

assignment_11

December 3, 2021

1 Image De-blurring by Supervised Learning

1.1 Connect Google Drive

```
[463]: 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

```
[464]: 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 math import log10
from tqdm import tqdm
import random
import os
```

1.3 load data

```
[465]: directory_data = 'drive/MyDrive'
filename_data = 'assignment_11_data.npz'
data = np.load(os.path.join(directory_data, filename_data))

original_train = data['original_train']
blur_train = data['blur_train']

original_test = data['original_test']
blur_test = data['blur_test']
```

```

num_data_train = original_train.shape[0]
num_data_test  = original_test.shape[0]

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

```

```

*****
size of original_train : (200, 256, 256)
size of blur_train : (200, 256, 256)
*****
size of original_test : (100, 256, 256)
size of blur_test : (100, 256, 256)
*****
number of training image : 200
height of training image : 256
width of training image : 256
*****
number of testing image : 100
height of testing image : 256
width of testing image : 256
*****

```

1.4 hyper-parameters

```

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

number_epoch    = 500
size_minibatch  = 8
learning_rate   = 0.01
weight_decay    = 0.000001

```

1.5 custom data loader for the PyTorch framework

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

    def __init__(self, original, blur, transform=False):

        self.original = original
        self.blur = blur
        self.transform = transform

    def __getitem__(self, index):

        original = self.original[index]
        blur = self.blur[index]

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

        if self.transform:

            crop_size = [256, 256]
            # random crop
            top = random.randint(0, original.shape[1] - crop_size[0])
            left = random.randint(0, original.shape[2] - crop_size[1])
            original = transforms.functional.crop(original, top, left, ↵
↵crop_size[0], crop_size[1])
            blur = transforms.functional.crop(blur, top, left, ↵
↵crop_size[0], crop_size[1])

            # random horizontal flip
            if random.random() > 0.5:
                original = transforms.functional.hflip(original)
                blur = transforms.functional.hflip(blur)

            # random vertical flip
            if random.random() > 0.5:
                original = transforms.functional.vflip(original)
                blur = transforms.functional.vflip(blur)

            if random.random() > 0.5:
                original = transforms.functional.hflip(original)
                blur = transforms.functional.hflip(blur)

            # random vertical flip
            if random.random() > 0.5:
                original = transforms.functional.vflip(original)
                blur = transforms.functional.vflip(blur)
```

```

        return (original, blur)

    def __len__(self):

        return self.original.shape[0]

```

1.6 construct datasets and dataloaders for training and testing

```

[468]: dataset_train_transform = dataset(original_train, blur_train, transform=True)
dataset_train                = dataset(original_train, blur_train)
dataset_test                 = dataset(original_test, blur_test)

dataloader_train_transform   = DataLoader(dataset_train_transform,
    ↪ batch_size=size_minibatch, shuffle=True, drop_last=True)
dataloader_train             = DataLoader(dataset_train, batch_size=1,
    ↪ shuffle=False, drop_last=True)
dataloader_test              = DataLoader(dataset_test, batch_size=1,
    ↪ shuffle=False, drop_last=True)

```

1.7 shape of the data when using the data loader

```

[469]: (original_train, blur_train) = dataset_train[0]
(original_test, blur_test)       = dataset_test[0]
(original_train_transform, blur_train_transform) = dataset_train_transform[0]
print('*****')
print('shape of the original in the training dataset:', original_train.shape)
print('shape of the blur in the training dataset:', blur_train.shape)
print('*****')
print('shape of the original in the testing dataset:', original_test.shape)
print('shape of the blur in the testing dataset:', blur_test.shape)
print('*****')
print('shape of the original in the training transform dataset:',
    ↪ original_train_transform.shape)
print('shape of the blur in the training transform dataset:',
    ↪ blur_train_transform.shape)
print('*****')

```

```

*****
shape of the original in the training dataset: torch.Size([1, 256, 256])
shape of the blur in the training dataset: torch.Size([1, 256, 256])
*****
shape of the original in the testing dataset: torch.Size([1, 256, 256])
shape of the blur in the testing dataset: torch.Size([1, 256, 256])
*****
shape of the original in the training transform dataset: torch.Size([1, 128,

```

128])

shape of the blur in the training transform dataset: torch.Size([1, 128, 128])

1.8 class for the neural network

```
[470]: # class Network(nn.Module):

#         def __init__(self, in_channel=1, out_channel=1,
#             dim_feature=4, threshold_ReLU=0.02):

#             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,
#             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
#             dim_feature * 1)
#             self.ebn2            = nn.BatchNorm2d(dim_feature
#             dim_feature * 2)
#             self.ebn3            = nn.BatchNorm2d(dim_feature
#             dim_feature * 4)
#             self.mbn             = nn.BatchNorm2d(dim_feature
#             dim_feature * 8)
#             self.dbn3            = nn.BatchNorm2d(dim_feature
#             dim_feature * 4)
#             self.dbn2            = nn.BatchNorm2d(dim_feature
#             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(c)
#                db3 = self.dbn3(y3)
#                d3  = self.activation(db3)

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

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

```

```

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

#             return y

```

```

[471]: 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 network

```

[472]: model      = Network().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate,
    ↪weight_decay=weight_decay)
#optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, betas=(0.
    ↪9, 0.999), eps=1e-08, weight_decay=weight_decay, amsgrad=False)

```

1.10 compute the prediction

```

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

    return prediction

```

1.11 compute the loss

```

[474]: def compute_loss(prediction, label):
    # =====
    # fill up the blank
    #
    criterion = nn.MSELoss()
    #
    # =====

    loss = criterion(prediction, label)

    return loss

```

1.12 compute the accuracy

```
[475]: def compute_accuracy(prediction, label):  
  
    prediction = prediction.squeeze(axis=1)  
    label      = label.squeeze(axis=1)  
    mse_loss   = torch.mean((prediction - label) ** 2)  
  
    if mse_loss == 0.0:  
        psnr = 100  
    else:  
        psnr = 10 * torch.log10(1 / mse_loss)  
  
    psnr = psnr.item()  
  
    return psnr
```

1.13 variables for the learning curve

```
[476]: 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

```
[477]: def train(model, dataloader):  
  
    loss_epoch      = []  
    accuracy_epoch  = []  
  
    model.train()  
  
    for index_batch, (original, blur) in enumerate(dataloader):  
  
        original     = original.to(device)  
        blur         = blur.to(device)  
  
        prediction   = compute_prediction(model, blur)  
        loss         = compute_loss(prediction, original)  
        accuracy     = compute_accuracy(prediction, original)
```

```

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

loss_epoch.append(loss.item())
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

```

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

    loss_epoch = []
    accuracy_epoch = []

    model.eval()

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

        original = original.to(device)
        blur = blur.to(device)

        prediction = compute_prediction(model, blur)
        loss = compute_loss(prediction, original)
        accuracy = compute_accuracy(prediction, original)

        loss_epoch.append(loss.item())
        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

```
[479]: #  
#-----  
#  
# 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 [14:07<00:00, 1.69s/it]
```

2 functions for visualizing the results

2.1 plot curve

```
[480]: 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()
```

```
[481]: 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()
```

```
[482]: 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()
```

```
[483]: 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))
```

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

    data_last = data[index_start:]

    return data_last
```

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

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

    return value
```

```
[486]: 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

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

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

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

    plot_data_grid(data_train, index_data, nRow, nCol)

[488]: def function_result_02():

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

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

    plot_data_grid(data_train, index_data, nRow, nCol)

[489]: def function_result_03():

    print('[plot examples of the training de-blurring results]')
    print('')

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

    plot_data_tensor_grid(prediction_train, index_data, nRow, nCol)

[490]: def function_result_04():

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

```

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

plot_data_grid(data_test, index_data, nRow, nCol)

```

```

[491]: def function_result_05():

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

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

    plot_data_grid(data_test, index_data, nRow, nCol)

```

```

[492]: def function_result_06():

    print('[plot examples of the testing de-blurring results]')
    print('')

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

    plot_data_tensor_grid(prediction_test, index_data, nRow, nCol)

```

```

[493]: def function_result_07():

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

    plot_curve_error(loss_mean_train, loss_std_train, 'epoch', 'loss', 'loss_
↪(training)')

```

```

[494]: def function_result_08():

    print('[plot the training accuracy]')

```



```

print('')

plot_curve_error(accuracy_mean_train, accuracy_std_train, 'epoch', '
→'accuracy', 'accuracy (training)')

```

```

[495]: def function_result_09():

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

    plot_curve_error(loss_mean_test, loss_std_test, 'epoch', 'loss', 'loss_
→(testing)')

```

```

[496]: def function_result_10():

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

    plot_curve_error(accuracy_mean_test, accuracy_std_test, 'epoch', '
→'accuracy', 'accuracy (testing)')

```

```

[497]: 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)

```

```

[498]: 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)

```

```

[499]: 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)

```

```
[500]: 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)
```

```
[501]: 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))
```

```
[502]: 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

```
[503]: 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 original images]
```



 #

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

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



#

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

[plot examples of the training de-blurring results]
```



 #


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

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

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

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

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



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

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

[plot examples of the testing de-blurring results]
```



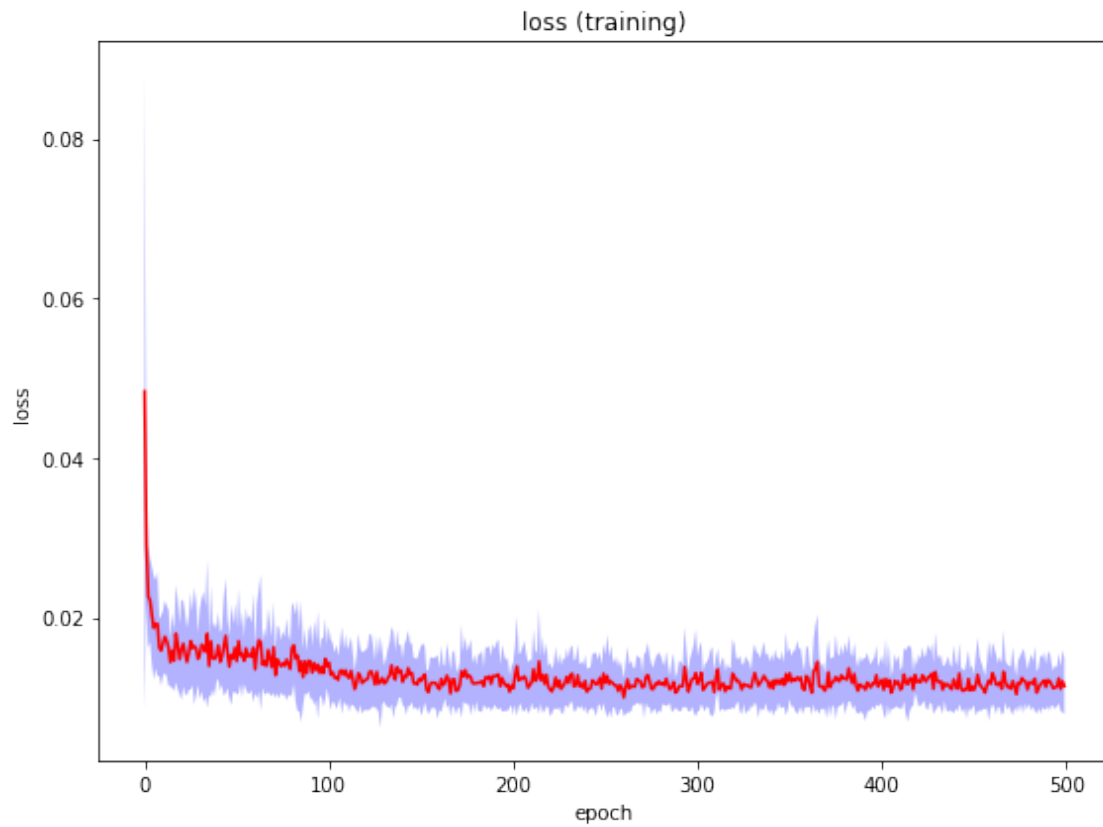
```
#####
#
```

```
# RESULT # 07
```

```
#
```

```
#####
```

```
[plot the training loss]
```



```
#####
```

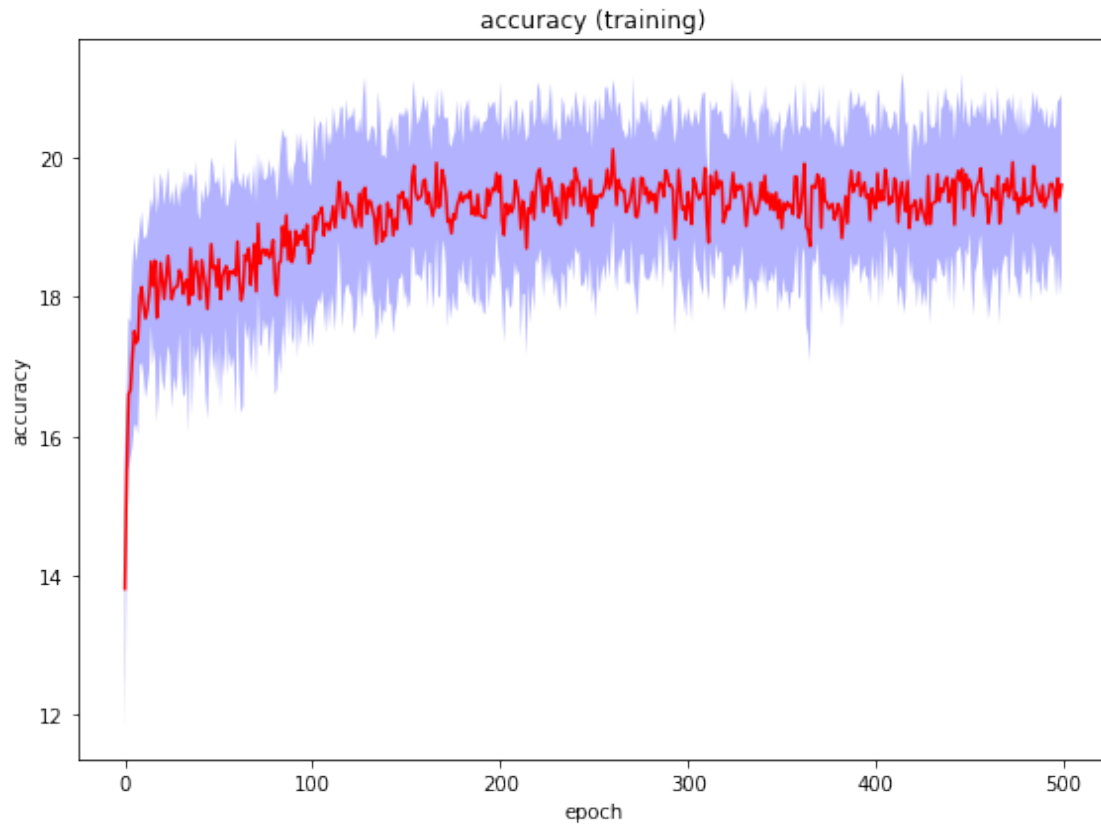
```
#
```

```
# RESULT # 08
```

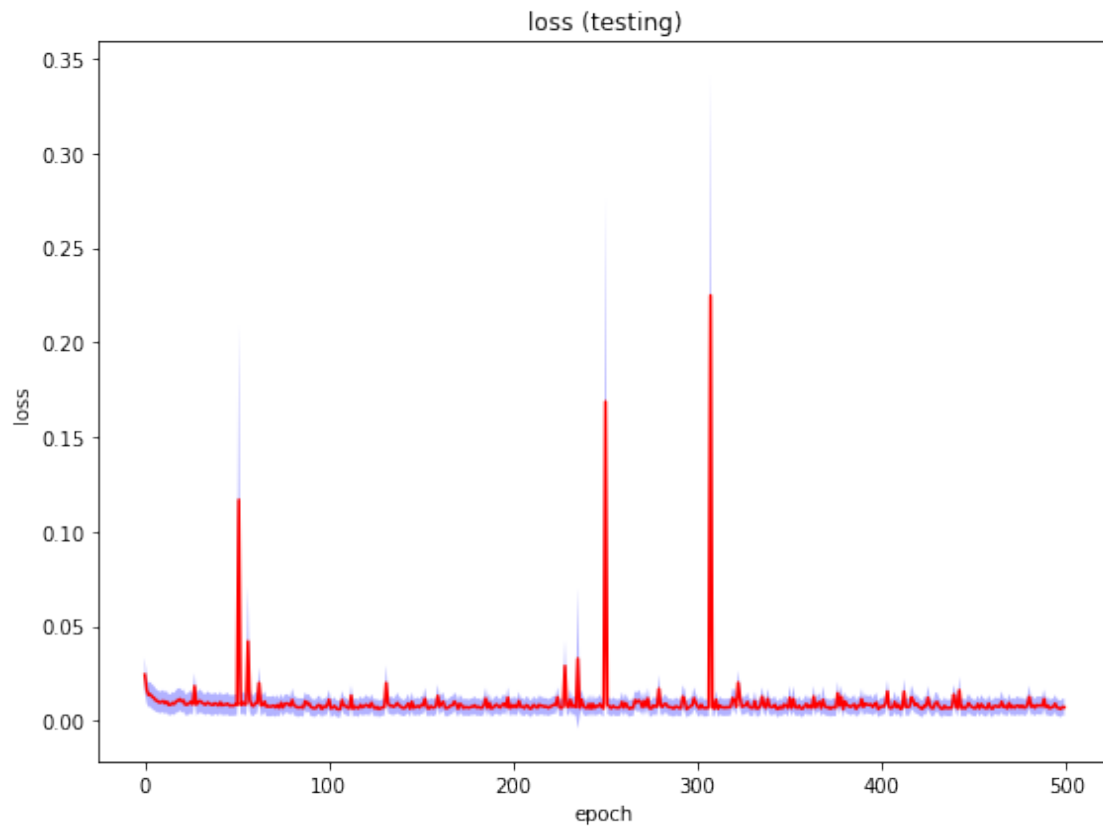
```
#
```

```
#####
```

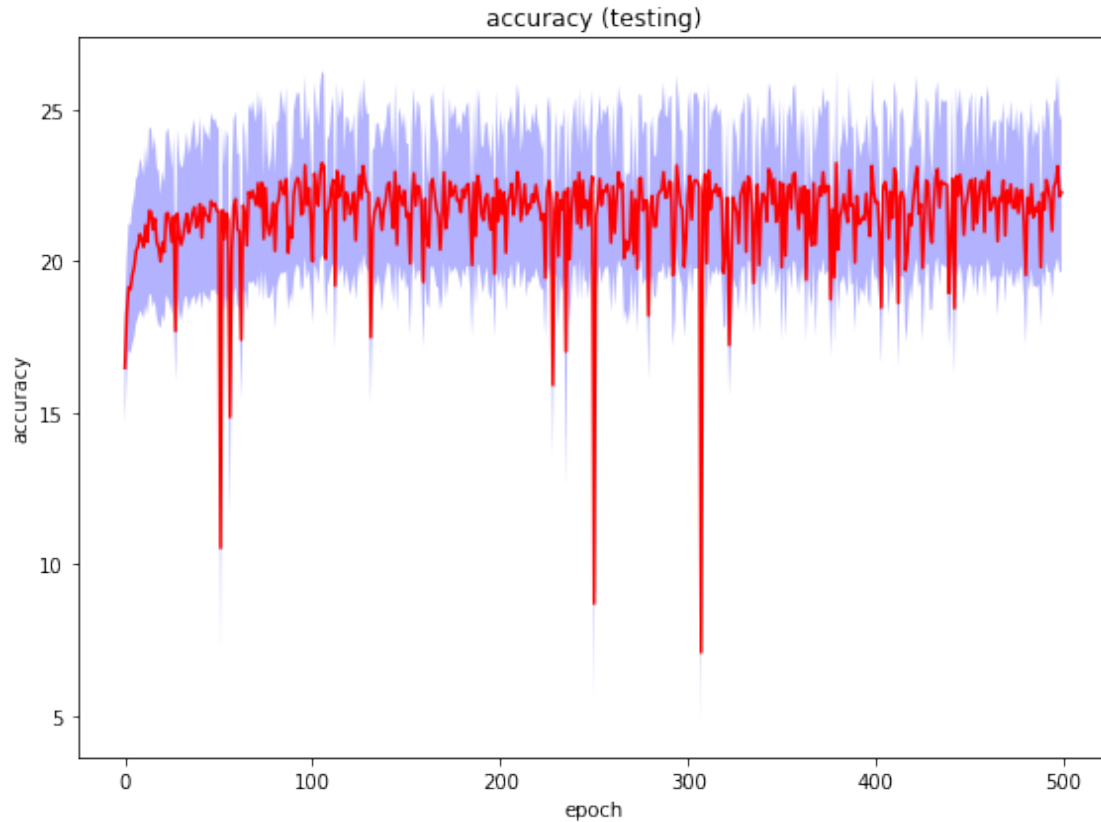
```
[plot the training accuracy]
```



```
#####  
#  
# 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 = 0.0121782186  
index = 1, value = 0.0120348612  
index = 2, value = 0.0116502666  
index = 3, value = 0.0111217045  
index = 4, value = 0.0112249208  
index = 5, value = 0.0113574982  
index = 6, value = 0.0123727315  
index = 7, value = 0.0110572469  
index = 8, value = 0.0119945704  
index = 9, value = 0.0114230361
```

```
#####
```

```

#
# RESULT # 12
#
#####

[print the training accuracy at the last 10 epochs]

index = 0, value = 19.2945107269
index = 1, value = 19.3952584076
index = 2, value = 19.4451042938
index = 3, value = 19.6232826233
index = 4, value = 19.6274692535
index = 5, value = 19.5404056549
index = 6, value = 19.2351422119
index = 7, value = 19.7019449615
index = 8, value = 19.4347164917
index = 9, value = 19.6141997528

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

[print the testing loss at the last 10 epochs]

index = 0, value = 0.0079851752
index = 1, value = 0.0065397952
index = 2, value = 0.0070088404
index = 3, value = 0.0073244836
index = 4, value = 0.0089900636
index = 5, value = 0.0069429583
index = 6, value = 0.0066246835
index = 7, value = 0.0061320923
index = 8, value = 0.0072112249
index = 9, value = 0.0070778896

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

[print the testing accuracy at the last 10 epochs]

index = 0, value = 21.7103660965
index = 1, value = 22.6840489006
index = 2, value = 22.3951174927

```

```
index = 3, value = 22.0108262444
index = 4, value = 21.0109188843
index = 5, value = 22.4603137589
index = 6, value = 22.6298655128
index = 7, value = 23.1412124634
index = 8, value = 22.1368829536
index = 9, value = 22.2815217972
```

```
#####
#
# RESULT # 15
#
#####
```

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

```
best training accuracy = 19.7019449615
```

```
#####
#
# RESULT # 16
#
#####
```

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

```
best testing accuracy = 23.1412124634
```

[503]: