Q4 Shoulders of Giants (15 points)

As we have already seen, deep networks can sometimes be hard to optimize. Often times they heavily overfit on small training sets. Many approaches have been proposed to counter this, eg, [Krahenbuhl et al. (ICLR'16) (http://arxiv.org/pdf/1511.06856.pdf)](http://arxiv.org/pdf/1511.06856.pdf), self-supervised learning, etc. However, the most effective approach remains pre-training the network on large, well-labeled supervised datasets such as ImageNet.

While training on the full ImageNet data is beyond the scope of this assignment, people have already trained many popular/standard models and released them online. In this task, we will initialize a ResNet-18 model with pre-trained ImageNet weights (from `torchvision`), and finetune the network for PASCAL classification.

4.1 Load Pre-trained Model (7 pts)\

Load the pre-trained weights up to the second last layer, and initialize last weights and biases from scratch.

The model loading mechanism is based on names of the weights. It is easy to load pretrained models from `torchvision.models`, even when your model uses different names for weights. Please briefly explain how to load the weights correctly if the names do not match ([hint (https://discuss.pytorch.org/t/loading-weights-from-pretrained-model-with-different-module-names/11841)](https://discuss.pytorch.org/t/loading-weights-from-pretrained-model-with-different-module-names/11841)).

YOUR ANSWER HERE

If you are loading pretrained parameters into a model and the names of the parameters do not match, you can still iterate through the layers of the model and load those weights. First, when you lod your pretrained parameters, turn those parameters into a list. You will be iterating through the list later. Build a for loop to iterate through your new models parameters. At each iteration of the for loop, set the layer_name and weights of the new model to be equal to the name and parameters of the pretrained model (which was turned into a list prior to the for loop). Iterate through the layers of the new model until the end. You should have a pretrained model-imported new model now.

```
pre_trained_model=torch.load("Path to the .pth file")
new=list(pre_trained.items())

my_model_kvpair=mymodel.state_dict()
count=0
for key,value in my_model_kvpair.item():
    layer_name,weights=new[count]
    mymodel_kvpair[key]=weights
    count+=1
```

In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models
import matplotlib.pyplot as plt
%matplotlib inline

import trainer
from utils import ARGS
from simple_cnn import SimpleCNN
from voc_dataset import VOCDataset


# Pre-trained weights up to second-to-last layer
# final layers should be initialized from scratcH!
class PretrainedResNet(nn.Module):
    def __init__(self):
        super().__init__()
        # load resnet model
        self.modelres = models.resnet18(pretrained = True)
        for params in self.modelres.parameters():
            params.requires_grad = False

        self.model= nn.Sequential(self.modelres,nn.Linear(1000,20,bias=True))

    def forward(self, x):
        return self.model(x)
```

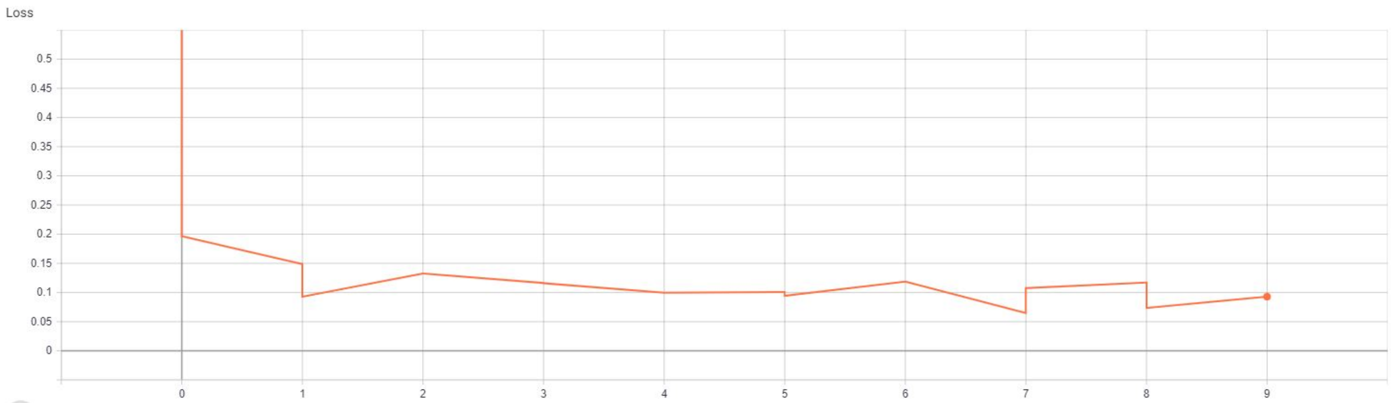
Use similar hyperparameter setup as in the scratch case. Show the learning curves (training loss, testing MAP) for 10 epochs. Please evaluate your model to calculate the MAP on the testing dataset every 100 iterations.

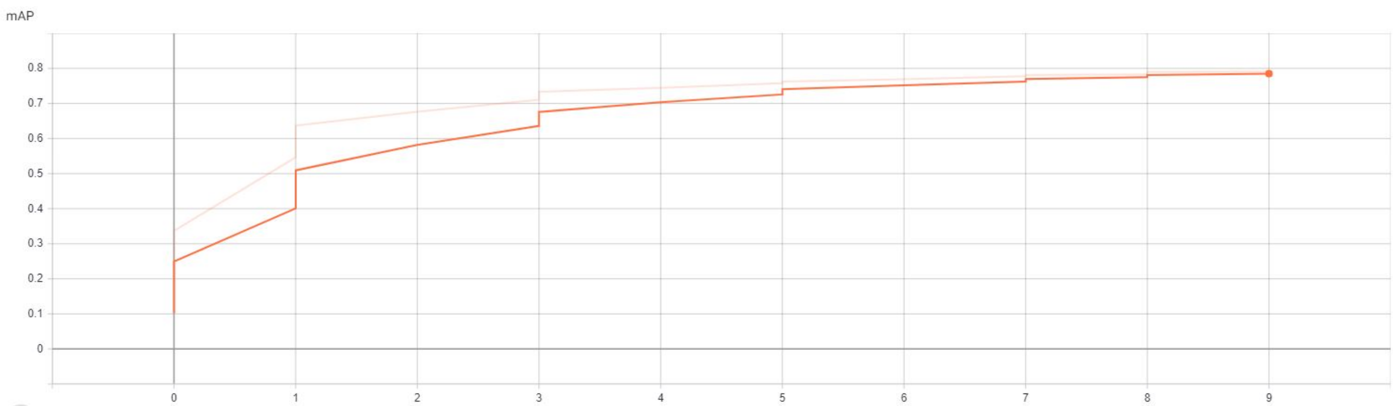
REMEMBER TO SAVE MODEL AT END OF TRAINING

In [ ]:
```
args = ARGS(batch_size = 32, epochs=10, lr = 0.0001)
args.gamma = 0.5
weightDecay = 5e-5
model = PretrainedResNet()
optimizer = torch.optim.Adam(model.parameters(), lr = args.lr, weight_decay =
weightDecay)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=15, gamma=arg
s.gamma)
if __name__ == '__main__':
    test_ap, test_map = trainer.train(args, model, optimizer, scheduler)
    print('test map:', test_map)
```

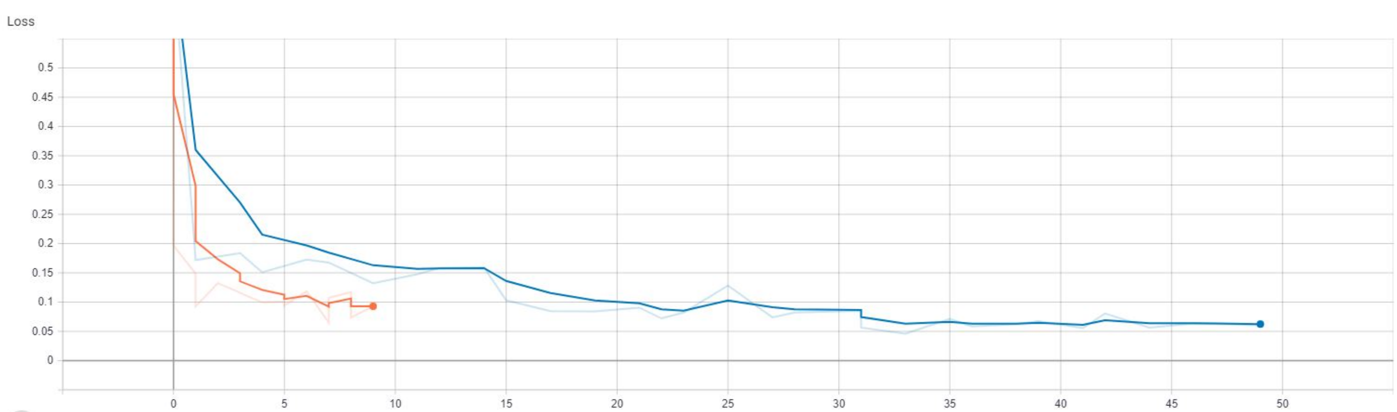
YOUR TB SCREENSHOTS HERE

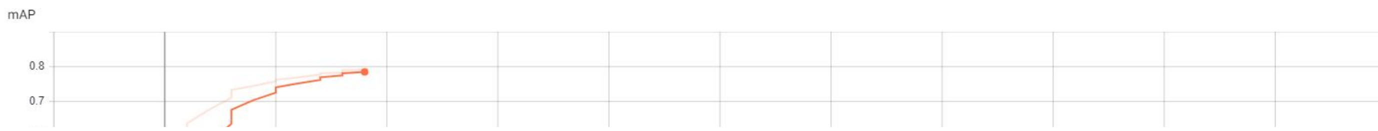
The following two figures shown below display the training loss and training mAP for the ResNet finetuned model.





The following two figures shown below compared the finetuned ResNet model with the ResNet model that was trained from scratch. The curves belonging to the transfer learning ResNet model are colored in light blue.





Appendex A: Training Epoch Information

The following log shown below displays the batch iteration, loss, and mAP calculation at various points during the training process:

Train Epoch: 0 [0 (0%)] Loss: 0.904385 | mAP: 0.112697
Train Epoch: 0 [100 (64%)] Loss: 0.246686 | mAP: 0.318172
Train Epoch: 1 [200 (27%)] Loss: 0.145483 | mAP: 0.512955
Train Epoch: 1 [300 (91%)] Loss: 0.148025 | mAP: 0.619263
Train Epoch: 2 [400 (55%)] Loss: 0.127986 | mAP: 0.674601
Train Epoch: 3 [500 (18%)] Loss: 0.116205 | mAP: 0.710900
Train Epoch: 3 [600 (82%)] Loss: 0.118125 | mAP: 0.730045
Train Epoch: 4 [700 (46%)] Loss: 0.111438 | mAP: 0.739866
Train Epoch: 5 [800 (10%)] Loss: 0.120211 | mAP: 0.755548
Train Epoch: 5 [900 (73%)] Loss: 0.099371 | mAP: 0.761425
Train Epoch: 6 [1000 (37%)] Loss: 0.103049 | mAP: 0.767812
Train Epoch: 7 [1100 (1%)] Loss: 0.095842 | mAP: 0.778130
Train Epoch: 7 [1200 (64%)] Loss: 0.111486 | mAP: 0.776830
Train Epoch: 8 [1300 (28%)] Loss: 0.082349 | mAP: 0.783933
Train Epoch: 8 [1400 (92%)] Loss: 0.100563 | mAP: 0.788158
Train Epoch: 9 [1500 (55%)] Loss: 0.095881 | mAP: 0.792471
test map: 0.7933945926532152