2. Image denoising

Read paper Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising (https://arxiv.org/abs/1608.03981). From the class drive download archive DnCNN.zip which contains the dataset, data preparation, training and testing scripts. You are provided with DnCNN-s model pretrained to denoise images with σ=25.

main references:

https://github.com/mozhuqing/fcn/blob/f6213010926c561b3e4f35eb5ee59b7ccc6390ec/data_augmenta.ipynb

https://github.com/jedichien/ssd_keras/blob/b3008773d833501da1b7a21e8c37e6640e229d5f/T3_Assioxes.ipynb
https://github.com/zplab-dev/Nicolette/blob/398be5b8a306972dfc4d530687b1a8ae290bcae5/image_analysis/analyze_images.py

a) Briefly explain how the authors have decided on the particular network depth. Which tasks is the DnCNN-3 designed to solve?

Answer:

i) The author sets the convolution kernel size of DnCNN to 3 * 3 and removes all pooling layers. For a single-layer convolution network, the size of the receptive field corresponding to each feature point on the feature map is equal to the size of the convolution layer filter; for the multi-layer convolution network, it can be fed back layer by layer, and the size of the receptive field in the original input image can be obtained through repeated iterations, that is, the size of the receptive field in the deep convolution layer behind is related to the filter size and step size of all previous network layers.

For DnCNN network, when the number of network layers is d, the receptive field of the network is (2d + 1) * (2d + 1). The receptive field of dncnn is related to the network depth D, while the receptive field of convolutional neural network can be compared with the effective patch size of traditional denoising algorithm. Therefore, the author refers to the most mainstream denoising algorithms, and according to 2d + 1 = effective patch size, reversely deduces a suitable network depth of DnCNN.

Finally,In order to capture enough spatial information, the author chooses 36 * 36 of EPLL as the reference standard with noise level of 25, because the effective patch size of EPLL is the smallest. The depth of dncnn for Gaussian denoising is 17, and the depth of DnCNN for general denoising task is 20.

ii) DnCNN-3 is used for image denoising. Batch normalization and residual learning are integrated to speed up the training process and improve denoising performance. This DnCNN-3 model has the ability to handle blind Gaussian denoising with unknown noise levels.

b) Use main_test.py to test the pretrained model on denoising the whole Set68 with σ=25 and report the obtained average PSNR and SSIM metrics. Download YourEmail_sigma.csv that contains specific value of σ. Test the same model on the same set corrupted by this particular σ and report the results.

In [ ]:

```
# %load main_test_b25.py

# ============================================================================
#  @article{zhang2017beyond,
#    title={Beyond a {Gaussian} denoiser: Residual learning of deep {CNN} for image denoising},
#    author={Zhang, Kai and Zuo, Wangmeng and Chen, Yunjin and Meng, Deyu and Zhang, Lei},
#    journal={IEEE Transactions on Image Processing},
#    year={2017},
#    volume={26},
```

```
#    number={7},
#    pages={3142-3155},
#  }
# modified version of the code from https://github.com/cszn/DnCNN
# ================================================================================

# run this to test the model

import argparse
import os, time, datetime
import numpy as np
import torch.nn as nn
import torch.nn.init as init
import torch
from skimage.measure import compare_psnr, compare_ssim
from skimage.io import imread, imsave


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--set_dir', default='data/Test', type=str, help='directory of test dataset
')
    parser.add_argument('--set_names', default=['Set68'], help='directory of test dataset')
    parser.add_argument('--sigma', default=25, type=int, help='noise level')
    parser.add_argument('--model_path', default='models/model_000.pth', type=str, help='the model n
ame')
    parser.add_argument('--result_dir', default='results', type=str, help='directory of test datase
t')
    parser.add_argument('--save_result', action='store_true', help='save the denoised image')
    return parser.parse_args()


def log(*args, **kwargs):
     print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:"), *args, **kwargs)


def save_result(result, path):
    path = path if path.find('.') != -1 else path+'.png'
    ext = os.path.splitext(path)[-1]
    if ext in ('.txt', '.dlm'):
        np.savetxt(path, result, fmt='%2.4f')
    else:
        imsave(path, np.uint8(np.clip(result*255.0, 0, 255)))


def show(x, title=None, cbar=False, figsize=None):
    import matplotlib.pyplot as plt
    plt.figure(figsize=figsize)
    plt.imshow(x, interpolation='nearest', cmap='gray')
    if title:
        plt.title(title)
    if cbar:
        plt.colorbar()
    plt.show()


class DnCNN(nn.Module):

    def __init__(self, depth=17, n_channels=64, image_channels=1, use_bnorm=True, kernel_size=3):
        super(DnCNN, self).__init__()
        kernel_size = 3
        padding = 1
        layers = []
        layers.append(nn.Conv2d(in_channels=image_channels, out_channels=n_channels, kernel_size=ke
rnel_size, padding=padding, bias=True))
        layers.append(nn.ReLU(inplace=True))
        for _ in range(depth-2):
            layers.append(nn.Conv2d(in_channels=n_channels, out_channels=n_channels, kernel_size=ke
rnel_size, padding=padding, bias=False))
            layers.append(nn.BatchNorm2d(n_channels))
            layers.append(nn.ReLU(inplace=True))
        layers.append(nn.Conv2d(in_channels=n_channels, out_channels=image_channels, kernel_size=ke
rnel_size, padding=padding, bias=False))
        self.dncnn = nn.Sequential(*layers)
        self._initialize_weights()

    def forward(self, x):
```

```
        y = x
        out = self.dncnn(x)
        return y-out

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                init.orthogonal_(m.weight)
                print('init weight')
                if m.bias is not None:
                    init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                init.constant_(m.weight, 1)
                init.constant_(m.bias, 0)


if __name__ == '__main__':

    args = parse_args()

    model = torch.load(args.model_path)
    #model = torch.load with map_location=torch.device('cpu')
    log('load trained model')

    model.eval()  # evaluation mode

    if torch.cuda.is_available():
        model = model.cuda()

    if not os.path.exists(args.result_dir):
        os.mkdir(args.result_dir)

    for set_cur in args.set_names:

        if not os.path.exists(os.path.join(args.result_dir, set_cur)):
            os.mkdir(os.path.join(args.result_dir, set_cur))
        psnrs = []
        ssims = []

        for im in os.listdir(os.path.join(args.set_dir, set_cur)):
            if im.endswith(".jpg") or im.endswith(".bmp") or im.endswith(".png"):

                x = np.array(imread(os.path.join(args.set_dir, set_cur, im)),
dtype=np.float32)/255.0
                np.random.seed(seed=0)  # for reproducibility
                y = x + np.random.normal(0, args.sigma/255.0, x.shape)  # Add Gaussian noise withou
clipping
                y = y.astype(np.float32)
                y_ = torch.from_numpy(y).view(1, -1, y.shape[0], y.shape[1])

                torch.cuda.synchronize()
                start_time = time.time()
                y_ = y_.cuda()
                x_ = model(y_)  # inference
                x_ = x_.view(y.shape[0], y.shape[1])
                x_ = x_.cpu()
                x_ = x_.detach().numpy().astype(np.float32)
                torch.cuda.synchronize()
                elapsed_time = time.time() - start_time
                print('%10s : %10s : %2.4f second' % (set_cur, im, elapsed_time))

                psnr_x_ = compare_psnr(x, x_)
                ssim_x_ = compare_ssim(x, x_)
                if args.save_result:
                    name, ext = os.path.splitext(im)
                    #show(np.hstack((y, x_)))  # show the image
                    save_result(x_, path=os.path.join(args.result_dir, set_cur, name+'_dncnn'+ext))
# save the denoised image
                psnrs.append(psnr_x_)
                ssims.append(ssim_x_)
        psnr_avg = np.mean(psnrs)
        ssim_avg = np.mean(ssims)
        psnrs.append(psnr_avg)
        ssims.append(ssim_avg)
        if args.save_result:
            save_result(np.hstack((psnrs, ssims)), path=os.path.join(args.result_dir, set_cur, 'res
ults.txt'))
```

```
        log('Datset: {0:10s} \n  PSNR = {1:2.2f}dB, SSIM = {2:1.4f}'.format(set_cur, psnr_avg, ssim
_avg))
```

In [2]:

```
%run main_test_b25.py
```

```
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:512: SourceChangeWarning: source cod
e of class '__main__.DnCNN' has changed. you can retrieve the original source code by accessing th
e object's source attribute or set `torch.nn.Module.dump_patches = True` and use the patch tool to
revert the changes.
  warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:512: SourceChangeWarning: source cod
e of class 'torch.nn.modules.conv.Conv2d' has changed. you can retrieve the original source code b
y accessing the object's source attribute or set `torch.nn.Module.dump_patches = True` and use the
patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:512: SourceChangeWarning: source cod
e of class 'torch.nn.modules.activation.ReLU' has changed. you can retrieve the original source co
de by accessing the object's source attribute or set `torch.nn.Module.dump_patches = True` and use
the patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:512: SourceChangeWarning: source cod
e of class 'torch.nn.modules.batchnorm.BatchNorm2d' has changed. you can retrieve the original sou
rce code by accessing the object's source attribute or set `torch.nn.Module.dump_patches = True` a
nd use the patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
/notebooks/main_test.py:140: UserWarning: DEPRECATED: skimage.measure.compare_psnr has been moved
to skimage.metrics.peak_signal_noise_ratio. It will be removed from skimage.measure in version
0.18.
  psnr_x_ = compare_psnr(x, x_)
/notebooks/main_test.py:141: UserWarning: DEPRECATED: skimage.measure.compare_ssim has been moved
to skimage.metrics.structural_similarity. It will be removed from skimage.measure in version 0.18.
  ssim_x_ = compare_ssim(x, x_)
```

```
2019-11-18 15:30:59: load trained model
     Set68 : test028.png : 0.0814 second
     Set68 : test006.png : 0.0789 second
     Set68 : test029.png : 0.0794 second
     Set68 : test017.png : 0.0789 second
     Set68 : test067.png : 0.0796 second
     Set68 : test057.png : 0.0789 second
     Set68 : test065.png : 0.0790 second
     Set68 : test051.png : 0.0790 second
     Set68 : test001.png : 0.0797 second
     Set68 : test013.png : 0.0790 second
     Set68 : test032.png : 0.0790 second
     Set68 : test058.png : 0.0790 second
     Set68 : test041.png : 0.0789 second
     Set68 : test060.png : 0.0790 second
     Set68 : test063.png : 0.0791 second
     Set68 : test054.png : 0.0796 second
     Set68 : test024.png : 0.0791 second
     Set68 : test004.png : 0.0791 second
     Set68 : test042.png : 0.0789 second
     Set68 : test066.png : 0.0791 second
     Set68 : test009.png : 0.0790 second
     Set68 : test059.png : 0.0792 second
     Set68 : test049.png : 0.0789 second
     Set68 : test012.png : 0.0791 second
     Set68 : test044.png : 0.0791 second
     Set68 : test031.png : 0.0795 second
     Set68 : test045.png : 0.0794 second
     Set68 : test047.png : 0.0791 second
     Set68 : test048.png : 0.0790 second
     Set68 : test019.png : 0.0789 second
     Set68 : test056.png : 0.0791 second
     Set68 : test046.png : 0.0791 second
     Set68 : test021.png : 0.0797 second
     Set68 : test022.png : 0.0793 second
     Set68 : test025.png : 0.0791 second
     Set68 : test050.png : 0.0792 second
     Set68 : test062.png : 0.0795 second
     Set68 : test002.png : 0.0797 second
     Set68 : test020.png : 0.0792 second
```

```
      Set68 : test014.png : 0.0791 second
      Set68 : test052.png : 0.0791 second
      Set68 : test040.png : 0.0791 second
      Set68 : test026.png : 0.0792 second
      Set68 : test036.png : 0.0792 second
      Set68 : test023.png : 0.0791 second
      Set68 : test016.png : 0.0793 second
      Set68 : test027.png : 0.0791 second
      Set68 : test034.png : 0.0796 second
      Set68 : test061.png : 0.0792 second
      Set68 : test010.png : 0.0792 second
      Set68 : test015.png : 0.0791 second
      Set68 : test035.png : 0.0792 second
      Set68 : test011.png : 0.0794 second
      Set68 : test003.png : 0.0796 second
      Set68 : test064.png : 0.0796 second
      Set68 : test030.png : 0.0791 second
      Set68 : test007.png : 0.0791 second
      Set68 : test053.png : 0.0792 second
      Set68 : test008.png : 0.0793 second
      Set68 : test039.png : 0.0797 second
      Set68 : test055.png : 0.0796 second
      Set68 : test005.png : 0.0791 second
      Set68 : test018.png : 0.0792 second
      Set68 : test033.png : 0.0792 second
      Set68 : test043.png : 0.0791 second
      Set68 : test038.png : 0.0796 second
      Set68 : test068.png : 0.0792 second
      Set68 : test037.png : 0.0791 second
2019-11-18 15:31:06: Datset: Set68
  PSNR = 29.26dB, SSIM = 0.9022
```

In [5]:

```
import pandas as pd
csv2=pd.read_csv('XZHANG6@TCD.IE_sigma.csv')
csv2.head(3)
```

Out[5]:

	37

In [ ]:

```
# %load main_test_b37.py

# ============================================================================
#  @article{zhang2017beyond,
#    title={Beyond a {Gaussian} denoiser: Residual learning of deep {CNN} for image denoising},
#    author={Zhang, Kai and Zuo, Wangmeng and Chen, Yunjin and Meng, Deyu and Zhang, Lei},
#    journal={IEEE Transactions on Image Processing},
#    year={2017},
#    volume={26},
#    number={7},
#    pages={3142-3155},
#  }
# modified version of the code from https://github.com/cszn/DnCNN
# ============================================================================

# run this to test the model

import argparse
import os, time, datetime
import numpy as np
import torch.nn as nn
import torch.nn.init as init
import torch
from skimage.measure import compare_psnr, compare_ssim
from skimage.io import imread, imsave


def parse_args():
    parser = argparse.ArgumentParser()
```

```
    parser.add_argument('--set_dir', default='data/Test', type=str, help='directory of test dataset
')
    parser.add_argument('--set_names', default=['Set68'], help='directory of test dataset')
    parser.add_argument('--sigma', default=37, type=int, help='noise level')
    parser.add_argument('--model_path', default='models/model_000.pth', type=str, help='the model n
ame')
    parser.add_argument('--result_dir', default='results', type=str, help='directory of test datase
t')
    parser.add_argument('--save_result', action='store_true', help='save the denoised image')
    return parser.parse_args()


def log(*args, **kwargs):
     print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:"), *args, **kwargs)


def save_result(result, path):
    path = path if path.find('.') != -1 else path+'.png'
    ext = os.path.splitext(path)[-1]
    if ext in ('.txt', '.dlm'):
        np.savetxt(path, result, fmt='%2.4f')
    else:
        imsave(path, np.uint8(np.clip(result*255.0, 0, 255)))


def show(x, title=None, cbar=False, figsize=None):
    import matplotlib.pyplot as plt
    plt.figure(figsize=figsize)
    plt.imshow(x, interpolation='nearest', cmap='gray')
    if title:
        plt.title(title)
    if cbar:
        plt.colorbar()
    plt.show()


class DnCNN(nn.Module):

    def __init__(self, depth=17, n_channels=64, image_channels=1, use_bnorm=True, kernel_size=3):
        super(DnCNN, self).__init__()
        kernel_size = 3
        padding = 1
        layers = []
        layers.append(nn.Conv2d(in_channels=image_channels, out_channels=n_channels, kernel_size=ke
rnel_size, padding=padding, bias=True))
        layers.append(nn.ReLU(inplace=True))
        for _ in range(depth-2):
            layers.append(nn.Conv2d(in_channels=n_channels, out_channels=n_channels, kernel_size=ke
rnel_size, padding=padding, bias=False))
            layers.append(nn.BatchNorm2d(n_channels))
            layers.append(nn.ReLU(inplace=True))
        layers.append(nn.Conv2d(in_channels=n_channels, out_channels=image_channels, kernel_size=ke
rnel_size, padding=padding, bias=False))
        self.dncnn = nn.Sequential(*layers)
        self._initialize_weights()

    def forward(self, x):
        y = x
        out = self.dncnn(x)
        return y-out

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                init.orthogonal_(m.weight)
                print('init weight')
                if m.bias is not None:
                    init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                init.constant_(m.weight, 1)
                init.constant_(m.bias, 0)


if __name__ == '__main__':

    args = parse_args()
```

```
    model = torch.load(args.model_path)
    #model = torch.load with map_location=torch.device('cpu')
    log('load trained model')

    model.eval()  # evaluation mode

    if torch.cuda.is_available():
        model = model.cuda()

    if not os.path.exists(args.result_dir):
        os.mkdir(args.result_dir)

    for set_cur in args.set_names:

        if not os.path.exists(os.path.join(args.result_dir, set_cur)):
            os.mkdir(os.path.join(args.result_dir, set_cur))
        psnrs = []
        ssims = []

        for im in os.listdir(os.path.join(args.set_dir, set_cur)):
            if im.endswith(".jpg") or im.endswith(".bmp") or im.endswith(".png"):

                x = np.array(imread(os.path.join(args.set_dir, set_cur, im)),
dtype=np.float32)/255.0
                np.random.seed(seed=0)  # for reproducibility
                y = x + np.random.normal(0, args.sigma/255.0, x.shape)  # Add Gaussian noise withou
clipping
                y = y.astype(np.float32)
                y_ = torch.from_numpy(y).view(1, -1, y.shape[0], y.shape[1])

                torch.cuda.synchronize()
                start_time = time.time()
                y_ = y_.cuda()
                x_ = model(y_)  # inference
                x_ = x_.view(y.shape[0], y.shape[1])
                x_ = x_.cpu()
                x_ = x_.detach().numpy().astype(np.float32)
                torch.cuda.synchronize()
                elapsed_time = time.time() - start_time
                print('%10s : %10s : %2.4f second' % (set_cur, im, elapsed_time))

                psnr_x_ = compare_psnr(x, x_)
                ssim_x_ = compare_ssim(x, x_)
                if args.save_result:
                    name, ext = os.path.splitext(im)
                    #show(np.hstack((y, x_)))  # show the image
                    save_result(x_, path=os.path.join(args.result_dir, set_cur, name+'_dncnn'+ext))
# save the denoised image
                psnrs.append(psnr_x_)
                ssims.append(ssim_x_)
        psnr_avg = np.mean(psnrs)
        ssim_avg = np.mean(ssims)
        psnrs.append(psnr_avg)
        ssims.append(ssim_avg)
        if args.save_result:
            save_result(np.hstack((psnrs, ssims)), path=os.path.join(args.result_dir, set_cur, 'res
ults.txt'))
        log('Datset: {0:10s} \n  PSNR = {1:2.2f}dB, SSIM = {2:1.4f}'.format(set_cur, psnr_avg, ssim
_avg))
```

In [10]:

```
%run main_test_b37.py
```

```
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:512: SourceChangeWarning: source cod
e of class '__main__.DnCNN' has changed. you can retrieve the original source code by accessing th
e object's source attribute or set `torch.nn.Module.dump_patches = True` and use the patch tool to
revert the changes.
  warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:512: SourceChangeWarning: source cod
e of class 'torch.nn.modules.conv.Conv2d' has changed. you can retrieve the original source code b
y accessing the object's source attribute or set `torch.nn.Module.dump_patches = True` and use the
patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:512: SourceChangeWarning: source cod
```

```
e of class `torch.nn.modules.activation.ReLU` has changed. you can retrieve the original source co
de by accessing the object's source attribute or set `torch.nn.Module.dump_patches = True` and use
the patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
/usr/local/lib/python3.6/dist-packages/torch/serialization.py:512: SourceChangeWarning: source cod
e of class 'torch.nn.modules.batchnorm.BatchNorm2d' has changed. you can retrieve the original sou
rce code by accessing the object's source attribute or set `torch.nn.Module.dump_patches = True` a
nd use the patch tool to revert the changes.
  warnings.warn(msg, SourceChangeWarning)
/notebooks/main_test_b37.py:140: UserWarning: DEPRECATED: skimage.measure.compare_psnr has been mo
ved to skimage.metrics.peak_signal_noise_ratio. It will be removed from skimage.measure in version
0.18.
  psnr_x_ = compare_psnr(x, x_)
/notebooks/main_test_b37.py:141: UserWarning: DEPRECATED: skimage.measure.compare_ssim has been mo
ved to skimage.metrics.structural_similarity. It will be removed from skimage.measure in version
0.18.
  ssim_x_ = compare_ssim(x, x_)


2019-11-18 15:54:46: load trained model
     Set68 : test028.png : 0.0793 second
     Set68 : test006.png : 0.0790 second
     Set68 : test029.png : 0.0794 second
     Set68 : test017.png : 0.0790 second
     Set68 : test067.png : 0.0795 second
     Set68 : test057.png : 0.0790 second
     Set68 : test065.png : 0.0791 second
     Set68 : test051.png : 0.0789 second
     Set68 : test001.png : 0.0796 second
     Set68 : test013.png : 0.0791 second
     Set68 : test032.png : 0.0791 second
     Set68 : test058.png : 0.0789 second
     Set68 : test041.png : 0.0790 second
     Set68 : test060.png : 0.0792 second
     Set68 : test063.png : 0.0791 second
     Set68 : test054.png : 0.0795 second
     Set68 : test024.png : 0.0793 second
     Set68 : test004.png : 0.0791 second
     Set68 : test042.png : 0.0790 second
     Set68 : test066.png : 0.0789 second
     Set68 : test009.png : 0.0791 second
     Set68 : test059.png : 0.0793 second
     Set68 : test049.png : 0.0791 second
     Set68 : test012.png : 0.0791 second
     Set68 : test044.png : 0.0790 second
     Set68 : test031.png : 0.0794 second
     Set68 : test045.png : 0.0797 second
     Set68 : test047.png : 0.0791 second
     Set68 : test048.png : 0.0791 second
     Set68 : test019.png : 0.0790 second
     Set68 : test056.png : 0.0790 second
     Set68 : test046.png : 0.0790 second
     Set68 : test021.png : 0.0796 second
     Set68 : test022.png : 0.0789 second
     Set68 : test025.png : 0.0791 second
     Set68 : test050.png : 0.0790 second
     Set68 : test062.png : 0.0797 second
     Set68 : test002.png : 0.0796 second
     Set68 : test020.png : 0.0791 second
     Set68 : test014.png : 0.0791 second
     Set68 : test052.png : 0.0789 second
     Set68 : test040.png : 0.0790 second
     Set68 : test026.png : 0.0791 second
     Set68 : test036.png : 0.0792 second
     Set68 : test023.png : 0.0792 second
     Set68 : test016.png : 0.0791 second
     Set68 : test027.png : 0.0790 second
     Set68 : test034.png : 0.0795 second
     Set68 : test061.png : 0.0791 second
     Set68 : test010.png : 0.0790 second
     Set68 : test015.png : 0.0790 second
     Set68 : test035.png : 0.0790 second
     Set68 : test011.png : 0.0790 second
     Set68 : test003.png : 0.0796 second
     Set68 : test064.png : 0.0797 second
     Set68 : test030.png : 0.0792 second
     Set68 : test007.png : 0.0789 second
     Set68 : test053.png : 0.0790 second
```

```
      Set68 : test008.png : 0.0790 second
      Set68 : test039.png : 0.0794 second
      Set68 : test055.png : 0.0795 second
      Set68 : test005.png : 0.0790 second
      Set68 : test018.png : 0.0791 second
      Set68 : test033.png : 0.0791 second
      Set68 : test043.png : 0.0791 second
      Set68 : test038.png : 0.0797 second
      Set68 : test068.png : 0.0791 second
      Set68 : test037.png : 0.0790 second
2019-11-18 15:54:53: Datset: Set68
  PSNR = 21.54dB, SSIM = 0.5650
```

c) Finetune the pre-trained model using script main_train.py for 1 epoch with learning rate 0.0001 with σ level given from YourEmail_sigma.csv and test its performance (PSRN/SSIM) on Set68 using this corruption level.

In [ ]:

```
# %load main_train_fine.py

# ============================================================================
#  @article{zhang2017beyond,
#    title={Beyond a {Gaussian} denoiser: Residual learning of deep {CNN} for image denoising},
#    author={Zhang, Kai and Zuo, Wangmeng and Chen, Yunjin and Meng, Deyu and Zhang, Lei},
#    journal={IEEE Transactions on Image Processing},
#    year={2017},
#    volume={26},
#    number={7},
#    pages={3142-3155},
#  }
# modified version of the code from https://github.com/cszn/DnCNN
# ============================================================================

# run this to train the model

import argparse
import re
import os, glob, datetime, time
import numpy as np
import torch
import torch.nn as nn
from torch.nn.modules.loss import _Loss
import torch.nn.init as init
from torch.utils.data import DataLoader
import torch.optim as optim
from torch.optim.lr_scheduler import MultiStepLR
import data_generator as dg
from data_generator import DenoisingDataset


# Params
parser = argparse.ArgumentParser(description='PyTorch DnCNN')
parser.add_argument('--batch_size', default=128, type=int, help='batch size')
parser.add_argument('--train_data', default='data/Train400', type=str, help='path of train data')
parser.add_argument('--sigma', default=37, type=int, help='noise level')
parser.add_argument('--epochs', default=1, type=int, help='number of train epochs')
parser.add_argument('--lr', default=1e-4, type=float, help='initial learning rate for Adam')
args = parser.parse_args()

batch_size = args.batch_size
cuda = torch.cuda.is_available()
n_epochs = args.epochs
sigma = args.sigma

save_dir = 'models'

if not os.path.exists(save_dir):
    os.mkdir(save_dir)


class DnCNN(nn.Module):
    def __init__(self, depth=17, n_channels=64, image_channels=1, use_bnorm=True, kernel_size=3):
        super(DnCNN, self).__init__()
        kernel_size = 3
        padding = 1
```

```
        layers = []

        layers.append(nn.Conv2d(in_channels=image_channels, out_channels=n_channels, kernel_size=ke
rnel_size, padding=padding, bias=True))
        layers.append(nn.ReLU(inplace=True))
        for _ in range(depth-2):
            layers.append(nn.Conv2d(in_channels=n_channels, out_channels=n_channels, kernel_size=ke
rnel_size, padding=padding, bias=False))
            layers.append(nn.BatchNorm2d(n_channels))
            layers.append(nn.ReLU(inplace=True))
        layers.append(nn.Conv2d(in_channels=n_channels, out_channels=image_channels, kernel_size=ke
rnel_size, padding=padding, bias=False))
        self.dncnn = nn.Sequential(*layers)
        self._initialize_weights()

    def forward(self, x):
        y = x
        out = self.dncnn(x)
        return y-out

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                init.orthogonal_(m.weight)
                if m.bias is not None:
                    init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                init.constant_(m.weight, 1)
                init.constant_(m.bias, 0)


class sum_squared_error(_Loss):
    """
    Definition: sum_squared_error = 1/2 * nn.MSELoss(reduction = 'sum')
    The backward is defined as: input-target
    """
    def __init__(self, size_average=None, reduce=None, reduction='sum'):
        super(sum_squared_error, self).__init__(size_average, reduce, reduction)

    def forward(self, input, target):
        return torch.nn.functional.mse_loss(input, target, size_average=None, reduce=None, reductio
n='sum').div_(2)


def findLastCheckpoint(save_dir):
    file_list = glob.glob(os.path.join(save_dir, 'model_*.pth'))
    if file_list:
        epochs_exist = []
        for file_ in file_list:
            result = re.findall(".*model_(.*).pth.*", file_)
            epochs_exist.append(int(result[0]))
        initial_epoch = max(epochs_exist)
    else:
        initial_epoch = 0
    return initial_epoch


def log(*args, **kwargs):
     print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:"), *args, **kwargs)


if __name__ == '__main__':
    # model selection
    print('===> Building model')
    model = DnCNN()

    initial_epoch = findLastCheckpoint(save_dir=save_dir)  # load the last model in matconvnet styl
e
    if initial_epoch > 0 or os.path.isfile(os.path.join(save_dir, 'model_%03d.pth' % initial_epoch)
):
        print('resuming by loading epoch %03d' % initial_epoch)
        model = torch.load(os.path.join(save_dir, 'model_%03d.pth' % initial_epoch))
    model.train()
    criterion = sum_squared_error()
    if cuda:
        model = model.cuda()
    optimizer = optim.Adam(model.parameters(), lr=args.lr)
```

```
    scheduler = MultiStepLR(optimizer, milestones=[15], gamma=0.1)  # learning rates
    for epoch in range(initial_epoch, n_epochs):

        scheduler.step(epoch)  # step to the learning rate in this epcoh
        xs = dg.datagenerator(data_dir=args.train_data)
        xs = xs.astype('float32')/255.0
        xs = torch.from_numpy(xs.transpose((0, 3, 1, 2)))  # tensor of the clean patches, NXCXHXW
        DDataset = DenoisingDataset(xs, sigma)
        DLoader = DataLoader(dataset=DDataset, num_workers=0, drop_last=True, batch_size=batch_size
, shuffle=True)
        epoch_loss = 0
        start_time = time.time()

        for n_count, batch_yx in enumerate(DLoader):
            optimizer.zero_grad()
            if cuda:
                batch_x, batch_y = batch_yx[1].cuda(), batch_yx[0].cuda()
            else:
                batch_x, batch_y = batch_yx[1], batch_yx[0]
            loss = criterion(model(batch_y), batch_x)
            epoch_loss += loss.item()
            loss.backward()
            optimizer.step()
            if n_count % 100 == 0:
                print('%4d %4d / %4d loss = %2.4f' % (epoch+1, n_count, xs.size(0)//batch_size, los
.item()/batch_size))
        elapsed_time = time.time() - start_time

        log('epcoh = %4d , loss = %4.4f , time = %4.2f s' % (epoch+1, epoch_loss/n_count, elapsed_t
ime))
        np.savetxt('train_result.txt', np.hstack((epoch+1, epoch_loss/n_count, elapsed_time)), fmt=
'%2.4f')
        torch.save(model, os.path.join(save_dir, 'model_%03d.pth' % (epoch+1)))
```

In [11]:

```
%run main_train_fine.py
```

```
===> Building model
resuming by loading epoch 000
```

```
/usr/local/lib/python3.6/dist-packages/torch/optim/lr_scheduler.py:122: UserWarning: Detected call
of `lr_scheduler.step()` before `optimizer.step()`. In PyTorch 1.1.0 and later, you should call th
em in the opposite order: `optimizer.step()` before `lr_scheduler.step()`.  Failure to do this wil
l result in PyTorch skipping the first value of the learning rate schedule.See more details at htt
ps://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate
  "https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate", UserWarning)
```

```
training data finished
   1    0 / 11390 loss = 3.6133
   1  100 / 11390 loss = 2.9871
   1  200 / 11390 loss = 2.6942
   1  300 / 11390 loss = 2.3620
   1  400 / 11390 loss = 2.1930
   1  500 / 11390 loss = 2.0399
   1  600 / 11390 loss = 2.1270
   1  700 / 11390 loss = 1.7493
   1  800 / 11390 loss = 1.6886
   1  900 / 11390 loss = 1.6603
   1 1000 / 11390 loss = 1.8002
   1 1100 / 11390 loss = 1.9602
   1 1200 / 11390 loss = 1.6682
   1 1300 / 11390 loss = 1.7139
   1 1400 / 11390 loss = 1.7712
   1 1500 / 11390 loss = 1.8223
   1 1600 / 11390 loss = 2.0812
   1 1700 / 11390 loss = 1.7096
   1 1800 / 11390 loss = 1.9449
   1 1900 / 11390 loss = 1.8372
   1 2000 / 11390 loss = 1.9960
   1 2100 / 11390 loss = 2.0903
   1 2200 / 11390 loss = 1.9423
   1 2300 / 11390 loss = 1.6355
   1 2400 / 11390 loss = 1.8820
```

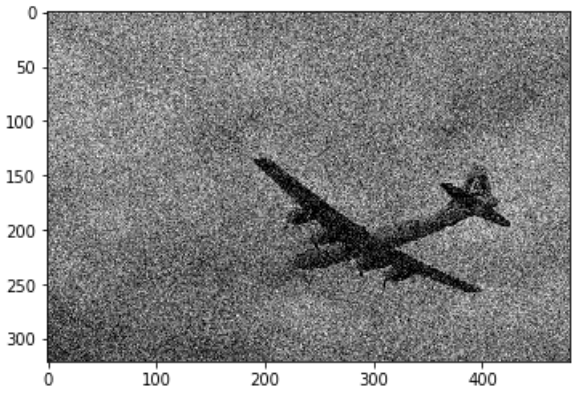
```
1 2500 / 11390 loss = 1.8893
1 2600 / 11390 loss = 1.5854
1 2700 / 11390 loss = 1.8562
1 2800 / 11390 loss = 2.0158
1 2900 / 11390 loss = 1.8101
1 3000 / 11390 loss = 1.8127
1 3100 / 11390 loss = 1.7136
1 3200 / 11390 loss = 1.6689
1 3300 / 11390 loss = 1.9814
1 3400 / 11390 loss = 1.7780
1 3500 / 11390 loss = 1.6775
1 3600 / 11390 loss = 1.5802
1 3700 / 11390 loss = 1.7287
1 3800 / 11390 loss = 1.8248
1 3900 / 11390 loss = 1.6918
1 4000 / 11390 loss = 1.8658
1 4100 / 11390 loss = 1.8874
1 4200 / 11390 loss = 1.7661
1 4300 / 11390 loss = 1.7228
1 4400 / 11390 loss = 1.6080
1 4500 / 11390 loss = 1.9002
1 4600 / 11390 loss = 1.8401
1 4700 / 11390 loss = 2.0212
1 4800 / 11390 loss = 1.6225
1 4900 / 11390 loss = 1.7425
1 5000 / 11390 loss = 1.6205
1 5100 / 11390 loss = 1.7129
1 5200 / 11390 loss = 1.6692
1 5300 / 11390 loss = 1.6559
1 5400 / 11390 loss = 1.9210
1 5500 / 11390 loss = 1.7337
1 5600 / 11390 loss = 1.6982
1 5700 / 11390 loss = 1.8424
1 5800 / 11390 loss = 1.6358
1 5900 / 11390 loss = 1.8081
1 6000 / 11390 loss = 1.6245
1 6100 / 11390 loss = 1.6830
1 6200 / 11390 loss = 1.6092
1 6300 / 11390 loss = 1.6706
1 6400 / 11390 loss = 1.8066
1 6500 / 11390 loss = 1.9122
1 6600 / 11390 loss = 1.7672
1 6700 / 11390 loss = 1.7631
1 6800 / 11390 loss = 1.7428
1 6900 / 11390 loss = 1.6622
1 7000 / 11390 loss = 1.6547
1 7100 / 11390 loss = 1.7500
1 7200 / 11390 loss = 1.8332
1 7300 / 11390 loss = 1.9402
1 7400 / 11390 loss = 1.6748
1 7500 / 11390 loss = 1.8857
1 7600 / 11390 loss = 1.7829
1 7700 / 11390 loss = 1.7797
1 7800 / 11390 loss = 1.7881
1 7900 / 11390 loss = 1.6660
1 8000 / 11390 loss = 1.9970
1 8100 / 11390 loss = 1.6728
1 8200 / 11390 loss = 1.8209
1 8300 / 11390 loss = 1.8218
1 8400 / 11390 loss = 1.8170
1 8500 / 11390 loss = 1.8652
1 8600 / 11390 loss = 1.6446
1 8700 / 11390 loss = 1.8744
1 8800 / 11390 loss = 1.8970
1 8900 / 11390 loss = 1.7119
1 9000 / 11390 loss = 1.6322
1 9100 / 11390 loss = 1.7965
1 9200 / 11390 loss = 1.6404
1 9300 / 11390 loss = 1.8917
1 9400 / 11390 loss = 1.9150
1 9500 / 11390 loss = 1.8927
1 9600 / 11390 loss = 1.6934
1 9700 / 11390 loss = 2.0384
1 9800 / 11390 loss = 1.6115
1 9900 / 11390 loss = 1.7821
1 10000 / 11390 loss = 1.7793
1 10100 / 11390 loss = 2.0286
```

```
   1 10200 / 11390 loss = 1.7847
   1 10300 / 11390 loss = 1.8575
   1 10400 / 11390 loss = 1.8661
   1 10500 / 11390 loss = 1.6661
   1 10600 / 11390 loss = 1.9295
   1 10700 / 11390 loss = 1.8009
   1 10800 / 11390 loss = 1.7453
   1 10900 / 11390 loss = 1.7981
   1 11000 / 11390 loss = 1.8992
   1 11100 / 11390 loss = 1.9330
   1 11200 / 11390 loss = 1.6897
   1 11300 / 11390 loss = 1.5796
2019-11-18 17:37:24: epcoh =    1 , loss = 234.9939 , time = 4524.41 s
```

d) Download YourEmail_XXX.png containing corrupted version of data/Test/Set68/testXXX.png. Denoise this image with the pre-trained model, submit it as YourEmail_XXX_pretrained.png and save both PSNR, SSIM values for this image in 2 columns preserving ordering, in a file named YourEmail_psnr.csv. Repeat the procedure with your finetuned model, save result to YourEmail_XXX_clean.png and report PSNR and SSIM of this image as a second row of YourEmail_psnr.csv.

In [15]:

```
import cv2
from matplotlib import pyplot as plt

img1 = cv2.imread ('./XZHANG6@TCD.IE_066.png', cv2.IMREAD_COLOR)
rgb_img1=img1[:,:,::-1]
plt.imshow(rgb_img1)
plt.show()
```



In [16]:

```
img2 = cv2.imread ('./test066.png', cv2.IMREAD_COLOR)
rgb_img2=img2[:,:,::-1]
plt.imshow(rgb_img2)
plt.show()
```

In [ ]:

```
# %load main_test_d_pre.py
```

```
# ============================================================================
#  @article{zhang2017beyond,
#    title={Beyond a {Gaussian} denoiser: Residual learning of deep {CNN} for image denoising},
#    author={Zhang, Kai and Zuo, Wangmeng and Chen, Yunjin and Meng, Deyu and Zhang, Lei},
#    journal={IEEE Transactions on Image Processing},
#    year={2017},
#    volume={26},
#    number={7},
#    pages={3142-3155},
#  }
# modified version of the code from https://github.com/cszn/DnCNN
# ============================================================================

# run this to test the model

import argparse
import os, time, datetime
import numpy as np
import torch.nn as nn
import torch.nn.init as init
import torch
from skimage.measure import compare_psnr, compare_ssim
from skimage.io import imread, imsave


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--set_dir', default='data', type=str, help='directory of test dataset')
    parser.add_argument('--set_names', default=['test066'], help='directory of test dataset')
    parser.add_argument('--sigma', default=25, type=int, help='noise level')
    parser.add_argument('--model_path', default='models/model_000.pth', type=str, help='the model n
ame')
    parser.add_argument('--result_dir', default='results', type=str, help='directory of test datase
t')
    parser.add_argument('--save_result', action='store_false', help='save the denoised image')
    return parser.parse_args()


def log(*args, **kwargs):
     print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:"), *args, **kwargs)


def save_result(result, path):
    path = path if path.find('.') != -1 else path+'.png'
    ext = os.path.splitext(path)[-1]
    if ext in ('.txt', '.dlm'):
        np.savetxt(path, result, fmt='%2.4f')
    else:
        imsave(path, np.uint8(np.clip(result*255.0, 0, 255)))


def show(x, title=None, cbar=False, figsize=None):
    import matplotlib.pyplot as plt
    plt.figure(figsize=figsize)
    plt.imshow(x, interpolation='nearest', cmap='gray')
    if title:
        plt.title(title)
    if cbar:
        plt.colorbar()
    plt.show()


class DnCNN(nn.Module):

    def __init__(self, depth=17, n_channels=64, image_channels=1, use_bnorm=True, kernel_size=3):
        super(DnCNN, self).__init__()
        kernel_size = 3
        padding = 1
        layers = []
        layers.append(nn.Conv2d(in_channels=image_channels, out_channels=n_channels, kernel_size=ke
rnel_size, padding=padding, bias=True))
        layers.append(nn.ReLU(inplace=True))
        for _ in range(depth-2):
            layers.append(nn.Conv2d(in_channels=n_channels, out_channels=n_channels, kernel_size=ke
rnel_size, padding=padding, bias=False))
            layers.append(nn.BatchNorm2d(n_channels))
            layers.append(nn.ReLU(inplace=True))
```

```
        layers.append(nn.Conv2d(in_channels=n_channels, out_channels=image_channels, kernel_size=ke
rnel_size, padding=padding, bias=False))
        self.dncnn = nn.Sequential(*layers)
        self._initialize_weights()

    def forward(self, x):
        y = x
        out = self.dncnn(x)
        return y-out

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                init.orthogonal_(m.weight)
                print('init weight')
                if m.bias is not None:
                    init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                init.constant_(m.weight, 1)
                init.constant_(m.bias, 0)


if __name__ == '__main__':

    args = parse_args()

    model = torch.load(args.model_path)
    #model = torch.load with map_location=torch.device('cpu')
    log('load trained model')

    model.eval()  # evaluation mode

    if torch.cuda.is_available():
        model = model.cuda()

    if not os.path.exists(args.result_dir):
        os.mkdir(args.result_dir)

    for set_cur in args.set_names:

        if not os.path.exists(os.path.join(args.result_dir, set_cur)):
            os.mkdir(os.path.join(args.result_dir, set_cur))
        psnrs = []
        ssims = []

        for im in os.listdir(os.path.join(args.set_dir, set_cur)):
            if im.endswith(".jpg") or im.endswith(".bmp") or im.endswith(".png"):

                x = np.array(imread(os.path.join(args.set_dir, set_cur, im)),
dtype=np.float32)/255.0
                np.random.seed(seed=0)  # for reproducibility
                #y = x + np.random.normal(0, args.sigma/255.0, x.shape)  # Add Gaussian noise witho
ut clipping
                y = x
                y = y.astype(np.float32)
                y_ = torch.from_numpy(y).view(1, -1, y.shape[0], y.shape[1])

                torch.cuda.synchronize()
                start_time = time.time()
                y_ = y_.cuda()
                x_ = model(y_)  # inference
                x_ = x_.view(y.shape[0], y.shape[1])
                x_ = x_.cpu()
                x_ = x_.detach().numpy().astype(np.float32)
                torch.cuda.synchronize()
                elapsed_time = time.time() - start_time
                print('%10s : %10s : %2.4f second' % (set_cur, im, elapsed_time))

                psnr_x_ = compare_psnr(x, x_)
                ssim_x_ = compare_ssim(x, x_)
                if args.save_result:
                    name, ext = os.path.splitext(im)
                    show(np.hstack((y, x_)))  # show the image
                    save_result(x_, path=os.path.join(args.result_dir, set_cur, name+'_pretrained'+
ext))  # save the denoised image
                psnrs.append(psnr_x_)
                ssims.append(ssim_x_)
```

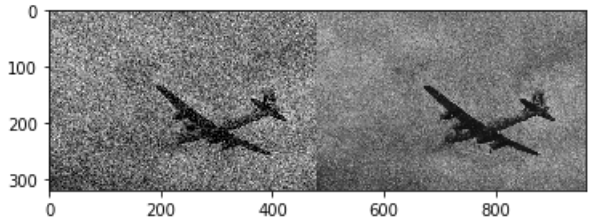
```
        psnr_avg = np.mean(psnrs)
        ssim_avg = np.mean(ssims)
        psnrs.append(psnr_avg)
        ssims.append(ssim_avg)
        if args.save_result:
            save_result(np.hstack((psnrs, ssims)), path=os.path.join(args.result_dir, set_cur, 'res
ults.txt'))
        log('Datset: {0:10s} \n  PSNR = {1:2.2f}dB, SSIM = {2:1.4f}'.format(set_cur, psnr_avg, ssim
_avg))
```

In [10]:

```
%run main_test_d_pre.py
```

```
2019-11-24 20:20:14: load trained model
   test066 : XZHANG6@TCD.IE_066.png : 0.0795 second
```

```
/notebooks/main_test_d_pre.py:141: UserWarning: DEPRECATED: skimage.measure.compare_psnr has been
moved to skimage.metrics.peak_signal_noise_ratio. It will be removed from skimage.measure in
version 0.18.
  psnr_x_ = compare_psnr(x, x_)
/notebooks/main_test_d_pre.py:142: UserWarning: DEPRECATED: skimage.measure.compare_ssim has been
moved to skimage.metrics.structural_similarity. It will be removed from skimage.measure in version
0.18.
  ssim_x_ = compare_ssim(x, x_)
```




```
2019-11-24 20:20:15: Datset: test066
  PSNR = 23.16dB, SSIM = 0.8391
```

In [ ]:

```
# %load main_test_d_fine.py

# ============================================================================
#  @article{zhang2017beyond,
#    title={Beyond a {Gaussian} denoiser: Residual learning of deep {CNN} for image denoising},
#    author={Zhang, Kai and Zuo, Wangmeng and Chen, Yunjin and Meng, Deyu and Zhang, Lei},
#    journal={IEEE Transactions on Image Processing},
#    year={2017},
#    volume={26},
#    number={7},
#    pages={3142-3155},
#  }
# modified version of the code from https://github.com/cszn/DnCNN
# ============================================================================

# run this to test the model

import argparse
import os, time, datetime
import numpy as np
import torch.nn as nn
import torch.nn.init as init
import torch
from skimage.measure import compare_psnr, compare_ssim
from skimage.io import imread, imsave


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--set_dir', default='data', type=str, help='directory of test dataset')
    parser.add_argument('--set_names', default=['test066'], help='directory of test dataset')
    parser.add_argument('--sigma', default=37, type=int, help='noise level')
```

```
    parser.add_argument('--sigma', default=37, type=int, help='noise level')
    parser.add_argument('--model_path', default='models/model_001.pth', type=str, help='the model name')
    parser.add_argument('--result_dir', default='results', type=str, help='directory of test dataset')
    parser.add_argument('--save_result', action='store_false', help='save the denoised image')
    return parser.parse_args()


def log(*args, **kwargs):
     print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:"), *args, **kwargs)


def save_result(result, path):
    path = path if path.find('.') != -1 else path+'.png'
    ext = os.path.splitext(path)[-1]
    if ext in ('.txt', '.dlm'):
        np.savetxt(path, result, fmt='%2.4f')
    else:
        imsave(path, np.uint8(np.clip(result*255.0, 0, 255)))


def show(x, title=None, cbar=False, figsize=None):
    import matplotlib.pyplot as plt
    plt.figure(figsize=figsize)
    plt.imshow(x, interpolation='nearest', cmap='gray')
    if title:
        plt.title(title)
    if cbar:
        plt.colorbar()
    plt.show()


class DnCNN(nn.Module):

    def __init__(self, depth=17, n_channels=64, image_channels=1, use_bnorm=True, kernel_size=3):
        super(DnCNN, self).__init__()
        kernel_size = 3
        padding = 1
        layers = []
        layers.append(nn.Conv2d(in_channels=image_channels, out_channels=n_channels, kernel_size=kernel_size, padding=padding, bias=True))
        layers.append(nn.ReLU(inplace=True))
        for _ in range(depth-2):
            layers.append(nn.Conv2d(in_channels=n_channels, out_channels=n_channels, kernel_size=kernel_size, padding=padding, bias=False))
            layers.append(nn.BatchNorm2d(n_channels))
            layers.append(nn.ReLU(inplace=True))
        layers.append(nn.Conv2d(in_channels=n_channels, out_channels=image_channels, kernel_size=kernel_size, padding=padding, bias=False))
        self.dncnn = nn.Sequential(*layers)
        self._initialize_weights()

    def forward(self, x):
        y = x
        out = self.dncnn(x)
        return y-out

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                init.orthogonal_(m.weight)
                print('init weight')
                if m.bias is not None:
                    init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                init.constant_(m.weight, 1)
                init.constant_(m.bias, 0)


if __name__ == '__main__':

    args = parse_args()

    model = torch.load(args.model_path)
    #model = torch.load with map_location=torch.device('cpu')
    log('load trained model')
```

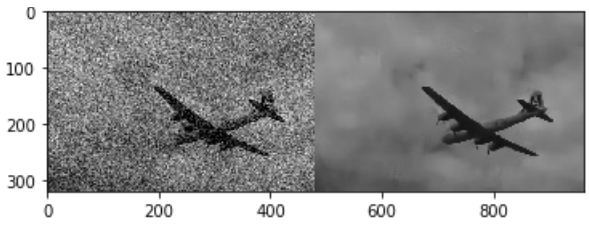
```
    model.eval()  # evaluation mode

    if torch.cuda.is_available():
        model = model.cuda()

    if not os.path.exists(args.result_dir):
        os.mkdir(args.result_dir)

    for set_cur in args.set_names:

        if not os.path.exists(os.path.join(args.result_dir, set_cur)):
            os.mkdir(os.path.join(args.result_dir, set_cur))
        psnrs = []
        ssims = []

        for im in os.listdir(os.path.join(args.set_dir, set_cur)):
            if im.endswith(".jpg") or im.endswith(".bmp") or im.endswith(".png"):

                x = np.array(imread(os.path.join(args.set_dir, set_cur, im)),
dtype=np.float32)/255.0
                np.random.seed(seed=0)  # for reproducibility
                #y = x + np.random.normal(0, args.sigma/255.0, x.shape)  # Add Gaussian noise witho
ut clipping
                y = x
                y = y.astype(np.float32)
                y_ = torch.from_numpy(y).view(1, -1, y.shape[0], y.shape[1])

                torch.cuda.synchronize()
                start_time = time.time()
                y_ = y_.cuda()
                x_ = model(y_)  # inference
                x_ = x_.view(y.shape[0], y.shape[1])
                x_ = x_.cpu()
                x_ = x_.detach().numpy().astype(np.float32)
                torch.cuda.synchronize()
                elapsed_time = time.time() - start_time
                print('%10s : %10s : %2.4f second' % (set_cur, im, elapsed_time))

                psnr_x_ = compare_psnr(x, x_)
                ssim_x_ = compare_ssim(x, x_)
                if args.save_result:
                    name, ext = os.path.splitext(im)
                    show(np.hstack((y, x_)))  # show the image
                    save_result(x_, path=os.path.join(args.result_dir, set_cur, name+'_clean'+ext))
# save the denoised image
                psnrs.append(psnr_x_)
                ssims.append(ssim_x_)
        psnr_avg = np.mean(psnrs)
        ssim_avg = np.mean(ssims)
        psnrs.append(psnr_avg)
        ssims.append(ssim_avg)
        if args.save_result:
            save_result(np.hstack((psnrs, ssims)), path=os.path.join(args.result_dir, set_cur, 'res
ults.txt'))
        log('Datset: {0:10s} \n  PSNR = {1:2.2f}dB, SSIM = {2:1.4f}'.format(set_cur, psnr_avg, ssim
_avg))
```

In [12]:

```
%run main_test_d_fine.py
```

```
2019-11-24 20:20:39: load trained model
   test066 : XZHANG6@TCD.IE_066.png : 0.0797 second
```

```
/notebooks/main_test_d_fine.py:141: UserWarning: DEPRECATED: skimage.measure.compare_psnr has been
moved to skimage.metrics.peak_signal_noise_ratio. It will be removed from skimage.measure in
version 0.18.
  psnr_x_ = compare_psnr(x, x_)
/notebooks/main_test_d_fine.py:142: UserWarning: DEPRECATED: skimage.measure.compare_ssim has been
moved to skimage.metrics.structural_similarity. It will be removed from skimage.measure in version
0.18.
  ssim_x_ = compare_ssim(x, x_)
```




```
2019-11-24 20:20:40: Datset: test066
  PSNR = 16.99dB, SSIM = 0.2012
```

In [13]:

```
import pandas as pd
final=pd.DataFrame([['23.16dB','0.8391'],['16.99dB','0.2012']])
final.to_csv('XZHANG6@TCD.IE_psnr.csv',index=['pre','fine'],header=['PSNR','SSIM'])
final.head(3)
```

Out[13]:

	0	1
0	23.16dB	0.8391
1	16.99dB	0.2012

e) Modify the network to output the estimated noise (Residual image in Fig.1) instead of the clean image. Generate noise from YourEmail_XXX.png by your finetuned network and save it as a grayscale image centered around intensity 127 named YourEmail_XXX_noise.png. Report the standard deviation of the noise in this image.

In [ ]:

```
# %load main_test_e.py

# ================================================================================
#  @article{zhang2017beyond,
#    title={Beyond a {Gaussian} denoiser: Residual learning of deep {CNN} for image denoising},
#    author={Zhang, Kai and Zuo, Wangmeng and Chen, Yunjin and Meng, Deyu and Zhang, Lei},
#    journal={IEEE Transactions on Image Processing},
#    year={2017},
#    volume={26},
#    number={7},
#    pages={3142-3155},
#  }
# modified version of the code from https://github.com/cszn/DnCNN
# ================================================================================

# run this to test the model

import argparse
import os, time, datetime
import numpy as np
import torch.nn as nn
import torch.nn.init as init
import torch
from skimage.measure import compare_psnr, compare_ssim
from skimage.io import imread, imsave
from skimage import exposure
import cv2


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--set_dir', default='data', type=str, help='directory of test dataset')
    parser.add_argument('--set_names', default=['test066'], help='directory of test dataset')
    parser.add_argument('--sigma', default=37, type=int, help='noise level')
    parser.add_argument('--model_path', default='models/model_001.pth', type=str, help='the model n
ame')
    parser.add_argument('--result_dir', default='results', type=str, help='directory of test datase
t')
    parser.add_argument('--save_result', action='store_false', help='save the denoised image')
    return parser.parse_args()
```

```
    return parser.parse_args()


def log(*args, **kwargs):
     print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S:"), *args, **kwargs)


def save_result(result, path):
    path = path if path.find('.') != -1 else path+'.png'
    ext = os.path.splitext(path)[-1]
    if ext in ('.txt', '.dlm'):
        np.savetxt(path, result, fmt='%2.4f')
    else:
        imsave(path, np.uint8(np.clip(result*255.0, 0, 255)))


def show(x, title=None, cbar=False, figsize=None):
    import matplotlib.pyplot as plt
    plt.figure(figsize=figsize)
    plt.imshow(x, interpolation='nearest', cmap='gray')
    if title:
        plt.title(title)
    if cbar:
        plt.colorbar()
    plt.show()


class DnCNN(nn.Module):

    def __init__(self, depth=17, n_channels=64, image_channels=1, use_bnorm=True, kernel_size=3):
        super(DnCNN, self).__init__()
        kernel_size = 3
        padding = 1
        layers = []
        layers.append(nn.Conv2d(in_channels=image_channels, out_channels=n_channels, kernel_size=ke
rnel_size, padding=padding, bias=True))
        layers.append(nn.ReLU(inplace=True))
        for _ in range(depth-2):
            layers.append(nn.Conv2d(in_channels=n_channels, out_channels=n_channels, kernel_size=ke
rnel_size, padding=padding, bias=False))
            layers.append(nn.BatchNorm2d(n_channels))
            layers.append(nn.ReLU(inplace=True))
        layers.append(nn.Conv2d(in_channels=n_channels, out_channels=image_channels, kernel_size=ke
rnel_size, padding=padding, bias=False))
        self.dncnn = nn.Sequential(*layers)
        self._initialize_weights()

    def forward(self, x):
        y = x
        out = self.dncnn(x)
        return out

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                init.orthogonal_(m.weight)
                print('init weight')
                if m.bias is not None:
                    init.constant_(m.bias, 0)
            elif isinstance(m, nn.BatchNorm2d):
                init.constant_(m.weight, 1)
                init.constant_(m.bias, 0)


if __name__ == '__main__':

    args = parse_args()

    model = torch.load(args.model_path)
    #model = torch.load with map_location=torch.device('cpu')
    log('load trained model')

    model.eval()  # evaluation mode

    if torch.cuda.is_available():
        model = model.cuda()

    if not os.path.exists(args.result_dir):
```

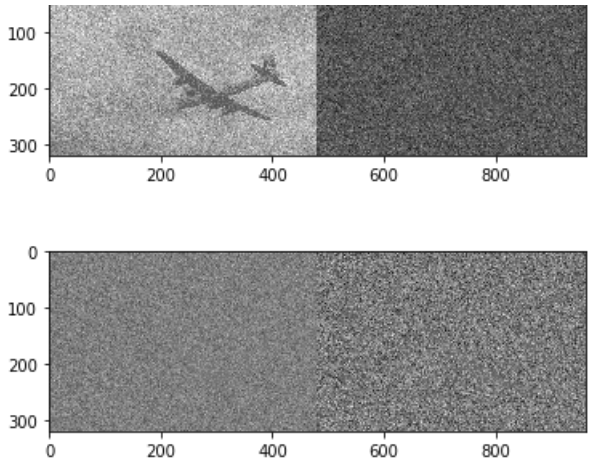
```
        os.mkdir(args.result_dir)

    for set_cur in args.set_names:

        if not os.path.exists(os.path.join(args.result_dir, set_cur)):
            os.mkdir(os.path.join(args.result_dir, set_cur))
        psnrs = []
        ssims = []

        for im in os.listdir(os.path.join(args.set_dir, set_cur)):
            if im.endswith(".jpg") or im.endswith(".bmp") or im.endswith(".png"):

                x = np.array(imread(os.path.join(args.set_dir, set_cur, im)),
dtype=np.float32)/255.0
                np.random.seed(seed=0)  # for reproducibility
                #y = x + np.random.normal(0, args.sigma/255.0, x.shape)  # Add Gaussian noise witho
ut clipping
                y = x
                y = y.astype(np.float32)
                y_ = torch.from_numpy(y).view(1, -1, y.shape[0], y.shape[1])

                torch.cuda.synchronize()
                start_time = time.time()
                y_ = y_.cuda()
                x_ = model(y_)  # inference
                x_ = x_.view(y.shape[0], y.shape[1])
                x_ = x_.cpu()
                x_ = x_.detach().numpy().astype(np.float32)
                torch.cuda.synchronize()
                elapsed_time = time.time() - start_time
                print('%10s : %10s : %2.4f second' % (set_cur, im, elapsed_time))

                psnr_x_ = compare_psnr(x, x_)
                ssim_x_ = compare_ssim(x, x_)
                if args.save_result:
                    name, ext = os.path.splitext(im)
                    show(np.hstack((y, x_)))  # show the image
                    xarray=np.array(x_)
                    xgray=exposure.rescale_intensity(xarray)
                    show(np.hstack((x_,xgray)))
                    save_result(x_, path=os.path.join(args.result_dir, set_cur, name+'_dncnn'+ext))
# save the denoised image
                    save_result(xgray, path=os.path.join(args.result_dir, set_cur, name+'_noise'+ex
t))  # save the denoised image
                psnrs.append(psnr_x_)
                ssims.append(ssim_x_)
        psnr_avg = np.mean(psnrs)
        ssim_avg = np.mean(ssims)
        psnrs.append(psnr_avg)
        ssims.append(ssim_avg)
        if args.save_result:
            save_result(np.hstack((psnrs, ssims)), path=os.path.join(args.result_dir, set_cur, 'res
ults.txt'))
        log('Datset: {0:10s} \n  PSNR = {1:2.2f}dB, SSIM = {2:1.4f}'.format(set_cur, psnr_avg, ssim
_avg))
```

In [17]:

```
%run main_test_e.py
```

```
2019-11-24 20:23:43: load trained model
   test066 : XZHANG6@TCD.IE_066.png : 0.0798 second
```

```
/notebooks/main_test_e.py:143: UserWarning: DEPRECATED: skimage.measure.compare_psnr has been move
d to skimage.metrics.peak_signal_noise_ratio. It will be removed from skimage.measure in version
0.18.
  psnr_x_ = compare_psnr(x, x_)
/notebooks/main_test_e.py:144: UserWarning: DEPRECATED: skimage.measure.compare_ssim has been move
d to skimage.metrics.structural_similarity. It will be removed from skimage.measure in version
0.18.
  ssim_x_ = compare_ssim(x, x_)
```




```
2019-11-24 20:23:44: Datset: test066
  PSNR = 6.49dB, SSIM = -0.0027
```