In [1]:
```
%matplotlib notebook
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
import socket

device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

```
cuda
```

In [2]:
```
## 1.
dataset_root_dir = os.path.join(socket.gethostname(), '/datasets/ee
285f-public/bsds')
print(os.path.isdir(dataset_root_dir))
```

```
True
```

In [3]:
```
## 2.
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])
        clean = clean.crop((i, j, i+self.image_size[0], j+self.imag
e_size[1]))
        transform = tv.transforms.Compose([
            tv.transforms.ToTensor(),    # convert numpy to tensor
            tv.transforms.Normalize(mean=[0.5,0.5,0.5],std=[0.5,0.5
,0.5]),    # normalized to [-1, 1]
            ])
        clean = transform(clean)
        noisy = clean + 2 / 255 * self.sigma * torch.randn(clean.sh
ape)
        return noisy, clean
```

In [4]:
```
## 3.
def myimshow(image, ax=plt):
    if image.requires_grad == True:
        image = image.to('cpu').detach().numpy()
    else:
        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
    ax.subplots()
    h = ax.imshow(image)
    ax.axis('off')
    return h

train_set = NoisyBSDSDataset(dataset_root_dir)
test_set = NoisyBSDSDataset(dataset_root_dir, mode = 'test', image_
size=(320, 320))
x = test_set[12]
for img in x:
    myimshow(img)
    print(img.shape)
```

```
torch.Size([3, 320, 320])
```

```
torch.Size([3, 320, 320])
```

In [5]:
```
## 4.
class NNRegressor(nt.NeuralNetwork):

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

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

In [5]:
```
## 5.
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 = (1,1)))
        for i in range(D):
            self.conv.append(nn.Conv2d(C, C, 3, padding = (1,1)))
        self.conv.append(nn.Conv2d(C, 3, 3, padding = (1,1)))

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


    def forward(self, x):
        D = self.D
        h = F.relu(self.conv[0](x))
        for j in range(D):
            h = F.relu(self.bn[j](self.conv[j+1](h)))
        y = self.conv[D+1](h) + x
        return y
```

In [26]:
```
## 6.
x = train_set[-1][0].reshape(1,3,180,180)
net0 = DnCNN(0)
net1 = DnCNN(1)
net2 = DnCNN(2)
net4 = DnCNN(4)
net8 = DnCNN(8)

y = [net0.forward(x), net1.forward(x), net2.forward(x), net4.forwar
d(x), net8.forward(x)]
for ind,img in enumerate(y, -1):
    print('Numerical result of D = %d is ' % (0 if 2**ind <1 else 2
**ind))
    print(x[0]-img[0])
    print('Display of D = %d is ' % (0 if 2**ind <1 else 2**ind))
    myimshow(x[0])
    myimshow(img[0])
    myimshow(x[0]-img[0])
```

```
Numerical result of D = 0 is
tensor([[[ 6.2274e-02,  1.8706e-01,  1.5285e-01,  ...,  2.3298e-01
,
           1.3050e-01, -1.5422e-02],
         [ 5.7557e-02,  1.9780e-01,  2.3984e-01,  ...,  1.3681e-01
,
           2.5443e-01,  1.4006e-01],
```

```
          [ 1.2923e-01,  2.4658e-01,  3.3670e-01,  ...,  1.7770e-01
,
            1.0402e-01,  1.6969e-04],
          ...,
          [-4.4843e-03,  1.5826e-01,  1.8145e-01,  ...,  4.1205e-02
,
           -3.1641e-02, -1.3810e-02],
          [ 2.0685e-02,  1.7772e-01,  2.2258e-01,  ..., -2.0373e-02
,
            3.1848e-02, -3.1762e-02],
          [-4.4620e-02,  1.3901e-01,  3.3130e-02,  ...,  3.9849e-02
,
            4.2394e-02,  2.0435e-02]],

         [[-8.8697e-02,  7.8360e-02, -1.2359e-02,  ...,  2.3813e-01
,
            6.7967e-02,  7.5422e-02],
          [-1.5517e-01, -2.1468e-01, -1.7138e-01,  ...,  1.2210e-01
,
            1.0906e-01,  8.6288e-02],
          [-1.7163e-01, -2.4433e-01, -1.0436e-01,  ...,  1.1116e-01
,
            1.6898e-01,  1.4921e-01],
          ...,
          [-2.2479e-02, -1.9080e-01, -2.3822e-01,  ..., -6.2467e-02
,
           -1.0712e-01, -1.2457e-01],
          [-1.8005e-01, -1.6666e-01, -2.1358e-01,  ..., -1.6578e-01
,
           -2.0013e-01, -1.1535e-01],
          [-4.6944e-02, -2.7846e-01, -1.5479e-01,  ..., -2.3522e-01
,
           -1.3593e-01, -1.0797e-01]],

         [[-1.7374e-01, -2.1642e-01, -2.3575e-01,  ...,  1.0029e-01
,
            9.1280e-02,  1.7975e-01],
          [-6.8309e-02, -8.4667e-02, -1.5486e-01,  ...,  1.3918e-01
,
            1.1591e-01,  2.0294e-01],
          [-2.9767e-02,  1.2235e-02, -3.6261e-02,  ...,  2.2317e-01
,
            1.8622e-01,  1.9328e-01],
          ...,
          [ 2.7364e-02,  2.5722e-02,  1.3134e-02,  ..., -5.8299e-02
,
            3.7648e-02,  1.1697e-01],
          [ 3.5608e-02,  1.1723e-02,  6.2212e-03,  ...,  9.3607e-02
,
            2.1000e-01,  1.2456e-01],
          [ 1.3760e-01,  6.7204e-02,  7.4328e-02,  ...,  1.8222e-01
,
            1.8709e-01,  1.1401e-01]]], grad_fn=<SubBackward0>)
```

```
Display of D = 0 is
```

```
Numerical result of D = 1 is
tensor([[[ 0.0221,  0.0217,  0.0205,  ...,  0.0236,  0.0254,  0.02
38],
         [ 0.0231,  0.0229,  0.0254,  ...,  0.0209,  0.0264,  0.01
87],
         [ 0.0216,  0.0250,  0.0259,  ...,  0.0212,  0.0243,  0.02
05],
         ...,
         [ 0.0253,  0.0248,  0.0209,  ...,  0.0236,  0.0265,  0.01
95],
         [ 0.0243,  0.0233,  0.0215,  ...,  0.0238,  0.0270,  0.02
14],
         [ 0.0217,  0.0226,  0.0226,  ...,  0.0203,  0.0222,  0.01
85]],

        [[ 0.0032, -0.0002,  0.0014,  ...,  0.0051,  0.0019,  0.00
45],
         [-0.0011,  0.0013, -0.0005,  ...,  0.0019,  0.0017,  0.00
47],
         [-0.0023, -0.0037, -0.0009,  ...,  0.0003,  0.0018,  0.00
29],
         ...,
         [ 0.0046,  0.0029,  0.0070,  ...,  0.0087,  0.0069,  0.00
40],
         [ 0.0024,  0.0042,  0.0077,  ...,  0.0091,  0.0065,  0.00
68],
         [ 0.0036,  0.0042,  0.0048,  ...,  0.0053,  0.0059,  0.00
49]],

        [[-0.0175, -0.0149, -0.0151,  ..., -0.0139, -0.0129, -0.01
13],
         [-0.0162, -0.0176, -0.0188,  ..., -0.0100, -0.0115, -0.01
52],
         [-0.0166, -0.0186, -0.0179,  ..., -0.0150, -0.0153, -0.01
47],
         ...,
         [-0.0144, -0.0150, -0.0148,  ..., -0.0140, -0.0127, -0.01
54],
         [-0.0118, -0.0151, -0.0107,  ..., -0.0139, -0.0102, -0.01
53],
         [-0.0097, -0.0147, -0.0119,  ..., -0.0128, -0.0112, -0.01
59]]],
       grad_fn=<SubBackward0>)
Display of D = 1 is
```

```
Numerical result of D = 2 is
tensor([[[-0.0333, -0.0334, -0.0334,  ..., -0.0332, -0.0332, -0.03
32],
         [-0.0332, -0.0333, -0.0332,  ..., -0.0330, -0.0330, -0.03
31],
         [-0.0332, -0.0332, -0.0331,  ..., -0.0330, -0.0330, -0.03
31],
         ...,
         [-0.0333, -0.0332, -0.0332,  ..., -0.0331, -0.0331, -0.03
31],
         [-0.0332, -0.0332, -0.0332,  ..., -0.0331, -0.0331, -0.03
31],
         [-0.0331, -0.0331, -0.0331,  ..., -0.0331, -0.0331, -0.03
31]],

        [[ 0.0045,  0.0045,  0.0044,  ...,  0.0046,  0.0046,  0.00
45],
         [ 0.0045,  0.0045,  0.0045,  ...,  0.0046,  0.0046,  0.00
47],
         [ 0.0044,  0.0044,  0.0044,  ...,  0.0047,  0.0046,  0.00
47],
         ...,
         [ 0.0045,  0.0046,  0.0045,  ...,  0.0045,  0.0046,  0.00
45],
         [ 0.0046,  0.0046,  0.0045,  ...,  0.0046,  0.0046,  0.00
45],
         [ 0.0046,  0.0046,  0.0046,  ...,  0.0046,  0.0046,  0.00
46]],

        [[-0.0041, -0.0041, -0.0041,  ..., -0.0041, -0.0041, -0.00
41],
         [-0.0041, -0.0041, -0.0042,  ..., -0.0040, -0.0041, -0.00
41],
         [-0.0041, -0.0041, -0.0041,  ..., -0.0040, -0.0041, -0.00
40],
         ...,
         [-0.0040, -0.0041, -0.0041,  ..., -0.0040, -0.0040, -0.00
40],
         [-0.0041, -0.0041, -0.0042,  ..., -0.0041, -0.0041, -0.00
41],
         [-0.0040, -0.0040, -0.0040,  ..., -0.0041, -0.0041, -0.00
41]]],
       grad_fn=<SubBackward0>)
Display of D = 2 is
```

```
Numerical result of D = 4 is
tensor([[[-0.0366, -0.0366, -0.0366,  ..., -0.0366, -0.0366, -0.03
66],
         [-0.0366, -0.0366, -0.0366,  ..., -0.0366, -0.0366, -0.03
66],
         [-0.0366, -0.0366, -0.0366,  ..., -0.0366, -0.0366, -0.03
66],
         ...,
         [-0.0366, -0.0366, -0.0366,  ..., -0.0366, -0.0366, -0.03
66],
         [-0.0366, -0.0366, -0.0366,  ..., -0.0366, -0.0366, -0.03
66],
         [-0.0366, -0.0366, -0.0366,  ..., -0.0366, -0.0366, -0.03
66]],

        [[-0.0375, -0.0375, -0.0375,  ..., -0.0375, -0.0375, -0.03
75],
         [-0.0375, -0.0375, -0.0375,  ..., -0.0375, -0.0375, -0.03
75],
         [-0.0375, -0.0375, -0.0375,  ..., -0.0375, -0.0375, -0.03
75],
         ...,
         [-0.0375, -0.0375, -0.0375,  ..., -0.0375, -0.0375, -0.03
75],
         [-0.0375, -0.0375, -0.0375,  ..., -0.0375, -0.0375, -0.03
75],
         [-0.0375, -0.0375, -0.0375,  ..., -0.0375, -0.0375, -0.03
75]],

        [[ 0.0373,  0.0373,  0.0373,  ...,  0.0373,  0.0373,  0.03
73],
         [ 0.0373,  0.0373,  0.0373,  ...,  0.0373,  0.0373,  0.03
73],
         [ 0.0373,  0.0373,  0.0373,  ...,  0.0373,  0.0373,  0.03
73],
         ...,
         [ 0.0373,  0.0373,  0.0373,  ...,  0.0373,  0.0373,  0.03
73],
         [ 0.0373,  0.0373,  0.0373,  ...,  0.0373,  0.0373,  0.03
73],
         [ 0.0373,  0.0373,  0.0373,  ...,  0.0373,  0.0373,  0.03
73]]],
       grad_fn=<SubBackward0>)
Display of D = 4 is
```

```
Numerical result of D = 8 is
tensor([[[0.0374, 0.0374, 0.0374,  ..., 0.0374, 0.0374, 0.0374],
         [0.0374, 0.0374, 0.0374,  ..., 0.0374, 0.0374, 0.0374],
         [0.0374, 0.0374, 0.0374,  ..., 0.0374, 0.0374, 0.0374],
         ...,
         [0.0374, 0.0374, 0.0374,  ..., 0.0374, 0.0374, 0.0374],
         [0.0374, 0.0374, 0.0374,  ..., 0.0374, 0.0374, 0.0374],
         [0.0374, 0.0374, 0.0374,  ..., 0.0374, 0.0374, 0.0374]],

        [[0.0110, 0.0110, 0.0110,  ..., 0.0110, 0.0110, 0.0110],
         [0.0110, 0.0110, 0.0110,  ..., 0.0110, 0.0110, 0.0110],
         [0.0110, 0.0110, 0.0110,  ..., 0.0110, 0.0110, 0.0110],
         ...,
         [0.0110, 0.0110, 0.0110,  ..., 0.0110, 0.0110, 0.0110],
         [0.0110, 0.0110, 0.0110,  ..., 0.0110, 0.0110, 0.0110],
         [0.0110, 0.0110, 0.0110,  ..., 0.0110, 0.0110, 0.0110]],

        [[0.0366, 0.0366, 0.0366,  ..., 0.0366, 0.0366, 0.0366],
         [0.0366, 0.0366, 0.0366,  ..., 0.0366, 0.0366, 0.0366],
         [0.0366, 0.0366, 0.0366,  ..., 0.0366, 0.0366, 0.0366],
         ...,
         [0.0366, 0.0366, 0.0366,  ..., 0.0366, 0.0366, 0.0366],
         [0.0366, 0.0366, 0.0366,  ..., 0.0366, 0.0366, 0.0366],
         [0.0366, 0.0366, 0.0366,  ..., 0.0366, 0.0366, 0.0366]]],
       grad_fn=<SubBackward0>)
Display of D = 8 is
```

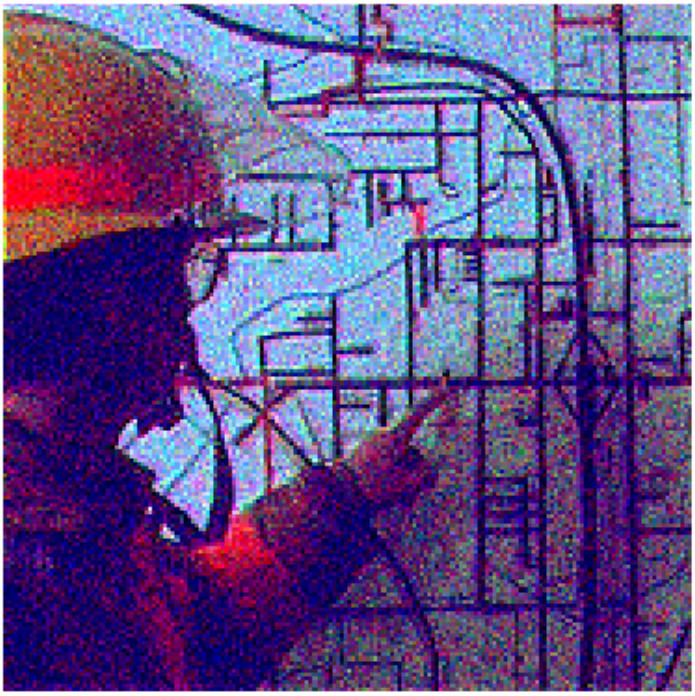
With D increasing, the difference between x[0] and y[0] decreases. In other word, the output of the D convolutions of the network gradually tends to be zero(array(0,0,0) with each component ranged in [-1, 1] is gray). The network with zero-initialized weights cannot learn anything.

The back-propagation will be very hard to train the network to learn the residual(negative noise). This way of initialization would increase the probability of learning the same thing and learning with different speeds for each layers, which is bad for learning.

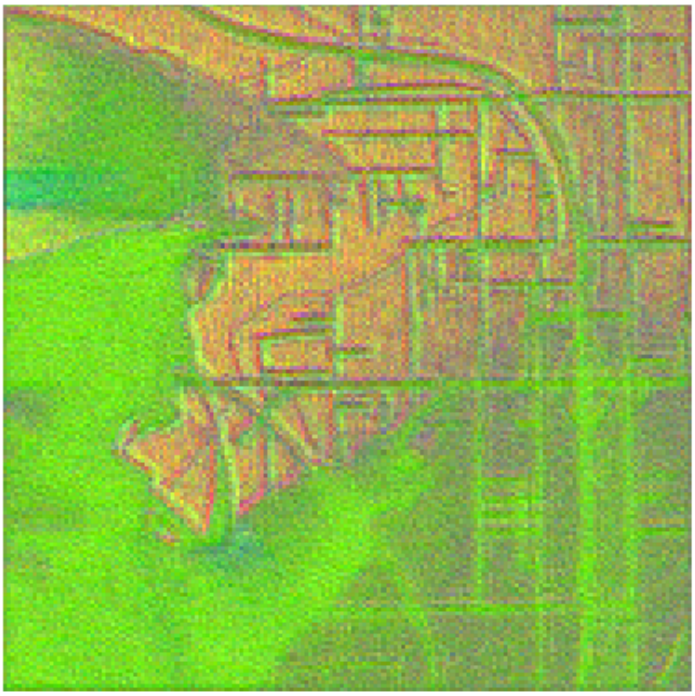
In [14]:
```
## 7.
class DnCNN(NNRegressor):

    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 = (1,1)))
        for i in range(D):
            self.conv.append(nn.Conv2d(C, C, 3, padding = (1,1)))
        self.conv.append(nn.Conv2d(C, 3, 3, padding = (1,1)))
        # initialize of conv
        for i in range(D+1):
            nn.init.kaiming_normal_(self.conv[i].weight.data, mode=
'fan_in', nonlinearity='relu')

        self.bn = nn.ModuleList()
        for k in range(D):
            self.bn.append(nn.BatchNorm2d(C, C))
        # initialize of bn
        for k in range(D):
            nn.init.constant_(self.bn[k].weight.data, 1.25 * np.sqr
t(C))


    def forward(self, x):
        D = self.D
        h = F.relu(self.conv[0](x))
        for j in range(D):
            h = F.relu(self.bn[j](self.conv[j+1](h)))
        y = self.conv[D+1](h) + x
        return y
```

In [28]:
```
## 8.
x = train_set[-1][0].reshape(1,3,180,180)
net0 = DnCNN(0)
net1 = DnCNN(1)
net2 = DnCNN(2)
net4 = DnCNN(4)
net8 = DnCNN(8)

y = [net0.forward(x), net1.forward(x), net2.forward(x), net4.forwar
d(x), net8.forward(x)]
for ind,img in enumerate(y, -1):
    print('Numerical result of D = %d is ' % (0 if 2**ind <1 else 2
**ind))
    print(x[0]-img[0])
    print('Display of D = %d is ' % (0 if 2**ind <1 else 2**ind))
    myimshow(x[0])
    myimshow(img[0])
    myimshow(x[0]-img[0])
```

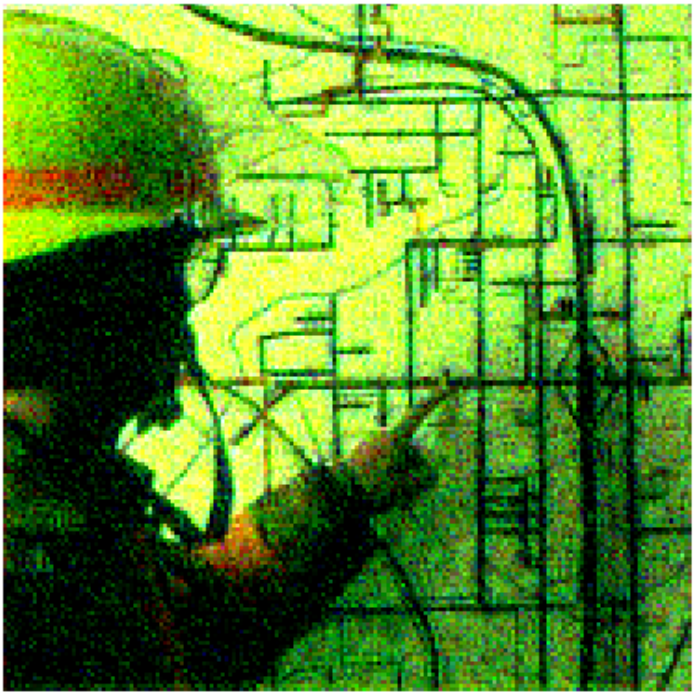
```
Numerical result of D = 0 is
tensor([[[-0.0293,  0.0953,  0.1313,  ...,  0.0275,  0.1438,  0.14
46],
         [ 0.2264,  0.2657,  0.4598,  ...,  0.2771,  0.2055,  0.20
11],
         [ 0.2382,  0.1785,  0.3231,  ...,  0.3514,  0.4231,  0.24
06],
         ...,
         [-0.0822, -0.1958, -0.0503,  ...,  0.0191, -0.2084, -0.03
69],
         [-0.4186, -0.1963, -0.3750,  ..., -0.1078, -0.0013, -0.15
45],
         [-0.3622, -0.3000, -0.3359,  ..., -0.1194, -0.0970, -0.06
25]],

        [[ 0.3308,  0.5799,  0.4620,  ...,  0.2137,  0.0356, -0.03
50],
         [ 0.0046,  0.0458,  0.1212,  ...,  0.5445,  0.0152, -0.23
66],
         [-0.0187, -0.0255, -0.2285,  ...,  0.3358, -0.1041, -0.01
73],
         ...,
         [ 0.5497,  0.6772,  0.5919,  ...,  0.3352,  0.1752,  0.16
26],
         [ 0.4077,  0.8869,  0.6718,  ...,  0.2527,  0.3352, -0.03
57],
         [ 0.3082,  0.4487,  0.4685,  ...,  0.0019,  0.0323,  0.02
91]],

        [[-0.2143, -0.3566, -0.4787,  ..., -0.2981, -0.3039, -0.10
14],
         [-0.0566, -0.2229, -0.2009,  ..., -0.4121, -0.3083, -0.17
09],
         [-0.1031, -0.2611, -0.0207,  ..., -0.3322, -0.1943, -0.11
60],
         ...,
         [-0.6107, -0.5518, -0.6357,  ..., -0.3072, -0.3242, -0.40
14],
         [-0.7348, -0.9129, -0.8854,  ..., -0.2779, -0.3009, -0.12
42],
         [-0.6473, -0.6328, -0.8038,  ..., -0.3920, -0.2415, -0.11
80]]],
       grad_fn=<SubBackward0>)
Display of D = 0 is
```

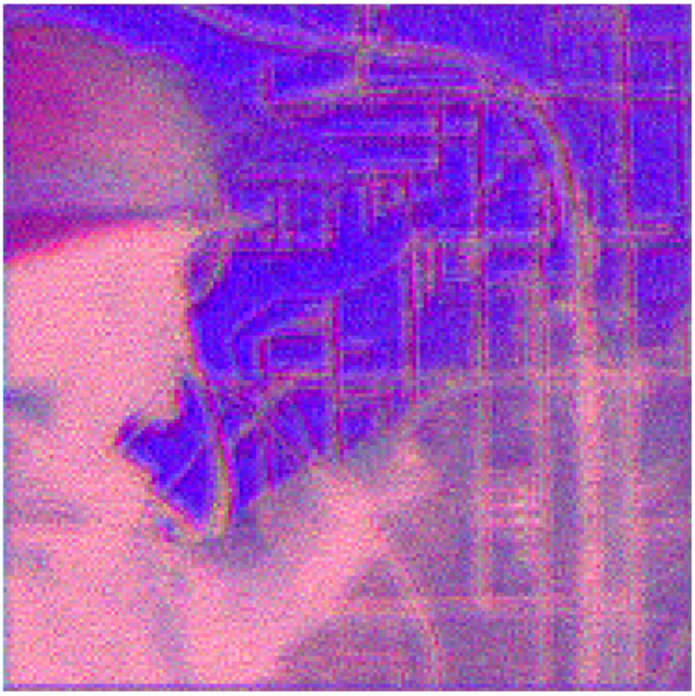


```
/opt/conda/lib/python3.6/site-packages/matplotlib/pyplot.py:514: R
untimeWarning: More than 20 figures have been opened. Figures crea
ted through the pyplot interface (`matplotlib.pyplot.figure`) are
retained until explicitly closed and may consume too much memory.
(To control this warning, see the rcParam `figure.max_open_warning
`).
  max_open_warning, RuntimeWarning)
```

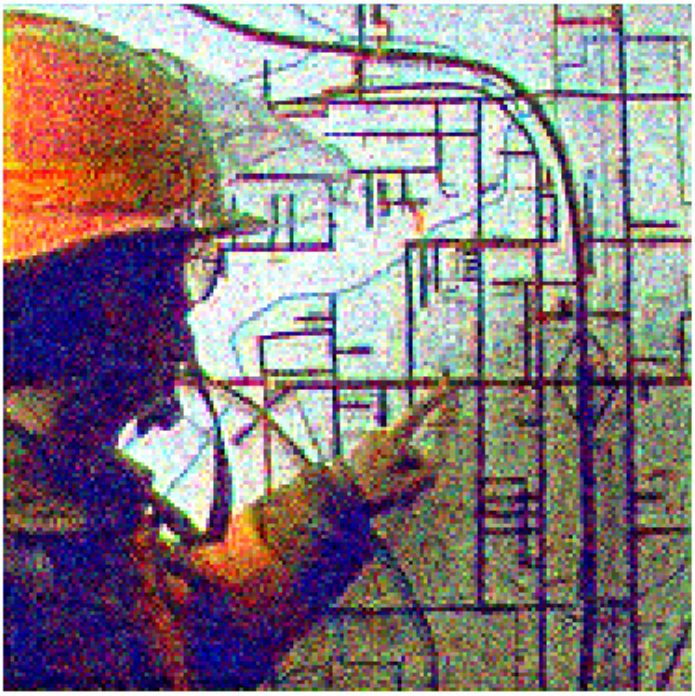


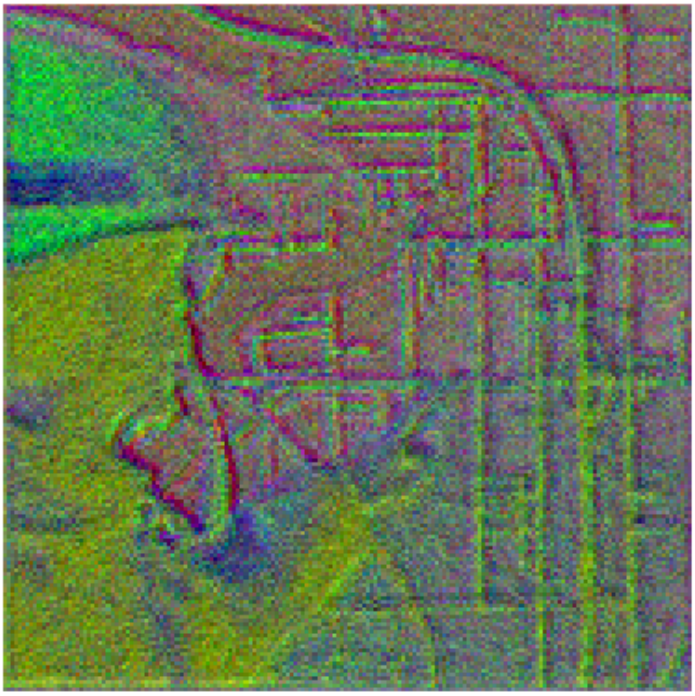
```
Numerical result of D = 1 is
tensor([[[-0.1840, -0.3858, -0.0164,  ...,  0.1856, -0.1003, -0.17
74],
         [ 0.0424, -0.0167, -0.3193,  ..., -0.3401,  0.0370, -0.01
64],
         [-0.1718,  0.0513,  0.2925,  ...,  0.1179, -0.1456, -0.09
65],
         ...,
         [ 0.0608,  0.1889,  0.6344,  ...,  0.1287,  0.1267, -0.03
50],
         [ 0.0707,  0.2405,  0.2714,  ...,  0.1848,  0.0587,  0.07
08],
         [-0.1525,  0.1645,  0.1005,  ...,  0.1751, -0.0807,  0.14
89]],

        [[-0.1746, -0.8217, -0.8130,  ..., -0.6779, -0.4865, -0.47
65],
         [-0.6363, -0.6087, -0.4212,  ..., -0.6185, -0.6156, -0.47
77],
         [-0.5132, -0.4857, -0.7854,  ..., -0.7892, -0.6540, -0.54
97],
         ...,
         [ 0.4023,  0.3333,  0.1864,  ..., -0.4801, -0.3904, -0.27
90],
         [ 0.1039, -0.1883, -0.3303,  ..., -0.4600, -0.4641, -0.39
36],
         [-0.1561, -0.1387, -0.1199,  ..., -0.3468, -0.3403, -0.31
56]],

        [[ 0.1305,  0.4272,  0.4841,  ...,  0.4676,  0.4886,  0.30
42],
         [ 0.1078,  0.5214,  0.2932,  ...,  0.6530,  0.7113,  0.62
31],
         [ 0.2661,  0.5085,  0.4727,  ...,  0.7846,  0.6504,  0.51
43],
         ...,
         [ 0.5835,  0.3976,  0.1834,  ...,  0.3547,  0.3455,  0.27
46],
         [ 0.5444,  0.6049,  1.1027,  ...,  0.4315,  0.2864,  0.45
93],
         [ 0.4824,  0.2651, -0.0567,  ...,  0.4827,  0.1499,  0.41
18]]],
       grad_fn=<SubBackward0>)
Display of D = 1 is
```





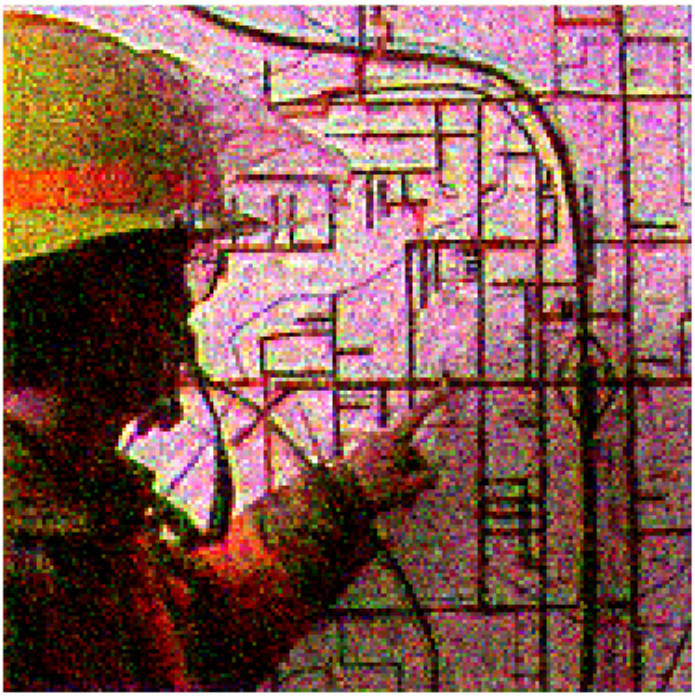
```
Numerical result of D = 2 is
tensor([[[ 0.3755,  0.2773,  0.5530,  ...,  0.3062,  0.2026,  0.31
47],
         [ 0.2975, -0.0969, -0.1228,  ..., -0.1601, -0.0681,  0.03
22],
         [ 0.2547,  0.0234,  0.2458,  ...,  0.0782, -0.0116, -0.10
37],
         ...,
         [ 0.0942, -0.1367,  0.1249,  ..., -0.0579, -0.2833, -0.09
59],
         [-0.0902, -0.2019,  0.0360,  ..., -0.1767, -0.1445, -0.00
95],
         [-0.1336, -0.1160, -0.2305,  ..., -0.1828, -0.1873,  0.08
88]],

        [[-0.2391, -0.1353, -0.0276,  ..., -0.0224, -0.0447, -0.13
40],
         [-0.3324, -0.5906, -0.8151,  ...,  0.0879,  0.0357, -0.16
26],
         [-0.1821, -0.5894, -0.1965,  ...,  0.1053, -0.0532, -0.14
84],
         ...,
         [ 0.3150,  0.2864,  0.5117,  ...,  0.1077, -0.2886, -0.18
93],
         [ 0.3707,  0.5696,  0.2334,  ...,  0.0358,  0.0241, -0.31
12],
         [ 0.0254, -0.2110, -0.1914,  ..., -0.0817, -0.2973, -0.30
35]],

        [[ 0.0832,  0.1622, -0.0368,  ..., -0.1432,  0.0318,  0.05
76],
         [-0.1314,  0.2365,  0.2787,  ..., -0.0818, -0.1754, -0.03
97],
         [-0.1178,  0.1877,  0.0990,  ..., -0.3654, -0.0856, -0.12
84],
         ...,
         [-0.4810, -0.3167, -0.6096,  ..., -0.2273, -0.0767, -0.09
01],
         [-0.5257, -0.6496, -0.7343,  ..., -0.1185, -0.1807,  0.00
43],
         [-0.2402, -0.0996,  0.0548,  ..., -0.1657,  0.2636,  0.11
68]]],
       grad_fn=<SubBackward0>)
Display of D = 2 is
```

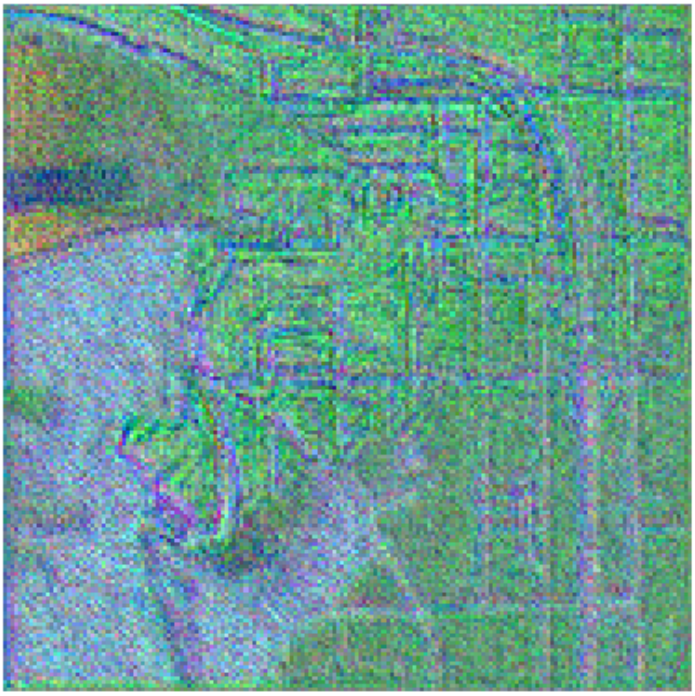




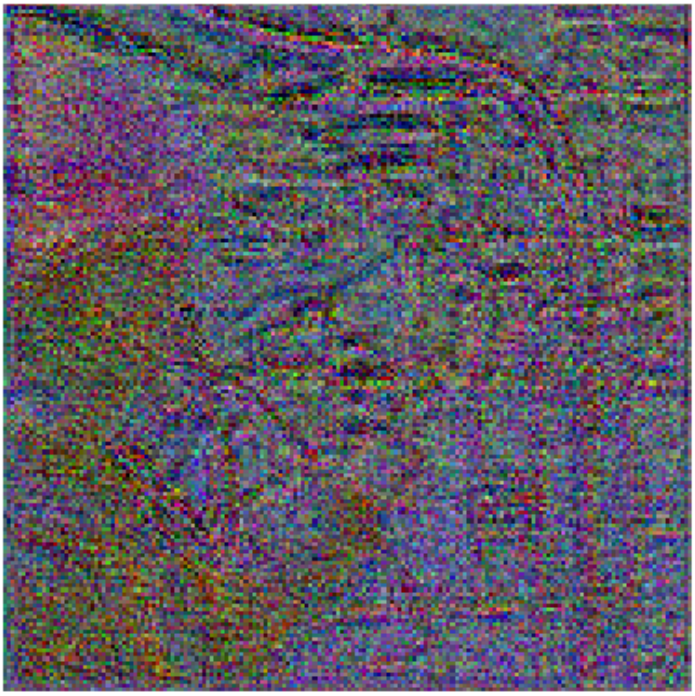
```
Numerical result of D = 4 is
tensor([[[-1.7887e-01, -9.6072e-02, -3.7828e-01,  ..., -1.6629e-01
,
          -8.9043e-02, -1.8285e-01],
         [-1.3692e-01, -3.8259e-01, -8.4187e-02,  ..., -1.9159e-01
,
          -2.2681e-01,  2.4340e-03],
         [-4.7016e-01, -6.4157e-01, -1.7216e-01,  ..., -5.3810e-01
,
          -3.5669e-01, -1.5631e-01],
         ...,
         [-4.6467e-01, -5.9407e-01,  1.2744e-01,  ..., -4.6407e-01
,
          -4.3034e-01, -2.7152e-01],
         [-5.4999e-01, -1.0580e+00, -2.3774e-01,  ..., -3.1334e-01
,
          -3.7341e-01, -1.6800e-04],
         [-4.7676e-01, -2.4757e-01, -2.9400e-02,  ..., -2.6140e-02
,
           1.8551e-01,  3.2417e-02]],

        [[ 1.3817e-01,  1.5705e-01,  1.1878e-01,  ...,  1.7514e-01
,
          -5.8405e-02, -1.1321e-01],
         [ 2.0631e-01,  1.2577e-01, -6.6776e-02,  ...,  2.5635e-01
,
          -7.2907e-02,  1.6035e-01],
         [-9.1022e-02, -1.0286e-01,  1.9157e-01,  ...,  4.9497e-01
```

```
,
           4.6072e-02,  2.4821e-01],
         ...,
         [ 4.9732e-01,  1.3838e-01,  7.2432e-01,  ...,  2.7991e-01
,
           2.9524e-01,  3.3278e-02],
         [ 3.6340e-01, -4.9483e-02, -9.5011e-02,  ...,  3.5761e-01
,
           1.8051e-01,  6.5149e-02],
         [ 1.2798e-01,  4.5271e-01,  2.0284e-01,  ...,  1.5257e-02
,
           3.0448e-02, -7.5885e-02]],

        [[ 1.5227e-01,  1.7567e-01,  3.9348e-01,  ..., -2.0990e-01
,
           8.5232e-02,  1.4604e-01],
         [-5.4254e-02,  6.0439e-02,  9.2185e-02,  ..., -5.0790e-01
,
          -4.4467e-01, -6.4706e-02],
         [ 1.5021e-01,  4.0884e-01,  7.8687e-01,  ..., -5.5141e-01
,
          -6.7188e-02, -1.1042e-01],
         ...,
         [ 5.8945e-01,  1.4148e-02,  8.6864e-02,  ..., -2.2524e-01
,
          -2.0672e-01, -1.3926e-01],
         [ 9.4806e-02,  2.2198e-01,  1.5447e-01,  ..., -1.4459e-01
,
          -2.4066e-01, -2.3433e-02],
         [ 1.2976e-01,  2.4623e-01,  1.3413e-01,  ..., -2.2379e-02
,
           3.2447e-02, -9.1812e-02]]], grad_fn=<SubBackward0>)
Display of D = 4 is
```





```
Numerical result of D = 8 is
tensor([[[-0.0410,  0.0159,  0.0643,  ..., -0.0705, -0.3546, -0.02
63],
         [-0.0062, -0.9252, -0.5514,  ..., -0.3576, -0.4458, -0.03
49],
         [-0.1552, -0.6526, -0.5914,  ..., -0.2355, -0.4055, -0.20
02],
         ...,
         [-0.3559, -0.1293, -0.2418,  ..., -0.5934, -0.0564, -0.40
85],
         [-0.4066, -0.3774, -0.2447,  ..., -0.2039, -0.1796, -0.21
96],
         [-0.4560, -0.3071, -0.1793,  ...,  0.0289,  0.1588,  0.10
58]],

        [[-0.1711, -0.6762, -0.7909,  ...,  0.0096, -0.3502, -0.25
30],
         [-0.2955, -0.7733, -0.6223,  ..., -0.6019, -0.3939, -0.45
62],
         [-0.3490, -0.9723,  0.3166,  ..., -0.4702, -0.5459, -0.23
22],
         ...,
         [ 0.2652, -0.4094, -0.1401,  ...,  0.1142, -0.4296, -0.22
02],
         [-0.4231, -0.3279, -0.3155,  ..., -0.1992, -0.4303,  0.20
35],
         [-0.2885, -0.2776,  0.0537,  ...,  0.0052, -0.4356, -0.14
60]],

        [[-0.0104, -0.3837, -0.4915,  ...,  0.0172, -0.4762, -0.20
17],
         [ 0.0856,  0.2457, -0.1526,  ...,  0.1323, -0.3641, -0.25
29],
         [-0.0412,  0.2568, -0.2486,  ...,  0.0722, -0.2798,  0.14
12],
         ...,
         [-0.2850,  0.2924, -0.3449,  ..., -0.0607, -0.5706, -0.05
91],
         [-0.2444,  0.0808,  0.2233,  ..., -0.3048, -0.1050, -0.04
44],
         [-0.2620, -0.0847, -0.3681,  ..., -0.5653,  0.1592,  0.17
01]]],
       grad_fn=<SubBackward0>)
Display of D = 8 is
```



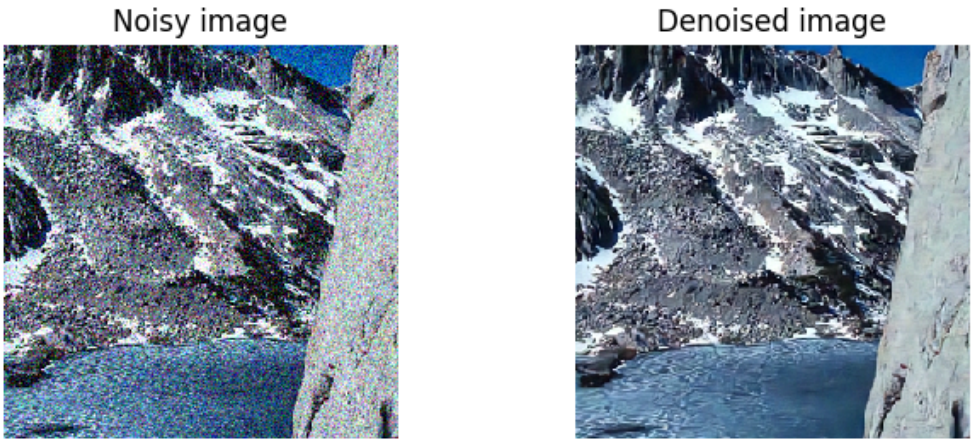
8. I observe that the He initialization makes the network learn more features about residual(negative noisy). With D increasing, the better result the network can generate. The reason is that the weight is no longer zero so its forward pass can have some reasonable response to noisy image.

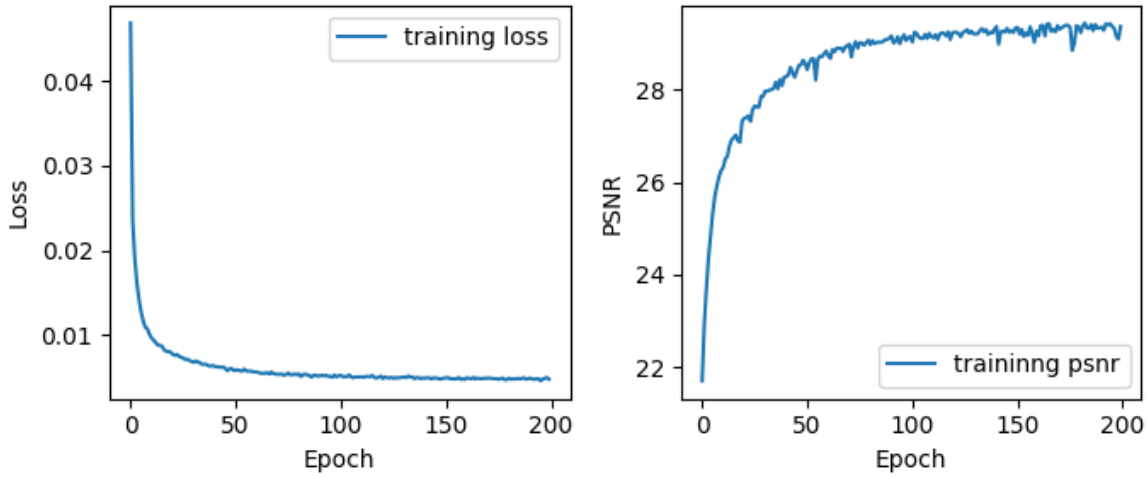
During back-propagation, it can prevent units from learning the same things and promote the dynamic of the outputs and gradients at each layers.

In [9]:
```
## 9.
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 = y.shape[1] * y.shape[2] * y.shape[3] # consider the wid
th and height and three color
        y1 = y.view(y.shape[0],-1)  # flatten to (batch size * feat
ures)
        d1 = d.view(d.shape[0],-1)
        norm = torch.norm((y1-d1), p=2, dim=1).pow(2)
        self.running_psnr += (10 * torch.log10((4 * n)/norm)).mean(
)  # find average insides batch

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

In [15]:
```
## 10.
lr = 1e-3
net = DnCNN(6)
net = net.to(device)
adam = torch.optim.Adam(net.parameters(), lr=lr)
stats_manager = DenoisingStatsManager()
exp1 = nt.Experiment(net, train_set, test_set, adam, stats_manager,
               output_dir="denoising1", batch_size=4, perform_valid
ation_during_training=True)
```

In [42]:
```
## 11.
def myimshow1(image, ax=plt):
    if image.requires_grad == True:
        image = image.to('cpu').detach().numpy()
    else:
        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

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()
    myimshow1(noisy, ax=axes[0][0])
    axes[0][0].set_title('Noisy image')
    myimshow1(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.e
poch)],
                    label="training loss")
    axes[1][1].plot([exp.history[k][1]['PSNR'] for k in range(exp.e
poch)],
                    label="traininng psnr")
    axes[1][0].legend()
    axes[1][0].set_ylabel('Loss')
    axes[1][0].set_xlabel('Epoch')

    axes[1][1].legend()
    axes[1][1].set_ylabel('PSNR')
    axes[1][1].set_xlabel('Epoch')
    plt.tight_layout()
    fig.canvas.draw()

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






```
Start/Continue training from epoch 0
Epoch 1 (Time: 7.02s)
Epoch 2 (Time: 7.47s)
Epoch 3 (Time: 7.42s)
Epoch 4 (Time: 7.21s)
Epoch 5 (Time: 7.28s)
Epoch 6 (Time: 7.23s)
Epoch 7 (Time: 7.28s)
Epoch 8 (Time: 7.16s)
Epoch 9 (Time: 7.33s)
Epoch 10 (Time: 7.28s)
Epoch 11 (Time: 7.15s)
Epoch 12 (Time: 7.12s)
Epoch 13 (Time: 7.20s)
Epoch 14 (Time: 7.27s)
Epoch 15 (Time: 7.33s)
Epoch 16 (Time: 7.15s)
Epoch 17 (Time: 7.59s)
Epoch 18 (Time: 7.37s)
Epoch 19 (Time: 7.29s)
Epoch 20 (Time: 7.06s)
Epoch 21 (Time: 7.16s)
Epoch 22 (Time: 7.27s)
Epoch 23 (Time: 7.19s)
```

```
Epoch 24 (Time: 7.18s)
Epoch 25 (Time: 7.30s)
Epoch 26 (Time: 7.31s)
Epoch 27 (Time: 7.26s)
Epoch 28 (Time: 7.37s)
Epoch 29 (Time: 7.20s)
Epoch 30 (Time: 7.36s)
Epoch 31 (Time: 7.27s)
Epoch 32 (Time: 7.17s)
Epoch 33 (Time: 7.22s)
Epoch 34 (Time: 7.27s)
Epoch 35 (Time: 7.27s)
Epoch 36 (Time: 7.29s)
Epoch 37 (Time: 7.24s)
Epoch 38 (Time: 7.31s)
Epoch 39 (Time: 7.37s)
Epoch 40 (Time: 7.11s)
Epoch 41 (Time: 7.17s)
Epoch 42 (Time: 7.33s)
Epoch 43 (Time: 7.37s)
Epoch 44 (Time: 7.20s)
Epoch 45 (Time: 7.39s)
Epoch 46 (Time: 7.35s)
Epoch 47 (Time: 7.25s)
Epoch 48 (Time: 7.10s)
Epoch 49 (Time: 7.20s)
Epoch 50 (Time: 7.22s)
Epoch 51 (Time: 7.20s)
Epoch 52 (Time: 7.14s)
Epoch 53 (Time: 7.22s)
Epoch 54 (Time: 7.16s)
Epoch 55 (Time: 7.12s)
Epoch 56 (Time: 6.95s)
Epoch 57 (Time: 7.23s)
Epoch 58 (Time: 7.14s)
Epoch 59 (Time: 7.17s)
Epoch 60 (Time: 7.10s)
Epoch 61 (Time: 7.19s)
Epoch 62 (Time: 7.13s)
Epoch 63 (Time: 7.28s)
Epoch 64 (Time: 7.03s)
Epoch 65 (Time: 7.25s)
Epoch 66 (Time: 7.20s)
Epoch 67 (Time: 7.20s)
Epoch 68 (Time: 6.96s)
Epoch 69 (Time: 7.43s)
Epoch 70 (Time: 7.30s)
Epoch 71 (Time: 7.16s)
Epoch 72 (Time: 7.18s)
Epoch 73 (Time: 7.04s)
Epoch 74 (Time: 7.19s)
Epoch 75 (Time: 7.12s)
Epoch 76 (Time: 7.18s)
```

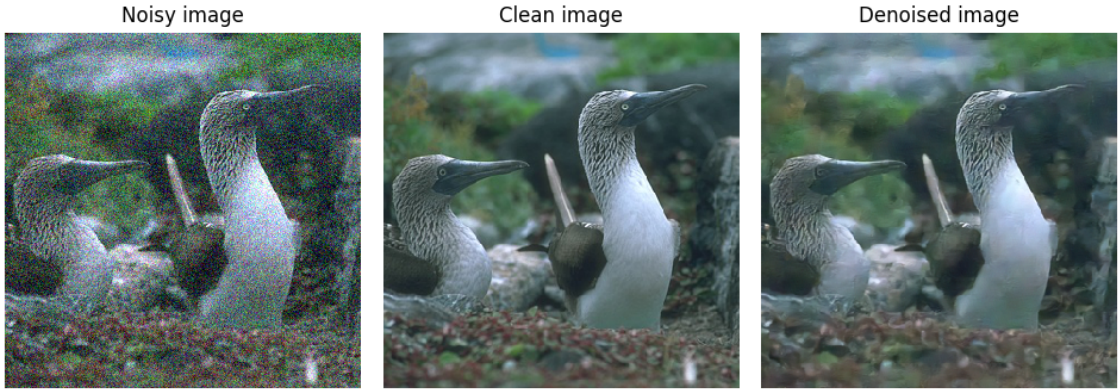
```
Epoch 77 (Time: 7.26s)
Epoch 78 (Time: 7.24s)
Epoch 79 (Time: 7.15s)
Epoch 80 (Time: 7.11s)
Epoch 81 (Time: 8.08s)
Epoch 82 (Time: 7.24s)
Epoch 83 (Time: 7.06s)
Epoch 84 (Time: 6.99s)
Epoch 85 (Time: 7.13s)
Epoch 86 (Time: 7.47s)
Epoch 87 (Time: 7.18s)
Epoch 88 (Time: 6.95s)
Epoch 89 (Time: 7.05s)
Epoch 90 (Time: 7.12s)
Epoch 91 (Time: 7.15s)
Epoch 92 (Time: 7.19s)
Epoch 93 (Time: 7.24s)
Epoch 94 (Time: 7.13s)
Epoch 95 (Time: 7.02s)
Epoch 96 (Time: 7.06s)
Epoch 97 (Time: 7.18s)
Epoch 98 (Time: 7.25s)
Epoch 99 (Time: 7.18s)
Epoch 100 (Time: 7.34s)
Epoch 101 (Time: 6.95s)
Epoch 102 (Time: 7.04s)
Epoch 103 (Time: 7.15s)
Epoch 104 (Time: 7.11s)
Epoch 105 (Time: 7.97s)
Epoch 106 (Time: 7.16s)
Epoch 107 (Time: 7.04s)
Epoch 108 (Time: 7.06s)
Epoch 109 (Time: 7.36s)
Epoch 110 (Time: 7.04s)
Epoch 111 (Time: 7.14s)
Epoch 112 (Time: 7.02s)
Epoch 113 (Time: 7.26s)
Epoch 114 (Time: 7.09s)
Epoch 115 (Time: 6.97s)
Epoch 116 (Time: 6.99s)
Epoch 117 (Time: 7.30s)
Epoch 118 (Time: 7.19s)
Epoch 119 (Time: 7.10s)
Epoch 120 (Time: 7.13s)
Epoch 121 (Time: 7.07s)
Epoch 122 (Time: 7.05s)
Epoch 123 (Time: 7.16s)
Epoch 124 (Time: 7.13s)
Epoch 125 (Time: 7.07s)
Epoch 126 (Time: 7.14s)
Epoch 127 (Time: 7.06s)
Epoch 128 (Time: 7.04s)
Epoch 129 (Time: 7.16s)
```

```
Epoch 130 (Time: 7.16s)
Epoch 131 (Time: 7.11s)
Epoch 132 (Time: 7.09s)
Epoch 133 (Time: 7.04s)
Epoch 134 (Time: 7.04s)
Epoch 135 (Time: 6.91s)
Epoch 136 (Time: 7.15s)
Epoch 137 (Time: 7.09s)
Epoch 138 (Time: 7.12s)
Epoch 139 (Time: 7.08s)
Epoch 140 (Time: 7.02s)
Epoch 141 (Time: 7.06s)
Epoch 142 (Time: 7.52s)
Epoch 143 (Time: 7.19s)
Epoch 144 (Time: 7.17s)
Epoch 145 (Time: 7.02s)
Epoch 146 (Time: 7.11s)
Epoch 147 (Time: 7.09s)
Epoch 148 (Time: 7.17s)
Epoch 149 (Time: 8.48s)
Epoch 150 (Time: 7.17s)
Epoch 151 (Time: 7.15s)
Epoch 152 (Time: 7.20s)
Epoch 153 (Time: 7.02s)
Epoch 154 (Time: 7.16s)
Epoch 155 (Time: 7.10s)
Epoch 156 (Time: 7.07s)
Epoch 157 (Time: 7.09s)
Epoch 158 (Time: 6.86s)
Epoch 159 (Time: 7.04s)
Epoch 160 (Time: 7.03s)
Epoch 161 (Time: 7.06s)
Epoch 162 (Time: 7.14s)
Epoch 163 (Time: 7.15s)
Epoch 164 (Time: 7.23s)
Epoch 165 (Time: 7.11s)
Epoch 166 (Time: 6.94s)
Epoch 167 (Time: 7.10s)
Epoch 168 (Time: 7.07s)
Epoch 169 (Time: 7.07s)
Epoch 170 (Time: 7.17s)
Epoch 171 (Time: 7.17s)
Epoch 172 (Time: 7.07s)
Epoch 173 (Time: 7.15s)
Epoch 174 (Time: 7.21s)
Epoch 175 (Time: 7.10s)
Epoch 176 (Time: 7.10s)
Epoch 177 (Time: 7.09s)
Epoch 178 (Time: 7.04s)
Epoch 179 (Time: 7.07s)
Epoch 180 (Time: 7.19s)
Epoch 181 (Time: 7.13s)
Epoch 182 (Time: 7.10s)
```

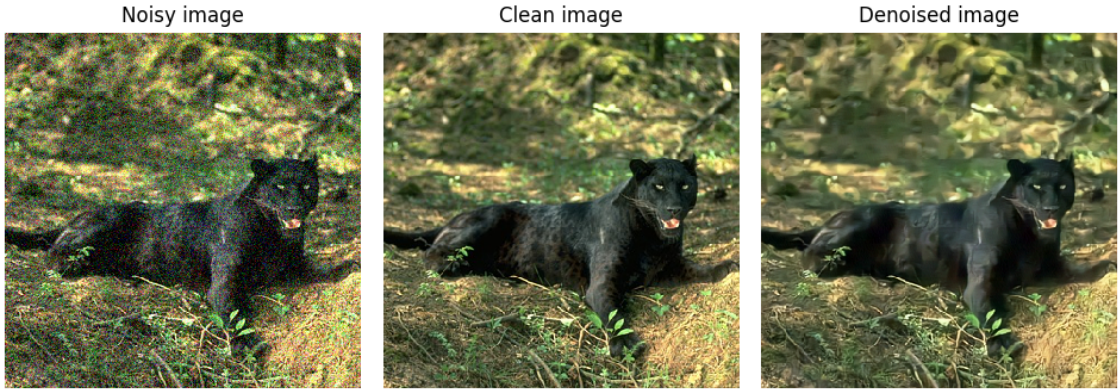
```
Epoch 183 (Time: 7.19s)
Epoch 184 (Time: 7.15s)
Epoch 185 (Time: 7.10s)
Epoch 186 (Time: 7.09s)
Epoch 187 (Time: 7.16s)
Epoch 188 (Time: 7.21s)
Epoch 189 (Time: 7.19s)
Epoch 190 (Time: 7.15s)
Epoch 191 (Time: 7.13s)
Epoch 192 (Time: 7.12s)
Epoch 193 (Time: 7.15s)
Epoch 194 (Time: 7.09s)
Epoch 195 (Time: 8.55s)
Epoch 196 (Time: 7.20s)
Epoch 197 (Time: 6.97s)
Epoch 198 (Time: 6.93s)
Epoch 199 (Time: 7.10s)
Epoch 200 (Time: 7.14s)
Finish training for 200 epochs
```

In [11]:
```
## 12.
def myimshow1(image, ax=plt):
    if image.requires_grad == True:
        image = image.to('cpu').detach().numpy()
    else:
        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

def plot_test(exp, clean, noisy):
    fig, axes = plt.subplots(ncols=3, nrows=1, figsize=(10,9), shar
ex = 'all', sharey = 'all')
    with torch.no_grad():
            denoised = exp.net(noisy[np.newaxis].to(exp.net.device)
)[0]
    axes[0].clear()
    axes[1].clear()
    axes[2].clear()
    myimshow1(noisy, ax=axes[0])
    axes[0].set_title('Noisy image')
    myimshow1(clean, ax=axes[1])
    axes[1].set_title('Clean image')
    myimshow1(denoised, ax=axes[2])
    axes[2].set_title('Denoised image')
    plt.tight_layout()
    fig.canvas.draw()

pickind = np.random.permutation(100)[:5]
for i in range(len(pickind)):
    test_img = test_set[pickind[i]]
    noisy = test_img[0]
    clean = test_img[1]
    plot_test(exp1, clean, noisy)
```

Noisy image
Clean image
Denoised image


Noisy image
Clean image
Denoised image

Noisy image
Clean image
Denoised image


Noisy image
Clean image
Denoised image

From visual inspection, the denoised image has recovered the clean image in a decent quality.

However, The network makes the output image looks more "smooth" compared with clean image. Detailed texture is lost after denoising.

In [26]:
```
## 13.
pytorch_total_params = sum(p.numel() for p in net.parameters())
print('Number of trainable parameters of the DnCNN(6) is ', pytorch
_total_params)
```

```
Number of trainable parameters of the DnCNN(6) is  225859
```

The number of trainable parameters of DnCNN(D) is $23649 + 67 + D(64649 + 64 + 2\ 64) = 37056D + 3523$

The number of total parameters of DnCNN(D) is $23649 + 67 + D(64649 + 64 + 4\ 64) = 37184D + 3523$

The receptive field of DnCNN(D) is $1 + 2(D + 2) = 2D + 5$

In [ ]:
```
## 14.
```

From the receptive field equation deducted above, we know that: D >= 14. In this case, trainable parameters would be 522307.

The number of parameters would increase a lot according to the equation of parameter number and the computation time would also increase. This network cannot denoising efficiently while remain a fast computation and deep architecture. New network need to be implement.

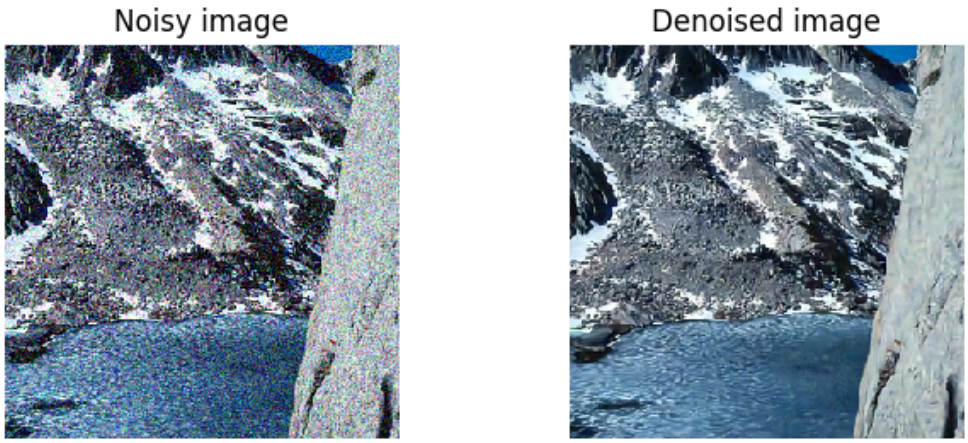
In [16]:
```
## 15.
class UDnCNN(NNRegressor):

    def __init__(self, D, C=64):  # D has to be even
        super(UDnCNN, self).__init__()
        self.D = int((D - 2)/2)     # the repeat part is layer with
pooling/unpooling
        # conv
        self.conv = nn.ModuleList()
        self.conv.append(nn.Conv2d(3, C, 3, padding = (1,1)))
        for i in range(self.D):
            self.conv.append(nn.Conv2d(C, C, 3, padding = (1,1)))
        self.conv.append(nn.Conv2d(C, C, 3, padding = (1,1)))
        self.conv.append(nn.Conv2d(C, C, 3, padding = (1,1)))
        for i in range(self.D):
            self.conv.append(nn.Conv2d(C, C, 3, padding = (1,1)))
        self.conv.append(nn.Conv2d(C, 3, 3, padding = (1,1)))
        # initialize of conv
        for i in range(D+1):  # without ReLu not initialize
            nn.init.kaiming_normal_(self.conv[i].weight.data, mode=
'fan_in', nonlinearity='relu')

        # bn
        self.bn = nn.ModuleList()
        for k in range(D):
            self.bn.append(nn.BatchNorm2d(C, C))
        # initialize of bn
        for k in range(D):
            nn.init.constant_(self.bn[k].weight.data, 1.25 * np.sqr
t(C))


    def forward(self, x):
        D = int(self.D * 2 + 2)
        h = F.relu(self.conv[0](x))
        # pooling
        poolind = []
        outsize = []
        feature = []
        outsize.append(h.size())
        feature.append(h)
```

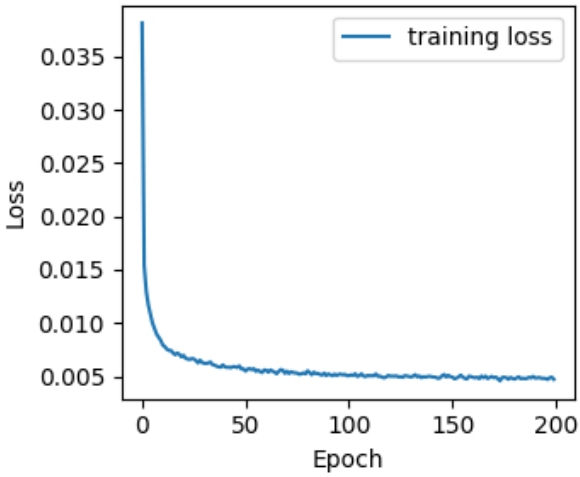
```
        for j in range(self.D):
            h = F.relu(self.bn[j](self.conv[j+1](h)))
            h, index = F.max_pool2d(h, kernel_size = 2, stride=2, r
eturn_indices=True)
            poolind.append(index)
            if j != self.D-1:
                feature.append(h)
                outsize.append(h.size())
        # reverse poolind and outsize and feature
        poolind.reverse()
        outsize.reverse()
        feature.reverse()
        h1 = F.relu(self.bn[j+1](self.conv[j+2](h)))
        h2 = F.relu(self.bn[j+2](self.conv[j+3](h1)))
        h = (h + h2)/(2**0.5)
        # unpooling
        for l in range(self.D):
            h = F.max_unpool2d(h, poolind[l], 2, stride=2, output_s
ize=outsize[l])
            h = F.relu(self.bn[l+j+3](self.conv[l+j+4](h)))
            h = (h + feature[l])/(2**0.5)
        y = self.conv[D+1](h) + x
        return y
```

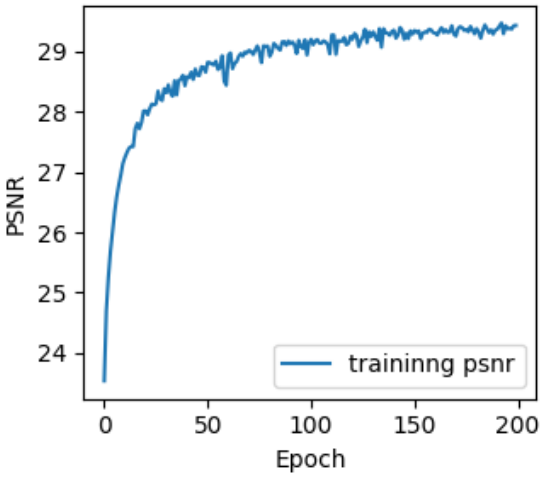
In [17]:

```
## 16.
lr = 1e-3
net = UDnCNN(6)
net = net.to(device)
adam = torch.optim.Adam(net.parameters(), lr=lr)
stats_manager = DenoisingStatsManager()
exp2 = nt.Experiment(net, train_set, test_set, adam, stats_manager,
               output_dir="denoising2", batch_size=4, perform_valid
ation_during_training=True)
```

In [54]:
```
def myimshow1(image, ax=plt):
    if image.requires_grad == True:
        image = image.to('cpu').detach().numpy()
    else:
        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

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()
    myimshow1(noisy, ax=axes[0][0])
    axes[0][0].set_title('Noisy image')
    myimshow1(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.e
poch)],
                    label="training loss")
    axes[1][1].plot([exp.history[k][1]['PSNR'] for k in range(exp.e
poch)],
                    label="traininng psnr")
    axes[1][0].legend()
    axes[1][0].set_ylabel('Loss')
    axes[1][0].set_xlabel('Epoch')

    axes[1][1].legend()
    axes[1][1].set_ylabel('PSNR')
    axes[1][1].set_xlabel('Epoch')
    plt.tight_layout()
    fig.canvas.draw()

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








```
Start/Continue training from epoch 0
Epoch 1 (Time: 6.18s)
Epoch 2 (Time: 6.19s)
Epoch 3 (Time: 6.22s)
Epoch 4 (Time: 6.31s)
Epoch 5 (Time: 6.41s)
Epoch 6 (Time: 6.27s)
Epoch 7 (Time: 6.31s)
Epoch 8 (Time: 6.26s)
Epoch 9 (Time: 6.24s)
Epoch 10 (Time: 6.27s)
Epoch 11 (Time: 6.19s)
Epoch 12 (Time: 6.18s)
Epoch 13 (Time: 6.23s)
Epoch 14 (Time: 6.15s)
Epoch 15 (Time: 6.28s)
Epoch 16 (Time: 6.19s)
Epoch 17 (Time: 6.17s)
Epoch 18 (Time: 6.16s)
Epoch 19 (Time: 6.23s)
Epoch 20 (Time: 6.07s)
Epoch 21 (Time: 6.28s)
Epoch 22 (Time: 6.29s)
Epoch 23 (Time: 6.21s)
```

```
Epoch 24 (Time: 6.28s)
Epoch 25 (Time: 6.34s)
Epoch 26 (Time: 6.29s)
Epoch 27 (Time: 6.28s)
Epoch 28 (Time: 6.19s)
Epoch 29 (Time: 6.25s)
Epoch 30 (Time: 6.14s)
Epoch 31 (Time: 6.12s)
Epoch 32 (Time: 6.15s)
Epoch 33 (Time: 6.23s)
Epoch 34 (Time: 6.16s)
Epoch 35 (Time: 6.25s)
Epoch 36 (Time: 6.18s)
Epoch 37 (Time: 5.97s)
Epoch 38 (Time: 6.20s)
Epoch 39 (Time: 6.19s)
Epoch 40 (Time: 6.17s)
Epoch 41 (Time: 6.31s)
Epoch 42 (Time: 6.04s)
Epoch 43 (Time: 6.08s)
Epoch 44 (Time: 6.03s)
Epoch 45 (Time: 6.31s)
Epoch 46 (Time: 6.16s)
Epoch 47 (Time: 6.22s)
Epoch 48 (Time: 6.40s)
Epoch 49 (Time: 6.65s)
Epoch 50 (Time: 6.27s)
Epoch 51 (Time: 6.03s)
Epoch 52 (Time: 6.34s)
Epoch 53 (Time: 6.25s)
Epoch 54 (Time: 6.67s)
Epoch 55 (Time: 6.25s)
Epoch 56 (Time: 6.35s)
Epoch 57 (Time: 6.17s)
Epoch 58 (Time: 6.44s)
Epoch 59 (Time: 6.23s)
Epoch 60 (Time: 6.20s)
Epoch 61 (Time: 6.18s)
Epoch 62 (Time: 6.21s)
Epoch 63 (Time: 6.22s)
Epoch 64 (Time: 6.14s)
Epoch 65 (Time: 6.27s)
Epoch 66 (Time: 6.22s)
Epoch 67 (Time: 6.27s)
Epoch 68 (Time: 6.34s)
Epoch 69 (Time: 6.23s)
Epoch 70 (Time: 6.25s)
Epoch 71 (Time: 6.23s)
Epoch 72 (Time: 6.33s)
Epoch 73 (Time: 6.11s)
Epoch 74 (Time: 6.14s)
Epoch 75 (Time: 6.21s)
Epoch 76 (Time: 6.34s)
```

```
Epoch 77 (Time: 6.21s)
Epoch 78 (Time: 6.05s)
Epoch 79 (Time: 6.29s)
Epoch 80 (Time: 6.38s)
Epoch 81 (Time: 6.32s)
Epoch 82 (Time: 6.29s)
Epoch 83 (Time: 6.24s)
Epoch 84 (Time: 6.29s)
Epoch 85 (Time: 6.19s)
Epoch 86 (Time: 6.35s)
Epoch 87 (Time: 6.09s)
Epoch 88 (Time: 6.25s)
Epoch 89 (Time: 6.20s)
Epoch 90 (Time: 6.26s)
Epoch 91 (Time: 6.19s)
Epoch 92 (Time: 5.79s)
Epoch 93 (Time: 6.07s)
Epoch 94 (Time: 6.20s)
Epoch 95 (Time: 6.24s)
Epoch 96 (Time: 6.07s)
Epoch 97 (Time: 6.28s)
Epoch 98 (Time: 6.22s)
Epoch 99 (Time: 6.15s)
Epoch 100 (Time: 6.27s)
Epoch 101 (Time: 6.27s)
Epoch 102 (Time: 6.16s)
Epoch 103 (Time: 6.18s)
Epoch 104 (Time: 6.32s)
Epoch 105 (Time: 6.14s)
Epoch 106 (Time: 5.92s)
Epoch 107 (Time: 6.37s)
Epoch 108 (Time: 6.44s)
Epoch 109 (Time: 6.24s)
Epoch 110 (Time: 6.31s)
Epoch 111 (Time: 6.26s)
Epoch 112 (Time: 6.35s)
Epoch 113 (Time: 6.24s)
Epoch 114 (Time: 6.35s)
Epoch 115 (Time: 6.14s)
Epoch 116 (Time: 6.28s)
Epoch 117 (Time: 6.21s)
Epoch 118 (Time: 6.28s)
Epoch 119 (Time: 6.69s)
Epoch 120 (Time: 6.25s)
Epoch 121 (Time: 6.24s)
Epoch 122 (Time: 6.49s)
Epoch 123 (Time: 6.21s)
Epoch 124 (Time: 6.06s)
Epoch 125 (Time: 6.19s)
Epoch 126 (Time: 6.25s)
Epoch 127 (Time: 6.16s)
Epoch 128 (Time: 6.21s)
Epoch 129 (Time: 6.36s)
```

```
Epoch 130 (Time: 6.21s)
Epoch 131 (Time: 6.21s)
Epoch 132 (Time: 6.30s)
Epoch 133 (Time: 6.12s)
Epoch 134 (Time: 6.18s)
Epoch 135 (Time: 6.36s)
Epoch 136 (Time: 6.29s)
Epoch 137 (Time: 6.25s)
Epoch 138 (Time: 6.27s)
Epoch 139 (Time: 6.36s)
Epoch 140 (Time: 6.28s)
Epoch 141 (Time: 6.26s)
Epoch 142 (Time: 6.21s)
Epoch 143 (Time: 6.23s)
Epoch 144 (Time: 6.28s)
Epoch 145 (Time: 6.17s)
Epoch 146 (Time: 6.30s)
Epoch 147 (Time: 6.15s)
Epoch 148 (Time: 6.08s)
Epoch 149 (Time: 6.04s)
Epoch 150 (Time: 6.23s)
Epoch 151 (Time: 6.08s)
Epoch 152 (Time: 6.24s)
Epoch 153 (Time: 5.99s)
Epoch 154 (Time: 6.20s)
Epoch 155 (Time: 6.17s)
Epoch 156 (Time: 6.18s)
Epoch 157 (Time: 6.32s)
Epoch 158 (Time: 6.21s)
Epoch 159 (Time: 6.25s)
Epoch 160 (Time: 6.28s)
Epoch 161 (Time: 6.29s)
Epoch 162 (Time: 6.30s)
Epoch 163 (Time: 6.30s)
Epoch 164 (Time: 6.33s)
Epoch 165 (Time: 6.21s)
Epoch 166 (Time: 6.39s)
Epoch 167 (Time: 6.30s)
Epoch 168 (Time: 6.23s)
Epoch 169 (Time: 6.20s)
Epoch 170 (Time: 6.38s)
Epoch 171 (Time: 6.19s)
Epoch 172 (Time: 6.28s)
Epoch 173 (Time: 6.24s)
Epoch 174 (Time: 6.22s)
Epoch 175 (Time: 6.27s)
Epoch 176 (Time: 6.26s)
Epoch 177 (Time: 6.19s)
Epoch 178 (Time: 6.28s)
Epoch 179 (Time: 6.01s)
Epoch 180 (Time: 6.27s)
Epoch 181 (Time: 6.14s)
Epoch 182 (Time: 6.12s)
```

```
Epoch 183 (Time: 6.08s)
Epoch 184 (Time: 6.28s)
Epoch 185 (Time: 6.21s)
Epoch 186 (Time: 6.24s)
Epoch 187 (Time: 6.31s)
Epoch 188 (Time: 6.12s)
Epoch 189 (Time: 6.09s)
Epoch 190 (Time: 5.95s)
Epoch 191 (Time: 6.16s)
Epoch 192 (Time: 6.23s)
Epoch 193 (Time: 6.23s)
Epoch 194 (Time: 6.38s)
Epoch 195 (Time: 6.20s)
Epoch 196 (Time: 6.23s)
Epoch 197 (Time: 6.20s)
Epoch 198 (Time: 6.30s)
Epoch 199 (Time: 6.22s)
Epoch 200 (Time: 6.32s)
Finish training for 200 epochs
```

In [22]:

```
## 17.
pytorch_total_params = sum(p.numel() for p in net.parameters())
print('Number of trainable parameters of the UDnCNN(6) is ', pytorc
h_total_params)
```

```
Number of trainable parameters of the UDnCNN(6) is  225859
```

The number of trainable parameters of the UDnCNN(D) is *23649* + *67* + *D*(*64649* + 64 + 2 *64)* = *37056*D + 3523

The receptive field of UDnCNN(D) is 2^((D+4)/2) + 1

I don't expect it can beat DnCNN(D) in both speed and quality. The UDnCNN(D) has the same number of trainable parameters with DnCNN(D), so they have simiiar training speed. The denoising quality of UDnCNN(D) would be worse than DnCNN(D) since its pooling layer would lose detailed texture information, leading to a worse PSNR.

In [55]:

```
## 18.
print('DnCNN(6)\'s performance: ', exp1.evaluate())
print('UDnCNN(6)\'s performance: ', exp2.evaluate())
```

```
DnCNN(6)'s performance:  {'loss': 0.005034405728802085, 'PSNR': te
nsor(29.4287, device='cuda:0')}
UDnCNN(6)'s performance:  {'loss': 0.0051128056086599824, 'PSNR':
tensor(29.3628, device='cuda:0')}
```

The result shows that the performance of UDnCNN(6) is similiar to DnCNN(6). The loss of UDnCNN(6) is worse than DnCNN(6) and the PSNR is also better than DnCNN(6). Although UDnCNN(6) have a larger receptive field, its pooling layers lead to spatial information loss. DnCNN(6) does not lose any information via convolution.

In [18]:
```
## 19.
class DUDnCNN(NNRegressor):

    def __init__(self, D, C=64):  # D has to be even
        super(DUDnCNN, self).__init__()
        self.D = int((D - 2)/2)     # the repeat part is layer with
pooling/unpooling
        # conv
        self.conv = nn.ModuleList()
        self.conv.append(nn.Conv2d(3, C, 3, padding = (1,1)))
        for k in range(self.D):
            dilation = 2**(k)
            self.conv.append(nn.Conv2d(C, C, 3, padding = dilation,
dilation = dilation))
        k += 1
        dilation = 2**(k)
        self.conv.append(nn.Conv2d(C, C, 3, padding = dilation, dil
ation = dilation))
        self.conv.append(nn.Conv2d(C, C, 3, padding = dilation, dil
ation = dilation))
        for l in range(self.D):
            dilation = 2**(k - (l+1))
            self.conv.append(nn.Conv2d(C, C, 3, padding = dilation,
dilation = dilation))
        self.conv.append(nn.Conv2d(C, 3, 3, padding = (1,1)))
        # initialize of conv
        for i in range(D+1):  # without ReLu not initialize
            nn.init.kaiming_normal_(self.conv[i].weight.data, mode=
'fan_in', nonlinearity='relu')

        # bn
        self.bn = nn.ModuleList()
        for k in range(D):
            self.bn.append(nn.BatchNorm2d(C, C))
        # initialize of bn
        for k in range(D):
            nn.init.constant_(self.bn[k].weight.data, 1.25 * np.sqr
t(C))


    def forward(self, x):
        D = int(self.D * 2 + 2)
        h = F.relu(self.conv[0](x))
        # pooling
        feature = []
        feature.append(h)
```

```
        for j in range(self.D):
            h = F.relu(self.bn[j](self.conv[j+1](h)))
            if j != self.D-1:
                feature.append(h)
        # reverse feature
        feature.reverse()
        h1 = F.relu(self.bn[j+1](self.conv[j+2](h)))
        h2 = F.relu(self.bn[j+2](self.conv[j+3](h1)))
        h = (h + h2)/(2**0.5)
        # unpooling
        for l in range(self.D):
            h = F.relu(self.bn[l+j+3](self.conv[l+j+4](h)))
            h = (h + feature[l])/(2**0.5)
        y = self.conv[D+1](h) + x
        return y
```

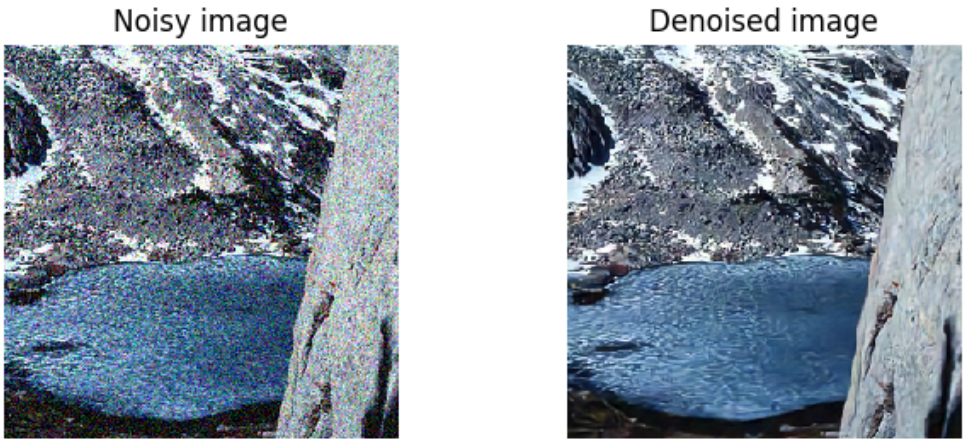
In [48]:

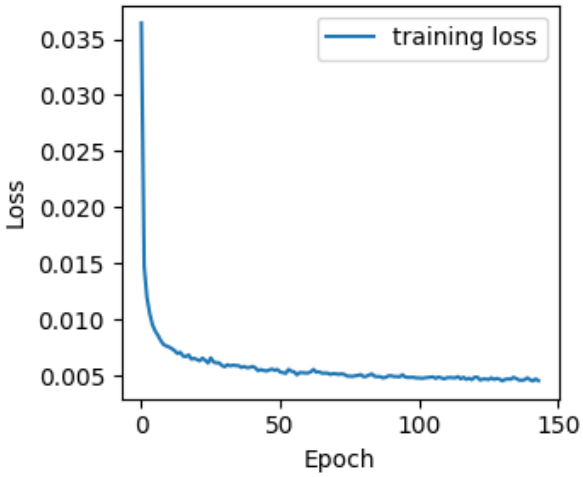
```
## 20.
class DUDnCNN(NNRegressor):

    def __init__(self, D, C=64):  # D has to be even
        super(DUDnCNN, self).__init__()
        self.D = int((D - 2)/2)     # the repeat part is layer with
pooling/unpooling
        # conv
        self.conv = nn.ModuleList()
        self.conv.append(nn.Conv2d(3, C, 3, padding = (1,1)))
        for k in range(self.D):
            dilation = 2**(k)
            self.conv.append(nn.Conv2d(C, C, 3, padding = dilation,
dilation = dilation))
        k += 1
        dilation = 2**(k)
        self.conv.append(nn.Conv2d(C, C, 3, padding = dilation, dil
ation = dilation))
        self.conv.append(nn.Conv2d(C, C, 3, padding = dilation, dil
ation = dilation))
        for l in range(self.D):
            dilation = 2**(k - (l+1))
            self.conv.append(nn.Conv2d(C, C, 3, padding = dilation,
dilation = dilation))
        self.conv.append(nn.Conv2d(C, 3, 3, padding = (1,1)))
        # initialize of conv
        for i in range(D+1):  # without ReLu not initialize
            nn.init.kaiming_normal_(self.conv[i].weight.data, mode=
'fan_in', nonlinearity='relu')

        # bn
        self.bn = nn.ModuleList()
        for k in range(D):
            self.bn.append(nn.BatchNorm2d(C, C))
        # initialize of bn
        for k in range(D):
            nn.init.constant_(self.bn[k].weight.data, 1.25 * np.sqr
```

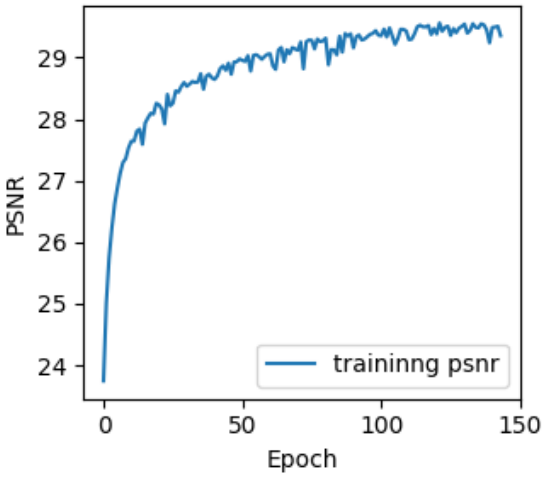
```
t(C))


    def forward(self, x):
        D = int(self.D * 2 + 2)
        h = F.relu(self.conv[0](x))
        # pooling
        feature = []
        feature.append(h)
        torch.backends.cudnn.benchmark=True
        for j in range(self.D):
            h = F.relu(self.bn[j](self.conv[j+1](h)))
            if j != self.D-1:
                feature.append(h)
        torch.backends.cudnn.benchmark=False
        # reverse feature
        feature.reverse()
        torch.backends.cudnn.benchmark=True
        h1 = F.relu(self.bn[j+1](self.conv[j+2](h)))
        h2 = F.relu(self.bn[j+2](self.conv[j+3](h1)))
        h = (h + h2)/(2**0.5)
        # unpooling
        for l in range(self.D):
            h = F.relu(self.bn[l+j+3](self.conv[l+j+4](h)))
            h = (h + feature[l])/(2**0.5)
        torch.backends.cudnn.benchmark=False
        y = self.conv[D+1](h) + x
        return y
```

In [ ]:

```
## 21.
lr = 1e-3
net = DUDnCNN(6)
net = net.to(device)
adam = torch.optim.Adam(net.parameters(), lr=lr)
stats_manager = DenoisingStatsManager()
exp3 = nt.Experiment(net, train_set, test_set, adam, stats_manager,
               output_dir="denoising3", batch_size=4, perform_valid
ation_during_training=True)

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








```
Start/Continue training from epoch 0
Epoch 1 (Time: 8.47s)
Epoch 2 (Time: 8.45s)
Epoch 3 (Time: 8.59s)
Epoch 4 (Time: 8.36s)
Epoch 5 (Time: 8.55s)
Epoch 6 (Time: 8.66s)
Epoch 7 (Time: 8.55s)
Epoch 8 (Time: 8.55s)
Epoch 9 (Time: 8.48s)
Epoch 10 (Time: 8.56s)
Epoch 11 (Time: 8.52s)
Epoch 12 (Time: 8.51s)
Epoch 13 (Time: 8.58s)
Epoch 14 (Time: 8.65s)
Epoch 15 (Time: 8.56s)
Epoch 16 (Time: 8.54s)
Epoch 17 (Time: 8.61s)
Epoch 18 (Time: 8.54s)
Epoch 19 (Time: 8.46s)
Epoch 20 (Time: 8.54s)
Epoch 21 (Time: 8.44s)
Epoch 22 (Time: 8.50s)
Epoch 23 (Time: 8.92s)
```

```
Epoch 24 (Time: 8.55s)
Epoch 25 (Time: 8.51s)
Epoch 26 (Time: 8.65s)
Epoch 27 (Time: 8.54s)
Epoch 28 (Time: 8.53s)
Epoch 29 (Time: 8.48s)
Epoch 30 (Time: 8.54s)
Epoch 31 (Time: 8.57s)
Epoch 32 (Time: 8.55s)
Epoch 33 (Time: 8.56s)
Epoch 34 (Time: 8.73s)
Epoch 35 (Time: 8.67s)
Epoch 36 (Time: 8.41s)
Epoch 37 (Time: 8.47s)
Epoch 38 (Time: 8.45s)
Epoch 39 (Time: 8.38s)
Epoch 40 (Time: 8.62s)
Epoch 41 (Time: 8.58s)
Epoch 42 (Time: 8.54s)
Epoch 43 (Time: 8.79s)
Epoch 44 (Time: 8.55s)
Epoch 45 (Time: 8.46s)
Epoch 46 (Time: 8.56s)
Epoch 47 (Time: 8.51s)
Epoch 48 (Time: 8.63s)
Epoch 49 (Time: 8.63s)
Epoch 50 (Time: 8.63s)
Epoch 51 (Time: 8.64s)
Epoch 52 (Time: 8.48s)
Epoch 53 (Time: 8.59s)
Epoch 54 (Time: 8.46s)
Epoch 55 (Time: 8.47s)
Epoch 56 (Time: 8.79s)
Epoch 57 (Time: 8.48s)
Epoch 58 (Time: 8.45s)
Epoch 59 (Time: 8.68s)
Epoch 60 (Time: 8.53s)
Epoch 61 (Time: 8.59s)
Epoch 62 (Time: 8.51s)
Epoch 63 (Time: 8.52s)
Epoch 64 (Time: 8.49s)
Epoch 65 (Time: 8.50s)
Epoch 66 (Time: 8.72s)
Epoch 67 (Time: 8.60s)
Epoch 68 (Time: 8.59s)
Epoch 69 (Time: 8.98s)
Epoch 70 (Time: 8.66s)
Epoch 71 (Time: 8.57s)
Epoch 72 (Time: 8.46s)
Epoch 73 (Time: 8.55s)
Epoch 74 (Time: 8.61s)
Epoch 75 (Time: 8.50s)
Epoch 76 (Time: 8.33s)
```

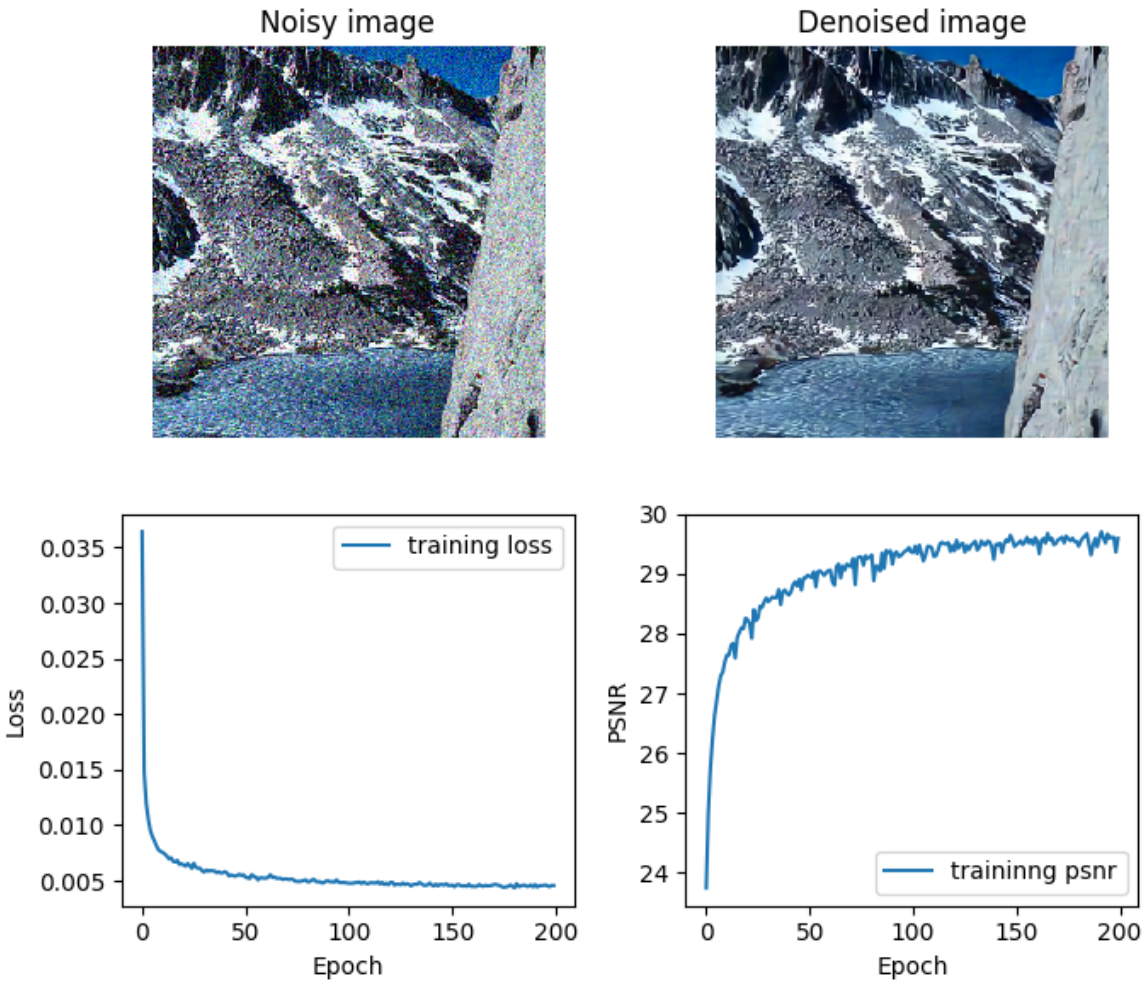
```
Epoch 77 (Time: 8.65s)
Epoch 78 (Time: 8.58s)
Epoch 79 (Time: 8.47s)
Epoch 80 (Time: 8.39s)
Epoch 81 (Time: 8.66s)
Epoch 82 (Time: 8.66s)
Epoch 83 (Time: 8.52s)
Epoch 84 (Time: 8.59s)
Epoch 85 (Time: 8.59s)
Epoch 86 (Time: 8.61s)
Epoch 87 (Time: 8.63s)
Epoch 88 (Time: 8.57s)
Epoch 89 (Time: 8.38s)
Epoch 90 (Time: 8.42s)
Epoch 91 (Time: 8.52s)
Epoch 92 (Time: 8.54s)
Epoch 93 (Time: 8.34s)
Epoch 94 (Time: 8.46s)
Epoch 95 (Time: 8.39s)
Epoch 96 (Time: 8.36s)
Epoch 97 (Time: 8.47s)
Epoch 98 (Time: 8.44s)
Epoch 99 (Time: 8.23s)
Epoch 100 (Time: 8.30s)
Epoch 101 (Time: 8.41s)
Epoch 102 (Time: 8.33s)
Epoch 103 (Time: 8.33s)
Epoch 104 (Time: 8.42s)
Epoch 105 (Time: 8.38s)
Epoch 106 (Time: 8.36s)
Epoch 107 (Time: 8.26s)
Epoch 108 (Time: 8.55s)
Epoch 109 (Time: 8.45s)
Epoch 110 (Time: 8.55s)
Epoch 111 (Time: 9.90s)
Epoch 112 (Time: 8.53s)
Epoch 113 (Time: 8.53s)
Epoch 114 (Time: 8.29s)
Epoch 115 (Time: 8.52s)
Epoch 116 (Time: 8.32s)
Epoch 117 (Time: 8.39s)
Epoch 118 (Time: 8.32s)
Epoch 119 (Time: 8.44s)
Epoch 120 (Time: 8.33s)
Epoch 121 (Time: 8.25s)
Epoch 122 (Time: 8.28s)
Epoch 123 (Time: 8.33s)
Epoch 124 (Time: 8.43s)
Epoch 125 (Time: 8.34s)
Epoch 126 (Time: 8.41s)
Epoch 127 (Time: 8.36s)
Epoch 128 (Time: 8.23s)
Epoch 129 (Time: 8.44s)
```

```
Epoch 130 (Time: 8.42s)
Epoch 131 (Time: 8.25s)
Epoch 132 (Time: 8.19s)
Epoch 133 (Time: 8.12s)
Epoch 134 (Time: 8.29s)
Epoch 135 (Time: 8.22s)
Epoch 136 (Time: 8.15s)
Epoch 137 (Time: 8.49s)
Epoch 138 (Time: 8.50s)
Epoch 139 (Time: 8.50s)
Epoch 140 (Time: 8.40s)
Epoch 141 (Time: 8.38s)
Epoch 142 (Time: 8.24s)
Epoch 143 (Time: 8.59s)
Epoch 144 (Time: 8.48s)
```

In [11]:

```
lr = 1e-3
net = DUDnCNN(6)
net = net.to(device)
adam = torch.optim.Adam(net.parameters(), lr=lr)
stats_manager = DenoisingStatsManager()
exp3 = nt.Experiment(net, train_set, test_set, adam, stats_manager,
               output_dir="denoising3", batch_size=4, perform_valid
ation_during_training=True)

def myimshow1(image, ax=plt):
    if image.requires_grad == True:
        image = image.to('cpu').detach().numpy()
    else:
        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

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()
    myimshow1(noisy, ax=axes[0][0])
    axes[0][0].set_title('Noisy image')
    myimshow1(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.e
poch)],
                    label="training loss")
    axes[1][1].plot([exp.history[k][1]['PSNR'] for k in range(exp.e
poch)],
                    label="traininng psnr")
    axes[1][0].legend()
    axes[1][0].set_ylabel('Loss')
    axes[1][0].set_xlabel('Epoch')

    axes[1][1].legend()
    axes[1][1].set_ylabel('PSNR')
    axes[1][1].set_xlabel('Epoch')
    plt.tight_layout()
    fig.canvas.draw()


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



```
Start/Continue training from epoch 174
Epoch 175 (Time: 40.82s)
Epoch 176 (Time: 42.46s)
Epoch 177 (Time: 42.95s)
Epoch 178 (Time: 42.63s)
Epoch 179 (Time: 42.85s)
Epoch 180 (Time: 42.59s)
Epoch 181 (Time: 42.93s)
Epoch 182 (Time: 42.83s)
Epoch 183 (Time: 42.99s)
Epoch 184 (Time: 43.01s)
Epoch 185 (Time: 43.18s)
Epoch 186 (Time: 42.93s)
Epoch 187 (Time: 42.97s)
Epoch 188 (Time: 42.91s)
Epoch 189 (Time: 43.04s)
Epoch 190 (Time: 42.88s)
Epoch 191 (Time: 42.89s)
Epoch 192 (Time: 42.95s)
Epoch 193 (Time: 43.13s)
Epoch 194 (Time: 42.85s)
Epoch 195 (Time: 43.05s)
Epoch 196 (Time: 42.91s)
Epoch 197 (Time: 42.98s)
Epoch 198 (Time: 42.77s)
Epoch 199 (Time: 43.14s)
Epoch 200 (Time: 42.88s)
Finish training for 200 epochs
```

In [19]:

```
## 22

lr = 1e-3
net = DUDnCNN(6)
net = net.to(device)
adam = torch.optim.Adam(net.parameters(), lr=lr)
stats_manager = DenoisingStatsManager()
exp3 = nt.Experiment(net, train_set, test_set, adam, stats_manager,
               output_dir="denoising3", batch_size=4, perform_valid
ation_during_training=True)


def myimshow1(image, ax=plt):
    if image.requires_grad == True:
        image = image.to('cpu').detach().numpy()
    else:
        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
```

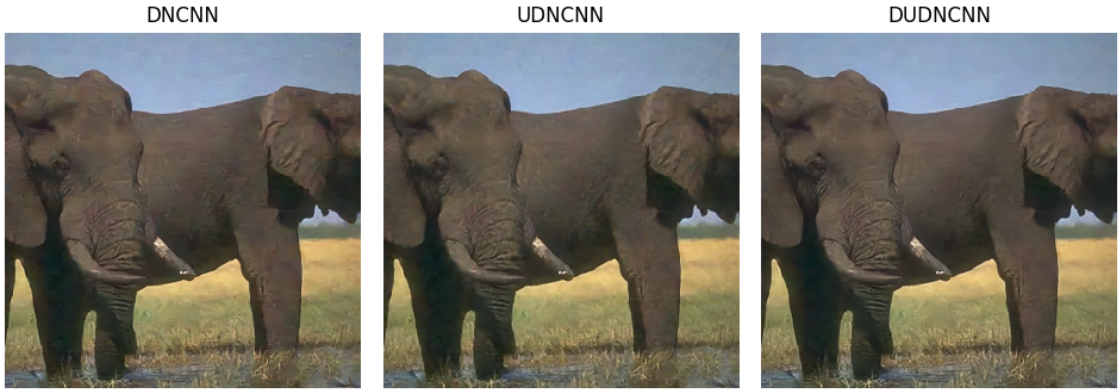
```
def plot_test(exp1, exp2, exp3, noisy):
    fig, axes = plt.subplots(ncols=3, nrows=1, figsize=(10,9), shar
ex = 'all', sharey = 'all')
    with torch.no_grad():
            denoised1 = exp1.net(noisy[np.newaxis].to(exp1.net.devi
ce))[0]
            denoised2 = exp2.net(noisy[np.newaxis].to(exp2.net.devi
ce))[0]
            denoised3 = exp3.net(noisy[np.newaxis].to(exp3.net.devi
ce))[0]
    axes[0].clear()
    axes[1].clear()
    axes[2].clear()
    myimshow1(denoised1, ax=axes[0])
    axes[0].set_title('DNCNN')
    myimshow1(denoised2, ax=axes[1])
    axes[1].set_title('UDNCNN')
    myimshow1(denoised3, ax=axes[2])
    axes[2].set_title('DUDNCNN')
    plt.tight_layout()
    fig.canvas.draw()


pickind = np.random.permutation(100)[:5]
# fig, axes = plt.subplots(ncols=2, nrows=1, figsize=(10,9), sharex
= 'all', sharey = 'all')
for i in range(len(pickind)):
    noisy = test_set[pickind[i]][0]
    plot_test(exp1, exp2, exp3, noisy)
# noisy=test_set[73][0]
```

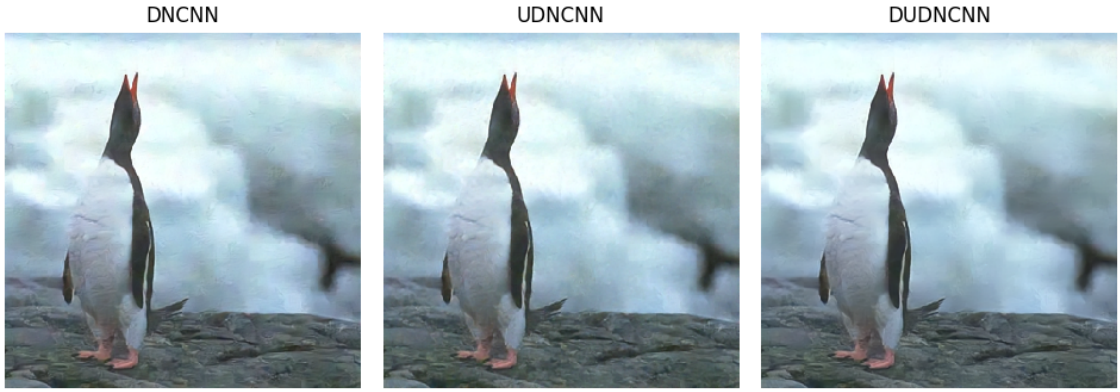
DNCNN

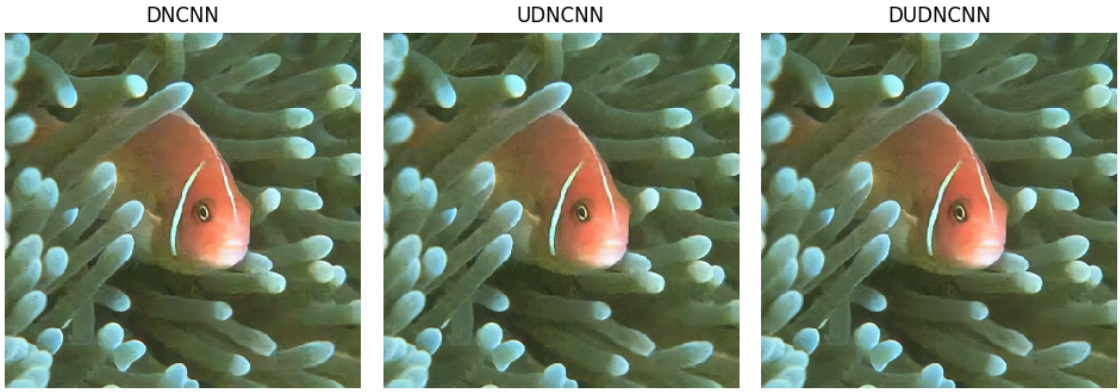
UDNCNN

DUDNCNN

DNCNN
UDNCNN
DUDNCNN


DNCNN
UDNCNN
DUDNCNN


DNCNN
UDNCNN
DUDNCNN



In [ ]:
```
## 23.
```

The number of trainable parameters of the DUDnCNN(D) is also $23649 + 67 + D(64649 + 64 + 2\ 64) = 37056D + 3523$

23.

receptive field DUDnCNN(D).

$$r_1 = 1 + 2 + \left[2 + 2^2 + 2^3 + \cdots + 2^{\left(\frac{D-2}{2}\right)}\right] + 2 \times 2^{\frac{D-2}{2}+1}$$

input conv[0] contracting path core conv

$$r_2 = r_1 + \left[2^{\left(\frac{D-2}{2}\right)} + \cdots + 2^3 + 2^2 + 2\right] + 2$$

expansive path conv[-1]

$$r = r_1 + r_2 = 1 + 4 + 2^{\frac{D+2}{2}} + \frac{4\left(1 - 2^{\frac{D-2}{2}}\right)}{1-2} = 1 + 2^{\frac{D+4}{2}}$$

The receptive field of DUDnCNN(D) is 2^((D+4)/2) + 1. Same with the UDnCNN(D) network.

Conclusion: All these three network have the same number of trainable parameters but with different receptive fields or level of information loss.

1. The UDnCNN and DUDnCNN has the same size of receptive field, however, DUDnCNN can achieve a better performance since it won't loss any spatial information with dilation convolution.
2. The DnCNN has smaller receptive field and its denoising performance is between UDnCNN and DUDnCNN.
3. The training time: UDnCNN < DnCNN < DUDnCNN