

final_10

March 26, 2023

```
[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

```
[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]: # 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)))
```



table seal bus cup

Choose a Device

```
[4]: # 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

```
[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.LeakyReLU()
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.relu2 = nn.LeakyReLU()
        self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.pool3 = nn.MaxPool2d(2, 2)
        self.relu3 = nn.SELU()
        self.fc1 = nn.Linear(256 * 4 * 4, 1024)
        self.bn4 = nn.BatchNorm1d(1024)
        self.relu4 = nn.SELU()
        self.fc2 = nn.Linear(1024, 100)

    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 = x.view(-1, 256 * 4 * 4)
        x = self.relu4(self.fc1(x))
        x = 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): LeakyReLU(negative_slope=0.01)
    (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): LeakyReLU(negative_slope=0.01)
    (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): SELU()
    (fc1): Linear(in_features=4096, out_features=1024, bias=True)
    (bn4): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu4): SELU()
    (fc2): Linear(in_features=1024, out_features=100, bias=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=0.0001, momentum=0.9)
    opt = optim.Adam(net.parameters(), lr=0.0001)

```

Training Procedure

```

[7]: import sys
    from tqdm.notebook import tqdm

    avg_losses = []    # Avg. losses.
    epochs = 8         # 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()

# 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: 4.378
[epoch: 0, i:   999] avg mini-batch loss: 4.028
[epoch: 0, i:  1499] avg mini-batch loss: 3.797
[epoch: 0, i:  1999] avg mini-batch loss: 3.621
[epoch: 0, i:  2499] avg mini-batch loss: 3.518
[epoch: 0, i:  2999] avg mini-batch loss: 3.421
[epoch: 0, i:  3499] avg mini-batch loss: 3.331
[epoch: 0, i:  3999] avg mini-batch loss: 3.258
[epoch: 0, i:  4499] avg mini-batch loss: 3.191
[epoch: 0, i:  4999] avg mini-batch loss: 3.052
[epoch: 0, i:  5499] avg mini-batch loss: 3.089
[epoch: 0, i:  5999] avg mini-batch loss: 3.008
[epoch: 0, i:  6499] avg mini-batch loss: 2.986
[epoch: 0, i:  6999] avg mini-batch loss: 2.900
[epoch: 0, i:  7499] avg mini-batch loss: 2.861
[epoch: 0, i:  7999] avg mini-batch loss: 2.898
[epoch: 0, i:  8499] avg mini-batch loss: 2.838
[epoch: 0, i:  8999] avg mini-batch loss: 2.751
[epoch: 0, i:  9499] avg mini-batch loss: 2.815
[epoch: 0, i:  9999] avg mini-batch loss: 2.692
[epoch: 0, i: 10499] avg mini-batch loss: 2.684
[epoch: 0, i: 10999] avg mini-batch loss: 2.721

```

```
[epoch: 0, i: 11499] avg mini-batch loss: 2.706
[epoch: 0, i: 11999] avg mini-batch loss: 2.673
[epoch: 0, i: 12499] avg mini-batch loss: 2.699
```

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

```
[epoch: 1, i: 499] avg mini-batch loss: 2.447
[epoch: 1, i: 999] avg mini-batch loss: 2.395
[epoch: 1, i: 1499] avg mini-batch loss: 2.479
[epoch: 1, i: 1999] avg mini-batch loss: 2.456
[epoch: 1, i: 2499] avg mini-batch loss: 2.412
[epoch: 1, i: 2999] avg mini-batch loss: 2.366
[epoch: 1, i: 3499] avg mini-batch loss: 2.430
[epoch: 1, i: 3999] avg mini-batch loss: 2.429
[epoch: 1, i: 4499] avg mini-batch loss: 2.449
[epoch: 1, i: 4999] avg mini-batch loss: 2.362
[epoch: 1, i: 5499] avg mini-batch loss: 2.342
[epoch: 1, i: 5999] avg mini-batch loss: 2.277
[epoch: 1, i: 6499] avg mini-batch loss: 2.359
[epoch: 1, i: 6999] avg mini-batch loss: 2.337
[epoch: 1, i: 7499] avg mini-batch loss: 2.317
[epoch: 1, i: 7999] avg mini-batch loss: 2.363
[epoch: 1, i: 8499] avg mini-batch loss: 2.338
[epoch: 1, i: 8999] avg mini-batch loss: 2.287
[epoch: 1, i: 9499] avg mini-batch loss: 2.300
[epoch: 1, i: 9999] avg mini-batch loss: 2.279
[epoch: 1, i: 10499] avg mini-batch loss: 2.317
[epoch: 1, i: 10999] avg mini-batch loss: 2.245
[epoch: 1, i: 11499] avg mini-batch loss: 2.241
[epoch: 1, i: 11999] avg mini-batch loss: 2.276
[epoch: 1, i: 12499] avg mini-batch loss: 2.225
```

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

```
[epoch: 2, i: 499] avg mini-batch loss: 2.003
[epoch: 2, i: 999] avg mini-batch loss: 1.920
[epoch: 2, i: 1499] avg mini-batch loss: 2.081
[epoch: 2, i: 1999] avg mini-batch loss: 1.972
[epoch: 2, i: 2499] avg mini-batch loss: 1.947
[epoch: 2, i: 2999] avg mini-batch loss: 2.074
[epoch: 2, i: 3499] avg mini-batch loss: 1.972
[epoch: 2, i: 3999] avg mini-batch loss: 2.005
[epoch: 2, i: 4499] avg mini-batch loss: 1.991
[epoch: 2, i: 4999] avg mini-batch loss: 2.018
[epoch: 2, i: 5499] avg mini-batch loss: 1.920
[epoch: 2, i: 5999] avg mini-batch loss: 2.044
[epoch: 2, i: 6499] avg mini-batch loss: 1.979
[epoch: 2, i: 6999] avg mini-batch loss: 2.005
[epoch: 2, i: 7499] avg mini-batch loss: 1.971
[epoch: 2, i: 7999] avg mini-batch loss: 1.979
```

```

[epoch: 2, i: 8499] avg mini-batch loss: 2.006
[epoch: 2, i: 8999] avg mini-batch loss: 1.926
[epoch: 2, i: 9499] avg mini-batch loss: 1.993
[epoch: 2, i: 9999] avg mini-batch loss: 1.940
[epoch: 2, i: 10499] avg mini-batch loss: 1.947
[epoch: 2, i: 10999] avg mini-batch loss: 1.917
[epoch: 2, i: 11499] avg mini-batch loss: 1.977
[epoch: 2, i: 11999] avg mini-batch loss: 1.953
[epoch: 2, i: 12499] avg mini-batch loss: 1.936

```

```

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

```

```

[epoch: 3, i: 499] avg mini-batch loss: 1.543
[epoch: 3, i: 999] avg mini-batch loss: 1.668
[epoch: 3, i: 1499] avg mini-batch loss: 1.641
[epoch: 3, i: 1999] avg mini-batch loss: 1.572
[epoch: 3, i: 2499] avg mini-batch loss: 1.664
[epoch: 3, i: 2999] avg mini-batch loss: 1.634
[epoch: 3, i: 3499] avg mini-batch loss: 1.648
[epoch: 3, i: 3999] avg mini-batch loss: 1.641
[epoch: 3, i: 4499] avg mini-batch loss: 1.620
[epoch: 3, i: 4999] avg mini-batch loss: 1.668
[epoch: 3, i: 5499] avg mini-batch loss: 1.688
[epoch: 3, i: 5999] avg mini-batch loss: 1.678
[epoch: 3, i: 6499] avg mini-batch loss: 1.596
[epoch: 3, i: 6999] avg mini-batch loss: 1.672
[epoch: 3, i: 7499] avg mini-batch loss: 1.669
[epoch: 3, i: 7999] avg mini-batch loss: 1.649
[epoch: 3, i: 8499] avg mini-batch loss: 1.741
[epoch: 3, i: 8999] avg mini-batch loss: 1.666
[epoch: 3, i: 9499] avg mini-batch loss: 1.680
[epoch: 3, i: 9999] avg mini-batch loss: 1.625
[epoch: 3, i: 10499] avg mini-batch loss: 1.701
[epoch: 3, i: 10999] avg mini-batch loss: 1.663
[epoch: 3, i: 11499] avg mini-batch loss: 1.678
[epoch: 3, i: 11999] avg mini-batch loss: 1.623
[epoch: 3, i: 12499] avg mini-batch loss: 1.707

```

```

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

```

```

[epoch: 4, i: 499] avg mini-batch loss: 1.175
[epoch: 4, i: 999] avg mini-batch loss: 1.310
[epoch: 4, i: 1499] avg mini-batch loss: 1.231
[epoch: 4, i: 1999] avg mini-batch loss: 1.252
[epoch: 4, i: 2499] avg mini-batch loss: 1.279
[epoch: 4, i: 2999] avg mini-batch loss: 1.275
[epoch: 4, i: 3499] avg mini-batch loss: 1.284
[epoch: 4, i: 3999] avg mini-batch loss: 1.347
[epoch: 4, i: 4499] avg mini-batch loss: 1.292
[epoch: 4, i: 4999] avg mini-batch loss: 1.342

```

```

[epoch: 4, i: 5499] avg mini-batch loss: 1.245
[epoch: 4, i: 5999] avg mini-batch loss: 1.358
[epoch: 4, i: 6499] avg mini-batch loss: 1.329
[epoch: 4, i: 6999] avg mini-batch loss: 1.397
[epoch: 4, i: 7499] avg mini-batch loss: 1.368
[epoch: 4, i: 7999] avg mini-batch loss: 1.290
[epoch: 4, i: 8499] avg mini-batch loss: 1.370
[epoch: 4, i: 8999] avg mini-batch loss: 1.371
[epoch: 4, i: 9499] avg mini-batch loss: 1.310
[epoch: 4, i: 9999] avg mini-batch loss: 1.310
[epoch: 4, i: 10499] avg mini-batch loss: 1.355
[epoch: 4, i: 10999] avg mini-batch loss: 1.391
[epoch: 4, i: 11499] avg mini-batch loss: 1.375
[epoch: 4, i: 11999] avg mini-batch loss: 1.310
[epoch: 4, i: 12499] avg mini-batch loss: 1.365

```

```

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

```

```

[epoch: 5, i: 499] avg mini-batch loss: 0.854
[epoch: 5, i: 999] avg mini-batch loss: 0.913
[epoch: 5, i: 1499] avg mini-batch loss: 0.918
[epoch: 5, i: 1999] avg mini-batch loss: 0.917
[epoch: 5, i: 2499] avg mini-batch loss: 0.933
[epoch: 5, i: 2999] avg mini-batch loss: 0.990
[epoch: 5, i: 3499] avg mini-batch loss: 0.916
[epoch: 5, i: 3999] avg mini-batch loss: 0.942
[epoch: 5, i: 4499] avg mini-batch loss: 0.987
[epoch: 5, i: 4999] avg mini-batch loss: 0.969
[epoch: 5, i: 5499] avg mini-batch loss: 0.966
[epoch: 5, i: 5999] avg mini-batch loss: 0.943
[epoch: 5, i: 6499] avg mini-batch loss: 0.912
[epoch: 5, i: 6999] avg mini-batch loss: 1.000
[epoch: 5, i: 7499] avg mini-batch loss: 1.029
[epoch: 5, i: 7999] avg mini-batch loss: 1.062
[epoch: 5, i: 8499] avg mini-batch loss: 1.033
[epoch: 5, i: 8999] avg mini-batch loss: 1.096
[epoch: 5, i: 9499] avg mini-batch loss: 1.060
[epoch: 5, i: 9999] avg mini-batch loss: 0.966
[epoch: 5, i: 10499] avg mini-batch loss: 0.985
[epoch: 5, i: 10999] avg mini-batch loss: 1.030
[epoch: 5, i: 11499] avg mini-batch loss: 1.003
[epoch: 5, i: 11999] avg mini-batch loss: 1.113
[epoch: 5, i: 12499] avg mini-batch loss: 1.127

```

```

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

```

```

[epoch: 6, i: 499] avg mini-batch loss: 0.548
[epoch: 6, i: 999] avg mini-batch loss: 0.566
[epoch: 6, i: 1499] avg mini-batch loss: 0.582
[epoch: 6, i: 1999] avg mini-batch loss: 0.546

```



```

[epoch: 6, i: 2499] avg mini-batch loss: 0.613
[epoch: 6, i: 2999] avg mini-batch loss: 0.648
[epoch: 6, i: 3499] avg mini-batch loss: 0.602
[epoch: 6, i: 3999] avg mini-batch loss: 0.691
[epoch: 6, i: 4499] avg mini-batch loss: 0.665
[epoch: 6, i: 4999] avg mini-batch loss: 0.697
[epoch: 6, i: 5499] avg mini-batch loss: 0.684
[epoch: 6, i: 5999] avg mini-batch loss: 0.742
[epoch: 6, i: 6499] avg mini-batch loss: 0.693
[epoch: 6, i: 6999] avg mini-batch loss: 0.689
[epoch: 6, i: 7499] avg mini-batch loss: 0.660
[epoch: 6, i: 7999] avg mini-batch loss: 0.735
[epoch: 6, i: 8499] avg mini-batch loss: 0.684
[epoch: 6, i: 8999] avg mini-batch loss: 0.687
[epoch: 6, i: 9499] avg mini-batch loss: 0.735
[epoch: 6, i: 9999] avg mini-batch loss: 0.734
[epoch: 6, i: 10499] avg mini-batch loss: 0.757
[epoch: 6, i: 10999] avg mini-batch loss: 0.765
[epoch: 6, i: 11499] avg mini-batch loss: 0.726
[epoch: 6, i: 11999] avg mini-batch loss: 0.734
[epoch: 6, i: 12499] avg mini-batch loss: 0.714

```

```

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

```

```

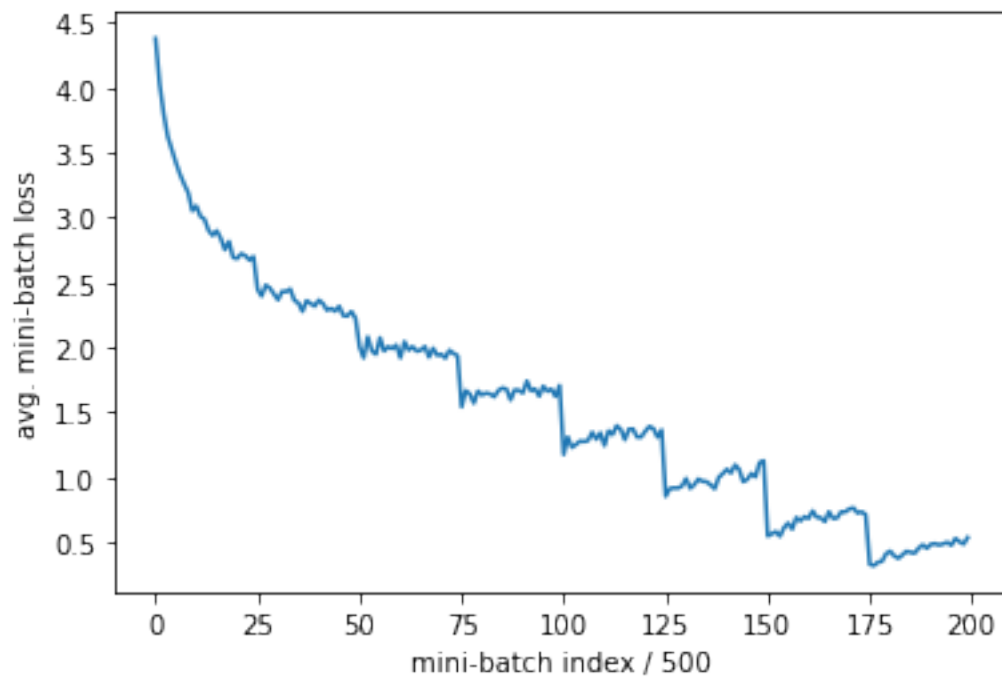
[epoch: 7, i: 499] avg mini-batch loss: 0.331
[epoch: 7, i: 999] avg mini-batch loss: 0.320
[epoch: 7, i: 1499] avg mini-batch loss: 0.347
[epoch: 7, i: 1999] avg mini-batch loss: 0.353
[epoch: 7, i: 2499] avg mini-batch loss: 0.412
[epoch: 7, i: 2999] avg mini-batch loss: 0.431
[epoch: 7, i: 3499] avg mini-batch loss: 0.395
[epoch: 7, i: 3999] avg mini-batch loss: 0.377
[epoch: 7, i: 4499] avg mini-batch loss: 0.402
[epoch: 7, i: 4999] avg mini-batch loss: 0.431
[epoch: 7, i: 5499] avg mini-batch loss: 0.422
[epoch: 7, i: 5999] avg mini-batch loss: 0.417
[epoch: 7, i: 6499] avg mini-batch loss: 0.455
[epoch: 7, i: 6999] avg mini-batch loss: 0.480
[epoch: 7, i: 7499] avg mini-batch loss: 0.454
[epoch: 7, i: 7999] avg mini-batch loss: 0.485
[epoch: 7, i: 8499] avg mini-batch loss: 0.489
[epoch: 7, i: 8999] avg mini-batch loss: 0.479
[epoch: 7, i: 9499] avg mini-batch loss: 0.488
[epoch: 7, i: 9999] avg mini-batch loss: 0.500
[epoch: 7, i: 10499] avg mini-batch loss: 0.477
[epoch: 7, i: 10999] avg mini-batch loss: 0.529
[epoch: 7, i: 11499] avg mini-batch loss: 0.504
[epoch: 7, i: 11999] avg mini-batch loss: 0.486
[epoch: 7, i: 12499] avg mini-batch loss: 0.537

```

Finished Training.

Training Loss Curve

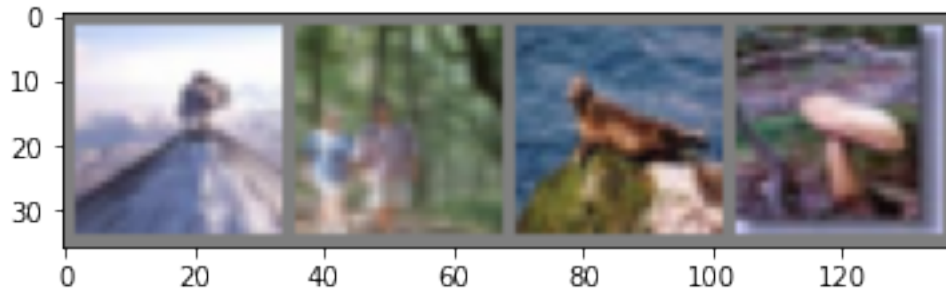
```
[8]: plt.plot(avg_losses)
plt.xlabel('mini-batch index / {}'.format(print_freq))
plt.ylabel('avg. mini-batch loss')
plt.show()
```



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: dolphin kangaroo otter mushroom

```
[10]: # 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: 44 %

```
[11]: # Get test accuracy for each class.
class_correct = [0] * len(classes)
class_total = [0] * len(classes)
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(len(labels)):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1
```

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

Accuracy of apple : 76 %
Accuracy of aquarium_fish : 61 %
Accuracy of baby : 33 %
Accuracy of bear : 11 %
Accuracy of beaver : 34 %
Accuracy of bed : 44 %
Accuracy of bee : 43 %
Accuracy of beetle : 62 %
Accuracy of bicycle : 51 %
Accuracy of bottle : 66 %
Accuracy of bowl : 33 %
Accuracy of boy : 30 %
Accuracy of bridge : 41 %
Accuracy of bus : 39 %
Accuracy of butterfly : 30 %
Accuracy of camel : 25 %
Accuracy of can : 39 %
Accuracy of castle : 72 %
Accuracy of caterpillar : 39 %
Accuracy of cattle : 36 %
Accuracy of chair : 68 %
Accuracy of chimpanzee : 57 %
Accuracy of clock : 39 %
Accuracy of cloud : 71 %
Accuracy of cockroach : 66 %
Accuracy of couch : 29 %
Accuracy of crab : 46 %
Accuracy of crocodile : 23 %
Accuracy of cup : 65 %
Accuracy of dinosaur : 43 %
Accuracy of dolphin : 55 %
Accuracy of elephant : 45 %
Accuracy of flatfish : 37 %
Accuracy of forest : 34 %
Accuracy of fox : 43 %
Accuracy of girl : 24 %
Accuracy of hamster : 42 %
Accuracy of house : 48 %
Accuracy of kangaroo : 31 %
Accuracy of keyboard : 42 %
Accuracy of lamp : 35 %
Accuracy of lawn_mower : 68 %
Accuracy of leopard : 43 %

Accuracy of lion : 47 %
Accuracy of lizard : 21 %
Accuracy of lobster : 23 %
Accuracy of man : 29 %
Accuracy of maple_tree : 59 %
Accuracy of motorcycle : 75 %
Accuracy of mountain : 60 %
Accuracy of mouse : 40 %
Accuracy of mushroom : 48 %
Accuracy of oak_tree : 62 %
Accuracy of orange : 57 %
Accuracy of orchid : 56 %
Accuracy of otter : 14 %
Accuracy of palm_tree : 61 %
Accuracy of pear : 46 %
Accuracy of pickup_truck : 55 %
Accuracy of pine_tree : 39 %
Accuracy of plain : 66 %
Accuracy of plate : 64 %
Accuracy of poppy : 65 %
Accuracy of porcupine : 37 %
Accuracy of possum : 12 %
Accuracy of rabbit : 33 %
Accuracy of raccoon : 37 %
Accuracy of ray : 32 %
Accuracy of road : 71 %
Accuracy of rocket : 57 %
Accuracy of rose : 31 %
Accuracy of sea : 64 %
Accuracy of seal : 17 %
Accuracy of shark : 40 %
Accuracy of shrew : 30 %
Accuracy of skunk : 65 %
Accuracy of skyscraper : 57 %
Accuracy of snail : 36 %
Accuracy of snake : 17 %
Accuracy of spider : 37 %
Accuracy of squirrel : 21 %
Accuracy of streetcar : 49 %
Accuracy of sunflower : 63 %
Accuracy of sweet_pepper : 52 %
Accuracy of table : 49 %
Accuracy of tank : 53 %
Accuracy of telephone : 48 %
Accuracy of television : 55 %
Accuracy of tiger : 37 %
Accuracy of tractor : 49 %
Accuracy of train : 34 %

Accuracy of trout : 55 %
Accuracy of tulip : 41 %
Accuracy of turtle : 38 %
Accuracy of wardrobe : 67 %
Accuracy of whale : 49 %
Accuracy of willow_tree : 23 %
Accuracy of wolf : 33 %
Accuracy of woman : 24 %
Accuracy of worm : 44 %

```
[12]: # 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.
```