Daniel, Eric, Tony

Data Science Lab: Lab 7 part I (of III)

Submit:

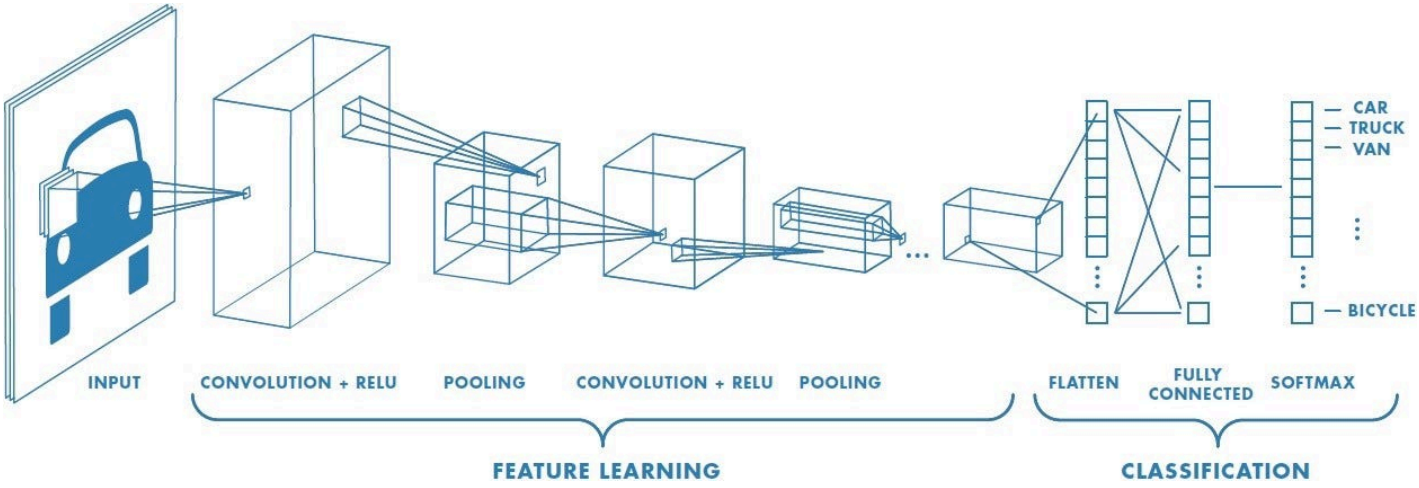
A pdf of your notebook with solutions. A link to your colab notebook or also upload your .ipynb if not working on colab.

Goals of this Lab

There are a number of goals of this notebook:

1. Learning to put together a basic model beyond what we did in the previous notebook. Main emphasis: new layers, including convolution and max pooling.
2. Learning the basics of training.
3. Learning basics of loading data and visualizing.
4. Learning the basics of printing out a model.

Also useful to see https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html




```
import torchvision
import torchvision.transforms as transforms
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
batch_size = 4

# the tutorial calls the dataloader twice -- this code defines a function
# that will do this for the train/test data.

def fetch_dataloader(batch_size, transform=None, is_train=True):
    """
    We saw the data loaders in the previous lab.
    This creates a method for us to get (image, label) pairs.
    We can use it so that we do not have to load everything into memory
    at once.
    """
    data = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=

    # Custom train/val split.
    indices = [i for i in range(len(data)) if (i%10 > 0) == is_train]

    data = torch.utils.data.Subset(data, indices)
    loader = torch.utils.data.DataLoader(data, batch_size=batch_size, shuffle=True, num_work
    return loader


train_transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

val_transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

data_train = fetch_dataloader(batch_size, train_transform, is_train=True)
data_val = fetch_dataloader(batch_size, val_transform, is_train=False)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-p
100%|██████████| 170M/170M [00:08<00:00, 20.3MB/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Problem 1 (nothing to turn in)

Read about transforms. These are routinely used when loading images. We've used a particular set of parameters. But you will see many other choices.

https://pytorch.org/vision/stable/transforms.html

Problem 2

Figure out how to use the data loader to display the data. For example, you could look here

https://towardsdatascience.com/beginners-guide-to-loading-image-data-with-pytorch-289c60b7afec and then https://stackoverflow.com/questions/51756581/how-do-i-turn-a-pytorch-dataloader-into-a-numpy-array-to-display-image-data-with

or you could look at the pytorch CIFAR10 tutorial

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

```
import torchvision.utils
import matplotlib.pyplot as plt
import numpy as np

data_iter = iter(data_train)
images, labels = next(data_iter)

print("Batch size:", images.size(0))  #images per match
print("Image shape:", images[0].shape)  # shape of image tensor

img = torchvision.utils.make_grid(images, nrow=4) / 2 + 0.5
npimg = img.numpy()
plt.imshow(np.transpose(npimg, (1, 2, 0)))
plt.axis('off')
plt.show()
```

```
Batch size: 4
Image shape: torch.Size([3, 32, 32])
```

Model Implementation.

Now it is time to implement our CNN. You will recognize syntax from the previous Lab. However, whereas in the last lab we had only fully connected layers and ReLU layers, here we will use more options. In addition to fully connected and ReLU layers, we want to use:

- Convolutional layers: `torch.nn.Conv2d`
- Max Pooling Layers: `torch.nn.MaxPool2d`
- Average Pooling Layers: `torch.nn.AvgPool2d`

Fully connected layers primarily had 3 parameters: input size, output size, and bias.

Convolutional layers have many more parameters, as we discussed in class, In particular, recall:

- kernel_size
- stride
- padding
- dilation

Read about these in the Pytorch documentation:

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

Next we see some basic code, similar in spirit to our previous colab notebook, but, critically, adding a new type of layer: a convolutional layer.

```
# We define two different models, using different syntax.
# CNNClassifier is basic code borrowed from Phil Krahenbuhl
# ConvNet comes from the pytorch tutorial linked above.
#

class CNNClassifier(torch.nn.Module):
    def __init__(self, input_channels, num_classes):
        """
        Define the layer(s) needed for the model.
        Feel free to define additional input arguments.
        """
        super().__init__()
        self.conv = nn.Conv2d(input_channels, 16, 7, 2, 3)
        self.cls = nn.Linear(16, num_classes)
        self.ReLU = nn.ReLU()
        # or could do
        # self.CNN = nn.Sequential(torch.nn.Conv2d(input_channels, 16, 7, 2, 3), torch.nn.Re

    def forward(self, x):
        """
        Calculate the classification score (logits).

        Input:
            x (float tensor N x 3 x 32 x 32): input images
        Output:
            y (float tensor N x 10): classification scores (logits) for each class
        """
        x = self.conv(x)

        # Add a ReLU
        x = self.ReLU(x)
        # Add global average pooling
        x = x.mean(dim=(2,3))
```

```
            return self.cls(x)

        def predict(self, image):
            return self(image).argmax(1)

    # From the pytorch tutorial linked above.
    class ConvNet(nn.Module):
        def __init__(self):
            super().__init__()
            self.relu = nn.ReLU()
            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.conv1(x)
            x = self.relu(x)
            x = self.pool(x)
            x = self.conv2(x)
            x = self.relu(x)
            x = self.pool(x)
            x = torch.flatten(x, 1) # flatten all dimensions except batch
            x = self.fc1(x)
            x = self.relu(x)
            x = self.fc2(x)
            x = self.relu(x)
            x = self.fc3(x)
            return x

    # We can rewrite it equivalently this way
    class ConvNet2(nn.Module):
        def __init__(self):
            super().__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 = torch.flatten(x, 1) # flatten all dimensions except batch
            x = F.relu(self.fc1(x))
            x = F.relu(self.fc2(x))
            x = self.fc3(x)
            return x
```

```
# This way is also equivalent
class ConvNet3(nn.Module):
    def __init__(self):
        super().__init__()
        self.relu = nn.ReLU()
        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(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Model Training and Validation

```
# Now we train the first model.
#


# Choose a model to train
model = CNNClassifier(3, 10)
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
model.train()


for epoch in range(5):  # loop over the dataset multiple times

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

        # zero the parameter gradients
```

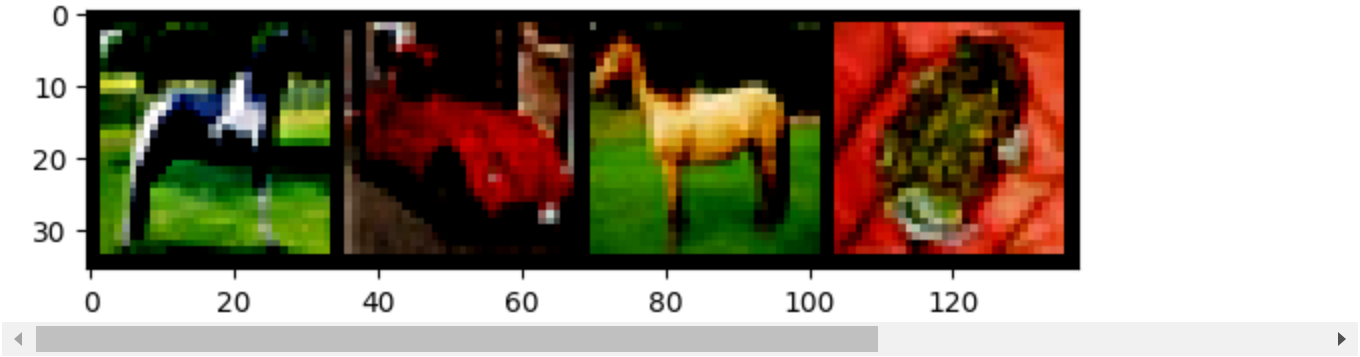
```
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = model(inputs)
            loss = loss_function(outputs, labels)
            loss.backward()
            optimizer.step()

            # 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))
                running_loss = 0.0

    print('Finished Training')
```

```
[1,  2000] loss: 2.211
[1,  4000] loss: 2.080
[1,  6000] loss: 2.032
[1,  8000] loss: 1.979
[1, 10000] loss: 1.941
[2,  2000] loss: 1.894
[2,  4000] loss: 1.863
[2,  6000] loss: 1.837
[2,  8000] loss: 1.823
[2, 10000] loss: 1.821
[3,  2000] loss: 1.795
[3,  4000] loss: 1.796
[3,  6000] loss: 1.785
[3,  8000] loss: 1.789
[3, 10000] loss: 1.794
[4,  2000] loss: 1.784
[4,  4000] loss: 1.781
[4,  6000] loss: 1.764
[4,  8000] loss: 1.753
[4, 10000] loss: 1.758
[5,  2000] loss: 1.742
[5,  4000] loss: 1.739
[5,  6000] loss: 1.750
[5,  8000] loss: 1.754
[5, 10000] loss: 1.752
Finished Training
```

```
    # Let's see how well this trained model performs on a couple data points.
    dataiter = iter(data_val)
    images, labels = next(dataiter)

    # print images
    plt.imshow(torchvision.utils.make_grid(images).permute(1,2,0))
    print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))

    # and now what our model thinks
```

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

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data
GroundTruth:  horse   car horse  frog
Predicted:  truck  ship  bird   dog
```



Problem 3

Use your validation data loader (data_val) to assess the accuracy on the entire data set.

If you did this correctly, you should find that your accuracy is somewhere in the 30-40% range -- not great, but a lot better than guessing, and much better than we were able to do with trees.

```
model.eval()
correct = 0
total = 0
for images, labels in data_val:
  outputs = model(images)
  val, predicted = torch.max(outputs, 1)  # highest predicted class
  total += labels.size(0)
  correct += (predicted == labels).sum().item() # count number of correct
print(f'Accuracy: {100 * correct / total:.2f}%')
```

```
Accuracy: 34.86%
```

ANSWER: Accuracy: 34.86%

Time to train the deeper model

```
# Now we train the second model. We see that it does quite a bit better than the first.
#
```

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


# Choose a model to train
model2 = ConvNet() # note that this command resets the parameters
model2.to(device)
model2.train()

# Set the loss function and optimizer
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model2.parameters(), lr=0.001, momentum=0.9)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')




for epoch in range(2):  # loop over the dataset multiple times

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

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model2(inputs)
        loss = loss_function(outputs, labels)
        loss.backward()
        optimizer.step()

        # 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))
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 2.228
[1,  4000] loss: 1.844
[1,  6000] loss: 1.670
[1,  8000] loss: 1.582
[1, 10000] loss: 1.503
[2,  2000] loss: 1.409
[2,  4000] loss: 1.379
[2,  6000] loss: 1.355
```

```
[2,  8000] loss: 1.332
[2, 10000] loss: 1.314
Finished Training
```

Problem 4

Repeat what you did for Problem 3 above: use your validation data loader (data_val) to assess the accuracy on the entire data set.

If you did this correctly, you should find that your accuracy is somewhere in the 50-60% range. Far from perfect, but still better than how we did above.

```
model2.eval()
correct = 0
total = 0
for images, labels in data_val:
  outputs = model2(images)
  val, predicted = torch.max(outputs, 1)  # highest predicted class
  total += labels.size(0)
  correct += (predicted == labels).sum().item() # count number of correct
print(f'Accuracy: {100 * correct / total:.2f}%')
```

```
Accuracy: 55.06%
```

ANSWER: Accuracy: 55.06%

Problem 5

How many parameters does your model have? You can either compute by hand, or much better, use the summary command from torchsummary

```
from torchsummary import summary
```

Note that you'll have to figure out how to use use, and also you will have to enter the size of your input.

Also try using the command

```
for name, layer in model2.named_modules():
    print(name,layer)
```

This one is very useful when we want to download someone else's netework, and we need to know what the names of the layers are.

```
from torchsummary import summary

summary(model2, input_size=(3, 32, 32)) # found in problem 1

for name, layer in model2.named_modules():
  print(name,layer)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             456
              ReLU-2            [-1, 6, 28, 28]               0
         MaxPool2d-3            [-1, 6, 14, 14]               0
            Conv2d-4           [-1, 16, 10, 10]           2,416
              ReLU-5           [-1, 16, 10, 10]               0
         MaxPool2d-6             [-1, 16, 5, 5]               0
            Linear-7                  [-1, 120]          48,120
              ReLU-8                  [-1, 120]               0
            Linear-9                   [-1, 84]          10,164
             ReLU-10                   [-1, 84]               0
           Linear-11                   [-1, 10]             850
================================================================
Total params: 62,006
Trainable params: 62,006
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.11
Params size (MB): 0.24
Estimated Total Size (MB): 0.36
----------------------------------------------------------------
 ConvNet(
  (relu): ReLU()
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
relu ReLU()
conv1 Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
pool MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
conv2 Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
fc1 Linear(in_features=400, out_features=120, bias=True)
fc2 Linear(in_features=120, out_features=84, bias=True)
fc3 Linear(in_features=84, out_features=10, bias=True)
```

ANSWER: Total params = 62,006

Problem 6

How much can you decrease the error? Try to make your own convolutional neural network for CIFAR-10. Don't make it so big that you can't train it on Colab! You can play around with the parameters of the network and also with the parameters of training (train for longer -- more epochs, possibly using other parameters for step size, momentum, etc.).

We will give 20 lab bonus points to the team with the best accuracy.

```
# This way is also equivalent
class ConvNetCustom(nn.Module):
    def __init__(self):
        super().__init__()
        self.relu = nn.ReLU()
        self.conv1 = nn.Conv2d(3, 64, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(64, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 400)
        self.fc2_5 = nn.Linear(400, 84)
        self.fc2 = nn.Linear(400, 84)
        self.fc3 = nn.Linear(84, 10)
        self.dropout = nn.Dropout(0.5) # just cuts half the neurons

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        # x = self.pool(self.relu(self.conv1(x)))
        # x = self.pool(self.relu(self.conv2(x)))
        # #x = self.pool2(self.relu(self.conv3(x)))
        # x = torch.flatten(x, 1) # flatten all dimensions except batch
        # x = self.relu(self.fc1(x))
        # #x = self.dropout(x)
        # x = self.relu(self.fc2(x))
        # #x = self.dropout(x)
        # #x = self.relu(self.fc2_5(x))
        # x = self.fc3(x)
        return x


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model3 = ConvNetCustom()
```

```
model3.to(device)
model3.train()

loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model3.parameters(), lr=0.0015, momentum=0.85)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


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

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

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model3(inputs)
        loss = loss_function(outputs, labels)
        loss.backward()
        optimizer.step()

        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))
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 2.183
[1,  4000] loss: 1.904
[1,  6000] loss: 1.742
[1,  8000] loss: 1.676
[1, 10000] loss: 1.611
[2,  2000] loss: 1.542
[2,  4000] loss: 1.487
[2,  6000] loss: 1.451
[2,  8000] loss: 1.424
[2, 10000] loss: 1.397
[3,  2000] loss: 1.341
[3,  4000] loss: 1.336
[3,  6000] loss: 1.325
[3,  8000] loss: 1.278
[3, 10000] loss: 1.304
[4,  2000] loss: 1.226
[4,  4000] loss: 1.228
```

```
[4,  6000] loss: 1.222
[4,  8000] loss: 1.212
[4, 10000] loss: 1.196
[5,  2000] loss: 1.166
[5,  4000] loss: 1.158
[5,  6000] loss: 1.161
[5,  8000] loss: 1.152
[5, 10000] loss: 1.147
[6,  2000] loss: 1.088
[6,  4000] loss: 1.102
[6,  6000] loss: 1.115
[6,  8000] loss: 1.105
[6, 10000] loss: 1.095
[7,  2000] loss: 1.031
[7,  4000] loss: 1.045
[7,  6000] loss: 1.068
[7,  8000] loss: 1.070
[7, 10000] loss: 1.077
Finished Training
```

```
model3.eval()
correct = 0
total = 0
for images, labels in data_val:
  outputs = model3(images)
  val, predicted = torch.max(outputs, 1)  # highest predicted class
  total += labels.size(0)
  correct += (predicted == labels).sum().item() # count number of correct
print(f'Accuracy: {100 * correct / total:.2f}%')
```

```
Accuracy: 65.82%
```

ANSWER: Accuracy: 65.82%

Data Science Lab: Lab 7 part II (of III)

Submit:

A pdf of your notebook with solutions. A link to your colab notebook or also upload your .ipynb if not working on colab.

Goals of this Lab: Transfer Learning

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

In the previous notebook, we learned how to put together various different types of layers to build a convolutional neural network (CNN) to classify CIFAR-10. Now we are going to learn about the principle of transfer learning.

We will create 2 data sets: one with the first five labels of CIFAR-10 (plane, car, bird, cat, deer), and one with the remaining 5. We will see how training on the second data set, can actually help us with the first.

Note: because CIFAR-10 is a relatively small data set, the numbers are not overly compelling. But we can greatly amplify this if we use larger data sets.

Problem 1.

Make train and validation (testing) data loaders for the first five labels, and the second five labels.

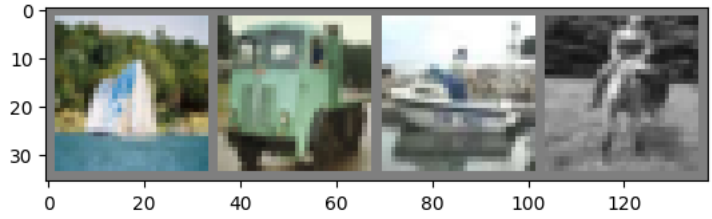
You should have 4 dataloaders when done.

```
# First we download the data. This is quite similar to what we've seen before.
# The main difference will be that we need to make two different loaders,
# one for the first five labels, and one for the remaining five.

import torchvision
import torchvision.transforms as transforms
import torch
import torch.nn as nn
import torch.nn.functional as F
torch.manual_seed(42)


classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

batch_size = 4

# the tutorial calls the dataloader twice -- this code defines a function
# that will do this for the train/test data.

def fetch_dataloader(batch_size, label_subset,transform=None, is_train=True):
    """
    Loads data from disk and returns a data_loader.
    A DataLoader is similar to a list of (image, label) tuples.
    You do not need to fully understand this code to do this assignment, we're happy to explain though.
    """
    data = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)

    # replaced TO DOs with following 5 lines
    indices = []
    for i, sample in enumerate(data):
        if sample[1] in label_subset:
            indices.append(i)
    data = torch.utils.data.Subset(data, indices)

    loader = torch.utils.data.DataLoader(data, batch_size=batch_size, shuffle=True, num_workers=2)
    return loader


train_transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

val_transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

```
"""
TO DO

Complete these four using the function you wrote above, to get the 4 data loaders you need:
two for labels 1-5, and two for labels 6-10.
"""
data15_train = fetch_dataloader(batch_size, [0, 1, 2, 3, 4], transform=train_transform, is_train=True)
data15_val = fetch_dataloader(batch_size, [0, 1, 2, 3, 4], transform=val_transform, is_train=False)
data610_train = fetch_dataloader(batch_size, [5, 6, 7, 8, 9], transform=train_transform, is_train=True)
data610_val = fetch_dataloader(batch_size, [5, 6, 7, 8, 9], transform=val_transform, is_train=False)
```

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

```
# Now let's visualize a few images, to see if this worked.
# Same exact syntax as in the previous notebook.
# Nothing to do here, except run it to make sure it worked.
import matplotlib.pyplot as plt
import numpy as np
from torchvision.utils import make_grid
%matplotlib inline

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(data610_train)
images, labels = next(dataiter)

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




```
ship truck  ship horse
```

```
# We define the same model from the last notebook
# that we got from the Pytorch ConvNet tutorial.

class ConvNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.relu = nn.ReLU()
        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.conv1(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.pool(x)
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = self.fc1(x)
        x = self.relu(x)
```

```
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x


# We train the model on the first five labels to see how well we do after a fixed amount of training.
#
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


# Choose a model to train
model1 = ConvNet() # note that this command resets the parameters
model1.to(device)
model1.train()

# Set the loss function and optimizer
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model1.parameters(), lr=0.001, momentum=0.9)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')




for epoch in range(1):  # a single pas through the data. Don't change this.

    running_loss = 0.0
    for i, data in enumerate(data15_train, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model1(inputs)
        loss = loss_function(outputs, labels)
        loss.backward()
        optimizer.step()

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

print('Finished Training')
```

```
[1,   200] loss: 2.020
[1,   400] loss: 1.613
[1,   600] loss: 1.574
[1,   800] loss: 1.484
[1,  1000] loss: 1.406
[1,  1200] loss: 1.361
[1,  1400] loss: 1.289
[1,  1600] loss: 1.274
[1,  1800] loss: 1.286
[1,  2000] loss: 1.254
[1,  2200] loss: 1.198
[1,  2400] loss: 1.164
[1,  2600] loss: 1.196
[1,  2800] loss: 1.138
[1,  3000] loss: 1.166
[1,  3200] loss: 1.159
[1,  3400] loss: 1.081
[1,  3600] loss: 1.104
[1,  3800] loss: 1.105
[1,  4000] loss: 1.084
[1,  4200] loss: 1.097
[1,  4400] loss: 1.046
[1,  4600] loss: 1.039
[1,  4800] loss: 1.042
[1,  5000] loss: 1.049
[1,  5200] loss: 1.046
[1,  5400] loss: 1.010
[1,  5600] loss: 0.982
[1,  5800] loss: 1.056
```

```
[1,  6000] loss: 1.033
[1,  6200] loss: 1.008
Finished Training
```

Problem 2

In the cell above, you trained your model on labels 1-5 for one single epoch.

How well does this do on the validation data set?

This is nearly identical to problems you solved in part I.

ANSWER: Model 1 accuracy: 61.94%

```
model1.eval()
correct = 0
total = 0
for images, labels in data15_val:
  outputs = model1(images)
  val, predicted = torch.max(outputs, 1)  # highest predicted class
  total += labels.size(0)
  correct += (predicted == labels).sum().item() # count number of correct
print(f'Accuracy: {100 * correct / total:.2f}%')
```

```
Accuracy: 61.94%
```

```
# Now we train the model on the last 5 labels to use for transfer learning.

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


# Choose a model to train
model2 = ConvNet() # note that this command resets the parameters
model2.to(device)
model2.train()

# Set the loss function and optimizer
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model2.parameters(), lr=0.001, momentum=0.9)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


for epoch in range(6):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(data610_train, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        labels = labels-5 # hack b/c labels are 2-9 instead of 0-7
        inputs = inputs.to(device)
        labels = labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model2(inputs)
        loss = loss_function(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 500 == 499:    # print every 500 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 500))
            running_loss = 0.0

print('Finished Training')
```

```
[2,  4000] loss: 0.737
[2,  4500] loss: 0.722
[2,  5000] loss: 0.717
[2,  5500] loss: 0.681
[2,  6000] loss: 0.678
[3,   500] loss: 0.605
[3,  1000] loss: 0.654
[3,  1500] loss: 0.644
[3,  2000] loss: 0.616
[3,  2500] loss: 0.620
[3,  3000] loss: 0.597
[3,  3500] loss: 0.637
[3,  4000] loss: 0.604
[3,  4500] loss: 0.624
[3,  5000] loss: 0.595
[3,  5500] loss: 0.607
[3,  6000] loss: 0.616
[4,   500] loss: 0.522
[4,  1000] loss: 0.540
[4,  1500] loss: 0.532
[4,  2000] loss: 0.558
[4,  2500] loss: 0.554
[4,  3000] loss: 0.541
[4,  3500] loss: 0.528
[4,  4000] loss: 0.559
[4,  4500] loss: 0.542
[4,  5000] loss: 0.593
[4,  5500] loss: 0.561
[4,  6000] loss: 0.537
[5,   500] loss: 0.492
[5,  1000] loss: 0.501
[5,  1500] loss: 0.481
[5,  2000] loss: 0.476
[5,  2500] loss: 0.514
[5,  3000] loss: 0.466
[5,  3500] loss: 0.490
[5,  4000] loss: 0.500
[5,  4500] loss: 0.501
[5,  5000] loss: 0.479
[5,  5500] loss: 0.492
[5,  6000] loss: 0.477
[6,   500] loss: 0.429
[6,  1000] loss: 0.459
[6,  1500] loss: 0.401
[6,  2000] loss: 0.449
[6,  2500] loss: 0.426
[6,  3000] loss: 0.417
[6,  3500] loss: 0.466
[6,  4000] loss: 0.474
[6,  4500] loss: 0.464
[6,  5000] loss: 0.485
[6,  5500] loss: 0.451
[6,  6000] loss: 0.424
Finished Training
```

Now we see how we can use this model for the first 5 labels.

```
# We define a new model to fine-tune
model3 = model2
```

We have a new model: model3. This model has the same weights as model2, hence it is good at classifying labels, 6-10, and does not know about labels 1-5. We will have to reset and retrain the last layer.

Problem 3

Let's pretend that didn't build model3 ourselves. In this case, to reset the last layer, we would need to figure out what its name is (we know it's fc3, but we're pretending we don't).

Find a command that will tell you the name and type of the last layer. Print them out.

```
for name, layer in model3.named_modules():
    print(name, layer)
```

```
ConvNet(
  (relu): ReLU()
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

```
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
relu ReLU()
conv1 Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
pool MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
conv2 Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
fc1 Linear(in_features=400, out_features=120, bias=True)
fc2 Linear(in_features=120, out_features=84, bias=True)
fc3 Linear(in_features=84, out_features=10, bias=True)
```

ANSWER: We can see above that the last layer is called fc3 and is a linear layer

Problem 4

If we did the above correctly, then we learned that the last layer is called fc3 and it is a linear layer.

We want to redefine it giving a command like

```
model3.fc3 = nn.Linear(input_features,output_features)
```

- What should be the value of output_features for our new last layer that we are defining? The answer to this comes from the problem we want to solve.
- What is the number of input features that go into this last layer? You'll have to look at the size of the layer defined in the network. Again, pretend we didn't make the network, and we just downloaded it.

So in other words, I'm asking you to use commands to find out how many input features there are for the last layer which (we know from above) is called fc3.

ANSWER: output_features should be 5 because we are predicting for 5 classes. The number of input features is the number of output features of the previous layer fc2.

Problem 5

Now redefine the last layer. You will do this with a command like

```
model3.fc3 = # TO DO
```

Note how important it is for us to know what the names of the layers are.

```
model3.fc3 = nn.Linear(model3.fc2.out_features, 5)
model3 = model3.to(device)
model3.train()

# Let's again check the layers and see the change we made
# (doesn't actually look different since we split 5 and 5)
for name, layer in model2.named_modules():
  print(name,layer)
```

```
ConvNet(
  (relu): ReLU()
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=5, bias=True)
)
relu ReLU()
conv1 Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
pool MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
conv2 Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
fc1 Linear(in_features=400, out_features=120, bias=True)
fc2 Linear(in_features=120, out_features=84, bias=True)
fc3 Linear(in_features=84, out_features=5, bias=True)
```

```
# Set the optimizer (loss function already set)
optimizer = torch.optim.SGD(model3.parameters(), lr=0.001, momentum=0.9)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Problem 6

Check that model3 is no good for classes 1-5. (It shouldn't be any good! We just added a randomly initialized last layer). The accuracy should be close to 20%, which is how good random guessing would be for 5 labels.

Evaluate its accuracy on the validation set for labels 1-5.

ANSWER: Model 3 accuracy before training: 16.51%

```
model3.eval()
correct = 0
total = 0
for images, labels in data15_val:
  outputs = model3(images)
  val, predicted = torch.max(outputs, 1)  # highest predicted class
  total += labels.size(0)
  correct += (predicted == labels).sum().item() # count number of correct
print(f'Accuracy: {100 * correct / total:.2f}%')
```

```
Accuracy: 16.51%
```

```
# Now we train model3 on the first five labels, and evaluate accuracy.
# Nothing for you to do here but run this cell.
# Note that we are also just using one single pass, to make it a fair
# comparison, and so that we can try to see the benefit of transfer learning.
#

for epoch in range(1):  # one single pass -- don't change this

    running_loss = 0.0
    for i, data in enumerate(data15_train, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model3(inputs)
        loss = loss_function(outputs, labels)
        loss.backward()
        optimizer.step()

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

print('Finished Training')
```

```
[1,   200] loss: 1.182
[1,   400] loss: 1.043
[1,   600] loss: 1.036
[1,   800] loss: 1.027
[1,  1000] loss: 1.002
[1,  1200] loss: 0.991
[1,  1400] loss: 0.920
[1,  1600] loss: 0.917
[1,  1800] loss: 0.950
[1,  2000] loss: 0.935
[1,  2200] loss: 0.924
[1,  2400] loss: 0.902
[1,  2600] loss: 0.896
[1,  2800] loss: 0.862
[1,  3000] loss: 0.888
[1,  3200] loss: 0.808
```

```
[1,  3400] loss: 0.869
[1,  3600] loss: 0.915
[1,  3800] loss: 0.925
[1,  4000] loss: 0.838
[1,  4200] loss: 0.846
[1,  4400] loss: 0.846
[1,  4600] loss: 0.862
[1,  4800] loss: 0.799
[1,  5000] loss: 0.829
[1,  5200] loss: 0.814
[1,  5400] loss: 0.839
[1,  5600] loss: 0.841
[1,  5800] loss: 0.890
[1,  6000] loss: 0.753
[1,  6200] loss: 0.815
Finished Training
```

Problem 7

Evaluate the accuracy of model3 (after 1 epoch of training) on the validation set. Compare with the accuracy you found above in Problem 2.

ANSWER: Model 3 accuracy after training: 70.07% which is about 8% higher than the one found in problem 2

```
model3.eval()
correct = 0
total = 0
for images, labels in data15_val:
  outputs = model3(images)
  val, predicted = torch.max(outputs, 1)  # highest predicted class
  total += labels.size(0)
  correct += (predicted == labels).sum().item() # count number of correct
print(f'Accuracy: {100 * correct / total:.2f}%')
```

```
Accuracy: 70.07%
```

Data Science Lab: Lab 7 part III (of III)

Submit:

A pdf of your notebook with solutions. A link to your colab notebook or also upload your .ipynb if not working on colab.

Goals of this Lab: Fine Tuning and Transfer Learning

This assignment completes our segment on convolutional networks, computer vision and transfer learning. The goal of this colab notebook and lab is to get down the basics of Transfer learning, and harnessing the power of pre-trained notebooks, in a more substantial setting than CIFAR-10. Along the way, we get experience with creating and labeling data sets, and a number of other Python tools.

Specifically, we do the following:

- In this colab notebook, we create our own labeled image data set, with calls to the Bing API. We use this to download images with user-specified labels, into train and test directories.
- We preprocess all the images and use the directory names as lables to create our training and testing data sets.
- Then we download a pre-trained convolutional neural network from Pytorch. There is wide selection here. These have been trained on ImageNet. See references below.
- Then we add a new last layer, and train. Note that the last layer we add has to have the right size, namely, the number of classes in our data.
- Things to play with: Choosing different pre-trained models; Fine tuning the entire network vs freezing and fine tuning only the last layer or layers; Possibly adding more of our own layers at the end. (See the last optional problem at the end).

Some references:

The bing image downloader package from here https://pypi.org/project/bing-image-downloader/ (https://pypi.org/project/bing-image-downloader/)

The following will allow us to call the Bing Image Downloader

In [1]:

```
!pip install bing-image-downloader

```

```
Requirement already satisfied: bing-image-downloader in c:\users\tonys\anaconda3\envs\gpu_
pytorchenv\lib\site-packages (1.1.2)
```

In [6]:

```
from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import time
import os
import copy
import requests
from bing_image_downloader import downloader
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random
import math

plt.ion()   # interactive mode

```

Out[6]:

```
<contextlib.ExitStack at 0x1fb263f53a0>
```

Downloading the images we need

In [4]:

```python
# These are the functions we need for asking for query terms, number of images
# and the fraction of train/test split
def get_query_terms():
    """Ask the user for a list of query terms."""
    queries = input("Enter query terms separated by commas: ").split(',')
    return [query.strip() for query in queries]


def get_positive_integers():
    """Ask the user for a list of positive integers."""
    while True:
        numbers = input("Enter positive integers corresponding to each query, separated
        try:
            # Convert string inputs to integers
            numbers = [int(num.strip()) for num in numbers]

            # Check if all numbers are positive
            if all(num > 0 for num in numbers):
                return numbers
            else:
                print("All numbers must be positive. Please try again.")
        except ValueError:
            print("Invalid input. Please enter integers only.")


def get_train_test_proportion():
    """Ask the user for a number between 0 and 1 and ensure it is in that range."""
    while True:
        try:
            # Ask the user for a number and try to convert it to a float
            num = float(input("What fraction of the data should be training? For exampl

            # Check if the number is between 0 and 1 (inclusive)
            if 0 <= num <= 1:
                return num
            else:
                print("The number must be between 0 and 1. Please try again.")
        except ValueError:
            print("Invalid input. Please enter a valid number between 0 and 1.")
```

Problem 0

Run the above code, and the code below, to make your own data set. Choose terms that aren't already one of the 1000 classes in the Imagenet dataset.

I recommend starting small: 2 or 3 different terms, and maybe 100-200 images max from each.

In [8]:

```
import shutil

queries = get_query_terms()
# And the number of each
numbers = get_positive_integers()
# And the fraction of training data
frac_train = get_train_test_proportion()
# Check if number of queries matches number of positive integers
if len(queries) != len(numbers):
  print("Number of queries and integers do not match!")

for query, n in zip(queries, numbers):
    # Download all images to a common directory
    downloader.download(query, limit=n, output_dir='dataset/all_images', adult_filter_o

    # Directory paths
    source_dir = f'dataset/all_images/{query}'
    train_dir = f'dataset/train/{query}'
    test_dir = f'dataset/test/{query}'

    # Create train and test directories if they don't exist
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)

    # Get all downloaded files
    all_files = [os.path.join(source_dir, f) for f in os.listdir(source_dir)]
    random.shuffle(all_files)  # Shuffle to randomize distribution

    # Determine split counts
    n_train = int(math.floor(len(all_files) * frac_train))
    train_files = all_files[:n_train]
    test_files = all_files[n_train:]

    # Move files to respective directories
    for f in train_files:
        shutil.move(f, train_dir)
    for f in test_files:
        shutil.move(f, test_dir)

    # Optionally, remove the source directory if it's now empty
    if not os.listdir(source_dir):
        os.rmdir(source_dir)

```

```
Enter query terms separated by commas: 'vintage_typewriter', 'antique_compass'
Enter positive integers corresponding to each query, separated by commas: 150,150
What fraction of the data should be training? For example, enter 0.75 for 75 percent: 0.8
[%] Downloading Images to C:\Users\tonys\Desktop\Fall 2024\Data Science Lab\Lab homework\d
ataset\all_images\'vintage_typewriter'
[Error]Invalid image, not saving https://preview.redd.it/up2w6modfti51.jpg?auto=webp&amp;s
=7e393f006607bf946bd5f741b9f8819de5a30070 (https://preview.redd.it/up2w6modfti51.jpg?auto=
webp&amp;s=7e393f006607bf946bd5f741b9f8819de5a30070)

[!] Issue getting: https://preview.redd.it/up2w6modfti51.jpg?auto=webp&amp;s=7e393f006607b
f946bd5f741b9f8819de5a30070 (https://preview.redd.it/up2w6modfti51.jpg?auto=webp&amp;s=7e3
93f006607bf946bd5f741b9f8819de5a30070)
[!] Error:: Invalid image, not saving https://preview.redd.it/up2w6modfti51.jpg?auto=webp&
amp;s=7e393f006607bf946bd5f741b9f8819de5a30070 (https://preview.redd.it/up2w6modfti51.jpg?
auto=webp&amp;s=7e393f006607bf946bd5f741b9f8819de5a30070)

[Error]Invalid image, not saving https://i.redd.it/2zl58ghgfl421.jpg (https://i.redd.it/2z
l58ghgfl421.jpg)

[!] Issue getting: https://i.redd.it/2zl58ghgfl421.jpg (https://i.redd.it/2zl58ghgfl421.jp
g)
[!] Error:: Invalid image, not saving https://i.redd.it/2zl58ghgfl421.jpg (https://i.redd.
it/2zl58ghgfl421.jpg)

[!] Issue getting: https://c1.wallpaperflare.com/preview/962/977/65/typewriter-vintage-old
-vintage-typewriter.jpg (https://c1.wallpaperflare.com/preview/962/977/65/typewriter-vinta
ge-old-vintage-typewriter.jpg)
[!] Error:: <urlopen error [SSL] record layer failure (_ssl.c:1133)>
[Error]Invalid image, not saving https://images.squarespace-cdn.com/content/v1/5ca223529d4
14920186973a5/1678120691582-VWX48PM0BMGJCUVO6BXA/Remington+Model+5+Vintage+Manual+Typewrit
er.jpg (https://images.squarespace-cdn.com/content/v1/5ca223529d414920186973a5/16781206915
82-VWX48PM0BMGJCUVO6BXA/Remington+Model+5+Vintage+Manual+Typewriter.jpg)

[!] Issue getting: https://images.squarespace-cdn.com/content/v1/5ca223529d414920186973a5/
1678120691582-VWX48PM0BMGJCUVO6BXA/Remington+Model+5+Vintage+Manual+Typewriter.jpg (http
s://images.squarespace-cdn.com/content/v1/5ca223529d414920186973a5/1678120691582-VWX48PM0B
MGJCUVO6BXA/Remington+Model+5+Vintage+Manual+Typewriter.jpg)
[!] Error:: Invalid image, not saving https://images.squarespace-cdn.com/content/v1/5ca223
529d414920186973a5/1678120691582-VWX48PM0BMGJCUVO6BXA/Remington+Model+5+Vintage+Manual+Typ
ewriter.jpg (https://images.squarespace-cdn.com/content/v1/5ca223529d414920186973a5/167812
0691582-VWX48PM0BMGJCUVO6BXA/Remington+Model+5+Vintage+Manual+Typewriter.jpg)

[Error]Invalid image, not saving https://preview.redd.it/ypnikseb7u431.jpg?auto=webp&amp;s
=0c7b3fcd9087ed8cb6c9eb257e68e9d8f84b031e (https://preview.redd.it/ypnikseb7u431.jpg?auto=
webp&amp;s=0c7b3fcd9087ed8cb6c9eb257e68e9d8f84b031e)

[!] Issue getting: https://preview.redd.it/ypnikseb7u431.jpg?auto=webp&amp;s=0c7b3fcd9087e
d8cb6c9eb257e68e9d8f84b031e (https://preview.redd.it/ypnikseb7u431.jpg?auto=webp&amp;s=0c7
b3fcd9087ed8cb6c9eb257e68e9d8f84b031e)
[!] Error:: Invalid image, not saving https://preview.redd.it/ypnikseb7u431.jpg?auto=webp&
amp;s=0c7b3fcd9087ed8cb6c9eb257e68e9d8f84b031e (https://preview.redd.it/ypnikseb7u431.jpg?
auto=webp&amp;s=0c7b3fcd9087ed8cb6c9eb257e68e9d8f84b031e)



[%] Done. Downloaded 150 images.
[%] Downloading Images to C:\Users\tonys\Desktop\Fall 2024\Data Science Lab\Lab homework\d
ataset\all_images\'antique_compass'
[!] Issue getting: http://www.geodus.com/globe-map/tresgrand/am_CO012.jpg (http://www.geod
us.com/globe-map/tresgrand/am_CO012.jpg)
[!] Error:: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: una
ble to get local issuer certificate (_ssl.c:1133)>
[!] Issue getting: https://churchantiques.com/wp-content/uploads/2021/01/IMG_20201123_1636
38.jpg (https://churchantiques.com/wp-content/uploads/2021/01/IMG_20201123_163638.jpg)
[!] Error:: HTTP Error 404: Not Found
[!] Issue getting: https://www.craple.com/wp-content/uploads/2023/03/Collecting-Antique-Co
```

```
mpasses.webp (https://www.craple.com/wp-content/uploads/2023/03/Collecting-Antique-Compass
es.webp)
[!] Error:: Remote end closed connection without response
[!] Issue getting: http://www.scientificcollectables.com/PHOTOS (http://www.scientificcoll
ectables.com/PHOTOS) 2019/sc2302c.jpg
[!] Error:: URL can't contain control characters. '/PHOTOS 2019/sc2302c.jpg' (found at lea
st ' ')
[!] Issue getting: https://media.istockphoto.com/photos/antique-compass-picture-id17354383
7 (https://media.istockphoto.com/photos/antique-compass-picture-id173543837)
[!] Error:: HTTP Error 400: Bad Request
[!] Issue getting: https://www.crushpixel.com/big-static11/preview4/old-vintage-compass-on
-ancient-648094.jpg (https://www.crushpixel.com/big-static11/preview4/old-vintage-compass-
on-ancient-648094.jpg)
[!] Error:: HTTP Error 403: Forbidden
[Error]Invalid image, not saving http://www.windandweather.com/medias/sys_master/images/im
ages/hb7/hc0/8838039896094/IN7303SInglesx.jpg (http://www.windandweather.com/medias/sys_ma
ster/images/images/hb7/hc0/8838039896094/IN7303SInglesx.jpg)

[!] Issue getting: http://www.windandweather.com/medias/sys_master/images/images/hb7/hc0/8
838039896094/IN7303SInglesx.jpg (http://www.windandweather.com/medias/sys_master/images/im
ages/hb7/hc0/8838039896094/IN7303SInglesx.jpg)
[!] Error:: Invalid image, not saving http://www.windandweather.com/medias/sys_master/imag
es/images/hb7/hc0/8838039896094/IN7303SInglesx.jpg (http://www.windandweather.com/medias/s
ys_master/images/images/hb7/hc0/8838039896094/IN7303SInglesx.jpg)

[Error]Invalid image, not saving https://www.nauticalia-trade-sales.com/index.php?route=pr
oduct/product/saveImage&amp;image=image/catalog/products_images/4429.jpg (https://www.naut
icalia-trade-sales.com/index.php?route=product/product/saveImage&amp;image=image/catalog/p
roducts_images/4429.jpg)

[!] Issue getting: https://www.nauticalia-trade-sales.com/index.php?route=product/product/
saveImage&amp;image=image/catalog/products_images/4429.jpg (https://www.nauticalia-trade-s
ales.com/index.php?route=product/product/saveImage&amp;image=image/catalog/products_image
s/4429.jpg)
[!] Error:: Invalid image, not saving https://www.nauticalia-trade-sales.com/index.php?rou
te=product/product/saveImage&amp;image=image/catalog/products_images/4429.jpg (https://ww
w.nauticalia-trade-sales.com/index.php?route=product/product/saveImage&amp;image=image/cat
alog/products_images/4429.jpg)

[!] Issue getting: https://antiqueitemstore.com/wp-content/uploads/2020/04/5-6-644x1024.jp
g (https://antiqueitemstore.com/wp-content/uploads/2020/04/5-6-644x1024.jpg)
[!] Error:: Remote end closed connection without response


[%] Done. Downloaded 150 images.
```

Let's make sure we downloaded enough of each

The code above is just making calls to the Bing API, and as you'll see there are a number of errors. So we should make sure that we have in fact downloaded the images we need. Print the number of files

In [10]:

```python
import os

def count_files_in_directory(directory, prefix=""):
    """
    Recursively counts the number of files in each directory and subdirectory.

    Args:
    - directory: The directory path to start counting from.
    - prefix: A string used for indentation to visualize the folder structure.
    """
    num_files = sum([len(files) for r, d, files in os.walk(directory)])
    print(f"{prefix}{os.path.basename(directory)}: {num_files} files")

    for subdir in next(os.walk(directory))[1]:  # List subdirectories of the current di
        path = os.path.join(directory, subdir)  # Full path of the subdirectory
        count_files_in_directory(path, prefix + "  ")  # Recursively count in this subd

# Path to the dataset directory
dataset_dir = 'C:\\Users\\tonys\\Desktop\\Fall 2024\\Data Science Lab\\Lab homework\\da

# Start the recursive count
count_files_in_directory(dataset_dir)

```

```
dataset: 312 files
  all_images: 0 files
  test: 63 files
    'antique_compass': 30 files
    'vintage_typewriter': 33 files
  train: 249 files
    'antique_compass': 120 files
    'vintage_typewriter': 129 files
```

Preprocessing the data

Now that we've downloaded the data we want, we do some basic preprocessing with functions from Torchvision.

For the training data, we normalize and also use some data augmentation.

For the validation set, we just normalize.

Problem 1 (Nothing to turn in)

Read about this here: https://pytorch.org/vision/stable/transforms.html (https://pytorch.org/vision/stable/transforms.html)

Make dataloaders out of our training and testing data

In [11]:

```
# Data augmentation and normalization for training
# Just normalization for validation

# The directory that contains the data
data_dir = 'dataset'

# Now we apply the usual transformations
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'test': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'test']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                              shuffle=True, num_workers=2)
               for x in ['train', 'test']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'test']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Problem 2

Write code that calls the dataloaders and displays some number of images, with their true labels.

In [12]:

```
import matplotlib.pyplot as plt

def imshow(inp, title=None):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit to update plots

# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

#Make a grid from batch
out = torchvision.utils.make_grid(inputs)

# Visualize
imshow(out, title=[class_names[x] for x in classes])

```

Download a pre-trained model

We will now download a pretrained model that has been trained on Imagenet. I recomend starting with Resnet18. See here for other larger or smaller pre-trained modfels https://pytorch.org/vision/stable/models.html (https://pytorch.org/vision/stable/models.html)

In [13]:

```
# Now we load a pre-trained model, and replace the last layer with a non-initialized li

# This command loads a pre-trained model
model_ft = models.resnet18(weights = models.ResNet18_Weights.DEFAULT)

```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to C:\Users\tony
s/.cache\torch\hub\checkpoints\resnet18-f37072fd.pth
100%|████████████████████████████████████████████████████████████████| 44.7M/
44.7M [00:01<00:00, 40.2MB/s]
```

Problem 3

Redefine the last layer of your neural network (called model_ft) to be a linear (fully connected layer) whose input size is whatever the input size is of the the current last layer, and whose output size is the number of labels your new data set has.

You need to figure out how to do this. It's not too difficult. You need will use similar commands as you used in the last 2 homeworks in order to define a linear layer. The key will be to find out what the last layer is called, and also to find

In [14]:
```
num_features = model_ft.fc.in_features  # Get input size of the last layer
num_classes = len(class_names)  # Number of classes in your dataset

# Replace the last fully connected layer
model_ft.fc = nn.Linear(num_features, num_classes)

```

In [26]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft = model_ft.to(device)
```

In [ ]:
```

```

Training

- We are going to fine-tune by training all the layers.
- We can also freeze the old layers, and only update the last layers that we added. (This is not implemented below.)

In [28]:

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'test']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step() # updates the learning rate

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase == 'test' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best test Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
```

```
67      return model
```

In [29]:

```python
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = model.to(device)  # Move model to GPU
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        for phase in ['train', 'test']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()  # Set model to evaluation mode

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)  # Move inputs to GPU
                labels = labels.to(device)  # Move labels to GPU

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)  # Forward pass
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()  # Backpropagation
                        optimizer.step()  # Optimize weights

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

            if phase == 'test' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
    print(f'Best test Acc: {best_acc:.4f}')

    model.load_state_dict(best_model_wts)
    return model

```

Parameters of the optimization

Problem 4 (Optional)

Explore choosing different learning rates or optimizers to see how things go.

In [30]:
```
# We need to set the loss function, the optimizer, and the learning rate scheduler
criterion = nn.CrossEntropyLoss()
# though we're fine tuning, we're updating all parameters
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
# the scheduler decreases the learning rate by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

Train

In [31]:
```
# Now we train
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,num_epochs=2
```

```
Epoch 0/24
----------
train Loss: 0.4797 Acc: 0.7751
test Loss: 0.0168 Acc: 1.0000

Epoch 1/24
----------
train Loss: 0.4086 Acc: 0.8835
test Loss: 0.0281 Acc: 0.9841

Epoch 2/24
----------
train Loss: 0.2769 Acc: 0.9157
test Loss: 0.0614 Acc: 0.9683

Epoch 3/24
----------
train Loss: 0.2740 Acc: 0.8996
test Loss: 0.0466 Acc: 0.9841
```

In [18]:
```

```

Problem 5

Report your accuracy on the testing set. You should compute this explicitly by running all your testing examples through the model and checking them against their true labels.

In [32]:

```python
def evaluate_model(model, dataloader):
    model.eval()  # Set to evaluation mode
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (preds == labels).sum().item()

    accuracy = correct / total
    print(f"Test Accuracy: {accuracy:.4f}")
    return accuracy

evaluate_model(model_ft, dataloaders['test'])

```

```
Test Accuracy: 1.0000
```

Out[32]: 1.0

Now let's visualize to see how well we did

Problem 6

Print out images from the test set, reporting the predicted labels and the true labels.

In [33]:

```python
def visualize_predictions(model, dataloader, num_images=6):
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for i in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images // 2, 2, images_so_far)
                ax.axis('off')
                ax.set_title(f'True: {class_names[labels[i]]} \nPred: {class_names[pred
                imshow(inputs.cpu().data[i])

                if images_so_far == num_images:
                    return

visualize_predictions(model_ft, dataloaders['test'])
```

True: 'vintage_typewriter'
Pred: 'vintage_typewriter'

True: 'antique_compass'
Pred: 'antique_compass'

True: 'antique_compass'
Pred: 'antique_compass'

True: 'vintage_typewriter'
Pred: 'vintage_typewriter'

True: 'antique_compass'
Pred: 'antique_compass'

True: 'antique_compass'
Pred: 'antique_compass'

Problem 7 (Optional)

Experiment with some/all of the following:

- Play with different pre-trained models
- Experiment with different pre-processing of the data (e.g., turn data augmentation on or off).
- Experiment with adding more/fewer layers, and/or layers of different size.
- Fine tune by training everything or only the last (new) layer -- this requires figuring out how to only update some of the layers.
- Try to reduce the number of images you use for training. How few can you use and still get good accuracy? Remember that you were all able to learn what a Goblin shark is with only one single example.

In [ ]:

```
1
```