Final Project

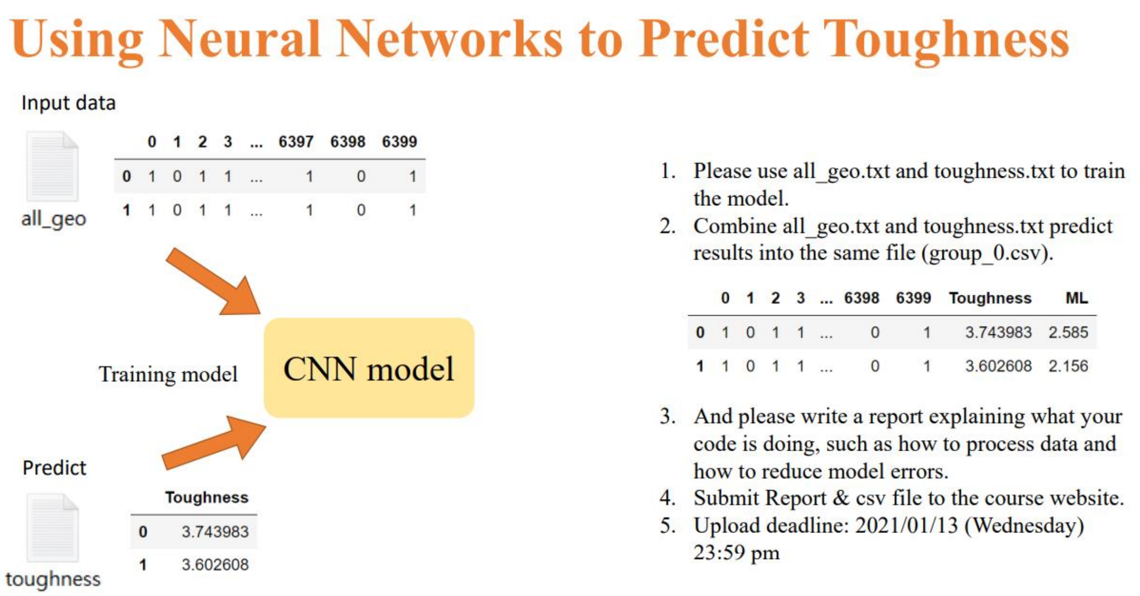
Using Neural Networks to Predict Toughness

Group_3

N96094196 張維峻 N96094023 林宣伯

N96094073 廖烽凱 N96091091 趙宥齊 N96094031 吳俊達

0.摘要



1.資料前處理

1.把 256*256 的彩色圖片轉成 80*80 的灰階圖片

```python
#圖片512*512轉成80*80
import cv2
import matplotlib.pyplot as plt

folder = 'group_3/'
for i in range(0,10000,1):
    filename = str(i)+'.png'

    img = cv2.imread(folder+filename, cv2.IMREAD_GRAYSCALE)
    print(type(img))
    print('shape =' ,img.shape)
    img_re = cv2.resize(img,(80,80), interpolation=cv2.INTER_CUBIC)
    print('re shape = ', img_re.shape)
    cv2.imwrite('./group_3re/'+str(i)+'.png', img_re)
    print('done')
```

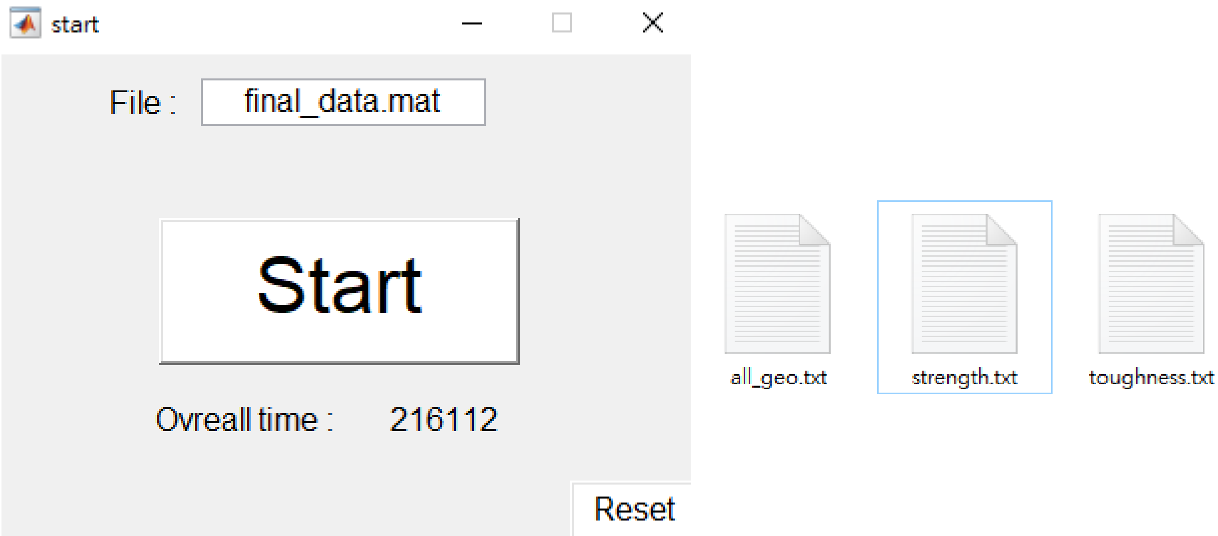
2. 讀取灰階圖片 將 pixel 換成 0/1 ,接所有結果加入

final_data[],輸出成 final_data.mat

```
#  讀取灰階圖片  將pixel換成0/1  ,接所有結果加入final_data[],輸出成  final_data.mat
import  cv2
import  matplotlib.pyplot  as  plt
from  scipy.io  import  savemat
from  sklearn.preprocessing  import  Binarizer
from  sklearn  import  preprocessing
import  numpy  as  np
folder  =  'group_3re/'
encoder  =  Binarizer()
final_data  =  np.array([])
size=2


for  i  in  range(0,size,1):
    filename  =  str(i)+'.png'
    img  =  cv2.imread(folder+filename,  cv2.IMREAD_GRAYSCALE)
    np.savetxt('img_0.txt',  img)
    #  先資料做標準化StandardScaler
    scaler  =  preprocessing.StandardScaler().fit(img)
    X_scaled  =  scaler.transform(img)
    np.savetxt('x_sc.txt',X_scaled)
    #  再做二元化Binarizer
    encoder.fit(X_scaled)
    img  =  encoder.transform(X_scaled)

    np.savetxt('img_re.txt',  img,  fmt='%i')
    final_data  =  np.append(final_data,img)

    print(i,'done')


print('fin=',final_data)
print(final_data.shape)
final_data  =  final_data.reshape(size,6400)
print('re_fin=  ',final_data)
print(final_data.shape)
savemat('final_data.mat',  {'random':final_data})
np.savetxt('final_data.txt',  final_data,  fmt='%i')
```

3. final_data.mat 資料 丟進 Matlab 進行計算,完成後會得

到 all_geo.txt，strength.txt，toughness.txt



4. 進入 model 前先將 all_geo.txt 和 toughness.txt

合起來變成一個 Fin_train.csv 檔方便使用。

```
from google.colab import drive
drive.mount('/content/gdrive')
import pandas as pd
import numpy as np
data = pd.read_table('/content/gdrive/My Drive/ML_Final/all_geo.txt',header=None,encoding='gb2312',sep=' ')
print(data.shape)
data_0=data.iloc[:,:-1]
print(data_0.shape)

toug = pd.read_table('/content/gdrive/My Drive/ML_Final/toughness.txt',header=None,encoding='gb2312')
traindata=np.column_stack((data_0,toug))
traindata_pd = pd.DataFrame(data=traindata)
traindata_pd.to_csv('Fin_train.csv',index=0)
traindata_pd=traindata_pd.rename(columns={traindata_pd.columns[6400]:'Toughness'},inplace=True)
```

```
raw_file = pd.read_csv('/content/gdrive/My Drive/ML_Final/Fin_train.csv')
print(raw_file.head(5))
```

```
   0  1  2  3  4  5  6  7  ...  6393  6394  6395  6396  6397  6398  6399  Toughness
0  1  1  1  1  0  0  0  0  ...     0     0     0     1     1     1     1   2.413495
1  1  1  1  0  0  0  0  1  ...     0     0     0     0     1     1     1   2.900767
2  1  1  1  0  0  0  0  0  ...     0     0     0     0     1     1     1   3.326615
3  1  1  1  1  1  0  1  1  ...     1     0     1     1     1     1     1   1.638111
4  0  0  0  1  1  1  1  0  ...     1     1     1     1     0     0     0   6.759200

[5 rows x 6401 columns]
```

2.建構模型

1.讀入資料切分成訓練集和驗證集

```
from sklearn.model_selection import train_test_split
train_file, val_file = train_test_split(raw_file, random_state=777, train_size=0.8)

train_data=train_file.iloc[:,:-1]
train_label=train_file.iloc[:,-1:]
val_data=val_file.iloc[:,:-1]
val_label=val_file.iloc[:,-1:]
```

2.將訓練集和測試集合轉換成要丟進模型的形狀

```
#訓練集 前處理
print('x_train_image: ',train_data.shape)
print('y_train_image: ',train_label.shape)
train_data_re = train_data.values.reshape(8000,80,80)
print('train data re',train_data_re.shape)
train_label_re = train_label.values.reshape(8000,)
print('train label re',train_label_re.shape)
print('train label re data type is',train_label_re[0].dtype)
print('train_data_re[0]',train_data_re[0].shape)
```

```
x_train_image: (8000, 6400)
y_train_image: (8000, 1)
train data re (8000, 80, 80)
train label re (8000,)
train label re data type is float64
train_data_re[0] (80, 80)
```

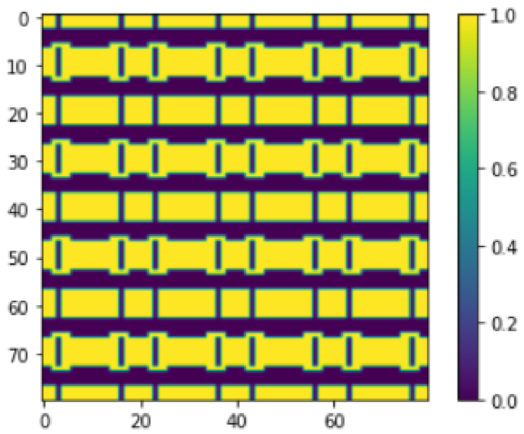
```
#驗證集 前處理
print('val data: ',val_data.shape)
print('val label: ',val_label.shape)

val_data_re = val_data.values.reshape(2000,80,80)
print('val_data_re: ',val_data_re.shape)

val_label_re = val_label.values.reshape(2000,)
print('val_label_re',val_label_re.shape)
```

```
val data: (2000, 6400)
val label: (2000, 1)
val_data_re: (2000, 80, 80)
val_label_re (2000,)
```

```
train_data4D = train_data_re.reshape(train_data_re.shape[0],80,80,1).astype('float32')
val_data4D = val_data_re.reshape(val_data_re.shape[0],80,80,1).astype('float32')
print('train_data4D',train_data4D.shape)
print('val_data4D',val_data4D.shape)
```

```
train_data4D (8000, 80, 80, 1)
val_data4D (2000, 80, 80, 1)
```

```
from matplotlib.pyplot import *
plt.figure()
plt.imshow(train_data_re[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



3.建立模型

```python
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D
```

```python
model = Sequential()
```

```python
model.add(Conv2D(
            filters=16,
            kernel_size=(5,5),
            padding='same',
            input_shape=(80,80,1),
            activation='relu'              ))
```

```python
model.add(MaxPooling2D(pool_size=(2,2)))
```

```python
model.add(Conv2D(filters=36,
             kernel_size=(5,5),
             padding = 'same',
             activation='relu'))
```

```python
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='relu'))
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['accuracy'])
```

```python
model.save('initial.h5')
```

```python
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param
=================================================================
conv2d (Conv2D)              (None, 80, 80, 16)        416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 40, 40, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 40, 40, 36)        14436
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 20, 36)        0
_________________________________________________________________
dropout (Dropout)            (None, 20, 20, 36)        0
_________________________________________________________________
flatten (Flatten)            (None, 14400)             0
_________________________________________________________________
dense (Dense)                (None, 128)               184332
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,858,309
Trainable params: 1,858,309
Non-trainable params: 0
_________________________________________________________________
None
```

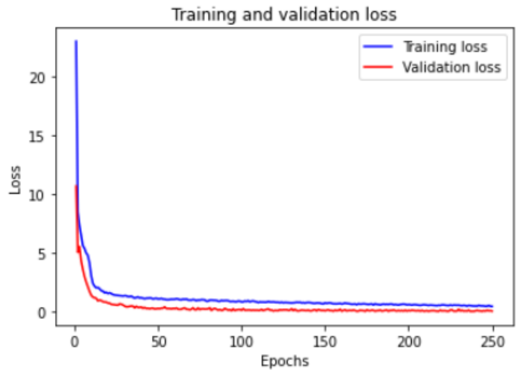
4.進行訓練 250epochs

```python
train_history=model.fit(x=train_data4D,
                        y=train_label_re,validation_split=0.2,
                        epochs=250,batch_size=500,verbose=2,
                        validation_data=(val_data4D, val_label_re))
```

```
Epoch 1/250
13/13 - 9s - loss: 22.9415 - accuracy: 0.0000e+00 - val_loss: 10.6694 - val_accuracy: 0.0000e+0
Epoch 2/250
13/13 - 1s - loss: 8.5026 - accuracy: 0.0000e+00 - val_loss: 5.0632 - val_accuracy: 0.0000e+00
Epoch 3/250
13/13 - 1s - loss: 7.3551 - accuracy: 0.0000e+00 - val_loss: 5.5618 - val_accuracy: 0.0000e+00
Epoch 4/250
13/13 - 1s - loss: 6.5116 - accuracy: 0.0000e+00 - val_loss: 4.3365 - val_accuracy: 0.0000e+00
Epoch 5/250
13/13 - 1s - loss: 5.6409 - accuracy: 0.0000e+00 - val_loss: 3.6371 - val_accuracy: 0.0000e+00

Epoch 245/250
13/13 - 1s - loss: 0.5273 - accuracy: 0.0000e+00 - val_loss: 0.1206 - val_accuracy: 0.0000e+00
Epoch 246/250
13/13 - 1s - loss: 0.4996 - accuracy: 0.0000e+00 - val_loss: 0.0880 - val_accuracy: 0.0000e+00
Epoch 247/250
13/13 - 1s - loss: 0.5006 - accuracy: 0.0000e+00 - val_loss: 0.1549 - val_accuracy: 0.0000e+00
Epoch 248/250
13/13 - 1s - loss: 0.5691 - accuracy: 0.0000e+00 - val_loss: 0.1079 - val_accuracy: 0.0000e+00
Epoch 249/250
13/13 - 1s - loss: 0.4968 - accuracy: 0.0000e+00 - val_loss: 0.1264 - val_accuracy: 0.0000e+00
Epoch 250/250
13/13 - 1s - loss: 0.4858 - accuracy: 0.0000e+00 - val_loss: 0.0872 - val_accuracy: 0.0000e+00
```

5. loss /epochs 圖表

```python
%matplotlib inline
acc = train_history.history['accuracy']
val_acc = train_history.history['val_accuracy']
loss = train_history.history['loss']
val_loss = train_history.history['val_loss']

epochs = range(1, len(acc) + 1)

# "b" is for "blue line"
plt.plot(epochs, loss, 'b', label='Training loss')
# r is for "solid red line"
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
# plt.xlim((0, 250))
# plt.ylim((0, 10000))
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



3.預測 Toughness

1.讀入目標輸出資料集，並轉成模型接受的形狀

```python
# 目標輸出
raw_data=raw_file.iloc[:,:-1]
raw_label=raw_file.iloc[:,-1:]
print('label:',raw_label.shape)
raw_label_re = raw_label.values.reshape(10000,)
print('label re:',raw_label_re.shape)
print(raw_data.head(5))
print(raw_data.shape)
raw_data_re = raw_data.values.reshape(10000,80,80)
print(raw_data_re.shape)
raw_data4D = raw_data_re.reshape(raw_data_re.shape[0],80,80,1).astype('float32')
print(raw_data4D.shape)
```

```
label: (10000, 1)
label re: (10000,)
   0  1  2  3  4  5  6  7  ...  6392  6393  6394  6395  6396  6397  6398  6399
0  1  1  1  1  0  0  0  0  ...     0     0     0     0     1     1     1     1
1  1  1  1  0  0  0  0  1  ...     1     0     0     0     0     1     1     1
2  1  1  1  0  0  0  0  0  ...     0     0     0     0     0     1     1     1
3  1  1  1  1  1  0  1  1  ...     1     1     0     1     1     1     1     1
4  0  0  0  1  1  1  1  0  ...     0     1     1     1     1     0     0     0

[5 rows x 6400 columns]
(10000, 6400)
(10000, 80, 80)
(10000, 80, 80, 1)
```

2.評估預測誤差

```python
# 評估預測誤差
def Mape(data_true, data_pred):
    return np.mean(np.abs((data_true - data_pred) / data_true )) * 100

Mape(raw_label_re,data_pred)
```

```
5.722297580361968
```

3. 輸出目標資料檔 group_3.csv

```
fin = pd.read_csv('/content/gdrive/My Drive/ML_Final/group_3.csv')
print(fin.head(5))
```

```
   0  1  2  3  4  5  6  ...  6395  6396  6397  6398  6399  Toughness        ML
0  1  1  1  1  0  0  0  ...     0     1     1     1     1   2.413495  2.498423
1  1  1  1  0  0  0  0  ...     0     0     1     1     1   2.900767  2.856366
2  1  1  1  0  0  0  0  ...     0     0     1     1     1   3.326615  3.262297
3  1  1  1  1  1  0  1  ...     1     1     1     1     1   1.638111  1.662913
4  0  0  0  1  1  1  1  ...     1     1     0     0     0   6.759200  6.111736

[5 rows x 6402 columns]
```

4.與其他模型比較

1. 模型架構

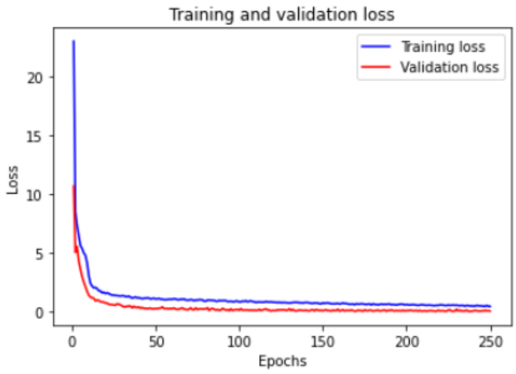
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 80, 80, 16)        416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 40, 40, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 40, 40, 36)        14436
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 20, 36)        0
_________________________________________________________________
dropout (Dropout)            (None, 20, 20, 36)        0
_________________________________________________________________
flatten (Flatten)            (None, 14400)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1843328
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,858,309
Trainable params: 1,858,309
Non-trainable params: 0
_________________________________________________________________
None
```

模型一(2*Convolution Layer 模型)

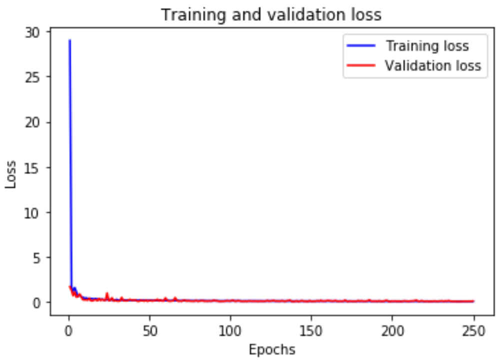
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                204832
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 206,977
Trainable params: 206,977
Non-trainable params: 0
_________________________________________________________________
None
```

模型二(hw4 模型)

2. loss /epochs 圖表



圖表一



圖表二

3. 評估預測誤差

```
#  評估預測誤差
def  Mape(data_true,  data_pred):
    return  np.mean(np.abs((data_true  -  data_pred)  /  data_true  ))  *  100

Mape(raw_label_re,data_pred)
```

5.722297580361968

結果一

```
# 評估誤差誤差
import numpy as np
def Mape(data_true, data_pred):
  return np.mean(np.abs((data_true - data_pred) / data_true )) * 100
ans=Mape(output['Toughness'],output['ML'])
ans
```

6.139292834639335

結果二

5. 提升模型準確度

1.增加 Convolution Layer:

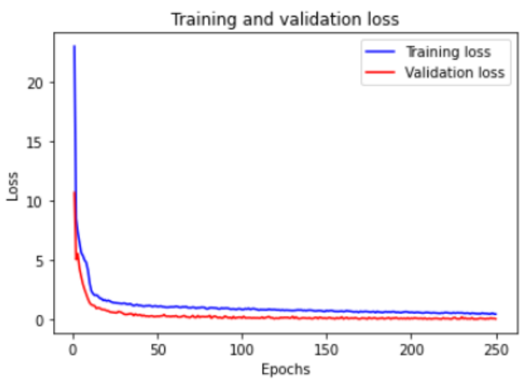
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 80, 80, 16)        416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 40, 40, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 40, 40, 36)        14436
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 20, 36)        0
_________________________________________________________________
dropout (Dropout)            (None, 20, 20, 36)        0
_________________________________________________________________
flatten (Flatten)            (None, 14400)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1843328
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,858,309
Trainable params: 1,858,309
Non-trainable params: 0
_________________________________________________________________
None
```

一開始設計的模型架構

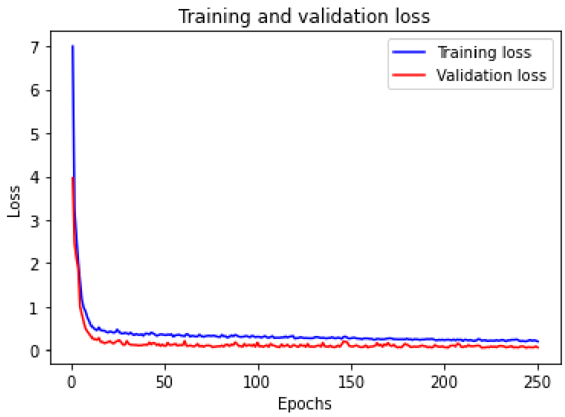
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 80, 80, 16)        416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 40, 40, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 40, 40, 32)        12832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 20, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 20, 20, 64)        51264
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 20, 20, 128)       204928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 12800)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1638528
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,908,097
Trainable params: 1,908,097
Non-trainable params: 0
_________________________________________________________________
None
```

新增兩層卷積層,並且拿掉 flatten 前的 dropout 層

2.loss/epoch 曲線 ， Mape 只有 3.548%



原始模型的 loss/epoch 曲線



新模型的 loss/epoch 曲線

```
# 評估預測誤差
def Mape(data_true, data_pred):
    return np.mean(np.abs((data_true - data_pred) / data_true )) * 100

Mape(raw_label_re,data_pred)
```

3.548472889463808

6. Conclusion

一開始使用 hw4 的模型(4*Dense Layer)和設計的 2* Convolution Layer + 2*Pooling Layer + 2*Dense Layer 模型來對資料進行評估。MAPE(平均絕對百分比誤差)分別是 6.139%和 5.722%。在這點上可以看到,CNN 的強大之處,只要把資料前處理完畢,丟進 model 內,儘管 model 架構很簡單,model 可以給出不錯的回饋。

為了提升模型預測數據的準確度,我嘗試在 model 中新增幾層 Convolution Layer ，建立一個包含 4*Convolution Layer + 2*Pooling Layer + 2*Dense Layer 的模型。除此之外將 Dense Layer 前,在 Convolution Layer 階段的 dropout 拿掉,避免丟失特徵值造成模型預測時,準確度飄忽不定的狀況。對資料進行評估，MAPE 的結果是 3.548%。