學號：B05901111 系級： 電機三 姓名：陳建成

1. (2%) 請說明你實作的 CNN model，其模型架構、訓練參數和準確率為何？並請用與上述 CNN 接近的參數量，實做簡單的 DNN model，同時也說明其模型架構、訓練參數和準確率為何？並說明你觀察到了什麼？
(Collaborators: None)

答：CNN model 的部分，我挑選 Kaggle 上第一個以單一 model （即非 ensemble ）於 public Score 上表現最佳的結構，然後盡量模仿其各層參數量並調整去實作一個簡單的 DNN model。

1) CNN model
   a) 結構
   參考 VGG 的堆疊方法，使用 Leaky ReLU，並開 dropout 0.4

```
model = Sequential()
model.add(BatchNormalization(input_shape=input_shape))
model.add(Conv2D(64, (5, 5), padding='same'))
model.add(LeakyReLU(0.2))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.4))

model.add(BatchNormalization(input_shape=input_shape))
model.add(Conv2D(128, (5, 5), padding='same'))
model.add(LeakyReLU(0.2))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))

model.add(BatchNormalization(input_shape=input_shape))
model.add(Conv2D(256, (5, 5), padding='same'))
model.add(LeakyReLU(0.2))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.4))

model.add(BatchNormalization(input_shape=input_shape))
model.add(Conv2D(256, (5, 5), padding='same'))
model.add(LeakyReLU(0.2))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.4))

model.add(Flatten())

model.add(Dense(256, use_bias=True))
model.add(LeakyReLU(0.2))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
```

Layer (type)	Output Shape	Param #
batch_normalization_9 (Batch	(None, 48, 48, 1)	4
conv2d_9 (Conv2D)	(None, 48, 48, 64)	1664
leaky_re_lu_11 (LeakyReLU)	(None, 48, 48, 64)	0
max_pooling2d_9 (MaxPooling2	(None, 24, 24, 64)	0
dropout_11 (Dropout)	(None, 24, 24, 64)	0
batch_normalization_10 (Batc	(None, 24, 24, 64)	256
conv2d_10 (Conv2D)	(None, 24, 24, 128)	204928
leaky_re_lu_12 (LeakyReLU)	(None, 24, 24, 128)	0
max_pooling2d_10 (MaxPooling	(None, 12, 12, 128)	0
dropout_12 (Dropout)	(None, 12, 12, 128)	0
batch_normalization_11 (Batc	(None, 12, 12, 128)	512
conv2d_11 (Conv2D)	(None, 12, 12, 256)	819456
leaky_re_lu_13 (LeakyReLU)	(None, 12, 12, 256)	0
max_pooling2d_11 (MaxPooling	(None, 6, 6, 256)	0
dropout_13 (Dropout)	(None, 6, 6, 256)	0
batch_normalization_12 (Batc	(None, 6, 6, 256)	1024
conv2d_12 (Conv2D)	(None, 6, 6, 256)	1638656
leaky_re_lu_14 (LeakyReLU)	(None, 6, 6, 256)	0
max_pooling2d_12 (MaxPooling	(None, 3, 3, 256)	0
dropout_14 (Dropout)	(None, 3, 3, 256)	0
flatten_3 (Flatten)	(None, 2304)	0
dense_5 (Dense)	(None, 256)	590080
leaky_re_lu_15 (LeakyReLU)	(None, 256)	0
dropout_15 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 7)	1799

   b) 參數量 = 3,258,379
   其中 3,257,481 可訓練
   898 不可訓練

   c) 準確率
   我是參照助教的方式（以前沒試過 Kaggle 的 callback），去存下 validation accuracy 最高

的 model 參數來做 prediction

最佳 validation accuracy 出現於第90個epoch，此時
training 準確率與 loss 值（在 training set 與 validation set 上）為
loss: 0.8997 - acc: 0.6592 - val_loss: 0.8740 - val_acc: 0.68234
而當訓練結束時最後的時候則是
loss: 0.8852 - acc: 0.6685 - val_loss: 0.8844 - val_acc: 0.6813
上傳 Kaggle 後，private score 為 0.67344 而 public score 為 0.67205

2) DNN model
   a) 結構
   原則上層數跟剛才差不多（把過一次Dense或Convolution才當作一層的話），參數量是盡量調得跟上面差不多（只差16個，夠接近了吧！）。然後在中間原本convolution層相對位置不放bias，並用原本的feature map的倍數設置neurons

```
model = Sequential()
model.add(BatchNormalization(input_shape=input_shape))
model.add(Flatten())
model.add(Dense(24 * 24 * 2))
model.add(Dropout(0.4))
model.add(LeakyReLU(0.2))

model.add(BatchNormalization(input_shape=input_shape))
model.add(Dense(12 * 12 * 3, use_bias=False))
model.add(Dropout(0.4))
model.add(LeakyReLU(0.2))

model.add(BatchNormalization(input_shape=input_shape))
model.add(Dense(6 * 6 * 5, use_bias=False))
model.add(Dropout(0.4))
model.add(LeakyReLU(0.2))

model.add(BatchNormalization(input_shape=input_shape))
model.add(Dense(3 * 3 * 10, use_bias=False))
model.add(Dropout(0.4))
model.add(LeakyReLU(0.2))

model.add(Dense(44, use_bias=True))
model.add(Dropout(0.5))
model.add(LeakyReLU(0.2))
model.add(Dense(7, activation='softmax'))
```

Layer (type)	Output Shape	Param #
batch_normalization_1 (Batch	(None, 48, 48, 1)	4
flatten_1 (Flatten)	(None, 2304)	0
dense_1 (Dense)	(None, 1152)	2655360
dropout_1 (Dropout)	(None, 1152)	0
leaky_re_lu_1 (LeakyReLU)	(None, 1152)	0
batch_normalization_2 (Batch	(None, 1152)	4608
dense_2 (Dense)	(None, 432)	497664
dropout_2 (Dropout)	(None, 432)	0
leaky_re_lu_2 (LeakyReLU)	(None, 432)	0
batch_normalization_3 (Batch	(None, 432)	1728
dense_3 (Dense)	(None, 180)	77760
dropout_3 (Dropout)	(None, 180)	0
leaky_re_lu_3 (LeakyReLU)	(None, 180)	0
batch_normalization_4 (Batch	(None, 180)	720
dense_4 (Dense)	(None, 90)	16200
dropout_4 (Dropout)	(None, 90)	0
leaky_re_lu_4 (LeakyReLU)	(None, 90)	0
dense_5 (Dense)	(None, 44)	4004
dropout_5 (Dropout)	(None, 44)	0
leaky_re_lu_5 (LeakyReLU)	(None, 44)	0
dense_6 (Dense)	(None, 7)	315

   b) 參數量 = 3,258,363
   其中 3,254,833 可訓練
   3,530 不可訓練

   c) 準確率
   最佳 validation accuracy 也恰巧出現於第90個epoch,
   此時 training 準確率與 loss 值

（在 training set 與 validation set 上）為
loss: 1.5945 - acc: 0.3768 - val_loss: 1.4431 - val_acc: 0.4528
而當訓練結束時最後的時候則是
loss: 1.5939 - acc: 0.3773 - val_loss: 1.4318 - val_acc: 0.4500
上傳 Kaggle 後，private score 為 0.44692 而 public score 為 0.44441

觀察：很明顯CNN的訓練結果在相同參數量下遠比DNN高，且準確率上升得快、loss降得快並較晚出現收斂。從下面圖中更明顯可看出此結論。當然還有一個差異：CNN train起來明顯跑得比較慢……

2. (1%) 承上題，請分別畫出這兩個model的訓練過程 (i.e., loss/accuracy v.s. epoch)
(Collaborators: None)

答：各自訓練 100 個 epochs

CNN model 的訓練過程：

DNN model 的訓練過程：

3. (1%) 請嘗試 data normalization, 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？
(Collaborators: None)

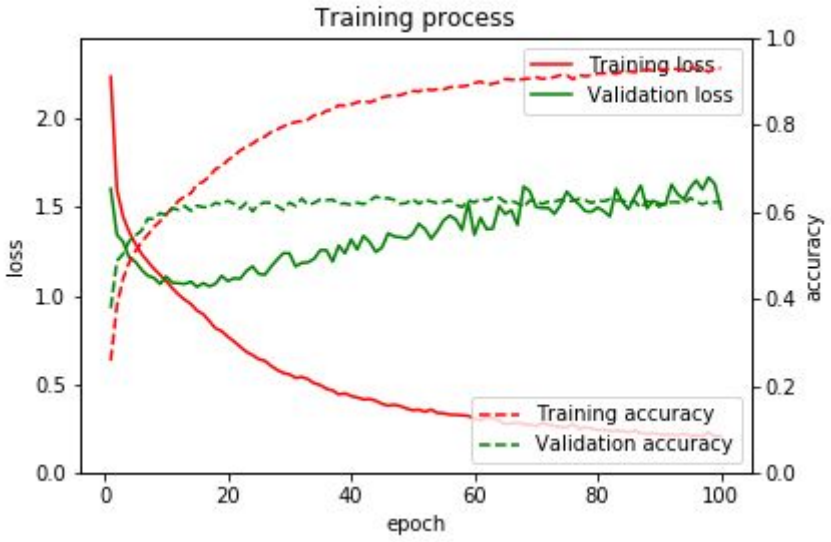
答： 我的data normalization使用把data除以255的方式將值都限制在0～1之間， 而data augmentation 則除了手動作左右翻轉外，還加上Keras內的ImageDataGenerator當場生成額外的資料。（當然其中horizontal_flip是與手動的重複了，但是這樣感覺總data量比較多？）

```
datagen = ImageDataGenerator(
    horizontal_flip=True,
    zoom_range=0.1,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1
    )
```

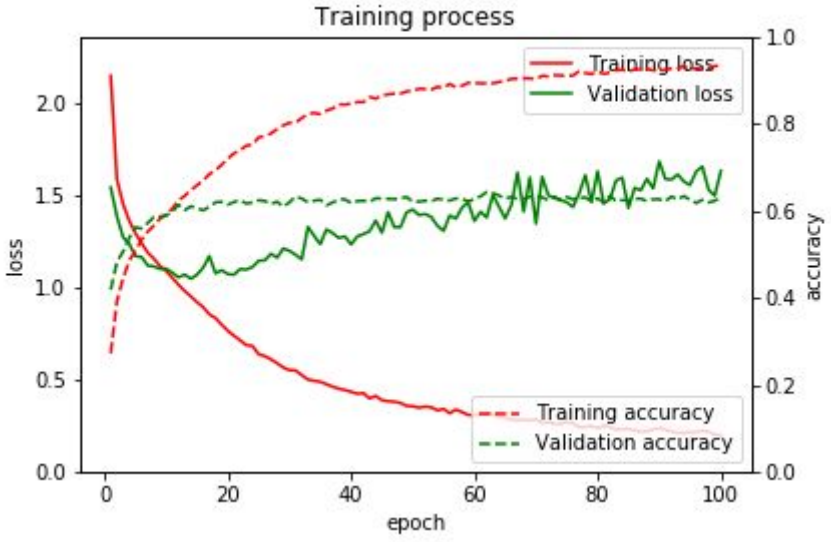
依照助教給定的要求，分成三種case去實驗：

1) 兩者都不使用
Best epoch: 44/100

loss: 0.4070 - acc: 0.8559 - val_loss: 1.2466 - val_acc: 0.63427
Last epoch
loss: 0.2067 - acc: 0.9289 - val_loss: 1.4902 - val_acc: 0.6217
Kaggle：private score = 0.61186， public score = 0.62886



2) 只使用 data normalization
Best epoch: 62/100
loss: 0.3185 - acc: 0.8917 - val_loss: 1.3770 - val_acc: 0.64228
Last epoch
loss: 0.1961 - acc: 0.9322 - val_loss: 1.6292 - val_acc: 0.6294
Kaggle：private score = 0.60908， public score = 0.62385



3) 同時做 data normalization + data augmentation
Best epoch: 99/100
loss: 0.8860 - acc: 0.6666 - val_loss: 0.8706 - val_acc: 0.6855

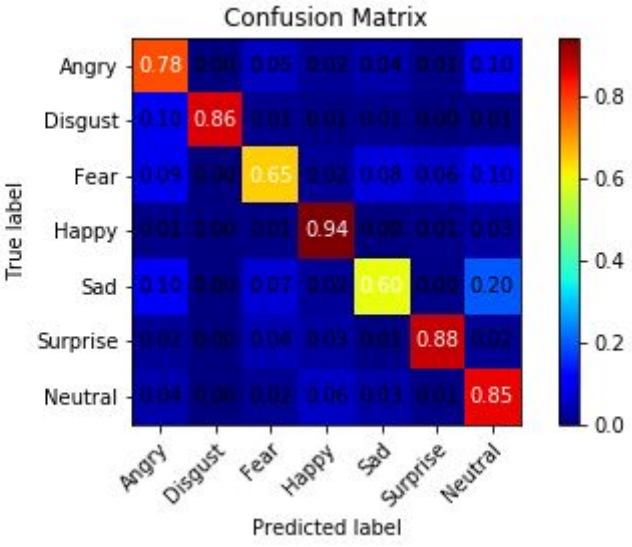
Last epoch
loss: 0.8870 - acc: 0.6707 - val_loss: 0.9190 - val_acc: 0.6764
Kaggle：private score = 0.67010， public score = 0.67957

從三個實驗去比較，其實我們可以明顯看出，加了data augmentation的準確率明顯上升，而比對了training process就可以發現，data augmentation可以大大降低overfitting的狀況（前兩者在training上的accuracy都明顯高到90%但validation上並沒有）。至於data normalization是有一些小幫助，但是看不太出來，或許是因為除以255這種normalization太簡單，或許用標準化（減去平均除以標準差）會更好。

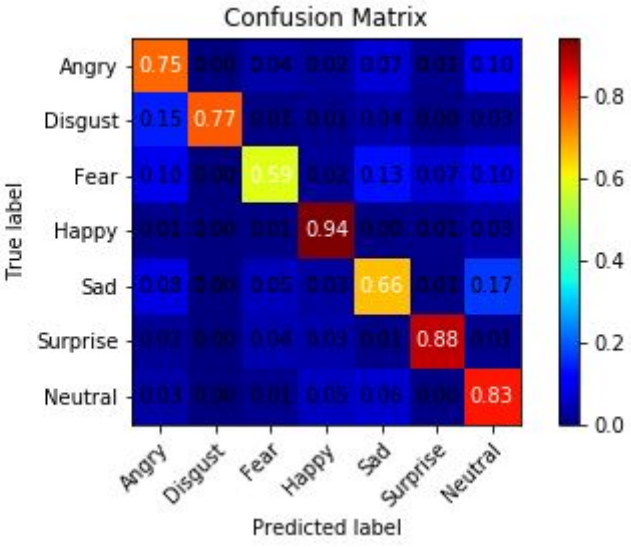
4. (1%) 觀察答錯的圖片中，哪些 class 彼此間容易用混？[繪出 confusion matrix 分析]
(Collaborators: B05901192 張晁維、B05202061 陳威旭)

答：分別就上傳到Kaggle上單一model最高public score與ensemble後的model繪出confusion matrix 分析

1) 單一model



2) ensemble五個model後



基本上從兩個model都可以看出來，fear和sad比較容易判斷錯誤，其中fear會和angry、sad與neutral搞混，而sad則很容易被判斷成neutral（所有判斷錯誤狀況中出現機率最高）。原因應該是sad的表情與neutral相近，都是比較不突出的表情（面無表情有可能其實是sad？），而fear也是較為不明顯的表情。有趣的是，disgust幾乎不會被其他表情用混（就是非disgust的表情都幾乎不可能被判斷為disgust，在兩個model中column都是0.00）或許是因為其特徵明顯（換言之，model說是disgust就幾乎一定是）；另一種特徵明顯則表現在happy上，但是是正確率高的層面。