4th Tutorial - Convolutional Neural Networks in PyTorch

In this tutorial, we want to you to get familiar with the implementation of Convolutional Neural Networks in PyTorch and let you investigate how the single neural network layers contribute to the final result. Sections which are marked as Assignments there you have to add something related to the tasks between the comments ### TODO Start ### and ### TODO End ###.

recommended literature for Convolutional Neural Networks:

- *Deep Learning (Ian Goodfellow and Yoshua Bengio and Aaron Courville)* (https://www.deeplearningbook.org/contents/convnets.html)

In [10]:
```
%matplotlib inline
```

In [2]:
```
#%pip install scipy --user
#%pip install matplotlib --user
#%pip install torch --user
#%pip install torchvision --user
#%pip install pandas --user
```

Training a Classifier

This is it. You have seen how to define neural networks, compute loss and make updates to the weights of the network.

Now you might be thinking,

What about data?

Generally, when you have to deal with image, text, audio or video data, you can use standard python packages that load data into a numpy array. Then you can convert this array into a `torch.*Tensor`.

- For images, packages such as Pillow, OpenCV are useful
- For audio, packages such as scipy and librosa
- For text, either raw Python or Cython based loading, or NLTK and SpaCy are useful

Specifically for vision, we have created a package called `torchvision`, that has data loaders for common datasets such as Imagenet, CIFAR10, MNIST, etc. and data transformers for images, viz., `torchvision.datasets` and `torch.utils.data.DataLoader`.

This provides a huge convenience and avoids writing boilerplate code.

For this tutorial, we will use the CIFAR10 dataset. It has the classes: ‘airplane’, ‘automobile’, ‘bird’, ‘cat’, ‘deer’, ‘dog’, ‘frog’, ‘horse’, ‘ship’, ‘truck’. The images in CIFAR-10 are of size

3x32x32, i.e. 3-channel color images of 32x32 pixels in size.

Training an image classifier

We will do the following steps in order:

1. Load and normalizing the CIFAR10 training and test datasets using `torchvision`
2. Define a Convolutional Neural Network
3. Define a loss function
4. Train the network on the training data
5. Test the network on the test data

In [2]:
```
import os
import torch
import torchvision
import torchvision.transforms as transforms
```

In [3]:
```
!pwd
```

```
/Users/ysl/Desktop/DLAI24:25/Exercises01-05/04
```

In [4]:
```
import ssl
# fix for https://github.com/pytorch/pytorch/issues/33288
ssl._create_default_https_context = ssl._create_unverified_context
```

1. Loading and normalizing CIFAR10

Using `torchvision`, it's extremely easy to load CIFAR10.

In [5]:
```
import os
import scipy
import numpy as np
import torchvision
```

The output of torchvision datasets are numpy images of range [0, 255].

In [35]:
```
dataset_root="./cifar_10/"
cifar10_data = torchvision.datasets.CIFAR10(root=f"{dataset_root}", download=True)
classes_uq = list(np.unique(cifar10_data.targets))
num_classes = len(classes_uq)
print(f"{num_classes} classes: {classes_uq}")
cifar10_data.data[0][0][:10]
```

```
Files already downloaded and verified
10 classes: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Out[35]:
```
array([[ 59,  62,  63],
       [ 43,  46,  45],
       [ 50,  48,  43],
       [ 68,  54,  42],
       [ 98,  73,  52],
       [119,  91,  63],
       [139, 107,  75],
       [145, 110,  80],
       [149, 117,  89],
       [149, 120,  93]], dtype=uint8)
```

We transform them to Tensors of normalized range [-1, 1].

> **Note**
>
> If running on Windows and you get a BrokenPipeError, try setting the num_worker of torch.utils.data.DataLoader() to 0.

In [ ]:
```
#num_workers = 0
num_workers = os.cpu_count()
bs=32

print(f"{num_workers} workers set for data_loader.")

transform = transforms.Compose(
    [
    transforms.ToTensor()  # ToTensor will scale PILImage pixels with values from [0
    ,transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) # Normalize will first s
    ])

trainset = torchvision.datasets.CIFAR10(root=f"{dataset_root}", train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=bs, shuffle=True, nu

testset = torchvision.datasets.CIFAR10(root=f"{dataset_root}", train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=bs,
                                         shuffle=False, num_workers=num_workers)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
print(trainset[0])
print(trainset[0][0].size(), trainset[0][0].min(), trainset[0][0].max())
```

```
10 workers set for data_loader.
Files already downloaded and verified
Files already downloaded and verified
(tensor([[[-0.5373, -0.6627, -0.6078,  ...,  0.2392,  0.1922,  0.1608],
         [-0.8745, -1.0000, -0.8588,  ..., -0.0353, -0.0667, -0.0431],
         [-0.8039, -0.8745, -0.6157,  ..., -0.0745, -0.0588, -0.1451],
         ...,
         [ 0.6314,  0.5765,  0.5529,  ...,  0.2549, -0.5608, -0.5843],
         [ 0.4118,  0.3569,  0.4588,  ...,  0.4431, -0.2392, -0.3490],
         [ 0.3882,  0.3176,  0.4039,  ...,  0.6941,  0.1843, -0.0353]],

        [[-0.5137, -0.6392, -0.6235,  ...,  0.0353, -0.0196, -0.0275],
         [-0.8431, -1.0000, -0.9373,  ..., -0.3098, -0.3490, -0.3176],
         [-0.8118, -0.9451, -0.7882,  ..., -0.3412, -0.3412, -0.4275],
         ...,
         [ 0.3333,  0.2000,  0.2627,  ...,  0.0431, -0.7569, -0.7333],
         [ 0.0902, -0.0353,  0.1294,  ...,  0.1608, -0.5137, -0.5843],
         [ 0.1294,  0.0118,  0.1137,  ...,  0.4431, -0.0745, -0.2784]],

        [[-0.5059, -0.6471, -0.6627,  ..., -0.1529, -0.2000, -0.1922],
         [-0.8431, -1.0000, -1.0000,  ..., -0.5686, -0.6078, -0.5529],
         [-0.8353, -1.0000, -0.9373,  ..., -0.6078, -0.6078, -0.6706],
         ...,
         [-0.2471, -0.7333, -0.7961,  ..., -0.4510, -0.9451, -0.8431],
         [-0.2471, -0.6706, -0.7647,  ..., -0.2627, -0.7333, -0.7333],
         [-0.0902, -0.2627, -0.3176,  ...,  0.0980, -0.3412, -0.4353]]]), 6)
torch.Size([3, 32, 32]) tensor(-1.) tensor(1.)
```

In [40]:
```
print(len(trainloader))
print(len(testloader))
```

```
1563
313
```

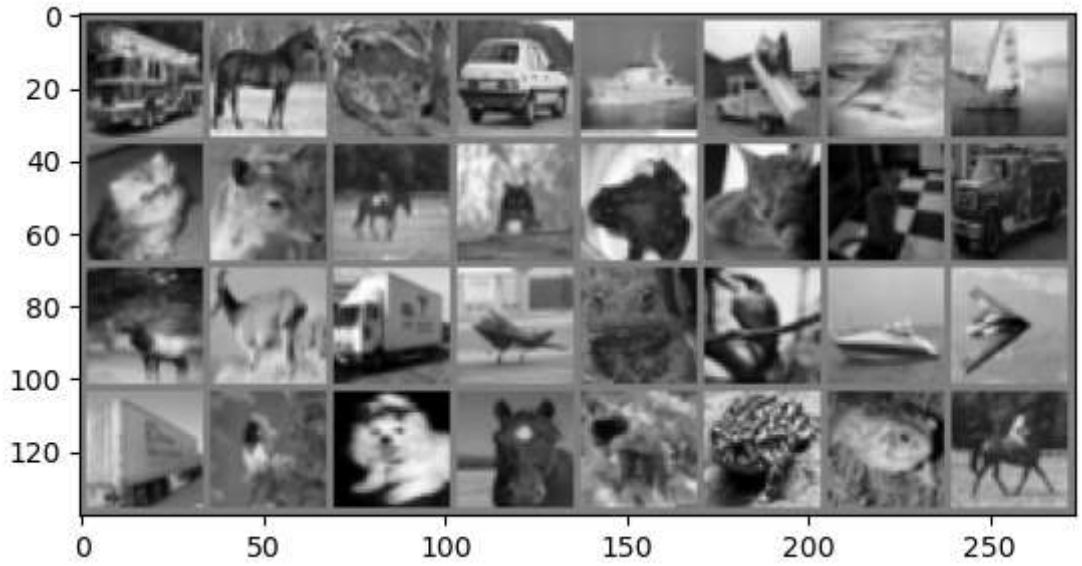
In [41]:
```
print(f"dataset.shape: {trainset.data.shape}")
print(f"dataset.targets")
for tg in np.unique(trainset.targets):
    print(f"{tg} -> {classes[tg]}")
```

```
dataset.shape: (50000, 32, 32, 3)
dataset.targets
0 -> plane
1 -> car
2 -> bird
3 -> cat
4 -> deer
5 -> dog
6 -> frog
7 -> horse
8 -> ship
9 -> truck
```

Let us show some of the training images, for fun.

In [42]:
```
import matplotlib.pyplot as plt
import numpy as np
import torchvision

# 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 = next(dataiter)  # Change from dataiter.next() to next(dataiter)

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



truck horse frog car

2. Define a Convolutional Neural Network

Create a CNN with the following architecture:

conv2d, in=3, out=6, kernel=5
ReLU
maxpool2d, kernel=2

conv2d, in=6, out=16, kernel=5
ReLU
maxpool2d, kernel=2

flatten

linear, in=16x5x5, out=120
ReLU
linear, in=120, out=84
ReLu
linear, in=84, out=number_of_possible_CIFAR10_classes

In [43]:
```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

3. Define a Loss function and optimizer

Let's use a Classification Cross-Entropy loss and SGD with momentum 0.9 and a learning rate of 0.001.

In [44]:
```
import torch.optim as optim

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

4. Train the network

This is when things start to get interesting. We simply have to loop over our data iterator, and feed the inputs to the network and optimize.

In [45]:
```
for epoch in range(2):  # loop over the dataset multiple times

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

        # zero the parameter gradients
        optimizer.zero_grad()  # IMPORTANT: gradients accumulate by default in PyTor

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()  # compute gradients with respect to each parameter (hidden
        optimizer.step()  # update parameters
```

```
        # print statistics
        running_loss += loss.item()
        # if i % 2000 == 1999:    # print every 2000 mini-batches
        if i % 200 == 199:
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 200))
            running_loss = 0.0

print('Finished Training')
```

```
[1,   200] loss: 2.304
[1,   400] loss: 2.301
[1,   600] loss: 2.298
[1,   800] loss: 2.291
[1,  1000] loss: 2.270
[1,  1200] loss: 2.194
[1,  1400] loss: 2.117
[2,   200] loss: 2.023
[2,   400] loss: 1.972
[2,   600] loss: 1.927
[2,   800] loss: 1.901
[2,  1000] loss: 1.856
[2,  1200] loss: 1.780
[2,  1400] loss: 1.730
Finished Training
```

Let's quickly save our trained model:

In [46]:
```
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
!dir | grep pth
```

```
zsh:1: command not found: dir
```

See `here <https://pytorch.org/docs/stable/notes/serialization.html>` _ for more details on saving PyTorch models.

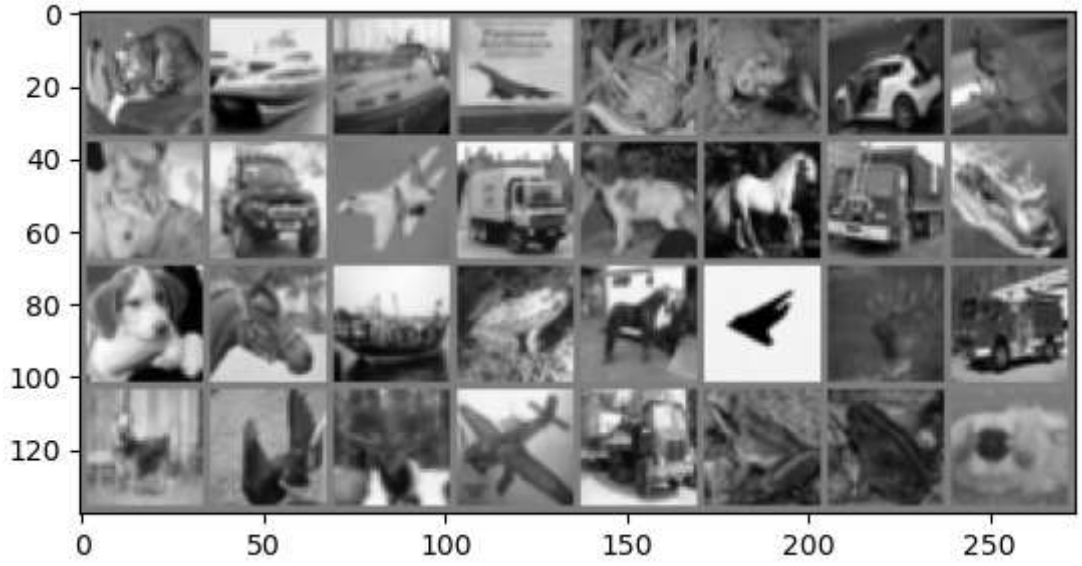
5. Test the network on the test data

We have trained the network for 2 passes over the training dataset. But we need to check if the network has learnt anything at all.

We will check this by predicting the class label that the neural network outputs, and checking it against the ground-truth. If the prediction is correct, we add the sample to the list of correct predictions.

Okay, first step. Let us display an image from the test set to get familiar.

In [47]:
```
import torchvision

dataiter = iter(testloader)
images, labels = next(dataiter)

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




```
GroundTruth:    cat  ship  ship plane
```

Next, let's load back in our saved model (note: saving and re-loading the model wasn't necessary here, we only did it to illustrate how to do so):

In [48]:
```
net = Net()
net.load_state_dict(torch.load(PATH))
```

Out[48]:
```
<All keys matched successfully>
```

Okay, now let us see what the neural network thinks these examples above are:

In [49]:
```
outputs = net(images)
print(f"outputs.shape: {outputs.shape}")

num_samples=4
outputs[:num_samples]
```

Out[49]:
```
outputs.shape: torch.Size([32, 10])
tensor([[-0.6824,  0.2023,  0.3139,  0.5896,  0.1053, -0.2206,  0.5234, -1.1445,
         -0.6728, -0.6204],
        [ 2.3391,  3.0623, -1.2953, -2.3532, -1.6795, -3.6094, -4.0431, -1.7841,
          3.6844,  2.6879],
        [ 1.4079,  2.2155, -0.9111, -1.4790, -1.3265, -2.3144, -2.8224, -0.8142,
          2.1795,  2.0772],
        [ 2.7911,  0.7291,  0.5248, -1.6887, -0.1513, -2.5693, -3.3297, -0.9880,
          2.4694,  0.1966]], grad_fn=<SliceBackward0>)
```

The outputs are energies for the 10 classes of 4 images. Since energies give no sense about the confidence for a class, lets transform the energies into proabilites by applying the Softmax function:

In [50]:
```
def get_outputs_proba(ooutputs):
    softmax = nn.Softmax(dim=1)
    outputs_proba = softmax(ooutputs)
    return outputs_proba

outputs_proba = get_outputs_proba(outputs[:num_samples])
outputs_proba
```

Out[50]:
```
tensor([[5.1211e-02, 1.2405e-01, 1.3870e-01, 1.8272e-01, 1.1258e-01, 8.1270e-02,
         1.7102e-01, 3.2261e-02, 5.1703e-02, 5.4489e-02],
        [1.1916e-01, 2.4560e-01, 3.1460e-03, 1.0923e-03, 2.1423e-03, 3.1099e-04,
         2.0155e-04, 1.9296e-03, 4.5751e-01, 1.6890e-01],
        [1.2945e-01, 2.9030e-01, 1.2734e-02, 7.2170e-03, 8.4060e-03, 3.1301e-03,
         1.8833e-03, 1.4031e-02, 2.8003e-01, 2.5281e-01],
        [4.7075e-01, 5.9879e-02, 4.8814e-02, 5.3361e-03, 2.4826e-02, 2.2120e-03,
         1.0340e-03, 1.0754e-02, 3.4124e-01, 3.5157e-02]],
       grad_fn=<SoftmaxBackward0>)
```

The higher the confidence (probability) for a class, the more the network thinks that the image is of the particular class.
So, let's sort the outputs by confidence:

In [51]:
```
def get_outputs_proba_sorted(ooutputs_proba_):
    return torch.sort(ooutputs_proba_.detach(), dim=1, descending=True)

outputs_proba_sorted = get_outputs_proba_sorted(outputs_proba[:num_samples])
outputs_proba_sorted
```

Out[51]:
```
torch.return_types.sort(
values=tensor([[1.8272e-01, 1.7102e-01, 1.3870e-01, 1.2405e-01, 1.1258e-01, 8.1270e-02,
         5.4489e-02, 5.1703e-02, 5.1211e-02, 3.2261e-02],
        [4.5751e-01, 2.4560e-01, 1.6890e-01, 1.1916e-01, 3.1460e-03, 2.1423e-03,
         1.9296e-03, 1.0923e-03, 3.1099e-04, 2.0155e-04],
        [2.9030e-01, 2.8003e-01, 2.5281e-01, 1.2945e-01, 1.4031e-02, 1.2734e-02,
         8.4060e-03, 7.2170e-03, 3.1301e-03, 1.8833e-03],
        [4.7075e-01, 3.4124e-01, 5.9879e-02, 4.8814e-02, 3.5157e-02, 2.4826e-02,
         1.0754e-02, 5.3361e-03, 2.2120e-03, 1.0340e-03]]),
indices=tensor([[3, 6, 2, 1, 4, 5, 9, 8, 0, 7],
        [8, 1, 9, 0, 2, 4, 7, 3, 5, 6],
        [1, 8, 9, 0, 7, 2, 4, 3, 5, 6],
        [0, 8, 1, 2, 9, 4, 7, 3, 5, 6]]))
```

In [ ]:
```
```

Store the confidence per image in a list together with its predicted class label and ground truth class label!

In [52]:
```
def get_predictions(outputs_, labels_, classes, highest_only=False):
    preds=[]

    outputs_proba_ = get_outputs_proba(outputs_)
    outputs_proba_sorted_ = get_outputs_proba_sorted(outputs_proba_)

    for sample_idx in range(0, len(outputs_proba_sorted_.values)):
        pred = []
        for idx in range(0, len(outputs_proba_sorted_.indices[sample_idx, :])):
            pred_val = outputs_proba_sorted_.values[sample_idx, idx]
            pred_cls = outputs_proba_sorted_.indices[sample_idx, idx]
            gt_class = classes[labels_[sample_idx]]
            pred_class = classes[pred_cls]
            pred_conf = round(pred_val.item(),5)
            pred.append(f"image{sample_idx} - gt:{gt_class}|pred:{pred_class} ({pred
            if highest_only: break
        preds.append(pred)
    return preds

num_samples=4
get_predictions(outputs[:num_samples], labels[:num_samples], classes, highest_only=F
```

Out[52]:
```
[['image0 - gt:cat|pred:cat (0.18272)',
  'image0 - gt:cat|pred:frog (0.17102)',
  'image0 - gt:cat|pred:bird (0.1387)',
  'image0 - gt:cat|pred:car (0.12405)',
  'image0 - gt:cat|pred:deer (0.11258)',
  'image0 - gt:cat|pred:dog (0.08127)',
  'image0 - gt:cat|pred:truck (0.05449)',
  'image0 - gt:cat|pred:ship (0.0517)',
  'image0 - gt:cat|pred:plane (0.05121)',
  'image0 - gt:cat|pred:horse (0.03226)'],
 ['image1 - gt:ship|pred:ship (0.45751)',
  'image1 - gt:ship|pred:car (0.2456)',
  'image1 - gt:ship|pred:truck (0.1689)',
  'image1 - gt:ship|pred:plane (0.11916)',
  'image1 - gt:ship|pred:bird (0.00315)',
  'image1 - gt:ship|pred:deer (0.00214)',
  'image1 - gt:ship|pred:horse (0.00193)',
  'image1 - gt:ship|pred:cat (0.00109)',
  'image1 - gt:ship|pred:dog (0.00031)',
  'image1 - gt:ship|pred:frog (0.0002)'],
 ['image2 - gt:ship|pred:car (0.2903)',
  'image2 - gt:ship|pred:ship (0.28003)',
  'image2 - gt:ship|pred:truck (0.25281)',
  'image2 - gt:ship|pred:plane (0.12945)',
  'image2 - gt:ship|pred:horse (0.01403)',
  'image2 - gt:ship|pred:bird (0.01273)',
  'image2 - gt:ship|pred:deer (0.00841)',
  'image2 - gt:ship|pred:cat (0.00722)',
  'image2 - gt:ship|pred:dog (0.00313)',
  'image2 - gt:ship|pred:frog (0.00188)'],
 ['image3 - gt:plane|pred:plane (0.47075)',
  'image3 - gt:plane|pred:ship (0.34124)',
  'image3 - gt:plane|pred:car (0.05988)',
  'image3 - gt:plane|pred:bird (0.04881)',
  'image3 - gt:plane|pred:truck (0.03516)',
  'image3 - gt:plane|pred:deer (0.02483)',
  'image3 - gt:plane|pred:horse (0.01075)',
  'image3 - gt:plane|pred:cat (0.00534)',
  'image3 - gt:plane|pred:dog (0.00221)',
  'image3 - gt:plane|pred:frog (0.00103)']]
```

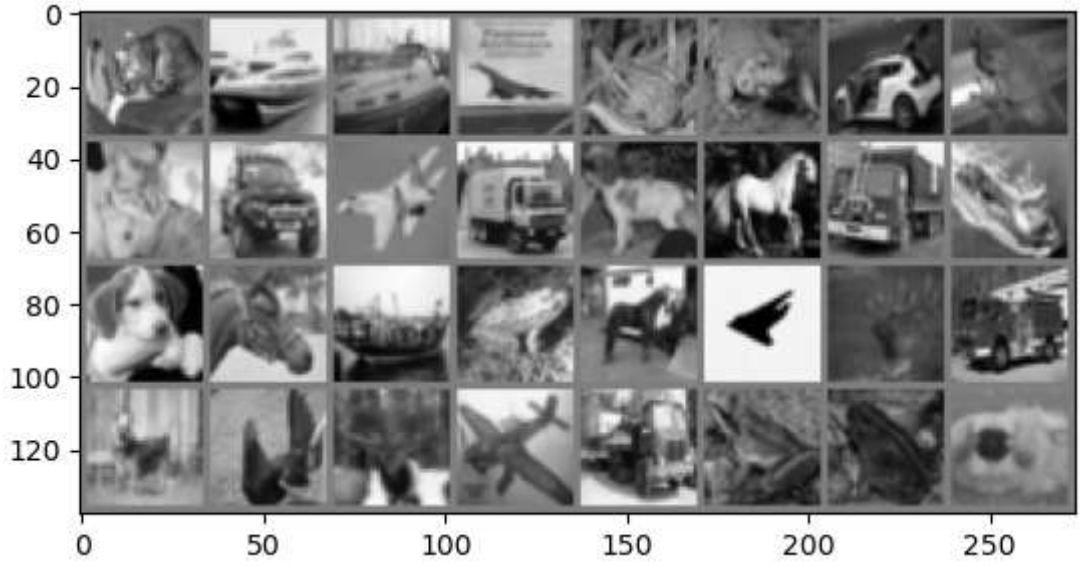
In [53]:
```
get_predictions(outputs[:num_samples], labels[:num_samples], classes, highest_only=T
```

Out[53]:
```
[['image0 - gt:cat|pred:cat (0.18272)'],
 ['image1 - gt:ship|pred:ship (0.45751)'],
 ['image2 - gt:ship|pred:car (0.2903)'],
 ['image3 - gt:plane|pred:plane (0.47075)']]
```

As one can observe, the CNN has even for the highest confidence not actually a high confidence.
Nevertheless, filter for each image the output with the highest confidence and plot the related predicted class label together with the ground truth label and image:

In [54]:
```
_, predicted = torch.max(outputs, 1)

print('  Predicted: ', ' '.join('%5s' % classes[predicted[j]] for j in range(num_sa
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(num_sampl
imshow(torchvision.utils.make_grid(images))
```

```
  Predicted:    cat  ship   car plane
GroundTruth:    cat  ship  ship plane
```



Let us look at how the network performs on the whole dataset by creating a classification report with sklearn.

In [55]:
```
from sklearn.metrics import classification_report

def evaluate_performance(test_model, testloader, classes):
    correct = 0
    total = 0
    predictions=[]
    groundtruth=[]
    device = 'cpu'
    test_model = test_model.to(device)
    test_model.eval()
    with torch.no_grad():
        for data in testloader:
            x_batch, y_batch = data[0].to(device), data[1].to(device)
            outputs = test_model(x_batch.to(device))
            _, predicted = torch.max(outputs.data, 1)
            predictions.extend(predicted.cpu())
            groundtruth.extend(y_batch.cpu())
    print(classification_report(y_true=groundtruth, y_pred=predictions, target_names

evaluate_performance(net, testloader, classes)
```

	precision	recall	f1-score	support
plane	0.44	0.54	0.49	1000
car	0.51	0.50	0.50	1000
bird	0.27	0.13	0.17	1000
cat	0.31	0.18	0.23	1000
deer	0.29	0.48	0.36	1000
dog	0.47	0.20	0.28	1000
frog	0.42	0.49	0.45	1000
horse	0.33	0.61	0.43	1000
ship	0.47	0.28	0.35	1000
truck	0.44	0.46	0.45	1000
accuracy			0.39	10000
macro avg	0.40	0.39	0.37	10000
weighted avg	0.40	0.39	0.37	10000

Accuracy of the network on the 10000 test images: ~40 % That looks better than chance, which is 10% accuracy (randomly picking a class out of 10 classes). Seems like the network

learned something.

Ok, lets try to improve the performance by using ResNet18. Since it has much more weights to train compared to our simple CNN we need to speed up the training process by utilizing a GPU. But how do we run train neural networks on the GPU?

Training on GPU

Just like how you transfer a Tensor onto the GPU, you transfer the neural net onto the GPU.

Let's first define our device as the first visible cuda device if we have CUDA available:

In [25]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Assuming that we are on a CUDA machine, this should print a CUDA device:

print(device)
```

```
cuda:0
```

Did you receive a response with device "cuda" or "cuda:0"?

If yes, then the rest of this section assumes that `device` is a CUDA device.

Then these methods will recursively go over all modules and convert their parameters and buffers to CUDA tensors:

.. code:: python

```
    model.to(device)
```

Remember that you will have to send the inputs and targets at every step to the GPU too:

.. code:: python

```
        x_batch, y_batch = data[0].to(device), data[1].to(device)
```

In [26]:
```
print(f"{num_classes} classes:")
print(classes_uq)
```

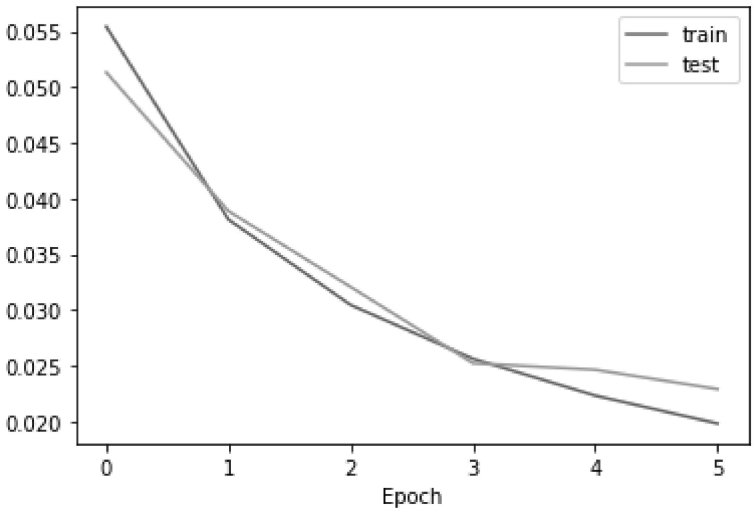
```
10 classes:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [27]:
```
import torchvision.models as models
import pandas as pd

model = models.resnet18(pretrained=True)

# since resnet18 ends up with a final linear layer (model.fc) of size 1000 we need t
model.fc = nn.Linear(512, num_classes)

# assign model to device (GPU or CPU)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.Adam(model.parameters(), lr=0.003)

epochs = 6

# train
trn_losses=[]
tst_losses=[]
for epoch in range(epochs):
    model.train()
    bidx=0
    batch_losses=[]
    for data in trainloader:
        bidx+=1

        optimizer.zero_grad()
        x_batch, y_batch = data[0].to(device), data[1].to(device)

        y_batch_preds = model(x_batch)
        loss = criterion(input=y_batch_preds, target=y_batch)
        loss.backward()
        batch_loss = loss.item()/x_batch.shape[0]
        batch_losses.append(batch_loss)

        optimizer.step()

        if bidx % 1000 == 0:
            print('Epoch {}/{} b{} - train'.format(epoch + 1, epochs, bidx))

    trn_losses.append(np.array(batch_losses).mean())

    model.eval()
    with torch.no_grad():
        bidx=0
        batch_losses=[]
        for data in testloader:
            bidx+=1
            x_batch, y_batch = data[0].to(device), data[1].to(device)

            y_batch_preds = model(x_batch)
            loss = criterion(input=y_batch_preds, target=y_batch)

            batch_loss = loss.item()/x_batch.shape[0]
            batch_losses.append(batch_loss)

            if bidx % 1000 == 0:
                print('Epoch {}/{} b{} - test'.format(epoch + 1, epochs, bidx))

        tst_losses.append(np.array(batch_losses).mean())

    print('Epoch {}/{} trn/tst: {}/{}'.format(epoch + 1, epochs, trn_losses[-1], tst

df=pd.DataFrame({'Epoch': [e for e in range(epoch + 1)], 'train': trn_losses, 'test'
df.plot(x=0, xticks=df.index.tolist())
```

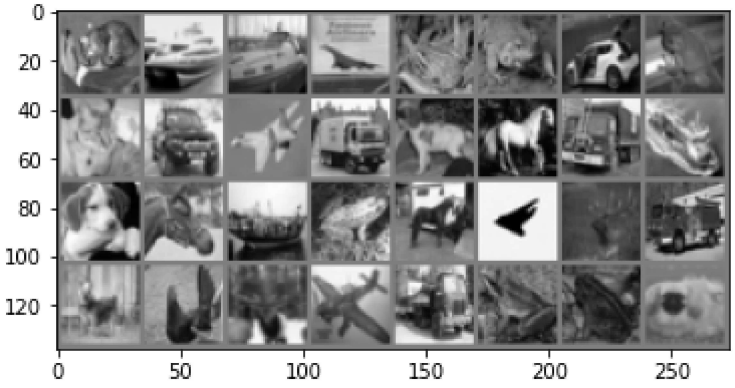
```
Epoch 1/6 b1000 - train
Epoch 1/6 trn/tst: 0.05540723172722531/0.05130889023335788
Epoch 2/6 b1000 - train
Epoch 2/6 trn/tst: 0.03811311976113002/0.03886772583301265
Epoch 3/6 b1000 - train
Epoch 3/6 trn/tst: 0.03045710134340816/0.032075629316675014
Epoch 4/6 b1000 - train
Epoch 4/6 trn/tst: 0.02564047567930098 6/0.025236212154523062
Epoch 5/6 b1000 - train
Epoch 5/6 trn/tst: 0.022336867551533215/0.024652200920608477
Epoch 6/6 b1000 - train
Epoch 6/6 trn/tst: 0.01983594451747925/0.022921628077713828
```

Out[27]: <AxesSubplot:xlabel='Epoch'>



In [30]:
```
outputs = model(images.to(device))
_, predicted = torch.max(outputs, 1)

print('  Predicted: ', ' '.join('%5s' % classes[predicted[j]] for j in range(4)))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
imshow(torchvision.utils.make_grid(images))
```

```
  Predicted:    cat  ship  ship plane
GroundTruth:    cat  ship  ship plane
```



In [31]:
```
num_samples=4
get_predictions(outputs[:num_samples], labels[:num_samples], classes, highest_only=F
```

Out[31]:
```
[['image0 - gt:cat|pred:cat (0.80886)',
  'image0 - gt:cat|pred:dog (0.13547)',
  'image0 - gt:cat|pred:frog (0.02346)',
  'image0 - gt:cat|pred:bird (0.01122)',
  'image0 - gt:cat|pred:ship (0.01102)',
  'image0 - gt:cat|pred:deer (0.00437)',
  'image0 - gt:cat|pred:plane (0.00275)',
  'image0 - gt:cat|pred:horse (0.00175)',
  'image0 - gt:cat|pred:car (0.00075)',
  'image0 - gt:cat|pred:truck (0.00035)'],
 ['image1 - gt:ship|pred:ship (0.9168)',
  'image1 - gt:ship|pred:car (0.05652)',
  'image1 - gt:ship|pred:plane (0.01489)',
  'image1 - gt:ship|pred:truck (0.01084)',
  'image1 - gt:ship|pred:deer (0.00043)',
  'image1 - gt:ship|pred:cat (0.00026)',
  'image1 - gt:ship|pred:bird (0.00018)',
  'image1 - gt:ship|pred:horse (5e-05)',
  'image1 - gt:ship|pred:dog (2e-05)',
  'image1 - gt:ship|pred:frog (1e-05)'],
 ['image2 - gt:ship|pred:ship (0.45822)',
  'image2 - gt:ship|pred:plane (0.42554)',
  'image2 - gt:ship|pred:truck (0.02701)',
  'image2 - gt:ship|pred:bird (0.02695)',
  'image2 - gt:ship|pred:cat (0.01947)',
  'image2 - gt:ship|pred:deer (0.01656)',
  'image2 - gt:ship|pred:car (0.01318)',
  'image2 - gt:ship|pred:horse (0.00894)',
  'image2 - gt:ship|pred:dog (0.00349)',
  'image2 - gt:ship|pred:frog (0.00064)'],
 ['image3 - gt:plane|pred:plane (0.7542)',
  'image3 - gt:plane|pred:ship (0.13881)',
  'image3 - gt:plane|pred:truck (0.05599)',
  'image3 - gt:plane|pred:deer (0.01832)',
  'image3 - gt:plane|pred:bird (0.0133)',
  'image3 - gt:plane|pred:horse (0.00685)',
  'image3 - gt:plane|pred:cat (0.00605)',
  'image3 - gt:plane|pred:car (0.00471)',
  'image3 - gt:plane|pred:dog (0.00137)',
  'image3 - gt:plane|pred:frog (0.00039)']]
```

In [32]:
```
get_predictions(outputs[:num_samples], labels[:num_samples], classes, highest_only=T
```

Out[32]:
```
[['image0 - gt:cat|pred:cat (0.80886)'],
 ['image1 - gt:ship|pred:ship (0.9168)'],
 ['image2 - gt:ship|pred:ship (0.45822)'],
 ['image3 - gt:plane|pred:plane (0.7542)']]
```

Now we observe that the confidences for the highest class are much more certain compared to those of the first small model.

Lets also evaluate the overall performance:

In [33]:
```
evaluate_performance(model, testloader, classes)
```

	precision	recall	f1-score	support
plane	0.73	0.75	0.74	1000
car	0.89	0.85	0.87	1000
bird	0.63	0.69	0.66	1000
cat	0.61	0.55	0.58	1000
deer	0.71	0.74	0.73	1000
dog	0.72	0.59	0.65	1000
frog	0.73	0.88	0.80	1000
horse	0.82	0.78	0.80	1000
ship	0.83	0.86	0.85	1000
truck	0.85	0.81	0.83	1000
accuracy			0.75	10000
macro avg	0.75	0.75	0.75	10000
weighted avg	0.75	0.75	0.75	10000

So as you see, using a pretrained resnet18 improves the network performance significantly but there is still space for improvement.

Why dont I notice MASSIVE speedup compared to CPU? Because your network is really small.

Exercise: Try increasing the width of your network (argument 2 of the first `nn.Conv2d`, and argument 1 of the second `nn.Conv2d` – they need to be the same number), see what kind of speedup you get.

Goals achieved:

- Understanding PyTorch's Tensor library and neural networks at a high level.
- Train a small neural network to classify images
- Train on CPU and GPU

In [ ]: