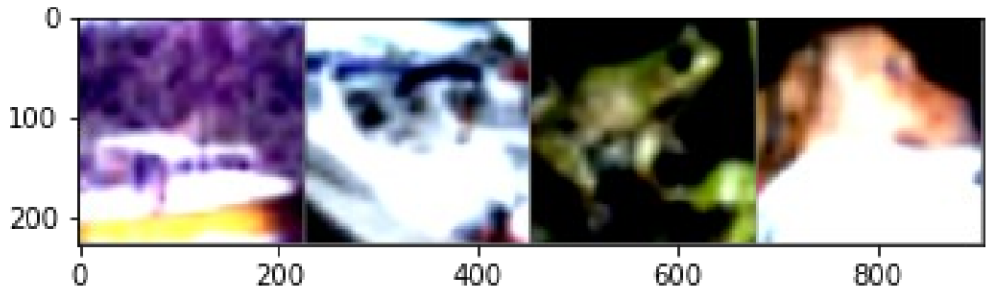
```
import torch
import torchvision
import torchvision.transforms as transforms

transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]),
])

train_data = torchvision.datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(train_data, batch_size=4,
shuffle=True, num_workers=2)

test_data = torchvision.datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform)
testloader = torch.utils.data.DataLoader(test_data, batch_size=4,
shuffle=False, num_workers=2)

classes = ('Airplane', 'Car', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog',
'Horse', 'Ship', 'Truck')

import matplotlib.pyplot as plt
import numpy as np

# functions 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 = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```




```
 Ship  Ship  Frog   Dog
```

```
# using the AlexNet
AlexNet_Model = torch.hub.load('pytorch/vision:v0.6.0', 'alexnet',
pretrained=True)
AlexNet_Model.eval()
```

```
import torch.nn as nn
AlexNet_Model.classifier[1] = nn.Linear(9216,4096)
AlexNet_Model.classifier[4] = nn.Linear(4096,1024)
AlexNet_Model.classifier[6] = nn.Linear(1024,10)

AlexNet_Model.eval()
```

```
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=False)
    (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=False)
    (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=False)
  )
  (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=1024, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=1024, out_features=10, bias=True)
  )
)
```

```
# move the input and model to GPU for speed if available
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")

print(device)
```

```
cuda:0
```

```
AlexNet_Model.to(device)
```

```
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=False)
    (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=False)
    (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=False)
  )
  (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=1024, bias=True)
    (5): ReLU(inplace=True)
```

```
    (6): Linear(in_features=1024, out_features=10, bias=True)
  )
)
```

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(AlexNet_Model.parameters(), lr=0.001,
momentum=0.9)
```

```
import time

for epoch in range(7):  # loop over the dataset multiple times

    running_loss = 0.0
    start_time = time.time()
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data[0].to(device), data[1].to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        output = AlexNet_Model(inputs)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()

        #Time
        end_time = time.time()
        time_taken = end_time - start_time

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1,
running_loss / 2000))
            print('Time:',time_taken)
            running_loss = 0.0

print('Finished Training of AlexNet')
```

```
[1,  2000] loss: 1.202
Time: 26.783158779144287
[1,  4000] loss: 0.892
Time: 52.17550587654114
[1,  6000] loss: 0.831
Time: 76.89599442481995
[1,  8000] loss: 0.741
```

```
Time: 102.43446254730225
[1, 10000] loss: 0.703
Time: 127.73311996459961
[1, 12000] loss: 0.652
Time: 153.21423721313477
[2,  2000] loss: 0.517
Time: 25.728044033050537
[2,  4000] loss: 0.518
Time: 51.18855333328247
[2,  6000] loss: 0.503
Time: 76.28134536743164
[2,  8000] loss: 0.503
Time: 102.24490404129028
[2, 10000] loss: 0.498
Time: 128.30848574638367
[2, 12000] loss: 0.479
Time: 154.10634303092957
[3,  2000] loss: 0.344
Time: 25.733392238616943
[3,  4000] loss: 0.345
Time: 51.15397524833679
[3,  6000] loss: 0.361
Time: 76.16496968269348
[3,  8000] loss: 0.353
Time: 101.2636661529541
[3, 10000] loss: 0.362
Time: 126.73455238342285
[3, 12000] loss: 0.367
Time: 152.39756155014038
[4,  2000] loss: 0.228
Time: 25.683215379714966
[4,  4000] loss: 0.247
Time: 50.77170968055725
[4,  6000] loss: 0.271
Time: 76.3136351108551
[4,  8000] loss: 0.266
Time: 101.3867175579071
[4, 10000] loss: 0.279
Time: 126.63472771644592
[4, 12000] loss: 0.286
Time: 152.09565353393555
[5,  2000] loss: 0.159
Time: 25.56282687187195
[5,  4000] loss: 0.189
Time: 51.04423999786377
[5,  6000] loss: 0.188
Time: 76.4984700679779
[5,  8000] loss: 0.216
Time: 101.08033537864685
[5, 10000] loss: 0.219
```

```
Time: 126.45636343955994
[5, 12000] loss: 0.234
Time: 151.6725754737854
[6,  2000] loss: 0.134
Time: 25.55684208869934
[6,  4000] loss: 0.154
Time: 51.152565240859985
[6,  6000] loss: 0.162
Time: 76.65935111045837
[6,  8000] loss: 0.166
Time: 101.64705610275269
[6, 10000] loss: 0.162
Time: 127.05616068840027
[6, 12000] loss: 0.171
Time: 152.73403024673462
[7,  2000] loss: 0.102
Time: 25.616260290145874
[7,  4000] loss: 0.132
Time: 51.10871887207031
[7,  6000] loss: 0.131
Time: 76.59881019592285
[7,  8000] loss: 0.131
Time: 101.46219182014465
[7, 10000] loss: 0.135
Time: 126.87981843948364
[7, 12000] loss: 0.121
Time: 152.46708822250366
Finished Training of AlexNet
```

```
#Testing Accuracy
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = AlexNet_Model(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: %.2f %%' %
(100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 83.02 %
```

```
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[0].to(device), data[1].to(device)
        outputs = AlexNet_Model(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]))
```

```
Accuracy of Airplane : 83 %
Accuracy of   Car : 93 %
Accuracy of  Bird : 82 %
Accuracy of   Cat : 70 %
Accuracy of  Deer : 82 %
Accuracy of   Dog : 78 %
Accuracy of  Frog : 88 %
Accuracy of Horse : 84 %
Accuracy of  Ship : 75 %
Accuracy of Truck : 90 %
```

```
#Verifying average accuracy of the network
avg = 0
for i in range(10):
  temp = (100 * class_correct[i] / class_total[i])
  avg = avg + temp
avg = avg/10
print('Average accuracy = ', avg)
```

```
Average accuracy =  83.02000000000001
```