ECE 47300 Assignment 5

Your Name: Owen Semeter

Prepare the pacakge we will use.

In [ ]:
```
import time
from typing import List, Dict

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.models as models
import torchvision.transforms as transforms

import matplotlib.pyplot as plt

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Exercise 1: Why use a CNN rather than only fully connected layers? (40 points)

In this exercise, you will build two models for the **MNIST** dataset: one uses only fully connected layers and another uses a standard CNN layout (convolution layers everywhere except the last layer is fully connected layer). Note, you will need to use cross entropy loss as your objective function. The two models should be built with roughly the same accuracy performance, your task is to compare the number of network parameters (a huge number of parameters can affect training/testing time, memory requirements, overfitting, etc.).

Task 1: Prepare train and test function

We will create our train and test procedure in these two functions. The train function should apply one epoch of training. The functions inputs should take everything we need for training and testing and return some logs.

Arguments requirement:

- For the `train` function, it takes the `model`, `loss_fn`, `optimizer`, `train_loader`, and `epoch` as arguments.
  - `model`: the classifier, or deep neural network, should be an instance of `nn.Module`.
  - `loss_fn`: the loss function instance. For example, `nn.CrossEntropy()`, or `nn.L1Loss()`, etc.
  - `optimizer`: should be an instance of `torch.optim.Optimizer`. For example, it could be `optim.SGD()` or `optim.Adam()`, etc.
  - `train_loader`: should be an instance of `torch.utils.data.DataLoader`.
  - `epoch`: the current number of epoch. Only used for log printing.(default: 1.)
- For the `test` function, it takes all the inputs above except for the optimizer (and it takes a test loader instead of a train loader).

Log requirement:

Here are some further requirements:

- In the `train` function, print the log 8-10 times per epoch. The print statement should be:

```
print(f'Epoch {epoch}: [{batch_idx*len(images)}/{len(train_loader.dataset)}] Loss: {loss.item():.3f}')
```

- In the `test` function, print the log after the testing. The print statement is:

```
print(f"Test result on epoch {epoch}: total sample: {total_num}, Avg loss: {test_stat['loss']:.3f}, Acc: {100*test_stat['accuracy']:.3f}%")
```

Return requirement

- The `train` function should return a list, which the element is the loss per batch, i.e., one loss value for every batch.
- The `test` function should return a dictionary with three keys: "loss", "accuracy", and "prediction". The values are the average loss of all the testset, average accuracy of all the test dataset, and the prediction of all test dataset.

Other requirement:

- In the `train` function, the model should be updated in-place, i.e., do not copy the model inside `train` function.

In [94]:
```
def train(model: nn.Module,
          loss_fn: nn.modules.loss._Loss,
```

```
          optimizer: torch.optim.Optimizer,
          train_loader: torch.utils.data.DataLoader,
          epoch: int=0)-> List:
    # ----------- <Your code> ---------------
    model = model.to(device)
    model.train()
    train_loss = list()

    for batch_idx, (images, target) in enumerate(train_loader):
      optimizer.zero_grad()
      images = images.to(device)
      target = target.to(device)
      outputs = model(images)
      loss = loss_fn(outputs, target)
      loss.backward()
      optimizer.step()

      train_loss.append(loss.item())

      if batch_idx % (len(train_loader) // 8) == 0:
        print(f"Epoch {epoch}: [{batch_idx*len(images)}/{len(train_loader.dataset)}


    # ----------- <End Your code> ---------------
    assert len(train_loss) == len(train_loader)
    return train_loss

def test(model: nn.Module,
         loss_fn: nn.modules.loss._Loss,
         test_loader: torch.utils.data.DataLoader,
         epoch: int=0)-> Dict:
    # ----------- <Your code> ---------------
    model = model.to(device)
    model.eval()
    total_num = len(test_loader.dataset)
    t_loss = 0
    num_correct = 0
    predictions = list()

    with torch.no_grad():
      for images, targets in test_loader:
        images = images.to(device)
        targets = targets.to(device)
        out = model(images)
        loss = loss_fn(out, targets)
        t_loss += loss.item()
        pred = out.data.max(1, keepdim=True)[1]
        num_correct += pred.eq(targets.data.view_as(pred)).sum()
        predictions.extend(pred.tolist())

    avg_loss = t_loss / len(test_loader.dataset)
    accuracy = num_correct / total_num

    test_stat = {"loss": avg_loss, "accuracy": accuracy, "prediction": torch.tensor
    print(f"Test result on epoch {epoch}: total sample: {total_num}, Avg loss:{test
```

```
    # ----------- <Your code> ---------------
    # dictionary should include loss, accuracy and prediction
    assert "loss" and "accuracy" and "prediction" in test_stat.keys()
    # "prediction" value should be a 1D tensor
    assert len(test_stat["prediction"]) == len(test_loader.dataset)
    assert isinstance(test_stat["prediction"], torch.Tensor)
    return test_stat
```

Task 2: Following the structure used in the instructions, you should create

One network named `OurFC` which should consist with only fully connected layers

- You need to add one `nn.Linear(*, 256)` where one of the dimension is `256` and decide how many other layers and hidden dimensions you want in your network (apart from this).
- Your final accuracy on the test dataset should lie roughly around 97% (±2)
- There is no need to make the neural network unnecessarily complex, your total training time should no longer than 3 mins

Another network named `OurCNN` which applys a standard CNN structure

- You should have one `nn.Conv2d(*, *, kernel_size=5)` convolutional layer with `kernel_size=5`, and again, you should decide how many layers and channels you want for each layer.
- Your final accuracy on the test dataset should lie roughly around 97% ($\pm$2%)
- A standard CNN structure can be composed as **[Conv2d, MaxPooling, ReLU] x num_conv_layers + FC x num_fc_layers**
- Train and test your network on MNIST data as in the instructions.
- Notice You can always use the `train` and `test` function you write throughout this assignment.
- The code below will also print out the number of parameters for both neural networks to allow comparison.
- (You can use multiple cells if helpful but make sure to run all of them to receive credit.)

In [ ]:
```
# Download MNIST and transformation
# ----------- <Your code> ---------------
transform = torchvision.transforms.Compose([torchvision.transforms.ToTensor(), torc

train_dataset = torchvision.datasets.MNIST('data', train=True, download=True, trans
test_dataset = torchvision.datasets.MNIST('data', train=False, download=True, trans

print(train_dataset)
batch_size_train, batch_size_test = 64, 1000

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size_tra
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size_test,
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
to data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9.91M/9.91M [00:00<00:00, 16.0MB/s]
Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
to data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28.9k/28.9k [00:00<00:00, 475kB/s]
```

```
Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
to data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.40MB/s]
Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
to data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4.54k/4.54k [00:00<00:00, 6.32MB/s]
Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw

Dataset MNIST
    Number of datapoints: 60000
    Root location: data
    Split: Train
    StandardTransform
Transform: Compose(
               ToTensor()
               Normalize(mean=(0.1307,), std=(0.3081,))
           )
```

In [ ]:

```python
# Build OurFC class and OurCNN class.
# ----------- <Your code> ---------------
class OurFC(nn.Module):
  def __init__(self, input_size = 28*28, hidden_layers = [512, 256]):
    super(OurFC, self).__init__()
    layers = list()
    input_features = input_size
    for output_features in hidden_layers:
      layers.append(nn.Linear(input_features, output_features))
      layers.append(nn.ReLU())
      input_features = output_features

    layers.append(nn.Linear(input_features, 256))
    self.fc_layers = nn.Sequential(*layers)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    return self.fc_layers(x)


class OurCNN(nn.Module):
  def __init__(self, num_conv_layers = 2, num_fc_layers = 1, num_channels = 3):
    super(OurCNN, self).__init__()
```

```
    con_layers = list()
    input_channels = 1
    for x in range(num_conv_layers):
      con_layers.append(nn.Conv2d(input_channels, num_channels, kernel_size=5))
      con_layers.append(nn.MaxPool2d(2))
      con_layers.append(nn.ReLU())
      input_channels = num_channels
    self.con_layers = nn.Sequential(*con_layers)
    self.fc_layers = nn.Sequential(nn.Linear(4 * 4 * input_channels, 256), nn.ReLU(

  def forward(self, x):
    x = self.con_layers(x)
    x = x.view(x.size(0), -1)
    return self.fc_layers(x)
```

In [ ]:

```
# Let's first train the FC model. Below are there common hyperparameters.
criterion = nn.CrossEntropyLoss()

start = time.time()
max_epoch = 3
# ----------- <Your code> ---------------
classifier = OurFC()
optimizer = optim.Adam(classifier.parameters(), lr=0.001)
for epoch in range(1, max_epoch + 1):
  train_losses = train(classifier, criterion, optimizer, train_loader, epoch)
  test_acc = test(classifier, criterion, test_loader, epoch)
# ----------- <End Your code> ---------------
end = time.time()
print(f'Finished Training after {end-start} s ')
```

```
Epoch 1: [0/60000] Loss: 5.530
Epoch 1: [7488/60000] Loss: 0.309
Epoch 1: [14976/60000] Loss: 0.131
Epoch 1: [22464/60000] Loss: 0.186
Epoch 1: [29952/60000] Loss: 0.259
Epoch 1: [37440/60000] Loss: 0.134
Epoch 1: [44928/60000] Loss: 0.182
Epoch 1: [52416/60000] Loss: 0.123
Epoch 1: [59904/60000] Loss: 0.186
Test result on epoch 1: total sample: 10000, Avg loss:0.000, Acc: 96.110%
Epoch 2: [0/60000] Loss: 0.091
Epoch 2: [7488/60000] Loss: 0.113
Epoch 2: [14976/60000] Loss: 0.099
Epoch 2: [22464/60000] Loss: 0.098
Epoch 2: [29952/60000] Loss: 0.072
Epoch 2: [37440/60000] Loss: 0.058
Epoch 2: [44928/60000] Loss: 0.012
Epoch 2: [52416/60000] Loss: 0.063
Epoch 2: [59904/60000] Loss: 0.081
Test result on epoch 2: total sample: 10000, Avg loss:0.000, Acc: 97.140%
Epoch 3: [0/60000] Loss: 0.029
Epoch 3: [7488/60000] Loss: 0.083
Epoch 3: [14976/60000] Loss: 0.037
Epoch 3: [22464/60000] Loss: 0.070
Epoch 3: [29952/60000] Loss: 0.037
Epoch 3: [37440/60000] Loss: 0.108
Epoch 3: [44928/60000] Loss: 0.040
Epoch 3: [52416/60000] Loss: 0.032
Epoch 3: [59904/60000] Loss: 0.166
Test result on epoch 3: total sample: 10000, Avg loss:0.000, Acc: 97.630%
Finished Training after 57.88507032394409 s
```

In [ ]:

```python
# Let's then train the OurCNN model.
start = time.time()
# ----------- <Your code> ---------------
classifier = OurCNN()
optimizer = optim.Adam(classifier.parameters(), lr=0.001)
for epoch in range(1, max_epoch + 1):
  train_losses = train(classifier, criterion, optimizer, train_loader, epoch)
  test_acc = test(classifier, criterion, test_loader, epoch)

# ----------- <End Your code> ---------------
end = time.time()
print(f'Finished Training after {end-start} s ')
```

```
Epoch 1: [0/60000] Loss: 2.314
Epoch 1: [7488/60000] Loss: 0.275
Epoch 1: [14976/60000] Loss: 0.424
Epoch 1: [22464/60000] Loss: 0.286
Epoch 1: [29952/60000] Loss: 0.219
Epoch 1: [37440/60000] Loss: 0.074
Epoch 1: [44928/60000] Loss: 0.047
Epoch 1: [52416/60000] Loss: 0.075
Epoch 1: [59904/60000] Loss: 0.096
Test result on epoch 1: total sample: 10000, Avg loss:0.000, Acc: 96.250%
Epoch 2: [0/60000] Loss: 0.040
Epoch 2: [7488/60000] Loss: 0.079
Epoch 2: [14976/60000] Loss: 0.079
Epoch 2: [22464/60000] Loss: 0.149
Epoch 2: [29952/60000] Loss: 0.089
Epoch 2: [37440/60000] Loss: 0.021
Epoch 2: [44928/60000] Loss: 0.053
Epoch 2: [52416/60000] Loss: 0.114
Epoch 2: [59904/60000] Loss: 0.050
Test result on epoch 2: total sample: 10000, Avg loss:0.000, Acc: 97.210%
Epoch 3: [0/60000] Loss: 0.047
Epoch 3: [7488/60000] Loss: 0.019
Epoch 3: [14976/60000] Loss: 0.147
Epoch 3: [22464/60000] Loss: 0.092
Epoch 3: [29952/60000] Loss: 0.038
Epoch 3: [37440/60000] Loss: 0.163
Epoch 3: [44928/60000] Loss: 0.095
Epoch 3: [52416/60000] Loss: 0.019
Epoch 3: [59904/60000] Loss: 0.025
Test result on epoch 3: total sample: 10000, Avg loss:0.000, Acc: 97.910%
Finished Training after 54.90977215766907 s
```

In [ ]:
```python
ourfc = OurFC()
total_params = sum(p.numel() for p in ourfc.parameters())
print(f'OurFC has a total of {total_params} parameters')

ourcnn = OurCNN()
total_params = sum(p.numel() for p in ourcnn.parameters())
print(f'OurCNN has a total of {total_params} parameters')
```

```
OurFC has a total of 599040 parameters
OurCNN has a total of 15420 parameters
```

Questions (0 points, just for understanding): Which one has more parameters? Which one is likely to have less computational cost when deployed? Which one took longer to train?

Exercise 2: Train classifier on CIFAR-10 data. (30 points)

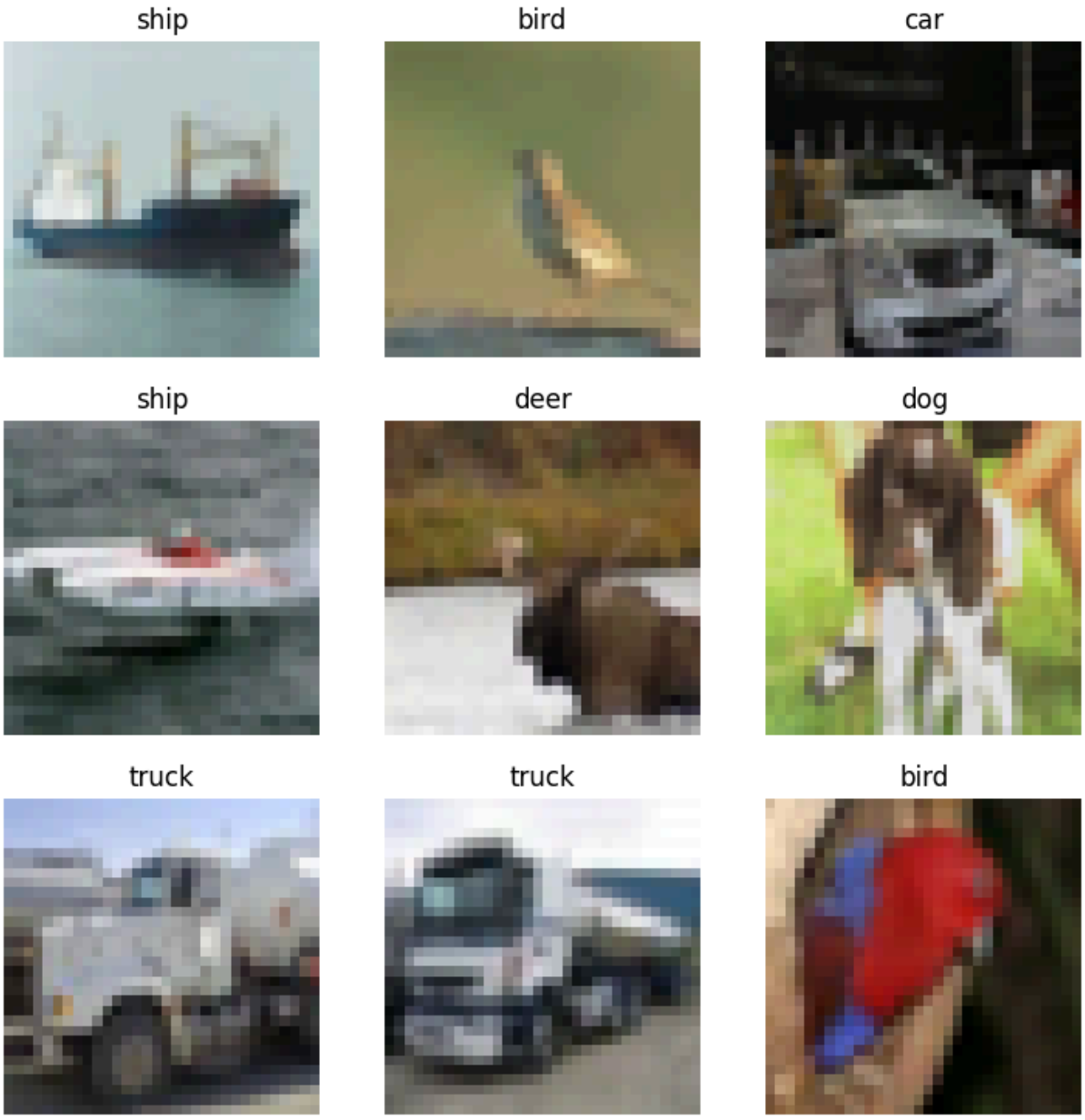
Now, lets move our dataset to color images. CIFAR-10 dataset is another widely used dataset. Here all images have colors, i.e each image has 3 color channels instead of only one channel in MNIST. You need to pay more attention to the dimension of the data as it passes through the layers of your network.

Task 1: Create data loaders

- Load CIFAR10 train and test datas with appropriate composite transform where the normalize transform should be `transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])`.
- Set up a `train_loader` and `test_loader` for the CIFAR-10 data with a batch size of 9 similar to the instructions.
- The code below will plot a 3 x 3 subplot of images including their labels. (do not modify)

In [ ]:
```
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', '

# load data and transformations
# ----------- <Your code> ---------------
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, t
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, t


# ----------- <End Your code> ---------------


# Define trainloader and testloader
# ----------- <Your code> ---------------
train_loader = torch.utils.data.DataLoader(trainset, batch_size=9, shuffle=True)
test_loader = torch.utils.data.DataLoader(testset, batch_size=9, shuffle=False)

# ----------- <End Your code> ---------------

# Code to display images
batch_idx, (images, targets) = next(enumerate(train_loader))
fig, ax = plt.subplots(3,3,figsize = (9,9))
for i in range(3):
    for j in range(3):
        image = images[i*3+j].permute(1,2,0)
        image = image/2 + 0.5
        ax[i,j].imshow(image)
        ax[i,j].set_axis_off()
        ax[i,j].set_title(f'{classes[targets[i*3+j]]}')
fig.show()
```

```
Files already downloaded and verified
Files already downloaded and verified
```



Task 2: Create CNN and train it

Set up a convolutional neural network and have your data trained on it. You have to decide all the details in your network, overall your neural network should meet the following standards to receive full credit:

- You should not use more than three convolutional layers and three fully connected layers
- Accuracy on the test dataset should be **above** 50%

In [ ]:
```
# Create CNN network.
# ----------- <Your code> ---------------
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(3, 6, 5)
    self.conv2 = nn.Conv2d(6, 16, 5)
    self.pool = nn.MaxPool2d(2, 2)
    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

# ----------- <End Your code> ---------------
```

In [ ]:

```
# Train your neural network here.
start = time.time()
max_epoch = 4
# ----------- <Your code> ---------------
# Define net
net = Net()
optimizer = optim.Adam(net.parameters(), 0.001)

for epoch in range(1, max_epoch + 1):
  train_losses = train(net, criterion, optimizer, train_loader, epoch)
  test_acc = test(net, criterion, test_loader, epoch)

# ----------- <End Your code> ---------------
output = test(net, criterion, test_loader, epoch)
end = time.time()
print(f'Finished Training after {end-start} s ')
```

```
Epoch 1: [0/50000] Loss: 2.321
Epoch 1: [6246/50000] Loss: 1.560
Epoch 1: [12492/50000] Loss: 2.035
Epoch 1: [18738/50000] Loss: 1.630
Epoch 1: [24984/50000] Loss: 1.613
Epoch 1: [31230/50000] Loss: 1.060
Epoch 1: [37476/50000] Loss: 2.149
Epoch 1: [43722/50000] Loss: 1.080
Epoch 1: [49968/50000] Loss: 1.651
Test result on epoch 1: total sample: 10000, Avg loss:0.147, Acc: 52.650%
Epoch 2: [0/50000] Loss: 1.322
Epoch 2: [6246/50000] Loss: 1.051
Epoch 2: [12492/50000] Loss: 1.653
Epoch 2: [18738/50000] Loss: 1.700
Epoch 2: [24984/50000] Loss: 1.384
Epoch 2: [31230/50000] Loss: 0.844
Epoch 2: [37476/50000] Loss: 1.117
Epoch 2: [43722/50000] Loss: 1.349
Epoch 2: [49968/50000] Loss: 1.577
Test result on epoch 2: total sample: 10000, Avg loss:0.139, Acc: 54.610%
Epoch 3: [0/50000] Loss: 0.938
Epoch 3: [6246/50000] Loss: 0.917
Epoch 3: [12492/50000] Loss: 0.767
Epoch 3: [18738/50000] Loss: 1.808
Epoch 3: [24984/50000] Loss: 0.908
Epoch 3: [31230/50000] Loss: 0.804
Epoch 3: [37476/50000] Loss: 1.381
Epoch 3: [43722/50000] Loss: 0.589
Epoch 3: [49968/50000] Loss: 1.070
Test result on epoch 3: total sample: 10000, Avg loss:0.128, Acc: 59.070%
Epoch 4: [0/50000] Loss: 1.031
Epoch 4: [6246/50000] Loss: 1.479
Epoch 4: [12492/50000] Loss: 1.208
Epoch 4: [18738/50000] Loss: 1.063
Epoch 4: [24984/50000] Loss: 1.416
Epoch 4: [31230/50000] Loss: 1.404
Epoch 4: [37476/50000] Loss: 1.228
Epoch 4: [43722/50000] Loss: 0.775
Epoch 4: [49968/50000] Loss: 1.353
Test result on epoch 4: total sample: 10000, Avg loss:0.133, Acc: 57.580%
Test result on epoch 4: total sample: 10000, Avg loss:0.133, Acc: 57.580%
Finished Training after 188.48106908798218 s
```

Task 3: Plot misclassified test images

Plot some misclassified images in your test dataset:

- select five images that are **misclassified** for `class_id in {1,3,5,7,9}` by your neural network, one image each (i.e., the true label is `class_id` but the predicted label is not `class_id`).
- label each images with true label and predicted label
- use `detach().cpu()` when plotting images if the image is in gpu

In [ ]:
```python
total_images = 5
predictions = output['prediction']
targets = torch.tensor(testset.targets)
# ----------- <Your code> ---------------
mask = predictions == targets.unsqueeze(1)
```

```
data = testset.data
images = list()
labels = list()
class_ids = {1, 3, 5, 7, 9}

fig, ax = plt.subplots(1,5,figsize = (9,2))

for class_id in class_ids:
  for idx, val in enumerate(mask):
    if not val[0] and targets[idx] == class_id:
      images.append(data[idx])
      labels.append((predictions[idx], targets[idx]))
      break

for x in range(total_images):
  image = images[x]
  ax[x].imshow(image)
  ax[x].set_axis_off()
  ax[x].set_title(f'Actual: {classes[labels[x][1]]}\nMislabel: {classes[labels[x][0

fig.show()

# ----------- <End Your code> ---------------
```

Questions (0 points): Are the mis-classified images also misleading to human eyes?

Exercise 3: Transfer Learning (30 points)

In practice, people won't train an entire CNN from scratch, because it is relatively rare to have a dataset of sufficient size (or sufficient computational power). Instead, it is common to pretrain a CNN on a very large dataset and then use the CNN either as an initialization or a fixed feature extractor for the task of interest.

In this task, you will learn how to use a pretrained CNN for CIFAR-10 classification.

Task1: Load pretrained model

`torchvision.models` (https://pytorch.org/vision/stable/models.html) contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection and video classification.

First, you should load the **pretrained** ResNet-18 that has already been trained on ImageNet using `torchvision.models`. If you are interested in more details about Resnet-18, read this paper "Deep Residual Learning for Image Recognition".

In [ ]:
```
resnet18 = models.resnet18(pretrained=True)
resnet18 = resnet18.to(device)
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarnin
g: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the fut
ure, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarnin
g: Arguments other than a weight enum or `None` for 'weights' are deprecated since
0.13 and may be removed in the future. The current behavior is equivalent to passing
`weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet18_Weight
s.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

Task2: Create data loaders for CIFAR-10

Then you need to create a modified dataset and dataloader for CIFAR-10. Importantly, the model you load has been trained on **ImageNet** and it expects inputs as mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be **at least** 224. So you need to preprocess the CIFAR-10 data to make sure it has a height and width of 224. Thus, you should add a transform when loading the CIFAR10 dataset (see `torchvision.transforms.Resize`). This should be added appropriately to the `transform` you created in a previous task.

In [ ]:
```
# Create your dataloader here
# ----------- <Your code> ---------------
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, t
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, t

train_loader = torch.utils.data.DataLoader(trainset, batch_size=9, shuffle=True)
test_loader = torch.utils.data.DataLoader(testset, batch_size=9, shuffle=False, pin
# ----------- <End Your code> ---------------
```

```
Files already downloaded and verified
Files already downloaded and verified
```

Task3: Classify test data on pretrained model

Use the model you load to classify the **test** CIFAR-10 data and print out the test accuracy.

Don't be surprised if the accuracy is bad!

In [ ]:
```
# ----------- <Your code> ---------------
resnet18.eval()
total_num = len(test_loader.dataset)
t_loss = 0
num_correct = 0
predictions = list()

with torch.no_grad():
  for images, targets in test_loader:
    images = images.to(device)
    targets = targets.to(device)
    out = resnet18(images)
    loss = criterion(out, targets)
    t_loss += loss.item()
    pred = out.data.max(1, keepdim=True)[1]
    num_correct += pred.eq(targets.data.view_as(pred)).sum()
    predictions.extend(pred.tolist())
accuracy = num_correct / total_num
```

```
print(f"Resnet18 accuracy: {accuracy:.3f}%")
# ----------- <End Your code> ---------------
```

```
Resnet18 accuracy: 0.000%
```

Task 4: Fine-tune (i.e., update) the pretrained model for CIFAR-10

Now try to improve the test accuracy. We offer several possible solutions:

(1) You can try to directly continue to train the model you load with the CIFAR-10 training data.

(2) For efficiency, you can try to freeze part of the parameters of the loaded models. For example, you can first freeze all parameters by

```
for param in model.parameters():
    param.requires_grad = False
```

and then unfreeze the last few layers by setting `somelayer.requires_grad=True`.

You are also welcome to try any other approach you can think of.

Note: You must print out the test accuracy and to get full credits, the test accuracy should be at least **80%**.

In [ ]:
```
# Directly train the whole model.
start = time.time()
#----------- <Your code> ---------------



# ----------- <End Your code> ---------------
test(resnet18, criterion, test_loader, epoch)
end = time.time()
print(f'Finished Training after {end-start} s ')
```

In [100…
```
# Load another resnet18 instance, only unfreeze the outer layers.
# ----------- <Your code> ---------------
for param in resnet18.parameters():
  param.requires_grad = False

params = list(resnet18.parameters())

for param in params[-5:]:
  param.requires_grad = True


# ----------- <End Your code> ---------------
```

In [101…
```
# Train the model!!
start = time.time()
# ----------- <Your code> ---------------
optimizer = optim.SGD(resnet18.parameters(), lr=0.002, momentum=0.9)
for epoch in range(1, 6):
  train_loss = train(resnet18, criterion, optimizer, train_loader, epoch)


# ----------- <End Your code> ---------------
test(resnet18, criterion, test_loader)
end = time.time()
print(f'Finished Training after {end-start} s ')
```

```
Epoch 1: [0/50000] Loss: 0.048
Epoch 1: [6246/50000] Loss: 0.269
Epoch 1: [12492/50000] Loss: 0.756
Epoch 1: [18738/50000] Loss: 0.676
Epoch 1: [24984/50000] Loss: 0.775
Epoch 1: [31230/50000] Loss: 0.486
Epoch 1: [37476/50000] Loss: 0.391
Epoch 1: [43722/50000] Loss: 0.445
Epoch 1: [49968/50000] Loss: 0.400
Epoch 2: [0/50000] Loss: 0.607
Epoch 2: [6246/50000] Loss: 0.547
Epoch 2: [12492/50000] Loss: 0.442
Epoch 2: [18738/50000] Loss: 0.172
Epoch 2: [24984/50000] Loss: 0.804
Epoch 2: [31230/50000] Loss: 0.149
Epoch 2: [37476/50000] Loss: 0.533
Epoch 2: [43722/50000] Loss: 0.510
Epoch 2: [49968/50000] Loss: 0.371
Epoch 3: [0/50000] Loss: 1.090
Epoch 3: [6246/50000] Loss: 0.559
Epoch 3: [12492/50000] Loss: 0.073
Epoch 3: [18738/50000] Loss: 0.308
Epoch 3: [24984/50000] Loss: 0.487
Epoch 3: [31230/50000] Loss: 0.191
Epoch 3: [37476/50000] Loss: 0.723
Epoch 3: [43722/50000] Loss: 0.364
Epoch 3: [49968/50000] Loss: 0.449
Epoch 4: [0/50000] Loss: 1.242
Epoch 4: [6246/50000] Loss: 0.452
Epoch 4: [12492/50000] Loss: 0.467
Epoch 4: [18738/50000] Loss: 0.964
Epoch 4: [24984/50000] Loss: 0.319
Epoch 4: [31230/50000] Loss: 0.379
Epoch 4: [37476/50000] Loss: 0.355
Epoch 4: [43722/50000] Loss: 0.159
Epoch 4: [49968/50000] Loss: 0.663
Epoch 5: [0/50000] Loss: 0.415
Epoch 5: [6246/50000] Loss: 0.202
Epoch 5: [12492/50000] Loss: 0.582
Epoch 5: [18738/50000] Loss: 0.255
Epoch 5: [24984/50000] Loss: 0.406
Epoch 5: [31230/50000] Loss: 0.173
Epoch 5: [37476/50000] Loss: 0.297
Epoch 5: [43722/50000] Loss: 0.184
Epoch 5: [49968/50000] Loss: 0.114
Test result on epoch 0: total sample: 10000, Avg loss:0.051, Acc: 84.590%
Finished Training after 556.9676067829132 s
```