Assignment 1 Part 3B: Developing Your Own Classifier

In [1]:

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

```
Drive already mounted at /content/drive; to attempt to forcibly remoun
t, call drive.mount("/content/drive", force_remount=True).
```

In [3]:

```
!pip3 install torch==1.5 torchvision==0.6
```

```
Collecting torch==1.5
  Downloading https://files.pythonhosted.org/packages/76/58/668ffb25215
b3f8231a550a227be7f905f514859c70a65ca59d28f9b7f60/torch-1.5.0-cp37-cp37
m-manylinux1_x86_64.whl (752.0MB)
     |████████████████████████████████| 752.0MB 24kB/s
Collecting torchvision==0.6
  Downloading https://files.pythonhosted.org/packages/7b/ed/a894f274a77
33d6492e438a5831a95b507c5ec777edf6d8c3b97574e08c4/torchvision-0.6.0-cp3
7-cp37m-manylinux1_x86_64.whl (6.6MB)
     |████████████████████████████████| 6.6MB 45.0MB/s
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-
packages (from torch==1.5) (0.16.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-p
ackages (from torch==1.5) (1.19.5)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.
7/dist-packages (from torchvision==0.6) (7.0.0)
Installing collected packages: torch, torchvision
  Found existing installation: torch 1.7.1+cu101
    Uninstalling torch-1.7.1+cu101:
      Successfully uninstalled torch-1.7.1+cu101
  Found existing installation: torchvision 0.8.2+cu101
    Uninstalling torchvision-0.8.2+cu101:
      Successfully uninstalled torchvision-0.8.2+cu101
Successfully installed torch-1.5.0 torchvision-0.6.0
```

In [2]:

```
%cd '/content/drive/MyDrive/DL/assignment1-part3/assignment1-part3'
```

```
/content/drive/MyDrive/DL/assignment1-part3/assignment1-part3
```

In [3]:

```
!ls
```

```
A1_P3A_Introduction.ipynb        download_data.sh   VOCdevkit
A1_P3B_Develop_Classifier.ipynb  __pycache__        voc_simple_classifi
er.pth
classifier.py                    voc_dataloader.py  VOCtrainval_06-Nov-
2007.tar
```

In [4]:

```
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
%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 performances. Hints will be given in the below instructions.

What to submit

Check the submission template for details what to submit.

In [5]:

```
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()
```

In [6]:

```
def test_classifier(test_loader, classifier, criterion, print_ind_class
es=True, print_total=True):
    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()), axi
s=0)
            y_score = np.concatenate((y_score, logits.cpu().numpy()), a
xis=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_CLASSES[i], ap))
            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
```

In [7]:

```
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()
```

In [8]:
```
def train(classifier, num_epochs, train_loader, val_loader, criterion,
optimizer, test_frequency=5):
    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, criteri
on, optimizer)
        train_losses.append(train_loss)
        print("Loss for Training on Epoch " +str(epoch) + " is "+ str(t
rain_loss))
        if(epoch%test_frequency==0 or epoch==1):
            mAP_train, _, _ = test_classifier(train_loader, classifier,
criterion, False, False)
            train_mAPs.append(mAP_train)
            mAP_val, val_loss, _ = test_classifier(val_loader, classifi
er, criterion)
            print('Evaluating classifier')
            print("Mean Precision Score for Testing on Epoch " +str(epo
ch) + " is "+ str(mAP_val))
            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 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
- Loss function: You might want to add weighting to the `MultiLabelSoftMarginLoss` for classes that are less well represented or 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 early idea whether your 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.

In [10]:

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

```
/content/drive/My Drive/DL/assignment1-part3/assignment1-part3/voc_data
loader.py:109: VisibleDeprecationWarning: Creating an ndarray from ragg
ed nested sequences (which is a list-or-tuple of lists-or-tuples-or nda
rrays with different lengths or shapes) is deprecated. If you meant to
do this, you must specify 'dtype=object' when creating the ndarray
  return np.array(names), np.array(labels).astype(np.float32), np.array
```

```
    return np.array(names), np.array(labels).astype(np.float32), np.array
(box_indices), label_order
```

In [11]:

```
num_epochs = 100
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)
```

In [55]:

```
c = SimpleClassifier()
c.parameters()
```

```
[autoreload of classifier failed: Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/IPython/extensions/autor
eload.py", line 247, in check
    superreload(m, reload, self.old_objects)
ValueError: __init__() requires a code object with 1 free vars, not 0
]
```

Out[55]:

```
<generator object Module.parameters at 0x7f95e673e950>
```

In [56]:

```
cl = Classifier()
cl.parameters()
```

Out[56]:

```
<generator object Module.parameters at 0x7f95e673e4d0>
```

In [60]:

```
# TODO: Run your own classifier here
c = Classifier()

criterion = nn.MultiLabelSoftMarginLoss()

# optimizer = torch.optim.SGD(classifier.parameters(), lr=0.01, momentu
m=0.9)
optimizer = torch.optim.Adam(c.parameters(), lr=1e-4)

classifier, train_losses, val_losses, train_mAPs, val_mAPs = train(c, n
um_epochs, train_loader, val_loader, criterion, optimizer, test_frequen
cy)
```

```
-----------------------------------------------------------------------
----
ValueError                                Traceback (most recent call l
ast)
<ipython-input-60-7b8d18f8e118> in <module>()
      5
      6 # optimizer = torch.optim.SGD(classifier.parameters(), lr=0.01,
momentum=0.9)
----> 7 optimizer = torch.optim.Adam(c.parameters(), lr=1e-4)
      8
      9 classifier, train_losses, val_losses, train_mAPs, val_mAPs = tr
ain(c, num_epochs, train_loader, val_loader, criterion, optimizer, test
_frequency)

/usr/local/lib/python3.7/dist-packages/torch/optim/adam.py in __init__
(self, params, lr, betas, eps, weight_decay, amsgrad)
     42         defaults = dict(lr=lr, betas=betas, eps=eps,
     43                         weight_decay=weight_decay, amsgrad=amsg
rad)
---> 44         super(Adam, self).__init__(params, defaults)
     45
     46     def __setstate__(self, state):

/usr/local/lib/python3.7/dist-packages/torch/optim/optimizer.py in __in
it__(self, params, defaults)
     44         param_groups = list(params)
     45         if len(param_groups) == 0:
```

```
---> 46                 raise ValueError("optimizer got an empty parameter
 list")
     47             if not isinstance(param_groups[0], dict):
     48                 param_groups = [{'params': param_groups}]

ValueError: optimizer got an empty parameter list
```

In [ ]:

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

In [ ]:

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

In [ ]:

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

In [ ]: