2019 IIE
Individual Research Project

以深度學習方法實作簡單語音辨識模型

107034703 陳家安

outline

01 Introduction

02 Data - preprocessing

03 Model Architecture

04 Parameters tuning & Training experiments

05 Conclusion

I NTRODUCTI ON

深度學習的應用領域中，最大宗的莫過於圖像辨識（Image Recognition）及自然語言處理（Natural Language Processing），因此本研究想藉此專題的實作機會，針對後者的領域進行一簡單的程式實作。

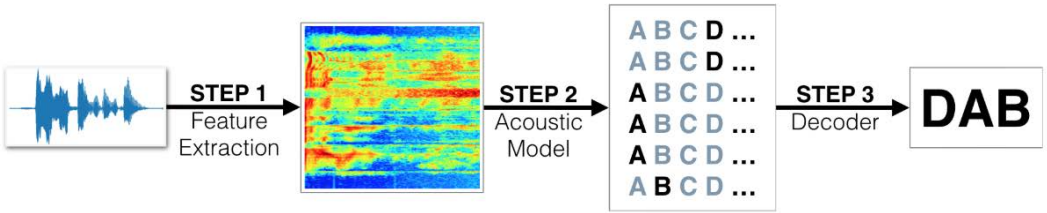
根據課堂所學以及網路上所閱讀之多元教材，利用卷積神經網路(Convolutional Neural Network，CNN)及長短期記憶模型（Long Short-Term Memory，LSTM）實作一簡易語音辨識模型，希望能對簡單的單詞進行辨識，也透過調參設計及實驗，以期發展一高準確率的辨識模型。

CNN

由卷積層、全連通層、池化層組成，搭配反向傳播演算法的運算，能夠利用輸入資料的二維結構擷取特徵並適當的收斂與學習，在圖像和語音辨識方面有出色的表現

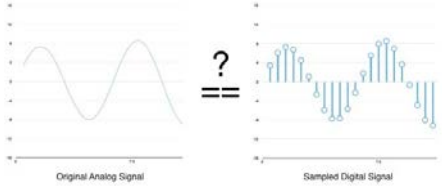
LSTM(RNN)

輸出不僅與當前輸入和網路的權值有關，也與之前網路的輸入有關，常用在處理時序資料，並大量運用在自然語言理解領域。



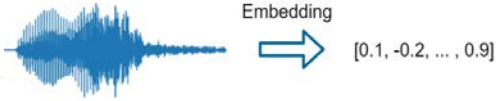
深度網路實現自動語音辨識(Auto Speech Recognition, ASR)的主要流程

Data- preprocessing



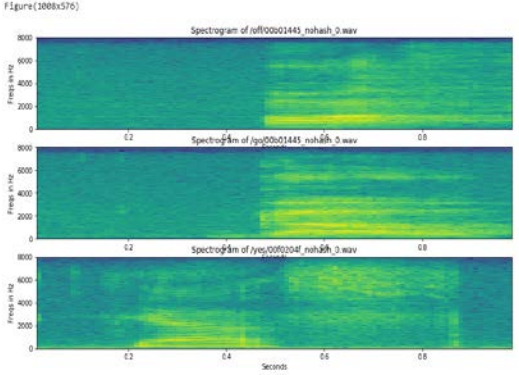
採用Google的公開訓練資料集—**Speech Commands Dataset**
內含30種不同單詞音檔，每個詞都有約2300~2400個原始wav音檔。

01 讀取wav檔，並將字串用array 的方式呈現(Embedding)




```python
# 如果長度太長則隨機截斷
if len(wav) > L:
    i = np.random.randint(0, len(wav) - L)
    wav = wav[i:(i+L)]
# 若太短則隨機加上silence
elif len(wav) < L:
    rem_len = L - len(wav)
    silence_part = np.random.randint(-100,100,16000).astype(np.float32) / np.iinfo(np.int16).max
    j = np.random.randint(0, rem_len)
    silence_part_left  = silence_part[0:j]
    silence_part_right = silence_part[j:rem_len]
    wav = np.concatenate([silence_part_left, wav, silence_part_right])
```

02 將音檔與依照給定的sample rate進行比較，長度太長的部分隨機截斷，長度太短則隨機加上silence的部分。



03 用Short-time Fourier Transform(STFT)的方法轉換成時頻譜(spectrogram)

```python
# 拆分測試資料
def apply_train_test_split(self, test_size, random_state):
    self.df_train, self.df_test = train_test_split(self.df,
                                                   test_size=test_size,
                                                   random_state=random_state)
# 拆分驗證資料
def apply_train_val_split(self, val_size, random_state):
    self.df_train, self.df_val = train_test_split(self.df_train,
                                                  test_size=val_size,
                                                  random_state=random_state)
```

04 將資料以 5:2:3 的比例隨機拆分為training、validation及testing三部分

Model 1 _ CNN

包含三個卷積層、三個最大池化層、
一個扁平層及兩個全連接層。

Training epoch = 15

```
model = deep_cnn(INPUT_SHAPE, NUM_CLASSES)
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['acc'])
```

```
callbacks = [EarlyStopping(monitor='val_acc', patience=4, verbose=1, mode='max')]

history = model.fit_generator(generator=dsGen.generator(BATCH, mode='train'),
                              steps_per_epoch=int(np.ceil(len(dsGen.df_train)/BATCH)),
                              epochs=EPOCHS,
                              verbose=1,
                              callbacks=callbacks,
                              validation_data=dsGen.generator(BATCH, mode='val'),
                              validation_steps=int(np.ceil(len(dsGen.df_val)/BATCH)))
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout, Flatten, Conv2D, MaxPooling2D, BatchNormalization

def deep_cnn(features_shape, num_classes, act='relu'):

    x = Input(name='inputs', shape=features_shape, dtype='float32')
    y = x

    # Block 1
    y = Conv2D(32, (3, 3), activation=act, padding='same', strides=1, name='block1_conv', input_shape=features_shape)(y)
    y = MaxPooling2D((3, 3), strides=(2,2), padding='same', name='block1_pool')(y)
    y = BatchNormalization(name='block1_norm')(y)

    # Block 2
    y = Conv2D(32, (3, 3), activation=act, padding='same', strides=1, name='block2_conv')(y)
    y = MaxPooling2D((3, 3), strides=(2,2), padding='same', name='block2_pool')(y)
    y = BatchNormalization(name='block2_norm')(y)

    # Block 3
    y = Conv2D(32, (3, 3), activation=act, padding='same', strides=1, name='block3_conv')(y)
    y = MaxPooling2D((3, 3), strides=(2,2), padding='same', name='block3_pool')(y)
    y = BatchNormalization(name='block3_norm')(y)

    # Flatten
    y = Flatten(name='flatten')(y)

    # Dense layer
    y = Dense(64, activation=act, name='dense')(y)
    y = BatchNormalization(name='dense_norm')(y)
    y = Dropout(0.2, name='dropout')(y)

    # Predictions
    y = Dense(num_classes, activation='softmax', name='pred')(y)

    # Print network summary
    Model(inputs=x, outputs=y).summary()

    return Model(inputs=x, outputs=y)
```

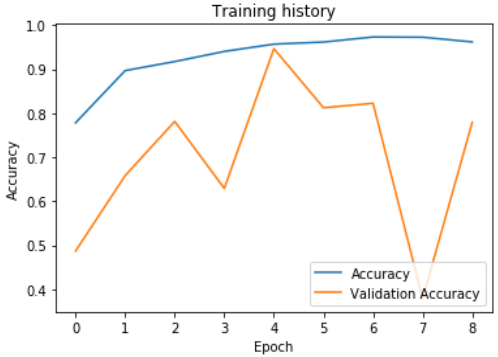
Python / Keras

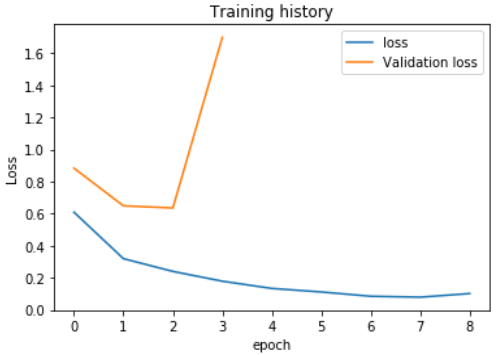
Parameters tuning

- batch size ：32
- Activation ：sigmoid
- Optimizer ：adam

EXPERIMENT 1

Accuracy = 77.8%

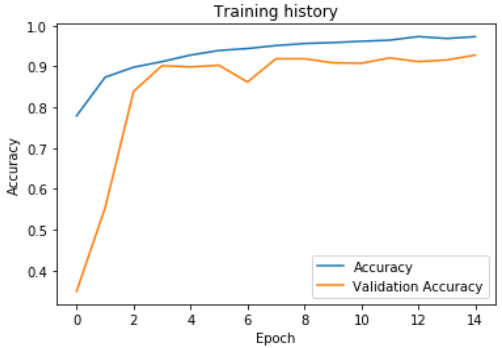


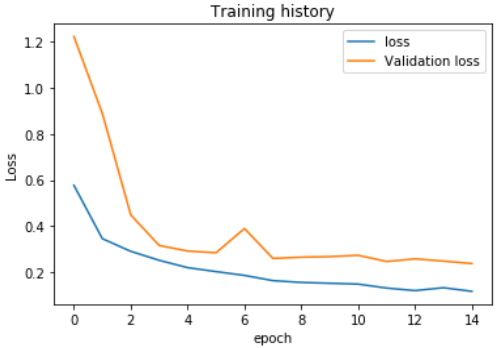


- batch size ：32
- Activation ：softmax
- Optimizer ：adagrad

EXPERIMENT 2

Accuracy = 92.0%



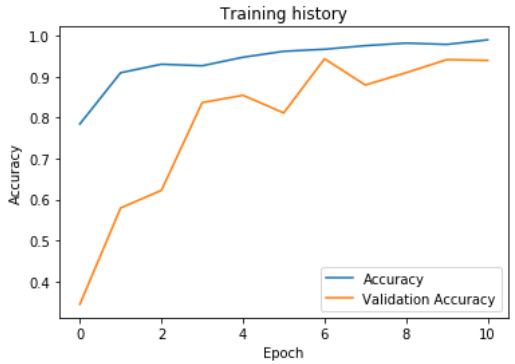


Parameters tuning

- batch size ：32
- Activation：softmax
- Optimizer：adam

EXPERIMENT 3

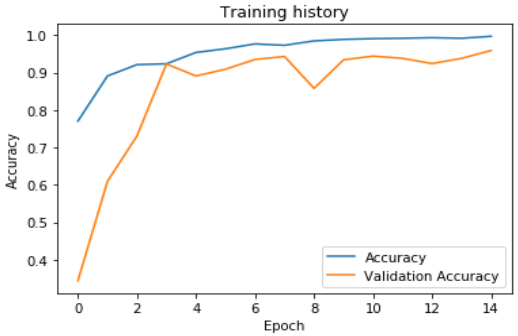
Accuracy = 94.4%

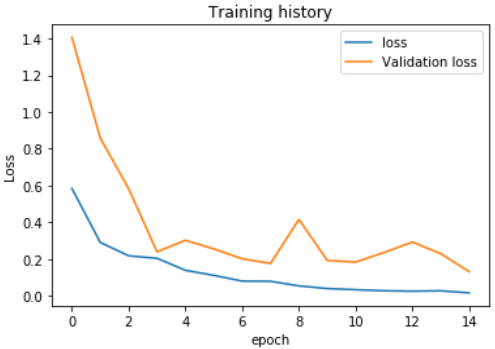


- batch size：64
- Activation：softmax
- Optimizer：adam

EXPERIMENT 4

Accuracy = 95.6%



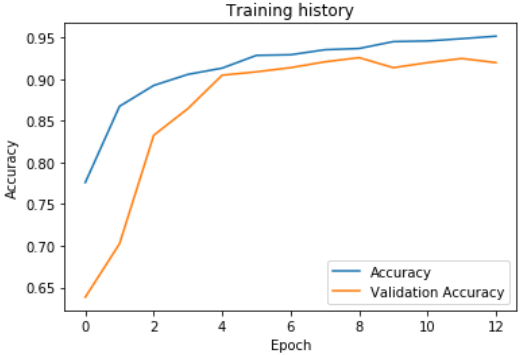


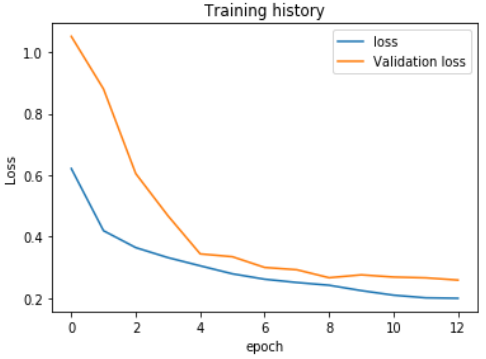
Parameters tuning

EXPERIMENT 5

- batch size ：32
- Activation：sigmoid
- Optimizer：adagrad

Accuracy = 92.7%

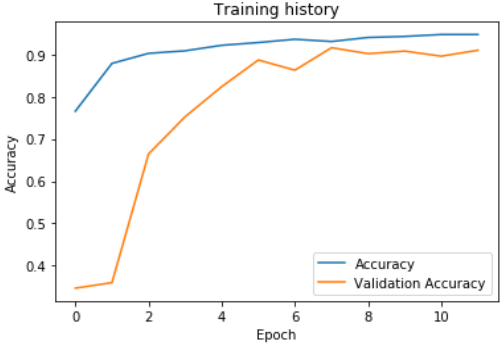


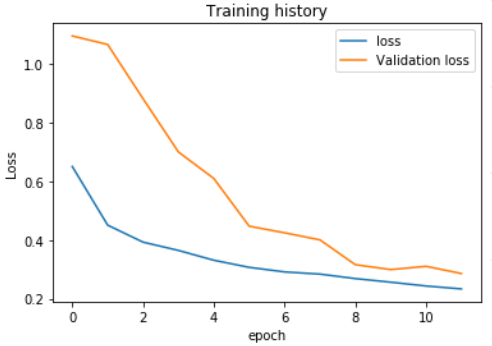


EXPERIMENT 6

- batch size：64
- Activation：sigmoid
- Optimizer：adagrad

Accuracy = 92.5%





Model 2 _ LSTM

Training epoch = 15

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.layers import LSTM

model = Sequential()
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Python / Keras

```python
callbacks = [EarlyStopping(monitor='val_accuracy', patience=4, verbose=1, mode='max')]
history = model.fit_generator(generator=dsGen.generator(BATCH, mode='train'),
                              steps_per_epoch=int(np.ceil(len(dsGen.df_train)/BATCH)),
                              epochs=EPOCHS,
                              verbose=1,
                              callbacks=callbacks,
                              validation_data=dsGen.generator(BATCH, mode='val'),
                              validation_steps=int(np.ceil(len(dsGen.df_val)/BATCH)))
```

Parameters tuning

EXPERIMENT 7

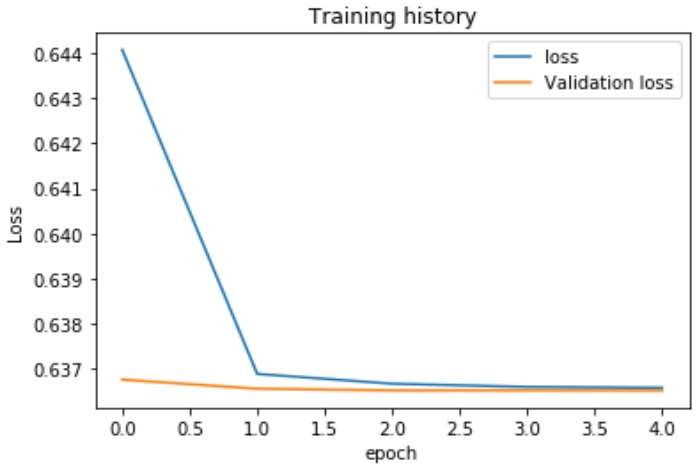
- batch size ：32
- Activation：sigmoid
- Optimizer：adam

Accuracy = 66.7%

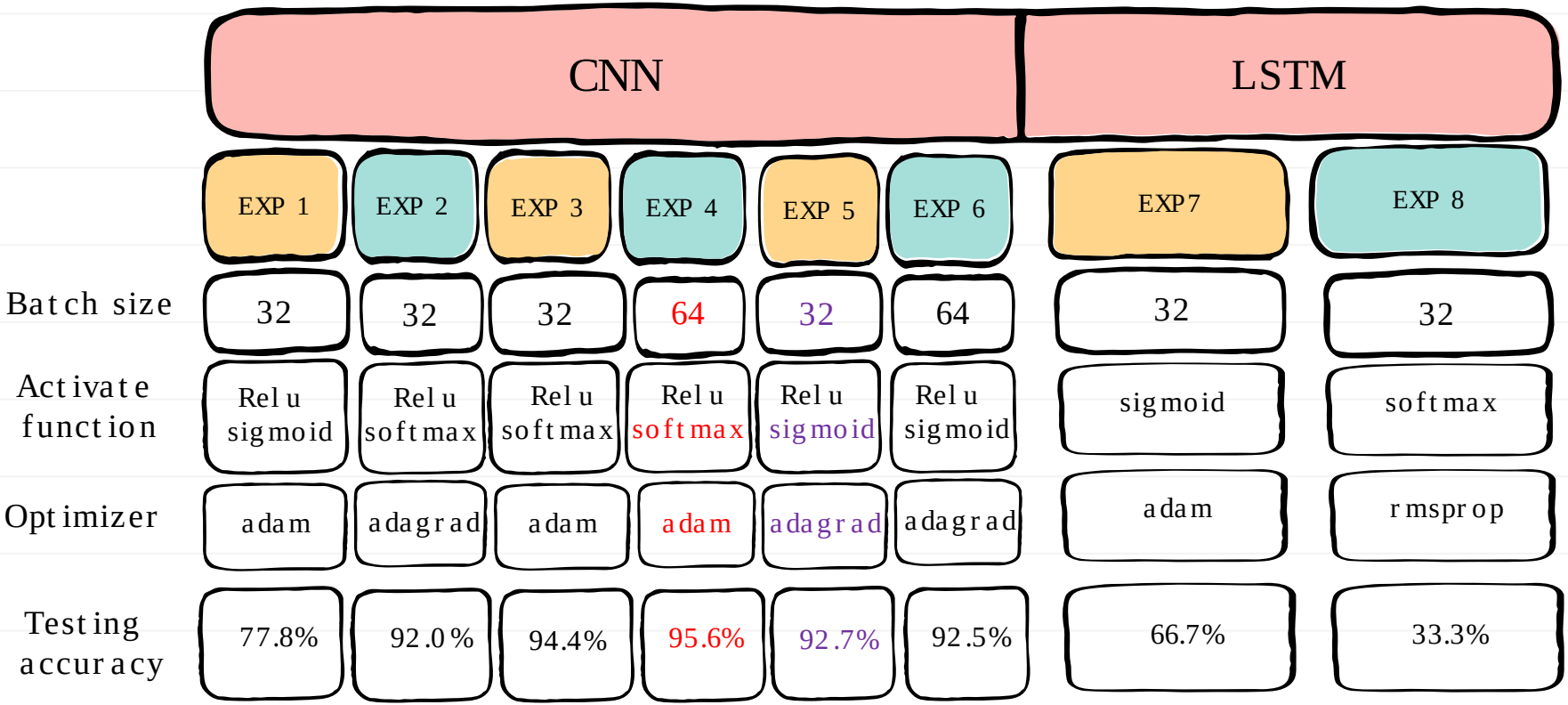
EXPERIMENT 8

- batch size：64
- Activation：softmax
- Optimizer：rmsprop

Accuracy = 33.3%

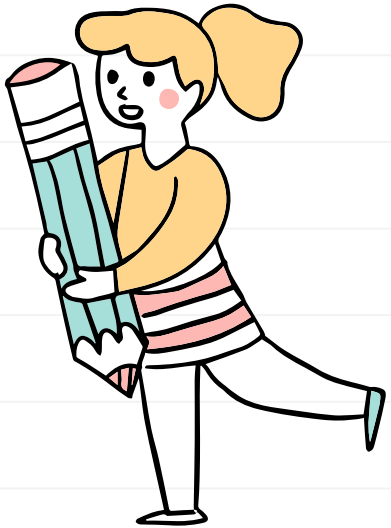


SUMMARY


CNN
LSTM
EXP 1
EXP 2
EXP 3
EXP 4
EXP 5
EXP 6
EXP7
EXP 8
Batch size
32
32
32
64
32
64
32
32
Activate function
Relu sigmoid
Relu softmax
Relu softmax
Relu softmax
Relu sigmoid
Relu sigmoid
sigmoid
softmax
Optimizer
adam
adagrad
adam
adam
adagrad
adagrad
adam
rmsprop
Testing accuracy
77.8%
92.0%
94.4%
95.6%
92.7%
92.5%
66.7%
33.3%

CONCLUSION

本研究利用CNN 及LSTM 兩種深度學習模型進行語音辨識應用的實作，由多次調參的實驗結果可以發現，CNN 對於圖片資料的學習與辨識度較高，大多數皆有9成以上的準確率，經過優化後準確率更可達95.6%。LSTM 雖有記憶特性，但在此例中相對表現較差，未來可考慮結合兩種神經網路，以期進一步提升模型準確率。



THANKS!