Pattern Recognition: Homework 11

Due date: 2023.5.16

Build a Real ResNet (100 pt)

Deep convolutional neural networks (CNNs) have achieved remarkable success in various computer vision tasks. One common approach to improving the performance of these networks is to increase their depth, which has given rise to architectures like ResNet.

In `cifar10_tutorial.ipynb`, we have provided the code of a simplified version of ResNet with a single-stage structure (the resolution of feature map and the number of channels do not change for each block) that contains multiple residual blocks, which is something like:

```
class BasicBlock(nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.conv1 = nn.Conv2d(dim, dim, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(dim)
        self.conv2 = nn.Conv2d(dim, dim, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(dim)
        self.shortcut = nn.Sequential()

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(self, n_blocks, num_classes=10):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.blocks = nn.Sequential(*[
            BasicBlock(dim=32) for _ in range(n_blocks)
        ])
        self.linear = nn.Linear(32, num_classes)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.blocks(out)
        out = F.avg_pool2d(out, 32)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```

```

net = ResNet(n_blocks=6).cuda()
```

However, in practice, the ResNet architecture is actually a multi-stage structure with multiple resolutions. Multi-stage architectures have been shown to provide better performance by allowing the network to capture features at different scales, thereby offering a more comprehensive understanding of the input data.

1 Task Description

In this assignment, you will be able to build a **real** ResNet with multi-stage architecture by extending the previous single-stage structure. Specifically, a real resnet has the following stages:

- Stage 1: resolution=32x32, num_channels=16
- Stage 2: resolution=16x16, num_channels=32
- Stage 3: resolution=8x8, num_channels=64

The key of this architecture is the transitioning between stages, where the resolution is reduced by a factor of 2 and the number of channels are multiplied by a factor of 2. And this is achieved by a class called `DownSamplingBlock`, and **implementing this is the core of your task**. You may find the following PyTorch code snippet helpful for implementing the downsampling block:

```
class DownSamplingBlock(nn.Module):
    def __init__(self, dim_in, dim):
        super().__init__()
        assert int(2 * dim_in) == dim  # the number of channels are multiplied by
    ↪ a factor of 2
        self.conv1 = nn.Conv2d(dim_in, dim, kernel_size=3, padding=1, stride=2)
        self.shortcut = nn.Sequential(
            nn.Conv2d(dim_in, dim, kernel_size=1, stride=2, bias=False),
            nn.BatchNorm2d(dim)
        )
        # ...
```

2 Assignment Objectives

To complete this assignment, you should accomplish the following objectives:

1. Prepare the **CIFAR-100** (not CIFAR-10!) dataset and the dataloader.
2. **Finish the multi-stage ResNet.**
3. Define the optimizer and loss function: please use Adam optimizer with `learning_rate=0.001` and `CrossEntropyLoss` as the loss function.
4. Write the training loop and perform training for **10** epochs.
5. Evaluate the model on the test dataset.

We have provided the template code in `main.py` with the code that requires you to finish marked in TODO. Note that only the second task needs careful implementation. The other parts can be finished by directly copying the code from `cifar10_tutorial.ipynb` in lecture 10 with little changes (*e.g.*, change the dataset).

3 Submission Form

Your submission should include:

1. Screenshot of each part of the code that you have finished (5 screenshots in total).
2. The mean training loss of each epoch.
3. The accuracy of the network on the 10000 test images (which should be more than 40%).
4. The complete code of `main.py`.

Bonus (20 pt)

Do you want to know how much a multi-stage version can improve the performance of ResNet? You can test the performance of the original single-stage ResNet (provided in `cifar10_tutorial.ipynb`, remember to change `num_classes` variable) on CIFAR-100 and compare it with the performance of the multi-staged version.

Note:

1. You should use the same hyperparameters as the multi-stage version, including the same optimizer, learning rate, loss function, and training epochs, etc.

2. You should report the mean training loss of each epoch of the single-staged ResNet and the final accuracy of the network on the 10000 test images.

3. Based on the results, draw a conclusion as to why the multi-stage version is better than the single-stage one: Does the multi-stage ResNet improve performance by 1) reducing overfitting or 2) reducing underfitting?