assignment_09

November 19, 2021

1 Image Segmentation by Supervised Learning

1.1 Connect Google Drive

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

1.2 import libraries

[596]:
```
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

[674]:
```
directory_data  = 'drive/MyDrive'
filename_data   = 'assignment_09_data.npz'
data            = np.load(os.path.join(directory_data, filename_data))

x_train = data['x_train']
y_train = data['y_train']

x_test  = data['x_test']
y_test  = data['y_test']

num_data_train  = x_train.shape[0]
```

```
num_data_test   = x_test.shape[0]

print('************************************************')
print('size of x_train :', x_train.shape)
print('size of y_train :', y_train.shape)
print('************************************************')
print('size of x_test :', x_test.shape)
print('size of y_test :', y_test.shape)
print('************************************************')
print('number of training image :', x_train.shape[0])
print('height of training image :', x_train.shape[1])
print('width of training image :', x_train.shape[2])
print('************************************************')
print('number of testing image :', x_test.shape[0])
print('height of testing image :', x_test.shape[1])
print('width of testing image :', x_test.shape[2])
print('************************************************')
```

```
************************************************
size of x_train : (600, 128, 128)
size of y_train : (600, 128, 128)
************************************************
size of x_test : (400, 128, 128)
size of y_test : (400, 128, 128)
************************************************
number of training image : 600
height of training image : 128
width of training image : 128
************************************************
number of testing image : 400
height of testing image : 128
width of testing image : 128
************************************************
```

1.4 hyper-parameters

[713]:
```
device          = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

number_epoch    = 500
size_minibatch  = 8
learning_rate   = 0.05
weight_decay    = 0.0002
```

1.5 custom data loader for the PyTorch framework

[714]:
```
class dataset(Dataset):

    def __init__(self, image, label, transform=False,prob=0.5):

        self.image  = image
        self.label  = label
        self.transform = transform
        self.prob = prob

    def __getitem__(self, index):

        image   = self.image[index]
        label   = self.label[index]

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

        image   = transforms.Resize(size=[64,64], interpolation=transforms.
↪InterpolationMode('bilinear'))(image)
        label   = transforms.Resize(size=[64,64], interpolation=transforms.
↪InterpolationMode('nearest'))(label)

        if self.transform:
            # ================================================
            # you can apply data augmentation here
            #
            if random.random() < self.prob:
                image = transforms.functional.hflip(image)
                label = transforms.functional.hflip(label)
            if random.random() < self.prob:
                image = transforms.functional.vflip(image)
                label = transforms.functional.vflip(label)
            if random.random() < self.prob:
                image = transforms.functional.adjust_brightness(image,0.7)
                label = transforms.functional.adjust_brightness(label,0.7)
            # ================================================
        return (image, label)

    def __len__(self):

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

1.6 construct datasets and dataloaders for training and testing

[715]:
```
dataset_train_transform = dataset(x_train, y_train, transform=True,prob=0.5)
dataset_train           = dataset(x_train, y_train)
dataset_test            = dataset(x_test, y_test)

dataloader_train_transform  = DataLoader(dataset_train_transform,␣
 ↪batch_size=size_minibatch, shuffle=True, drop_last=True)
dataloader_train            = DataLoader(dataset_train,␣
 ↪batch_size=size_minibatch, shuffle=False, 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

[716]:
```
(image_train, label_train)  = dataset_train[0]
(image_test, label_test)    = dataset_test[0]
(image_train_transform, label_train_transform)  = dataset_train_transform[0]
print(len(dataset_train_transform))
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('**********************************************************************')
print('shape of the image in the training transform dataset:',␣
 ↪image_train_transform.shape)
print('shape of the label in the training transform dataset:',␣
 ↪label_train_transform.shape)
print('**********************************************************************')
```

```
600
**********************************************************************
shape of the image in the training dataset: torch.Size([1, 64, 64])
shape of the label in the training dataset: torch.Size([1, 64, 64])
**********************************************************************
shape of the image in the testing dataset: torch.Size([1, 64, 64])
shape of the label in the testing dataset: torch.Size([1, 64, 64])
**********************************************************************
shape of the image in the training transform dataset: torch.Size([1, 64, 64])
shape of the label in the training transform dataset: torch.Size([1, 64, 64])
**********************************************************************
```

1.8 class for the neural network

[717]:
```
class DoubleConv(nn.Module):
    """   conv - BN - ReLU     """
    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)

class Down(nn.Module):
    """Downscaling with maxpool then double conv"""

    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)

class Up(nn.Module):
    """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=False):
        super().__init__()

        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear',␣
↪align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels , in_channels // 2,␣
↪kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)
```

```
    def forward(self, x1, x2):
        x1 = self.up(x1)
        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])

        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)

class OutConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(OutConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        return self.conv(x)

class UNet(nn.Module):
    def __init__(self, n_channels, n_classes, bilinear=True):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear

        self.inc = DoubleConv(n_channels, 64)
        self.down1 = Down(64, 128)
        self.down2 = Down(128, 256)
        self.down3 = Down(256, 512)
        factor = 2 if bilinear else 1
        self.down4 = Down(512, 1024 // factor)
        self.up1 = Up(1024, 512 // factor, bilinear)
        self.up2 = Up(512, 256 // factor, bilinear)
        self.up3 = Up(256, 128 // factor, bilinear)
        self.up4 = Up(128, 64, bilinear)
        self.outc = OutConv(64, n_classes)
        self.activation_out        = nn.Sigmoid()

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
```

```
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        logits = self.outc(x)
        y = self.activation_out(logits)
        return y
```

[718]:
```
class Network(nn.Module):

        def __init__(self, in_channel=1, out_channel=1, dim_feature=8,␣
↪dim_code=8, threshold_ReLU=0.01):

                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_middle        = nn.Conv2d(dim_feature * 2,␣
↪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.mbn                        = 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)

                m   = self.conv_middle(e2)
                mb  = self.mbn(m)
                c   = self.activation(mb)

                y2  = nn.Upsample(scale_factor=2, mode='bilinear',␣
↪align_corners=False)(c)
                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

[719]:
```
model       = UNet(1,1).to(device)
optimizer   = torch.optim.SGD(model.parameters(), lr=learning_rate,␣
↪weight_decay=weight_decay)
```

1.10 compute the prediction

[720]:
```
def compute_prediction(model, input):
    # ================================================
    # fill up the blank
    #

    prediction = model(input)

    #
    # ================================================
    return prediction
```

1.11 compute the loss

[721]:
```
def compute_loss(prediction, label):
    # ================================================
    # fill up the blank
    #

    criterion = nn.MSELoss()
    loss = criterion(prediction,label)

    #
    # ================================================
    loss_value  = loss.item()

    return loss, loss_value
```

1.12 compute the accuracy

[722]:
```
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

[723]:
```
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

[724]:
```
def train(model, dataloader):

    loss_epoch      = []
    accuracy_epoch  = []

    model.train()

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

        image = image.to(device)
        label = label.to(device)

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

        prediction          = compute_prediction(model,image)
        loss, loss_value    = compute_loss(prediction,label)
        accuracy            = compute_accuracy(prediction,label)

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

        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

[725]:
```
def test(model, dataloader):

    loss_epoch      = []
    accuracy_epoch  = []

    model.eval()

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

        image = image.to(device)
        label = label.to(device)

        # ================================================
        # fill up the blank
        #
        prediction          = compute_prediction(model,image)
        loss, loss_value    = compute_loss(prediction,label)
        accuracy            = compute_accuracy(prediction,label)
        #
        # ================================================

        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)
```

1.16 train and test

[726]:
```
#␣
 ↪============================================================================
#
# iterations for epochs
#
#␣
 ↪============================================================================
for i in tqdm(range(number_epoch)):

    #␣
 ↪============================================================================
    #
    # training
    #
    #␣
 ↪============================================================================
    (loss_train, accuracy_train) = train(model, dataloader_train_transform)

    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%|      | 500/500 [37:05<00:00,  4.45s/it]
```

2 functions for visualizing the results

2.1 plot curve

[727]:
```
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()
```

[728]:
```
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()
```

[729]:
```
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()
```

[730]:
```
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))
```

[731]:
```
def get_data_last(data, index_start):

    data_last = data[index_start:]

    return data_last
```

[732]:
```
def get_max_last_range(data, index_start):

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

    return value
```

[733]:
```
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

---

[734]:
```
def function_result_01():

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

    nRow = 8
    nCol = 6
    index_data  = np.arange(0, nRow * nCol)
    image_train,_ = dataset_train[index_data]
    image_train = image_train[0]

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

[735]:
```
def function_result_02():

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

    nRow = 8
    nCol = 6
    index_data  = np.arange(0, nRow * nCol)
    _,label_train = dataset_train[index_data]
    label_train = label_train[0]

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

[736]:
```
def function_result_03():

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

    nRow = 8
    nCol = 6
    index_data          = np.arange(0, nRow * nCol)
    image_train,_       = dataset_train[index_data]
    image_train         = image_train[0].unsqueeze(dim=1).to(device)
    prediction_train    = compute_prediction(model, image_train)

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

[737]:
```
def function_result_04():

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

    nRow = 8
    nCol = 6
    index_data      = np.arange(0, nRow * nCol)
    image_test,_    = dataset_test[index_data]
    image_test      = image_test[0]

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

[738]:
```
def function_result_05():

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

    nRow = 8
    nCol = 6
    index_data = np.arange(0, nRow * nCol)
    _,label_test = dataset_test[index_data]
    label_test = label_test[0]

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

[739]:
```
def function_result_06():

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

    nRow = 8
    nCol = 6
    index_data      = np.arange(0, nRow * nCol)
    image_test,_    = dataset_test[index_data]
    image_test      = image_test[0].unsqueeze(dim=1).to(device)
    prediction_test = compute_prediction(model, image_test)


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

[740]:
```
def function_result_07():

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

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

[741]:
```
def function_result_08():

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

    plot_curve_error(accuracy_mean_train, accuracy_std_train, 'epoch',␣
↪'accuracy', 'accuracy (training)')
```

[742]:
```
def function_result_09():

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

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

[743]:
```
def function_result_10():

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

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

[744]:
```
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)
```

[745]:
```
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)
```

[746]:
```
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)
```

[747]:
```
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)
```

[748]:
```
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))
```

[749]:
```
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

---

[750]:
```
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 images]
```

##
#

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

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

##
#

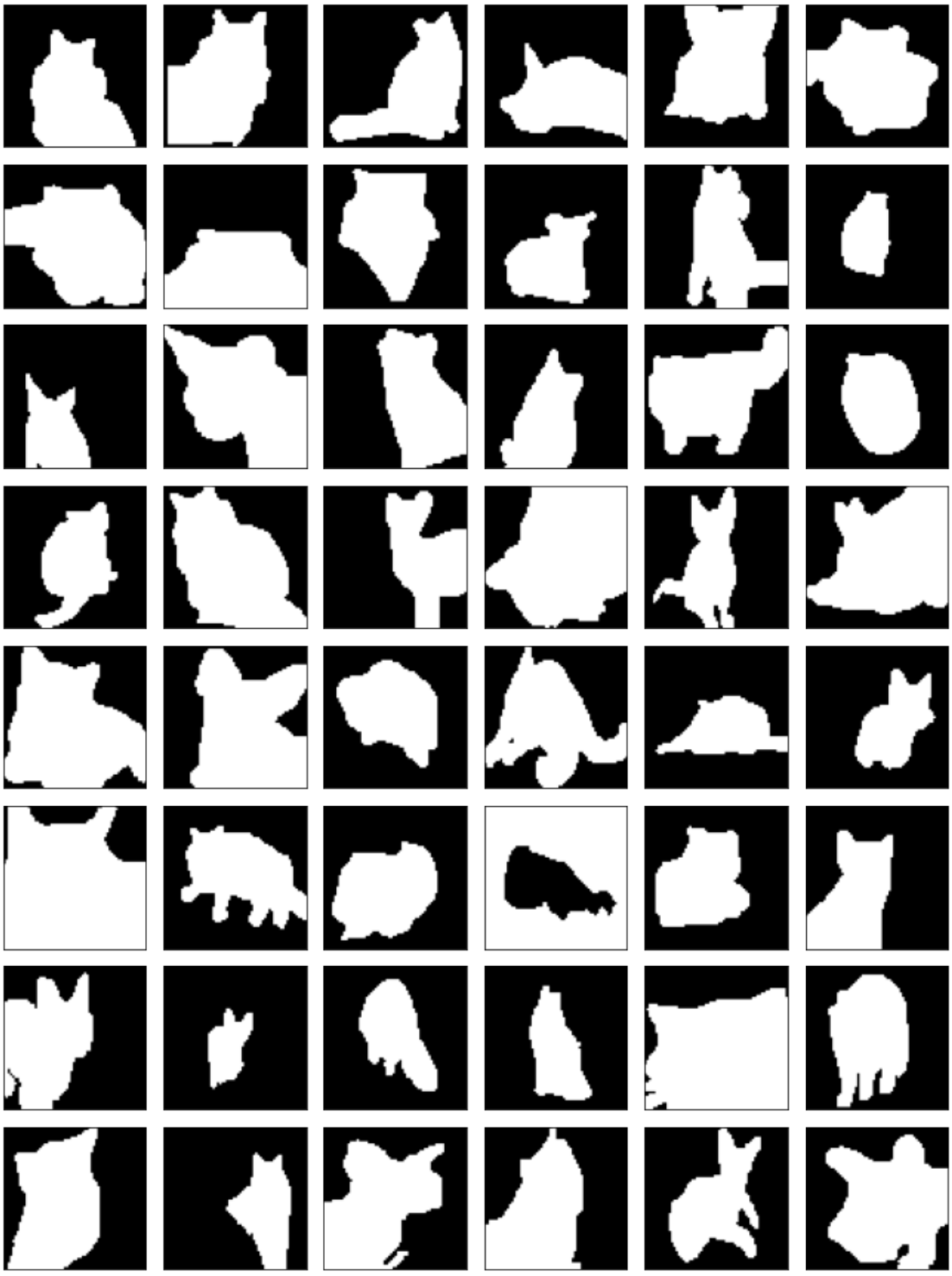
```
# RESULT # 03
#
################################################################################

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

##
#

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

[plot examples of the testing images]
```

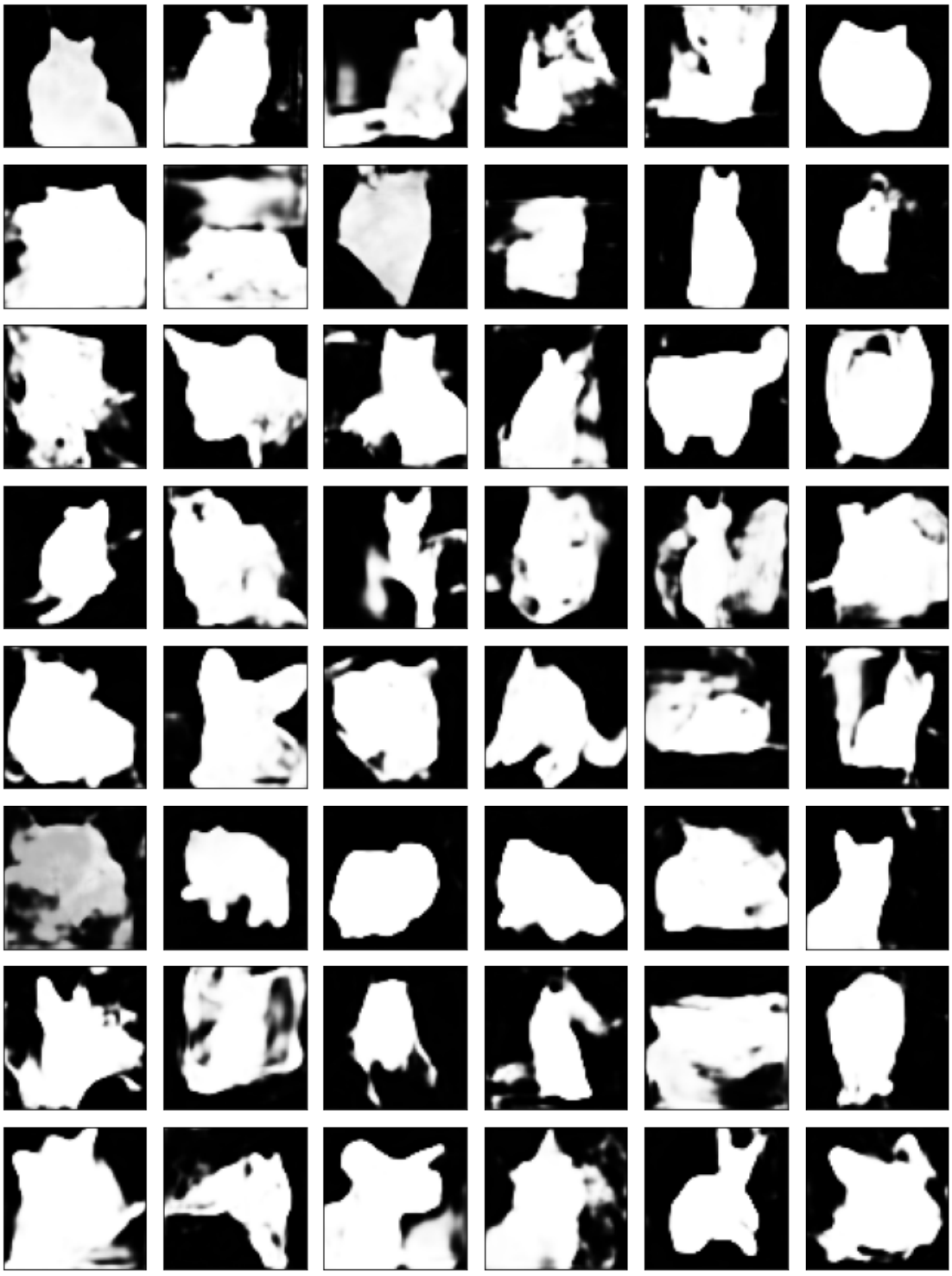
##
#

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

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



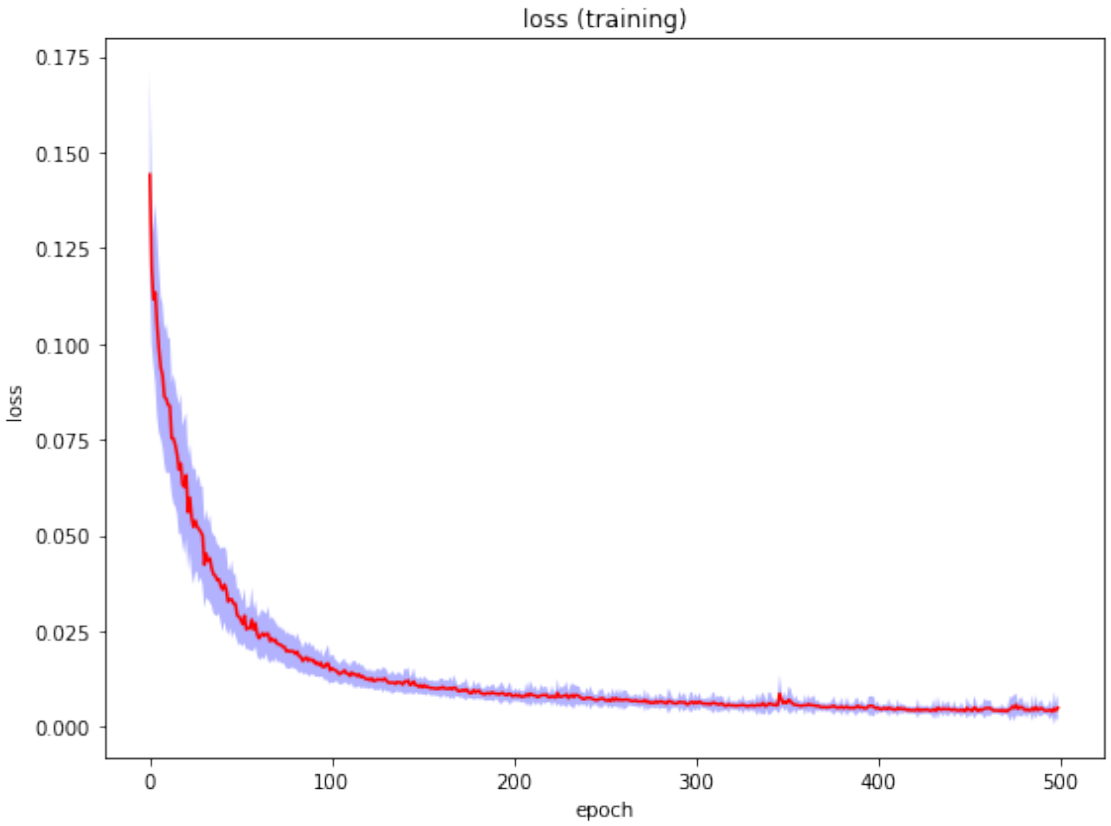
##
#

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

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

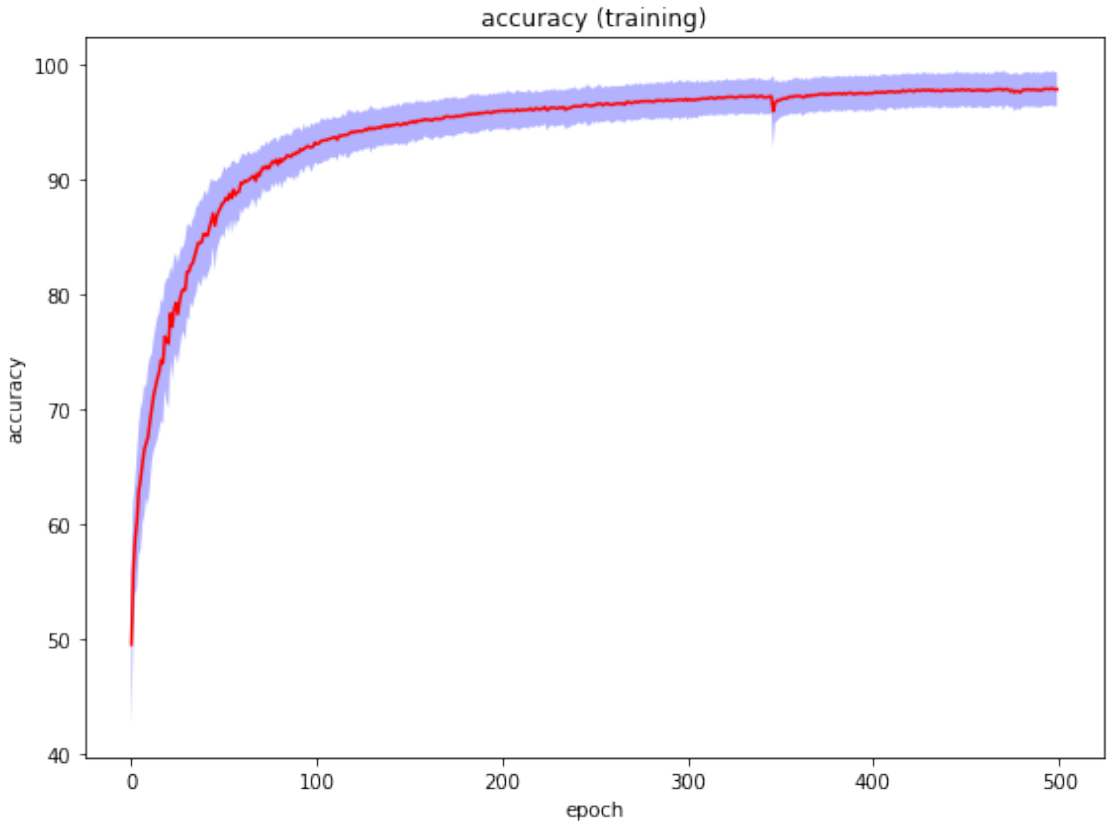


##
#

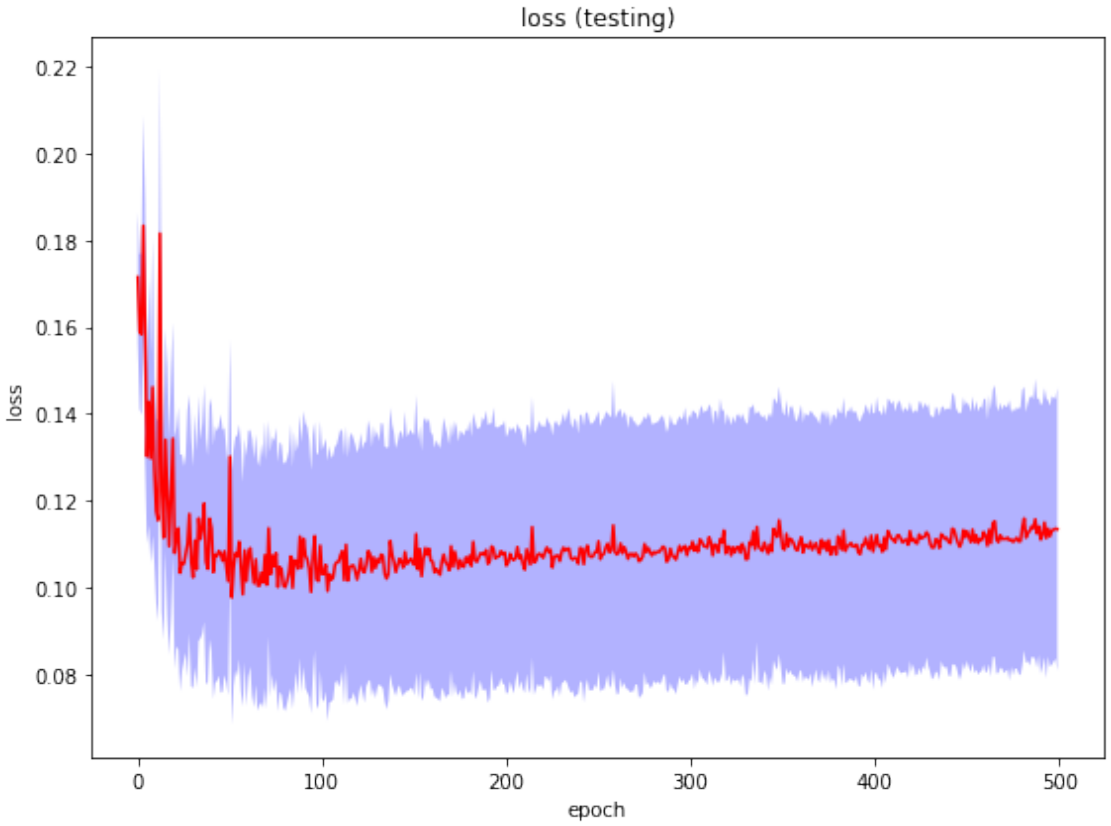
```
# RESULT # 07
#
################################################################################

[plot the training loss]
```



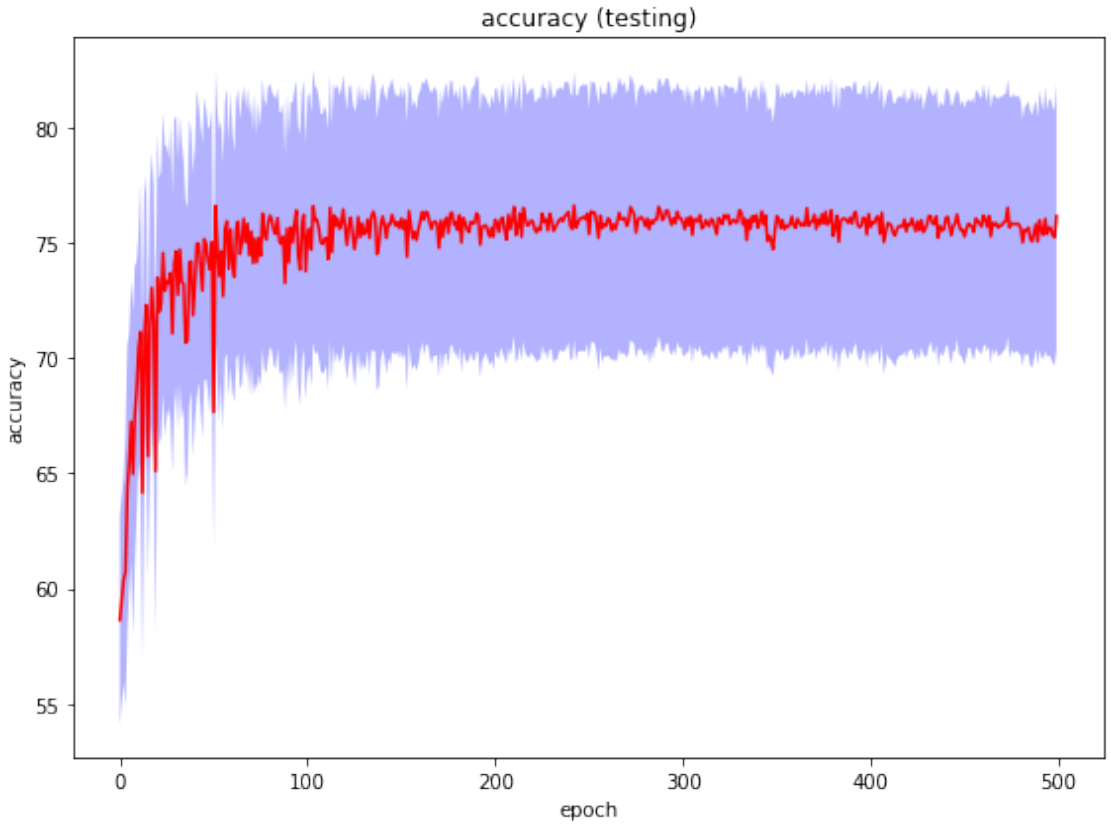

```
################################################################################
#
# RESULT # 08
#
################################################################################

[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 = 0.0049731206
index =  1, value = 0.0046070855
index =  2, value = 0.0042393796
index =  3, value = 0.0043552472
index =  4, value = 0.0043175510
index =  5, value = 0.0045242869
index =  6, value = 0.0039720613
index =  7, value = 0.0049017525
index =  8, value = 0.0043481766
index =  9, value = 0.0050341336

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

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

[print the training accuracy at the last 10 epochs]

index =  0, value = 97.7139025397
index =  1, value = 97.7496074926
index =  2, value = 97.8497427895
index =  3, value = 97.8548694759
index =  4, value = 97.8567574910
index =  5, value = 97.8220214584
index =  6, value = 97.8906104167
index =  7, value = 97.8560349436
index =  8, value = 97.8083335183
index =  9, value = 97.8247215820

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

[print the testing loss at the last 10 epochs]

index =  0, value = 0.1109726766
index =  1, value = 0.1120983627
index =  2, value = 0.1148860795
index =  3, value = 0.1112893097
index =  4, value = 0.1135071030
index =  5, value = 0.1117482653
index =  6, value = 0.1130559492
index =  7, value = 0.1132752594
index =  8, value = 0.1134637559
index =  9, value = 0.1133748698

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

[print the testing accuracy at the last 10 epochs]

index =  0, value = 75.9768249889
index =  1, value = 75.5539470240
index =  2, value = 75.3445617935
```

```
index =  3, value = 76.0421393794
index =  4, value = 75.3881419972
index =  5, value = 75.6722546662
index =  6, value = 75.6331447442
index =  7, value = 75.3360659623
index =  8, value = 75.2446803434
index =  9, value = 76.1637805547

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

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

best training accuracy = 97.8906104167

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

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

best testing accuracy = 76.1637805547
```

[750]: