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) 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). A visualization of the network is available here (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 [2]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision.models.utils import load_state_dict_from_url
import os
import matplotlib.pyplot as plt
%matplotlib inline

import trainer
from utils import ARGS
from simple_cnn import SimpleCNN
from voc_dataset import VOCDataset
%env CUDA_VISIBLE_DEVICES=0

%matplotlib inline



class CaffeNet(nn.Module):
    def __init__(self):
        super().__init__()
        conv1 = [nn.Conv2d(in_channels=3,out_channels=96,kernel_size=11,

        self.conv_first = nn.Sequential(*conv1)

        self.conv1_features = [] #epoch,features of filters
        conv_layers = [
                        nn.ReLU(inplace=True),
                       nn.MaxPool2d(kernel_size=3,stride=2),
                       nn.Conv2d(in_channels=96,out_channels=256,stride=
                       nn.ReLU(inplace=True),
                       nn.MaxPool2d(kernel_size=3,stride=2),

                       nn.Conv2d(in_channels=256,out_channels=384,stride
                       nn.ReLU(inplace=True),

                       nn.Conv2d(in_channels=384,out_channels=384,stride
                       nn.ReLU(inplace=True),

                       nn.Conv2d(in_channels=384,out_channels=256,stride
                       nn.ReLU(inplace=True),
                       nn.MaxPool2d(kernel_size=3,stride=2)
                      ]
        self.conv_layers = nn.Sequential(*conv_layers)
        fc = [nn.Linear(256*11*11,4096,bias=True),
             nn.ReLU(inplace=True),
             nn.Dropout(0.5),

              nn.Linear(4096,4096,bias=True),
            nn.ReLU(inplace=True),
              nn.Dropout(0.5),

              nn.Linear(4096,20,bias=True)
             ]
        self.fc_layers = nn.Sequential(*fc)
#         self.conv_first.register_forward_hook(self.save_outputs_hook()
```

```
#     def save_outputs_hook(self):
#         def hook(model,inp, output):
#             self.conv1_features.append(output.detach())
#         return hook



    def forward(self, x):
        x=self.conv_first(x)
        x=self.conv_layers(x)
        x=torch.flatten(x,start_dim=1)
        out=self.fc_layers(x)
        return out
```

```
env: CUDA_VISIBLE_DEVICES=0
```

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)

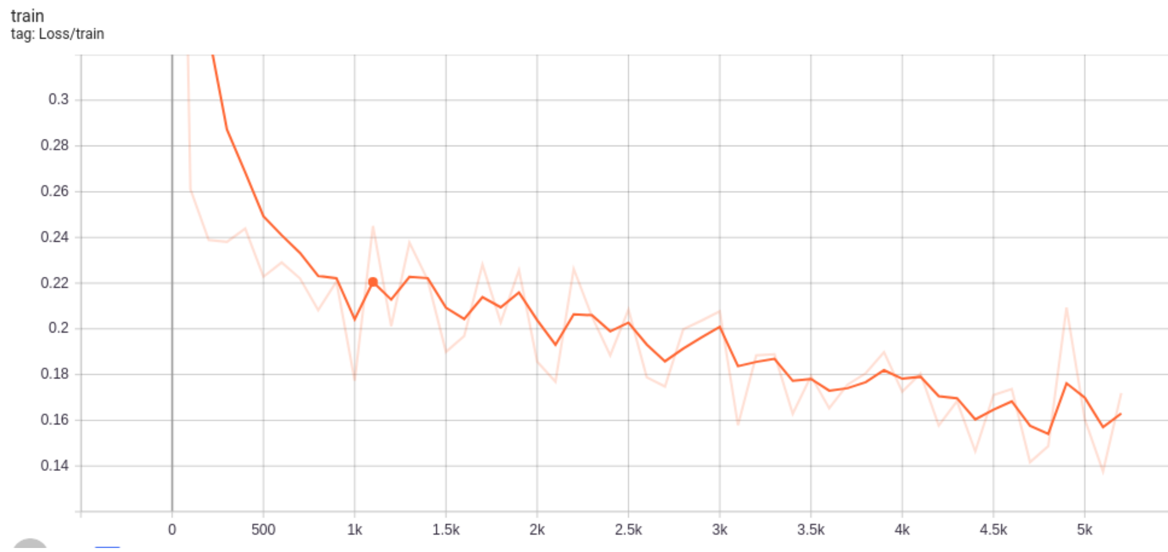
In [3]:
```
args = ARGS(epochs=50, batch_size=32,test_batch_size=32, lr=0.0001,val_e
model = CaffeNet()
optimizer = torch.optim.Adam(model.parameters(),lr=args.lr)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,step_size=5,gamma=
test_ap, test_map = trainer.train(args, model, optimizer, scheduler)
print('test map:', test_map)
```

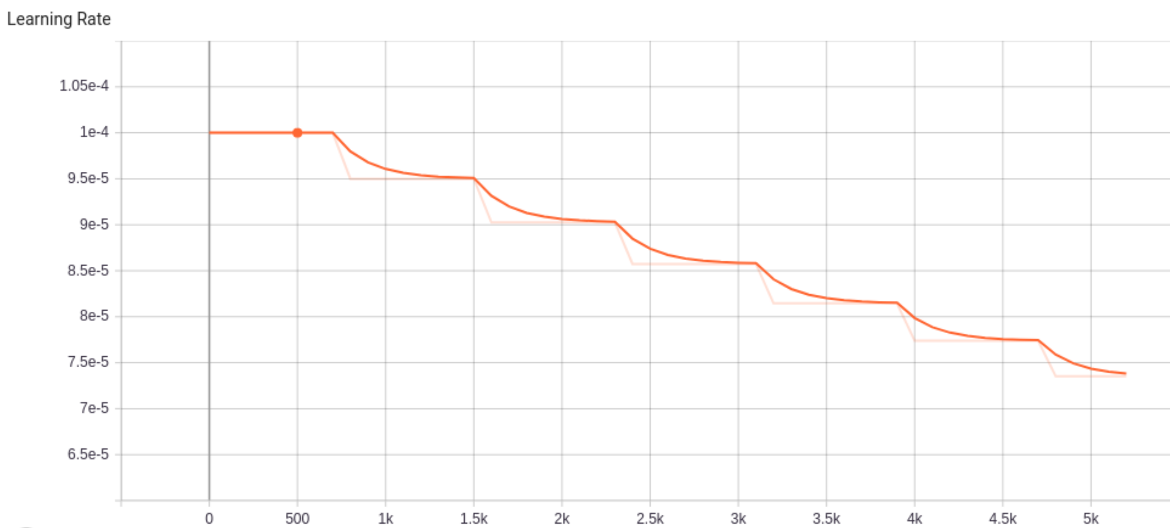
```
Train Epoch: 0 [0 (0%)] Loss: 0.692977
Test Epoch: 0 [0 (0%)]  mAP: 0.069898
Train Epoch: 0 [100 (64%)]      Loss: 0.261110
Train Epoch: 1 [200 (27%)]      Loss: 0.238920
Test Epoch: 1 [250 (59%)]       mAP: 0.115613
Train Epoch: 1 [300 (91%)]      Loss: 0.238063
Train Epoch: 2 [400 (55%)]      Loss: 0.243902
Train Epoch: 3 [500 (18%)]      Loss: 0.222822
Test Epoch: 3 [500 (18%)]       mAP: 0.155766
Train Epoch: 3 [600 (82%)]      Loss: 0.229039
Train Epoch: 4 [700 (46%)]      Loss: 0.222055
Test Epoch: 4 [750 (78%)]       mAP: 0.169797
Train Epoch: 5 [800 (10%)]      Loss: 0.208249
Train Epoch: 5 [900 (73%)]      Loss: 0.220676
Train Epoch: 6 [1000 (37%)]     Loss: 0.177393
Test Epoch: 6 [1000 (37%)]      mAP: 0.193221
Train Epoch: 7 [1100 (1%)]      Loss: 0.244983
Train Epoch: 7 [1200 (64%)]     Loss: 0.201182
Test Epoch: 7 [1250 (96%)]      mAP: 0.210451
Train Epoch: 8 [1300 (28%)]     Loss: 0.237658
Train Epoch: 8 [1400 (92%)]     Loss: 0.221375
Train Epoch: 9 [1500 (55%)]     Loss: 0.189915
Test Epoch: 9 [1500 (55%)]      mAP: 0.222596
Train Epoch: 10 [1600 (19%)]    Loss: 0.196881
Train Epoch: 10 [1700 (83%)]    Loss: 0.228156
Test Epoch: 11 [1750 (15%)]     mAP: 0.241162
Train Epoch: 11 [1800 (46%)]    Loss: 0.202781
Train Epoch: 12 [1900 (10%)]    Loss: 0.225615
Train Epoch: 12 [2000 (74%)]    Loss: 0.185650
Test Epoch: 12 [2000 (74%)]     mAP: 0.227675
Train Epoch: 13 [2100 (38%)]    Loss: 0.176899
Train Epoch: 14 [2200 (1%)]     Loss: 0.226242
Test Epoch: 14 [2250 (33%)]     mAP: 0.272072
Train Epoch: 14 [2300 (65%)]    Loss: 0.205388
Train Epoch: 15 [2400 (29%)]    Loss: 0.188402
Train Epoch: 15 [2500 (92%)]    Loss: 0.208372
Test Epoch: 15 [2500 (92%)]     mAP: 0.269799
Train Epoch: 16 [2600 (56%)]    Loss: 0.178793
Train Epoch: 17 [2700 (20%)]    Loss: 0.174762
Test Epoch: 17 [2750 (52%)]     mAP: 0.285524
Train Epoch: 17 [2800 (83%)]    Loss: 0.199813
Train Epoch: 18 [2900 (47%)]    Loss: 0.203604
Train Epoch: 19 [3000 (11%)]    Loss: 0.207692
Test Epoch: 19 [3000 (11%)]     mAP: 0.275282
Train Epoch: 19 [3100 (75%)]    Loss: 0.157850
Train Epoch: 20 [3200 (38%)]    Loss: 0.188402
Test Epoch: 20 [3250 (70%)]     mAP: 0.314228
Train Epoch: 21 [3300 (2%)]     Loss: 0.188862
Train Epoch: 21 [3400 (66%)]    Loss: 0.162842
Train Epoch: 22 [3500 (29%)]    Loss: 0.179340
```

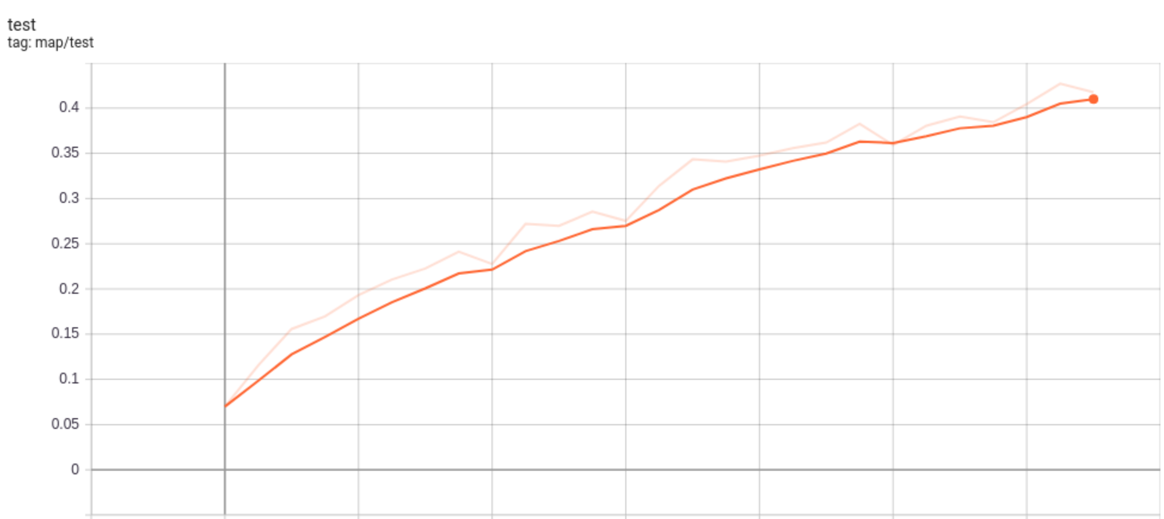
```
Test Epoch: 22 [3500 (29%)]     mAP: 0.343395
Train Epoch: 22 [3600 (93%)]    Loss: 0.165279
Train Epoch: 23 [3700 (57%)]    Loss: 0.175619
Test Epoch: 23 [3750 (89%)]     mAP: 0.340914
Train Epoch: 24 [3800 (20%)]    Loss: 0.180571
Train Epoch: 24 [3900 (84%)]    Loss: 0.189798
Train Epoch: 25 [4000 (48%)]    Loss: 0.172531
Test Epoch: 25 [4000 (48%)]     mAP: 0.347360
Train Epoch: 26 [4100 (11%)]    Loss: 0.180401
Train Epoch: 26 [4200 (75%)]    Loss: 0.157703
Test Epoch: 27 [4250 (7%)]      mAP: 0.355838
Train Epoch: 27 [4300 (39%)]    Loss: 0.168389
Train Epoch: 28 [4400 (3%)]     Loss: 0.146437
Train Epoch: 28 [4500 (66%)]    Loss: 0.171070
Test Epoch: 28 [4500 (66%)]     mAP: 0.362001
Train Epoch: 29 [4600 (30%)]    Loss: 0.173725
Train Epoch: 29 [4700 (94%)]    Loss: 0.141663
Test Epoch: 30 [4750 (25%)]     mAP: 0.382660
Train Epoch: 30 [4800 (57%)]    Loss: 0.148725
Train Epoch: 31 [4900 (21%)]    Loss: 0.209353
Train Epoch: 31 [5000 (85%)]    Loss: 0.160433
Test Epoch: 31 [5000 (85%)]     mAP: 0.359178
Train Epoch: 32 [5100 (48%)]    Loss: 0.137650
Train Epoch: 33 [5200 (12%)]    Loss: 0.171916
Test Epoch: 33 [5250 (44%)]     mAP: 0.380550
Train Epoch: 33 [5300 (76%)]    Loss: 0.167809
Train Epoch: 34 [5400 (39%)]    Loss: 0.195813
Train Epoch: 35 [5500 (3%)]     Loss: 0.151520
Test Epoch: 35 [5500 (3%)]      mAP: 0.390740
Train Epoch: 35 [5600 (67%)]    Loss: 0.121309
Train Epoch: 36 [5700 (31%)]    Loss: 0.178890
Test Epoch: 36 [5750 (62%)]     mAP: 0.384516
Train Epoch: 36 [5800 (94%)]    Loss: 0.155795
Train Epoch: 37 [5900 (58%)]    Loss: 0.166594
Train Epoch: 38 [6000 (22%)]    Loss: 0.142097
Test Epoch: 38 [6000 (22%)]     mAP: 0.404675
Train Epoch: 38 [6100 (85%)]    Loss: 0.161921
Train Epoch: 39 [6200 (49%)]    Loss: 0.141919
Test Epoch: 39 [6250 (81%)]     mAP: 0.427050
Train Epoch: 40 [6300 (13%)]    Loss: 0.113942
Train Epoch: 40 [6400 (76%)]    Loss: 0.167998
Train Epoch: 41 [6500 (40%)]    Loss: 0.138742
Test Epoch: 41 [6500 (40%)]     mAP: 0.417689
Train Epoch: 42 [6600 (4%)]     Loss: 0.139222
Train Epoch: 42 [6700 (68%)]    Loss: 0.153477
Test Epoch: 42 [6750 (99%)]     mAP: 0.435103
Train Epoch: 43 [6800 (31%)]    Loss: 0.146585
Train Epoch: 43 [6900 (95%)]    Loss: 0.141861
Train Epoch: 44 [7000 (59%)]    Loss: 0.123305
Test Epoch: 44 [7000 (59%)]     mAP: 0.408931
Train Epoch: 45 [7100 (22%)]    Loss: 0.131223
Train Epoch: 45 [7200 (86%)]    Loss: 0.145928
Test Epoch: 46 [7250 (18%)]     mAP: 0.424120
Train Epoch: 46 [7300 (50%)]    Loss: 0.128794
Train Epoch: 47 [7400 (13%)]    Loss: 0.158533
Train Epoch: 47 [7500 (77%)]    Loss: 0.111411
Test Epoch: 47 [7500 (77%)]     mAP: 0.415200
```

```
Train Epoch: 48 [7600 (41%)]    Loss: 0.134618
Train Epoch: 49 [7700 (4%)]     Loss: 0.139118
Test Epoch: 49 [7750 (36%)]     mAP: 0.419936
Train Epoch: 49 [7800 (68%)]    Loss: 0.130249
test map: 0.41278551962164645
```

INSERT YOUR TENSORBOARD SCREENSHOTS HERE



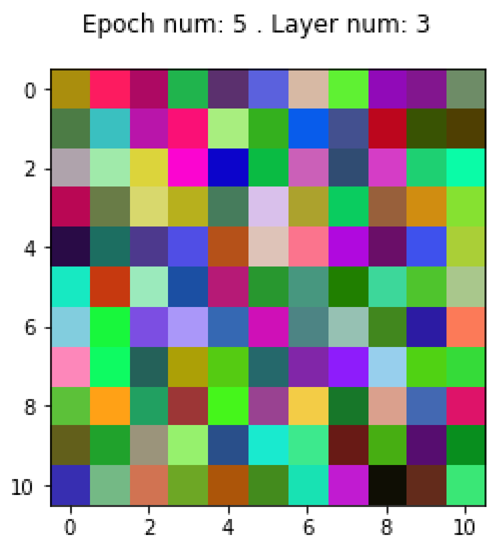


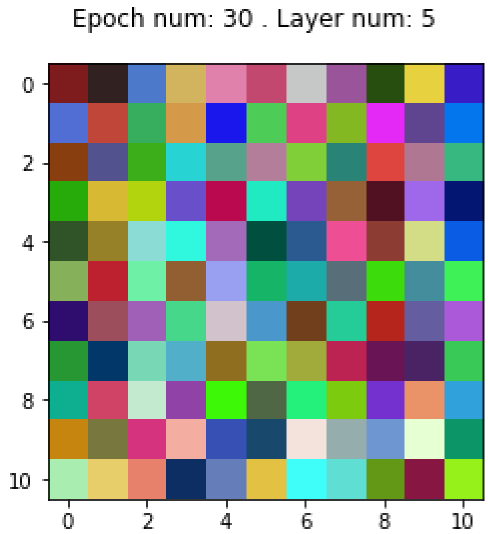


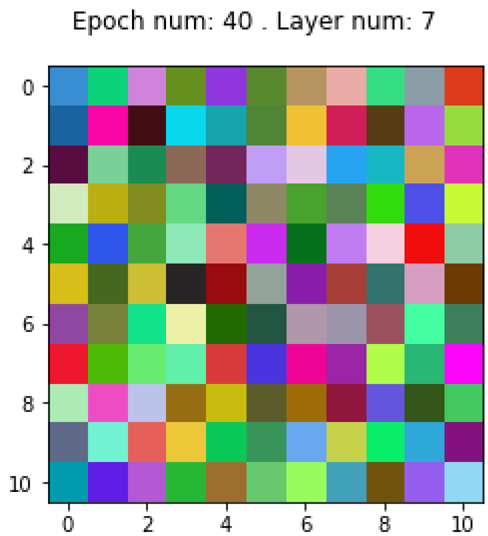
2.4 Visualizing: Conv-1 filters (5pts)

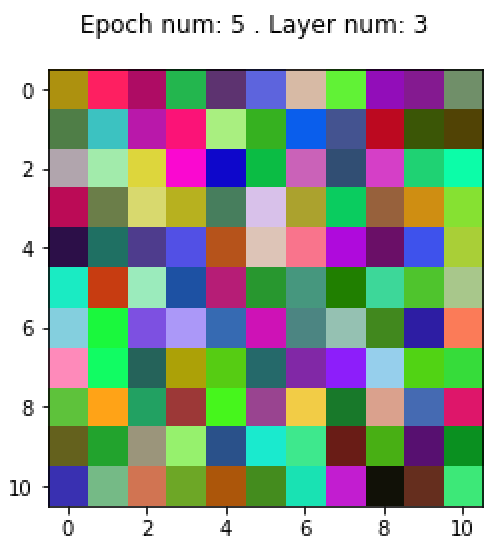
Extract and compare the conv1 filters, at different stages of the training (at least from 3 different iterations). Show at least 5 filters.

In [48]:
```
# visualize below
model_weights_5 = torch.load(os.path.join(model.__class__.__name__,"mode
model_weights_30 = torch.load(os.path.join(model.__class__.__name__,"mod
model_weights_40 = torch.load(os.path.join(model.__class__.__name__,"mod
# conv1_weight = chk["model_state_dict"]["conv_first.weight"]
wt_5 = model_weights_5['conv_first.0.weight']
wt_30 = model_weights_30['conv_first.0.weight']
wt_40 = model_weights_40['conv_first.0.weight']
weights = [wt.detach().cpu().numpy() for wt in [wt_5,wt_30,wt_40]]
layer_num = [3,5,7]
epoch_num = [5,30,40]
weights =  [wt[layer_num] for wt in weights] #get wieghts at 3rd 5th and
filters_epochs = [weight.transpose(0,2,3,1) for weight in weights]
for filters_epoch in filters_epochs:
    for idx,filter_ in enumerate(filters_epoch):
        plt.figure()
        plt.suptitle('Epoch num: {} . Layer num: {}'.format(epoch_num[id
        filter_ = (filter_-filter_.min())/(filter_.max()-filter_.min())
        plt.imshow(filter_)
        plt.show()
```

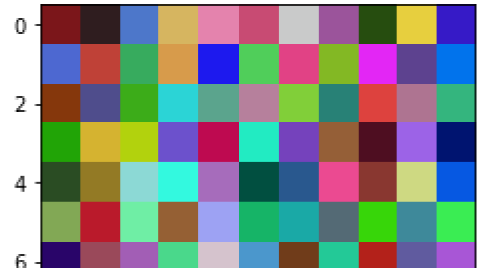



Epoch num: 30 . Layer num: 5
0
2
4
6
8
10

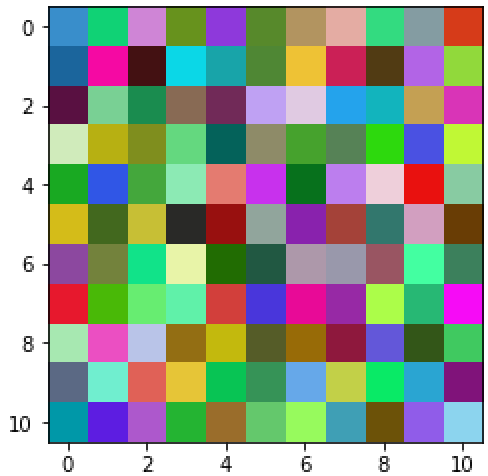

Epoch num: 40 . Layer num: 7
0
2
4
6
8
10


Epoch num: 5 . Layer num: 3
0
2
4
6
8
10

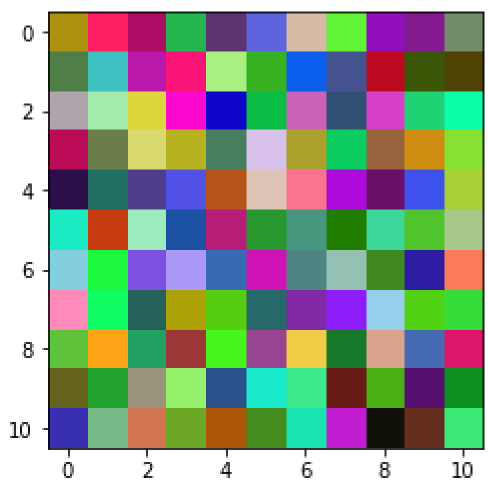
Epoch num: 30 . Layer num: 5



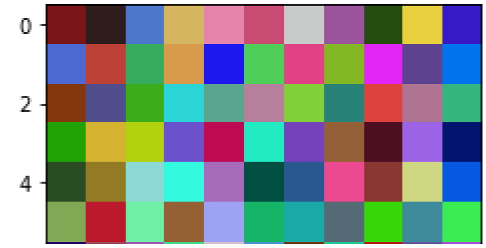
Epoch num: 40 . Layer num: 7



Epoch num: 5 . Layer num: 3



Epoch num: 30 . Layer num: 5



Epoch num: 40 . Layer num: 7