COMP 4437 Artificial Neural Networks, Spring'24
Homework 2

Lecturer: Arman Savran

Question:	1	2	3	Total
Points:	50	30	20	100

Late submission: 10 points less grade for each day late. If you have a valid excuse, explain it to me and your submission may be accepted without a penalty (have a health report) or with some mild penalty.

Code of Honor: By submitting this assignment, you accept the honor code: "I affirm that wherever I received any help on this assignment, I have cited the source or the person clearly, otherwise I affirm that I have not given or received any unauthorized help on this assignment and that this work is my own." If you get help from your classmate but do not cite him/her, it is cheating and you and your friend can both get a big penalty. If you reference your friend, then your friend will not have any penalty and you may get a partial penalty or may not get a penalty at all depending on the help received. Copy-paste is always subject to some penalty. Similarly, you should reference your source if you get help from the internet.

Submission:

- **Notebook.** All of your solutions must be included in one notebook which must be submitted after converting it to an HTML file. Clean your intermediate results before submitting, do not have a messy submission. You are given plenty of time, so prepare a neat submission. You can use markdown cells for text explanations, latex formulas, or image embeddings. Don't forget to put the title as **HOMEWORK NO, your full name, and your student ID at the top of the notebook!**
- **References.** Cite all the sources that you benefited from, e.g., for explanation or code. It could be the URL of a webpage, or a book, your classmate, or some other person.

IMPORTANT: For **each question** below, you will work on the CIFAR-10 dataset, use the Cross-Entropy loss function, and compare the methods by reporting the final test accuracies as well as plotting the learning (training set) and the generalization (validation set) curves of the accuracies for your best solution. You must have at least 70% accuracy on the test set at each question. Higher accuracies are welcome.

1. **(50 points) Bottleneck residual block.** Explain the **COMPLETE** design of the bottleneck residual block used in ResNet (you can write or show by a figure). Recall that it consists of convolutional layers, ReLU, batch normalization, and a residual connection. Implement **exactly what you explained** in PyTorch by writing a class that accepts all the design parameters in its constructor. Then, implement your CNN model class that includes bottleneck residual blocks (you can include additional layers or blocks as well). Design, train, tune, and evaluate. Your training must include the learning and the generalization curves, and your evaluation must be on the test set, as explained above.
   - **Batch Size:** 128
   - **Epochs:** 20
   - **Regulizer:** L_2 regularization (MUST be explicitly implemented via a loss function)
   - **Optimizer:** SGD with momentum
   - **Learning rate:** fixed

Except for the above constraints, you are free in your design and for the hyperparameter values. Optimize your hyperparameters! Report your best training hyperparameters. Also, report your best architecture and its complexity in your notebook by using the torchinfo package (**10 points**).

Suggestion: For quicker development and optimization, you can apply the easy-to-hard progressive strategy (start with a smaller model, a training subset, fewer iterations, fewer epochs, coarser parameter search, etc.). Be careful with underfitting (can be detected on the learning curve) and overfitting (can be detected on the learning + generalization curve) problems.

You can use or modify the visualization function below to display some example prediction results in your report or to monitor during training (to monitor you need to use the axis argument not to display images on the same plot by update instead of creating a new figure).

Use the below code for all the questions:

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, random_split
import torchvision
from torchvision import models, transforms, datasets
import matplotlib.pyplot as plt
import numpy as np
import time

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

# Load and prepare the dataset
transform = 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])])

trn_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
vld_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
tst_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

# Split the training set into training and validation partitions
trn_size = int(0.8 * len(trn_dataset))
vld_size = len(trn_dataset) - train_size
torch.manual_seed(0)
trn_dataset, vld_dataset = random_split(trn_dataset, [trn_size, vld_size])

classes = 'Airplane', 'Car', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck'
num_classes = len(classes)


batch_size = 128
trn_loader = DataLoader(trn_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
vld_loader = DataLoader(vld_dataset, batch_size=batch_size, shuffle=False, num_workers=2)
tst_loader = DataLoader(tst_dataset, batch_size=batch_size, shuffle=False, num_workers=2)


def visualize_model_predictions(model, loader=tstloader, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()
    with torch.no_grad():
        for i, (inputs, labels) in enumerate(loader):
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title('predicted: ' + classes[preds[j]])
                imshow(inputs.cpu().data[j])
                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
    model.train(mode=was_training)
```

2. **Different training dynamics** You will use your final best architecture in Question 1 but will change your training dynamics. Keep all the hyperparameters the same except below. Don't forget to report the training curves, test set evaluation, and the optimized hyperparameters.

   Note: If you could not do the Question 1, you can use the model of Question 3, but repeating with both SGD + L_2 + fixed learning rate as well.

   (a) **(10 points) ADAM with weight decay**. Instead of SGD and L_2 regularization, use ADAM with weight decay. Optimize these new hyperparameters together with the learning rate. Is the same learning rate that you used before with SGD still very good for ADAM or not?

   (b) **(20 points) Learning rate scheduling**. Continue using ADAM with weight decay and add the scheduler, **ReduceLROnPlateau** PyTorch class, to automatically reduce the learning rate when the validation accuracy stops improving. Look at the documentation (in torch.optim.lr_scheduler) to learn about this PyTorch class. The critical argument is the "mode" argument. Set it correctly (**no points if incorrect**). Since you will not apply many epochs, set the patience number to a small number, for example, to 2. Naturally, you should run the scheduler after **each validation** evaluation. Together with the learning and generalization curves, plot the evolution of the learning rate over epochs. You can get the value of the learning rate at each epoch as below.

   ```
   current_lr = optimizer.param_groups[0]["lr"]
   ```

   You can tune the learning rate or use the same learning rate as the previous part.

3. **(20 points) Transfer learning.** You will apply the pre-trained ResNet-50 that was trained in ImageNet. You can download it and freeze all the layers as shown below. Don't forget to report the training curves, test set evaluation, and the optimized hyperparameters.

   ```
   from torchvision import models
   model = models.resnet50(pretrained=True)

   # To freeze all the layers (to prevent update of all the parameters)
   for param in model.parameters():
       param.requires_grad = False
   ```

   When you change some layers, by default they are trainable, so no need to unfreeze them.

   Only modify and train the last layer of ResNet-50 so that it can classify the CIFAR-10 classes.