Assignment 3 Part 1: Developing Your Own Classifier

```
from google.colab import drive
drive.mount('/content/gdrive/',force_remount=True)

import sys
sys.path.append('/content/gdrive/My Drive/assignment3_p1_starterkit')
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_1 import SimpleClassifier, Classifier#, AlexNet
from voc_dataloader import VocDataset, VOC_CLASSES

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

```
Mounted at /content/gdrive/
```

```
!pip install torch==1.5.1+cu101 torchvision==0.6.1+cu101 -f https://download.pytorch.
```

```
Looking in links: https://download.pytorch.org/whl/torch_stable.html
Collecting torch==1.5.1+cu101
  Downloading https://download.pytorch.org/whl/cu101/torch-1.5.1%2Bcu101-cp36-cp
     |████████████████████████████████| 704.4MB 25kB/s
Collecting torchvision==0.6.1+cu101
  Downloading https://download.pytorch.org/whl/cu101/torchvision-0.6.1%2Bcu101-c
     |████████████████████████████████| 6.6MB 111kB/s
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.6/dist-pa
Installing collected packages: torch, torchvision
  Found existing installation: torch 1.6.0+cu101
    Uninstalling torch-1.6.0+cu101:
      Successfully uninstalled torch-1.6.0+cu101
  Found existing installation: torchvision 0.7.0+cu101
    Uninstalling torchvision-0.7.0+cu101:
      Successfully uninstalled torchvision-0.7.0+cu101
Successfully installed torch-1.5.1+cu101 torchvision-0.6.1+cu101
```

```
import os
```

```
import os
os.chdir('/content/gdrive/My Drive/assignment3_p1_starterkit')
```

```
import shutil
shutil.copyfile("VOCtrainval_06-Nov-2007.tar.1", "/content/VOCtrainval_06-Nov-2007.tar
!tar -xf "/content/VOCtrainval_06-Nov-2007.tar.1" -C "/content/"
shutil.move("/content/VOCdevkit/", "/content/VOCdevkit_2007")
```

```
'/content/VOCdevkit_2007'
```

Part 1B: 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, print_
    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_CI
            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, test
    train_losses = []
    train_mAPs = []
    val_losses = []
    val_mAPs = []

    for epoch in range(1,num_epochs+1):
```

```
    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, Fal
            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 "+ s
            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 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.RandomResizedCrop(227),
            transforms.CenterCrop(227),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            normalize
        ])

test_transform = transforms.Compose([
            transforms.Resize(227),
            #transforms.RandomResizedCrop(227),
            transforms.CenterCrop(227),
            #transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            normalize,
        ])
```

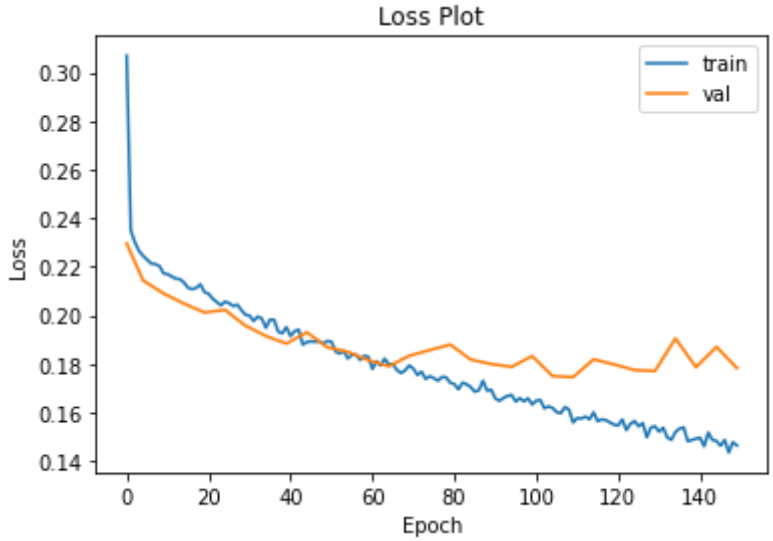
```
ds_train = VocDataset('/content/VOCdevkit_2007/VOC2007/','train',train_transform)
```

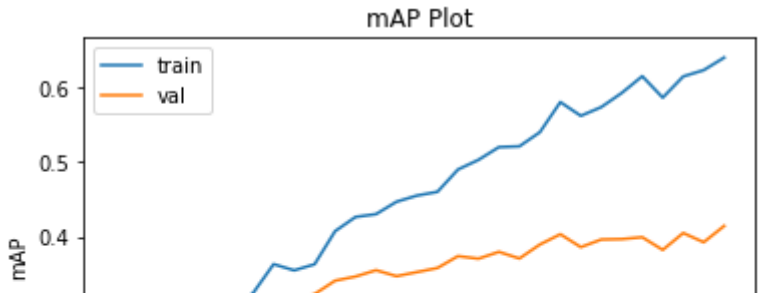
```
ds_val = VocDataset('/content/VOCdevkit_2007/VOC2007/','val',test_transform)
ds_test = VocDataset('/content/gdrive/My Drive/assignment3_p1_starterkit/VOCdevkit_20(



num_epochs = 150
test_frequency = 5
batch_size = 32

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

val_loader = torch.utils.data.DataLoader(dataset=ds_val,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=1,drop_last=True)

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


# TODO: Run your own classifier here

classifier = Classifier().to(device)

criterion = nn.MultiLabelSoftMarginLoss()
#optimizer = torch.optim.SGD(classifier.parameters(), lr=0.01, momentum=0.98)
#optimizer = torch.optim.Adam(classifier.parameters(), lr=1e-4)
optimizer = torch.optim.SGD(classifier.parameters(), lr=0.1, momentum=0.9)
torch.optim.lr_scheduler.StepLR(optimizer, 4, gamma=0.001, last_epoch=-1)

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

```
mAP: 0.4048
Avg loss: 0.17869987988319153
Evaluating classifier
Mean Precision Score for Testing on Epoch 140 is 0.4048474727796231
Starting epoch number 141
Loss for Training on Epoch 141 is 0.1496322900056839
Starting epoch number 142
Loss for Training on Epoch 142 is 0.14615283906459808
Starting epoch number 143
Loss for Training on Epoch 143 is 0.15177685022354126
Starting epoch number 144
Loss for Training on Epoch 144 is 0.14875881373882294

Starting epoch number 145
Loss for Training on Epoch 145 is 0.14830011129379272
-------  Class: aeroplane        AP:   0.5540  -------
-------  Class: bicycle          AP:   0.4485  -------
```

```
-------  Class: bird             AP:   0.3263  -------
-------  Class: boat             AP:   0.4344  -------
-------  Class: bottle           AP:   0.1377  -------
-------  Class: bus              AP:   0.3090  -------
-------  Class: car              AP:   0.6141  -------
-------  Class: cat              AP:   0.3140  -------
-------  Class: chair            AP:   0.4652  -------
-------  Class: cow              AP:   0.1800  -------
-------  Class: diningtable      AP:   0.3499  -------
-------  Class: dog              AP:   0.2944  -------
-------  Class: horse            AP:   0.5940  -------
-------  Class: motorbike        AP:   0.4895  -------
-------  Class: person           AP:   0.7725  -------
-------  Class: pottedplant      AP:   0.1949  -------
-------  Class: sheep            AP:   0.1644  -------
-------  Class: sofa             AP:   0.3142  -------
-------  Class: train            AP:   0.5654  -------
-------  Class: tvmonitor        AP:   0.3288  -------
mAP: 0.3925
Avg loss: 0.18705657258247718
Evaluating classifier
Mean Precision Score for Testing on Epoch 145 is 0.3925477928112914
Starting epoch number 146
Loss for Training on Epoch 146 is 0.1463354527950287
Starting epoch number 147
Loss for Training on Epoch 147 is 0.14870448410511017
Starting epoch number 148
Loss for Training on Epoch 148 is 0.14357000589370728
Starting epoch number 149
Loss for Training on Epoch 149 is 0.14779342710971832
Starting epoch number 150
Loss for Training on Epoch 150 is 0.14645610749721527
-------  Class: aeroplane        AP:   0.6103  -------
-------  Class: bicycle          AP:   0.4474  -------
-------  Class: bird             AP:   0.3492  -------
-------  Class: boat             AP:   0.4299  -------
-------  Class: bottle           AP:   0.1596  -------
-------  Class: bus              AP:   0.3542  -------
-------  Class: car              AP:   0.6007  -------
-------  Class: cat              AP:   0.3593  -------
-------  Class: chair            AP:   0.4732  -------
-------  Class: cow              AP:   0.2129  -------
-------  Class: diningtable      AP:   0.3529  -------
-------  Class: dog              AP:   0.2870  -------
```

```
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:   0.5853  -------
    -------  Class: bicycle          AP:   0.3948  -------
    -------  Class: bird             AP:   0.3162  -------
    -------  Class: boat             AP:   0.3650  -------
    -------  Class: bottle           AP:   0.1579  -------
    -------  Class: bus              AP:   0.3160  -------
    -------  Class: car              AP:   0.6146  -------
    -------  Class: cat              AP:   0.3580  -------
    -------  Class: chair            AP:   0.4145  -------
    -------  Class: cow              AP:   0.1921  -------
    -------  Class: diningtable      AP:   0.2587  -------
    -------  Class: dog              AP:   0.2990  -------
    -------  Class: horse            AP:   0.6802  -------
    -------  Class: motorbike        AP:   0.5494  -------
    -------  Class: person           AP:   0.8033  -------
    -------  Class: pottedplant      AP:   0.2133  -------
    -------  Class: sheep            AP:   0.3216  -------
    -------  Class: sofa             AP:   0.3264  -------
    -------  Class: train            AP:   0.5123  -------
    -------  Class: tvmonitor        AP:   0.3114  -------
    mAP: 0.3995
    Avg loss: 0.1766747386231051
    0.3995088353656625
```

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

```
device
```

```
device(type='cuda', index=1)
```