Assignment 2 Part 2: Developing Your Own Classifier

```
!wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar
!tar -xf VOCtrainval_06-Nov-2007.tar
!mv VOCdevkit VOCdevkit_2007
# download test and combine into same directory
!wget http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest_06-Nov-2007.tar
!tar -xf VOCtest_06-Nov-2007.tar
!mv VOCdevkit/VOC2007 VOCdevkit_2007/VOC2007test
!rmdir VOCdevkit
```

```
--2021-10-05 14:46:57--  http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_
Resolving host.robots.ox.ac.uk (host.robots.ox.ac.uk)... 129.67.94.152
Connecting to host.robots.ox.ac.uk (host.robots.ox.ac.uk)|129.67.94.152|:80... conne
HTTP request sent, awaiting response... 200 OK
Length: 460032000 (439M) [application/x-tar]
Saving to: ‘VOCtrainval_06-Nov-2007.tar’

VOCtrainval_06-Nov- 100%[===================>] 438.72M  4.89MB/s    in 93s

2021-10-05 14:48:31 (4.71 MB/s) - ‘VOCtrainval_06-Nov-2007.tar’ saved [460032000/460

--2021-10-05 14:48:33--  http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest_06-N
Resolving host.robots.ox.ac.uk (host.robots.ox.ac.uk)... 129.67.94.152
Connecting to host.robots.ox.ac.uk (host.robots.ox.ac.uk)|129.67.94.152|:80... conne
HTTP request sent, awaiting response... 200 OK
Length: 451020800 (430M) [application/x-tar]
Saving to: ‘VOCtest_06-Nov-2007.tar’

VOCtest_06-Nov-2007 100%[===================>] 430.13M  4.94MB/s    in 92s

2021-10-05 14:50:05 (4.70 MB/s) - ‘VOCtest_06-Nov-2007.tar’ saved [451020800/4510208
```

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

from torchvision import transforms
from sklearn.metrics import average_precision_score
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
from kaggle_submission import output_submission_csv
from classifier import SimpleClassifier, Classifier#, AlexNet
from voc_dataloader import VocDataset, VOC_CLASSES

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

✓ 0s completed at 12:06 PM

```
torch.Size([2, 21])
```

Double-click (or enter) to edit

Part 2: Design your own network

In this notebook, your task is to create and train your own model for multi-label classification on VOC Pascal.

What to do

1. You will make change on network architecture in `classifier.py`.
2. You may also want to change other hyperparameters to assist your training to get a better performances. Hints will be given in the below instructions.

What to submit

Check the submission template for details what to submit.

```
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()
```

```
def test_classifier(test_loader, classifier, criterion, print_ind_classes=True, prin
    classifier.eval()
    losses = []
    with torch.no_grad():
        y_true = np.zeros((0,21))
        y_score = np.zeros((0,21))
        for i, (images, labels) in enumerate(test_loader):
            images, labels = images.to(device), labels.to(device)
            logits = classifier(images)
            y_true = np.concatenate((y_true, labels.cpu().numpy()), axis=0)
        for i in range(1, y_true.shape[1]):
            ap = average_precision_score(y_true[:, i], y_score[:, i])
            if print_ind_classes:
                print('-------  Class: {:<12}     AP: {:>8.4f}  -------'.format(VOC_
            aps.append(ap)

        mAP = np.mean(aps)
        test_loss = np.mean(losses)
        if print_total:
            print('mAP: {0:.4f}'.format(mAP))
            print('Avg loss: {}'.format(test_loss))

    return mAP, test_loss, aps
```

```
def plot_losses(train, val, test_frequency, num_epochs):
    plt.plot(train, label="train")
    indices = [i for i in range(num_epochs) if ((i+1)%test_frequency == 0 or i ==0)]
    plt.plot(indices, val, label="val")
    plt.title("Loss Plot")
    plt.ylabel("Loss")
    plt.xlabel("Epoch")
    plt.legend()
    plt.show()

def plot_mAP(train, val, test_frequency, num_epochs):
    indices = [i for i in range(num_epochs) if ((i+1)%test_frequency == 0 or i ==0)]
    plt.plot(indices, train, label="train")
    plt.plot(indices, val, label="val")
    plt.title("mAP Plot")
    plt.ylabel("mAP")
    plt.xlabel("Epoch")
    plt.legend()
    plt.show()

```

```

def train(classifier, num_epochs, train_loader, val_loader, criterion, optimizer, te
    train_losses = []
    train_mAPs = []
    val_losses = []
    val_mAPs = []

    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):
```

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

Developing Your Own Model

Goal

To meet the benchmark for this assignment you will need to improve the network. Note you should have noticed pretrained Alenxt performs really well, but training Alexnet from scratch performs much worse. We hope you can design a better architecture over both the simple classifier and AlexNet to train from scratch.

How to start

You may take inspiration from other published architectures and architectures discussed in lecture. However, you are NOT allowed to use predefined models (e.g. models from torchvision) or use pretrained weights. Training must be done from scratch with your own custom model.

Some hints

There are a variety of different approaches you should try to improve performance from the simple classifier:

- Network architecture changes
  - Number of layers: try adding layers to make your network deeper
  - Batch normalization: adding batch norm between layers will likely give you a significant performance increase
  - Residual connections: as you increase the depth of your network, you will find that having residual connections like those in ResNet architectures will be helpful
- Optimizer: Instead of plain SGD, you may want to add a learning rate schedule, add momentum, or use one of the other optimizers you have learned about like Adam. Check the `torch.optim` package for other optimizers
- Data augmentation: You should use the `torchvision.transforms` module to try adding random resized crops and horizontal flips of the input data. Check `transforms.RandomResizedCrop` and `transforms.RandomHorizontalFlip` for this. Feel free to apply more transforms for data augmentation which can lead to better performance.
- Epochs: Once you have found a generally good hyperparameter setting try training for more epochs

What to submit

Submit your best model to Kaggle and save all plots for the writeup.

```
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])

train_transform = transforms.Compose([
            transforms.Resize(227),
            transforms.CenterCrop(227),
            transforms.ToTensor(),
            normalize
        ])

test_transform = transforms.Compose([
            transforms.Resize(227),
            transforms.CenterCrop(227),
            transforms.ToTensor(),
            normalize,
        ])

ds_train = VocDataset('VOCdevkit_2007/VOC2007/','train',train_transform)
ds_val = VocDataset('VOCdevkit_2007/VOC2007/','val',test_transform)
ds_test = VocDataset('VOCdevkit_2007/VOC2007test/','test', test_transform)

```

```
/content/voc_dataloader.py:137: VisibleDeprecationWarning: Creating an ndarray from
  np.array(box_indices),
```

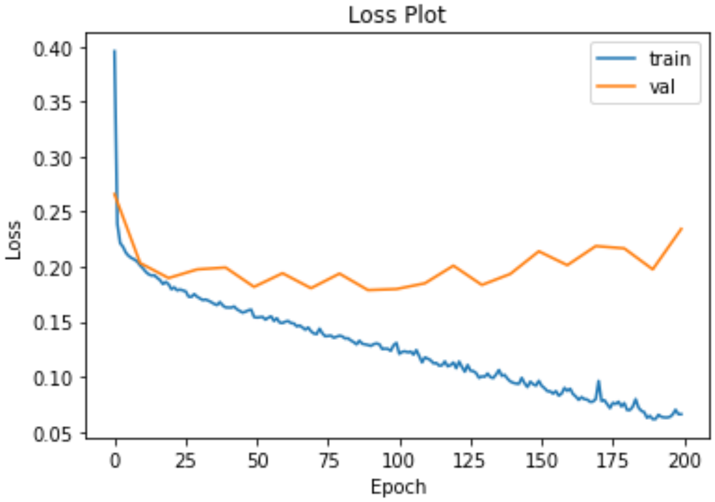
```
num_epochs = 200
test_frequency = 10
batch_size = 64

train_loader = torch.utils.data.DataLoader(dataset=ds_train,
                                               batch_size=batch_size,
                                               shuffle=True,
                                               num_workers=1)

val_loader = torch.utils.data.DataLoader(dataset=ds_val,
                                               batch_size=batch_size,
                                               shuffle=True,
                                               num workers=1)
```

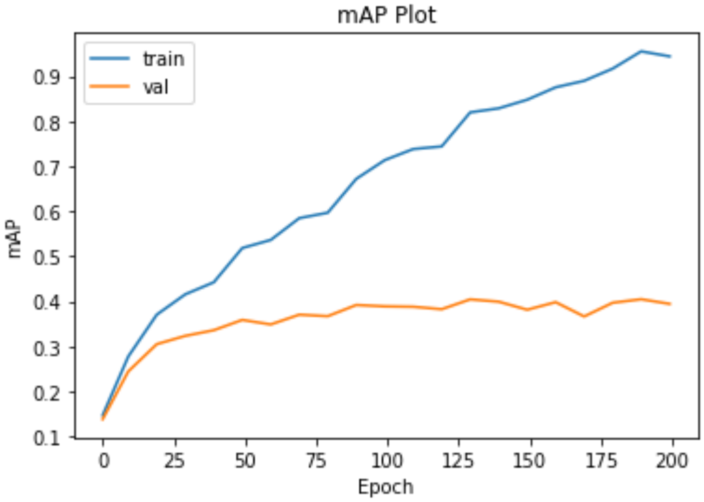
```
optimizer = torch.optim.Adam(classifier.parameters(), lr=4e-4)

classifier, train_losses, val_losses, train_mAPs, val_mAPs = train(classifier, num_e

```

```
Classifier(
  (conv1): Conv2d(3, 64, kernel_size=(5, 5), stride=(2, 2), padding=(3, 3))
  (conv1_bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
  (relu): ReLU()
  (pool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (b_layer1): Sequential(
    (0): block(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (conv1_bn): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (conv2_bn): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (relu): ReLU()
      (iden_downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1))
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_st
      )
    )
    (1): block(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (conv1_bn): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (conv2_bn): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_runni
      (relu): ReLU()
    )
  )
  (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2_bn): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc1): Linear(in_features=128, out_features=21, bias=True)
)
Starting epoch number 1
Loss for Training on Epoch 1 is 0.3958989977836609
-------  Class: aeroplane       AP:   0.2831  -------
-------  Class: bicycle         AP:   0.1067  -------
-------  Class: bird            AP:   0.1282  -------
-------  Class: boat            AP:   0.1298  -------
-------  Class: bottle          AP:   0.0804  -------
-------  Class: bus             AP:   0.0645  -------
-------  Class: car             AP:   0.2611  -------
-------  Class: cat             AP:   0.1412  -------
-------  Class: chair           AP:   0.1956  -------
```

```
Loss for Training on Epoch 2 is 0.2389794886112213
```

```
plot_losses(train_losses, val_losses, test_frequency, num_epochs)
plot_mAP(train_mAPs, val_mAPs, test_frequency, num_epochs)
```





```
mAP_test, test_loss, test_aps = test_classifier(test_loader, classifier, criterion)
print(mAP_test)
```

```
-------  Class: aeroplane        AP:   0.6446  -------
```

```
mAP: 0.3880
Avg loss: 0.23487901764038283
0.3879707478604367
```

```
torch.save(classifier.state_dict(), './voc_my_best_classifier.pth')
output_submission_csv('my_solution.csv', test_aps)
```