# final

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',
```

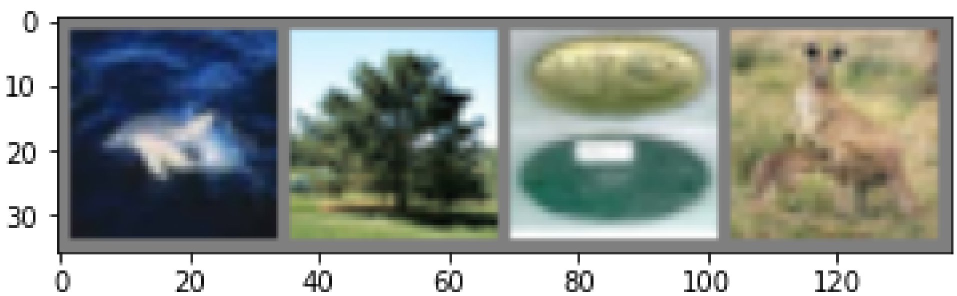
```
            'palm_tree', 'pear', 'pickup_truck', 'pine_tree', 'plain',
            'plate', 'poppy', 'porcupine', 'possum', 'rabbit', 'raccoon',
            'ray', 'road', 'rocket', 'rose', 'sea', 'seal', 'shark',
            'shrew', 'skunk', 'skyscraper', 'snail', 'snake', 'spider',
            '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)))
```



```
dolphin pine_tree plate kangaroo
```

### 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.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.fc1 = nn.Linear(256 * 4 * 4, 1024)
        self.bn4 = nn.BatchNorm1d(1024)
        self.relu4 = nn.ReLU()
        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): 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()
  (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): ReLU()
  (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=0.0005, momentum=0.9)
```

### Training Procedure

[7]:
```
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()

        # 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: 4.603
[epoch: 0, i:   999] avg mini-batch loss: 4.598
[epoch: 0, i:  1499] avg mini-batch loss: 4.593
[epoch: 0, i:  1999] avg mini-batch loss: 4.572
[epoch: 0, i:  2499] avg mini-batch loss: 4.537
[epoch: 0, i:  2999] avg mini-batch loss: 4.439
[epoch: 0, i:  3499] avg mini-batch loss: 4.342
[epoch: 0, i:  3999] avg mini-batch loss: 4.239
[epoch: 0, i:  4499] avg mini-batch loss: 4.162
[epoch: 0, i:  4999] avg mini-batch loss: 4.089
[epoch: 0, i:  5499] avg mini-batch loss: 4.036
[epoch: 0, i:  5999] avg mini-batch loss: 4.038
[epoch: 0, i:  6499] avg mini-batch loss: 3.987
[epoch: 0, i:  6999] avg mini-batch loss: 3.924
[epoch: 0, i:  7499] avg mini-batch loss: 3.885
[epoch: 0, i:  7999] avg mini-batch loss: 3.829
[epoch: 0, i:  8499] avg mini-batch loss: 3.855
[epoch: 0, i:  8999] avg mini-batch loss: 3.798
[epoch: 0, i:  9499] avg mini-batch loss: 3.743
[epoch: 0, i:  9999] avg mini-batch loss: 3.743
[epoch: 0, i: 10499] avg mini-batch loss: 3.712
[epoch: 0, i: 10999] avg mini-batch loss: 3.656
[epoch: 0, i: 11499] avg mini-batch loss: 3.637
```

```
[epoch: 0, i: 11999] avg mini-batch loss: 3.615
[epoch: 0, i: 12499] avg mini-batch loss: 3.552
```

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

```
[epoch: 1, i:   499] avg mini-batch loss: 3.484
[epoch: 1, i:   999] avg mini-batch loss: 3.514
[epoch: 1, i:  1499] avg mini-batch loss: 3.426
[epoch: 1, i:  1999] avg mini-batch loss: 3.432
[epoch: 1, i:  2499] avg mini-batch loss: 3.363
[epoch: 1, i:  2999] avg mini-batch loss: 3.370
[epoch: 1, i:  3499] avg mini-batch loss: 3.290
[epoch: 1, i:  3999] avg mini-batch loss: 3.296
[epoch: 1, i:  4499] avg mini-batch loss: 3.317
[epoch: 1, i:  4999] avg mini-batch loss: 3.288
[epoch: 1, i:  5499] avg mini-batch loss: 3.293
[epoch: 1, i:  5999] avg mini-batch loss: 3.263
[epoch: 1, i:  6499] avg mini-batch loss: 3.175
[epoch: 1, i:  6999] avg mini-batch loss: 3.236
[epoch: 1, i:  7499] avg mini-batch loss: 3.203
[epoch: 1, i:  7999] avg mini-batch loss: 3.098
[epoch: 1, i:  8499] avg mini-batch loss: 3.130
[epoch: 1, i:  8999] avg mini-batch loss: 3.045
[epoch: 1, i:  9499] avg mini-batch loss: 3.118
[epoch: 1, i:  9999] avg mini-batch loss: 3.048
[epoch: 1, i: 10499] avg mini-batch loss: 3.156
[epoch: 1, i: 10999] avg mini-batch loss: 3.025
[epoch: 1, i: 11499] avg mini-batch loss: 2.982
[epoch: 1, i: 11999] avg mini-batch loss: 3.031
[epoch: 1, i: 12499] avg mini-batch loss: 2.957
```

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

```
[epoch: 2, i:   499] avg mini-batch loss: 2.813
[epoch: 2, i:   999] avg mini-batch loss: 2.789
[epoch: 2, i:  1499] avg mini-batch loss: 2.846
[epoch: 2, i:  1999] avg mini-batch loss: 2.879
[epoch: 2, i:  2499] avg mini-batch loss: 2.915
[epoch: 2, i:  2999] avg mini-batch loss: 2.844
[epoch: 2, i:  3499] avg mini-batch loss: 2.730
[epoch: 2, i:  3999] avg mini-batch loss: 2.768
[epoch: 2, i:  4499] avg mini-batch loss: 2.765
[epoch: 2, i:  4999] avg mini-batch loss: 2.822
[epoch: 2, i:  5499] avg mini-batch loss: 2.750
[epoch: 2, i:  5999] avg mini-batch loss: 2.754
[epoch: 2, i:  6499] avg mini-batch loss: 2.749
[epoch: 2, i:  6999] avg mini-batch loss: 2.687
[epoch: 2, i:  7499] avg mini-batch loss: 2.730
[epoch: 2, i:  7999] avg mini-batch loss: 2.721
[epoch: 2, i:  8499] avg mini-batch loss: 2.701
```

```
[epoch: 2, i:  8999] avg mini-batch loss: 2.690
[epoch: 2, i:  9499] avg mini-batch loss: 2.621
[epoch: 2, i:  9999] avg mini-batch loss: 2.630
[epoch: 2, i: 10499] avg mini-batch loss: 2.651
[epoch: 2, i: 10999] avg mini-batch loss: 2.647
[epoch: 2, i: 11499] avg mini-batch loss: 2.603
[epoch: 2, i: 11999] avg mini-batch loss: 2.602
[epoch: 2, i: 12499] avg mini-batch loss: 2.593
```

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

```
[epoch: 3, i:   499] avg mini-batch loss: 2.361
[epoch: 3, i:   999] avg mini-batch loss: 2.410
[epoch: 3, i:  1499] avg mini-batch loss: 2.350
[epoch: 3, i:  1999] avg mini-batch loss: 2.399
[epoch: 3, i:  2499] avg mini-batch loss: 2.407
[epoch: 3, i:  2999] avg mini-batch loss: 2.373
[epoch: 3, i:  3499] avg mini-batch loss: 2.350
[epoch: 3, i:  3999] avg mini-batch loss: 2.412
[epoch: 3, i:  4499] avg mini-batch loss: 2.423
[epoch: 3, i:  4999] avg mini-batch loss: 2.423
[epoch: 3, i:  5499] avg mini-batch loss: 2.445
[epoch: 3, i:  5999] avg mini-batch loss: 2.323
[epoch: 3, i:  6499] avg mini-batch loss: 2.386
[epoch: 3, i:  6999] avg mini-batch loss: 2.291
[epoch: 3, i:  7499] avg mini-batch loss: 2.346
[epoch: 3, i:  7999] avg mini-batch loss: 2.348
[epoch: 3, i:  8499] avg mini-batch loss: 2.319
[epoch: 3, i:  8999] avg mini-batch loss: 2.345
[epoch: 3, i:  9499] avg mini-batch loss: 2.314
[epoch: 3, i:  9999] avg mini-batch loss: 2.328
[epoch: 3, i: 10499] avg mini-batch loss: 2.349
[epoch: 3, i: 10999] avg mini-batch loss: 2.334
[epoch: 3, i: 11499] avg mini-batch loss: 2.247
[epoch: 3, i: 11999] avg mini-batch loss: 2.334
[epoch: 3, i: 12499] avg mini-batch loss: 2.304
```

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

```
[epoch: 4, i:   499] avg mini-batch loss: 1.921
[epoch: 4, i:   999] avg mini-batch loss: 2.030
[epoch: 4, i:  1499] avg mini-batch loss: 2.007
[epoch: 4, i:  1999] avg mini-batch loss: 2.024
[epoch: 4, i:  2499] avg mini-batch loss: 2.050
[epoch: 4, i:  2999] avg mini-batch loss: 1.966
[epoch: 4, i:  3499] avg mini-batch loss: 2.048
[epoch: 4, i:  3999] avg mini-batch loss: 2.025
[epoch: 4, i:  4499] avg mini-batch loss: 2.008
[epoch: 4, i:  4999] avg mini-batch loss: 2.039
[epoch: 4, i:  5499] avg mini-batch loss: 2.038
```

```
[epoch: 4, i:  5999] avg mini-batch loss: 2.092
[epoch: 4, i:  6499] avg mini-batch loss: 2.013
[epoch: 4, i:  6999] avg mini-batch loss: 2.021
[epoch: 4, i:  7499] avg mini-batch loss: 2.080
[epoch: 4, i:  7999] avg mini-batch loss: 2.009
[epoch: 4, i:  8499] avg mini-batch loss: 2.073
[epoch: 4, i:  8999] avg mini-batch loss: 1.984
[epoch: 4, i:  9499] avg mini-batch loss: 1.955
[epoch: 4, i:  9999] avg mini-batch loss: 1.977
[epoch: 4, i: 10499] avg mini-batch loss: 1.967
[epoch: 4, i: 10999] avg mini-batch loss: 1.970
[epoch: 4, i: 11499] avg mini-batch loss: 1.981
[epoch: 4, i: 11999] avg mini-batch loss: 2.001
[epoch: 4, i: 12499] avg mini-batch loss: 2.021
```

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

```
[epoch: 5, i:   499] avg mini-batch loss: 1.577
[epoch: 5, i:   999] avg mini-batch loss: 1.627
[epoch: 5, i:  1499] avg mini-batch loss: 1.617
[epoch: 5, i:  1999] avg mini-batch loss: 1.618
[epoch: 5, i:  2499] avg mini-batch loss: 1.671
[epoch: 5, i:  2999] avg mini-batch loss: 1.634
[epoch: 5, i:  3499] avg mini-batch loss: 1.651
[epoch: 5, i:  3999] avg mini-batch loss: 1.624
[epoch: 5, i:  4499] avg mini-batch loss: 1.712
[epoch: 5, i:  4999] avg mini-batch loss: 1.606
[epoch: 5, i:  5499] avg mini-batch loss: 1.683
[epoch: 5, i:  5999] avg mini-batch loss: 1.664
[epoch: 5, i:  6499] avg mini-batch loss: 1.650
[epoch: 5, i:  6999] avg mini-batch loss: 1.688
[epoch: 5, i:  7499] avg mini-batch loss: 1.733
[epoch: 5, i:  7999] avg mini-batch loss: 1.671
[epoch: 5, i:  8499] avg mini-batch loss: 1.699
[epoch: 5, i:  8999] avg mini-batch loss: 1.688
[epoch: 5, i:  9499] avg mini-batch loss: 1.779
[epoch: 5, i:  9999] avg mini-batch loss: 1.662
[epoch: 5, i: 10499] avg mini-batch loss: 1.711
[epoch: 5, i: 10999] avg mini-batch loss: 1.745
[epoch: 5, i: 11499] avg mini-batch loss: 1.666
[epoch: 5, i: 11999] avg mini-batch loss: 1.647
[epoch: 5, i: 12499] avg mini-batch loss: 1.742
```

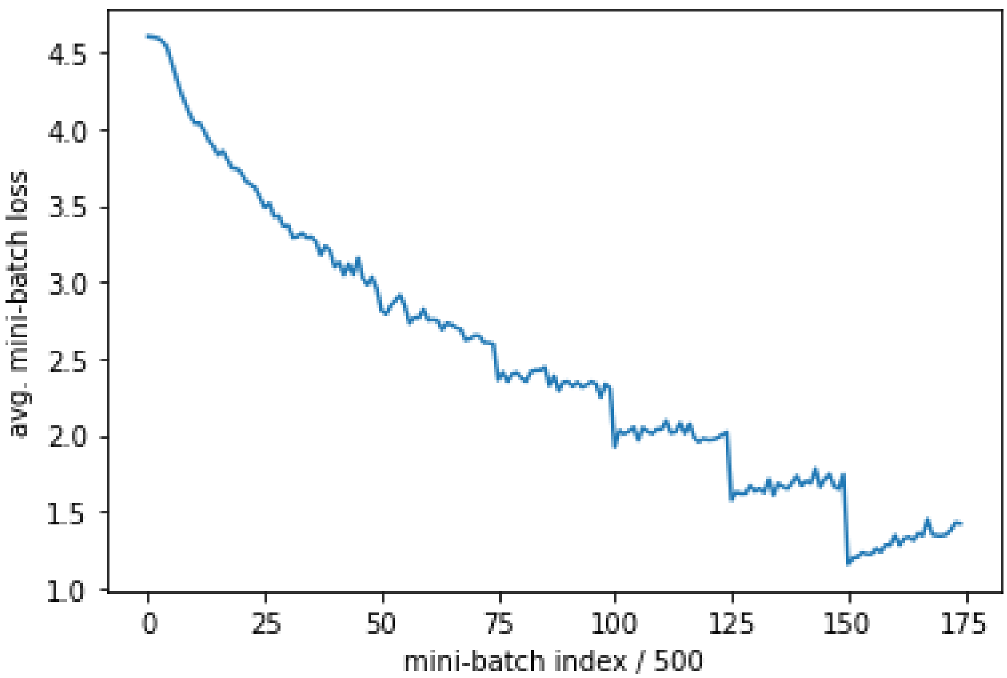
```
  0%|          | 0/12500 [00:00<?, ?it/s]
```

```
[epoch: 6, i:   499] avg mini-batch loss: 1.158
[epoch: 6, i:   999] avg mini-batch loss: 1.198
[epoch: 6, i:  1499] avg mini-batch loss: 1.205
[epoch: 6, i:  1999] avg mini-batch loss: 1.234
[epoch: 6, i:  2499] avg mini-batch loss: 1.223
```

```
[epoch: 6, i:  2999] avg mini-batch loss: 1.223
[epoch: 6, i:  3499] avg mini-batch loss: 1.259
[epoch: 6, i:  3999] avg mini-batch loss: 1.239
[epoch: 6, i:  4499] avg mini-batch loss: 1.283
[epoch: 6, i:  4999] avg mini-batch loss: 1.283
[epoch: 6, i:  5499] avg mini-batch loss: 1.346
[epoch: 6, i:  5999] avg mini-batch loss: 1.281
[epoch: 6, i:  6499] avg mini-batch loss: 1.327
[epoch: 6, i:  6999] avg mini-batch loss: 1.335
[epoch: 6, i:  7499] avg mini-batch loss: 1.315
[epoch: 6, i:  7999] avg mini-batch loss: 1.360
[epoch: 6, i:  8499] avg mini-batch loss: 1.345
[epoch: 6, i:  8999] avg mini-batch loss: 1.451
[epoch: 6, i:  9499] avg mini-batch loss: 1.354
[epoch: 6, i:  9999] avg mini-batch loss: 1.346
[epoch: 6, i: 10499] avg mini-batch loss: 1.345
[epoch: 6, i: 10999] avg mini-batch loss: 1.352
[epoch: 6, i: 11499] avg mini-batch loss: 1.382
[epoch: 6, i: 11999] avg mini-batch loss: 1.431
[epoch: 6, i: 12499] avg mini-batch loss: 1.424
Finished Training.
```

**Training Loss Curve**

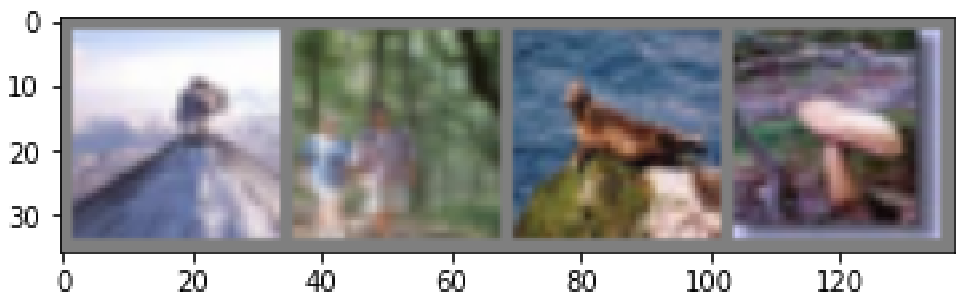
[8]:
```
plt.plot(avg_losses)
plt.xlabel('mini-batch index / {}'.format(print_freq))
plt.ylabel('avg. mini-batch loss')
plt.show()
```



**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:  spider forest camel trout
```

[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: 42 %
```

[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_6876/2959918759.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
```

[ ]:
```python
# 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.
```