

## 1.請說明你實作的 CNN model，其模型架構、訓練過程和準確率為何？

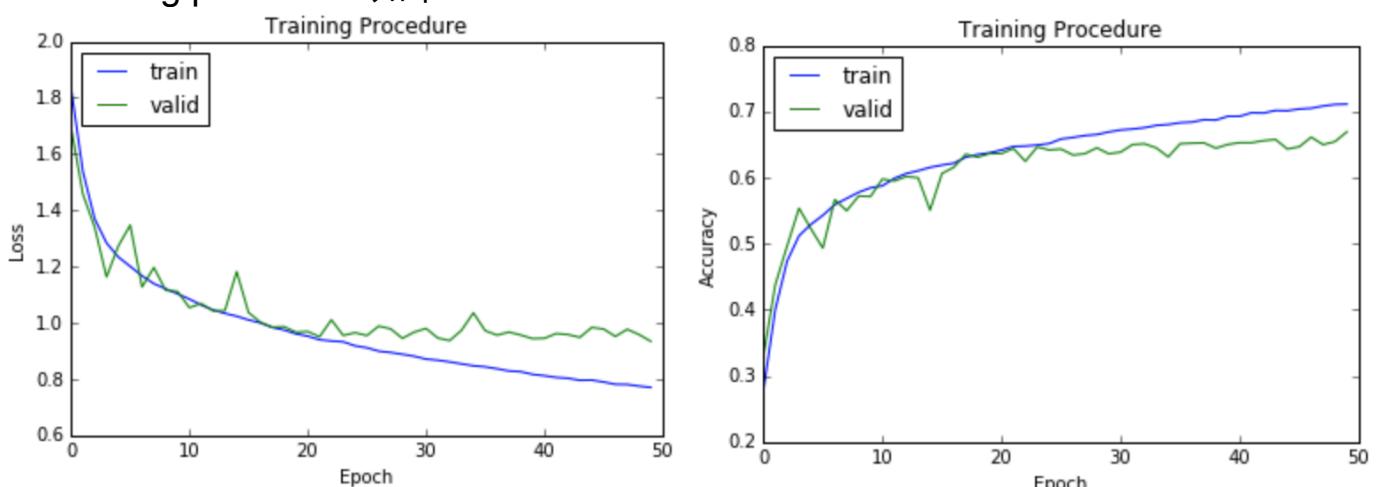
Layer (type)	Output Shape	Param #	
conv2d_11 (Conv2D)	(None, 48, 48, 64)	1664	conv2d_14 (Conv2D) (None, 12, 12, 256) 295168
batch_normalization_15 (BatchNormalization)	(None, 48, 48, 64)	256	batch_normalization_18 (BatchNormalization) (None, 12, 12, 256) 1024
leaky_re_lu_15 (LeakyReLU)	(None, 48, 48, 64)	0	leaky_re_lu_18 (LeakyReLU) (None, 12, 12, 256) 0
dropout_15 (Dropout)	(None, 48, 48, 64)	0	max_pooling2d_11 (MaxPooling2D) (None, 6, 6, 256) 0
conv2d_12 (Conv2D)	(None, 48, 48, 64)	36928	dropout_18 (Dropout) (None, 6, 6, 256) 0
batch_normalization_16 (BatchNormalization)	(None, 48, 48, 64)	256	conv2d_15 (Conv2D) (None, 6, 6, 512) 1180160
leaky_re_lu_16 (LeakyReLU)	(None, 48, 48, 64)	0	batch_normalization_19 (BatchNormalization) (None, 6, 6, 512) 2048
max_pooling2d_9 (MaxPooling2D)	(None, 24, 24, 64)	0	leaky_re_lu_19 (LeakyReLU) (None, 6, 6, 512) 0
dropout_16 (Dropout)	(None, 24, 24, 64)	0	max_pooling2d_12 (MaxPooling2D) (None, 3, 3, 512) 0
conv2d_13 (Conv2D)	(None, 24, 24, 128)	73856	dropout_19 (Dropout) (None, 3, 3, 512) 0
batch_normalization_17 (BatchNormalization)	(None, 24, 24, 128)	512	flatten_3 (Flatten) (None, 4608) 0
leaky_re_lu_17 (LeakyReLU)	(None, 24, 24, 128)	0	dense_7 (Dense) (None, 512) 2359808
max_pooling2d_10 (MaxPooling2D)	(None, 12, 12, 128)	0	batch_normalization_20 (BatchNormalization) (None, 512) 2048
dropout_17 (Dropout)	(None, 12, 12, 128)	0	leaky_re_lu_20 (LeakyReLU) (None, 512) 0
			dropout_20 (Dropout) (None, 512) 0
			dense_8 (Dense) (None, 256) 131328
			batch_normalization_21 (BatchNormalization) (None, 256) 1024
			leaky_re_lu_21 (LeakyReLU) (None, 256) 0
			dropout_21 (Dropout) (None, 256) 0
			dense_9 (Dense) (None, 7) 1799
<hr/>			
Total params: 4,087,879			
Trainable params: 4,084,295			
Non-trainable params: 3,584			

我總共用了 5 個 CNN Blocks，2 個 Fully Connected Blocks。

各 CNN Block 皆依序經過 Conv2D, BatchNormalization, LeakyReLU, MaxPooling2D 以及 Dropout，( 但第一個 Block 沒有 Dropout )。Conv2D 的 filters 數量依序為 64, 64, 128, 256, 512、shape 除了第一個 Block 為 (5,5) 其他都是 (3,3)、padding 都是 same。BN 用預設參數，LeakyReLU 的  $\alpha = 0.1$ ，Pooling 的 pool\_size 皆為 (2,2)，Dropout 的 rate 依序為 0.2, 0.2, 0.3, 0.4, 0.4。

經過 flatten 後進入 FC，每個 FC Block 也依序經過 Dense, BN, LeakyReLU 與 Dropout，Dense 的 units 依序為 512, 256，BN 與 ReLU 同上，dropout\_rate 皆為 0.5。最後再進 7 units, softmax 的 Dense 預測標籤。

訓練過程運用 keras 的 ImageDataGenerator 對 train data 做 rotate, zoom, horizon\_flip, shear 來增加 data 量並提升 model 穩定性。fit\_generator 時我的 step\_per\_epoch=500, batch\_size=128，亦即每個 epoch 會處理 64,000 筆。Training procedure 如下：



大約在 30 個 epoch 後 valid 就會趨於穩定。上述 model 在 testing 上的平均 accuracy 為 0.6719，若把 validation data 一起 train，則可以達到 0.6755。我在 Kaggle 上表現最佳的為 ensemble 5 個相似 model，可提升至 0.6833。

## 2. 承上題，請用與上述 CNN 接近的參數量，實做簡單的 DNN model。其模型架構、訓練過程和準確率為何？試與上題結果做比較，並說明你觀察到了什麼？

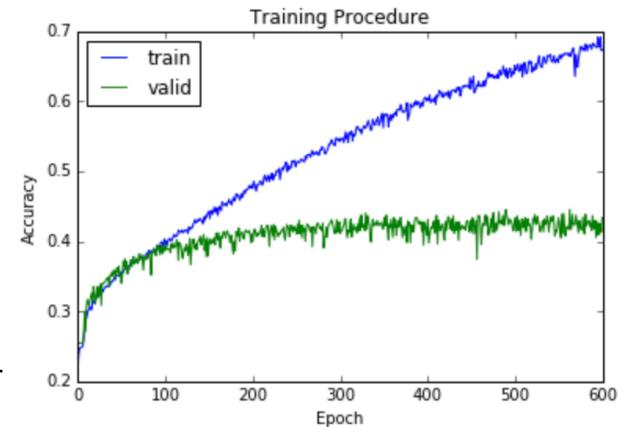
DNN 與 CNN 參數量皆約為 409 萬。我在 DNN 中用了 5 個 FC Blocks。每個 block 包含 Dense, LeakyReLU, Dropout。Dense units 依序為 1024, 1024, 512, 256, 128，ReLU 同上。dropout\_rate 依序為 0.3, 0.3, 0.4, 0.5, 0.5。最後再進 7 units, softmax 的 Dense 預測標籤。Optimizer 是預設參數的 adam，Loss=crossentropy。另外需注意，在 DNN 中沒有做 Data Augmentation。所以每個 epoch 處理的 Datat 量會變成原先的 1/3 左右，epoch 數應該相應提升。

Layer (type)	Output Shape	Param #
dense_40 (Dense)	(None, 1024)	2360320
leaky_re_lu_33 (LeakyReLU)	(None, 1024)	0
dropout_33 (Dropout)	(None, 1024)	0
dense_41 (Dense)	(None, 1024)	1049600
leaky_re_lu_34 (LeakyReLU)	(None, 1024)	0
dropout_34 (Dropout)	(None, 1024)	0
dense_42 (Dense)	(None, 512)	524800
leaky_re_lu_35 (LeakyReLU)	(None, 512)	0
dropout_35 (Dropout)	(None, 512)	0
dense_43 (Dense)	(None, 256)	131328
leaky_re_lu_36 (LeakyReLU)	(None, 256)	0
dropout_36 (Dropout)	(None, 256)	0
dense_44 (Dense)	(None, 128)	32896
leaky_re_lu_37 (LeakyReLU)	(None, 128)	0
dropout_37 (Dropout)	(None, 128)	0
dense_45 (Dense)	(None, 7)	903

Total params: 4,099,847  
Trainable params: 4,099,847  
Non-trainable params: 0

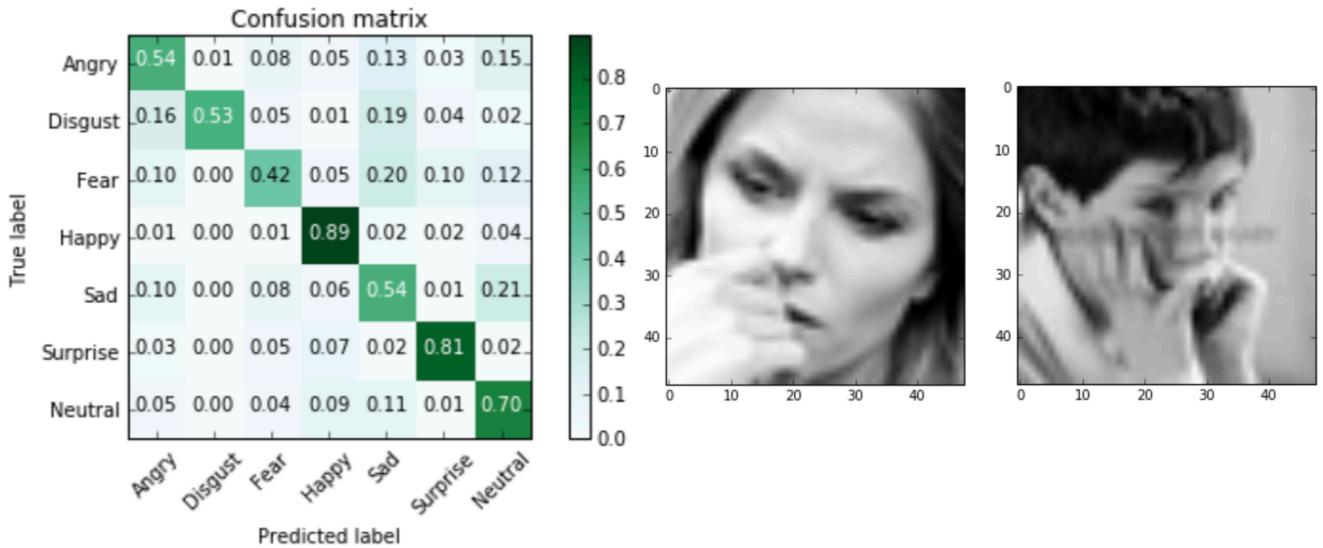
由 Training Procedure 可以發現 DNN 大約在 100 多個 epoch 後收斂。但相同 epoch 下成果比 CNN 差很多，且無論在 valid 或 test 上最終能達到的效果都遠遜 CNN (DNN acc 都是 0.44 左右)。如果加入 BN，則收斂速度會極快，約 10 個 epoch 即可，但 acc 也差不多糟。

另外，DNN 在 training 速度上遠快於 CNN，一個 epoch 僅需 2 秒左右 (CNN 則需 60 秒)，推測是因為做 convolution 時所需運算量較大。



## 3. 觀察答錯的圖片中，哪些 class 彼此間容易用混？[繪出 confusion matrix 分析]

觀察 confusion matrix 可發現，Fear 很容易被誤判為 Sad，Sad 則容易被誤判為 Neutral。以下兩者各舉一個例子：女性為 Fear，男孩為 Sad。



對女性預測出的機率為 [7.5e-2, 1.1e-2, 0.33, 1.6e-4, **0.52**, 4.4e-4, 5.9e-2]。

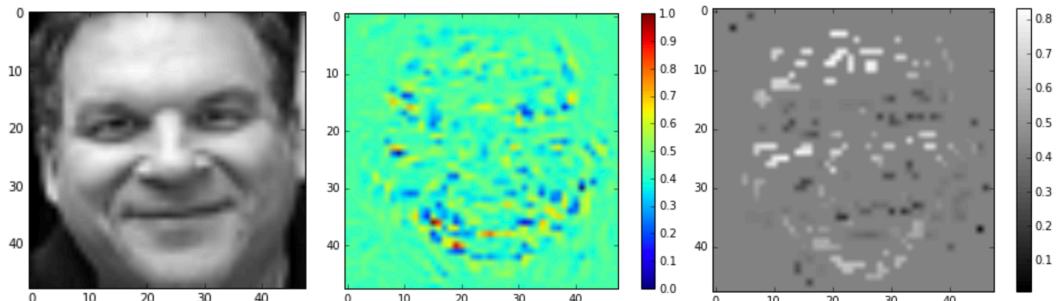
可以發現 Fear 的確是第二高的，且遠高於剩下其他 label。

對男孩預測出的機率為 [ 0.02, 0, 0.2, 0.01, 0.33, 0.09, **0.35**]。

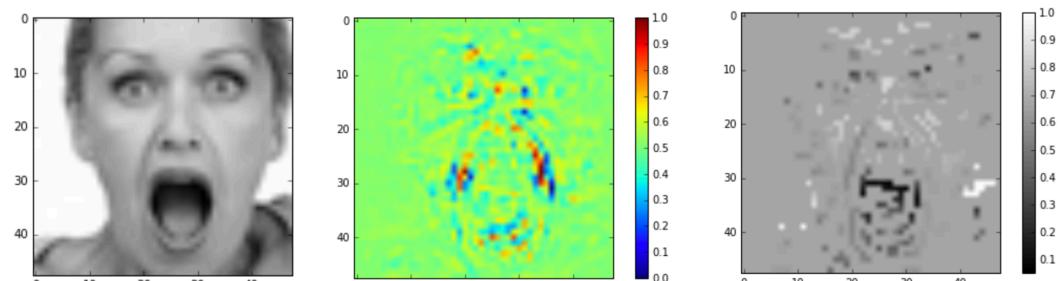
可以發現 Sad 的確是第二高、和 Neutral 很相近，且遠高於剩下其他 label。

就我自己觀察認為，上述兩張圖片其實都不容易判斷。要說 model 判出來的才是正解其實我也很能接受，甚至我認為男孩應該更接近 Sad 才對。情緒這麼主觀的東西有時連人都判不出來了、機器判錯也是很合理的。

#### 4. 從(1)(2)可以發現，使用 CNN 的確有些好處，試繪出其 saliency maps，觀察模型在做 classification 時，是 focus 在圖片的哪些部份？



例子 1 是個 Happy 臉，我的 model 做了正確預測。可以發現模型似乎著重其額頭、嘴巴以及兩頰的輪廓。我覺得嘴形跟兩頰的確是判斷 Happy 的重要依據，但額頭為什麼也有很高的值我就不太清楚了。

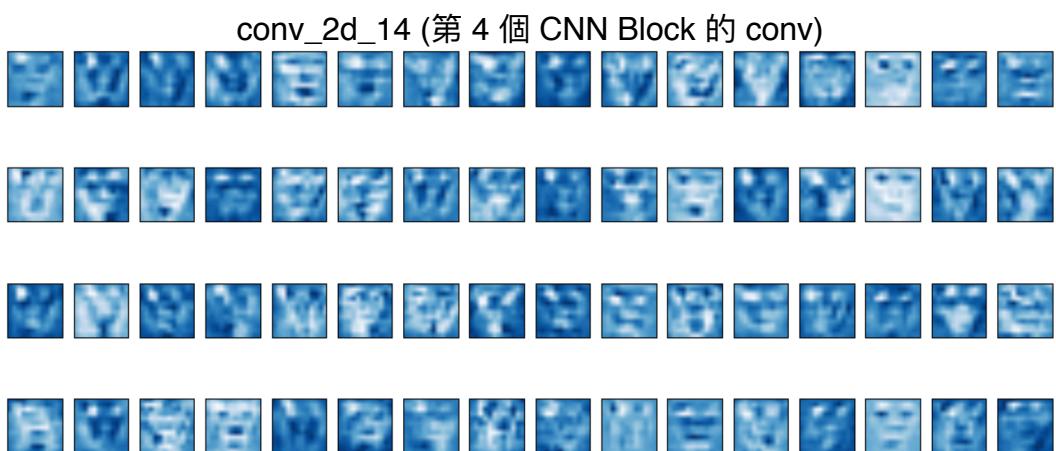
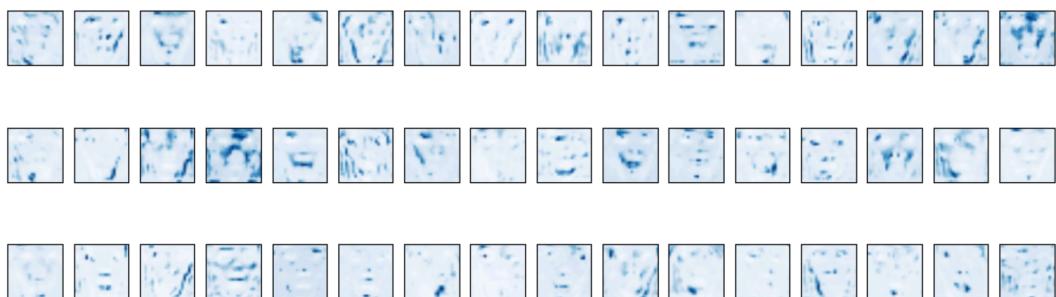
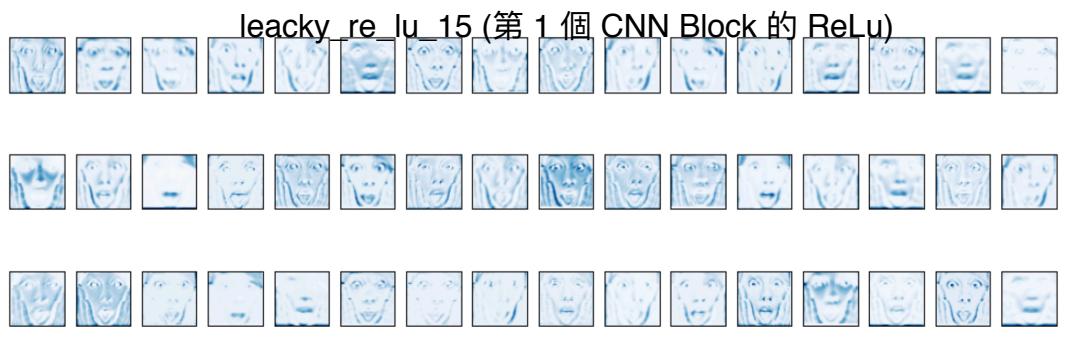
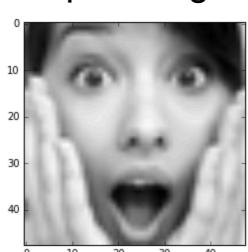


例子 2 則是 Surprise 臉，我的 model 一樣預測正確。模型著重其上揚的眉毛、張大的嘴巴以及被牽動的法令紋，感覺還算合理。

5. 承(1)(2)，利用上課所提到的 gradient ascent 方法，觀察特定層的filter最容易被哪種圖片 activate。

(1) Filters output:

Input image:



上述四張圖為由淺至深的 filters output。可以明顯觀察到較淺層的 output 會抓到整張臉的輪廓，尤其第一層 ReLu 非常明顯。而越深看起來就越抽象了，像 conv\_2d\_14 可以看見一些眼睛、嘴巴，到 leaky\_re\_lu\_19，也就是我最後一個 CNN Block 的 ReLu，就看不太出明顯的輪廓或器官，比較像是偵測眼睛、嘴巴的位置。

## (2) Gradient Ascent:



能 active 較淺層的圖片看起來是一些有粗有細的紋路，許多紋路都很類似，只是方向不同，推測是因為輸入的圖片臉會有各種角度，所以都需要偵測出來。對於較深層的 filters，對應圖片則較複雜多樣，其中有滿多螺旋狀的紋路，有些看起來像眼睛，有些則太抽象以致難以判斷是什麼。