Computer Vision
Homework 3: Big vs Small Models

Problem1

First model is resnet18 and weights is None.

Later Change model to resnet50, and weights is IMAGENET1.

Change the output of the model to 10 class.

```
# HINT: Remember to change the model to 'resnet50' and the weights to weights="IMAGENET1K_V1" when needed.
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50', weights="IMAGENET1K_V1")
#model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet18', weights=None)
num_fc = model.fc.in_features
model.fc = nn.Linear(num_fc, 10)
# Background: The original resnet18 is designed for ImageNet dataset to predict 1000 classes.
# TODO: Change the output of the model to 10 class.
```

Here I am going to make the training model. Set the variable “learning rate” to 0.01. Input shape is (-1,3,32,32) because the structure of the model need to train on multiples of three. Epoch set to 10.

```
# TODO: Fill in the code cell according to the pytorch tutorial we gave.
LR=0.001
optimizer = torch.optim.Adam(model.parameters(), lr=LR)
loss_func = nn.CrossEntropyLoss()
input_shape = (-1,3,32,32)
for epoch in range(5):
    model.train()
    correct_train = 0
    total_train = 0
    for step, (x, y) in enumerate(train_dataloader):
        b_x = Variable(x, requires_grad=False)
        b_y = Variable(y, requires_grad=False)
        out = model(b_x)
        loss = loss_func(out, b_y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        predicted = torch.max(out.data, 1)[1]
        total_train += len(b_y)
        correct_train += (predicted == b_y).float().sum()
    train_accuracy = 100 * correct_train / float(total_train)
    print('Epoch: {} | accuracy: {}% | Loss: {}'.format(epoch + 1, train_accuracy, loss))
```

Problem2

resnet50

sixteenth_train_dataloader	half_train_dataloader	train_dataloader
Epoch: 1 | accuracy: 67.68000030517578% | Loss: 1.6098675727844238 Epoch: 2 | accuracy: 64.92800140380096% | Loss: 1.3581033945083618 Epoch: 3 | accuracy: 65.40799713134766% | Loss: 1.1370474100112915 Epoch: 4 | accuracy: 65.47200012207031% | Loss: 0.8004204630851746 Epoch: 5 | accuracy: 67.87200164794922% | Loss: 0.9551518559455872 sixteenth_train_dataloader | accuracy: 74.23000335693359%	Epoch: 1 | accuracy: 78.25199890136719% | Loss: 0.7256434559822083 Epoch: 2 | accuracy: 76.05200195312500% | Loss: 0.8472979664802551 Epoch: 3 | accuracy: 79.10800170898438% | Loss: 0.6622571349143982 Epoch: 4 | accuracy: 81.09999847412109% | Loss: 0.4149971902370453 Epoch: 5 | accuracy: 83.48000335693359% | Loss: 0.3739603459835052 half_train_dataloader | accuracy: 83.23000335693359%	Epoch: 1 | accuracy: 82.44999694824219% | Loss: 0.4391198158264160 Epoch: 2 | accuracy: 85.78600311279297% | Loss: 0.3361054062843323 Epoch: 3 | accuracy: 84.40399932861328% | Loss: 0.5226349732398987 Epoch: 4 | accuracy: 75.64600372314453% | Loss: 0.4834958109402776 Epoch: 5 | accuracy: 83.19999694824219% | Loss: 0.4919958114624023 train_dataloader | accuracy: 84.83999633789062%

resnet18

sixteenth_train_dataloader	half_train_dataloader	train_dataloader
Epoch: 1 | accuracy: 20.927999496459961% | Loss: 2.173978090286255 Epoch: 2 | accuracy: 31.968000411987305% | Loss: 1.8966121673583984 Epoch: 3 | accuracy: 35.327999114990234% | Loss: 1.6123600006103516 Epoch: 4 | accuracy: 37.183998107910156% | Loss: 1.6879534721374512 Epoch: 5 | accuracy: 40.223999023437500% | Loss: 1.6870961189270020 sixteenth_train_dataloader | accuracy: 46.06999969482422%	Epoch: 1 | accuracy: 46.340000152587891% | Loss: 1.3304163217544556 Epoch: 2 | accuracy: 54.860000610351562% | Loss: 1.2427078485488892 Epoch: 3 | accuracy: 58.627998352050781% | Loss: 0.9849798083305359 Epoch: 4 | accuracy: 62.768001556396484% | Loss: 1.0059294700622559 Epoch: 5 | accuracy: 64.767997741699219% | Loss: 1.0151878595352173 half_train_dataloader | accuracy: 68.15000152587891%	Epoch: 1 | accuracy: 42.734001159667969% | Loss: 1.3658219146728516 Epoch: 2 | accuracy: 56.987998962402344% | Loss: 0.9367364645004272 Epoch: 3 | accuracy: 63.021999359130859% | Loss: 1.3379069972038269 Epoch: 4 | accuracy: 66.613998413085938% | Loss: 0.8827414512634277 Epoch: 5 | accuracy: 69.174003601074219% | Loss: 0.6846130490303040 train_dataloader | accuracy: 71.76999664306641%

Problem3

Best performance

Model: resnet18 to training and weights is None

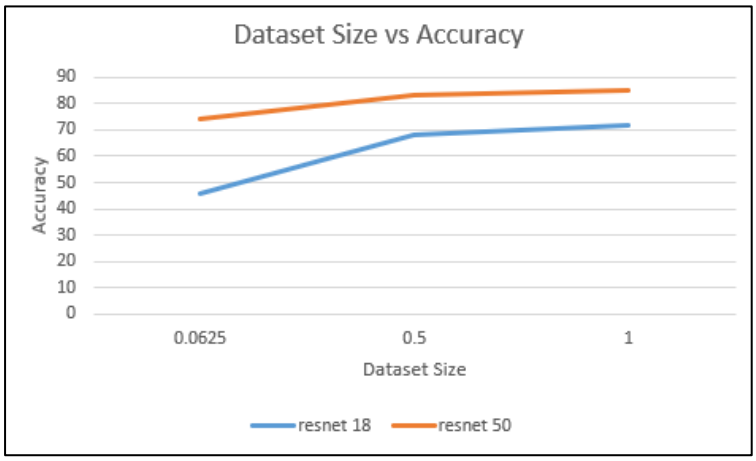
Train data: train_dataloader

Epoch: 10

```
Epoch: 1 | accuracy: 43.27799987792969% | Loss: 1.1996657848358154
Epoch: 2 | accuracy: 57.70000076293945% | Loss: 1.036866307258606
Epoch: 3 | accuracy: 63.104000091552734% | Loss: 1.058481216430664
Epoch: 4 | accuracy: 66.60600280761719% | Loss: 0.9383122324943542
Epoch: 5 | accuracy: 69.93800354003906% | Loss: 1.035589575767517
Epoch: 6 | accuracy: 71.94000244140625% | Loss: 0.7491894960403442
Epoch: 7 | accuracy: 73.54799652099961% | Loss: 0.8618677854537964
Epoch: 8 | accuracy: 75.20999908447266% | Loss: 0.7155677080154419
Epoch: 9 | accuracy: 76.20400238037711% | Loss: 0.44183674454689026
Epoch: 10 | accuracy: 77.42600250244141% | Loss: 0.435247004032135
train_dataloader | accuracy: 77.27999877929688%
```

Discussion1

Model resnet50 with pretrained is better than resnet18 in all dataset size. Bigger dataset size can get better accuracy. Resnet50 is a more powerful architecture and performs better. Resnet18 may be a more practical choice for simpler tasks or when computational resources are limited. But if resnet50 without pretrained, final result won't be better than resnet50, so I choose resnet18 without pretrained to implement problem 3.



	0.0625	0.5	1
resnet 18	46.06	68.15	71.76
resnet 50	74.23	83.23	84.83

Discussion2

The weights parameter in the context of deep learning models typically refers to the model with pre-trained weights or train it from scratch. If we have limited data or computational resources and want to benefit from pre-trained features, using pre-trained weights from a model like "IMAGENET1K_V1" is a common and beneficial choice. Pretrained models is a common strategy in many computers vision tasks, it allows us to leverage the knowledge learned from a large and diverse dataset and adapt it to a specific problem. Requiring less training data and fewer training iterations

compared to training from scratch.