Assignment 4

December 1, 2019

Shakeel Ahamed Mansoor Shaikna | A53315574

1 Getting started

[1]:
```
%matplotlib inline

import os
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.data as td
import torchvision as tv
from PIL import Image
import matplotlib.pyplot as plt
import nntools as nt
```

[2]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

```
cuda
```

2 Creating noisy images of BSDS dataset with DataSet

Question 1

[3]:
```
dataset_root_dir = "/datasets/ee285f-public/bsds/"
```

Question 2

[4]:
```
class NoisyBSDSDataset(td.Dataset):

    def __init__(self, root_dir, mode='train', image_size=(180, 180), sigma=30):
        super(NoisyBSDSDataset, self).__init__()
        self.mode = mode
        self.image_size = image_size
        self.sigma = sigma
        self.images_dir = os.path.join(root_dir, mode)
        self.files = os.listdir(self.images_dir)
```

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

    def __repr__(self):
        return "NoisyBSDSDataset(mode={}, image_size={}, sigma={})". \
            format(self.mode, self.image_size, self.sigma)

    def __getitem__(self, idx):
        img_path = os.path.join(self.images_dir, self.files[idx])
        clean = Image.open(img_path).convert('RGB')
        i = np.random.randint(clean.size[0] - self.image_size[0])
        j = np.random.randint(clean.size[1] - self.image_size[1])

        transform = tv.transforms.Compose([
        tv.transforms.Resize(self.image_size),
        tv.transforms.ToTensor(),
        tv.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
        ])
        clean = clean.crop([i, j, i+self.image_size[0], j+self.image_size[1]])
        clean = transform(clean)

        noisy = clean + 2 / 255 * self.sigma * torch.randn(clean.shape)
        return noisy, clean
```

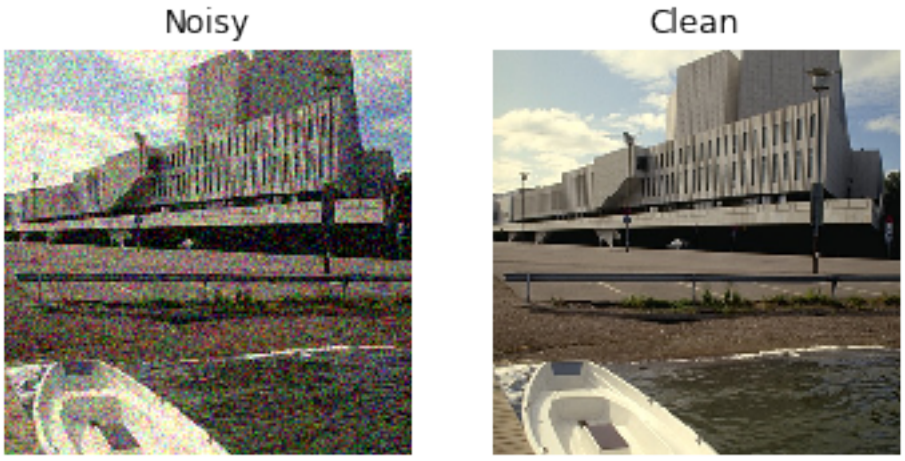
Question 3

[5]:
```
train_set = NoisyBSDSDataset(dataset_root_dir, mode='train',␣
 ↪image_size=(180,180))
test_set = NoisyBSDSDataset(dataset_root_dir, mode='test', image_size=(320,320))
```

[6]:
```
def myimshow(image, ax=plt):
    image = image.to('cpu').numpy()
    image = np.moveaxis(image, [0, 1, 2], [2, 0, 1])
    image = (image + 1) / 2
    image[image < 0] = 0
    image[image > 1] = 1
    h = ax.imshow(image)
    ax.axis('off')
    return h
```

[7]:
```
x = test_set.__getitem__(12)
fig, axes = plt.subplots(ncols=2)
myimshow(x[0], ax=axes[0])
axes[0].set_title('Noisy')
myimshow(x[1], ax=axes[1])
axes[1].set_title('Clean')
```

[7]: Text(0.5, 1.0, 'Clean')



3 DnCNN

Question 4

[8]:
```
class NNRegressor(nt.NeuralNetwork):

    def __init__(self):
        super(NNRegressor, self).__init__()
        self.MSE = nn.MSELoss()

    def criterion(self, y, d):
        return self.MSE(y, d)
```

Question 5

[9]:
```
class DnCNN(NNRegressor):

    def __init__(self, D, C=64):
        super(DnCNN, self).__init__()
        self.D = D
        self.conv = nn.ModuleList()
        self.conv.append(nn.Conv2d(3, C, 3, padding=1))

        for i in range(D):
            self.conv.append(nn.Conv2d(C, C, 3, padding=1))
        self.conv.append(nn.Conv2d(C, 3, 3, padding=1))

        self.bn = nn.ModuleList()
        for k in range(D):
```

```
            self.bn.append(nn.BatchNorm2d(C))

    def forward(self, x):
        D = self.D
        h = F.relu(self.conv[0](x))

        for i in range(D):
            h = F.relu(self.bn[i](self.conv[i+1](h)))

        y = self.conv[D+1](h) + x
        return y
```

Question 6

[10]:
```
class DenoisingStatsManager(nt.StatsManager):

    def __init__(self):
        super(DenoisingStatsManager, self).__init__()

    def init(self):
        super(DenoisingStatsManager, self).init()
        self.running_PSNR = 0

    def accumulate(self, loss, x, y, d):
        super(DenoisingStatsManager, self).accumulate(loss, x, y, d)
        n = x.shape[0] * x.shape[1] * x.shape[2] * x.shape[3]
        self.running_PSNR += 10*torch.log10(4*n/(torch.norm(y-d)**2))

    def summarize(self):
        loss = super(DenoisingStatsManager, self).summarize()
        PSNR = self.running_PSNR / self.number_update
        return {'loss': loss, 'PSNR': PSNR}
```

Question 7

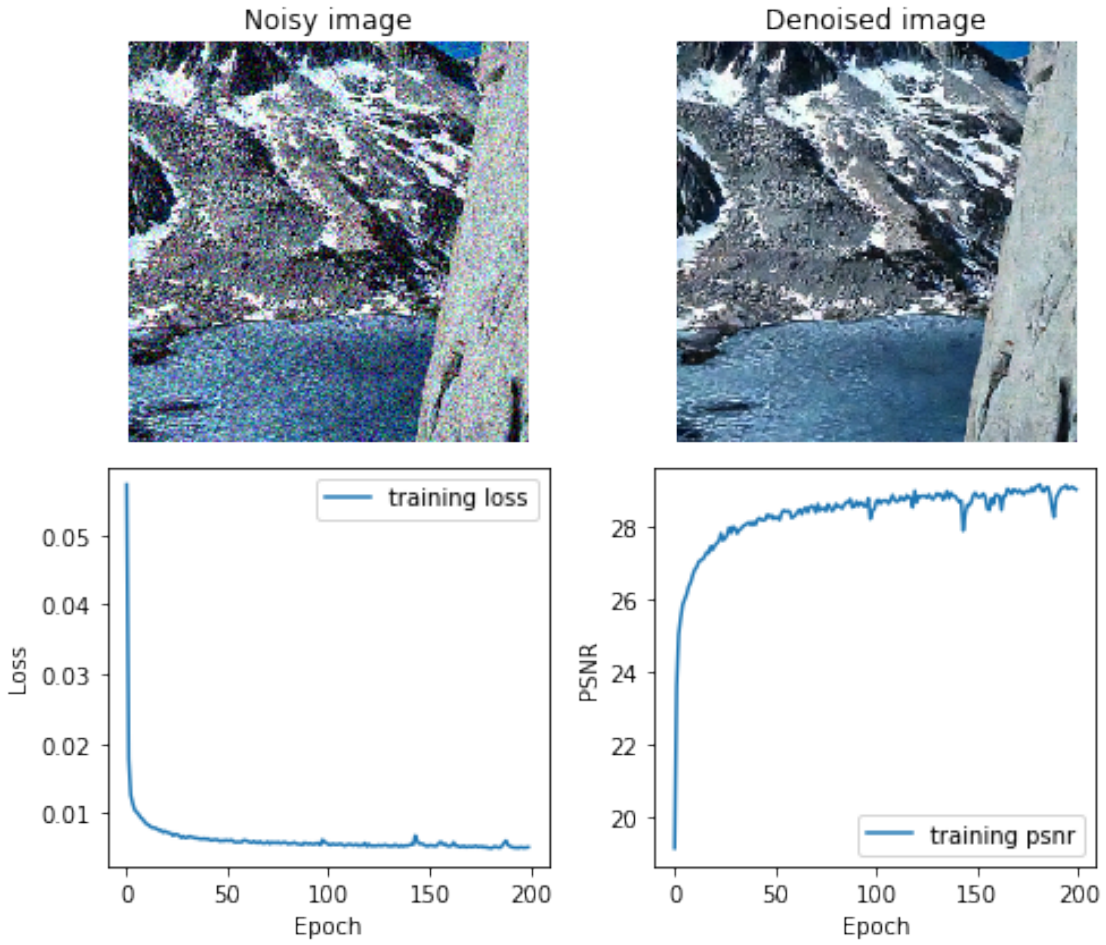
[11]:
```
Dncnn = DnCNN(D = 6)
Dncnn = Dncnn.to(device)
lr = 1e-3
adam = torch.optim.Adam(Dncnn.parameters(), lr=lr)
stats_manager = DenoisingStatsManager()
exp1 = nt.Experiment(Dncnn, train_set, test_set, adam, stats_manager,
                output_dir="denoising1", batch_size=4,␣
↪perform_validation_during_training=True)
```

Question 8

[12]:
```
def plot(exp, fig, axes, noisy, visu_rate=2):
    if exp.epoch % visu_rate != 0:
        return
    with torch.no_grad():
        denoised = exp.net(noisy[np.newaxis].to(exp.net.device))[0]
    axes[0][0].clear()
    axes[0][1].clear()
    axes[1][0].clear()
    axes[1][1].clear()
    myimshow(noisy, ax=axes[0][0])
    axes[0][0].set_title('Noisy image')

    myimshow(denoised, ax=axes[0][1])
    axes[0][1].set_title('Denoised image')

    axes[1][0].plot([exp.history[k][0]['loss'] for k in range(exp.epoch)],␣
↪label='training loss')
    axes[1][0].set_ylabel('Loss')
    axes[1][0].set_xlabel('Epoch')
    axes[1][0].legend()

    axes[1][1].plot([exp.history[k][0]['PSNR'] for k in range(exp.epoch)],␣
↪label='training psnr')
    axes[1][1].set_ylabel('PSNR')
    axes[1][1].set_xlabel('Epoch')
    axes[1][1].legend()

    plt.tight_layout()
    fig.canvas.draw()
```

[13]:
```
fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(7,6))
exp1.run(num_epochs=200, plot=lambda exp: plot(exp, fig=fig, axes=axes,
                                               noisy=test_set[73][0]))
```

```
Start/Continue training from epoch 200
Finish training for 200 epochs
```



Question 9

[14]:
```
img = []
model = exp1.net.to(device)
titles = ['clean', 'noise', 'denoise']

x, clean = test_set[12]
x = x.unsqueeze(0).to(device)
img.append(clean)
img.append(x[0])

model.eval()
with torch.no_grad():
    y = model.forward(x)
img.append(y[0])

fig, axes = plt.subplots(ncols=3, figsize=(9,5), sharex='all', sharey='all')
for i in range(len(img)):
```

```
    myimshow(img[i], ax=axes[i])
    axes[i].set_title(f'{titles[i]}')
```

Although the visual quality of the denoised image looks same as the clean image, based on PSNR we can infer that there still exist some noise. Some areas in the denoised image have lost the information.

Question 10

[15]:
```
print("Parameters of DnCNN")
for name, param in model.named_parameters():
    print(name, param.size(), param.requires_grad)
```

```
Parameters of DnCNN
conv.0.weight torch.Size([64, 3, 3, 3]) True
conv.0.bias torch.Size([64]) True
conv.1.weight torch.Size([64, 64, 3, 3]) True
conv.1.bias torch.Size([64]) True
conv.2.weight torch.Size([64, 64, 3, 3]) True
conv.2.bias torch.Size([64]) True
conv.3.weight torch.Size([64, 64, 3, 3]) True
conv.3.bias torch.Size([64]) True
conv.4.weight torch.Size([64, 64, 3, 3]) True
conv.4.bias torch.Size([64]) True
conv.5.weight torch.Size([64, 64, 3, 3]) True
conv.5.bias torch.Size([64]) True
conv.6.weight torch.Size([64, 64, 3, 3]) True
conv.6.bias torch.Size([64]) True
conv.7.weight torch.Size([3, 64, 3, 3]) True
conv.7.bias torch.Size([3]) True
bn.0.weight torch.Size([64]) True
bn.0.bias torch.Size([64]) True
bn.1.weight torch.Size([64]) True
bn.1.bias torch.Size([64]) True
```

```
bn.2.weight torch.Size([64]) True
bn.2.bias torch.Size([64]) True
bn.3.weight torch.Size([64]) True
bn.3.bias torch.Size([64]) True
bn.4.weight torch.Size([64]) True
bn.4.bias torch.Size([64]) True
bn.5.weight torch.Size([64]) True
bn.5.bias torch.Size([64]) True
```

Number of parameters of DnCNN(D):

Convolution Layers: 64x3x3x3x2 + 64+3 + 64x64x3x3xD + 64xD, BatchNorm Layers: 64x2xD, Total: 3523 + 37056xD parameters. Substituting D=6, we get 225859 parameters.

Receptive Field of DnCNN(D):

Equation to compute the receptive field: $2^{k-l+1} \times padding\ size$, where k and l are the number of the pooling and unpooling layers respectively. Since there are no pooling and unpooling layers in DnCNN network, k=0, l=0. The receptive field of the input layer is 1, and the receptive field next increases after each convolution by $2^{0-0+1} = 2$.

So, the receptive field of DnCNN(D) is $(1+2(D+2)) \times (1+2(D+2))$. For DnCNN(6), we get the receptive field as 17x17.

Question 11

It is given that a pixel should be influenced by at least 33x33 pixels. Implies, the receptive field is 33x33. Since receptive field is $(1+2(D+2)) \times (1+2(D+2))$, we can equate 1+2(D+2) = 33 and get D=14.

Number of parameters is 3523 + 37056xD = 3523 + 37056x14 = 522307. As the number of parameters increases, obviously the computation time would also increase.

4 U-net like CNNs

Question 12

[16]:
```
class UDnCNN(NNRegressor):

    def __init__(self, D, C=64):
        super(UDnCNN, self).__init__()
        self.D = D

        self.conv = nn.ModuleList()
        self.conv.append(nn.Conv2d(3, C, 3, padding=1))
        for i in range(D):
            self.conv.append(nn.Conv2d(C, C, 3, padding=1))
        self.conv.append(nn.Conv2d(C, 3, 3, padding=1))

        self.bn = nn.ModuleList()
        for k in range(D):
            self.bn.append(nn.BatchNorm2d(C))
```

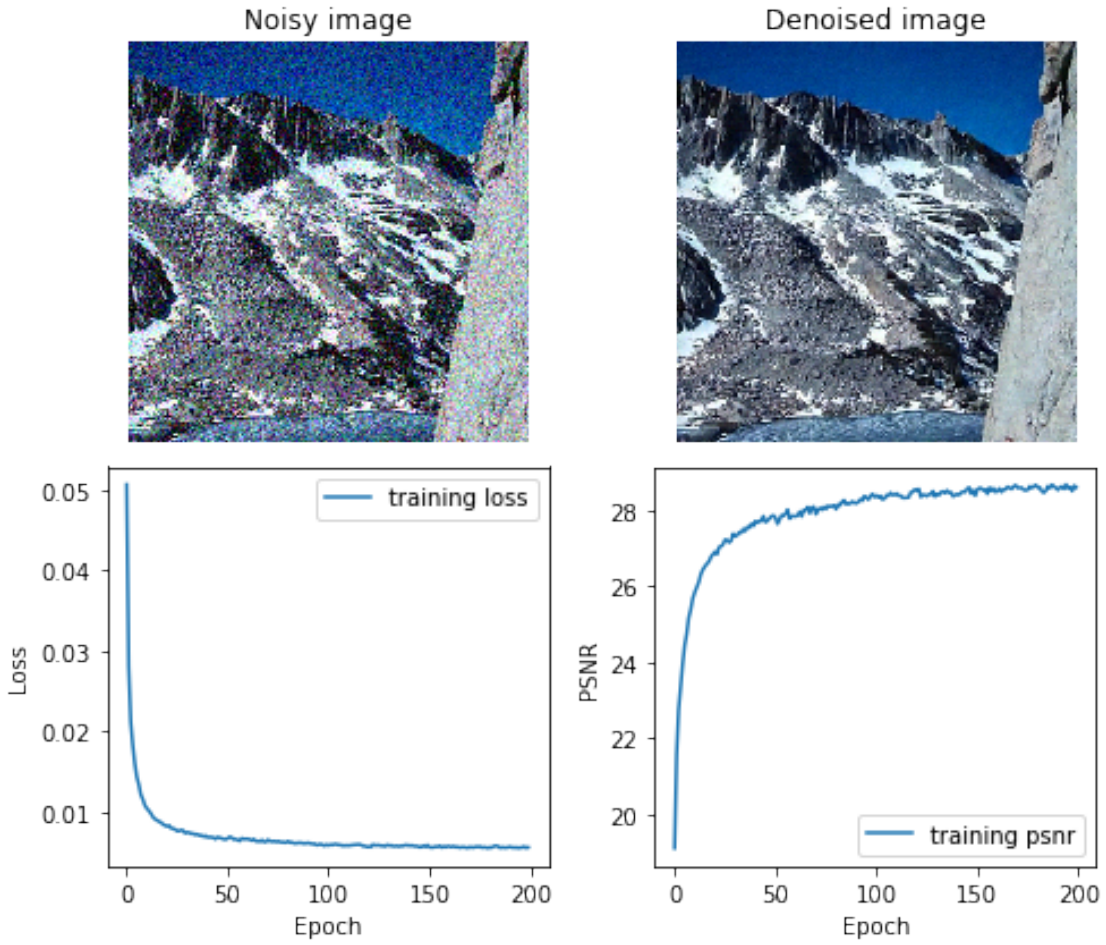
```
    def forward(self, x):
        D = self.D
        h = F.relu(self.conv[0](x))
        features = []
        maxpool_idx = []
        spatial_dim = []
        k = D//2
        for i in range(k-1):
            spatial_dim.append(h.shape)
            h, idx = F.max_pool2d(F.relu(self.bn[i](self.conv[i+1](h))),
                                  kernel_size=(2,2), return_indices=True)
            features.append(h)
            maxpool_idx.append(idx)
        for i in range(k-1, k+1):
            h = F.relu(self.bn[i](self.conv[i+1](h)))
        for i in range(k+1, D):
            j = i - (k + 1) + 1
            h = F.max_unpool2d(F.relu(self.bn[i](self.
↪conv[i+1]((h+features[-j])/np.sqrt(2)))),
                               maxpool_idx[-j], kernel_size=(2,2),␣
↪output_size=spatial_dim[-j])

        y = self.conv[D+1](h) + x
        return y
```

Question 13

[17]:
```
Udncnn = UDnCNN(D = 6)
Udncnn = Udncnn.to(device)
lr = 1e-3
adam = torch.optim.Adam(Udncnn.parameters(), lr=lr)
stats_manager = DenoisingStatsManager()
exp2 = nt.Experiment(Udncnn, train_set, test_set, adam, stats_manager,
              output_dir="denoising2", batch_size=4,␣
↪perform_validation_during_training=True)
```

[18]:
```
fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(7,6))
exp2.run(num_epochs=200, plot=lambda exp: plot(exp, fig=fig, axes=axes,
                                               noisy=test_set[73][0]))
```

```
Start/Continue training from epoch 200
Finish training for 200 epochs
```



Question 14

[19]:
```
print("Parameters of UDnCNN")
for name, param in exp2.net.named_parameters():
    print(name, param.size(), param.requires_grad)
```

```
Parameters of UDnCNN
conv.0.weight torch.Size([64, 3, 3, 3]) True
conv.0.bias torch.Size([64]) True
conv.1.weight torch.Size([64, 64, 3, 3]) True
conv.1.bias torch.Size([64]) True
conv.2.weight torch.Size([64, 64, 3, 3]) True
conv.2.bias torch.Size([64]) True
conv.3.weight torch.Size([64, 64, 3, 3]) True
conv.3.bias torch.Size([64]) True
conv.4.weight torch.Size([64, 64, 3, 3]) True
conv.4.bias torch.Size([64]) True
conv.5.weight torch.Size([64, 64, 3, 3]) True
```

```
conv.5.bias torch.Size([64]) True
conv.6.weight torch.Size([64, 64, 3, 3]) True
conv.6.bias torch.Size([64]) True
conv.7.weight torch.Size([3, 64, 3, 3]) True
conv.7.bias torch.Size([3]) True
bn.0.weight torch.Size([64]) True
bn.0.bias torch.Size([64]) True
bn.1.weight torch.Size([64]) True
bn.1.bias torch.Size([64]) True
bn.2.weight torch.Size([64]) True
bn.2.bias torch.Size([64]) True
bn.3.weight torch.Size([64]) True
bn.3.bias torch.Size([64]) True
bn.4.weight torch.Size([64]) True
bn.4.bias torch.Size([64]) True
bn.5.weight torch.Size([64]) True
bn.5.bias torch.Size([64]) True
```

Number of parameters of UDnCNN(D):

Convolution Layers: 64x3x3x3x2 + 64+3 + 64x64x3x3xD + 64xD, BatchNorm Layers: 64x2xD, Total: 3523 + 37056xD parameters. Substituting D=6, we get 225859 parameters which is the same as DnCNN.

Receptive Field of UDnCNN(D):

First Layer: 1, Convolution Layer: 2, Pooling & Unpooling: $\sum_{i=1}^{D/2} 2^i$.

Receptive Field output: $(1 + 2 + \sum_{i=1}^{D/2} 2^i + \sum_{i=1}^{D/2} 2^i + 2) \times (1 + 2 + \sum_{i=1}^{D/2} 2^i + \sum_{i=1}^{D/2} 2^i + 2) = (5 + \sum_{i=1}^{D/2} 2^{i+1}) \times (5 + \sum_{i=1}^{D/2} 2^{i+1})$.

So, the receptive field of UDnCNN(D) is $(5 + \sum_{i=1}^{D/2} 2^{i+1}) \times (5 + \sum_{i=1}^{D/2} 2^{i+1})$. For UDnCNN(6), we get the receptive field as 33x33.

From the performance evaluation, I get PSNR for UDnCNN as 28.4177 and PSNR for DnCNN as 28.4030. So, I expect UDnCNN to beat DnCNN, because UDnCNN performance was slightly higher than DnCNN performance.

Question 15

[20]:
```
exp1.evaluate()
```

[20]:
```
{'loss': 0.0058896875567734245, 'PSNR': tensor(28.4030, device='cuda:0')}
```

[21]:
```
exp2.evaluate()
```

[21]:
```
{'loss': 0.005863057672977448, 'PSNR': tensor(28.4177, device='cuda:0')}
```

5 U-net like CNNs with dilated convolutions

Question 16, 17

[22]:
```
class DUDnCNN(NNRegressor):

    def __init__(self, D, C=64):
        super(DUDnCNN, self).__init__()
        self.D = D

        # compute k(max_pool) and l(max_unpool)
        k = [0]
        k.extend([i for i in range(D//2)])
        k.extend([k[-1] for _ in range(D//2, D+1)])
        l = [0 for _ in range(D//2+1)]
        l.extend([i for i in range(D+1-(D//2+1))])
        l.append(l[-1])

        # holes and dilations for convolution layers
        holes = [2**(kl[0]-kl[1])-1 for kl in zip(k,l)]
        dilations = [i+1 for i in holes]

        # convolution layers
        self.conv = nn.ModuleList()
        self.conv.append(nn.Conv2d(3, C, 3, padding=dilations[0],␣
↪dilation=dilations[0]))
        self.conv.extend([nn.Conv2d(C, C, 3, padding=dilations[i+1],␣
↪dilation=dilations[i+1]) for i in range(D)])
        self.conv.append(nn.Conv2d(C, 3, 3, padding=dilations[-1],␣
↪dilation=dilations[-1]))

        # batch normalization
        self.bn = nn.ModuleList()
        self.bn.extend([nn.BatchNorm2d(C, C) for _ in range(D)])


    def forward(self, x):
        D = self.D
        h = F.relu(self.conv[0](x))
        features = []

        for i in range(D//2 - 1):
            torch.backends.cudnn.benchmark = True
            h = self.conv[i+1](h)
            torch.backends.cudnn.benchmark = False
            h = F.relu(self.bn[i](h))
            features.append(h)

        for i in range(D//2 - 1, D//2 + 1):
            torch.backends.cudnn.benchmark = True
            h = self.conv[i+1](h)
```

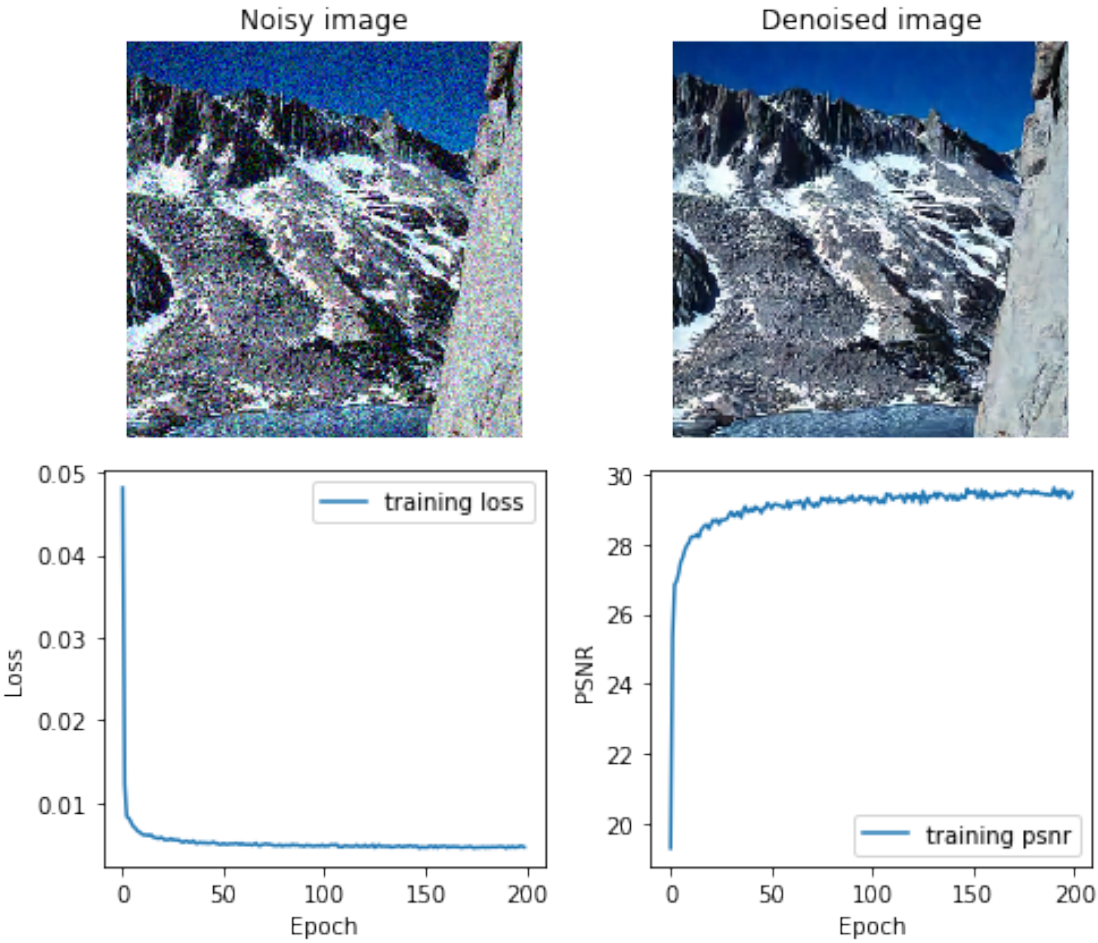
```
            torch.backends.cudnn.benchmark = False
            h = F.relu(self.bn[i](h))

        for i in range(D//2 + 1, D):
            j = i - (D//2 + 1) + 1
            torch.backends.cudnn.benchmark = True
            h = self.conv[i+1]((h + features[-j]) / np.sqrt(2))
            torch.backends.cudnn.benchmark = False
            h = F.relu(self.bn[i](h))

        y = self.conv[D+1](h) + x
        return y
```

Question 18

[23]:
```
Dudncnn = DUDnCNN(D = 6)
Dudncnn = Dudncnn.to(device)
lr = 1e-3
adam = torch.optim.Adam(Dudncnn.parameters(), lr=lr)
stats_manager = DenoisingStatsManager()
exp3 = nt.Experiment(Dudncnn, train_set, test_set, adam, stats_manager,
                output_dir="denoising3", batch_size=4,␣
 ↪perform_validation_during_training=True)
```

[24]:
```
fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(7,6))
exp3.run(num_epochs=200, plot=lambda exp: plot(exp, fig=fig, axes=axes,
                                               noisy=test_set[73][0]))
```

```
Start/Continue training from epoch 200
Finish training for 200 epochs
```



Question 19

[25]:
```
exp1.evaluate()
```

[25]: {'loss': 0.005901629235595464, 'PSNR': tensor(28.3888, device='cuda:0')}

[26]:
```
exp2.evaluate()
```

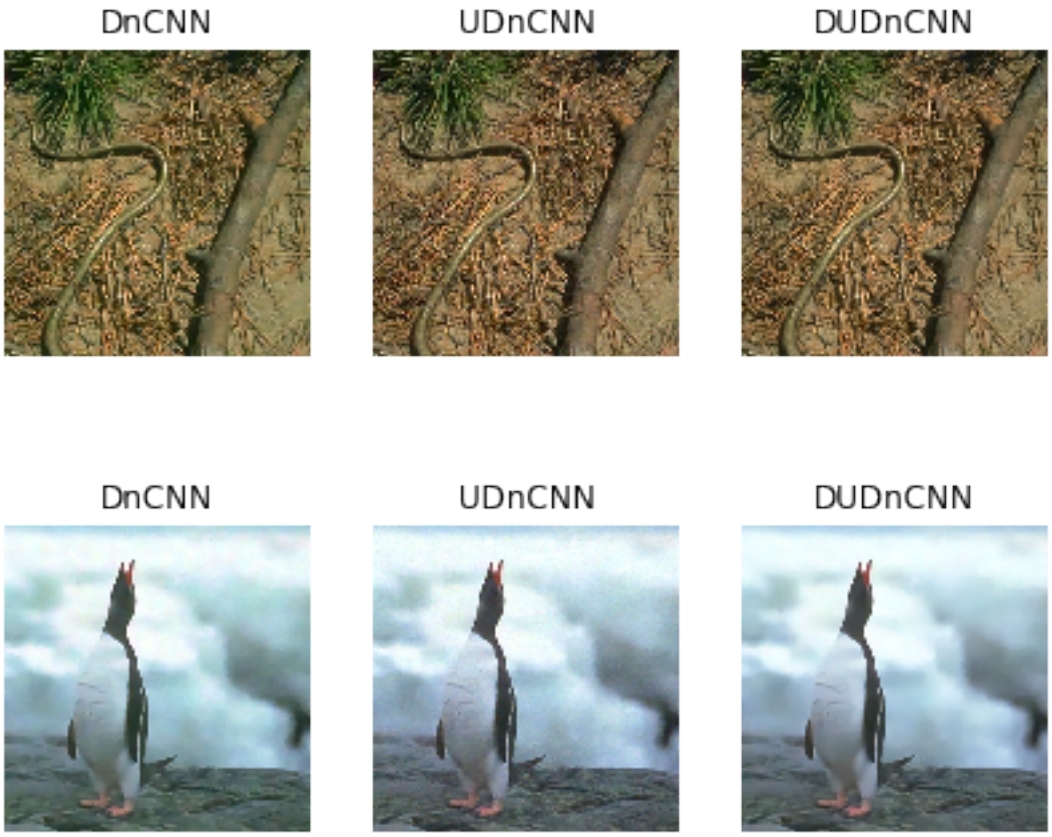
[26]: {'loss': 0.005895433761179447, 'PSNR': tensor(28.3928, device='cuda:0')}

[27]:
```
exp3.evaluate()
```

[27]: {'loss': 0.0048937905300408605, 'PSNR': tensor(29.2206, device='cuda:0')}

[28]:
```
num = 2
img = []
nets = [exp1.net, exp2.net, exp3.net]
titles = ['DnCNN', 'UDnCNN', 'DUDnCNN']
```

```
for i in range(num):
    x, _ = test_set[10*i+15]
    x = x.unsqueeze(0).to(device)
    img.append(x)

fig, axes = plt.subplots(nrows=num, ncols=3, figsize=(7,6), sharex='all',␣
 ↪sharey='all')
for i in range(num):
    for j in range(len(nets)):
        model = nets[j].to(device)
        model.eval()
        with torch.no_grad():
            y = model.forward(img[i])

        myimshow(y[0], ax=axes[i][j])
        axes[i][j].set_title(f'{titles[j]}')
```



Question 20

[29]:
```
print("Parameters of DUDnCNN")
for name, param in exp3.net.named_parameters():
    print(name, param.size(), param.requires_grad)
```

```
Parameters of DUDnCNN
conv.0.weight torch.Size([64, 3, 3, 3]) True
conv.0.bias torch.Size([64]) True
conv.1.weight torch.Size([64, 64, 3, 3]) True
conv.1.bias torch.Size([64]) True
conv.2.weight torch.Size([64, 64, 3, 3]) True
conv.2.bias torch.Size([64]) True
conv.3.weight torch.Size([64, 64, 3, 3]) True
conv.3.bias torch.Size([64]) True
conv.4.weight torch.Size([64, 64, 3, 3]) True
conv.4.bias torch.Size([64]) True
conv.5.weight torch.Size([64, 64, 3, 3]) True
conv.5.bias torch.Size([64]) True
conv.6.weight torch.Size([64, 64, 3, 3]) True
conv.6.bias torch.Size([64]) True
conv.7.weight torch.Size([3, 64, 3, 3]) True
conv.7.bias torch.Size([3]) True
bn.0.weight torch.Size([64]) True
bn.0.bias torch.Size([64]) True
bn.1.weight torch.Size([64]) True
bn.1.bias torch.Size([64]) True
bn.2.weight torch.Size([64]) True
bn.2.bias torch.Size([64]) True
bn.3.weight torch.Size([64]) True
bn.3.bias torch.Size([64]) True
bn.4.weight torch.Size([64]) True
bn.4.bias torch.Size([64]) True
bn.5.weight torch.Size([64]) True
bn.5.bias torch.Size([64]) True
```

Number of parameters of DUDnCNN(D):

Convolution Layers: 64x3x3x3x2 + 64+3 + 64x64x3x3xD + 64xD, BatchNorm Layers: 64x2xD, Total: 3523 + 37056xD parameters. Substituting D=6, we get 225859 parameters which is the same as DnCNN and UDnCNN.

Receptive Field of DUDnCNN(D):

The receptive field of DUDnCNN(D) is $(5+\sum_{i=1}^{D/2} 2^{i+1}) \times (5+\sum_{i=1}^{D/2} 2^{i+1})$. For DUDnCNN(6), we get the receptive field as 33x33.