Project

March 9, 2020

[6]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import scipy.io as scio
import cv2 # version: 3.4.2
import numpy as np
from PIL import Image
from matplotlib import pyplot as plt
from collections import defaultdict
import warnings
warnings.filterwarnings('ignore')
np.random.seed(7)
device = None
if torch.cuda.device_count() == 0:
    device = torch.device('cpu')
else:
    device = torch.device('cuda')
print('Use device ', device)
```

```
Use device  cuda
```

[2]:
```
TRAIN_PATH='./train_32x32.mat'
TEST_PATH='./test_32x32.mat'
EXTRA_PATH='./extra_32x32.mat'

def loadData(path):
    res = scio.loadmat(path)
    return res['X'], res['y']
Xtr, Ytr = loadData(TRAIN_PATH)
Xte, Yte = loadData(TEST_PATH)
Xex, Yex = loadData(EXTRA_PATH)
print(Xtr.shape)
```

```
(32, 32, 3, 73257)
```

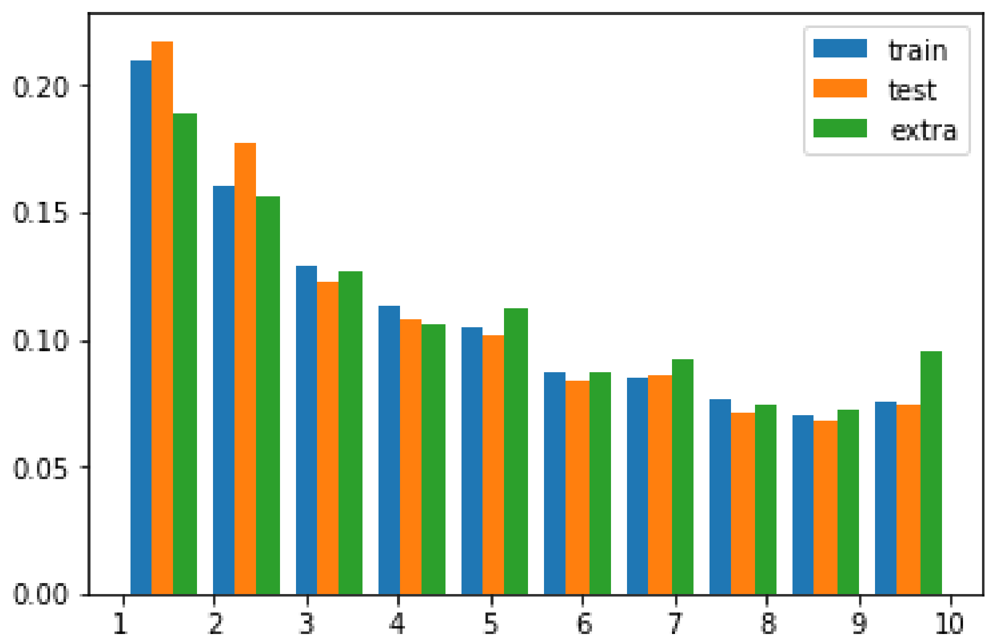
0.1 Data analysis

0.1.1 Class distribution

[19]:
```
print('Train set: {} Test set: {}, Extra: {}'.format(len(Ytr), len(Yte),␣
↪len(Yex)))
plt.hist([Ytr, Yte, Yex], 10, density=True,histtype='bar',label=['train',␣
↪'test', 'extra'])
plt.xticks(range(1,11))
plt.legend()
```

```
Train set: 73257 Test set: 26032, Extra: 531131
```

[19]:
```
<matplotlib.legend.Legend at 0x7f6fe1e88390>
```



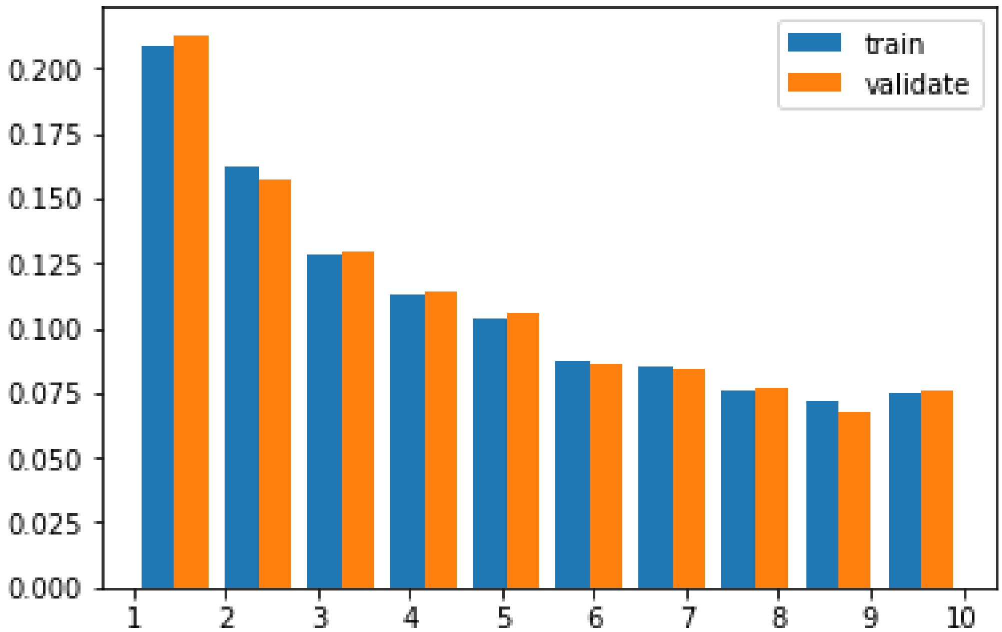
0.1.2 data preprocess

1. shuffle training set
2. split original training set into training and validation set: 7:3
3. grey image and RBG image z-score normalize

[3]:
```
indice = np.random.permutation(len(Ytr))
valSize = int(len(Ytr) * 0.3)
finalX, finalY = Xtr, Ytr
Xtr, Ytr = Xtr[:, :, :, indice], Ytr[indice]
Xva, Yva = Xtr[:,:,:,:valSize], Ytr[:valSize]
Xtr, Ytr = Xtr[:,:,:,valSize:], Ytr[valSize:]
```

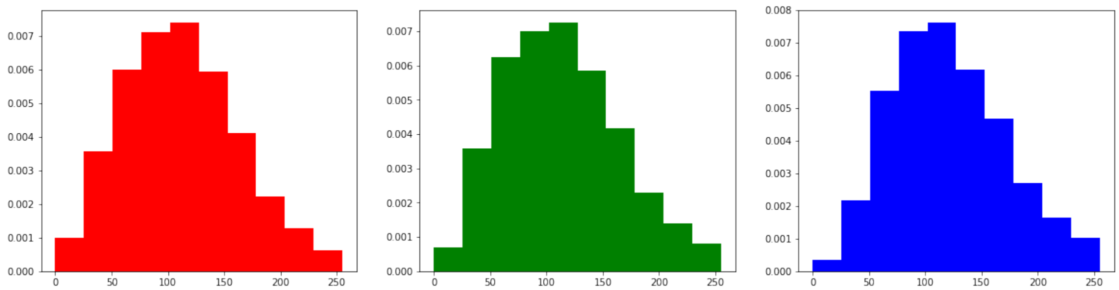
```
print('Train set: {} Val set: {}'.format(len(Ytr), len(Yva)))
plt.hist([Ytr, Yva], 10, density=True,histtype='bar',label=['train',␣
 ↪'validate'])
plt.xticks(range(1,11))
plt.legend()
print(Xtr.shape)
```

```
Train set: 51280 Val set: 21977
(32, 32, 3, 51280)
```



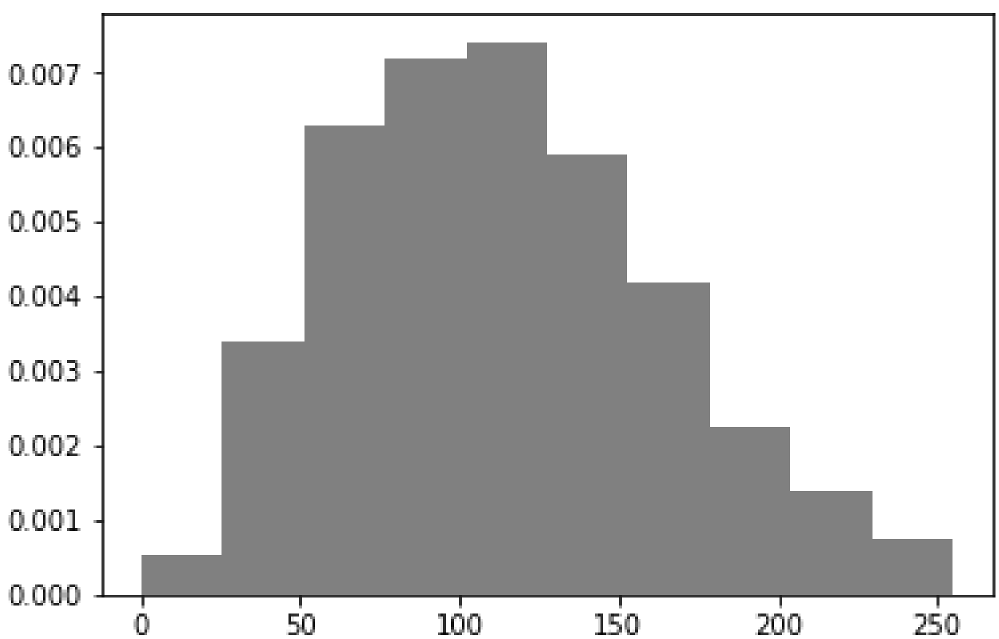
[21]:
```
# data distribution by channel
def describeData(batch, **kwargs):
    batch = batch.flatten()
    m = batch.mean()
    s = batch.std()
    print("mean: {:.4f} std: {:.4f}".format(m, s))
    plt.hist(batch, density=True, **kwargs)
    return m, s
plt.figure(figsize=(20,5))
plt.subplot(131)
stat_red = describeData(Xtr[:,:,0], color='red')
plt.subplot(132)
stat_green = describeData(Xtr[:,:,1], color='green')
plt.subplot(133)
stat_blue = describeData(Xtr[:,:,2], color='blue')
```

```
mean: 111.4888 std: 50.5229
mean: 113.0466 std: 51.2343
mean: 120.4928 std: 50.2066
```



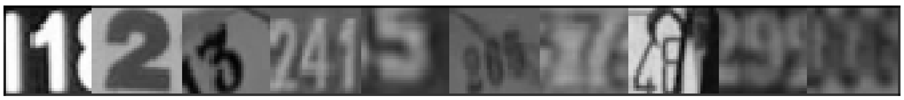
[22]:
```
# grey images
grey_images = np.array([cv2.cvtColor(Xtr[:,:,:,i], cv2.COLOR_RGB2GRAY) for i in␣
 ↪range(Xtr.shape[-1])])
print(grey_images.shape)
plt.figure()
stat_grey = describeData(grey_images, color='grey')
```

```
(51280, 32, 32)
mean: 113.4298 std: 50.2126
```



[23]:
```python
images_origin = []
for i in range(1, 11):
    idx = np.where(Ytr==i)[0][0]
    images_origin.append(grey_images[idx])
images_to_show = np.hstack(images_origin)
plt.figure()
plt.imshow(images_to_show,cmap='gray',interpolation='none')
plt.xticks([]); plt.yticks([])
```

[23]:
```
([], <a list of 0 Text yticklabel objects>)
```



0.2 Handcraft Feature

[5]:
```python
# Feature extractor
def extract_features(image_array):
    '''image_array: numpy array(RBG)'''
    gray = cv2.cvtColor(image_array, cv2.COLOR_RGB2GRAY)
    sift = cv2.xfeatures2d.SIFT_create()
    keypoints, descriptor = sift.detectAndCompute(gray, None)
    return descriptor

print(extract_features(Xtr[:,:,:,0]).shape)
```

```
(7, 128)
```

0.3 Convolutional Network

[4]:
```python
import torchvision.transforms as transforms
class SVHN(torch.utils.data.Dataset):
    def __init__(self, X, Y, transform=None):
        super(SVHN, self).__init__()
        X = np.transpose(X, (3, 2, 0, 1)) # [b, c, h, w]
        self.X = X
        self.Y = np.reshape(Y, (-1,))
        np.place(self.Y, self.Y == 10, 0)
        if transform is None:
            self.transform = transforms.ToTensor()
        else:
            self.transform = transform
    def __getitem__(self, idx):
```

```
        x = Image.fromarray(np.transpose(self.X[idx], (1,2,0)))
        x = self.transform(x)
        return x, int(self.Y[idx])
    def __len__(self):
        return len(self.X)

class ConvNet(nn.Module):
    def __init__(self, ncls=10, input_dim=3):
        super(ConvNet, self).__init__()
        self.ncls=ncls
        self.input_dim=input_dim

        self.model = nn.Sequential(
            nn.Conv2d(input_dim, 6, 5), # 32*32 --> 28 * 28
            nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(6, 16, 5), # 14*14 --> 10 * 10
            nn.ReLU(), nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(16*5*5, 120), nn.ReLU(),
            nn.Linear(120, 84), nn.ReLU(),
            nn.Linear(84, ncls)
        )

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

[5]:

```
def test(model, valSet, batch_size=512, return_pred=False):
    dataloader = torch.utils.data.DataLoader(valSet, batch_size=batch_size,␣
↪shuffle=False, num_workers=4)
    model.train(False)
    correct = 0
    total = 0
    loss = 0
    if return_pred:
        preds = []
    for x, y in dataloader:
        total += len(y)
        x = x.to(device)
        y = y.to(device)
        output = model(x)
        loss += F.cross_entropy(output, y, reduction='sum').item()
        pred_y = torch.argmax(output,1)
        correct += torch.sum(pred_y == y).item()
        if return_pred:
            preds.append(pred_y.cpu().numpy())
    if return_pred:
        preds = np.hstack(preds)
```

```
        return correct/total, loss/total, preds
    return correct / total, loss/total

def train(epoch, model, trainSet, valSet, optimizer, batch_size = 100,␣
↪report_epoch=20, scheduler = None):
    dataloader = torch.utils.data.DataLoader(trainSet, batch_size=batch_size,␣
↪shuffle=True, num_workers=4)
    model.train()
    status = defaultdict(list)
    for e in range(epoch):
        for step, (x, y) in enumerate(dataloader):
            x = x.to(device)
            y = y.to(device)
            out = model(x)
            loss = F.cross_entropy(out, y)
            model.zero_grad()
            loss.backward()
            optimizer.step()
        if scheduler is not None:
            scheduler.step()
        if e % report_epoch == 0:
            accTr, lossTr = test(model, trainSet)
            accVa, lossVa = test(model, valSet)
            print('Epoch {}: accTr: {:.4f}, lossTr: {:.4f}, accVa: {:.4f},␣
↪lossVa: {:.4f}'\
                  .format(e, accTr, lossTr, accVa, lossVa))
            status['accTr'].append(accTr)
            status['lossTr'].append(lossTr)
            status['accVa'].append(accVa)
            status['lossVa'].append(lossVa)
            status['epoch'].append(e)
    return model, status

def plotStatus(stat, l=''):
    plt.subplot(1,2,1)
    plt.plot(stat['epoch'], stat['accTr'], label='train_'+l)
    plt.xlabel('nEpoch')
    plt.ylabel('Accuracy')

    plt.subplot(1,2,2)
    plt.plot(stat['epoch'], stat['lossTr'], label='train_'+l)
    plt.xlabel('nEpoch')
    plt.ylabel('Loss')

    plt.subplot(1,2,1)
    plt.plot(stat['epoch'], stat['accVa'], label='val_'+l)
    plt.legend()
```

```
    plt.subplot(1,2,2)
    plt.plot(stat['epoch'], stat['lossVa'], label='val_'+l)
    plt.legend()
```

[26]:
```
EPOCH=200
LEARNING_RATE = 0.0001
BATCH_SIZE=128
```

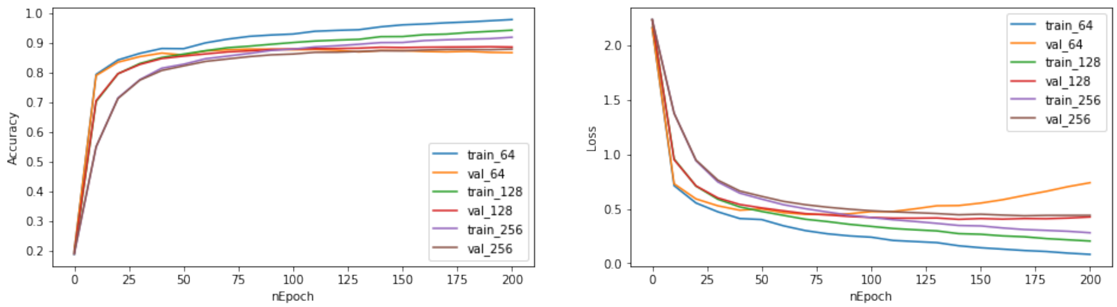
0.3.1 Batch size

[57]:
```
bsize = [64, 128, 256]
transform = transforms.Compose([transforms.Grayscale(), transforms.ToTensor()])
trainset = SVHN(Xtr, Ytr, transform)
valset = SVHN(Xva, Yva, transform)
status = []
for b in bsize:
    net = ConvNet(input_dim=1).to(device)
    opt = torch.optim.Adam(net.parameters(),lr=LEARNING_RATE)
    _, a = train(EPOCH+1, net, trainset, valset, opt, b)
    status.append(a)
```

```
Epoch 0: accTr: 0.1964, lossTr: 2.1624, accVa: 0.1993, lossVa: 2.1605
Epoch 10: accTr: 0.7937, lossTr: 0.7131, accVa: 0.7901, lossVa: 0.7301
Epoch 20: accTr: 0.8425, lossTr: 0.5540, accVa: 0.8349, lossVa: 0.5914
Epoch 30: accTr: 0.8647, lossTr: 0.4728, accVa: 0.8538, lossVa: 0.5281
Epoch 40: accTr: 0.8811, lossTr: 0.4110, accVa: 0.8656, lossVa: 0.4877
Epoch 50: accTr: 0.8804, lossTr: 0.4019, accVa: 0.8591, lossVa: 0.4992
Epoch 60: accTr: 0.9001, lossTr: 0.3443, accVa: 0.8728, lossVa: 0.4653
Epoch 70: accTr: 0.9125, lossTr: 0.3003, accVa: 0.8766, lossVa: 0.4499
Epoch 80: accTr: 0.9219, lossTr: 0.2713, accVa: 0.8790, lossVa: 0.4477
Epoch 90: accTr: 0.9265, lossTr: 0.2519, accVa: 0.8786, lossVa: 0.4550
Epoch 100: accTr: 0.9298, lossTr: 0.2402, accVa: 0.8768, lossVa: 0.4746
Epoch 110: accTr: 0.9392, lossTr: 0.2102, accVa: 0.8777, lossVa: 0.4760
Epoch 120: accTr: 0.9420, lossTr: 0.2000, accVa: 0.8752, lossVa: 0.5003
Epoch 130: accTr: 0.9442, lossTr: 0.1903, accVa: 0.8697, lossVa: 0.5282
Epoch 140: accTr: 0.9541, lossTr: 0.1612, accVa: 0.8743, lossVa: 0.5304
Epoch 150: accTr: 0.9607, lossTr: 0.1432, accVa: 0.8741, lossVa: 0.5529
Epoch 160: accTr: 0.9637, lossTr: 0.1304, accVa: 0.8711, lossVa: 0.5836
Epoch 170: accTr: 0.9676, lossTr: 0.1176, accVa: 0.8698, lossVa: 0.6235
Epoch 180: accTr: 0.9706, lossTr: 0.1086, accVa: 0.8716, lossVa: 0.6610
Epoch 190: accTr: 0.9745, lossTr: 0.0936, accVa: 0.8680, lossVa: 0.7042
Epoch 200: accTr: 0.9785, lossTr: 0.0823, accVa: 0.8673, lossVa: 0.7399
Epoch 0: accTr: 0.1881, lossTr: 2.2361, accVa: 0.1917, lossVa: 2.2336
Epoch 10: accTr: 0.7026, lossTr: 0.9577, accVa: 0.7054, lossVa: 0.9517
Epoch 20: accTr: 0.7961, lossTr: 0.7090, accVa: 0.7963, lossVa: 0.7108
Epoch 30: accTr: 0.8307, lossTr: 0.5882, accVa: 0.8275, lossVa: 0.6004
```

```
Epoch 40: accTr: 0.8509, lossTr: 0.5178, accVa: 0.8472, lossVa: 0.5403
Epoch 50: accTr: 0.8622, lossTr: 0.4762, accVa: 0.8555, lossVa: 0.5087
Epoch 60: accTr: 0.8737, lossTr: 0.4401, accVa: 0.8628, lossVa: 0.4819
Epoch 70: accTr: 0.8837, lossTr: 0.4047, accVa: 0.8695, lossVa: 0.4563
Epoch 80: accTr: 0.8890, lossTr: 0.3834, accVa: 0.8730, lossVa: 0.4445
Epoch 90: accTr: 0.8953, lossTr: 0.3582, accVa: 0.8777, lossVa: 0.4300
Epoch 100: accTr: 0.9012, lossTr: 0.3391, accVa: 0.8795, lossVa: 0.4215
Epoch 110: accTr: 0.9062, lossTr: 0.3199, accVa: 0.8806, lossVa: 0.4148
Epoch 120: accTr: 0.9093, lossTr: 0.3080, accVa: 0.8811, lossVa: 0.4145
Epoch 130: accTr: 0.9118, lossTr: 0.2978, accVa: 0.8822, lossVa: 0.4162
Epoch 140: accTr: 0.9210, lossTr: 0.2726, accVa: 0.8852, lossVa: 0.4037
Epoch 150: accTr: 0.9213, lossTr: 0.2669, accVa: 0.8843, lossVa: 0.4109
Epoch 160: accTr: 0.9277, lossTr: 0.2519, accVa: 0.8854, lossVa: 0.4056
Epoch 170: accTr: 0.9295, lossTr: 0.2437, accVa: 0.8860, lossVa: 0.4114
Epoch 180: accTr: 0.9349, lossTr: 0.2273, accVa: 0.8863, lossVa: 0.4097
Epoch 190: accTr: 0.9388, lossTr: 0.2159, accVa: 0.8870, lossVa: 0.4158
Epoch 200: accTr: 0.9427, lossTr: 0.2047, accVa: 0.8860, lossVa: 0.4254
Epoch 0: accTr: 0.1881, lossTr: 2.2376, accVa: 0.1917, lossVa: 2.2358
Epoch 10: accTr: 0.5496, lossTr: 1.3783, accVa: 0.5504, lossVa: 1.3767
Epoch 20: accTr: 0.7145, lossTr: 0.9413, accVa: 0.7128, lossVa: 0.9479
Epoch 30: accTr: 0.7763, lossTr: 0.7471, accVa: 0.7747, lossVa: 0.7634
Epoch 40: accTr: 0.8148, lossTr: 0.6443, accVa: 0.8072, lossVa: 0.6642
Epoch 50: accTr: 0.8282, lossTr: 0.5897, accVa: 0.8226, lossVa: 0.6148
Epoch 60: accTr: 0.8467, lossTr: 0.5379, accVa: 0.8377, lossVa: 0.5687
Epoch 70: accTr: 0.8555, lossTr: 0.5016, accVa: 0.8458, lossVa: 0.5379
Epoch 80: accTr: 0.8646, lossTr: 0.4711, accVa: 0.8538, lossVa: 0.5153
Epoch 90: accTr: 0.8745, lossTr: 0.4420, accVa: 0.8597, lossVa: 0.4971
Epoch 100: accTr: 0.8790, lossTr: 0.4203, accVa: 0.8625, lossVa: 0.4827
Epoch 110: accTr: 0.8864, lossTr: 0.4002, accVa: 0.8683, lossVa: 0.4728
Epoch 120: accTr: 0.8903, lossTr: 0.3842, accVa: 0.8686, lossVa: 0.4650
Epoch 130: accTr: 0.8953, lossTr: 0.3669, accVa: 0.8711, lossVa: 0.4571
Epoch 140: accTr: 0.9010, lossTr: 0.3476, accVa: 0.8737, lossVa: 0.4459
Epoch 150: accTr: 0.9013, lossTr: 0.3434, accVa: 0.8732, lossVa: 0.4509
Epoch 160: accTr: 0.9077, lossTr: 0.3248, accVa: 0.8759, lossVa: 0.4430
Epoch 170: accTr: 0.9105, lossTr: 0.3102, accVa: 0.8778, lossVa: 0.4364
Epoch 180: accTr: 0.9126, lossTr: 0.3025, accVa: 0.8777, lossVa: 0.4401
Epoch 190: accTr: 0.9143, lossTr: 0.2941, accVa: 0.8771, lossVa: 0.4404
Epoch 200: accTr: 0.9190, lossTr: 0.2807, accVa: 0.8791, lossVa: 0.4406
```

[61]:
```
batchSize_status = status
plt.figure(figsize=(16,4))
for b, stat in zip(bsize, batchSize_status):
    plotStatus(stat, str(b))
```



choose batch_size=128

0.3.2 Learning rate

[66]:
```
lrates = [0.001,0.0005, 0.0001]
transform = transforms.Compose([transforms.Grayscale(), transforms.ToTensor()])
trainset = SVHN(Xtr, Ytr, transform)
valset = SVHN(Xva, Yva, transform)
status = []
for lr in lrates:
    net = ConvNet(input_dim=1).to(device)
    opt = torch.optim.Adam(net.parameters(),lr=lr)
    _, a = train(EPOCH+1, net, trainset, valset, opt, BATCH_SIZE)
    status.append(a)
```

```
Epoch 0: accTr: 0.6104, lossTr: 1.2116, accVa: 0.6120, lossVa: 1.2067
Epoch 20: accTr: 0.9323, lossTr: 0.2275, accVa: 0.8918, lossVa: 0.3877
Epoch 40: accTr: 0.9639, lossTr: 0.1172, accVa: 0.8866, lossVa: 0.5153
Epoch 60: accTr: 0.9811, lossTr: 0.0576, accVa: 0.8832, lossVa: 0.7548
Epoch 80: accTr: 0.9879, lossTr: 0.0363, accVa: 0.8811, lossVa: 0.9806
Epoch 100: accTr: 0.9931, lossTr: 0.0221, accVa: 0.8786, lossVa: 1.2107
Epoch 120: accTr: 0.9954, lossTr: 0.0147, accVa: 0.8785, lossVa: 1.3987
Epoch 140: accTr: 0.9948, lossTr: 0.0155, accVa: 0.8805, lossVa: 1.5421
Epoch 160: accTr: 0.9955, lossTr: 0.0142, accVa: 0.8773, lossVa: 1.6608
Epoch 180: accTr: 0.9942, lossTr: 0.0178, accVa: 0.8789, lossVa: 1.7543
Epoch 200: accTr: 0.9735, lossTr: 0.0950, accVa: 0.8659, lossVa: 1.8075
Epoch 0: accTr: 0.3587, lossTr: 1.8620, accVa: 0.3586, lossVa: 1.8578
Epoch 20: accTr: 0.8881, lossTr: 0.3719, accVa: 0.8696, lossVa: 0.4366
Epoch 40: accTr: 0.9245, lossTr: 0.2511, accVa: 0.8812, lossVa: 0.4177
Epoch 60: accTr: 0.9510, lossTr: 0.1644, accVa: 0.8826, lossVa: 0.4782
Epoch 80: accTr: 0.9610, lossTr: 0.1230, accVa: 0.8723, lossVa: 0.6318
Epoch 100: accTr: 0.9771, lossTr: 0.0743, accVa: 0.8743, lossVa: 0.7800
Epoch 120: accTr: 0.9854, lossTr: 0.0457, accVa: 0.8709, lossVa: 0.9805
Epoch 140: accTr: 0.9908, lossTr: 0.0304, accVa: 0.8683, lossVa: 1.1746
Epoch 160: accTr: 0.9968, lossTr: 0.0142, accVa: 0.8669, lossVa: 1.3361
Epoch 180: accTr: 0.9969, lossTr: 0.0124, accVa: 0.8685, lossVa: 1.5193
```

```
Epoch 200: accTr: 0.9990, lossTr: 0.0056, accVa: 0.8689, lossVa: 1.6440
Epoch 0: accTr: 0.1881, lossTr: 2.2264, accVa: 0.1917, lossVa: 2.2245
Epoch 20: accTr: 0.8065, lossTr: 0.6648, accVa: 0.7986, lossVa: 0.6851
Epoch 40: accTr: 0.8571, lossTr: 0.5041, accVa: 0.8451, lossVa: 0.5439
Epoch 60: accTr: 0.8800, lossTr: 0.4247, accVa: 0.8632, lossVa: 0.4877
Epoch 80: accTr: 0.8928, lossTr: 0.3736, accVa: 0.8710, lossVa: 0.4639
Epoch 100: accTr: 0.9059, lossTr: 0.3280, accVa: 0.8757, lossVa: 0.4488
Epoch 120: accTr: 0.9150, lossTr: 0.2934, accVa: 0.8776, lossVa: 0.4440
Epoch 140: accTr: 0.9225, lossTr: 0.2661, accVa: 0.8787, lossVa: 0.4471
Epoch 160: accTr: 0.9335, lossTr: 0.2321, accVa: 0.8807, lossVa: 0.4545
Epoch 180: accTr: 0.9389, lossTr: 0.2099, accVa: 0.8796, lossVa: 0.4688
Epoch 200: accTr: 0.9488, lossTr: 0.1819, accVa: 0.8792, lossVa: 0.4875
```

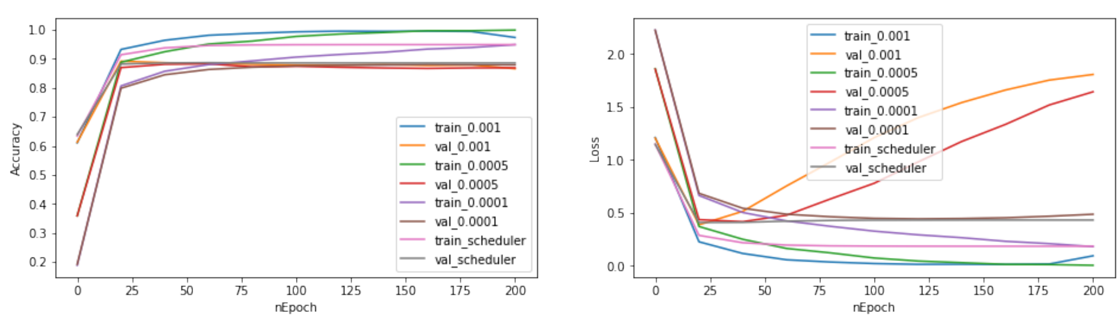
[70]:

```python
net = ConvNet(input_dim=1).to(device)
opt = torch.optim.Adam(net.parameters(),lr=0.001)
scheduler = torch.optim.lr_scheduler.ExponentialLR(opt, 0.95)
net, a = train(EPOCH+1, net, trainset, valset, opt, BATCH_SIZE,
 ↪scheduler=scheduler)
```

```
Epoch 0: accTr: 0.6338, lossTr: 1.1491, accVa: 0.6390, lossVa: 1.1477
Epoch 20: accTr: 0.9140, lossTr: 0.2881, accVa: 0.8819, lossVa: 0.4098
Epoch 40: accTr: 0.9378, lossTr: 0.2184, accVa: 0.8858, lossVa: 0.4112
Epoch 60: accTr: 0.9459, lossTr: 0.1963, accVa: 0.8867, lossVa: 0.4226
Epoch 80: accTr: 0.9480, lossTr: 0.1891, accVa: 0.8858, lossVa: 0.4289
Epoch 100: accTr: 0.9486, lossTr: 0.1868, accVa: 0.8859, lossVa: 0.4320
Epoch 120: accTr: 0.9488, lossTr: 0.1860, accVa: 0.8858, lossVa: 0.4322
Epoch 140: accTr: 0.9489, lossTr: 0.1857, accVa: 0.8858, lossVa: 0.4327
Epoch 160: accTr: 0.9489, lossTr: 0.1856, accVa: 0.8857, lossVa: 0.4328
Epoch 180: accTr: 0.9489, lossTr: 0.1856, accVa: 0.8858, lossVa: 0.4329
Epoch 200: accTr: 0.9489, lossTr: 0.1856, accVa: 0.8858, lossVa: 0.4329
```

[73]:

```python
LR_status = status
plt.figure(figsize=(16,4))
for l, stat in zip(lrates, LR_status):
    plotStatus(stat, str(l))
scheduler_status = a
plotStatus(scheduler_status, 'scheduler')
```



choose lr=0.001 and exponential lr scheduler alpha=0.95

0.3.3 Normalization

[32]:
```
transform = transforms.Compose([transforms.Grayscale(), transforms.ToTensor(),
                               transforms.Normalize((stat_grey[0]/255,),␣
↪(stat_grey[1]/255, ))])
trainset = SVHN(Xtr, Ytr, transform)
valset = SVHN(Xva, Yva, transform)
net = ConvNet(input_dim=1).to(device)
opt = torch.optim.Adam(net.parameters(),lr=0.001)
scheduler = torch.optim.lr_scheduler.ExponentialLR(opt, 0.95)
net, status4 = train(EPOCH, net, trainset, valset, opt, BATCH_SIZE,␣
↪scheduler=scheduler)
```

```
Epoch 0: accTr: 0.7917, lossTr: 0.7056, accVa: 0.7894, lossVa: 0.7228
Epoch 20: accTr: 0.9574, lossTr: 0.1532, accVa: 0.8907, lossVa: 0.4301
Epoch 40: accTr: 0.9818, lossTr: 0.0759, accVa: 0.8871, lossVa: 0.5799
Epoch 60: accTr: 0.9889, lossTr: 0.0525, accVa: 0.8837, lossVa: 0.6834
Epoch 80: accTr: 0.9912, lossTr: 0.0447, accVa: 0.8824, lossVa: 0.7317
Epoch 100: accTr: 0.9920, lossTr: 0.0419, accVa: 0.8819, lossVa: 0.7514
Epoch 120: accTr: 0.9922, lossTr: 0.0410, accVa: 0.8821, lossVa: 0.7589
Epoch 140: accTr: 0.9923, lossTr: 0.0406, accVa: 0.8820, lossVa: 0.7616
Epoch 160: accTr: 0.9924, lossTr: 0.0405, accVa: 0.8821, lossVa: 0.7627
Epoch 180: accTr: 0.9924, lossTr: 0.0405, accVa: 0.8821, lossVa: 0.7630
```

normalization doesn't provide much improvement

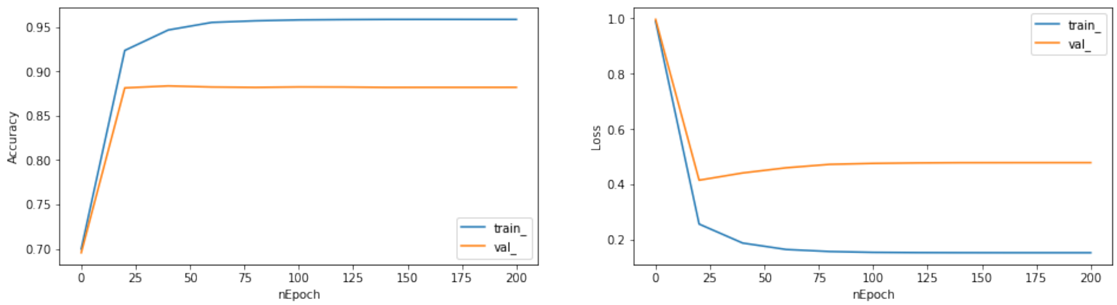
0.3.4 Color information

[30]:
```
net = ConvNet().to(device)
opt = torch.optim.Adam(net.parameters(),lr=0.001)
scheduler = torch.optim.lr_scheduler.ExponentialLR(opt, 0.95)
trainset = SVHN(Xtr, Ytr)
valset = SVHN(Xva, Yva)
net, status1 = train(EPOCH+1, net, trainset, valset, opt, BATCH_SIZE,␣
↪scheduler=scheduler)
```

```
Epoch 0: accTr: 0.6998, lossTr: 0.9899, accVa: 0.6956, lossVa: 0.9968
Epoch 20: accTr: 0.9235, lossTr: 0.2570, accVa: 0.8813, lossVa: 0.4155
Epoch 40: accTr: 0.9465, lossTr: 0.1884, accVa: 0.8835, lossVa: 0.4417
Epoch 60: accTr: 0.9551, lossTr: 0.1651, accVa: 0.8823, lossVa: 0.4604
Epoch 80: accTr: 0.9569, lossTr: 0.1578, accVa: 0.8819, lossVa: 0.4728
Epoch 100: accTr: 0.9579, lossTr: 0.1547, accVa: 0.8824, lossVa: 0.4763
Epoch 120: accTr: 0.9582, lossTr: 0.1538, accVa: 0.8823, lossVa: 0.4781
Epoch 140: accTr: 0.9584, lossTr: 0.1535, accVa: 0.8819, lossVa: 0.4788
Epoch 160: accTr: 0.9585, lossTr: 0.1534, accVa: 0.8819, lossVa: 0.4789
```

```
Epoch 180: accTr: 0.9584, lossTr: 0.1533, accVa: 0.8819, lossVa: 0.4790
Epoch 200: accTr: 0.9584, lossTr: 0.1533, accVa: 0.8820, lossVa: 0.4790
```

[31]:
```
plt.figure(figsize=(16,4))
plotStatus(status1)
```



0.3.5 Dropout

[50]:
```
class ConvNetDropout(nn.Module):
    def __init__(self, ncls=10, input_dim=3):
        super(ConvNetDropout, self).__init__()
        self.ncls=ncls
        self.input_dim=input_dim

        self.model = nn.Sequential(
            nn.Conv2d(input_dim, 6, 5), # 32*32 --> 28 * 28
            nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(6, 16, 5), # 14*14 --> 10 * 10
            nn.ReLU(), nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(16*5*5, 120), nn.ReLU(),nn.Dropout(),
            nn.Linear(120, 84), nn.ReLU(),nn.Dropout(),
            nn.Linear(84, ncls)
        )

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

[52]:
```
transform = transforms.Compose([transforms.Grayscale(), transforms.ToTensor()])
trainset = SVHN(Xtr, Ytr, transform)
valset = SVHN(Xva, Yva, transform)
net = ConvNetDropout(input_dim=1).to(device)
opt = torch.optim.Adam(net.parameters(),lr=0.001)
scheduler = torch.optim.lr_scheduler.ExponentialLR(opt, 0.95)
```

```
net_drop, a = train(201, net, trainset, valset, opt, BATCH_SIZE,␣
↪scheduler=scheduler)
```

```
Epoch 0: accTr: 0.4042, lossTr: 1.7752, accVa: 0.4036, lossVa: 1.7818
Epoch 20: accTr: 0.9259, lossTr: 0.2542, accVa: 0.8838, lossVa: 0.3981
Epoch 40: accTr: 0.9461, lossTr: 0.1914, accVa: 0.8855, lossVa: 0.4272
Epoch 60: accTr: 0.9550, lossTr: 0.1655, accVa: 0.8842, lossVa: 0.4408
Epoch 80: accTr: 0.9576, lossTr: 0.1583, accVa: 0.8834, lossVa: 0.4504
Epoch 100: accTr: 0.9584, lossTr: 0.1556, accVa: 0.8834, lossVa: 0.4538
Epoch 120: accTr: 0.9586, lossTr: 0.1547, accVa: 0.8832, lossVa: 0.4552
Epoch 140: accTr: 0.9588, lossTr: 0.1544, accVa: 0.8831, lossVa: 0.4555
Epoch 160: accTr: 0.9588, lossTr: 0.1542, accVa: 0.8831, lossVa: 0.4557
Epoch 180: accTr: 0.9588, lossTr: 0.1542, accVa: 0.8831, lossVa: 0.4557
Epoch 200: accTr: 0.9588, lossTr: 0.1542, accVa: 0.8831, lossVa: 0.4557
```

0.3.6 better architecture

[7]:
```
def get_resnet18():
    model_ft = torchvision.models.resnet18(pretrained=True)
    model_ft.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,␣
↪bias=False)
    num_ftrs = model_ft.fc.in_features
    model_ft.fc = nn.Linear(num_ftrs, 10)
    model_ft = model_ft.to(device)
    return model_ft
```

[17]:
```
model_ft = get_resnet18()
transform = transforms.Compose([transforms.ToTensor(),
                               transforms.Normalize([0.485, 0.456, 0.406], [0.
↪229, 0.224, 0.225])])
trainset = SVHN(Xtr, Ytr, transform)
valset = SVHN(Xva, Yva, transform)
opt = torch.optim.Adam(model_ft.parameters(),lr=0.001)
scheduler = torch.optim.lr_scheduler.ExponentialLR(opt, 0.95)
model_ft, a = train(41, model_ft, trainset, valset, opt, 128,␣
↪scheduler=scheduler, report_epoch=5)
```

```
Epoch 0: accTr: 0.9245, lossTr: 0.2558, accVa: 0.9108, lossVa: 0.2990
Epoch 5: accTr: 0.9757, lossTr: 0.0934, accVa: 0.9329, lossVa: 0.2441
Epoch 10: accTr: 0.9938, lossTr: 0.0225, accVa: 0.9377, lossVa: 0.2886
Epoch 15: accTr: 0.9978, lossTr: 0.0081, accVa: 0.9387, lossVa: 0.3570
Epoch 20: accTr: 0.9980, lossTr: 0.0069, accVa: 0.9382, lossVa: 0.4137
Epoch 25: accTr: 0.9998, lossTr: 0.0008, accVa: 0.9419, lossVa: 0.4401
Epoch 30: accTr: 0.9999, lossTr: 0.0001, accVa: 0.9443, lossVa: 0.5497
Epoch 35: accTr: 1.0000, lossTr: 0.0000, accVa: 0.9438, lossVa: 0.6606
Epoch 40: accTr: 1.0000, lossTr: 0.0000, accVa: 0.9442, lossVa: 0.7420
```

0.3.7 Semi-supervised learning

[8]:
```
LEARNING_RATE = 0.001
MIN_LR = 1e-4
EPOCH = 40
update_params = {
    't1': 10, 't2': 60, 'af': 0.3, 'n_labels': 4000
}
```

[9]:
```
def unlabeled_weight(epoch, T1, T2, af):
    alpha = 0.0
    if epoch > T1:
        alpha = (epoch-T1) / (T2-T1)*af
        if epoch > T2:
            alpha = af
    return alpha
```

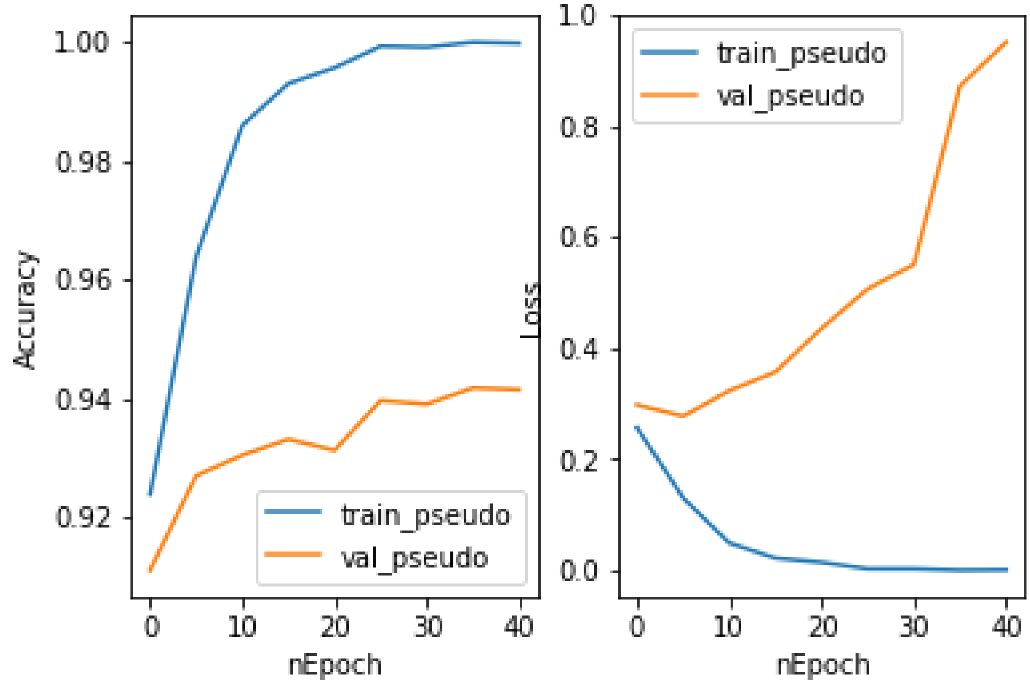
[10]:
```
def pseudo_label_train(epoch, model, trainSet, valSet, exSet, optimizer,␣
↪scheduler = None, batch_size = 100, report_epoch=20):
    dataloader = torch.utils.data.DataLoader(trainSet, batch_size=batch_size,␣
↪shuffle=True, num_workers=4)
    exloader = torch.utils.data.DataLoader(exSet, batch_size=2*batch_size,␣
↪shuffle=True, num_workers=4)
    model.train()
    status = defaultdict(list)
    exiter = iter(exloader)
    for e in range(epoch):
        for step, (x, y) in enumerate(dataloader):
            x = x.to(device)
            y = y.to(device)
            out = model(x)
            loss = F.cross_entropy(out, y)
            try:
                unlabeled, _ = next(exiter)
            except StopIteration:
                exiter = iter(exloader)
                unlabeled, _ = next(exiter)
            unlabeled = unlabeled.to(device)
            out = model(unlabeled)
            with torch.no_grad():
                pseudo_labeled = out.max(1)[1]
            w = unlabeled_weight(e, update_params['t1'], update_params['t2'],␣
↪update_params['af'])
            loss += w * F.cross_entropy(out, pseudo_labeled)
            model.zero_grad()
            loss.backward()
            optimizer.step()
```

```
        if scheduler is not None:
            scheduler.step()
        if e % report_epoch == 0:
            accTr, lossTr = test(model, trainSet)
            accVa, lossVa = test(model, valSet)
            print('Epoch {}: accTr: {:.4f}, lossTr: {:.4f}, accVa: {:.4f},␣
↪lossVa: {:.4f}'\
                  .format(e,accTr, lossTr, accVa, lossVa))
            status['accTr'].append(accTr)
            status['lossTr'].append(lossTr)
            status['accVa'].append(accVa)
            status['lossVa'].append(lossVa)
            status['epoch'].append(e)
    return model, status
```

[13]:
```
semi_net = get_resnet18()
optimizer = torch.optim.Adam(semi_net.parameters(), LEARNING_RATE)
scheduler = torch.optim.lr_scheduler.
↪CosineAnnealingLR(optimizer,T_max=EPOCH,eta_min=MIN_LR)
transform = transforms.Compose([transforms.ToTensor(),
                               transforms.Normalize([0.485, 0.456, 0.406], [0.
↪229, 0.224, 0.225])])
trainset = SVHN(Xtr, Ytr, transform)
valset = SVHN(Xva, Yva, transform)
extraset = SVHN(Xex, Yex, transform)

net, status = pseudo_label_train(EPOCH+1, semi_net, trainset, valset, extraset,␣
↪optimizer, scheduler, batch_size=128, report_epoch=5)
plotStatus(status, 'pseudo')
```

```
Epoch 0: accTr: 0.9239, lossTr: 0.2571, accVa: 0.9111, lossVa: 0.2981
Epoch 5: accTr: 0.9638, lossTr: 0.1299, accVa: 0.9270, lossVa: 0.2777
Epoch 10: accTr: 0.9859, lossTr: 0.0493, accVa: 0.9305, lossVa: 0.3235
Epoch 15: accTr: 0.9930, lossTr: 0.0220, accVa: 0.9332, lossVa: 0.3571
Epoch 20: accTr: 0.9956, lossTr: 0.0145, accVa: 0.9313, lossVa: 0.4357
Epoch 25: accTr: 0.9992, lossTr: 0.0029, accVa: 0.9397, lossVa: 0.5069
Epoch 30: accTr: 0.9991, lossTr: 0.0028, accVa: 0.9391, lossVa: 0.5502
Epoch 35: accTr: 0.9999, lossTr: 0.0004, accVa: 0.9418, lossVa: 0.8721
Epoch 40: accTr: 0.9997, lossTr: 0.0010, accVa: 0.9415, lossVa: 0.9509
```



1 model evaluation

1.1 simple model

[70]:

```
transform = transforms.Compose([transforms.Grayscale(), transforms.ToTensor()])
trainset = SVHN(finalX, finalY, transform)
valset = SVHN(Xva, Yva, transform)
net = ConvNet(input_dim=1).to(device)
opt = torch.optim.Adam(net.parameters(),lr=0.001)
scheduler = torch.optim.lr_scheduler.ExponentialLR(opt, 0.95)
net1, a = train(201, net, trainset, valset, opt, BATCH_SIZE,␣
 ↪scheduler=scheduler)
```

```
Epoch 0: accTr: 0.7175, lossTr: 0.9226, accVa: 0.7159, lossVa: 0.9262
Epoch 20: accTr: 0.9207, lossTr: 0.2715, accVa: 0.9206, lossVa: 0.2766
Epoch 40: accTr: 0.9393, lossTr: 0.2162, accVa: 0.9378, lossVa: 0.2203
Epoch 60: accTr: 0.9449, lossTr: 0.1981, accVa: 0.9440, lossVa: 0.2017
Epoch 80: accTr: 0.9473, lossTr: 0.1917, accVa: 0.9469, lossVa: 0.1951
Epoch 100: accTr: 0.9480, lossTr: 0.1896, accVa: 0.9477, lossVa: 0.1929
Epoch 120: accTr: 0.9481, lossTr: 0.1888, accVa: 0.9478, lossVa: 0.1920
Epoch 140: accTr: 0.9482, lossTr: 0.1885, accVa: 0.9478, lossVa: 0.1917
Epoch 160: accTr: 0.9483, lossTr: 0.1884, accVa: 0.9479, lossVa: 0.1916
Epoch 180: accTr: 0.9483, lossTr: 0.1884, accVa: 0.9479, lossVa: 0.1916
Epoch 200: accTr: 0.9483, lossTr: 0.1884, accVa: 0.9479, lossVa: 0.1915
```

[71]:
```
testset = SVHN(Xte, Yte, transform)
accuracy, _, preds = test(net1, testset, batch_size=512, return_pred=True)
print(accuracy)
```

```
0.884219422249539
```

1.2 Simple model with dropout

[73]:
```
transform = transforms.Compose([transforms.Grayscale(), transforms.ToTensor()])
trainset = SVHN(finalX, finalY, transform)
valset = SVHN(Xva, Yva, transform)
opt = torch.optim.Adam(net_drop.parameters(),lr=0.001)
scheduler = torch.optim.lr_scheduler.ExponentialLR(opt, 0.95)
net_drop, a = train(101, net_drop, trainset, valset, opt, BATCH_SIZE,␣
↪scheduler=scheduler)
```

```
Epoch 0: accTr: 0.8819, lossTr: 0.3942, accVa: 0.8798, lossVa: 0.3989
Epoch 20: accTr: 0.9650, lossTr: 0.1264, accVa: 0.9639, lossVa: 0.1264
Epoch 40: accTr: 0.9812, lossTr: 0.0787, accVa: 0.9809, lossVa: 0.0771
Epoch 60: accTr: 0.9860, lossTr: 0.0638, accVa: 0.9856, lossVa: 0.0619
Epoch 80: accTr: 0.9878, lossTr: 0.0581, accVa: 0.9877, lossVa: 0.0562
Epoch 100: accTr: 0.9884, lossTr: 0.0563, accVa: 0.9886, lossVa: 0.0542
```

[75]:
```
accuracy_drop, _, preds_drop = test(net_drop, testset, batch_size=512,␣
↪return_pred=True)
print(accuracy_drop)
```

```
0.8725030731407498
```

1.3 resnet18

[18]:
```
transform = transforms.Compose([transforms.ToTensor(),
                               transforms.Normalize([0.485, 0.456, 0.406], [0.
↪229, 0.224, 0.225])])
testset = SVHN(Xte, Yte, transform)
accuracy_res, _, preds_res = test(model_ft, testset, batch_size=512,␣
↪return_pred=True)
print(accuracy_res)
```

```
0.9442224953902889
```

[19]:
```
accuracy_semi, _, preds_semi = test(semi_net, testset, batch_size=512,␣
↪return_pred=True)
print(accuracy_semi)
```

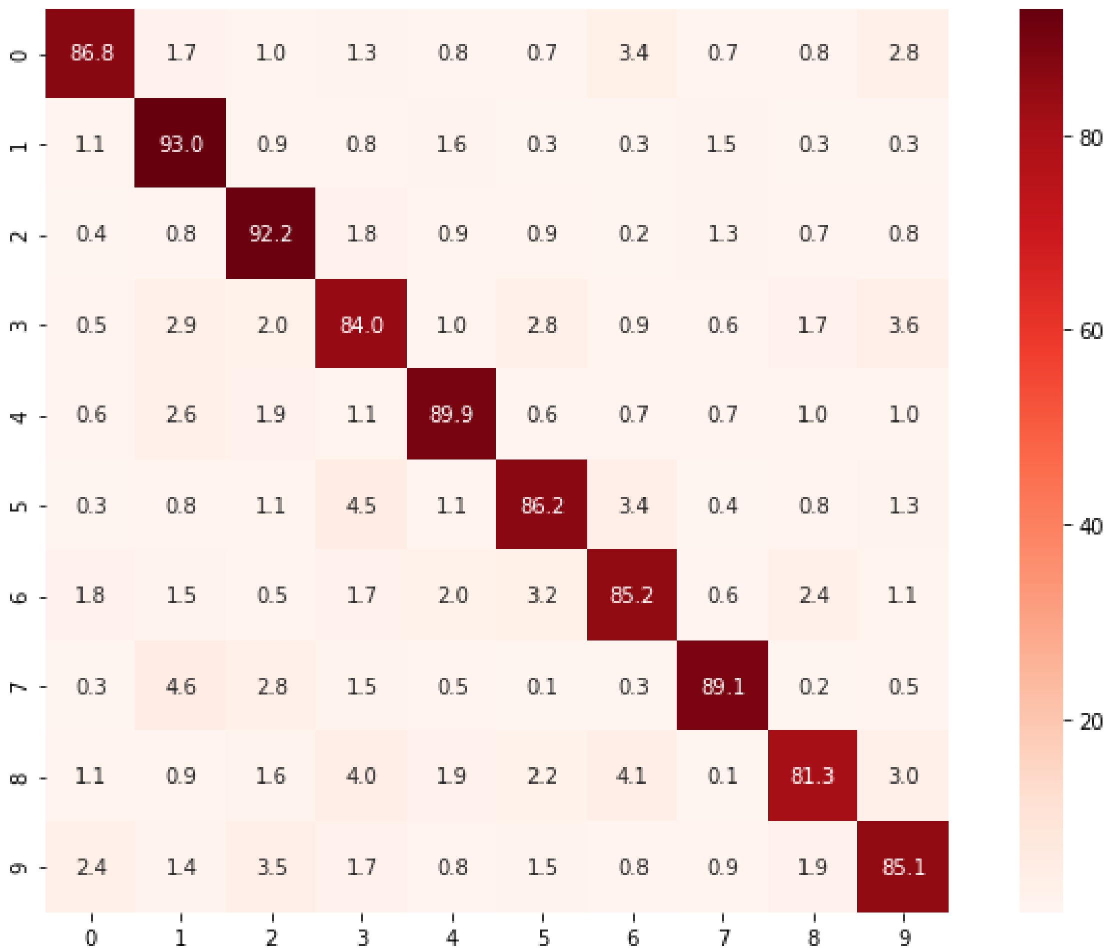
```
0.9405731407498463
```

1.4 result visualization

[21]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

[80]:
```
plt.figure(figsize=(12, 8))
cm = confusion_matrix(y_true=Yte, y_pred=preds)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] * 100.0
sns.heatmap(cm, annot=True, cmap='Reds', fmt='.1f', square=True)
```

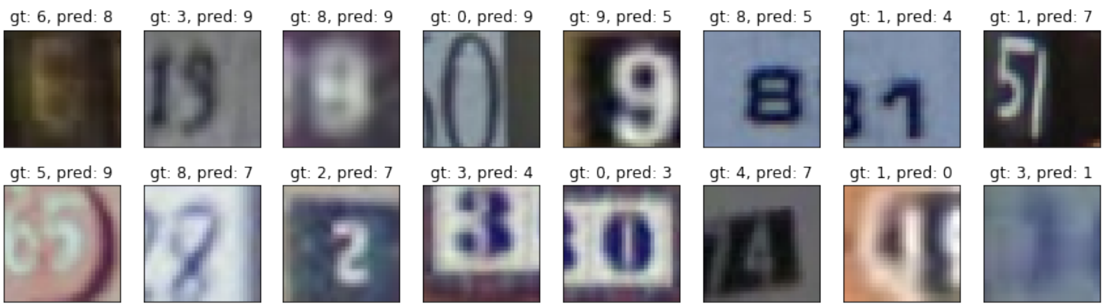
[80]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6fe2b8d278>
```



[103]:
```
mistake = np.where(Yte.flatten() != preds)[0][:19]
print(mistake.shape)
mistakeX = Xte[:,:,:,mistake]
mistakeY = Yte[mistake]
mistake_preds = preds[mistake]
```

```python
fig, axes = plt.subplots(2, 8)
fig.set_size_inches((15,4))
for i, ax in enumerate(axes.flat):
    title = "gt: {}, pred: {}".format(mistakeY[i][0], mistake_preds[i])
    ax.imshow(mistakeX[:,:,:, i])
    ax.set_title(title)
    ax.set_xticks([])
    ax.set_yticks([])
```

```
(19,)
```



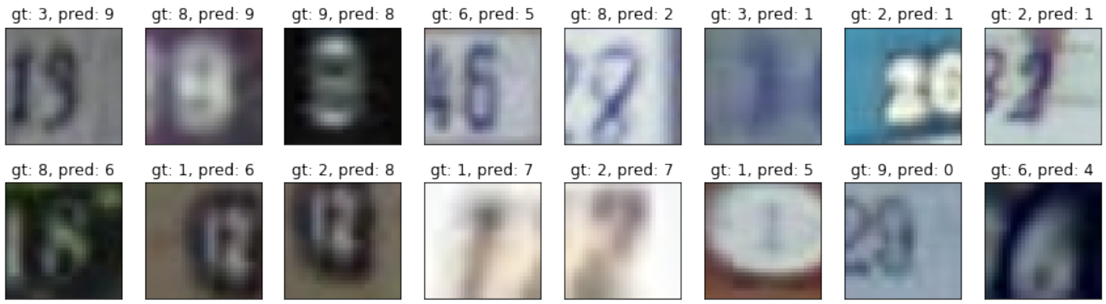
[22]:
```python
plt.figure(figsize=(12, 8))
cm = confusion_matrix(y_true=Yte, y_pred=preds_res)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] * 100.0
sns.heatmap(cm, annot=True, cmap='Reds', fmt='.1f', square=True)
```

[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7feaae0f1128>
```

[23]:

```
mistake = np.where(Yte.flatten() != preds_res)[0][:19]
print(mistake.shape)
mistakeX = Xte[:,:,:,mistake]
mistakeY = Yte[mistake]
mistake_preds = preds_res[mistake]
fig, axes = plt.subplots(2, 8)
fig.set_size_inches((15,4))
for i, ax in enumerate(axes.flat):
    title = "gt: {}, pred: {}".format(mistakeY[i][0], mistake_preds[i])
    ax.imshow(mistakeX[:,:,:, i])
    ax.set_title(title)
    ax.set_xticks([])
    ax.set_yticks([])
```

```
(19,)
```



2 Summary

1. Influence of batch size
2. Influence of learning rate
3. Influence of normalization
4. Influence of color information
5. Influence of dropout
6. Influence of auxiliary data without labels (semi-supervised learning)
7. Influence of better architecture

[ ]:
```

```