Notebook

Assignment 1 Part 3A Introduction: Multi-label Image Classificatio

```
# 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')
```

```
--2021-02-26 14:36:46--  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.1’

.tar.1              16%[==>                 ]  70.20M  11.6MB/s    eta 39s    ^C
```

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

```
!pip install torch==1.5.1 torchvision==0.6.1
```

```
Collecting torch==1.5.1
  Using cached https://files.pythonhosted.org/packages/a4/cf/007b6de316c9f3d4cb315a60c308342cc
Collecting torchvision==0.6.1
  Downloading https://files.pythonhosted.org/packages/f4/ef/c32275c040f12f24adcd6302f0f3cd12cc
     |████████████████████████████████| 6.6MB 3.9MB/s
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages (from torch==1
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torch==1.
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.7/dist-packages (from t
Installing collected packages: torch, torchvision
  Found existing installation: torchvision 0.8.1+cu101
    Uninstalling torchvision-0.8.1+cu101:
      Successfully uninstalled torchvision-0.8.1+cu101
Successfully installed torch-1.5.1 torchvision-0.6.1
```

```
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
```

```
# from kaggle_submission import output_submission_csv
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
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Multi-label Classification

In this assignment, you train a classifier to do multi-label classificaton on the PASCAL VOC 2007 dataset. Th
which can appear in any given image. Your classifier will predict whether each class appears in an image. Th
exclusive multiclass classification like the ImageNet competition where only a single most appropriate class

Part 3A

You will use this notebook to warm up with pytorch.

What to do

In part 3A, You are asked to run below experiments. You don't need to change hyperparameters for this Part
code provides everything that you will need.)

1. to train a simple network (defined in `classifiers.py`)
2. to train the AlexNet (PyTorch built-in)
    - from scratch
    - finetuning AlexNet pretrained on ImageNet

What to submit

We ask you to run the following code and report the results in your homework submission. You may want to
familiar with PyTorch.

You will the need the numbers and plots this notebook outputs for reports, but you are not required to submi

Reading Pascal Data

Loading Training Data

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

```
# Transforms applied to the training data
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std= [0.229, 0.224, 0.225])
```

```
4
5 train_transform = transforms.Compose([
6             transforms.Resize(227),
7             transforms.CenterCrop(227),
8             transforms.ToTensor(),
9             normalize
10        ])
11
```

```
1 ds_train = VocDataset('VOCdevkit_2007/VOC2007/','train',train_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_orc
```

Loading Validation Data

We will load the test data for the PASCAL VOC 2007 dataset. Do **NOT** add data augmentation transforms to

```
1 # Transforms applied to the testing data
2 test_transform = transforms.Compose([
3             transforms.Resize(227),
4             transforms.CenterCrop(227),
5             transforms.ToTensor(),
6             normalize,
7         ])
```

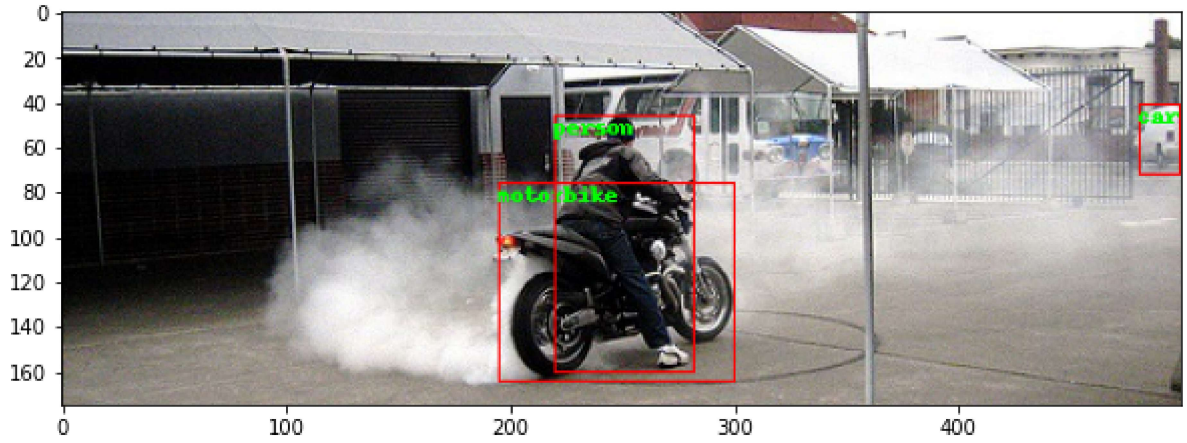
```
1 ds_val = VocDataset('VOCdevkit_2007/VOC2007/','val',test_transform)
```

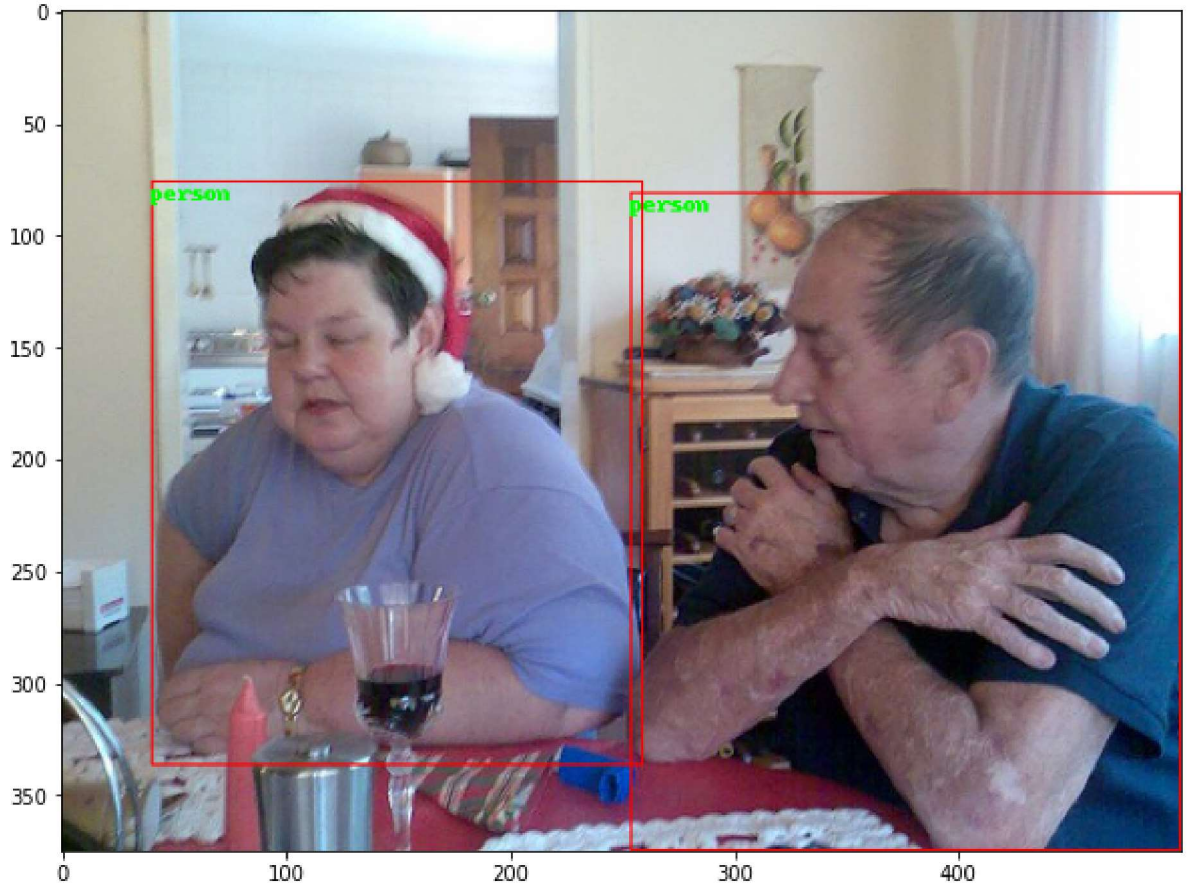
```
/content/assignment1/assignment1-part3/voc_dataloader.py:109: VisibleDeprecationWarning: Creati
  return np.array(names), np.array(labels).astype(np.float32), np.array(box_indices), label_orc
```

Visualizing the Data

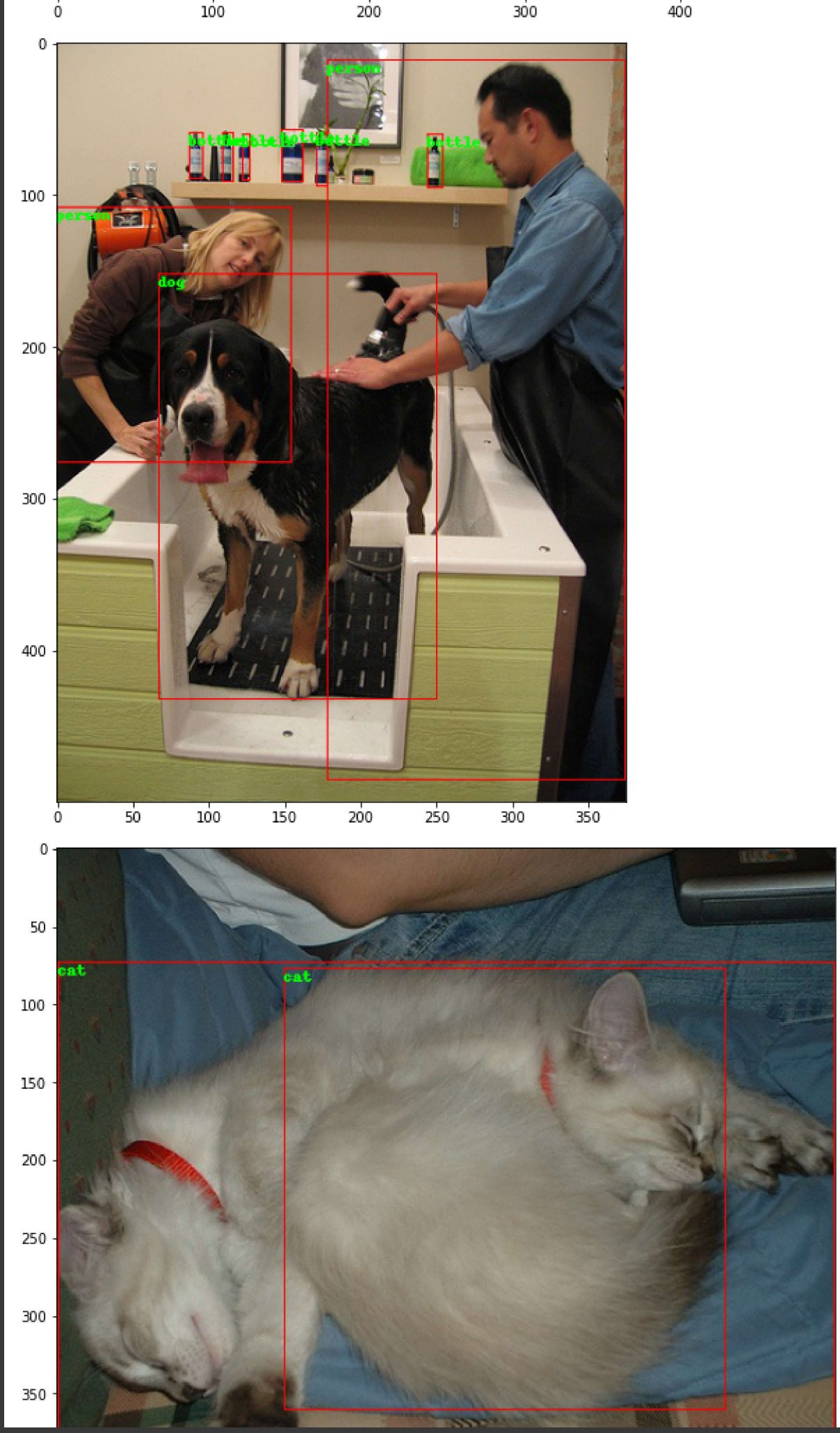
PASCAL VOC has bounding box annotations in addition to class labels. Use the following code to visualize s
corresponding annotations from the train set.

```
1 for i in range(5):
2     idx = np.random.randint(0, len(ds_train.names)+1)
3     _imgpath = os.path.join('VOCdevkit_2007/VOC2007/', 'JPEGImages', ds_train.names[
4     img = Image.open(_imgpath).convert('RGB')
5     draw = ImageDraw.Draw(img)
6     for j in range(len(ds_train.box_indices[idx])):
7         obj = ds_train.box_indices[idx][j]
8         draw.rectangle(list(obj), outline=(255,0,0))
9         draw.text(list(obj[0:2]), ds_train.classes[ds_train.label_order[idx][j]], fi
10    plt.figure(figsize = (10,10))
11    plt.imshow(np.array(img))
```


person
motorbike
car


person
person


motorbike



Classification

```
# def collate_fn(batch):
#     return tuple(zip(*batch))

```

```
# declare what device to use: gpu/cpu
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
train_loader = torch.utils.data.DataLoader(dataset=ds_train,
                                           batch_size=50,
                                           shuffle=True,
                                        #    collate_fn=lambda x: x,
                                           num_workers=1)
```

```
val_loader = torch.utils.data.DataLoader(dataset=ds_val,
                                           batch_size=50,
                                           shuffle=True,
                                        #    collate_fn=lambda x: x,
                                           num_workers=1)
```

```
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
```

```
# plot functions
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()

```

Training the network

The simple network you are given as is will allow you to reach around 0.15-0.2 mAP. In this project, you will f
Save plots and final test mAP scores as you will be adding these to the writeup.

```
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):
            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
```

```
classifier = SimpleClassifier().to(device)
# You can can use this function to reload a network you have already saved previousl
#classifier.load_state_dict(torch.load('voc_classifier.pth'))
```

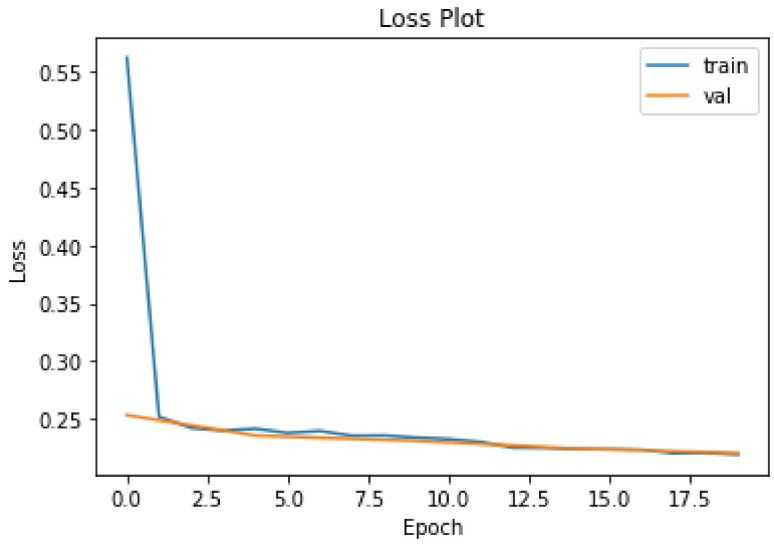
```
criterion = nn.MultiLabelSoftMarginLoss()
optimizer = torch.optim.SGD(classifier.parameters(), lr=0.01, momentum=0.9)
```

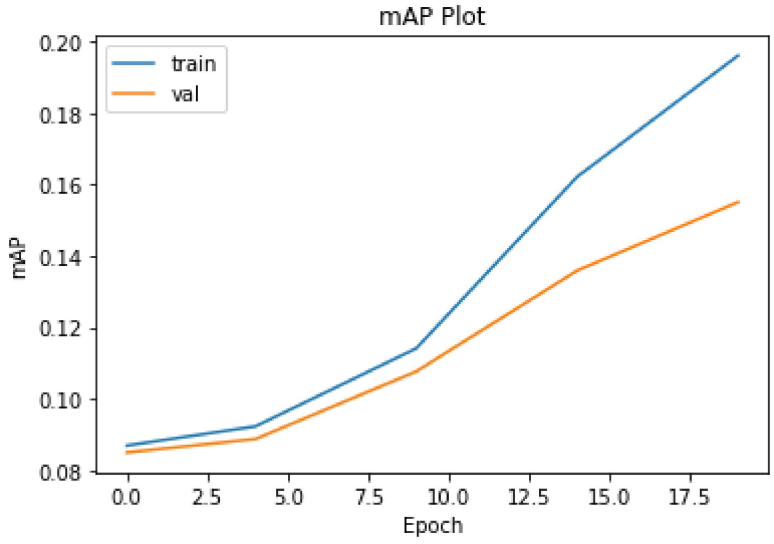
```
# Training the Classifier
num_epochs = 20
test_frequency = 5

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

```

```
Loss for Training on Epoch 15 is 0.22407136857509613
-------  Class: aeroplane        AP:   0.2722  -------
-------  Class: bicycle          AP:   0.0608  -------
-------  Class: bird             AP:   0.1066  -------
-------  Class: boat             AP:   0.2111  -------
-------  Class: bottle           AP:   0.0804  -------
-------  Class: bus              AP:   0.0600  -------
-------  Class: car              AP:   0.1826  -------
-------  Class: cat              AP:   0.1271  -------
-------  Class: chair            AP:   0.2060  -------
-------  Class: cow              AP:   0.0485  -------
-------  Class: diningtable      AP:   0.1284  -------
-------  Class: dog              AP:   0.1444  -------
-------  Class: horse            AP:   0.0782  -------
-------  Class: motorbike        AP:   0.0528  -------
-------  Class: person           AP:   0.5101  -------
-------  Class: pottedplant      AP:   0.0605  -------
-------  Class: sheep            AP:   0.0459  -------
-------  Class: sofa             AP:   0.1509  -------
-------  Class: train            AP:   0.1132  -------
-------  Class: tvmonitor        AP:   0.0776  -------
mAP: 0.1359
Avg loss: 0.22459923753551408
Evaluating classifier
Mean Precision Score for Testing on Epoch 15 is 0.13586376672585931
Starting epoch number 16
Loss for Training on Epoch 16 is 0.224028080701828
Starting epoch number 17
Loss for Training on Epoch 17 is 0.22357989847660065
Starting epoch number 18
Loss for Training on Epoch 18 is 0.22061200439929962
Starting epoch number 19
Loss for Training on Epoch 19 is 0.2209300994873047
Starting epoch number 20
Loss for Training on Epoch 20 is 0.21946430206298828
-------  Class: aeroplane        AP:   0.3129  -------
```

```
-------  Class: bicycle          AP:   0.0734  -------
-------  Class: bird             AP:   0.1073  -------
-------  Class: boat             AP:   0.2140  -------
-------  Class: bottle           AP:   0.0912  -------
-------  Class: bus              AP:   0.0709  -------
-------  Class: car              AP:   0.2269  -------
-------  Class: cat              AP:   0.1345  -------
-------  Class: chair            AP:   0.2370  -------
-------  Class: cow              AP:   0.0630  -------
-------  Class: diningtable      AP:   0.1707  -------
-------  Class: dog              AP:   0.1389  -------
-------  Class: horse            AP:   0.1153  -------
-------  Class: motorbike        AP:   0.0736  -------
-------  Class: person           AP:   0.5127  -------
-------  Class: pottedplant      AP:   0.0728  -------
-------  Class: sheep            AP:   0.0414  -------
-------  Class: sofa             AP:   0.1775  -------
-------  Class: train            AP:   0.1517  -------
-------  Class: tvmonitor        AP:   0.1150  -------
mAP: 0.1550
Avg loss: 0.22070468407051236
Evaluating classifier
Mean Precision Score for Testing on Epoch 20 is 0.155030745622131
```

```
# Compare train and validation metrics
plot_losses(train_losses, val_losses, test_frequency, num_epochs)
plot_mAP(train_mAPs, val_mAPs, test_frequency, num_epochs)
```






```
# Save the clssifier network
# Suggestion: you can save checkpoints of your network during training and reload th
```

```
# Suggestion: You can save checkpoints of your network during training and reload th
torch.save(classifier.state_dict(), './voc_simple_classifier.pth')
```

Evaluate on test set

```
ds_test = VocDataset('VOCdevkit_2007/VOC2007test/','test', test_transform)

test_loader = torch.utils.data.DataLoader(dataset=ds_test,
                                          batch_size=50,
                                          shuffle=False,
                                          num_workers=1)

# Transforms applied to the testing data
test_transform = transforms.Compose([
            transforms.Resize(227),
            transforms.CenterCrop(227),
            transforms.ToTensor(),
            normalize,
        ])

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

```
/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_orc
-------  Class: aeroplane       AP:   0.3025  -------
-------  Class: bicycle         AP:   0.0615  -------
-------  Class: bird            AP:   0.1073  -------
-------  Class: boat            AP:   0.1665  -------
-------  Class: bottle          AP:   0.0851  -------
-------  Class: bus             AP:   0.0614  -------
-------  Class: car             AP:   0.2676  -------
-------  Class: cat             AP:   0.1126  -------
-------  Class: chair           AP:   0.2408  -------
-------  Class: cow             AP:   0.0645  -------
-------  Class: diningtable     AP:   0.1223  -------
-------  Class: dog             AP:   0.1564  -------
-------  Class: horse           AP:   0.1079  -------
-------  Class: motorbike       AP:   0.0791  -------
-------  Class: person          AP:   0.5287  -------
-------  Class: pottedplant     AP:   0.0658  -------
-------  Class: sheep           AP:   0.0713  -------
-------  Class: sofa            AP:   0.1770  -------
-------  Class: train           AP:   0.1716  -------
-------  Class: tvmonitor       AP:   0.0955  -------
mAP: 0.1523
Avg loss: 0.2167697401344776
```

```
output_submission_csv('my_solution.csv', test_aps)
```

AlexNet Baselines (From Scratch)

AlexNet was one of the earliest deep learning models to have success in classification. In this section we wi
AlexNet as a baseline. Furthermore, we will run an ImageNet-pretrained AlexNet to observe the impact of we

final test mAP scores as you will be adding these to the writeup.

Running AlexNet

In this section, we train AlexNet from scratch using the same hyperparameters as our previous experiment.

```
num_epochs = 20
test_frequency = 5

# Change classifier to AlexNet
classifier = torchvision.models.alexnet(pretrained=False)
classifier.classifier._modules['6'] = nn.Linear(4096, 21)
classifier = classifier.to(device)

criterion = nn.MultiLabelSoftMarginLoss()

optimizer = torch.optim.SGD(classifier.parameters(), lr=0.01, momentum=0.9)
```

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

```
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("Test mAP: ", mAP_test)
```

You should notice somewhat poor performance. You could try running AlexNet with an Adam optimizer inste
that makes a difference. This experiment is not required for the writeup, but it may show you the importance
optimizer.

Pretrained AlexNet

Here we look at the impact of pretrained features. This model's weights were trained on ImageNet, which is a
pretrained features perform on VOC? Why do you think there is such a large difference in performance?

```
num_epochs = 20
test_frequency = 5

# Load Pretrained AlexNet
classifier = torchvision.models.alexnet(pretrained=True)
classifier.classifier._modules['6'] = nn.Linear(4096, 21)
classifier = classifier.to(device)
optimizer = torch.optim.SGD(classifier.parameters(), lr=0.01, momentum=0.9)
```

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