COMP 448/548 – Medical Image Analysis
Homework #3

Due: 23:55, May 12, 2021

In this homework, you will design a convolutional neural network (CNN) classifier for colon tissue image classification. For that, you will make use of transfer learning by finetuning the pretrained AlexNet network on the tissue image dataset that you also used for your second homework.

What to submit: Submit the following via blackboard.

- Your source codes. Put all of your source codes in a zip file called ***Lastname_FirstName_HW3.zip***
- A maximum of 2-pages report. You may use the following guideline in writing your reports. It also gives you an outline for the experiments that you need to conduct. Similar to the previous assignments, you are expected to write your report neatly and properly. The format, structure, and writing style of your report as well as the quality of the tables will be a part of your grade. Use reasonable font sizes, spacing, margin sizes, etc. You may submit either a one-column or a double-column document. The filename of your report should be ***Lastname_FirstName_HW3.pdf***

Implementation details

Specify the platform that you used for your implementation.

Explain how you made use of the pretrained AlexNet to design your own classifier:

1. How did you make the input size compatible with the AlextNet network?
2. How did you normalize the input?
3. What parts of the AlextNet architecture did you modify? How did you modify the last layer?
4. What loss function did you use in backpropagation?
5. How did you select the parameters related to backpropagation? For example, did you use any optimizer? If so, what were the parameters of this optimizer and how did you select their values?
6. How did you address the class-imbalance problem?

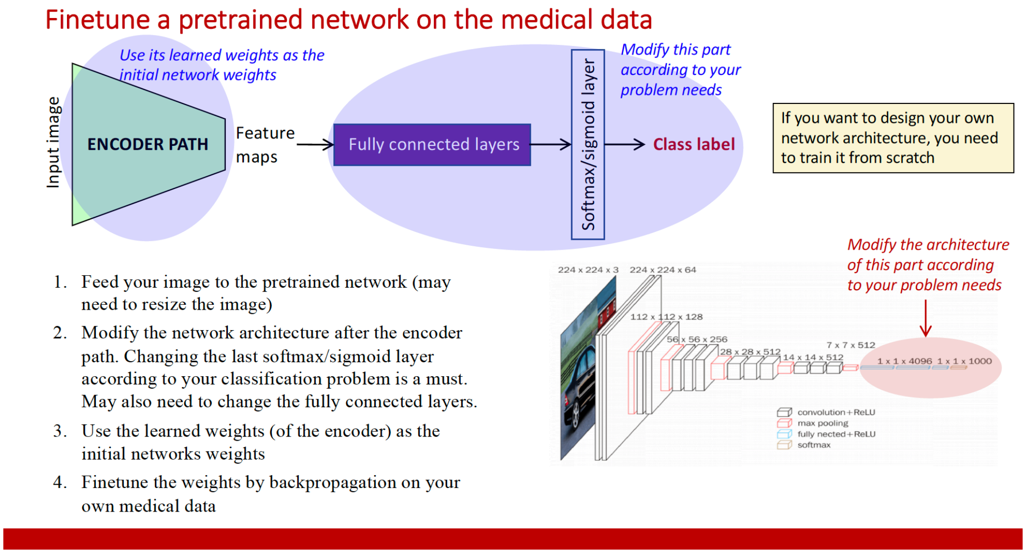
Additional comments, if you have any.

	Training portion of the training set				Validation portion of the training set				Test set			
	Class 1	Class 2	Class 3	Overall	Class 1	Class 2	Class 3	Overall	Class 1	Class 2	Class 3	Overall
With input normalization and with addressing the class-imbalance problem												
With input normalization and without addressing the class-imbalance problem												
Without input normalization and with addressing the class-imbalance problem												

You may make use of the following resources/references in the implementation your classifier (please use them partially or entirely if you think that they are useful, otherwise there is no requirement of using them). If you use any other resources, please give the necessary references.

1. The following paper in which the AlexNet network was proposed by Krizhevsky et al. in 2012. https://www.cs.toronto.edu/~kriz/imagenet_classification_with_deep_convolutional.pdf

2. Course slides (10_CNNs.pdf). The necessary slide is also copied below.



3. You may use any platform and/or any programming language in your implementation. Your TA, Batuhan Özyurt, has prepared a sample program in pytorch for your use. This program is incomplete on purpose. It is just to give you some idea and a starting point. You may use it partially or entirely. If you want to use it, you need to complete its missing parts (especially focus on ?? parts). You may ask your questions related to this program directly to your TA. However, do not ask questions such as "*what loss function should I use?*", "*how can I perform normalization?*", "*what will be the parameters of the optimizer?*", etc.

4. You may run your codes on any machine. However, if you do not have sufficient computational resources (server and/or GPU) to run your code, you may use Google Colab. Your TA has also prepared a document on the use of Google Colab. You may also ask your questions related to this document directly to your TA.

Transfer learning with TorchVision[1]

(*prepared by Batuhan Özyurt, bozyurt20@ku.edu.tr*)

This document includes an example of how you can employ the pretrained AlexNet network for an arbitrary image classification task.

Originally, AlexNet was designed for 1000-class classification task. When your classification task is different than this original task, you may use the pretrained weights of the previous convolutional layers of AlexNet. However, you have to modify the last fully connected layer(s) of the network so that it will work for the number of classes in your classification task. After this modification, you will not train the entire model from scratch. Instead, you will freeze the weights of the first layers (those before the last one) and have to train the network only for the layers that you will have modified.

Below are the code fragments that you may use in your implementation. Note that these codes are not complete; they are just to give you a starting point. Also note that if you'd prefer, you may ignore these fragments and write your own implementation in any platform from scratch.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
#use the GPU if available
```

Download the pretrained AlexNet model

```
model_conv = models.alexnet(pretrained = True)
```

Freeze the entire network before resetting the final layer. You need to set `requires_grad = False` to freeze the parameters so that the gradients are not computed in `backward()`.

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

Now reset the final layer. The parameters of newly constructed modules have `requires_grad = True` by default. Check the following documentation of torch.nn for more information: https://pytorch.org/docs/stable/nn.html. Note that you may use `print(model_conv)` to display the architecture of the model and find the index of the last layer(s).

```
model_conv.classifier[ ?? ] = nn.??( ?? )
model_conv = model_conv.to(device) # use the GPU
```

Choose an appropriate loss function. You may find more information in the torch.nn documentation.

```
criterion = nn.??
```

Select the optimizer for the stochastic gradient descent implementation. You may also find more information in https://pytorch.org/docs/master/generated/torch.optim.SGD.html

```
optimizer_conv = optim.SGD( ?? )
exp_lr_scheduler = lr_scheduler.StepLR( optimizer_conv, ?? )
```

[1] This document is prepared using https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

You may finetune this model using a function similar to **train_model** given below. The **train_model** function uses a portion of the training set to learn the weights and the remaining portion as the validation data to decide in which epoch the learned network is the best (this function will return this best network). These two portions will be referred as `'train'` and `'valid'`, respectively in the following function. You will use the network (model) returned by this function to classify images in the training and test sets and report the accuracies as explained above.

```
def train_model(image_datasets, model, criterion, optimizer,
                scheduler, num_epochs):

   dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x],
                 batch_size = 4, shuffle = True, num_workers = 4)
                 for x in ['train', 'valid']}

   best_model_wts = copy.deepcopy(model.state_dict())
   best_no_corrects = 0
   for epoch in range(num_epochs):
      # Set the model to the training mode for updating the weights using
      # the first portion of training images
      model.train()
      for inputs, labels in dataloaders['train']: # iterate over data
         inputs = inputs.to(device)
         labels = labels.to(device)
         optimizer.zero_grad()
         with torch.set_grad_enabled(True):
            outputs = model(inputs)
            _, preds = torch.max(outputs, ??)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

      # Set the model to the evaluation mode for selecting the best network
      # based on the number of correctly classified validation images
      model.eval()
      no_corrects = 0
      for inputs, labels in dataloaders['valid']:
         inputs = inputs.to(device)
         labels = labels.to(device)
         with torch.set_grad_enabled(False):
            outputs = model(inputs)
            _, preds = torch.max(outputs, ??)
            no_corrects += torch.sum(preds == labels.data)
      if no_corrects > best_no_corrects:
         best_no_corrects = no_corrects
         best_model_wts = copy.deepcopy(model.state_dict())

      scheduler.step()

   # Load the weights of the best network
   model.load_state_dict(best_model_wts)
   return model
```

The **train_model** function will expect image datasets as its input. For that, you need to load and preprocess the image data. Below are some code fragments that you may make use of. The following code (the **ImageFolder** function) assumes that the datasets are arranged in the following directory format. Note that you are given only the training and test datasets but you also need validation images for the network training. Thus, you need to select and put some portion of the training images into the valid directory. Also note that you need to select this portion considering the class labels as well.

```
main_dir/
    train/
        class-0/
            img1.jpg
            ...
        class-1/
            img2.jpg
            ...
        class-2/
            img3.jpg
            ...
    valid/
        class-0/
            img4.jpg
            ...
        class-1/
            img5.jpg
            ...
        class-2/
            img6.jpg
            ...
    test/
        class-0/
            img7.jpg
            ...
        class-1/
            img8.jpg
            ...
        class-2/
            img9.jpg
            ...
```

```
data_dir = '/content/gdrive/.../main_dir'
data_transforms = {
   'train': transforms.Compose([
            # put the input to Tensor format in order to use torch
            transforms.ToTensor(),
            transforms.Normalize( ?? )
   ]),
   'valid': ...
   'test': ...
}
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                    data_transforms[x]) for x in ['train', 'valid', 'test']}
```

Additionally, these are the imports you may find useful

```
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 matplotlib.pyplot as plt
import time
import os
import copy
```

How to use Google Colab

(*prepared by Batuhan Özyurt, bozyurt20@ku.edu.tr*)

Google Colab allows you to write and execute Python code in your browser with no configuration required, also with free access to GPUs. Training neural networks take shorter time on GPUs, so if you do not have a machine with a GPU, you can use Google Colab. If you have used Jupyter Notebook before, you can think of Google Colab as a Jupyter Notebook stored in Google Drive. You can learn the basics about Colab by watching this video.

You may follow the following steps to run your codes on Google Colab:

1. Go to your Google Drive.
2. Upload the dataset for the homework to your Google Drive.
3. Create a new Colab notebook by clicking on New → More → Google Colaboratory. The "new" button is on the top left corner, below the "Drive" logo.
4. Write your code on the notebook.
5. To use a GPU, click on "edit" at the bar above and then choose "notebook settings". Choose the hardware accelerator as GPU and save changes.
6. To use the dataset you uploaded to your Drive, you need to mount your Drive to Colab. To do that, add a new cell to your notebook by clicking on the "+ Code" button at the top and write the following code in the cell and run it:

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

7. Open the link.
8. Choose your Google account.
9. Allow access to your Google account.
10. Copy the code and paste it in the text box, and press enter.
11. While accessing the dataset, the directory address should start with `"/content/gdrive/..."`