02-620 HW4 Programming - Image classification using Pytorch

In this last homework, we will learn basic deep learning. Deep learning is a fast moving research area. There is ongoing research on why it works so well, but so far the theory of deep learning is still building in progress. Due to this nature of the field, hands-on experience is the most important in deep learning.

We will use Pytorch, one of the popular deep learning frameworks. If you are unfamiliar with the Pytorch, please watch the recitation video and look at the recitation material, where we covered basic understanding of Pytorch.

Because our course is an introductory machine learning course, we covered broad topics in machine learning and thus we can't go very deep into deep learning. So we will go through only the basics of deep learning. Throughout the homework, we hope you'll get familiar with deep learning and Pytorch framework.

Referenced 16-720 and 16-824, great courses in CMU

```
#Feel free to import any required library you need.
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
from torchvision.transforms import ToTensor
import glob
from torchvision import transforms
import seaborn as sns
import numpy as np

import matplotlib.pyplot as plt
import os
import pandas as pd
torch.manual_seed(0) #For reproducibility
```

```
    <torch._C.Generator at 0x7f7d38b00310>
```

General Task description

In this assignment, our task is to build a workflow of classifying the objects using deep learning models. The dataset we are interested in is CIFAR10(https://www.cs.toronto.edu/~kriz/cifar.html). This dataset includes 60,000 32x32 color images in 10 clases. It has 50,000 training samples and 10,000 test samples. You can download the dataset using torchvision.datasets.CIFAR10()

TODOs

1. Change path to your desired path!!!
2. Once you download your data, you can change it to False

```
train_transform = transforms.Compose([
    transforms.Resize((256)),
    transforms.ToTensor(),
    transforms.RandomRotation(degrees=(0, 180)),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

test_transform = transforms.Compose([
    transforms.Resize((256)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])


# use this one
# path = "./"
# download=True
# trainset = torchvision.datasets.CIFAR10(root = path, train=True,download=download,
# testset = torchvision.datasets.CIFAR10( root = path ,train=False, download=downloa
```

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

```
# For task 1
path = "./"
download=True
trainset = torchvision.datasets.CIFAR10(root = path, train=True,download=download, t
testset = torchvision.datasets.CIFAR10( root = path ,train=False, download=download,
```

```
    Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./cifar
    100%|██████████| 170498071/170498071 [00:05<00:00, 29387067.54it/s]
    Extracting ./cifar-10-python.tar.gz to ./
    Files already downloaded and verified
```

Task 1 Build your own DataLoader(10 Points)

As we covered in the recitation, Pytorch uses the DataLoader class to bring datapoints to the neural network. You can do any preprocessing in this step.

Take a look at https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

There can be possibility that you don't have enough dataset. In this case, we can populate the dataset using data augmentation. For example, in Computer Vision, one way to augment your data is adding rotated images.

BE CAREFUL: In Pytorch, one of the argument in the DataLoader is indicating train or test. This is very important argument because you do data augmentation with training samples, but not with test samples. You should consider this when you implement DataLoader.

Tip:

1. It is very important to understand what is your data and how they are organized. Observe how folder and files strutured in the downloaded CIFAR10. See the annotation files, open the images, try to see if there is any pattern in file name, etc.
2. For data augmentation, feel free to use methods in torchvision.transforms

TODOs

1. Form your own DataLoader
2. In the **getitem**, add your own data augmentation.
3. You should have an variable that controls size of the image. For this assignment, use size of 256: that is, image should be (256, 256, 3). Please be mind that when you load this image to the Tensor, it might be changed to (3, 256, 256)

```
batch_size = 64

# DataLoader from the package
train_dataloader = DataLoader(trainset, batch_size=batch_size)
test_dataloader = DataLoader(testset, batch_size=batch_size)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

```
Shape of X [N, C, H, W]: torch.Size([64, 3, 256, 256])
Shape of y: torch.Size([64]) torch.int64
```

```
# # Trying Task 1
# (tried task 1, could not finish it)

# class CustomImageDataset(Dataset):
#     def __init__(self, trainset.targets, trainset.data, transform=None, target_tra
#         self.img_labels = trainset.target
#         self.img_dir = trainset.data
#         self.transform = transform

#     def __len__(self):
#         return len(self.img_labels)

#     def __getitem__(self, idx):
#         img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
#         image = read_image(img_path)
#         label = self.img_labels.iloc[idx, 1]
#         if self.transform:
#             image = self.transform(image)
#         if self.target_transform:
#             label = self.target_transform(label)
#         return image, label
```

Now you should be able to bring and use your Dataloader

```
# continue task 1
# loading it through custom data loader

# newTrainSet = CustomImageDataset(trainset.targets, trainset.data, transform = trai
# newTestSet = CustomImageDataset(testset.targets, testset.data, transform = test_tr
# augTrainSet = CustomImageDataset(trainset.targets, trainset.data, transform = augT


# used data loader from the package to continue with the rest of the tasks in the as
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,shuffle=True, num_
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False, num_
```

Task 2 Build Neural Network(5 Points)

If you look at some of the deep learning papers, they provide their model architecture, which is how they organized their neural network. Through this task, you should be able to rebuild the neural network given model architecture. The model we are interested in is AlexNet(https://arxiv.org/abs/1404.5997). For simplicity, we are going to implement a slightly simplified version.

This is the model you should rebuild:

AlexNet(

(features): Sequential(

```
(0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
(1): ReLU(inplace)
(2): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
(3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
(4): ReLU(inplace)
(5): MaxPool2d(kernel_size=(3, 3), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
(6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(7): ReLU(inplace)
(8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(9): ReLU(inplace)
(10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11): ReLU(inplace))
```

(classifier): Sequential(

```
(0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
(1): ReLU(inplace)
(2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
```

```
 (3): ReLU(inplace)
 (4): Conv2d(256, 20, kernel_size=(1, 1), stride=(1, 1)))
```

)

TODO: Implement the model architcture

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=10):
        super(AlexNet, self).__init__()
        #TODO: Define Features
        self.features=nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2), dilation=(1, 1), ceil_mo
            nn.Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2), dilation=(1, 1), ceil_mo
            nn.Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.ReLU(),
            nn.Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.ReLU())


        #TODO: Define Classifiers
        self.classifier=nn.Sequential(
            nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1)),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1)),
            nn.ReLU(),
            nn.Conv2d(256, 10, kernel_size=(1, 1), stride=(1, 1)))

        #We will give this initialization for you
        for neuron in self.features:
            if isinstance(neuron,nn.Conv2d):
                nn.init.xavier_uniform_(neuron.weight)


    def forward(self, x):
        #TODO: Define forward pass
        x = self.features(x)
        x = self.classifier(x)

        return x
```

Task 3 Build deep learning pipeline(20 Points)

You have Dataset, DataLoader, and your model. It's time to make a pipeline with ingredients. In the recitation, we covered that before we build a training loop, we need to define loss and optimizer. Due to the limited time, we will provide you with loss and optimizer. Use the given parameters.

```
#Do not change this cell
device = "cuda" if torch.cuda.is_available() else "mps" if torch.backends.mps.is_ava
model = AlexNet().to(device)
num_iter=20
loss_fn=nn.BCELoss()
optimizer=torch.torch.optim.SGD(model.parameters(),lr=0.1)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min')
```

Now, build a deep learning pipeline.

TODOs: Finish the pipeline. Bring your DataLoader and the model here. Organize the train loop and test loop, and then train and test your model. At the end of the pipeline, your pipeline should be able to provide graphs of training accuracy, test accuracy and training loss, and report final test accuracy. If you implement it well, your test accuracy should be around 65%.

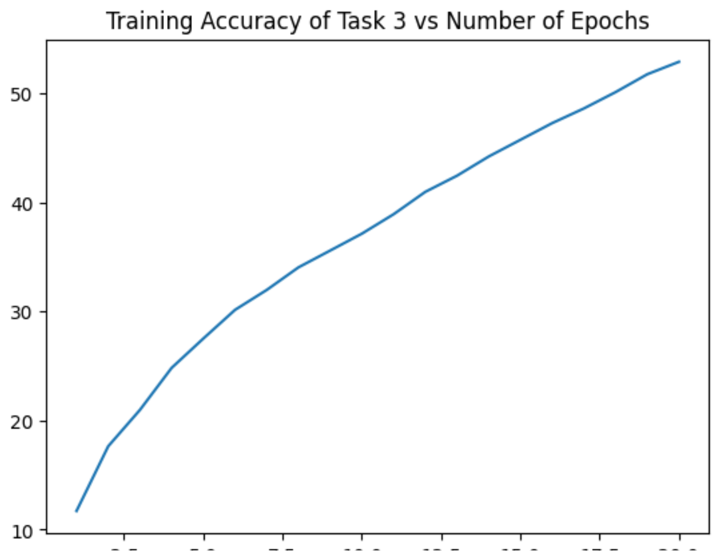
IMPORTANT: After you get predictions from your model, please add below codes before you put the prediction into loss function. Remember below lines should be added in both training loop and test loop!

```
(assume you used pred=model(X))
'''
final_layer=nn.MaxPool2d((pred.size(2),pred.size(3)))
pred=final_layer(pred)
pred=torch.reshape(pred,(-1,10))#(-1,10)
pred=F.sigmoid(pred)
'''
Then loss(pred,y) and goes on
```

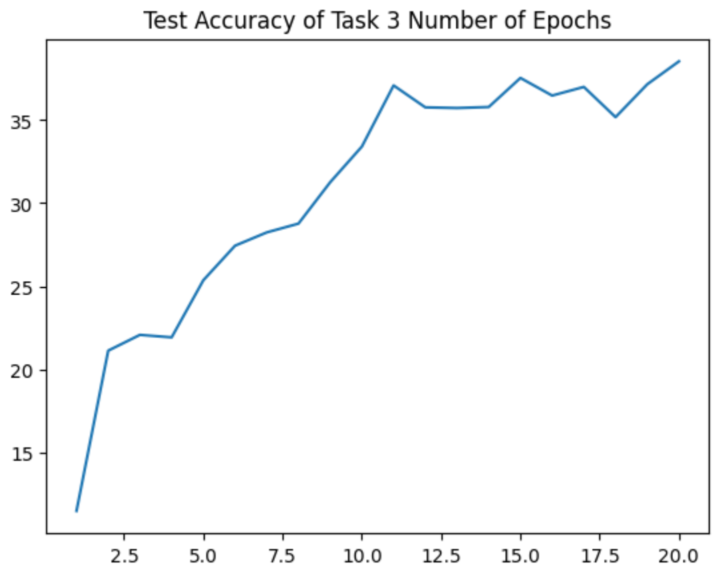
```
#Your code here
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    trainAcc = 0
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)
        pred=model(X)
        final_layer=nn.MaxPool2d((pred.size(2),pred.size(3)))
        yOH = F.one_hot(y, num_classes=10)
        yOH = yOH.to(device).float()
        # Compute prediction error
        pred = final_layer(pred)
        pred=torch.reshape(pred,(-1,10))#(-1,10)
        pred=F.sigmoid(pred)
        loss = loss_fn(pred, yOH)
        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        trainAcc += (pred.argmax(1) == y).type(torch.float).sum().item()
        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

    return trainAcc, loss



def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            yOH = F.one_hot(y, num_classes=10)
            yOH = yOH.to(device).float()
            pred = model(X)
            final_layer=nn.MaxPool2d((pred.size(2),pred.size(3)))
            pred = final_layer(pred)
            pred=torch.reshape(pred,(-1,10))#(-1,10)
            pred=F.sigmoid(pred)
            test_loss += loss_fn(pred, yOH).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>
    return correct*100


# run for 20 and plot
train_accuracy = []
test_accuracy = []
train_loss = []
epochs = 20
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_accu, loss = train(train_dataloader, model, loss_fn, optimizer)
    train_accuracy.append(train_accu)
    train_loss.append(loss)
    correct = test(test_dataloader, model, loss_fn)
    test_accuracy.append(correct)
print("Done!")
print(f"TrainAccuracy: {train_accuracy}")
print(f"TestAccuracy: {test_accuracy}")
print(f"TrainLoss: {train_loss}")
```

```
    loss: 0.225736  [19264/50000]
    loss: 0.233228  [25664/50000]
    loss: 0.246762  [32064/50000]
    loss: 0.242246  [38464/50000]
    loss: 0.251717  [44864/50000]
    Test Error: 
     Accuracy: 37.5%, Avg loss: 0.268782 

    Epoch 16
    -------------------------------
    loss: 0.235936  [   64/50000]
    loss: 0.209610  [ 6464/50000]
    loss: 0.210312  [12864/50000]
    loss: 0.228126  [19264/50000]
    loss: 0.217282  [25664/50000]
    loss: 0.256226  [32064/50000]
    loss: 0.228044  [38464/50000]
    loss: 0.247436  [44864/50000]
    Test Error: 
     Accuracy: 36.5%, Avg loss: 0.272661 

    Epoch 17
    -------------------------------
    loss: 0.241970  [   64/50000]
    loss: 0.217149  [ 6464/50000]
    loss: 0.231873  [12864/50000]
    loss: 0.216801  [19264/50000]
    loss: 0.214846  [25664/50000]
    loss: 0.243000  [32064/50000]
    loss: 0.235954  [38464/50000]
    loss: 0.255420  [44864/50000]
    Test Error: 
     Accuracy: 37.0%, Avg loss: 0.270271 

    Epoch 18
    -------------------------------
    loss: 0.232499  [   64/50000]
    loss: 0.217952  [ 6464/50000]
    loss: 0.213424  [12864/50000]
    loss: 0.200582  [19264/50000]
    loss: 0.212237  [25664/50000]
    loss: 0.228785  [32064/50000]
    loss: 0.228600  [38464/50000]
    loss: 0.254390  [44864/50000]
    Test Error: 
     Accuracy: 35.2%, Avg loss: 0.280925 

    Epoch 19
```

```
y = []
for i in range(len(train_accuracy)):
  y.append(100*train_accuracy[i]/len(train_dataloader.dataset))

# Plot training accuracy
x = list(range(1, 21))
plt.plot(x, y)
plt.title("Training Accuracy of Task 3 vs Number of Epochs")
plt.show()
```

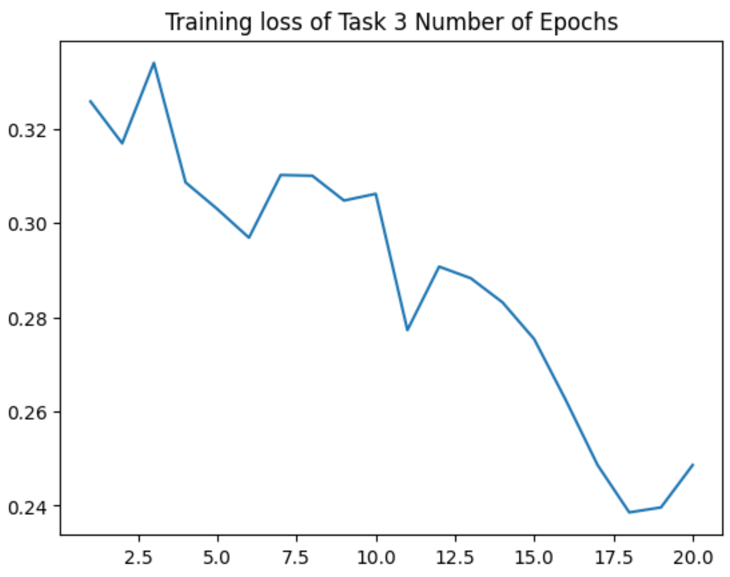


```
# Plot test accuracy
y = test_accuracy
x = list(range(1, 21))
plt.plot(x, y)
plt.title("Test Accuracy of Task 3 Number of Epochs")
plt.show()
```



```
# Plot training loss
y = []
```

```
for i in train_loss:
  y.append(i.item())
x = list(range(1, 21))
plt.plot(x, y)
plt.title("Training loss of Task 3 Number of Epochs")
plt.show()
```



Task 4 Weight transfer(Transfer learning)(5 Points)

There can be two possible scenarios when you are given neural network architcture: One is build everything entirely from the scratch, as you implemented above, and the other is using the pretrained model. The former is preferred when there is no pretrained model or you are training with novel(or unpopular) dataset. The latter is generally more preferred, especially if you are working in computer vision or natural langauge processing related area, because most of the pretrained neural network works very well and their pretrained dataset is very large scale such that starting from raw training might consume great amount of time. For the latter case, it's just one line of code:

```
#model = torch.hub.load('pytorch/vision:v0.10.0', 'alexnet', pretrained=True) #DO N(
```

In this task, we will slightly tweak: You will not use the above code to bring a pretrained model. Instead, you will transfer the weights of the pretrained model to the model you built above. Knowing this will help you later when you want to build your own model but basic flow comes from previous existing models.

One thing you should know, and should remember for your future is that Pytorch stores trained weights as a dictionary data structure, the key and value, and their name of the key has a pattern such that you can easily load your desired key and corresponding weights.

If your implementation is correct, your final accuracy should be around 80%

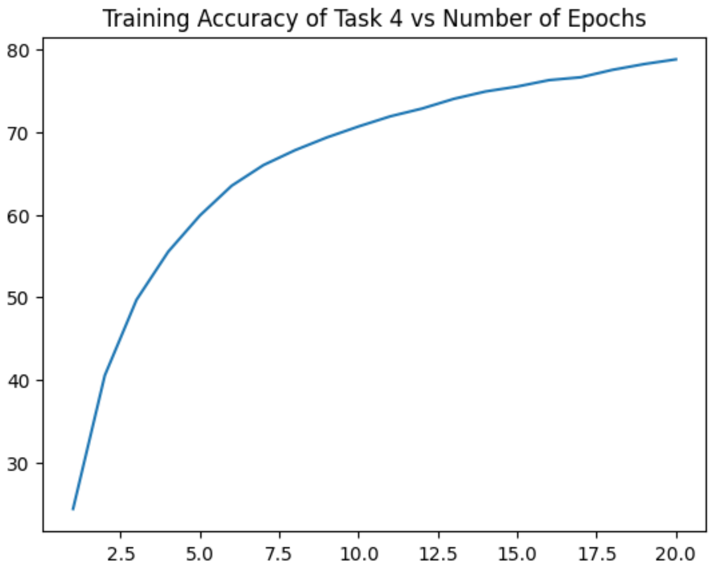
TODOs: Implement below weight transfer function. Then train and test your pretrained model.Provide graphs of training accuracy, test accuracy and training loss. Report final accuracy

```
from torchvision._internally_replaced_utils import load_state_dict_from_url # This i

def Load_Alexnet_Weight_Transferred(pretrained=True, **kwargs):
    r"""AlexNet model architecture from the
    `"One weird trick..." <https://arxiv.org/abs/1404.5997>`_ paper.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model_urls = {
            'alexnet': 'https://download.pytorch.org/models/alexnet-owt-4df8aa71.pth
    }
    model = AlexNet().to(device)
    modelDict = model.state_dict()
    state_dict = torch.hub.load_state_dict_from_url(model_urls['alexnet'], progress
    #TODO: Initialize weights correctly based on whethet it is pretrained or not
    if pretrained:
        #Your code here
        model_pretrained = {k: v for k, v in state_dict.items() if k in modelDict an
        modelDict.update(model_pretrained)
        model.load_state_dict(modelDict)
        # for k in state_dict:
        #   model.state_dict()[k] = state_dict[k]
    return model


model_pretrained = Load_Alexnet_Weight_Transferred(pretrained=True)


#Use your previous training and test loops, but don't forget to use above model
trainAccu = []
loss2 = []
testAccu = []
epochs = 20
optimizer=torch.torch.optim.SGD(model_pretrained.parameters(),lr=0.1)
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_accu, loss = train(train_dataloader, model_pretrained, loss_fn, optimizer
    trainAccu.append(train_accu)
    loss2.append(loss)
    correct = test(test_dataloader, model_pretrained , loss_fn)
    testAccu.append(correct)
print("Done!")
print(f"TrainAccuracy: {trainAccu}")
print(f"TestAccuracy: {testAccu}")
```

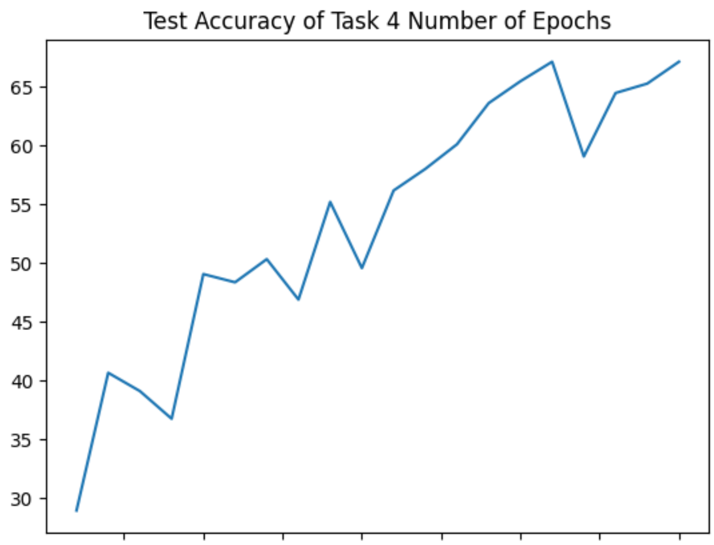
```python
print(f"TrainLoss: {loss2}")
# modelDup = model_pretrained
# torch.save(model_pretrained.state_dict(), "model_pretrained.pt")
torch.save(model_pretrained.state_dict(), "model_pretrained.pth")

# model_pretrained = AlexNet().to(device)
# model_pretrained.load_state_dict(torch.load('./model_pretrained.pt', map_location=
```

```
    Epoch 1
    -------------------------------
    loss: 0.719297  [   64/50000]
    loss: 0.321809  [ 6464/50000]
    loss: 0.314105  [12864/50000]
    loss: 0.320717  [19264/50000]
    loss: 0.297116  [25664/50000]
    loss: 0.299236  [32064/50000]
    loss: 0.272702  [38464/50000]
    loss: 0.276424  [44864/50000]
    Test Error:
     Accuracy: 28.9%, Avg loss: 0.277450

    Epoch 2
    -------------------------------
    loss: 0.279558  [   64/50000]
    loss: 0.258976  [ 6464/50000]
    loss: 0.257205  [12864/50000]
    loss: 0.248174  [19264/50000]
    loss: 0.242184  [25664/50000]
    loss: 0.258015  [32064/50000]
    loss: 0.245365  [38464/50000]
    loss: 0.245362  [44864/50000]
    Test Error:
     Accuracy: 40.7%, Avg loss: 0.250098

    Epoch 3
    -------------------------------
    loss: 0.256372  [   64/50000]
    loss: 0.221296  [ 6464/50000]
    loss: 0.228528  [12864/50000]
    loss: 0.220592  [19264/50000]
    loss: 0.229670  [25664/50000]
    loss: 0.221554  [32064/50000]
    loss: 0.225487  [38464/50000]
    loss: 0.230136  [44864/50000]
    Test Error:
     Accuracy: 39.1%, Avg loss: 0.266214

    Epoch 4
    -------------------------------
    loss: 0.258976  [   64/50000]
    loss: 0.210925  [ 6464/50000]
    loss: 0.203458  [12864/50000]
    loss: 0.196298  [19264/50000]
    loss: 0.204240  [25664/50000]
    loss: 0.218365  [32064/50000]
    loss: 0.214091  [38464/50000]
    loss: 0.211613  [44864/50000]
    Test Error:
     Accuracy: 36.8%, Avg loss: 0.280328

    Epoch 5
    -------------------------------
    loss: 0.236149  [   64/50000]
    loss: 0.194910  [ 6464/50000]
    loss: 0.186671  [12864/50000]
    loss: 0.189963  [19264/50000]
```

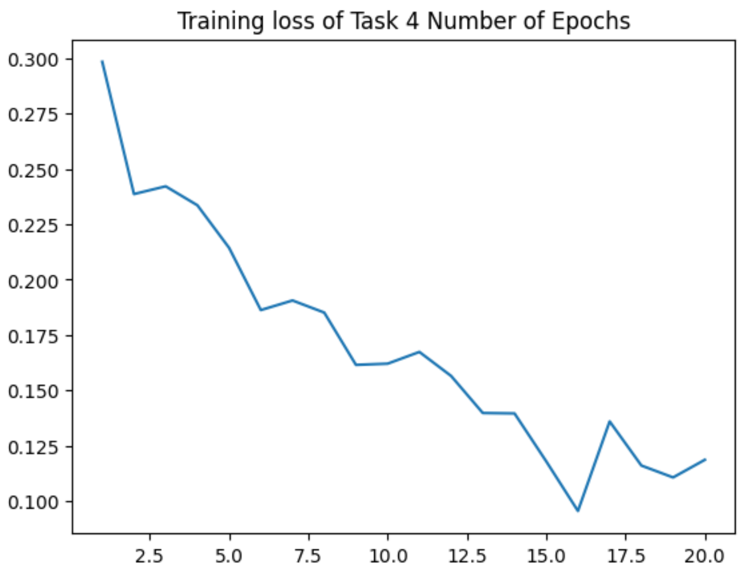
```
torch.save({
    'trainAccu': trainAccu,
    'testAccu' : testAccu,
    'loss2': loss2
}, 'vals.pth'
)


y = []
for i in range(len(trainAccu)):
  y.append(100*trainAccu[i]/len(train_dataloader.dataset))

# Plot training accuracy
x = list(range(1, 21))
plt.plot(x, y)
plt.title("Training Accuracy of Task 4 vs Number of Epochs")
plt.show()
```



```
# Plot test accuracy
y = testAccu
x = list(range(1, 21))
plt.plot(x, y)
plt.title("Test Accuracy of Task 4 Number of Epochs")
plt.show()
```



```
# Plot training loss
y = []
for i in loss2:
  y.append(i.item())
x = list(range(1, 21))
plt.plot(x, y)
plt.title("Training loss of Task 4 Number of Epochs")
plt.show()
```



Task 5 Evaluate your model(10 Points)

You have a full loop of both train and test. Let's see how good your model is. In the lecture, we learned precision, recall. In addition to these metrics, one way to visualize our model performance is to show a heatmap of classification result. You need to build a (# classes)x(# classes) matrix. Then, for each sample in the test set with a true label ith class, we can get jth class through your model. Then we add a value to matrix[i][j], then we normalize the matrix. By doing so, we can visually show our performance. Remember that in CIFAR10, we have 10 classes.

TODOs

1. Visualize the heatmap, and report which class showed most accurate, and which 'task' showed most mistakes, i.e, predicted j when the true label is i.
2. Provide 3 cases of failed prediction with most mistakes 'task'. So you should show a total of 6 images here. Briefly write why you think the model can't predict well.

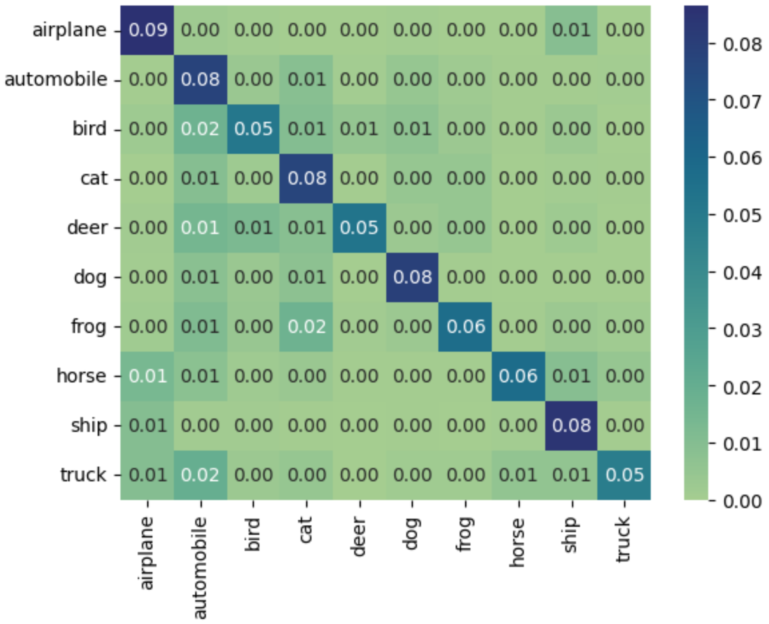
```
device = "cuda" if torch.cuda.is_available() else "mps" if torch.backends.mps.is_ava
dataloader = test_dataloader
model = AlexNet().to(device)
model.eval()
model.load_state_dict(torch.load('./model_pretrained.pth', map_location=torch.device
model.eval()
model.cuda()
matrix = np.zeros((10,10))
#optimizer=torch.torch.optim.SGD(model.parameters(),lr=0.1)
#scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min')
cell1 = -1
cell2 = -1
cell3 = -1
totalPred = []
with torch.no_grad():
  for X, y in dataloader:
      X, y = X.to(device), y.to(device)
      pred = model(X)
      yOH = F.one_hot(y, num_classes=10)
      final_layer=nn.MaxPool2d((pred.size(2),pred.size(3)))
      pred = final_layer(pred)
      pred = torch.reshape(pred,(-1,10))#(-1,10)
      pred = F.sigmoid(pred)

      totalPred.append(pred)

      pred = pred.float()
      yOH = yOH.float()
      l = len(yOH.argmax(1))
      # 3 cases of failed predictions
      for i in range(l):
        if cell1 == -1 and yOH.argmax(1)[i] == 1 and pred.argmax(1)[i] == 2:
          cell1 = i
        if cell2 == -1 and yOH.argmax(1)[i] == 1 and pred.argmax(1)[i] == 3:
          cell2 = i
        if cell3 == -1 and yOH.argmax(1)[i] == 1 and pred.argmax(1)[i] == 9:
          cell3 = i
        matrix[yOH.argmax(1)[i]-1][pred.argmax(1)[i]-1] += 1

totalPred = torch.vstack(totalPred).argmax(1)
totalPred = totalPred.tolist()
# print(totalPred)
```

```
    [2, 8, 8, 0, 6, 6, 9, 6, 4, 1, 2, 9, 2, 7, 9, 1, 5, 9, 9, 6, 9, 0, 4, 9, 4, 2,
```

```
l = dict()
for k, v in testset.class_to_idx.items():
  l[v] = k
labels = []
for key in l.keys():
  labels.append(l[key])
```

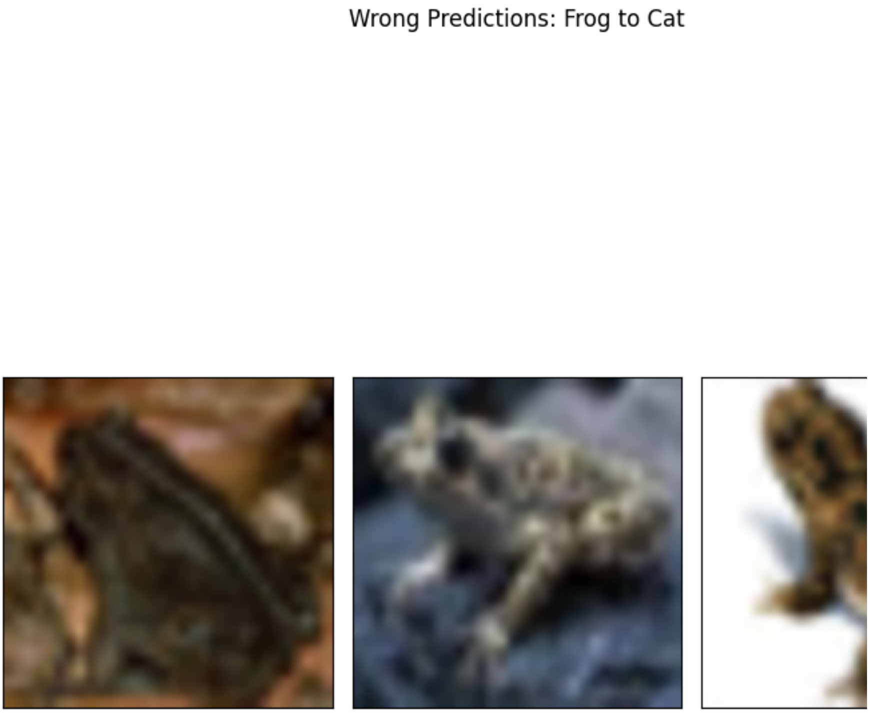
```
sns.heatmap(matrix/np.sum(matrix), annot = True,  fmt = '.2f',cmap="crest", yticklal
```

```
    <Axes: >
```



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
automobile	0.00	0.08	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
bird	0.00	0.02	0.05	0.01	0.01	0.01	0.00	0.00	0.00	0.00
cat	0.00	0.01	0.00	0.08	0.00	0.00	0.00	0.00	0.00	0.00
deer	0.00	0.01	0.01	0.01	0.05	0.00	0.00	0.00	0.00	0.00
dog	0.00	0.01	0.00	0.01	0.00	0.08	0.00	0.00	0.00	0.00
frog	0.00	0.01	0.00	0.02	0.00	0.00	0.06	0.00	0.00	0.00
horse	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.06	0.01	0.00
ship	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.00
truck	0.01	0.02	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.05

```
from PIL import Image

l = dict()
for k, v in testset.class_to_idx.items():
  l[v] = k
labels = []
for key in l.keys():
  labels.append(l[key])


# from ipywidgets.widgets.widget_media import Image

graph, axes = plt.subplots(1,3, figsize = (8,8))
count = 0
targets = testset.targets

for i, (a,b) in enumerate(zip(targets, totalPred)):
  correct = l[a]
  prediction = l[b]
  if correct == 'frog':
    if prediction == 'cat':
      image = testset.data[i]
      image = Image.fromarray(image)
      image = image.resize((128, 128))
      axes[count].imshow(image)
      axes[count].set_xticks([])
      axes[count].set_yticks([])
      count += 1
      if count == 3:
        graph.suptitle("Wrong Predictions: Frog to Cat")
        graph.tight_layout()
        break;
```



Write analysis here

Task 6 Create your own model(10 Points OR 10 Points + 10 Bonus Points)

You have two options, and this task is open ended.

Make your own model. Name the model class with your name(e.g. YoungJeNet). Briefly describe how you organized your model architecture and pipeline, and why you proposed such architecture (feel free to take inspiration from common architectures, and please reference any tutorials/guides that you use). Feel free to use your designed DataLoader and feel free to weight shift from any different model. Also feel free to use your training and test loop. With your defined model, run your pipeline on a new dataset and provide graphs of training accuracy and training loss, and report final accuracy.

Do NOT just bring pretrained model: Like model=some_kind_of_pretrained_model(pretrain=True)

For your new dataset, choose one of the following options:

1. (10 Points) One of the following standard datasets: CIFAR10, PASCAL VOC, CALTECH256, or ImageNet2012.
2. (10 Points + 10 Bonus Points) Find a dataset of your interest in biology. We are giving bonus points here because you need to describe further about your dataset and may need to work more with Dataloader.

Please be mind that if you choose 2, this should not be part of your project.

Your score will be determined based on 1. Description of dataset 2. Clearly stated idea and correctly implemented the idea. 3. Not too low accuracy. Please be aware that accuracy is not the only criterion here.

```
#Your Model
```

(Option) For your career, we encourage you to upload your work on Github. Github is a repository for programmers. This repo can be useful to your future career, especially if you aim to work in a computational job: provide your Github link to recruiter/research POI. By doing so, you can show your recruiter/research POI that you have fundamental ability to work with deep learning and being able to code Pytorch.

Congratulations! Now you understand the basic flow of deep learning workflow. CMU provides a variety of deep learning courses, so we recommend taking any of them if you are interested or strengthen your knowledge and skills in deep learning. If you want to learn general deep learning, consider 11-685 Introduction to Deep learning. Be careful that this course is very hectic. LTI and

RI offer domain specific deep learning courses(such as Natural Language Processing, Visual Learning and Recognition). If you want to know deep learning theory, consider 10-707. Please be aware that what we've covered in this assignment is very basic: this is going to be assignment 0 for other deep learning courses. However, we believe that this assignment will work as a immigration assignment to deep learning.

✓ 0s completed at 8:49 PM ● ×