simclr_mnist

December 13, 2023

1 Contrastive Learning on non-spurious MNIST dataset

[1]:
```
import torch
import torch.optim as optim
import torch.nn as nn
import torch.nn.functional as F

from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np
```

[35]:
```
torch.cuda.is_available()
```

[35]:
```
True
```

2 Utils

[2]:
```
from collections import defaultdict

class MetricMonitor:
    def __init__(self, float_precision=4):
        self.float_precision = float_precision
        self.reset()

    def reset(self):
        self.metrics = defaultdict(lambda: {"val": 0, "count": 0, "avg": 0})

    def update(self, metric_name, val):
        metric = self.metrics[metric_name]

        metric["val"] += val
        metric["count"] += 1
        metric["avg"] = metric["val"] / metric["count"]

    def __str__(self):
        return " | ".join(
```

```
            [
                "{metric_name}: {avg:.{float_precision}f}".format(
                    metric_name=metric_name, avg=metric["avg"],␣
↪float_precision=self.float_precision
                )
                for (metric_name, metric) in self.metrics.items()
            ]
        )
```

[3]:
```
class EarlyStopping:
    def __init__(self, patience=7, verbose=False, delta=0, path='checkpoint.
↪pt', trace_func=print):
        """
        Source:
            https://github.com/Bjarten/early-stopping-pytorch/blob/master/
↪pytorchtools.py
        Args:
            patience (int): How long to wait after last time validation loss␣
↪improved.
                            Default: 7
            verbose (bool): If True, prints a message for each validation loss␣
↪improvement.
                            Default: False
            delta (float): Minimum change in the monitored quantity to qualify␣
↪as an improvement.
                            Default: 0
            path (str): Path for the checkpoint to be saved to.
                            Default: 'checkpoint.pt'
            trace_func (function): trace print function.
                            Default: print
        """
        self.patience = patience
        self.verbose = verbose
        self.counter = 0
        self.best_score = None
        self.early_stop = False
        self.val_loss_min = np.Inf
        self.delta = delta
        self.path = path
        self.trace_func = trace_func
    def __call__(self, val_loss, model):

        score = -val_loss

        if self.best_score is None:
            self.best_score = score
```

```python
            self.save_checkpoint(val_loss, model)
        elif score < self.best_score + self.delta:
            self.counter += 1
            self.trace_func(f'> early stopping counter: {self.counter} out of␣
↪{self.patience}')
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_score = score
            self.save_checkpoint(val_loss, model)
            self.counter = 0

    def save_checkpoint(self, val_loss, model):
        '''Saves model when validation loss decrease.'''
        if self.verbose:
            self.trace_func(f'validation loss decreased ({self.val_loss_min:.
↪6f} --> {val_loss:.6f}).  Saving model ...')
        torch.save(model.state_dict(), self.path)
        self.val_loss_min = val_loss
```

[4]:

```python
def calculate_accuracy(output, target):
    "Calculates accuracy"
    output = output.data.max(dim=1,keepdim=True)[1]
    output = output == 1.0
    output = torch.flatten(output)
    target = target == 1.0
    target = torch.flatten(target)
    return torch.true_divide((target == output).sum(dim=0), output.size(0)).
↪item()
```

[5]:

```python
def save_model(model, optimizer, epoch, save_file):
    print('\n==> Saving...')
    state = {
        'model': model.state_dict(),
        'optimizer': optimizer.state_dict(),
        'epoch': epoch,
    }
    torch.save(state, save_file)
    del state
```

[6]:

```python
class TwoCropTransform:
    """Create two crops of the same image"""
    def __init__(self, transform):
        self.transform = transform

    def __call__(self, x):
        return [self.transform(x), self.transform(x)]
```

3 Method

[7]:
```python
class Encoder(torch.nn.Module):
    "Encoder network"
    def __init__(self):
        super(Encoder, self).__init__()
        # L1 (?, 28, 28, 3) -> (?, 28, 28, 32) -> (?, 14, 14, 32)
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.BatchNorm2d(32),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            torch.nn.Dropout(p=0.2)
            )
        # L2 (?, 14, 14, 32) -> (?, 14, 14, 64) -> (?, 7, 7, 64)
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.BatchNorm2d(64),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            torch.nn.Dropout(p=0.2)
            )
        # L3 (?, 7, 7, 64) -> (?, 7, 7, 128) -> (?, 4, 4, 128)
        self.layer3 = torch.nn.Sequential(
            torch.nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            torch.nn.BatchNorm2d(128),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
            torch.nn.Dropout(p=0.2)
            )
        self._to_linear = 4 * 4 * 128

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = x.view(x.size(0), -1) # Flatten them for FC
        return x


class LinearClassifier(torch.nn.Module):
    """Linear classifier"""
    def __init__(self):
        super(LinearClassifier, self).__init__()
        self.fc = torch.nn.Sequential(
            torch.nn.Linear(4 * 4 * 128, 10),
            )
```

```
    def forward(self, x):
        x = self.fc(x)
        probs = torch.nn.functional.softmax(x, dim=0)
        return probs
```

[8]:
```
class SupCon(nn.Module):
    """encoder + projection head"""
    def __init__(self, model, head='mlp', feat_dim=128):
        super(SupCon, self).__init__()

        self.dim_in = model._to_linear
        self.encoder = model

        if head == 'linear':
            self.head = nn.Linear(self.dim_in, feat_dim)
        elif head == 'mlp':
            self.head = nn.Sequential(
                nn.Linear(self.dim_in, self.dim_in),
                nn.ReLU(inplace=True),
                nn.Linear(self.dim_in, feat_dim)
            )
        else:
            raise NotImplementedError('Head not supported: {}'.format(head))

    def forward(self, x):
        feat = self.encoder(x)
        feat = F.normalize(self.head(feat), dim=1)
        return feat
```

[9]:
```
"""
Author: Yonglong Tian (yonglong@mit.edu)
Date: May 07, 2020
"""
class SupConLoss(nn.Module):
    def __init__(self, temperature=0.07, contrast_mode='all',
                 base_temperature=0.07):
        super(SupConLoss, self).__init__()
        self.temperature = temperature
        self.contrast_mode = contrast_mode
        self.base_temperature = base_temperature

    def forward(self, features, labels=None, mask=None):
        device = (torch.device('cuda')
                  if features.is_cuda
                  else torch.device('cpu'))
```

```python
        if len(features.shape) < 3:
            raise ValueError('`features` needs to be [bsz, n_views, ...],'
                             'at least 3 dimensions are required')
        if len(features.shape) > 3:
            features = features.view(features.shape[0], features.shape[1], -1)

        batch_size = features.shape[0]
        if labels is not None and mask is not None:
            raise ValueError('Cannot define both `labels` and `mask`')
        elif labels is None and mask is None:
            mask = torch.eye(batch_size, dtype=torch.float32).to(device)
        elif labels is not None:
            labels = labels.contiguous().view(-1, 1)
            if labels.shape[0] != batch_size:
                raise ValueError('Num of labels does not match num of features')
            mask = torch.eq(labels, labels.T).float().to(device)
        else:
            mask = mask.float().to(device)

        contrast_count = features.shape[1]
        contrast_feature = torch.cat(torch.unbind(features, dim=1), dim=0)
        if self.contrast_mode == 'one':
            anchor_feature = features[:, 0]
            anchor_count = 1
        elif self.contrast_mode == 'all':
            anchor_feature = contrast_feature
            anchor_count = contrast_count
        else:
            raise ValueError('Unknown mode: {}'.format(self.contrast_mode))

        # compute logits
        anchor_dot_contrast = torch.div(
            torch.matmul(anchor_feature, contrast_feature.T),
            self.temperature)
        # for numerical stability
        logits_max, _ = torch.max(anchor_dot_contrast, dim=1, keepdim=True)
        logits = anchor_dot_contrast - logits_max.detach()

        # tile mask
        mask = mask.repeat(anchor_count, contrast_count)
        # mask-out self-contrast cases
        logits_mask = torch.scatter(
            torch.ones_like(mask),
            1,
            torch.arange(batch_size * anchor_count).view(-1, 1).to(device),
            0
        )
```

```
        mask = mask * logits_mask

        # compute log_prob
        exp_logits = torch.exp(logits) * logits_mask
        log_prob = logits - torch.log(exp_logits.sum(1, keepdim=True))

        # compute mean of log-likelihood over positive
        mean_log_prob_pos = (mask * log_prob).sum(1) / mask.sum(1)

        # loss
        loss = - (self.temperature / self.base_temperature) * mean_log_prob_pos
        loss = loss.view(anchor_count, batch_size).mean()

        return loss
```

4 Training

[10]:
```
def pretraining(epoch, model, contrastive_loader, optimizer, criterion,␣
↪method='SimCLR'):
    "Contrastive pre-training over an epoch"
    metric_monitor = MetricMonitor()
    model.train()
    for batch_idx, (data,labels) in enumerate(contrastive_loader):
        # print(batch_idx)
        # print(data[0].shape)
        # print(data[1].shape)
        # print(labels.shape)
        data = torch.cat([data[0], data[1]], dim=0)
        # print(data.shape)
        if torch.cuda.is_available():
            data,labels = data.cuda(), labels.cuda()
        data, labels = torch.autograd.Variable(data,False), torch.autograd.
↪Variable(labels)
        bsz = labels.shape[0]
        features = model(data)
        f1, f2 = torch.split(features, [bsz, bsz], dim=0)
        features = torch.cat([f1.unsqueeze(1), f2.unsqueeze(1)], dim=1)
        if method == 'SupCon':
            loss = criterion(features, labels)
        elif method == 'SimCLR':
            loss = criterion(features)
        else:
            raise ValueError('contrastive method not supported: {}'.
↪format(method))
        metric_monitor.update("Loss", loss.item())
        metric_monitor.update("Learning Rate", optimizer.param_groups[0]['lr'])
```

```
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print("[Epoch: {epoch:03d}] Contrastive Pre-train | {metric_monitor}".
↪format(epoch=epoch, metric_monitor=metric_monitor))
    return metric_monitor.metrics['Loss']['avg'], metric_monitor.
↪metrics['Learning Rate']['avg']
```

[11]:
```
def training(epoch, model, classifier, train_loader, optimizer, criterion):
    "Training over an epoch"
    metric_monitor = MetricMonitor()
    model.eval()
    classifier.train()
    for batch_idx, (data,labels) in enumerate(train_loader):
        # print(batch_idx)
        # print(data.shape)
        # print(labels.shape)
        if torch.cuda.is_available():
            data,labels = data.cuda(), labels.cuda()
        data, labels = torch.autograd.Variable(data,False), torch.autograd.
↪Variable(labels)
        with torch.no_grad():
            features = model.encoder(data)
        output = classifier(features.float())
        loss = criterion(output, labels)
        accuracy = calculate_accuracy(output, labels)
        metric_monitor.update("Loss", loss.item())
        metric_monitor.update("Accuracy", accuracy)
        data.detach()
        labels.detach()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print("[Epoch: {epoch:03d}] Train       | {metric_monitor}".
↪format(epoch=epoch, metric_monitor=metric_monitor))
    return metric_monitor.metrics['Loss']['avg'], metric_monitor.
↪metrics['Accuracy']['avg']
```

[12]:
```
def validation(epoch, model, classifier, valid_loader, criterion):
    "Validation over an epoch"
    metric_monitor = MetricMonitor()
    model.eval()
    classifier.eval()
    with torch.no_grad():
        for batch_idx, (data,labels) in enumerate(valid_loader):
            if torch.cuda.is_available():
                data,labels = data.cuda(), labels.cuda()
```

```
            data, labels = torch.autograd.Variable(data,False), torch.autograd.
↪Variable(labels)
            features = model.encoder(data)
            output = classifier(features.float())
            loss = criterion(output,labels)
            accuracy = calculate_accuracy(output, labels)
            metric_monitor.update("Loss", loss.item())
            metric_monitor.update("Accuracy", accuracy)
            data.detach()
            labels.detach()
    print("[Epoch: {epoch:03d}] Validation | {metric_monitor}".
↪format(epoch=epoch, metric_monitor=metric_monitor))
    return metric_monitor.metrics['Loss']['avg'], metric_monitor.
↪metrics['Accuracy']['avg']
```

5 Data

[13]:
```
contrastive_transform = transforms.Compose([
  transforms.Grayscale(num_output_channels = 3),  # Convert to 3 color channel
  transforms.RandomHorizontalFlip(),
  transforms.RandomResizedCrop(size=28, scale=(0.2, 1.)),
  transforms.ToTensor(),
  transforms.Normalize((0.5,), (0.5,)),
])
train_transform = transforms.Compose([
  transforms.Grayscale(num_output_channels = 3),  # Convert to 3 color channel
  transforms.RandomHorizontalFlip(),
  transforms.ToTensor(),
  transforms.Normalize((0.5,), (0.5,)),
])
valid_transform = transforms.Compose([
  transforms.Grayscale(num_output_channels = 3),  # Convert to 3 color channel
  transforms.ToTensor(),
  transforms.Normalize((0.5,), (0.5,)),
])

contrastive_set = datasets.MNIST('./data', download=True, train=True,␣
↪transform=TwoCropTransform(contrastive_transform))
train_set = datasets.MNIST('./data', download=True, train=True,␣
↪transform=train_transform)
valid_set = datasets.MNIST('./data', download=True, train=False,␣
↪transform=valid_transform)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
./data/MNIST/raw/train-images-idx3-ubyte.gz
```

```
100%|          | 9912422/9912422 [00:00<00:00, 176856960.98it/s]

Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw


Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
./data/MNIST/raw/train-labels-idx1-ubyte.gz

100%|          | 28881/28881 [00:00<00:00, 33124335.20it/s]

Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
./data/MNIST/raw/t10k-images-idx3-ubyte.gz


100%|          | 1648877/1648877 [00:00<00:00, 39444547.47it/s]

Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
./data/MNIST/raw/t10k-labels-idx1-ubyte.gz

100%|          | 4542/4542 [00:00<00:00, 5719162.04it/s]

Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

[14]:

```python
print(contrastive_set)
print(contrastive_set.classes)
print(len(contrastive_set.classes))
print(len(contrastive_set))
print(len(contrastive_set[0])) # 2 entries (2 crops of same img for a given␣
 ↪label)
print(len(contrastive_set[0][0])) # img, label
print(len(contrastive_set[0][0][0])) # color channel
print(len(contrastive_set[0][0][0][0])) # dim
print(len(contrastive_set[0][0][0][0][0])) # dim
```

```
Dataset MNIST
    Number of datapoints: 60000
    Root location: ./data
    Split: Train
    StandardTransform
Transform: <__main__.TwoCropTransform object at 0x7913f5092a70>
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 -
six', '7 - seven', '8 - eight', '9 - nine']
```

```
10
60000
2
2
3
28
28
```

[15]:
```python
print(train_set)
print(train_set.classes)
print(len(train_set.classes))
print(len(train_set))
print(len(train_set[0])) # img, label
print(len(train_set[0][0])) # color channel
print(len(train_set[0][0][0])) # dim
print(len(train_set[0][0][0][0])) # dim
```

```
Dataset MNIST
    Number of datapoints: 60000
    Root location: ./data
    Split: Train
    StandardTransform
Transform: Compose(
               Grayscale(num_output_channels=3)
               RandomHorizontalFlip(p=0.5)
               ToTensor()
               Normalize(mean=(0.5,), std=(0.5,))
           )
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 -
six', '7 - seven', '8 - eight', '9 - nine']
10
60000
2
3
28
28
```

[16]:
```python
print(valid_set)
print(valid_set.classes)
print(len(valid_set.classes))
print(len(valid_set))
print(len(valid_set[0])) # img, label
print(len(valid_set[0][0])) # color channel
print(len(valid_set[0][0][0])) # dim
print(len(valid_set[0][0][0][0])) # dim
```

```
Dataset MNIST
    Number of datapoints: 10000
```

```
    Root location: ./data
    Split: Test
    StandardTransform
Transform: Compose(
               Grayscale(num_output_channels=3)
               ToTensor()
               Normalize(mean=(0.5,), std=(0.5,))
           )
['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 -
six', '7 - seven', '8 - eight', '9 - nine']
10
10000
2
3
28
28
```

[33]:
```python
import torchvision.transforms as T
T.ToPILImage()(contrastive_set[0][0][0]).resize((50,50))
```

[33]:

6 4. Main

[36]:
```python
import os

def main():

    num_epochs = 50
    use_early_stopping = True
    use_scheduler = True
    head_type = 'mlp' # choose among 'mlp' and 'linear'
    save_file = os.path.join('./results/', 'model.pth')
    if not os.path.isdir('./results/'):
         os.makedirs('./results/')

    contrastive_loader = torch.utils.data.DataLoader(contrastive_set,␣
↪batch_size=64, shuffle=True)
    train_loader = torch.utils.data.DataLoader(train_set, batch_size=64,␣
↪shuffle=True)
    valid_loader = torch.utils.data.DataLoader(valid_set, batch_size=64,␣
↪shuffle=True)
```

```
# Part 1
encoder = Encoder()
model = SupCon(encoder, head=head_type, feat_dim=128)
criterion = SupConLoss(temperature=0.07)
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, betas=(0.9, 0.
↪999), eps=1e-08, weight_decay=1e-3)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=20,␣
↪gamma=0.9)

contrastive_loss, contrastive_lr = [], []

for epoch in range(1, num_epochs+1):
    print('Epoch {} start'.format(epoch))
    loss, lr = pretraining(epoch, model, contrastive_loader, optimizer,␣
↪criterion, method='SimCLR')
    if use_scheduler:
        scheduler.step()
    contrastive_loss.append(loss)
    contrastive_lr.append(lr)
    print('Epoch {} end'.format(epoch))

save_model(model, optimizer, num_epochs, save_file)

plt.plot(range(1,len(contrastive_lr)+1),contrastive_lr, color='b', label =␣
↪'learning rate')
plt.legend(), plt.ylabel('loss'), plt.xlabel('epochs'), plt.title('Learning␣
↪Rate'), plt.show()

plt.plot(range(1,len(contrastive_loss)+1),contrastive_loss, color='b',␣
↪label = 'loss')
plt.legend(), plt.ylabel('loss'), plt.xlabel('epochs'), plt.title('Loss'),␣
↪plt.show()

# Part 2
model = SupCon(encoder, head=head_type, feat_dim=128)
classifier = LinearClassifier()
criterion = torch.nn.CrossEntropyLoss()

ckpt = torch.load(save_file, map_location='cpu')
state_dict = ckpt['model']
new_state_dict = {}
for k, v in state_dict.items():
```

```python
        k = k.replace("module.", "")
        new_state_dict[k] = v
    state_dict = new_state_dict
    model.load_state_dict(state_dict)

    if torch.cuda.is_available():
        model = model.cuda()
        classifier = classifier.cuda()
        criterion = criterion.cuda()

    train_losses , train_accuracies = [],[]
    valid_losses , valid_accuracies = [],[]

    if use_early_stopping:
        early_stopping = EarlyStopping(patience=30, verbose=False, delta=1e-4)

    for epoch in range(1, num_epochs+1):
        print('Epoch {} start'.format(epoch))

        train_loss, train_accuracy = training(epoch, model, classifier,␣
↪train_loader, optimizer, criterion)
        valid_loss, valid_accuracy = validation(epoch, model, classifier,␣
↪valid_loader, criterion)

        if use_scheduler:
            scheduler.step()

        train_losses.append(train_loss)
        train_accuracies.append(train_accuracy)
        valid_losses.append(valid_loss)
        valid_accuracies.append(valid_accuracy)

        print('Epoch {} end'.format(epoch))

        if use_early_stopping:
            early_stopping(valid_loss, model)

            if early_stopping.early_stop:
                print('Early stopping at epoch', epoch)
                break

    plt.plot(range(1,len(train_losses)+1), train_losses, color='b', label =␣
↪'training loss')
    plt.plot(range(1,len(valid_losses)+1), valid_losses, color='r',␣
↪linestyle='dashed', label = 'validation loss')
    plt.legend(), plt.ylabel('loss'), plt.xlabel('epochs'), plt.title('Loss'),␣
↪plt.show()
```

```
    plt.plot(range(1,len(train_accuracies)+1),train_accuracies, color='b',␣
↪label = 'training accuracy')
    plt.plot(range(1,len(valid_accuracies)+1),valid_accuracies, color='r',␣
↪linestyle='dashed', label = 'validation accuracy')
    plt.legend(), plt.ylabel('loss'), plt.xlabel('epochs'), plt.
↪title('Accuracy'), plt.show()
```

[37]:
```
main()
```

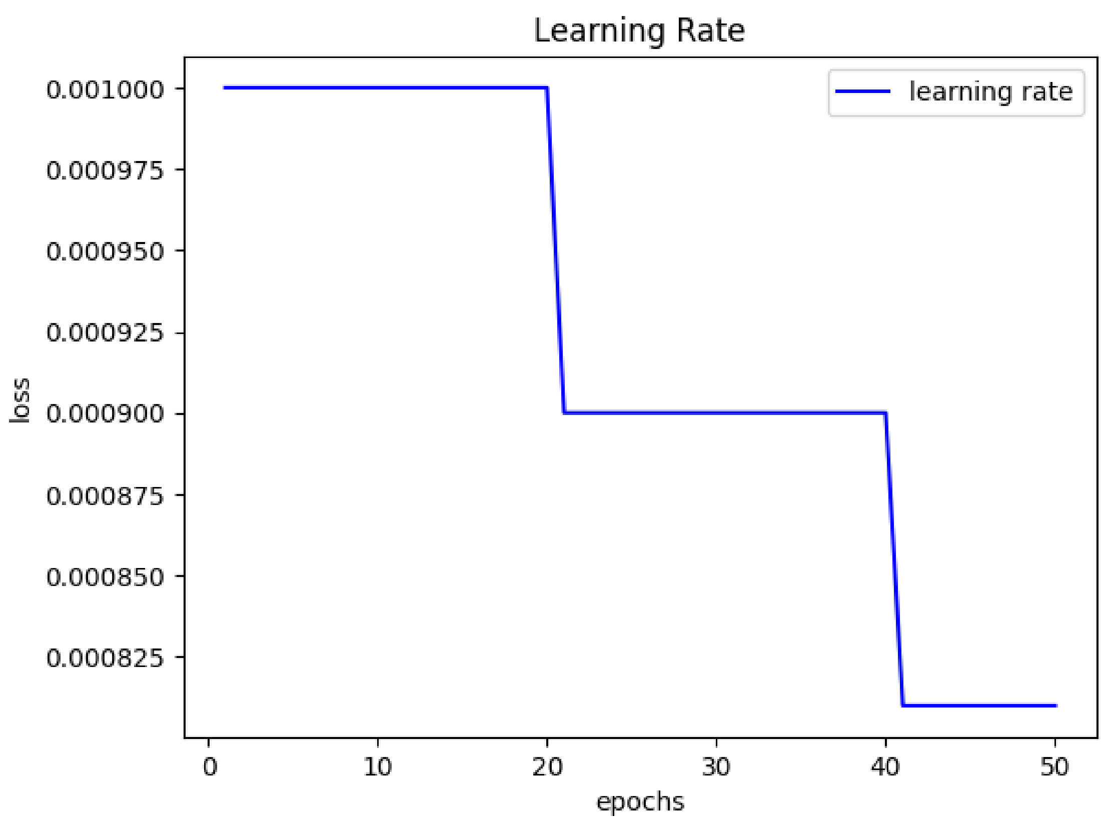
```
Epoch 1 start
[Epoch: 001] Contrastive Pre-train | Loss: 2.6896 | Learning Rate: 0.0010
Epoch 1 end
Epoch 2 start
[Epoch: 002] Contrastive Pre-train | Loss: 1.7506 | Learning Rate: 0.0010
Epoch 2 end
Epoch 3 start
[Epoch: 003] Contrastive Pre-train | Loss: 1.4733 | Learning Rate: 0.0010
Epoch 3 end
Epoch 4 start
[Epoch: 004] Contrastive Pre-train | Loss: 1.3309 | Learning Rate: 0.0010
Epoch 4 end
Epoch 5 start
[Epoch: 005] Contrastive Pre-train | Loss: 1.2423 | Learning Rate: 0.0010
Epoch 5 end
Epoch 6 start
[Epoch: 006] Contrastive Pre-train | Loss: 1.1636 | Learning Rate: 0.0010
Epoch 6 end
Epoch 7 start
[Epoch: 007] Contrastive Pre-train | Loss: 1.1083 | Learning Rate: 0.0010
Epoch 7 end
Epoch 8 start
[Epoch: 008] Contrastive Pre-train | Loss: 1.0682 | Learning Rate: 0.0010
Epoch 8 end
Epoch 9 start
[Epoch: 009] Contrastive Pre-train | Loss: 1.0319 | Learning Rate: 0.0010
Epoch 9 end
Epoch 10 start
[Epoch: 010] Contrastive Pre-train | Loss: 1.0043 | Learning Rate: 0.0010
Epoch 10 end
Epoch 11 start
[Epoch: 011] Contrastive Pre-train | Loss: 0.9945 | Learning Rate: 0.0010
Epoch 11 end
Epoch 12 start
[Epoch: 012] Contrastive Pre-train | Loss: 0.9756 | Learning Rate: 0.0010
Epoch 12 end
Epoch 13 start
```

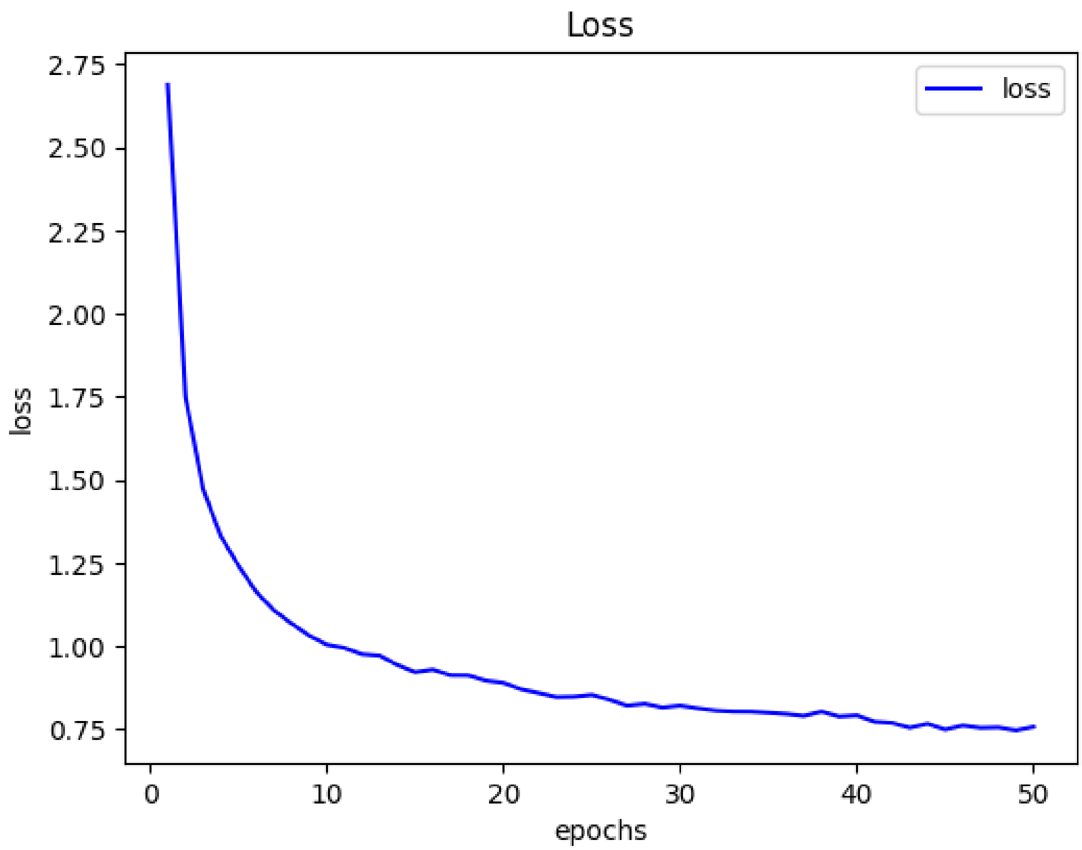
```
[Epoch: 013] Contrastive Pre-train | Loss: 0.9712 | Learning Rate: 0.0010
Epoch 13 end
Epoch 14 start
[Epoch: 014] Contrastive Pre-train | Loss: 0.9434 | Learning Rate: 0.0010
Epoch 14 end
Epoch 15 start
[Epoch: 015] Contrastive Pre-train | Loss: 0.9217 | Learning Rate: 0.0010
Epoch 15 end
Epoch 16 start
[Epoch: 016] Contrastive Pre-train | Loss: 0.9292 | Learning Rate: 0.0010
Epoch 16 end
Epoch 17 start
[Epoch: 017] Contrastive Pre-train | Loss: 0.9125 | Learning Rate: 0.0010
Epoch 17 end
Epoch 18 start
[Epoch: 018] Contrastive Pre-train | Loss: 0.9121 | Learning Rate: 0.0010
Epoch 18 end
Epoch 19 start
[Epoch: 019] Contrastive Pre-train | Loss: 0.8963 | Learning Rate: 0.0010
Epoch 19 end
Epoch 20 start
[Epoch: 020] Contrastive Pre-train | Loss: 0.8896 | Learning Rate: 0.0010
Epoch 20 end
Epoch 21 start
[Epoch: 021] Contrastive Pre-train | Loss: 0.8702 | Learning Rate: 0.0009
Epoch 21 end
Epoch 22 start
[Epoch: 022] Contrastive Pre-train | Loss: 0.8589 | Learning Rate: 0.0009
Epoch 22 end
Epoch 23 start
[Epoch: 023] Contrastive Pre-train | Loss: 0.8467 | Learning Rate: 0.0009
Epoch 23 end
Epoch 24 start
[Epoch: 024] Contrastive Pre-train | Loss: 0.8478 | Learning Rate: 0.0009
Epoch 24 end
Epoch 25 start
[Epoch: 025] Contrastive Pre-train | Loss: 0.8534 | Learning Rate: 0.0009
Epoch 25 end
Epoch 26 start
[Epoch: 026] Contrastive Pre-train | Loss: 0.8387 | Learning Rate: 0.0009
Epoch 26 end
Epoch 27 start
[Epoch: 027] Contrastive Pre-train | Loss: 0.8208 | Learning Rate: 0.0009
Epoch 27 end
Epoch 28 start
[Epoch: 028] Contrastive Pre-train | Loss: 0.8265 | Learning Rate: 0.0009
Epoch 28 end
Epoch 29 start
```

```
[Epoch: 029] Contrastive Pre-train | Loss: 0.8152 | Learning Rate: 0.0009
Epoch 29 end
Epoch 30 start
[Epoch: 030] Contrastive Pre-train | Loss: 0.8211 | Learning Rate: 0.0009
Epoch 30 end
Epoch 31 start
[Epoch: 031] Contrastive Pre-train | Loss: 0.8127 | Learning Rate: 0.0009
Epoch 31 end
Epoch 32 start
[Epoch: 032] Contrastive Pre-train | Loss: 0.8060 | Learning Rate: 0.0009
Epoch 32 end
Epoch 33 start
[Epoch: 033] Contrastive Pre-train | Loss: 0.8032 | Learning Rate: 0.0009
Epoch 33 end
Epoch 34 start
[Epoch: 034] Contrastive Pre-train | Loss: 0.8028 | Learning Rate: 0.0009
Epoch 34 end
Epoch 35 start
[Epoch: 035] Contrastive Pre-train | Loss: 0.7995 | Learning Rate: 0.0009
Epoch 35 end
Epoch 36 start
[Epoch: 036] Contrastive Pre-train | Loss: 0.7959 | Learning Rate: 0.0009
Epoch 36 end
Epoch 37 start
[Epoch: 037] Contrastive Pre-train | Loss: 0.7900 | Learning Rate: 0.0009
Epoch 37 end
Epoch 38 start
[Epoch: 038] Contrastive Pre-train | Loss: 0.8028 | Learning Rate: 0.0009
Epoch 38 end
Epoch 39 start
[Epoch: 039] Contrastive Pre-train | Loss: 0.7881 | Learning Rate: 0.0009
Epoch 39 end
Epoch 40 start
[Epoch: 040] Contrastive Pre-train | Loss: 0.7916 | Learning Rate: 0.0009
Epoch 40 end
Epoch 41 start
[Epoch: 041] Contrastive Pre-train | Loss: 0.7725 | Learning Rate: 0.0008
Epoch 41 end
Epoch 42 start
[Epoch: 042] Contrastive Pre-train | Loss: 0.7688 | Learning Rate: 0.0008
Epoch 42 end
Epoch 43 start
[Epoch: 043] Contrastive Pre-train | Loss: 0.7546 | Learning Rate: 0.0008
Epoch 43 end
Epoch 44 start
[Epoch: 044] Contrastive Pre-train | Loss: 0.7666 | Learning Rate: 0.0008
Epoch 44 end
Epoch 45 start
```

```
[Epoch: 045] Contrastive Pre-train | Loss: 0.7491 | Learning Rate: 0.0008
Epoch 45 end
Epoch 46 start
[Epoch: 046] Contrastive Pre-train | Loss: 0.7613 | Learning Rate: 0.0008
Epoch 46 end
Epoch 47 start
[Epoch: 047] Contrastive Pre-train | Loss: 0.7541 | Learning Rate: 0.0008
Epoch 47 end
Epoch 48 start
[Epoch: 048] Contrastive Pre-train | Loss: 0.7554 | Learning Rate: 0.0008
Epoch 48 end
Epoch 49 start
[Epoch: 049] Contrastive Pre-train | Loss: 0.7467 | Learning Rate: 0.0008
Epoch 49 end
Epoch 50 start
[Epoch: 050] Contrastive Pre-train | Loss: 0.7570 | Learning Rate: 0.0008
Epoch 50 end

==> Saving…
```



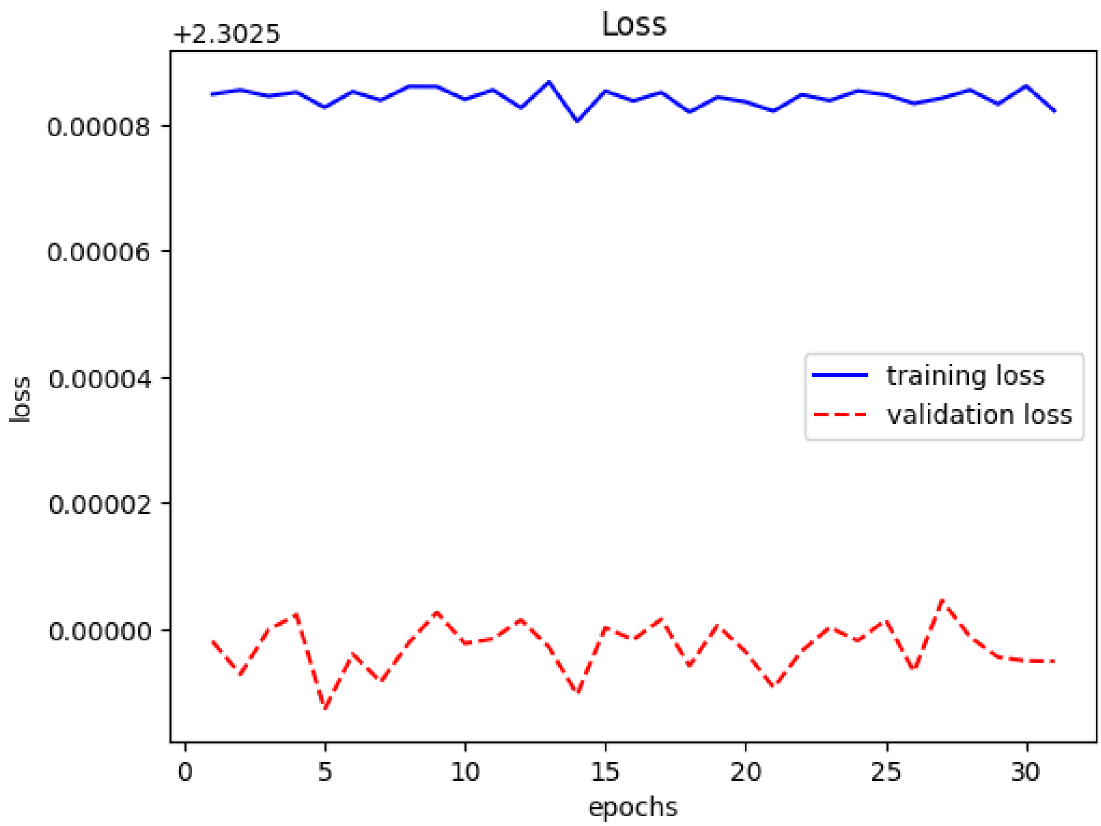


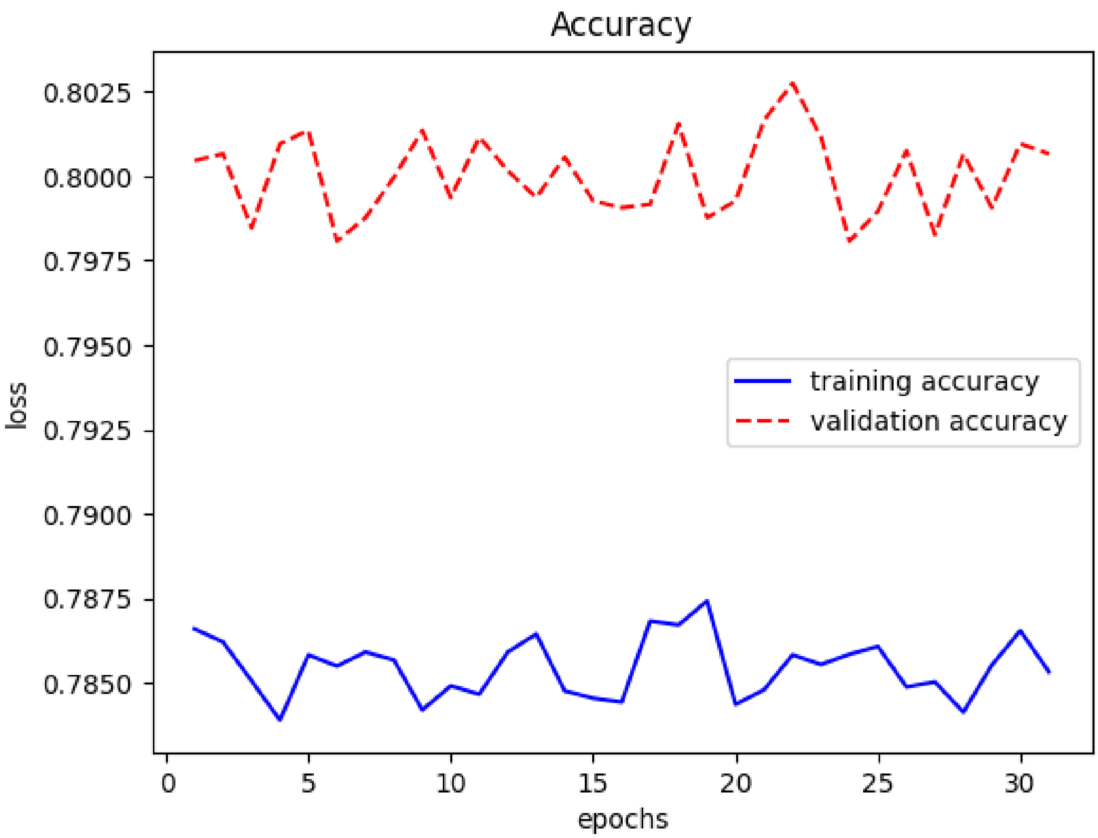

```
Epoch 1 start
[Epoch: 001] Train      | Loss: 2.3026 | Accuracy: 0.7866
[Epoch: 001] Validation | Loss: 2.3025 | Accuracy: 0.8005
Epoch 1 end
Epoch 2 start
[Epoch: 002] Train      | Loss: 2.3026 | Accuracy: 0.7862
[Epoch: 002] Validation | Loss: 2.3025 | Accuracy: 0.8007
Epoch 2 end
> early stopping counter: 1 out of 30
Epoch 3 start
[Epoch: 003] Train      | Loss: 2.3026 | Accuracy: 0.7851
[Epoch: 003] Validation | Loss: 2.3025 | Accuracy: 0.7985
Epoch 3 end
> early stopping counter: 2 out of 30
Epoch 4 start
[Epoch: 004] Train      | Loss: 2.3026 | Accuracy: 0.7839
[Epoch: 004] Validation | Loss: 2.3025 | Accuracy: 0.8010
Epoch 4 end
> early stopping counter: 3 out of 30
Epoch 5 start
```

```
[Epoch: 005] Train      | Loss: 2.3026 | Accuracy: 0.7858
[Epoch: 005] Validation | Loss: 2.3025 | Accuracy: 0.8014
Epoch 5 end
> early stopping counter: 4 out of 30
Epoch 6 start
[Epoch: 006] Train      | Loss: 2.3026 | Accuracy: 0.7855
[Epoch: 006] Validation | Loss: 2.3025 | Accuracy: 0.7981
Epoch 6 end
> early stopping counter: 5 out of 30
Epoch 7 start
[Epoch: 007] Train      | Loss: 2.3026 | Accuracy: 0.7859
[Epoch: 007] Validation | Loss: 2.3025 | Accuracy: 0.7988
Epoch 7 end
> early stopping counter: 6 out of 30
Epoch 8 start
[Epoch: 008] Train      | Loss: 2.3026 | Accuracy: 0.7857
[Epoch: 008] Validation | Loss: 2.3025 | Accuracy: 0.8000
Epoch 8 end
> early stopping counter: 7 out of 30
Epoch 9 start
[Epoch: 009] Train      | Loss: 2.3026 | Accuracy: 0.7842
[Epoch: 009] Validation | Loss: 2.3025 | Accuracy: 0.8014
Epoch 9 end
> early stopping counter: 8 out of 30
Epoch 10 start
[Epoch: 010] Train      | Loss: 2.3026 | Accuracy: 0.7849
[Epoch: 010] Validation | Loss: 2.3025 | Accuracy: 0.7994
Epoch 10 end
> early stopping counter: 9 out of 30
Epoch 11 start
[Epoch: 011] Train      | Loss: 2.3026 | Accuracy: 0.7847
[Epoch: 011] Validation | Loss: 2.3025 | Accuracy: 0.8012
Epoch 11 end
> early stopping counter: 10 out of 30
Epoch 12 start
[Epoch: 012] Train      | Loss: 2.3026 | Accuracy: 0.7859
[Epoch: 012] Validation | Loss: 2.3025 | Accuracy: 0.8002
Epoch 12 end
> early stopping counter: 11 out of 30
Epoch 13 start
[Epoch: 013] Train      | Loss: 2.3026 | Accuracy: 0.7864
[Epoch: 013] Validation | Loss: 2.3025 | Accuracy: 0.7994
Epoch 13 end
> early stopping counter: 12 out of 30
Epoch 14 start
[Epoch: 014] Train      | Loss: 2.3026 | Accuracy: 0.7848
[Epoch: 014] Validation | Loss: 2.3025 | Accuracy: 0.8006
Epoch 14 end
```

```
> early stopping counter: 13 out of 30
Epoch 15 start
[Epoch: 015] Train      | Loss: 2.3026 | Accuracy: 0.7845
[Epoch: 015] Validation | Loss: 2.3025 | Accuracy: 0.7993
Epoch 15 end
> early stopping counter: 14 out of 30
Epoch 16 start
[Epoch: 016] Train      | Loss: 2.3026 | Accuracy: 0.7844
[Epoch: 016] Validation | Loss: 2.3025 | Accuracy: 0.7991
Epoch 16 end
> early stopping counter: 15 out of 30
Epoch 17 start
[Epoch: 017] Train      | Loss: 2.3026 | Accuracy: 0.7868
[Epoch: 017] Validation | Loss: 2.3025 | Accuracy: 0.7992
Epoch 17 end
> early stopping counter: 16 out of 30
Epoch 18 start
[Epoch: 018] Train      | Loss: 2.3026 | Accuracy: 0.7867
[Epoch: 018] Validation | Loss: 2.3025 | Accuracy: 0.8016
Epoch 18 end
> early stopping counter: 17 out of 30
Epoch 19 start
[Epoch: 019] Train      | Loss: 2.3026 | Accuracy: 0.7874
[Epoch: 019] Validation | Loss: 2.3025 | Accuracy: 0.7988
Epoch 19 end
> early stopping counter: 18 out of 30
Epoch 20 start
[Epoch: 020] Train      | Loss: 2.3026 | Accuracy: 0.7844
[Epoch: 020] Validation | Loss: 2.3025 | Accuracy: 0.7993
Epoch 20 end
> early stopping counter: 19 out of 30
Epoch 21 start
[Epoch: 021] Train      | Loss: 2.3026 | Accuracy: 0.7848
[Epoch: 021] Validation | Loss: 2.3025 | Accuracy: 0.8017
Epoch 21 end
> early stopping counter: 20 out of 30
Epoch 22 start
[Epoch: 022] Train      | Loss: 2.3026 | Accuracy: 0.7858
[Epoch: 022] Validation | Loss: 2.3025 | Accuracy: 0.8027
Epoch 22 end
> early stopping counter: 21 out of 30
Epoch 23 start
[Epoch: 023] Train      | Loss: 2.3026 | Accuracy: 0.7855
[Epoch: 023] Validation | Loss: 2.3025 | Accuracy: 0.8012
Epoch 23 end
> early stopping counter: 22 out of 30
Epoch 24 start
[Epoch: 024] Train      | Loss: 2.3026 | Accuracy: 0.7858
```

```
[Epoch: 024] Validation | Loss: 2.3025 | Accuracy: 0.7981
Epoch 24 end
> early stopping counter: 23 out of 30
Epoch 25 start
[Epoch: 025] Train      | Loss: 2.3026 | Accuracy: 0.7861
[Epoch: 025] Validation | Loss: 2.3025 | Accuracy: 0.7990
Epoch 25 end
> early stopping counter: 24 out of 30
Epoch 26 start
[Epoch: 026] Train      | Loss: 2.3026 | Accuracy: 0.7849
[Epoch: 026] Validation | Loss: 2.3025 | Accuracy: 0.8008
Epoch 26 end
> early stopping counter: 25 out of 30
Epoch 27 start
[Epoch: 027] Train      | Loss: 2.3026 | Accuracy: 0.7850
[Epoch: 027] Validation | Loss: 2.3025 | Accuracy: 0.7983
Epoch 27 end
> early stopping counter: 26 out of 30
Epoch 28 start
[Epoch: 028] Train      | Loss: 2.3026 | Accuracy: 0.7841
[Epoch: 028] Validation | Loss: 2.3025 | Accuracy: 0.8007
Epoch 28 end
> early stopping counter: 27 out of 30
Epoch 29 start
[Epoch: 029] Train      | Loss: 2.3026 | Accuracy: 0.7855
[Epoch: 029] Validation | Loss: 2.3025 | Accuracy: 0.7991
Epoch 29 end
> early stopping counter: 28 out of 30
Epoch 30 start
[Epoch: 030] Train      | Loss: 2.3026 | Accuracy: 0.7865
[Epoch: 030] Validation | Loss: 2.3025 | Accuracy: 0.8010
Epoch 30 end
> early stopping counter: 29 out of 30
Epoch 31 start
[Epoch: 031] Train      | Loss: 2.3026 | Accuracy: 0.7853
[Epoch: 031] Validation | Loss: 2.3025 | Accuracy: 0.8007
Epoch 31 end
> early stopping counter: 30 out of 30
Early stopping at epoch 31
```


Loss
+2.3025
0.00008
0.00006
0.00004
0.00002
0.00000
loss
training loss
validation loss
0
5
10
15
20
25
30
epochs


Accuracy
0.8025
0.8000
0.7975
0.7950
0.7925
0.7900
0.7875
0.7850
loss
training accuracy
validation accuracy
0
5
10
15
20
25
30
epochs