Assignment 2 Part 1 Introduction: Multi-label Image Classification

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

```
Mounted at /content/drive
```

```
# !mkdir data1
# !wget http://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar -P data/
# # !wget https://storage.googleapis.com/coco-dataset/external/PASCAL_VOC.zip -P data/
# !tar -xf data/VOCtrainval_06-Nov-2007.tar -C data/
# !unzip data/PASCAL_VOC.zip -d data/
# !rm -rf data/PASCAL_VOC.zip data/VOCtrainval_06-Nov-2007.tar
!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
```

```
mkdir: cannot create directory ‘data1’: File exists
```

```
!rmdir data1
```

```
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
```

Multi-label Classificati

dataset. The dataset has 20 different class which can appear in any given image. Your classifier will predict whether each class appears in an image. This task is slightly different from exclusive multiclass classification like the ImageNet competition where only a single most appropriate class is predicted for an image.

Part 1

You will use this notebook to warm up with pytorch and the code+dataset that we will use for assignment3.

What to do

In part 1, You are asked to run below experiments. You don't need to change hyperparameters for this Part 1's experiments. (the following 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 leverage this part 1 get yourself familiar with PyTorch.

You will the need the numbers and plots this notebook outputs for reports, but you are not required to submit this notebook as a printed pdf.

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])

train_transform = transforms.Compose([
            transforms.Resize(227),
            transforms.CenterCrop(227),
```

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

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

Loading Validation Data

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

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

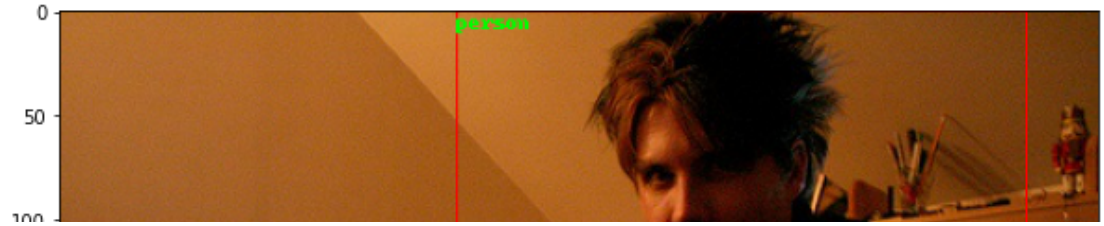
```
ds_val = VocDataset('VOCdevkit_2007/VOC2007/','val',test_transform)
```

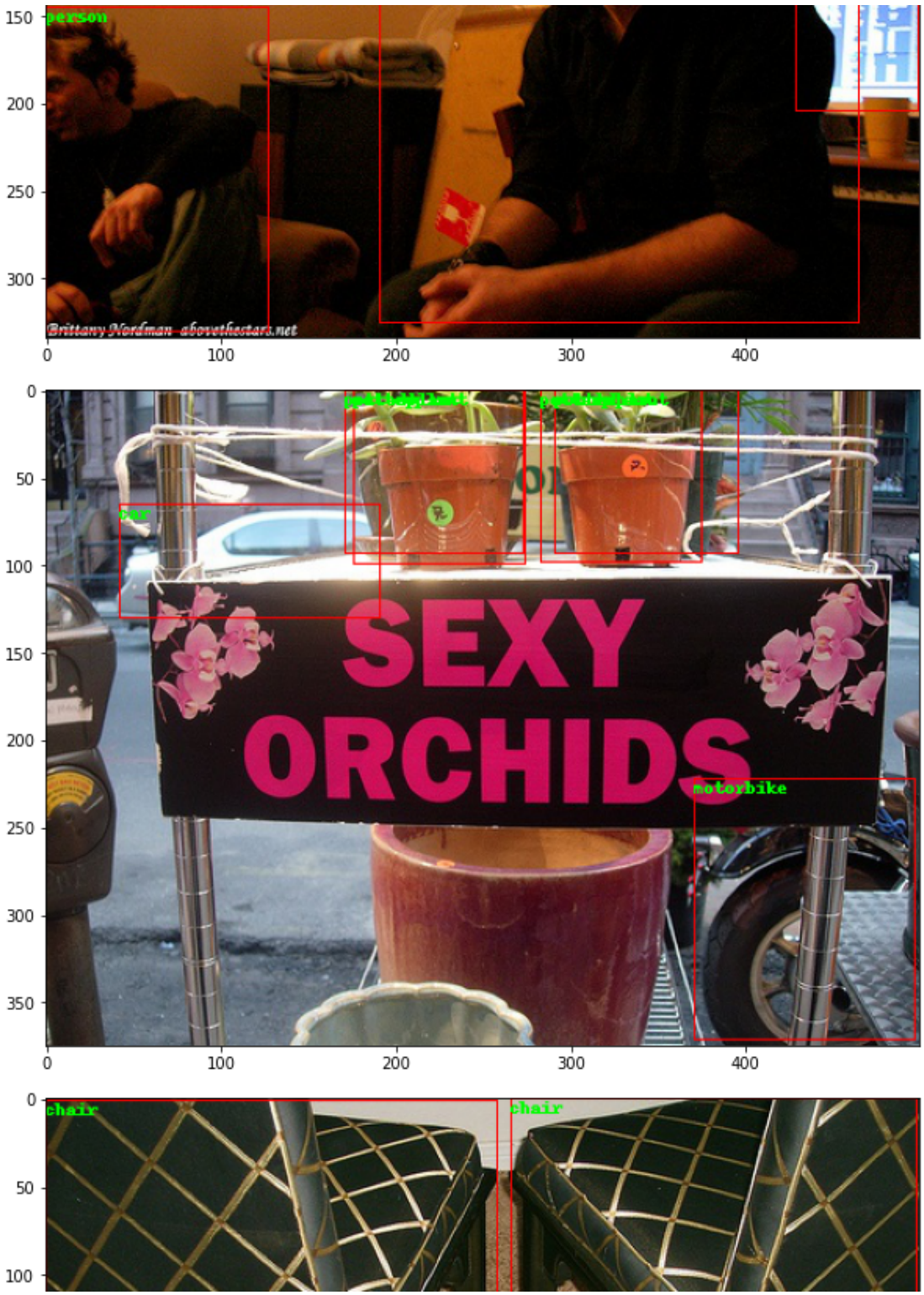
```
/content/voc_dataloader.py:137: VisibleDeprecationWarning: Creating an ndarray from
  np.array(box_indices),
```

Visualizing the Data

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

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





Classification

```
val_loader = torch.utils.data.DataLoader(dataset=ds_val,
                                         batch_size=50,
                                         shuffle=True,
                                         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, print_tot
    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_CLASS
            aps.append(ap)

        mAP = np.mean(aps)
        test_loss = np.mean(losses)
```

```
    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 find ways to design a better network. Save plots and final test mAP scores as you will be adding these to the writeup.

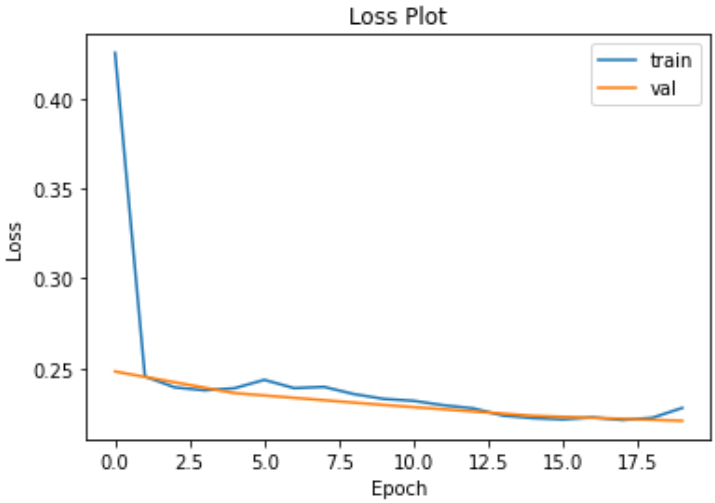
```
def train(classifier, num_epochs, train_loader, val_loader, criterion, optimizer, test_fr
    train_losses = []
    train_mAPs = []
    val_losses = []
    val_mAPs = []

    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, False,
            train_mAPs.append(mAP_train)
            mAP_val, val_loss, _ = test_classifier(val_loader, classifier, criterion)
            print('Evaluating classifier')
```

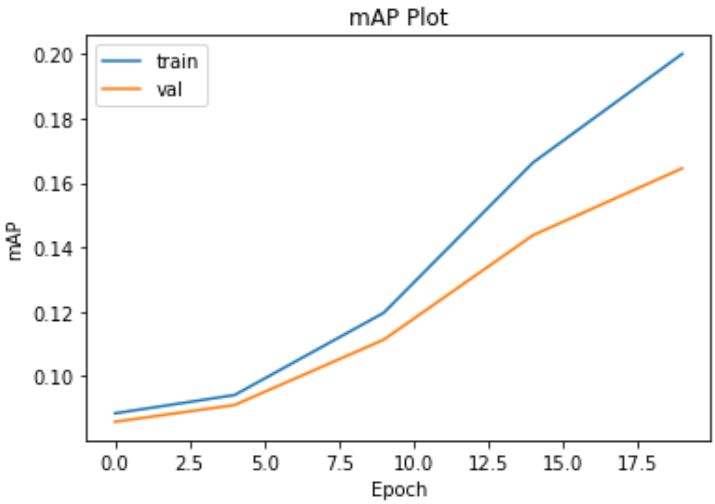
```
# Training the Classifier
num_epochs = 20
test_frequency = 5

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

```
Starting epoch number 1
Loss for Training on Epoch 1 is 0.4254213273525238
-------  Class: aeroplane        AP:   0.0766  -------
-------  Class: bicycle          AP:   0.0435  -------
-------  Class: bird             AP:   0.1188  -------
-------  Class: boat             AP:   0.0693  -------
-------  Class: bottle           AP:   0.0447  -------
-------  Class: bus              AP:   0.0271  -------
-------  Class: car              AP:   0.1078  -------
-------  Class: cat              AP:   0.1011  -------
-------  Class: chair            AP:   0.1249  -------
-------  Class: cow              AP:   0.0315  -------
-------  Class: diningtable      AP:   0.0558  -------
-------  Class: dog              AP:   0.1287  -------
-------  Class: horse            AP:   0.0518  -------
-------  Class: motorbike        AP:   0.0429  -------
-------  Class: person           AP:   0.4261  -------
-------  Class: pottedplant      AP:   0.0469  -------
-------  Class: sheep            AP:   0.0328  -------
-------  Class: sofa             AP:   0.1009  -------
-------  Class: train            AP:   0.0348  -------
-------  Class: tvmonitor        AP:   0.0510  -------
mAP: 0.0858
Avg loss: 0.24824131470100552
Evaluating classifier
Mean Precision Score for Testing on Epoch 1 is 0.08583970711222547
Starting epoch number 2
Loss for Training on Epoch 2 is 0.24549204111099243
Starting epoch number 3
Loss for Training on Epoch 3 is 0.23941317200660706
Starting epoch number 4
Loss for Training on Epoch 4 is 0.23780259490013123
Starting epoch number 5
Loss for Training on Epoch 5 is 0.23895296454429626
-------  Class: aeroplane        AP:   0.0798  -------
-------  Class: bicycle          AP:   0.0431  -------
```

```
Avg loss: 0.2362529913000032
Evaluating classifier
Mean Precision Score for Testing on Epoch 5 is 0.09103170214300192
```

```
# 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)
```





```
            transforms.Resize(227),
            transforms.CenterCrop(227),
            transforms.ToTensor(),
            normalize,
        ])

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

```
/content/voc_dataloader.py:137: VisibleDeprecationWarning: Creating an ndarray from
  np.array(box_indices),
-------  Class: aeroplane        AP:   0.3125  -------
-------  Class: bicycle          AP:   0.0656  -------
-------  Class: bird             AP:   0.1189  -------
-------  Class: boat             AP:   0.2022  -------
-------  Class: bottle           AP:   0.0666  -------
-------  Class: bus              AP:   0.0638  -------
-------  Class: car              AP:   0.3120  -------
-------  Class: cat              AP:   0.1090  -------
-------  Class: chair            AP:   0.2125  -------
-------  Class: cow              AP:   0.0645  -------
-------  Class: diningtable      AP:   0.1030  -------
-------  Class: dog              AP:   0.1490  -------
-------  Class: horse            AP:   0.1305  -------
-------  Class: motorbike        AP:   0.1116  -------
-------  Class: person           AP:   0.5364  -------
-------  Class: pottedplant      AP:   0.0704  -------
-------  Class: sheep            AP:   0.0602  -------
-------  Class: sofa             AP:   0.1890  -------
-------  Class: train            AP:   0.2000  -------
-------  Class: tvmonitor        AP:   0.1016  -------
mAP: 0.1590
Avg loss: 0.21825524017214776
```

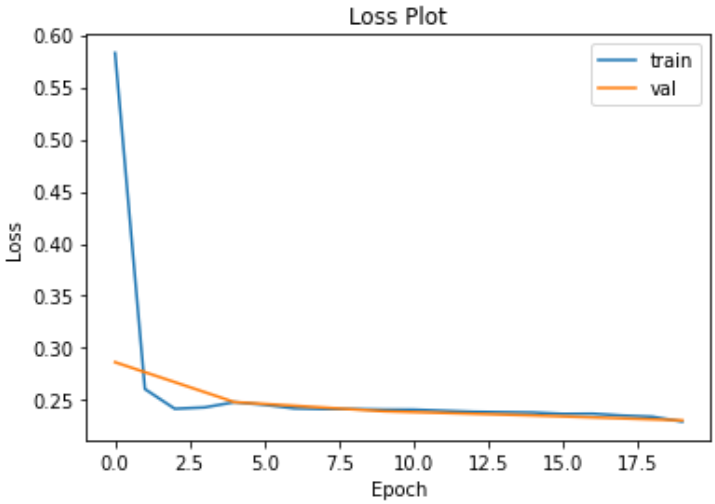
```
output_submission_csv('my_solution.csv', test_aps)
```

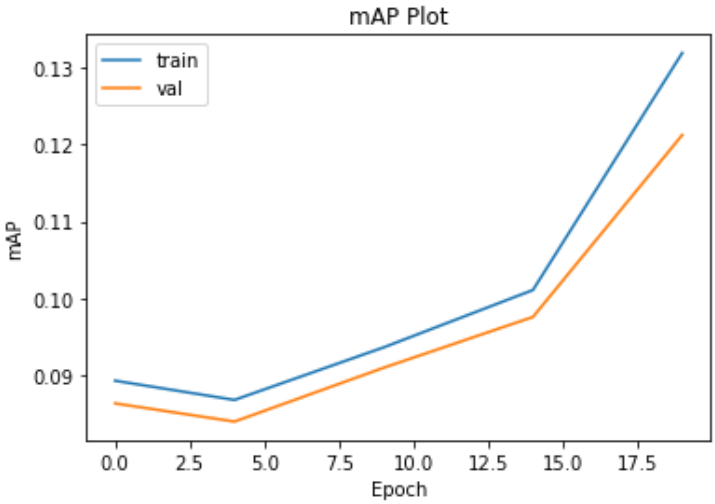
```
classifier = torchvision.models.alexnet(pretrained=False)
classifier.classifier._modules['6'] = nn.Linear(4096, 21)
classifier = classifier.to(device)

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_epochs
```

```
Starting epoch number 1
Loss for Training on Epoch 1 is 0.582821786403656
-------  Class: aeroplane        AP:   0.1285  -------
-------  Class: bicycle          AP:   0.0457  -------
-------  Class: bird             AP:   0.1066  -------
-------  Class: boat             AP:   0.1229  -------
-------  Class: bottle           AP:   0.0395  -------
-------  Class: bus              AP:   0.0298  -------
-------  Class: car              AP:   0.1955  -------
-------  Class: cat              AP:   0.0908  -------
-------  Class: chair            AP:   0.1106  -------
-------  Class: cow              AP:   0.0351  -------
-------  Class: diningtable      AP:   0.0493  -------
-------  Class: dog              AP:   0.1099  -------
-------  Class: horse            AP:   0.0459  -------
-------  Class: motorbike        AP:   0.0356  -------
-------  Class: person           AP:   0.3288  -------
-------  Class: pottedplant      AP:   0.0430  -------
-------  Class: sheep            AP:   0.0346  -------
-------  Class: sofa             AP:   0.0889  -------
-------  Class: train            AP:   0.0405  -------
-------  Class: tvmonitor        AP:   0.0477  -------
mAP: 0.0865
```

```
-------  Class: sheep            AP:   0.0394  -------
-------  Class: sofa             AP:   0.0986  -------
-------  Class: train            AP:   0.0403  -------
-------  Class: tvmonitor        AP:   0.0491  -------
mAP: 0.0841
Avg loss: 0.24812841035571753
Evaluating classifier
Mean Precision Score for Testing on Epoch 5 is 0.08409754367684243
```

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





```
-------  Class: motorbike        AP:   0.0424  -------
-------  Class: person           AP:   0.5275  -------
-------  Class: pottedplant      AP:   0.0544  -------
-------  Class: sheep            AP:   0.0605  -------
-------  Class: sofa             AP:   0.1181  -------
-------  Class: train            AP:   0.0895  -------
-------  Class: tvmonitor        AP:   0.0578  -------
mAP: 0.1175
Avg loss: 0.2271171696484089
Test mAP:  0.1175058537564142
```

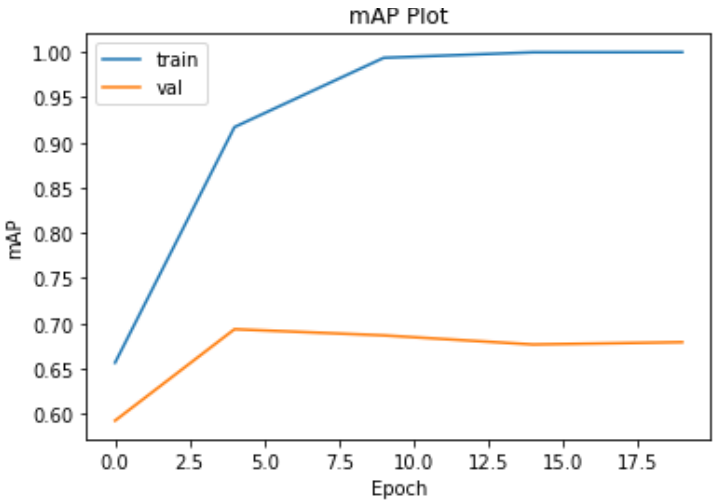
You should notice somewhat poor performance. You could try running AlexNet with an Adam optimizer instead with learning rate 1e-4 to see if that makes a difference. This experiment is not required for the writeup, but it may show you the importance of a good learning rate and optimizer.

Pretrained AlexNet

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

```
num_epochs = 20
```

```
-------  Class: pottedplant      AP:   0.3332  -------
-------  Class: sheep            AP:   0.3481  -------
-------  Class: sofa             AP:   0.4380  -------
-------  Class: train            AP:   0.8474  -------
-------  Class: tvmonitor        AP:   0.5393  -------
mAP: 0.5923
Avg loss: 0.13437851109341079
Evaluating classifier
Mean Precision Score for Testing on Epoch 1 is 0.592251797485317
Starting epoch number 2
Loss for Training on Epoch 2 is 0.1219232827425003
Starting epoch number 3
Loss for Training on Epoch 3 is 0.10006953775882721
Starting epoch number 4
Loss for Training on Epoch 4 is 0.08705829828977585
Starting epoch number 5
Loss for Training on Epoch 5 is 0.07687181234359741
-------  Class: aeroplane        AP:   0.8710  -------
-------  Class: bicycle          AP:   0.7420  -------
-------  Class: bird             AP:   0.8709  -------
-------  Class: boat             AP:   0.7717  -------
-------  Class: bottle           AP:   0.3699  -------
-------  Class: bus              AP:   0.5950  -------
-------  Class: car              AP:   0.8275  -------
-------  Class: cat              AP:   0.7973  -------
-------  Class: chair            AP:   0.6128  -------
-------  Class: cow              AP:   0.4736  -------
```




```
mAP_test, test_loss, test_aps = test_classifier(test_loader, classifier, criterion)
print("Test mAP: ", mAP_test)
```

```
-------  Class: aeroplane        AP:   0.8399  -------
-------  Class: bicycle          AP:   0.7430  -------
-------  Class: bird             AP:   0.8257  -------
```