final_1

March 26, 2023

```python
[1]: #!pip install torch torchvision
     %matplotlib inline
     import matplotlib.pyplot as plt
     import numpy as np
     import torch
     import torchvision
     import torchvision.transforms as transforms
     import torch.nn as nn
     import torch.nn.functional as F
     import torch.optim as optim
```

Prepare for Dataset

```python
[2]: transform = transforms.Compose(
         [transforms.ToTensor(),
          transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

     trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
                                              download=True, transform=transform)
     trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                               shuffle=True, num_workers=2)

     testset = torchvision.datasets.CIFAR100(root='./data', train=False,
                                             download=True, transform=transform)
     testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                              shuffle=False, num_workers=2)

     classes = ('apple', 'aquarium_fish', 'baby', 'bear', 'beaver',
                'bed', 'bee', 'beetle', 'bicycle', 'bottle', 'bowl',
                'boy', 'bridge', 'bus', 'butterfly', 'camel', 'can',
                'castle', 'caterpillar', 'cattle', 'chair',
                'chimpanzee', 'clock', 'cloud', 'cockroach', 'couch',
                'crab', 'crocodile', 'cup', 'dinosaur', 'dolphin',
                'elephant', 'flatfish', 'forest', 'fox', 'girl',
                'hamster', 'house', 'kangaroo', 'keyboard', 'lamp',
                'lawn_mower', 'leopard', 'lion', 'lizard', 'lobster',
                'man', 'maple_tree', 'motorcycle', 'mountain', 'mouse',
                'mushroom', 'oak_tree', 'orange', 'orchid', 'otter',
```

```
        'palm_tree', 'pear', 'pickup_truck', 'pine_tree', 'plain',
        'plate', 'poppy', 'porcupine', 'possum', 'rabbit', 'raccoon',
        'ray', 'road', 'rocket', 'rose', 'sea', 'seal', 'shark',
        'shrew', 'skunk', 'skyscraper', 'snail', 'snake', 'spider',
        'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table',
        'tank', 'telephone', 'television', 'tiger', 'tractor', 'train',
        'trout', 'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree',
        'wolf', 'woman', 'worm')

#classes = ('plane', 'car', 'bird', 'cat',
#           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

[3]:
```python
# The function to show an image.
def imshow(img):
    img = img / 2 + 0.5     # Unnormalize.
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# Get some random training images.
dataiter = iter(trainloader)
images, labels = next(dataiter)
# Show images.
imshow(torchvision.utils.make_grid(images))
# Print labels.
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```
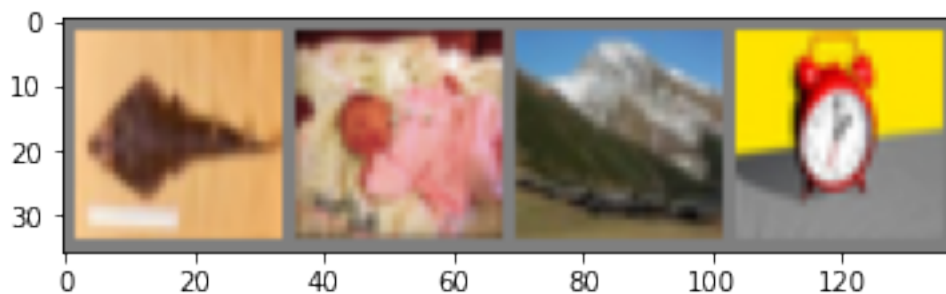


```
 ray   baby mountain clock
```

Choose a Device

[4]:
```python
# If there are GPUs, choose the first one for computing. Otherwise use CPU.
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
print(device)
# If 'cuda:0' is printed, it means GPU is available.
```

cuda:0

Network Definition   This modified version of the Net class adds a fourth convolutional layer
with 512 output channels, followed by a dropout layer with a dropout rate of 0.5. It also increases
the size of the fully connected layer to 1024.

```
[5]: class Net(nn.Module):
         def __init__(self):
             super(Net, self).__init__()
             self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3,␣
       ↪padding=1)
             self.bn1 = nn.BatchNorm2d(64)
             self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
             self.relu1 = nn.ReLU()
             self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
             self.bn2 = nn.BatchNorm2d(128)
             self.pool2 = nn.MaxPool2d(2, 2)
             self.relu2 = nn.ReLU()
             self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
             self.bn3 = nn.BatchNorm2d(256)
             self.pool3 = nn.MaxPool2d(2, 2)
             self.relu3 = nn.ReLU()
             self.conv4 = nn.Conv2d(256, 1024, 3, padding=1)  # modified
             self.bn4 = nn.BatchNorm2d(1024)
             self.pool4 = nn.MaxPool2d(2, 2)
             self.relu4 = nn.ReLU()
             self.dropout = nn.Dropout(0.5)
             self.fc1 = nn.Linear(1024 * 2 * 2, 1024)  # modified
             self.bn5 = nn.BatchNorm1d(1024)
             self.relu5 = nn.ReLU()
             self.fc2 = nn.Linear(1024, 100)
             self.bn6 = nn.BatchNorm1d(100)  # modified

         def forward(self, x):
             x = self.pool1(self.relu1(self.conv1(x)))
             x = self.pool2(self.relu2(self.conv2(x)))
             x = self.pool3(self.relu3(self.conv3(x)))
             x = self.pool4(self.relu4(self.bn4(self.conv4(x))))  # modified
             x = x.view(-1, 1024 * 2 * 2)  # modified
             x = self.relu5(self.bn5(self.fc1(x)))
             x = self.bn6(self.fc2(x))
             return x
```

```
net = Net()        # Create the network instance.
net.to(device)     # Move the network parameters to the specified device.
```

[5]: Net(
    (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
    (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
    ceil_mode=False)
    (relu1): ReLU()
    (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
    (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
    ceil_mode=False)
    (relu2): ReLU()
    (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
    (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
    ceil_mode=False)
    (relu3): ReLU()
    (conv4): Conv2d(256, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn4): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
    (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
    ceil_mode=False)
    (relu4): ReLU()
    (dropout): Dropout(p=0.5, inplace=False)
    (fc1): Linear(in_features=4096, out_features=1024, bias=True)
    (bn5): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
    (relu5): ReLU()
    (fc2): Linear(in_features=1024, out_features=100, bias=True)
    (bn6): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True,
    track_running_stats=True)
)
```

**Optimizer and Loss Function**

```
[6]: # We use cross-entropy as loss function.
loss_func = nn.CrossEntropyLoss()
# We use stochastic gradient descent (SGD) as optimizer.
opt = optim.SGD(net.parameters(), lr=0.0005, momentum=0.9)
#scheduler = optim.lr_scheduler.StepLR(opt, step_size=2, gamma=0.1)
```

Training Procedure   added learning rate scheduler

A learning rate scheduler adjusts the learning rate during training, based on a pre-defined schedule. The learning rate determines how quickly the model parameters are updated during training. If the learning rate is too high, the optimization process may overshoot the optimal parameters and fail to converge. If the learning rate is too low, the optimization process may take a long time to converge.

A learning rate scheduler can help address this problem by gradually decreasing the learning rate during training, allowing the model to converge more slowly and potentially to a better solution. There are different types of learning rate schedules, such as step decay, exponential decay, and cosine annealing, among others. Each type has its own formula for computing the learning rate at each epoch or iteration.

Another way to improve accuracy is by using learning rate scheduling. You can decrease the learning rate over time, so that the model can converge to a better minimum in the optimization landscape.

we define a StepLR scheduler with a step size of 2 epochs and a gamma of 0.1. This means that the learning rate will be multiplied by 0.1 every 2 epochs. We then use the scheduler to update the learning rate after each optimization step with the line scheduler.step().

```python
import sys
from tqdm.notebook import tqdm

avg_losses = []    # Avg. losses.
epochs = 7        # Total epochs.
print_freq = 500  # Print frequency.

for epoch in range(epochs):  # Loop over the dataset multiple times.
    running_loss = 0.0        # Initialize running loss.
    for i, data in enumerate(tqdm(trainloader), 0):
        # Get the inputs.
        inputs, labels = data

        # Move the inputs to the specified device.
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients.
        opt.zero_grad()

        # Forward step.
        outputs = net(inputs)
        loss = loss_func(outputs, labels)

        # Backward step.
        loss.backward()

        # Optimization step (update the parameters).
        opt.step()
```

```python
        # Update the learning rate.
        #scheduler.step()

        # Print statistics.
        running_loss += loss.item()
        if i % print_freq == print_freq - 1: # Print every several mini-batches.
            avg_loss = running_loss / print_freq
            print('[epoch: {}, i: {:5d}] avg mini-batch loss: {:.3f}'.
format(epoch, i, avg_loss),flush=True)
            sys.stdout.flush()
            avg_losses.append(avg_loss)
            running_loss = 0.0

print('Finished Training.')
```

```
  0%|          | 0/12500 [00:00<?, ?it/s]
[epoch: 0, i:   499] avg mini-batch loss: 4.850
[epoch: 0, i:   999] avg mini-batch loss: 4.737
[epoch: 0, i:  1499] avg mini-batch loss: 4.544
[epoch: 0, i:  1999] avg mini-batch loss: 4.474
[epoch: 0, i:  2499] avg mini-batch loss: 4.396
[epoch: 0, i:  2999] avg mini-batch loss: 4.315
[epoch: 0, i:  3499] avg mini-batch loss: 4.306
[epoch: 0, i:  3999] avg mini-batch loss: 4.308
[epoch: 0, i:  4499] avg mini-batch loss: 4.268
[epoch: 0, i:  4999] avg mini-batch loss: 4.225
[epoch: 0, i:  5499] avg mini-batch loss: 4.193
[epoch: 0, i:  5999] avg mini-batch loss: 4.196
[epoch: 0, i:  6499] avg mini-batch loss: 4.156
[epoch: 0, i:  6999] avg mini-batch loss: 4.167
[epoch: 0, i:  7499] avg mini-batch loss: 4.151
[epoch: 0, i:  7999] avg mini-batch loss: 4.157
[epoch: 0, i:  8499] avg mini-batch loss: 4.117
[epoch: 0, i:  8999] avg mini-batch loss: 4.080
[epoch: 0, i:  9499] avg mini-batch loss: 4.088
[epoch: 0, i:  9999] avg mini-batch loss: 4.075
[epoch: 0, i: 10499] avg mini-batch loss: 4.063
[epoch: 0, i: 10999] avg mini-batch loss: 4.055
[epoch: 0, i: 11499] avg mini-batch loss: 4.020
[epoch: 0, i: 11999] avg mini-batch loss: 4.045
[epoch: 0, i: 12499] avg mini-batch loss: 4.000

  0%|          | 0/12500 [00:00<?, ?it/s]
[epoch: 1, i:   499] avg mini-batch loss: 3.991
[epoch: 1, i:   999] avg mini-batch loss: 3.984
[epoch: 1, i:  1499] avg mini-batch loss: 3.976
[epoch: 1, i:  1999] avg mini-batch loss: 3.961
```

```
[epoch: 1, i:  2499] avg mini-batch loss: 3.968
[epoch: 1, i:  2999] avg mini-batch loss: 3.979
[epoch: 1, i:  3499] avg mini-batch loss: 3.962
[epoch: 1, i:  3999] avg mini-batch loss: 3.951
[epoch: 1, i:  4499] avg mini-batch loss: 3.929
[epoch: 1, i:  4999] avg mini-batch loss: 3.950
[epoch: 1, i:  5499] avg mini-batch loss: 3.940
[epoch: 1, i:  5999] avg mini-batch loss: 3.932
[epoch: 1, i:  6499] avg mini-batch loss: 3.948
[epoch: 1, i:  6999] avg mini-batch loss: 3.920
[epoch: 1, i:  7499] avg mini-batch loss: 3.928
[epoch: 1, i:  7999] avg mini-batch loss: 3.896
[epoch: 1, i:  8499] avg mini-batch loss: 3.875
[epoch: 1, i:  8999] avg mini-batch loss: 3.835
[epoch: 1, i:  9499] avg mini-batch loss: 3.906
[epoch: 1, i:  9999] avg mini-batch loss: 3.890
[epoch: 1, i: 10499] avg mini-batch loss: 3.871
[epoch: 1, i: 10999] avg mini-batch loss: 3.853
[epoch: 1, i: 11499] avg mini-batch loss: 3.828
[epoch: 1, i: 11999] avg mini-batch loss: 3.859
[epoch: 1, i: 12499] avg mini-batch loss: 3.857

  0%|          | 0/12500 [00:00<?, ?it/s]

[epoch: 2, i:   499] avg mini-batch loss: 3.839
[epoch: 2, i:   999] avg mini-batch loss: 3.782
[epoch: 2, i:  1499] avg mini-batch loss: 3.803
[epoch: 2, i:  1999] avg mini-batch loss: 3.778
[epoch: 2, i:  2499] avg mini-batch loss: 3.790
[epoch: 2, i:  2999] avg mini-batch loss: 3.752
[epoch: 2, i:  3499] avg mini-batch loss: 3.771
[epoch: 2, i:  3999] avg mini-batch loss: 3.779
[epoch: 2, i:  4499] avg mini-batch loss: 3.756
[epoch: 2, i:  4999] avg mini-batch loss: 3.757
[epoch: 2, i:  5499] avg mini-batch loss: 3.738
[epoch: 2, i:  5999] avg mini-batch loss: 3.714
[epoch: 2, i:  6499] avg mini-batch loss: 3.744
[epoch: 2, i:  6999] avg mini-batch loss: 3.736
[epoch: 2, i:  7499] avg mini-batch loss: 3.732
[epoch: 2, i:  7999] avg mini-batch loss: 3.720
[epoch: 2, i:  8499] avg mini-batch loss: 3.698
[epoch: 2, i:  8999] avg mini-batch loss: 3.696
[epoch: 2, i:  9499] avg mini-batch loss: 3.705
[epoch: 2, i:  9999] avg mini-batch loss: 3.708
[epoch: 2, i: 10499] avg mini-batch loss: 3.705
[epoch: 2, i: 10999] avg mini-batch loss: 3.659
[epoch: 2, i: 11499] avg mini-batch loss: 3.664
[epoch: 2, i: 11999] avg mini-batch loss: 3.633
[epoch: 2, i: 12499] avg mini-batch loss: 3.650
```

```
  0%|          | 0/12500 [00:00<?, ?it/s]
[epoch: 3, i:   499] avg mini-batch loss: 3.630
[epoch: 3, i:   999] avg mini-batch loss: 3.627
[epoch: 3, i:  1499] avg mini-batch loss: 3.560
[epoch: 3, i:  1999] avg mini-batch loss: 3.570
[epoch: 3, i:  2499] avg mini-batch loss: 3.675
[epoch: 3, i:  2999] avg mini-batch loss: 3.650
[epoch: 3, i:  3499] avg mini-batch loss: 3.626
[epoch: 3, i:  3999] avg mini-batch loss: 3.546
[epoch: 3, i:  4499] avg mini-batch loss: 3.576
[epoch: 3, i:  4999] avg mini-batch loss: 3.591
[epoch: 3, i:  5499] avg mini-batch loss: 3.570
[epoch: 3, i:  5999] avg mini-batch loss: 3.574
[epoch: 3, i:  6499] avg mini-batch loss: 3.577
[epoch: 3, i:  6999] avg mini-batch loss: 3.538
[epoch: 3, i:  7499] avg mini-batch loss: 3.521
[epoch: 3, i:  7999] avg mini-batch loss: 3.560
[epoch: 3, i:  8499] avg mini-batch loss: 3.590
[epoch: 3, i:  8999] avg mini-batch loss: 3.549
[epoch: 3, i:  9499] avg mini-batch loss: 3.545
[epoch: 3, i:  9999] avg mini-batch loss: 3.503
[epoch: 3, i: 10499] avg mini-batch loss: 3.529
[epoch: 3, i: 10999] avg mini-batch loss: 3.517
[epoch: 3, i: 11499] avg mini-batch loss: 3.589
[epoch: 3, i: 11999] avg mini-batch loss: 3.523
[epoch: 3, i: 12499] avg mini-batch loss: 3.522
  0%|          | 0/12500 [00:00<?, ?it/s]
[epoch: 4, i:   499] avg mini-batch loss: 3.481
[epoch: 4, i:   999] avg mini-batch loss: 3.507
[epoch: 4, i:  1499] avg mini-batch loss: 3.459
[epoch: 4, i:  1999] avg mini-batch loss: 3.447
[epoch: 4, i:  2499] avg mini-batch loss: 3.411
[epoch: 4, i:  2999] avg mini-batch loss: 3.486
[epoch: 4, i:  3499] avg mini-batch loss: 3.424
[epoch: 4, i:  3999] avg mini-batch loss: 3.431
[epoch: 4, i:  4499] avg mini-batch loss: 3.404
[epoch: 4, i:  4999] avg mini-batch loss: 3.444
[epoch: 4, i:  5499] avg mini-batch loss: 3.436
[epoch: 4, i:  5999] avg mini-batch loss: 3.432
[epoch: 4, i:  6499] avg mini-batch loss: 3.404
[epoch: 4, i:  6999] avg mini-batch loss: 3.445
[epoch: 4, i:  7499] avg mini-batch loss: 3.408
[epoch: 4, i:  7999] avg mini-batch loss: 3.432
[epoch: 4, i:  8499] avg mini-batch loss: 3.411
[epoch: 4, i:  8999] avg mini-batch loss: 3.416
[epoch: 4, i:  9499] avg mini-batch loss: 3.422
```

```
[epoch: 4, i:  9999] avg mini-batch loss: 3.383
[epoch: 4, i: 10499] avg mini-batch loss: 3.417
[epoch: 4, i: 10999] avg mini-batch loss: 3.370
[epoch: 4, i: 11499] avg mini-batch loss: 3.351
[epoch: 4, i: 11999] avg mini-batch loss: 3.398
[epoch: 4, i: 12499] avg mini-batch loss: 3.374

  0%|          | 0/12500 [00:00<?, ?it/s]

[epoch: 5, i:   499] avg mini-batch loss: 3.325
[epoch: 5, i:   999] avg mini-batch loss: 3.292
[epoch: 5, i:  1499] avg mini-batch loss: 3.323
[epoch: 5, i:  1999] avg mini-batch loss: 3.304
[epoch: 5, i:  2499] avg mini-batch loss: 3.321
[epoch: 5, i:  2999] avg mini-batch loss: 3.279
[epoch: 5, i:  3499] avg mini-batch loss: 3.255
[epoch: 5, i:  3999] avg mini-batch loss: 3.300
[epoch: 5, i:  4499] avg mini-batch loss: 3.324
[epoch: 5, i:  4999] avg mini-batch loss: 3.345
[epoch: 5, i:  5499] avg mini-batch loss: 3.317
[epoch: 5, i:  5999] avg mini-batch loss: 3.291
[epoch: 5, i:  6499] avg mini-batch loss: 3.294
[epoch: 5, i:  6999] avg mini-batch loss: 3.273
[epoch: 5, i:  7499] avg mini-batch loss: 3.288
[epoch: 5, i:  7999] avg mini-batch loss: 3.285
[epoch: 5, i:  8499] avg mini-batch loss: 3.282
[epoch: 5, i:  8999] avg mini-batch loss: 3.278
[epoch: 5, i:  9499] avg mini-batch loss: 3.299
[epoch: 5, i:  9999] avg mini-batch loss: 3.253
[epoch: 5, i: 10499] avg mini-batch loss: 3.264
[epoch: 5, i: 10999] avg mini-batch loss: 3.219
[epoch: 5, i: 11499] avg mini-batch loss: 3.278
[epoch: 5, i: 11999] avg mini-batch loss: 3.215
[epoch: 5, i: 12499] avg mini-batch loss: 3.227

  0%|          | 0/12500 [00:00<?, ?it/s]

[epoch: 6, i:   499] avg mini-batch loss: 3.153
[epoch: 6, i:   999] avg mini-batch loss: 3.242
[epoch: 6, i:  1499] avg mini-batch loss: 3.202
[epoch: 6, i:  1999] avg mini-batch loss: 3.179
[epoch: 6, i:  2499] avg mini-batch loss: 3.161
[epoch: 6, i:  2999] avg mini-batch loss: 3.167
[epoch: 6, i:  3499] avg mini-batch loss: 3.146
[epoch: 6, i:  3999] avg mini-batch loss: 3.156
[epoch: 6, i:  4499] avg mini-batch loss: 3.202
[epoch: 6, i:  4999] avg mini-batch loss: 3.132
[epoch: 6, i:  5499] avg mini-batch loss: 3.176
[epoch: 6, i:  5999] avg mini-batch loss: 3.162
[epoch: 6, i:  6499] avg mini-batch loss: 3.158
```

```
[epoch: 6, i:  6999] avg mini-batch loss: 3.168
[epoch: 6, i:  7499] avg mini-batch loss: 3.152
[epoch: 6, i:  7999] avg mini-batch loss: 3.131
[epoch: 6, i:  8499] avg mini-batch loss: 3.113
[epoch: 6, i:  8999] avg mini-batch loss: 3.171
[epoch: 6, i:  9499] avg mini-batch loss: 3.146
[epoch: 6, i:  9999] avg mini-batch loss: 3.131
[epoch: 6, i: 10499] avg mini-batch loss: 3.106
[epoch: 6, i: 10999] avg mini-batch loss: 3.167
[epoch: 6, i: 11499] avg mini-batch loss: 3.120
[epoch: 6, i: 11999] avg mini-batch loss: 3.113
[epoch: 6, i: 12499] avg mini-batch loss: 3.133
Finished Training.
```

Training Loss Curve

```
[8]: plt.plot(avg_losses)
     plt.xlabel('mini-batch index / {}'.format(print_freq))
     plt.ylabel('avg. mini-batch loss')
     plt.show()
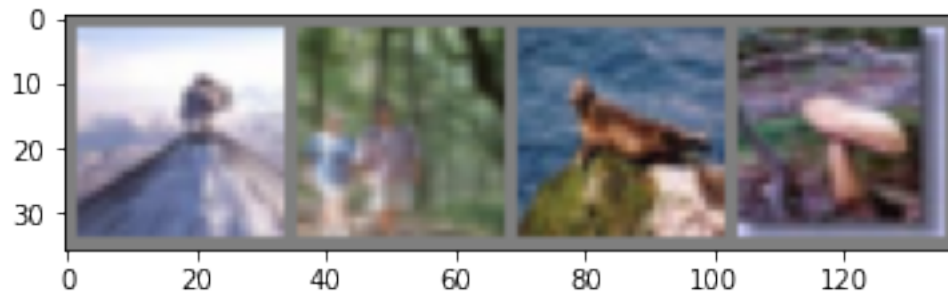```
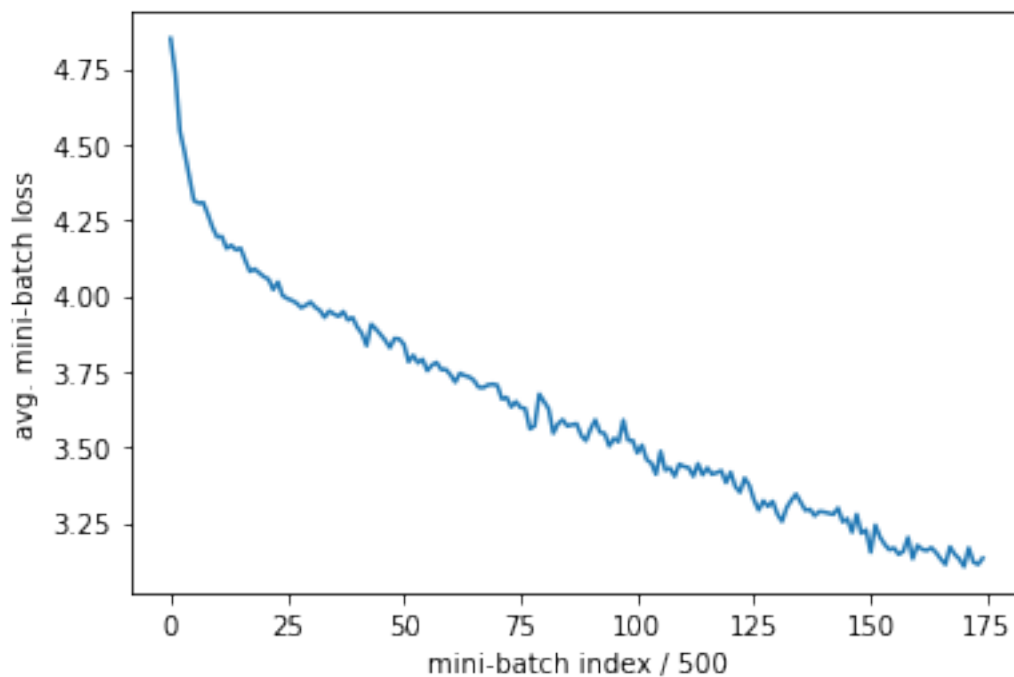


**Evaluate on Test Dataset**

```
[9]: # Check several images.
     dataiter = iter(testloader)
     images, labels = next(dataiter)
```

```
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
outputs = net(images.to(device))
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```



```
GroundTruth:   mountain forest   seal mushroom
Predicted:   rocket caterpillar beaver skunk
```

[10]:
```python
# Get test accuracy.
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 21 %
```

[11]:
```python
# Get test accuracy for each class.
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
```

```
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1

for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipykernel_14672/2959918759.py in <module>
     11             for i in range(4):
     12                 label = labels[i]
---> 13                 class_correct[label] += c[i].item()
     14                 class_total[label] += 1
     15

IndexError: list index out of range
```

```
[ ]: # One of the changes I made was that I added another layer in the network
     # that takes the output from the second convolutional layer, applies ReLU␣
      ↪activation,
     # and passes it through a 2x2 AvgPool layer to capture more relationships.
     # I also reduced the learning rate of the optimizer to half (0.0005) which
     # which allowed the optimizer to take smaller steps towards the minimum of
     # the loss function which might allow for a better chance of finding the global␣
      ↪min of loss.
```