Statistical Learning and Deep Learning, Fall 2024 HW3

B11705009 An-Che, Liang

Part1: Resnet as a Feature Extractor, after FC Layer

In [2]:
```
import torch
import torchvision
from torchvision import models, transforms, datasets
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np
from tqdm import tqdm
```

In [3]:
```
imagenet_stats = [(0.485, 0.456, 0.406), (0.229, 0.224, 0.225)]

valid_tfms = transforms.Compose([
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(imagenet_stats[0], imagenet_stats[1])
])

batch_size = 196

trainset = torchvision.datasets.Food101(root='./food101', split="train", dow
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, s

validset = torchvision.datasets.Food101(root='./food101', split="test", dowr
validloader = torch.utils.data.DataLoader(validset, batch_size=batch_size, s

assert trainset.classes == validset.classes
classes = trainset.classes
num_classes = len(classes)

print("Number of classes =", num_classes)
```

```
Number of classes = 101
```

In [4]:
```
device = torch.device("mps")
resnet50 = models.resnet50(weights="IMAGENET1K_V2")
resnet50.fc = torch.nn.Identity()  # Replace the fully connected layer with
resnet50 = resnet50.to(device)
resnet50.eval()
```

Out[4]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil
_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
```

```
unning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
    )
```

```
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
```

```
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
```

```
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=Fal
se)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
```

```
    (fc): Identity()
  )
```

In [5]:
```python
def extract_features(dataloader, model):
    features, labels = [], []
    with torch.no_grad():
        for images, targets in tqdm(dataloader, desc="Extracting Features"):
            images = images.to(device)
            outputs = model(images)  # Extract 1000-dimensional features
            features.append(outputs.cpu().numpy())
            labels.append(targets.numpy())
    features = np.concatenate(features)
    labels = np.concatenate(labels)
    return features, labels
```

In [6]:
```python
train_features, train_labels = extract_features(trainloader, resnet50)
test_features, test_labels = extract_features(validloader, resnet50)
```

```
Extracting Features: 100%|██████████| 387/387 [17:00<00:00,  2.64s/it]
Extracting Features: 100%|██████████| 129/129 [06:17<00:00,  2.92s/it]
```

In [7]:
```python
logreg = LogisticRegression(max_iter=1000, verbose=1, n_jobs=-1)
logreg.fit(train_features, train_labels)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =       206949     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  4.61512D+00    |proj g|=  4.86234D-02
 This problem is unconstrained.
```

```
At iterate   50    f=  1.27395D+00    |proj g|=  7.07186D-03

At iterate  100    f=  9.93092D-01    |proj g|=  9.82457D-03

At iterate  150    f=  8.90531D-01    |proj g|=  3.46589D-03

At iterate  200    f=  8.52607D-01    |proj g|=  9.07444D-04

At iterate  250    f=  8.38238D-01    |proj g|=  6.39770D-04

At iterate  300    f=  8.32456D-01    |proj g|=  4.41992D-04

At iterate  350    f=  8.30074D-01    |proj g|=  1.98946D-04

At iterate  400    f=  8.29195D-01    |proj g|=  1.66219D-04

At iterate  450    f=  8.28829D-01    |proj g|=  8.90317D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
*****    450    456      1     0     0   8.903D-05   8.288D-01
  F =  0.82882867776498426

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```

Out[7]:

```
▾ LogisticRegression
LogisticRegression(max_iter=1000, n_jobs=-1, verbose=1)
```

In [8]:

```
test_predictions = logreg.predict(test_features)
accuracy = accuracy_score(test_labels, test_predictions)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Test Accuracy: 60.64%
```

Part2: Resnet as a Feature Extractor, before FC Layer

In [9]:

```
resnet50.fc = torch.nn.Identity()  # Remove the final fc layer
resnet50 = resnet50.to(device)
resnet50.eval()
```

Out[9]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runnin
g_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil
_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_ru
nning_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
```

```
unning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
    )
```

```
    (3): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fal
se)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=Fals
e)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
```

```
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
```

```
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=Fal
se)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=Fa
lse)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
```

```
    (fc): Identity()
  )
```

In [10]:
```
train_features_2048, train_labels_2048 = extract_features(trainloader, resne
test_features_2048, test_labels_2048 = extract_features(validloader, resnet5
```

```
Extracting Features: 100%|██████████| 387/387 [15:13<00:00,  2.36s/it]
Extracting Features: 100%|██████████| 129/129 [05:11<00:00,  2.42s/it]
```

In [11]:
```
logreg_2048 = LogisticRegression(max_iter=1000, verbose=1, n_jobs=-1)
logreg_2048.fit(train_features_2048, train_labels_2048)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =       206949     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  4.61512D+00    |proj g|=  4.86234D-02
 This problem is unconstrained.
```

```
At iterate   50    f=  1.27395D+00    |proj g|=  7.07186D-03

At iterate  100    f=  9.93092D-01    |proj g|=  9.82449D-03

At iterate  150    f=  8.90532D-01    |proj g|=  3.47315D-03

At iterate  200    f=  8.52266D-01    |proj g|=  1.13297D-03

At iterate  250    f=  8.38093D-01    |proj g|=  1.29713D-03

At iterate  300    f=  8.32523D-01    |proj g|=  1.29053D-03

At iterate  350    f=  8.30143D-01    |proj g|=  2.18735D-04

At iterate  400    f=  8.29217D-01    |proj g|=  3.16976D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
*****    422    428      1     0     0   9.763D-05   8.290D-01
  F =  0.82899594787211117

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```

Out[11]:

```
▾ LogisticRegression
LogisticRegression(max_iter=1000, n_jobs=-1, verbose=1)
```

In [13]:

```python
test_predictions_2048 = logreg_2048.predict(test_features_2048)
accuracy_2048 = accuracy_score(test_labels_2048, test_predictions_2048)
print(f"Test Accuracy using 2048 features: {accuracy_2048 * 100:.2f}%")
```

```
Test Accuracy using 2048 features: 60.58%
```

We can observe that the two approaches yield virtually no difference. The 1000-feature representation achieves an accuracy of 60.64%, while the 2048-feature representation achieves 60.58%. This similarity arises because most of the feature extraction is performed within the residual blocks. As a result, including or excluding the final fully connected (FC) layer has minimal impact on performance.

Part3: Resnet Finetuning

In [1]:
```
from torchvision import transforms
import torch
import torchvision

# data transformation during training
train_tfms = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22
])

# data transformation during validation
valid_tfms = transforms.Compose([
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22
])

# Load the training and validation datasets
trainset = torchvision.datasets.Food101(root='./food101', split="train", dow
validset = torchvision.datasets.Food101(root='./food101', split="test", dowr

trainloader = torch.utils.data.DataLoader(trainset, batch_size=256, shuffle=
validloader = torch.utils.data.DataLoader(validset, batch_size=256, shuffle=

num_classes = len(trainset.classes)
print(f"Number of classes: {num_classes}")
```

```
Number of classes: 101
```

In [2]:
```
import torch.nn as nn
from torchvision import models

# load the pre-trained ResNet-50 model
resnet50 = models.resnet50(weights="IMAGENET1K_V2")
resnet50.fc = nn.Linear(2048, num_classes)  # 替換最後一層

# set the device to GPU
device = torch.device("mps" if torch.cuda.is_available() else "cpu")
resnet50 = resnet50.to(device)
```

In [3]:
```
optimizer = torch.optim.Adam(resnet50.parameters(), lr=1e-4)
criterion = nn.CrossEntropyLoss()
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min'
```

In [4]:
```
from sklearn.metrics import f1_score
import numpy as np
from tqdm import tqdm


def train_and_evaluate(
    model,
    trainloader,
    validloader,
```

```
    optimizer,
    criterion,
    scheduler,
    num_epochs=100,
    patience=5,
):
    best_val_f1 = 0
    patience_counter = 0
    train_losses, val_losses, val_accuracies, val_f1s = [], [], [], []

    for epoch in tqdm(range(num_epochs), desc="Training", total=num_epochs):
        # 訓練階段
        model.train()
        running_loss = 0
        for inputs, targets in trainloader:
            inputs, targets = inputs.to(device), targets.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        train_losses.append(running_loss / len(trainloader))

        # 驗證階段
        model.eval()
        val_loss = 0
        all_preds, all_targets = [], []
        with torch.no_grad():
            for inputs, targets in validloader:
                inputs, targets = inputs.to(device), targets.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, targets)
                val_loss += loss.item()

                preds = torch.argmax(outputs, dim=1)
                all_preds.extend(preds.cpu().numpy())
                all_targets.extend(targets.cpu().numpy())

        val_losses.append(val_loss / len(validloader))
        val_acc = np.mean(np.array(all_preds) == np.array(all_targets))
        val_f1 = f1_score(all_targets, all_preds, average="macro")

        val_accuracies.append(val_acc)
        val_f1s.append(val_f1)

        print(
            f"Epoch {epoch+1}: Train Loss = {train_losses[-1]:.4f}, Val Loss
            f"Val Acc = {val_acc:.4f}, Val Macro F1 = {val_f1:.4f}"
        )

        # Early stopping
        if val_f1 > best_val_f1:
            best_val_f1 = val_f1
            patience_counter = 0
```

```python
            torch.save(model.state_dict(), "best_model.pth")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print("Early stopping triggered.")
                break

        scheduler.step(val_loss)

    return train_losses, val_losses, val_accuracies, val_f1s
```

In [ ]:

```python
train_losses_all, val_losses_all, val_accs_all, val_f1s_all = train_and_eval
    resnet50, trainloader, validloader, optimizer, criterion, scheduler
)
```

```
Training:   0%|          | 0/100 [00:00<?, ?it/s]
```