group-project-VGG-1

July 13, 2019

In [5]:
```
import torch
import pandas as pd
import numpy as np
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
from torch.utils.data.sampler import SubsetRandomSampler

import matplotlib.pyplot as plt

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision

device = torch.device('cuda:0')
```

In [6]:
```
class PUBG_imglike_dataset(Dataset):
    def __init__(self, csv_file, transform=None):
        self.frame = pd.read_csv(csv_file)
        self.transform = transform

    def __len__(self):
        return len(self.frame)

    def __getitem__(self, idx):
        def transfrom2imglike(input):
            output = np.zeros((3,32,32))
            temp = np.array(input)
            for x in range(23):
                for y in range(23):
                    if(x == y):
                        output[0][x][y] = temp[x]
                        output[1][x][y] = temp[x]
                        output[2][x][y] = temp[x]
            return output
        # get one line in csv
        player_id = self.frame.iloc [idx, 0]
```

```
            player_stats = self.frame.iloc [idx, [x for x in range(3, 27) if x != 15]].value
            player_stats = torch.tensor(transfrom2imglike(player_stats))
            win_place_perc = torch.tensor(self.frame.iloc [idx, 28])
            if self.transform:
                player_stats = self.transform(player_stats)
            sample = {
                "player_id": player_id,
                "player_stats": player_stats,
                "win_place_perc": win_place_perc
            }
            return sample

    def get_dataset(csv_file, train_dataset_size_ratio, batch_size):
        dataset = PUBG_imglike_dataset(csv_file)
        #  `torch.utils.data.random_split` meets server problem and lead to CRASH
        # see also:
        # - a denied fix PR for this problem: https://github.com/pytorch/pytorch/pull/9237
        #train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, t
        dataset_size = len(dataset)
        indices = list(range(dataset_size))
        split = int(np.floor((1-train_dataset_size_ratio) * dataset_size))
        train_indices, val_indices = indices[split:], indices[:split]

        train_sampler = SubsetRandomSampler(train_indices)
        valid_sampler = SubsetRandomSampler(val_indices)

        train_loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, sampler=t
        test_loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, sampler=va
        print("load dataset: train dataset: {}, test dataset: {}.".format(len(train_loader)*
        return (train_loader, test_loader)

    # load dataset
    csv_file = 'train_small.csv'
    train_dataset_size_ratio = 0.9
    batch_size = 128
    train_loader, test_loader = get_dataset(csv_file, train_dataset_size_ratio, batch_size)
```

```
load dataset: train dataset: 1152, test dataset: 128.
```

In [7]:
```
def show_curve(ys, title):
    x = np.array(range(len(ys)))
    y = np.array(ys)
    plt.plot(x, y, c='b')
    plt.axis()
    plt.title('{} curve'.format(title))
    plt.xlabel('epoch')
    plt.ylabel('{}'.format(title))
    plt.show()
```

```
In [8]: def train(model, train_loader, loss_func, optimizer, device):
            total_loss = 0
            # train the model using minibatch
            for i, data in enumerate(train_loader):
                stats, prec = data['player_stats'], data['win_place_perc']
                stats, prec = stats.to(torch.float32).to(device), prec.to(device)

                # forward
                outputs = model(stats)
                loss = loss_func(outputs, prec)

                # backward and optimize
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

                total_loss += loss.item()

                #if (i + 1) % 10 == 0:
                #    print ("Step [{}/{}] Train Loss: {:.4f}".format(i+1, len(train_loader), los
            #print ("Train Loss: {:.4f}".format(loss.item()))
            return total_loss / len(train_loader)

        def evaluate(model, val_loader, device):

            model.eval()
            with torch.no_grad():
                loss = 0
                total = 0

                for i, data in enumerate(val_loader):
                    stats, prec = data['player_stats'], data['win_place_perc']
                    stats, prec = stats.to(torch.float32).to(device), prec.to(device)

                    outputs = model(stats)


                    loss += (torch.abs(torch.t(outputs) - prec)).sum()
                    total += prec.size(0)


                accuracy = loss / total
                #print('Test Loss: {:.4f}'.format(accuracy))
                return accuracy

        def fit(model, num_epochs, optimizer, device):
            loss_func = nn.MSELoss()
            model.to(device)
```

```
    if device == torch.device('cuda'):
        model = torch.nn.DataParallel(model)
        cudnn.benchmark = True
    loss_func.to(device)
    losses = []
    accs = []

    for epoch in range(num_epochs):


        # train step
        loss = train(model, train_loader, loss_func, optimizer, device)
        losses.append(loss)

        # evaluate step
        accuracy = evaluate(model, test_loader, device)
        accs.append(accuracy)

        # print loss
        if (epoch+1) % 10 == 0:
            print("Epoch {}/{}".format(epoch+1, num_epochs))
            print("Train Loss: {:.4f}".format(loss))
            print('Test Loss: {:.4f}'.format(accuracy))

    show_curve(losses, "train loss")
    show_curve(accs, "test loss")
```

In [9]:
```
from torch import nn
import math

class VGG(nn.Module):
    def __init__(self, cfg, num_classes=10):
        super(VGG, self).__init__()
        self.features = self._make_layers(cfg)
        # linear layer
        self.classifier = nn.Linear(512, num_classes)

    def forward(self, x):
        out = self.features(x)
        out = out.view(out.size(0), -1)
        out = self.classifier(out)
        return out

    def _make_layers(self, cfg):
        """
        cfg: a list define layers this layer contains
            'M': MaxPool, number: Conv2d(out_channels=number) -> BN -> ReLU
        """
```

```
                layers = []
                in_channels = 3
                for x in cfg:
                    if x == 'M':
                        layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
                    else:
                        layers += [nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
                                   nn.BatchNorm2d(x),
                                   nn.ReLU(inplace=True)]
                        in_channels = x
                layers += [nn.AvgPool2d(kernel_size=1, stride=1)]
                return nn.Sequential(*layers)

In [10]: cfg = {
             'VGG11': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
             'VGG13': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
             'VGG16': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512,
             'VGG19': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512,
         }
         vggnet11 = VGG(cfg['VGG11'], 1)
         print(vggnet11)

VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU(inplace)
    (11): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): ReLU(inplace)
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (16): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (17): ReLU(inplace)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (20): ReLU(inplace)
    (21): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (22): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (23): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

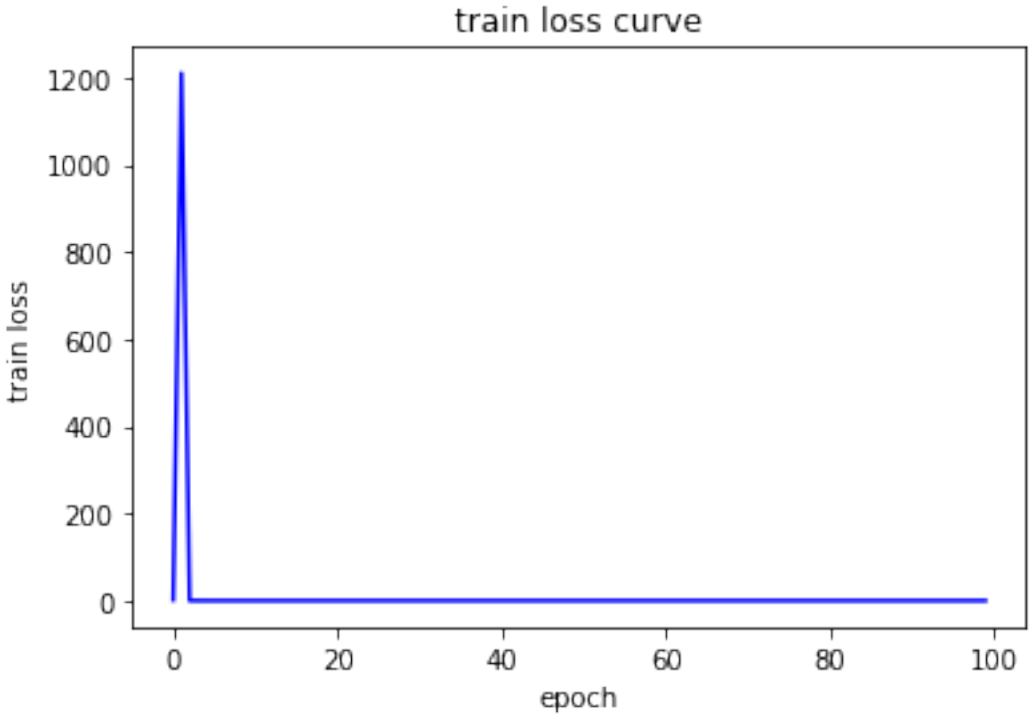
```
    (24): ReLU(inplace)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (27): ReLU(inplace)
    (28): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (29): AvgPool2d(kernel_size=1, stride=1, padding=0)
  )
  (classifier): Linear(in_features=512, out_features=1, bias=True)
)
```

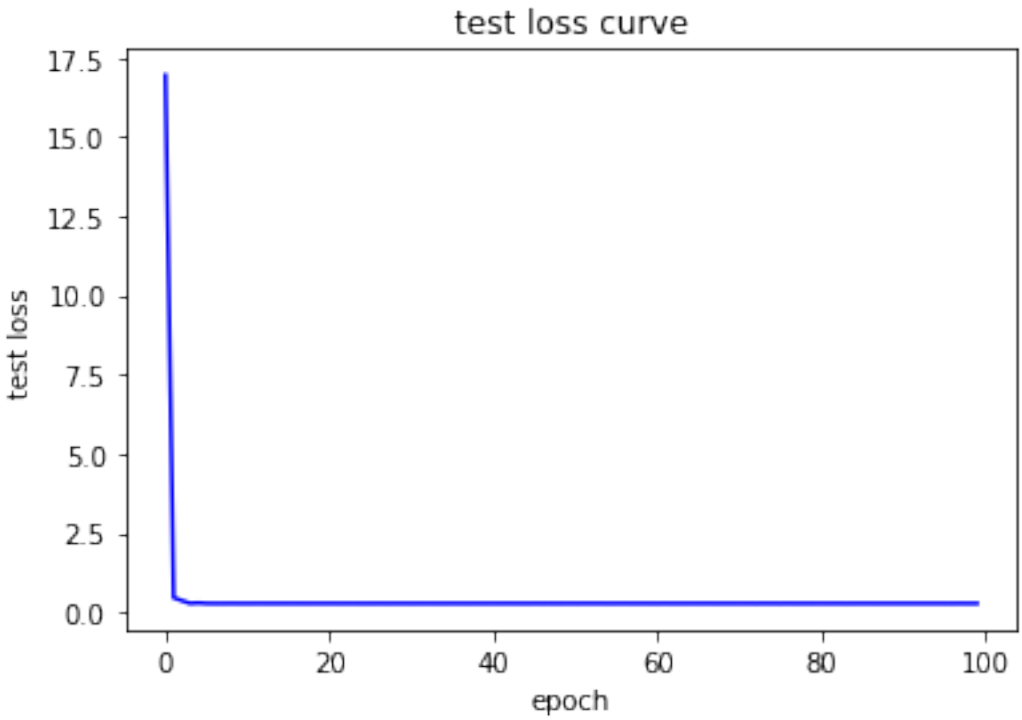
```
In [7]: # training setting
        # hyper parameters
        num_epochs = 100
        lr = 0.01
        image_size = 32

        # Device configuration, cpu, cuda:0/1/2/3 available
        device = torch.device('cuda:0')

        optimizer = torch.optim.Adam(vggnet11.parameters(), lr=lr)
```

```
In [8]: fit(vggnet11, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0971
Test Loss: 0.2787
Epoch 20/100
Train Loss: 0.0966
Test Loss: 0.2788
Epoch 30/100
Train Loss: 0.0913
Test Loss: 0.2793
Epoch 40/100
Train Loss: 0.0952
Test Loss: 0.2786
Epoch 50/100
Train Loss: 0.0876
Test Loss: 0.2788
Epoch 60/100
Train Loss: 0.0914
Test Loss: 0.2787
Epoch 70/100
Train Loss: 0.0941
Test Loss: 0.2789
Epoch 80/100
Train Loss: 0.0922
Test Loss: 0.2786
```

```
Epoch 90/100
Train Loss: 0.0943
Test Loss: 0.2785
Epoch 100/100
Train Loss: 0.0923
Test Loss: 0.2792
```

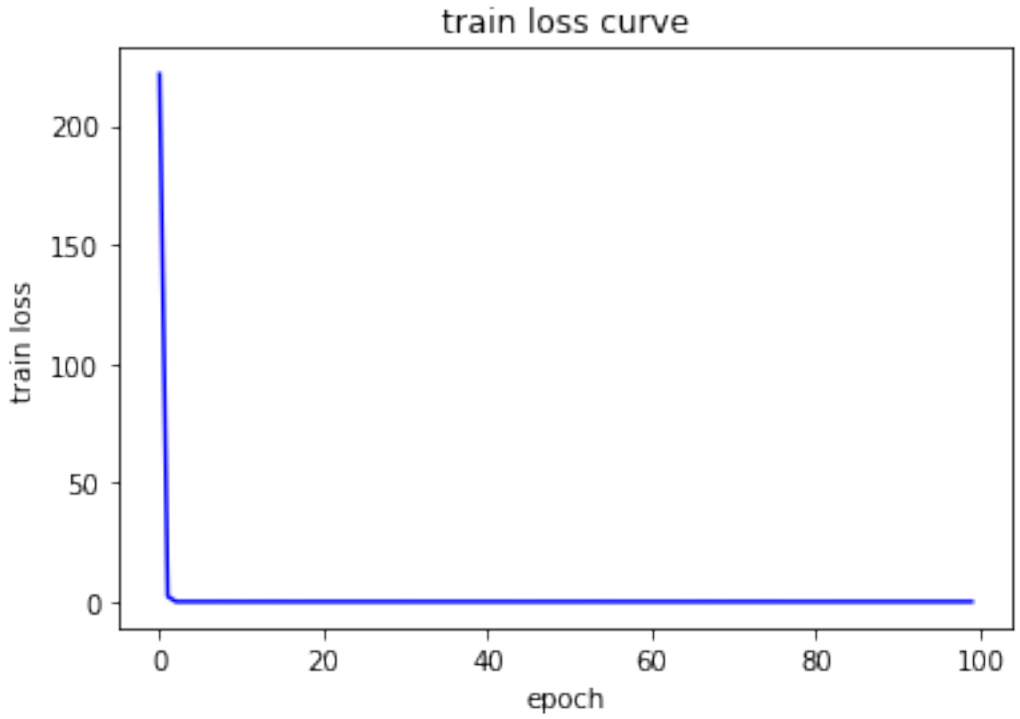


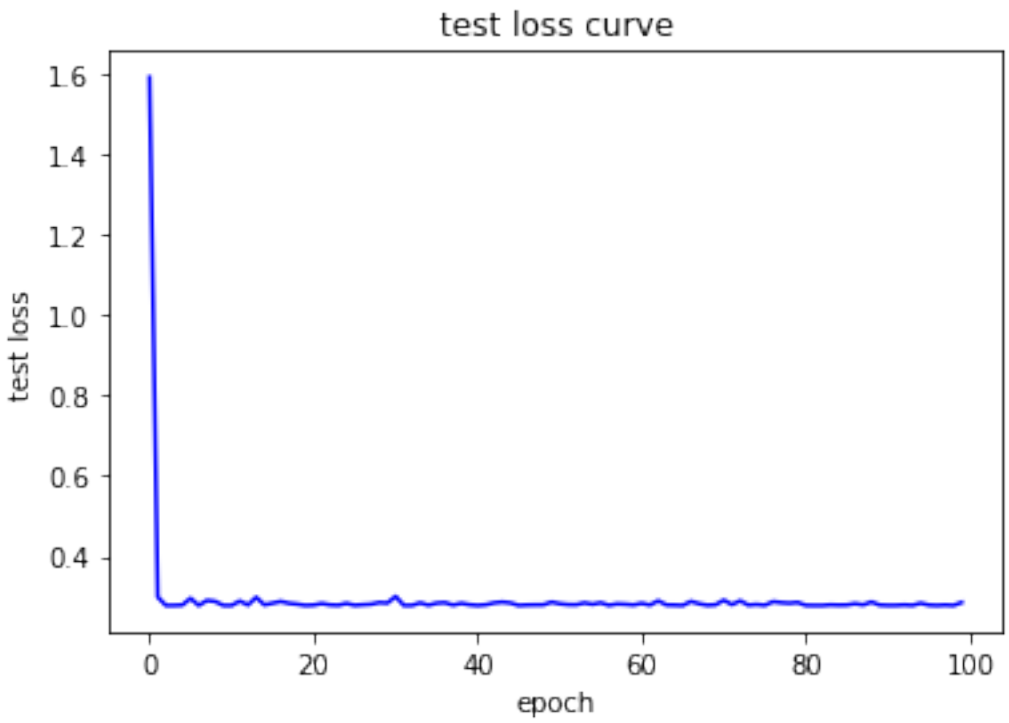


In [12]:
```
# training setting
# hyper parameters
num_epochs = 100
lr = 0.01
image_size = 32

# Device configuration, cpu, cuda:0/1/2/3 available
device = torch.device('cuda:1')

optimizer = torch.optim.SGD(vggnet11.parameters(), lr=lr)
fit(vggnet11, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0923
Test Loss: 0.2777
Epoch 20/100
Train Loss: 0.0932
Test Loss: 0.2777
Epoch 30/100
Train Loss: 0.0921
Test Loss: 0.2837
Epoch 40/100
Train Loss: 0.0948
Test Loss: 0.2795
```

```
Epoch 50/100
Train Loss: 0.0944
Test Loss: 0.2861
Epoch 60/100
Train Loss: 0.0932
Test Loss: 0.2789
Epoch 70/100
Train Loss: 0.0902
Test Loss: 0.2793
Epoch 80/100
Train Loss: 0.0927
Test Loss: 0.2852
Epoch 90/100
Train Loss: 0.0947
Test Loss: 0.2787
Epoch 100/100
Train Loss: 0.0919
Test Loss: 0.2852
```

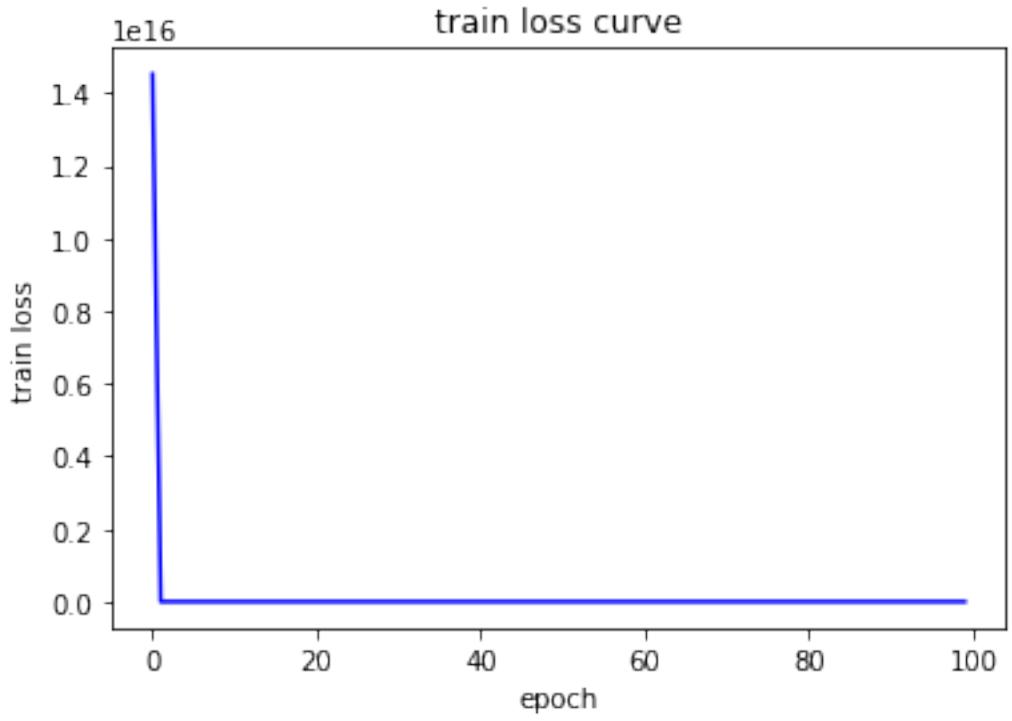


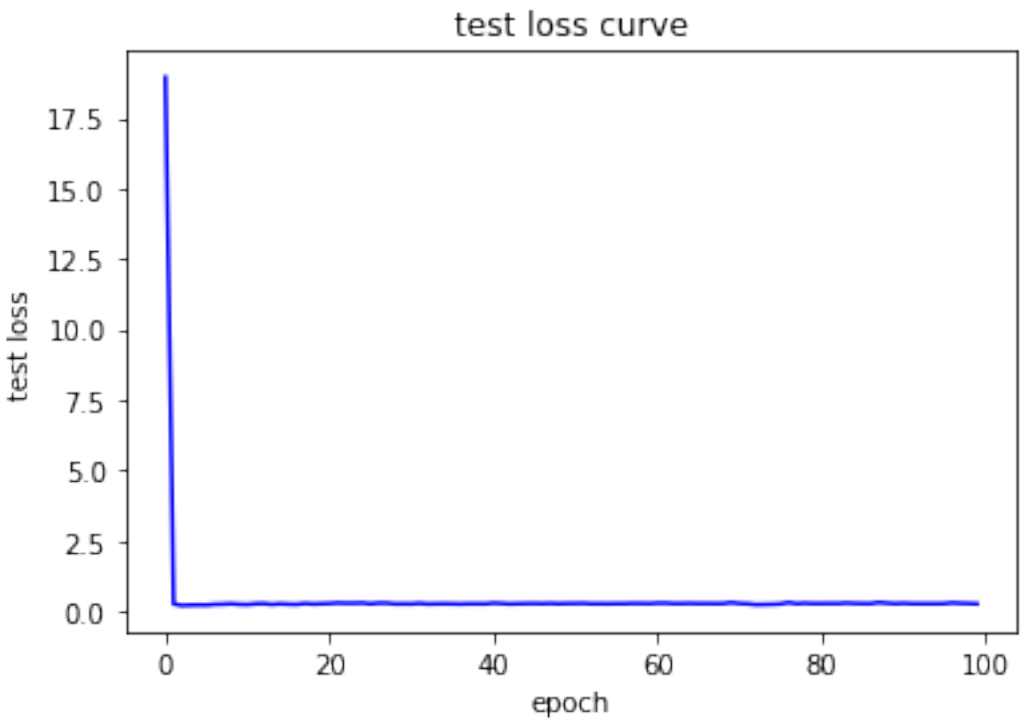


In [13]:
```
# training setting
# hyper parameters
num_epochs = 100
lr = 0.01
image_size = 32

# Device configuration, cpu, cuda:0/1/2/3 available
device = torch.device('cuda:1')

optimizer = torch.optim.RMSprop(vggnet11.parameters(), lr=lr)
fit(vggnet11, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0999
Test Loss: 0.2482
Epoch 20/100
Train Loss: 0.0941
Test Loss: 0.2695
Epoch 30/100
Train Loss: 0.0919
Test Loss: 0.2664
Epoch 40/100
Train Loss: 0.0923
Test Loss: 0.2649
```

```
Epoch 50/100
Train Loss: 0.0984
Test Loss: 0.2782
Epoch 60/100
Train Loss: 0.1055
Test Loss: 0.2747
Epoch 70/100
Train Loss: 0.1032
Test Loss: 0.3104
Epoch 80/100
Train Loss: 0.0968
Test Loss: 0.2753
Epoch 90/100
Train Loss: 0.0998
Test Loss: 0.2723
Epoch 100/100
Train Loss: 0.0926
Test Loss: 0.2662
```



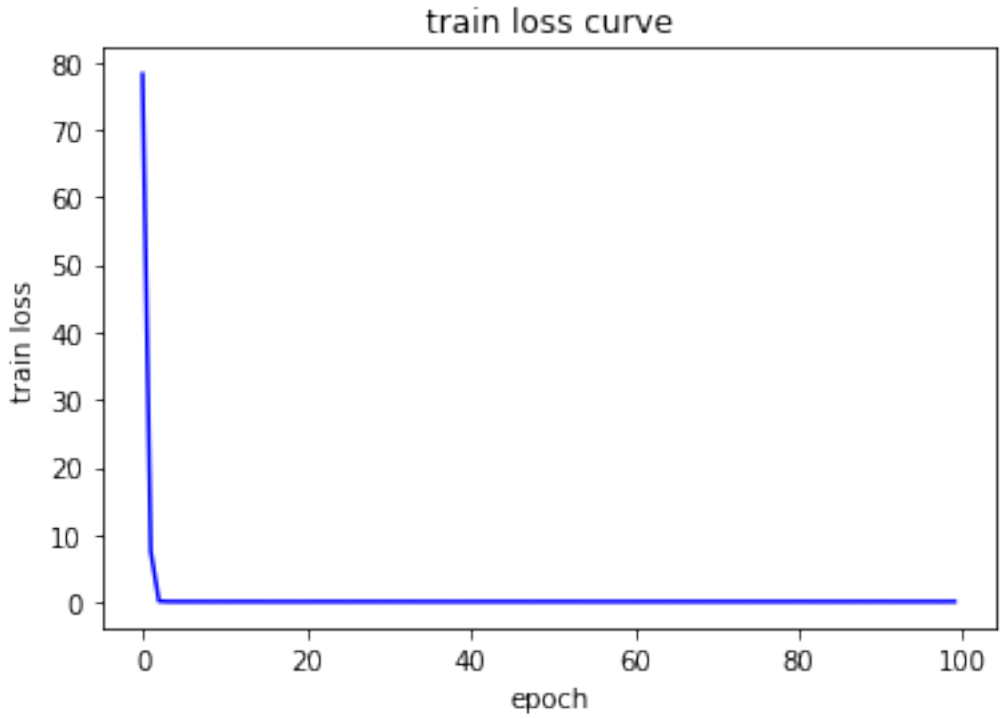


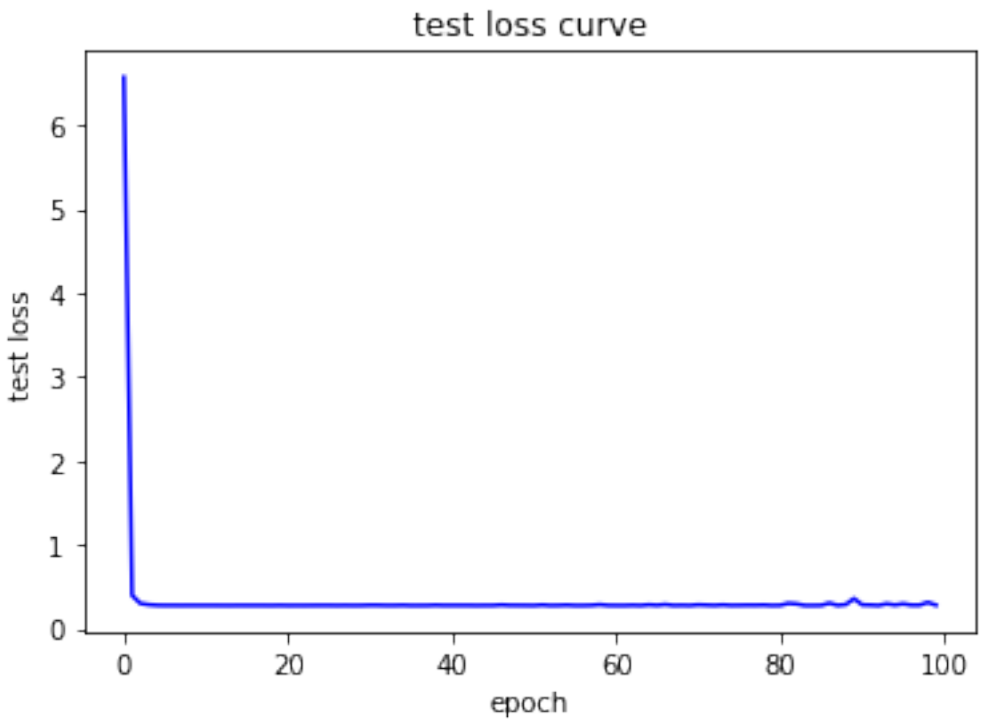
RMSprop [U+6548] [U+679C] [U+6700] [U+597D]

```
In [15]: vggnet13 = VGG(cfg['VGG13'], 1)
         # training setting
         # hyper parameters
         num_epochs = 100
         lr = 0.01
         image_size = 32

         # Device configuration, cpu, cuda:0/1/2/3 available
         device = torch.device('cuda:1')

         optimizer = torch.optim.RMSprop(vggnet13.parameters(), lr=lr)
         fit(vggnet13, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0912
Test Loss: 0.2789
Epoch 20/100
Train Loss: 0.0890
Test Loss: 0.2794
Epoch 30/100
Train Loss: 0.0932
Test Loss: 0.2788
Epoch 40/100
```

```
Train Loss: 0.0883
Test Loss: 0.2793
Epoch 50/100
Train Loss: 0.0953
Test Loss: 0.2793
Epoch 60/100
Train Loss: 0.0910
Test Loss: 0.2794
Epoch 70/100
Train Loss: 0.0949
Test Loss: 0.2784
Epoch 80/100
Train Loss: 0.0937
Test Loss: 0.2784
Epoch 90/100
Train Loss: 0.0972
Test Loss: 0.3606
Epoch 100/100
Train Loss: 0.1006
Test Loss: 0.2816
```

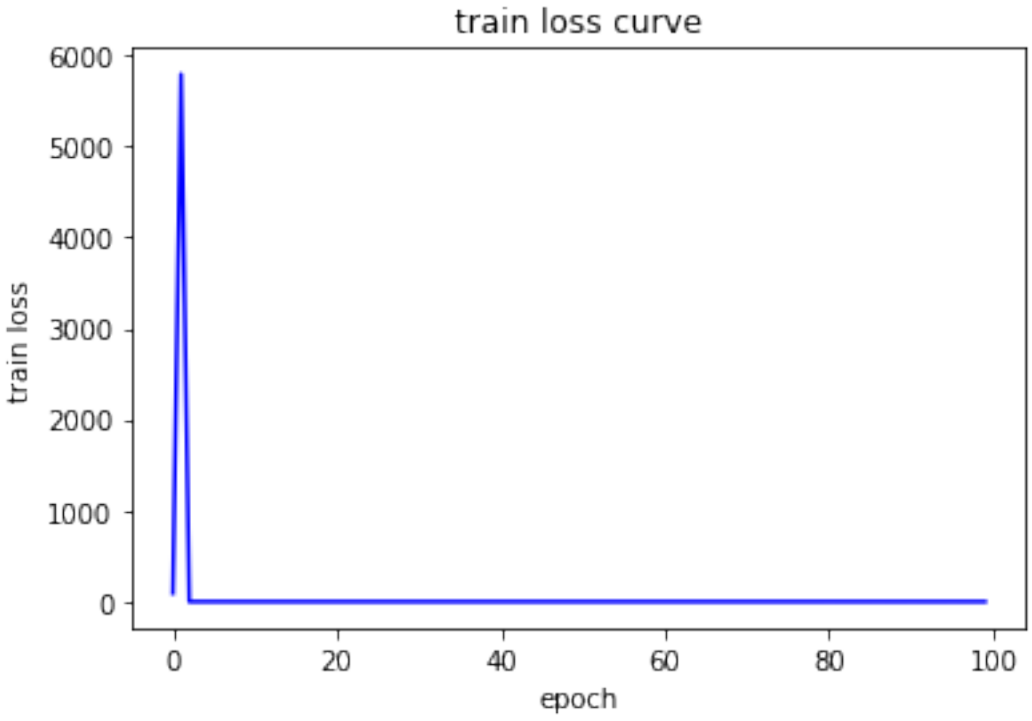


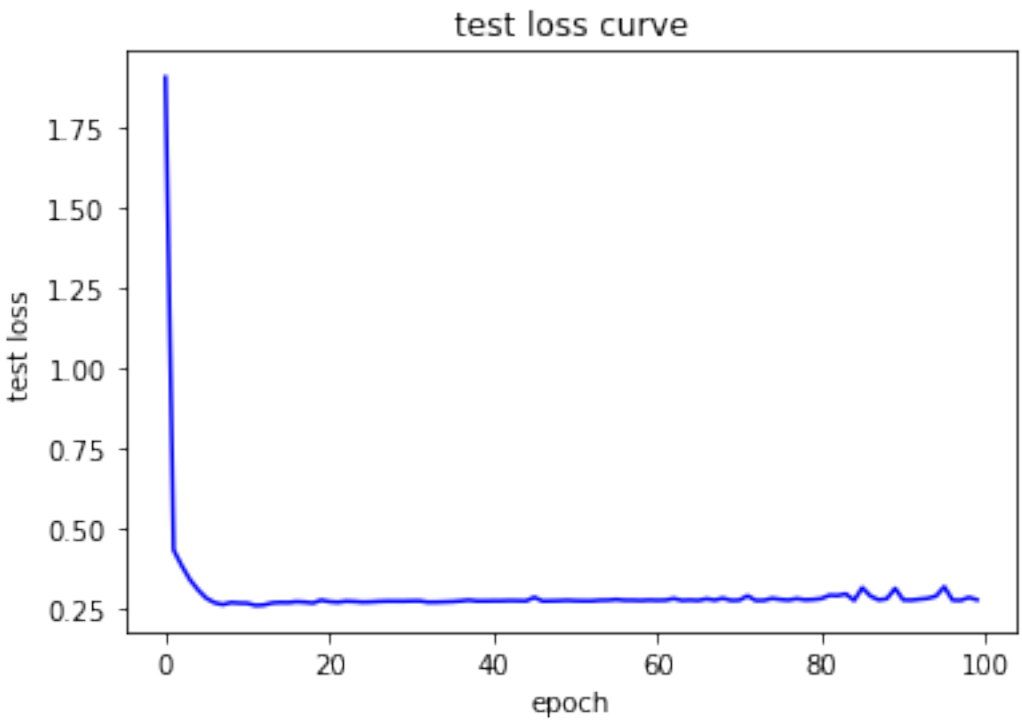



```
In [16]: vggnet16 = VGG(cfg['VGG16'], 1)
         # training setting
         # hyper parameters
         num_epochs = 100
         lr = 0.01
         image_size = 32

         # Device configuration, cpu, cuda:0/1/2/3 available
         device = torch.device('cuda:1')

         optimizer = torch.optim.RMSprop(vggnet16.parameters(), lr=lr)
         fit(vggnet16, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0912
Test Loss: 0.2699
Epoch 20/100
Train Loss: 0.0937
Test Loss: 0.2789
Epoch 30/100
Train Loss: 0.0902
Test Loss: 0.2757
Epoch 40/100
Train Loss: 0.0900
```

```
Test Loss: 0.2763
Epoch 50/100
Train Loss: 0.0944
Test Loss: 0.2783
Epoch 60/100
Train Loss: 0.0925
Test Loss: 0.2791
Epoch 70/100
Train Loss: 0.0879
Test Loss: 0.2780
Epoch 80/100
Train Loss: 0.0918
Test Loss: 0.2810
Epoch 90/100
Train Loss: 0.0973
Test Loss: 0.3156
Epoch 100/100
Train Loss: 0.0925
Test Loss: 0.2799
```

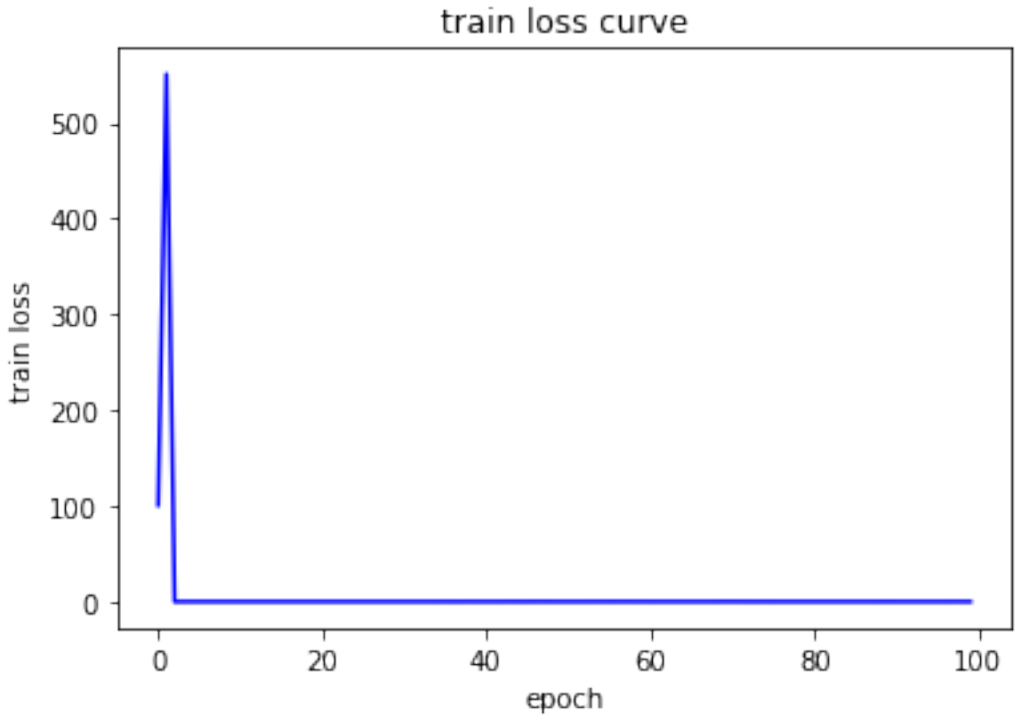


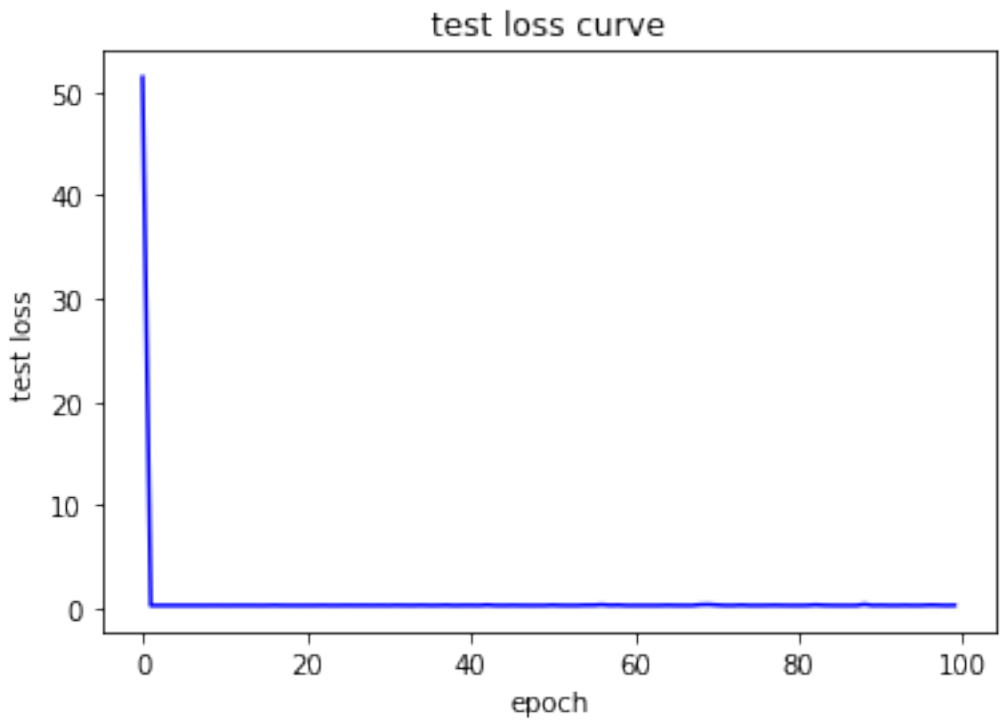



```
In [17]: vggnet19 = VGG(cfg['VGG19'], 1)
         # training setting
         # hyper parameters
         num_epochs = 100
         lr = 0.01
         image_size = 32

         # Device configuration, cpu, cuda:0/1/2/3 available
         device = torch.device('cuda:1')

         optimizer = torch.optim.RMSprop(vggnet19.parameters(), lr=lr)
         fit(vggnet19, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0925
Test Loss: 0.2824
Epoch 20/100
Train Loss: 0.0930
Test Loss: 0.2782
Epoch 30/100
Train Loss: 0.0978
Test Loss: 0.2900
Epoch 40/100
Train Loss: 0.0941
```

```
Test Loss: 0.2903
Epoch 50/100
Train Loss: 0.0954
Test Loss: 0.2797
Epoch 60/100
Train Loss: 0.1097
Test Loss: 0.2786
Epoch 70/100
Train Loss: 0.1253
Test Loss: 0.3763
Epoch 80/100
Train Loss: 0.0979
Test Loss: 0.2789
Epoch 90/100
Train Loss: 0.1141
Test Loss: 0.2840
Epoch 100/100
Train Loss: 0.0958
Test Loss: 0.2929
```





[U+5176] [U+4E2D] vgg11 [U+7684] [U+6548] [U+679C] [U+6700] [U+597D]

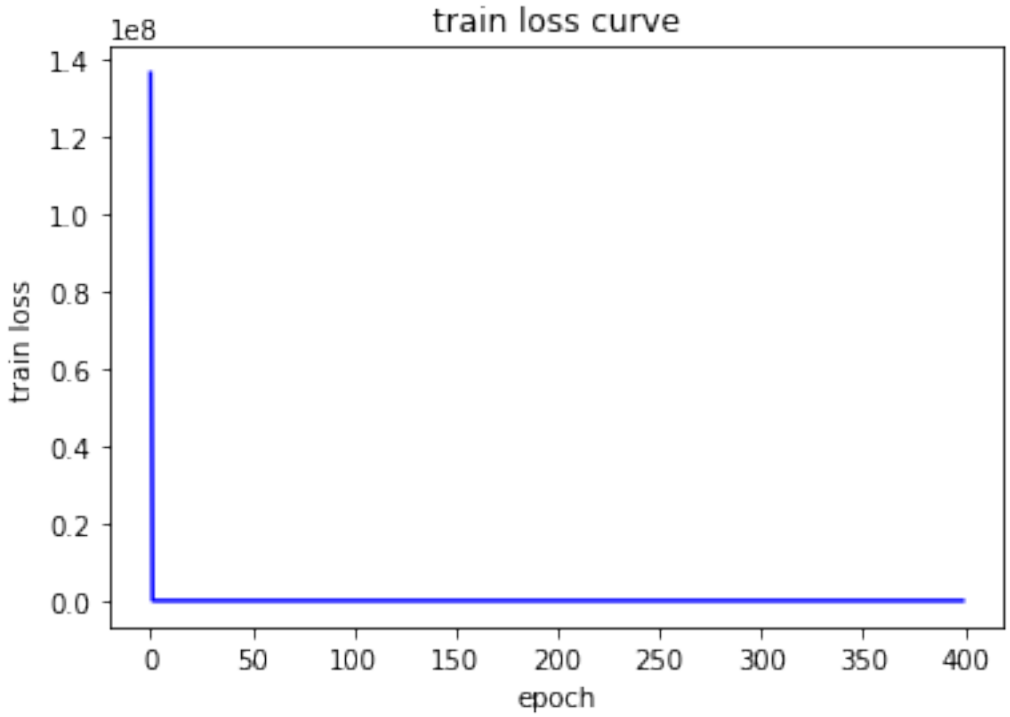
In [19]:
```
from torch.optim import lr_scheduler
def fit2(model, num_epochs, optimizer, device):
    loss_func = nn.MSELoss()
    model.to(device)
    if device == torch.device('cuda'):
        model = torch.nn.DataParallel(model)
        cudnn.benchmark = True
    loss_func.to(device)
    losses = []
    accs = []

    scheduler = lr_scheduler.StepLR(optimizer,step_size=100,gamma=0.1)

    for epoch in range(num_epochs):


        # train step
        loss = train(model, train_loader, loss_func, optimizer, device)
        losses.append(loss)

        # evaluate step
        accuracy = evaluate(model, test_loader, device)
        accs.append(accuracy)
```

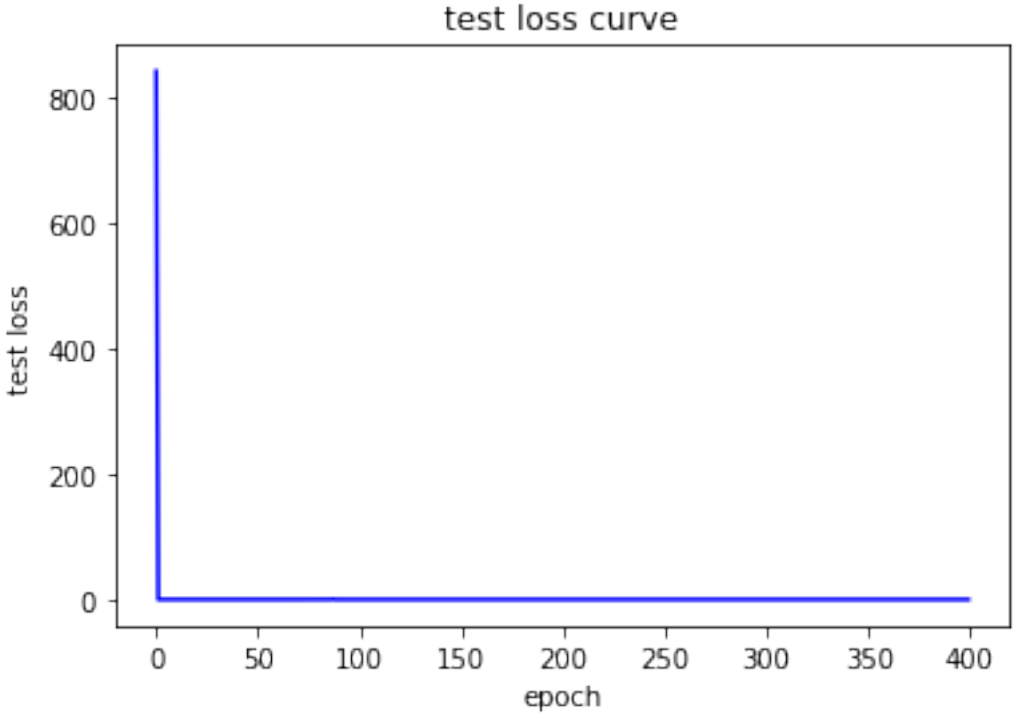
```
                # change the learning rate by scheduler
                scheduler.step()

                # print loss
                if (epoch+1) % 10 == 0:
                    print("Epoch {}/{}".format(epoch+1, num_epochs))
                    print("Train Loss: {:.4f}".format(loss))
                    print('Test Loss: {:.4f}'.format(accuracy))

            show_curve(losses, "train loss")
            show_curve(accs, "test loss")

        # training setting
        # hyper parameters
        num_epochs = 400
        lr = 0.01
        image_size = 32

        # Device configuration, cpu, cuda:0/1/2/3 available
        device = torch.device('cuda:1')

        optimizer = torch.optim.RMSprop(vggnet11.parameters(), lr=lr)
        fit2(vggnet11, num_epochs, optimizer, device)
```

```
Epoch 10/400
Train Loss: 0.1014
Test Loss: 0.3474
Epoch 20/400
Train Loss: 0.0961
Test Loss: 0.3093
Epoch 30/400
Train Loss: 0.1207
Test Loss: 0.3056
Epoch 40/400
Train Loss: 0.0977
Test Loss: 0.2883
Epoch 50/400
Train Loss: 0.1231
Test Loss: 0.2857
Epoch 60/400
Train Loss: 0.0987
Test Loss: 0.3579
Epoch 70/400
Train Loss: 0.1014
Test Loss: 0.3044
Epoch 80/400
Train Loss: 0.1113
Test Loss: 0.3369
```

```
Epoch 90/400
Train Loss: 0.0936
Test Loss: 0.2938
Epoch 100/400
Train Loss: 0.0956
Test Loss: 0.2779
Epoch 110/400
Train Loss: 0.0941
Test Loss: 0.2771
Epoch 120/400
Train Loss: 0.0923
Test Loss: 0.2778
Epoch 130/400
Train Loss: 0.0911
Test Loss: 0.2816
Epoch 140/400
Train Loss: 0.0907
Test Loss: 0.2747
Epoch 150/400
Train Loss: 0.0948
Test Loss: 0.2805
Epoch 160/400
Train Loss: 0.0921
Test Loss: 0.2764
Epoch 170/400
Train Loss: 0.0914
Test Loss: 0.2754
Epoch 180/400
Train Loss: 0.0953
Test Loss: 0.2748
Epoch 190/400
Train Loss: 0.0920
Test Loss: 0.2778
Epoch 200/400
Train Loss: 0.0937
Test Loss: 0.2823
Epoch 210/400
Train Loss: 0.0925
Test Loss: 0.2734
Epoch 220/400
Train Loss: 0.0912
Test Loss: 0.2724
Epoch 230/400
Train Loss: 0.0944
Test Loss: 0.2738
Epoch 240/400
Train Loss: 0.0925
Test Loss: 0.2762
```

```
Epoch 250/400
Train Loss: 0.0923
Test Loss: 0.2761
Epoch 260/400
Train Loss: 0.0969
Test Loss: 0.2762
Epoch 270/400
Train Loss: 0.0944
Test Loss: 0.2795
Epoch 280/400
Train Loss: 0.0947
Test Loss: 0.2771
Epoch 290/400
Train Loss: 0.0956
Test Loss: 0.2773
Epoch 300/400
Train Loss: 0.0936
Test Loss: 0.2757
Epoch 310/400
Train Loss: 0.0907
Test Loss: 0.2747
Epoch 320/400
Train Loss: 0.0976
Test Loss: 0.2753
Epoch 330/400
Train Loss: 0.0913
Test Loss: 0.2758
Epoch 340/400
Train Loss: 0.0886
Test Loss: 0.2754
Epoch 350/400
Train Loss: 0.0956
Test Loss: 0.2761
Epoch 360/400
Train Loss: 0.0930
Test Loss: 0.2754
Epoch 370/400
Train Loss: 0.0911
Test Loss: 0.2753
Epoch 380/400
Train Loss: 0.0945
Test Loss: 0.2754
Epoch 390/400
Train Loss: 0.0909
Test Loss: 0.2760
Epoch 400/400
Train Loss: 0.0919
Test Loss: 0.2773
```


train loss curve
1e8
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
train loss
0
50
100
150
200
250
300
350
400
epoch


test loss curve
800
600
400
200
0
test loss
0
50
100
150
200
250
300
350
400
epoch