assignment_07

November 5, 2021

1 Supervised image denoising

1.1 Import libraries

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
import torch
import torchvision
from torch.utils.data import Dataset
from os import listdir
from os.path import join
from torchvision.transforms import Compose, ToTensor, ToPILImage, Resize,␣
↪Lambda, Normalize, Grayscale
from torch.utils.data import DataLoader
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from math import log10
from tqdm.notebook import tqdm
import os
```

1.2 Load data

[ ]:
```
directory_data  = 'drive/MyDrive/'
filename_data   = 'assignment_07_data.npz'
data            = np.load(os.path.join(directory_data, filename_data))


original_train = data['original_train']
noise_train    = data['noise_train']
```

```
original_test = data['original_test']
noise_test    = data['noise_test']

print('***************************************************')
print('size of original_train : ', original_train.shape)
print('size of noise_train    : ', noise_train.shape)
print('***************************************************')
print('size of original_test : ', original_test.shape)
print('size of noise_test    : ', noise_test.shape)
print('***************************************************')
print('number of training image :', original_train.shape[0])
print('height of training image :', original_train.shape[1])
print('width of training image  :', original_train.shape[2])
print('***************************************************')
print('number of testing image :', original_test.shape[0])
print('height of testing image :', original_test.shape[1])
print('width of testing image  :', original_test.shape[2])
print('***************************************************')
```

```
***************************************************
size of original_train :  (2000, 64, 64)
size of noise_train    :  (2000, 64, 64)
***************************************************
size of original_test :  (900, 64, 64)
size of noise_test    :  (900, 64, 64)
***************************************************
number of training image : 2000
height of training image : 64
width of training image  : 64
***************************************************
number of testing image : 900
height of testing image : 64
width of testing image  : 64
***************************************************
```

1.3 Hyper parameters

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

number_epoch    = 500
size_minibatch  = 8
learning_rate   = 0.1
momentum        = 0.9
weight_decay    = 0.0001
```

1.4 Costumize dataloader for pytorch

[ ]:
```
class dataset (Dataset):
  def  __init__(self, original,noise):

    self.original = original
    self.noise    = noise

  def __getitem__(self, index):

    original    = self.original[index]
    noise       = self.noise[index]

    original   = torch.FloatTensor(original).unsqueeze(dim=0)
    noise      = torch.FloatTensor(noise).unsqueeze(dim=0)


    return (original , noise)

  def __len__(self):

     return self.original.shape[0]
```

1.5 Construct datasets and dataloaders for training and testing

[ ]:
```
dataset_train = dataset(original_train, noise_train)
dataset_test  = dataset(original_test, noise_test)

dataloader_train = DataLoader(dataset_train, batch_size=size_minibatch,␣
 ↪shuffle=True, drop_last=True, num_workers=2)
dataloader_test  = DataLoader(dataset_test,  batch_size=size_minibatch,␣
 ↪shuffle=False, drop_last=True, num_workers=2)
```

1.6 Shape of the data with data loader

[ ]:
```
(original_train, noise_train)   = dataset_train[0]
(original_test, noise_test)     = dataset_test[0]

print('************************************************************')
print('shape of the original image in the training dataset:', original_train.
 ↪shape)
print('shape of the noisy image in the training dataset:', noise_train.shape)
print('************************************************************')
print('shape of the original image in the testing dataset:', original_test.
 ↪shape)
```

```
print('shape of the noisy image in the testing dataset:', noise_test.shape)
print('************************************************************')
```

```
************************************************************
shape of the original image in the training dataset: torch.Size([1, 64, 64])
shape of the noisy image in the training dataset: torch.Size([1, 64, 64])
************************************************************
shape of the original image in the testing dataset: torch.Size([1, 64, 64])
shape of the noisy image in the testing dataset: torch.Size([1, 64, 64])
************************************************************
```

1.7 Class for the neural network

[ ]:
```
class Network(nn.Module):
    def __init__(self):
        super(Network,self).__init__()

        # ------------------------------------------------
        # Encoder
        # ------------------------------------------------
        self.e_layer1 = nn.Sequential(
                        nn.Conv2d(in_channels=1, out_channels=256,␣
↪kernel_size=3, stride=1, padding=1, bias=True),
                        nn.MaxPool2d(2,2),
                        nn.ReLU(),
                        nn.BatchNorm2d(256),
        )

        # ------------------------------------------------
        # Decoder
        # ------------------------------------------------
        # self.d_layer1 = nn.Sequential(
        #                 nn.Upsample(scale_factor=2, mode='bilinear',␣
↪align_corners=False),
        #                 nn.Conv2d(in_channels=16, out_channels=8,␣
↪kernel_size=3, stride=1, padding=1, bias=True),
        #                 nn.ReLU(),
        #                 nn.BatchNorm2d(8),
        # )

        self.d_layer1 = nn.Sequential(
                        nn.Upsample(scale_factor=2, mode='bilinear',␣
↪align_corners=False),
                        nn.Conv2d(in_channels=256, out_channels=1,␣
↪kernel_size=3, stride=1, padding=1, bias=True),
                        nn.Sigmoid(),
        )
```

```
        # ------------------------------------------------
        # Network
        # ------------------------------------------------
        self.network = nn.Sequential(
                        self.e_layer1,
                        #self.e_layer2,
                        self.d_layer1,
                        #self.d_layer2,
        )

        self.initialize_weight()

    def forward(self,x):

        out = self.network(x)

        return out

    # ========================================================================
    # initialize weights
    # ========================================================================
    def initialize_weight(self):

        for m in self.network.modules():

            if isinstance(m, nn.Conv2d):
                nn.init.xavier_uniform_(m.weight)
                #nn.init.constant_(m.weight, 0.1)

                if m.bias is not None:

                    nn.init.constant_(m.bias, 1)
                    pass

            elif isinstance(m, nn.BatchNorm2d):

                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 1)

            elif isinstance(m, nn.Linear):
                nn.init.xavier_uniform_(m.weight)
                #nn.init.constant_(m.weight, 0.1)

                if m.bias is not None:

                    nn.init.constant_(m.bias, 1)
```

```
                    pass
```

1.8 Build the network

[ ]:
```
model = Network().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate,␣
 ↪momentum=momentum , weight_decay=weight_decay)
```

1.9 Compute prediction (denoised image)

[ ]:
```
def compute_prediction(model, input):

    denoise = model(input)

    return denoise
```

1.10 Compute loss

[ ]:
```
def compute_loss(noise, original):

    criterion   = nn.MSELoss()
    loss        = criterion(noise, original)
    loss_value  = loss.item()

    return loss, loss_value
```

1.11 Compute PSNR metric

[ ]:
```
def compute_PSNR (loss):

  if (loss==0.):
    PSNR=100
  else :
    PSNR=10*log10(1/loss)

  return PSNR
```

1.12 Variable for the learning curves

[ ]:
```
loss_mean_train     = np.zeros(number_epoch)
loss_std_train      = np.zeros(number_epoch)
PSNR_mean_train     = np.zeros(number_epoch)
PSNR_std_train      = np.zeros(number_epoch)

loss_mean_test      = np.zeros(number_epoch)
```

```
loss_std_test       = np.zeros(number_epoch)
PSNR_mean_test      = np.zeros(number_epoch)
PSNR_std_test       = np.zeros(number_epoch)
```

1.13 Train

[ ]:
```
def train(model, dataloader):

    loss_epoch      = []
    psnr_epoch      = []

    model.train()

    for index_batch, (original, noise) in enumerate(dataloader):

        original = original.to(device)
        noise    = noise.to(device)
        #␣
↪==========================================================================␣
↪
        # complete the following codes
        #
        denoise             = compute_prediction(model,noise)
        loss, loss_value    = compute_loss(denoise,original)
        #␣
↪==========================================================================␣
↪
        psnr                = compute_PSNR(loss_value)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        loss_epoch.append(loss_value)
        psnr_epoch.append(psnr)

    loss_mean_epoch     = np.mean(loss_epoch)
    loss_std_epoch      = np.std(loss_epoch)

    psnr_mean_epoch = np.mean(psnr_epoch)
    psnr_std_epoch  = np.std(psnr_epoch)

    loss        = {'mean' : loss_mean_epoch, 'std' : loss_std_epoch}
    psnr        = {'mean' : psnr_mean_epoch, 'std' : psnr_std_epoch}

    return (loss, psnr)
```

1.14 Test

[ ]:
```
def test(model, dataloader):

    loss_epoch      = []
    psnr_epoch      = []

    model.eval()

    for index_batch, (original, noise) in enumerate(dataloader):

        original = original.to(device)
        noise    = noise.to(device)

        #␣
 ↪==========================================================================␣
 ↪
        # complete the following codes
        #
        denoise             = compute_prediction(model,noise)
        loss, loss_value    = compute_loss(denoise,original)
        #␣
 ↪==========================================================================␣
 ↪
        psnr                = compute_PSNR(loss_value)

        loss_epoch.append(loss_value)
        psnr_epoch.append(psnr)

    loss_mean_epoch     = np.mean(loss_epoch)
    loss_std_epoch      = np.std(loss_epoch)

    psnr_mean_epoch = np.mean(psnr_epoch)
    psnr_std_epoch  = np.std(psnr_epoch)

    loss        = {'mean' : loss_mean_epoch, 'std' : loss_std_epoch}
    psnr        = {'mean' : psnr_mean_epoch, 'std' : psnr_std_epoch}

    return (loss, psnr)
```

1.15 train and test

[ ]:
```
#␣
 ↪==========================================================================
#
# iterations for epochs
#
```

```
#␣
 ↪==========================================================================
for i in tqdm(range(number_epoch)):

    #␣
 ↪==========================================================================
    #
    # training
    #
    #␣
 ↪==========================================================================
    (loss_train, psnr_train) = train(model, dataloader_train)

    loss_mean_train[i]      = loss_train['mean']
    loss_std_train[i]       = loss_train['std']

    PSNR_mean_train[i]  = psnr_train['mean']
    PSNR_std_train[i]   = psnr_train['std']

    #␣
 ↪==========================================================================
    #
    # testing
    #
    #␣
 ↪==========================================================================
    (loss_test, psnr_test) = test(model, dataloader_test)

    loss_mean_test[i]      = loss_test['mean']
    loss_std_test[i]       = loss_test['std']

    PSNR_mean_test[i]  = psnr_test['mean']
    PSNR_std_test[i]   = psnr_test['std']
```

```
  0%|          | 0/500 [00:00<?, ?it/s]
```

---

2 DO NOT MODIFY THE CODES FROM HERE, BUT EXECUTE THEM

2.1 Plot functions

[ ]:
```
def plot_data_grid(data, index_data, nRow, nCol):

    fig, axes = plt.subplots(nRow, nCol, constrained_layout=True, figsize=(nCol␣
↪* 3, nRow * 3))

    for i in range(nRow):
        for j in range(nCol):

            k       = i * nCol + j
            index   = index_data[k]

            axes[i, j].imshow(data[index], cmap='gray', vmin=0, vmax=1)
            axes[i, j].xaxis.set_visible(False)
            axes[i, j].yaxis.set_visible(False)

    plt.show()
```

[ ]:
```
def plot_data_tensor_grid(data, index_data, nRow, nCol):

    fig, axes = plt.subplots(nRow, nCol, constrained_layout=True, figsize=(nCol␣
↪* 3, nRow * 3))

    data = data.detach().cpu().squeeze(axis=1)

    for i in range(nRow):
        for j in range(nCol):

            k       = i * nCol + j
            index   = index_data[k]

            axes[i, j].imshow(data[index], cmap='gray', vmin=0, vmax=1)
            axes[i, j].xaxis.set_visible(False)
            axes[i, j].yaxis.set_visible(False)

    plt.show()
```

[ ]:
```
def plot_curve_error(data_mean, data_std, x_label, y_label, title):

    plt.figure(figsize=(8, 6))
    plt.title(title)

    alpha = 0.3

    plt.plot(range(len(data_mean)), data_mean, '-', color = 'red')
```

```
    plt.fill_between(range(len(data_mean)), data_mean - data_std, data_mean +␣
↪data_std, facecolor = 'blue', alpha = alpha)

    plt.xlabel(x_label)
    plt.ylabel(y_label)

    plt.tight_layout()
    plt.show()
```

[ ]:
```
def print_curve(data, index):

    for i in range(len(index)):

        idx = index[i]
        val = data[idx]

        print('index = %2d, value = %12.10f' % (idx, val))
```

[ ]:
```
def get_data_last(data, index_start):

    data_last = data[index_start:]

    return data_last
```

[ ]:
```
def get_max_last_range(data, index_start):

    data_range = get_data_last(data, index_start)
    value = data_range.max()

    return value
```

[ ]:
```
def get_min_last_range(data, index_start):

    data_range = get_data_last(data, index_start)
    value = data_range.min()

    return value
```

---

3 functions for presenting the results

[ ]:
```
def function_result_01():

    print('[plot examples of the training clean images]')
```

```
    print('')

    nRow = 5
    nCol = 4
    index_data  = np.arange(0, nRow * nCol)
    original_train = dataset_train.original[index_data]

    plot_data_grid(original_train, index_data, nRow, nCol)
```

[ ]:
```
def function_result_02():

    print('[plot examples of the training noisy images]')
    print('')

    nRow = 5
    nCol = 4
    index_data  = np.arange(0, nRow * nCol)
    noise_train = dataset_train.noise[index_data]

    plot_data_grid(noise_train, index_data, nRow, nCol)
```

[ ]:
```
def function_result_03():

    print('[plot examples of the training denoising results]')
    print('')

    nRow = 5
    nCol = 4
    index_data          = np.arange(0, nRow * nCol)
    image_train         = torch.FloatTensor(dataset_train.original[index_data]).
↪unsqueeze(dim=1).to(device)
    prediction_train    = compute_prediction(model, image_train)

    plot_data_tensor_grid(prediction_train, index_data, nRow, nCol)
```

[ ]:
```
def function_result_04():

    print('[plot examples of the testing clean images]')
    print('')

    nRow = 5
    nCol = 4
    index_data = np.arange(0, nRow * nCol)
    original_test = dataset_test.original[index_data]

    plot_data_grid(original_test, index_data, nRow, nCol)
```

[ ]:
```
def function_result_05():

    print('[plot examples of the testing noise images]')
    print('')

    nRow = 5
    nCol = 4
    index_data = np.arange(0, nRow * nCol)
    noise_test = dataset_test.noise[index_data]

    plot_data_grid(noise_test, index_data, nRow, nCol)
```

[ ]:
```
def function_result_06():

    print('[plot examples of the testing denoising results]')
    print('')

    nRow = 5
    nCol = 4
    index_data      = np.arange(0, nRow * nCol)
    image_test      = torch.FloatTensor(dataset_test.original[index_data]).
↪unsqueeze(dim=1).to(device)
    prediction_test = compute_prediction(model, image_test)

    plot_data_tensor_grid(prediction_test, index_data, nRow, nCol)
```

[ ]:
```
def function_result_07():

    print('[plot the training loss]')
    print('')

    plot_curve_error(loss_mean_train, loss_std_train, 'epoch', 'loss', 'loss␣
↪(training)')
```

[ ]:
```
def function_result_08():

    print('[plot the training PSNR]')
    print('')

    plot_curve_error(PSNR_mean_train, PSNR_std_train, 'epoch', 'PSNR', 'PSNR␣
↪(training)')
```

[ ]:
```
def function_result_09():

    print('[plot the testing loss]')
    print('')
```

```
    plot_curve_error(loss_mean_test, loss_std_test, 'epoch', 'loss', 'loss␣
↪(testing)')
```

[ ]:
```
def function_result_10():

    print('[plot the testing PSNR]')
    print('')

    plot_curve_error(PSNR_mean_test, PSNR_std_test, 'epoch', 'PSNR', 'PSNR␣
↪(testing)')
```

[ ]:
```
def function_result_11():

    print('[print the training loss at the last 10 epochs]')
    print('')

    data_last = get_data_last(loss_mean_train, -10)
    index = np.arange(0, 10)
    print_curve(data_last, index)
```

[ ]:
```
def function_result_12():

    print('[print the training PSNR at the last 10 epochs]')
    print('')

    data_last = get_data_last(PSNR_mean_train, -10)
    index = np.arange(0, 10)
    print_curve(data_last, index)
```

[ ]:
```
def function_result_13():

    print('[print the testing loss at the last 10 epochs]')
    print('')

    data_last = get_data_last(loss_mean_test, -10)
    index = np.arange(0, 10)
    print_curve(data_last, index)
```

[ ]:
```
def function_result_14():

    print('[print the testing PSNR at the last 10 epochs]')
    print('')

    data_last = get_data_last(PSNR_mean_test, -10)
    index = np.arange(0, 10)
    print_curve(data_last, index)
```

[ ]:
```
def function_result_15():

    print('[print the best training PSNR within the last 10 epochs]')
    print('')

    value = get_max_last_range(PSNR_mean_train, -10)
    print('best training PSNR = %12.10f' % (value))
```

[ ]:
```
def function_result_16():

    print('[print the best testing PSNR within the last 10 epochs]')
    print('')

    value = get_max_last_range(PSNR_mean_test, -10)
    print('best testing PSNR = %12.10f' % (value))
```

---

4 RESULTS

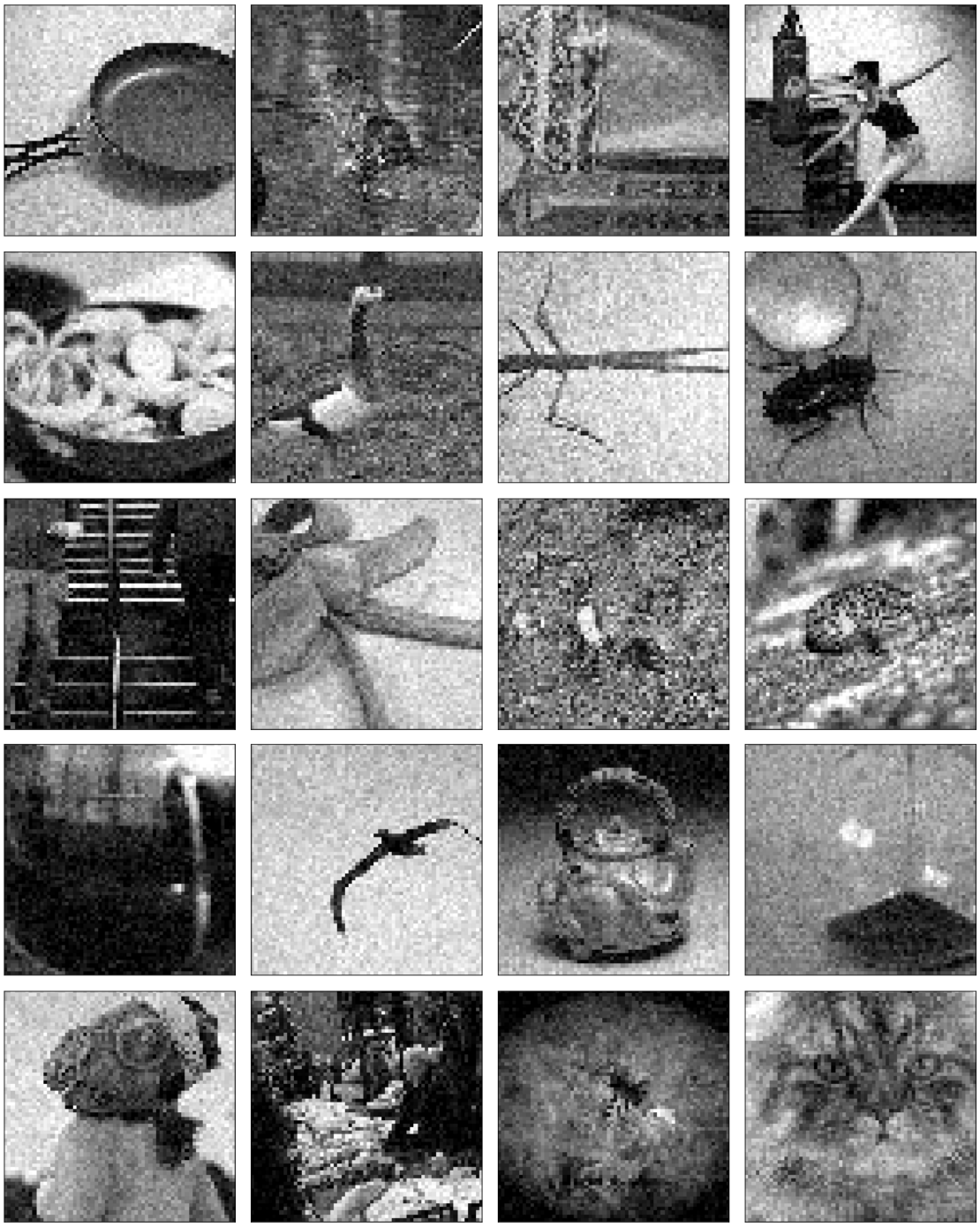
[ ]:
```
number_result = 16

for i in range(number_result):

    title           = '# RESULT # {:02d}'.format(i+1)
    name_function   = 'function_result_{:02d}()'.format(i+1)

    print('')
    ␣
 ↪print('################################################################################')
    print('#')
    print(title)
    print('#')
    ␣
 ↪print('################################################################################')
    print('')

    eval(name_function)
```

```
################################################################################
#
# RESULT # 01
#
################################################################################
```

[plot examples of the training clean images]

##

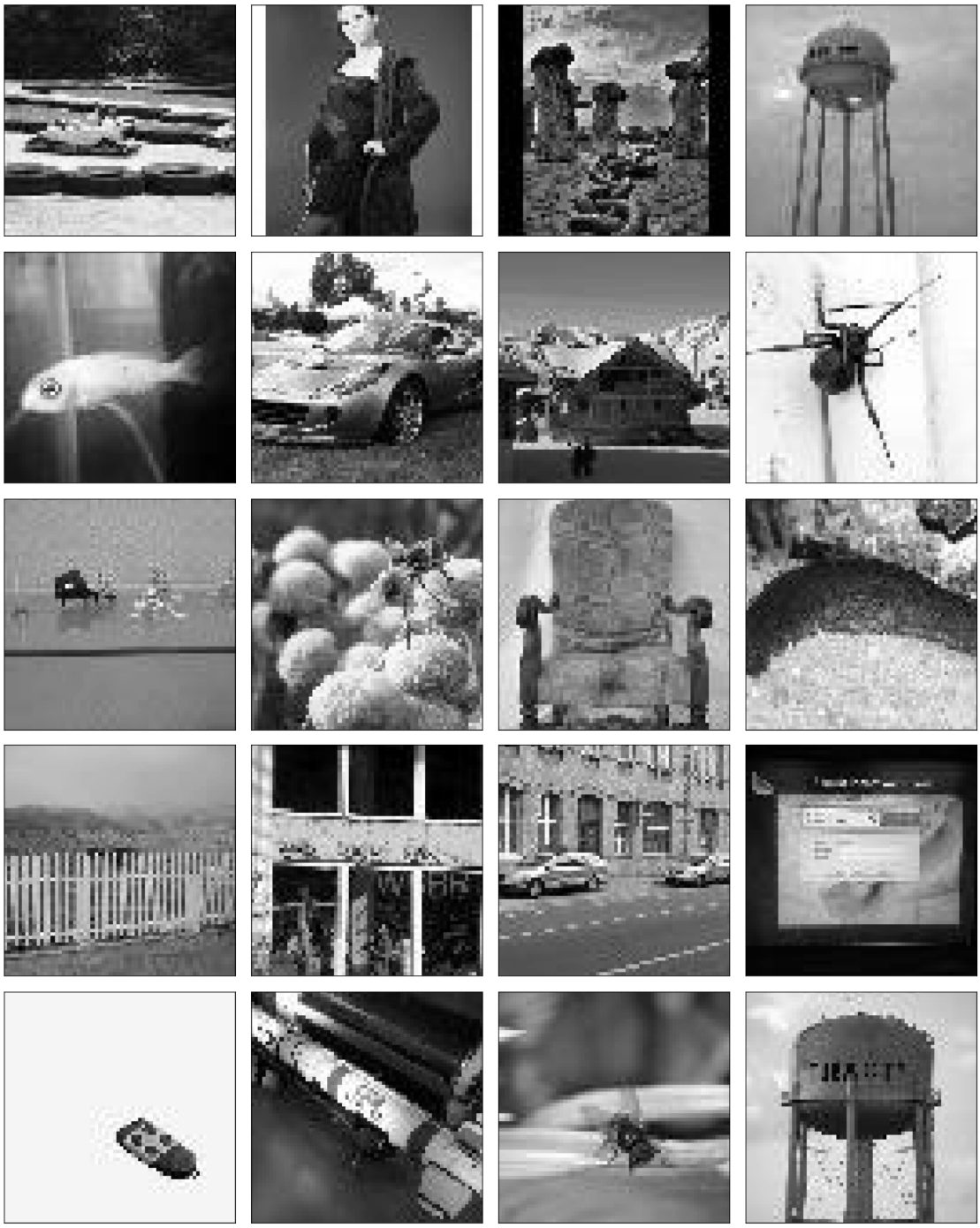
```
#
# RESULT # 02
#
################################################################################

[plot examples of the training noisy images]
```



```
##############################################################################
#
# RESULT # 03
#
##############################################################################

[plot examples of the training denoising results]
```

```
#############################################################################
#
# RESULT # 04
#
#############################################################################
```

[plot examples of the testing clean images]



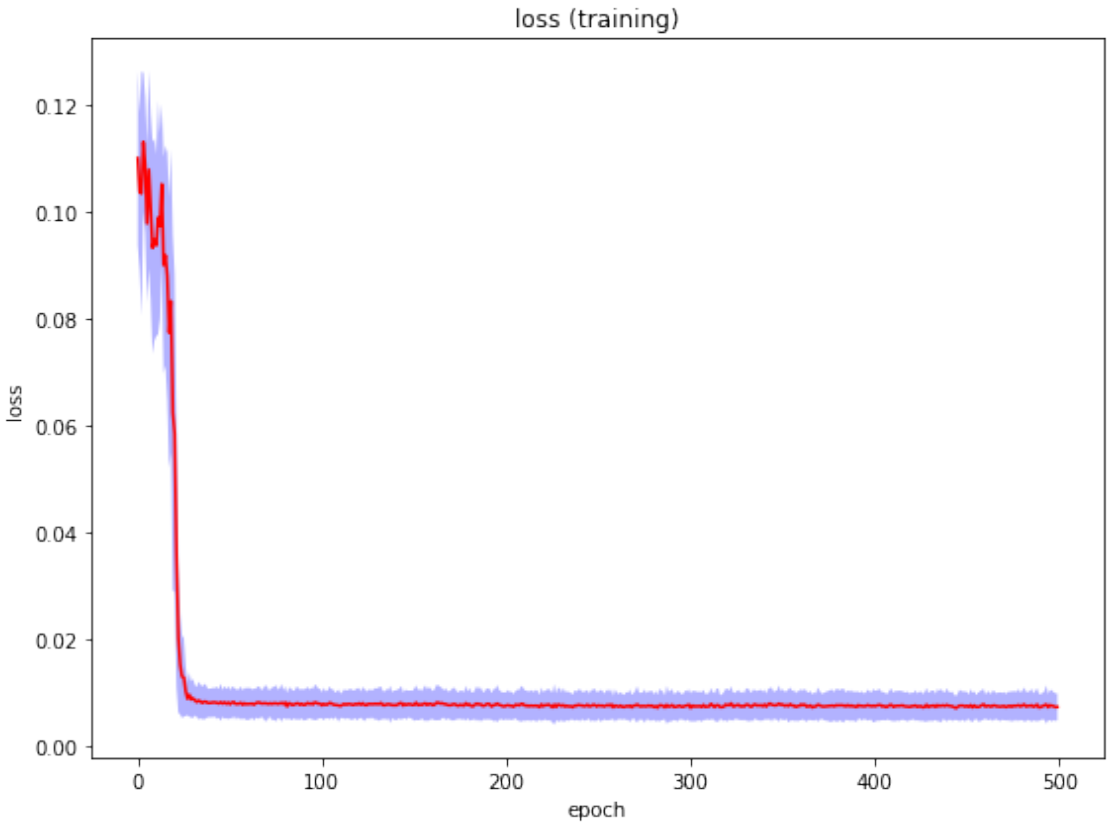
##

```
#
# RESULT # 05
#
################################################################################

[plot examples of the testing noise images]
```

```
################################################################################
#
# RESULT # 06
#
################################################################################

[plot examples of the testing denoising results]
```

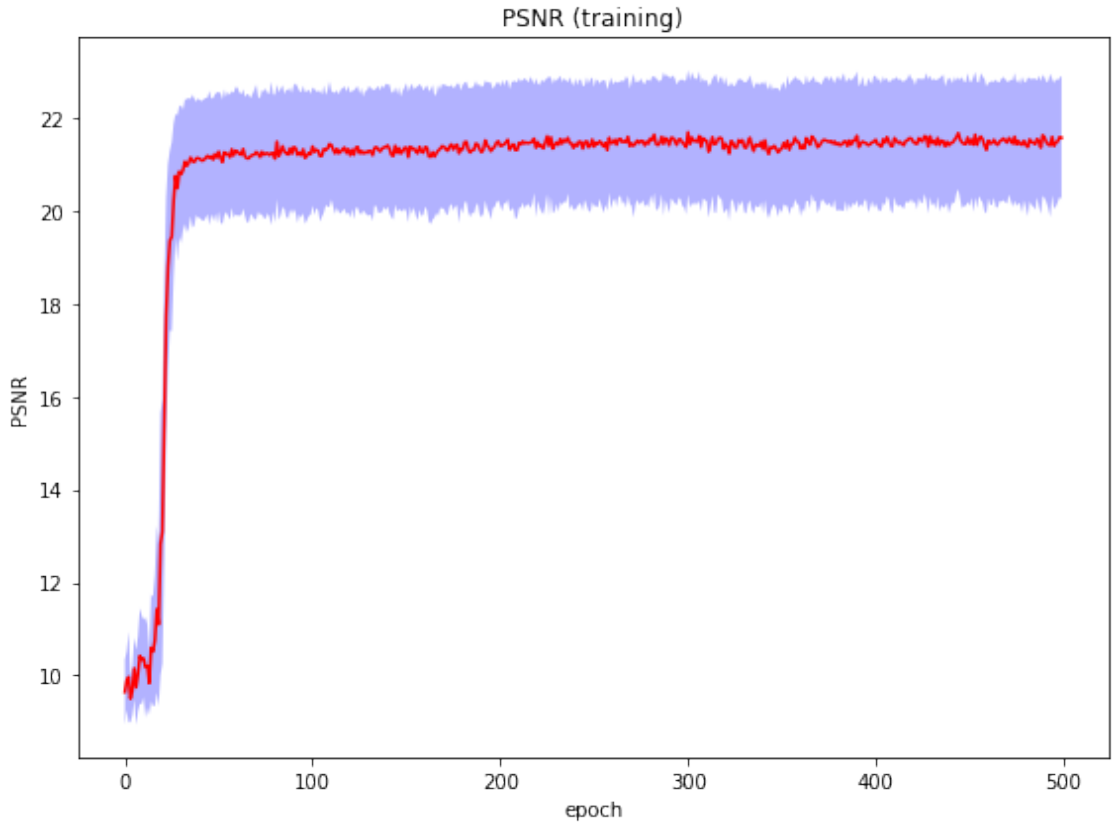
```
################################################################################
#
# RESULT # 07
#
################################################################################
```

[plot the training loss]

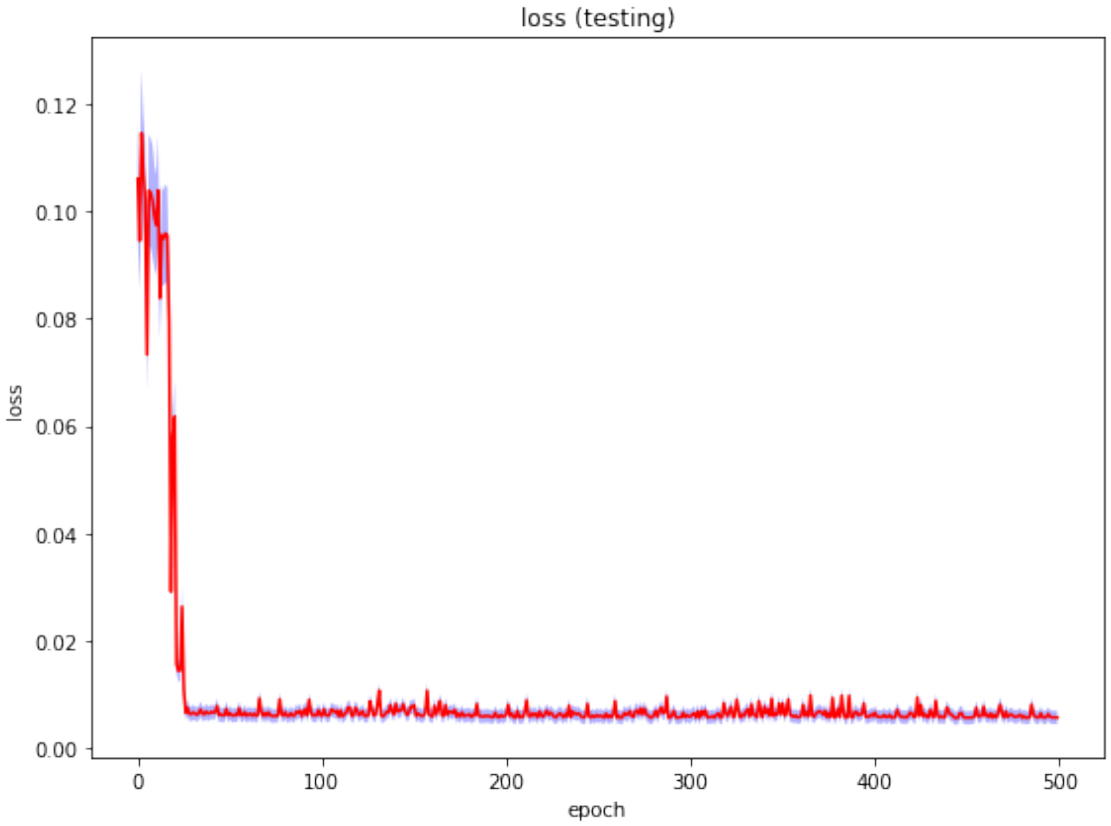



```
################################################################################
#
# RESULT # 08
#
################################################################################
```

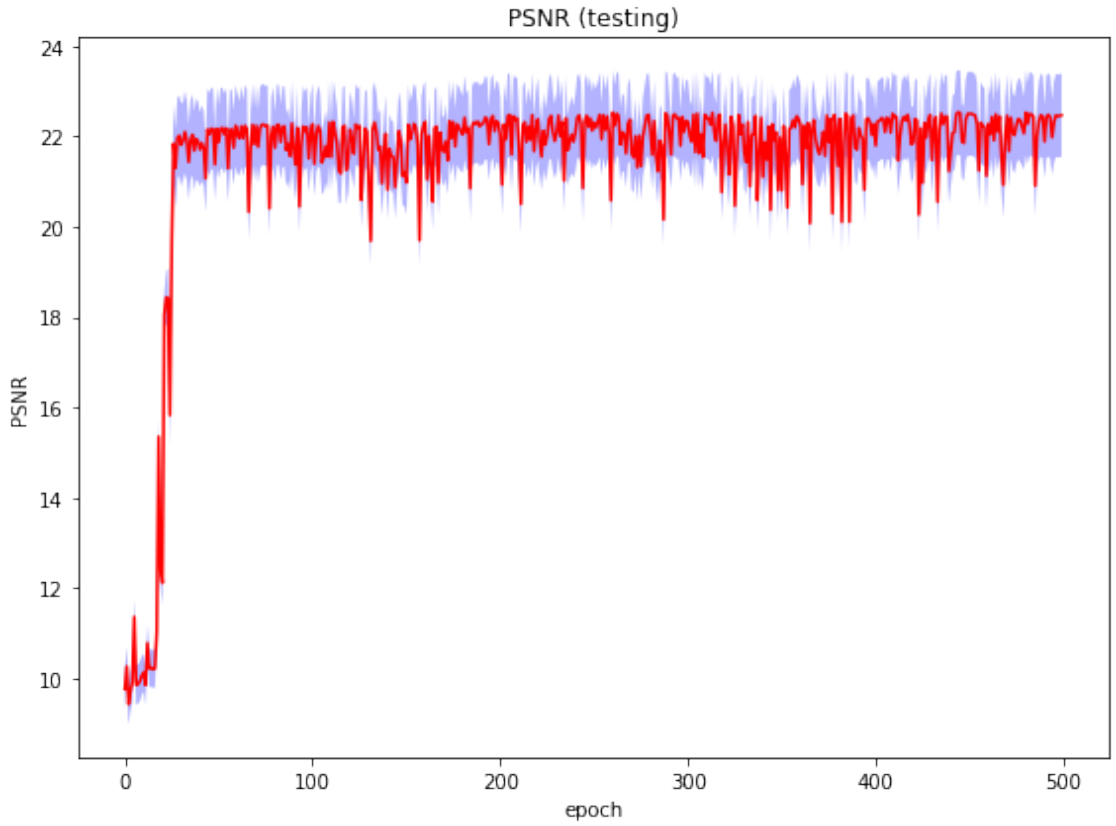
[plot the training PSNR]




```
################################################################################
#
# RESULT # 09
#
################################################################################

[plot the testing loss]
```




```
################################################################################
#
# RESULT # 10
#
################################################################################

[plot the testing PSNR]
```




```
################################################################################
#
# RESULT # 11
#
################################################################################

[print the training loss at the last 10 epochs]

index =  0, value = 0.0072790234
index =  1, value = 0.0074423958
index =  2, value = 0.0075753644
index =  3, value = 0.0077536066
index =  4, value = 0.0072317056
index =  5, value = 0.0076201677
index =  6, value = 0.0075205406
index =  7, value = 0.0074213842
index =  8, value = 0.0072590623
index =  9, value = 0.0072939374

################################################################################
```

```
#
# RESULT # 12
#
################################################################################

[print the training PSNR at the last 10 epochs]

index =  0, value = 21.5832241613
index =  1, value = 21.4867989431
index =  2, value = 21.4364820133
index =  3, value = 21.4227068090
index =  4, value = 21.6046217554
index =  5, value = 21.4080423482
index =  6, value = 21.4816717632
index =  7, value = 21.5080287184
index =  8, value = 21.6012141503
index =  9, value = 21.5844034491

################################################################################
#
# RESULT # 13
#
################################################################################

[print the testing loss at the last 10 epochs]

index =  0, value = 0.0065836407
index =  1, value = 0.0058320655
index =  2, value = 0.0057772769
index =  3, value = 0.0058598632
index =  4, value = 0.0064515677
index =  5, value = 0.0059959378
index =  6, value = 0.0057942235
index =  7, value = 0.0058181359
index =  8, value = 0.0058008575
index =  9, value = 0.0057928766

################################################################################
#
# RESULT # 14
#
################################################################################

[print the testing PSNR at the last 10 epochs]

index =  0, value = 21.8895612746
index =  1, value = 22.4364757526
index =  2, value = 22.4811760703
```

```
index =  3, value = 22.4165728093
index =  4, value = 21.9816613640
index =  5, value = 22.3141353617
index =  6, value = 22.4673756426
index =  7, value = 22.4487729235
index =  8, value = 22.4614343401
index =  9, value = 22.4695872789

################################################################################
#
# RESULT # 15
#
################################################################################

[print the best training PSNR within the last 10 epochs]

best training PSNR = 21.6046217554

################################################################################
#
# RESULT # 16
#
################################################################################

[print the best testing PSNR within the last 10 epochs]

best testing PSNR = 22.4811760703
```

[ ]: