assignment_08_solution_latest

November 12, 2021

1 Unsupervised image denoising

1.1 Connect Google Drive

[ ]:
```
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).
```

1.2 Import libraries

[ ]:
```
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
```

1.3 Load data

[ ]:
```
directory_data  = 'drive/MyDrive'
filename_data   = 'assignment_08_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 noise_train    : ', noise_train.shape)
print('***************************************************')
print('size of noise_test    : ', noise_test.shape)
print('***************************************************')
print('number of training image :', noise_train.shape[0])
print('height of training image :', noise_train.shape[1])
print('width of training image  :', noise_train.shape[2])
print('***************************************************')
print('number of testing image :', noise_test.shape[0])
print('height of testing image :', noise_test.shape[1])
print('width of testing image  :', noise_test.shape[2])
print('***************************************************')
```

```
***************************************************
size of noise_train    :  (2000, 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
***************************************************
```

1.4 Hyper parameters

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

# ================================================
# determine optimal hyper-parameters to obtain best testing performance
number_epoch    = 300
size_minibatch  = 32
learning_rate   = 0.1
momentum        = 0.9
weight_decay    = 0.0001
weight_total_variation = 0.01
# ================================================
```

1.5 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]
```

1.6 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)
```

1.7 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])
************************************************************
```

1.8 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=64,␣
↪kernel_size=3, stride=1, padding=1, bias=True),
                        nn.BatchNorm2d(64),
                        nn.ReLU(),
                        nn.Dropout(0.5),
                        #nn.MaxPool2d(2,2),

        )
        self.e_layer3 = nn.Sequential(
                        nn.Conv2d(in_channels=64, out_channels=256,␣
↪kernel_size=3, stride=1, padding=1, bias=True),
                        nn.BatchNorm2d(256),
                        nn.ReLU(),
                        nn.Dropout(0.5),
                        #nn.MaxPool2d(2,2),
        )
        # ------------------------------------------------
        # Decoder
        # ------------------------------------------------

        self.d_layer1 = nn.Sequential(
                        #nn.Upsample(scale_factor=2, mode='bilinear',␣
↪align_corners=False),
                        nn.ConvTranspose2d(in_channels=256, out_channels=64,␣
↪kernel_size=3, stride=1, padding=1, bias=True),
                        nn.BatchNorm2d(64),
                        nn.ReLU(),
```

```
                        nn.Dropout(0.5),

        )

        self.d_layer3 = nn.Sequential(
                        #nn.Upsample(scale_factor=2, mode='bilinear',␣
↪align_corners=False),
                        nn.ConvTranspose2d(in_channels=64, out_channels=1,␣
↪kernel_size=3, stride=1, padding=1, bias=True),
                        nn.Sigmoid(),
                        #nn.Dropout(0.5),
        )


        # -----------------------------------------------
        # Network
        # -----------------------------------------------
        self.network = nn.Sequential(
                        self.e_layer1,
                        #self.e_layer2,
                        self.e_layer3,
                        self.d_layer1,
                        #self.d_layer2,
                        self.d_layer3,
        )
        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)

                if m.bias is not None:

                    nn.init.constant_(m.bias, 0.01)
```

```
                pass

            elif isinstance(m, nn.BatchNorm2d):

                #nn.init.xavier_uniform_(m.weight)
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0.01)

            elif isinstance(m, nn.Linear):
                nn.init.xavier_uniform_(m.weight)
                # nn.init.constant_(m.weight, 1)

                if m.bias is not None:

                    nn.init.constant_(m.bias, 0.01)
                    pass
```

1.9 Build the network

[ ]:
```
model = Network().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate,␣
 ↪momentum=momentum , weight_decay=weight_decay)
#optimizer = torch.optim.Adam(model.parameters(),lr=0.001,betas=(0.9,0.
 ↪999),eps=1e-08,weight_decay=0,amsgrad=False)
#scheduler = optim.lr_scheduler.LambdaLR(optimizer=optimizer,lr_lambda=lambda␣
 ↪epoch: 0.95**epoch,last_epoch=-1,verbose=False)
```

1.10 Compute prediction (denoised image)

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

    # ================================================
    # fill up the blank
    prediction = model(input)
    # ================================================

    return prediction
```

1.11 Compute loss

[ ]:
```
def compute_fidelity(input, prediction):

    mse = nn.MSELoss()

    # ================================================
    # fill up the blank
```

```
    loss_mse = mse(input,prediction)
    # ================================================

    loss_mse_value = loss_mse.item()

    return loss_mse, loss_mse_value
```

[ ]:
```
def compute_regularization(data, weight):

    bs_img, c_img, h_img, w_img = data.size()

    tv_height = torch.abs(data[:, :, 1:, :] - data[:, :, :- 1, :]).sum()
    tv_width  = torch.abs(data[:, :, :, 1:] - data[:, :, :, :-1]).sum()

    total_variation = (tv_height + tv_width) / (bs_img * c_img * h_img * w_img)

    # ================================================
    # fill up the blank
    loss_regularization = total_variation * weight
    # ================================================

    loss_regularization_value = loss_regularization.item()

    return loss_regularization, loss_regularization_value
```

[ ]:
```
def compute_loss(input, prediction, weight):

    (loss_fidelity, loss_fidelity_value) = compute_fidelity(input, prediction)
    (loss_regularization, loss_regularization_value) =␣
 ↪compute_regularization(prediction, weight)

    # ================================================
    # fill up the blank
    loss = loss_fidelity + loss_regularization
    # ================================================

    loss_value = loss.item()

    return loss, loss_value , loss_fidelity_value, loss_regularization_value
```

1.12 Compute PSNR metric

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

    if (mse==0.):
        PSNR=100
```

```
    else :
        PSNR=10*log10(1.0 / mse)

    return PSNR
```

1.13 Variable for the learning curves

[ ]:
```
loss_fidelity_mean_train       = np.zeros(number_epoch)
loss_fidelity_std_train        = np.zeros(number_epoch)
loss_regularization_mean_train = np.zeros(number_epoch)
loss_regularization_std_train  = np.zeros(number_epoch)
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_fidelity_mean_test       = np.zeros(number_epoch)
loss_fidelity_std_test        = np.zeros(number_epoch)
loss_regularization_mean_test = np.zeros(number_epoch)
loss_regularization_std_test  = 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)
```

1.14 Train

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

    loss_epoch          = []
    loss_fidelity_epoch = []
    loss_reg_epoch      = []
    psnr_epoch          = []

    model.train()

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

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

        prediction = compute_prediction(noise, model)
        (loss, loss_value, loss_fidelity_value, loss_regularization_value) =␣
↪compute_loss(noise, prediction, weight_total_variation)

        # ================================================
```

```
        # fill up the blank
        (mse, mse_value)    = compute_fidelity(prediction,original)
        psnr                = compute_PSNR(mse_value)
        # ================================================

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

        loss_epoch.append(loss_value)
        loss_fidelity_epoch.append(loss_fidelity_value)
        loss_reg_epoch.append(loss_regularization_value)
        psnr_epoch.append(psnr)

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

    loss_fidelity_mean_epoch     = np.mean(loss_fidelity_epoch)
    loss_fidelity_std_epoch      = np.std(loss_fidelity_epoch)

    loss_reg_mean_epoch     = np.mean(loss_reg_epoch)
    loss_reg_std_epoch      = np.std(loss_reg_epoch)

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

    loss                        = {'mean' : loss_mean_epoch, 'std' :␣
↪loss_std_epoch}
    loss_fidelity               = {'mean' : loss_fidelity_mean_epoch, 'std' :␣
↪loss_fidelity_std_epoch}
    loss_regularization         = {'mean' : loss_reg_mean_epoch, 'std' :␣
↪loss_reg_std_epoch}
    psnr                        = {'mean' : psnr_mean_epoch, 'std' :␣
↪psnr_std_epoch}

    return (loss, loss_fidelity, loss_regularization, psnr)
```

1.15 Test

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

    loss_epoch          = []
    loss_fidelity_epoch = []
    loss_reg_epoch      = []
    psnr_epoch          = []
```

```
    model.eval()

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

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

        prediction = compute_prediction(noise, model)
        (loss, loss_value, loss_fidelity_value, loss_regularization_value) =␣
↪compute_loss(noise, prediction, weight_total_variation)

        # ================================================
        # fill up the blank
        (mse, mse_value)    = compute_fidelity(prediction,original)
        psnr                = compute_PSNR(mse_value)
        # ================================================

        loss_epoch.append(loss_value)
        loss_fidelity_epoch.append(loss_fidelity_value)
        loss_reg_epoch.append(loss_regularization_value)
        psnr_epoch.append(psnr)

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

    loss_fidelity_mean_epoch     = np.mean(loss_fidelity_epoch)
    loss_fidelity_std_epoch      = np.std(loss_fidelity_epoch)

    loss_reg_mean_epoch     = np.mean(loss_reg_epoch)
    loss_reg_std_epoch      = np.std(loss_reg_epoch)

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

    loss                        = {'mean' : loss_mean_epoch, 'std' :␣
↪loss_std_epoch}
    loss_fidelity               = {'mean' : loss_fidelity_mean_epoch, 'std' :␣
↪loss_fidelity_std_epoch}
    loss_regularization         = {'mean' : loss_reg_mean_epoch, 'std' :␣
↪loss_reg_std_epoch}
    psnr                        = {'mean' : psnr_mean_epoch, 'std' :␣
↪psnr_std_epoch}

    return (loss, loss_fidelity, loss_regularization, psnr)
```

1.16 train and test

[ ]:
```
#␣
↪==========================================================================
#
# iterations for epochs
#
#␣
↪==========================================================================
for i in tqdm(range(number_epoch)):

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

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

    loss_fidelity_mean_train[i] = loss_fidelity_train['mean']
    loss_fidelity_std_train[i]  = loss_fidelity_train['std']

    loss_regularization_mean_train[i] = loss_reg_train['mean']
    loss_regularization_std_train[i]  = loss_reg_train['std']

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

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

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

    loss_fidelity_mean_test[i] = loss_fidelity_test['mean']
```

```
    loss_fidelity_std_test[i]  = loss_fidelity_test['std']

    loss_regularization_mean_test[i] = loss_reg_test['mean']
    loss_regularization_std_test[i]  = loss_reg_test['std']

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

```
  0%|          | 0/300 [00:00<?, ?it/s]
```

---

2 functions for visualizing the results

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 images]')
    print('')

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

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

[ ]:
```
def function_result_02():

    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.noise[index_data]).
↪unsqueeze(dim=1).to(device)
    prediction_train    = compute_prediction(image_train,model)

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

[ ]:
```
def function_result_03():

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

    nRow = 5
    nCol = 4
    index_data = np.arange(0, nRow * nCol)
```

```
    image_test = dataset_test.noise[index_data]

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

[ ]:
```
def function_result_04():

    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.noise[index_data]).
 ↪unsqueeze(dim=1).to(device)
    prediction_test = compute_prediction(image_test,model)

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

[ ]:
```
def function_result_05():

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

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

[ ]:
```
def function_result_06():

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

    plot_curve_error(loss_fidelity_mean_train, loss_fidelity_std_train,␣
 ↪'epoch', 'loss', 'training loss (fidelity)')
```

[ ]:
```
def function_result_07():

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

    plot_curve_error(loss_regularization_mean_train,␣
 ↪loss_regularization_std_train, 'epoch', 'loss', 'training loss␣
 ↪(regularization)')
```

[ ]:
```
def function_result_08():

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

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

[ ]:
```
def function_result_09():

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

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

[ ]:
```
def function_result_10():

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

    plot_curve_error(loss_fidelity_mean_test, loss_fidelity_std_test, 'epoch',␣
↪'loss', 'testing loss (fidelity)')
```

[ ]:
```
def function_result_11():

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

    plot_curve_error(loss_regularization_mean_test,␣
↪loss_regularization_std_test, 'epoch', 'loss', 'testing loss␣
↪(regularization)')
```

[ ]:
```
def function_result_12():

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

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

[ ]:
```
def function_result_13():

    print('[print the training loss (mean) 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_14():

    print('[print the training PSNR (mean) 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_15():

    print('[print the testing loss (mean) 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_16():

    print('[print the testing PSNR (mean) 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_17():

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

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

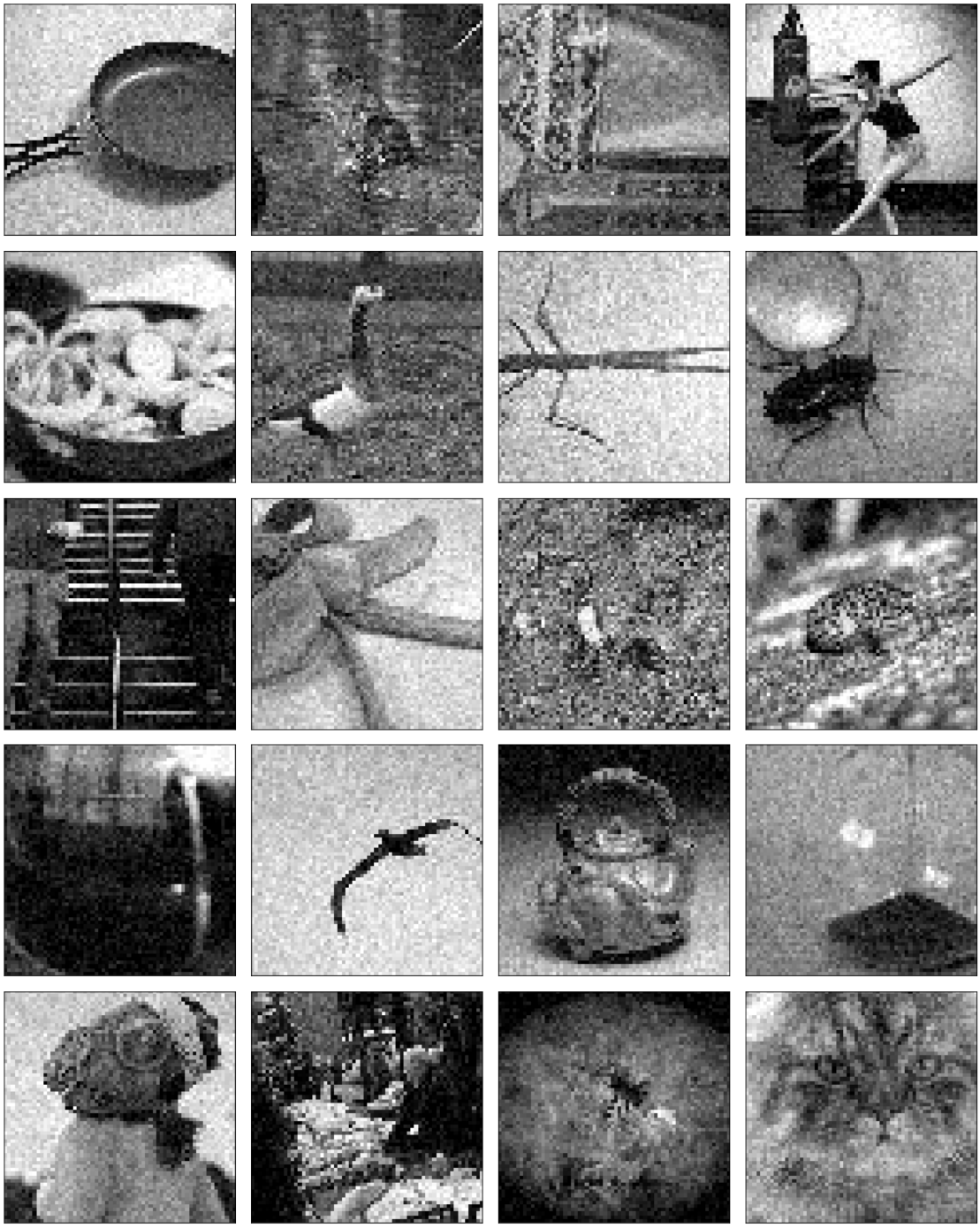
[ ]:
```
def function_result_18():

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

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

---

4 RESULTS

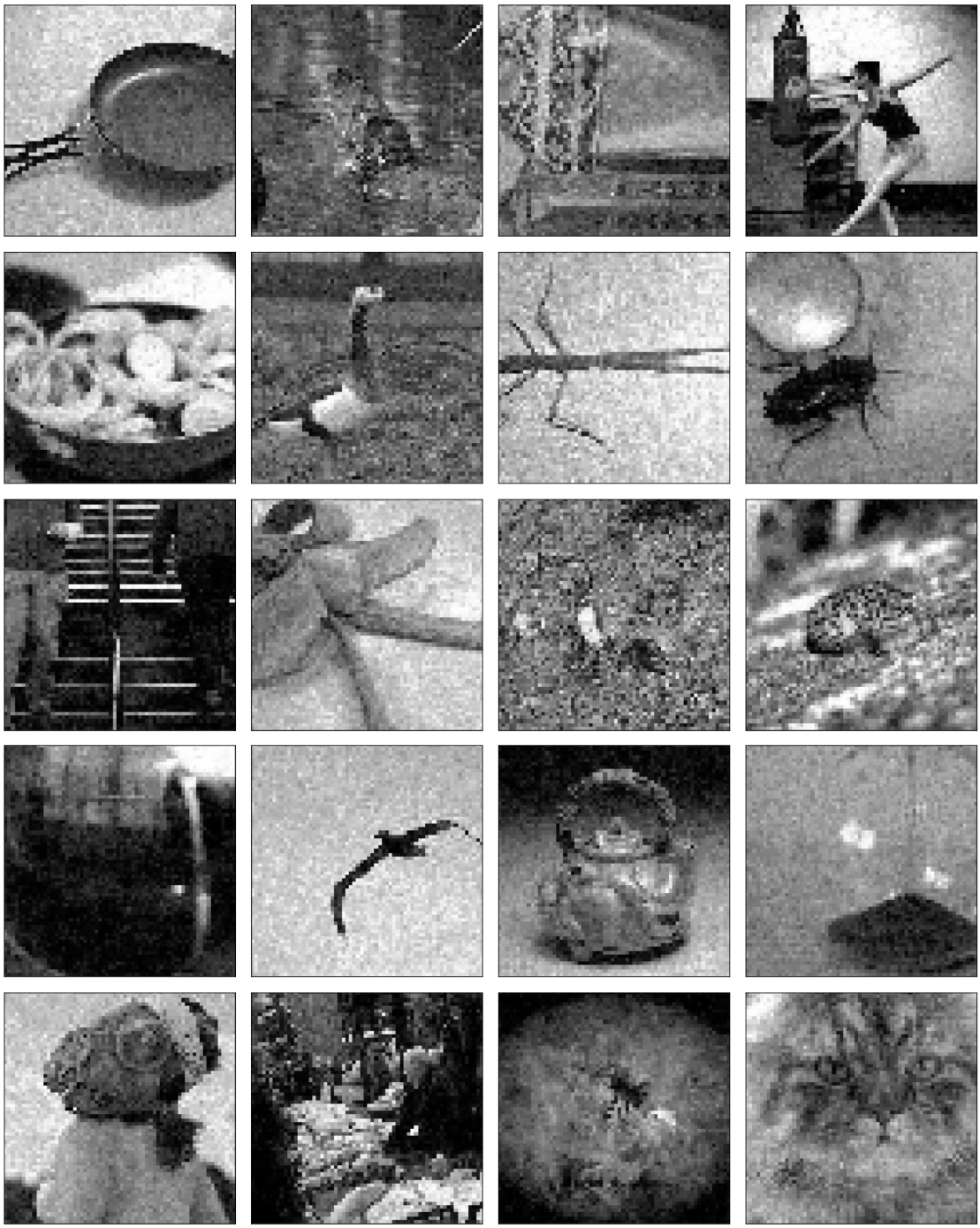
[ ]:
```
number_result = 18

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 images]
```



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

[plot examples of the training denoising results]



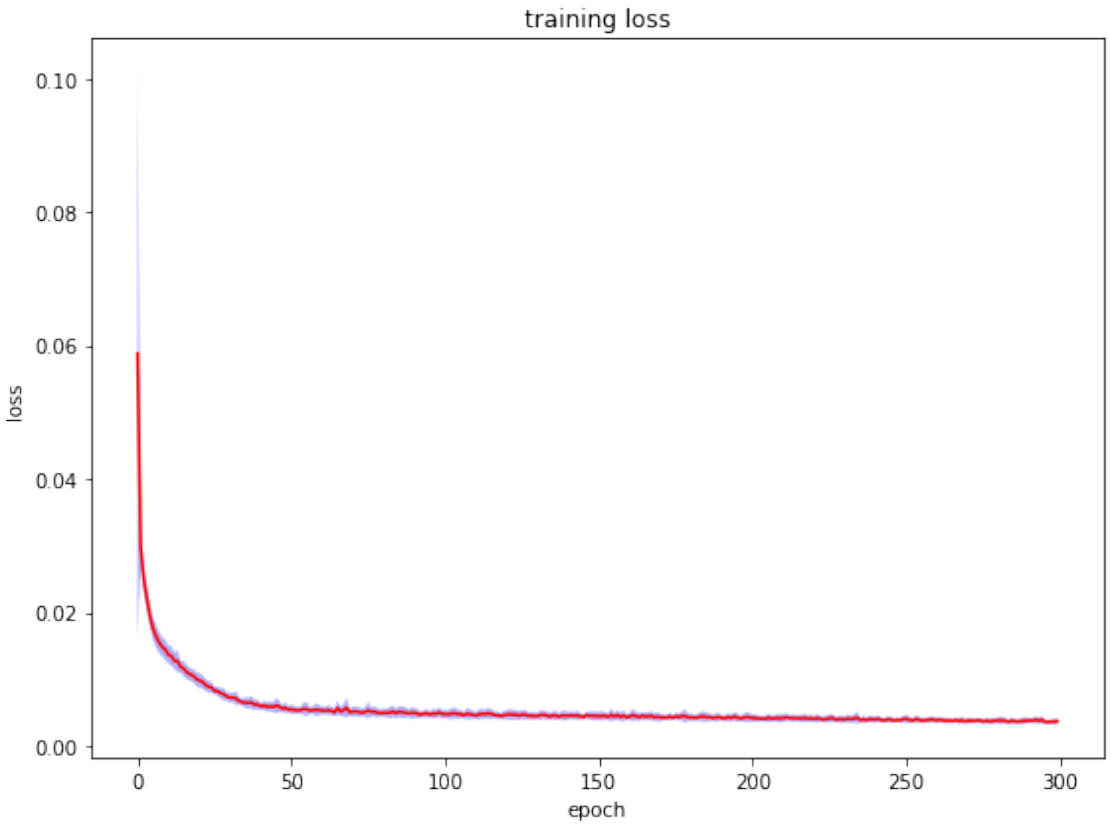
##

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

[plot examples of the testing images]
```

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

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

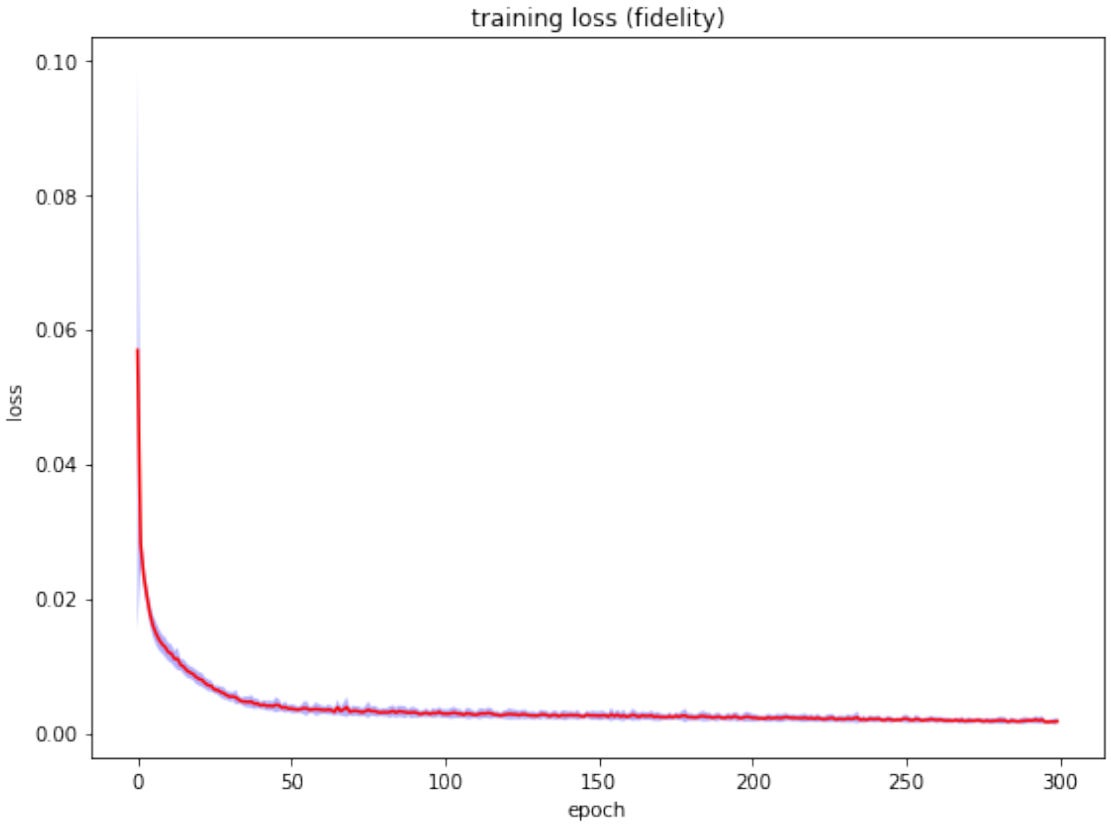
```
################################################################################
#
# RESULT # 05
#
################################################################################
```

[plot the training loss]

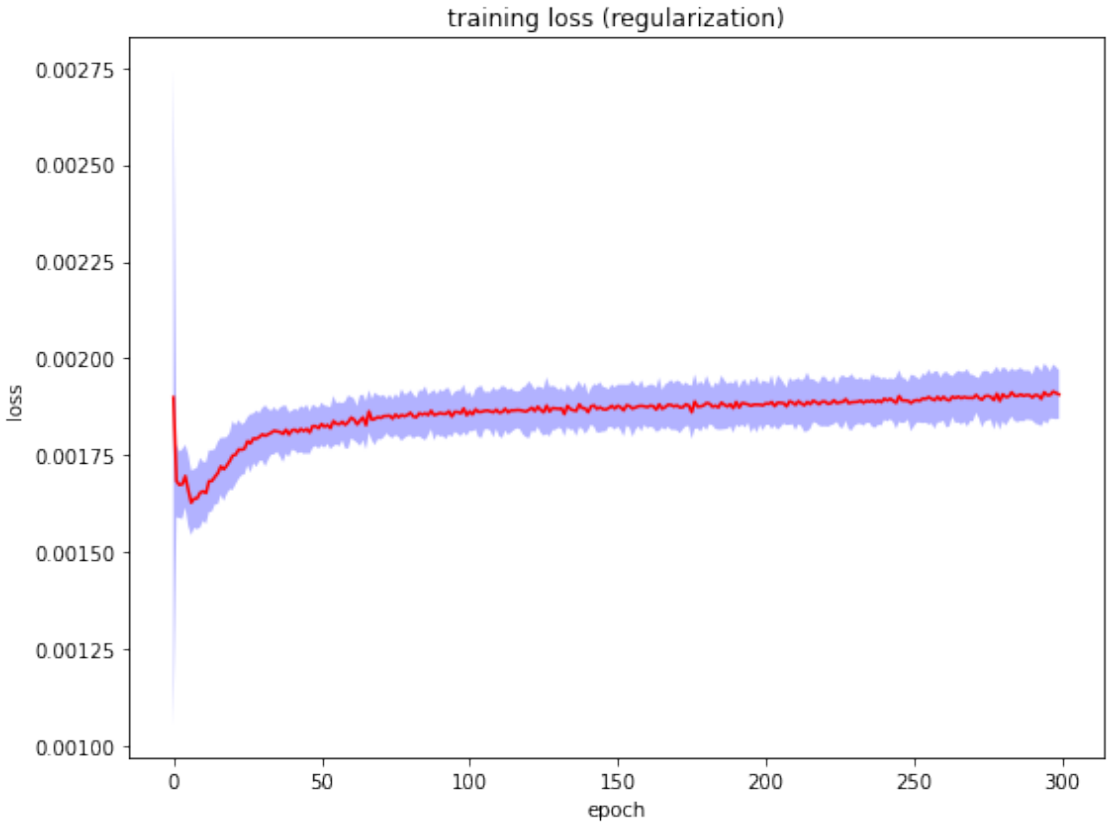



```
##############################################################################
#
# RESULT # 06
#
##############################################################################
```

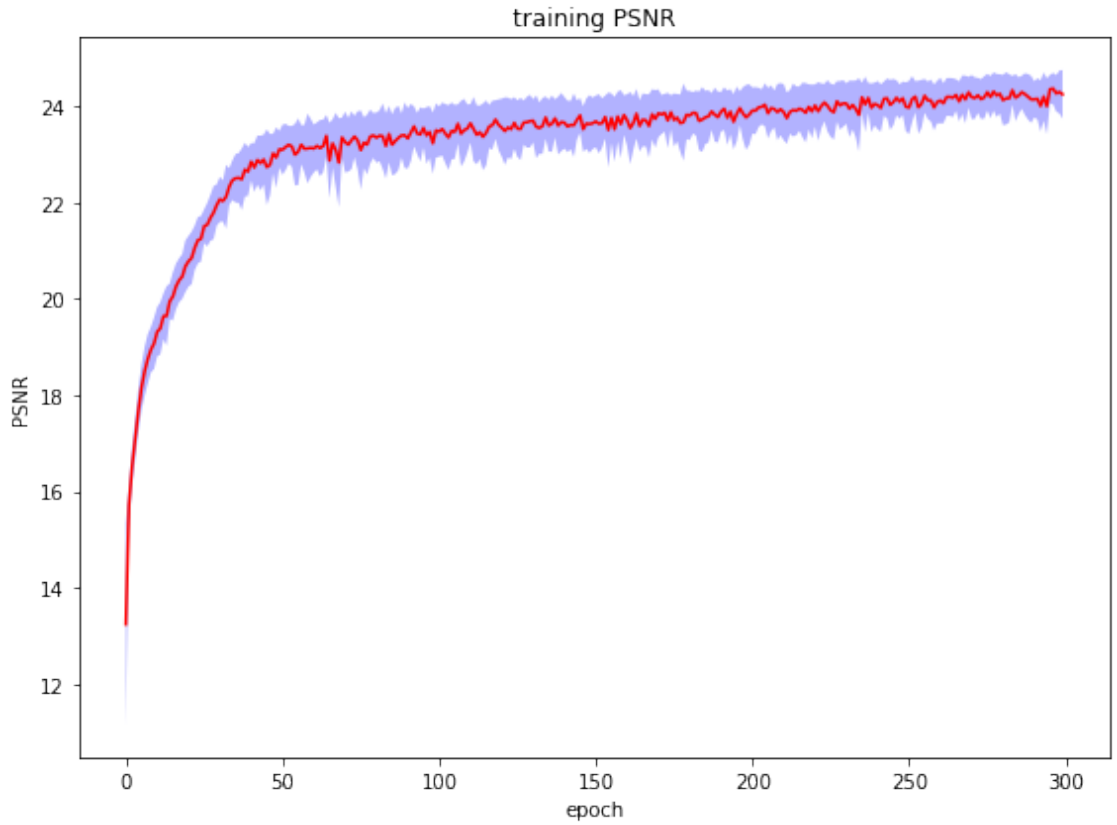
[plot the training fidelity loss]



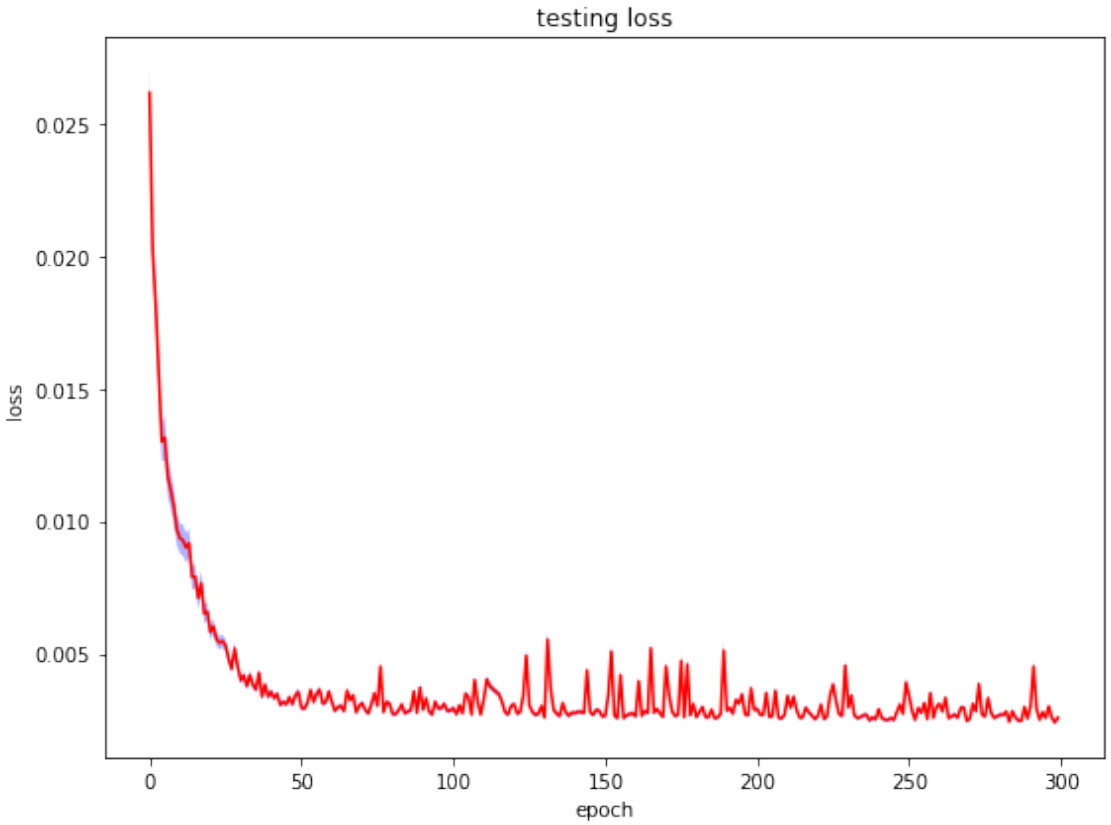

```
################################################################################
#
# RESULT # 07
#
################################################################################

[plot the training regularization loss]
```



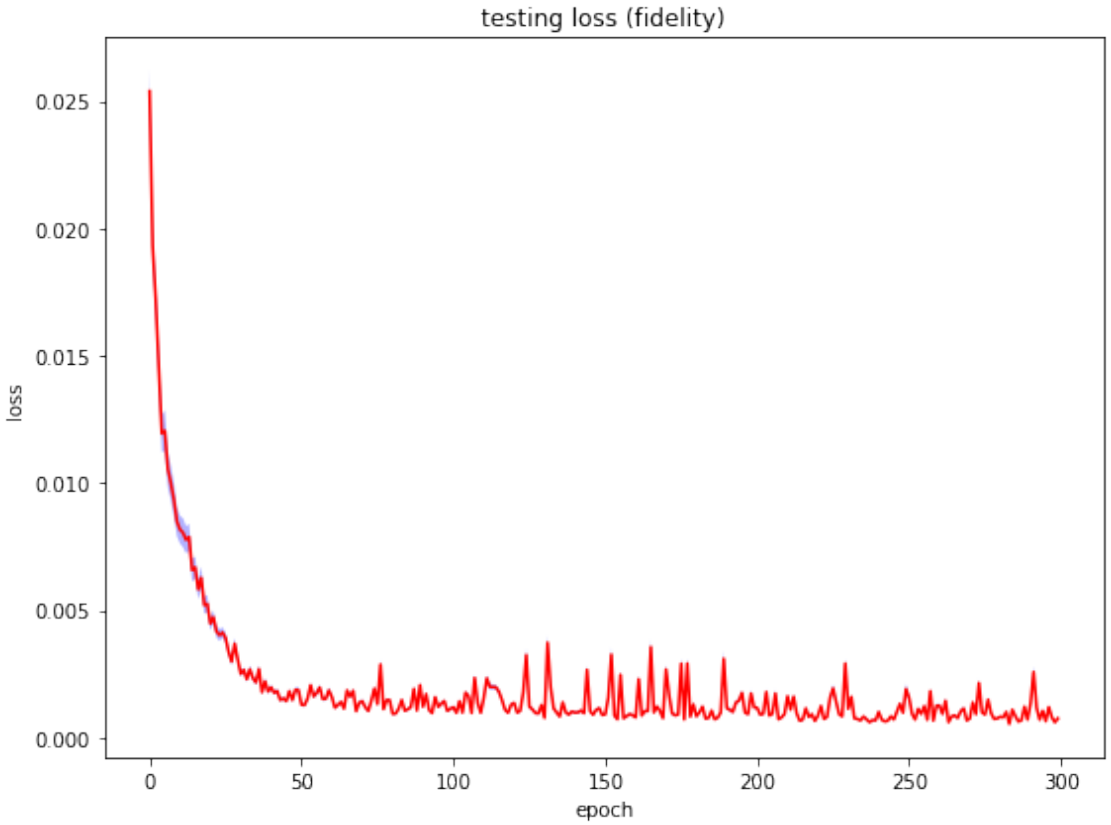

```
################################################################################
#
# RESULT # 08
#
################################################################################

[plot the training PSNR]
```



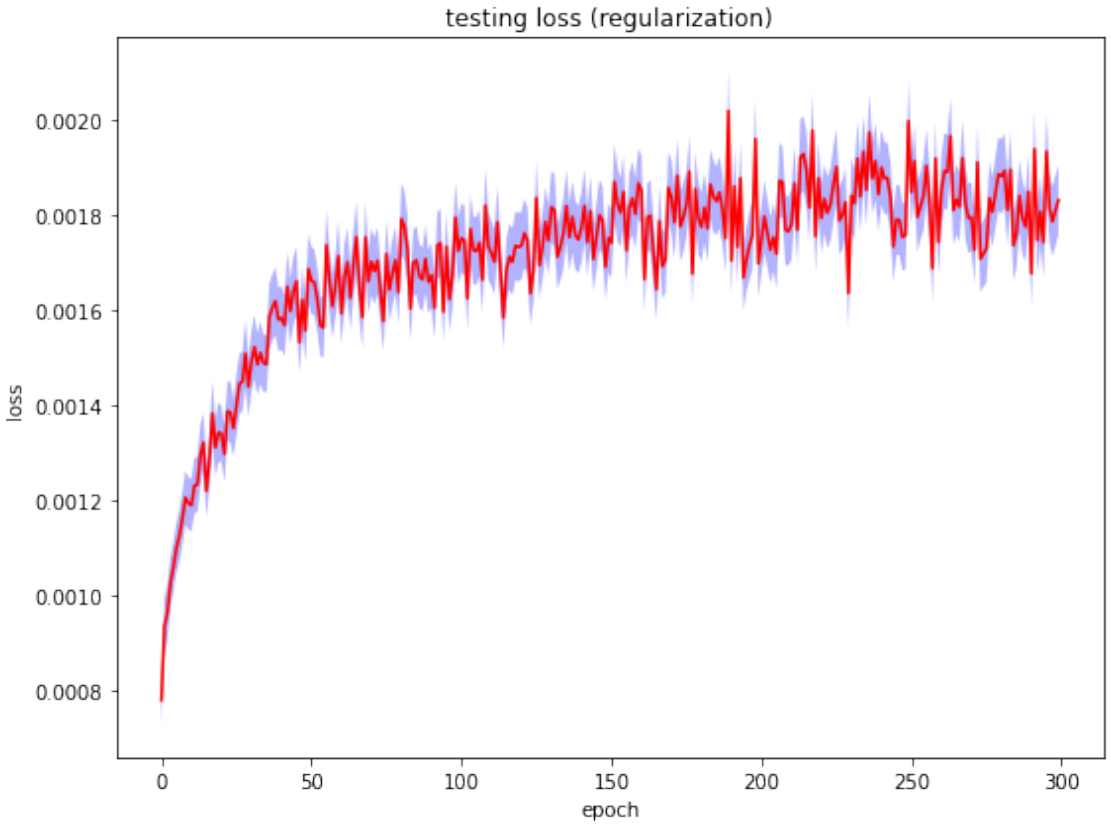

```
################################################################################
#
# RESULT # 09
#
################################################################################

[plot the testing loss]
```



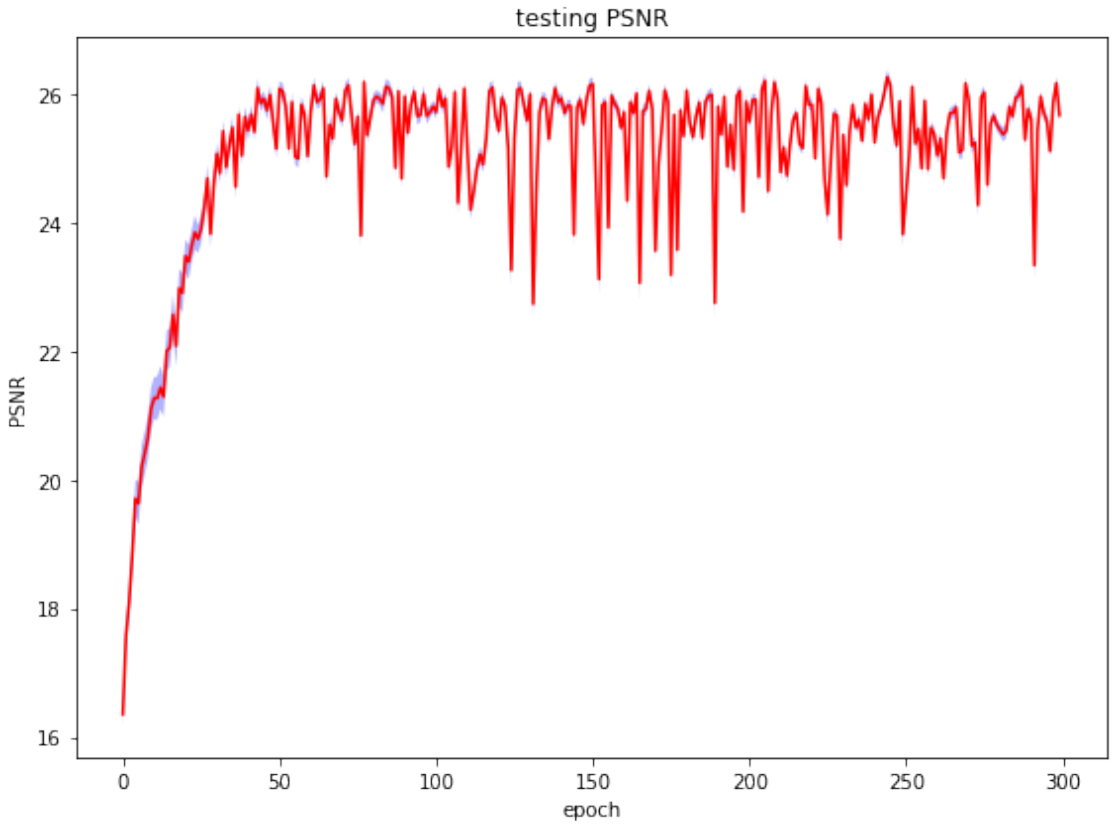

```
##############################################################################
#
# RESULT # 10
#
##############################################################################

[plot the testing fidelity loss]
```




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

[plot the testing regularization loss]
```




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

[plot the testing PSNR]
```




```
################################################################################
#
# RESULT # 13
#
################################################################################

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

index =  0, value = 0.0038002328
index =  1, value = 0.0037923141
index =  2, value = 0.0039318503
index =  3, value = 0.0037820195
index =  4, value = 0.0039264526
index =  5, value = 0.0036128355
index =  6, value = 0.0035836674
index =  7, value = 0.0036550146
index =  8, value = 0.0036404294
index =  9, value = 0.0036970274

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

```
#
# RESULT # 14
#
################################################################################

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

index =  0, value = 24.1475189535
index =  1, value = 24.1584881734
index =  2, value = 24.0106738102
index =  3, value = 24.2053302647
index =  4, value = 24.0047585863
index =  5, value = 24.3562387749
index =  6, value = 24.3720607861
index =  7, value = 24.2713487832
index =  8, value = 24.3005282829
index =  9, value = 24.2524470179

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

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

index =  0, value = 0.0030130520
index =  1, value = 0.0045153473
index =  2, value = 0.0029077708
index =  3, value = 0.0025105066
index =  4, value = 0.0027931629
index =  5, value = 0.0025774752
index =  6, value = 0.0030103918
index =  7, value = 0.0025834616
index =  8, value = 0.0024095097
index =  9, value = 0.0025826706

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

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

index =  0, value = 25.6085163090
index =  1, value = 23.3396811208
index =  2, value = 25.4476725142
```

```
index =  3, value = 25.9647118364
index =  4, value = 25.7093559312
index =  5, value = 25.5817037987
index =  6, value = 25.1144981877
index =  7, value = 25.8767380889
index =  8, value = 26.1725474610
index =  9, value = 25.6726276415

################################################################################
#
# RESULT # 17
#
################################################################################

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

best training PSNR = 24.3720607861

################################################################################
#
# RESULT # 18
#
################################################################################

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

best testing PSNR = 26.1725474610
```

[ ]: