Supervised image denoising

Import libraries

In [1]:

```
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
```

Load data

In [2]:

```
directory_data  = './'
filename_data   = 'assignment_07_data.npz'
data            = np.load(os.path.join(directory_data, filename_data))

train       = data['x_train']
train_noise = data['x_train_noise']

test        = data['x_test']
test_noise  = data['x_test_noise']

num_train   = train.shape[0]
num_test    = test.shape[0]
```

In [3]:

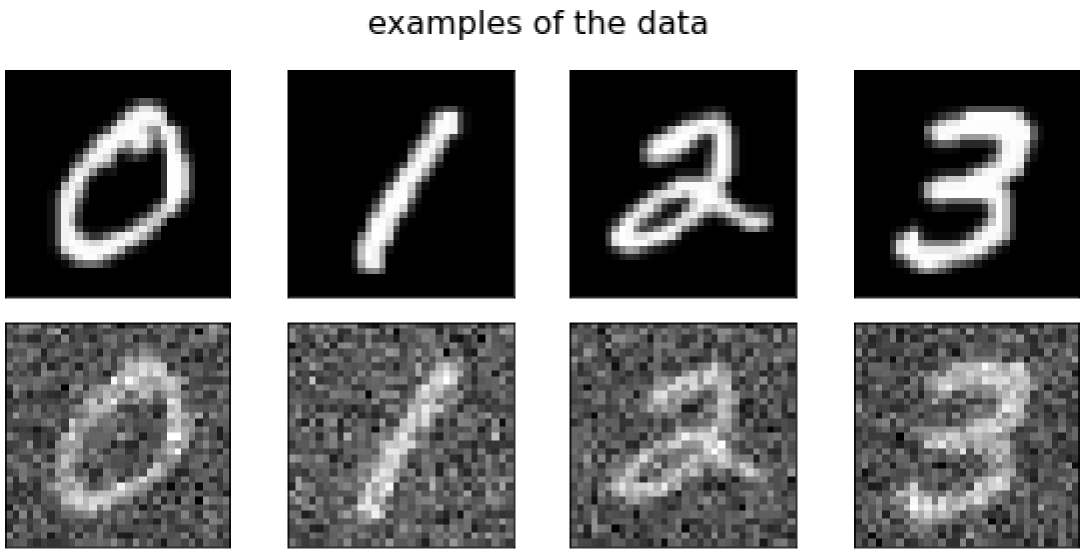
```
print('*************************************************')
print('size of train :', train.shape)
print('size of train_noise :', train_noise.shape)
print('*************************************************')
print('size of test :', test.shape)
print('size of test_noise :', test_noise.shape)
print('*************************************************')
```

```
*************************************************
size of train : (10000, 32, 32)
size of train_noise : (10000, 32, 32)
*************************************************
size of test : (5000, 32, 32)
size of test_noise : (5000, 32, 32)
*************************************************
```

plot examples of the data

In [4]:

```
nRow = 2
nCol = 4
size = 2

title       = 'examples of the data'
fig, axes   = plt.subplots(nRow, nCol, figsize=(size * nCol, size * nRow))
fig.suptitle(title, fontsize=16)

for c in range(nCol):

    axes[0, c].imshow(train[c * 1000], cmap='gray')
    axes[1, c].imshow(train_noise[c * 1000], cmap='gray')

    axes[0, c].xaxis.set_visible(False)
    axes[1, c].xaxis.set_visible(False)

    axes[0, c].yaxis.set_visible(False)
    axes[1, c].yaxis.set_visible(False)

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



custom data loader for the PyTorch framework

In [5]:

```
class dataset(Dataset):

    def __init__(self, image, image_noise):

        self.image          = image
        self.image_noise    = image_noise

    def __getitem__(self, index):

        image       = self.image[index]
        image_noise = self.image_noise[index]

        image       = torch.FloatTensor(image).unsqueeze(dim=0)
        image_noise = torch.FloatTensor(image_noise).unsqueeze(dim=0)

        return (image, image_noise)

    def __len__(self):

        number_image = self.image.shape[0]

        return number_image
```

setting device

In [6]:

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

In [7]:

```
print(device)
```

```
cuda
```

In [8]:

```
# random seed
import random
random.seed(20184757)
np.random.seed(20184757)
torch.manual_seed(20184757)
torch.cuda.manual_seed(20184757)
torch.cuda.manual_seed_all(20184757)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

construct datasets and dataloaders for training and testing

In [9]:

```
# ==================================================
# determine your own parameter value
#
size_minibatch      = 32
#
# ==================================================

dataset_train       = dataset(train, train_noise)
dataset_test        = dataset(test, test_noise)

dataloader_train    = torch.utils.data.DataLoader(dataset_train, batch_size=size_minibatch, shuffle=True, drop_last=True)
dataloader_test     = torch.utils.data.DataLoader(dataset_test, batch_size=size_minibatch, shuffle=True, drop_last=True)
```

shape of the data when using the data loader

In [29]:

```
train_image, train_image_noise = next(iter(dataloader_train))
test_image, test_image_noise = next(iter(dataloader_test))
```

In [30]:

```
print('************************************************************')
print('size of mini-batch of the training image:', train_image.shape)
print('************************************************************')
print('size of mini-batch of the training image_noise:', train_image_noise.shape)
print('************************************************************')
print('size of mini-batch of the testing image:', test_image.shape)
print('************************************************************')
print('size of mini-batch of the testing image_noise:', test_image_noise.shape)
print('************************************************************')
```

```
************************************************************
size of mini-batch of the training image: torch.Size([32, 1, 32, 32])
************************************************************
size of mini-batch of the training image_noise: torch.Size([32, 1, 32, 32])
************************************************************
size of mini-batch of the testing image: torch.Size([32, 1, 32, 32])
************************************************************
size of mini-batch of the testing image_noise: torch.Size([32, 1, 32, 32])
************************************************************
```

construct a neural network

In [10]:

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

        self.encoder_layer1 = nn.Sequential(
                        nn.Conv2d(in_channels=1, out_channels=64, kernel_size=3,
stride=2, padding=1, bias=True),
                        nn.ReLU(),
                        nn.BatchNorm2d(64),
        )
        self.encoder_layer2 = nn.Sequential(
                        nn.Conv2d(in_channels=64, out_channels=128, kernel_size=
3, stride=2, padding=1, bias=True),
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
        )
        self.encoder_layer3 = nn.Sequential(
                        nn.Conv2d(in_channels=128, out_channels=256, kernel_size
=3, stride=2, padding=1, bias=True),
                        nn.ReLU(),
                        nn.BatchNorm2d(256),
        )

        self.decoder_layer3 = nn.Sequential(
                        nn.ConvTranspose2d(in_channels=256, out_channels=128, ke
rnel_size=3, stride=2, padding=1, bias=True, output_padding=1),
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
        )
        self.decoder_layer2 = nn.Sequential(
                        nn.ConvTranspose2d(in_channels=128, out_channels=64, ker
nel_size=3, stride=2, padding=1, bias=True, output_padding=1),
                        nn.ReLU(),
                        nn.BatchNorm2d(64),
        )
        self.decoder_layer1 = nn.Sequential(
                        nn.ConvTranspose2d(in_channels=64, out_channels=32, kern
el_size=3, stride=2, padding=1, bias=True, output_padding=1),
                        nn.ReLU(),
                        nn.BatchNorm2d(32),
                        nn.Conv2d(in_channels=32, out_channels=1, kernel_size=3,
stride=1, padding=1, bias=True),
                        nn.Sigmoid(),
        )

        self.network = nn.Sequential(
            self.encoder_layer1,
            self.encoder_layer2,
            self.encoder_layer3,
            self.decoder_layer3,
            self.decoder_layer2,
            self.decoder_layer1,
        )

        self.initialize_weight()

    def forward(self,x):
```

```
        out = self.network(x)
        return out

    def initialize_weight(self):
        for m in self.network.modules():
            if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):
                nn.init.xavier_normal_(m.weight)
                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_normal_(m.weight)
                if m.bias is not None:
                    nn.init.constant_(m.bias, 1)
                    pass
```

build the network

In [11]:

```
# ==================================================
# determine your own parameter value
#
learning_rate   = 0.3
weight_decay    = 0.0001
number_epoch    = 344
#
# ==================================================

model       = Network().to(device)
optimizer   = torch.optim.SGD(model.parameters(), lr=learning_rate, weight_decay
=weight_decay)
```

compute the prediction

In [12]:

```
def compute_prediction(model, input):

    prediction = model(input)

    return prediction
```

compute the loss

- use the mean squared error for the loss
- use the following loss function `nn.MSELoss()`
- input dimension: mini-batch-size x channel x height x width

In [13]:

```
def compute_loss(prediction, original):
    # ==================================================
    # fill up the blank
    #
    loss = nn.MSELoss()(prediction, original)
    #
    # ==================================================

    return loss
```

compute the loss value

In [14]:

```
def compute_loss_value(loss):

    loss_value = loss.item()

    return loss_value
```

compute the PSNR metric

- input dimension: mini-batch-size x channel x height x width
- psnr is defined by $10 * \log_{10}(\frac{1}{\text{MSE}})$
- MSE is defined by $\text{MSE}(x, y) = \frac{1}{n} \|x - y\|_2^2$ where n is the length of data x and y

In [15]:

```
def compute_psnr(data1, data2):

    mse         = nn.MSELoss()(data1, data2)
    mse_value   = mse.item()
    psnr        = 10 * np.log10(1 / mse_value)

    return psnr
```

Variable for the learning curves

In [16]:

```
loss_train_mean = np.zeros(number_epoch)
loss_train_std  = np.zeros(number_epoch)
psnr_train_mean = np.zeros(number_epoch)
psnr_train_std  = np.zeros(number_epoch)

loss_test_mean  = np.zeros(number_epoch)
loss_test_std   = np.zeros(number_epoch)
psnr_test_mean  = np.zeros(number_epoch)
psnr_test_std   = np.zeros(number_epoch)
```

train

In [17]:

```
def train(model, optimizer, dataloader):

    loss_epoch      = []
    psnr_epoch      = []

    model.train()

    for index_batch, (image, image_noise) in enumerate(dataloader):

        image       = image.to(device)
        image_noise = image_noise.to(device)

        # ==================================================
        # fill up the blank
        #
        prediction  = compute_prediction(model, image_noise)
        loss        = compute_loss(prediction, image)
        loss_value  = compute_loss_value(loss)
        psnr        = compute_psnr(prediction, image)
        #
        # ==================================================

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

        # ==================================================
        # fill up the blank (update moodel parameters)
        #
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        #
        # ==================================================

    loss_mean   = np.mean(loss_epoch)
    loss_std    = np.std(loss_epoch)

    psnr_mean   = np.mean(psnr_epoch)
    psnr_std    = np.std(psnr_epoch)

    return (loss_mean, loss_std, psnr_mean, psnr_std)
```

test

In [18]:

```
def test(model, dataloader):

    loss_epoch      = []
    psnr_epoch      = []

    model.eval()

    for index_batch, (image, image_noise) in enumerate(dataloader):

        image       = image.to(device)
        image_noise = image_noise.to(device)

        # ==================================================
        # fill up the blank
        #
        prediction  = compute_prediction(model, image_noise)
        loss        = compute_loss(prediction, image)
        loss_value  = compute_loss_value(loss)
        psnr        = compute_psnr(prediction, image)
        #
        # ==================================================

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

    loss_mean   = np.mean(loss_epoch)
    loss_std    = np.std(loss_epoch)

    psnr_mean   = np.mean(psnr_epoch)
    psnr_std    = np.std(psnr_epoch)

    return (loss_mean, loss_std, psnr_mean, psnr_std)
```

train and test

In [19]:

```
# ==============================================================================
==
#
# iterations for epochs
#
# ==============================================================================
==
for i in tqdm(range(number_epoch)):

    # ==========================================================================
======
    #
    # training
    #
    # ==========================================================================
======
    (loss_train_mean_epoch, loss_train_std_epoch, psnr_train_mean_epoch, psnr_tr
ain_std_epoch) = train(model, optimizer, dataloader_train)

    loss_train_mean[i]  = loss_train_mean_epoch
    loss_train_std[i]   = loss_train_std_epoch

    psnr_train_mean[i]  = psnr_train_mean_epoch
    psnr_train_std[i]   = psnr_train_std_epoch

    # ==========================================================================
======
    #
    # testing
    #
    # ==========================================================================
======
    (loss_test_mean_epoch, loss_test_std_epoch, psnr_test_mean_epoch, psnr_test_
std_epoch) = test(model, dataloader_test)

    loss_test_mean[i]   = loss_test_mean_epoch
    loss_test_std[i]    = loss_test_std_epoch

    psnr_test_mean[i]   = psnr_test_mean_epoch
    psnr_test_std[i]    = psnr_test_std_epoch
```

functions for presenting the results

In [20]:

```
def function_result_01():

    title           = 'loss (training)'
    label_axis_x    = 'epoch'
    label_axis_y    = 'loss'
    color_mean      = 'red'
    color_std       = 'blue'
    alpha           = 0.3

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

    plt.plot(range(len(loss_train_mean)), loss_train_mean, '-', color = color_me
an)
    plt.fill_between(range(len(loss_train_mean)), loss_train_mean - loss_train_s
td, loss_train_mean + loss_train_std, facecolor = color_std, alpha = alpha)

    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

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

In [21]:

```
def function_result_02():

    title           = 'loss (testing)'
    label_axis_x    = 'epoch'
    label_axis_y    = 'loss'
    color_mean      = 'red'
    color_std       = 'blue'
    alpha           = 0.3

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

    plt.plot(range(len(loss_test_mean)), loss_test_mean, '-', color = color_mean
)
    plt.fill_between(range(len(loss_test_mean)), loss_test_mean - loss_test_std,
loss_test_mean + loss_test_std, facecolor = color_std, alpha = alpha)

    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

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

In [22]:

```
def function_result_03():

    title           = 'psnr (training)'
    label_axis_x    = 'epoch'
    label_axis_y    = 'psnr'
    color_mean      = 'red'
    color_std       = 'blue'
    alpha           = 0.3

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

    plt.plot(range(len(psnr_train_mean)), psnr_train_mean, '-', color = color_me
an)
    plt.fill_between(range(len(psnr_train_mean)), psnr_train_mean - psnr_train_s
td, psnr_train_mean + psnr_train_std, facecolor = color_std, alpha = alpha)

    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

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

In [23]:

```
def function_result_04():

    title           = 'psnr (testing)'
    label_axis_x    = 'epoch'
    label_axis_y    = 'psnr'
    color_mean      = 'red'
    color_std       = 'blue'
    alpha           = 0.3

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

    plt.plot(range(len(psnr_test_mean)), psnr_test_mean, '-', color = color_mean
)
    plt.fill_between(range(len(psnr_test_mean)), psnr_test_mean - psnr_test_std,
psnr_test_mean + psnr_test_std, facecolor = color_std, alpha = alpha)

    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

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

In [24]:

```
def function_result_05():

    nRow = 9
    nCol = 3
    size = 3

    title = 'training results'
    fig, axes = plt.subplots(nRow, nCol, figsize=(size * nCol, size * nRow))
    fig.suptitle(title, fontsize=16)

    index_image = np.array([0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000])

    image           = torch.FloatTensor(dataset_train.image[index_image]).unsque
eze(dim=1).to(device)
    image_noise     = torch.FloatTensor(dataset_train.image_noise[index_image]).
unsqueeze(dim=1).to(device)
    image_denoise   = compute_prediction(model, image_noise)

    image           = image.detach().cpu().squeeze(axis=1)
    image_noise     = image_noise.detach().cpu().squeeze(axis=1)
    image_denoise   = image_denoise.detach().cpu().squeeze(axis=1)

    nStep = 3

    for r in range(3):
        for c in range(nCol):

            axes[0 + r * nStep, c].imshow(image_noise[c + r * nStep], cmap='gra
y')
            axes[1 + r * nStep, c].imshow(image[c + r * nStep], cmap='gray', vmi
n=0, vmax=1)
            axes[2 + r * nStep, c].imshow(image_denoise[c + r * nStep], cmap='gr
ay', vmin=0, vmax=1)

            axes[0 + r * nStep, c].xaxis.set_visible(False)
            axes[1 + r * nStep, c].xaxis.set_visible(False)
            axes[2 + r * nStep, c].xaxis.set_visible(False)

            axes[0 + r * nStep, c].yaxis.set_visible(False)
            axes[1 + r * nStep, c].yaxis.set_visible(False)
            axes[2 + r * nStep, c].yaxis.set_visible(False)

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

In [25]:

```
def function_result_06():

    nRow = 9
    nCol = 3
    size = 3

    title = 'testing results'
    fig, axes = plt.subplots(nRow, nCol, figsize=(size * nCol, size * nRow))
    fig.suptitle(title, fontsize=16)

    index_image     = np.array([0, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000
])

    image           = torch.FloatTensor(dataset_test.image[index_image]).unsquee
ze(dim=1).to(device)
    image_noise     = torch.FloatTensor(dataset_test.image_noise[index_image]).u
nsqueeze(dim=1).to(device)
    image_denoise   = compute_prediction(model, image_noise)

    image           = image.detach().cpu().squeeze(axis=1)
    image_noise     = image_noise.detach().cpu().squeeze(axis=1)
    image_denoise   = image_denoise.detach().cpu().squeeze(axis=1)

    nStep = 3

    for r in range(3):
        for c in range(nCol):

            axes[0 + r * nStep, c].imshow(image_noise[c + r * nStep], cmap='gra
y')
            axes[1 + r * nStep, c].imshow(image[c + r * nStep], cmap='gray', vmi
n=0, vmax=1)
            axes[2 + r * nStep, c].imshow(image_denoise[c + r * nStep], cmap='gr
ay', vmin=0, vmax=1)

            axes[0 + r * nStep, c].xaxis.set_visible(False)
            axes[1 + r * nStep, c].xaxis.set_visible(False)
            axes[2 + r * nStep, c].xaxis.set_visible(False)

            axes[0 + r * nStep, c].yaxis.set_visible(False)
            axes[1 + r * nStep, c].yaxis.set_visible(False)
            axes[2 + r * nStep, c].yaxis.set_visible(False)

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

In [26]:

```
def function_result_07():

    print('final training psnr = %9.8f' % (psnr_train_mean[-1]))
```

In [27]:

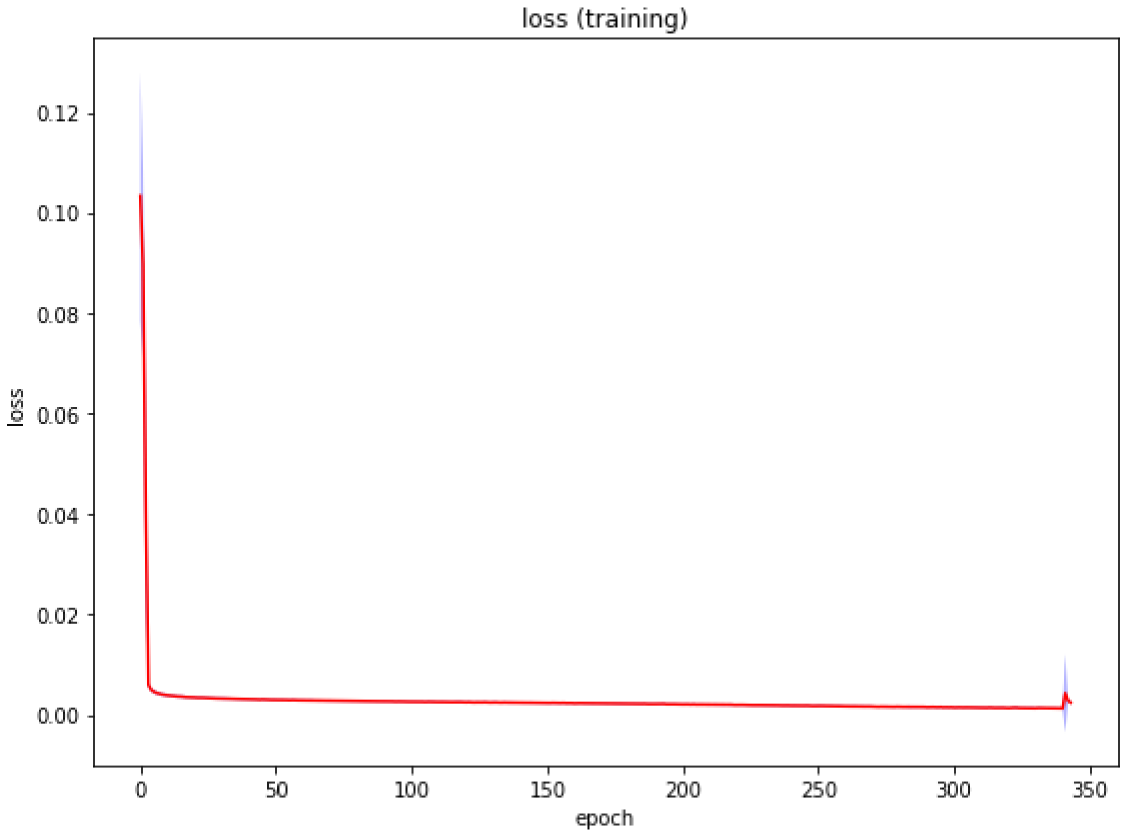
```
def function_result_08():

    print('final testing psnr = %9.8f' % (psnr_test_mean[-1]))
```

results

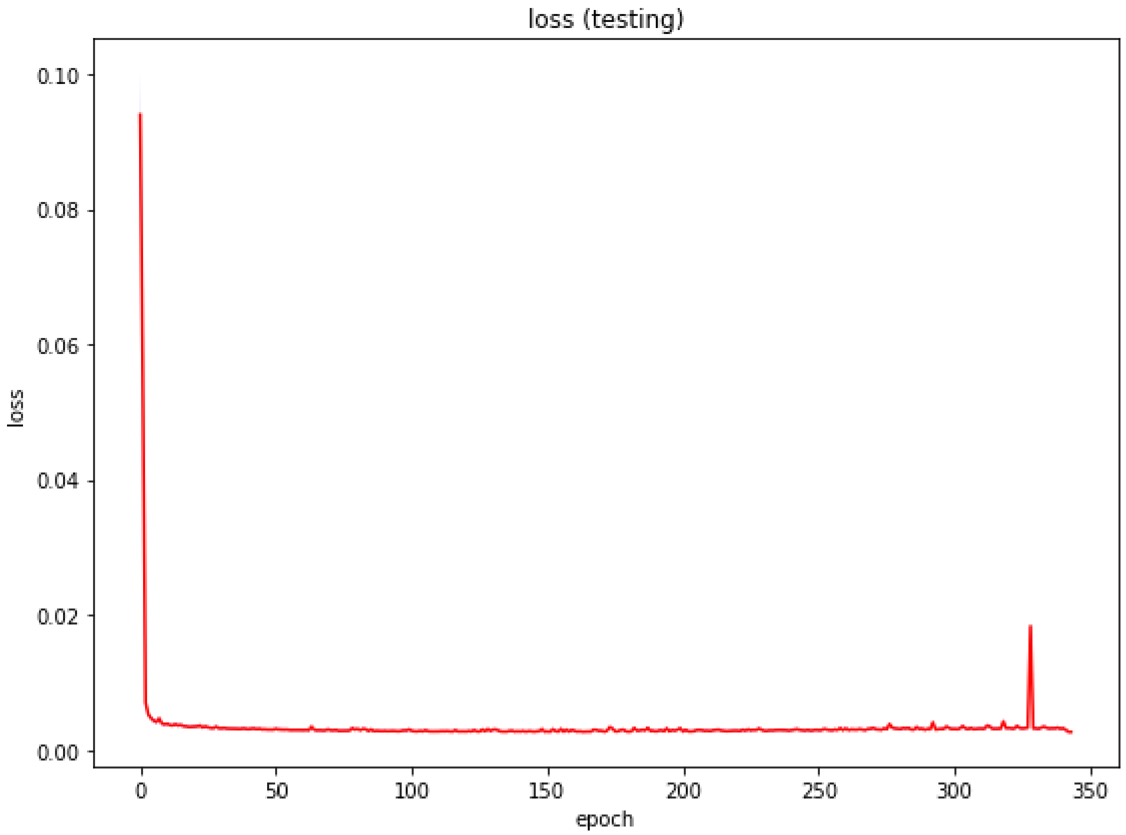
In [28]:

```
number_result = 8

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)
```

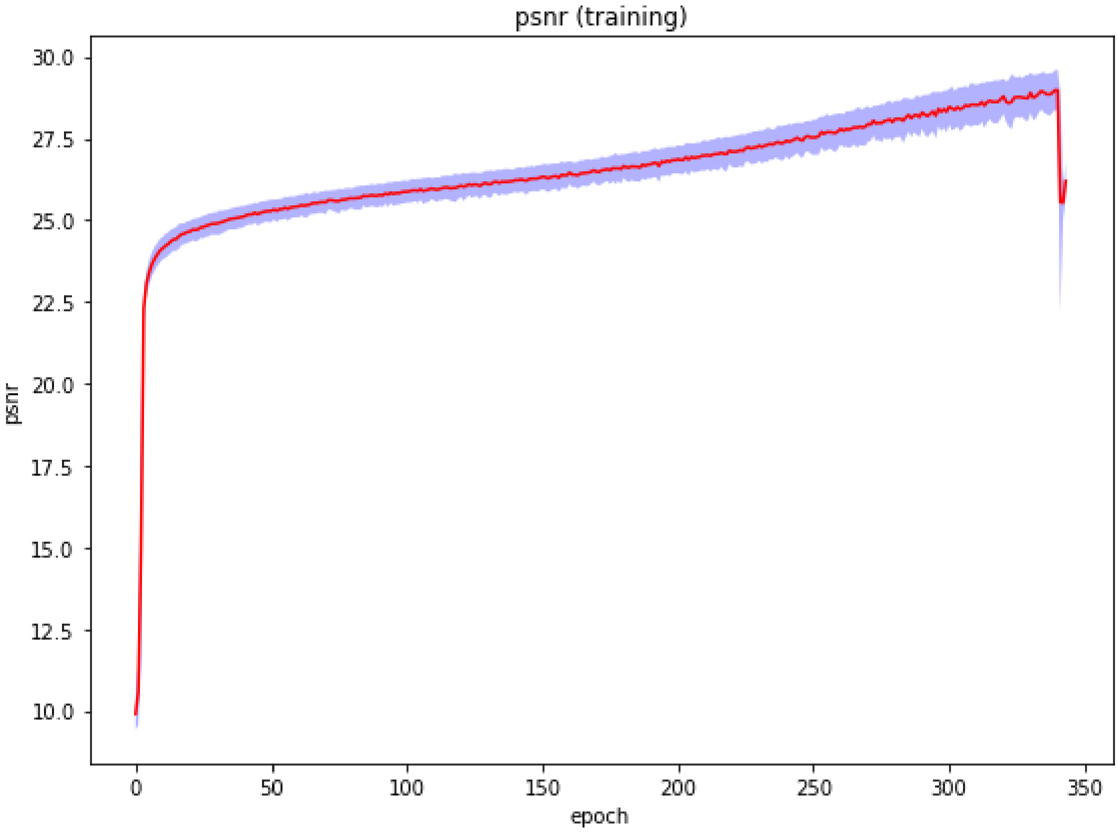
```
########################################################################
#
# RESULT # 01
#
########################################################################
```



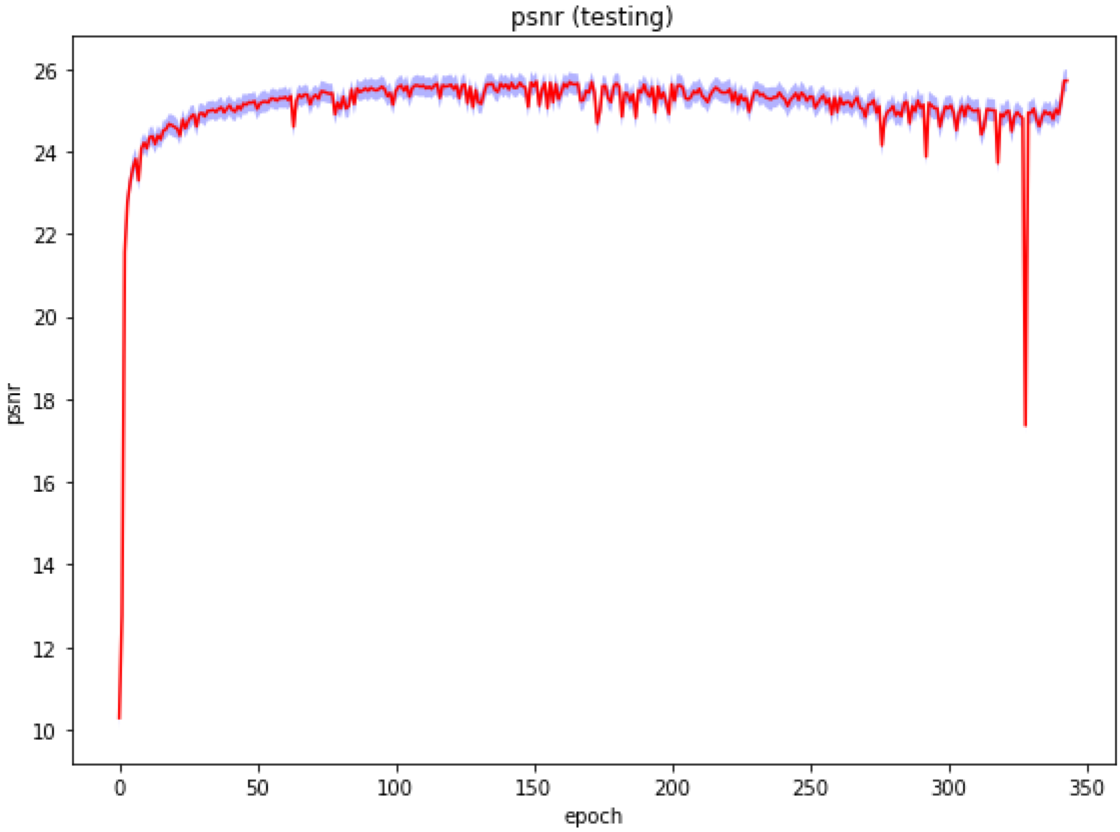

```
########################################################################
#
# RESULT # 02
#
########################################################################
```


loss (testing)
0.10
0.08
0.06
0.04
0.02
0.00
loss
0
50
100
150
200
250
300
350
epoch

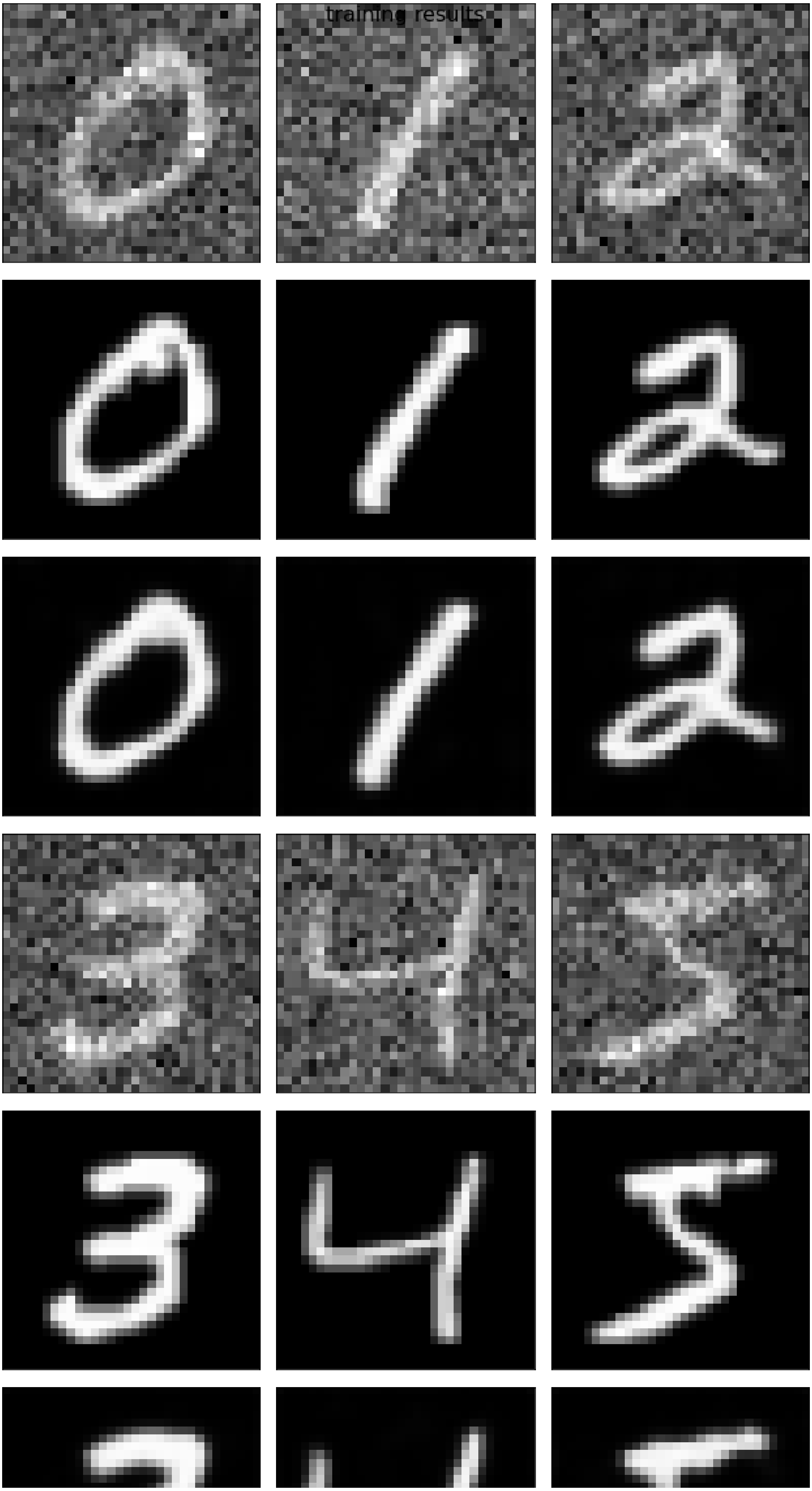
```
####################################################################
############
#
# RESULT # 03
#
####################################################################
############
```

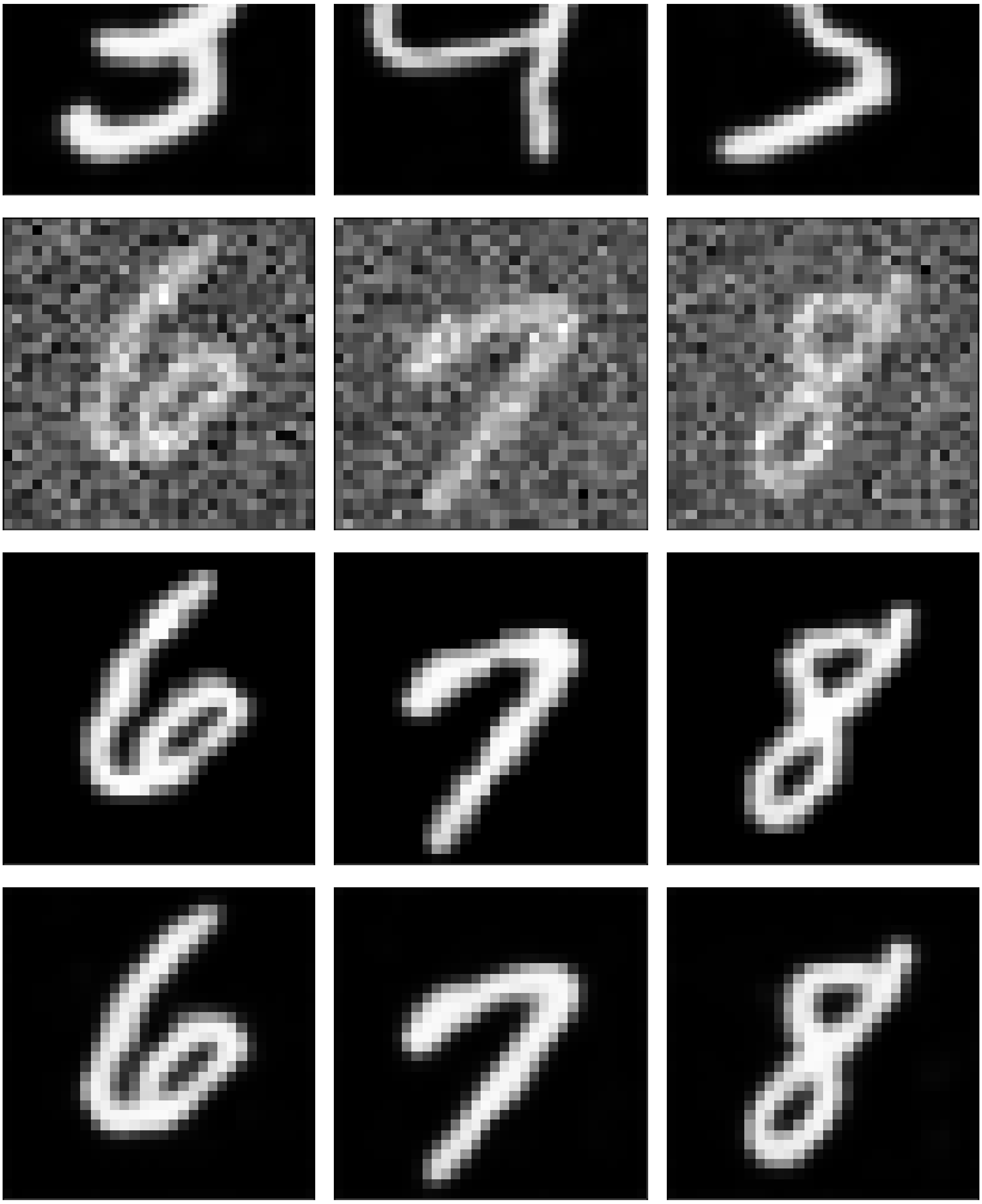

psnr (training)
psnr
epoch
30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
0
50
100
150
200
250
300
350

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

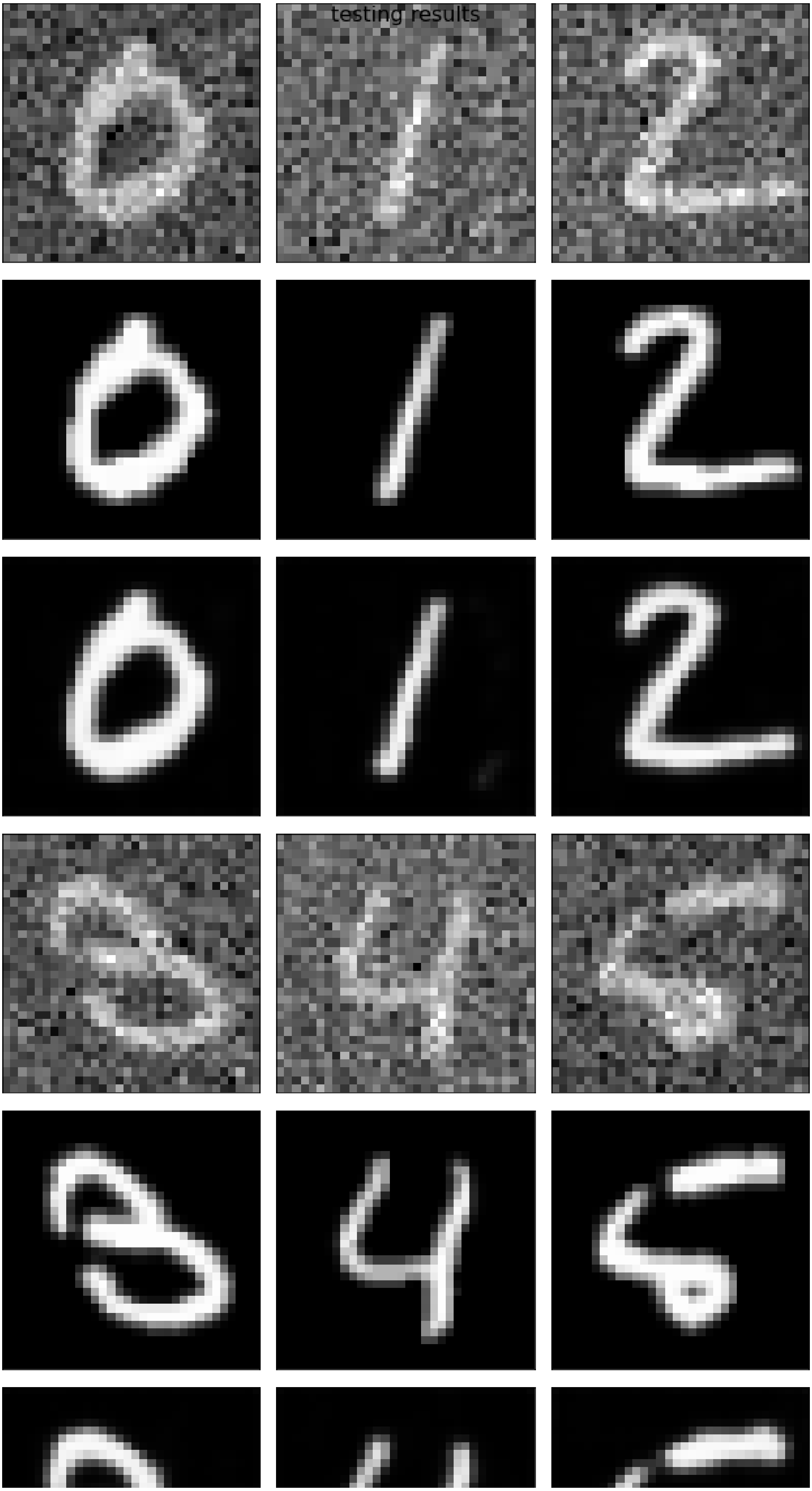

psnr (testing)
psnr
epoch
26
24
22
20
18
16
14
12
10
0
50
100
150
200
250
300
350

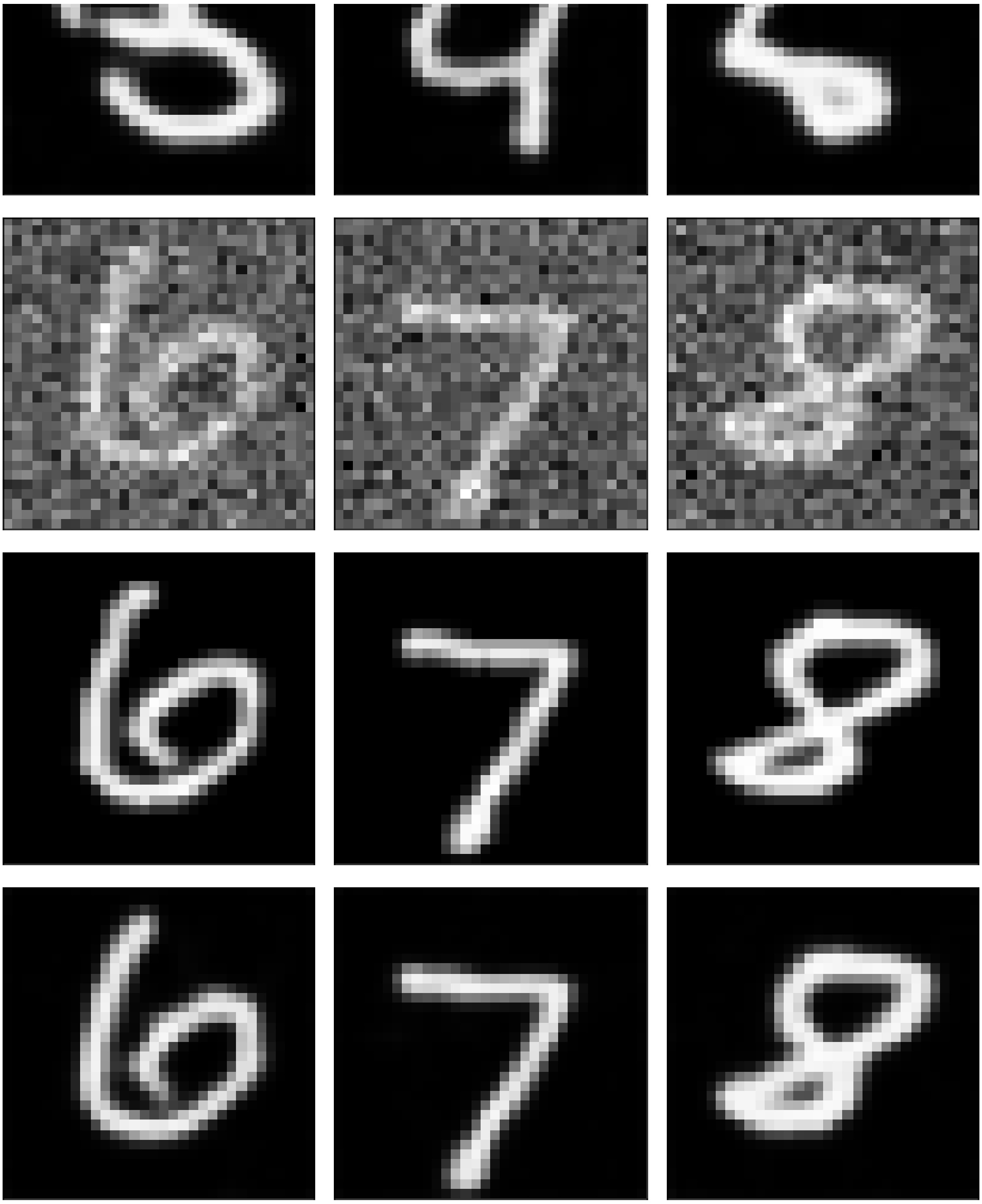
```
########################################################################
############
#
# RESULT # 05
#
########################################################################
############
```


training results



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


testing results



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

final training psnr = 26.19558784

################################################################################
#
# RESULT # 08
#
################################################################################

final testing psnr = 25.72501818
```

In [ ]: