Georgia Institute of Technology

ECE 4803: Fundamentamentals of Machine Learning (FunML)

Spring 2022

Final Take Home Exam

Due: Wednesday, May 4th, 2022 @11:59 PM

Please read the following instructions carefully.

- You may post questions to Piazza during the exam availability time. You **MAY NOT** post solutions or information related to how problems should be solved.
- Rename the PDF according to the format: ***LastName_FirstName_ECE_4803_sp22_final_exam.pdf***
- Late submission is not accepted unless arranged otherwise and in advance.
- **Remember to terminate your session if you pause using the Google colab. Otherwise you might run out of your daily gpu allocation quota.**

The Academic Honor Code will be strictly enforced. Forgeries and plagiarism are violations of the Georgia Tech honor code and will be referred to the Dean of Students for disciplinary action. You are not allowed to discuss exam content or to share any written, electronic, or any other form of exam information with anyone during or after the exam until the solutions have been posted. By submitting this exam, you affirm that you have neither given nor received inappropriate assistance during this exam. I have read the instructions above and affirm that I will abide by the guidelines provided.

Type your name here: Ting-Ying Yu (My kaggle name: tyu304)

Problem 1 : Perform Transfer Learning on CURE-OR (15pts)

The last set of problems in Assignment 5 required you to take one of the various popular computer vision network architectures and

- firstly, test them as they come pretrained on the large scale `ImageNet` dataset on data samples extracted from the `CURE-OR` dataset
- secondly, train them from scratch (after modification) on the training split from the `CURE-OR` dataset.

The first point as you saw last time is impratical for many reasons including where object categories are different and there is a shift in the target domain e.g., target examples contain specific artifacts not present in the source. The second option, as you might have guessed as well, is sub-optimal for situations involving a limited number of training examples in the target dataset (`CURE-OR` in this case), and might lead to a slower convergence and/or local minima in the training process. This is where the highly useful concept of *transfer learning* comes in.

As the name implies, it involves transferring prior knowledge from a source dataset/domain that is related to the target dataset/domain of interest. Its utility is justified mainly from the fact that neural networks are known to learn general, domain-invariant representations in the earlier layers, followed by more specific ones later. The representations learnt on a large source dataset like `ImageNet` therefore prove especially useful in other image-based tasks, datasets, and domains. To account for the distributional shift between the source and target domains, one usually finetunes the last few layers of the pretrained architecture with the limited labeled training examples from the target dataset.

Below, we are going to analyze the performance of various object classifiers after they have been finetuned on the training examples from `CURE-OR` dataset. Before we proceed, make sure that:

1) The runtime session is set to GPU mode
2) Your personal Drive storage is mounted and connected to the local session.
3) The directory structure we set up last time for `CURE-OR`, the various files we downloaded, and the training and test splits we created are still there. If not, you may want to redo step 1 of Problem 5 in Assignment 5.

Below, we will guide you in a step-by-step process to set things up from scratch before we get to the transfer learning part itself.

```
# mount google drive
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

(a) Run the code cell below to establish paths to various directories in the folder structre we set up last time

```
# for part (a)

import os

root = '/content/drive/MyDrive/ECE-4803-Assignment-5-files'
train_directory = '/content/drive/My Drive/ECE-4803-Assignment-5-files/data/train'
test_directory = '/content/drive/My Drive/ECE-4803-Assignment-5-files/data/test'
os.chdir(root)
```

(b) Run the code cell below to set up training and testing dataset classes and their respective loader objects for extracting training and testing images, respectively from the `CURE-OR` splits we created last time.

```
# create a dataset and dataloader for part (b)

import torch
from torchvision.datasets import ImageFolder
from torchvision import transforms

preprocess = transforms.Compose([
       transforms.Resize(256),
       transforms.CenterCrop(224),
       transforms.ToTensor(),
       transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
       ])

# set up train loader
```

```
train_dataset = ImageFolder(root=train_directory+'/train_imgs', transform=preprocess)
trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=1, shuffle=True)

# set up test loader
test_dataset = ImageFolder(root=test_directory+'/test_imgs', transform=preprocess)
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=1, shuffle=False)
```

(c) Using the code cell below as template, load the pretrained weights for the `Alexnet` we downloaded into the `\content\drive\MyDrive\ECE-4803-Assignment-5-files\pretrained_models` directory last time into an alexnet network object. Change the last linear layer of the network architecture to have the appropriate number of neurons, and afterwards, finetune *only this last layer* with a learning rate of $1e-4$ for 50 epochs, similar to last time.

Remember that you can choose which layers should be fixed and which ones should be trained by passing in the appropriate argument to the optimizer object. Last time, we wanted to train the whole network, so all if its parameters were passed into the optimizer object. But you can also be selective about the various layers you want finetuned. Remember also that you only load the pretrained weights when you are finetuning. You would not have done this step if you were retraining the network from scratch.

Problem 1 (c) Solution

```
import torch.nn as nn
from torchvision.models import alexnet, resnet18, vgg16
import torch.nn.functional as F

import numpy as np
```

```
# Finetune model for part c

# instantiate model object and
model =  alexnet(pretrained=True)
```

```
# change last layer of model
model =  alexnet(num_classes = 10)


model.to('cuda') # send model to cuda

# define loss
loss = nn.CrossEntropyLoss()
lr = 10e-4


# set up optimizer
optimizer = torch.optim.SGD(model.parameters(), lr = lr)

for epoch in range(50):
  model.train()

  for i, (x, y) in enumerate(trainloader):
    optimizer.zero_grad()
    input = x.to('cuda')
    target = y.to('cuda')

    ##TODO # obtain output posteriors
    ##TODO # obtain loss
    ##TODO # backproapagate loss
    ##TODO # perform gradient descent step using optimizer
    out = model(input)##TODO
    loss_val = loss(out, target)##TODO

    loss_val.backward()

    optimizer.step()

  print('Epoch: {} | Loss:{:0.4f}'.format(epoch, loss_val.item()))
```

```
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /root/.cache/torch/hub/checkpoints/alexnet-owt-7
```

100% 233M/233M [00:02<00:00, 153MB/s]

```
Epoch: 0 | Loss:2.3280
Epoch: 1 | Loss:2.2745
Epoch: 2 | Loss:2.2571
Epoch: 3 | Loss:2.2522
Epoch: 4 | Loss:2.3670
Epoch: 5 | Loss:2.2259
Epoch: 6 | Loss:2.4007
Epoch: 7 | Loss:2.4921
Epoch: 8 | Loss:2.1622
Epoch: 9 | Loss:2.2266
Epoch: 10 | Loss:2.2200
Epoch: 11 | Loss:2.4436
Epoch: 12 | Loss:2.1662
Epoch: 13 | Loss:2.1763
Epoch: 14 | Loss:2.6478
Epoch: 15 | Loss:2.3382
Epoch: 16 | Loss:2.4506
Epoch: 17 | Loss:2.4644
Epoch: 18 | Loss:2.1561
Epoch: 19 | Loss:2.4276
Epoch: 20 | Loss:2.2052
Epoch: 21 | Loss:2.0540
Epoch: 22 | Loss:1.4452
Epoch: 23 | Loss:1.3421
Epoch: 24 | Loss:2.3745
Epoch: 25 | Loss:2.2819
Epoch: 26 | Loss:1.4615
Epoch: 27 | Loss:2.0341
Epoch: 28 | Loss:2.1434
Epoch: 29 | Loss:1.9665
Epoch: 30 | Loss:0.6909
Epoch: 31 | Loss:2.6916
Epoch: 32 | Loss:0.8939
Epoch: 33 | Loss:1.2617
Epoch: 34 | Loss:0.7404
Epoch: 35 | Loss:0.1327
Epoch: 36 | Loss:0.7276
Epoch: 37 | Loss:1.3982
```

```
Epoch: 38 | Loss:0.4578
Epoch: 39 | Loss:2.1921
Epoch: 40 | Loss:0.0000
Epoch: 41 | Loss:0.3528
Epoch: 42 | Loss:0.6266
Epoch: 43 | Loss:0.3660
Epoch: 44 | Loss:0.4008
Epoch: 45 | Loss:1.7438
Epoch: 46 | Loss:0.0000
Epoch: 47 | Loss:0.2047
Epoch: 48 | Loss:2.5010
```

(d) As studied in class, a popular metric to evaluate object classification models is the top-k accuracy. The top-k accuracy essentially measures the accuracy rate on a given dataset by measuring the fraction of examples for which the correct label appears in the highest k scores of the classifier. Complete the function below to compute the top-k accuracy scores given a classifier's output vector of posterior probabilities and the ground-truth labels for those data examples. Use the function's docstring to guide you to complete the function. You are provided a dummy dataset and its ground-truth labels. Successfully running the cell should print an output for top-3 accuracy for this dataset as 0.33.

Problem 1 (d) Solution

```
# define top k error function for part d

import torch

def topk_accuracy(y_post, y_true, k):
  """Function outputs a binary vector correspodning to if true label was among top k
  labels for a classifier or not.

  Parameters
  ----------
  y_post: torch.tensor, shape(N, C), type=torch.float
```

```
    tensor containing posterior vectors for each of the N data points in the batch.

  y_true: torch.tensor, shape(N), type=torch.long
    tensor containing ground-truth labels for each of the N data points in the batch

  k: int
    integer specifying value of k

  Returns
  -------
  out: torch.tensor, shape(N)
    tensor containing 0 (if true label not among top k scores) or 1(if true label)
    among top k scores) for each of the N training examples."""

  ##TODO
  if (k==1):
    value, index = torch.topk(y_post, dim=1 ,k=1)
    out = torch.transpose(index, 0, 1)[0]
    out = out == y_true
    # print(out)
    return out


  value, index = torch.topk(y_post, dim=1 ,k=k)
  out = y_true
  i = 0
  for item in y_true:
    # print(item)
    # print(torch.sum(index[i,:] == item) > 0)
    if (torch.sum(index[i,:] == item) > 0):
      out[i] = 1
    else:
      out[i] = 0
    i=i+1

  return out
```

```
#----------------Don't change anything below----------------------#

dummy_posteriors = torch.tensor([[0.2, 0.3, 0.1, 0.25, 0.15],
                                 [0.6, 0.15, 0.2, 0.05, 0.0],
                                 [0.2, 0.1, 0.05, 0.3, 0.35]])

dummy_ground_truths = torch.tensor([0, 4, 2])
# dummy_ground_truths = torch.tensor([0, 0, 4])

k = 3
# k = 1

out = topk_accuracy(dummy_posteriors, dummy_ground_truths, k)
topk_score = out.sum() / torch.numel(out)
print('Top-3 accuracy score for dummy dataset: {:0.4f}'.format(topk_score.item()))
```

```
Top-3 accuracy score for dummy dataset: 0.3333
```

(e) Now we iterate through the training and testloaders as set up earlier to evaluate the trained model for the top-1 and top-3 error rates. Run the cells below and record the numbers in the tables provided.

Problem 1 (e) Solution

```
# top 1 and top 3 errors for training set (part e)

out_1 = torch.empty(0) # intialize an empy tensor
out_3 = torch.empty(0) # intialize an empy tensor

out_1 = out_1.to('cuda')
out_3 = out_3.to('cuda')
```

```
with torch.no_grad():
  for img, label in trainloader:
    img = img.to("cuda")
    label = label.to("cuda")

    y_post = model(img)

    top_1_vec = topk_accuracy(y_post, label, 1)
    top_3_vec = topk_accuracy(y_post, label, 3)

    out_1 = torch.cat((out_1, top_1_vec))
    out_3 = torch.cat((out_3, top_3_vec))

print('Training top-1 Score: {:0.4f}'.format(out_1.sum() / torch.numel(out_1)))
print('Training top-3 Score: {:0.4f}'.format(out_3.sum() / torch.numel(out_3)))
```

```
 Training top-1 Score: 0.8717
 Training top-3 Score: 0.9866
```

```
# top 1 and top 3 errors for test set (part e)

out_1 = torch.empty(0) # intialize an empy tensor
out_3 = torch.empty(0) # intialize an empy tensor

out_1 = out_1.to('cuda')
out_3 = out_3.to('cuda')

with torch.no_grad():
  for i, (img, label) in enumerate(testloader):
    img = img.to("cuda")
    label = label.to("cuda")

    y_post = model(img)

    top_1_vec = topk_accuracy(y_post, label, 1)
```

```
    top_3_vec = topk_accuracy(y_post, label, 3)

    out_1 = torch.cat((out_1, top_1_vec))
    out_3 = torch.cat((out_3, top_3_vec))

print('Test top-1 Score: {:0.4f}'.format(out_1.sum() / torch.numel(out_1)))
print('Test top-3 Score: {:0.4f}'.format(out_3.sum() / torch.numel(out_3)))
```

```
 Test top-1 Score: 0.2514
 Test top-3 Score: 0.5657
```

Problem 2: Evaluating the Effect of Pretraining and Finetuning on Different Network Architectures (15pts)

(a) In Problem 1 above, you took a pretrained `Alexnet` architecture and then finetuned the last linear layer only to then evaluate on the training and test sets in terms of the top-1 and top-3 accuracy scores. This corresponds to the second entry in the table below:

Alexnet	Top-1 Train Acc	Top-3 Train Acc	Top-1 Test Acc	Top-3 Test Acc
Training From Scratch				
Finetuning Last Linear Layer				
Finetuning Last 2 Linear Layers				
Finetuning the complete last classification module				

Complete the other entries of the table by finetuning (or retrainining from scratch) the different layers of the network as indicated. Notice that this would entail finetuning from the layer/block indicated until the end of the network. Note also that the first row is what you did in the last assignment, where a network intialized randomly is trained from scratch on the given training examples of a target dataset.

(b) In a similar fashion to above, take a `VGG-16` architecture and then finetuned (or retrain from scratch) to complete the table below:

VGG-16	Top-1 Train Acc	Top-3 Train Acc	Top-1 Test Acc	Top-3 Test Acc
Training From Scratch				
Finetuning Last Linear Layer				
Finetuning Last 2 Linear Layers				

VGG-16	Top-1 Train Acc	Top-3 Train Acc	Top-1 Test Acc	Top-3 Test Acc
Finetuning the complete last classification module				

(c) Lastly, repeat the process for the `Resnet-18` architecture to complete the table below:

Resnet-18	Top-1 Train Acc	Top-3 Train Acc	Top-1 Test Acc	Top-3 Test Acc
Training From Scratch				
Finetuning Last Linear Layer				
Finetuning Last Basic Block				
Finetuning Last two Basic Blocks				

Problem 2 Solution

```
print(alexnet())
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
```

```
      (1): Linear(in_features=9216, out_features=4096, bias=True)
      (2): ReLU(inplace=True)
      (3): Dropout(p=0.5, inplace=False)
      (4): Linear(in_features=4096, out_features=4096, bias=True)
      (5): ReLU(inplace=True)
      (6): Linear(in_features=4096, out_features=1000, bias=True)
    )
  )
```

```
model = alexnet(pretrained=True)
model = vgg16(pretrained=True)
model = resnet18(pretrained=True)
```

```
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /root/.cache/torch/hub/checkpoints/alexnet-owt-7
100%                                      233M/233M [00:01<00:00, 154MB/s]
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
100%                                      528M/528M [00:04<00:00, 117MB/s]
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072f
100%                                      44.7M/44.7M [00:00<00:00, 88.7MB/s]
```

```
## Complete problem 2 code here

#################################################################################################
# Alexnet
model = alexnet
model =  alexnet(pretrained=True)
model =  alexnet(num_classes = 10) # change last layer of model

# vgg
# model = vgg16(pretrained=True)
# model.classifier[6] = torch.nn.Linear(in_features=4096, out_features=10, bias=True)

# Resnet
# model = resnet18(pretrained=True)
```

```
# model.fc = torch.nn.Linear(in_features=512, out_features=10, bias=True)
model.to('cuda') # send model to cuda

#############################################################################################

# define loss
loss = nn.CrossEntropyLoss() ##TODO
# loss = nn.MSELoss()
# loss1 = nn.L1Loss()
lr = 10e-4 ##TODO

# set up optimizer
#############################################################################################
# Alexnet fintuning:
params =  model.classifier[6].parameters()
# params =  list(model.classifier[4].parameters()) + list(model.classifier[6].parameters())
# params =  list(model.classifier[1].parameters()) + list(model.classifier[4].parameters()) + list(model.classifier[6].parameters())

# vgg fintuning:
# params =  model.classifier[6].parameters()
# params =  list(model.classifier[3].parameters()) + list(model.classifier[6].parameters())
# params =  list(model.classifier[0].parameters()) + list(model.classifier[3].parameters()) + list(model.classifier[6].parameters())

# resnet fintuning:
# params =  model.fc.parameters()
# params =  list(model.layer4[1].parameters()) + list(model.fc.parameters())
# params =  list(model.layer4[0].parameters()) + list(model.layer4[1].parameters()) + list(model.fc.parameters())

optimizer = torch.optim.SGD(params, lr = lr, weight_decay=0.001) ##TODO
#############################################################################################

for epoch in range(100):
  model.train()

  for i, (x, y) in enumerate(trainloader):
    optimizer.zero_grad()
    input = x.to('cuda')
```

```
    target = y.to('cuda')

    out= model(input)##TODO
    loss_val = loss(out, target)##TODO

    loss_val.backward()

    optimizer.step()

  print('Epoch: {} | Loss:{:0.4f}'.format(epoch, loss_val.item()))
```

```
 Epoch: 0 | Loss:0.0000
 Epoch: 1 | Loss:0.0000
 Epoch: 2 | Loss:0.0000
 Epoch: 3 | Loss:0.0000
 Epoch: 4 | Loss:0.0000
 Epoch: 5 | Loss:0.0000
 Epoch: 6 | Loss:0.0001
 Epoch: 7 | Loss:0.0000
 Epoch: 8 | Loss:0.0000
 Epoch: 9 | Loss:0.0000
 Epoch: 10 | Loss:0.0000
 Epoch: 11 | Loss:0.0000
 Epoch: 12 | Loss:0.0000
 Epoch: 13 | Loss:0.0002
 Epoch: 14 | Loss:0.0000
 Epoch: 15 | Loss:0.0000
 Epoch: 16 | Loss:0.0000
 Epoch: 17 | Loss:0.0000
 Epoch: 18 | Loss:0.0000
 Epoch: 19 | Loss:0.0000
 Epoch: 20 | Loss:0.0000
 Epoch: 21 | Loss:0.0000
 Epoch: 22 | Loss:0.0012
 Epoch: 23 | Loss:0.0000
 Epoch: 24 | Loss:0.0000
 Epoch: 25 | Loss:0.0000
 Epoch: 26 | Loss:0.0000
 Epoch: 27 | Loss:0.0000
 Epoch: 28 | Loss:0.0000
```

```
Epoch: 29 | Loss:0.0000
Epoch: 30 | Loss:0.0000
Epoch: 31 | Loss:0.0000
Epoch: 32 | Loss:0.0000
Epoch: 33 | Loss:0.0000
Epoch: 34 | Loss:0.0008
Epoch: 35 | Loss:0.0000
Epoch: 36 | Loss:0.0000
Epoch: 37 | Loss:0.0000
Epoch: 38 | Loss:0.0000
Epoch: 39 | Loss:0.0000
Epoch: 40 | Loss:0.0000
Epoch: 41 | Loss:0.0000
Epoch: 42 | Loss:0.0000
Epoch: 43 | Loss:0.0000
Epoch: 44 | Loss:0.0000
Epoch: 45 | Loss:0.0000
Epoch: 46 | Loss:0.0000
Epoch: 47 | Loss:0.0000
Epoch: 48 | Loss:0.0000
Epoch: 49 | Loss:0.0000
Epoch: 50 | Loss:0.0000
Epoch: 51 | Loss:0.0000
Epoch: 52 | Loss:0.0000
Epoch: 53 | Loss:0.0000
Epoch: 54 | Loss:0.0000
Epoch: 55 | Loss:0.0000
Epoch: 56 | Loss:0.0000
Epoch: 57 | Loss:0.0000
```

```
####################
# output scores
####################
submission_list=[]

# Training set (part e)

out_1 = torch.empty(0) # intialize an empy tensor
out_3 = torch.empty(0) # intialize an empy tensor
```

```
out_1 = out_1.to('cuda')
out_3 = out_3.to('cuda')

# print(trainloader)


with torch.no_grad():
  for img, label in trainloader:

    img = img.to("cuda")
    label = label.to("cuda")

    y_post = model(img)
    # submission_list.append(y_post.argmax().item())

    top_1_vec = topk_accuracy(y_post, label, 1)
    top_3_vec = topk_accuracy(y_post, label, 3)

    out_1 = torch.cat((out_1, top_1_vec))
    out_3 = torch.cat((out_3, top_3_vec))

print('Training top-1 Score: {:0.4f}'.format(out_1.sum() / torch.numel(out_1)))
print('Training top-3 Score: {:0.4f}'.format(out_3.sum() / torch.numel(out_3)))

# Testing set (part e)

out_1 = torch.empty(0) # intialize an empy tensor
out_3 = torch.empty(0) # intialize an empy tensor

out_1 = out_1.to('cuda')
out_3 = out_3.to('cuda')

submission_list = []

with torch.no_grad():
  for i, (img, label) in enumerate(testloader):
```

```
    img = img.to("cuda")
    label = label.to("cuda")

    y_post = model(img)
    submission_list.append(y_post.argmax().item())
    # print(y_post.argmax())
    # print(label)

    top_1_vec = topk_accuracy(y_post, label, 1)
    top_3_vec = topk_accuracy(y_post, label, 3)

    out_1 = torch.cat((out_1, top_1_vec))
    out_3 = torch.cat((out_3, top_3_vec))
print('Test top-1 Score: {:0.4f}'.format(out_1.sum() / torch.numel(out_1)))
print('Test top-3 Score: {:0.4f}'.format(out_3.sum() / torch.numel(out_3)))
```

```
 Training top-1 Score: 1.0000
 Training top-3 Score: 1.0000
 Test top-1 Score: 0.7400
 Test top-3 Score: 0.9200
```

For Alexnet:

Alexnet	Top-1 Train Acc	Top-3 Train Acc	Top-1 Test Acc	Top-3 Test Acc
Training From Scratch	0.6209	0.8681	0.2229	0.56
Finetuning Last Linear Layer	0.7582	0.9176	0.2457	0.5086
Finetuning Last 2 Linear Layers	0.7912	0.9396	0.2343	0.5029
Finetuning the complete last classification module	0.8736	0.978	0.2057	0.5316

For VGG-16:

VGG-16	Top-1 Train Acc	Top-3 Train Acc	Top-1 Test Acc	Top-3 Test Acc
Training From Scratch	0.9792	0.9863	0.2629	0.5052
Finetuning Last Linear Layer	0.9251	0.9786	0.56	0.78
Finetuning Last 2 Linear Layers	0.984	0.9973	0.58	0.82

VGG-16	Top-1 Train Acc	Top-3 Train Acc	Top-1 Test Acc	Top-3 Test Acc
Finetuning the complete last classification module	1	1	0.54	0.8400

For Resnet-18:

Resnet-18	Top-1 Train Acc	Top-3 Train Acc	Top-1 Test Acc	Top-3 Test Acc
Training From Scratch	0.9873	1	0.3	0.48
Finetuning Last Linear Layer	1	1	0.3	0.52
Finetuning Last Basic Block	1	1	0.3	0.54

- I think the numbers make sense since after fine tuning, the accuracy increases.

Problem 3: CURE-OR Classification Competition on Kaggle (35pts)

After filling in the table in problem 2, you have had some basic understandings and observations how these networks perform in `CURE-OR` dataset. In this problem, you need to choose and design your own training strategy to achieve better top-1 accuracy in a `Kaggle` competition.

Link: https://www.kaggle.com/competitions/funml-sp22-final-exam-problem-3/

- You should start with one of pretrained `alexnet` or `vgg16` or `resnet18`.
- You should only use the train data for training your network. You can use any loss function, optimizer, regularization, data augumentation strategy or modify model structure to help youself get better test results.
- You should explain your strategy and desgin in detail.
- Use the cell provided below to submit test result as a `.csv` file in `Kaggle`.
- In `Kaggle`, 50% of your result will be used to calculate the top-1 accuracy for the score on the leaderboard. The rest of 50% will be used to calculate your actual score after the deadline. This means the score on the leaderboard only gives you a sense of how you perform, and it will not be your final score.
- You have 5 times to submit result in `Kaggle` every day. Please use it wisely.
- 10 pts will be according to your strategy design and the rest of 25 pts depends on your ranking.

$$\text{score} = (50 - \text{your ranking} + 1) \times 0.5$$

Problem 3 Solution

My strategy:

- Since VGG16 has the best testing accuracy based on my previous part's result, I choose its pretrained model for this part.
- I fine tune all the linear layers in the model

1. Since VGG16 has the best testing accuracy based on my previous part's result, I choose its pretrained model for this part.
2. I fine tune all the linear layer in the model
3. To improve the accuracy more, I do these folloing modifications and finetune linear layers in the model multiple times:

- I added weight decay for L2 regularization.
- I added some randomnize image preprocessing.
- Increase epoch number to 100
- Decrease the learning rate to 1e-6

My Kaggle Name: tyu304

```
# problem 3 code here

########################################
# Part 0: More to preprocess
########################################

preprocess = transforms.Compose([
       transforms.Resize(256),
       transforms.CenterCrop(224),
```

```
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])
preprocess1 = transforms.Compose([
        transforms.RandomAutocontrast(0.5), #random contrast
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])
preprocess2 = transforms.Compose([
        transforms.RandomGrayscale(0.5), #random grayscale
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])
preprocess3 = transforms.Compose([
        transforms.RandomHorizontalFlip(0.5), #random flip
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])
preprocess4 = transforms.Compose([
        transforms.RandomInvert(0.5), #random invert
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])

class ConcatDataset(torch.utils.data.Dataset):
    def __init__(self, *datasets):
        self.datasets = datasets

    def __getitem__(self, i):
```

```
        return tuple(d[i] for d in self.datasets)

    def __len__(self):
        return min(len(d) for d in self.datasets)

# set up train loader
train_dataset = ImageFolder(root=train_directory+'/train_imgs', transform=preprocess)
train_dataset1 = ImageFolder(root=train_directory+'/train_imgs', transform=preprocess1)
train_dataset2 = ImageFolder(root=train_directory+'/train_imgs', transform=preprocess2)
train_dataset3 = ImageFolder(root=train_directory+'/train_imgs', transform=preprocess3)

trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=1, shuffle=True)
# trainloader = torch.utils.data.DataLoader(train_dataset1, batch_size=1, shuffle=True)
# trainloader = torch.utils.data.DataLoader(train_dataset2, batch_size=1, shuffle=True)
# trainloader = torch.utils.data.DataLoader(train_dataset3, batch_size=1, shuffle=True)

# set up test loader
test_dataset = ImageFolder(root=test_directory+'/test_imgs', transform=preprocess)
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=1, shuffle=False)
```

```
# problem 3 code here

########################################
# Part 1: training
########################################

import copy

# model = vgg16(pretrained=True)
# model.classifier[6] = torch.nn.Linear(in_features=4096, out_features=10, bias=True)

model.to('cuda') # send model to cuda

########################################################################################
best_acc = 0
```

```
# define loss
loss = nn.CrossEntropyLoss() ##TODO
# loss = nn.MSELoss()
# loss1 = nn.L1Loss()
lr = 10e-6 ##TODO

# set up optimizer
########################################################################################
params =  model.classifier[6].parameters()
# params =  model.parameters()
optimizer = torch.optim.SGD(params, lr = lr, weight_decay=0.001) ##TODO
########################################################################################

for epoch in range(50):
  model.train()
  epoch_acc = 0

  for i, (x, y) in enumerate(trainloader):
    optimizer.zero_grad()
    input = x.to('cuda')
    target = y.to('cuda')

    out= model(input)##TODO
    loss_val = loss(out, target)##TODO

    loss_val.backward()

    optimizer.step()

    if (out.argmax().item() == target.item()):
      epoch_acc = epoch_acc + 1

  print('Epoch: {} | Loss:{:0.4f}'.format(epoch, loss_val.item()))
  # print(epoch_acc)

  if epoch_acc >= best_acc:
    best_acc = epoch_acc
```

```
    best_model_wts = copy.deepcopy(model.state_dict())

model.load_state_dict(best_model_wts)
```

```
Epoch: 0 | Loss:0.0000
Epoch: 1 | Loss:0.0000
Epoch: 2 | Loss:0.0000
Epoch: 3 | Loss:0.0000
Epoch: 4 | Loss:0.0000
Epoch: 5 | Loss:0.0000
Epoch: 6 | Loss:0.0000
Epoch: 7 | Loss:0.0000
Epoch: 8 | Loss:0.0000
Epoch: 9 | Loss:0.0000
Epoch: 10 | Loss:0.0001
Epoch: 11 | Loss:0.0000
Epoch: 12 | Loss:0.0000
Epoch: 13 | Loss:0.0000
Epoch: 14 | Loss:0.0000
Epoch: 15 | Loss:0.0000
Epoch: 16 | Loss:0.0000
Epoch: 17 | Loss:0.0000
Epoch: 18 | Loss:0.0001
Epoch: 19 | Loss:0.0000
Epoch: 20 | Loss:0.0000
Epoch: 21 | Loss:0.0160
Epoch: 22 | Loss:0.0000
Epoch: 23 | Loss:0.0000
Epoch: 24 | Loss:0.0004
Epoch: 25 | Loss:0.0009
Epoch: 26 | Loss:0.0056
Epoch: 27 | Loss:0.0000
Epoch: 28 | Loss:0.0000
Epoch: 29 | Loss:0.0000
Epoch: 30 | Loss:0.0000
Epoch: 31 | Loss:0.0000
Epoch: 32 | Loss:0.0002
Epoch: 33 | Loss:0.0000
Epoch: 34 | Loss:0.0000
Epoch: 35 | Loss:0.0000
```

```
Epoch: 36 | Loss:0.0000
Epoch: 37 | Loss:0.0000
Epoch: 38 | Loss:0.0000
Epoch: 39 | Loss:0.0016
Epoch: 40 | Loss:0.0000
Epoch: 41 | Loss:0.0000
Epoch: 42 | Loss:0.0000
Epoch: 43 | Loss:0.0000
Epoch: 44 | Loss:0.0000
Epoch: 45 | Loss:0.0000
Epoch: 46 | Loss:0.0000
Epoch: 47 | Loss:0.0000
Epoch: 48 | Loss:0.0000
Epoch: 49 | Loss:0.0000
<All keys matched successfully>
```

```
# problem 3 code here

########################################
# Part 2: output scores
########################################
submission_list=[]

# Training set (part e)

out_1 = torch.empty(0) # intialize an empy tensor
out_3 = torch.empty(0) # intialize an empy tensor

out_1 = out_1.to('cuda')
out_3 = out_3.to('cuda')

# print(trainloader)


with torch.no_grad():
  for img, label in trainloader:

    img = img.to("cuda")
```

```
    label = label.to("cuda")

    y_post = model(img)
    # submission_list.append(y_post.argmax().item())

    top_1_vec = topk_accuracy(y_post, label, 1)
    top_3_vec = topk_accuracy(y_post, label, 3)

    out_1 = torch.cat((out_1, top_1_vec))
    out_3 = torch.cat((out_3, top_3_vec))

print('Training top-1 Score: {:0.4f}'.format(out_1.sum() / torch.numel(out_1)))
print('Training top-3 Score: {:0.4f}'.format(out_3.sum() / torch.numel(out_3)))

# Testing set (part e)

out_1 = torch.empty(0) # intialize an empy tensor
out_3 = torch.empty(0) # intialize an empy tensor

out_1 = out_1.to('cuda')
out_3 = out_3.to('cuda')

submission_list = []

with torch.no_grad():
  for i, (img, label) in enumerate(testloader):
    img = img.to("cuda")
    label = label.to("cuda")

    y_post = model(img)
    submission_list.append(y_post.argmax().item())
    # print(y_post.argmax())
    # print(label)

    top_1_vec = topk_accuracy(y_post, label, 1)
    top_3_vec = topk_accuracy(y_post, label, 3)
```

```
    out_1 = torch.cat((out_1, top_1_vec))
    out_3 = torch.cat((out_3, top_3_vec))
print('Test top-1 Score: {:0.4f}'.format(out_1.sum() / torch.numel(out_1)))
print('Test top-3 Score: {:0.4f}'.format(out_3.sum() / torch.numel(out_3)))
```

```
Training top-1 Score: 0.9972
Training top-3 Score: 1.0000
Test top-1 Score: 0.8400
Test top-3 Score: 0.9200
```

```
# use this cell to create .csv file for submission

import numpy as np

# convert submission_list to numpy
submission_np = np.array(submission_list).reshape(-1, 1)

# generate submission.csv from submission_np
np.savetxt('/content/submission.csv', np.concatenate((np.arange(len(submission_list)).reshape(-1, 1), submission_np), axis=1), delim
```

Problem 4: CURE-OR Anomaly Detection (35pts)

In problem 3 from assginment 6, we learned how to implement anomaly detection using autoencoders. In this problem, we will implement a `CURE-OR` anomaly detector using a classifier instead.

In this problem, we assume the first five labels are in-distribution (ID) and the remaining five labels are out-of-distribution (OoD). We will train our `Alexnet` using only ID data in the train set, so we will remove those OoD data from the training set during training. Once the model is trained, we can apply it on ID and OoD data in the test set. The idea is as follows: although the OoD test data will still be categorized into one of ID classes, its softmax probablity will be relativly low compared to ID test data. This happens because in general when it comes to ID data, the model has more confidence to correctly predict it. On the other hand, when it comes to OoD data, the model usually has lower confidence.

Hence, we can use this strategy to perform anomaly detection by choosing a threshold λ that:

$$\hat{y} = \begin{cases} 0 \text{ (normal)}, & \max(f(x)) \geq \lambda \\ 1 \text{ (anomaly)}, & \text{otherwise.} \end{cases}$$

where x is input image, $f(\cdot)$ is the model, $\hat{y}$ is the anomaly prediction. Implement this anomaly detection algorithm and plot the ROC plot using different λ values on test data.

You will have to perfrom the following steps and your code should reflect these steps: (1) assume the first five labels are in-distribution (ID) and the remaining five labels are out-of-distribution (OoD); (2) train `Alexnet` using only ID data in the train set, i.e., remove OoD data from the training set during training; (3) apply the model on ID and OoD data in the test set; and (4) plot the ROC curve.

Problem 4 Solution

```
# problem 4 code here

################################################################################
# Step (1) and (2) Training only ID data
################################################################################

# instantiate model object and
model =  alexnet()

# change last layer of model
model =  alexnet(num_classes = 10)

model.to('cuda') # send model to cuda

# define loss
loss = nn.CrossEntropyLoss()
lr = 10e-4
```

```
# set up optimizer
optimizer = torch.optim.SGD(model.parameters(), lr = lr)

for epoch in range(100):
  model.train()

  for i, (x, y) in enumerate(trainloader):
    optimizer.zero_grad()
    input = x.to('cuda')
    target = y.to('cuda')

    if (target > 4):  # Skipping all data of label 5~9
      # print("skip")
      continue

    out = model(input)
    loss_val = loss(out, target)

    loss_val.backward()

    optimizer.step()
    # print("trained")

  print('Epoch: {} | Loss:{:0.4f}'.format(epoch, loss_val.item()))
```

```
Epoch: 42 | Loss:0.4711
Epoch: 43 | Loss:0.7933
Epoch: 44 | Loss:2.8927
Epoch: 45 | Loss:2.5066
Epoch: 46 | Loss:0.0007
Epoch: 47 | Loss:0.0001
Epoch: 48 | Loss:0.1077
Epoch: 49 | Loss:0.0545
Epoch: 50 | Loss:0.8452
Epoch: 51 | Loss:0.1283
Epoch: 52 | Loss:0.0001
```

```
Epoch: 53 | Loss:0.0000
Epoch: 54 | Loss:0.0450
Epoch: 55 | Loss:0.0133
Epoch: 56 | Loss:0.6145
Epoch: 57 | Loss:3.2916
Epoch: 58 | Loss:0.0154
Epoch: 59 | Loss:0.0000
Epoch: 60 | Loss:0.0016
Epoch: 61 | Loss:0.0014
Epoch: 62 | Loss:0.0451
Epoch: 63 | Loss:0.0001
Epoch: 64 | Loss:0.6703
Epoch: 65 | Loss:0.7733
Epoch: 66 | Loss:0.0002
Epoch: 67 | Loss:0.0791
Epoch: 68 | Loss:0.7438
Epoch: 69 | Loss:0.1570
Epoch: 70 | Loss:0.0001
Epoch: 71 | Loss:0.0000
Epoch: 72 | Loss:0.0007
Epoch: 73 | Loss:0.0000
Epoch: 74 | Loss:0.0071
Epoch: 75 | Loss:0.0082
Epoch: 76 | Loss:0.0072
Epoch: 77 | Loss:0.0068
Epoch: 78 | Loss:0.0326
Epoch: 79 | Loss:0.0000
Epoch: 80 | Loss:0.0076
Epoch: 81 | Loss:0.5290
Epoch: 82 | Loss:0.0263
Epoch: 83 | Loss:0.0000
Epoch: 84 | Loss:0.0000
Epoch: 85 | Loss:0.0134
Epoch: 86 | Loss:0.0000
Epoch: 87 | Loss:0.0000
Epoch: 88 | Loss:0.0000
Epoch: 89 | Loss:0.1429
Epoch: 90 | Loss:0.0000
Epoch: 91 | Loss:0.0023
Epoch: 92 | Loss:0.0000
Epoch: 93 | Loss:0.0006
```

```
Epoch: 94 | Loss:0.0028
Epoch: 95 | Loss:0.0000
Epoch: 96 | Loss:0.0000
Epoch: 97 | Loss:0.0057
Epoch: 98 | Loss:0.0000
Epoch: 99 | Loss:0.0000
```

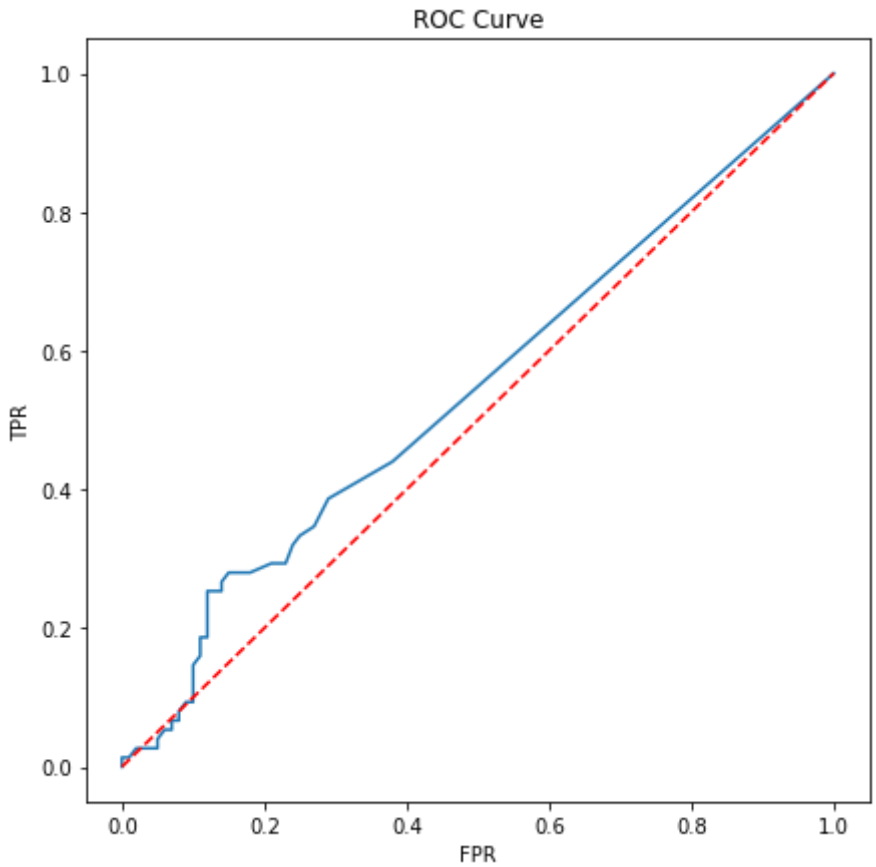
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
################################################################################
# (3) Put model on Test Data
################################################################################

pred_list = []
pred_label_list = []
y_test = []

with torch.no_grad():
  for i, (img, label) in enumerate(testloader):
    img = img.to("cuda")
    label = label.to("cuda")

    y_test.append(label.item())

    # train model and predict on test data
    y_post = model(img)

    out = torch.nn.functional.softmax(y_post, dim = 1)
    # print(out)
    pred, pred_label = out.max(dim=1)
    pred = pred.item()
    pred_label = pred_label.item()
```

```
    # pred_label = out.argmax(dim = 1).item()

    pred_list.append(pred)
    pred_label_list.append(pred_label)

    # print(y_post.max(dim=1))
    # print(out.max(dim=1))
    # pred_label = out.argmax(dim = 1).item() # obtain predicted label

    # Calculate TPR, FPR
# print(pred_list)
# print(pred_label_list)
# print(y_test)
```

```
################################################################################
# (4) Plot ROC Curve:
################################################################################

##### I implement anomaly detection as following based on the lecture slide:
# TPR = Anomaly Detected as Anomaly/ num_of_element
# FPR = Normal Detected as Anomaly/ num_of_element

th = np.linspace(-0.1, 1.1, 100)
# print(np.asarray(pred_list))
t = []
f = []

for ii in range(np.shape(th)[0]):
  # print(th[ii])
  # print(np.shape(np.reshape((np.asarray(pred_list) < th[ii]), [-1,1] )))
  # print(np.shape(np.reshape((np.array(y_test) > 4), [-1,1])))
  anomaly_detected = (np.asarray(pred_list) < th[ii]) & (np.array(y_test) > 4)
```

```
  normal_detected = (np.asarray(pred_list) < th[ii]) & (np.array(y_test) <= 4)

  # print(anomaly_detected)
  # print(normal_detected)

  truely_ID = np.sum(np.asarray(y_test) <= 4)
  truely_OoD = np.sum(np.asarray(y_test) > 4)

  TPR = np.zeros(np.shape(th)[0])
  FPR = np.zeros(np.shape(th)[0])

  TPR[ii] = np.sum(anomaly_detected)/truely_OoD
  t.append(np.sum(anomaly_detected)/truely_OoD)

  FPR[ii] = np.sum(normal_detected)/truely_ID
  f.append(np.sum(normal_detected)/truely_ID)

# print(t)
# print(f)

fig = plt.figure(figsize=(7,7))
plt.plot(f,t)
plt.plot([0,1],[0,1], linestyle='--', color='red')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC Curve')
plt.show()
```



- The performance is not great since we just use the pretrained model and randomly make label 0 ~ 4 ID and 5 ~ 9 Ood. However, we can see the curve following the general shape of an ROC curve.

Problem 5: CIOS Survey (Bonus)

If you include a proof (e.g., a screenshot without revealing your choices) that you completed the CIOS survey for the course, you will receive a bonus of 1% for the overall grade. This bonus opportunity will be available with your take-home final exam as well in case you do not have time

to complete the survey before the due date for hw#7. You will receive the bonus points once, either in hw#7 or in the final exam. More than that, if 90% of the class submits, everyone gets another 1%.

✓ 0s completed at 1:54 PM