Q2: Lets go deeper! CaffeNet for PASCAL classification (20 pts)

Note: You are encouraged to reuse code from the previous task. Finish Q1 if you haven't already!

As you might have seen, the performance of the SimpleCNN model was pretty low for PASCAL. This is expected as PASCAL is much more complex than FASHION MNIST, and we need a much beefier model to handle it.

In this task we will be constructing a variant of the [AlexNet (https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf)](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf) architecture, known as CaffeNet. If you are familiar with Caffe, a prototxt of the network is available [here (https://github.com/BVLC/caffe/blob/master/models/bvlc_reference_caffenet/train_val.prototxt)](https://github.com/BVLC/caffe/blob/master/models/bvlc_reference_caffenet/train_val.prototxt). A visualization of the network is available [here (http://ethereon.github.io/netscope/#/preset/caffenet)](http://ethereon.github.io/netscope/#/preset/caffenet).

2.1 Build CaffeNet (5 pts)

Here is the exact model we want to build. In this task, `torchvision.models.xxx()` is NOT allowed. Define your own CaffeNet! We use the following operator notation for the architecture:

1. Convolution: A convolution with kernel size k, stride s, output channels n, padding p is represented as $conv(k, s, n, p)$.
2. Max Pooling: A max pool operation with kernel size k, stride s as $maxpool(k, s)$.
3. Fully connected: For n output units, $FC(n)$.
4. ReLU: For rectified linear non-linearity $relu()$

```
ARCHITECTURE:
-> image
-> conv(11, 4, 96, ’VALID’)
-> relu()
-> max_pool(3, 2)
-> conv(5, 1, 256, 'SAME')
-> relu()
-> max_pool(3, 2)
-> conv(3, 1, 384, 'SAME')
-> relu()
-> conv(3, 1, 384, 'SAME')
-> relu()
-> conv(3, 1, 256, ’SAME’)
-> relu()
-> max_pool(3, 2)
-> flatten()
-> fully_connected(4096)
-> relu()
-> dropout(0.5)
-> fully_connected(4096)
-> relu()
-> dropout(0.5)
-> fully_connected(20)
```

In [ ]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
# %matplotlib inline

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


def get_fc(inp_dim, out_dim, non_linear='relu'):
    """
    Mid-level API. It is useful to customize your own for large code repo.
    :param inp_dim: int, intput dimension
    :param out_dim: int, output dimension
    :param non_linear: str, 'relu', 'softmax'
    :return: list of layers [FC(inp_dim, out_dim), (non linear layer)]
    """
    layers = []
    layers.append(nn.Linear(inp_dim, out_dim))
    if non_linear == 'relu':
        layers.append(nn.ReLU())
    elif non_linear == 'softmax':
        layers.append(nn.Softmax(dim=1))
    elif non_linear == 'none':
        pass
    else:
        raise NotImplementedError
    return layers

class CaffeNet(nn.Module):
    def __init__(self):
        super().__init__()
        c_dim = 3
        self.conv1 = nn.Conv2d(c_dim,96,11,4,padding=0) # valid padding
        self.pool1 = nn.MaxPool2d(3,2)
        self.conv2 = nn.Conv2d(96, 256, 5,padding=2) # same padding
        self.pool2 = nn.MaxPool2d(3,2)
        self.conv3 = nn.Conv2d(256,384,3,padding=1) # same padding
        self.conv4 = nn.Conv2d(384,384,3,padding=1) # same padding
        self.conv5 = nn.Conv2d(384,256,3,padding=1) # same padding
        self.pool3 = nn.MaxPool2d(3,2)
        self.flat_dim = 5*5*256 # replace with the actual value
        self.fc1 = nn.Sequential(*get_fc(self.flat_dim, 4096, 'relu'))
        self.dropout1 = nn.Dropout(p=0.5)
        self.fc2 = nn.Sequential(*get_fc(4096, 4096, 'relu'))
        self.dropout2 = nn.Dropout(p=0.5)
        self.fc3 = nn.Sequential(*get_fc(4096, 20, 'none'))

        self.nonlinear = lambda x: torch.clamp(x,0)

    def forward(self, x):
        N = x.size(0)
        x = self.conv1(x)
```

```
        x = self.nonlinear(x)
        x = self.pool1(x)

        x = self.conv2(x)
        x = self.nonlinear(x)
        x = self.pool2(x)

        x = self.conv3(x)
        x = self.nonlinear(x)
        x = self.conv4(x)
        x = self.nonlinear(x)
        x = self.conv5(x)
        x = self.nonlinear(x)
        x = self.pool3(x)
        x = x.view(N, self.flat_dim) # flatten the array

        out = self.fc1(x)
        out = self.nonlinear(out)
        out = self.dropout1(out)
        out = self.fc2(out)
        out = self.nonlinear(out)
        out = self.dropout2(out)
        out = self.fc3(out)

        return out
```

2.2 Save the Model (5 pts)

Finish code stubs for saving the model periodically into `trainer.py` . **You will need these models later**

2.3 Train and Test (5pts)

Show clear screenshots of testing MAP and training loss for 50 epochs. Please evaluate your model to calculate the MAP on the testing dataset every 250 iterations. Use the following hyperparamters:

- batch_size=32
- Adam optimizer with lr=0.0001

NOTE: SAVE AT LEAST 5 EVENLY SPACED CHECKPOINTS DURING TRAINING (1 at end)

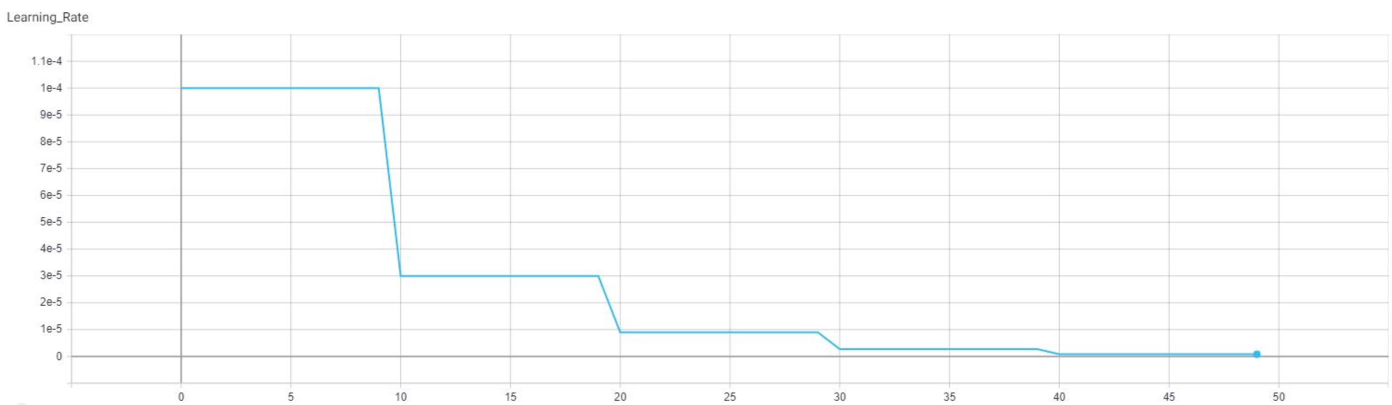
In [ ]:
```
def xavier_normal_init(m):
    if(type(m)==nn.Conv1d or type(m)==nn.Conv2d or type(m)==nn.Linear):
        torch.nn.init.xavier_normal_(m.weight.data)
        if(m.bias is not None):
            torch.nn.init.xavier_normal_(m.weight.data)
```

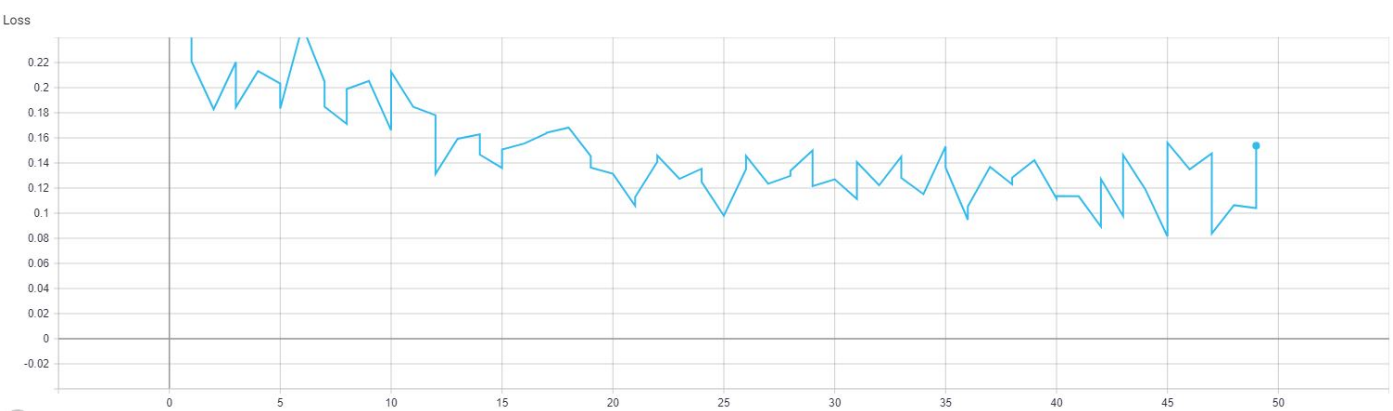
In [ ]:

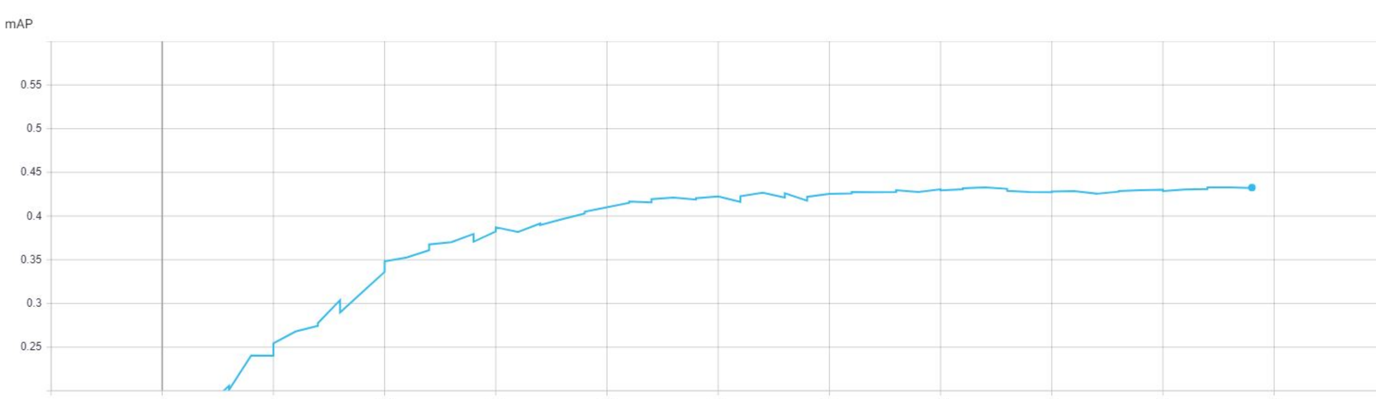
```
args = ARGS(batch_size = 32, epochs=50, lr = 0.0001)
args.gamma = 0.3
weightDecay = 5e-5
model = CaffeNet()
model.apply(xavier_normal_init)
optimizer = torch.optim.Adam(model.parameters(), lr = args.lr,weight_decay=weightDecay) # ,weight_decay=weightDecay
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=args.gamma)
if __name__ == '__main__':
    test_ap, test_map = trainer.train(args, model, optimizer, scheduler)
    print('test map:', test_map)
```

INSERT YOUR TENSORBOARD SCREENSHOTS HERE

The figures below display the learning rate, loss, and map, respectively, during training.



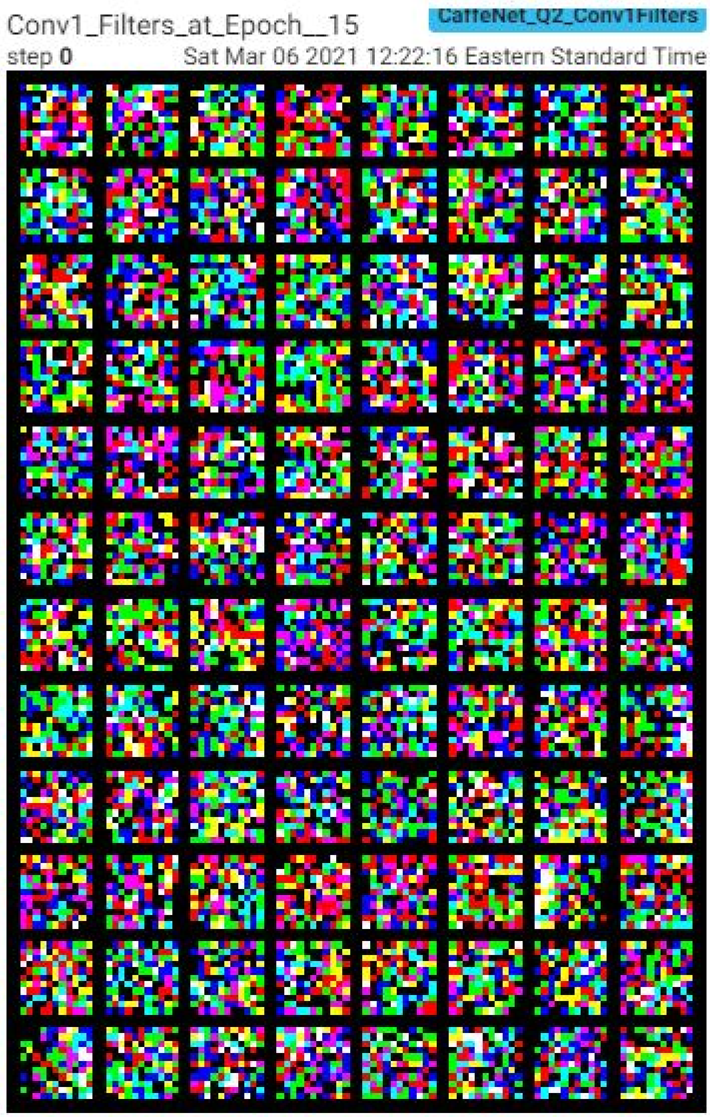




2.4 Visualizing: Conv-1 filters (5pts)

Extract and compare the conv1 filters, at different stages of the training (at least from 3 different iterations). Show at least 5 filters.

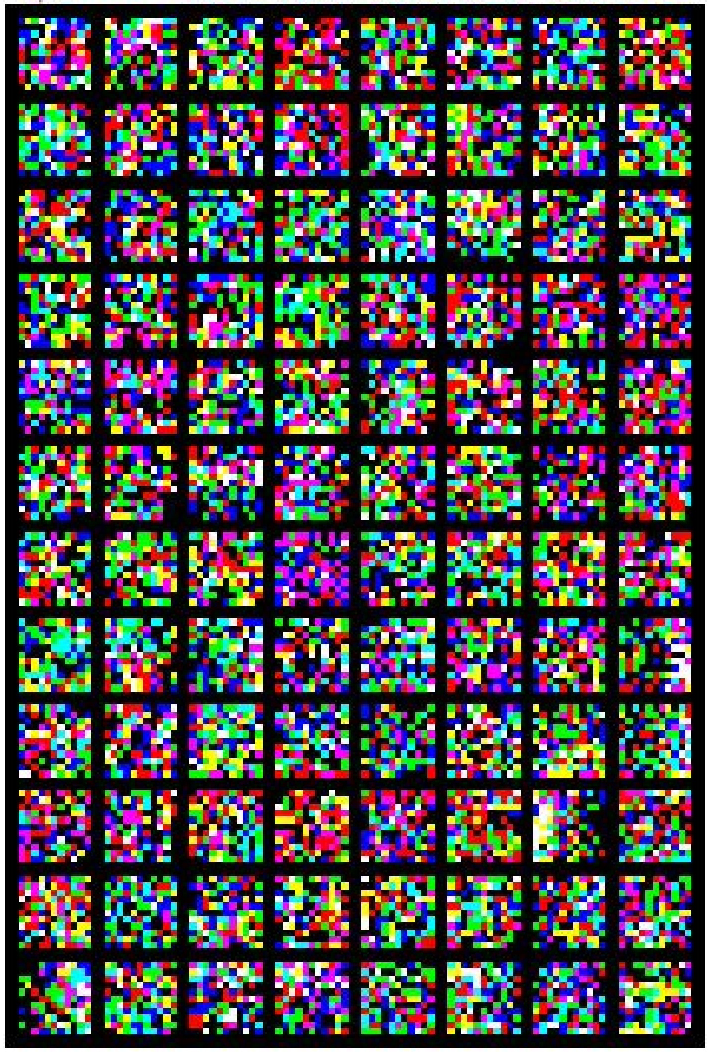
The following two image grids below show the Conv1 layer filters at epochs 15, 30, and 45, respectively:



Conv1_Filters_at_Epoch__30

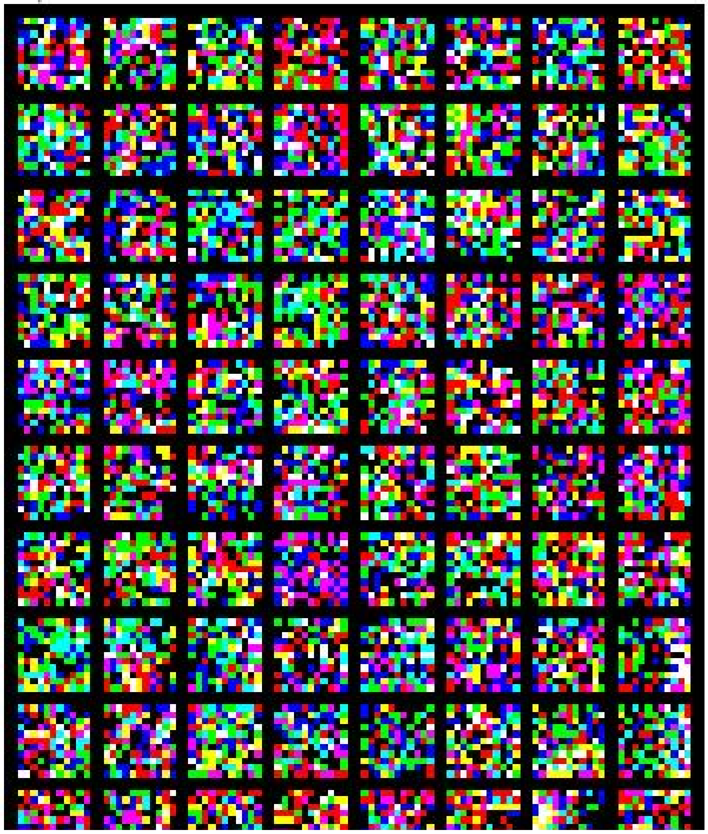
CaffeNet_Q2_Conv1Filters

step **0** Sat Mar 06 2021 12:38:59 Eastern Standard Time



Conv1_Filters_at_Epoch__45 CaffeNet_Q2_Conv1Filters

step **0** Sat Mar 06 2021 12:55:07 Eastern Standard Time



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: 0.694260 | mAP: 0.079919
Train Epoch: 0 [100 (64%)] Loss: 0.252534 | mAP: 0.106128
Train Epoch: 1 [200 (27%)] Loss: 0.245160 | mAP: 0.125702
Train Epoch: 1 [300 (91%)] Loss: 0.220970 | mAP: 0.154655
Train Epoch: 2 [400 (55%)] Loss: 0.182752 | mAP: 0.181409
Train Epoch: 3 [500 (18%)] Loss: 0.220406 | mAP: 0.205849
Train Epoch: 3 [600 (82%)] Loss: 0.184534 | mAP: 0.200986
Train Epoch: 4 [700 (46%)] Loss: 0.213212 | mAP: 0.240417
Train Epoch: 5 [800 (10%)] Loss: 0.203305 | mAP: 0.240243
Train Epoch: 5 [900 (73%)] Loss: 0.183403 | mAP: 0.254320
Train Epoch: 6 [1000 (37%)] Loss: 0.248874 | mAP: 0.267995
Train Epoch: 7 [1100 (1%)] Loss: 0.204796 | mAP: 0.274374
Train Epoch: 7 [1200 (64%)] Loss: 0.184777 | mAP: 0.277550
Train Epoch: 8 [1300 (28%)] Loss: 0.171141 | mAP: 0.303782
Train Epoch: 8 [1400 (92%)] Loss: 0.198896 | mAP: 0.289614
Train Epoch: 9 [1500 (55%)] Loss: 0.205384 | mAP: 0.312908
Train Epoch: 10 [1600 (19%)] Loss: 0.166079 | mAP: 0.336100
Train Epoch: 10 [1700 (83%)] Loss: 0.212687 | mAP: 0.348000
Train Epoch: 11 [1800 (46%)] Loss: 0.184672 | mAP: 0.352580
Train Epoch: 12 [1900 (10%)] Loss: 0.178017 | mAP: 0.360864
Train Epoch: 12 [2000 (74%)] Loss: 0.131287 | mAP: 0.367583
Train Epoch: 13 [2100 (38%)] Loss: 0.159262 | mAP: 0.370199
Train Epoch: 14 [2200 (1%)] Loss: 0.162815 | mAP: 0.379512
Train Epoch: 14 [2300 (65%)] Loss: 0.146647 | mAP: 0.370575
Train Epoch: 15 [2400 (29%)] Loss: 0.136011 | mAP: 0.382428
Train Epoch: 15 [2500 (92%)] Loss: 0.150734 | mAP: 0.387179
Train Epoch: 16 [2600 (56%)] Loss: 0.155456 | mAP: 0.381900
Train Epoch: 17 [2700 (20%)] Loss: 0.163627 | mAP: 0.391537
Train Epoch: 17 [2800 (83%)] Loss: 0.164096 | mAP: 0.389797
Train Epoch: 18 [2900 (47%)] Loss: 0.168196 | mAP: 0.396490
Train Epoch: 19 [3000 (11%)] Loss: 0.145444 | mAP: 0.402965
Train Epoch: 19 [3100 (75%)] Loss: 0.136387 | mAP: 0.404979
Train Epoch: 20 [3200 (38%)] Loss: 0.131393 | mAP: 0.410043
Train Epoch: 21 [3300 (2%)] Loss: 0.105851 | mAP: 0.415120
Train Epoch: 21 [3400 (66%)] Loss: 0.112801 | mAP: 0.416542
Train Epoch: 22 [3500 (29%)] Loss: 0.140922 | mAP: 0.415565
Train Epoch: 22 [3600 (93%)] Loss: 0.145867 | mAP: 0.419292
Train Epoch: 23 [3700 (57%)] Loss: 0.127371 | mAP: 0.421104
Train Epoch: 24 [3800 (20%)] Loss: 0.135417 | mAP: 0.418907
Train Epoch: 24 [3900 (84%)] Loss: 0.124882 | mAP: 0.420355
Train Epoch: 25 [4000 (48%)] Loss: 0.097941 | mAP: 0.422405
Train Epoch: 26 [4100 (11%)] Loss: 0.134959 | mAP: 0.416246
Train Epoch: 26 [4200 (75%)] Loss: 0.145727 | mAP: 0.422733

Train Epoch: 27 [4300 (39%)] Loss: 0.123460 | mAP: 0.426791
Train Epoch: 28 [4400 (3%)] Loss: 0.129733 | mAP: 0.421078
Train Epoch: 28 [4500 (66%)] Loss: 0.133712 | mAP: 0.426055
Train Epoch: 29 [4600 (30%)] Loss: 0.149955 | mAP: 0.417618
Train Epoch: 29 [4700 (94%)] Loss: 0.121479 | mAP: 0.421980
Train Epoch: 30 [4800 (57%)] Loss: 0.127036 | mAP: 0.425303
Train Epoch: 31 [4900 (21%)] Loss: 0.111305 | mAP: 0.425996
Train Epoch: 31 [5000 (85%)] Loss: 0.140682 | mAP: 0.427547
Train Epoch: 32 [5100 (48%)] Loss: 0.122193 | mAP: 0.427262
Train Epoch: 33 [5200 (12%)] Loss: 0.145076 | mAP: 0.427588
Train Epoch: 33 [5300 (76%)] Loss: 0.128083 | mAP: 0.429591
Train Epoch: 34 [5400 (39%)] Loss: 0.115151 | mAP: 0.427573
Train Epoch: 35 [5500 (3%)] Loss: 0.153220 | mAP: 0.430578
Train Epoch: 35 [5600 (67%)] Loss: 0.136315 | mAP: 0.429386
Train Epoch: 36 [5700 (31%)] Loss: 0.094817 | mAP: 0.430634
Train Epoch: 36 [5800 (94%)] Loss: 0.105419 | mAP: 0.431691
Train Epoch: 37 [5900 (58%)] Loss: 0.136911 | mAP: 0.432678
Train Epoch: 38 [6000 (22%)] Loss: 0.122926 | mAP: 0.431091
Train Epoch: 38 [6100 (85%)] Loss: 0.128397 | mAP: 0.428878
Train Epoch: 39 [6200 (49%)] Loss: 0.142264 | mAP: 0.427402
Train Epoch: 40 [6300 (13%)] Loss: 0.111605 | mAP: 0.427158
Train Epoch: 40 [6400 (76%)] Loss: 0.113659 | mAP: 0.428018
Train Epoch: 41 [6500 (40%)] Loss: 0.113470 | mAP: 0.428675
Train Epoch: 42 [6600 (4%)] Loss: 0.089273 | mAP: 0.425716
Train Epoch: 42 [6700 (68%)] Loss: 0.127156 | mAP: 0.425413
Train Epoch: 43 [6800 (31%)] Loss: 0.097778 | mAP: 0.427899
Train Epoch: 43 [6900 (95%)] Loss: 0.146353 | mAP: 0.428698
Train Epoch: 44 [7000 (59%)] Loss: 0.119124 | mAP: 0.429632
Train Epoch: 45 [7100 (22%)] Loss: 0.081511 | mAP: 0.430262
Train Epoch: 45 [7200 (86%)] Loss: 0.156132 | mAP: 0.428628
Train Epoch: 46 [7300 (50%)] Loss: 0.134881 | mAP: 0.430437
Train Epoch: 47 [7400 (13%)] Loss: 0.147624 | mAP: 0.430969
Train Epoch: 47 [7500 (77%)] Loss: 0.083810 | mAP: 0.432749
Train Epoch: 48 [7600 (41%)] Loss: 0.106350 | mAP: 0.432845
Train Epoch: 49 [7700 (4%)] Loss: 0.104012 | mAP: 0.431998
Train Epoch: 49 [7800 (68%)] Loss: 0.153784 | mAP: 0.432565
test map: 0.41263213323264036