group-project-DenseNet-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

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 [3]: 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 [4]: 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 [5]:

```
# Load necessary modules here
import math
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.backends.cudnn as cudnn
import os


class Bottleneck(nn.Module):
    '''
        the above mentioned bottleneck, including two conv layer, one's kernel size is 1

        after non-linear operation, concatenate the input to the output
    '''
    def __init__(self, in_planes, growth_rate):
        super(Bottleneck, self).__init__()
        self.bn1 = nn.BatchNorm2d(in_planes)
        self.conv1 = nn.Conv2d(in_planes, 4*growth_rate, kernel_size=1, bias=False)
        self.bn2 = nn.BatchNorm2d(4*growth_rate)
        self.conv2 = nn.Conv2d(4*growth_rate, growth_rate, kernel_size=3, padding=1, bia

    def forward(self, x):
        out = self.conv1(F.relu(self.bn1(x)))
        out = self.conv2(F.relu(self.bn2(out)))

        # input and output are concatenated here
        out = torch.cat([out,x], 1)
        return out


class Transition(nn.Module):
    '''
        transition layer is used for down sampling the feature

        when compress rate is 0.5, out_planes is a half of in_planes
    '''
    def __init__(self, in_planes, out_planes):
        super(Transition, self).__init__()
        self.bn = nn.BatchNorm2d(in_planes)
        self.conv = nn.Conv2d(in_planes, out_planes, kernel_size=1, bias=False)

    def forward(self, x):

        out = self.conv(F.relu(self.bn(x)))
        # use average pooling change the size of feature map here
        out = F.avg_pool2d(out, 2)
        return out


class DenseNet(nn.Module):
    def __init__(self, block, nblocks, growth_rate=12, reduction=0.5, num_classes=10):
        super(DenseNet, self).__init__()
        '''
        Args:
            block: bottleneck
            nblock: a list, the elements is number of bottleneck in each denseblock
            growth_rate: channel size of bottleneck's output
            reduction:
        '''
        self.growth_rate = growth_rate

        num_planes = 2*growth_rate
        self.conv1 = nn.Conv2d(3, num_planes, kernel_size=3, padding=1, bias=False)

        # a DenseBlock and a transition layer
        self.dense1 = self._make_dense_layers(block, num_planes, nblocks[0])
        num_planes += nblocks[0]*growth_rate
        # the channel size is superposed, mutiply by reduction to cut it down here, the
        out_planes = int(math.floor(num_planes*reduction))
        self.trans1 = Transition(num_planes, out_planes)
        num_planes = out_planes

        # a DenseBlock and a transition layer
        self.dense2 = self._make_dense_layers(block, num_planes, nblocks[1])
        num_planes += nblocks[1]*growth_rate
        # the channel size is superposed, mutiply by reduction to cut it down here, the
        out_planes = int(math.floor(num_planes*reduction))
        self.trans2 = Transition(num_planes, out_planes)
        num_planes = out_planes

        # a DenseBlock and a transition layer
        self.dense3 = self._make_dense_layers(block, num_planes, nblocks[2])
        num_planes += nblocks[2]*growth_rate
        # the channel size is superposed, mutiply by reduction to cut it down here, the
        out_planes = int(math.floor(num_planes*reduction))
        self.trans3 = Transition(num_planes, out_planes)
        num_planes = out_planes

        # only one DenseBlock
        self.dense4 = self._make_dense_layers(block, num_planes, nblocks[3])
        num_planes += nblocks[3]*growth_rate

        # the last part is a linear layer as a classifier
        self.bn = nn.BatchNorm2d(num_planes)
        self.linear = nn.Linear(num_planes, num_classes)

    def _make_dense_layers(self, block, in_planes, nblock):
        layers = []

        # number of non-linear transformations in one DenseBlock depends on the paramete
        for i in range(nblock):
            layers.append(block(in_planes, self.growth_rate))
            in_planes += self.growth_rate
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv1(x)
        out = self.trans1(self.dense1(out))
        out = self.trans2(self.dense2(out))
        out = self.trans3(self.dense3(out))
        out = self.dense4(out)
        out = F.avg_pool2d(F.relu(self.bn(out)), 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```

```
In [6]: # start training and testing
        #densenet = DenseNet(Bottleneck, [2, 5, 4, 6], num_classes=1)
        densenet52 = DenseNet(Bottleneck, [6, 6, 6, 6], num_classes=1)
        print(densenet52)
```

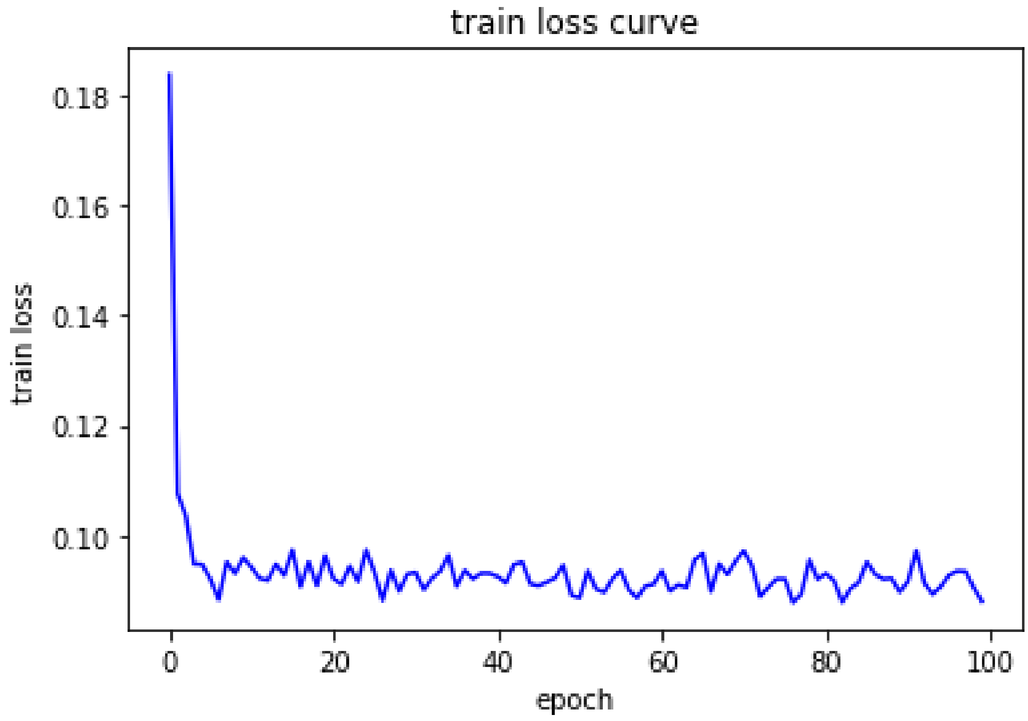
```
DenseNet(
  (conv1): Conv2d(3, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (dense1): Sequential(
    (0): Bottleneck(
      (bn1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(24, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (1): Bottleneck(
      (bn1): BatchNorm2d(36, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(36, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (2): Bottleneck(
      (bn1): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(48, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (3): Bottleneck(
      (bn1): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(60, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (4): Bottleneck(
      (bn1): BatchNorm2d(72, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(72, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (5): Bottleneck(
      (bn1): BatchNorm2d(84, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(84, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
  )
  (trans1): Transition(
    (bn): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv): Conv2d(96, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
  )
  (dense2): Sequential(
    (0): Bottleneck(
      (bn1): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(48, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (1): Bottleneck(
      (bn1): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(60, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (2): Bottleneck(
      (bn1): BatchNorm2d(72, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(72, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (3): Bottleneck(
      (bn1): BatchNorm2d(84, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(84, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (4): Bottleneck(
      (bn1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(96, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (5): Bottleneck(
      (bn1): BatchNorm2d(108, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(108, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
  )
  (trans2): Transition(
    (bn): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv): Conv2d(120, 60, kernel_size=(1, 1), stride=(1, 1), bias=False)
  )
  (dense3): Sequential(
    (0): Bottleneck(
      (bn1): BatchNorm2d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(60, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (1): Bottleneck(
      (bn1): BatchNorm2d(72, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(72, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (2): Bottleneck(
      (bn1): BatchNorm2d(84, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(84, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (3): Bottleneck(
      (bn1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(96, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (4): Bottleneck(
      (bn1): BatchNorm2d(108, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(108, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (5): Bottleneck(
      (bn1): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(120, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
  )
  (trans3): Transition(
    (bn): BatchNorm2d(132, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv): Conv2d(132, 66, kernel_size=(1, 1), stride=(1, 1), bias=False)
  )
  (dense4): Sequential(
    (0): Bottleneck(
      (bn1): BatchNorm2d(66, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(66, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (1): Bottleneck(
      (bn1): BatchNorm2d(78, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(78, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (2): Bottleneck(
      (bn1): BatchNorm2d(90, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(90, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (3): Bottleneck(
      (bn1): BatchNorm2d(102, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(102, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (4): Bottleneck(
      (bn1): BatchNorm2d(114, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(114, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
    (5): Bottleneck(
      (bn1): BatchNorm2d(126, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv1): Conv2d(126, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(48, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    )
  )
  (bn): BatchNorm2d(138, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (linear): Linear(in_features=138, out_features=1, bias=True)
)
```

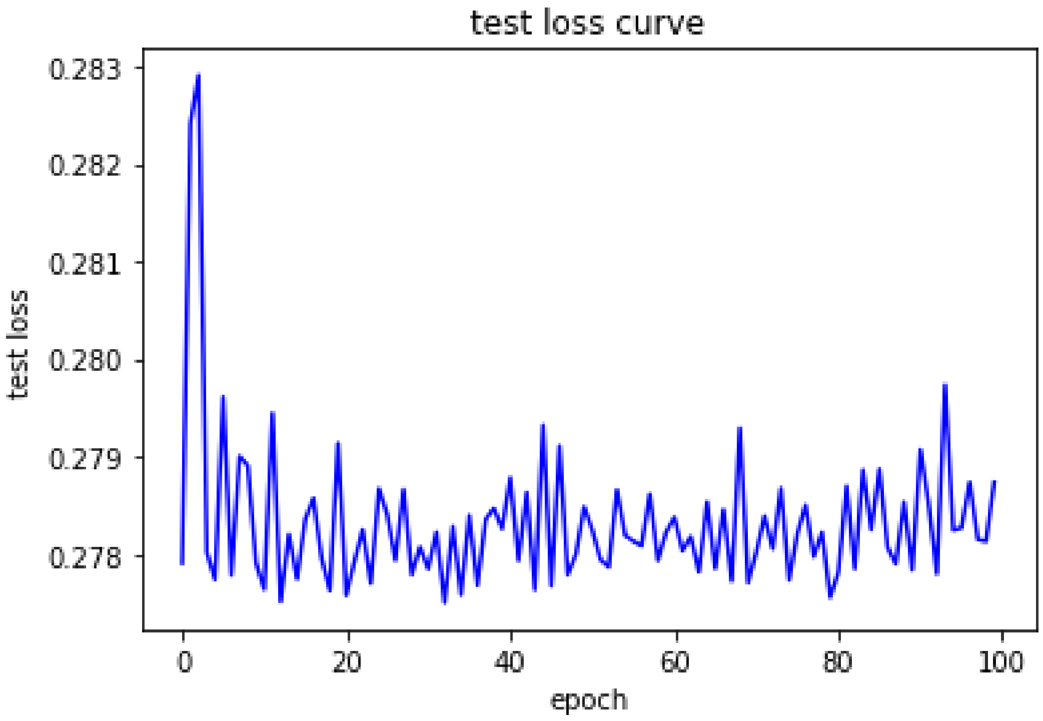
```
In [7]: # training setting
        # hyper parameters
        num_epochs = 100
        lr = 0.01
        image_size = 32

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

        optimizer = torch.optim.Adam(densenet52.parameters(), lr=lr)
```

```
In [8]: fit(densenet52, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0960
Test Loss: 0.2779
```

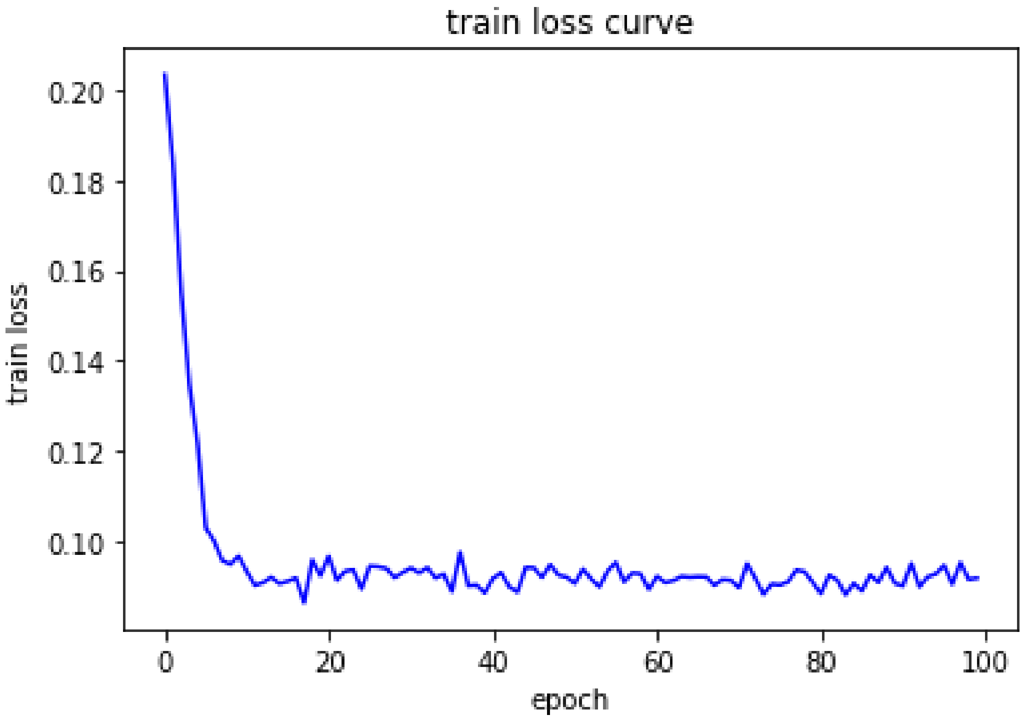
```
Epoch 20/100
Train Loss: 0.0964
Test Loss: 0.2791
Epoch 30/100
Train Loss: 0.0930
Test Loss: 0.2781
Epoch 40/100
Train Loss: 0.0932
Test Loss: 0.2783
Epoch 50/100
Train Loss: 0.0892
Test Loss: 0.2785
Epoch 60/100
Train Loss: 0.0912
Test Loss: 0.2782
Epoch 70/100
Train Loss: 0.0954
Test Loss: 0.2777
Epoch 80/100
Train Loss: 0.0921
Test Loss: 0.2776
Epoch 90/100
Train Loss: 0.0898
Test Loss: 0.2778
Epoch 100/100
Train Loss: 0.0881
Test Loss: 0.2787
```

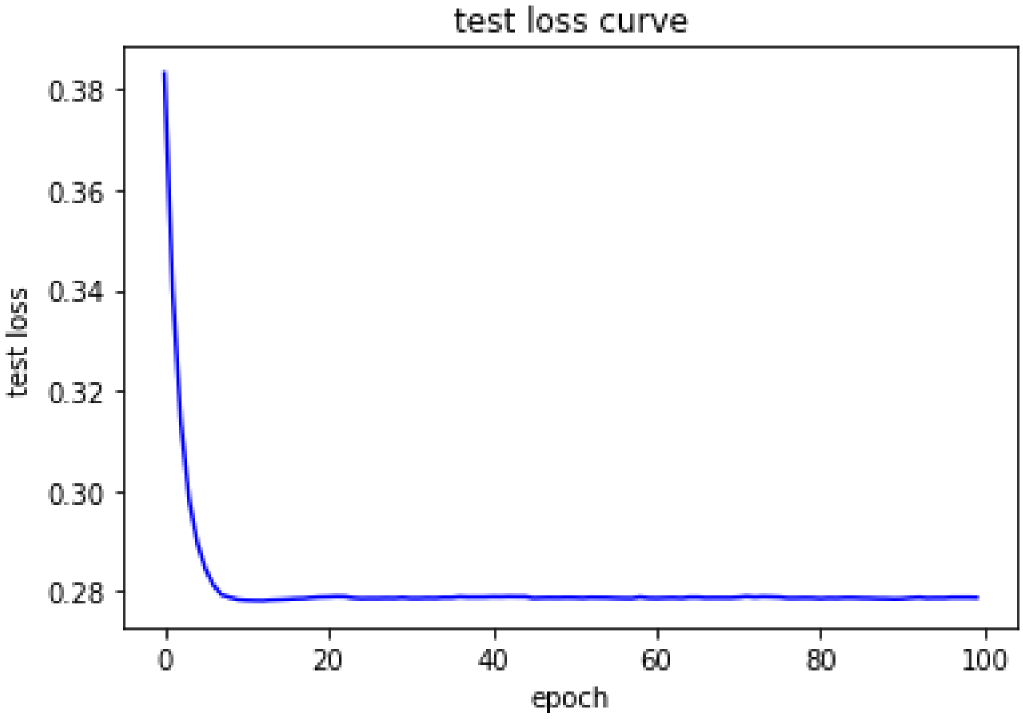

train loss curve
train loss
0.18
0.16
0.14
0.12
0.10
0
20
40
60
80
100
epoch


test loss curve
test loss
0.283
0.282
0.281
0.280
0.279
0.278
0
20
40
60
80
100
epoch

```
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.SGD(densenet52.parameters(), lr=lr)
        fit(densenet52, num_epochs, optimizer, device)
```

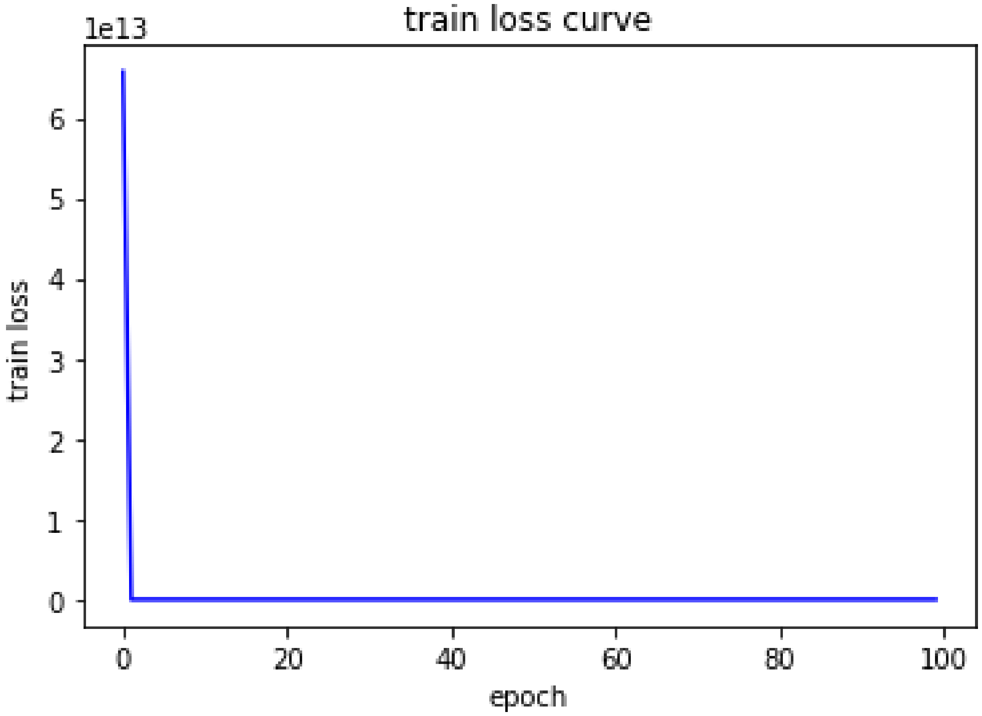
```
Epoch 10/100
Train Loss: 0.0966
Test Loss: 0.2783
Epoch 20/100
Train Loss: 0.0922
Test Loss: 0.2789
Epoch 30/100
Train Loss: 0.0929
Test Loss: 0.2787
Epoch 40/100
Train Loss: 0.0884
Test Loss: 0.2789
Epoch 50/100
Train Loss: 0.0919
Test Loss: 0.2788
Epoch 60/100
Train Loss: 0.0892
Test Loss: 0.2787
Epoch 70/100
Train Loss: 0.0912
Test Loss: 0.2787
Epoch 80/100
Train Loss: 0.0908
Test Loss: 0.2787
Epoch 90/100
Train Loss: 0.0906
Test Loss: 0.2785
Epoch 100/100
Train Loss: 0.0917
Test Loss: 0.2787
```

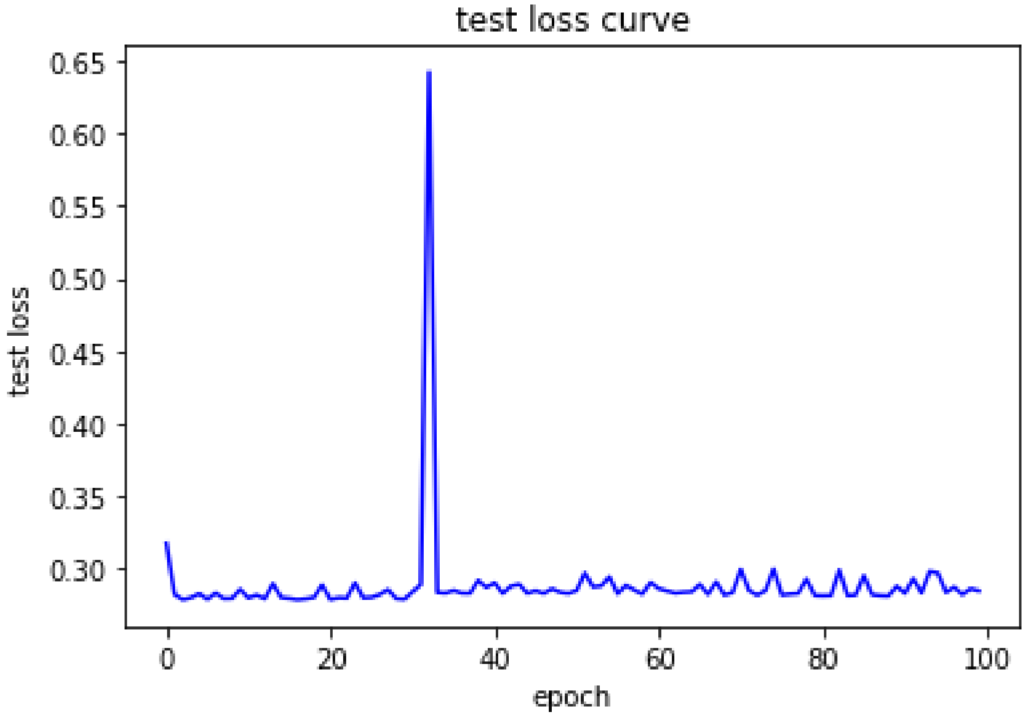

train loss curve
train loss
0.20
0.18
0.16
0.14
0.12
0.10
0
20
40
60
80
100
epoch


test loss curve
test loss
0.38
0.36
0.34
0.32
0.30
0.28
0
20
40
60
80
100
epoch

```
In [8]: # 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.RMSprop(densenet52.parameters(), lr=lr)
        fit(densenet52, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0972
Test Loss: 0.2861
Epoch 20/100
Train Loss: 0.0914
Test Loss: 0.2892
Epoch 30/100
Train Loss: 0.0908
Test Loss: 0.2793
Epoch 40/100
Train Loss: 0.0916
Test Loss: 0.2875
Epoch 50/100
Train Loss: 0.0928
Test Loss: 0.2833
Epoch 60/100
Train Loss: 0.0906
Test Loss: 0.2908
Epoch 70/100
Train Loss: 0.0905
Test Loss: 0.2839
Epoch 80/100
Train Loss: 0.0914
Test Loss: 0.2819
Epoch 90/100
Train Loss: 0.0918
Test Loss: 0.2883
Epoch 100/100
Train Loss: 0.0887
Test Loss: 0.2850
```

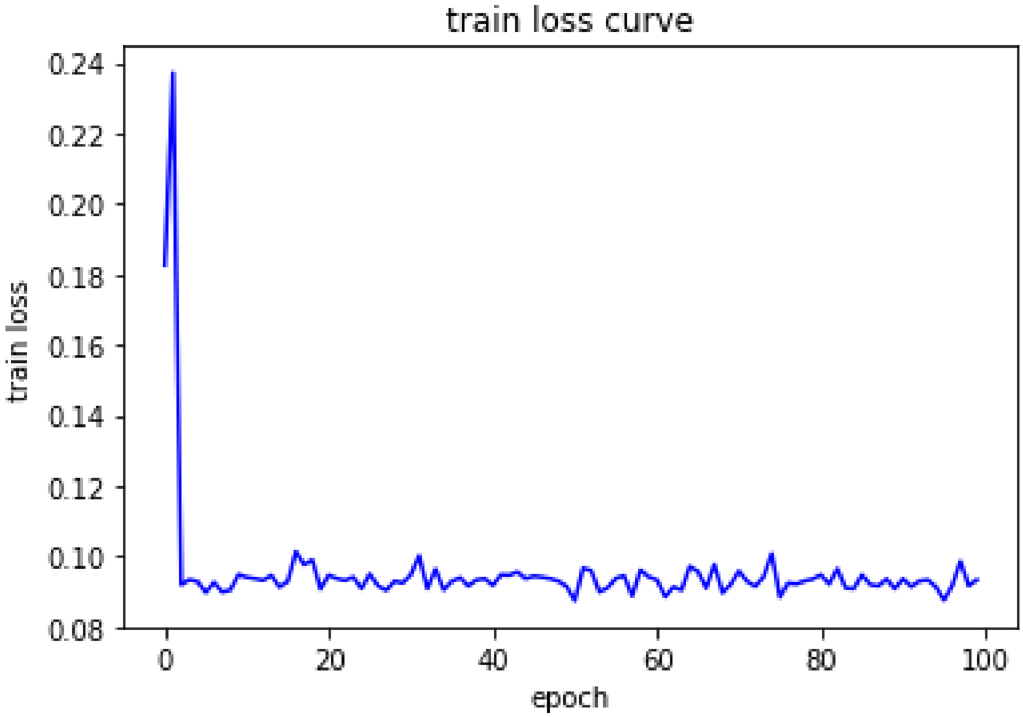


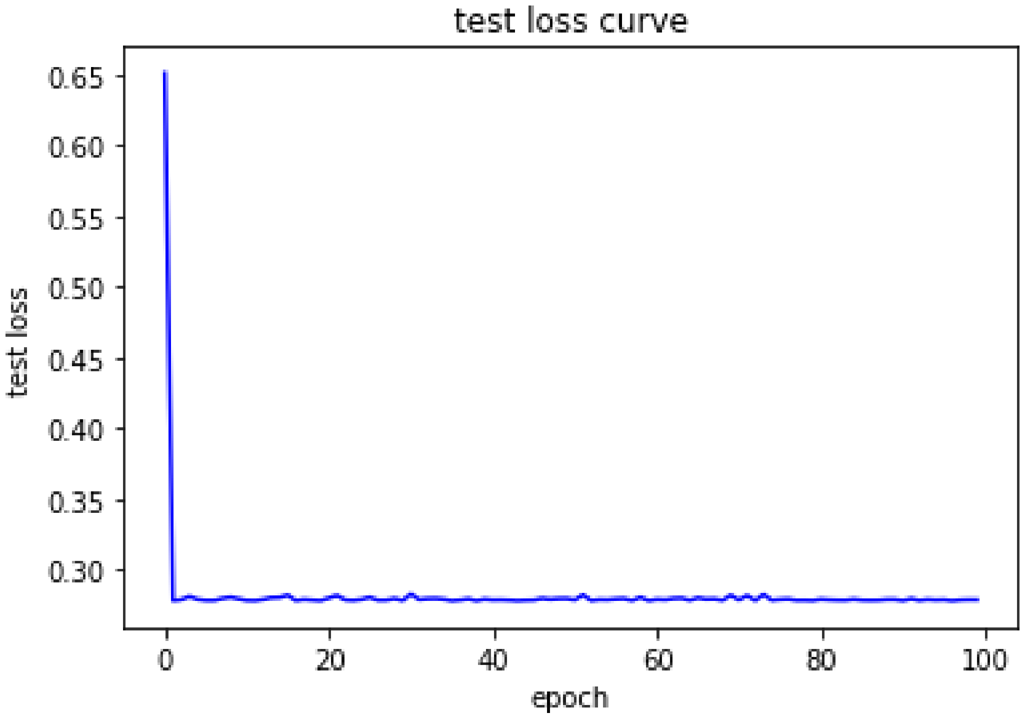


[U+603B] [U+4F53] [U+6765] [U+770B] [U+FF0C] Adam [U+7684] [U+6548] [U+679C] [U+66F4] [U+597D]

```
In [9]: # start training and testing
        #densenet = DenseNet(Bottleneck, [2, 5, 4, 6], num_classes=1)
        densenet121 = DenseNet(Bottleneck, [6, 12, 24, 16], num_classes=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:0')

        optimizer = torch.optim.Adam(densenet121.parameters(), lr=lr)
        fit(densenet121, num_epochs, optimizer, device)
```

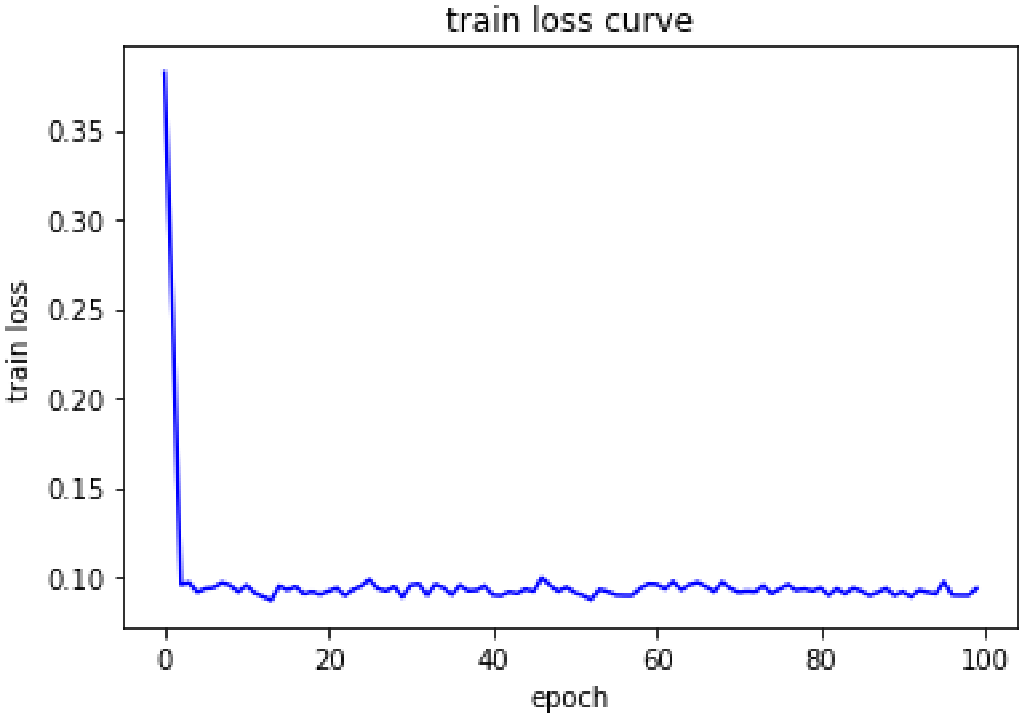
```
Epoch 10/100
Train Loss: 0.0948
Test Loss: 0.2795
Epoch 20/100
Train Loss: 0.0907
Test Loss: 0.2782
Epoch 30/100
Train Loss: 0.0925
Test Loss: 0.2782
Epoch 40/100
Train Loss: 0.0937
Test Loss: 0.2793
Epoch 50/100
Train Loss: 0.0915
Test Loss: 0.2797
Epoch 60/100
Train Loss: 0.0942
Test Loss: 0.2782
Epoch 70/100
Train Loss: 0.0921
Test Loss: 0.2820
Epoch 80/100
Train Loss: 0.0936
Test Loss: 0.2782
Epoch 90/100
Train Loss: 0.0908
Test Loss: 0.2790
Epoch 100/100
Train Loss: 0.0934
Test Loss: 0.2788
```

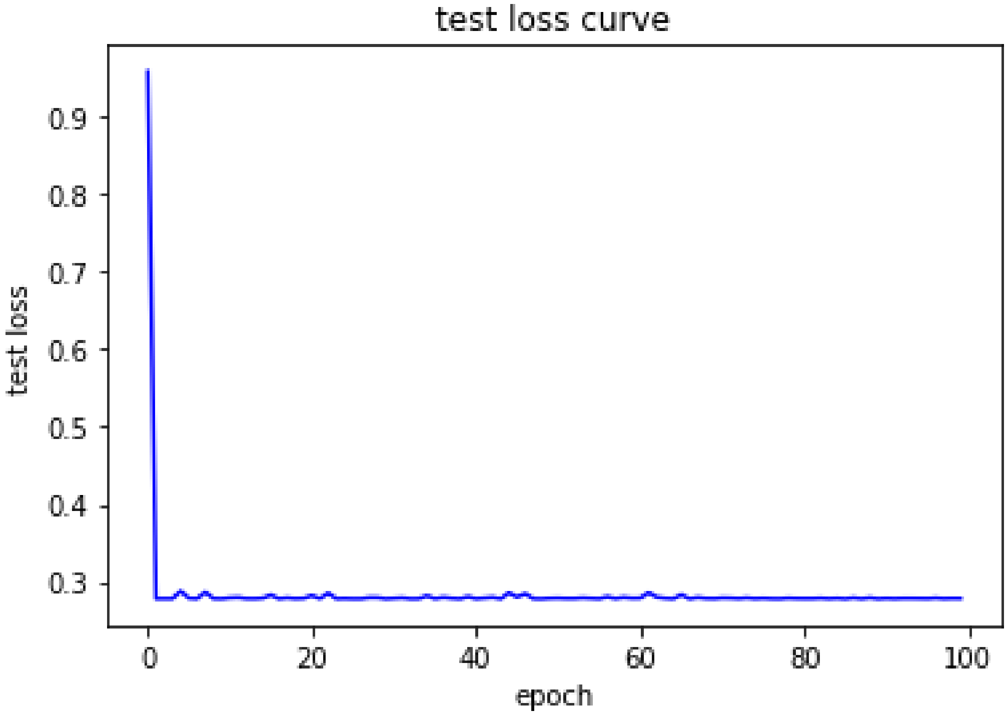

train loss curve
train loss
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0
20
40
60
80
100
epoch


test loss curve
test loss
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
20
40
60
80
100
epoch

```
In [10]: # start training and testing
         #densenet = DenseNet(Bottleneck, [2, 5, 4, 6], num_classes=1)
         densenet169 = DenseNet(Bottleneck, [6, 12, 32, 32], num_classes=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:0')

         optimizer = torch.optim.Adam(densenet169.parameters(), lr=lr)
         fit(densenet169, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0918
Test Loss: 0.2782
Epoch 20/100
Train Loss: 0.0903
Test Loss: 0.2789
Epoch 30/100
Train Loss: 0.0889
Test Loss: 0.2781
Epoch 40/100
Train Loss: 0.0952
Test Loss: 0.2809
Epoch 50/100
Train Loss: 0.0945
Test Loss: 0.2788
Epoch 60/100
Train Loss: 0.0962
Test Loss: 0.2789
Epoch 70/100
Train Loss: 0.0938
Test Loss: 0.2784
Epoch 80/100
Train Loss: 0.0923
Test Loss: 0.2787
Epoch 90/100
Train Loss: 0.0897
Test Loss: 0.2781
Epoch 100/100
Train Loss: 0.0938
Test Loss: 0.2786
```

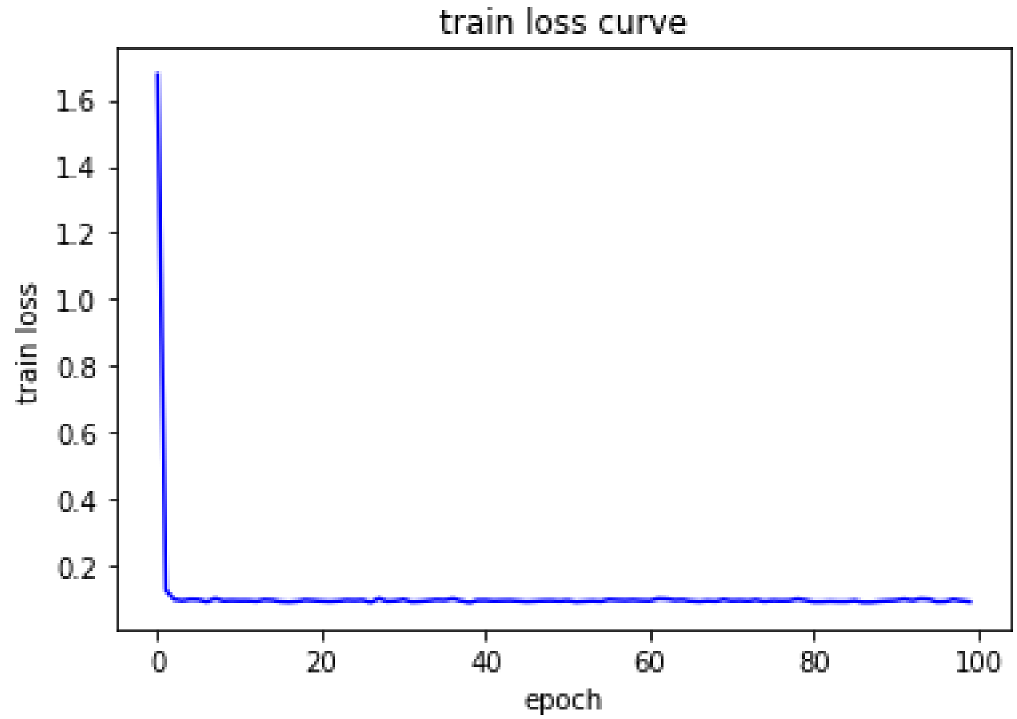

train loss curve
train loss
0.35
0.30
0.25
0.20
0.15
0.10
0
20
40
60
80
100
epoch

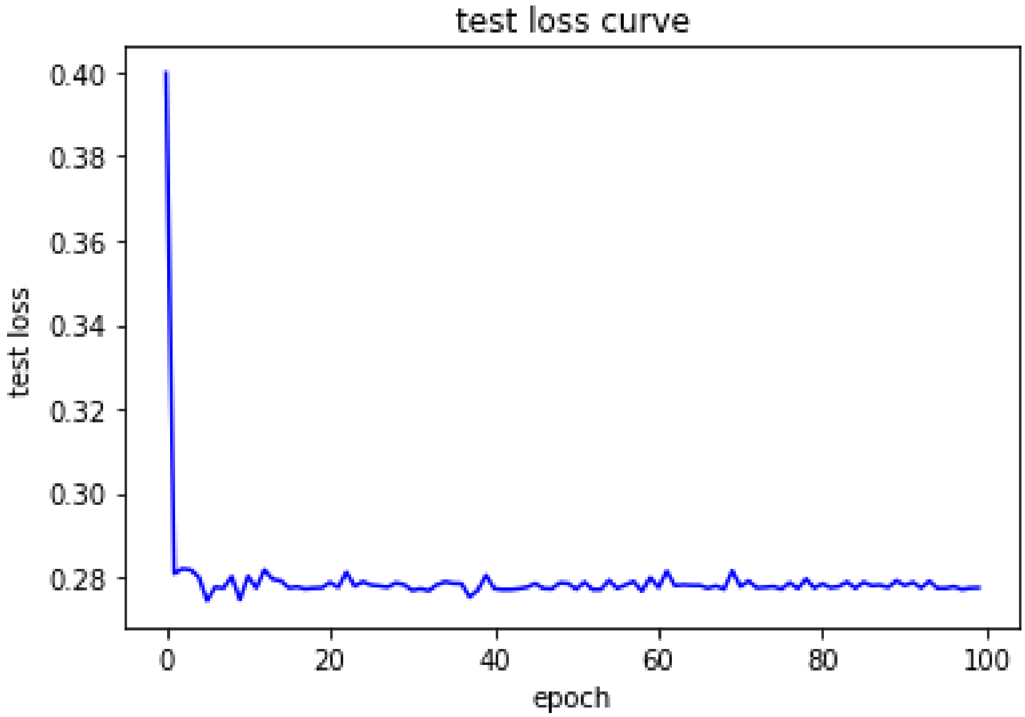

test loss curve
test loss
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100
epoch

In [13]:

```
# start training and testing
#densenet = DenseNet(Bottleneck, [2, 5, 4, 6], num_classes=1)
densenet201 = DenseNet(Bottleneck, [6, 12, 48, 32], num_classes=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.Adam(densenet201.parameters(), lr=lr)
fit(densenet201, num_epochs, optimizer, device)
```

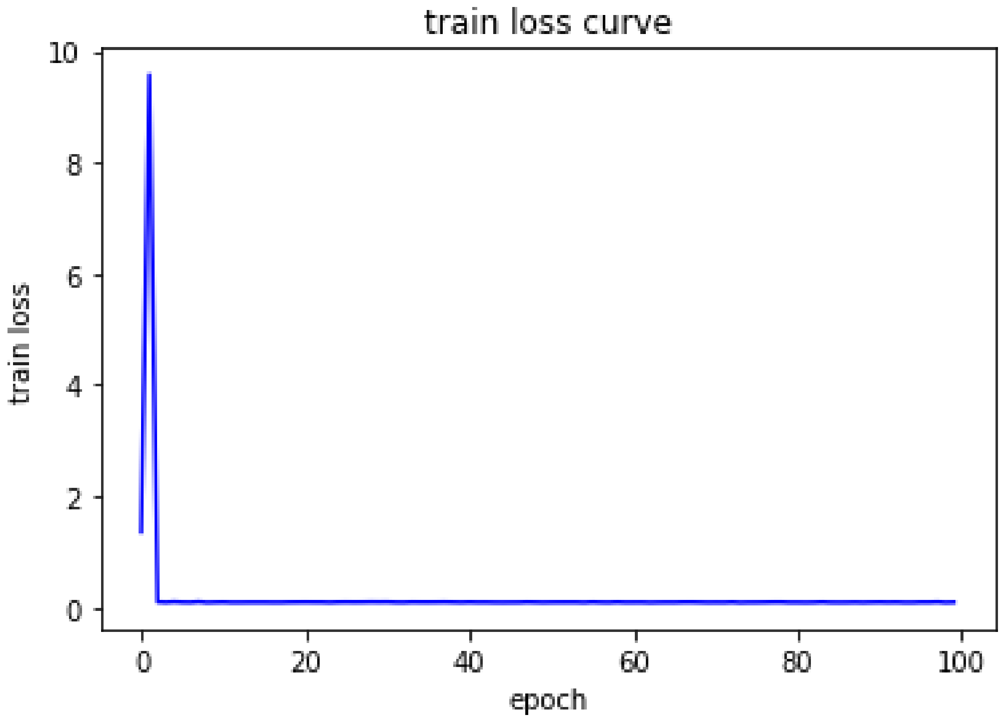
```
Epoch 10/100
Train Loss: 0.0938
Test Loss: 0.2747
Epoch 20/100
Train Loss: 0.0938
Test Loss: 0.2775
Epoch 30/100
Train Loss: 0.0927
Test Loss: 0.2783
Epoch 40/100
Train Loss: 0.0942
Test Loss: 0.2805
Epoch 50/100
Train Loss: 0.0917
Test Loss: 0.2785
Epoch 60/100
Train Loss: 0.0937
Test Loss: 0.2799
Epoch 70/100
Train Loss: 0.0964
Test Loss: 0.2815
Epoch 80/100
Train Loss: 0.0944
Test Loss: 0.2774
Epoch 90/100
Train Loss: 0.0928
Test Loss: 0.2792
Epoch 100/100
Train Loss: 0.0897
Test Loss: 0.2775
```

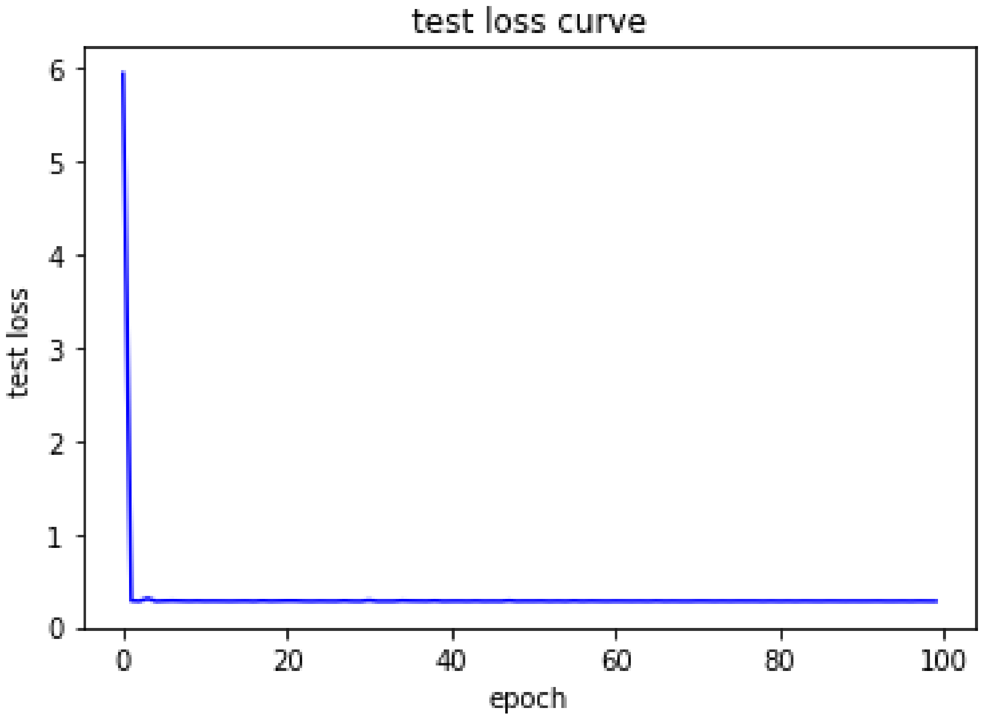

train loss curve
train loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100
epoch


test loss curve
test loss
0.40
0.38
0.36
0.34
0.32
0.30
0.28
0
20
40
60
80
100
epoch

```
In [16]: # start training and testing
         #densenet = DenseNet(Bottleneck, [2, 5, 4, 6], num_classes=1)
         densenet264 = DenseNet(Bottleneck, [6, 12, 64, 48], num_classes=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.Adam(densenet264.parameters(), lr=lr)
         fit(densenet264, num_epochs, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.0929
Test Loss: 0.2812
Epoch 20/100
Train Loss: 0.0953
Test Loss: 0.2810
Epoch 30/100
Train Loss: 0.0976
Test Loss: 0.2782
Epoch 40/100
Train Loss: 0.0915
Test Loss: 0.2782
Epoch 50/100
Train Loss: 0.0907
Test Loss: 0.2796
Epoch 60/100
Train Loss: 0.0913
Test Loss: 0.2784
Epoch 70/100
Train Loss: 0.0900
Test Loss: 0.2781
Epoch 80/100
Train Loss: 0.0904
Test Loss: 0.2795
Epoch 90/100
Train Loss: 0.0928
Test Loss: 0.2791
Epoch 100/100
Train Loss: 0.0932
Test Loss: 0.2783
```





densenet169 [U+7684] [U+6548] [U+679C] [U+6700] [U+597D] [U+FF0C] [U+518D] [U+6B21] [U+8BAD] [U+7EC3]

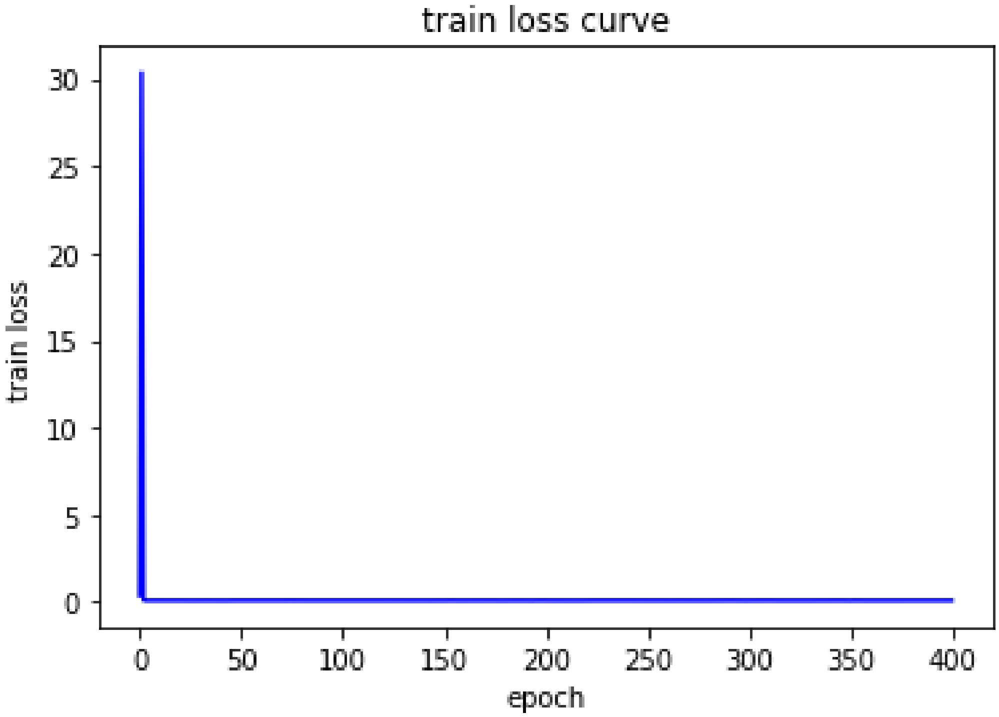
```
In [28]: from torch.optim import lr_scheduler
         def fit2(model, num_epochs, optimizer, device):
             loss_func = nn.MSELoss()
             model.to(device)
             if device == torch.device('cuda'):
                 model = torch.nn.DataParallel(model)
                 cudnn.benchmark = 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")


         # start training and testing
         #densenet = DenseNet(Bottleneck, [2, 5, 4, 6], num_classes=1)
         densenet169 = DenseNet(Bottleneck, [6, 12, 48, 32], num_classes=1)

         # 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')
```

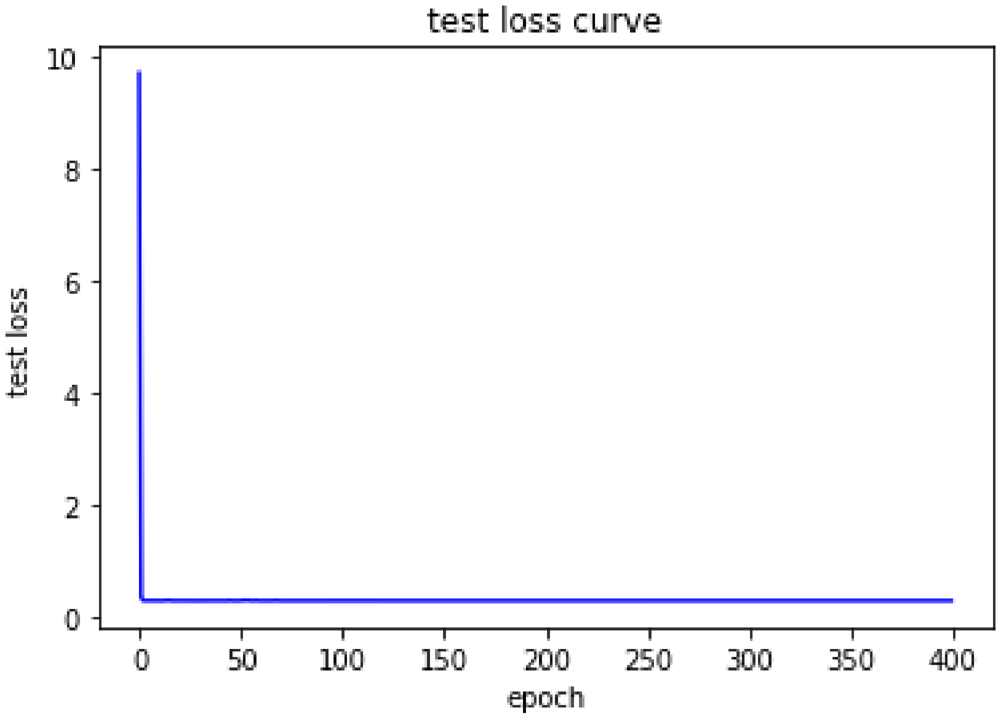
```
optimizer = torch.optim.Adam(densenet169.parameters(), lr=lr)
fit2(densenet169, num_epochs, optimizer, device)
```

```
Epoch 10/400
Train Loss: 0.0944
Test Loss: 0.2807
Epoch 20/400
Train Loss: 0.0891
Test Loss: 0.2799
Epoch 30/400
Train Loss: 0.0900
Test Loss: 0.2784
Epoch 40/400
Train Loss: 0.0931
Test Loss: 0.2800
Epoch 50/400
Train Loss: 0.0971
Test Loss: 0.2781
Epoch 60/400
Train Loss: 0.0964
Test Loss: 0.2782
Epoch 70/400
Train Loss: 0.0967
Test Loss: 0.2781
Epoch 80/400
Train Loss: 0.0953
Test Loss: 0.2804
Epoch 90/400
Train Loss: 0.0900
Test Loss: 0.2784
Epoch 100/400
Train Loss: 0.0900
Test Loss: 0.2786
Epoch 110/400
Train Loss: 0.0872
Test Loss: 0.2793
Epoch 120/400
Train Loss: 0.0929
Test Loss: 0.2792
Epoch 130/400
Train Loss: 0.0936
Test Loss: 0.2794
Epoch 140/400
Train Loss: 0.0905
Test Loss: 0.2785
Epoch 150/400
Train Loss: 0.0893
```

```
Test Loss: 0.2789
Epoch 160/400
Train Loss: 0.0900
Test Loss: 0.2794
Epoch 170/400
Train Loss: 0.0973
Test Loss: 0.2792
Epoch 180/400
Train Loss: 0.0925
Test Loss: 0.2789
Epoch 190/400
Train Loss: 0.0906
Test Loss: 0.2791
Epoch 200/400
Train Loss: 0.0896
Test Loss: 0.2793
Epoch 210/400
Train Loss: 0.0900
Test Loss: 0.2790
Epoch 220/400
Train Loss: 0.0931
Test Loss: 0.2788
Epoch 230/400
Train Loss: 0.0945
Test Loss: 0.2787
Epoch 240/400
Train Loss: 0.0963
Test Loss: 0.2789
Epoch 250/400
Train Loss: 0.0899
Test Loss: 0.2788
Epoch 260/400
Train Loss: 0.0956
Test Loss: 0.2787
Epoch 270/400
Train Loss: 0.0921
Test Loss: 0.2789
Epoch 280/400
Train Loss: 0.0907
Test Loss: 0.2787
Epoch 290/400
Train Loss: 0.0931
Test Loss: 0.2789
Epoch 300/400
Train Loss: 0.0938
Test Loss: 0.2789
Epoch 310/400
Train Loss: 0.0921
```

```
Test Loss: 0.2789
Epoch 320/400
Train Loss: 0.0913
Test Loss: 0.2789
Epoch 330/400
Train Loss: 0.0892
Test Loss: 0.2789
Epoch 340/400
Train Loss: 0.0913
Test Loss: 0.2789
Epoch 350/400
Train Loss: 0.0880
Test Loss: 0.2789
Epoch 360/400
Train Loss: 0.0913
Test Loss: 0.2788
Epoch 370/400
Train Loss: 0.0902
Test Loss: 0.2788
Epoch 380/400
Train Loss: 0.0910
Test Loss: 0.2788
Epoch 390/400
Train Loss: 0.0907
Test Loss: 0.2788
Epoch 400/400
Train Loss: 0.0919
Test Loss: 0.2788
```


train loss curve
train loss
30
25
20
15
10
5
0
0
50
100
150
200
250
300
350
400
epoch


test loss curve
test loss
10
8
6
4
2
0
0
50
100
150
200
250
300
350
400
epoch