group-project-ResNet-1

July 13, 2019

In [1]:
```
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 [2]:
```
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
```

In [3]:
```
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)
```

In [4]:
```
# 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 [5]:
```
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 [6]: 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 [7]:
```
# 3x3 convolution
def conv3x3(in_channels, out_channels, stride=1):
    return nn.Conv2d(in_channels, out_channels, kernel_size=3,
                     stride=stride, padding=1, bias=False)

# Residual block
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(ResidualBlock, self).__init__()
        self.conv1 = conv3x3(in_channels, out_channels, stride)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(out_channels, out_channels)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsample = downsample

    def forward(self, x):

        residual = x
        # if the size of input x changes, using downsample to change the size of residua
        if self.downsample:
```

```
            residual = self.downsample(x)
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        out += residual
        out = self.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes=10):
        super(ResNet, self).__init__()
        self.in_channels = 16
        self.conv = conv3x3(3, 16)
        self.bn = nn.BatchNorm2d(16)
        self.relu = nn.ReLU(inplace=True)
        # layer1: image size 32
        self.layer1 = self.make_layer(block, 16, num_blocks=layers[0])
        # layer2: image size 32 -> 16
        self.layer2 = self.make_layer(block, 32, num_blocks=layers[1], stride=2)
        # layer1: image size 16 -> 8
        self.layer3 = self.make_layer(block, 64, num_blocks=layers[2], stride=2)
        # global avg pool: image size 8 -> 1
        self.avg_pool = nn.AvgPool2d(8)

        self.fc = nn.Linear(64, num_classes)

    def make_layer(self, block, out_channels, num_blocks, stride=1):
        downsample = None
        if (stride != 1) or (self.in_channels != out_channels):
            downsample = nn.Sequential(
                conv3x3(self.in_channels, out_channels, stride=stride),
                nn.BatchNorm2d(out_channels))

        layers = []
        layers.append(block(self.in_channels, out_channels, stride, downsample))

        self.in_channels = out_channels

        for i in range(1, num_blocks):
            layers.append(block(out_channels, out_channels))

        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv(x)
        out = self.bn(out)
```

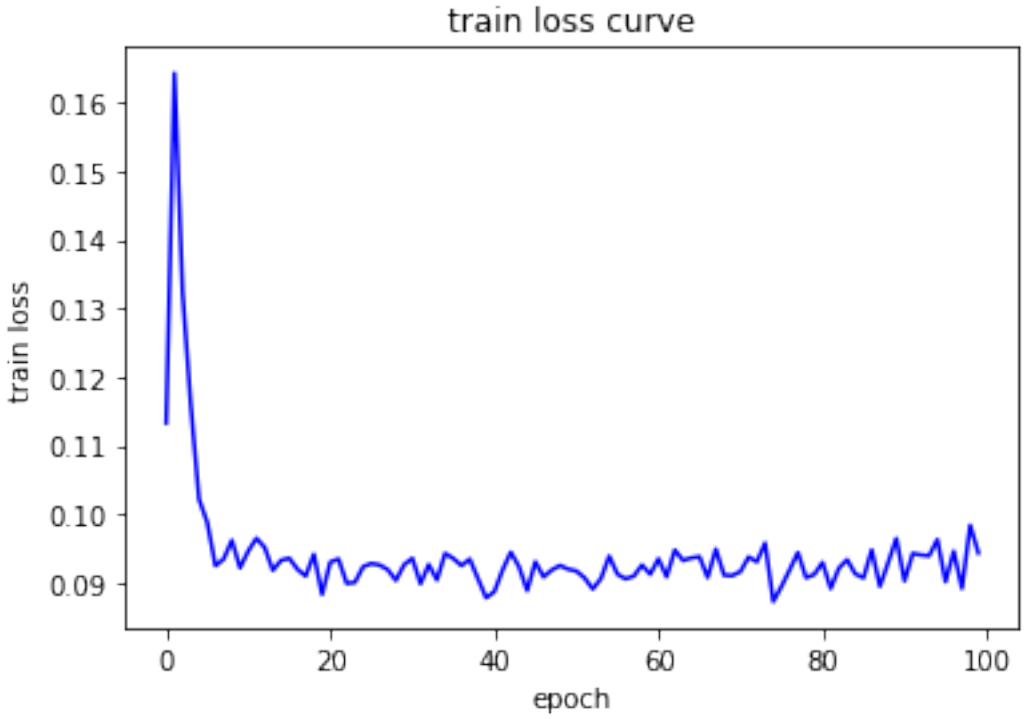
```
                out = self.relu(out)
                out = self.layer1(out)
                out = self.layer2(out)
                out = self.layer3(out)
                out = self.avg_pool(out)
                out = out.view(out.size(0), -1)
                out = self.fc(out)
                return out

In [8]: resnet = ResNet(ResidualBlock, [2, 2, 2], 1)
        print(resnet)

ResNet(
  (conv): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (layer1): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): ResidualBlock(
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): ResidualBlock(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
```

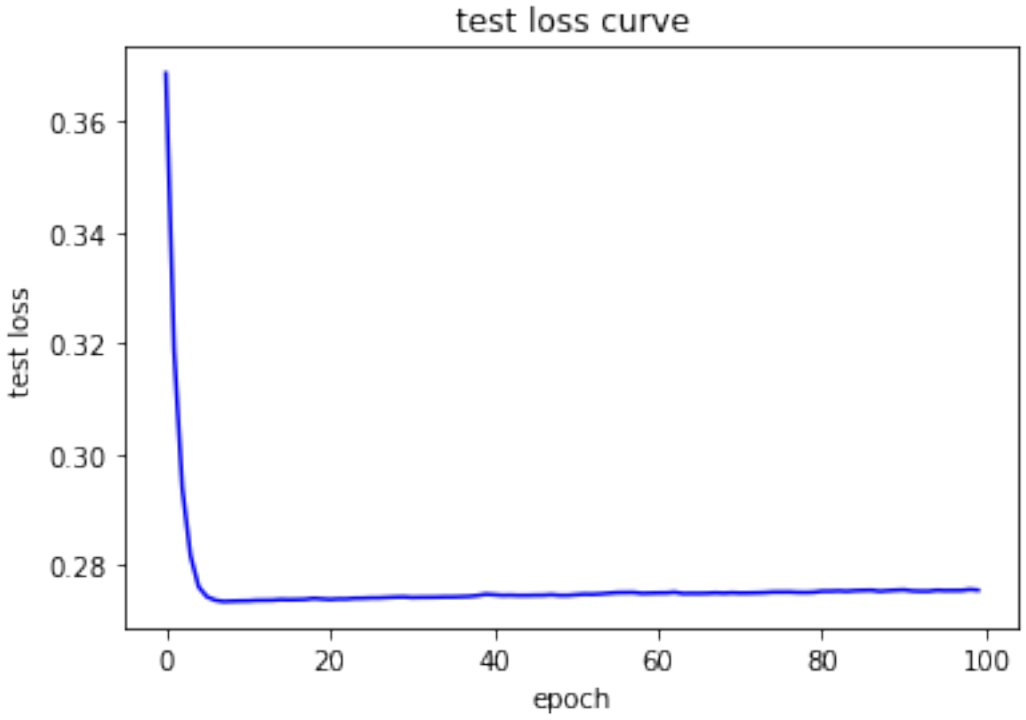
```
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): ResidualBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avg_pool): AvgPool2d(kernel_size=8, stride=8, padding=0)
  (fc): Linear(in_features=64, out_features=1, bias=True)
)
```

In [9]:
```
# training setting
# hyper parameters
num_epochs = 100
lr = 0.01
image_size = 32
num_classes = 1

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

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

```
Epoch 10/100
Train Loss: 0.0922
Test Loss: 0.2736
Epoch 20/100
Train Loss: 0.0884
Test Loss: 0.2739
```

```
Epoch 30/100
Train Loss: 0.0927
Test Loss: 0.2743
Epoch 40/100
Train Loss: 0.0878
Test Loss: 0.2748
Epoch 50/100
Train Loss: 0.0920
Test Loss: 0.2746
Epoch 60/100
Train Loss: 0.0913
Test Loss: 0.2750
Epoch 70/100
Train Loss: 0.0911
Test Loss: 0.2750
Epoch 80/100
Train Loss: 0.0913
Test Loss: 0.2751
Epoch 90/100
Train Loss: 0.0965
Test Loss: 0.2755
Epoch 100/100
Train Loss: 0.0944
Test Loss: 0.2755
```



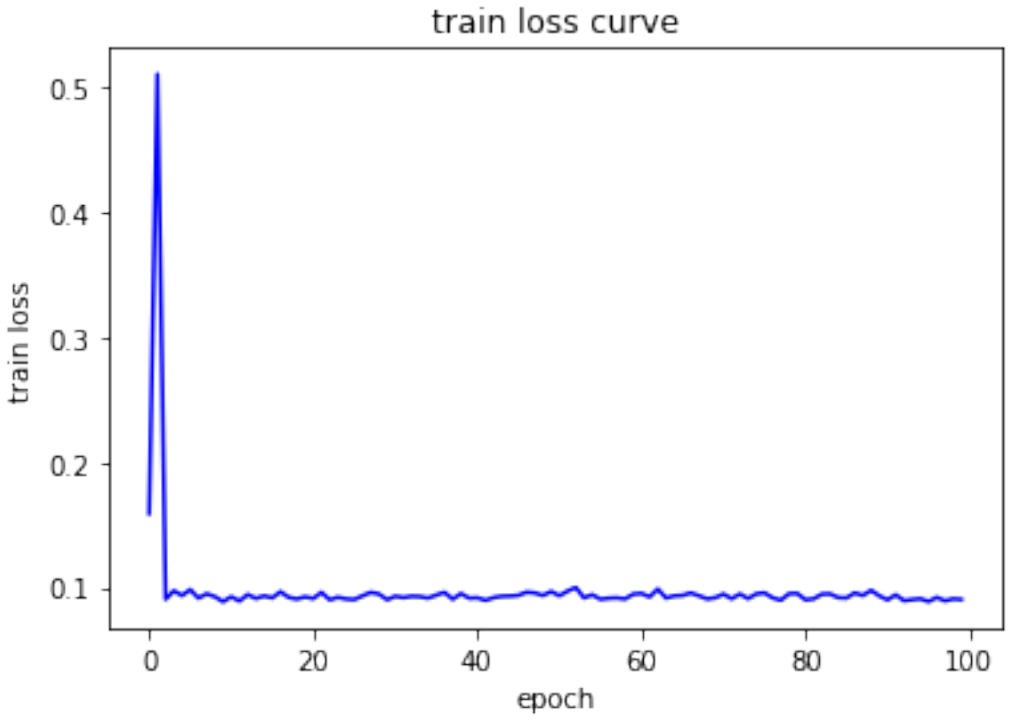


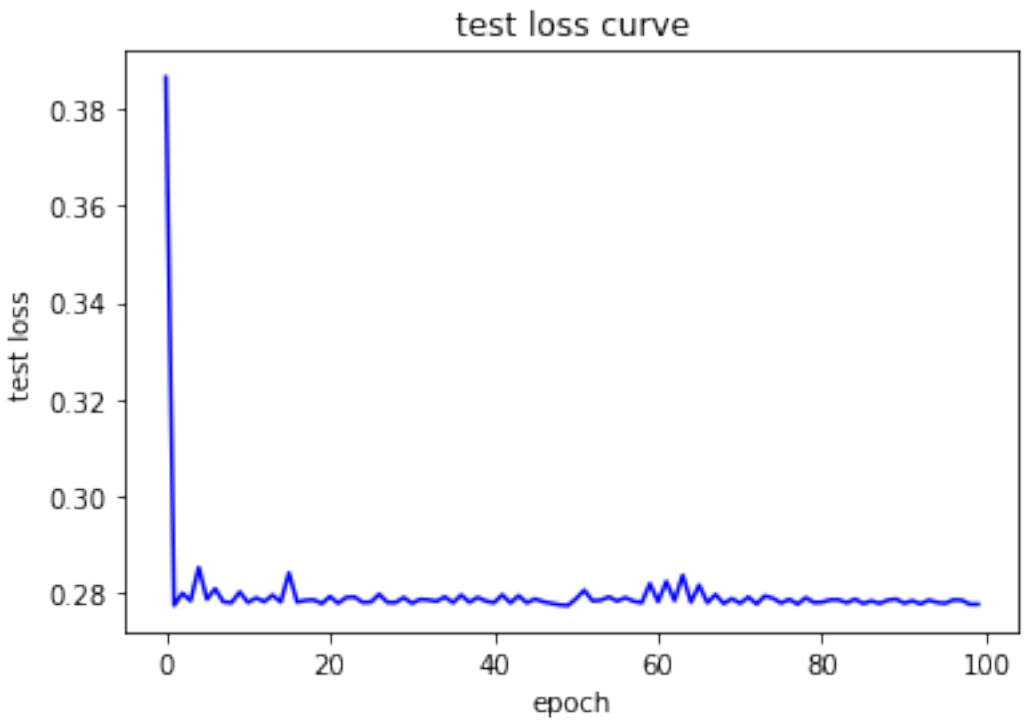
In [13]:
```
# training setting
# hyper parameters
num_epochs = 100
lr = 0.01
image_size = 32
num_classes = 1

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

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

In [14]:
```
fit(resnet, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0888
Test Loss: 0.2803
Epoch 20/100
Train Loss: 0.0926
Test Loss: 0.2779
Epoch 30/100
Train Loss: 0.0903
```

```
Test Loss: 0.2792
Epoch 40/100
Train Loss: 0.0916
Test Loss: 0.2784
Epoch 50/100
Train Loss: 0.0969
Test Loss: 0.2775
Epoch 60/100
Train Loss: 0.0948
Test Loss: 0.2820
Epoch 70/100
Train Loss: 0.0919
Test Loss: 0.2790
Epoch 80/100
Train Loss: 0.0953
Test Loss: 0.2781
Epoch 90/100
Train Loss: 0.0934
Test Loss: 0.2789
Epoch 100/100
Train Loss: 0.0906
Test Loss: 0.2778
```

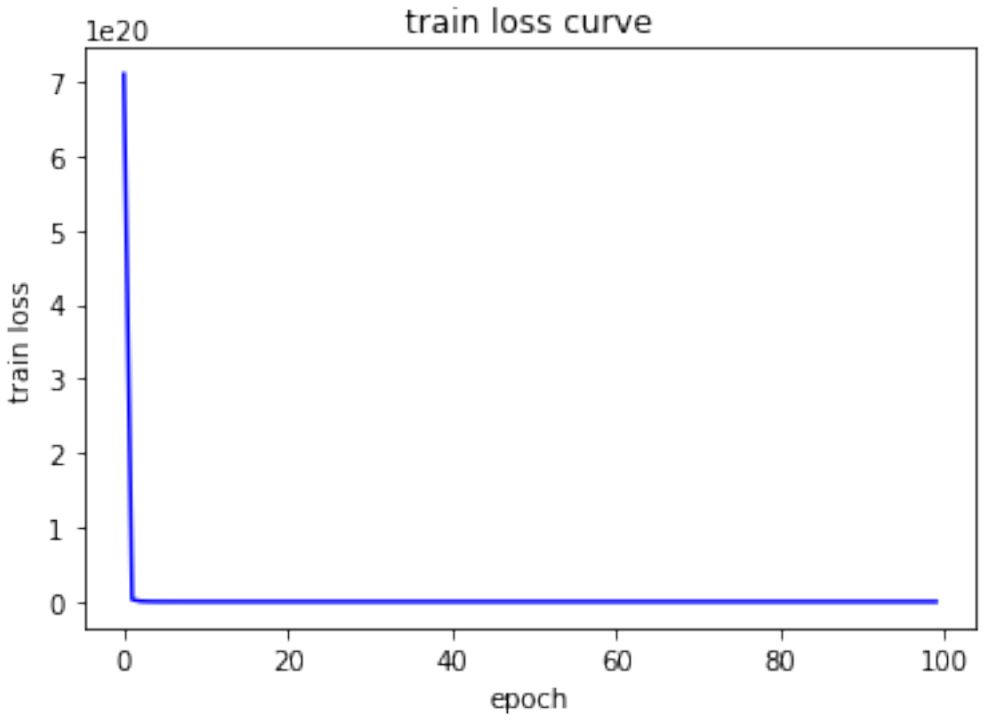


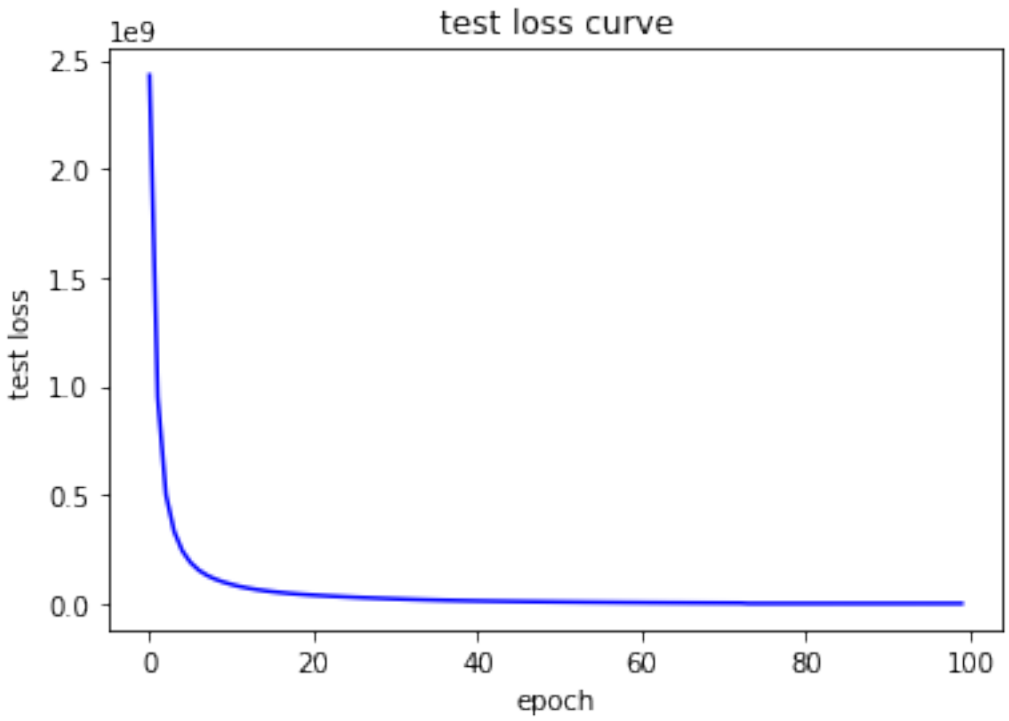


In [11]:
```
# training setting
# hyper parameters
num_epochs = 100
lr = 0.01
image_size = 32
num_classes = 1

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

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

```
Epoch 10/100
Train Loss: 12808430949731896.0000
Test Loss: 98556816.0000
Epoch 20/100
Train Loss: 2110637347263829.2500
Test Loss: 40848076.0000
Epoch 30/100
Train Loss: 636051135660032.0000
Test Loss: 22463044.0000
Epoch 40/100
Train Loss: 201165241785457.7812
```

```
Test Loss: 12369056.0000
Epoch 50/100
Train Loss: 71020349343516.4375
Test Loss: 7511150.0000
Epoch 60/100
Train Loss: 24612726024874.6680
Test Loss: 4340881.5000
Epoch 70/100
Train Loss: 7802336925013.3330
Test Loss: 2403489.5000
Epoch 80/100
Train Loss: 27337186.8889
Test Loss: 3823.0417
Epoch 90/100
Train Loss: 15108037.5556
Test Loss: 2817.3325
Epoch 100/100
Train Loss: 43980030.4444
Test Loss: 2161.7556
```





[U+4F7F] [U+7528] SGD [U+5F97] [U+5230] [U+7684] test loss [U+66F4] [U+5C0F] [U+FF0C] [U+6545] [U+4F18] [

In [10]:
```
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
        num_classes = 1

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

        optimizer = torch.optim.SGD(resnet.parameters(), lr=lr)
        fit2(resnet, num_epochs, optimizer, device)
```

```
Epoch 10/400
Train Loss: 0.0948
Test Loss: 0.2768
Epoch 20/400
Train Loss: 0.0900
Test Loss: 0.2770
Epoch 30/400
Train Loss: 0.0884
Test Loss: 0.2768
Epoch 40/400
Train Loss: 0.0904
Test Loss: 0.2771
Epoch 50/400
Train Loss: 0.0930
Test Loss: 0.2772
Epoch 60/400
Train Loss: 0.0935
Test Loss: 0.2770
Epoch 70/400
Train Loss: 0.0918
Test Loss: 0.2773
Epoch 80/400
```

```
Train Loss: 0.0917
Test Loss: 0.2769
Epoch 90/400
Train Loss: 0.0913
Test Loss: 0.2772
Epoch 100/400
Train Loss: 0.0919
Test Loss: 0.2777
Epoch 110/400
Train Loss: 0.0942
Test Loss: 0.2774
Epoch 120/400
Train Loss: 0.0906
Test Loss: 0.2773
Epoch 130/400
Train Loss: 0.0909
Test Loss: 0.2773
Epoch 140/400
Train Loss: 0.0915
Test Loss: 0.2773
Epoch 150/400
Train Loss: 0.0907
Test Loss: 0.2773
Epoch 160/400
Train Loss: 0.0925
Test Loss: 0.2773
Epoch 170/400
Train Loss: 0.0902
Test Loss: 0.2773
Epoch 180/400
Train Loss: 0.0892
Test Loss: 0.2773
Epoch 190/400
Train Loss: 0.0953
Test Loss: 0.2773
Epoch 200/400
Train Loss: 0.0927
Test Loss: 0.2772
Epoch 210/400
Train Loss: 0.0936
Test Loss: 0.2773
Epoch 220/400
Train Loss: 0.0912
Test Loss: 0.2772
Epoch 230/400
Train Loss: 0.0884
Test Loss: 0.2772
Epoch 240/400
```

```
Train Loss: 0.0946
Test Loss: 0.2773
Epoch 250/400
Train Loss: 0.0876
Test Loss: 0.2773
Epoch 260/400
Train Loss: 0.0935
Test Loss: 0.2773
Epoch 270/400
Train Loss: 0.0948
Test Loss: 0.2773
Epoch 280/400
Train Loss: 0.0923
Test Loss: 0.2773
Epoch 290/400
Train Loss: 0.0883
Test Loss: 0.2773
Epoch 300/400
Train Loss: 0.0906
Test Loss: 0.2773
Epoch 310/400
Train Loss: 0.0935
Test Loss: 0.2773
Epoch 320/400
Train Loss: 0.0915
Test Loss: 0.2773
Epoch 330/400
Train Loss: 0.0903
Test Loss: 0.2773
Epoch 340/400
Train Loss: 0.0926
Test Loss: 0.2773
Epoch 350/400
Train Loss: 0.0903
Test Loss: 0.2773
Epoch 360/400
Train Loss: 0.0887
Test Loss: 0.2773
Epoch 370/400
Train Loss: 0.0927
Test Loss: 0.2773
Epoch 380/400
Train Loss: 0.0920
Test Loss: 0.2773
Epoch 390/400
Train Loss: 0.0907
Test Loss: 0.2773
Epoch 400/400
```

```
Train Loss: 0.0871
Test Loss: 0.2773
```