Notebook voc_dataloader.py

```
# Steps to load extra files and download dataset.
import os
!git clone "https://vipulrawat008:Invincible007!@github.com/the-visionary/assignment
os.chdir('/content')
os.chdir('assignment1')
os.chdir('assignment1-part3')
assert (os.getcwd() == '/content/assignment1/assignment1-part3')

!bash download_data.sh
assert (os.getcwd() == '/content/assignment1/assignment1-part3')
```

```
Cloning into 'assignment1'...
remote: Enumerating objects: 59, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (51/51), done.
remote: Total 59 (delta 20), reused 32 (delta 4), pack-reused 0
Unpacking objects: 100% (59/59), done.
--2021-03-06 03:57:23--  http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-200
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... connected.
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  9.41MB/s    in 47s

2021-03-06 03:58:11 (9.26 MB/s) - ‘VOCtrainval_06-Nov-2007.tar’ saved [460032000/460032000]

--2021-03-06 03:58:13--  http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest_06-Nov-2007.tar
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... connected.
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  9.16MB/s    in 47s

2021-03-06 03:59:00 (9.10 MB/s) - ‘VOCtest_06-Nov-2007.tar’ saved [451020800/451020800]
```

```
import os
os.chdir('/content')
os.chdir('assignment1')
os.chdir('assignment1-part3')
```

Assignment 1 Part 3B: Developing Your Own Classifier

```
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 classifier import SimpleClassifier, Classifier#, AlexNet
from voc_dataloader import VocDataset, VOC_CLASSES

%matplotlib inline
%load_ext autoreload
# %reload_ext autoreload
%autoreload 2
```

Part 3B: 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 performanc 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)
```

```
            y_score = np.concatenate((y_score, logits.cpu().numpy()), axis=0)
            loss = criterion(logits, labels)
            losses.append(loss.item())
        aps = []
        # ignore first class which is background
        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):
```

```
        if(epoch%test_frequency==0 or epoch==1):
            mAP_train, _, _ = test_classifier(train_loader, classifier, criterion, F
            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 "+
            val_losses.append(val_loss)
            val_mAPs.append(mAP_val)

    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 noti
really well, but training Alexnet from scratch performs much worse. We hope you can design a better archite
and AlexNet to train from scratch.

How to start

You may take inspiration from other published architectures and architectures discussed in lecture. Howeve
predefined models (e.g. models from torchvision) or use pretrained weights. Training must be done from scr

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 performa
    - Residual connections: as you increase the depth of your network, you will find that having residua
    architectures will be helpful
- Optimizer: Instead of plain SGD, you may want to add a learning rate schedule, add momentum, or use
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
input data. Check `transforms.RandomResizedCrop` and `transforms.RandomHorizontalFlip` for this. Fe
data augmentation which can lead to better performance.
- Epochs: Once you have found a generally good hyperparameter setting try training for more epochs
- Loss function: You might want to add weighting to the `MultiLabelSoftMarginLoss` for classes that are
experiment with a different loss function

Note

We will soon be providing some initial expectations of mAP values as a function of epoch so you can get an
implementation works without waiting a long time for training to converge.

What to submit

Submit your best model 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/assignment1/assignment1-part3/voc_dataloader.py:109: VisibleDeprecationWarning: Creati
  return np.array(names), np.array(labels).astype(np.float32), np.array(box_indices), label_or
```

```
num_epochs = 15
test_frequency = 5
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)

test_loader = torch.utils.data.DataLoader(dataset=ds_test,
                                           batch_size=batch_size,
                                           shuffle=False,
                                           num_workers=1)
```

```
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.nn.functional as F
from torch import optim
```

```
import numpy as np

NUM_CLASSES = 21

class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        """
        Args:
          in_channels (int):  Number of input channels.
          out_channels (int): Number of output channels.
          stride (int):       Controls the stride.
        """
        super(ResBlock, self).__init__()

        self.skip = nn.Sequential()

        if stride != 1 or in_channels != out_channels:
          self.skip = nn.Sequential(
            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_siz
            nn.BatchNorm2d(out_channels))
        else:
          self.skip = None

        self.block = nn.Sequential(
            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_siz
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_siz
            nn.BatchNorm2d(out_channels))

    def forward(self, x):
        identity = x
        out = self.block(x)

        if self.skip is not None:
            identity = self.skip(x)

        out += identity
        out = F.relu(out)

        return out

class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, stride=1)
        self.conv1b = nn.Conv2d(32, 32, 3, stride=2)
        # self.conv2a = nn.Conv2d(48, 32, 1)
        # self.conv2b = nn.Conv2d(32, 32, 5, stride=1, padding=2)
        # self.conv2c = nn.Conv2d(32, 256, 1)
        self.conv3a = nn.Conv2d(32, 64, 1)
        self.res = ResBlock(64, 64)
        self.conv3e = nn.Conv2d(64, 256, 1)
        self.pool = nn.MaxPool2d(3, 2)
```

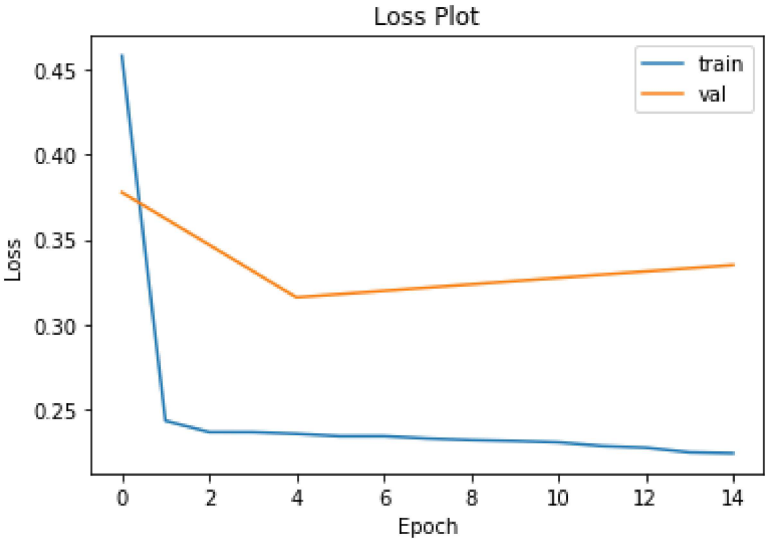
```
        self.fc1 = nn.Linear(2*2*256, 200)
        self.fc2 = nn.Linear(200, 100)
        self.fc3 = nn.Linear(100, NUM_CLASSES)

    def forward(self, x):
        #print(x.size())
        x = self.conv1(x)
        x = F.relu(x)
        #print(x.size())
        x = self.pool(x)
        #print(x.size())
        x = self.conv1b(x)
        x = F.relu(x)
        #print(x.size())
        x = self.conv1b(x)
        x = F.relu(x)
        #print(x.size())
        x = self.pool(x)
        #print(x.size())
        x = self.conv3a(x)
        x = F.relu(x)
        #print(x.size())
        x = self.res(x)
        x = self.res(x)
        x = self.res(x)
        x = F.relu(x)
        #print(x.size())
        x = self.pool(x)
        #print(x.size())
        x = self.conv3e(x)
        x = F.relu(x)
        #print(x.size())
        x = self.pool(x)
        #print(x.size())
        x = x.view(x.size()[0], 2*2*256)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

```

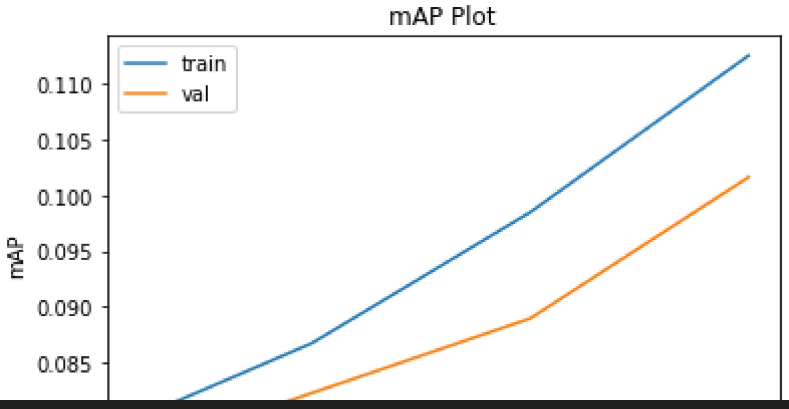
```
# TODO: Run your own classifier here
classifier = Classifier().to(device)

criterion = nn.MultiLabelSoftMarginLoss()
optimizer = torch.optim.SGD(classifier.parameters(), lr=0.005, momentum=0.9)

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

```

```
Loss for Training on Epoch 10 is 0.231260746717453
-------  Class: aeroplane       AP:   0.1011  -------
-------  Class: bicycle         AP:   0.0557  -------
-------  Class: bird            AP:   0.0556  -------
```

```
-------  Class: boat             AP:   0.0596  -------
-------  Class: bottle           AP:   0.0516  -------
-------  Class: bus              AP:   0.0402  -------
-------  Class: car              AP:   0.1610  -------
-------  Class: cat              AP:   0.0726  -------
-------  Class: chair            AP:   0.1156  -------
-------  Class: cow              AP:   0.0304  -------
-------  Class: diningtable      AP:   0.0539  -------
-------  Class: dog              AP:   0.0883  -------
-------  Class: horse            AP:   0.0684  -------
-------  Class: motorbike        AP:   0.0778  -------
-------  Class: person           AP:   0.4932  -------
-------  Class: pottedplant      AP:   0.0521  -------
-------  Class: sheep            AP:   0.0162  -------
-------  Class: sofa             AP:   0.0662  -------
-------  Class: train            AP:   0.0723  -------
-------  Class: tvmonitor        AP:   0.0472  -------
mAP: 0.0890
Avg loss: 0.32551265358924864
Evaluating classifier
Mean Precision Score for Testing on Epoch 10 is 0.08895318486778567
Starting epoch number 11
Loss for Training on Epoch 11 is 0.23047009110450745
Starting epoch number 12
Loss for Training on Epoch 12 is 0.2282872200012207
Starting epoch number 13
Loss for Training on Epoch 13 is 0.22733020782470703
Starting epoch number 14
Loss for Training on Epoch 14 is 0.22457727789878845
Starting epoch number 15
Loss for Training on Epoch 15 is 0.2240038812160492
-------  Class: aeroplane        AP:   0.1863  -------
-------  Class: bicycle          AP:   0.0631  -------
-------  Class: bird             AP:   0.0555  -------
-------  Class: boat             AP:   0.1307  -------
-------  Class: bottle           AP:   0.0493  -------
-------  Class: bus              AP:   0.0588  -------
-------  Class: car              AP:   0.2048  -------
-------  Class: cat              AP:   0.0671  -------
-------  Class: chair            AP:   0.1170  -------
-------  Class: cow              AP:   0.0281  -------
-------  Class: diningtable      AP:   0.0494  -------
-------  Class: dog              AP:   0.0760  -------
-------  Class: horse            AP:   0.0763  -------
-------  Class: motorbike        AP:   0.0940  -------
-------  Class: person           AP:   0.5150  -------
-------  Class: pottedplant      AP:   0.0475  -------
-------  Class: sheep            AP:   0.0143  -------
-------  Class: sofa             AP:   0.0600  -------
-------  Class: train            AP:   0.0920  -------
-------  Class: tvmonitor        AP:   0.0469  -------
mAP: 0.1016
Avg loss: 0.3349403478205204
Evaluating classifier
Mean Precision Score for Testing on Epoch 15 is 0.10161171225786791
```

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






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

```
-------  Class: aeroplane        AP:   0.2006  -------
-------  Class: bicycle          AP:   0.0592  -------
-------  Class: bird             AP:   0.0556  -------
-------  Class: boat             AP:   0.1014  -------
-------  Class: bottle           AP:   0.0618  -------
-------  Class: bus              AP:   0.0513  -------
-------  Class: car              AP:   0.2233  -------
-------  Class: cat              AP:   0.0587  -------
-------  Class: chair            AP:   0.1146  -------
-------  Class: cow              AP:   0.0237  -------
-------  Class: diningtable      AP:   0.0592  -------
-------  Class: dog              AP:   0.0713  -------
-------  Class: horse            AP:   0.0682  -------
-------  Class: motorbike        AP:   0.0846  -------
-------  Class: person           AP:   0.5183  -------
-------  Class: pottedplant      AP:   0.0542  -------
-------  Class: sheep            AP:   0.0133  -------
-------  Class: sofa             AP:   0.0593  -------
-------  Class: train            AP:   0.0883  -------
-------  Class: tvmonitor        AP:   0.0608  -------
mAP: 0.1014
Avg loss: 0.33337560563515395
0.10139077554847706
```

```
1 torch.save(classifier.state_dict(), './voc_my_best_classifier.pth')
2
3 # output_submission_csv('my_solution.csv', test_aps)
```

```
1
```