7123_miniproject_resnet

April 12, 2024

[1]:
```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
```

[2]:
```
def set_seed(seed = 16):
    np.random.seed(seed)
    torch.manual_seed(seed)
```

[ ]:
```
```

[ ]:
```
```

[4]:
```
"""
Cutout is used from https://github.com/uoguelph-mlrg/Cutout
"""
class Cutout(object):
    """Randomly mask out one or more patches from an image.

    Args:
        n_holes (int): Number of patches to cut out of each image.
        length (int): The length (in pixels) of each square patch.
    """
    def __init__(self, n_holes, length):
        self.n_holes = n_holes
        self.length = length

    def __call__(self, img):
        """
        Args:
            img (Tensor): Tensor image of size (C, H, W).
        Returns:
            Tensor: Image with n_holes of dimension length x length cut out of␣
 ↪it.
        """
        h = img.size(1)
```

```
        w = img.size(2)

        mask = np.ones((h, w), np.float32)

        for n in range(self.n_holes):
            y = np.random.randint(h)
            x = np.random.randint(w)

            y1 = np.clip(y - self.length // 2, 0, h)
            y2 = np.clip(y + self.length // 2, 0, h)
            x1 = np.clip(x - self.length // 2, 0, w)
            x2 = np.clip(x + self.length // 2, 0, w)

            mask[y1: y2, x1: x2] = 0.

        mask = torch.from_numpy(mask)
        mask = mask.expand_as(img)
        img = img * mask

        return img
```

[5]:
```
# Original image
normalize =  transforms.Normalize(mean=[x / 255.0 for x in [125.3, 123.0, 113.
 ↪9]],std=[x / 255.0 for x in [63.0, 62.1, 66.7]])
t0 = transforms.Compose([
                         transforms.ToTensor(),
                         transforms.Normalize((0.4914, 0.4822, 0.4465), (0.
 ↪2471, 0.2435, 0.2616)),
])
# Crop
t1 = transforms.Compose([
                         transforms.RandomResizedCrop(32,(0.8,1.0)),
                         transforms.ToTensor(),

])
# Vertical flip
t2 = transforms.Compose([
                         transforms.RandomVerticalFlip(),
                         transforms.ToTensor(),



])
# Horizontal flip
t3 = transforms.Compose([
                         transforms.RandomHorizontalFlip(),
                         transforms.ToTensor(),

```

```
])
# Distort
t4 = transforms.Compose([
                        transforms.ColorJitter(3),
                        transforms.ToTensor(),

])
# Rotate
t5 = transforms.Compose([
                        transforms.RandomRotation(270),
                        transforms.ToTensor(),

])
# Cutout
t6 = transforms.Compose([
                        transforms.ToTensor(),
                        Cutout(n_holes=1, length=16),

])
#GaussianBlur
t7 = transforms.Compose([
                        transforms.GaussianBlur(3),
                        transforms.ToTensor(),

])
t136 = transforms.Compose([
                          transforms.RandomResizedCrop(32,(0.8,1.0)),
                          transforms.RandomHorizontalFlip(),
                          transforms.ToTensor(),
                          Cutout(n_holes=1, length=16),
                          transforms.Normalize((0.4914, 0.4822, 0.4465), (0.
↪2471, 0.2435, 0.2616)),
])
```

[ ]:
```
trainingdata = torchvision.datasets.CIFAR10('./CIFAR-10/
↪',train=True,download=True,transform=t136)
testdata = torchvision.datasets.CIFAR10('./CIFAR-10/
↪',train=False,download=True,transform=t0)
```

[7]:
```
print(len(trainingdata))
print(len(testdata))
```

```
50000
10000
```

[8]:
```
trainDataLoader = torch.utils.data.
↪DataLoader(trainingdata,batch_size=64,shuffle=True)
```

```
testDataLoader = torch.utils.data.
↪DataLoader(testdata,batch_size=64,shuffle=False)
```

[9]:

```
# Model architecture
class BasicBlock(torch.nn.Module):
  expansion = 1

  def __init__(self, in_planes,planes, stride = 1):
    super(BasicBlock, self).__init__()
    self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride,␣
↪padding = 1, bias = False)
    self.bn1 = nn.BatchNorm2d(planes)
    self.conv2 = nn.Conv2d(planes, planes, kernel_size = 3, stride = 1, padding␣
↪= 1, bias = False)
    self.bn2 = nn.BatchNorm2d(planes)

    self.shortcut = nn.Sequential()
    if (stride != 1 or in_planes != self.expansion*planes):
      self.shortcut = nn.Sequential(nn.Conv2d(in_planes,self.
↪expansion*planes,kernel_size=1,stride=stride,bias=False),
                                    nn.BatchNorm2d(self.expansion*planes)
                                    )

  def forward(self, x):
    out = F.gelu(self.bn1(self.conv1(x)))
    out = self.bn2(self.conv2(out))
    out += self.shortcut(x)
    out = F.gelu(out)
    return out

class ResNet(nn.Module):
  def __init__(self,block,num_blocks, num_classes = 10):
    super(ResNet, self).__init__()
    self.in_planes = 32
    self.conv1 = nn.Conv2d(3,32,kernel_size=3,stride = 1, padding = 1, bias =␣
↪False)
    self.bn1 = nn.BatchNorm2d(32)
    self.layer1 = self._make_layer(block, 32, num_blocks[0], stride = 1)
    self.layer2 = self._make_layer(block, 64, num_blocks[1], stride = 2)
    self.layer3 = self._make_layer(block, 128, num_blocks[2], stride = 2)
    self.layer4 = self._make_layer(block, 256, num_blocks[3], stride = 2)
    self.linear = nn.Linear(256*block.expansion,num_classes)


  def _make_layer(self, block, planes, num_blocks, stride):
    strides = [stride] + [1]*(num_blocks-1)
    layers = []
```

```python
    for stride in strides:
      layers.append(block(self.in_planes,planes,stride))
      self.in_planes = planes * block.expansion
    return nn.Sequential(*layers)

  def forward(self, x):
    out = F.gelu(self.bn1(self.conv1(x)))
    out = self.layer1(out)
    out = self.layer2(out)
    out = self.layer3(out)
    out = self.layer4(out)
    out = F.avg_pool2d(out,4)
    out = out.view(out.size(0), -1)
    out = self.linear(out)
    return out

ResNet18 = ResNet(BasicBlock,[3,2,4,3]).cuda()
Loss = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(ResNet18.parameters(), lr=0.01, betas=(0.9, 0.
 ↪999), eps=1e-08)
#optimizer = torch.optim.SGD(ResNet18.parameters(), lr=0.05, momentum=0.9)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max = 200)
start_epoch = 0
best_acc = 0
```

[10]:
```python
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(count_parameters(ResNet18))
```

```
4587690
```

[11]:
```python
print(count_parameters(ResNet18))
```

```
4587690
```

[12]:
```python
train_loss_history = []
train_accuracy_history = []
test_loss_history = []
test_accuracy_history = []
```

[13]:
```python
def train(epoch):
  print('\nEpoch: %d' %epoch)
  ResNet18.train()
  train_loss = 0
  correct = 0
  total = 0
  for idx, (images,labels) in enumerate(trainDataLoader):
```

```
    images = images.cuda()
    labels = labels.cuda()
    optimizer.zero_grad()
    outputs = ResNet18(images)
    loss = Loss(outputs,labels)
    loss.backward()
    optimizer.step()

    train_loss += loss.item()
    _, predict = outputs.max(1)
    total += labels.size(0)
    correct += predict.eq(labels).sum().item()
  print('Train loss: {:.2f}, Train accuracy:{:.2f} %'.format(train_loss/
 ↪len(trainDataLoader),correct/total*100))
  train_loss_history.append(train_loss/len(trainDataLoader))
  train_accuracy_history.append(correct/total*100)
```

[14]:
```
def test(epoch):
  ResNet18.eval()
  global best_acc
  test_loss = 0
  correct = 0
  total = 0
  with torch.no_grad():
    for idx, (images,labels) in enumerate(testDataLoader):
      images = images.cuda()
      labels = labels.cuda()
      outputs = ResNet18(images)
      loss = Loss(outputs,labels)

      test_loss += loss.item()
      _, predict = outputs.max(1)
      total += labels.size(0)
      correct += predict.eq(labels).sum().item()
  acc = 100.*correct/total
  if acc > best_acc:
    best_acc = acc
  print('Test loss: {:.2f}, Test accuracy:{:.2f} %'.format(test_loss/
 ↪len(testDataLoader),acc))
  print('Best accuracy: {:.2f}%'.format(best_acc))
  test_loss_history.append(test_loss/len(testDataLoader))
  test_accuracy_history.append(acc)
```

[15]:
```
for epoch in range(start_epoch, start_epoch+120):
    train(epoch)
    test(epoch)
    scheduler.step()
```

```
start_epoch += 120
```

```
Epoch: 0
Train loss: 1.83, Train accuracy:31.87 %
Test loss: 1.56, Test accuracy:40.55 %
Best accuracy: 40.55%

Epoch: 1
Train loss: 1.45, Train accuracy:46.56 %
Test loss: 1.34, Test accuracy:51.17 %
Best accuracy: 51.17%

Epoch: 2
Train loss: 1.18, Train accuracy:57.49 %
Test loss: 1.02, Test accuracy:64.56 %
Best accuracy: 64.56%

Epoch: 3
Train loss: 0.97, Train accuracy:65.54 %
Test loss: 0.81, Test accuracy:71.34 %
Best accuracy: 71.34%

Epoch: 4
Train loss: 0.85, Train accuracy:70.16 %
Test loss: 0.68, Test accuracy:76.11 %
Best accuracy: 76.11%

Epoch: 5
Train loss: 0.75, Train accuracy:73.51 %
Test loss: 0.69, Test accuracy:76.84 %
Best accuracy: 76.84%

Epoch: 6
Train loss: 0.69, Train accuracy:75.69 %
Test loss: 0.67, Test accuracy:77.03 %
Best accuracy: 77.03%

Epoch: 7
Train loss: 0.64, Train accuracy:77.48 %
Test loss: 0.61, Test accuracy:79.98 %
Best accuracy: 79.98%

Epoch: 8
Train loss: 0.60, Train accuracy:79.01 %
Test loss: 0.59, Test accuracy:80.17 %
Best accuracy: 80.17%
```

```
Epoch: 9
Train loss: 0.56, Train accuracy:80.35 %
Test loss: 0.48, Test accuracy:83.74 %
Best accuracy: 83.74%

Epoch: 10
Train loss: 0.54, Train accuracy:80.99 %
Test loss: 0.53, Test accuracy:82.45 %
Best accuracy: 83.74%

Epoch: 11
Train loss: 0.51, Train accuracy:82.21 %
Test loss: 0.45, Test accuracy:85.08 %
Best accuracy: 85.08%

Epoch: 12
Train loss: 0.48, Train accuracy:83.11 %
Test loss: 0.53, Test accuracy:82.20 %
Best accuracy: 85.08%

Epoch: 13
Train loss: 0.46, Train accuracy:83.75 %
Test loss: 0.44, Test accuracy:85.26 %
Best accuracy: 85.26%

Epoch: 14
Train loss: 0.44, Train accuracy:84.68 %
Test loss: 0.43, Test accuracy:85.80 %
Best accuracy: 85.80%

Epoch: 15
Train loss: 0.42, Train accuracy:85.62 %
Test loss: 0.41, Test accuracy:86.84 %
Best accuracy: 86.84%

Epoch: 16
Train loss: 0.40, Train accuracy:85.96 %
Test loss: 0.44, Test accuracy:85.75 %
Best accuracy: 86.84%

Epoch: 17
Train loss: 0.38, Train accuracy:86.58 %
Test loss: 0.38, Test accuracy:87.64 %
Best accuracy: 87.64%

Epoch: 18
Train loss: 0.38, Train accuracy:86.75 %
Test loss: 0.38, Test accuracy:87.50 %
```

```
Best accuracy: 87.64%

Epoch: 19
Train loss: 0.36, Train accuracy:87.45 %
Test loss: 0.37, Test accuracy:88.21 %
Best accuracy: 88.21%

Epoch: 20
Train loss: 0.34, Train accuracy:87.98 %
Test loss: 0.40, Test accuracy:87.87 %
Best accuracy: 88.21%

Epoch: 21
Train loss: 0.33, Train accuracy:88.45 %
Test loss: 0.38, Test accuracy:88.08 %
Best accuracy: 88.21%

Epoch: 22
Train loss: 0.32, Train accuracy:88.61 %
Test loss: 0.39, Test accuracy:88.08 %
Best accuracy: 88.21%

Epoch: 23
Train loss: 0.31, Train accuracy:89.19 %
Test loss: 0.38, Test accuracy:88.17 %
Best accuracy: 88.21%

Epoch: 24
Train loss: 0.31, Train accuracy:89.21 %
Test loss: 0.37, Test accuracy:88.52 %
Best accuracy: 88.52%

Epoch: 25
Train loss: 0.29, Train accuracy:89.70 %
Test loss: 0.36, Test accuracy:88.95 %
Best accuracy: 88.95%

Epoch: 26
Train loss: 0.29, Train accuracy:89.94 %
Test loss: 0.37, Test accuracy:88.46 %
Best accuracy: 88.95%

Epoch: 27
Train loss: 0.27, Train accuracy:90.41 %
Test loss: 0.37, Test accuracy:88.85 %
Best accuracy: 88.95%

Epoch: 28
```

```
Train loss: 0.27, Train accuracy:90.77 %
Test loss: 0.33, Test accuracy:89.49 %
Best accuracy: 89.49%

Epoch: 29
Train loss: 0.27, Train accuracy:90.68 %
Test loss: 0.37, Test accuracy:89.16 %
Best accuracy: 89.49%

Epoch: 30
Train loss: 0.26, Train accuracy:91.03 %
Test loss: 0.35, Test accuracy:89.22 %
Best accuracy: 89.49%

Epoch: 31
Train loss: 0.25, Train accuracy:91.24 %
Test loss: 0.37, Test accuracy:88.90 %
Best accuracy: 89.49%

Epoch: 32
Train loss: 0.24, Train accuracy:91.47 %
Test loss: 0.37, Test accuracy:89.14 %
Best accuracy: 89.49%

Epoch: 33
Train loss: 0.24, Train accuracy:91.61 %
Test loss: 0.36, Test accuracy:89.07 %
Best accuracy: 89.49%

Epoch: 34
Train loss: 0.23, Train accuracy:91.84 %
Test loss: 0.36, Test accuracy:89.51 %
Best accuracy: 89.51%

Epoch: 35
Train loss: 0.23, Train accuracy:92.01 %
Test loss: 0.38, Test accuracy:88.92 %
Best accuracy: 89.51%

Epoch: 36
Train loss: 0.22, Train accuracy:92.27 %
Test loss: 0.36, Test accuracy:89.63 %
Best accuracy: 89.63%

Epoch: 37
Train loss: 0.22, Train accuracy:92.34 %
Test loss: 0.34, Test accuracy:89.59 %
Best accuracy: 89.63%
```

```
Epoch: 38
Train loss: 0.21, Train accuracy:92.52 %
Test loss: 0.34, Test accuracy:89.66 %
Best accuracy: 89.66%

Epoch: 39
Train loss: 0.21, Train accuracy:92.66 %
Test loss: 0.60, Test accuracy:87.04 %
Best accuracy: 89.66%

Epoch: 40
Train loss: 0.20, Train accuracy:93.07 %
Test loss: 0.37, Test accuracy:89.53 %
Best accuracy: 89.66%

Epoch: 41
Train loss: 0.20, Train accuracy:93.18 %
Test loss: 0.40, Test accuracy:88.55 %
Best accuracy: 89.66%

Epoch: 42
Train loss: 0.19, Train accuracy:93.12 %
Test loss: 0.38, Test accuracy:89.14 %
Best accuracy: 89.66%

Epoch: 43
Train loss: 0.19, Train accuracy:93.31 %
Test loss: 0.39, Test accuracy:89.55 %
Best accuracy: 89.66%

Epoch: 44
Train loss: 0.19, Train accuracy:93.34 %
Test loss: 0.34, Test accuracy:90.24 %
Best accuracy: 90.24%

Epoch: 45
Train loss: 0.18, Train accuracy:93.62 %
Test loss: 0.37, Test accuracy:89.70 %
Best accuracy: 90.24%

Epoch: 46
Train loss: 0.18, Train accuracy:93.82 %
Test loss: 0.38, Test accuracy:90.06 %
Best accuracy: 90.24%

Epoch: 47
Train loss: 0.18, Train accuracy:93.70 %
```

```
Test loss: 0.36, Test accuracy:90.50 %
Best accuracy: 90.50%

Epoch: 48
Train loss: 0.17, Train accuracy:93.84 %
Test loss: 0.34, Test accuracy:90.06 %
Best accuracy: 90.50%

Epoch: 49
Train loss: 0.17, Train accuracy:94.02 %
Test loss: 0.37, Test accuracy:90.08 %
Best accuracy: 90.50%

Epoch: 50
Train loss: 0.17, Train accuracy:94.11 %
Test loss: 0.34, Test accuracy:90.55 %
Best accuracy: 90.55%

Epoch: 51
Train loss: 0.16, Train accuracy:94.35 %
Test loss: 0.37, Test accuracy:90.14 %
Best accuracy: 90.55%

Epoch: 52
Train loss: 0.16, Train accuracy:94.54 %
Test loss: 0.35, Test accuracy:90.46 %
Best accuracy: 90.55%

Epoch: 53
Train loss: 0.16, Train accuracy:94.43 %
Test loss: 0.38, Test accuracy:89.98 %
Best accuracy: 90.55%

Epoch: 54
Train loss: 0.16, Train accuracy:94.50 %
Test loss: 0.35, Test accuracy:90.19 %
Best accuracy: 90.55%

Epoch: 55
Train loss: 0.15, Train accuracy:94.57 %
Test loss: 0.36, Test accuracy:90.08 %
Best accuracy: 90.55%

Epoch: 56
Train loss: 0.15, Train accuracy:94.75 %
Test loss: 0.35, Test accuracy:90.28 %
Best accuracy: 90.55%
```

```
Epoch: 57
Train loss: 0.14, Train accuracy:94.90 %
Test loss: 0.41, Test accuracy:89.32 %
Best accuracy: 90.55%

Epoch: 58
Train loss: 0.14, Train accuracy:95.01 %
Test loss: 0.36, Test accuracy:90.52 %
Best accuracy: 90.55%

Epoch: 59
Train loss: 0.14, Train accuracy:95.07 %
Test loss: 0.34, Test accuracy:90.96 %
Best accuracy: 90.96%

Epoch: 60
Train loss: 0.14, Train accuracy:95.15 %
Test loss: 0.35, Test accuracy:90.55 %
Best accuracy: 90.96%

Epoch: 61
Train loss: 0.14, Train accuracy:95.21 %
Test loss: 0.38, Test accuracy:89.92 %
Best accuracy: 90.96%

Epoch: 62
Train loss: 0.13, Train accuracy:95.29 %
Test loss: 0.36, Test accuracy:90.58 %
Best accuracy: 90.96%

Epoch: 63
Train loss: 0.14, Train accuracy:95.20 %
Test loss: 0.36, Test accuracy:90.68 %
Best accuracy: 90.96%

Epoch: 64
Train loss: 0.13, Train accuracy:95.61 %
Test loss: 0.37, Test accuracy:90.52 %
Best accuracy: 90.96%

Epoch: 65
Train loss: 0.13, Train accuracy:95.63 %
Test loss: 0.41, Test accuracy:89.76 %
Best accuracy: 90.96%

Epoch: 66
Train loss: 0.12, Train accuracy:95.67 %
Test loss: 0.36, Test accuracy:90.85 %
```

```
Best accuracy: 90.96%

Epoch: 67
Train loss: 0.13, Train accuracy:95.54 %
Test loss: 0.38, Test accuracy:90.73 %
Best accuracy: 90.96%

Epoch: 68
Train loss: 0.12, Train accuracy:95.72 %
Test loss: 0.36, Test accuracy:90.56 %
Best accuracy: 90.96%

Epoch: 69
Train loss: 0.12, Train accuracy:95.69 %
Test loss: 0.35, Test accuracy:91.06 %
Best accuracy: 91.06%

Epoch: 70
Train loss: 0.12, Train accuracy:95.86 %
Test loss: 0.35, Test accuracy:90.92 %
Best accuracy: 91.06%

Epoch: 71
Train loss: 0.11, Train accuracy:96.00 %
Test loss: 0.38, Test accuracy:90.39 %
Best accuracy: 91.06%

Epoch: 72
Train loss: 0.12, Train accuracy:95.89 %
Test loss: 0.33, Test accuracy:91.49 %
Best accuracy: 91.49%

Epoch: 73
Train loss: 0.11, Train accuracy:96.02 %
Test loss: 0.34, Test accuracy:91.48 %
Best accuracy: 91.49%

Epoch: 74
Train loss: 0.11, Train accuracy:96.15 %
Test loss: 0.35, Test accuracy:91.09 %
Best accuracy: 91.49%

Epoch: 75
Train loss: 0.11, Train accuracy:96.22 %
Test loss: 0.35, Test accuracy:90.96 %
Best accuracy: 91.49%

Epoch: 76
```

```
Train loss: 0.11, Train accuracy:96.32 %
Test loss: 0.34, Test accuracy:91.38 %
Best accuracy: 91.49%

Epoch: 77
Train loss: 0.11, Train accuracy:96.42 %
Test loss: 0.36, Test accuracy:90.96 %
Best accuracy: 91.49%

Epoch: 78
Train loss: 0.10, Train accuracy:96.44 %
Test loss: 0.38, Test accuracy:91.10 %
Best accuracy: 91.49%

Epoch: 79
Train loss: 0.10, Train accuracy:96.54 %
Test loss: 0.37, Test accuracy:91.38 %
Best accuracy: 91.49%

Epoch: 80
Train loss: 0.10, Train accuracy:96.59 %
Test loss: 0.36, Test accuracy:91.32 %
Best accuracy: 91.49%

Epoch: 81
Train loss: 0.10, Train accuracy:96.56 %
Test loss: 0.37, Test accuracy:91.44 %
Best accuracy: 91.49%

Epoch: 82
Train loss: 0.09, Train accuracy:96.72 %
Test loss: 0.34, Test accuracy:91.73 %
Best accuracy: 91.73%

Epoch: 83
Train loss: 0.09, Train accuracy:96.83 %
Test loss: 0.36, Test accuracy:91.45 %
Best accuracy: 91.73%

Epoch: 84
Train loss: 0.09, Train accuracy:96.87 %
Test loss: 0.37, Test accuracy:91.11 %
Best accuracy: 91.73%

Epoch: 85
Train loss: 0.09, Train accuracy:96.73 %
Test loss: 0.38, Test accuracy:91.10 %
Best accuracy: 91.73%
```

```
Epoch: 86
Train loss: 0.09, Train accuracy:96.87 %
Test loss: 0.36, Test accuracy:91.48 %
Best accuracy: 91.73%

Epoch: 87
Train loss: 0.09, Train accuracy:96.90 %
Test loss: 0.35, Test accuracy:91.57 %
Best accuracy: 91.73%

Epoch: 88
Train loss: 0.09, Train accuracy:97.04 %
Test loss: 0.39, Test accuracy:91.19 %
Best accuracy: 91.73%

Epoch: 89
Train loss: 0.09, Train accuracy:96.96 %
Test loss: 0.36, Test accuracy:91.39 %
Best accuracy: 91.73%

Epoch: 90
Train loss: 0.09, Train accuracy:97.02 %
Test loss: 0.36, Test accuracy:91.43 %
Best accuracy: 91.73%

Epoch: 91
Train loss: 0.08, Train accuracy:97.18 %
Test loss: 0.36, Test accuracy:91.37 %
Best accuracy: 91.73%

Epoch: 92
Train loss: 0.08, Train accuracy:97.24 %
Test loss: 0.35, Test accuracy:91.72 %
Best accuracy: 91.73%

Epoch: 93
Train loss: 0.08, Train accuracy:97.19 %
Test loss: 0.37, Test accuracy:91.24 %
Best accuracy: 91.73%

Epoch: 94
Train loss: 0.08, Train accuracy:97.21 %
Test loss: 0.35, Test accuracy:91.86 %
Best accuracy: 91.86%

Epoch: 95
Train loss: 0.08, Train accuracy:97.34 %
```

```
Test loss: 0.37, Test accuracy:91.41 %
Best accuracy: 91.86%

Epoch: 96
Train loss: 0.08, Train accuracy:97.28 %
Test loss: 0.35, Test accuracy:92.06 %
Best accuracy: 92.06%

Epoch: 97
Train loss: 0.08, Train accuracy:97.31 %
Test loss: 0.41, Test accuracy:90.69 %
Best accuracy: 92.06%

Epoch: 98
Train loss: 0.07, Train accuracy:97.46 %
Test loss: 0.38, Test accuracy:91.26 %
Best accuracy: 92.06%

Epoch: 99
Train loss: 0.07, Train accuracy:97.40 %
Test loss: 0.35, Test accuracy:91.89 %
Best accuracy: 92.06%

Epoch: 100
Train loss: 0.07, Train accuracy:97.58 %
Test loss: 0.36, Test accuracy:91.72 %
Best accuracy: 92.06%

Epoch: 101
Train loss: 0.07, Train accuracy:97.53 %
Test loss: 0.36, Test accuracy:91.72 %
Best accuracy: 92.06%

Epoch: 102
Train loss: 0.07, Train accuracy:97.55 %
Test loss: 0.41, Test accuracy:91.02 %
Best accuracy: 92.06%

Epoch: 103
Train loss: 0.07, Train accuracy:97.63 %
Test loss: 0.36, Test accuracy:91.96 %
Best accuracy: 92.06%

Epoch: 104
Train loss: 0.07, Train accuracy:97.78 %
Test loss: 0.36, Test accuracy:91.88 %
Best accuracy: 92.06%
```

```
Epoch: 105
Train loss: 0.07, Train accuracy:97.66 %
Test loss: 0.36, Test accuracy:91.39 %
Best accuracy: 92.06%

Epoch: 106
Train loss: 0.06, Train accuracy:97.75 %
Test loss: 0.33, Test accuracy:92.36 %
Best accuracy: 92.36%

Epoch: 107
Train loss: 0.06, Train accuracy:97.79 %
Test loss: 0.36, Test accuracy:91.92 %
Best accuracy: 92.36%

Epoch: 108
Train loss: 0.06, Train accuracy:97.92 %
Test loss: 0.37, Test accuracy:91.56 %
Best accuracy: 92.36%

Epoch: 109
Train loss: 0.06, Train accuracy:97.84 %
Test loss: 0.35, Test accuracy:92.09 %
Best accuracy: 92.36%

Epoch: 110
Train loss: 0.06, Train accuracy:97.92 %
Test loss: 0.34, Test accuracy:92.20 %
Best accuracy: 92.36%

Epoch: 111
Train loss: 0.06, Train accuracy:97.89 %
Test loss: 0.36, Test accuracy:92.08 %
Best accuracy: 92.36%

Epoch: 112
Train loss: 0.06, Train accuracy:97.90 %
Test loss: 0.36, Test accuracy:92.12 %
Best accuracy: 92.36%

Epoch: 113
Train loss: 0.06, Train accuracy:97.99 %
Test loss: 0.36, Test accuracy:91.94 %
Best accuracy: 92.36%

Epoch: 114
Train loss: 0.05, Train accuracy:98.10 %
Test loss: 0.37, Test accuracy:92.24 %
```

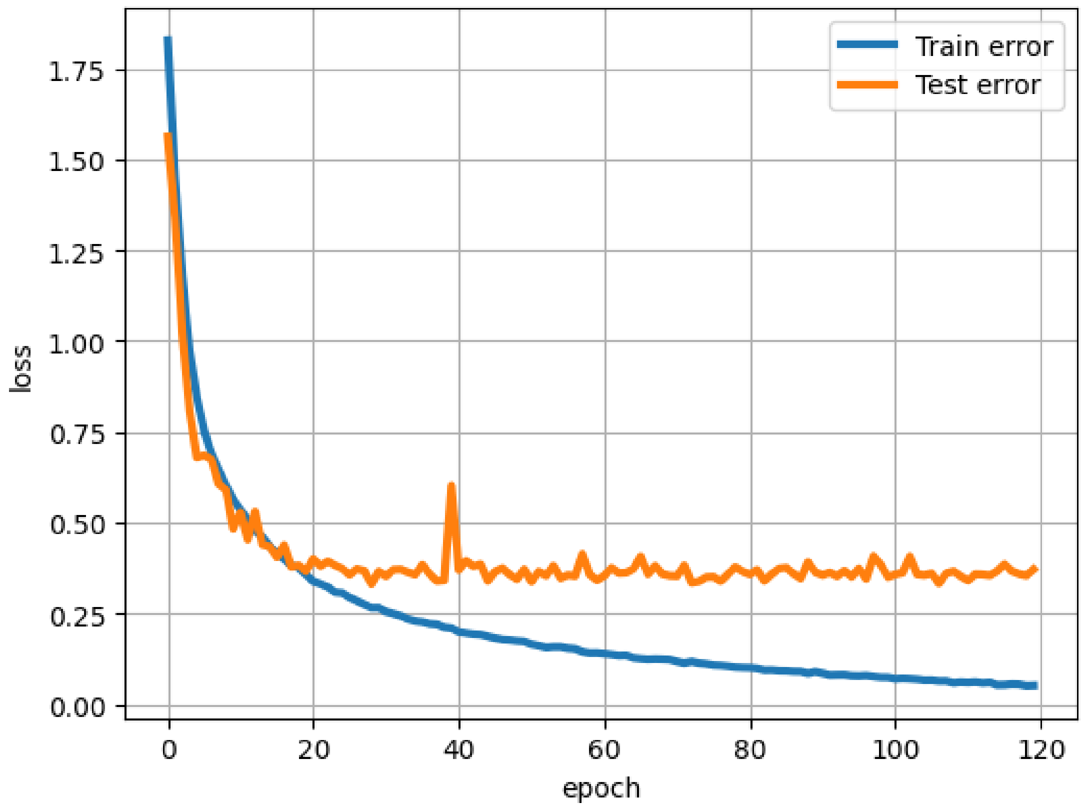
```
Best accuracy: 92.36%

Epoch: 115
Train loss: 0.05, Train accuracy:98.18 %
Test loss: 0.39, Test accuracy:91.69 %
Best accuracy: 92.36%

Epoch: 116
Train loss: 0.06, Train accuracy:98.13 %
Test loss: 0.37, Test accuracy:92.21 %
Best accuracy: 92.36%

Epoch: 117
Train loss: 0.06, Train accuracy:98.12 %
Test loss: 0.36, Test accuracy:92.08 %
Best accuracy: 92.36%

Epoch: 118
Train loss: 0.05, Train accuracy:98.20 %
Test loss: 0.35, Test accuracy:92.15 %
Best accuracy: 92.36%

Epoch: 119
Train loss: 0.05, Train accuracy:98.17 %
Test loss: 0.37, Test accuracy:92.16 %
Best accuracy: 92.36%
```

[18]:

```
import matplotlib.pyplot as plt
plt.
 ↪plot(range(len(train_loss_history)),train_loss_history,'-',linewidth=3,label='Train␣
 ↪error')
plt.
 ↪plot(range(len(test_loss_history)),test_loss_history,'-',linewidth=3,label='Test␣
 ↪error')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.grid(True)
plt.legend()
```

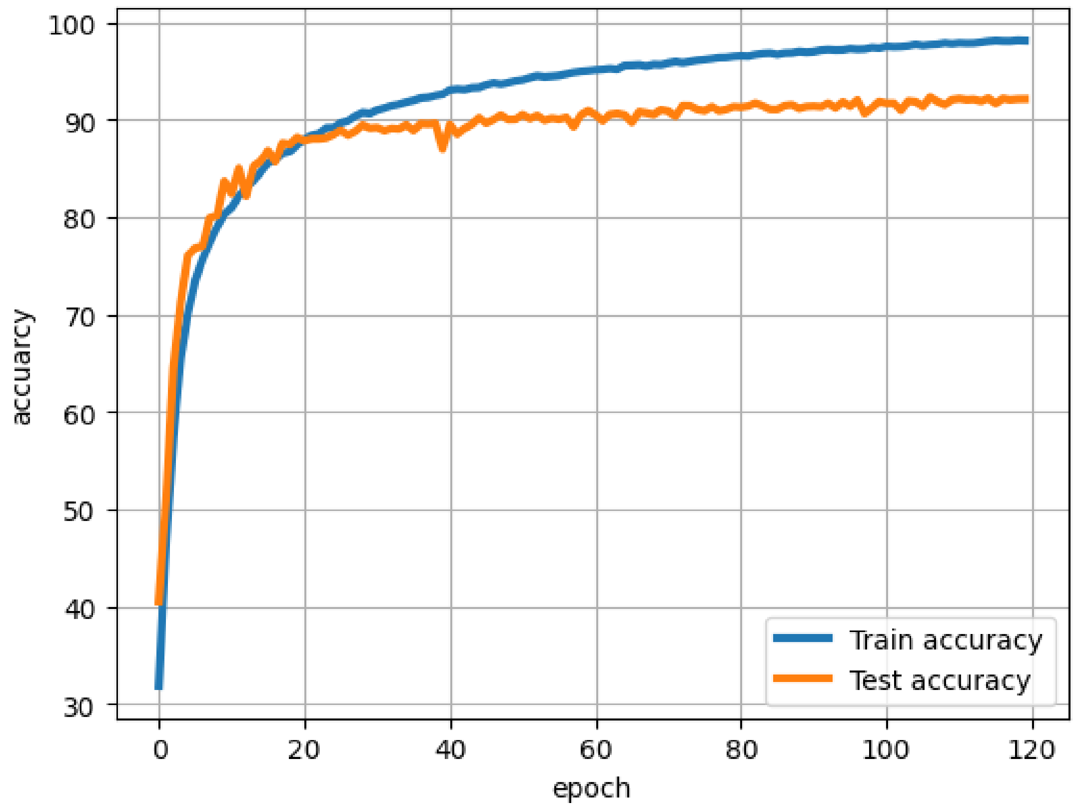
[18]:

```
<matplotlib.legend.Legend at 0x78be56c84580>
```



[19]:
```
plt.
 ↪plot(range(len(train_accuracy_history)),train_accuracy_history,'-',linewidth=3,label='Train␣
 ↪accuracy')
plt.
 ↪plot(range(len(test_accuracy_history)),test_accuracy_history,'-',linewidth=3,label='Test␣
 ↪accuracy')
plt.xlabel('epoch')
plt.ylabel('accuarcy')
plt.grid(True)
plt.legend()
```

[19]: <matplotlib.legend.Legend at 0x78be56d3b430>



[20]:

```
from google.colab import files
torch.save(ResNet18.state_dict(), 'model3.pt')

# download checkpoint file
files.download('model3.pt')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

[23]:

```
from torchsummary import summary
print(summary(model, (3, 32, 32)))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]             864
       BatchNorm2d-2           [-1, 32, 32, 32]              64
            Conv2d-3           [-1, 32, 32, 32]           9,216
       BatchNorm2d-4           [-1, 32, 32, 32]              64
            Conv2d-5           [-1, 32, 32, 32]           9,216
       BatchNorm2d-6           [-1, 32, 32, 32]              64
```

BasicBlock-7	[-1, 32, 32, 32]	0
Conv2d-8	[-1, 32, 32, 32]	9,216
BatchNorm2d-9	[-1, 32, 32, 32]	64
Conv2d-10	[-1, 32, 32, 32]	9,216
BatchNorm2d-11	[-1, 32, 32, 32]	64
BasicBlock-12	[-1, 32, 32, 32]	0
Conv2d-13	[-1, 32, 32, 32]	9,216
BatchNorm2d-14	[-1, 32, 32, 32]	64
Conv2d-15	[-1, 32, 32, 32]	9,216
BatchNorm2d-16	[-1, 32, 32, 32]	64
BasicBlock-17	[-1, 32, 32, 32]	0
Conv2d-18	[-1, 64, 16, 16]	18,432
BatchNorm2d-19	[-1, 64, 16, 16]	128
Conv2d-20	[-1, 64, 16, 16]	36,864
BatchNorm2d-21	[-1, 64, 16, 16]	128
Conv2d-22	[-1, 64, 16, 16]	2,048
BatchNorm2d-23	[-1, 64, 16, 16]	128
BasicBlock-24	[-1, 64, 16, 16]	0
Conv2d-25	[-1, 64, 16, 16]	36,864
BatchNorm2d-26	[-1, 64, 16, 16]	128
Conv2d-27	[-1, 64, 16, 16]	36,864
BatchNorm2d-28	[-1, 64, 16, 16]	128
BasicBlock-29	[-1, 64, 16, 16]	0
Conv2d-30	[-1, 128, 8, 8]	73,728
BatchNorm2d-31	[-1, 128, 8, 8]	256
Conv2d-32	[-1, 128, 8, 8]	147,456
BatchNorm2d-33	[-1, 128, 8, 8]	256
Conv2d-34	[-1, 128, 8, 8]	8,192
BatchNorm2d-35	[-1, 128, 8, 8]	256
BasicBlock-36	[-1, 128, 8, 8]	0
Conv2d-37	[-1, 128, 8, 8]	147,456
BatchNorm2d-38	[-1, 128, 8, 8]	256
Conv2d-39	[-1, 128, 8, 8]	147,456
BatchNorm2d-40	[-1, 128, 8, 8]	256
BasicBlock-41	[-1, 128, 8, 8]	0
Conv2d-42	[-1, 128, 8, 8]	147,456
BatchNorm2d-43	[-1, 128, 8, 8]	256
Conv2d-44	[-1, 128, 8, 8]	147,456
BatchNorm2d-45	[-1, 128, 8, 8]	256
BasicBlock-46	[-1, 128, 8, 8]	0
Conv2d-47	[-1, 128, 8, 8]	147,456
BatchNorm2d-48	[-1, 128, 8, 8]	256
Conv2d-49	[-1, 128, 8, 8]	147,456
BatchNorm2d-50	[-1, 128, 8, 8]	256
BasicBlock-51	[-1, 128, 8, 8]	0
Conv2d-52	[-1, 256, 4, 4]	294,912
BatchNorm2d-53	[-1, 256, 4, 4]	512
Conv2d-54	[-1, 256, 4, 4]	589,824

```
      BatchNorm2d-55            [-1, 256, 4, 4]             512
           Conv2d-56            [-1, 256, 4, 4]          32,768
      BatchNorm2d-57            [-1, 256, 4, 4]             512
       BasicBlock-58            [-1, 256, 4, 4]               0
           Conv2d-59            [-1, 256, 4, 4]         589,824
      BatchNorm2d-60            [-1, 256, 4, 4]             512
           Conv2d-61            [-1, 256, 4, 4]         589,824
      BatchNorm2d-62            [-1, 256, 4, 4]             512
       BasicBlock-63            [-1, 256, 4, 4]               0
           Conv2d-64            [-1, 256, 4, 4]         589,824
      BatchNorm2d-65            [-1, 256, 4, 4]             512
           Conv2d-66            [-1, 256, 4, 4]         589,824
      BatchNorm2d-67            [-1, 256, 4, 4]             512
       BasicBlock-68            [-1, 256, 4, 4]               0
           Linear-69                   [-1, 10]           2,570
================================================================
Total params: 4,587,690
Trainable params: 4,587,690
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 7.66
Params size (MB): 17.50
Estimated Total Size (MB): 25.17
----------------------------------------------------------------
None
```

[21]:
```
# read model file
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
model = ResNet18
model_path = './model3.pt'
model.load_state_dict(torch.load(model_path, map_location=device), strict=False)
model.eval()
```

[21]:
```
ResNet(
  (conv1): Conv2d(3, 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)
  (layer1): Sequential(
    (0): BasicBlock(
      (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)
      (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)
      (shortcut): Sequential()
    )
    (1): BasicBlock(
      (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)
      (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)
      (shortcut): Sequential()
    )
    (2): BasicBlock(
      (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)
      (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)
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (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)
      (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)
      (shortcut): Sequential(
        (0): Conv2d(32, 64, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (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)
      (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)
      (shortcut): Sequential()
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
    (2): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
    (3): BasicBlock(
```

```
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (shortcut): Sequential()
    )
    (2): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
        (shortcut): Sequential()
      )
    )
    (linear): Linear(in_features=256, out_features=10, bias=True)
  )
```

[ ]:
```python
def project1_model():
    return ResNet(BasicBlock, [3, 2, 4, 3])
```

[24]:
```python
def unpickle(file):
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict
```

[25]:
```python
import pickle
test_data = unpickle('cifar_test_nolabels.pkl')
```

[26]:
```python
test_images = test_data[b'data'].reshape((-1, 3, 32, 32)).astype(np.float32) /␣
 ↪255.0
test_images = torch.tensor(test_images)

mean = torch.tensor([0.4914, 0.4822, 0.4465]).view(1, 3, 1, 1)
std = torch.tensor([0.2023, 0.1994, 0.2010]).view(1, 3, 1, 1)

preprocessed_images = (test_images - mean) / std
```

[27]:
```python
from torch.utils.data import DataLoader
custom_testset = torch.utils.data.TensorDataset(preprocessed_images)
custom_testloader = DataLoader(custom_testset, batch_size=100, shuffle=False)
```

[28]:
```python
predictions = []
with torch.no_grad():
    for data in custom_testloader:
        images = data[0].to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        predictions.extend(predicted.cpu().tolist())
```

[29]:
```python
import csv
image_ids = np.arange(len(predictions))

output_csv_path = 'predictions.csv'

with open(output_csv_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['ID', 'Labels'])
    for img_id, prediction in zip(image_ids, predictions):
```

```python
        writer.writerow([img_id, prediction])

print(f'Predictions have been saved to {output_csv_path}')
```

```
Predictions have been saved to predictions.csv
```