Computer Vision

HW3 Big vs Small Models

學號:112061619

姓名:王證皓

1. Codes

(1) Problems 1

ResNet18

```
# 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', 'resnet18', weights=None)

# Background: The original resnet18 is designed for ImageNet dataset to predict 1000 classes.
# TODO: Change the output of the model to 10 class.
model.fc=nn.Linear(in_features=512,out_features=10,bias=True)
model=model.to(device)
```

ResNet50

```
# 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=None)

# Background: The original resnet18 is designed for ImageNet dataset to predict 1000 classes.
# TODO: Change the output of the model to 10 class.
model.fc=nn.Linear(in_features=2048,out_features=10,bias=True)
model=model.to(device)
```

此處因需修改輸出至 10，所以僅需修改最後的全連接層，

而 resnet18 有 512 個輸出，resnet50 有 2048 個輸出

(2) Problems 2

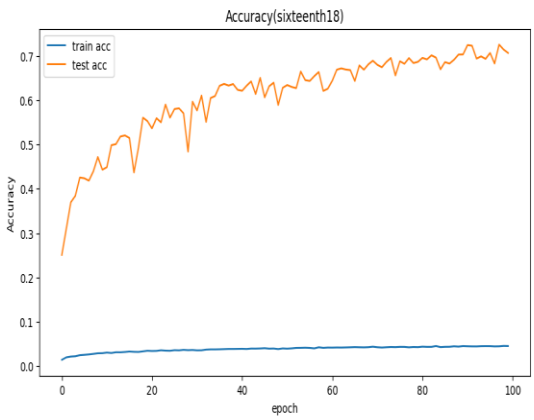
```
# TODO: Fill in the code cell according to the pytorch tutorial we gave.
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
def train(dataloader, model, loss_fn, optimizer):
    num_batches = len(dataloader)
    size = len(dataloader.dataset)
    epoch_loss = 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
```

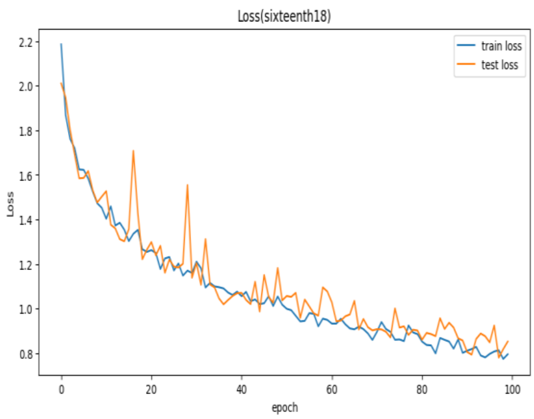
```python
def test(dataloader, model, loss_fn):
        num_batches = len(dataloader)
        size = len(dataloader.dataset)
        epoch_loss = 0
        correct = 0

        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
epochs = 100
train_acc_plot = []
test_acc_plot = []
train_loss_plot = []
test_loss_plot = []
for epoch in range(epochs):
        train_loss, train_acc = train(sixteenth_train_dataloader, model, loss_fn, optimizer)
        test_loss, test_acc = test(valid_dataloader, model, loss_fn)
        print(f"Epoch {epoch + 1:2d}: Loss = {train_loss:.4f} Acc = {train_acc:.2f} Test_Loss = {test_loss:.4f} Test_Acc = {test_acc:.2f}")
        train_acc_plot.append(train_acc)
        test_acc_plot.append(test_acc)
        train_loss_plot.append(train_loss)
        test_loss_plot.append(test_loss)
print("Done!")
plt.figure(figsize=(10,5))
plt.title("Accuracy")
plt.plot(train_acc_plot,label="train acc")
plt.plot(test_acc_plot,label="test acc")
plt.xlabel("epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure(figsize=(10,5))
plt.title("Loss")
plt.plot(train_loss_plot,label="train loss")
plt.plot(test_loss_plot,label="test loss")
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

2. Accuracy & Loss (epoch=100)

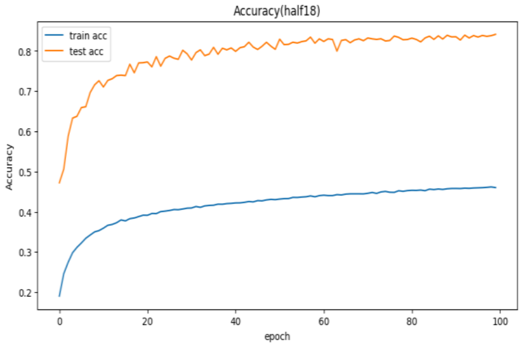
(1) Small model(ResNet18)

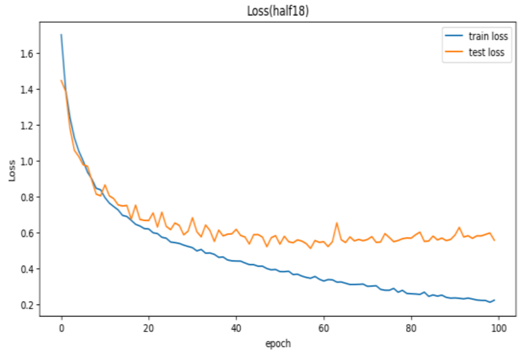
i. Sixteenth



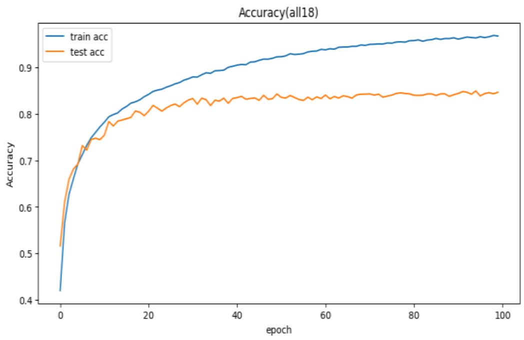


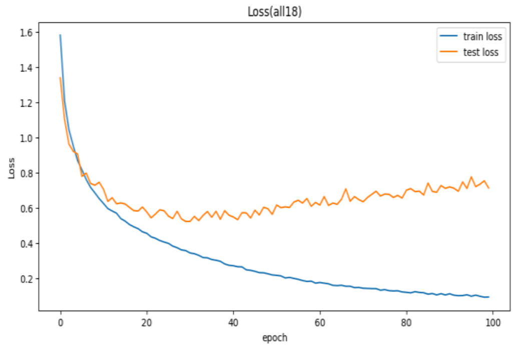
ii. half





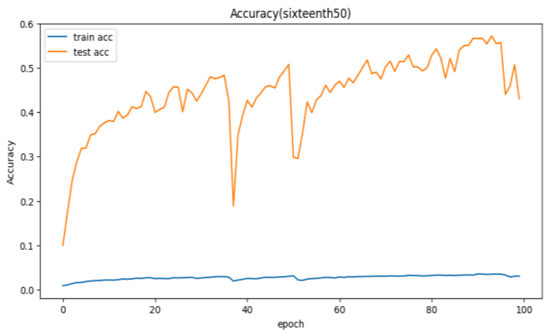
iii. all

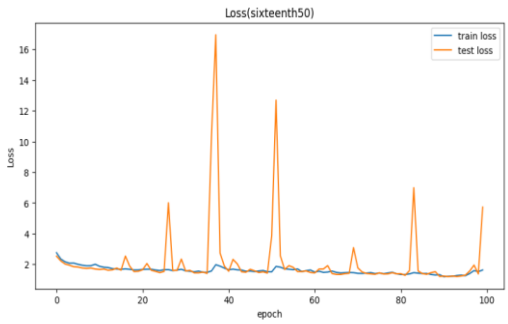




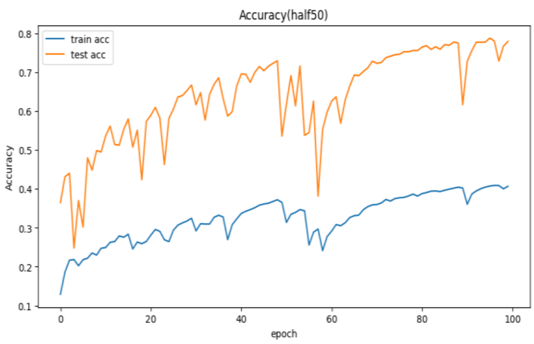
(2) Big model(ResNet50)

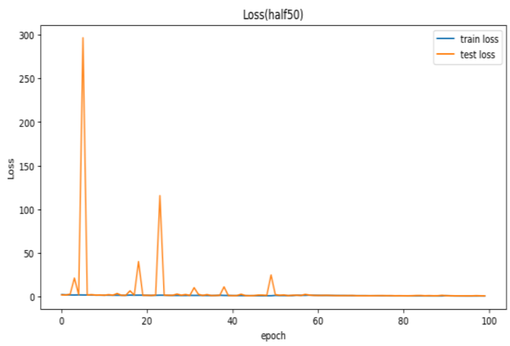
i. Sixteenth



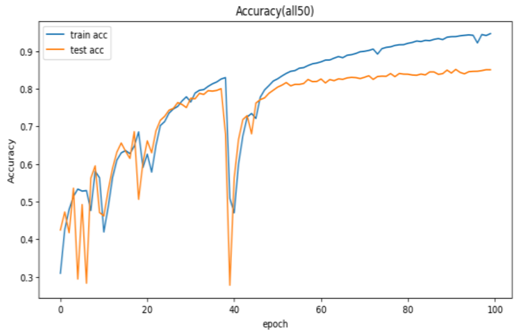


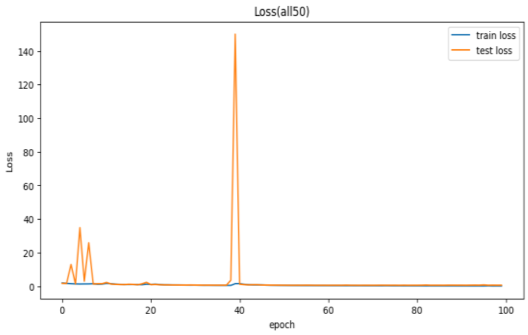
ii. half





iii. all



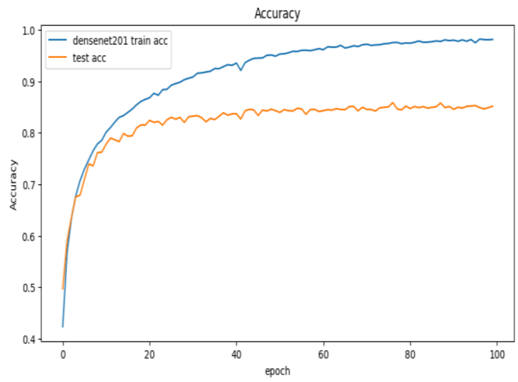


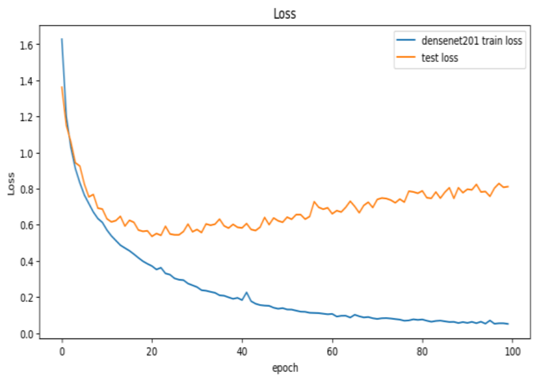
3. Best Performance

DenseNet-201

```
# 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', 'densenet201', weights=None)

# Background: The original resnet18 is designed for ImageNet dataset to predict 1000 classes.
# TODO: Change the output of the model to 10 class.
model.fc=nn.Linear(in_features=1920,out_features=10,bias=True)
model=model.to(device)
```

Accuracy & Loss



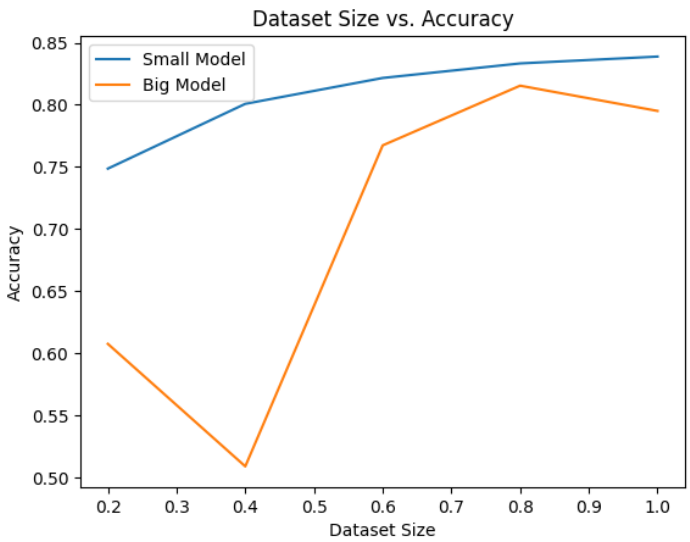



```
Epoch 100: Loss = 0.0509 Acc = 0.98 Test_Loss = 0.8111 Test_Acc = 0.85
Done!
```

此處使用了 DenseNet-201 這個模型，輸出層為 1920，並且從數據中可看出 DenseNet-201 的 Accuracy 在 epoch=100 時來到了 0.98，遠高出 test_acc 的 0.85，Loss 也較低，所以採用此套模型來達到更好的效能。

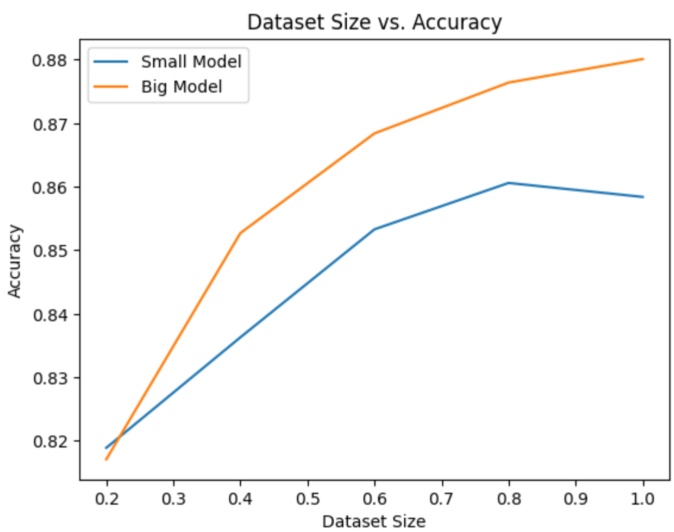
4. Discussion

(1) Relationship



由圖可知，當 Dataset size 增加時，Accuracy 也會隨之增加，但 Accuracy 並不會因為較大的 model 而遞增，不過可看出 Small Model 與 Big Model 的差距隨 Dataset size 增加而減少，顯示 Big Model 在 Dataset size 較大時，能發揮得更好。

(2) ImageNet initialized weights



由上圖可知，ResNet18 在處理較小的 dataset size 時，效能會優於 ResNet50，且訓練時間也較短，但相反的是，ResNet50 在處理較大的 dataset size 時，效能會較好，所以可得知 small model 用來處理小型數據，big model 處理大型數據。