have been replaced.")
```

```
Found a nn.Conv2d layer that was not replaced.
```

In [ ]:
```
sap_conv_model = LResnet()
to_sap_conv(sap_conv_model)
for module in sap_conv_model.modules():
    print(module)
```

```
LResnet(
  (loss_module): CrossEntropyLoss()
  (accuracy): MulticlassAccuracy()
  (model): ResNet(
    (conv1): SAP_Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3
, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, cei
l_mode=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      )
      (1): BasicBlock(
        (conv1): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      )
    )
    (layer2): Sequential(
```

```
      (0): BasicBlock(
        (conv1): SAP_Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padd
ing=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        (downsample): Sequential(
          (0): SAP_Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=F
alse)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      )
    )
    (layer3): Sequential(
      (0): BasicBlock(
        (conv1): SAP_Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), pad
ding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        (downsample): Sequential(
          (0): SAP_Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=
False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        (relu): ReLU(inplace=True)
```

```
        (conv2): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      )
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): SAP_Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), pad
ding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        (downsample): Sequential(
          (0): SAP_Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=
False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=10, bias=True)
  )
)
CrossEntropyLoss()
MulticlassAccuracy()
ResNet(
  (conv1): SAP_Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3,
3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_
mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding
```

```
=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
    (1): BasicBlock(
      (conv1): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): SAP_Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), paddin
g=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (downsample): Sequential(
        (0): SAP_Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=Fal
se)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
```

```
      (conv1): SAP_Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), paddi
ng=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (downsample): Sequential(
        (0): SAP_Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=Fa
lse)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): SAP_Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), paddi
ng=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (downsample): Sequential(
        (0): SAP_Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=Fa
lse)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), paddi
ng=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), paddi
```

```
ng=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
SAP_Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=Fa
lse)
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
ReLU(inplace=True)
MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
Sequential(
  (0): BasicBlock(
    (conv1): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
  )
  (1): BasicBlock(
    (conv1): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
  )
)
BasicBlock(
  (conv1): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  (relu): ReLU(inplace=True)
  (conv2): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
)
SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
```

```
ReLU(inplace=True)
SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
BasicBlock(
  (conv1): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  (relu): ReLU(inplace=True)
  (conv2): SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
)
SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
ReLU(inplace=True)
SAP_Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
alse)
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
ue)
Sequential(
  (0): BasicBlock(
    (conv1): SAP_Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (downsample): Sequential(
      (0): SAP_Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False
)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
```

```
  )
)
BasicBlock(
  (conv1): SAP_Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1
, 1), bias=False)
  (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (relu): ReLU(inplace=True)
  (conv2): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (downsample): Sequential(
    (0): SAP_Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  )
)
SAP_Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=
False)
BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
ReLU(inplace=True)
SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
Sequential(
  (0): SAP_Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
  (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
)
SAP_Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
BasicBlock(
  (conv1): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
  (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (relu): ReLU(inplace=True)
  (conv2): SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
  (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
)
SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
ReLU(inplace=True)
SAP_Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
```

```
BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
Sequential(
  (0): BasicBlock(
    (conv1): SAP_Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding
=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (downsample): Sequential(
      (0): SAP_Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
  )
)
BasicBlock(
  (conv1): SAP_Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(
1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (relu): ReLU(inplace=True)
  (conv2): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (downsample): Sequential(
    (0): SAP_Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  )
)
SAP_Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias
=False)
BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
ReLU(inplace=True)
```

```
SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
Sequential(
  (0): SAP_Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
  (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
)
SAP_Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
BasicBlock(
  (conv1): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (relu): ReLU(inplace=True)
  (conv2): SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
)
SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
ReLU(inplace=True)
SAP_Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
Sequential(
  (0): BasicBlock(
    (conv1): SAP_Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding
=(1, 1), bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (downsample): Sequential(
      (0): SAP_Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
```

```
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
  )
)
BasicBlock(
  (conv1): SAP_Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(
1, 1), bias=False)
  (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (relu): ReLU(inplace=True)
  (conv2): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
  (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (downsample): Sequential(
    (0): SAP_Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  )
)
SAP_Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias
=False)
BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
ReLU(inplace=True)
SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
Sequential(
  (0): SAP_Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
)
SAP_Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
BasicBlock(
  (conv1): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
  (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (relu): ReLU(inplace=True)
  (conv2): SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(
1, 1), bias=False)
  (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
)
```

```
SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
ReLU(inplace=True)
SAP_Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
AdaptiveAvgPool2d(output_size=(1, 1))
Linear(in_features=512, out_features=10, bias=True)
```

Training

Train an LResnet defended by SAP.

Your code should look very similar to the baseline example above. Be sure to save your model in the right place and to store your model in the `models` dictionary. You can adjust `max_epochs` (although early stopping should handle the cases you'd want to) or any other hyperparameters if you'd like. You will need to comment at the end on any changes you've made.

In [ ]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
sap_conv_model = LResnet().to(device)
to_sap_conv(sap_conv_model)



save_key = 'SAP_conv'
sap_trainer = L.Trainer(
    default_root_dir=SAVE_NAMES[save_key],  # Use the SAP_conv save key
    accelerator='auto',
    devices=1,
    max_epochs=50,  # Adjust as necessary
    callbacks=[
        ModelCheckpoint(
            dirpath=SAVE_NAMES[save_key],
            monitor='val_loss',
            save_top_k=1,
            mode='min',
            save_weights_only=True,
            every_n_epochs=1,
        ),
        EarlyStopping(
            monitor='val_loss',
            patience=3,
            verbose=True,
            mode='min',
        ),
        LearningRateMonitor('epoch'),
    ],
)

# Fit the model using the train and validation data loaders
sap_trainer.fit(sap_conv_model, trainloader, valloader)

best_sap_conv_model = LResnet.load_from_checkpoint(
    sap_trainer.checkpoint_callback.best_model_path
)

best_sap_conv_model = best_sap_conv_model.to(device)



# Store the model in the dictionary
models[save_key] = best_sap_conv_model
```

```
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used:
True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TP
U cores
INFO: IPU available: False, using: 0 IPUs
INFO:lightning.pytorch.utilities.rank_zero:IPU available: False, using: 0 IP
Us
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HP
Us
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICE
S: [0]
INFO:
  | Name        | Type               | Params | In sizes       | Out sizes
----------------------------------------------------------------------------
----
0 | loss_module | CrossEntropyLoss   | 0      | ?              | ?
1 | accuracy    | MulticlassAccuracy | 0      | ?              | ?
2 | model       | ResNet             | 11.2 M | [1, 3, 32, 32] | [1, 10]
----------------------------------------------------------------------------
----
11.2 M    Trainable params
0         Non-trainable params
11.2 M    Total params
44.727    Total estimated model params size (MB)
INFO:lightning.pytorch.callbacks.model_summary:
  | Name        | Type               | Params | In sizes       | Out sizes
----------------------------------------------------------------------------
----
0 | loss_module | CrossEntropyLoss   | 0      | ?              | ?
1 | accuracy    | MulticlassAccuracy | 0      | ?              | ?
2 | model       | ResNet             | 11.2 M | [1, 3, 32, 32] | [1, 10]
----------------------------------------------------------------------------
----
11.2 M    Trainable params
0         Non-trainable params
11.2 M    Total params
44.727    Total estimated model params size (MB)
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved. New best score: 2.045
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved. Ne
w best score: 2.045
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.262 >= min_delta = 0.0. New best score:
1.782
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.262 >= min_delta = 0.0. New best score: 1.782
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.181 >= min_delta = 0.0. New best score: 1.601
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.181 >= min_delta = 0.0. New best score: 1.601
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.109 >= min_delta = 0.0. New best score: 1.492
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.109 >= min_delta = 0.0. New best score: 1.492
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.092 >= min_delta = 0.0. New best score: 1.400
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.092 >= min_delta = 0.0. New best score: 1.400
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.067 >= min_delta = 0.0. New best score: 1.333
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.067 >= min_delta = 0.0. New best score: 1.333
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.061 >= min_delta = 0.0. New best score: 1.272
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.061 >= min_delta = 0.0. New best score: 1.272
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.048 >= min_delta = 0.0. New best score: 1.224
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.048 >= min_delta = 0.0. New best score: 1.224
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.033 >= min_delta = 0.0. New best score: 1.191
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.033 >= min_delta = 0.0. New best score: 1.191
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.062 >= min_delta = 0.0. New best score: 1.129
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.062 >= min_delta = 0.0. New best score: 1.129
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.017 >= min_delta = 0.0. New best score: 1.112
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.017 >= min_delta = 0.0. New best score: 1.112
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.031 >= min_delta = 0.0. New best score: 1.081
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.031 >= min_delta = 0.0. New best score: 1.081
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.033 >= min_delta = 0.0. New best score: 1.048
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.033 >= min_delta = 0.0. New best score: 1.048
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.015 >= min_delta = 0.0. New best score: 1.033
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.015 >= min_delta = 0.0. New best score: 1.033
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.043 >= min_delta = 0.0. New best score: 0.991
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.043 >= min_delta = 0.0. New best score: 0.991
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.007 >= min_delta = 0.0. New best score: 0.983
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.007 >= min_delta = 0.0. New best score: 0.983
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.023 >= min_delta = 0.0. New best score: 0.961
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.023 >= min_delta = 0.0. New best score: 0.961
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.012 >= min_delta = 0.0. New best score: 0.949
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.012 >= min_delta = 0.0. New best score: 0.949
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.021 >= min_delta = 0.0. New best score: 0.928
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.021 >= min_delta = 0.0. New best score: 0.928
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.025 >= min_delta = 0.0. New best score: 0.903
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.025 >= min_delta = 0.0. New best score: 0.903
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.018 >= min_delta = 0.0. New best score: 0.885
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.018 >= min_delta = 0.0. New best score: 0.885
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
INFO: Metric val_loss improved by 0.023 >= min_delta = 0.0. New best score:
0.862
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.023 >= min_delta = 0.0. New best score: 0.862
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.005 >= min_delta = 0.0. New best score:
0.857
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.005 >= min_delta = 0.0. New best score: 0.857
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.007 >= min_delta = 0.0. New best score:
0.850
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.007 >= min_delta = 0.0. New best score: 0.850
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.011 >= min_delta = 0.0. New best score:
0.840
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.011 >= min_delta = 0.0. New best score: 0.840
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.014 >= min_delta = 0.0. New best score:
0.826
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.014 >= min_delta = 0.0. New best score: 0.826
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.005 >= min_delta = 0.0. New best score:
0.821
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.005 >= min_delta = 0.0. New best score: 0.821
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.013 >= min_delta = 0.0. New best score:
0.808
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.013 >= min_delta = 0.0. New best score: 0.808
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.003 >= min_delta = 0.0. New best score:
0.805
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.003 >= min_delta = 0.0. New best score: 0.805
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.012 >= min_delta = 0.0. New best score:
0.792
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.012 >= min_delta = 0.0. New best score: 0.792
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.010 >= min_delta = 0.0. New best score:
0.783
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by
0.010 >= min_delta = 0.0. New best score: 0.783
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.009 >= min_delta = 0.0. New best score: 0.774
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.009 >= min_delta = 0.0. New best score: 0.774
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.000 >= min_delta = 0.0. New best score: 0.774
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.000 >= min_delta = 0.0. New best score: 0.774
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.008 >= min_delta = 0.0. New best score: 0.766
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.008 >= min_delta = 0.0. New best score: 0.766
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.015 >= min_delta = 0.0. New best score: 0.752
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.015 >= min_delta = 0.0. New best score: 0.752
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.005 >= min_delta = 0.0. New best score: 0.746
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.005 >= min_delta = 0.0. New best score: 0.746
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.003 >= min_delta = 0.0. New best score: 0.743
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.003 >= min_delta = 0.0. New best score: 0.743
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.000 >= min_delta = 0.0. New best score: 0.743
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.000 >= min_delta = 0.0. New best score: 0.743
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.007 >= min_delta = 0.0. New best score: 0.737
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.007 >= min_delta = 0.0. New best score: 0.737
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.004 >= min_delta = 0.0. New best score: 0.732
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.004 >= min_delta = 0.0. New best score: 0.732
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
INFO: Metric val_loss improved by 0.004 >= min_delta = 0.0. New best score: 0.728
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.004 >= min_delta = 0.0. New best score: 0.728
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.003 >= min_delta = 0.0. New best score: 0.725
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.003 >= min_delta = 0.0. New best score: 0.725
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: Metric val_loss improved by 0.008 >= min_delta = 0.0. New best score: 0.717
INFO:lightning.pytorch.callbacks.early_stopping:Metric val_loss improved by 0.008 >= min_delta = 0.0. New best score: 0.717
Validation: |          | 0/? [00:00<?, ?it/s]
INFO: `Trainer.fit` stopped: `max_epochs=50` reached.
INFO:lightning.pytorch.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=50` reached.
```

Evaluation

Methods

These two functions help us modularize the experiments we run. Complete `eval_attack` to compute the accuracy of each model (baseline, adversarially trained, SAP) on images. We take every batch in `loader`, apply `attack_method` to the batch, and check the accuracy of `model` in predicting the class of each adversarial image. Output a Float between 0 and 1.

`top_k` describes how we determine accuracy. For example, `top_k=2` means if the model predicts the correct class within its two highest-scoring classes, it's counted as correct.

Complete the next code block and just run the one after that.

In [ ]:
```
def eval_attack(model, attack_method, loader, top_k, max_batches=0):
    r"""
    Args:
        model (LResnet): Model to attack.
        attack_method (function): Adversarial generation method. One of id,
        loader (DataLoader): Data loader for the dataset to evaluate on.
        top_k (int): The number of top predictions to check for correctness.
        max_batches (int): Maximum number of batches to evaluate. If 0, eval
    Returns:
        float: Accuracy of the model on the (adversarially perturbed) datase
    """
    # YOUR CODE HERE
    model.eval()  # Set the model to evaluation mode
    correct = 0
    total = 0

    for batch_idx, (images, labels) in enumerate(loader):
        if max_batches and batch_idx >= max_batches:
            break  # Stop evaluation if max_batches is reached

        images, labels = images.to(device), labels.to(device)
        images.requires_grad = True
        adv_images = attack_method(model, images, labels)  # Generate advers

        outputs = model(adv_images)  # Get model predictions for adversarial
        _, pred = outputs.topk(top_k, 1, True, True)
        pred = pred.t()
        correct += pred.eq(labels.view(1, -1).expand_as(pred)).sum().item()

        total += labels.size(0)

    accuracy = correct / total
    return accuracy
```

In [ ]:
```
def run_experiment(model, attack, top_k=2, max_batches=0):
    # If we're re-running an experiment, remove the old results
    for i in range(len(results_dic['model'])):
        if results_dic['model'][i] == model and results_dic['attack'][i] ==
            results_dic['model'].pop(i)
            results_dic['attack'].pop(i)
            results_dic['top_k'].pop(i)
            results_dic['accuracy'].pop(i)
            break
    # Run the experiment
    acc = eval_attack(
        models[model],
        attacks[attack],
        testloader,
        top_k=top_k,
        max_batches=max_batches
    )
    # Store the results
    results_dic['model'].append(model)
    results_dic['attack'].append(attack)
    results_dic['top_k'].append(top_k)
    results_dic['accuracy'].append(acc)
```

Experiments

In [ ]:
```
torch.set_grad_enabled(True)
torch.autograd.set_detect_anomaly(True)
```

Out[ ]:
```
<torch.autograd.anomaly_mode.set_detect_anomaly at 0x7fb6c9ee2da0>
```

Baseline

The following code runs experiments with all three attacks (including the baseline identity) on the baseline model. Feel free to adjust the parameters or code how you'd like. You will need to comment later on any adjustments you've made.

Adversarially trained

Run the same experiments on the adversarially trained model. You should be able to use very similar code.

SAP

Run the same experiments on the model defended by SAP. You should be able to use very similar code.

In [ ]:
```
for top_k in [1, 2]:
    for model_key in models.keys():
        for attack_method in ['id', 'fgsm', 'pgd']:
            print(f"Running experiment {model_key} with attack {attack_methc
            mb = 100 if attack_method == 'pgd' else 0  # Adjust batch count
            run_experiment(model_key, attack_method, top_k=top_k, max_batche
```

```
Running experiment baseline with attack id and top_k=1...
Running experiment baseline with attack fgsm and top_k=1...
Running experiment baseline with attack pgd and top_k=1...
Running experiment adv_train with attack id and top_k=1...
Running experiment adv_train with attack fgsm and top_k=1...
Running experiment adv_train with attack pgd and top_k=1...
Running experiment SAP_conv with attack id and top_k=1...
Running experiment SAP_conv with attack fgsm and top_k=1...
Running experiment SAP_conv with attack pgd and top_k=1...
Running experiment baseline with attack id and top_k=2...
Running experiment baseline with attack fgsm and top_k=2...
Running experiment baseline with attack pgd and top_k=2...
Running experiment adv_train with attack id and top_k=2...
Running experiment adv_train with attack fgsm and top_k=2...
Running experiment adv_train with attack pgd and top_k=2...
Running experiment SAP_conv with attack id and top_k=2...
Running experiment SAP_conv with attack fgsm and top_k=2...
Running experiment SAP_conv with attack pgd and top_k=2...
```

Display results

We've already stored the results in a dictionary. Let's put them in a Pandas DataFrame to make them nicer to look at. Export your results to a CSV to save them.

It might take some manual work, but if you run any training loop more than once you should probably keep track, e.g. in a spreadsheet or in file names, of which one is which. In particular, always ensure you will know which model is the most recently trained: even better, ensure you'll still know in a month or more.

In [ ]:
```
df_results = pd.DataFrame(results_dic)
df_results
```

Out[ ]:

	model	attack	top_k	accuracy
0	baseline	id	1	0.7383
1	baseline	fgsm	1	0.3053
2	baseline	pgd	1	0.0823
3	adv_train	id	1	0.3044
4	adv_train	fgsm	1	0.5386
5	adv_train	pgd	1	0.3993
6	SAP_conv	id	1	0.7838
7	SAP_conv	fgsm	1	0.3303
8	SAP_conv	pgd	1	0.1547
9	baseline	id	2	0.8855
10	baseline	fgsm	2	0.4805
11	baseline	pgd	2	0.2726
12	adv_train	id	2	0.4827
13	adv_train	fgsm	2	0.7422
14	adv_train	pgd	2	0.6246
15	SAP_conv	id	2	0.9080
16	SAP_conv	fgsm	2	0.5103
17	SAP_conv	pgd	2	0.3690

In [ ]:

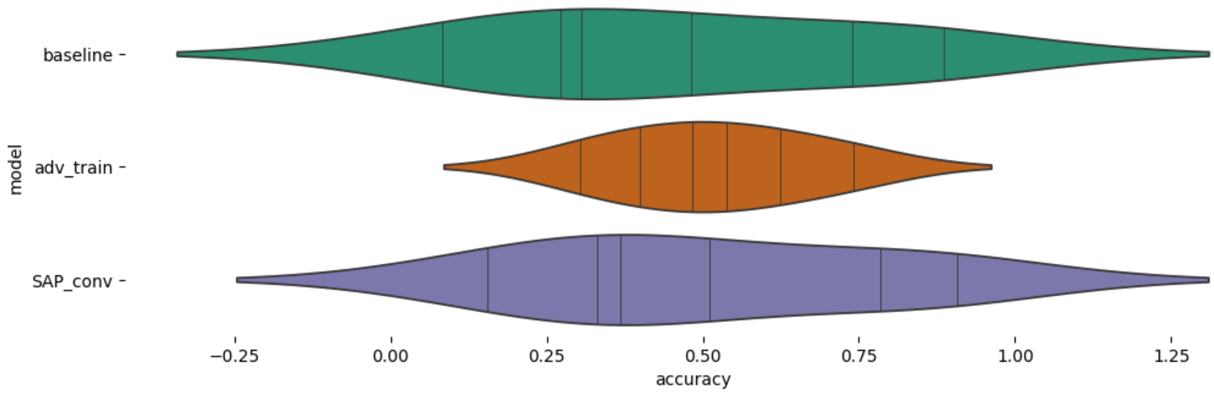
```
# @title model vs accuracy

from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(df_results['model'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(df_results, x='accuracy', y='model', inner='stick', palette='
sns.despine(top=True, right=True, bottom=True, left=True)
```

```
<ipython-input-53-99c32ae43ca2>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the
same effect.

  sns.violinplot(df_results, x='accuracy', y='model', inner='stick', palette
='Dark2')
```



In [ ]:
```
# YOUR CODE HERE
df_results.to_csv('model_evaluation_results.csv', index=False)
```

Tensorboard

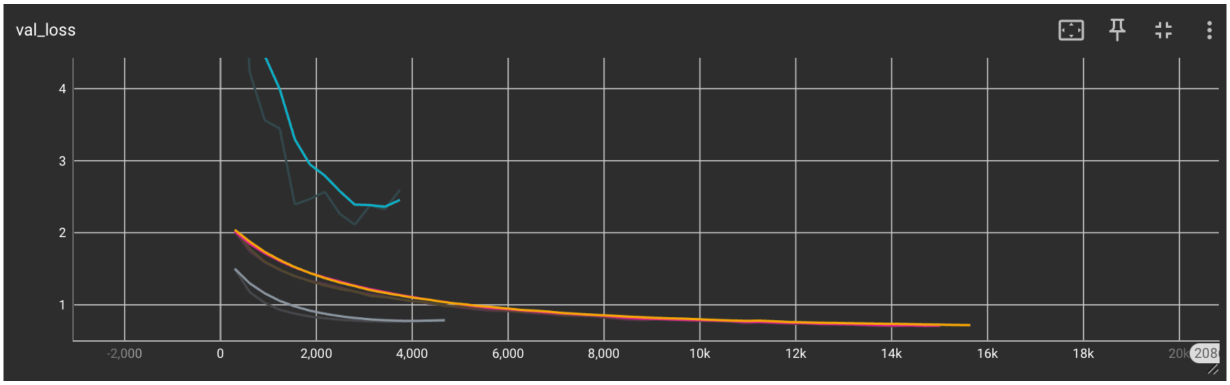
Use the cell below to open a Tensorboard session, and check out the train accuracy/loss and validation loss over the training period. Take screenshots or export images of some salient graphs. Briefly describe what you notice. See the documentation to find how to use Tensorboard with Google Colab.

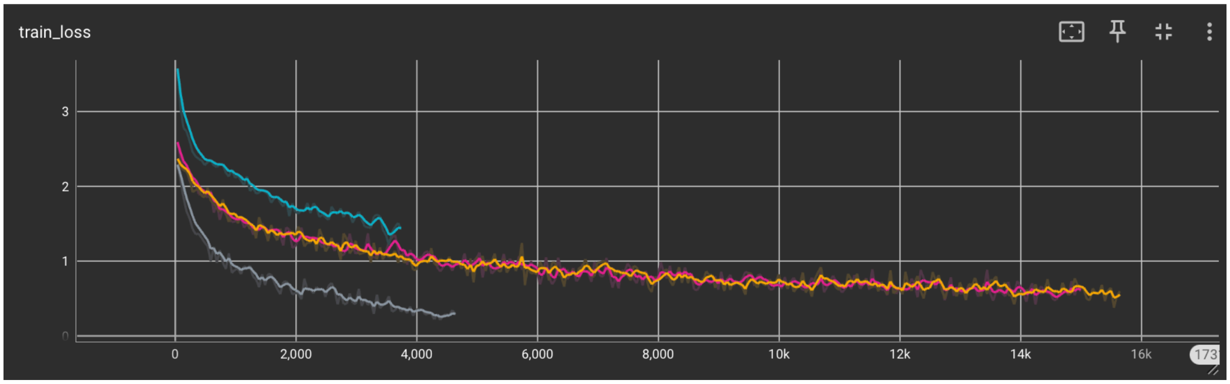
In [ ]:
```
%load_ext tensorboard
```

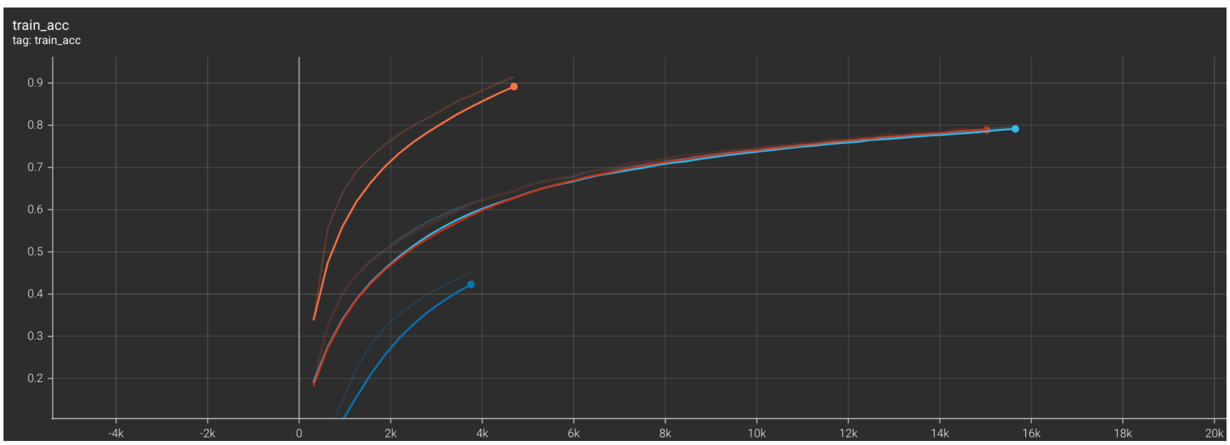
In [ ]:
```
%reload_ext tensorboard
```

In [ ]:
```
%tensorboard --logdir=models/checkpoints/
```

```
Reusing TensorBoard on port 6006 (pid 30247), started 0:00:09 ago. (Use '!ki
ll 30247' to kill it.)
```







In the first two: val loss and train loss, Blue is adversarially trained, Orange/red is SAP, and gray is baseline

In the last graph (train acc), orange is baseline, red/lightblue is SAP, and blue is adv train

the train loss of the SAP and baseline is somewhat similar to the validation loss but as expected, the adversarially trained one has lower train loss than val loss due to the nature of the pgd attack. In addition, we see that the SAP model takes much longer to converge to an optimal val loss than the baseline model due to its complexity and stochasticity slowing its learning

Final question

Comment on your results and any adjustments you've made to the experiments.

1. What did you expect? What met or differed from your expectations?
2. How would you compare the attacks?
3. How would you compare the defenses
   a. In raw performance?
   b. In performance against adversarial examples?
   c. In training time?

Expectations

Baseline Resnet: Expected to perform well under no attack (id) but to have significantly reduced accuracy under adversarial attacks (FGSM, PGD), due to its lack of adversarial training or defenses.

Adversarially Trained Resnet: Expected to show improved robustness against adversarial attacks compared to the baseline, at the cost of some performance under normal conditions.

SAP Defended Resnet: Anticipated to demonstrate enhanced resilience to adversarial attacks, similar to the adversarially trained Resnet, but with potentially better performance under normal conditions due to its defensive mechanism or the fact that it was trained for more epochs, although the baseline did plateau at 13 epochs while SAP ran for all.

Observations

Baseline Resnet's accuracy significantly drops from 73.83% (top-1) and 88.55% (top-2) under normal conditions to 30.53% and 48.05% under FGSM, and further down to 8.23% and 27.26% under PGD. This matches expectations, highlighting its vulnerability to adversarial attacks.

Adversarially Trained Resnet exhibits a drop in accuracy under normal conditions (30.44% top-1, 48.27% top-2) compared to the baseline, likely due to its focus on robustness over accuracy. However, it significantly outperforms the baseline under both attacks, achieving 53.86% and 74.22% under FGSM, and 39.93% and 62.46% under PGD for top-1 and top-2 accuracies, respectively. This improvement against attacks meets expectations.

SAP Defended Resnet shows the highest accuracy under normal conditions (78.38% top-1, 90.80% top-2), indicating superior performance without sacrificing robustness. Under FGSM and PGD attacks, it also demonstrates resilience, albeit not as robust as the adversarially trained model against these specific attacks, with top-1 accuracies of 33.03% and 15.47%, and top-2 accuracies of 51.03% and 36.90%, respectively.

Attacks

PGD is a more potent attack compared to FGSM, as evidenced by the lower accuracies across all models when subjected to PGD. This is expected since PGD is an iterative attack that can more precisely exploit model vulnerabilities.

Defenses

Raw Performance:

SAP Conv exhibits the best raw performance under normal conditions (id), followed by the baseline, and then the adversarially trained model. The adversarial training significantly compromises normal accuracy, likely due to its generalized nature aimed at improving robustness.

Performance Against Adversarial Examples:

Adversarially trained Resnet shows the best performance against both FGSM and PGD, highlighting its effectiveness as a robust defense mechanism against these types of attacks. SAP Conv, while more effective than the baseline, does not match the adversarially trained model's robustness against these attacks but offers a better balance between normal condition performance and defense.

Training Time:

Adversarially training a model generally requires more time due to the need to generate adversarial examples and retrain the model to defend against them. In contrast, SAP's impact on training time can vary depending on the implementation but is expected to be less than adversarially training from scratch since it's a post-processing step applied to an already trained model. However, specific training time data is not provided, making precise comparisons difficult.

Summary:

While adversarial training offers the best defense against FGSM and PGD attacks, it does so at the cost of raw performance under normal conditions. SAP Conv provides a promising balance between maintaining high accuracy under normal conditions and offering some level of defense against adversarial attacks. The choice between these defenses would depend on the specific requirements and constraints of the application, including the expected adversarial threat model and the importance of maintaining high accuracy under normal operating conditions.

Bonus

Technically, because SAP is stochastic, the authors average the outputs of 100 runs. Try implementing this. How does the model's regular performance change? How does its performance against adversarial attacks change?

In [ ]:
```
class SAP_Conv2d(nn.Conv2d):
    def __init__(
            self,
            in_channels,
            out_channels,
            kernel_size,
            stride=1,
            padding=0,
            groups=1,
            bias=True,
            dilation=1,
    ):
        super().__init__(in_channels, out_channels, kernel_size, stride,
                         padding, dilation, groups, bias)


    def _conv_forward(self, input, weight, bias):
        act = super()._conv_forward(input, weight, bias)

        # Initialize a tensor to accumulate SAP-modified activations
        sap_accumulated = torch.zeros_like(act)

        sap_iterations = 100
        for _ in range(sap_iterations):
            sap_modified = sap(act)  # Apply SAP to the activations
            sap_accumulated += sap_modified

        # Average the accumulated SAP-modified activations
        sap_averaged = sap_accumulated / sap_iterations
        print(sap_averaged)
        return sap_averaged
```

In [ ]:
```
# YOUR CODE HERE
for attack_method in ['id', 'fgsm', 'pgd']:
    print(f"Running experiment SAP with attack {attack_method}...")
    mb = 0
    # I've found 100 batches about matches the time of the other attacks' ex
    if attack_method == 'pgd':
        mb = 100
    run_experiment('SAP_conv', attack_method, max_batches=mb)
```

```
Running experiment SAP with attack id...
Running experiment SAP with attack fgsm...
Running experiment SAP with attack pgd...
```

In [ ]:
```
for attack_method in ['id', 'fgsm', 'pgd']:
    print(f"Running experiment SAP with attack {attack_method}...")
    mb = 0
    # I've found 100 batches about matches the time of the other attacks' ex
    if attack_method == 'pgd':
        mb = 100
    run_experiment('SAP_conv', attack_method, top_k=1, max_batches=mb)
```

```
Running experiment SAP with attack id...
Running experiment SAP with attack fgsm...
Running experiment SAP with attack pgd...
```

In [ ]:
```
df_results = pd.DataFrame(results_dic)
df_results
```

Out[ ]:

	model	attack	top_k	accuracy
0	baseline	id	1	0.7383
1	baseline	fgsm	1	0.3053
2	baseline	pgd	1	0.0823
3	adv_train	id	1	0.3044
4	adv_train	fgsm	1	0.5386
5	adv_train	pgd	1	0.3993
6	baseline	id	2	0.8855
7	baseline	fgsm	2	0.4805
8	baseline	pgd	2	0.2726
9	adv_train	id	2	0.4827
10	adv_train	fgsm	2	0.7422
11	adv_train	pgd	2	0.6246
12	SAP_conv	id	2	0.9080
13	SAP_conv	fgsm	2	0.5103
14	SAP_conv	pgd	2	0.3694
15	SAP_conv	id	1	0.7838
16	SAP_conv	fgsm	1	0.3303
17	SAP_conv	pgd	1	0.1550

Old outputs:

6 SAP_conv id 1 0.7838

7 SAP_conv fgsm 1 0.3303

8 SAP_conv pgd 1 0.1547

15 SAP_conv id 2 0.9080

16 SAP_conv fgsm 2 0.5103

17 SAP_conv pgd 2 0.3690

It seems to be the almost the same?

the only notable differences are in pgd with the older pgd accuracies being slightly lower than now by 0.0004

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```