Assignment 3 Part 3

```
import os
import numpy as np
import torch
import torch.nn as nn
import torchvision

from torchvision import transforms, datasets
# from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
import torchvision.models as models
import torch.nn.functional as F

%matplotlib inline
%load_ext autoreload
%autoreload 2
```

```
from gan.utils import sample_noise, show_images, deprocess_img, preprocess_img
```

Custom Loss Functions

From [Wanglei]

```
def linear_combination(x, y, epsilon):
    return epsilon * x + (1 - epsilon) * y


def reduce_loss(loss, reduction='mean'):
    return loss.mean() if reduction == 'mean' else loss.sum() if reduction == 'sum' els


class LabelSmoothingCrossEntropy(nn.Module):
    def __init__(self, epsilon: float = 0.1, reduction='mean'):
        super().__init__()
        self.epsilon = epsilon
        self.reduction = reduction

    def forward(self, preds, target):
        n = preds.size()[-1]
        log_preds = F.log_softmax(preds, dim=-1)
        loss = reduce_loss(-log_preds.sum(dim=-1), self.reduction)
        nll = F.nll_loss(log_preds, target, reduction=self.reduction)
        return linear_combination(loss / n, nll, self.epsilon)

```

0s completed at 8:05 PM

In the following cell we will load the training data and also apply some transforms to the data.

```
# get the imagenette dataset
!wget https://s3.amazonaws.com/fast-ai-imageclas/imagenette2.tgz
!tar -xzvf imagenette2.tgz
!rm imagenette2.tgz
```

Streaming output truncated to the last 5000 lines.

```
imagenette2/train/n03888257/n03888257_16077.JPEG
imagenette2/train/n03888257/n03888257_23339.JPEG
imagenette2/train/n03888257/n03888257_44204.JPEG
imagenette2/train/n03888257/n03888257_61633.JPEG
imagenette2/train/n03888257/n03888257_15067.JPEG
imagenette2/train/n03888257/n03888257_75365.JPEG
imagenette2/train/n03888257/n03888257_63966.JPEG
imagenette2/train/n03888257/n03888257_3927.JPEG
imagenette2/train/n03888257/n03888257_20684.JPEG
imagenette2/train/n03888257/ILSVRC2012_val_00047778.JPEG
imagenette2/train/n03888257/n03888257_14016.JPEG
imagenette2/train/n03888257/n03888257_37776.JPEG
imagenette2/train/n03888257/ILSVRC2012_val_00041706.JPEG
imagenette2/train/n03888257/n03888257_17513.JPEG
imagenette2/train/n03888257/n03888257_17143.JPEG
imagenette2/train/n03888257/n03888257_6738.JPEG
imagenette2/train/n03888257/n03888257_4355.JPEG
imagenette2/train/n03888257/ILSVRC2012_val_00030583.JPEG
imagenette2/train/n03888257/n03888257_18127.JPEG
imagenette2/train/n03888257/n03888257_10844.JPEG
imagenette2/train/n03888257/n03888257_6503.JPEG
imagenette2/train/n03888257/n03888257_12195.JPEG
imagenette2/train/n03888257/n03888257_14368.JPEG
imagenette2/train/n03888257/n03888257_11886.JPEG
imagenette2/train/n03888257/n03888257_5293.JPEG
imagenette2/train/n03888257/n03888257_10014.JPEG
imagenette2/train/n03888257/n03888257_30158.JPEG
imagenette2/train/n03888257/n03888257_11255.JPEG
imagenette2/train/n03888257/n03888257_38929.JPEG
imagenette2/train/n03888257/n03888257_10106.JPEG
imagenette2/train/n03888257/n03888257_36937.JPEG
imagenette2/train/n03888257/n03888257_128.JPEG
imagenette2/train/n03888257/n03888257_23723.JPEG
imagenette2/train/n03888257/n03888257_18023.JPEG
imagenette2/train/n03888257/n03888257_25024.JPEG
imagenette2/train/n03888257/n03888257_8225.JPEG
imagenette2/train/n03888257/n03888257_20676.JPEG
imagenette2/train/n03888257/n03888257_4154.JPEG
imagenette2/train/n03888257/n03888257_18664.JPEG
imagenette2/train/n03888257/n03888257_29569.JPEG
imagenette2/train/n03888257/n03888257_21159.JPEG
imagenette2/train/n03888257/ILSVRC2012_val_00022496.JPEG
imagenette2/train/n03888257/n03888257_15797.JPEG
imagenette2/train/n03888257/n03888257_75495.JPEG
imagenette2/train/n03888257/n03888257_4553.JPEG
imagenette2/train/n03888257/n03888257_24025.JPEG
```

```
imagenette2/train/n03888257/n03888257_10653.JPEG
imagenette2/train/n03888257/n03888257_50108.JPEG
imagenette2/train/n03888257/n03888257_10079.JPEG
imagenette2/train/n03888257/n03888257_8423.JPEG
imagenette2/train/n03888257/n03888257_41099.JPEG
imagenette2/train/n03888257/n03888257_27166.JPEG
imagenette2/train/n03888257/n03888257_10645.JPEG
```

```
nb_epochs = 45
batch_size = 64
learning_rate = 0.001
scale_size = 256

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std= [0.229, 0.224, 0.225])
data_dir = './imagenette2'
# print(os.listdir(data_dir))
train_dataset = ImageFolder(root=data_dir+'/train',
                            transform=transforms.Compose([
                              transforms.Resize((scale_size, scale_size)),
                              transforms.CenterCrop(256),
                              transforms.ToTensor(),
                              normalize,
                        ]))
val_dataset = ImageFolder(root=data_dir+'/val',
                            transform=transforms.Compose([
                                transforms.Resize((scale_size, scale_size)),
                                transforms.CenterCrop(256),
                                transforms.ToTensor(),
                                normalize,
                            ]))

train_loader_im = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_wo
val_loader_im = DataLoader(val_dataset , batch_size=batch_size, num_workers=1, pin_memo

imgs = train_loader_im.__iter__().next()[0].numpy().squeeze()
```

Pretrained Models

In this project, I used the pretrained AlexNet model

```
num_epochs = 4
test_frequency = 5

# Load Pretrained AlexNet
classifier = torchvision.models.alexnet(pretrained=True)
# set a new output size
```

```
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /root/
100%                                        233M/233M [00:04<00:00, 55.7MB/s]
```

Training the network

```python
def train_classifier(train_loader, classifier, criterion, optimizer):
    classifier.train()
    loss_ = 0.0
    losses = []
    for i, (images, labels) in enumerate(train_loader):
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        logits = classifier(images)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()
        losses.append(loss)
    return torch.stack(losses).mean().item()
```

```python
def test_classifier(test_loader, classifier, criterion, print_ind_classes=True, print_
    classifier.eval()
    losses = []
    correct = 0
    total = 0
    with torch.no_grad():
        for i, (images, labels) in enumerate(test_loader):
            images, labels = images.to(device), labels.to(device)
            logits = classifier(images)
            # loss = F.nll_loss(logits, labels)
            loss = criterion(logits, labels)
            losses.append(loss.item())
            # ignore first class which is background
            preds = logits.cpu().detach().numpy()
            correct += (np.argmax(preds, axis=1) == labels.cpu().detach().numpy()).su
            total += test_loader.batch_size

        mAP = float(correct) / float(total)
        test_loss = np.mean(losses)
        if print_total:
            print('Acc: {0:.4f}'.format(mAP))
            print('Avg loss: {}'.format(test_loss))

    return mAP, test_loss, 0
```

```
    for epoch in range(1,num_epochs+1):
        print("Starting epoch number " + str(epoch))
        train_loss = train_classifier(train_loader, classifier, criterion, optimizer)
        train_losses.append(train_loss)
        print("Loss for Training on Epoch " +str(epoch) + " is "+ str(train_loss))
        if(epoch%test_frequency==0 or epoch==1):
            mAP_train, _, _ = test_classifier(train_loader, classifier, criterion, Fals
            train_mAPs.append(mAP_train)
            mAP_val, val_loss, _ = test_classifier(val_loader, classifier, criterion)
            print('Evaluating classifier')
            print("Mean Precision Score for Testing on Epoch " +str(epoch) + " is "+ st
            val_losses.append(val_loss)
            val_mAPs.append(mAP_val)

    return classifier, train_losses, val_losses, train_mAPs, val_mAPs
```

```
classifier, train_losses, val_losses, train_mAPs, val_mAPs = train(classifier, num_epoc
```

```
Starting epoch number 1
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:718: UserWarning: Named
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
Loss for Training on Epoch 1 is 53.87113952636719
Acc: 0.9100
Avg loss: 46.27601300516436
Evaluating classifier
Mean Precision Score for Testing on Epoch 1 is 0.9100302419354839
Starting epoch number 2
Loss for Training on Epoch 2 is 40.755882263183594
Starting epoch number 3
Loss for Training on Epoch 3 is 36.26933288574219
Starting epoch number 4
Loss for Training on Epoch 4 is 34.52720260620117
```

```
# Test on clean data
mAP_test, test_loss, test_aps = test_classifier(val_loader_im, classifier, criterion)
print("Test accuracy: ", mAP_test)
```

```
Acc: 0.9378
Avg loss: 40.88100867117605
Test accuracy:  0.9377520161290323
```

Advererial Attack

```
# FGSM attack code
def fgsm_attack(image, epsilon, data_grad):
```

```
# Iterative projected gradient Method
def igm_attack(model, X, y, thres, alpha, num_iter):
    """ Construct igm adversarial examples on the examples X"""
    delta = torch.zeros_like(X, requires_grad=True)
    for t in range(num_iter):
        loss = F.nll_loss(model(X + delta), y)
        # loss = nn.CrossEntropyLoss()(model(X + delta), y)
        loss.backward()
        delta.data = (delta + alpha*delta.grad.detach().sign()).clamp(-thres, thres)
        delta.grad.zero_()
    return delta.detach() + X

```

```
#Evaluate results on adversarially perturbed
def eval_advAttack(model=None, test_loader=None, thres=0.1, num_iter=40):
    total = 0
    correct = 0
    print("Evaluating single model results on adv data")
    for data, target in test_loader:
      if torch.cuda.is_available():
        data, target = data.cuda(), target.cuda()
      # print('new data ---------------------------------------------')
      # disp_fake_images = deprocess_img(data.data)  # denormalize
      # imgs_numpy = (disp_fake_images).cpu().numpy()
      # show_images(imgs_numpy[0:4],color=True)
      # plt.show()
      # print()

      # Call IGM Attack
      perturbed_data = igm_attack(model, data, target, thres=thres, alpha=1e-3, num_ite

      # disp_fake_images = deprocess_img(perturbed_data.data)  # denormalize
      # imgs_numpy = (disp_fake_images).cpu().numpy()
      # show_images(imgs_numpy[0:4], color=True)
      # plt.show()
      # print()

      # Re-classify the perturbed image
      output = model(perturbed_data)
      # Check for success
      preds_np = output.cpu().detach().numpy()
      # finalPred = np.argmax(preds_np, axis=1)
      correct += (np.argmax(preds_np, axis=1) == target.cpu().detach().numpy()).sum()
      # correct += (finalPred == target.cpu().detach().numpy()).sum()
      total += test_loader.batch_size
```

Defense Mechanism

Firstly, we will use training the model with the adverserial data

```
#Adversarial Training
def adv_train(model, train_loader, criterion, thres=0.1, num_iter=40, nb_epochs=nb_ep
                batch_size=batch_size, train_end=-1, test_end=-1, learning_rate=learn

    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
    train_loss = []
    total = 0
    correct = 0
    totalAdv = 0
    correctAdv = 0
    step = 0
    # breakstep = 0
    for _epoch in range(nb_epochs):
      for data, target in train_loader:
        #Normal Training
        if torch.cuda.is_available():
          data1, target = data.cuda(), target.cuda()

        optimizer.zero_grad()
        preds = model(data1)
        # Calculate the loss
        loss = criterion(preds, target1)
        # loss = F.nll_loss(preds, target)
        model.zero_grad()
        loss.backward()  # calc gradients

        train_loss.append(loss.data.item())
        optimizer.step()  # update gradients

        preds_np = preds.cpu().detach().numpy()
        correct += (np.argmax(preds_np, axis=1) == target.cpu().detach().numpy()).sum
        total += train_loader.batch_size

        #Adversarial Training
        # Call IGM Attack
        data_adv = igm_attack(model, data, target, thres=thres, alpha=1e-3, num_iter=

        if torch.cuda.is_available():
```

```
        if total % 2000 == 0:
          acc = float(correct) / float(total)
          print('[%s] Clean Training accuracy: %.2f%%' % (step, acc * 100))
          total = 0
          correct = 0
          accAdv = float(correctAdv) / float(totalAdv)
          print('[%s] Adv Training accuracy: %.2f%%' % (step, accAdv * 100))
          totalAdv = 0
          correctAdv = 0

```

```
print("Training on Adversarial Samples")
adv_train(classifier, train_loader_im, criterion, thres=0.07, num_iter=20)
```

```
#Evaluating Again
mAP_test, test_loss, test_aps = test_classifier(val_loader_im, classifier, criterion)
print("Test accuracy: ", mAP_test)
eval_advAttack(classifier, val_loader_im, thres=0.01, num_iter=15)
```

```
mAP: 0.7926
Avg loss: 65.56647073068926
Test accuracy:  0.7925907258064516
Evaluating single model results on adv data
Test Accuracy = 2778 / 3968 = 70.010%
```

```

```