```
for i in [0, 2, 4, 8, 16, 20, 24, 28, 32, 40, 48, 56, 64, 80, 96, 112, 128]:
  print(i/256, end=", ")
```

```
0.0, 0.0078125, 0.015625, 0.03125, 0.0625, 0.078125, 0.09375, 0.109375, 0.125, 0.15625, 0.1875, 0.21875, 0.25, 0.3125, 0.
```

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remou
```

```
import os
os.chdir('/content/drive/Shared with me/')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-20-6ac6876c04b9> in <cell line: 2>()
      1 import os
----> 2 os.chdir('/content/drive/Shared with me/')

FileNotFoundError: [Errno 2] No such file or directory:
'/content/drive/Shared with me/'

SEARCH STACK OVERFLOW
```

```
from __future__ import print_function
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder

import numpy as np
import matplotlib.pyplot as plt

from tqdm.auto import tqdm
from time import sleep
```

Model Under Attack

```
from torchvision.models import resnet18, ResNet18_Weights
```

```
resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
```

```
ResNet(
  (conv1): 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): 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): 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)
    )
    (1): BasicBlock(
      (conv1): 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): 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)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): 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_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): 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_running_stats=True)
      (downsample): Sequential(
        (0): 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)
      )
    )
```

```
          (1): BasicBlock(
            (conv1): 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_running_stats=True)
            (relu): ReLU(inplace=True)
            (conv2): 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_running_stats=True)
          )
        )
        (layer3): Sequential(
          (0): BasicBlock(
            (conv1): 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_running_stats=True)
            (relu): ReLU(inplace=True)
            (conv2): 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_running_stats=True)
            (downsample): Sequential(
              (0): 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): 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_running_stats=True)
            (relu): ReLU(inplace=True)
```

```
# Initialize the Weight Transforms
weights     = ResNet18_Weights.IMAGENET1K_V1
preprocess  = weights.transforms()


# Initialize model
weights = ResNet18_Weights.IMAGENET1K_V1
model   = resnet18(weights=weights)

num_ftrs  = model.fc.in_features
model.fc  = nn.Linear(num_ftrs, 10)


# Load model weights after running the model for the  first time
model.load_state_dict(torch.load('./Resnet_10_class'))
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-18-30ea2ce5969f> in <cell line: 2>()
      1 # Load model weights after running the model for the  first time
----> 2 model.load_state_dict(torch.load('./Resnet_10_class'))

                                 2 frames
/usr/local/lib/python3.10/dist-packages/torch/serialization.py in __init__(self, name, mode)
    250 class _open_file(_opener):
    251     def __init__(self, name, mode):
--> 252         super().__init__(open(name, mode))
    253
    254     def __exit__(self, *args):

FileNotFoundError: [Errno 2] No such file or directory: './Resnet_10_class'

SEARCH STACK OVERFLOW
```

```
# Define what device we are using
use_cuda = True
print("CUDA Available: ",torch.cuda.is_available())
device = torch.device("cuda" if (use_cuda and torch.cuda.is_available()) else "cpu")

# Initialize the network
model = model.to(device)
```

```
CUDA Available:  False
```

```
# unzip the .tgz dataset file
#!tar -xzvf 'imagenette2-320.tgz'


# Dataloader for imagenette2-320

class Imagenette2DataLoader(torch.utils.data.DataLoader):
    def __init__(self, root_dir, batch_size, num_workers=4, shuffle=True):
        """
        Args:
            root_dir (string): Directory with all the images.
            batch_size (int): Number of images in each batch.
            num_workers (int): Number of subprocesses to use for data loading.
            shuffle (bool): Set to True to have the data reshuffled at every epoch.
        """
```

```
        self.root_dir     = root_dir
        self.batch_size   = batch_size
        self.num_workers  = num_workers
        self.shuffle      = shuffle

        # Define the data transforms
        self.transform = transforms.Compose([
            transforms.Resize(320),
            transforms.CenterCrop(320),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
        ])

        # Load the dataset
        self.dataset = ImageFolder(root=self.root_dir, transform=self.transform)

        # Initialize the PyTorch DataLoader
        super().__init__(dataset=self.dataset, batch_size=self.batch_size,
                         shuffle=self.shuffle, num_workers=self.num_workers)



train_loader = Imagenette2DataLoader(root_dir='./imagenette2-320/train', batch_size=16, num_workers=1, shuffle=True)
test_loader = Imagenette2DataLoader(root_dir='./imagenette2-320/val', batch_size=1, num_workers=1, shuffle=True)


print('Number of batches in train dataloader =', len(train_loader))
print('Number of batches in test dataloader =', len(test_loader))
```

```
Number of batches in train dataloader = 592
Number of batches in test dataloader = 3925
```

```
def adjust_learning_rate(optimizer, epoch, init_lr, decay_epochs=30):
    """Sets the learning rate to the initial LR decayed by 10 every 30 epochs"""
    lr = init_lr * (0.1 ** (epoch // decay_epochs))
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr


# Fine-tuning the RessNet18 model with the CrossEntropyLoss.

def train(net, criterion, optimizer, num_epochs, decay_epochs, init_lr):

    for epoch in range(num_epochs):  # loop over the dataset multiple times
        running_loss = 0.0
        running_correct = 0.0
        running_total = 0.0
        start_time = time.time()

        net.train()

        for i, data_pair in enumerate(tqdm(train_loader)):
            adjust_learning_rate(optimizer, epoch, init_lr, decay_epochs)

            # TODO: Set the data to the correct device
            images, labels = data_pair
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()

            outputs = net(images)
            loss = criterion(outputs, labels)
            loss.backward()

            optimizer.step()

            # TODO: Get predicted results
            predicted = torch.argmax(outputs, axis =1)

            # print statistics
            print_freq = 100
            running_loss += loss.item()

            # calc acc
            running_total += labels.size(0)
            running_correct += (predicted == labels).sum().item()

            if i % print_freq == (print_freq - 1):    # print every 2000 mini-batches
                print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / print_freq:.3f} acc: {100*running_correct / running_to
                running_loss, running_correct, running_total = 0.0, 0.0, 0.0
                start_time = time.time()
```

```
    print('Finished Training')


criterion   = nn.CrossEntropyLoss()
optimizer   = torch.optim.Adam(lr = 1e-3, params= model.parameters())

train(model, criterion, optimizer, num_epochs=10, decay_epochs=15, init_lr=1e-3)
```

```
100%                                      592/592 [1:24:33<00:00, 7.74s/it]
[1,   100] loss: 0.101 acc: 96.50 time: 863.84
[1,   200] loss: 0.127 acc: 96.44 time: 849.90
[1,   300] loss: 0.144 acc: 95.75 time: 872.49
[1,   400] loss: 0.141 acc: 95.06 time: 853.83
[1,   500] loss: 0.145 acc: 96.00 time: 861.76
100%                                      592/592 [01:15<00:00, 6.49it/s]
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7f
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    self._shutdown_workers()
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    if w.is_alive():
  File "/usr/lib/python3.10/multiprocessing/process.py", line 160, in is_alive
    assert self._parent_pid == os.getpid(), 'can only test a child process'
AssertionError: can only test a child process
[2,   100] loss: 0.104 acc: 96.31 time: 13.29
[2,   200] loss: 0.132 acc: 96.44 time: 12.68
[2,   300] loss: 0.115 acc: 96.50 time: 12.88
[2,   400] loss: 0.112 acc: 96.31 time: 12.77
[2,   500] loss: 0.111 acc: 96.62 time: 12.86
100%                                      592/592 [01:15<00:00, 8.59it/s]
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7f
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    self._shutdown_workers()
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    if w.is_alive():
  File "/usr/lib/python3.10/multiprocessing/process.py", line 160, in is_alive
    assert self._parent_pid == os.getpid(), 'can only test a child process'
AssertionError: can only test a child process
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7f
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    self._shutdown_workers()
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    if w.is_alive():
  File "/usr/lib/python3.10/multiprocessing/process.py", line 160, in is_alive
    assert self._parent_pid == os.getpid(), 'can only test a child process'
AssertionError: can only test a child process
[3,   100] loss: 0.091 acc: 97.00 time: 12.76
[3,   200] loss: 0.108 acc: 96.38 time: 12.75
[3,   300] loss: 0.096 acc: 96.75 time: 12.70
[3,   400] loss: 0.089 acc: 96.75 time: 12.78
[3,   500] loss: 0.103 acc: 96.88 time: 12.77
100%                                      592/592 [01:15<00:00, 8.81it/s]
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7f
Exception ignored in: Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    self._shutdown_workers()
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    if w.is_alive():
  File "/usr/lib/python3.10/multiprocessing/process.py", line 160, in is_alive
    assert self._parent_pid == os.getpid(), 'can only test a child process'
```

```
torch.save(model.state_dict(), './Resnet_10_class')
```

Fast Gradient Sign Attack

Different epsilons to be used to perturb images for adversarial attack

```
# These are normalized epsilon values, when scaled to 256, these are originally = [0, 2.0, 4.0, 8.0, 16.0, 20.0, 24.0, 28.0, 3
epsilons = [0, 0.0078125, 0.015625, 0.03125, 0.0625, 0.078125, 0.09375, 0.109375, 0.125, 0.15625, 0.1875, 0.21875, 0.25, 0.312
```

FGSM Attack

```
# FGSM attack code
def fgsm_attack(image, epsilon, data_grad):
    # Collect the element-wise sign of the data gradient
    sign_data_grad = data_grad.sign()
    # Create the perturbed image by adjusting each pixel of the input image
    perturbed_image = image + epsilon*sign_data_grad
    # Adding clipping to maintain [0,1] range
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    # Return the perturbed image
    return perturbed_image
```

Testing Function

```
def test_fgsm( model, device, test_loader, epsilon ):

    # Accuracy counter
    correct = 0
    adv_examples = []

    # Loop over all examples in test set
    for i, data_pair in enumerate(tqdm(test_loader)):
        sleep(0.01)

        data, target  = data_pair

        # Send the data and label to the device
        data, target = data.to(device), target.to(device)

        # Set requires_grad attribute of tensor. Important for Attack
        data.requires_grad = True

        # Forward pass the data through the model
        output = model(data)
        init_pred = output.max(1, keepdim=True)[1] # get the index of the max log-probability
        init_pred = init_pred.squeeze()

        # If the initial prediction is wrong, don't bother attacking, just move on
        if init_pred.item() != target.item():
            continue

        # Calculate the loss
        loss = F.nll_loss(output, target)

        # Zero all existing gradients
        model.zero_grad()

        # Calculate gradients of model in backward pass
        loss.backward()

        # Collect ``datagrad``
        data_grad = data.grad.data

        # Call FGSM Attack
        perturbed_data = fgsm_attack(data, epsilon, data_grad)

        # Re-classify the perturbed image
        output = model(perturbed_data)

        # Check for success
        final_pred = output.max(1, keepdim=True)[1] # get the index of the max log-probability
        if final_pred.item() == target.item():
```

```
            correct += 1
            # Special case for saving 0 epsilon examples
            if (epsilon == 0) and (len(adv_examples) < 5):
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex) )
        else:
            # Save some adv examples for visualization later
            if len(adv_examples) < 5:
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex) )

    # Calculate final accuracy for this epsilon
    final_acc = correct/float(len(test_loader))
    print("Epsilon: {}\tTest Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader), final_acc))

    # Return the accuracy and an adversarial example
    return final_acc, adv_examples
```

Run the attack

```
torch.cuda.empty_cache()
```

```
import gc
gc.collect()
```

```
7
```

```
torch.cuda.memory_summary(device=None, abbreviated=False)
```

```
'|===========================================================================
|\n|                  PyTorch CUDA memory summary, device ID 0                 
|\n|---------------------------------------------------------------------------
--|\n|            CUDA OOMs: 0            |        cudaMalloc retries: 0      
|\n|===========================================================================
==|\n|        Metric         | Cur Usage  | Peak Usage | Tot Alloc  | Tot Fre
ed  |\n|---------------------------------------------------------------------------
```

```
accuracies_fgsm = []
examples_fgsm = []

# Set the model in evaluation mode. In this case this is for the Dropout layers
model.eval()

# Run test for each epsilon
for eps in epsilons:
    print('Value of epsilon = ', eps)
    acc, ex = test_fgsm(model, device, test_loader, eps)
    accuracies_fgsm.append(acc)
    examples_fgsm.append(ex)
```

```
Value of epsilon =  0
```

```
100%                                      3925/3925 [01:39<00:00, 38.16it/s]
```

```
Epsilon: 0      Test Accuracy = 2344 / 3925 = 0.5971974522292993
Value of epsilon =  0.0078125
```

```
100%                                      3925/3925 [01:37<00:00, 43.17it/s]
```

```
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7f
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    self._shutdown_workers()
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    if w.is_alive():
  File "/usr/lib/python3.10/multiprocessing/process.py", line 160, in is_alive
    assert self._parent_pid == os.getpid(), 'can only test a child process'
AssertionError: can only test a child process
Epsilon: 0.0078125      Test Accuracy = 2182 / 3925 = 0.5559235668789809
Value of epsilon =  0.015625
```

```
100%                                      3925/3925 [01:38<00:00, 42.22it/s]
```

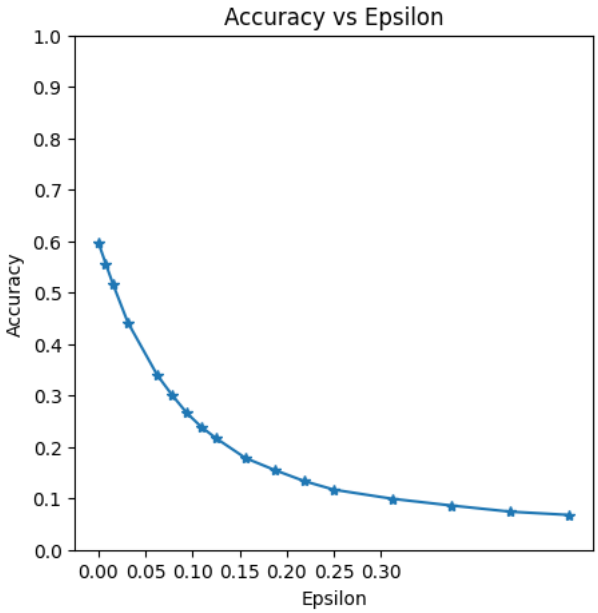
```
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7f
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    self._shutdown_workers()
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    if w.is_alive():
  File "/usr/lib/python3.10/multiprocessing/process.py", line 160, in is_alive
    assert self._parent_pid == os.getpid(), 'can only test a child process'
AssertionError: can only test a child process
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7f
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    self._shutdown_workers()
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    if w.is_alive():
  File "/usr/lib/python3.10/multiprocessing/process.py", line 160, in is_alive
    assert self._parent_pid == os.getpid(), 'can only test a child process'
AssertionError: can only test a child process
Epsilon: 0.015625       Test Accuracy = 2024 / 3925 = 0.5156687898089172
Value of epsilon =  0.03125
```

```
100%                                      3925/3925 [01:39<00:00, 40.67it/s]
```

```
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7f
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    self._shutdown_workers()
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    if w.is_alive():
  File "/usr/lib/python3.10/multiprocessing/process.py", line 160, in is_alive
    assert self._parent_pid == os.getpid(), 'can only test a child process'
AssertionError: can only test a child process
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7f
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    self._shutdown_workers()
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
    if w.is_alive():
  File "/usr/lib/python3.10/multiprocessing/process.py", line 160, in is_alive
```

Accuracy vs Epsilon

```
plt.figure(figsize=(5,5))
plt.plot(epsilons, accuracies_fgsm, "*-")
plt.yticks(np.arange(0, 1.1, step=0.1))
plt.xticks(np.arange(0, .35, step=0.05))
plt.title("Accuracy vs Epsilon")
plt.xlabel("Epsilon")
plt.ylabel("Accuracy")
plt.show()
```




```
# Plot several examples of adversarial samples at each epsilon
cnt = 0
plt.figure(figsize=(32,32))
for i in range(len(epsilons)):
    for j in range(len(examples_fgsm[i])):
        cnt += 1
        plt.subplot(len(epsilons),len(examples_fgsm[0]),cnt)
        plt.xticks([], [])
        plt.yticks([], [])
        if j == 0:
            plt.ylabel("Eps: {}".format(epsilons[i]), fontsize=14)
        orig,adv,ex = examples_fgsm[i][j]
        plt.title("{} -> {}".format(orig, adv))
        plt.imshow(ex.T)
plt.tight_layout()
plt.show()
```


5 -> 2
8 -> 8
0 -> 0
5 -> 9
8 -> 8
Eps: 0
0 -> 9
5 -> 4
3 -> 9
3 -> 9
3 -> 6
Eps: 0.0078125
3 -> 1
5 -> 4
3 -> 1
3 -> 9
1 -> 9
Eps: 0.015625
6 -> 0
7 -> 2
0 -> 3
5 -> 2
5 -> 2
Eps: 0.03125
7 -> 2
8 -> 4
8 -> 4
0 -> 9
6 -> 9
Eps: 0.0625
2 -> 9
5 -> 1
9 -> 4
7 -> 4
3 -> 1
Eps: 0.078125
6 -> 4
9 -> 4
1 -> 9
6 -> 9
4 -> 8
Eps: 0.09375
4 -> 9
6 -> 1
9 -> 3
0 -> 9
9 -> 8
Eps: 0.109375
5 -> 4
5 -> 2
4 -> 3
7 -> 2
5 -> 2
Eps: 0.125
9 -> 1
9 -> 1
6 -> 0
7 -> 9
8 -> 9
Eps: 0.14625
7 -> 2
1 -> 9
4 -> 9
9 -> 2
9 -> 0
Eps: 0.1875
0 -> 4
9 -> 1
3 -> 9
5 -> 9
4 -> 7
Eps: 0.21875
6 -> 2
9 -> 4
4 -> 1
4 -> 1
0 -> 1
Eps: 0.25
0 -> 9
3 -> 4
3 -> 8
1 -> 7
0 -> 9
Eps: 0.3125
9 -> 4
4 -> 8
6 -> 4
8 -> 9
0 -> 8
Eps: 0.375
4 -> 9
9 -> 1
1 -> 4
5 -> 9
6 -> 7
Eps: 0.4375
9 -> 8
7 -> 2
6 -> 4
5 -> 9
9 -> 1
Eps: 0.5

Iterative gradient sign method/Basic iterative method

```
# These are normalized epsilon values, when scaled to 256, these are originally = [0, 2.0, 4.0, 8.0, 16.0, 20.0, 24.0, 28.0, 3
epsilons  = [0, 0.0078125, 0.015625, 0.03125, 0.0625, 0.078125, 0.09375, 0.109375, 0.125, 0.15625, 0.1875, 0.21875, 0.25, 0.31
alpha     = 0.25
num_iter  = 10


def iterative_attack(model, image, label, epsilon, alpha, num_item):
    # Set the model to evaluation mode
    model.eval()

    # Create a copy of the input image to perturb
    perturbed_image = image.clone()

    for i in range(num_iter):
        # Set requires_grad attribute of tensor. Important for attack
        perturbed_image = image.clone().detach()
        perturbed_image.requires_grad = True

        # Forward pass the data through the model
        output = model(perturbed_image)

        # Calculate the loss
        loss = F.cross_entropy(output, label)

        # Zero all existing gradients
        model.zero_grad()

        # Calculate gradients of model in backward pass
        loss.backward()

        # Collect the element-wise sign of the data gradient
        data_grad       = perturbed_image.grad.data
        sign_data_grad  = data_grad.detach().sign()

        # Create the perturbed image by adjusting each pixel of the input image
        perturbed_image = perturbed_image + alpha*sign_data_grad

        # Project the perturbation onto an epsilon ball
        perturbed_image = torch.max(torch.min(perturbed_image, image + epsilon), image - epsilon).clamp(0, 1)

    # Return the perturbed image
    return perturbed_image



def test_bim(model, device, test_loader, epsilon, alpha, num_item):

    # Accuracy counter
    correct = 0
    adv_examples = []

    # Loop over all examples in test set
    for i, data_pair in enumerate(tqdm(test_loader)):
        sleep(0.01)

        data, target  = data_pair

        # Send the data and label to the device
        data, target = data.to(device), target.to(device)

        # Set requires_grad attribute of tensor. Important for Attack
        data.requires_grad = True

        # Forward pass the data through the model
        output = model(data)
        init_pred = output.max(1, keepdim=True)[1] # get the index of the max log-probability
        init_pred = init_pred.squeeze()

        # If the initial prediction is wrong, don't bother attacking, just move on
        if init_pred.item() != target.item():
            continue

        # Call Iterative attack Attack
        perturbed_data = iterative_attack(model, data, target, epsilon, alpha, num_item)

        # Re-classify the perturbed image
        output = model(perturbed_data)
```

```
        # Check for success
        final_pred = output.max(1, keepdim=True)[1] # get the index of the max log-probability
        if final_pred.item() == target.item():
            correct += 1
            # Special case for saving 0 epsilon examples
            if (epsilon == 0) and (len(adv_examples) < 5):
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex) )
        else:
            # Save some adv examples for visualization later
            if len(adv_examples) < 5:
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append( (init_pred.item(), final_pred.item(), adv_ex) )

    # Calculate final accuracy for this epsilon
    final_acc = correct/float(len(test_loader))
    print("Epsilon: {}\tTest Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader), final_acc))

    # Return the accuracy and an adversarial example
    return final_acc, adv_examples
```

Run the attack

```
torch.cuda.empty_cache()
```

```
import gc
gc.collect()
```

```
0
```

```
torch.cuda.memory_summary(device=None, abbreviated=False)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-21-a7e933d74b36> in <cell line: 1>()
----> 1 torch.cuda.memory_summary(device=None, abbreviated=False)

/usr/local/lib/python3.10/dist-packages/torch/cuda/memory.py in
memory_summary(device, abbreviated)
    516             prefix = metric_key + "." + submetric_key + "."
    517
--> 518             current = stats[prefix + "current"]
    519             peak = stats[prefix + "peak"]
    520             allocated = stats[prefix + "allocated"]

KeyError: 'allocated_bytes.all.current'
```

SEARCH STACK OVERFLOW

```
accuracies_bim = []
examples_bim = []

# Set the model in evaluation mode. In this case this is for the Dropout layers
model.eval()

# Run test for each epsilon
for eps in epsilons:
    print('Value of epsilon = ', eps)
    acc, ex = test_bim(model, device, test_loader, eps, alpha, num_iter)
    accuracies_bim.append(acc)
    examples_bim.append(ex)
```