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
```

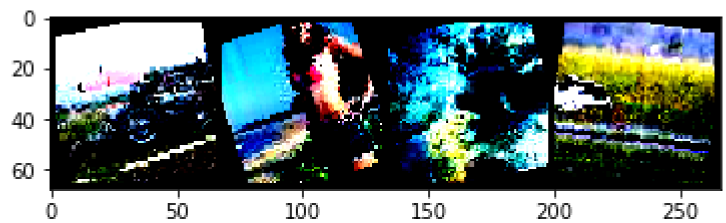
In [4]:
```
# If there are GPUs, choose the first one for computing. Otherwise use CPU.
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
# If 'cuda:0' is printed, it means GPU is available.
```

```
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_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 [35]:

```
net = alexnet.AlexNet()   # Create the network instance.
net.to(device)
```

Out[35]:

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fa
lse)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=Fa
lse)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=F
alse)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

In [36]:

```
# 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)
```

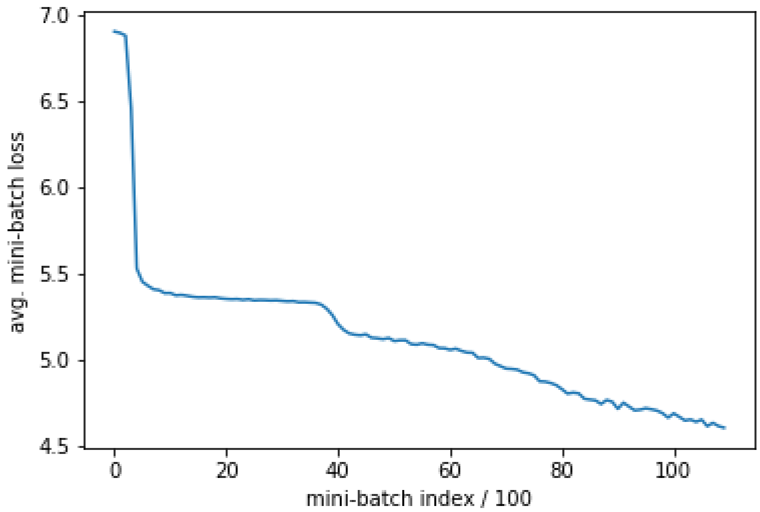
In [37]:
```python
avg_losses = []   # Avg. losses.
epochs = 11       # Total epochs.
print_freq = 100  # Print frequency.

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: 6.905
[epoch: 0, i:   199] avg mini-batch loss: 6.897
[epoch: 0, i:   299] avg mini-batch loss: 6.881
[epoch: 0, i:   399] avg mini-batch loss: 6.454
[epoch: 0, i:   499] avg mini-batch loss: 5.529
[epoch: 0, i:   599] avg mini-batch loss: 5.452
[epoch: 0, i:   699] avg mini-batch loss: 5.428
[epoch: 0, i:   799] avg mini-batch loss: 5.407
[epoch: 0, i:   899] avg mini-batch loss: 5.402
[epoch: 0, i:   999] avg mini-batch loss: 5.385
[epoch: 1, i:    99] avg mini-batch loss: 5.386
[epoch: 1, i:   199] avg mini-batch loss: 5.372
[epoch: 1, i:   299] avg mini-batch loss: 5.375
[epoch: 1, i:   399] avg mini-batch loss: 5.370
[epoch: 1, i:   499] avg mini-batch loss: 5.364
[epoch: 1, i:   599] avg mini-batch loss: 5.360
```

```
[epoch: 1, i:   699] avg mini-batch loss: 5.361
[epoch: 1, i:   799] avg mini-batch loss: 5.359
[epoch: 1, i:   899] avg mini-batch loss: 5.361
[epoch: 1, i:   999] avg mini-batch loss: 5.355
[epoch: 2, i:    99] avg mini-batch loss: 5.352
[epoch: 2, i:   199] avg mini-batch loss: 5.350
[epoch: 2, i:   299] avg mini-batch loss: 5.351
[epoch: 2, i:   399] avg mini-batch loss: 5.346
[epoch: 2, i:   499] avg mini-batch loss: 5.350
[epoch: 2, i:   599] avg mini-batch loss: 5.343
[epoch: 2, i:   699] avg mini-batch loss: 5.345
[epoch: 2, i:   799] avg mini-batch loss: 5.344
[epoch: 2, i:   899] avg mini-batch loss: 5.342
[epoch: 2, i:   999] avg mini-batch loss: 5.343
[epoch: 3, i:    99] avg mini-batch loss: 5.340
[epoch: 3, i:   199] avg mini-batch loss: 5.337
[epoch: 3, i:   299] avg mini-batch loss: 5.339
[epoch: 3, i:   399] avg mini-batch loss: 5.332
[epoch: 3, i:   499] avg mini-batch loss: 5.333
[epoch: 3, i:   599] avg mini-batch loss: 5.330
[epoch: 3, i:   699] avg mini-batch loss: 5.329
[epoch: 3, i:   799] avg mini-batch loss: 5.319
[epoch: 3, i:   899] avg mini-batch loss: 5.294
[epoch: 3, i:   999] avg mini-batch loss: 5.258
[epoch: 4, i:    99] avg mini-batch loss: 5.205
[epoch: 4, i:   199] avg mini-batch loss: 5.172
[epoch: 4, i:   299] avg mini-batch loss: 5.151
[epoch: 4, i:   399] avg mini-batch loss: 5.143
[epoch: 4, i:   499] avg mini-batch loss: 5.140
[epoch: 4, i:   599] avg mini-batch loss: 5.146
[epoch: 4, i:   699] avg mini-batch loss: 5.126
[epoch: 4, i:   799] avg mini-batch loss: 5.123
[epoch: 4, i:   899] avg mini-batch loss: 5.117
[epoch: 4, i:   999] avg mini-batch loss: 5.124
[epoch: 5, i:    99] avg mini-batch loss: 5.106
[epoch: 5, i:   199] avg mini-batch loss: 5.111
[epoch: 5, i:   299] avg mini-batch loss: 5.111
[epoch: 5, i:   399] avg mini-batch loss: 5.089
[epoch: 5, i:   499] avg mini-batch loss: 5.085
[epoch: 5, i:   599] avg mini-batch loss: 5.093
[epoch: 5, i:   699] avg mini-batch loss: 5.085
[epoch: 5, i:   799] avg mini-batch loss: 5.083
[epoch: 5, i:   899] avg mini-batch loss: 5.065
[epoch: 5, i:   999] avg mini-batch loss: 5.064
[epoch: 6, i:    99] avg mini-batch loss: 5.055
[epoch: 6, i:   199] avg mini-batch loss: 5.062
[epoch: 6, i:   299] avg mini-batch loss: 5.049
[epoch: 6, i:   399] avg mini-batch loss: 5.040
[epoch: 6, i:   499] avg mini-batch loss: 5.038
[epoch: 6, i:   599] avg mini-batch loss: 5.008
[epoch: 6, i:   699] avg mini-batch loss: 5.009
[epoch: 6, i:   799] avg mini-batch loss: 5.002
[epoch: 6, i:   899] avg mini-batch loss: 4.974
[epoch: 6, i:   999] avg mini-batch loss: 4.960
[epoch: 7, i:    99] avg mini-batch loss: 4.946
[epoch: 7, i:   199] avg mini-batch loss: 4.944
[epoch: 7, i:   299] avg mini-batch loss: 4.940
```

```
[epoch: 7, i:   399] avg mini-batch loss: 4.925
[epoch: 7, i:   499] avg mini-batch loss: 4.919
[epoch: 7, i:   599] avg mini-batch loss: 4.907
[epoch: 7, i:   699] avg mini-batch loss: 4.872
[epoch: 7, i:   799] avg mini-batch loss: 4.870
[epoch: 7, i:   899] avg mini-batch loss: 4.862
[epoch: 7, i:   999] avg mini-batch loss: 4.849
[epoch: 8, i:    99] avg mini-batch loss: 4.826
[epoch: 8, i:   199] avg mini-batch loss: 4.800
[epoch: 8, i:   299] avg mini-batch loss: 4.806
[epoch: 8, i:   399] avg mini-batch loss: 4.801
[epoch: 8, i:   499] avg mini-batch loss: 4.770
[epoch: 8, i:   599] avg mini-batch loss: 4.765
[epoch: 8, i:   699] avg mini-batch loss: 4.761
[epoch: 8, i:   799] avg mini-batch loss: 4.740
[epoch: 8, i:   899] avg mini-batch loss: 4.763
[epoch: 8, i:   999] avg mini-batch loss: 4.753
[epoch: 9, i:    99] avg mini-batch loss: 4.712
[epoch: 9, i:   199] avg mini-batch loss: 4.747
[epoch: 9, i:   299] avg mini-batch loss: 4.725
[epoch: 9, i:   399] avg mini-batch loss: 4.703
[epoch: 9, i:   499] avg mini-batch loss: 4.707
[epoch: 9, i:   599] avg mini-batch loss: 4.715
[epoch: 9, i:   699] avg mini-batch loss: 4.709
[epoch: 9, i:   799] avg mini-batch loss: 4.702
[epoch: 9, i:   899] avg mini-batch loss: 4.686
[epoch: 9, i:   999] avg mini-batch loss: 4.660
[epoch: 10, i:    99] avg mini-batch loss: 4.685
[epoch: 10, i:   199] avg mini-batch loss: 4.665
[epoch: 10, i:   299] avg mini-batch loss: 4.644
[epoch: 10, i:   399] avg mini-batch loss: 4.649
[epoch: 10, i:   499] avg mini-batch loss: 4.636
[epoch: 10, i:   599] avg mini-batch loss: 4.649
[epoch: 10, i:   699] avg mini-batch loss: 4.610
[epoch: 10, i:   799] avg mini-batch loss: 4.630
[epoch: 10, i:   899] avg mini-batch loss: 4.611
[epoch: 10, i:   999] avg mini-batch loss: 4.602
Finished Training.
```

In [38]:
```python
plt.plot(avg_losses)
plt.xlabel('mini-batch index / {}'.format(print_freq))
plt.ylabel('avg. mini-batch loss')
plt.show()
```



In [39]:
```python
# 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: 20 %
```

In [ ]:
```python

```