



Food Multi-classification

資工碩一 M1354024 戴育琪

Part1

為一個圖像分類的任務設置數據加載流程，並準備訓練和測試數據集。它使用 PyTorch 庫來處理數據加載、預處理和建模。

1. 引入相關庫

```
import numpy as np
import pandas as pd
import torch
import os
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
import torchvision.transforms as transforms
from PIL import Image
from torch.utils.data import ConcatDataset, DataLoader, Subset, Dataset
from torchvision.datasets import DatasetFolder, VisionDataset
from tqdm.auto import tqdm
import random
from torchvision import transforms as T
import matplotlib.pyplot as plt
from PIL import Image
```



1. 引入相關庫 -code解析

- numpy 和 pandas: 用來進行數據處理。
- torch 和相關模塊: PyTorch 是這個專案的主要深度學習框架。
- torchvision.transforms: 用來對圖像進行變換與預處理。
- PIL.Image: 用來處理和加載圖像數據。
- tqdm: 提供一個進度條, 用來顯示處理進度。

2. 設置隨機種子

```
myseed = 6666
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(myseed)
torch.manual_seed(myseed)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(myseed)
```



2. 設置隨機種子 -code解析

- 設置隨機種子 `myseed = 6666`, 這樣可以保證結果的可重現性。
PyTorch 和 numpy 的隨機性會根據這個種子來控制。
- `torch.backends.cudnn.deterministic = True`: 確保每次運行相同的操作產生相同的結果(但會降低運行速度)。
- `torch.backends.cudnn.benchmark = False`: 禁用基於硬體條件優化的算法搜索, 同樣是為了可重現性。

3. 圖像的預處理變換

```
train_tfm = transforms.Compose([  
    # Resize the image into a fixed shape (height = width = 128)  
    transforms.Resize((128, 128)),  
    transforms.ToTensor(),  
])
```

```
test_tfm = transforms.Compose([  
    # Resize the image into a fixed shape (height = width = 128)  
    transforms.Resize((128, 128)),  
    transforms.ToTensor(),  
])
```



3. 圖像的預處理變換 -code解析

- `train_tfm` 和 `test_tfm` 定義了對訓練和測試數據進行的變換。
- `Resize`: 把圖像縮放到 128x128 的固定大小。
- `ToTensor`: 把圖像從 PIL 格式轉換成 PyTorch 張量(並且將像素值從 [0, 255] 正規化到 [0, 1])。

4. 自定義數據集類 FoodDataset

```
class FoodDataset(Dataset):
```

這裡直接使用老師給的範例程式，應圖片過長，只擷取開頭

4. 自定義數據集類 FoodDataset-code解析

- `__init__`: 初始化數據集, 指定路徑 `path`, 並自動將指定目錄下的所有 `.jpg` 文件載入到 `self.files` 中。`files` 參數允許手動指定文件列表。變換 `tfm` 用來對圖像進行處理。
- `__len__`: 返回數據集的長度, 即圖像的數量。
- `__getitem__`: 給定索引 `idx`, 返回對應的圖像和標籤。它會嘗試打開圖像, 並應用之前定義的變換。如果圖像有問題無法打開, 會跳過這張圖像。標籤是通過文件名解析出來的, 假設文件名的格式為 `<label>_<第幾張圖>.jpg`。

5. 創建數據加載器

```
# Data loading
train_dataset = FoodDataset("/home/yuchi/AI/Dataset/training", tfm=train_tfm) # Replace with the training dataset path
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True) # Increase batch size for training

test_dataset = FoodDataset("/home/yuchi/AI/Dataset/testing", tfm=test_tfm) # Replace with the testing dataset path
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

Output:

```
One /home/yuchi/AI/Dataset/training sample: /home/yuchi/AI/Dataset/training/0_0.jpg
One /home/yuchi/AI/Dataset/testing sample: /home/yuchi/AI/Dataset/testing/0001.jpg
```



5. 創建數據加載器 -code解析

- 創建 `train_dataset` 和 `test_dataset`, 指定了訓練和測試數據的目錄。並使用之前定義的圖像變換(`train_tfm` 和 `test_tfm`)。
- `DataLoader` 是 PyTorch 的數據加載器, 它將數據集分成批次進行加載。這裡設置批次大小為 64。
 - a. 訓練集 (`train_loader`) 使用 `shuffle=True`, 確保每個 epoch 中數據以隨機順序加載。
 - b. 測試集 (`test_loader`) 則不需要打亂順序, 因此使用 `shuffle=False`。

Part2

基於 ResNet-50 架構的深度學習模型，包含了 Bottleneck 殘差塊和 ResNet50 類，並模擬了 ResNet-50 的核心結構。

1. Bottleneck

```
class Bottleneck(nn.Module):
```

```
# 殘差塊定義
```

```
extention = 4 # 每個 Bottleneck block 的擴展倍率
```

```
def __init__(self, inplanes, planes, stride, downsample=None):
```

```
    super(Bottleneck, self).__init__()
```

```
    # 1x1卷積層，用於減少維度
```

```
    self.conv1 = nn.Conv2d(inplanes, planes, kernel_size=1, stride=stride, bias=False)
```

```
    self.bn1 = nn.BatchNorm2d(planes)
```

```
    # 3x3卷積層，保持特徵圖的大小（除非有 stride 設定）
```

```
    self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
```

```
    self.bn2 = nn.BatchNorm2d(planes)
```

```
    # 1x1卷積層，用於恢復維度（乘上擴展倍率）
```

```
    self.conv3 = nn.Conv2d(planes, planes * self.extention, kernel_size=1, stride=1, bias=False)
```

```
    self.bn3 = nn.BatchNorm2d(planes * self.extention)
```

```
    self.relu = nn.ReLU(inplace=True)
```

```
    # 如果下採樣不為 None，則應用下採樣
```

```
    self.downsample = downsample
```

```
    self.stride = stride
```

```

def forward(self, x):
    shortcut = x # shortcut 是跳躍連接的原始輸入
    # 第一個 1x1 卷積層和 BN 操作
    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)
    # 第二個 3x3 卷積層和 BN 操作
    out = self.conv2(out)
    out = self.bn2(out)
    out = self.relu(out)
    # 第三個 1x1 卷積層和 BN 操作
    out = self.conv3(out)
    out = self.bn3(out)
    # 如果需要，對 shortcut 使用下采樣
    if self.downsample is not None:
        shortcut = self.downsample(x)
    # 殘差加和
    out = out + shortcut
    out = self.relu(out) # 最後一層 ReLU 激活函數
    return out

```

Bottleneck 是 ResNet-50 中的基本殘差塊結構。

用於深層神經網路中的信息傳遞和梯度流通。



1. Bottleneck-code解析

- 這個類用於定義一個三層卷積結構(1x1、3x3 和 1x1)，其中殘差連接(shortcut connection)使模型能夠避免梯度消失問題，並允許在更深的網路中有效地進行訓練。
- 參數downsample 用來處理不同大小的輸入與輸出，以確保殘差連接的尺寸匹配。

2. ResNet50

```
class ResNet50(nn.Module):
    def __init__(self, block, layers, num_class):
        self.inplane = 64 # 初始輸入通道數
        super(ResNet50, self).__init__()

        self.block = block
        self.layers = layers
        # 第一層卷積：7x7大小的卷積核，stride=2，padding=3，處理輸入圖像
        self.conv1 = nn.Conv2d(3, self.inplane, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(self.inplane)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        # 四個 block stage，依次增加通道數
        self.stage1 = self.make_layer(self.block, 64, layers[0], stride=1)
        self.stage2 = self.make_layer(self.block, 128, layers[1], stride=2)
        self.stage3 = self.make_layer(self.block, 256, layers[2], stride=2)
        self.stage4 = self.make_layer(self.block, 512, layers[3], stride=2)

        # Change the avgpool kernel size based on output size after stage4
        # 四個 block stage，依次增加通道數
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1)) # Adaptive pooling to output (1, 1) # 自適應池化
        self.fc = nn.Linear(512 * block.extention, num_class)
```

```
def forward(self, x):  
    # 第一層卷積和池化  
    out = self.conv1(x)  
    out = self.bn1(out)  
    out = self.relu(out)  
    out = self.maxpool(out)  
    # Block部分 # 四個 stage  
    out = self.stage1(out)  
    out = self.stage2(out)  
    out = self.stage3(out)  
    out = self.stage4(out)  
    # 池化和全連接層  
    out = self.avgpool(out)  
    out = torch.flatten(out, 1)  
    out = self.fc(out)  
  
    return out
```

```
# 定義如何堆疊多個 Bottleneck block
def make_layer(self, block, plane, block_num, stride=1):
    block_list = []
    downsample = None
    # 如果需要改變尺寸或通道數，定義下采樣
    if (stride != 1 or self.inplane != plane * block.extention):
        downsample = nn.Sequential(
            nn.Conv2d(self.inplane, plane * block.extention, stride=stride, kernel_size=1, bias=False),
            nn.BatchNorm2d(plane * block.extention)
        )
    # 第一個 block 處理 stride 和 downsample
    conv_block = block(self.inplane, plane, stride=stride, downsample=downsample)
    block_list.append(conv_block)
    self.inplane = plane * block.extention
    # 添加剩下的 block，這些 block 沒有下采樣
    for i in range(1, block_num):
        block_list.append(block(self.inplane, plane, stride=1))

    return nn.Sequential(*block_list)
```



2. ResNet50 -code解析

- ResNet50 類定義了整個網絡結構。它通過使用 `make_layer` 方法來堆疊多個 `Bottleneck` 區塊。
- 輸入先經過一個大卷積層和池化，然後通過四個主要的階段(stage)逐步提取特徵。
- 最後使用全局平均池化(`AdaptiveAvgPool2d`)壓縮特徵圖，再通過全連接層得到最終分類結果。

`make_layer` 是一個輔助函數，用來創建每個 stage 的多個殘差區塊。第一個 block 可能會改變尺寸，之後的 block 保持尺寸不變。

測試代碼

```
...  
if __name__ == "__main__":  
    resnet = ResNet50(Bottleneck, [3, 4, 6, 3], 11) # Assuming you want 11 classes  
    x = torch.randn(64, 3, 128, 128) # Change to 128x128 input  
    x = resnet(x)  
    print(x.shape) # Should output: torch.Size([64, 11])  
...
```

Output:

```
torch.Size([64, 11])
```



測試代碼 -code解析

- 這段代碼是用來測試 **ResNet50** 模型是否正常工作。
- 使用隨機生成的 64 張大小為 128x128 的 3 通道圖像作為輸入，通過模型後，輸出應該是 [64, 11]，表示對 64 張圖像分別進行 11 類分類的結果。

Part3

使用預訓練的 ResNet-50 模型進行多類別圖像分類任務的訓練和驗證，並儲存最佳模型參數。

1. 載入模型

```
# 2. load model
num_class = 11
model = models.resnet50(pretrained=True)
fc_inputs = model.fc.in_features
model.fc = nn.Linear(fc_inputs, num_class)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
```




1. 載入模型 -code解析

- 載入預訓練的 ResNet-50 模型 (`pretrained=True`), 使用在 ImageNet 上預訓練的權重。
- 將 ResNet-50 最後一層全連接層 (`model.fc`) 替換為一個新的線性層, 輸出大小與任務的分類數目 (`num_class = 11`) 匹配。`fc_inputs` 取得原始全連接層的輸入維度。
- 模型被移動到 GPU (若可用), 否則使用 CPU。

2. 訓練參數設定

```
# 3. prepare super parameters
criterion = nn.CrossEntropyLoss()
learning_rate = 0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
epoch = 15
```



2. 訓練參數設定 -code解析

- 設定損失函數為 `CrossEntropyLoss`, 這是多分類任務常用的損失函數。
- 使用 Adam 優化器, 學習率設定為 `0.0001`。
- 訓練的總迭代次數(`epoch`)為 `15`。



submission.csv

Complete · 5h ago

`<-- epoch = 15`

0.85653



submission.csv

Complete · 7h ago

`<-- epoch = 20`

0.83297



submission.csv

Complete · 11h ago

`<-- epoch = 10`

0.84082



3. 訓練過程

```
# 4. train
val_acc_list = []
for epoch in range(0, epoch):
    print('\nEpoch: %d' % (epoch + 1))
    model.train()
    sum_loss = 0.0
    correct = 0.0
    total = 0.0
    for batch_idx, (images, labels) in enumerate(train_loader):
        length = len(train_loader)
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images) # torch.size([batch_size, num_class])
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        sum_loss += loss.item()
        _, predicted = torch.max(outputs.data, dim=1)
        total += labels.size(0)
        correct += predicted.eq(labels.data).cpu().sum()
    print('[epoch:%d, iter:%d] Loss: %.03f | Acc: %.3f%% '
          % (epoch + 1, (batch_idx + 1 + epoch * length), sum_loss / (batch_idx + 1), 100. * correct / total))
```



3. 訓練過程 -code解析

- 設定模型為訓練模式 (`model.train()`)。
- 對每個 batch, 首先將圖像和標籤移到 GPU(或 CPU), 進行前向傳播、計算損失, 並使用反向傳播更新權重。
- 每個 epoch 都會輸出當前的訓練損失和準確率。

4. 驗證過程

```
#get the ac with testdataset in each epoch
print('Waiting Val...')
with torch.no_grad():
    correct = 0.0
    total = 0.0
    for batch_idx, (images, labels) in enumerate(test_loader):
        model.eval()
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, dim=1)
        total += labels.size(0)
        correct += (predicted == labels).sum()
    print('Val\'s ac is: %.3f%%' % (100 * correct / total))

acc_val = 100 * correct / total
val_acc_list.append(acc_val)
```



4. 驗證過程 -code解析

- 使用 `torch.no_grad()` 關閉梯度計算, 減少內存佔用並加速推理。
- 設定模型為驗證模式 (`model.eval()`), 在每個 epoch 結束後用測試集進行評估, 並計算準確率。
- 將每次驗證的準確率加入 `val_acc_list`。

5. 儲存模型

```
torch.save(model.state_dict(), "/home/yuchi/AI/M1354024_best.ckpt")
if acc_val == max(val_acc_list):
    torch.save(model.state_dict(), "/home/yuchi/AI/M1354024_best.ckpt")
    print("save epoch {} model".format(epoch))
```



5. 儲存模型 -code解析

- 在每個 epoch 後儲存模型的狀態字典，儲存到指定路徑 `/home/yuchi/AI/M1354024_best.ckpt`。
- 如果當前 epoch 的驗證準確率是歷史最高，則保存該模型，作為“最佳”模型。

Output:(epoch2~epoch14的部分被自動摺疊了)

```
Epoch: 1
/home/yuchi/anaconda3/envs/torch230/lib/python3.10/site-packages/torch/autograd/graph.py:744: UserWarning: Plan failed with a cudnnException: CUDNN
  return Variable._execution_engine.run_backward( # Calls into the C++ engine to run the backward pass

[epoch:1, iter:1] Loss: 2.487 | Acc: 9.375%
[epoch:1, iter:2] Loss: 2.406 | Acc: 10.156%
[epoch:1, iter:3] Loss: 2.352 | Acc: 16.146%
[epoch:1, iter:4] Loss: 2.308 | Acc: 17.578%
[epoch:1, iter:5] Loss: 2.247 | Acc: 21.562%
[epoch:1, iter:6] Loss: 2.199 | Acc: 23.958%
[epoch:1, iter:7] Loss: 2.155 | Acc: 27.232%
[epoch:1, iter:8] Loss: 2.091 | Acc: 30.078%
[epoch:1, iter:9] Loss: 2.046 | Acc: 31.944%
[epoch:1, iter:10] Loss: 2.004 | Acc: 33.750%
[epoch:1, iter:11] Loss: 1.972 | Acc: 34.375%
[epoch:1, iter:12] Loss: 1.920 | Acc: 37.500%
[epoch:1, iter:13] Loss: 1.881 | Acc: 38.822%
[epoch:1, iter:14] Loss: 1.848 | Acc: 40.067%
[epoch:1, iter:15] Loss: 1.808 | Acc: 41.875%
[epoch:1, iter:16] Loss: 1.780 | Acc: 42.676%
[epoch:1, iter:17] Loss: 1.753 | Acc: 43.474%
[epoch:1, iter:18] Loss: 1.721 | Acc: 44.444%
[epoch:1, iter:19] Loss: 1.694 | Acc: 45.641%
[epoch:1, iter:20] Loss: 1.666 | Acc: 46.797%
[epoch:1, iter:21] Loss: 1.635 | Acc: 47.917%
[epoch:1, iter:22] Loss: 1.612 | Acc: 48.580%
[epoch:1, iter:23] Loss: 1.580 | Acc: 49.592%
[epoch:1, iter:24] Loss: 1.549 | Acc: 50.521%
[epoch:1, iter:25] Loss: 1.518 | Acc: 51.688%
...
[epoch:15, iter:1530] Loss: 0.015 | Acc: 99.586%
Waiting Val...
Val's ac is: 0.000%
```

Part4

加載已訓練的最佳模型並對測試數據集進行預測，最後生成 CSV 檔案以提交預測結果。

1. 載入預訓練模型並設置

```
model_best = models.resnet50(pretrained=True)
fc_inputs = model_best.fc.in_features
model_best.fc = nn.Linear(fc_inputs, 11)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model_best.to(device)
model_best.load_state_dict(torch.load(f"/home/yuchi/AI/M1354024_best.ckpt"))
model_best.eval()
prediction = []
```



1. 載入預訓練模型並設置 -code解析

- 使用預訓練的 ResNet-50 模型(`pretrained=True`), 並將其最後的全連接層(`model.fc`) 替換為一個具有 11 類別輸出的線性層。
- 設定裝置為 GPU 或 CPU, 取決於當前硬件支持。
- 加載之前訓練過程中保存的最佳模型權重 (`M1354024_best.ckpt`)。
- 設置模型為推理模式 (`eval()`), 這會關閉一些影響推理的功能, 比如 dropout 和 batch normalization 的更新。

2. 預測

```
prediction = []  
with torch.no_grad():  
    for data, _ in test_loader:  
        test_pred = model_best(data.to(device))  
        test_label = np.argmax(test_pred.cpu().data.numpy(), axis=1)  
        prediction += test_label.squeeze().tolist()
```



2. 預測 -code解析

- 使用 `torch.no_grad()` 關閉梯度計算, 這在推理時可以節省內存並加速運算。
- 對測試數據集 (`test_loader`) 中的每個批次進行前向傳播, 取得模型的輸出 (`test_pred`)。
- 使用 `np.argmax` 將模型輸出的概率轉換為預測的類別索引, 並將其存入 `prediction` 列表中。

3. 生成 CSV 文件

```
#create test csv
def pad4(i):
    return "0"*(4-len(str(i)))+str(i)
df = pd.DataFrame()
df["Id"] = [pad4(i) for i in range(1,len(test_dataset)+1)]
df["Category"] = prediction
df.to_csv("/home/yuchi/AI/submission.csv",index = False)
print("done!")
```








3. 生成 CSV 文件 -code解析

- `pad4` 函數用於格式化測試樣本的編號，將數字轉換為固定四位的字符串，例如 0001、0002 等，這在生成提交文件時經常需要。
- 使用 `pd.DataFrame` 建立一個包含預測結果的資料框，兩列分別是 `Id`(樣本編號)和 `Category`(預測類別)。
- 最後，將結果保存為 CSV 文件，路徑為 `/home/yuchi/AI/submission.csv`，並且不包含索引。

心得

Leaderboard

#	Team	Members	Score	Entries	Last	Join
1	M1354002		0.88008	33	2d	
2	M1254020		0.87865	11	3h	
3	M1354024_Vicky		0.85653	6	5h	
4	M1354024_TEST		0.85653	18	5h	




Your Best Entry!
Your most recent submission scored 0.85653, which is an improvement of your previous score of 0.62455. Great job!

[Tweet this](#)

因為當時沒注意不小心用了其他帳號 Submit Prediction ,
所以有兩個帳號都是 M1354024, 不好意思 QQ

心得

- 
- CNN分類不難，但各個"層"之間的連接及"維度"的降級/擴級很複雜。這也是讓我花最多時間的地方。
 - 過擬合是需要特別注意的，有時候要反覆嘗試找出最佳訓練次數，不過不知道為什麼用了ResNet卻還是會出現過擬合的情況。

這次的作業過程中學到了如何使用 PyTorch 訓練模型和進行推理，並熟悉了模型的優化和調參步驟。整體來說，這是一個簡單但收穫頗豐的實驗，讓我對深度學習的應用有了更多信心！