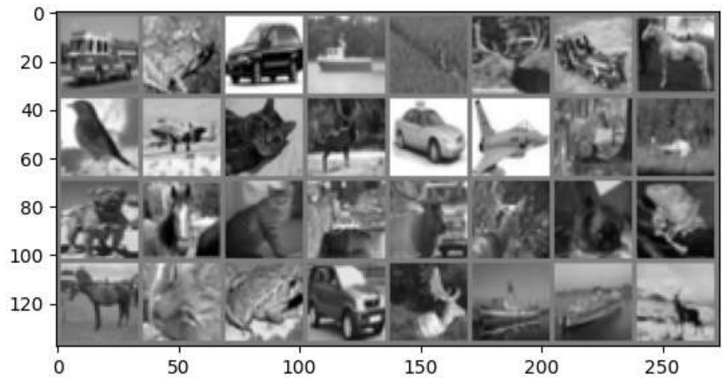


truck frog car ship

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

```
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.

```
import torch.optim as optim
```

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

双击（或按回车键）即可修改

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.

```
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 PyTorch

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()  # compute gradients with respect to each parameter (hidden attribute .grad)
        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:

```
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

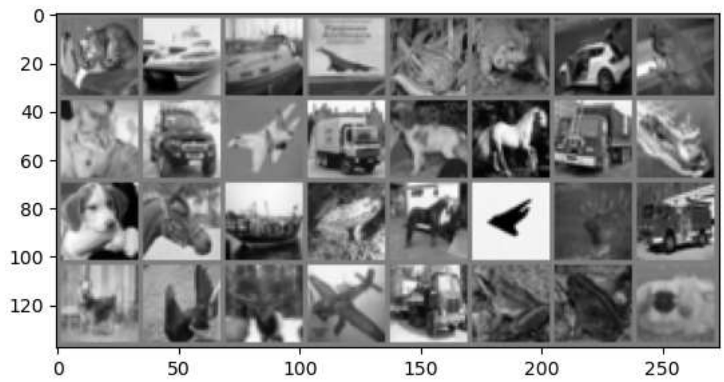
We have trained the network for 2 passes over the training dataset. But we need to check if the network has learnt anything at all.

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.

```
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):

```
net = Net()
net.load_state_dict(torch.load(PATH))
```

```
<All keys matched successfully>
```

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

```
outputs = net(images)
print(f"outputs.shape: {outputs.shape}")

num_samples=4
outputs[:num_samples]
```

```
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:

```
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
```

```
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:

```
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
```

```
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]]))
```

开始借助 AI 编写或生成代码。

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

```
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_conf})")
            if highest_only: break
        preds.append(pred)
    return preds

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

```
[['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)']]
```

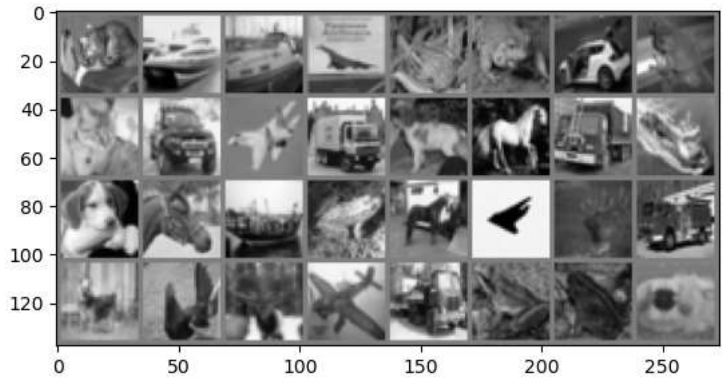
```
get_predictions(outputs[:num_samples], labels[:num_samples], classes, highest_only=True)
```

```
[['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:

```
_, predicted = torch.max(outputs, 1)

print('   Predicted: ', ' '.join('%5s' % classes[predicted[j]] for j in range(num_samples)))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(num_samples)))
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.

```
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=classes))

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:

```
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)
```

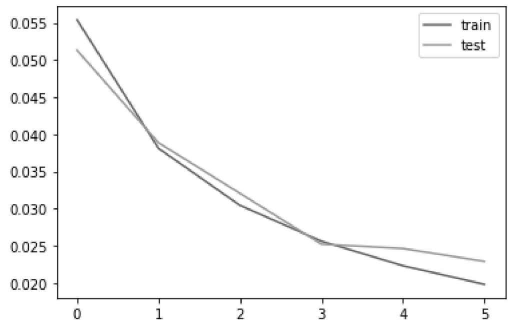
```
print(f"{num_classes} classes:")
print(classes_uq)
```

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

```
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 to adapt it to our number of total classes
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()
```

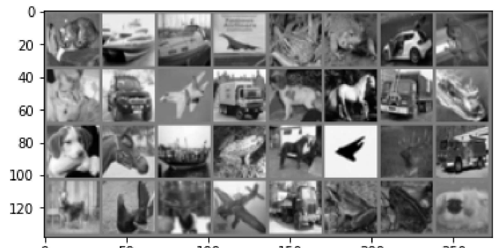
```
        if bidx % 1000 == 0:
            print('Epoch {}/{} b{} - train'.format(epoch + 1, epochs, bidx))

    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_losses[-1]))

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

```
Epoch 1/6 b1000 - train
Epoch 1/6 trn/tst: 0.05540723172722531/0.051308889023335788
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: 0.025640475679300986/0.025236212154523062
Epoch 5/6 b1000 - train
Epoch 5/6 trn/tst: 0.022336867551533215/0.024652200920608477
Epoch 6/6 b1000 - train
Epoch 6/6 trn/tst: 0.019835944517479 25/0.022921628077713828
<AxesSubplot:xlabel='Epoch'>
```




```
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
```




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

```
[['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)']]
```

```
get_predictions(outputs[:num_samples], labels[:num_samples], classes, highest_only=True)
```

```
[['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:

```
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

开始借助 AI 编写或生成代码。