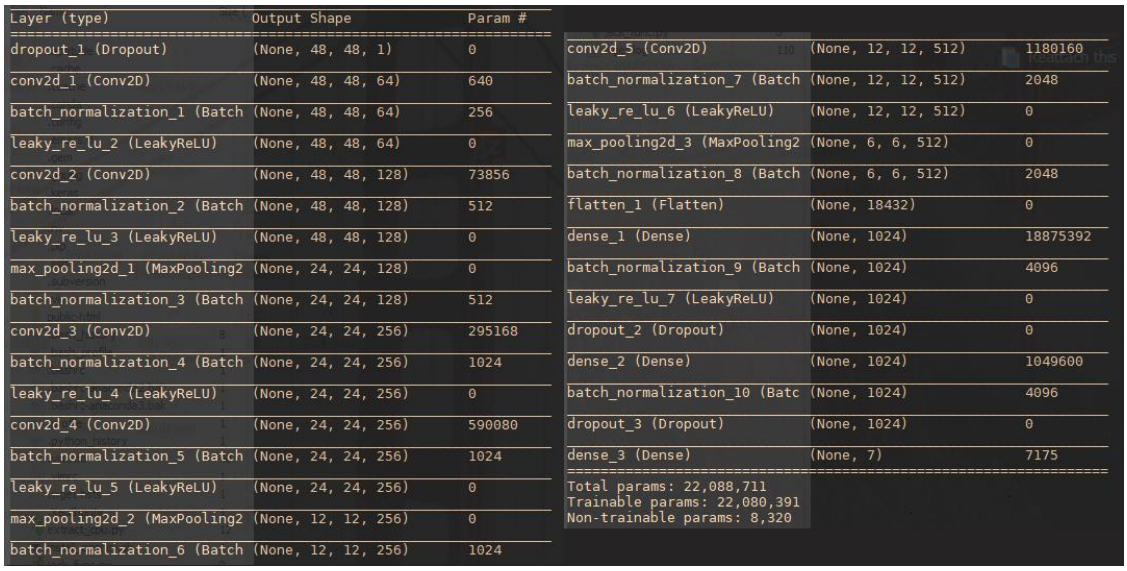
Homework 3 Report

學號：B05505004 系級：工海三 姓名:朱信靂

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？



Layer (type)	Output Shape	Param #
dropout_1 (Dropout)	(None, 48, 48, 1)	0
conv2d_1 (Conv2D)	(None, 48, 48, 64)	640
batch_normalization_1 (Batch	(None, 48, 48, 64)	256
leaky_re_lu_2 (LeakyReLU)	(None, 48, 48, 64)	0
conv2d_2 (Conv2D)	(None, 48, 48, 128)	73856
batch_normalization_2 (Batch	(None, 48, 48, 128)	512
leaky_re_lu_3 (LeakyReLU)	(None, 48, 48, 128)	0
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 128)	0
batch_normalization_3 (Batch	(None, 24, 24, 128)	512
conv2d_3 (Conv2D)	(None, 24, 24, 256)	295168
batch_normalization_4 (Batch	(None, 24, 24, 256)	1024
leaky_re_lu_4 (LeakyReLU)	(None, 24, 24, 256)	0
conv2d_4 (Conv2D)	(None, 24, 24, 256)	590080
batch_normalization_5 (Batch	(None, 24, 24, 256)	1024
leaky_re_lu_5 (LeakyReLU)	(None, 24, 24, 256)	0
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 256)	0
batch_normalization_6 (Batch	(None, 12, 12, 256)	1024
conv2d_5 (Conv2D)	(None, 12, 12, 512)	1180160
batch_normalization_7 (Batch	(None, 12, 12, 512)	2048
leaky_re_lu_6 (LeakyReLU)	(None, 12, 12, 512)	0
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 512)	0
batch_normalization_8 (Batch	(None, 6, 6, 512)	2048
flatten_1 (Flatten)	(None, 18432)	0
dense_1 (Dense)	(None, 1024)	18875392
batch_normalization_9 (Batch	(None, 1024)	4096
leaky_re_lu_7 (LeakyReLU)	(None, 1024)	0
dropout_2 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 1024)	1049600
batch_normalization_10 (Batc	(None, 1024)	4096
dropout_3 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 7)	7175

Total params: 22,088,711
Trainable params: 22,080,391
Non-trainable params: 8,320

模型架構：

我的 model 是使用 5 層的 convolution layer，其中每一層都包含 batch normalization，並且使用 leaky relu 當作我的 activation function。另外我每經過兩層的 convolution layer 才會再加入一層 max pooling layer，總共加入了三次。最後會產生維度 $6 \times 6 \times 512$的 feature map。

抽完 feature 後會把 feature map 拉平過後再經過兩層包含 1024 個 neuron 的 fully connected layer，並且一樣都加入 batch normalization，並且使用 leaky relu 當作我的 activation function，比較不一樣的是這兩層我還有加入 dropout 避免 overfit 的情況發生。

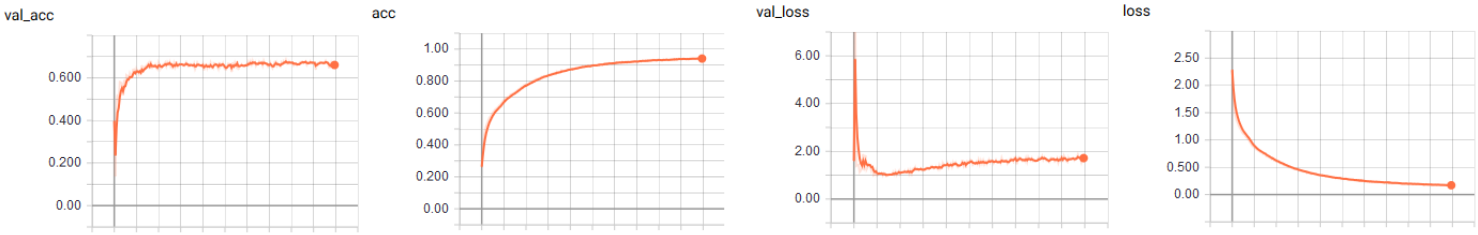
訓練過程：

在訓練的一開始我還有使用 keras 內建的 ImageDataGenerator，來做 data augmentation 的操作，我自己是使用了平移、旋轉、反轉這幾個功能。這個步驟增加了 training data 的多樣性，使的訓練出來的 model 比較不會 overfit 在 training data 上。也使的 model 上升了約 1%的準確率。

在這之前我都有使用一成的 training data 作為 validation 使用，然而為了要增加 model 的準確性，並且不要浪費這一成的 training data，於是我將

training data 切成了十等分，每次都用不同的一份當作 validation。如此最後會得到 10 個基於不同 training data train 出來的 model，去掉其中表現最不好的一個，最後可以達到高達七成的準確率。

下方的圖則是我訓練單一 model 的訓練過程，左邊是 validation 的圖形，右邊是 training 的圖形，一共訓練 400 epoch，並且會取出訓練過程中 val_acc 最高時的 model 最為最後使用的 model。



準確率：

在做 bagging 前，單一的 model 的準確率大約為 public score:0.64725 private score: 0.66536，然而當照上面的方法選擇了 9 個 model bagging 後，準確率上升至 public score:0.71579 private score: 0.72415。

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？

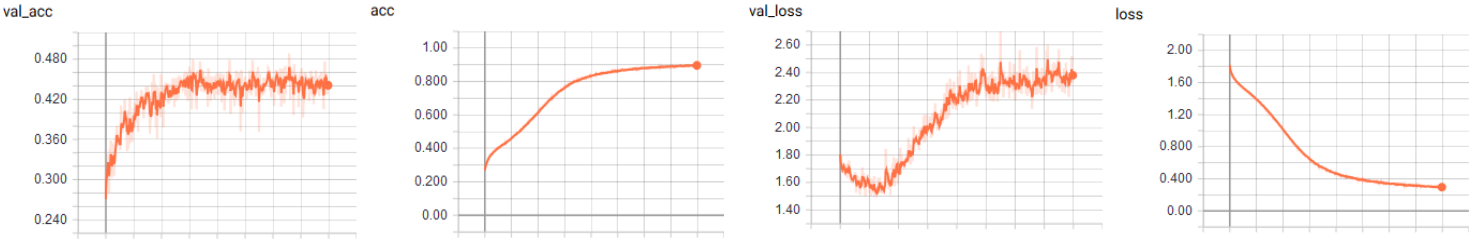
模型架構：

為了使 DNN 的參數量和 CNN 差不多，於是我使用了 6 層的 dense layer，neuron 數分別是 1024、2048、4096、2048、512、7。為了模擬跟 CNN model 類似，我一樣在每一層 dense layer 都加入 batch normalization，並且使用 leaky relu 當作我的 activation function。而整個 model 的架構如上圖所示，同樣都有約 22M 個參數。

訓練過程：

訓練方面同樣為了使 CNN 和 DNN 的 model 比較有意義，我同樣在 dnn 有使用 data augmentation 增加 training data 的多樣性。

跟上面 CNN 的訓練過程比較的話，可以看到不管是 dnn 的 model loss 不管在 validation 還是 training 上，相對於 CNN 的圖形都有極高的震動幅度，極不穩定。甚至訓練到後期產生嚴重的 overfit，相對於 CNN 的 model val_loss 只有略為上升。而準確率的部分可以看到不管是在 training 還是 validation，他得到的結果都相對 CNN 來的差，同時 CNN 大概會在 50 epoch 時開始收斂，可是 dnn 卻直到約 100 epoch 時才開始有收斂的傾向。

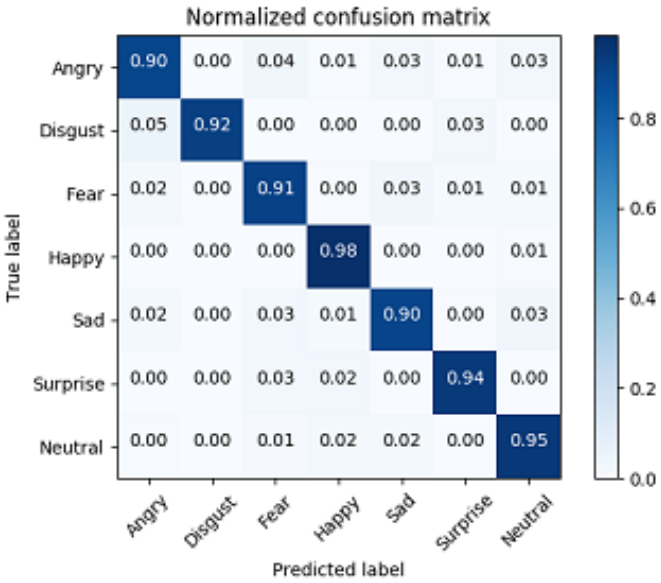


準確率：

準確率的部分 DNN 在 kaggle 僅拿到 public score:0.45750 private score: 0.486762 的成績，其相對於 CNN 掉了快 20%的準確率。

3 (1%)

- 觀察答錯的圖片中，哪些 class 彼此間容易用混？ 並說明你觀察到了什麼？ [繪出 confusion matrix 分析]



右圖的數據是我用單一 model 預測 validation set 所畫出的 confusion matrix，可以看到這個我的 model 在 Angry、Sad、Fear 有較差的表現。其中又可以發現 Angry 的圖片又很常會被誤判為 Fear。而在這七類分類中 model 在 Happy 這類的分類表現得最好，有高達 0.98 的準確率，領先第二高的 Neural 約 4%的準確率。然而其實回到 training data 來看的話會發現這七類的 training data 其實蠻不平均的，像是 Happy 的照片數是其他類別將近兩倍，這也難怪這個 model 在預測 Happy 可以有那麼好的 performance。

手寫

4.a Layer A: $(2\times2\times5+1)\times6=126$

Layer B: $(2\times2\times6+1)\times4=100$

4.b Layer A:

$\frac{(8-2+2\times0)}{3}+1=3$

乘法: $(2\times2\times5)\times3^2\times6=1080$

加法 $(2\times2\times5-1)\times3^2\times6=1026$ ※

Layer B:

乘法: $(2\times2\times6)\times1^2\times4=96$

加法: $(2\times2\times6-1)\times1^2\times4=92$ ※

4.c the output size of layer $\bar{L}$: $\left(\left(\frac{n_{\bar{L}}-k_{\bar{L}}+2P_{\bar{L}}}{S_{\bar{L}}}+1\right),\left(\frac{n_{\bar{L}}-k_{\bar{L}}+2P_{\bar{L}}}{S_{\bar{L}}}+1\right),C_{\bar{L}}\right)$

$\Rightarrow$ time complexity: $O\left(\sum_{\bar{L}=1}^{l}\left[k_{\bar{L}}^2\times C_{\bar{L}-1}\times\left(\frac{n_{\bar{L}}-k_{\bar{L}}+2P_{\bar{L}}}{S_{\bar{L}}}+1\right)^2\times C_{\bar{L}}\right]\right)$, C_0 is input channel number ※

5. $X=\begin{bmatrix}1&3&3\\4&8&5\\3&13&9\\1&8&5\\5&14&2\\7&4&1\\9&8&9\\3&8&1\\11&5&6\\10&11&7\end{bmatrix}$ mean=[5.4 8 4.8] std=[3.6596 3.6818 3.0111] scaling $\Rightarrow$ $\begin{bmatrix}-1.20299&-1.63964&-0.59999\\-0.38397&0&0.06642\\-0.65619&1.08643&1.39484\\-1.20299&0&0.06642\\-0.10936&1.63964&-0.92990\\0.43745&-1.08643&-1.26200\\0.98434&0&1.39484\\-0.65617&0&-1.26200\\1.53107&-0.81482&0.39853\\1.25917&0.81482&0.73063\end{bmatrix}=Y$ ※

each row vector is a sample.

set $S=\text{conv}(X)=\frac{1}{10-1}Y^TY=\begin{bmatrix}13.3778&0.55556&3.64444\\0.55556&13.55556&3.22222\\3.64444&3.22222&9.06667\end{bmatrix}$

$\det(S-\lambda I)=0\Rightarrow\begin{vmatrix}13.3778-\lambda&0.55556&3.64444\\0.55556&13.55556-\lambda&3.22222\\3.64444&3.22222&9.06667-\lambda\end{vmatrix}=0$

eigenvalue: 6.08, 12.9228, 16.9972

eigenvector: $\begin{bmatrix}-0.399855\\-0.337589\\0.852144\end{bmatrix}$, $\begin{bmatrix}-0.678179\\0.734390\\-0.029286\end{bmatrix}$, $\begin{bmatrix}-0.616595\\-0.588816\\-0.522596\end{bmatrix}$

(a) the corresponding eigenvector are principle axis.

(b) sort eigenvector with eigenvalue.

$P=\begin{bmatrix}-0.616595&-0.678179&-0.399855\\-0.588816&0.734390&-0.337589\\-0.522596&-0.029286&0.852144\end{bmatrix}$, $XP=\begin{bmatrix}-3.362015&0.708743&1.481399\\-9.789888&3.025974&-0.039412\\-13.618941&6.532569&2.418663\\-7.94013&5.060511&1.160153\\-12.371591&6.835993&-5.021233\\-7.194025&-1.836979&-3.299197\\-14.963247&-0.474065&1.369889\\-7.082909&3.813297&-3.048133\\-12.862201&-3.951735&-0.973486\\-16.301098&1.105498&-1.747021\end{bmatrix}$ ※

(c) $\hat{X}^{(PCA)}=\begin{bmatrix}-3.362015&0.708743&0\\-9.789888&3.025974&0\\-13.618941&6.532569&0\\-7.94013&5.060511&0\\-12.371591&6.835993&0\\-7.194025&-1.836979&0\\-14.963247&-0.474065&0\\-7.082909&-3.813297&0\\-12.862201&-3.951735&0\\-16.301098&1.105498&0\end{bmatrix}$

the reconstruction error:

$E=\|X-\hat{X}^{(PCA)}\|^2=(1.481399+0.039412^2+2.418663^2+1.160153^2+5.021233^2+3.299197^2+1.369889^2+3.048133^2+0.973486^2+1.747021^2)$

$=60.64$ ※