Cifar Lab

Dr. Denton Bobeldyk

In this lab, you will use pytorch to create and train a CNN on the CIFAR-10 dataset. It is recommended you select a group member's laptop to use for this lab. You may also ssh into the EOS machines to do the lab.

Before we begin, it is important to familiarize yourself with the dataset you will be working with.

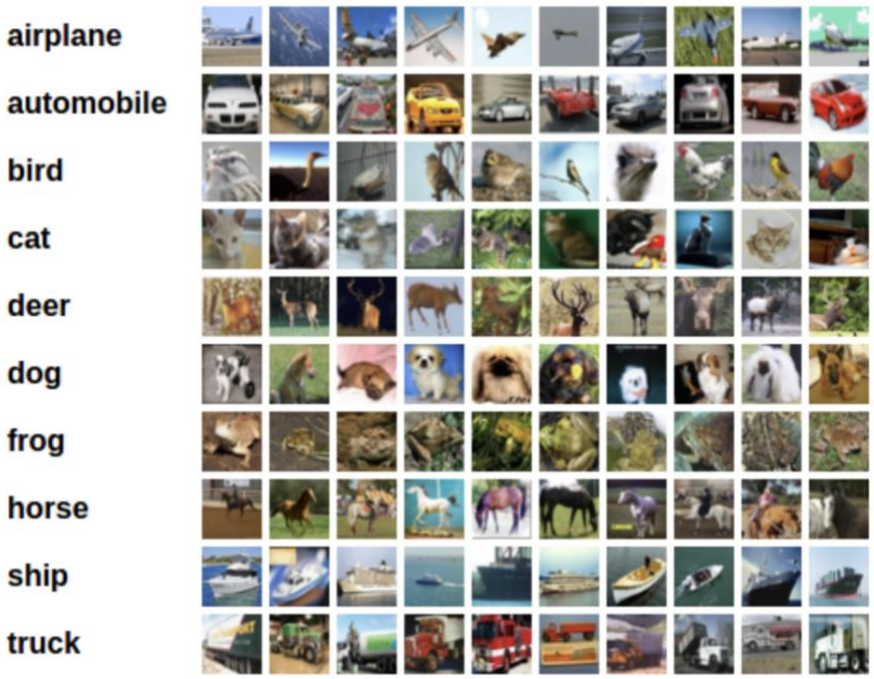
There are 10 classes in the CIFAR-10 dataset:

'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'

The images in CIFAR-10 are of size 3x32x32.

3 represents the color channels (RGB)

32x32 represents the width and height (respectively)



1. Create a new python script named 'cifar10.py'. For this lab, as you add commands to the script, run the script and observe the output.

2. For each of the 'Questions', answer them in a separate document that you will submit to blackboard as a group.

3. Add the following import statements for the modules you will need for this lab:

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
import multiprocessing

def main():
        #insert all new commands here

If __name__ == '__main__':
        multiprocessing.freeze_support()
        print("Lab One")
        main()
```

4. Add a print "Lab One" to your script and ensure all of the modules are available. If not, you will need to install them using the 'pip install' command. Call the instructor over for assistance with this or feel free to search up instructions online

5. In order to download the CIFAR10 dataset from the following url: http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

   You will need to add the following line to override the default download (which uses ssl and may cause a certificate issue):

   torchvision.datasets.CIFAR10.url=http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

6. Next we need to set the transform which normalizes the images:
   transform = transforms.Compose(
   [transforms.ToTensor(),
   transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

   Question 1) What do the 0.5's represent?

7. For this lab, we're going to use a batch size of 4

batch_size = 4

8. The following lines will download the dataset as well as define the train and test set

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

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

9. CIFAR-10 uses 10 default classes which need to be assigned below:
   classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

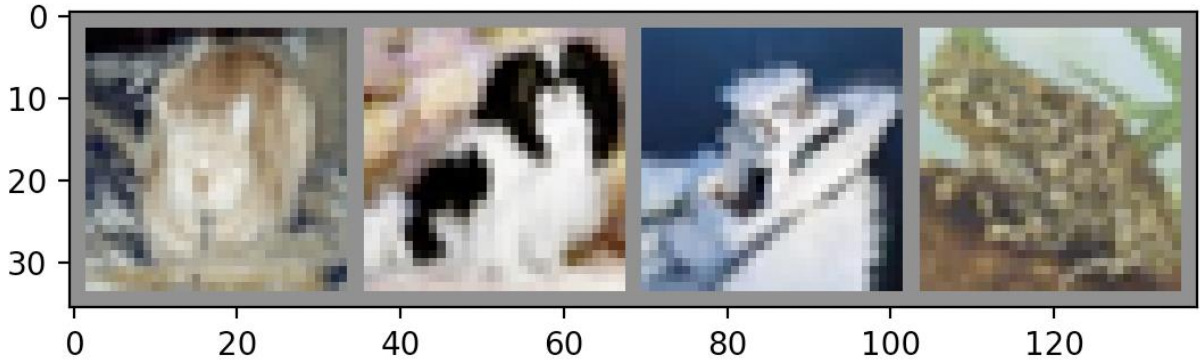
10. We need to create a helper function to display the image:

```
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
```

11. Load up some random training images that can be displayed:
    dataiter = iter(trainloader)
    images, labels = next(dataiter)

12. Display the images using the imshow command:
    imshow(torchvision.utils.make_grid(images))

    You should see something like below:



13. Print out the labels:
    print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))

    cat dog ship frog

Now that we're familiar with the dataset, let's create the convolutional neural network that will learn to predict the labels for the images.

14. Add the following code to your script:

```
class Net(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
```

net = Net()

Question 2) What do the arguments in the conv2d represent?
Question 3) What do the arguments in the MaxPool represent?
Question 4) What do the arguments in the Linear represent?
Question 5) What is the flatten layer doing?

15. Let's define the loss function and use momentum with stochastic gradient descent

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

16. Now the network is setup for training. We simply loop over the data iterator and feed the inputs to the network and optimize.

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

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

        # zero the parameter gradients
        optimizer.zero_grad()

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

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

print('Finished Training')

17. Now that the model has been trained and resides in your system memory, you can save it for future use so as to not require training again.

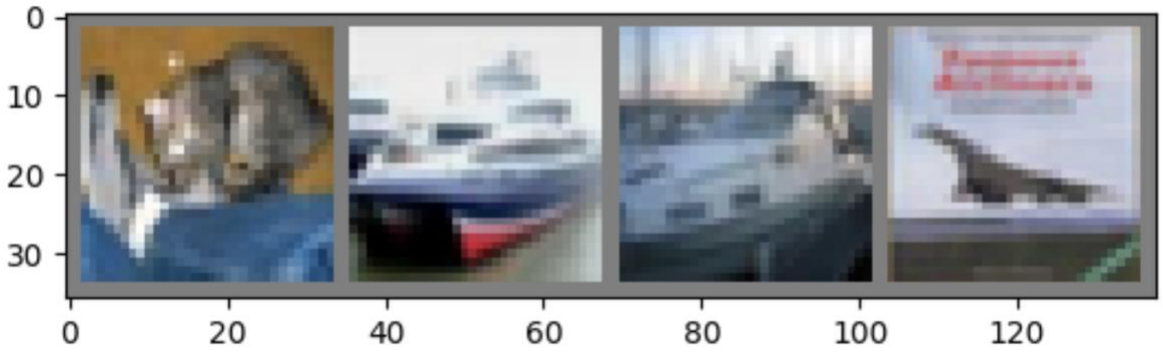
    PATH = './cifar_net.pth'
    torch.save(net.state_dict(), PATH)

    Run your script again and ensure the 'cifar_net.pth' file is there.

18. Now that you have a trained model, it's time to test the network on the test data. Before we do that, let's take a look at some of the test images:
    dataiter = iter(testloader)
    images, labels = next(dataiter)

    # print images
    imshow(torchvision.utils.make_grid(images))

    You should see something similar to:



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

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

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

```
outputs = net(images)
```

20. The outputs are energies for the 10 classes. The higher the energy for a class, the more the network thinks that the image is of the particular class. So, let's get the index of the highest energy:

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

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

Output shown below:

```
cat      ship     truck    ship
```

21. Let us look at how the network performs on the whole dataset.

```
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```

You should see something similar to:

```
Accuracy of the network on the 10000 test images: 54 %
```

```
Accuracy for class: plane is 54.9 %
Accuracy for class: car   is 62.3 %
Accuracy for class: bird  is 41.3 %
Accuracy for class: cat   is 19.7 %
Accuracy for class: deer  is 31.6 %
Accuracy for class: dog   is 59.0 %
Accuracy for class: frog  is 75.6 %
Accuracy for class: horse is 55.9 %
Accuracy for class: ship  is 74.9 %
Accuracy for class: truck is 71.9 %
```

22. The results look promising and way better than chance. Chance would be 10% accuracy (randomly picking a class out of 10 classes).

23. It is good to further analyze the results to determine which classes are performing well and which ones aren't.

```
# prepare to count predictions for each class
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

# again no gradients needed
with torch.no_grad():
   for data in testloader:
      images, labels = data
      outputs = net(images)
      _, predictions = torch.max(outputs, 1)
      # collect the correct predictions for each class
      for label, prediction in zip(labels, predictions):
         if label == prediction:
            correct_pred[classes[label]] += 1
         total_pred[classes[label]] += 1
```

```
# print accuracy for each class
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

Question 6:

Create a separate script file to load in the saved model and do the above prediction. This simulates training a model (which can take a VERY long time) and using it for future use later (like a production environment).

You can use the following to load the saved model:

```
saveLocation = './cifar_net.pth'

net.load_state_dict(torch.load(saveLocation))
```

Copy and paste each of the scripts into the word document for this 'answer'

Question 7:

Modify the above code to only learn 5 classes

Copy and paste your modified script for question 7.

Question 8:

Modify the above code to only learn 2 classes

Copy and paste your modified script for question 8

Question 9:

Repeat the above exercise but using the CIFAR-100 dataset located here:

https://www.cs.toronto.edu/~kriz/cifar.html

Copy and paste the code

Extra:

Can you improve the performance or your neural network by changing the layers/parameters of the model?

Which group was able to achieve the best performance?