1112 Deep Learning – Homework 2 Due: 11:59 pm, 12/12, 2023

111753127 資碩工二 蘇冠華

黑底白字:程式碼 ；#:註解

For the following questions, please upload the source code to Moodle and explain the results in your report. Please submit your homework using the IPython Notebook (.ipynb), Python script (.py), and/or a PDF file (code needs to be turned in by .py or .ipynb files).

If your computer doesn’ t have a GPU, you can work on Google Colab.

1.**Classification task (Cat and Dog):**

Please download the dataset using the following link:

https://drive.google.com/drive/folders/12J0JtSrqrHAjt2_olcB3tLVL6WIKlq5l?usp=sharing.

The dataset includes two files: train.zip and test.zip.

Utilize the training data to train two models, **AlexNet** and **ResNet**. After training, assess the performance of both models using the test data. Report the **accuracy** of the results obtained from testing. Additionally, please provide visualizations of the **training loss** changes for both models.

Sol:

Step1.Training data、Test data 及 Model權重設定

```
#將載入的圖片 調整大小、轉換成PyTorch格式
transform = transforms.Compose([transforms.Resize((224, 224)),transforms.ToTensor(),])
#將載入的資料集套用上一行寫的格式
train_dataset = datasets.ImageFolder(root='hw2train', transform=transform)
test_dataset = datasets.ImageFolder(root='hw2test', transform=transform)
#Loader資料:每次載入32張、在訓練集中隨機載入(shuffle=True)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32, shuffle=False)
#AlexNet Model權重設定，用預訓練的權重，並將連接層輸出成2個類別(Dog、Cat)
alexnet_model = models.alexnet(weights="AlexNet_Weights.DEFAULT")
alexnet_model.classifier[6] = nn.Linear(4096, 2)
#ResNet Model權重設定，用預訓練的權重，並將連接層輸出成2個類別(Dog、Cat)
resnet_model = models.resnet18(weights="ResNet18_Weights.DEFAULT")
resnet_model.fc = nn.Linear(512, 2)
#定義Loss Function為CrossEntropyLoss()
criterion = nn.CrossEntropyLoss()
#模型優化器:使用SGD optimizer、learning rate =0.00005、momentum = 0.9
alexnet_optimizer = optim.SGD(alexnet_model.parameters(), lr=0.0005, momentum=0.9)
resnet_optimizer = optim.SGD(resnet_model.parameters(), lr=0.0005, momentum=0.9)
```

Step2.Training過程

計算Training loss、Validation loss

```
def train_and_validate(model, optimizer, criterion, train_loader, test_loader, num_epochs=3):
    training_loss = []     #儲存每次迭代的Training loss
    validation_loss = []  #儲存每次迭代的Validation loss
    #開始訓練
    for epoch in range(num_epochs):
        model.train()       #設定model為訓練模式
```

```
running_loss = 0.0
#依照batch_size逐次處理Training資料集
for inputs, labels in train_loader:
    optimizer.zero_grad()           #重設梯度
    outputs = model(inputs)         #將模型預測結果儲存至outputs
    loss = criterion(outputs, labels)#計算loss
    loss.backward()                 #將計算好的loss進行反向傳播
    optimizer.step()                #更新模型權重
    running_loss += loss.item()
average_loss = running_loss / len(train_loader)    #計算average training loss
training_loss.append(average_loss)          #將training loss依序加進列表
```

計算Accuracy:

```
model.eval() #設定model為評估模式
correct = 0             #儲存預測正確的數量
total = 0               #儲存總數量
with torch.no_grad():           #不需計算梯度(因不需反向傳播)
    for inputs, labels in test_loader:
        outputs = model(inputs)         #模型進行正向傳播輸出的結果
        _, predicted = torch.max(outputs.data, 1)#找最大值作為預測的答案類別
        total += labels.size(0)  #更新總數量
        correct += (predicted == labels).sum().item()#更新預測正確的數量
accuracy = correct / total      #計算準確度
```

Step3.使用torch.save儲存訓練後的模型

```
torch.save(alexnet_model.state_dict(), 'alexnet_model_weights.pth')
torch.save(resnet_model.state_dict(), 'resnet_model_weights.pth')
```

AlexNet Accuracy = 100%

ResNet Accuracy = 100%

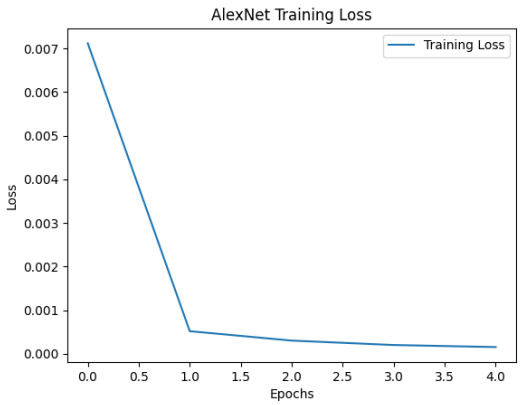
以下為訓練過程及Training loss資訊

```
Begin training
Epoch 1/5: 100%|
Epoch [1/5], Training Loss: 0.034933035384491086
Epoch 2/5: 100%|
Epoch [2/5], Training Loss: 0.002968834159336984
Epoch 3/5: 100%|
Epoch [3/5], Training Loss: 0.0017846328940242528
Epoch 4/5: 100%|
Epoch [4/5], Training Loss: 0.0012874940624460578
Epoch 5/5: 100%|
Epoch [5/5], Training Loss: 0.0010079632394015788
Test Accuracy: 1.0
ResNet has finished
```

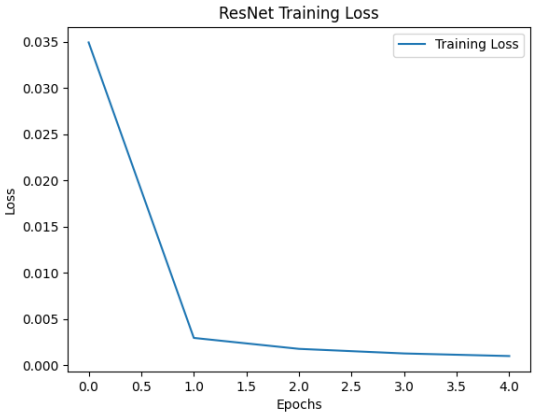
AlexNet Treminal

```
Training on cuda
Begin training
Epoch 1/5: 100%|
Epoch [1/5], Training Loss: 0.007119413751317188
Epoch 2/5: 100%|
Epoch [2/5], Training Loss: 0.0005208801927510649
Epoch 3/5: 100%|
Epoch [3/5], Training Loss: 0.0003050808296247851
Epoch 4/5: 100%|
Epoch [4/5], Training Loss: 0.00020399571923189797
Epoch 5/5: 100%|
Epoch [5/5], Training Loss: 0.00015682629567745606
Test Accuracy: 1.0
AlexNet has finished
```

ResNet Terminal



AlexNet Training Loss



ResNet Training Loss

2.**Classification Task (MNIST - Multiple Classes):**

Please download the dataset using the following code:

Follow the format of Question 1 and utilize the training data to train two different models, including **VGG** and a **CNN model you designed**. After training, assess the performance of both models using the test data. Report the **accuracy** of the results obtained from testing. Additionally, please provide visualizations of the **training loss** changes for both models.

Step1.下載資料集及調整資料格式

```
#將載入的圖片調整大小、轉換成PyTorch格式
transform = transforms.Compose([transforms.Resize((224, 224)), transforms.ToTensor()])
#將MNIST資料集載入，並套用上一行寫的格式
train_dataset = datasets.MNIST(root='data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='data', train=False, download=True, transform=transform)
#Loader資料:每次載入64張、在訓練集中隨機載入，並把資料轉至GPU運行
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False, pin_memory=True)
#把models、optimizers、loss function都用GPU跑
cnn_model = CNNModel().to(device)
vgg_model = VGGModel().to(device)
#使用Adam優化器套用於CNN、VGG模型的參數，更新模型的權重
cnn_optimizer = optim.Adam(cnn_model.parameters(), lr=0.001)
vgg_optimizer = optim.Adam(vgg_model.parameters(), lr=0.001)
#定義Loss Function為CrossEntropyLoss()，並移至GPU運行
criterion = nn.CrossEntropyLoss().to(device)
```

Step2.CNN模型設置

```
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
```

```
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 56 * 56, 128)
        self.fc2 = nn.Linear(128, 10)
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 56 * 56)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Step3.VGG模型設置

```
class VGGModel(nn.Module):
    def __init__(self):
        super(VGGModel, self).__init__()
        self.vgg11 = models.vgg11(weights='VGG11_Weights.DEFAULT').to(device)
        self.vgg11.features[0] = nn.Conv2d(1, 64, kernel_size=3, padding=1)
    def forward(self, x):
        return self.vgg11(x)
```

Step4. Training過程

```
#計算Training loss
def train(model, criterion, optimizer, train_loader, epochs=3):
    model.train()#設定model為訓練模式
    train_losses = []       #儲存每次迭代的Training loss
    for epoch in range(epochs):
        running_loss = 0.0
        #依照batch_size逐次處理Training資料集
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()  #重設梯度
            outputs = model(inputs)          #將模型預測結果儲存至outputs
            loss = criterion(outputs, labels)#計算loss
            loss.backward()          #將計算好的loss進行反向傳播
            optimizer.step()         #更新模型權重
            running_loss += loss.item()
        average_loss = running_loss / len(train_loader)    #計算average training loss
        train_losses.append(average_loss)#將training loss依序加進列表
    return train_losses
```

Step5. 計算Accuracy

```
def evaluate(model, test_loader):
```

```
    model.eval() #設定model為評估模式
    correct = 0              #儲存預測正確的數量
    total = 0                #儲存總數量
    with torch.no_grad():          #不需計算梯度(因不需反向傳播)
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device) #將資料移到GPU運行
            outputs = model(inputs)          #模型進行正向傳播輸出的結果
            _, predicted = torch.max(outputs.data, 1)#找最大值作為預測的答案類別
            total += labels.size(0)  #更新總數量
            correct += (predicted == labels).sum().item()#更新預測正確的數量
    accuracy = correct / total      #計算準確度
return accuracy
```

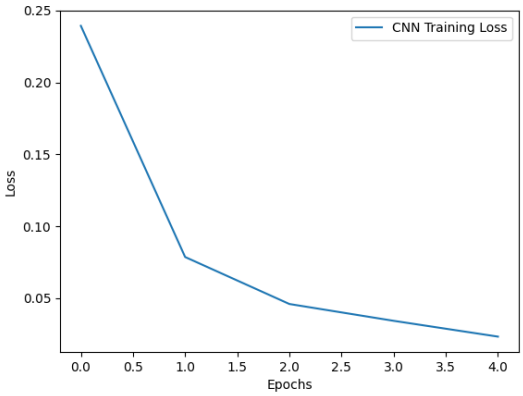
CNN Accuracy = 97.78%

VGG Accuracy = 99.01%

以下為訓練過程及Training loss資訊

```
Training on cuda
Epoch 1/5, Loss: 0.24191699466749486
Epoch 2/5, Loss: 0.08801510713017507
Epoch 3/5, Loss: 0.05332983277531079
Epoch 4/5, Loss: 0.03575976058636094
Epoch 5/5, Loss: 0.0247448973828691
CNN Test Accuracy: 0.9778
```

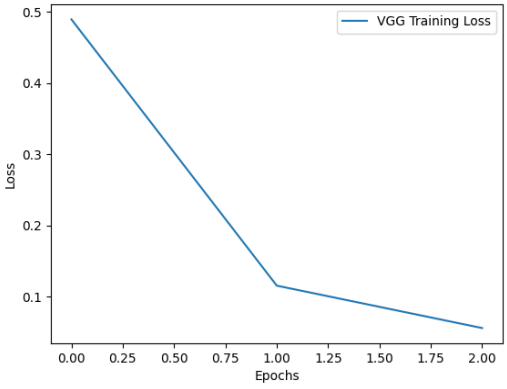
CNN Terminal Training Loss and Accuracy



CNN Training Loss可視化

```
start calculate loss
Epoch 1/3, Loss: 0.48950455384229674
start calculate loss
Epoch 2/3, Loss: 0.11567075886944336
start calculate loss
Epoch 3/3, Loss: 0.05613709342822846
VGG Test Accuracy: 0.9901
```

VGG Terminal Training Loss and Accuracy



VGG Training Loss可視化