Computer Vision hw3

110065502 李杰穎

Problems

1. Finish the rest of the codes for Problem 1 and Problem 2 according to the hint.

```python
# TODO: Change the output of the model to 10 class.
model.fc = nn.Linear(model.fc.in_features, 10)
model = model.to(device)
```

```python
# TODO: Fill in the code cell according to the pytorch tutorial we gave.
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

```python
def train(dataloader, model, loss_fn, optimizer):
    num_batches = len(dataloader)
    size = len(dataloader.dataset)
    epoch_loss = 0
    correct = 0

    model.train()

    for X, y in tqdm(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()
        pred = pred.argmax(dim=1, keepdim=True)
        correct += pred.eq(y.view_as(pred)).sum().item()

    avg_epoch_loss = epoch_loss / num_batches
    avg_acc = correct / size

    return avg_epoch_loss, avg_acc
```

```python
def test(dataloader, model, loss_fn):
    num_batches = len(dataloader)
    size = len(dataloader.dataset)
    epoch_loss = 0
    correct = 0

    model.eval()

    with torch.no_grad():
        for X, y in tqdm(dataloader):
            X, y = X.to(device), y.to(device)

            pred = model(X)

            epoch_loss += loss_fn(pred, y).item()
            pred = pred.argmax(dim=1, keepdim=True)
            correct += pred.eq(y.view_as(pred)).sum().item()

    avg_epoch_loss = epoch_loss / num_batches
    avg_acc = correct / size

    return avg_epoch_loss, avg_acc
```

```python
epochs = 50
train_loss_list = []
train_acc_list = []
test_loss_list = []
test_acc_list = []
best_acc = 0

for epoch in range(epochs):
    train_loss, train_acc = train(train_dataloader, model, loss_fn, optimizer)
    # train_loss, train_acc = train(
    #     sixteenth_train_dataloader, model, loss_fn, optimizer)
    # train_loss, train_acc = train(
    #     half_train_dataloader, model, loss_fn, optimizer)

    test_loss, test_acc = test(valid_dataloader, model, loss_fn)

    if test_acc > best_acc:
        best_acc = test_acc
        # torch.save(model.state_dict(), 'resnet18_all_IMAGENET.pth')
        # torch.save(model.state_dict(), 'resnet18_sixteenth_IMAGENET.pth')
        # torch.save(model.state_dict(), 'resnet18_half_IMAGENET.pth')
        torch.save(model.state_dict(), 'resnet50_all_IMAGENET.pth')
        # torch.save(model.state_dict(), 'resnet50_sixteenth_IMAGENET.pth')
        # torch.save(model.state_dict(), 'resnet50_half_IMAGENET.pth')

    train_loss_list.append(train_loss)
    train_acc_list.append(train_acc)
    test_loss_list.append(test_loss)
    test_acc_list.append(test_acc)

    print(f"Epoch {epoch + 1:2d}: Loss = {train_loss:.4f} Acc = {train_acc:.2f} Test_Loss = {test_loss:.4f} Test_Acc = {test_acc:.2f}")
print("Done!")
print(f"Best Accuracy: {best_acc:.2f}")
```

3. Achieve the best performance given all training data using whatever model and training strategy.

Model: ConvNeXt_Base

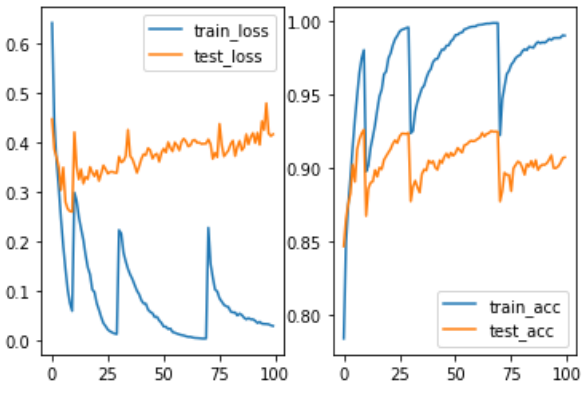
Optimizer: AdamW

Scheduler: CosineAnnealingWarmRestarts

Accuracy: 93%

```python
import torchvision.models
model = models.convnext_base(weights=models.ConvNeXt_Base_Weights.IMAGENET1K_V1)
model.classifier[2] = nn.Linear(model.classifier[2].in_features, 10)
model = model.to(device)
```

```python
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.AdamW(model.parameters(), lr=1e-3, weight_decay=1e-4)
scheduler = optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer, T_0=10, T_mult=2, eta_min=1e-6)
```

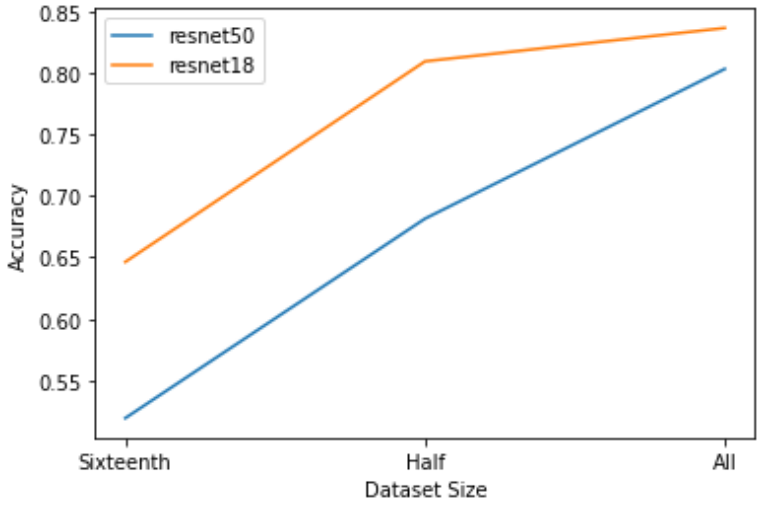


Discussion

- The relationship between the accuracy, model size, and the training dataset size.

The larger the amount of data, the higher the accuracy.

The larger the size of the model, accuracy is not necessarily higher.



- What if we train the ResNet with ImageNet initialized weights (weights="IMAGENET1K V1"), how would the relationship change?

In general, using pretrained weight will achieve better accuracy, and the deeper model will train better than the shallow model.