Machine Learning Project 08

December 11, 2019

[23]:
```
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader

import torchvision.transforms as transforms
from torch.autograd import Variable
import matplotlib.pyplot as plt
plt.rcParams.update({'figure.max_open_warning': 0})
import torchvision
```

[24]:
```
class numpyDataset(Dataset):
    def __init__(self, data, transform=None):
        self.data = torch.from_numpy(data).float()
        self.transform = transform

    def __getitem__(self, index):
        x = self.data[index]

        if self.transform:
            x = self.transform(x)

        return x

    def __len__(self):
        return len(self.data)
```

[25]:
```
def add_noise(img):
    mu=0
    sigma_list=[0.01,0.02,0.03,0.04]
    sigma= np.random.choice(sigma_list, 1, replace=True, p=None)
    noise= np.random.normal(mu, sigma[0])
    noisy_img = img + noise
    return noisy_img
```

[186]:
```
import torch.nn as nn
import torch.nn.functional as F

class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, padding=1)
        self.conv3 = nn.Conv2d(16, 8, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        self.t_conv1 = nn.ConvTranspose2d(8, 8, 2, stride=2)
        self.t_conv3 = nn.ConvTranspose2d(8, 16, 2, stride=2)
        self.conv_out = nn.Conv2d(16, 1, 3, padding=1)


    def forward(self, x):
        ## encode ##
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = self.pool(x)

        ## decode ##
        x = F.relu(self.t_conv1(x))
        x = F.relu(self.t_conv3(x))
        x = F.sigmoid(self.conv_out(x))

        return x
```

[ ]:
```
NUM_EPOCH       = 50
transform       = transforms.Compose([transforms.ToPILImage(),transforms.
 ↪Grayscale(num_output_channels=1),transforms.ToTensor(),])

batch_size = 120
learning_rate = 0.001


# for training
traindata       = np.load('train.npy')
traindataset    = numpyDataset(traindata, transform)
trainloader     = DataLoader(traindataset, batch_size=batch_size, shuffle=True,␣
 ↪num_workers=0)


device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = autoencoder().to(device)
criterion = nn.MSELoss()
```

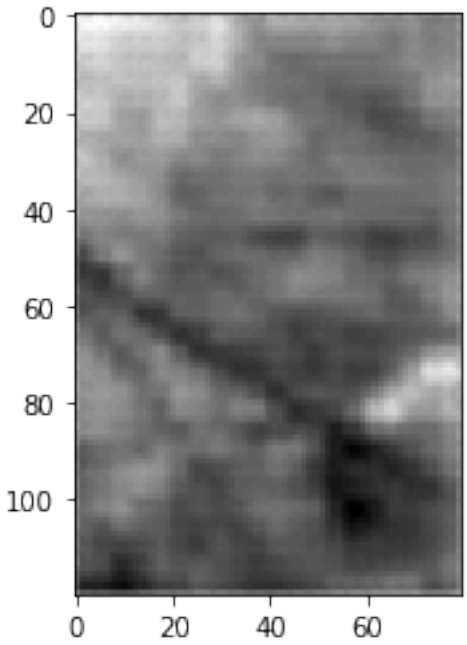
```
#optimizer = torch.optim.SGD( model.parameters(), lr=learning_rate,  momentum=0.
↪007)
optimizer = torch.optim.Adam( model.parameters(), lr=learning_rate,␣
↪weight_decay=0.0001)


loss_train_mean        = np.zeros(NUM_EPOCH)
loss_train_std         = np.zeros(NUM_EPOCH)



for epoch in range(NUM_EPOCH):
    loss_list = []

    for batch_idx, data in enumerate(trainloader):
        imagearr   = data
        noisy_img = add_noise(imagearr)


        optimizer.zero_grad()
        # ==================forward====================
        output = model(noisy_img)
        loss = criterion(output, imagearr)
        # ==================backward====================
        loss.backward()
        optimizer.step()

        loss_list.append(loss.item())


    loss_train_mean[epoch]     = np.mean(loss_list)
    print("""[EPOCH %5d ] LOSS     :(TRAIN) %3.
↪10f"""%(epoch,loss_train_mean[epoch]))
    loss_train_std[epoch]      = np.std(loss_list)
```

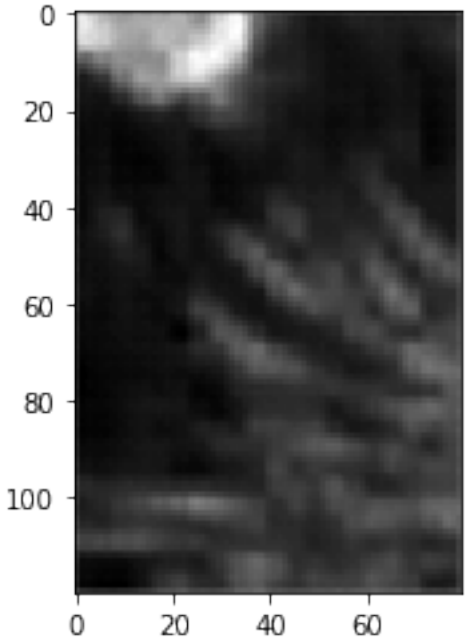
```
[EPOCH     0 ] LOSS     :(TRAIN) 0.0192138163
[EPOCH     1 ] LOSS     :(TRAIN) 0.0153734231
[EPOCH     2 ] LOSS     :(TRAIN) 0.0048541913
[EPOCH     3 ] LOSS     :(TRAIN) 0.0038991128
[EPOCH     4 ] LOSS     :(TRAIN) 0.0031227068
```

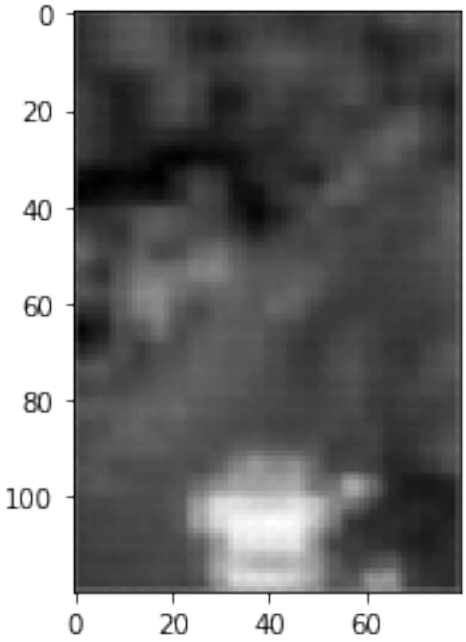
[165]:
```
print(output.shape)
#output_s = output.reshape(50,1,120,80)
to_img  = transforms.ToPILImage()
for i in range(10):
    show  = to_img(output[i])
```

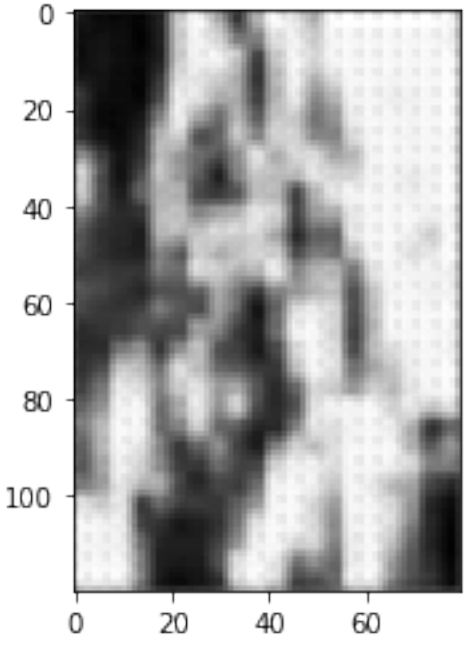
```
    fig     = plt.figure()
    ax      = fig.add_subplot(1, 1, 1)
    ax.imshow(show, cmap='gray')
```

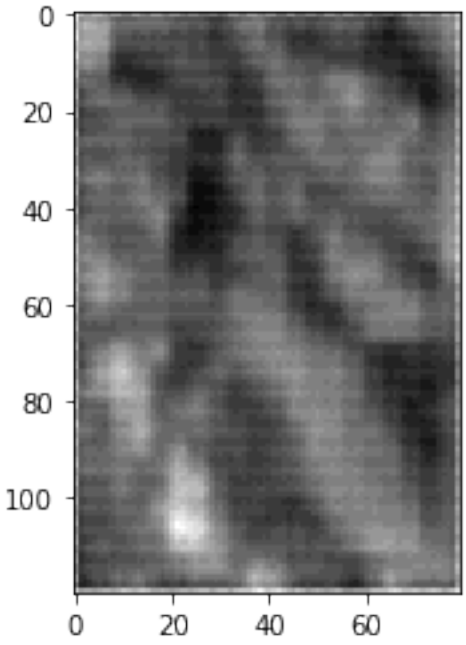
```
torch.Size([120, 1, 120, 80])
```

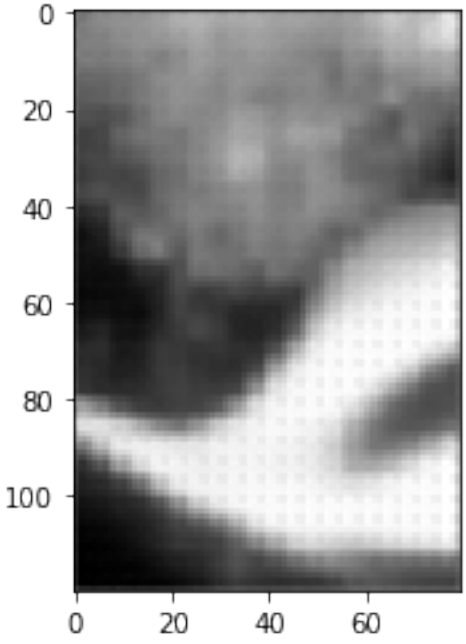


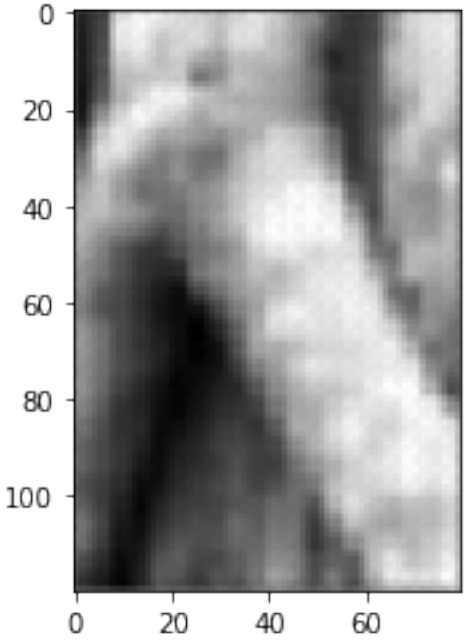


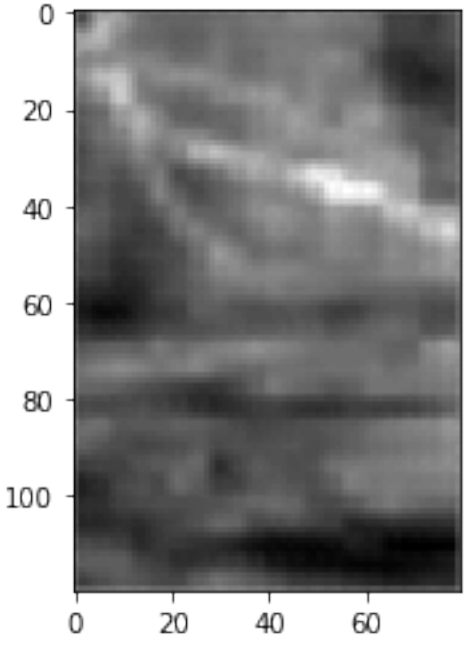

0
20
40
60
80
100
0
20
40
60

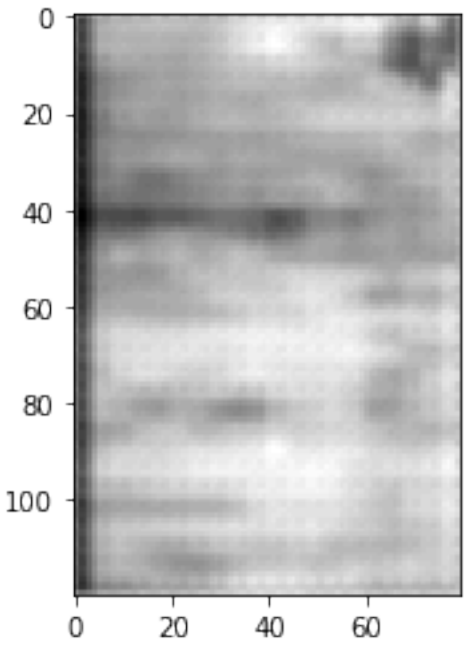

0
20
40
60
80
100
0
20
40
60

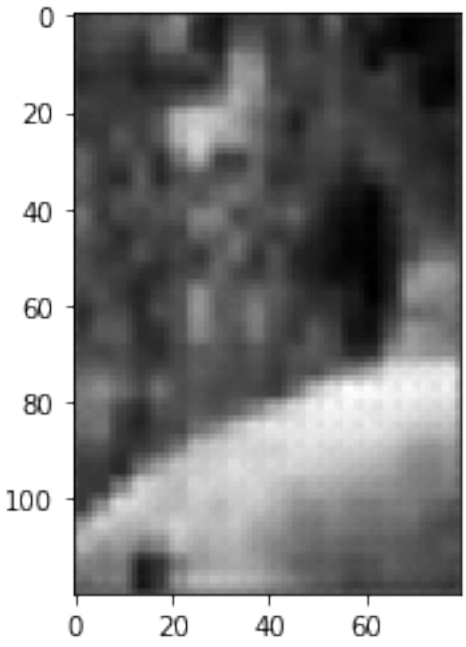

0
20
40
60
80
100
0
20
40
60


0
20
40
60
80
100
0
20
40
60


0
20
40
60
80
100
0
20
40
60

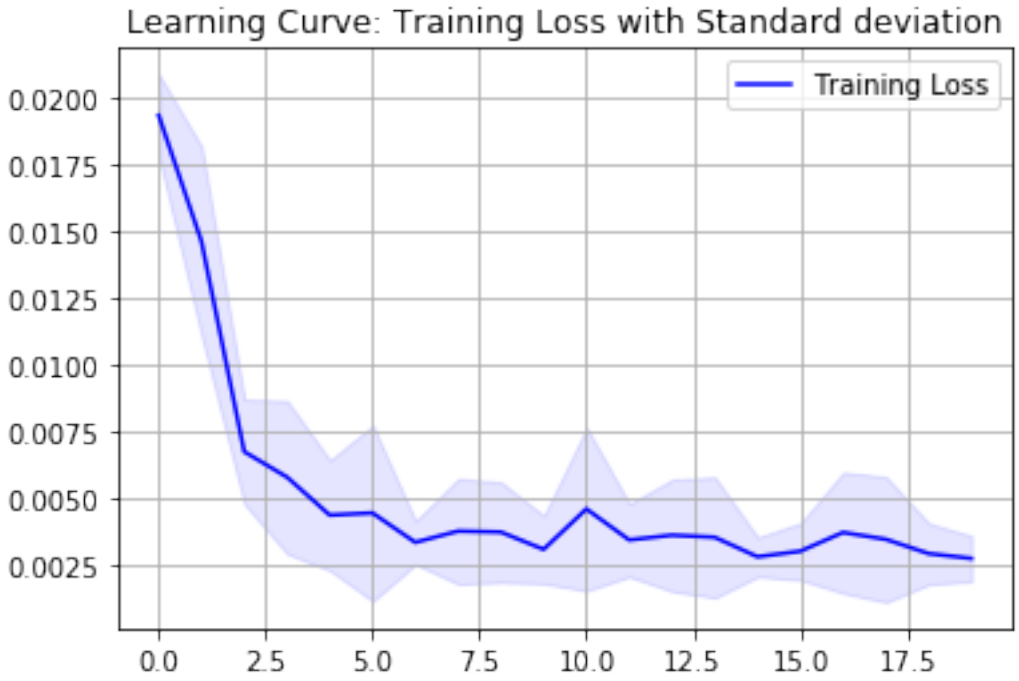

0
20
40
60
80
100
0
20
40
60





[178]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit
from sklearn.svm import SVC
```

```
from sklearn.datasets import load_digits


train_sizes=np.array(range(NUM_EPOCH))


plt.title("Learning Curve: Training Loss with Standard deviation")
plt.grid()
#plt.ylim([0.015,0.027])
#print(loss_train_std)
plt.fill_between(train_sizes,loss_train_mean  - loss_train_std, loss_train_mean␣
 ↪+ loss_train_std, alpha=0.1,color="b")
plt.plot(train_sizes, loss_train_mean, color="b",label="Training Loss")
plt.legend(loc='upper right')

plt.show()
```



[179]:
```
# for testing
testdata        = np.load('test.npy')
testdataset     = numpyDataset(testdata, transform)
testloader      = DataLoader(testdataset, batch_size=40, shuffle=False,␣
 ↪num_workers=0)

result_for_submit = None    # this is for submit file
```

```
for batch_idx, data in enumerate(testloader):

#    data = data.view(data.size(0), -1)
    result_of_test = model(data)

    if batch_idx == 0:
        result_for_submit = result_of_test
    else:
        try:
            result_for_submit = torch.cat([result_for_submit, result_of_test],␣
 ↪dim=0)

        except RuntimeError:
            transposed = torch.transpose(result_of_test, 2, 3)
            result_for_submit = torch.cat([result_for_submit, transposed],␣
 ↪dim=0)
print(result_of_test.shape)
print(result_for_submit.shape)
# the submit_file.shape must be (400,1,120,80)
#submit_file = result_for_submit.detach().numpy()
#np.save('your_name.npy', submit_file)
```

```
torch.Size([40, 1, 120, 80])
torch.Size([400, 1, 120, 80])
```

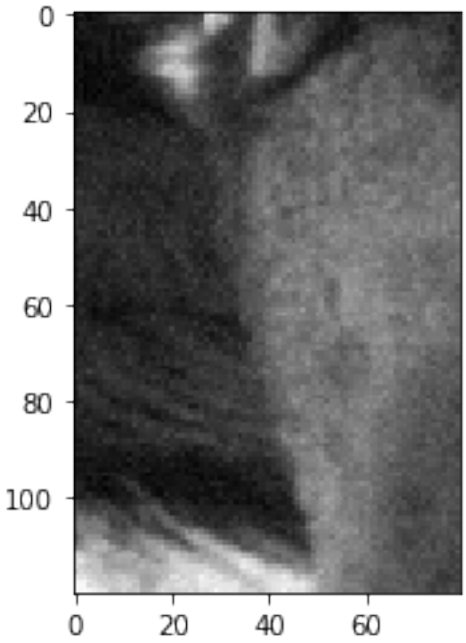
[173]:
```
testdata       = np.load('test.npy')
print(testdata)
print(result_for_submit)
plt.imshow(testdata[8], cmap='gray')
```

```
[[[0.25096598 0.2578184  0.24734384 … 0.40209818 0.40234357 0.41850954]
  [0.26115713 0.25937217 0.25825202 … 0.37769216 0.40756923 0.3883009 ]
  [0.25773162 0.27895033 0.26262528 … 0.40319365 0.40665805 0.40004042]
  …
  [0.32804954 0.32524568 0.34652016 … 0.36129624 0.39818776 0.4010844 ]
  [0.30458072 0.35293165 0.36487103 … 0.39366654 0.38533837 0.41148585]
  [0.31277347 0.3679903  0.40805683 … 0.37274158 0.37808508 0.3894653 ]]

 [[0.39479133 0.41145387 0.39531147 … 0.37814227 0.3823619  0.39712438]
  [0.40215704 0.414894   0.41485384 … 0.37735587 0.38867232 0.38689512]
  [0.40905064 0.40726143 0.40808472 … 0.39708203 0.4370551  0.41711485]
  …
  [0.38842326 0.36434427 0.34737778 … 0.39625934 0.36913863 0.36598665]
  [0.38612643 0.3632215  0.34981066 … 0.38379535 0.3747496  0.34133083]
  [0.41333926 0.39374337 0.36763892 … 0.3739893  0.40534595 0.36729744]]
```

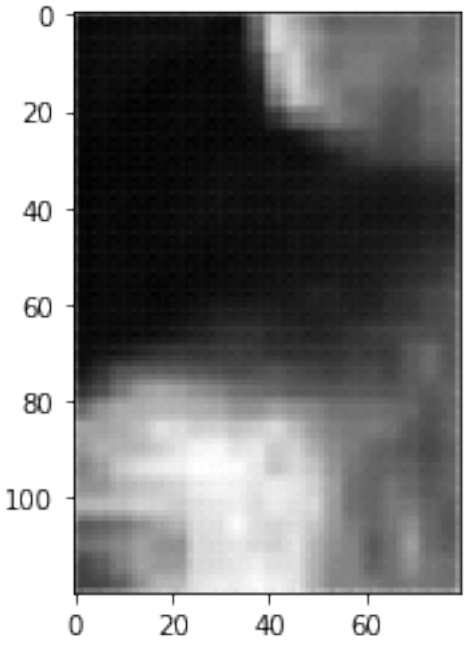
```
 [[0.41999075 0.41492543 0.39151874 … 0.2849076  0.3045111  0.345031  ]
  [0.40909025 0.39743546 0.41439608 … 0.3129584  0.33186692 0.33702153]
  [0.4079595  0.3947699  0.42385712 … 0.36053747 0.32750466 0.31053036]
  …
  [0.36163223 0.36892584 0.34765732 … 0.34643045 0.32289395 0.2873205 ]
  [0.36198923 0.35590383 0.33991244 … 0.35983855 0.34097502 0.32595918]
  [0.37538522 0.3832508  0.34591642 … 0.3737246  0.35645208 0.33843648]]

 …

 [[0.33813348 0.32374915 0.35279256 … 0.8033322  0.81650317 0.7580648 ]
  [0.25532717 0.23402254 0.23540476 … 0.84540534 0.8140267  0.6763711 ]
  [0.46140218 0.32514045 0.337802   … 0.8058277  0.81600994 0.88138175]
  …
  [0.19534433 0.269856   0.28706512 … 0.7646051  0.7142312  0.8472966 ]
  [0.48904485 0.21010774 0.23730643 … 0.75831354 0.7683295  0.6906784 ]
  [0.48410806 0.24042636 0.22724256 … 0.7048377  0.8840695  0.8101086 ]]

 [[0.7239136  0.74301165 0.7723005  … 0.5042487  0.36713988 0.49426988]
  [0.78021216 0.73244065 0.7068966  … 0.39891937 0.4606171  0.37879068]
  [0.7896176  0.7428718  0.7425291  … 0.49449518 0.49903578 0.46546167]
  …
  [0.78338647 0.80402315 0.7911368  … 0.8464895  0.7660241  0.749958  ]
  [0.78188026 0.78141403 0.76783854 … 0.7384347  0.8016539  0.76620156]
  [0.78447133 0.7706025  0.7676616  … 0.7393823  0.7724798  0.80585897]]

 [[0.42806453 0.42900765 0.428942   … 0.44377473 0.4218532  0.4315002 ]
  [0.45699078 0.4558399  0.4727716  … 0.505474   0.3232876  0.47982195]
  [0.4556573  0.46567237 0.52334607 … 0.43715724 0.41832465 0.3936198 ]
  …
  [0.7432134  0.8212377  0.77080256 … 0.5177118  0.56495464 0.6161549 ]
  [0.7753917  0.841381   0.73584366 … 0.6055991  0.60063356 0.6363352 ]
  [0.7846752  0.7664136  0.7490774  … 0.56434155 0.6118244  0.6277421 ]]]
tensor([[[[0.3621, 0.3121, 0.3131,  …, 0.4214, 0.4111, 0.4361],
          [0.3198, 0.2523, 0.2570,  …, 0.3959, 0.3942, 0.4158],
          [0.3101, 0.2510, 0.2547,  …, 0.3970, 0.3972, 0.4231],
          …,
          [0.3726, 0.3159, 0.3193,  …, 0.3794, 0.3752, 0.4017],
          [0.3648, 0.3207, 0.3225,  …, 0.3749, 0.3711, 0.4032],
          [0.4161, 0.3796, 0.3831,  …, 0.4134, 0.4164, 0.4331]]],


        [[[0.4430, 0.4132, 0.4064,  …, 0.4163, 0.4077, 0.4347],
          [0.4281, 0.3886, 0.3912,  …, 0.3915, 0.3917, 0.4150],
          [0.4182, 0.3941, 0.3985,  …, 0.3956, 0.3976, 0.4241],
          …,
          [0.3856, 0.3325, 0.3355,  …, 0.3644, 0.3622, 0.3935],
          [0.3759, 0.3359, 0.3378,  …, 0.3618, 0.3594, 0.3954],
```

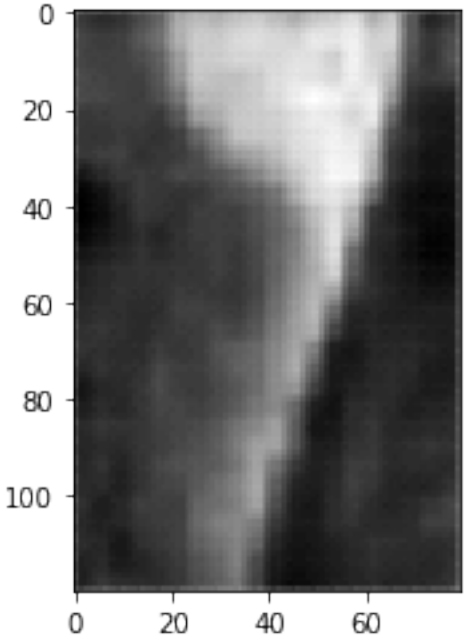
```
          [0.4224, 0.3895, 0.3944,  …, 0.4051, 0.4084, 0.4274]]],


        [[[0.4390, 0.4080, 0.4014,  …, 0.3639, 0.3592, 0.4050],
          [0.4229, 0.3815, 0.3841,  …, 0.3206, 0.3226, 0.3670],
          [0.4132, 0.3871, 0.3912,  …, 0.3249, 0.3254, 0.3710],
          …,
          [0.3917, 0.3402, 0.3427,  …, 0.3284, 0.3268, 0.3684],
          [0.3811, 0.3427, 0.3442,  …, 0.3270, 0.3238, 0.3683],
          [0.4255, 0.3939, 0.3988,  …, 0.3809, 0.3815, 0.4067]]],


        …,


        [[[0.3920, 0.3484, 0.3455,  …, 0.7013, 0.6633, 0.5857],
          [0.3611, 0.3012, 0.3046,  …, 0.7757, 0.7558, 0.6590],
          [0.3530, 0.3055, 0.3071,  …, 0.7734, 0.7716, 0.6932],
          …,
          [0.3021, 0.2305, 0.2343,  …, 0.7378, 0.7158, 0.6303],
          [0.2908, 0.2275, 0.2302,  …, 0.7091, 0.7008, 0.6365],
          [0.3626, 0.3054, 0.3049,  …, 0.6326, 0.6429, 0.6039]]],


        [[[0.6196, 0.6422, 0.6239,  …, 0.4621, 0.4487, 0.4582],
          [0.6616, 0.7042, 0.6997,  …, 0.4536, 0.4498, 0.4528],
          [0.6455, 0.7072, 0.7157,  …, 0.4549, 0.4557, 0.4643],
          …,
          [0.6807, 0.7250, 0.7075,  …, 0.7608, 0.7373, 0.6445],
          [0.6438, 0.6996, 0.6957,  …, 0.7388, 0.7295, 0.6571],
          [0.5819, 0.6271, 0.6410,  …, 0.6582, 0.6683, 0.6229]]],


        [[[0.4743, 0.4531, 0.4437,  …, 0.4420, 0.4305, 0.4475],
          [0.4700, 0.4449, 0.4461,  …, 0.4268, 0.4245, 0.4362],
          [0.4590, 0.4508, 0.4565,  …, 0.4300, 0.4311, 0.4472],
          …,
          [0.7035, 0.7518, 0.7306,  …, 0.5694, 0.5604, 0.5263],
          [0.6684, 0.7299, 0.7254,  …, 0.5491, 0.5453, 0.5267],
          [0.5994, 0.6516, 0.6665,  …, 0.5252, 0.5340, 0.5224]]]],
       grad_fn=<CatBackward>)
```

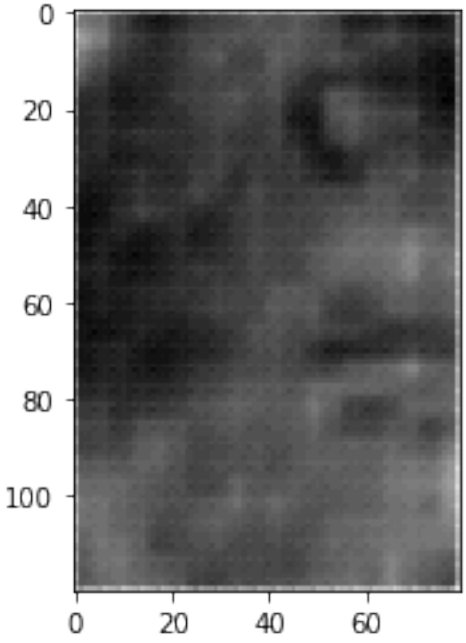
[173]: `<matplotlib.image.AxesImage at 0x2c196e80ac8>`

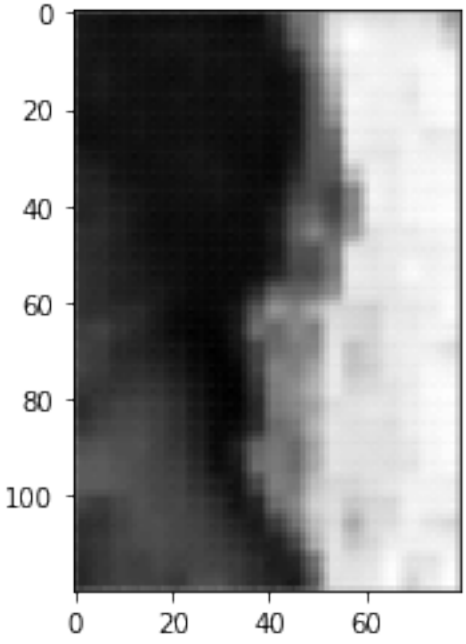


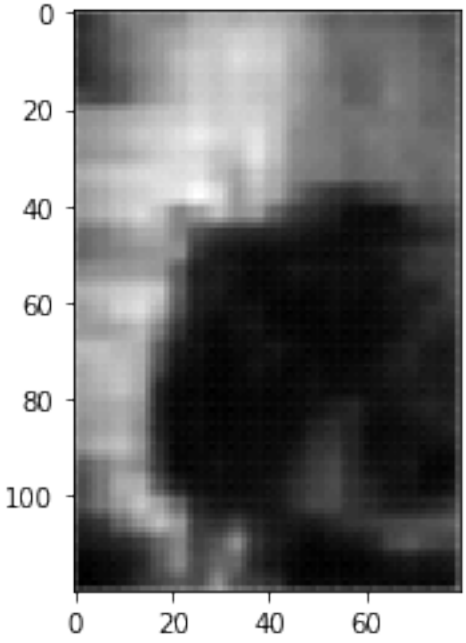
[180]:
```
for i in range(10):
#    result_for_submit=result_for_submit.reshape(400,1,120,80)
    to_img  = transforms.ToPILImage()
    show    = to_img(result_for_submit[i])
    fig     = plt.figure()
    ax      = fig.add_subplot(1, 1, 1)
    ax.imshow(show, cmap='gray')
```

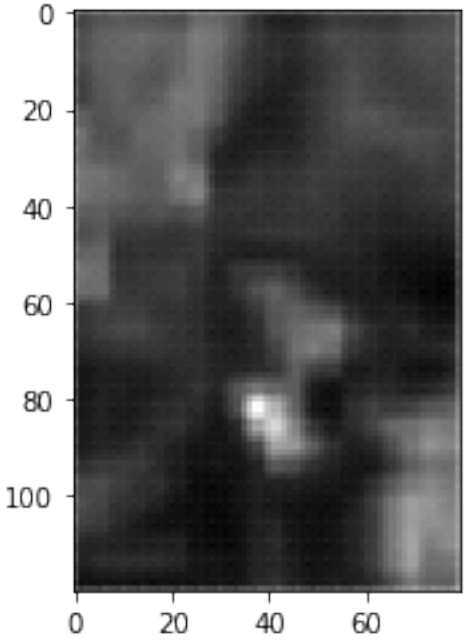

0
20
40
60
80
100
0
20
40
60

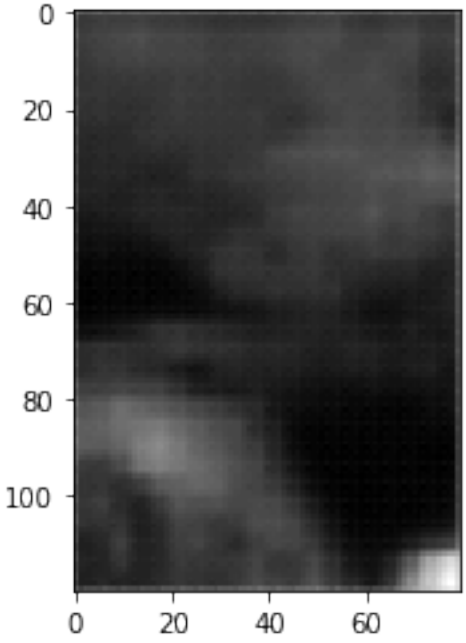

0
20
40
60
80
100
0
20
40
60

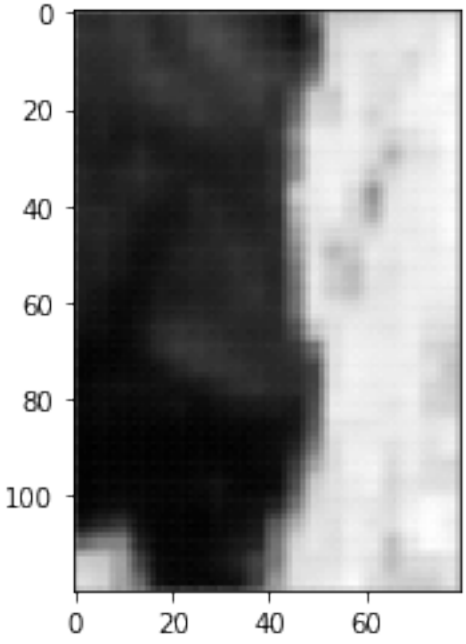

0
20
40
60
80
100
0
20
40
60

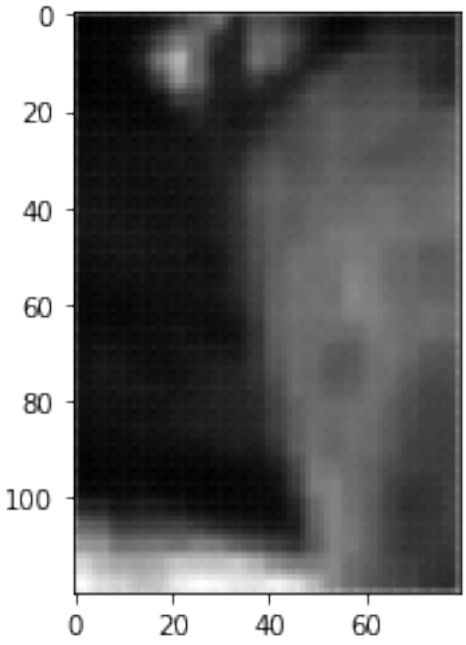

0
20
40
60
80
100
0
20
40
60

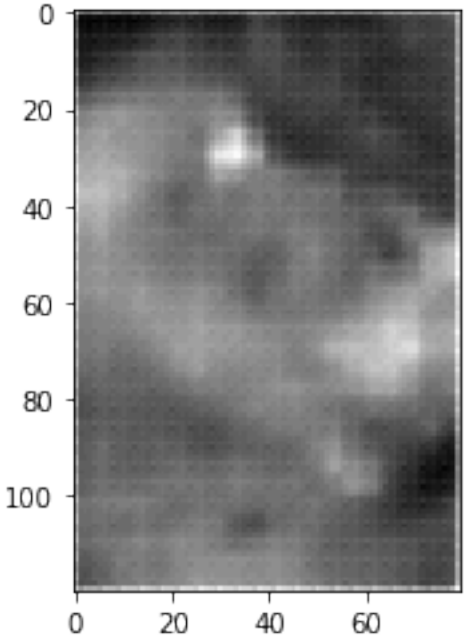

0
20
40
60
80
100
0
20
40
60


0
20
40
60
80
100
0
20
40
60


0
20
40
60
80
100
0
20
40
60


0
20
40
60
80
100
0
20
40
60


0
20
40
60
80
100
0
20
40
60


0
20
40
60
80
100
0
20
40
60

[ ]: