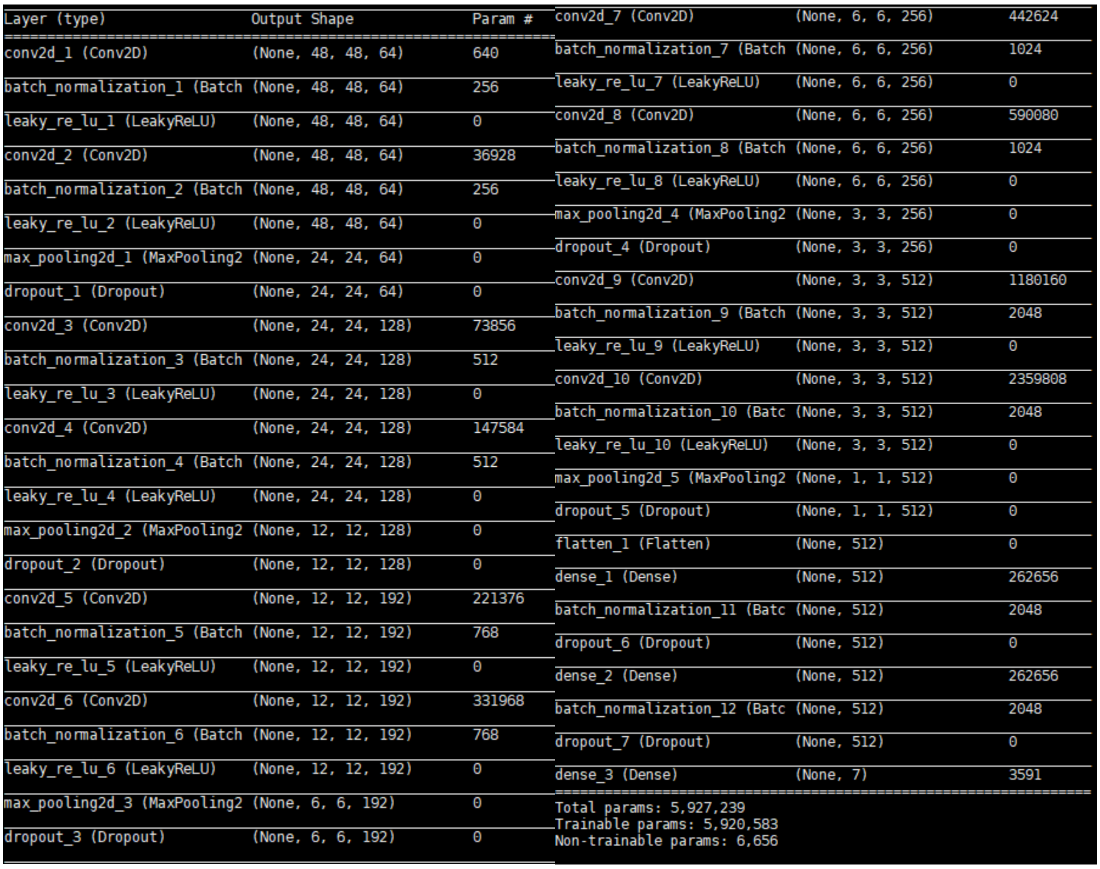
Homework 3 Report Problem Set

Professor Pei-Yuan Wu
EE5184 - Machine Learning

電子所碩一 R07943004 莊育權

Problem 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 #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 64)        640
batch_normalization_1 (Batch (None, 48, 48, 64)        256
leaky_re_lu_1 (LeakyReLU)    (None, 48, 48, 64)        0
conv2d_2 (Conv2D)            (None, 48, 48, 64)        36928
batch_normalization_2 (Batch (None, 48, 48, 64)        256
leaky_re_lu_2 (LeakyReLU)    (None, 48, 48, 64)        0
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 64)        0
dropout_1 (Dropout)          (None, 24, 24, 64)        0
conv2d_3 (Conv2D)            (None, 24, 24, 128)       73856
batch_normalization_3 (Batch (None, 24, 24, 128)       512
leaky_re_lu_3 (LeakyReLU)    (None, 24, 24, 128)       0
conv2d_4 (Conv2D)            (None, 24, 24, 128)       147584
batch_normalization_4 (Batch (None, 24, 24, 128)       512
leaky_re_lu_4 (LeakyReLU)    (None, 24, 24, 128)       0
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 128)       0
dropout_2 (Dropout)          (None, 12, 12, 128)       0
conv2d_5 (Conv2D)            (None, 12, 12, 192)       221376
batch_normalization_5 (Batch (None, 12, 12, 192)       768
leaky_re_lu_5 (LeakyReLU)    (None, 12, 12, 192)       0
conv2d_6 (Conv2D)            (None, 12, 12, 192)       331968
batch_normalization_6 (Batch (None, 12, 12, 192)       768
leaky_re_lu_6 (LeakyReLU)    (None, 12, 12, 192)       0
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 192)         0
dropout_3 (Dropout)          (None, 6, 6, 192)         0
conv2d_7 (Conv2D)            (None, 6, 6, 256)         442624
batch_normalization_7 (Batch (None, 6, 6, 256)         1024
leaky_re_lu_7 (LeakyReLU)    (None, 6, 6, 256)         0
conv2d_8 (Conv2D)            (None, 6, 6, 256)         590080
batch_normalization_8 (Batch (None, 6, 6, 256)         1024
leaky_re_lu_8 (LeakyReLU)    (None, 6, 6, 256)         0
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 256)         0
dropout_4 (Dropout)          (None, 3, 3, 256)         0
conv2d_9 (Conv2D)            (None, 3, 3, 512)         1180160
batch_normalization_9 (Batch (None, 3, 3, 512)         2048
leaky_re_lu_9 (LeakyReLU)    (None, 3, 3, 512)         0
conv2d_10 (Conv2D)           (None, 3, 3, 512)         2359808
batch_normalization_10 (Batc (None, 3, 3, 512)         2048
leaky_re_lu_10 (LeakyReLU)   (None, 3, 3, 512)         0
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 512)         0
dropout_5 (Dropout)          (None, 1, 1, 512)         0
flatten_1 (Flatten)          (None, 512)               0
dense_1 (Dense)              (None, 512)               262656
batch_normalization_11 (Batc (None, 512)               2048
dropout_6 (Dropout)          (None, 512)               0
dense_2 (Dense)              (None, 512)               262656
batch_normalization_12 (Batc (None, 512)               2048
dropout_7 (Dropout)          (None, 512)               0
dense_3 (Dense)              (None, 7)                 3591
=================================================================
Total params: 5,927,239
Trainable params: 5,920,583
Non-trainable params: 6,656
```

模型架構:

前面 Convolution 2D 每層架構為 conv2D (3,3) -> batch normalization -> activation (‘leaky relu’, alpha=0.1) -> conv2D (3,3) -> batch normalization -> activation (‘leaky relu’, alpha=0.1) -> max pooling2D (2,2) -> drop out (0.3)，總共有 5 層，對應的 filter 數為 64、128、192，256 和 512。

後面 Dense 架構為 dense (512, ‘relu’) -> batch normalization -> drop out (0.5) -> dense (512, ‘relu’) -> batch normalization -> drop out (0.5) -> dense (7, ‘softmax’)

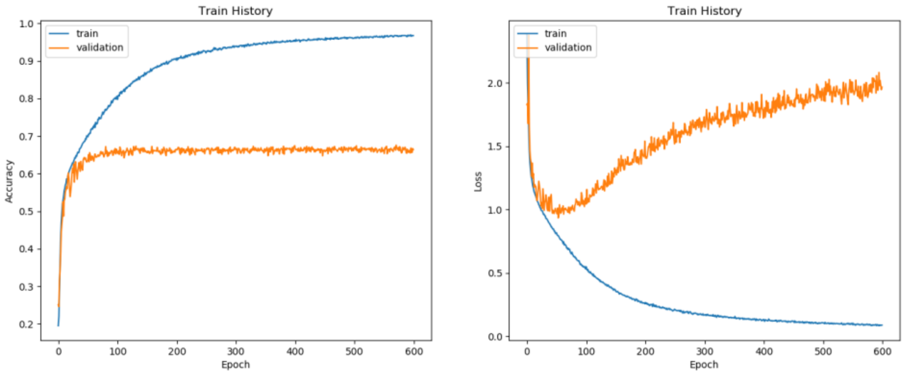
整體模型的 loss 是由 categorical crossentropy 決定，optimizer 是 adam。

訓練過程:

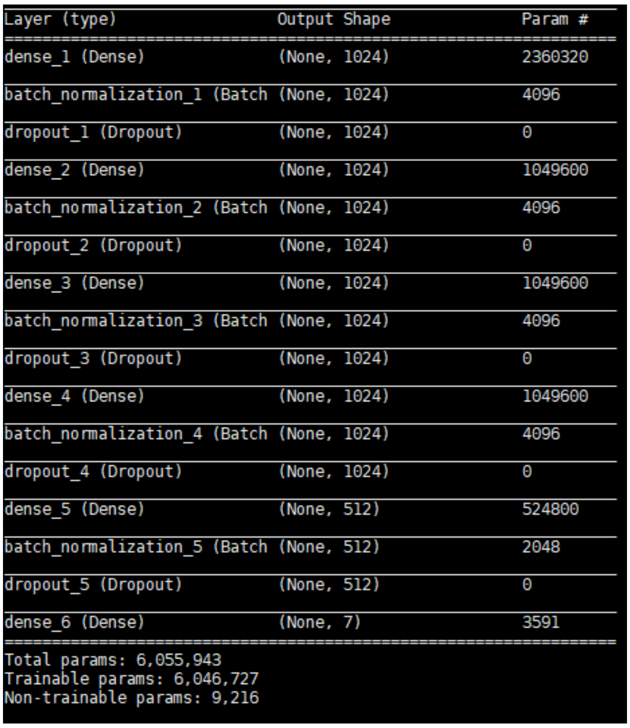
在資料前處理的部分，將 5000 筆訓練資料當作 validation data，而剩餘資料當作 training data。訓練過程中使用 Image Generator 把資料做旋轉(10 度以內)和水平上下移(10%以內)，用來增加訓練資料量。並在訓練的過程中使用 callbacks 函數中的 ModelCheckpoints，根據每個 epoch 訓練出來的 validation accuracy 將最好的 model 存取下來。

準確率:

此模型訓練 600 個 epoch 後，訓練中最好的 validation accuracy 為 0.6840，而 kaggle public score 為 0.68459，從下圖可以看到 validation accuracy 到最後會趨於飽和，但是 loss 會上升，有 overfitting 的現象。



Problem 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？



```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              2360320
_________________________________________________________________
batch_normalization_1 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
batch_normalization_2 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              1049600
_________________________________________________________________
batch_normalization_3 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              1049600
_________________________________________________________________
batch_normalization_4 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               524800
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 7)                 3591
=================================================================
Total params: 6,055,943
Trainable params: 6,046,727
Non-trainable params: 9,216
```

模型架構:

DNN 架構為 Dense (1024, ‘relu’) -> batch normalization -> drop out (0.5) -> Dense (1024, ‘relu’) -> batch normalization -> drop out (0.5) -> Dense (1024, ‘relu’) -> batch normalization -> drop out (0.5) -> Dense (1024, ‘relu’) -> batch normalization -> drop out (0.5) -> Dense (512, ‘relu’) -> batch normalization -> drop out (0.5) -> Dense (7, ‘softmax’)

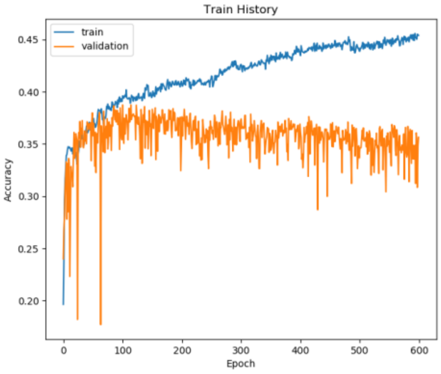
整體模型的 loss 是由 categorical crossentropy 決定，optimizer 是 adam。

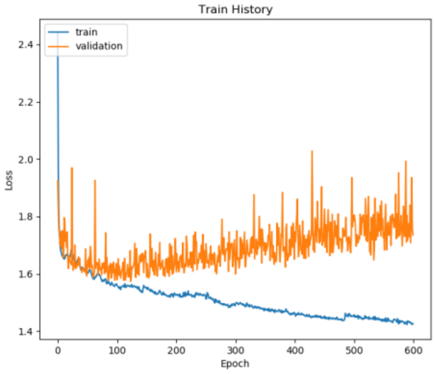
訓練過程:

在資料前處理的部分，將 5000 筆訓練資料當作 validation data，而剩餘資料當作 training data。並在訓練的過程中使用 callbacks 函數中的 ModelCheckpoints，根據每個 epoch 訓練出來的 validation accuracy 將最好的 model 存取下來。

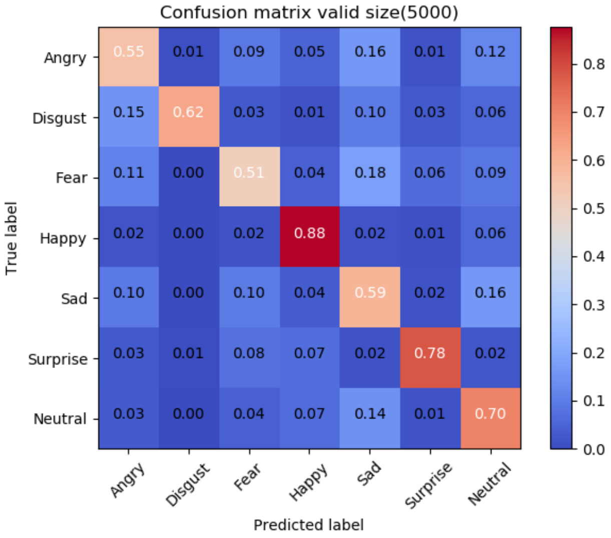
準確率:

此模型訓練 600 個 epoch 後，訓練中最好的 validation accuracy 為 0.3874，下圖顯示 validation accuracy 和 error 震盪都蠻嚴重的。可以知道在差不多參數量的情況下，DNN 的預測能力非常差，我覺得是因為 DNN 相對 CNN 而言並沒有將圖片分區塊辨識的能力，所以當圖片歧異度很高時，純粹用數值的訓練不容易成功。





Problem 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]



	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
Angry	0.55	0.01	0.09	0.05	0.16	0.01	0.12
Disgust	0.15	0.62	0.03	0.01	0.10	0.03	0.06
Fear	0.11	0.00	0.51	0.04	0.18	0.06	0.09
Happy	0.02	0.00	0.02	0.88	0.02	0.01	0.06
Sad	0.10	0.00	0.10	0.04	0.59	0.02	0.16
Surprise	0.03	0.01	0.08	0.07	0.02	0.78	0.02
Neutral	0.03	0.00	0.04	0.07	0.14	0.01	0.70

如同訓練 CNN 時的假設，將前 5000 筆資料當作 validation set，所以這邊是用 validation set 的 data 去做 confusion matrix。

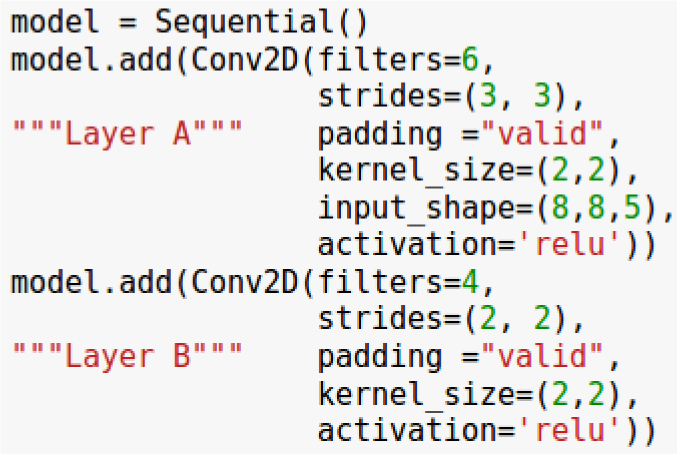
根據 confusion matrix 可以看出 angry 中最容易被錯誤判斷成 sad，disgust 中最容易被錯誤判斷成 angry，fear 中最容易被錯誤判斷成 sad，sad 中最容易被錯誤判斷成 angry，從這四個觀察點可以

發現，我們人類對於這四個情緒的分類比較屬於「負面情緒」，因此可能 CNN 在學習的時候會比較容易將這四種情緒搞混。這種現象也可以在「正面情緒」happy 和 surprise 中觀察到。

同時根據 confusion matrix 可以看出 happy 和 surprise 的準確率最高，推測可能是因為正面情緒的表情較為誇張，例如微笑的嘴角上揚等特徵，且正面情緒選擇較少，使得 CNN 比較容易能分辨這兩個情緒。而負面情緒較低，可能是因為負面情緒除了選擇多，加上表情小還可能跟 neutral 搞混的原因。

Problem 4. (1.5%, each 0.5%) CNN time/space complexity

For a. b. Given a CNN model as



```
model = Sequential()
model.add(Conv2D(filters=6,
                 strides=(3, 3),
"""Layer A"""    padding ="valid",
                 kernel_size=(2,2),
                 input_shape=(8,8,5),
                 activation='relu'))
model.add(Conv2D(filters=4,
                 strides=(2, 2),
"""Layer B"""    padding ="valid",
                 kernel_size=(2,2),
                 activation='relu'))
```

And for the c. given the parameter as:

kernel size = (k,k);

channel size = c;

filter size = f;

input shape = (n,n);

padding = 1;

strides = (s,s);

a. How many parameters are there in each layer(Hint: you may consider whether the number of parameter is related with)

Parameter: $(\text{kernel size} \times \text{kernel size} \times \text{number of channels} + 1) \times \text{number of filters}$

Layer A: $(2 \times 2 \times 5 + 1) \times 6 = 126$

Layer B: $(2 \times 2 \times 6 + 1) \times 4 = 100$

b. How many multiplications/additions are needed for a forward pass (each layer).

Multiplications:

$$\text{kernel size} \times \text{kernel size} \times \text{number of channels} \times \text{output size} \times \text{number of filters}$$

Additions:

kernel size × kernel size × number of channels × output size × number of filters

Layer A:

Multiplications: $2 \times 2 \times 5 \times 9 \times 6 = 1080$

Additions: $2 \times 2 \times 5 \times 9 \times 6 = 1080$

Layer B:

Multiplications: $2 \times 2 \times 6 \times 1 \times 4 = 96$

Additions: $2 \times 2 \times 6 \times 1 \times 4 = 96$

c. **What is the time complexity of convolutional neural networks?(note: you must use big-O upper bound, and there are l layer, you can use C_l,C_{l-1}as lth and l-1th layer)**

Time Complexity of CNN

$$= O\left(\sum_{i=1}^{total\ layers} (number\ of\ input\ channel) \times (length\ of\ filter)^2 \times (number\ of\ filters) \times (length\ of\ output\ feature\ map)^2\right)$$

$$= O\left(\sum_{i=1}^{l} (f_{l-1}) \times (k)^2 \times (f_l) \times \left(\left\lfloor \frac{n-k}{s} \right\rfloor\right)^2\right)$$

Ref: https://arxiv.org/pdf/1412.1710.pdf

Problem 5. (1.5%, each 0.5%) PCA practice: Given 10 samples in 3D space

(1,2,3),(4,8,5),(3,12,9),(1,8,5),(5,14,2), (7,4,1),(9,8,9),(3,8,1),(11,5,6),(10,11,7)

(1) What are the principle axes?

a. Compute the covariance matrix

$$C = \frac{1}{N} X X^T = \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{x_i} - \boldsymbol{\mu})(\boldsymbol{x_i} - \boldsymbol{\mu})^T = U \Lambda U^T$$

$$X = \begin{bmatrix} 1 & 4 & 3 & 1 & 5 & 7 & 9 & 3 & 11 & 10 \\ 2 & 8 & 12 & 8 & 14 & 4 & 8 & 8 & 5 & 11 \\ 3 & 5 & 9 & 5 & 2 & 1 & 9 & 1 & 6 & 7 \end{bmatrix} = [\boldsymbol{x_1}\ \boldsymbol{x_2}\ \boldsymbol{x_3}\ \boldsymbol{x_4}\ \boldsymbol{x_5}\ \boldsymbol{x_6}\ \boldsymbol{x_7}\ \boldsymbol{x_8}\ \boldsymbol{x_9}\ \boldsymbol{x_{10}}]$$

b. 用 numpy.linalg.eig 去對 covariance matrix 做 eigenvalue decomposition

$$C \approx \begin{bmatrix} 0.39985541 & -0.67817891 & -0.6165947 \\ 0.33758926 & 0.73439013 & -0.58881629 \\ -0.85214385 & -0.02728563 & -0.52259579 \end{bmatrix} \begin{bmatrix} 5.47203291 & 0 & 0 \\ 0 & 11.63052369 & 0 \\ 0 & 0 & 15.2974434 \end{bmatrix}$$

$$\begin{bmatrix} 0.39985541 & -0.67817891 & -0.6165947 \\ 0.33758926 & 0.73439013 & -0.58881629 \\ -0.85214385 & -0.02728563 & -0.52259579 \end{bmatrix}^T = U\Lambda U^T$$

c. 總共有三條 principal axes，依照 eigenvalue 大小順序分別為(大到小)

$$\boldsymbol{u_1} = [-0.6165947 \quad -0.58881629 \quad -0.52259579]$$

$$\boldsymbol{u_2} = [-0.67817891 \quad 0.73439013 \quad -0.02728563]$$

$$\boldsymbol{u_3} = [0.39985541 \quad 0.33758926 \quad -0.85214385]$$

(2) Compute the principal components for each sample.

$\mathbf{y} = \boldsymbol{U^T x} = [\boldsymbol{u_1}\ \boldsymbol{u_2}\ \boldsymbol{u_3}\ ]^T[\boldsymbol{x_1}\ \boldsymbol{x_2}\ \boldsymbol{x_3}\ \boldsymbol{x_4}\ \boldsymbol{x_5}\ \boldsymbol{x_6}\ \boldsymbol{x_7}\ \boldsymbol{x_8}\ \boldsymbol{x_9}\ \boldsymbol{x_{10}}]$

$$\mathbf{y} = \begin{bmatrix} -0.6165947 & -0.67817891 & 0.39985541 \\ -0.58881629 & 0.73439013 & 0.33758926 \\ -0.52259579 & -0.02728563 & -0.85214385 \end{bmatrix}^T \begin{bmatrix} 1 & 4 & 3 & 1 & 5 & 7 & 9 & 3 & 11 & 10 \\ 2 & 8 & 12 & 8 & 14 & 4 & 8 & 8 & 5 & 11 \\ 3 & 5 & 9 & 5 & 2 & 1 & 9 & 1 & 6 & 7 \end{bmatrix}$$

$$\approx \begin{bmatrix} -3.362 & -9.79 & -13.62 & -7.940 & -12.37 & -7.194 & -14.96 & -7.083 & -12.86 & -16.30 \\ 0.709 & 3.026 & 6.533 & 5.062 & 6.836 & -1.837 & -0.474 & 3.813 & -3.952 & 1.106 \\ -1.481 & 0.039 & -2.419 & -1.160 & 5.021 & 3.297 & -1.370 & 3.048 & 0.973 & 1.747 \end{bmatrix}$$

$= [\boldsymbol{y_1}\ \boldsymbol{y_2}\ \boldsymbol{y_3}\ \boldsymbol{y_4}\ \boldsymbol{y_5}\ \boldsymbol{y_6}\ \boldsymbol{y_7}\ \boldsymbol{y_8}\ \boldsymbol{y_9}\ \boldsymbol{y_{10}}]$

$[\boldsymbol{y_1}\ \boldsymbol{y_2}\ \boldsymbol{y_3}\ \boldsymbol{y_4}\ \boldsymbol{y_5}\ \boldsymbol{y_6}\ \boldsymbol{y_7}\ \boldsymbol{y_8}\ \boldsymbol{y_9}\ \boldsymbol{y_{10}}]$ 分別為 $[\boldsymbol{x_1}\ \boldsymbol{x_2}\ \boldsymbol{x_3}\ \boldsymbol{x_4}\ \boldsymbol{x_5}\ \boldsymbol{x_6}\ \boldsymbol{x_7}\ \boldsymbol{x_8}\ \boldsymbol{x_9}\ \boldsymbol{x_{10}}]$ 對應$[\boldsymbol{u_1}\ \boldsymbol{u_2}\ \boldsymbol{u_3}\ ]$的 principal components

(3) Reconstruction error if reduced to 2D. (Calculate the L2-norm)

另$\boldsymbol{\tilde{x}}$為從 2D 到 3D 的 reconstruction samples

$$\boldsymbol{\tilde{x}} = \mathbf{U}[:,:\mathbf{2}]\boldsymbol{y}[:2,:]$$

Reconstruction error

$$= \sum_{i=1}^{10} (\boldsymbol{x_i} - \boldsymbol{\tilde{x}_i})^2 \approx 60.644$$