學號：B06901053　　系級：電機三　　姓名：謝承延

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%)

Batch size:64

epoch time:100

learning rate:0.001

我參考 AlexNet 的架構做出了5層 convolution layer 以及3層的 neural network,

每一層 convolution layer 完之後做 batch normalization,activation function 使用 Relu, 為了避免 overfitting, 在 neural network 的部份加入 drop out (probability 為0.5)

每一層 layer 參數如下

	Kernel size	Channels
Convolution layer	3	64
Batch Normalization		
ReLU		
MaxPooling	2X2	
Convolution layer	3	128
Batch Normalization		
ReLU		
MaxPooling	2X2	
Convolution layer	3	256
Batch Normalization		
ReLU		
MaxPooling	2X2	
Convolution layer	3	512
Batch Normalization		
ReLU		
MaxPooling	2X2	
Convolution layer	3	512
Batch Normalization		

ReLU	
MaxPooling	2X2

NN:

	Neuron
Hidden layer	1024
Drop out(0.5)	
Hidden layer	512
Drop out(0.5)	
Output layer	11

Train accuracy 達到：0.971946

kaggle public set accuracy：0.81589

AlexNet 中將3～5層的 maxpooling 省去了，可能可以讓模型更加複雜，但因為 gpu 內存限制我沒有使用此方式

然而我認為若是把 maxpooling 省去有機會達到更高的準確率， 因為我在做 train validation 的時候 有發現一點 underfitting 的狀況，雖然 epoch time 拉高以後就沒有這樣的狀況，但是讓 model 更複雜有可能達到更好的 performance

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%)

batch size 跟 epoch 次數皆固定

```
batch_size = 64
num_epoch = 30
```

原本的 model ：

```
self.cnn = nn.Sequential(
    nn.Conv2d(3, 64, 3, 1, 1),  # [64, 128, 128]
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),      # [64, 64, 64]

    nn.Conv2d(64, 128, 3, 1, 1),  # [128, 64, 64]
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),      # [128, 32, 32]

    nn.Conv2d(128, 256, 3, 1, 1),  # [256, 32, 32]
    nn.BatchNorm2d(256),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),      # [256, 16, 16]

    nn.Conv2d(256, 512, 3, 1, 1),  # [512, 16, 16]
    nn.BatchNorm2d(512),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),       # [512, 8, 8]

    nn.Conv2d(512, 512, 3, 1, 1),  # [512, 8, 8]
    nn.BatchNorm2d(512),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),       # [512, 4, 4]
```

5次的 convolution

每次 convolution 做完以後，做 batch normalization 以確保 layer 之間的 internal covariate shift 不要太大，（套在 cnn 的話 這種問題可能就意謂著不同 convolution layer 的影響力會不同,難以找到正確參數），並做 ReLU 後丟入 maxpooling

```
[030/030] 50.93 sec(s) Train Acc: 0.879485 Loss: 0.005471 | Val Acc: 0.668222 loss: 0.019835
```

達到 train accuracy:0.879485 validation accuracy:0.668222

更改後的 model:

```
self.cnn = nn.Sequential(
    nn.Conv2d(3, 64, 3, 1, 1),  # [64, 128, 128]
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),      # [64, 64, 64]

    nn.Conv2d(64, 128, 3, 1, 1),  # [128, 64, 64]
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.MaxPool2d(16, 16, 0),      # [128, 32, 32]
```

```
self.fc = nn.Sequential(
    nn.Linear(128*4*4, 2),
    nn.ReLU(),
    nn.Linear(2, 512*4*4),
    nn.ReLU(),
    # 改
    # 改
    nn.Linear(512*4*4, 1024),
    nn.ReLU(),
    nn.Linear(1024, 512),
    nn.ReLU(),
    nn.Linear(512, 11)
)
```

convolution layer 減少3層

neural network 的 layer 數則增加兩層

變化的參數量為 ：-(9*256+9*512+9*512)+128*4*4*2+2*512*4*4=8960

結果：

```
35.33 sec(s) Train Acc: 0.154166 Loss: 0.036056 | Val Acc: 0.145773 loss: 0.036147
```

慘不忍睹的 underfitting ---accuracy 僅僅只有約0.15

我認為可能的原因是有一層 neural network 的 hidden layer 只有2個 neuron，這兩個 neuron 要將資訊傳給下一層的時候無法提供足夠有力的資訊，因此雖然參數量增加，反而無法提供更精確的 funtion

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%)

DNN 的架構為 input layer, hidden layerX1, output layer

activation function 為 relu 和 sigmoid function

```
        )
        self.fc = nn.Sequential(
            nn.Linear(3*128*128, 181),
            nn.ReLU(),
            nn.Linear(181, 11),
            nn.Sigmoid()
            # nn.Linear(512*4*4, 1024),
            # nn.ReLU(),
            # nn.Linear(1024, 512),
            # nn.ReLU(),
            # nn.Linear(512, 11)
        )
```

參數量為：3*128*128*181+181*11 ＝8898503

跟原本 model 相比的參數變化量為 ：

9*(64+128+256+512+512)+512*4*4*1024+1024*512+512*11-3*128*128*181 -181*11＝33273

結果為

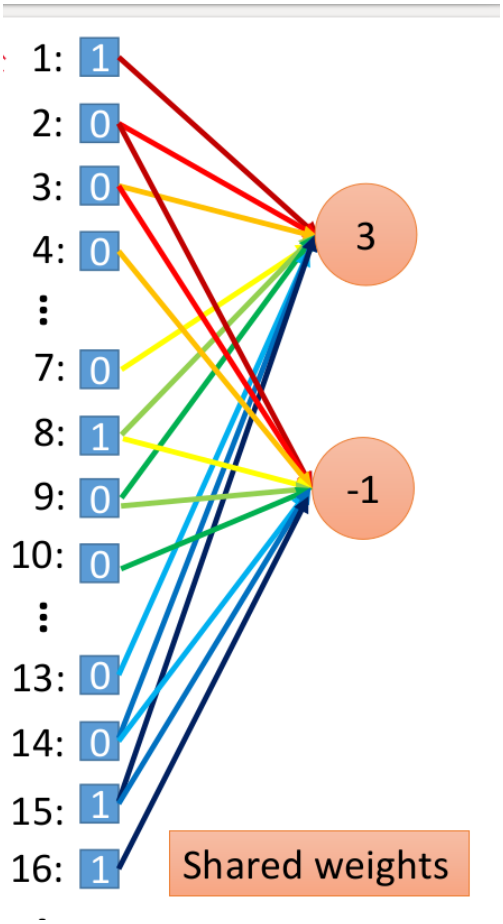
```
5.60 sec(s) Train Acc: 0.259984 Loss: 0.032524 | Val Acc: 0.260641 loss: 0.032458
```

結果也是很慘：accuracy 只有0.26

我認為 DNN 要以 ”參數量”為標準去跟 cnn 比較 performance 是行不通的, 因為實際上 cnn 中每一層 convolution layer 是利用同一組參數去對很多不同的 input 做事情,所以它所執行的動作換到 dnn 的話相當於是更多的參數量,但因為是共用參數，所以表面上看起來參數量很少　　dnn 若用相同的參數量很明顯 model 會不夠力，導致 underfitting

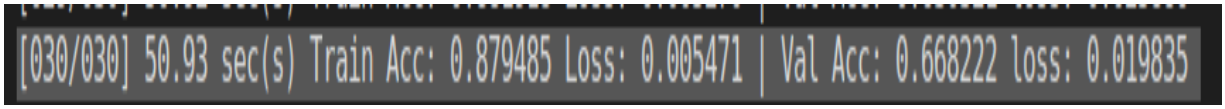
4. 請說明由 1 ~ 3 題的實驗中你觀察到了什麼？(1%)

首先，若將減少的 convolution layer 的參數量拿去建構 DNN，這會使得 model 的泛化能力明顯下降，我認為是因為 convolution layer 雖然是共用參數，但是卻能同時餵給很多不同 input，若將 cnn 的概念想像成 dnn，則每個 neuron 會連到 number of kernel 個 input，而有幾個 filter channel 就代表有幾個 neuron，而換成同樣的參數量在 DNN 的時候，就變成只有一個 neuron 連著 number of kernel 個 input，很明顯 model 的複雜度就下降了很多



所以從前面題目的實驗中，可以了解到 convolution layer 的好處在於節省了很多參數所佔空間，卻依然能考慮到很多不同的 input 資訊，在記憶體的使用上也比較友善（如我的 gpu 只有4GB 就感受深刻啊）

接著 我發現最開始的 model 有 overfitting 的問題，training accuracy 跟 validation 的差的很多，我認為有可能是 fully connected network 的參數太多，也有可能是 training data 不夠，因此打算嘗試 regularization 以及 data augmentation



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%)

batch size:64

epoch time:15

Model: 5層 convolution layer 3層 neural network

activation function: Relu

```
self.cnn = nn.Sequential(
    nn.Conv2d(3, 64, 3, 1, 1),  # [64, 128, 128]
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),      # [64, 64, 64]

    nn.Conv2d(64, 128, 3, 1, 1),  # [128, 64, 64]
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),      # [128, 32, 32]

    nn.Conv2d(128, 256, 3, 1, 1),  # [256, 32, 32]
    nn.BatchNorm2d(256),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),      # [256, 16, 16]

    nn.Conv2d(256, 512, 3, 1, 1),  # [512, 16, 16]
    nn.BatchNorm2d(512),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),       # [512, 8, 8]

    nn.Conv2d(512, 512, 3, 1, 1),  # [512, 8, 8]
    nn.BatchNorm2d(512),
    nn.ReLU(),
    nn.MaxPool2d(2, 2, 0),       # [512, 4, 4]
```

結果如下

	Augmentation and Normalization	Nomalization	None
Train accuracy	0.716501	0.807318	0.437462
Validation accuracy	0.576968	0.583090	0.390087

Details:

執行了 data augmentation + data normalization:

1.做圖片水平翻轉

2.做隨機旋轉

3.用 toTensor 做 normalization

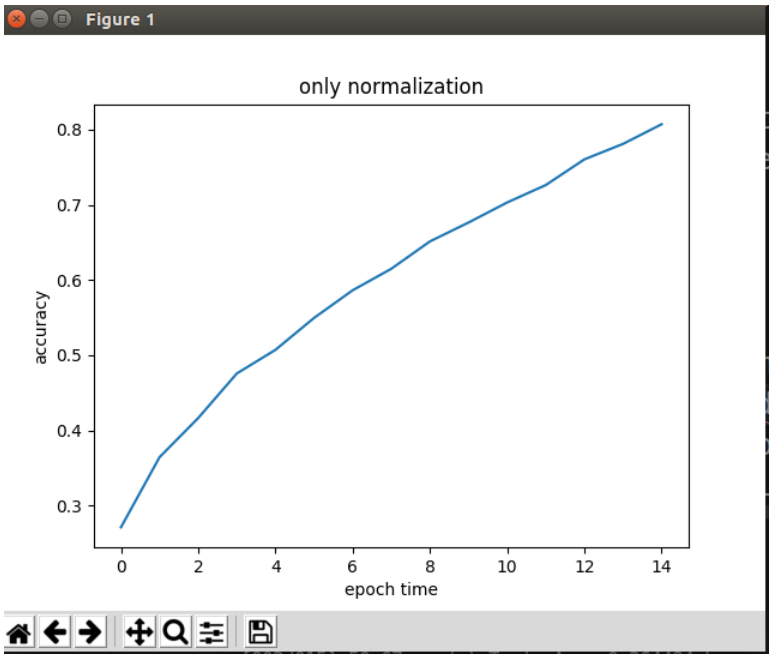
```
# training 時做 data augmentation
train_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.RandomHorizontalFlip(),  # 隨機將圖片水平翻轉
    transforms.RandomRotation(15),  # 隨機旋轉圖片
    # 將圖片轉成 Tensor，並把數值normalize到[0,1](data normalization)
    transforms.ToTensor(),
])
# testing 時不需做 data augmentation
test_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.ToTensor(),
])
```

結果為：train accuracy:0.716501 validation accuracy:0.576968

```
[015/015] 50.89 sec(s) Train Acc: 0.716501 Loss: 0.012736 | Val Acc: 0.576968 loss: 0.021399
```

只進行 normalization:

同上步驟但是把水平翻轉 旋轉的部份弄掉




```
49.60 sec(s) Train Acc: 0.807318 Loss: 0.008465 | Val Acc: 0.583090 loss: 0.023023
```

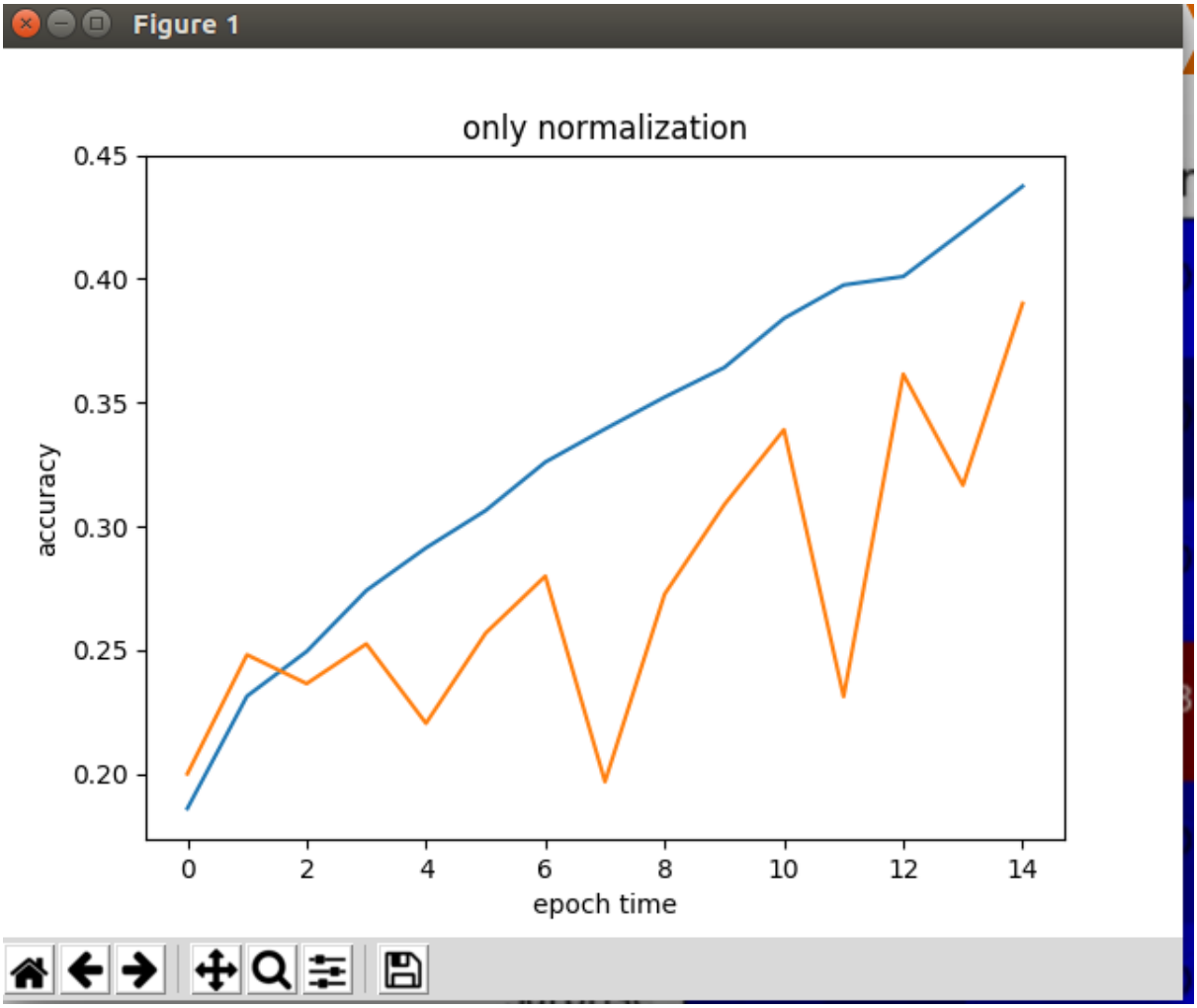
train accuracy:0.807318 validation accuracy:0.583090

train accuracy 比起 augmentation normalization 都做的狀況進步了很多，比較令我意外的是 validation accuracy 也上升了

我認為可能的原因是 data augmentation 是一種可以預防 overfitting 的方法，但是因為 epoch time 並不高，有可能 model 根本還處在 underfitting 的狀況，因此 data augmentation 的效果並不好

without data augmentation and Normalization:

作法：把 torchvision transform 的部份全部取消，transform 的 function 用 numpy 轉換 tensor 的函式取代，將原本（H,W,C）的 numpy 轉換成為（C,H,W）的 tensor



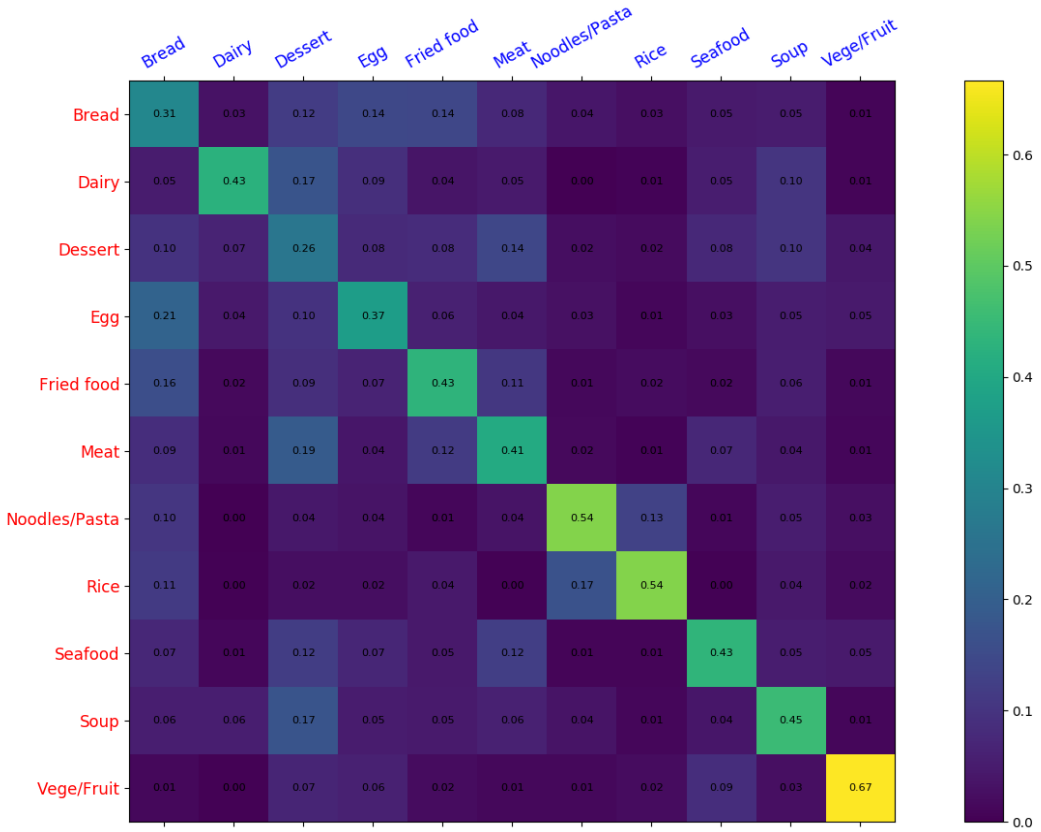

```
49.75 sec(s) Train Acc: 0.437462 Loss: 0.024979 | Val Acc: 0.390087 loss: 0.028198
```

train accuracy:0.437462 validation accuracy:0.390087

沒有做 normalization 的話 input 的 value 會在0～255之間，我認為這會使得特定 convolution 時不容易偵測出特定 feature（例如255對結果的影響 會遠大於0對結果的影響）

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%)

紅色的字代表真實 label



顏色越靠近黃色代表預測機率越高

藍色的字代表預測的 label

先吐嘈一下這個悲慘的準確率，並列出每個 label 容易預測錯的前三名

BREAD	Egg	Seafood	Dessert
DAIRY PRODUCT	Dessert	Egg	Soup
Dessert	Bread	Meat	Soup
Egg	Bread	Dessert	Fried food
Fried food	Bread	Meat	Dessert
Meat	Dessert	Fried food	Bread
Noodles/Pasta	Rice	Bread	Dessert
Rice	Noodles	Bread	Fried food
Seafood	Dessert	Meat	Egg
Soup	Dessert	Bread	Meat
Vegetable/Fruit	Seafood	Dessert	Egg

可以觀察到的是，BREAD 幾乎在每一個 label 都有出現，Dessert 也很常出現，這兩種食物在此 model 的預測似乎不太精確，這一點也可以從 confusion matrix 看出來，你可以看到這兩種食物的預測準確率都不是太好。

這張 confusion matrix 可以看到，Vegetable/fruilt 的預測率最好，所以在上面表格當中你會發現他完全沒有出現過。

觀察以上現象我得到的結論是:造成預測錯誤的狀況發生，很大一個原因是 model 不太能準確地去辨識那些被誤判的 label，以 bread 而言，因為 model 不太能辨識 bread，所以在預測別的 class 的時候，model 就是常會誤認其他 class 為 BREAD。