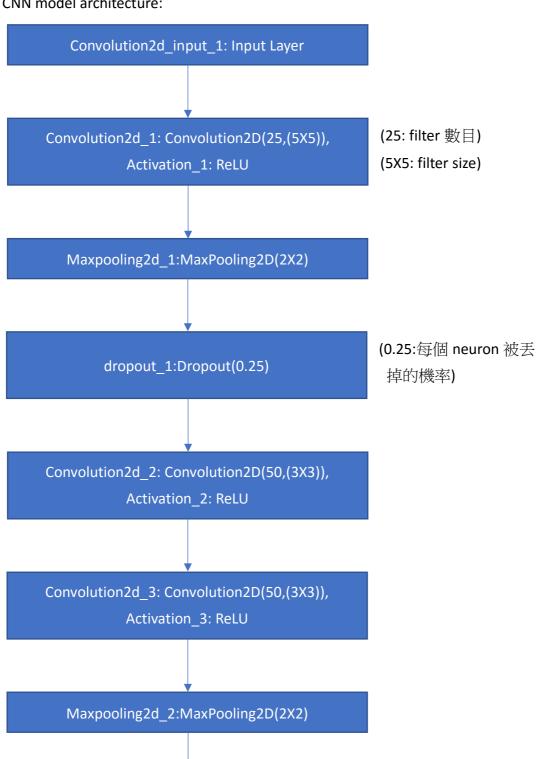
學號:R05942101 系級: 電信一 姓名:陳泓弦

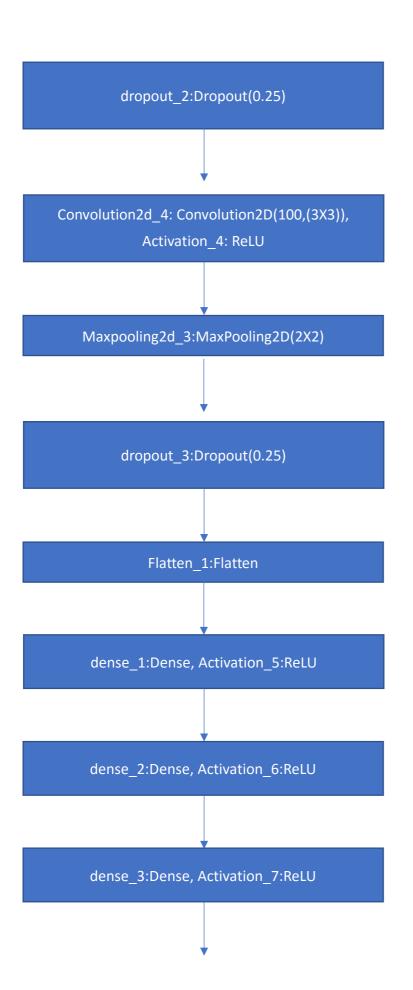
1.(1%) 請說明你實作的 CNN model, 其模型架構、訓練過程和準確率為

何?

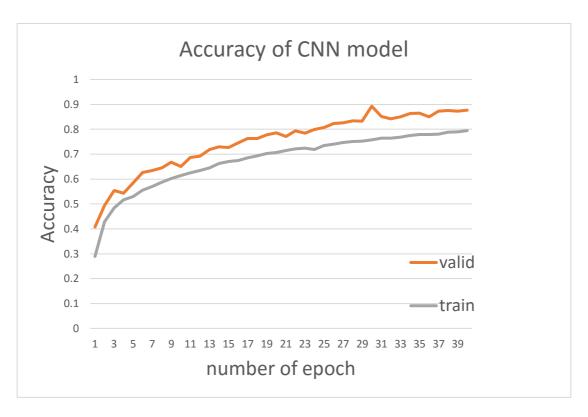
答:

## CNN model architecture:



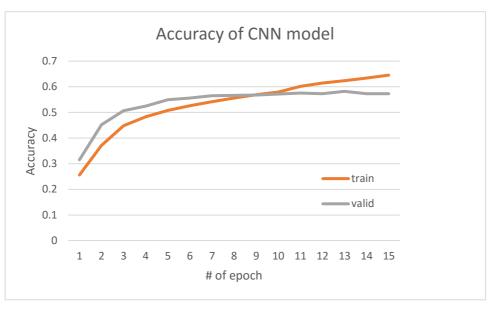


圖(一)是 Kaggle 得到 0.60435 的 CNN model 的 Accuracy,在訓練前我將原本的圖片做反轉,額外產生一組圖片做為 training set。比較令我感到困惑的是 validation data 的 accuracy 一直比 training data 高,後來發現不小心將 validation data 放進新產生的 data 中,模型在訓練的時候其實就會看到 validation set 的資料 雖然有變形但都差不多,所以會學到 validation set 的圖片,導致 validation 的 accuracy 一直都比 training 的 accuracy 高。雖然從圖(一)可以看到 valid\_accur可以到 0.9,但在 kaggle 只拿到了 0.60435,很明顯這個 validaiiton set 没有什麼可信度,因為他已經被訓練過了。



圖(一) Accuracy of CNN model

後來改掉上面的問題,將後 7000 筆 data 當做 validation set,得到 accuracy 如圖 (二),可以發現在第 10 個 epoch,train 和 valid 的 accuracy 交叉了,之後 train 繼續上升,valid 差不多到極限 0.58,與丟到 kaggle 上的分數 0.57286 很接近,明顯這組 validation set 很有可信度,可以用來 predict 在 testing set 得到的結果。



圖(二) Accuracy of CNN model

2.(1%) 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?

答:

從問題一我們已經知道 validation set 可以 predict 在 kaggle 上的分數。

CNN 模型的總參數: 351863, epoch = 40, val\_accr = 0.58

flatten_1 (Flatten)	(None,	900)	0
haha (Dense)	(None,	256)	238656
hehe (Dense) le "CNN py", line 135, in <mc< td=""><td>(None; (None;</td><td>1128)TA: 864</td><td>s — loss: 1<b>32896</b> — acc:</td></mc<>	(None; (None;	1128)TA: 864	s — loss: 1 <b>32896</b> — acc:
dense_1 (Dense)	(None,	64) Versions/3.6,	alidation_data_(x_valid /lib/pythono.6/site-pac
dropout_4 (Dropout)	n.frdNone,	<b>64)</b> ions/3.6	/lib/pythor <b>0</b> .6/site-pac
prediction/(Dense)ks/Pytho	n.frd(None,	¼ <b>?)</b> sions/3.6,	/lib/pythor <b>455</b> /site-pac
Total params: 351,863 Trainable params: 351,863 Non-trainable params: 0			/lib/python3.6/site-pac /lib/python3.6/site-pac

圖(三) Total number of parameters of CNN model

DNN 模型的總參數: 384757, epoch =40, val accr = 0.3641

Using TensorFlow backend.	· · · · · · · · · · · · · · · · · · ·	
Layer (type)/python3.6/s	sit Output Shape Lensor	flow/p <b>Param</b> n#client/
dense_1 (Dense)	(None, 100)	230500
dense_2 (Dense)	(None, 250) tensor	710W/p <mark>25250</mark> 7/client/
dense_3 (Dense)	(None, 500)	125500
dense_4 (Dense)	(None, 7)	3507
Total params: 384,757 Trainable params: 384,757 Non-trainable params: 0	.cc:45] The TensorFl ne and could speed u .cc:45] The TensorFl	

圖(四) Total number of parameters of DNN model

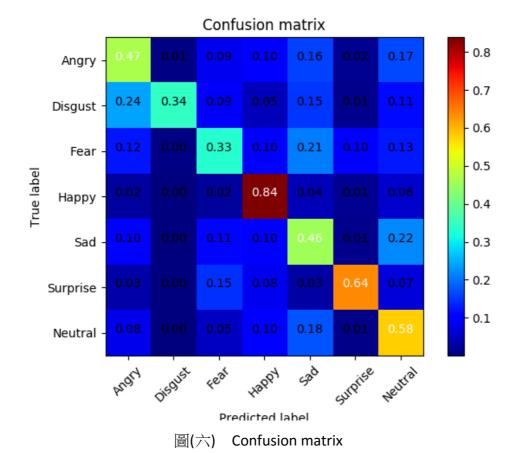
在 DNN 模型中做了 4 層 neural network,包含 input layer 和 output layer 雖然兩個 model 用的總參數量是差不多的(甚至 DNN 還高出一些),而且訓練用的 data 量以及 batch、epoch 值都相同,但從 validation data 的準確率可以發現到 CNN 的 performance 遠比 DNN 好很多,而且 DNN 大概 train 到第 33 個 epoch之後,train accu 仍然會上升,但 val\_accu 就開始震盪,有 overfitting 的問題,使 DNN 最後的 performance 是很差的。

```
Epoch 31/40
21531/21531
                                              - 2s - loss: 1.4766 - acc: 0.4245 - val_loss: 1.5756 - val_acc: 0.3860
 poch 32/40
1531/21531
                                               - 3s - loss: 1.4725 - acc: 0.4290 - val_loss: 1.6000 - val_acc: 0.3855
  och 33/40
                                            =] - 3s - loss: 1.4529 - acc: 0.4312 - val_loss: 1.5803 - val_acc: 0.3931
 1531/21531
 poch 34/40
1531/21531
                                              - 2s - loss: 1.4432 - acc: 0.4373 - val_loss: 1.6100 - val_acc: 0.3884
                                              - 3s - loss: 1.4362 - acc: 0.4388 - val_loss: 1.5913 - val_acc: 0.3852
  och 36/40
 1531/21531
                                              - 2s - loss: 1.4258 - acc: 0.4457 - val_loss: 1.6142 - val_acc: 0.3770
 boch 37/40
 1531/21531
                                               - 2s - loss: 1.4242 - acc: 0.4470 - val_loss: 1.6479 - val_acc: 0.3831
     /21531
                                                2s - loss: 1.4191 - acc: 0.4471 - val_loss: 1.6238 - val_acc: 0.3870
     39/40
                                           =] - 3s - loss: 1.4071 - acc: 0.4526 - val_loss: 1.6324 - val_acc: 0.3767
```

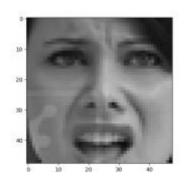
圖(五) training procedure of DNN model

3. (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]

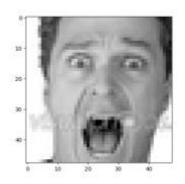
答:



圖(六)是將原本 data 的後 7000 筆當做 validation set,丟進 CNN 模型得到的 Confusion matrix,可以發現"Happy"類的圖片預測的蠻準的,有 0.84 的正確率,而 Disgust 和 Fear 的正確率最低,才 0.33。 Disgust class 有 0.24 會被 predict 成 angry,所以我分別找了一張 angry 和一張 disgust 的圖片來看,如下:左邊是 disgust,右邊是 angry,其實我才分不出來哪張才是 angry,如果要做出更好的 classification,可能要增加 filter 數目才能提高 accuracy,或是增加辨別度比較高的資料,能明顯看出是屬於 disgust 還是 angry。



Class Disgust



Class Angry

4.(1%) 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? 答:

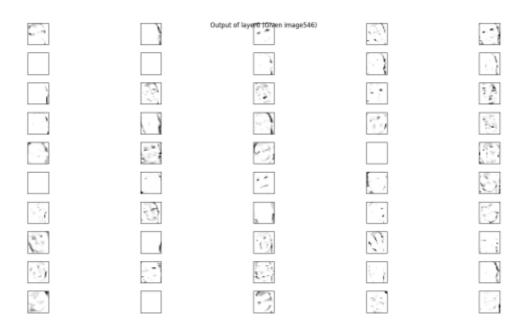
5.(1%) 承(1)(2), 利用上課所提到的 gradient ascent 方法, 觀察特定層的 filter 最容易被哪種圖片 activate。

答:

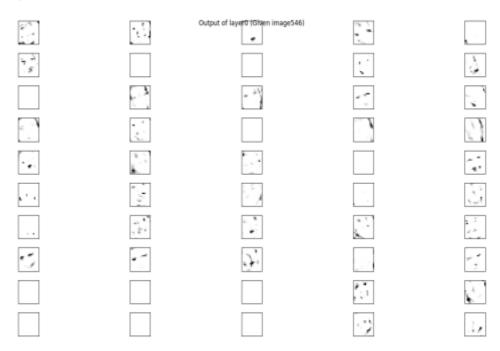
我選了下面這張圖片做為 input ,這張圖片是屬於"Happy",觀察每層 filter 看到了什麼。



首先,第一層 filter 看到的重點如下圖,可以看出臉大致的輪廓,但像是頭髮, 皮膚膚色就看不到,推測第一層把一些較平滑的像素 filter 掉了,只留下 gradient 差很多的部分,即臉的輪廓。



下圖是第二層 filter 看到的影像,某些臉的輪廓已經看不清楚了,大概只能看出眼睛和嘴巴附近的輪廓,但某些 filter 還能看出臉的輪廓。我覺的很神奇的地方是,當我看一個人的臉來分辨這個人的情緒,會看他的眼睛,眉毛是否上揚或是皺眉,或嘴巴上彎或露牙齒是笑,嘴巴若是下彎可能是在生氣,而這些我們用來判斷的依據,機器都有學到重點。

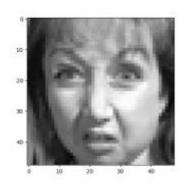


[Bonus] (1%) 從 training data 中移除部份 label, 實做 semisupervised learning

把 testing data 丟進 problem 1 訓練好的 model,根據 predict 出來的結果當做他們的新 label,接著再把這些新的 data 放進 training data 拿進去 train,得到的新 model 在 kaggle 拿到了 0.56395,而原本的 model 在 kaggle 上的分數是 0.57286,會變差的原因推測因為原本的 model 準確率並不高,所以可能會餵一 堆錯誤的 data 讓結果變差。

[Bonus] (1%) 在 Problem 5 中,提供了 3 個 hint,可以嘗試實作及觀察 (但也可以不限於 hint 所提到的方向,也可以自己去研究更多關於 CNN 細節的資料),並說明你做了些什麼? [完成 1 個: +0.4%,完成 2 個: +0.7%,完成 3 個: +1%]

我另外找了一張圖片如下,這張圖片屬於"Disgust",觀察他的第一層和第二層看到的影像,可以看出第一層還能看出臉的輪廓,但第二層就看不出什麼臉或眼睛的輪廓,只看到一塊一塊黑色區塊,所以可能容易被 predict 成另一種 class,從這可以推測出 Disgust 的準確率可能不會太高,problem3 也可以輔助這個論點。



第一層 filter:

	i 4_4	1.7	Output of layer0 (Ghen image546)		
					7.44
		<u> </u>	FF.	* · · · ·	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
			( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )	£	75
	224 	(3)		(AC)	
		35	8	F	(22)
			7	(a.)	7. 8
		7			
	2,		23	13	
	95	- 1	* <u>1</u> 1	2	100
	W-4		-74	0 mg	100 kg
第二層 filter:	W				
		1 9	Output of layer0 (Gityen image546)	-1.	- ;
	3 12			200	4.
		13	[ · ; ]	7.5	
	\$ 42 #	****** ********		100	33
	C.E.	6.5	1		27.5
	14	-	<u> </u>		
	. ,	3			ia'.
			E3	4.1	
				or the	73

## hint1:

我另外 train 了一個很差的 model,同樣拿 problem5 的圖片,觀察第一層 filter 看到的影像,但幾乎看不出臉的輪廓,效果非常差。

