assignment_10

November 26, 2021

1 Image Segmentation by unsupervised Learning

1.1 Connect Google Drive

[1]:

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

```
Mounted at /content/drive
```

1.2 import libraries

[2]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import numpy as np
import matplotlib.pyplot as plt
import math
from tqdm import tqdm
import random
import os
```

1.3 load data

[3]:

```
directory_data  = 'drive/MyDrive'
filename_data   = 'assignment_10_data.npz'
data            = np.load(os.path.join(directory_data, filename_data))
image_clean     = torch.from_numpy(data['real_images']).float()
```

1.4 custom data loader for the PyTorch framework

[135]:

```
class dataset (Dataset):
    def  __init__(self, data, std_noise):
        noise = torch.randn(data.size()) * std_noise

        self.clean  = data
        self.noisy  = data + noise

    def __getitem__(self, index):

        clean    = self.clean[index]
        noisy    = self.noisy[index]

        clean = torch.FloatTensor(clean).unsqueeze(dim=0)
        noisy = torch.FloatTensor(noisy).unsqueeze(dim=0)

        return (clean, noisy)

    def __len__(self):

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

[136]:

```
image_train = image_clean[::2]
image_test  = image_clean[1::2]

dataset_train    = dataset(image_train, 0.5)
dataset_test     = dataset(image_test, 0.5)
```

1.5 hyper-parameters

[373]:

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

number_epoch    = 300
size_minibatch  = 16
learning_rate   = 0.1
weight_decay    = 0.001
weight_regular  = 0.001
```

1.6 construct datasets and dataloaders for training and testing

[374]:

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

1.7 shape of the data when using the data loader

[375]:
```
(image_train, label_train)  = dataset_train[0]
(image_test, label_test)    = dataset_test[0]
print('*******************************************************************')
print('shape of the image in the training dataset:', image_train.shape)
print('shape of the label in the training dataset:', label_train.shape)
print('*******************************************************************')
print('shape of the image in the testing dataset:', image_test.shape)
print('shape of the label in the testing dataset:', label_test.shape)
print('*******************************************************************')
```

```
*******************************************************************
shape of the image in the training dataset: torch.Size([1, 32, 32])
shape of the label in the training dataset: torch.Size([1, 32, 32])
*******************************************************************
shape of the image in the testing dataset: torch.Size([1, 32, 32])
shape of the label in the testing dataset: torch.Size([1, 32, 32])
*******************************************************************
```

1.8 class for the neural network

[376]:
```
class Network(nn.Module):

        def __init__(self, in_channel=1, out_channel=1, dim_feature=8):

                super(Network, self).__init__()

                self.in_channel         = in_channel
                self.out_channel        = out_channel
                self.dim_feature        = dim_feature

                self.conv_encode1        = nn.Conv2d(in_channel , dim_feature *␣
 ↪1, kernel_size=3, stride=2, padding=1, bias=True)
                self.conv_encode2        = nn.Conv2d(dim_feature * 1,␣
 ↪dim_feature * 2, kernel_size=3, stride=2, padding=1, bias=True)
                self.conv_encode3        = nn.Conv2d(dim_feature * 2,␣
 ↪dim_feature * 4, kernel_size=3, stride=2, padding=1, bias=True)
                self.conv_middle        = nn.Conv2d(dim_feature * 4,␣
 ↪dim_feature * 8, kernel_size=3, stride=1, padding=1, bias=True)
                self.conv_decode3         = nn.Conv2d(dim_feature * 8,␣
 ↪dim_feature * 4, kernel_size=3, stride=1, padding=1, bias=True)
                self.conv_decode2         = nn.Conv2d(dim_feature * 4,␣
 ↪dim_feature * 2, kernel_size=3, stride=1, padding=1, bias=True)
                self.conv_decode1         = nn.Conv2d(dim_feature * 2,␣
 ↪dim_feature * 1, kernel_size=3, stride=1, padding=1, bias=True)
```

```
        self.conv_out                 = nn.Conv2d(dim_feature * 1,␣
↪out_channel,        kernel_size=1, stride=1, padding=0, bias=True)

        self.ebn1                        = nn.BatchNorm2d(dim_feature *␣
↪1)
        self.ebn2                        = nn.BatchNorm2d(dim_feature *␣
↪2)
        self.ebn3                        = nn.BatchNorm2d(dim_feature *␣
↪4)
        self.mbn                         = nn.BatchNorm2d(dim_feature *␣
↪8)
        self.dbn3                        = nn.BatchNorm2d(dim_feature *␣
↪4)
        self.dbn2                        = nn.BatchNorm2d(dim_feature *␣
↪2)
        self.dbn1                        = nn.BatchNorm2d(dim_feature *␣
↪1)

        self.activation                = nn.ReLU(inplace=True)
        self.activation_out        = nn.Sigmoid()

        #␣
↪*********************************************************************
        # forward propagation
        #␣
↪*********************************************************************
    def forward(self, x):

        x1  = self.conv_encode1(x)
        eb1 = self.ebn1(x1)
        e1  = self.activation(eb1)

        x2  = self.conv_encode2(e1)
        eb2 = self.ebn2(x2)
        e2  = self.activation(eb2)

        x3  = self.conv_encode3(e2)
        eb3 = self.ebn3(x3)
        e3  = self.activation(eb3)

        m   = self.conv_middle(e3)
        mb  = self.mbn(m)
        c   = self.activation(mb)

        y3  = nn.Upsample(scale_factor=2, mode='bilinear',␣
↪align_corners=False)(c)
```

```
                y3  = self.conv_decode3(y3)
                db3 = self.dbn3(y3)
                d3  = self.activation(db3)

                y2  = nn.Upsample(scale_factor=2, mode='bilinear',␣
↪align_corners=False)(d3)
                y2  = self.conv_decode2(y2)
                db2 = self.dbn2(y2)
                d2  = self.activation(db2)

                y1  = nn.Upsample(scale_factor=2, mode='bilinear',␣
↪align_corners=False)(d2)
                y1  = self.conv_decode1(y1)
                db1 = self.dbn1(y1)
                d1  = self.activation(db1)

                y1  = self.conv_out(d1)
                y = self.activation_out(y1)

                return y
```

1.9 build network

[377]:
```
model       = Network().to(device)
optimizer   = torch.optim.SGD(model.parameters(), lr=learning_rate,␣
↪weight_decay=weight_decay)
```

1.10 compute the prediction

[378]:
```
def compute_prediction(model, input):
    # ==================================================
    # fill up the blank
    #

    prediction = model(input)

    #
    # ==================================================

    return prediction
```

[379]:
```
def compute_estimate(input, prediction):

    number_phase = 2 # bi-partitioning
    (batch, channel, height, width) = input.size()

    estimate = torch.zeros(number_phase, batch, channel).to(device)

    prediction_inside   = prediction
    prediction_outside  = 1 - prediction

    # ==================================================
    # fill up the blank for the estimate of the inside of segmenting region
    #
    estimate[0] = torch.sum(input*prediction_inside,dim=(2,3)) / torch.
 ↪sum(prediction_inside,dim=(2,3))
    #
    # ==================================================

    # ==================================================
    # fill up the blank for the estimate of the outside of segmenting region
    #
    estimate[1] = torch.sum(input*prediction_outside,dim=(2,3)) / torch.
 ↪sum(prediction_outside,dim=(2,3))
    #
    # ==================================================

    return estimate
```

[380]:

```
def compute_loss_data(input, prediction):

    (batch, channel, height, width) = input.size()
    estimate = compute_estimate(input, prediction)

    prediction_inside   = prediction
    prediction_outside  = 1 - prediction

    estimate0           = torch.unsqueeze(torch.unsqueeze(estimate[0], dim=-1),␣
 ↪dim=-1)
    estimate1           = torch.unsqueeze(torch.unsqueeze(estimate[1], dim=-1),␣
 ↪dim=-1)

    residual_inside     = torch.square(input - estimate0)
    residual_outside    = torch.square(input - estimate1)

    # ==================================================
    # fill up the blank for the data fidelity of the inside of segmenting region
    #
    fidelity_inside     = (residual_inside*prediction_inside).sum()
    #
    # ==================================================

    # ==================================================
    # fill up the blank for the data fidelity of the inside of segmenting region
    #
    fidelity_outside    = (residual_outside*prediction_outside).sum()
    #
    # ==================================================

    loss_data       = (fidelity_inside + fidelity_outside) / (batch * channel *␣
↪height * width)
    loss_data_value = loss_data.item()

    return loss_data, loss_data_value
```

[381]:

```
def compute_regularization(prediction):

    (batch, channel, height, width) = prediction.size()

    gradient_height = torch.abs(prediction[:, :, 1:, :] - prediction[:, :, :-␣
↪1, :]).sum()
    gradient_width  = torch.abs(prediction[:, :, :, 1:] - prediction[:, :, :, :
↪-1]).sum()

    loss_regularization         = (gradient_height + gradient_width) / (batch *␣
↪channel * height * width)
    loss_regularization_value   = loss_regularization.item()

    return loss_regularization, loss_regularization_value
```

1.11 compute the loss

[382]:

```
def compute_loss(input, prediction, alpha):

    (loss_data, _)              = compute_loss_data(input, prediction)
    (loss_regularization, _)    = compute_regularization(prediction)

    # ==================================================
    # fill up the blank for the loss that consists of the data fidelity and the␣
↪regularization with a weight
    #
    loss        = loss_data + alpha * loss_regularization
    #
    # ==================================================

    loss_value  = loss.item()

    return loss, loss_value
```

1.12 compute the accuracy

[383]:
```
def compute_accuracy(prediction, label):

    prediction  = prediction.squeeze(axis=1)
    label       = label.squeeze(axis=1)

    prediction_binary   = (prediction >= 0.5).cpu().numpy()
    label               = label.bool().cpu().numpy()

    region_intersection = prediction_binary & label
    region_union        = prediction_binary | label

    area_intersection   = region_intersection.sum(axis=1).sum(axis=1).
 ↪astype(float)
    area_union          = region_union.sum(axis=1).sum(axis=1).astype(float)

    eps         = np.finfo(float).eps
    correct     = area_intersection / (area_union + eps)
    accuracy    = correct.mean() * 100.0

    return accuracy
```

1.13 variables for the learning curve

[384]:
```
loss_mean_train     = np.zeros(number_epoch)
loss_std_train      = np.zeros(number_epoch)
accuracy_mean_train = np.zeros(number_epoch)
accuracy_std_train  = np.zeros(number_epoch)

loss_mean_test      = np.zeros(number_epoch)
loss_std_test       = np.zeros(number_epoch)
accuracy_mean_test  = np.zeros(number_epoch)
accuracy_std_test   = np.zeros(number_epoch)
```

1.14 train

[385]:
```
def train(model, dataloader):

    loss_epoch      = []
    accuracy_epoch  = []

    model.train()

    for index_batch, (clean, noisy) in enumerate(dataloader):

        clean = clean.to(device)
        noisy = noisy.to(device)

        prediction          = compute_prediction(model, noisy)
        loss, loss_value    = compute_loss(noisy, prediction, weight_regular)
        accuracy1           = compute_accuracy(prediction, clean)
        accuracy2           = compute_accuracy(1 - prediction, clean)
        accuracy            = np.maximum(accuracy1, accuracy2)

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

        loss_epoch.append(loss_value)
        accuracy_epoch.append(accuracy)

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

    accuracy_mean_epoch = np.mean(accuracy_epoch)
    accuracy_std_epoch  = np.std(accuracy_epoch)

    loss        = {'mean' : loss_mean_epoch, 'std' : loss_std_epoch}
    accuracy    = {'mean' : accuracy_mean_epoch, 'std' : accuracy_std_epoch}

    return (loss, accuracy)
```

1.15 test

[386]:
```
def test(model, dataloader):

    loss_epoch      = []
    accuracy_epoch  = []

    model.eval()

    for index_batch, (clean, noisy) in enumerate(dataloader):

        clean = clean.to(device)
        noisy = noisy.to(device)

        prediction          = compute_prediction(model, noisy)
        loss, loss_value    = compute_loss(noisy, prediction, weight_regular)
        accuracy1           = compute_accuracy(prediction, clean)
        accuracy2           = compute_accuracy(1 - prediction, clean)
        accuracy            = np.maximum(accuracy1, accuracy2)

        loss_epoch.append(loss_value)
```

```python
        accuracy_epoch.append(accuracy)

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

    accuracy_mean_epoch = np.mean(accuracy_epoch)
    accuracy_std_epoch  = np.std(accuracy_epoch)

    loss        = {'mean' : loss_mean_epoch, 'std' : loss_std_epoch}
    accuracy    = {'mean' : accuracy_mean_epoch, 'std' : accuracy_std_epoch}

    return (loss, accuracy)
```

1.16 train and test

[387]:

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

    # ================================================================================
    #
    # training
    #
    # ================================================================================
    (loss_train, accuracy_train) = train(model, dataloader_train)

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

    accuracy_mean_train[i]  = accuracy_train['mean']
    accuracy_std_train[i]   = accuracy_train['std']

    # ================================================================================
    #
    # testing
    #
    # ================================================================================
    (loss_test, accuracy_test) = test(model, dataloader_test)

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

    accuracy_mean_test[i]   = accuracy_test['mean']
    accuracy_std_test[i]    = accuracy_test['std']
```

```
100%|     | 300/300 [10:05<00:00,  2.02s/it]
```

---

2 functions for visualizing the results

2.1 plot curve

[388]:
```
def plot_data_grid(data, index_data, nRow, nCol):

    size_col = 1.5
    size_row = 1.5

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

    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()
```

[389]:
```
def plot_data_tensor_grid(data, index_data, nRow, nCol):

    size_col = 1.5
    size_row = 1.5

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

    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()
```

[390]:
```
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()
```

[391]:
```
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))
```

[392]:
```
def get_data_last(data, index_start):

    data_last = data[index_start:]

    return data_last
```

[393]:
```
def get_max_last_range(data, index_start):
    data_range = get_data_last(data, index_start)
    value = data_range.max()

    return value
```

```
[394]: 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

```
[395]: def function_result_01():

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

           nRow = 8
           nCol = 6

           number_data = len(dataset_train)
           step        = np.floor(number_data / (nRow * nCol))
           index_data  = np.arange(0, number_data, step)
           index_plot  = np.arange(0, nRow * nCol)

           _, data     = dataset_train[index_data]
           data        = data[0]

           plot_data_grid(data, index_plot, nRow, nCol)
```

```
[396]: def function_result_02():

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

           nRow = 8
           nCol = 6

           number_data = len(dataset_train)
           step        = np.floor(number_data / (nRow * nCol))
           index_data  = np.arange(0, number_data, step)
```

```
    index_plot  = np.arange(0, nRow * nCol)

    data, _     = dataset_train[index_data]
    data        = data[0]

    plot_data_grid(data, index_plot, nRow, nCol)
```

[397]:
```
def function_result_03():

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

    nRow = 8
    nCol = 6

    number_data = len(dataset_train)
    step        = np.floor(number_data / (nRow * nCol))
    index_data  = np.arange(0, number_data, step)
    index_plot  = np.arange(0, nRow * nCol)

    _, data     = dataset_train[index_data]
    data        = data[0].unsqueeze(dim=1).to(device)
    prediction  = compute_prediction(model, data)

    plot_data_tensor_grid(prediction, index_plot, nRow, nCol)
```

[398]:
```
def function_result_04():

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

    nRow = 8
    nCol = 6

    number_data = len(dataset_test)
    step        = np.floor(number_data / (nRow * nCol))
    index_data  = np.arange(0, number_data, step)
    index_plot  = np.arange(0, nRow * nCol)

    _, data     = dataset_test[index_data]
    data        = data[0]

    plot_data_grid(data, index_plot, nRow, nCol)
```

[399]:
```
def function_result_05():

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

    nRow = 8
    nCol = 6

    number_data = len(dataset_test)
    step        = np.floor(number_data / (nRow * nCol))
    index_data  = np.arange(0, number_data, step)
    index_plot  = np.arange(0, nRow * nCol)

    data, _     = dataset_test[index_data]
    data        = data[0]

    plot_data_grid(data, index_plot, nRow, nCol)
```

[400]:
```
def function_result_06():

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

    nRow = 8
    nCol = 6

    number_data = len(dataset_test)
    step        = np.floor(number_data / (nRow * nCol))
    index_data  = np.arange(0, number_data, step)
    index_plot  = np.arange(0, nRow * nCol)

    _, data     = dataset_test[index_data]
    data        = data[0].unsqueeze(dim=1).to(device)
    prediction  = compute_prediction(model, data)

    plot_data_tensor_grid(prediction, index_plot, nRow, nCol)
```

[401]:
```
def function_result_07():

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

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

[402]:
```
def function_result_08():

    print('[plot the training accuracy]')
    print('')
    plot_curve_error(accuracy_mean_train, accuracy_std_train, 'epoch',␣
 ↪'accuracy', 'accuracy (training)')
```

[403]:

```
def function_result_09():

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

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

[404]:

```
def function_result_10():

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

    plot_curve_error(accuracy_mean_test, accuracy_std_test, 'epoch',␣
 ↪'accuracy', 'accuracy (testing)')
```

[405]:

```
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)
```

[406]:

```
def function_result_12():

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

    data_last   = get_data_last(accuracy_mean_train, -10)
    index       = np.arange(0, 10)

    print_curve(data_last, index)
```

[407]:

```
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)
```

[408]:
```
def function_result_14():

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

    data_last   = get_data_last(accuracy_mean_test, -10)
    index       = np.arange(0, 10)

    print_curve(data_last, index)
```

[409]:
```
def function_result_15():

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

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

[410]:
```
def function_result_16():

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

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

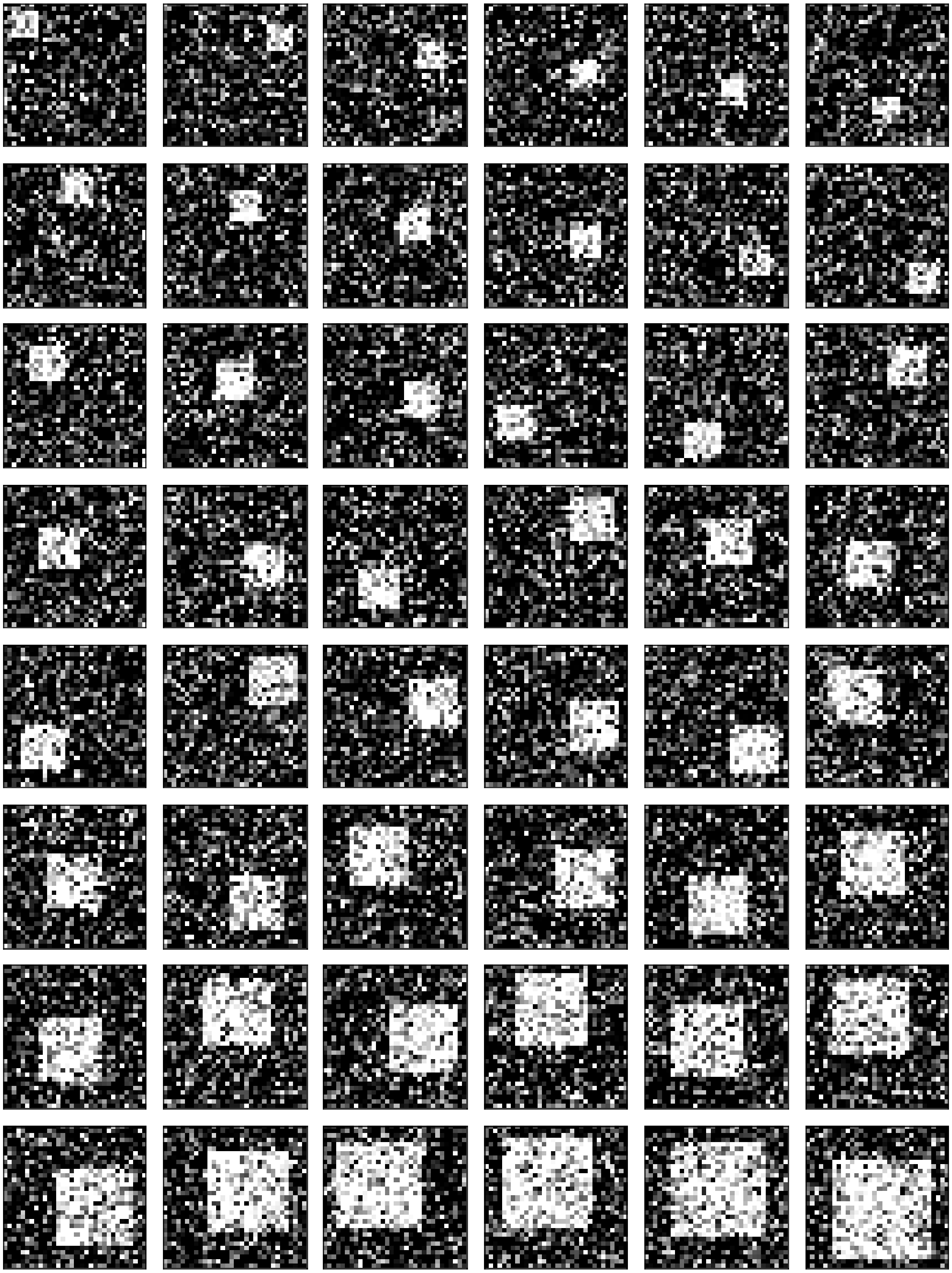
---

4 RESULTS

---

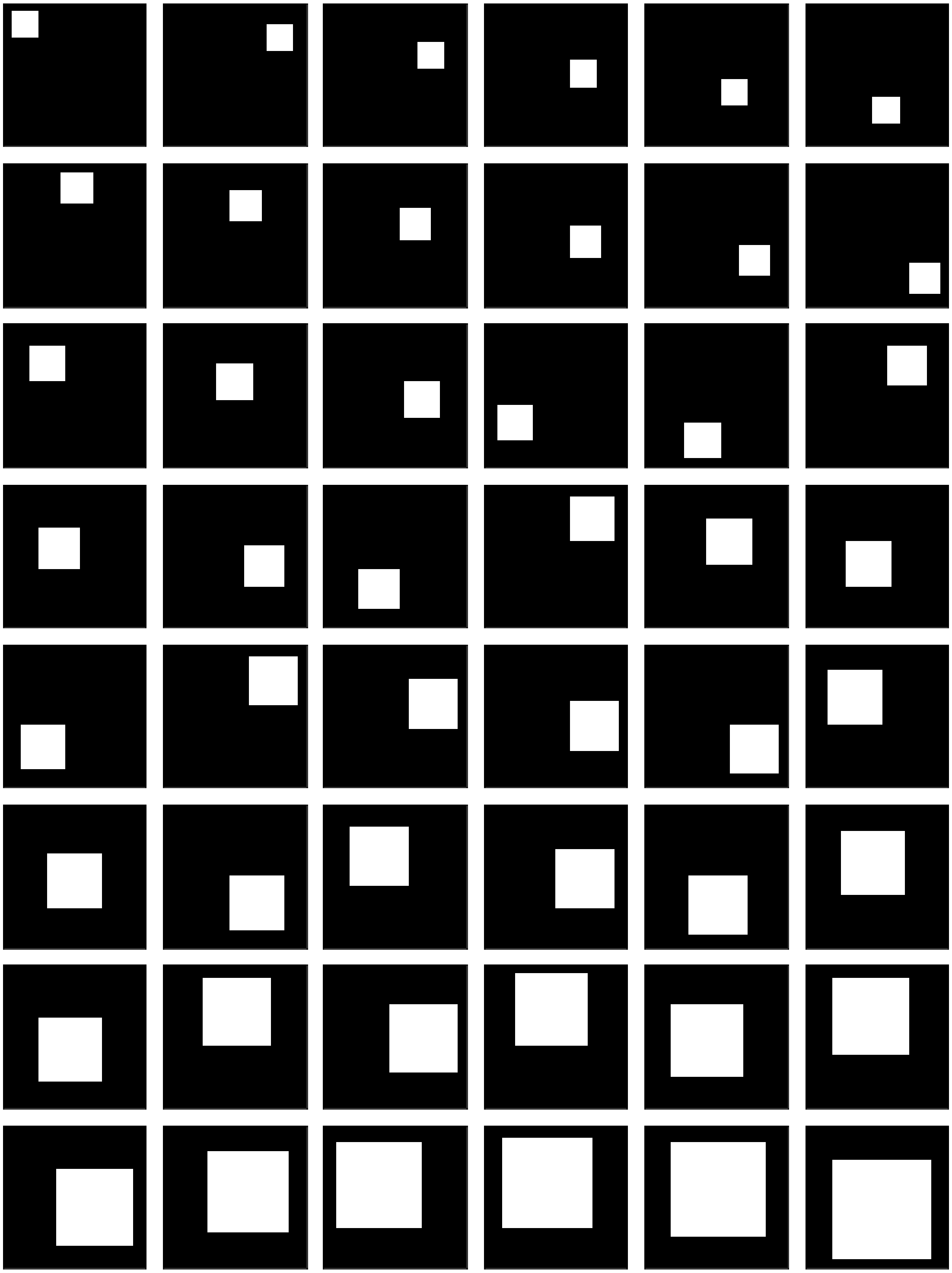
[411]:
```
number_result = 16

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 noisy images]
```



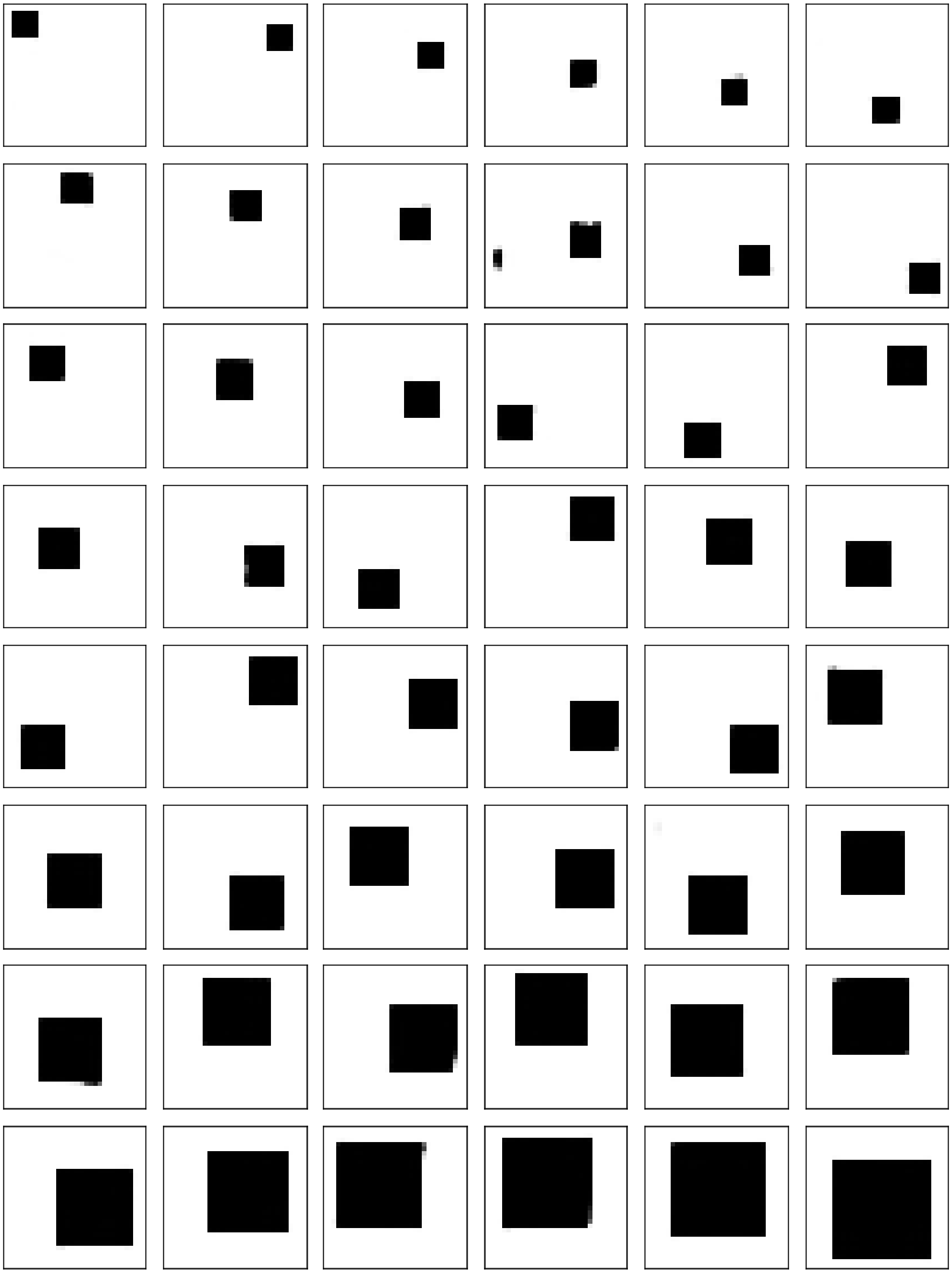
```
#######################################################################################
#
```

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

[plot examples of the training ground truth images]
```



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

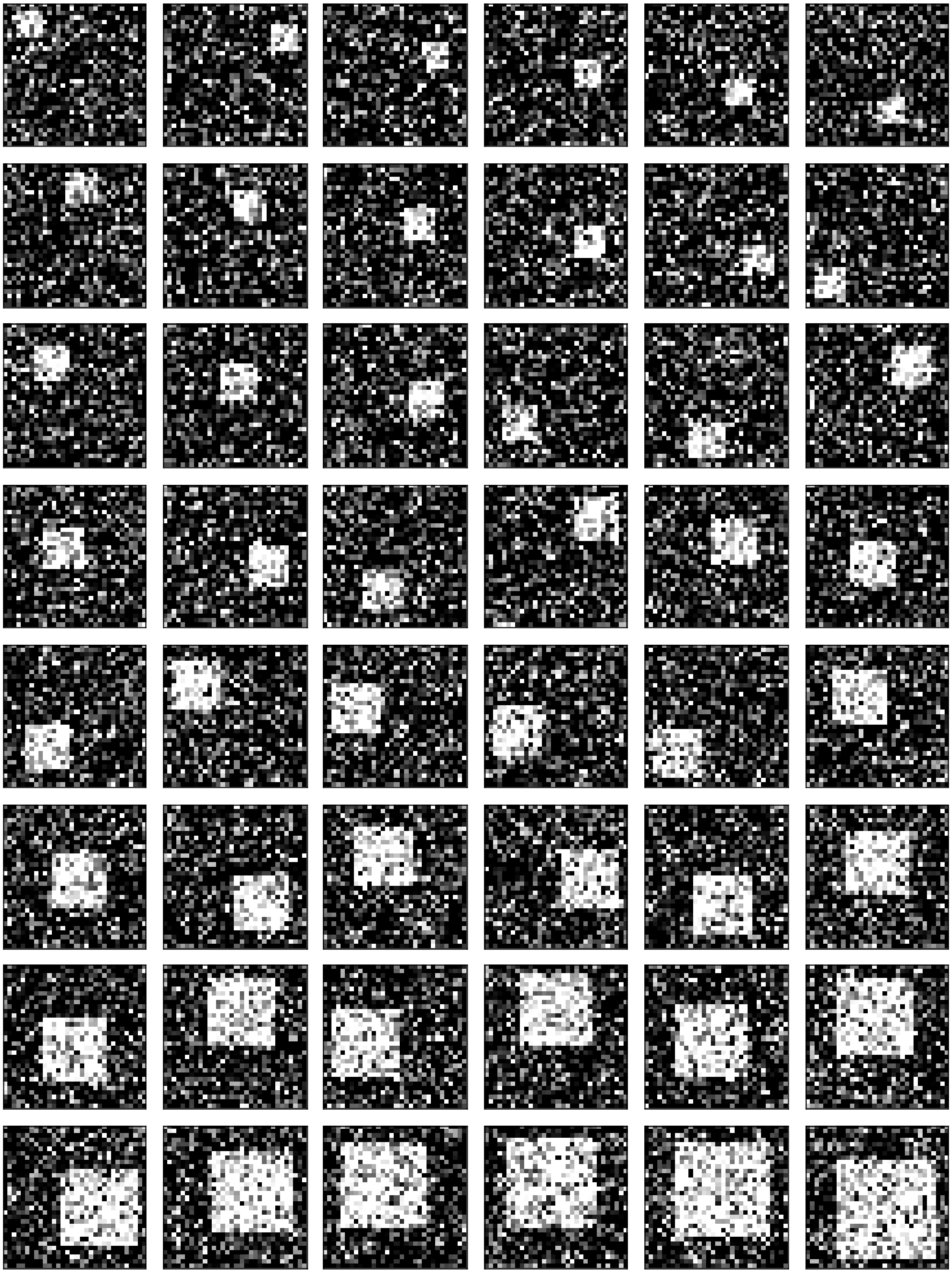
```
# RESULT # 03
#
################################################################################

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



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

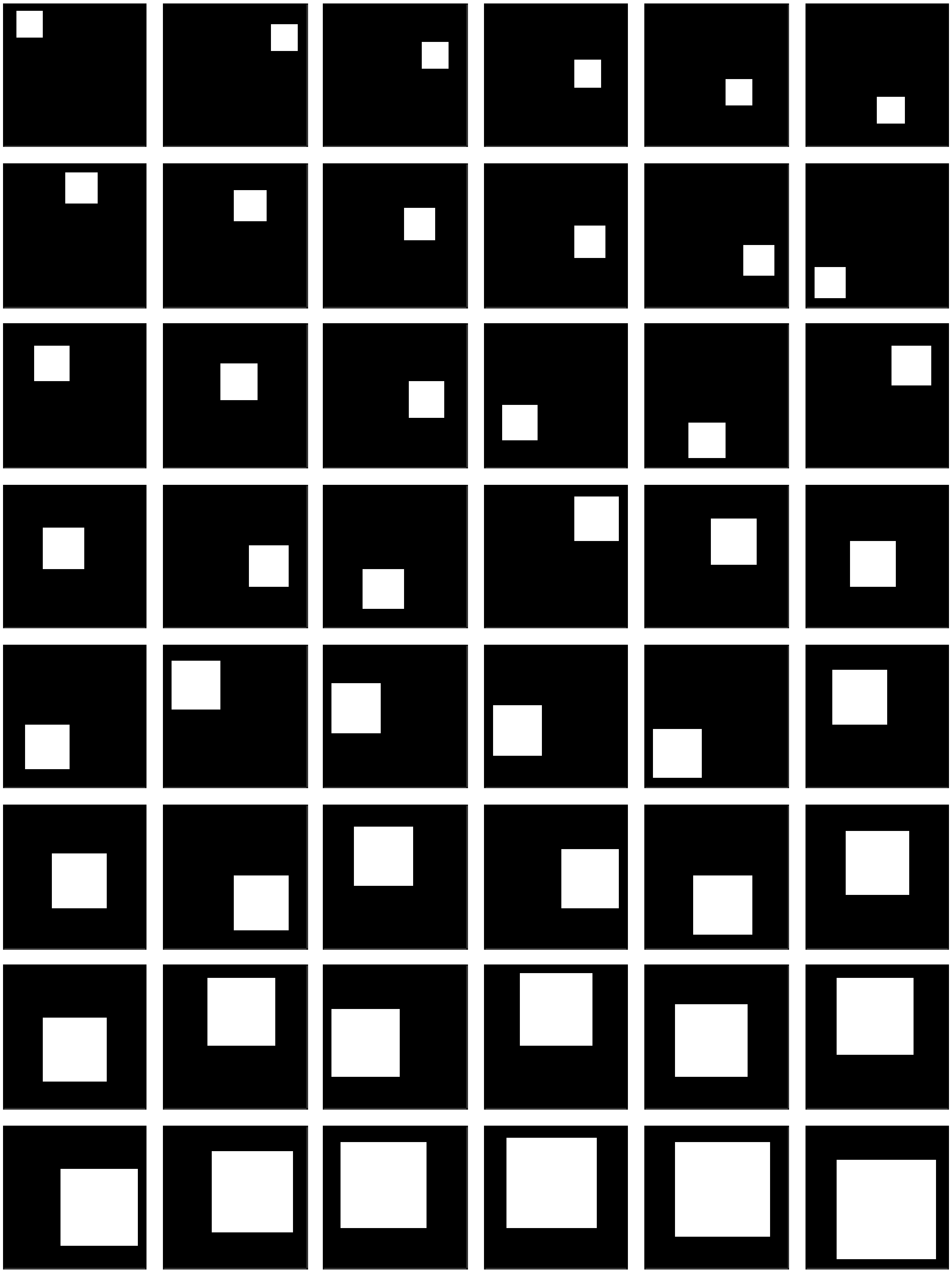
```
# RESULT # 04
#
################################################################################
```

[plot examples of the testing noisy images]



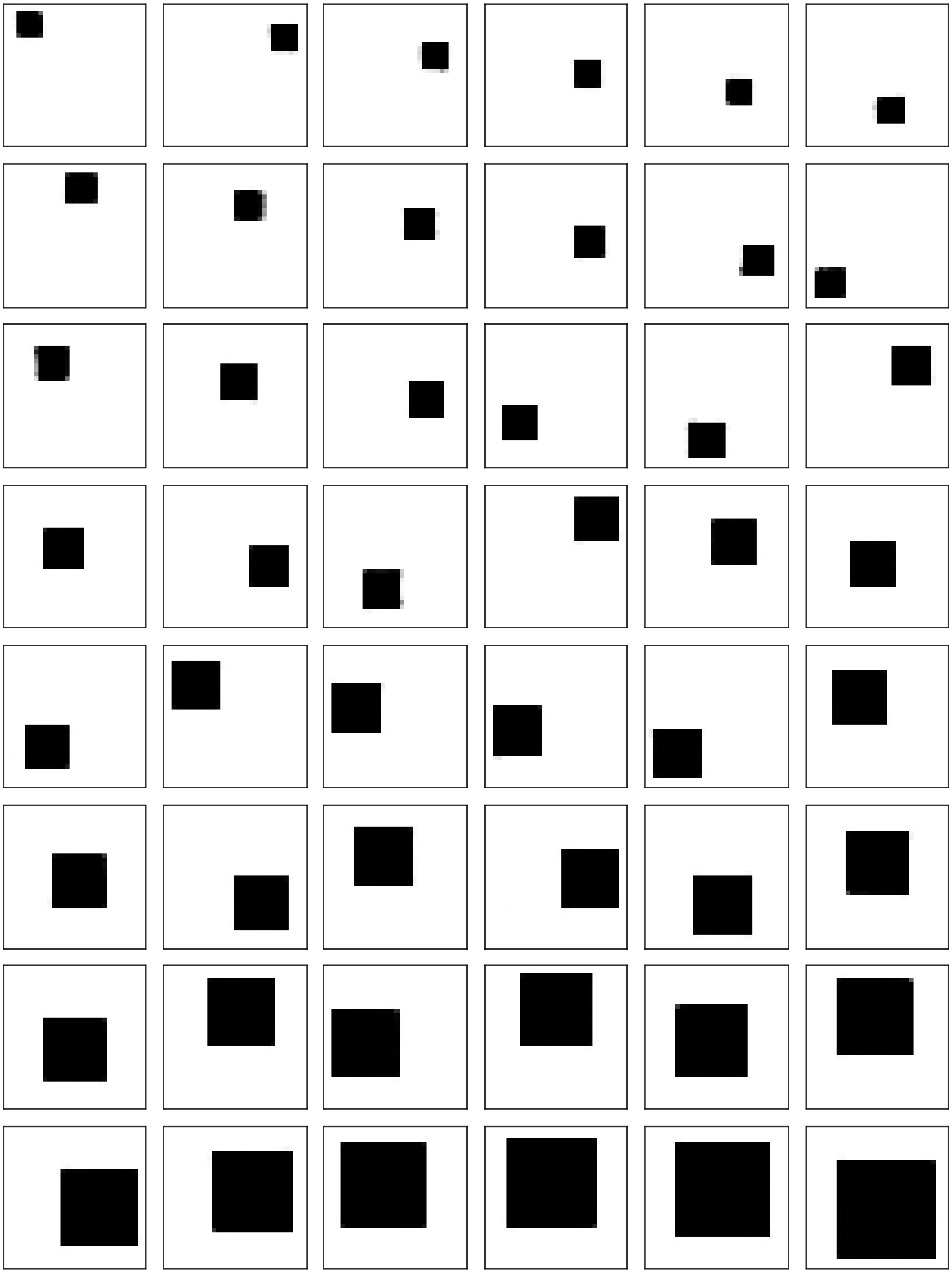
```
#######################################################################################
#
```

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

[plot examples of the testing ground truth images]
```



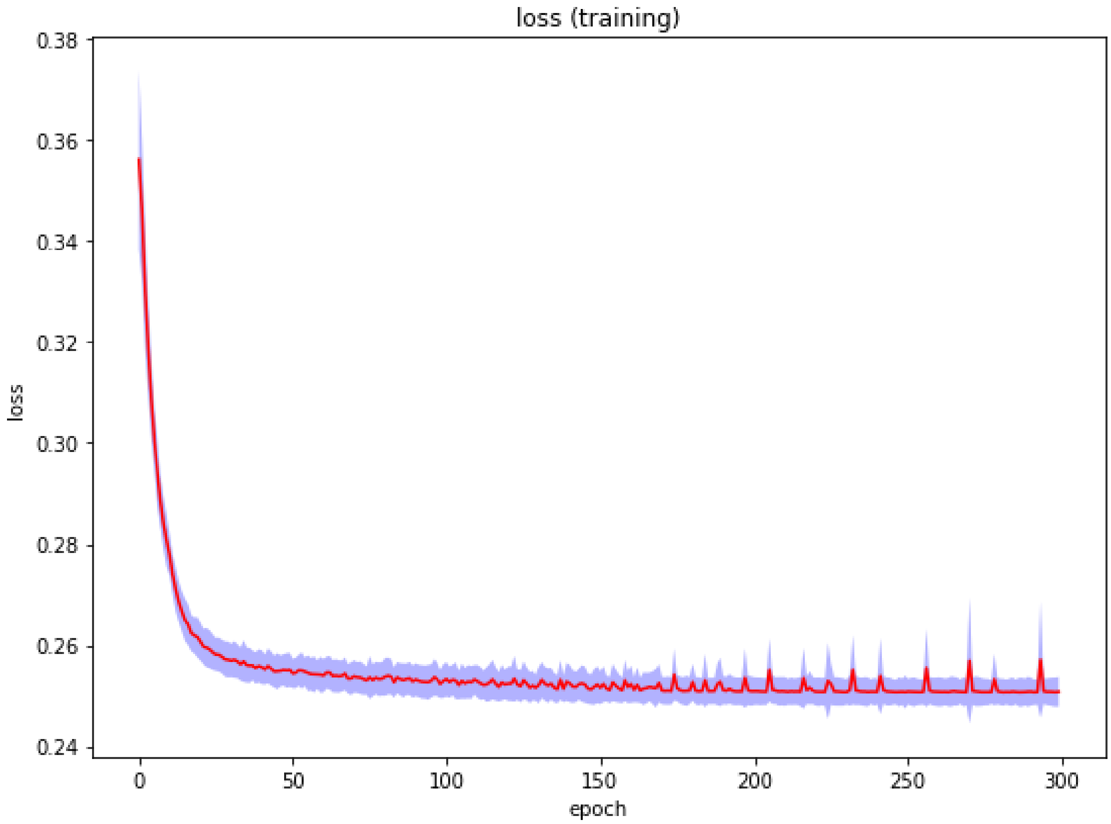
```
#################################################################################
#
```

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

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

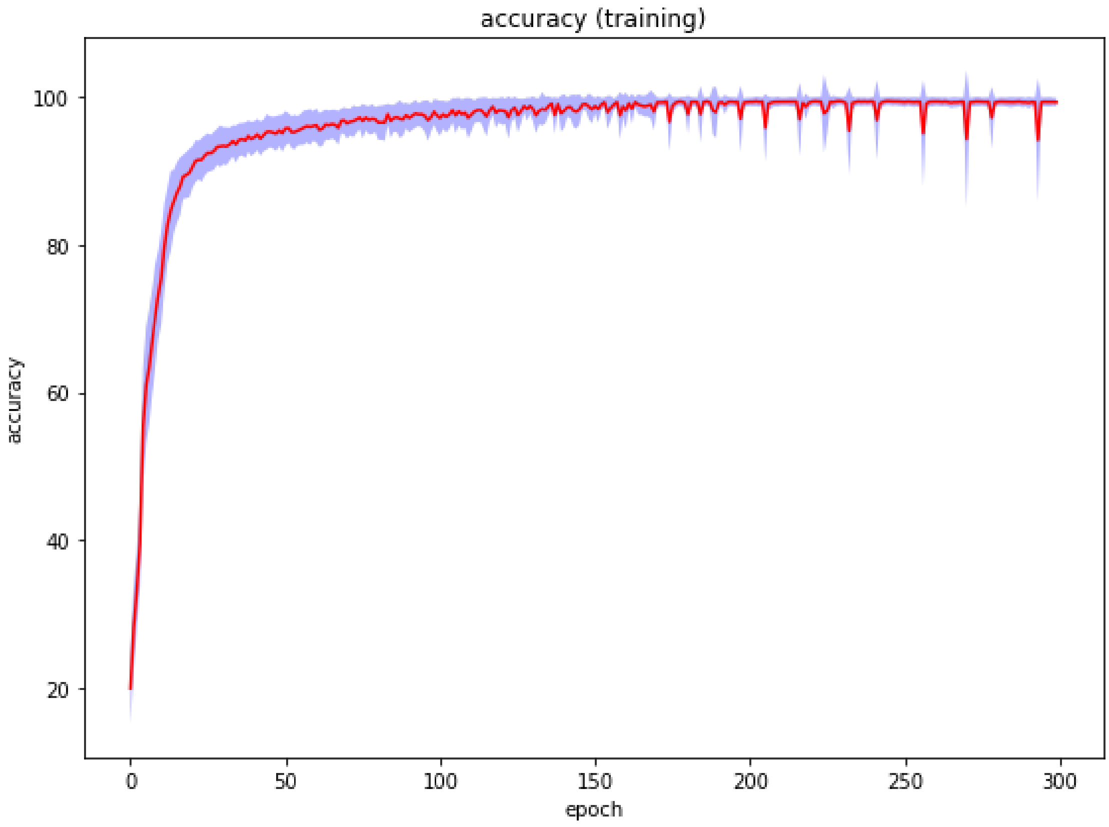


```
########################################################################################
#
```

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

[plot the training loss]
```

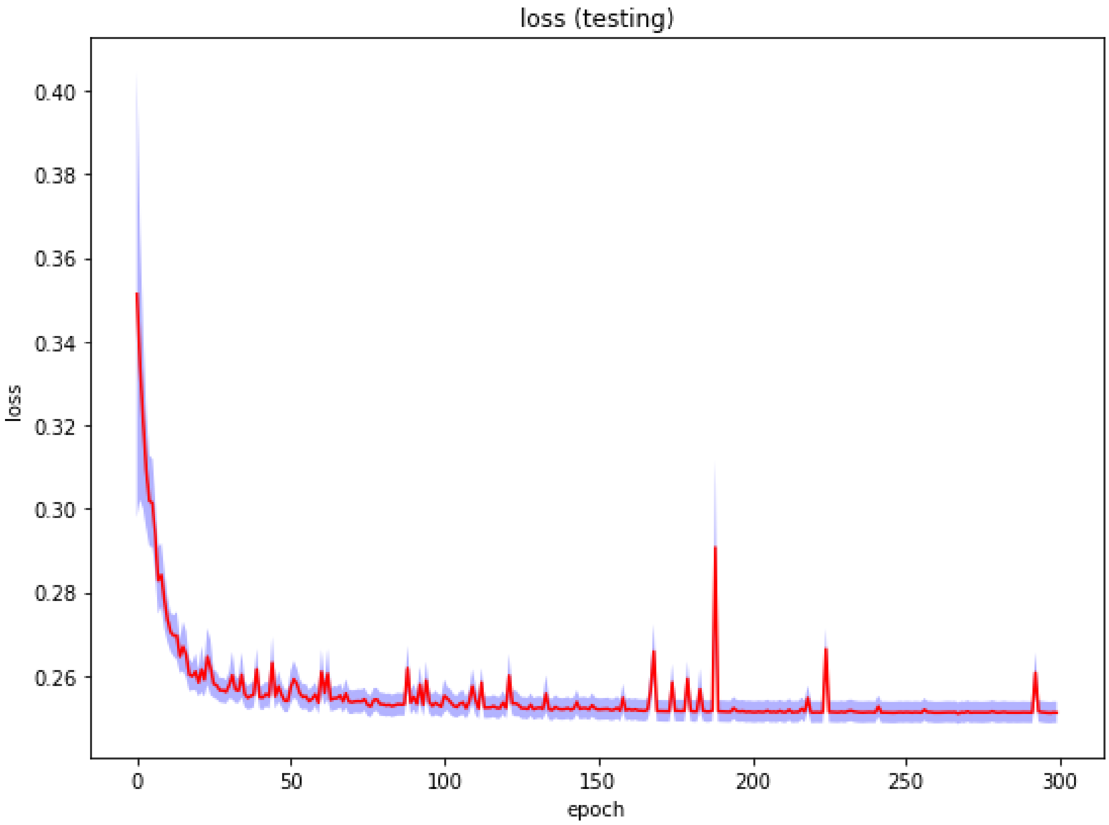


```
################################################################################
#
# RESULT # 08
#
################################################################################

[plot the training accuracy]
```

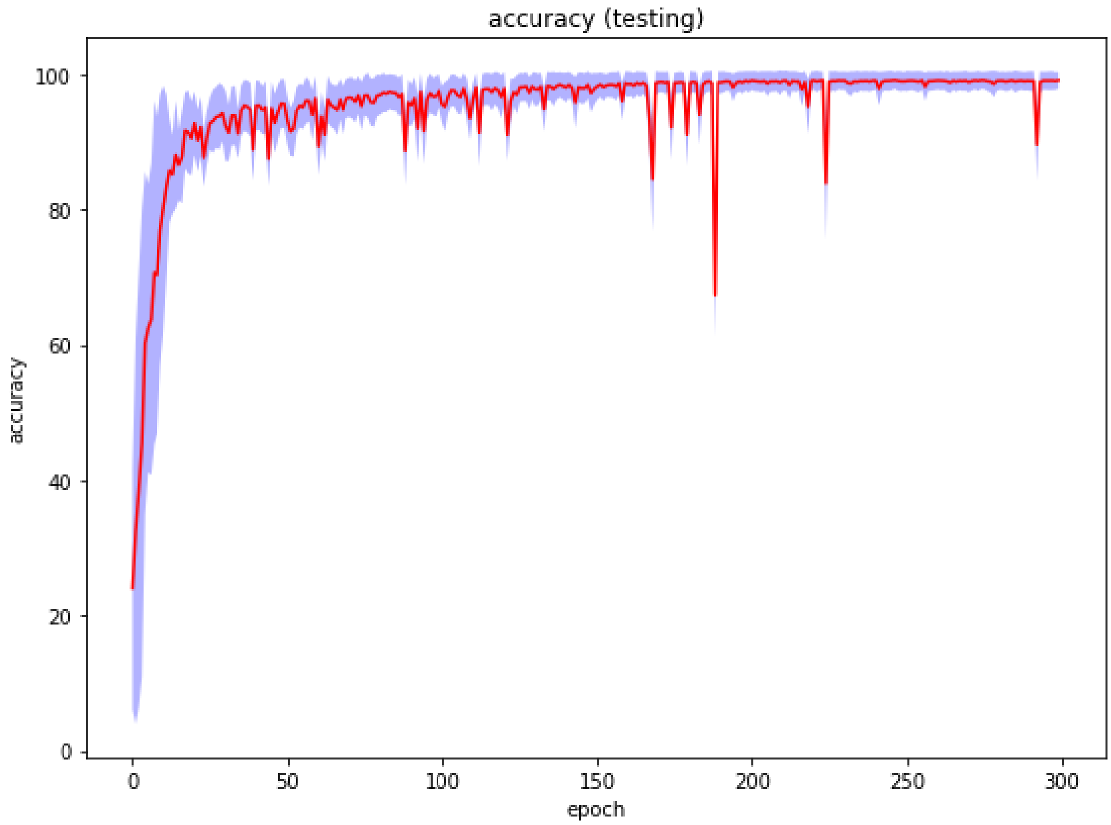


```
################################################################################
#
# RESULT # 09
#
################################################################################
```

[plot the testing loss]



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

[plot the testing accuracy]
```



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

[print the training loss at the last 10 epochs]

index =  0, value = 0.2508142391
index =  1, value = 0.2506943366
index =  2, value = 0.2507651859
index =  3, value = 0.2571615773
index =  4, value = 0.2509107372
index =  5, value = 0.2507848499
index =  6, value = 0.2508032464
index =  7, value = 0.2507485370
index =  8, value = 0.2507146729
index =  9, value = 0.2507966308

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

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

[print the training accuracy at the last 10 epochs]

index =  0, value = 99.2660382541
index =  1, value = 99.3560405666
index =  2, value = 99.3313315137
index =  3, value = 94.1665207295
index =  4, value = 99.3140943799
index =  5, value = 99.3830433342
index =  6, value = 99.3606859692
index =  7, value = 99.3740857529
index =  8, value = 99.3655432274
index =  9, value = 99.3251453961

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

[print the testing loss at the last 10 epochs]

index =  0, value = 0.2512596765
index =  1, value = 0.2512590512
index =  2, value = 0.2607073412
index =  3, value = 0.2515475055
index =  4, value = 0.2513491399
index =  5, value = 0.2512563592
index =  6, value = 0.2512153895
index =  7, value = 0.2511668811
index =  8, value = 0.2512799274
index =  9, value = 0.2512426695

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

[print the testing accuracy at the last 10 epochs]

index =  0, value = 99.0080253609
index =  1, value = 99.0201527804
index =  2, value = 89.6424898584
index =  3, value = 98.8456966803
index =  4, value = 99.0776522543
index =  5, value = 99.0672543309
index =  6, value = 99.0731572331
index =  7, value = 99.0935956118
index =  8, value = 99.0374127261
index =  9, value = 99.1267377244

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

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

best training accuracy = 99.3830433342

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

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

best testing accuracy = 99.1267377244
```

[411]: