

ML HW3 Report

學號：b07902040 | 系級：資工二 | 姓名：吳承軒

1. 請說明你實作的 CNN 模型(best model)，其模型架構、訓練參數量和準確率為何？(1%)

對vgg16架構略做修改:

模型架構:

```
1 self.cnn = nn.Sequential(  
2     nn.Conv2d(3, 64, 3, 1, 1), # [64, 128, 128]  
3     nn.BatchNorm2d(64),  
4     nn.ReLU(),  
5     nn.Conv2d(64, 64, 3, 1, 1),  
6     nn.BatchNorm2d(64),  
7     nn.ReLU(),  
8     nn.MaxPool2d(2, 2, 0), # [64, 64, 64]  
9  
10    nn.Conv2d(64, 128, 3, 1, 1), # [64, 64, 64]  
11    nn.BatchNorm2d(128),  
12    nn.ReLU(),  
13    nn.Conv2d(128, 128, 3, 1, 1),  
14    nn.BatchNorm2d(128),  
15    nn.ReLU(),  
16    nn.MaxPool2d(2, 2, 0), # [128, 32, 32]  
17  
18    nn.Conv2d(128, 256, 3, 1, 1), # [128, 32, 32]  
19    nn.BatchNorm2d(256),  
20    nn.ReLU(),  
21    nn.Conv2d(256, 256, 3, 1, 1),  
22    nn.BatchNorm2d(256),  
23    nn.ReLU(),  
24    nn.Conv2d(256, 256, 3, 1, 1),  
25    nn.BatchNorm2d(256),  
26    nn.ReLU(),  
27    nn.MaxPool2d(2, 2, 0), # [256, 16, 16]  
28  
29    nn.Conv2d(256, 512, 3, 1, 1), # [256, 16, 16]  
30    nn.BatchNorm2d(512),  
31    nn.ReLU(),  
32    nn.Conv2d(512, 512, 3, 1, 1),  
33    nn.BatchNorm2d(512),  
34    nn.ReLU(),  
35    nn.Conv2d(512, 512, 3, 1, 1),  
36    nn.BatchNorm2d(512),  
37    nn.ReLU(),  
38    nn.MaxPool2d(2, 2, 0), # [512, 8, 8]  
39  
40    nn.Conv2d(512, 512, 3, 1, 1), # [512, 8, 8]  
41    nn.BatchNorm2d(512),  
42    nn.ReLU(),
```

```

43         nn.Conv2d(512, 512, 3, 1, 1), # [512, 8, 8]
44         nn.BatchNorm2d(512),
45         nn.ReLU(),
46         nn.Conv2d(512, 512, 3, 1, 1), # [512, 8, 8]
47         nn.BatchNorm2d(512),
48         nn.ReLU(),
49         nn.MaxPool2d(2, 2, 0),          # [512, 4, 4]
50     )
51     self.fc = nn.Sequential(
52         nn.Linear(512*4*4, 4096),
53         nn.Dropout(0.5),
54         nn.ReLU(),
55         nn.Linear(4096, 2048),
56         nn.Dropout(0.5),
57         nn.ReLU(),
58         nn.Linear(2048, 512),
59         nn.Dropout(0.5),
60         nn.ReLU(),
61         nn.Linear(512, 11)
62     )

```

參數:

```

1  batch_size = 128
2  learning_rate = 0.001
3  num_epoch = 200
4  input_size = 3*128*128

```

訓練參數量: **57,707,715 parameters**

```

1  INPUT: [128x128x3] weights: 0
2  CONV3-64: [128x128x64] weights: (3*3*3)*64 = 1,728
3  CONV3-64: [128x128x64] weights: (3*3*64)*64 = 36,864
4  POOL2: [64x64x64] weights: 0
5  CONV3-128: [64x64x128] weights: (3*3*64)*128 = 73,728
6  CONV3-128: [64x64x128] weights: (3*3*128)*128 = 147,456
7  POOL2: [32x32x128] weights: 0
8  CONV3-256: [32x32x256] weights: (3*3*128)*256 = 294,912
9  CONV3-256: [32x32x256] weights: (3*3*256)*256 = 589,824
10 CONV3-256: [32x32x256] weights: (3*3*256)*256 = 589,824
11 POOL2: [16x16x256] weights: 0
12 CONV3-512: [16x16x512] weights: (3*3*256)*512 = 1,179,648
13 CONV3-512: [16x16x512] weights: (3*3*512)*512 = 2,359,296
14 CONV3-512: [16x16x512] weights: (3*3*512)*512 = 2,359,296
15 POOL2: [8x8x512] weights: 0
16 CONV3-512: [8x8x512] weights: (3*3*512)*512 = 2,359,296
17 CONV3-512: [8x8x512] weights: (3*3*512)*512 = 2,359,296
18 CONV3-512: [8x8x512] weights: (3*3*512)*512 = 2,359,296
19 POOL2: [4x4x512] weights: 0
20 FC: [1x1x4096] weights: 4*4*512*4096 = 33,554,432
21 FC: [1x1x2048] weights: 4096*2048 = 8,388,608
22 FC: [1x1x512] weights: 2048*512 = 1,048,579
23 FC: [1x1x11] weights: 512*11 = 5632
24 Total = 以上總和 = 57,707,715

```

正確率

在kaggle public上的準確率為84.518%

2. 請實作與第一題接近的參數量，但 CNN 深度 (CNN 層數) 減半的模型，並說明其模型架構、訓練參數量和準確率為何？(1%)

模型架構:

```
1 self.cnn = nn.Sequential(  
2     nn.Conv2d(3, 64, 3, 1, 1), # [64, 128, 128]  
3     nn.BatchNorm2d(64),  
4     nn.ReLU(),  
5     nn.MaxPool2d(2, 2, 0), # [64, 64, 64]  
6  
7     nn.Conv2d(64, 128, 3, 1, 1), # [64, 64, 64]  
8     nn.BatchNorm2d(128),  
9     nn.ReLU(),  
10    nn.Conv2d(128, 128, 3, 1, 1),  
11    nn.BatchNorm2d(128),  
12    nn.ReLU(),  
13    nn.MaxPool2d(2, 2, 0), # [128, 32, 32]  
14  
15    nn.Conv2d(128, 256, 3, 1, 1), # [128, 32, 32]  
16    nn.BatchNorm2d(256),  
17    nn.ReLU(),  
18    nn.MaxPool2d(2, 2, 0), # [256, 16, 16]  
19  
20    nn.Conv2d(256, 512, 3, 1, 1), # [256, 16, 16]  
21    nn.BatchNorm2d(512),  
22    nn.ReLU(),  
23    nn.MaxPool2d(2, 2, 0), # [512, 8, 8]  
24  
25    nn.Conv2d(512, 512, 3, 1, 1), # [512, 8, 8]  
26    nn.BatchNorm2d(512),  
27    nn.ReLU(),  
28    nn.MaxPool2d(2, 2, 0), # [512, 4, 4]  
29 )  
30 self.fc = nn.Sequential(  
31     nn.Linear(512*4*4, 4096),  
32     nn.Dropout(0.5),  
33     nn.ReLU(),  
34     nn.Linear(4096, 2048),  
35     nn.Dropout(0.5),  
36     nn.ReLU(),  
37     nn.Linear(2048, 512),  
38     nn.Dropout(0.5),  
39     nn.ReLU(),  
40     nn.Linear(512, 11)  
41 )
```

參數:

```

1 batch_size = 128
2 learning_rate = 0.001
3 num_epoch = 200
4 input_size = 3*128*128

```

訓練參數量: 55,274,649 parameters

```

1 INPUT: [128x128x3] weights: 0
2 CONV3-64: [128x128x64] weights: (3*3*3)*64 = 1,728
3 POOL2: [64x64x64] weights: 0
4 CONV3-128: [64x64x128] weights: (3*3*64)*128 = 73,728
5 CONV3-128: [64x64x128] weights: (3*3*128)*128 = 147,456
6 POOL2: [32x32x128] weights: 0
7 CONV3-256: [32x32x256] weights: (3*3*128)*256 = 294,912
8 POOL2: [16x16x256] weights: 0
9 CONV3-512: [16x16x512] weights: (3*3*256)*512 = 1,179,648
10 POOL2: [8x8x512] weights: 0
11 CONV3-512: [8x8x512] weights: (3*3*512)*512 = 2,359,296
12 POOL2: [4x4x512] weights: 0
13 FC: [1x1x4096] weights: 4*4*512*4096 = 33,554,432
14 FC: [1x1x4096] weights: 4096*4096 = 16,777,216
15 FC: [1x1x512] weights: 4096*512 = 2,097,158
16 FC: [1x1x11] weights: 512*11 = 5632
17 Total = 以上總和 = 55,274,694

```

正確率

在kaggle public上的準確率為80.394%

3. 請實作與第一題接近的參數量，簡單的 DNN 模型，同時也說明其模型架構、訓練參數和準確率為何？(1%)

模型架構:

```

1 self.cnn = nn.Sequential(
2     nn.MaxPool2d(2, 2, 0)
3 )
4 self.fc = nn.Sequential(
5     nn.Linear(3*64*64, 4096),
6     nn.Dropout(0.5),
7     nn.ReLU(),
8     nn.Linear(4096, 2048),
9     nn.Dropout(0.5),
10    nn.ReLU(),
11    nn.Linear(2048, 512),
12    nn.Dropout(0.5),
13    nn.ReLU(),
14    nn.Linear(512, 11)
15 )

```

參數:

```
1 batch_size = 128
2 learning_rate = 0.001
3 num_epoch = 200
4 input_size = 3*128*128
```

訓練參數量: 59,774,476 parameters

```
1 INPUT: [128x128x3] weights: 0
2 POOL2: [64x64x3] weights: 0
3 FC: [1x1x4096] weights: 3*64*64*4096 = 50,331,648
4 FC: [1x1x2048] weights: 4096*2048 = 8,388,608
5 FC: [1x1x512] weights: 2048*512 = 1,048,579
6 FC: [1x1x11] weights: 512*11 = 5632
7 Total = 以上總和 = 59,774,476
```

正確率

在kaggle public上的準確率為22.414%

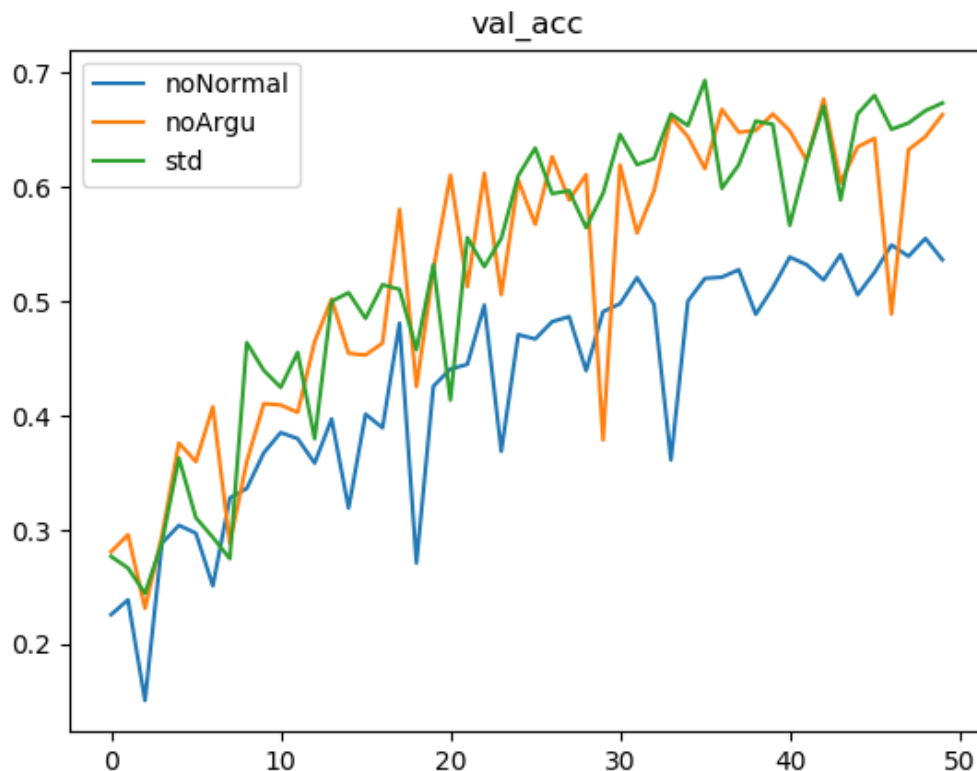
4. 請說明由 1 ~ 3 題的實驗中你觀察到了什麼？(1%)

可以發現在此題在參數量接近時，細長的架構(layer多，每層layer小)通常會比粗短(layer少，每層layer大)的來的好，尤其在圖片分類的訓練中，找出2D中的特徵特別重要。第3題只使用DNN的做法完全忽略了2D上的分布，將圖片直接拉為一直線，所以正確率相當低。

5. 請嘗試 data normalization 及 data augmentation，說明實作方法並且說明實行前後對準確率有什麼樣的影響？(1%)

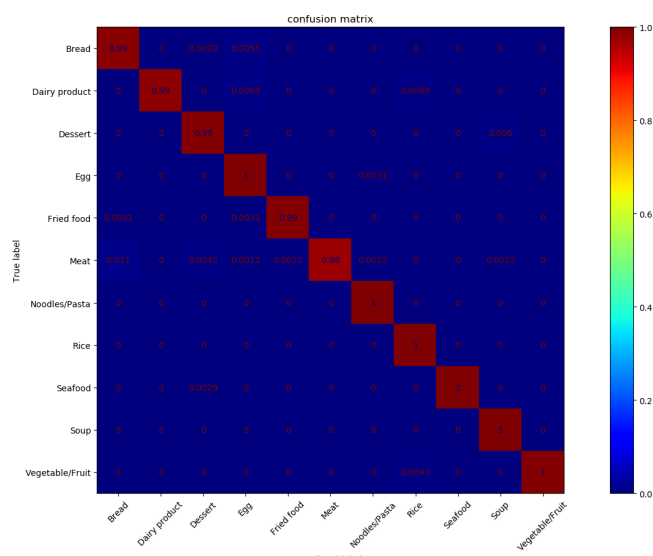
data normalization實行方法:由於transforms.ToTensor()會自動做normalization，所以由是否對每個x乘255控制。 data augmentation實行方法:由是否加入以下兩行控制。

```
1 transforms.RandomHorizontalFlip(), # 隨機將圖片水平翻轉
2 transforms.RandomRotation(15), # 隨機旋轉圖片
```



std為data normalization和data augmentation都有做 可以發現 1.取消normalize之後，其正確率顯著地較另外兩個case低，可能是因為取消normalize後，其loss function的空間較為崎嶇，gradient descent的過程較長且彎曲。 2.取消augmentation之後的波動較大，可能是因為其model學習到了在training data上順序的pattern，而這種pattern對val來說是毫無道理的，從而導致over fitting，在val上的表現較std差。

6. 觀察答錯的圖片中，哪些 class 彼此間容易用混？[繪出 confusion matrix 分析] (1%)



這是在val上的confusion matrix。由於我在訓練時為了增加資料量，把val併入了train一起訓練，所以這些val都是model看過的樣本，所以正確率才這麼高，但仍看得出來哪些class之間較易弄混。最常出錯的是誤認Dairy product為Egg和Rice。次為誤認Bread為Egg