Classification for Multiple Categories using Pytorch

Build a classifier for the digit classification task with 10 classes on the MNIST dataset

```
import os

# load data
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

# train
import torch
from torch import nn
from torch.nn import functional as F
import numpy as np

# visualization
import matplotlib.pyplot as plt
import pandas as pd
```

1. Data

- apply normalization

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,),(0.3081,)),  # mean value = 0.1307, standard deviation value = 0.3081
])
```

- load the MNIST dataset

```
data_path = './MNIST'

training_set = datasets.MNIST(root = data_path, train= True, download=True, transform= transform)
testing_set = datasets.MNIST(root = data_path, train= False, download=True, transform= transform)
```

2. Model

- design a neural network that consists of three fully connected layers with an activation function of Sigmoid
- the activation function for the output layer is LogSoftmax

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

        # construct layers for a neural network
        self.classifier1 = nn.Sequential(
            nn.Linear(in_features=28*28, out_features=20*20),
            nn.Sigmoid(),
        )
        self.classifier2 = nn.Sequential(
            nn.Linear(in_features=20*20, out_features=10*10),
            nn.Sigmoid(),
        )
```

```
        self.classifier3 = nn.Sequential(
            nn.Linear(in_features=10*10, out_features=10),
            nn.LogSoftmax(dim=1),
        )


    def forward(self, inputs):                # [batchSize, 1, 28, 28]
        x = inputs.view(inputs.size(0), -1)    # [batchSize, 28*28]
        x = self.classifier1(x)                # [batchSize, 20*20]
        x = self.classifier2(x)                # [batchSize, 10*10]
        out = self.classifier3(x)              # [batchSize, 10]

        return out
```

3. Loss function

- the log of softmax
- the negative log likelihood loss

```
criterion = nn.NLLLoss()
```

4. Optimization

- use a stochastic gradient descent algorithm with different mini-batch sizes of 32, 64, 128
- use a constant learning rate for all the mini-batch sizes
- do not use any regularization algorithm such as dropout or weight decay
- compute the average loss and the average accuracy for all the mini-batches within each epoch

```
def init_optimizer(learning_rate_value):
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    print(device)
    classifier = classification().to(device)
    optimizer = torch.optim.SGD(classifier.parameters(), lr=learning_rate_value)

    return device, classifier, optimizer
```

5. Train

```
def train(model, batch_size, optimizer, criterion):
    model.train()
    train_accuracy = 0.0
    train_loss = 0.0
    total = 0
    train_loader = torch.utils.data.DataLoader(dataset=training_set, batch_size=batch_size, shuffle=True)

    for train_img, train_label in train_loader:
        train_img, train_label = train_img.to(device), train_label.to(device)

        optimizer.zero_grad()
        train_output = model(train_img)
        loss = criterion(train_output, train_label)
        loss.backward()
        optimizer.step()

        train_loss += loss
        _, argmax = torch.max(train_output, 1)
```

```
        total += train_label.size(0)
        train_accuracy += (train_label == argmax).sum().item()

    print("Training Loss: {:.4f} ".format(train_loss/len(train_loader)),
          "Train Accuracy: {:.4f}".format(train_accuracy/total))

    return train_loss/len(train_loader), train_accuracy/total


def test(model, batch_size, optimizer, criterion):
    model.eval()
    total = 0
    test_loss = 0.0
    test_accuracy = 0.0
    test_loader = torch.utils.data.DataLoader(dataset=testing_set, batch_size=batch_size, shuffle=True)
    with torch.no_grad():
        for test_img, test_label in test_loader:
            test_img, test_label = test_img.to(device), test_label.to(device)
            test_output = model(test_img)
            test_loss += criterion(test_output, test_label)

            _, argmax = torch.max(test_output, 1)
            total += test_label.size(0)
            test_accuracy += (test_label == argmax).sum().item()

    print("Test Loss: {:.4f} ".format(test_loss/len(test_loader)),
          "Test Accuracy: {:.4f}".format(test_accuracy / total))

    return test_loss/len(test_loader), test_accuracy /total
```

```
def run_epoch(model, batch_size, optimizer, criterion):
    train_loss_list, train_acc_list, test_loss_list, test_acc_list = [], [], [], []
    for epoch in range(epochs):
        print("Epoch: {}/{} : ".format(epoch+1, epochs))
        train_loss, train_acc = train(model, batch_size, optimizer, criterion)
        test_loss, test_acc = test(model, batch_size, optimizer, criterion)

        train_loss_list.append(train_loss)
        train_acc_list.append(train_acc)
        test_loss_list.append(test_loss)
        test_acc_list.append(test_acc)

    return train_loss_list, test_loss_list, train_acc_list, test_acc_list
```

6. Visualization

```
def draw_graph(idx, train_data, test_data, batch_size):
  fig = plt.figure(figsize=(8,8))
  # plot the loss curve
  if (idx == 0):
    train_label = 'train loss'
    test_label = 'test loss'
    title = 'loss (Batch size = '+str(batch_size)+')'
    legend_loc = 'upper right'
  # plot the accuracy curve
  elif (idx == 1):
    train_label = 'train accuracy'
    test_label = 'test accuracy'
    title = 'accuracy (Batch size = '+str(batch_size)+')'
    legend_loc = 'lower right'

  plt.plot(np.array(range(epochs)), train_data, c = 'r', label = train_label)
```

```
    plt.plot(np.array(range(epochs)), test_data, c = 'b', label = test_label)
    plt.legend(loc = legend_loc)
    plt.title(title)
    plt.show()

    return fig
```

7. Start Learning

Init learning value

```
train_loss_result, test_loss_result, train_acc_result, test_acc_result= [], [], [], []
```

```
def final_result(train_loss_list, test_loss_list, train_acc_list, test_acc_list):
  train_loss_result.append(train_loss_list[-1])
  test_loss_result.append(test_loss_list[-1])
  train_acc_result.append(train_acc_list[-1])
  test_acc_result.append(test_acc_list[-1])
```

```
epochs = 70
lr = 0.01
```

Learning All

```
# mini-batch size = 32
device, classifier, optimizer = init_optimizer(lr)
train_loss_list, test_loss_list, train_acc_list, test_acc_list = run_epoch(classifier, 32, optimizer, criterion)
final_result(train_loss_list, test_loss_list, train_acc_list, test_acc_list)
fig_1 = draw_graph(0, train_loss_list, test_loss_list, 32)
fig_1.savefig('loss curve (Batch size =32).png')
fig_2 = draw_graph(1, train_acc_list, test_acc_list, 32)
fig_2.savefig('accuracy curve (Batch size =32).png')

# mini-batch size = 64
device, classifier, optimizer = init_optimizer(lr)
train_loss_list, test_loss_list, train_acc_list, test_acc_list = run_epoch(classifier, 64, optimizer, criterion)
final_result(train_loss_list, test_loss_list, train_acc_list, test_acc_list)
fig_3 = draw_graph(0, train_loss_list, test_loss_list, 64)
fig_3.savefig('loss curve (Batch size = 64).png')
fig_4 = draw_graph(1, train_acc_list, test_acc_list, 64)
fig_4.savefig('accuracy curve (Batch size = 64).png')

# mini-batch size = 128
device, classifier, optimizer = init_optimizer(lr)
train_loss_list, test_loss_list, train_acc_list, test_acc_list = run_epoch(classifier, 128, optimizer, criterion)
final_result(train_loss_list, test_loss_list, train_acc_list, test_acc_list)
fig_5 = draw_graph(0, train_loss_list, test_loss_list, 128)
fig_5.savefig('loss curve (Batch size = 128).png')
fig_6 = draw_graph(1, train_acc_list, test_acc_list, 128)
fig_6.savefig('accuracy curve (Batch size = 128).png')
```

Learning Each mini-batch

```
# mini-batch size = 32
device, classifier, optimizer = init_optimizer(lr)
```

```
device, classifier, optimizer = init_optimizer(lr)
train_loss_list, test_loss_list, train_acc_list, test_acc_list = run_epoch(classifier, 32, optimizer, criterion)
```

```
final_result(train_loss_list, test_loss_list, train_acc_list, test_acc_list)
fig_1 = draw_graph(0, train_loss_list, test_loss_list, 32)
fig_1.savefig('loss curve (Batch size =32).png')
fig_2 = draw_graph(1, train_acc_list, test_acc_list, 32)
fig_2.savefig('accuracy curve (Batch size =32).png')
```

```
# mini-batch size = 64
device, classifier, optimizer= init_optimizer(lr)
train_loss_list, test_loss_list, train_acc_list, test_acc_list = run_epoch(classifier, 64, optimizer, criterion)
```

```
final_result(train_loss_list, test_loss_list, train_acc_list, test_acc_list)
fig_3 = draw_graph(0, train_loss_list, test_loss_list, 64)
fig_3.savefig('loss curve (Batch size = 64).png')
fig_4 = draw_graph(1, train_acc_list, test_acc_list, 64)
fig_4.savefig('accuracy curve (Batch size = 64).png')
```

```
# mini-batch size =
device, classifier, optimizer = init_optimizer(lr)
train_loss_list, test_loss_list, train_acc_list, test_acc_list = run_epoch(classifier, 128, optimizer, criterion)
```



```
final_result(train_loss_list, test_loss_list, train_acc_list, test_acc_list)
fig_5 = draw_graph(0, train_loss_list, test_loss_list, 128)
fig_5.savefig('loss curve (Batch size = 128).png')
fig_6 = draw_graph(1, train_acc_list, test_acc_list, 128)
fig_6.savefig('accuracy curve (Batch size = 128).png')
```

Print learning results as a table using pandas

```
result_loss = pd.DataFrame({'32':[train_loss_result[0].item(), test_loss_result[0].item()],
                            '64':[train_loss_result[1].item(), test_loss_result[1].item()],
                            '128':[train_loss_result[2].item(), test_loss_result[2].item()]}, index = ['training l
result_loss
```

	32	64	128
training loss	0.074559	0.151474	0.249550
testing loss	0.093673	0.158547	0.250557

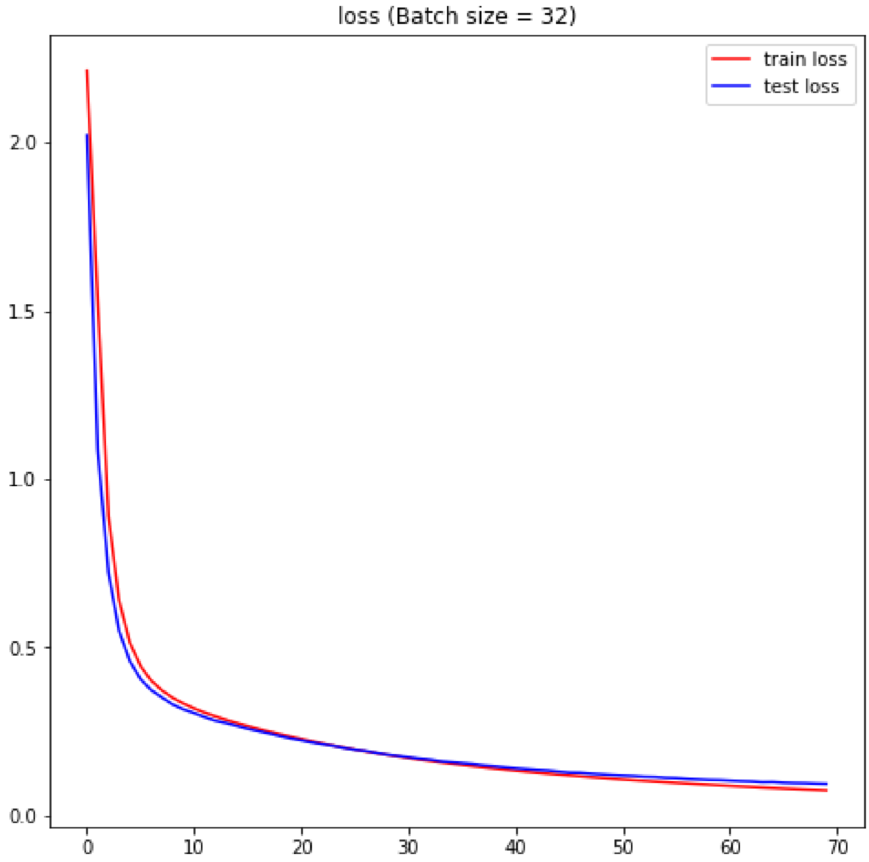
```
result_acc = pd.DataFrame({'32':[train_acc_result[0], test_acc_result[0]],
                           '64':[train_acc_result[1], test_acc_result[1]],
                           '128':[train_acc_result[2], test_acc_result[2]]}, index = ['training accuracy','testin
result_acc
```

	32	64	128
training accuracy	0.98025	0.957133	0.92815
testing accuracy	0.97160	0.955300	0.92840

Output

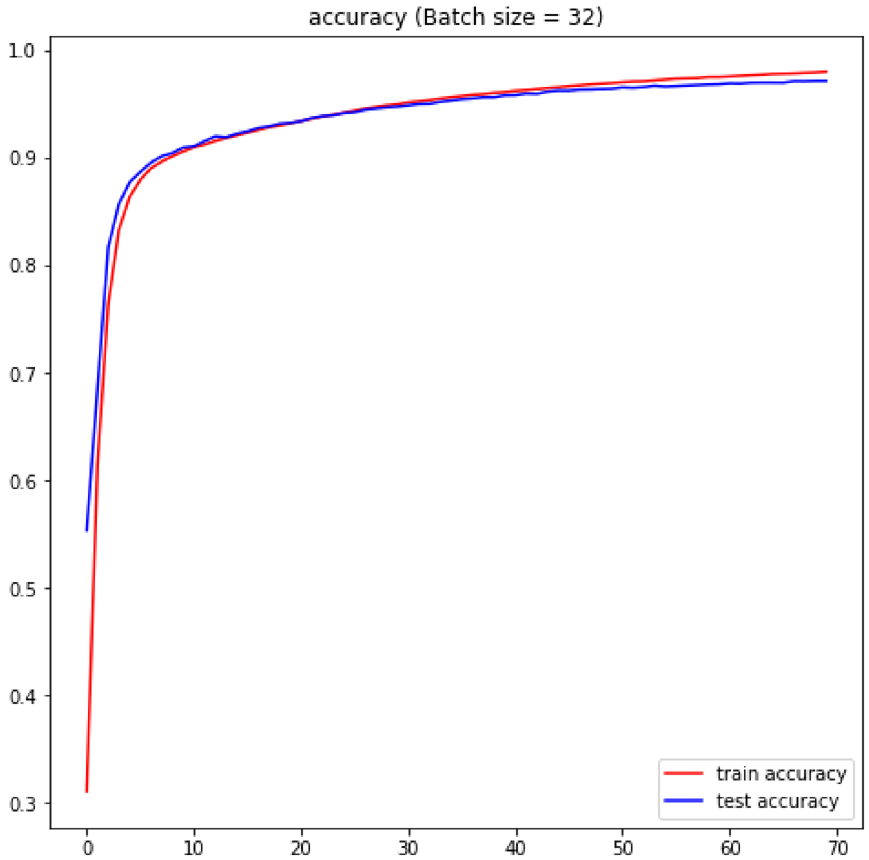
1. Plot the training and testing losses with a batch size of 32 [4pt]

```
fig_1
```



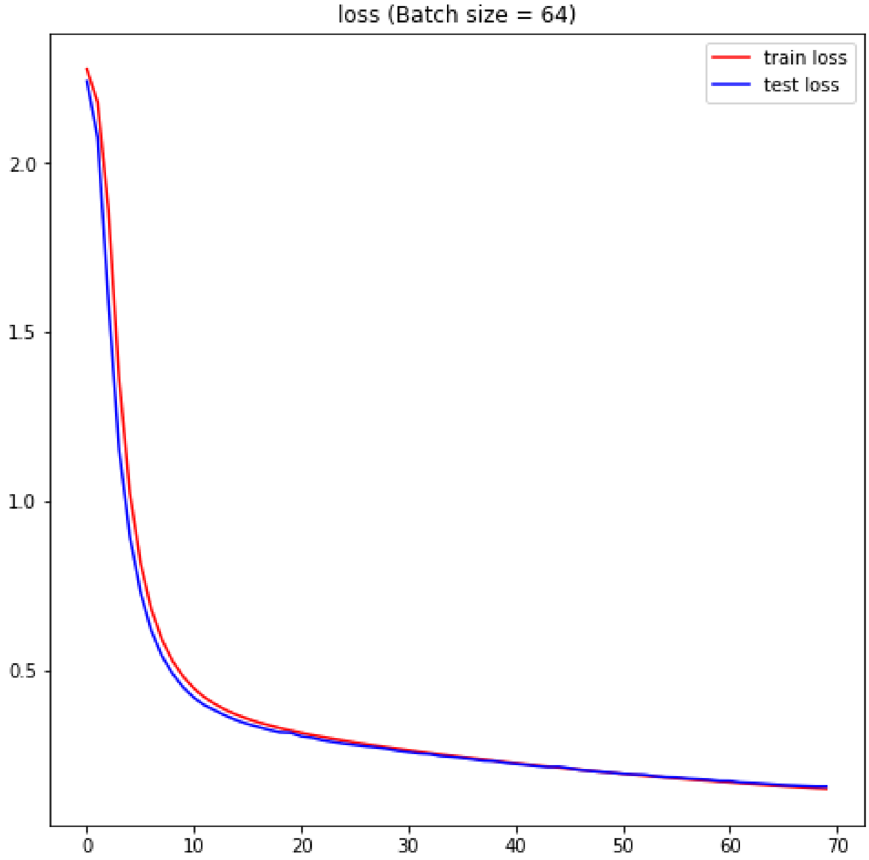
2. Plot the training and testing accuracies with a batch size of 32 [4pt]

```
fig_2
```



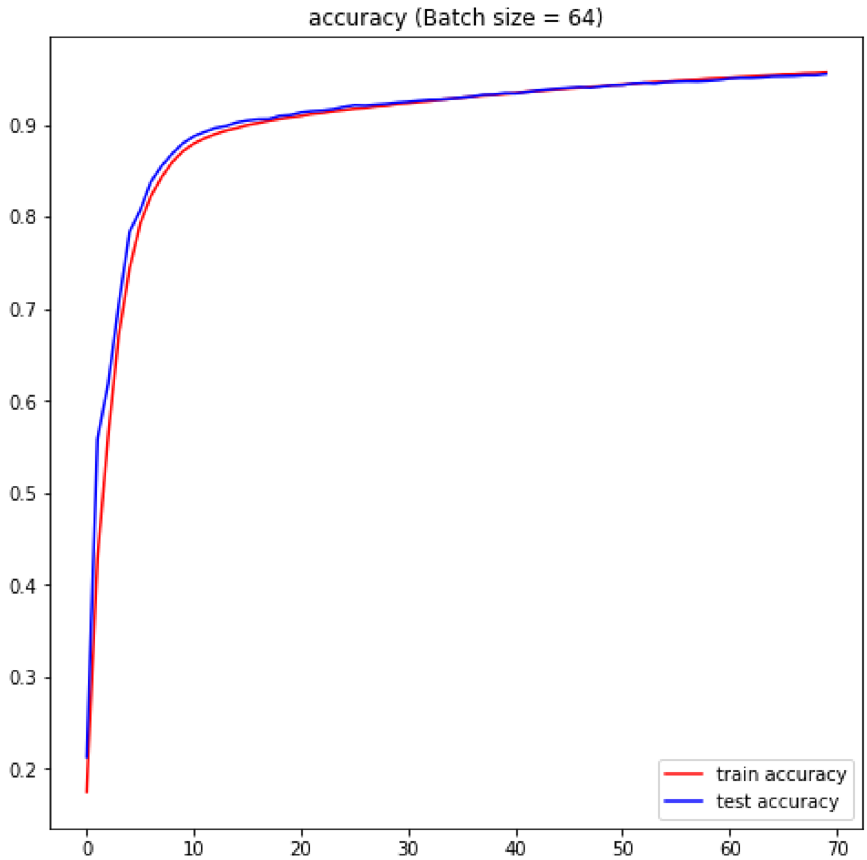
3. Plot the training and testing losses with a batch size of 64 [4pt]

```
fig_3
```



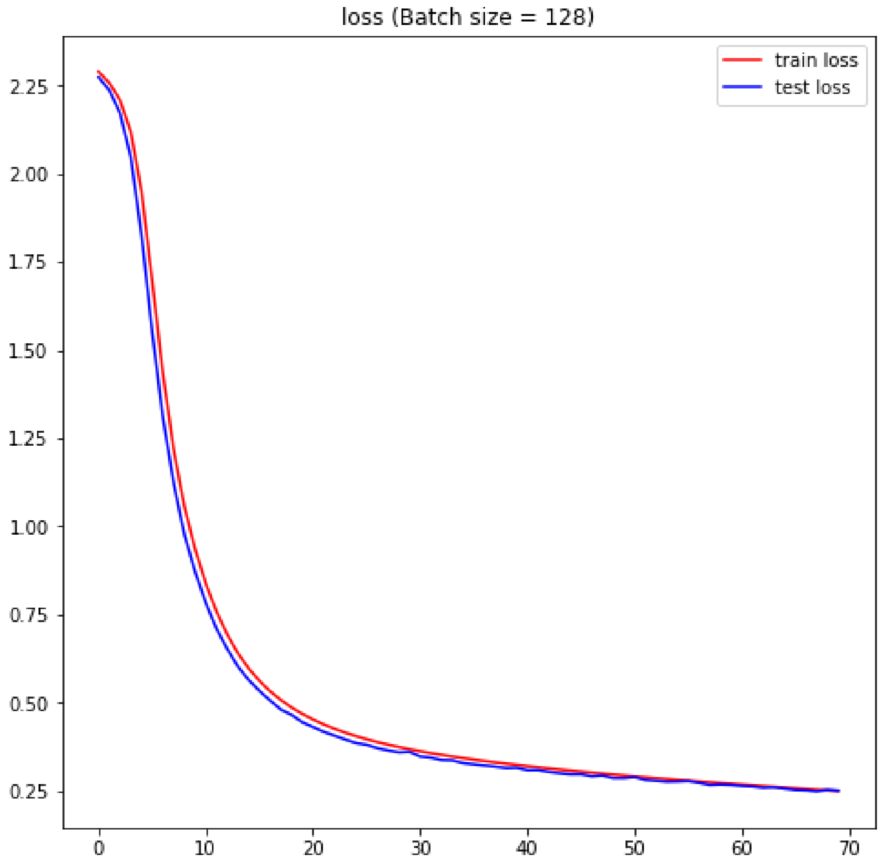
4. Plot the training and testing accuracies with a batch size of 64 [4pt]

```
fig_4
```



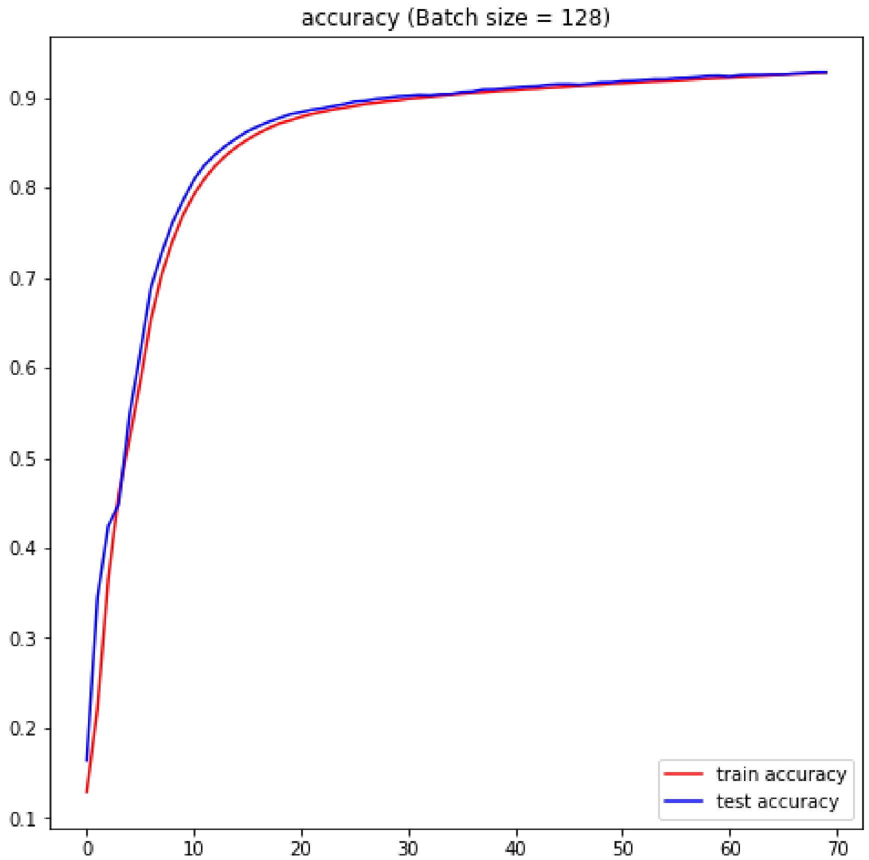
5. Plot the training and testing losses with a batch size of 128 [4pt]

```
fig_5
```



6. Plot the training and testing accuracies with a batch size of 128 [4pt]

```
fig_6
```



7. Print the loss at convergence with different mini-batch sizes [3pt]

```
result_loss
```

	32	64	128
training loss	0.074559	0.151474	0.249550
testing loss	0.093673	0.158547	0.250557

8. Print the accuracy at convergence with different mini-batch sizes [3pt]

```
result_acc
```

	32	64	128
training accuracy	0.98025	0.957133	0.92815
testing accuracy	0.97160	0.955300	0.92840