CS398-Deep Learning
Homework 5
Lingyi Xu (lingyix2)

1. Description of Implementation

Part1:

First, I load and prepare the training data and test data of CIFAR100. When loading data, I used Transforms to normalize and enlarge the dataset to help training.

```
train_transform = transforms.Compose([transforms.RandomCrop(32, padding=4), transforms.RandomHorizontalFlip(p=0.5), transforms.ToTensor(), transforms.Normalize(rgb_mean, rgb_std),])
test_transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(rgb_mean, rgb_std)])

trainset = torchvision.datasets.CIFAR100(root='~/scratch/Data4', train=True, download=True, transform=train_transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR100(root='~/scratch/Data4', train=False, download=True, transform=test_transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=128, shuffle=False, num_workers=2)
```

Then, I define the basic convolutional block for resNet

```
class convBlock(nn.Module):
    def __init__(self, inChannel, outChannel, instride, inpadding):
        super(convBlock, self).__init__()
        self.convB1 = nn.Conv2d(inChannel, outChannel, kernel_size=3,stride=instride, padding=inpadding)
        self.convB2 = nn.Conv2d(outChannel, outChannel, kernel_size=3,stride=1, padding=1)
        self.bnB = nn.BatchNorm2d(outChannel)
        self.shortcut = nn.Sequential()
        if inChannel != outChannel or instride != 1:
            self.shortcut = nn.Sequential(nn.Conv2d(inChannel, outChannel,kernel_size=1, stride=instride),nn.BatchNorm2d(outChannel))

    def forward(self, x):
        out = self.bnB(self.convB1(x))
        out = F.relu(out)
        out = self.bnB(self.convB2(out))
        shortcut = self.shortcut(x)
        out = out + shortcut
        return out
```

Then, I define and build the resNet network.

```
class ResNetNetwork(nn.Module):
    def __init__(self):
        super(ResNetNetwork, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.bn = nn.BatchNorm2d(32)
        self.drop = nn.Dropout(p=0.2)

        self.conv21 = convBlock(32, 32, instride=1, inpadding=1)
        self.conv22 = convBlock(32, 32, instride=1, inpadding=1)
        self.conv31 = convBlock(32, 64, instride=2, inpadding=1)
        self.conv32 = convBlock(64, 64, instride=1, inpadding=1)
        self.conv33 = convBlock(64, 64, instride=1, inpadding=1)
        self.conv34 = convBlock(64, 64, instride=1, inpadding=1)
        self.conv41 = convBlock(64, 128, instride=2, inpadding=1)
        self.conv42 = convBlock(128, 128, instride=1, inpadding=1)
        self.conv43 = convBlock(128, 128, instride=1, inpadding=1)
        self.conv44 = convBlock(128, 128, instride=1, inpadding=1)
        self.conv51 = convBlock(128, 256, instride=2, inpadding=1)
        self.conv52 = convBlock(256, 256, instride=1, inpadding=1)

        self.fc = nn.Linear(256*2*2, 100)
```

```
def forward(self, x):

    x = self.conv1(x)
    x = self.bn(x)
    x = F.relu(x)

    x = self.drop(x)

    x = self.conv21(x)
    x = self.conv22(x)

    x = self.conv31(x)
    x = self.conv32(x)
    x = self.conv33(x)
    x = self.conv34(x)

    x = self.conv41(x)
    x = self.conv42(x)
    x = self.conv43(x)
    x = self.conv44(x)

    x = self.conv51(x)
    x = self.conv52(x)

    x = F.max_pool2d(x, kernel_size=3, stride=2, padding=1)
    x = x.view(-1, 256*2*2)
    x = self.fc(x)

    return x
```

Train the model with GPU acceleration. For this particular dataset, I use the following parameters:

Iterations: 40
Learning rate = 0.001*0.9^num_iterations

And trained the model on the dataset

```
with torch.no_grad():
    for tes_data in testloader:
        tes_image, tes_lab = tes_data
        tes_lab = tes_lab.to(torch.device("cuda"))
        tes_out = res(tes_image)
        _, tes_predicted = torch.max(tes_out.data, 1)
        test_size += tes_lab.size(0)
        test_acc += (tes_predicted == tes_lab).sum().item()

train_accuracy = train_count/train_size
test_accuracy = test_acc/test_size
print('epoch number: '+str(epoch+1))
print('test accuracy: '+str(test_accuracy))
```

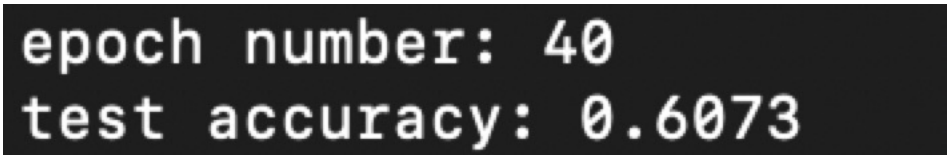
Part2:

The implementation of part 2 is mostly same as part 1. The only difference is that we need to load the pretrained model:

```
file_dir = os.getcwd() + '/' + 'resnet18-5c106cde.pth'
def resnet18(pretrained=True):
    model = torchvision.models.resnet.ResNet(torchvision.models.resnet.BasicBlock, [2, 2, 2, 2])
    if pretrained:
        model.load_state_dict(torch.utils.model_zoo.load_url(file_dir, model_dir='./'))
    return model
```

2. **Final Test Accuracy:**

Part1: 0.6073



Part2: 0.7245