In [2]:
```
#upload tiny imagenet folder into jupyter project

#import zipfile as zf
#files = zf.ZipFile("tiny-imagenet-200.zip", 'r')
#files.extractall()
#files.close
```

In [3]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import torch
import torchvision
import torchvision.datasets as datasets
import torch.utils.data as data
from torchvision.utils import make_grid
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import os

import vgg
import resnet
import googlenet
import alexnet
```

In [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.
```

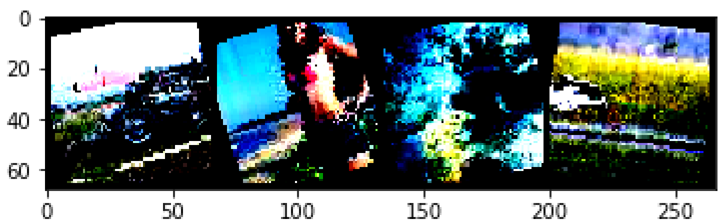
```
cuda:0
```

In [5]:

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomRotation(20),
        transforms.RandomHorizontalFlip(0.5),
        transforms.ToTensor(),
        transforms.Normalize([0.4802, 0.4481, 0.3975], [0.2302, 0.2265, 0.2262])
    ]),
    'val': transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.4802, 0.4481, 0.3975], [0.2302, 0.2265, 0.2262])
    ]),
    'test': transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.4802, 0.4481, 0.3975], [0.2302, 0.2265, 0.2262])
    ])
}

data_dir = 'tiny-imagenet-200/'

num_workers = {
    'train' : 100,
    'val'   : 0,
    'test'  : 0
}

image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val','test']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=100,
                                             shuffle=True,num_workers=num_worker
              for x in ['train', 'val', 'test']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val', 'test']}
```

In [26]:
```
def imshow(img):
    img = img.numpy().transpose((1, 2, 0))
    img = np.clip(img, 0, 1)
    plt.imshow(img)

images, labels = next(iter(dataloaders['train']))
print(labels)
grid = make_grid(images[:4], nrow=4)
imshow(grid)
```

```
tensor([164, 120, 196,  20, 146, 131,  38,  12, 139,   0,  72,   9, 188,  11,
        101,   6,  16, 157,  36, 189, 106,  66,  25, 194, 140, 187, 151,  68,
         62, 195, 159, 131,  85,  47,  11, 156,  99, 141,  48, 160,  65, 112,
         65,  68,  14,  35,   0, 101, 190, 167, 156, 168,   2, 149, 186,   7,
        107,  60,  37,  76, 169,   9, 157,  83,  62, 181,  97, 164,  48, 149,
         85,  96, 107, 144, 149,  24, 113,  65,  89,  16,  19,  31, 153,   8,
         59,  98, 135, 116, 110,  75,  44, 175,  37, 187, 180,  47, 148, 167,
        107, 176])
```



In [30]:
```
net = resnet.resnet18()   # Create the network instance.
net.to(device)
```

Out[30]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), b
ias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_m
ode=False)
  (layer1): Sequential(
    (0): 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_runn
ing_stats=True)
      (relu): ReLU(inplace=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_runn
ing_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_runn
ing_stats=True)
      (relu): ReLU(inplace=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_runn
ing_stats=True)
    )
  )
  (layer2): 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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
      (downsample): 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_run
ning_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_run
```

```
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
    )
  )
  (layer3): 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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
      (downsample): 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_run
ning_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_run
ning_stats=True)
      (relu): ReLU(inplace=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_run
ning_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
```

```
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

In [31]:

```
# 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=0.001, momentum=0.9)
```

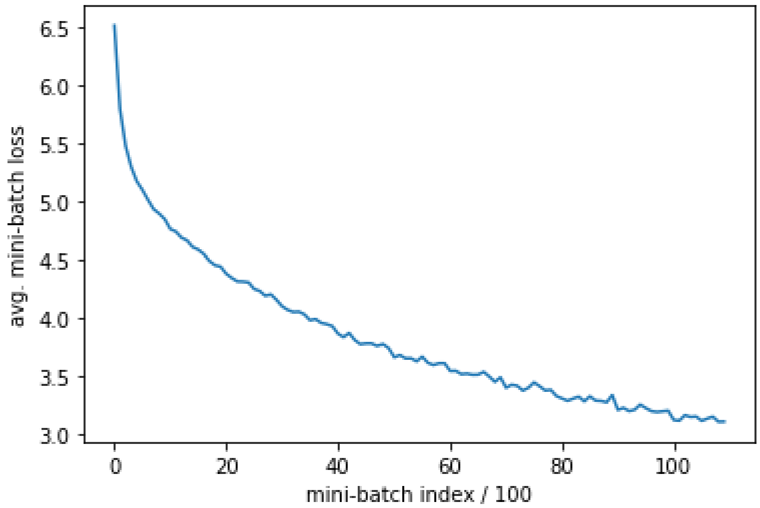
In [32]:
```python
avg_losses = []   # Avg. losses.
epochs = 11       # Total epochs.
print_freq = 100  # Print frequency.

for epoch in range(epochs):  # Loop over the dataset multiple times.
    running_loss = 0.0       # Initialize running loss.
    for i, data in enumerate(dataloaders['train'], 0):

        net.train()

        # 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))
            avg_losses.append(avg_loss)
            running_loss = 0.0

print('Finished Training.')
```

```
[epoch: 0, i:    99] avg mini-batch loss: 6.518
[epoch: 0, i:   199] avg mini-batch loss: 5.798
[epoch: 0, i:   299] avg mini-batch loss: 5.481
[epoch: 0, i:   399] avg mini-batch loss: 5.299
[epoch: 0, i:   499] avg mini-batch loss: 5.175
[epoch: 0, i:   599] avg mini-batch loss: 5.101
[epoch: 0, i:   699] avg mini-batch loss: 5.015
[epoch: 0, i:   799] avg mini-batch loss: 4.935
[epoch: 0, i:   899] avg mini-batch loss: 4.896
[epoch: 0, i:   999] avg mini-batch loss: 4.849
[epoch: 1, i:    99] avg mini-batch loss: 4.763
[epoch: 1, i:   199] avg mini-batch loss: 4.741
[epoch: 1, i:   299] avg mini-batch loss: 4.689
[epoch: 1, i:   399] avg mini-batch loss: 4.665
[epoch: 1, i:   499] avg mini-batch loss: 4.608
[epoch: 1, i:   599] avg mini-batch loss: 4.586
```

```
[epoch: 1, i:   699] avg mini-batch loss: 4.548
[epoch: 1, i:   799] avg mini-batch loss: 4.490
[epoch: 1, i:   899] avg mini-batch loss: 4.454
[epoch: 1, i:   999] avg mini-batch loss: 4.442
[epoch: 2, i:    99] avg mini-batch loss: 4.381
[epoch: 2, i:   199] avg mini-batch loss: 4.344
[epoch: 2, i:   299] avg mini-batch loss: 4.314
[epoch: 2, i:   399] avg mini-batch loss: 4.313
[epoch: 2, i:   499] avg mini-batch loss: 4.306
[epoch: 2, i:   599] avg mini-batch loss: 4.250
[epoch: 2, i:   699] avg mini-batch loss: 4.232
[epoch: 2, i:   799] avg mini-batch loss: 4.191
[epoch: 2, i:   899] avg mini-batch loss: 4.203
[epoch: 2, i:   999] avg mini-batch loss: 4.156
[epoch: 3, i:    99] avg mini-batch loss: 4.101
[epoch: 3, i:   199] avg mini-batch loss: 4.070
[epoch: 3, i:   299] avg mini-batch loss: 4.051
[epoch: 3, i:   399] avg mini-batch loss: 4.054
[epoch: 3, i:   499] avg mini-batch loss: 4.031
[epoch: 3, i:   599] avg mini-batch loss: 3.980
[epoch: 3, i:   699] avg mini-batch loss: 3.990
[epoch: 3, i:   799] avg mini-batch loss: 3.957
[epoch: 3, i:   899] avg mini-batch loss: 3.946
[epoch: 3, i:   999] avg mini-batch loss: 3.929
[epoch: 4, i:    99] avg mini-batch loss: 3.865
[epoch: 4, i:   199] avg mini-batch loss: 3.834
[epoch: 4, i:   299] avg mini-batch loss: 3.870
[epoch: 4, i:   399] avg mini-batch loss: 3.810
[epoch: 4, i:   499] avg mini-batch loss: 3.774
[epoch: 4, i:   599] avg mini-batch loss: 3.780
[epoch: 4, i:   699] avg mini-batch loss: 3.781
[epoch: 4, i:   799] avg mini-batch loss: 3.758
[epoch: 4, i:   899] avg mini-batch loss: 3.774
[epoch: 4, i:   999] avg mini-batch loss: 3.739
[epoch: 5, i:    99] avg mini-batch loss: 3.660
[epoch: 5, i:   199] avg mini-batch loss: 3.680
[epoch: 5, i:   299] avg mini-batch loss: 3.651
[epoch: 5, i:   399] avg mini-batch loss: 3.652
[epoch: 5, i:   499] avg mini-batch loss: 3.626
[epoch: 5, i:   599] avg mini-batch loss: 3.666
[epoch: 5, i:   699] avg mini-batch loss: 3.613
[epoch: 5, i:   799] avg mini-batch loss: 3.594
[epoch: 5, i:   899] avg mini-batch loss: 3.609
[epoch: 5, i:   999] avg mini-batch loss: 3.609
[epoch: 6, i:    99] avg mini-batch loss: 3.542
[epoch: 6, i:   199] avg mini-batch loss: 3.544
[epoch: 6, i:   299] avg mini-batch loss: 3.515
[epoch: 6, i:   399] avg mini-batch loss: 3.520
[epoch: 6, i:   499] avg mini-batch loss: 3.510
[epoch: 6, i:   599] avg mini-batch loss: 3.511
[epoch: 6, i:   699] avg mini-batch loss: 3.534
[epoch: 6, i:   799] avg mini-batch loss: 3.494
[epoch: 6, i:   899] avg mini-batch loss: 3.449
[epoch: 6, i:   999] avg mini-batch loss: 3.487
[epoch: 7, i:    99] avg mini-batch loss: 3.397
[epoch: 7, i:   199] avg mini-batch loss: 3.424
[epoch: 7, i:   299] avg mini-batch loss: 3.415
```

```
[epoch: 7, i:   399] avg mini-batch loss: 3.374
[epoch: 7, i:   499] avg mini-batch loss: 3.398
[epoch: 7, i:   599] avg mini-batch loss: 3.444
[epoch: 7, i:   699] avg mini-batch loss: 3.411
[epoch: 7, i:   799] avg mini-batch loss: 3.375
[epoch: 7, i:   899] avg mini-batch loss: 3.380
[epoch: 7, i:   999] avg mini-batch loss: 3.327
[epoch: 8, i:    99] avg mini-batch loss: 3.306
[epoch: 8, i:   199] avg mini-batch loss: 3.285
[epoch: 8, i:   299] avg mini-batch loss: 3.304
[epoch: 8, i:   399] avg mini-batch loss: 3.321
[epoch: 8, i:   499] avg mini-batch loss: 3.280
[epoch: 8, i:   599] avg mini-batch loss: 3.324
[epoch: 8, i:   699] avg mini-batch loss: 3.288
[epoch: 8, i:   799] avg mini-batch loss: 3.284
[epoch: 8, i:   899] avg mini-batch loss: 3.272
[epoch: 8, i:   999] avg mini-batch loss: 3.336
[epoch: 9, i:    99] avg mini-batch loss: 3.206
[epoch: 9, i:   199] avg mini-batch loss: 3.225
[epoch: 9, i:   299] avg mini-batch loss: 3.196
[epoch: 9, i:   399] avg mini-batch loss: 3.208
[epoch: 9, i:   499] avg mini-batch loss: 3.253
[epoch: 9, i:   599] avg mini-batch loss: 3.223
[epoch: 9, i:   699] avg mini-batch loss: 3.196
[epoch: 9, i:   799] avg mini-batch loss: 3.189
[epoch: 9, i:   899] avg mini-batch loss: 3.194
[epoch: 9, i:   999] avg mini-batch loss: 3.200
[epoch: 10, i:    99] avg mini-batch loss: 3.120
[epoch: 10, i:   199] avg mini-batch loss: 3.114
[epoch: 10, i:   299] avg mini-batch loss: 3.161
[epoch: 10, i:   399] avg mini-batch loss: 3.146
[epoch: 10, i:   499] avg mini-batch loss: 3.150
[epoch: 10, i:   599] avg mini-batch loss: 3.113
[epoch: 10, i:   699] avg mini-batch loss: 3.132
[epoch: 10, i:   799] avg mini-batch loss: 3.149
[epoch: 10, i:   899] avg mini-batch loss: 3.105
[epoch: 10, i:   999] avg mini-batch loss: 3.106
Finished Training.
```

In [33]:
```
plt.plot(avg_losses)
plt.xlabel('mini-batch index / {}'.format(print_freq))
plt.ylabel('avg. mini-batch loss')
plt.show()
```



In [34]:
```
# Get test accuracy.
correct = 0
total = 0
with torch.no_grad():
    for i, data in enumerate(dataloaders['test']):
        net.eval()
        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: 15 %
```

In [ ]: