American Sign Language Alphabet

-- Classification with CNN

英文字母手語辨識

Group8

卓妤庭 108034548
鄧宏宇 108034550
葉紹康 108034552
林呈昱 108034558

Outline

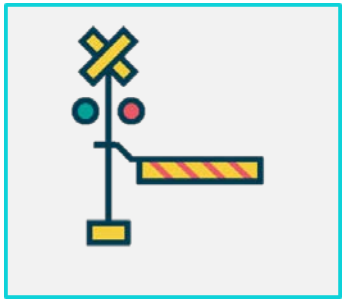
1. 主題說明
2. 資料整理
3. 模型設定
4. 訓練過程
5. 結論與未來展望

主題說明

主題說明-問題描述

聽障者日常溝通不容易

在公共場合需要他人協助才能溝通



主題說明-5W1H

WHY

聽障者無法簡單順利的
獨自完成日常生活溝通自理

WHAT

建立一個辨識手語字母的系統

WHERE

聽障者所在的任何場所

WHEN

聽障者需要與不懂手語的人進行溝通時

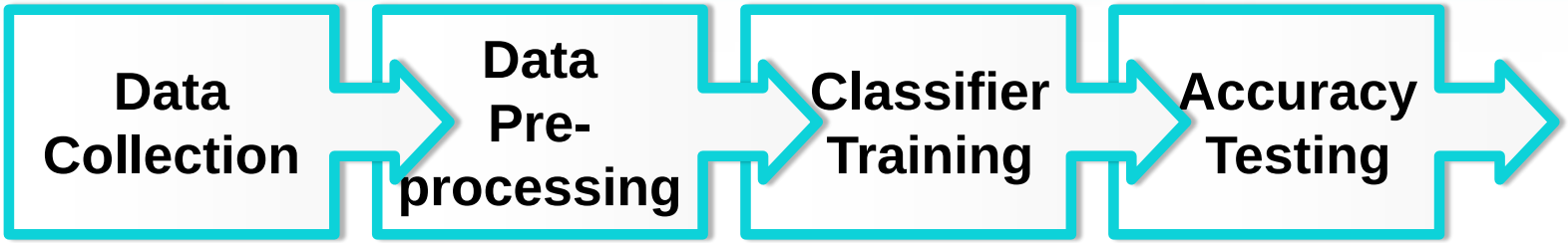
WHO

需要使用手語進行溝通之群體

HOW

收集大量手語字母動作照片，利用卷積神經網路訓練這些照片，以及判斷實際比劃出的動作代表甚麼字母

執行過程



01 **Data Collection**
由Kaggle公開數據集取得資料

02 **Data Pre-processing**
資料前處理

03 **Classifier Training**
建立模型與進行訓練

04 **Accuracy Testing**
投入測試集並得出最終準確率

資料前處理

資料前處理

- 資料集及分類建資料夾
- 將每張圖片格式統一，使每張圖片大小皆為64x64的圖像
- 轉換顏色空間，將圖片從BGR空間轉換到RGB空間
- 讀取各個資料夾，為同類別資料夾內的圖片進行labeling，其值為0-28共29種
- 將圖像做歸一化，使其像素值從 [0-255] 縮放到 [0-1]
- 將labels進行one-hot encoding
- 利用隨機樹種子的設定分割訓練及測試集

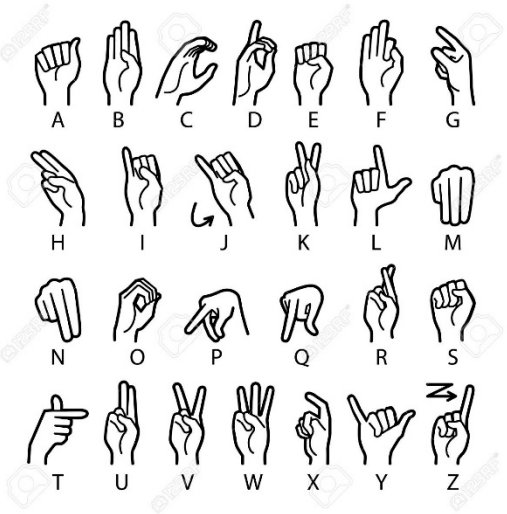
資料前處理

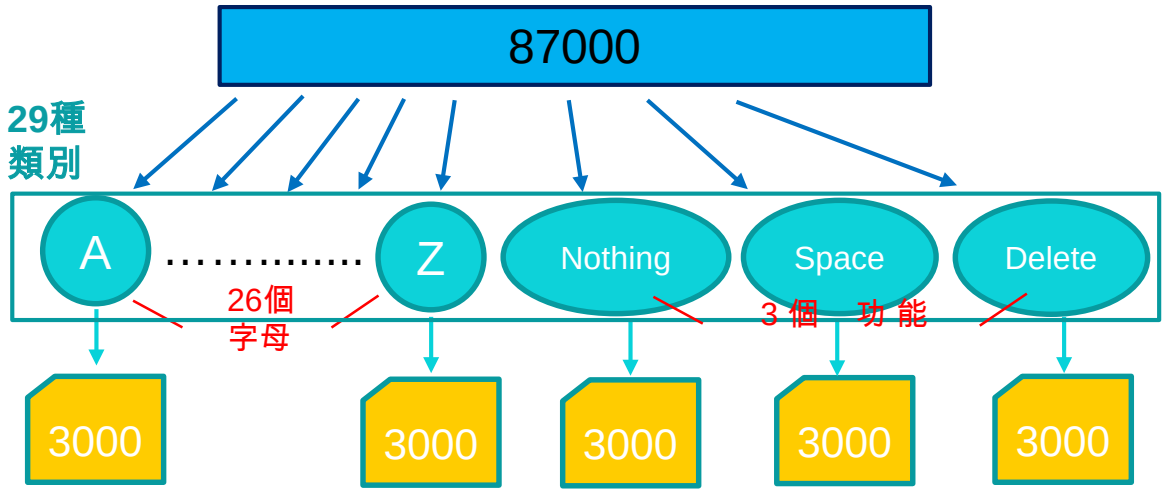
資料集及分類

由Kaggle公開數據集中取得英文手語字母的圖像資料集

共分成29種類別，分別包含26個英文字母，以及3種功能(Nothing、Space、Delete)

每一類別包含3000張照片，總共87000張照片，依類別分別分成29個資料夾。





資料前處理

1. 將每張圖片格式統一，使每張圖片大小皆為64x64的圖像

```
def load_unique():
    size_img = 64,64
    images_for_plot = []
    labels_for_plot = []
    for folder in os.listdir(train_dir):
        for file in os.listdir(train_dir + '/' + folder):
            filepath = train_dir + '/' + folder + '/' + file
            image = cv2.imread(filepath)
            final_img = cv2.resize(image, size_img)
            final_img = cv2.cvtColor(final_img, cv2.COLOR_BGR2RGB)
            images_for_plot.append(final_img)
            labels_for_plot.append(folder)
            break
    return images_for_plot, labels_for_plot

images_for_plot, labels_for_plot = load_unique()
print("unique_labels = ", labels_for_plot)
labels_dict = {'A':0,'B':1,'C':2,'D':3,'E':4,'F':5,'G':6,'H':7,'I':8,'J':9,'K':10,'L':11,'M':12,
               'N':13,'O':14,'P':15,'Q':16,'R':17,'S':18,'T':19,'U':20,'V':21,'W':22,'X':23,'Y':24,
               'Z':25,'space':26,'del':27,'nothing':28}
```

資料前處理

2. 轉換顏色空間，將圖片從BGR空間轉換到RGB空間

```
def load_unique():
    size_img = 64,64
    images_for_plot = []
    labels_for_plot = []
    for folder in os.listdir(train_dir):
        for file in os.listdir(train_dir + '/' + folder):
            filepath = train_dir + '/' + folder + '/' + file
            image = cv2.imread(filepath)
            final_img = cv2.resize(image, size_img)
            final_img = cv2.cvtColor(final_img, cv2.COLOR_BGR2RGB)
            images_for_plot.append(final_img)
            labels_for_plot.append(folder)
            break
    return images_for_plot, labels_for_plot

images_for_plot, labels_for_plot = load_unique()
print("unique_labels = ", labels_for_plot)
labels_dict = {'A':0,'B':1,'C':2,'D':3,'E':4,'F':5,'G':6,'H':7,'I':8,'J':9,'K':10,'L':11,'M':12,
               'N':13,'O':14,'P':15,'Q':16,'R':17,'S':18,'T':19,'U':20,'V':21,'W':22,'X':23,'Y':24,
               'Z':25,'space':26,'del':27,'nothing':28}
```

資料前處理

3. 讀取各個資料夾，為同類別資料夾內的圖片進行labeling，其值為0-28共29種

```
labels_dict = {'A':0,'B':1,'C':2,'D':3,'E':4,'F':5,'G':6,'H':7,'I':8,'J':9,'K':10,'L':11,'M':12,
                   'N':13,'O':14,'P':15,'Q':16,'R':17,'S':18,'T':19,'U':20,'V':21,'W':22,'X':23,'Y':24,
                   'Z':25,'space':26,'del':27,'nothing':28}

labels1_dict = {'A':0,'B':1,'C':2,'D':3,'E':4,'F':5,'G':6,'H':7,'I':8,'J':9,'K':10,'L':11,'M':12,
                   'N':13,'O':14,'P':15,'Q':16,'R':17,'S':18,'T':19,'U':20,'V':21,'W':22,'X':23,'Y':24,
                   'Z':25,'space':26,'del':27,'nothing':28}

def load_data():
    images = []
    labels = []
    size = 64,64
    print("LOADING DATA FROM : ",end = "")
    for folder in os.listdir(train_dir):
        print(folder, end = ' | ')
        for image in os.listdir(train_dir + "/" + folder):
            temp_img = cv2.imread(train_dir + '/' + folder + '/' + image)
            temp_img = cv2.resize(temp_img, size)
            images.append(temp_img)
            if folder == 'A':
                labels.append(labels_dict['A'])
            elif folder == 'B':
                labels.append(labels_dict['B'])
            elif folder == 'C':
                labels.append(labels_dict['C'])
            elif folder == 'D':
                labels.append(labels_dict['D'])
            elif folder == 'E':
                labels.append(labels_dict['E'])
            elif folder == 'F':
                labels.append(labels_dict['F'])
            elif folder == 'G':
                labels.append(labels_dict['G'])
            elif folder == 'H':
                labels.append(labels_dict['H'])
            elif folder == 'I':
                labels.append(labels_dict['I'])
            elif folder == 'J':
                labels.append(labels_dict['J'])
            elif folder == 'K':
                labels.append(labels_dict['K'])
            elif folder == 'L':
                labels.append(labels_dict['L'])
            elif folder == 'M':
                labels.append(labels_dict['M'])
            elif folder == 'N':
                labels.append(labels_dict['N'])
            elif folder == 'O':
                labels.append(labels_dict['O'])
            elif folder == 'P':
                labels.append(labels_dict['P'])
            elif folder == 'Q':
                labels.append(labels_dict['Q'])
            elif folder == 'R':
                labels.append(labels_dict['R'])
            elif folder == 'S':
                labels.append(labels_dict['S'])
            elif folder == 'T':
                labels.append(labels_dict['T'])
            elif folder == 'U':
                labels.append(labels_dict['U'])
            elif folder == 'V':
                labels.append(labels_dict['V'])
            elif folder == 'W':
                labels.append(labels_dict['W'])
            elif folder == 'X':
                labels.append(labels_dict['X'])
            elif folder == 'Y':
                labels.append(labels_dict['Y'])
            elif folder == 'Z':
                labels.append(labels_dict['Z'])
            elif folder == 'space':
                labels.append(labels_dict['space'])
            elif folder == 'del':
                labels.append(labels_dict['del'])
            elif folder == 'nothing':
                labels.append(labels_dict['nothing'])
```

資料前處理

4. 將圖像做歸一化，使其像素值從 [0-255] 縮放到 [0-1]

```
images = np.array(images)
print("images_for_plot = ", images)
images = images.astype('float32')/255.0
print("images_for_plot = ", images)
```

```
[[196   8   8]
 [ 65  44  45]
 [ 60  41  39]
 ...
 [100 118 117]
 [100 119 119]
 [182 110 106]]

[[209   5   5]
 [ 96  10  13]
 [ 93   8   9]
 ...
 [183 107 107]
 [180 108 109]
 [228 100  97]]]]
```

歸一化後

```
[[0.76862746 0.03137255 0.03137255]
 [0.25490198 0.17254902 0.1764706 ]
 [0.23529412 0.16078432 0.15294118]
 ...
 [0.39215687 0.4627451  0.45882353]
 [0.39215687 0.46666667 0.46666667]
 [0.7137255  0.43137255 0.41568628]]

[[0.81960785 0.01960784 0.01960784]
 [0.3764706  0.03921569 0.05098039]
 [0.3647059  0.03137255 0.03529412]
 ...
 [0.7176471  0.41960785 0.41960785]
 [0.7058824  0.42352942 0.42745098]
 [0.89411765 0.39215687 0.38039216]]]]
```

資料前處理

5. 將labels進行one-hot encoding

```
labels = keras.utils.to_categorical(labels)   #one-hot encoding
print(labels)

X_train, X_test, Y_train, Y_test = train_test_split(images, labels, t

print()
print('Loaded', len(X_train),'images for training,','Train data shape
print('Loaded', len(X_test),'images for testing','Test data shape =',
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.
 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 0. 0. 0.]
```

資料前處理

6. 將90%的資料當作訓練集；10%的資料當作測試集

(之後在模型訓練過程中將訓練集中的10%分割出來當作驗證集)

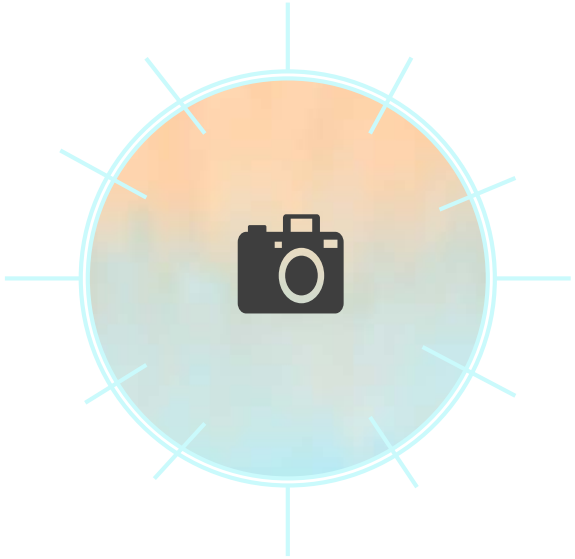
(訓練集 70470 張，驗證集 7830 張，測試集 8700 張)

其中分割出來的測試集利用隨機樹種子的設定，後續測試不同模型時所抽取出的測試集會與之前相同

```
labels = keras.utils.to_categorical(labels)    #one-hot encoding
print(labels)

X_train, X_test, Y_train, Y_test = train_test_split(images, labels, test_size = 0.1, random_state = 3)
```

```
def fit_model():
    history = model.fit(X_train, Y_train, batch_size = 64, epochs = 10, validation_split = 0.1)
    return history
```



模型設定

模型設定

設計出之卷積神經網路模型：

1.進行兩個卷積層

2.經過一次比例為0.5之dropout

3.進行兩個卷積層

4.經過一次比例為0.5之dropout

5.進行兩個卷積層

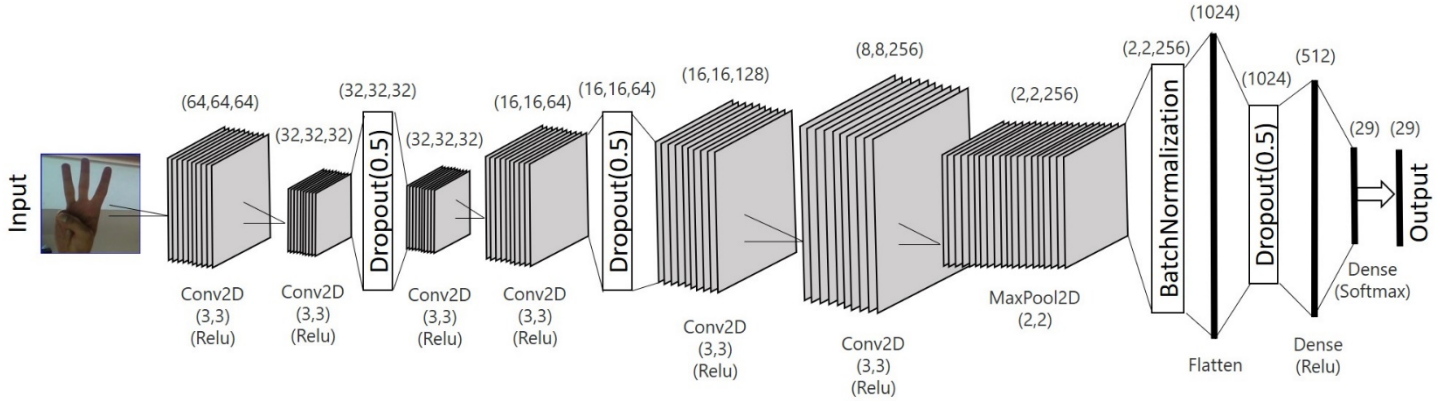
6.進行一個池化層

7.進行Flatten拉直

8.經過一次比例為0.5之dropout

9.進行兩次全連接層

10.輸出29單位之輸出項



模型設定

	Loss	卷積層、池化層 C: Conv2D D: Dropout(0.5) P: MaxPool2D(3)	全連接層	激活函數	最後一層 全連接層 激活函數	優化器	Epoch
1.	categorical	2C1D2C1D2C1P	2	relu	softmax	adam	10

```
def build_model():

    model = Sequential()

    model.add(Conv2D(64, kernel_size = 3, padding = 'same', activation = 'relu', input_shape = (64,64,3)))
    model.add(Conv2D(32, kernel_size = 3, padding = 'same', strides = 2, activation = 'relu'))
    model.add(Dropout(0.5))

    model.add(Conv2D(32, kernel_size = 3, padding = 'same', activation = 'relu'))
    model.add(Conv2D(64, kernel_size = 3, padding = 'same', strides = 2, activation = 'relu'))
    model.add(Dropout(0.5))

    model.add(Conv2D(128, kernel_size = 3, padding = 'same', activation = 'relu'))
    model.add(Conv2D(256, kernel_size = 3, padding = 'same', strides = 2 , activation = 'relu'))
    model.add(MaxPool2D(3))

    model.add(BatchNormalization())

    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(512, activation = 'relu'))
    model.add(Dense(29, activation = 'softmax'))

    model.compile(optimizer = 'adam', loss = tensorflow.keras.losses.categorical_crossentropy, metrics = ["accuracy"])
```

1.進行兩個卷積層

2.經過一次比例為0.5之dropout

3.進行兩個卷積層

4.經過一次比例為0.5之dropout

5.進行兩個卷積層

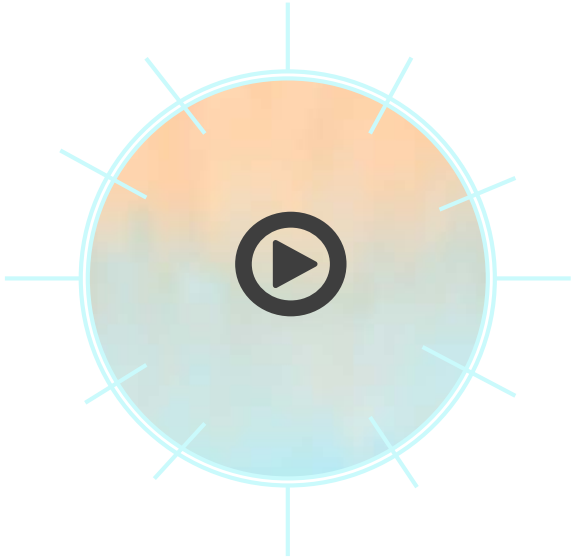
6.進行一個池化層

7.進行Flatten拉直

8.經過一次比例為0.5之dropout

9.進行兩次全連接層

10.輸出29單位之輸出項



訓練過程

訓練過程-測試(1)

使用原先之最初始模型

	Loss	卷積層、池化層 C: Conv2D D: Dropout(0.5) P: MaxPool2D(3)	全連接層	激活函數	最後一層 全連接層 激活函數	優化器	Epoch
1.	categorical	2C1D2C1D2C1P	2	relu	softmax	adam	10

CREATED
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 64)	1792
conv2d_1 (Conv2D)	(None, 32, 32, 32)	18464
dropout (Dropout)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	9248
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18496
dropout_1 (Dropout)	(None, 16, 16, 64)	0
conv2d_4 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_5 (Conv2D)	(None, 8, 8, 256)	295168
max_pooling2d (MaxPooling2D)	(None, 2, 2, 256)	0
batch_normalization (BatchNo	(None, 2, 2, 256)	1024
flatten (Flatten)	(None, 1024)	0
dropout_2 (Dropout)	(None, 1024)	0
dense (Dense)	(None, 512)	524800
dense_1 (Dense)	(None, 29)	14877

```
Train on 70470 samples, validate on 7830 samples
Epoch 1/10
70470/70470 [==============================] - 276s 4ms/sample - loss: 1.6008 - accuracy: 0.5063 - val_loss: 0.7665 - val_accuracy: 0.7386
Epoch 2/10
70470/70470 [==============================] - 277s 4ms/sample - loss: 0.3839 - accuracy: 0.8674 - val_loss: 0.2628 - val_accuracy: 0.9114
Epoch 3/10
70470/70470 [==============================] - 277s 4ms/sample - loss: 0.2227 - accuracy: 0.9229 - val_loss: 0.1001 - val_accuracy: 0.9645
Epoch 4/10
70470/70470 [==============================] - 277s 4ms/sample - loss: 0.1514 - accuracy: 0.9474 - val_loss: 0.4122 - val_accuracy: 0.8765
Epoch 5/10
70470/70470 [==============================] - 277s 4ms/sample - loss: 0.1251 - accuracy: 0.9581 - val_loss: 0.1217 - val_accuracy: 0.9572
Epoch 6/10
70470/70470 [==============================] - 278s 4ms/sample - loss: 0.1057 - accuracy: 0.9634 - val_loss: 0.0703 - val_accuracy: 0.9757
Epoch 7/10
70470/70470 [==============================] - 278s 4ms/sample - loss: 0.0975 - accuracy: 0.9678 - val_loss: 0.1628 - val_accuracy: 0.9489
Epoch 8/10
70470/70470 [==============================] - 278s 4ms/sample - loss: 0.0889 - accuracy: 0.9708 - val_loss: 0.0269 - val_accuracy: 0.9900
Epoch 9/10
70470/70470 [==============================] - 277s 4ms/sample - loss: 0.0767 - accuracy: 0.9742 - val_loss: 0.0257 - val_accuracy: 0.9902
Epoch 10/10
70470/70470 [==============================] - 277s 4ms/sample - loss: 0.0704 - accuracy: 0.9770 - val_loss: 0.0484 - val_accuracy: 0.9824
Final Accuracy: 97.70%
Validation Set Accuracy: 98.24%
```

Total params: 957,725
Trainable params: 957,213
Non-trainable params: 512

==========] - 8s 925us/sample - loss: 0.1029 - accuracy: 0.9811

Test loss: 0.05193019951383273
Test accuracy: 0.98114944

訓練過程-測試(2)

設立第二個模型：
更改損失函數為binary

	Loss	卷積層、池化層 C: Conv2D D: Dropout(0.5) P: MaxPool2D(3)	全連接層	激活函數	最後一層 全連接層 激活函數	優化器	Epoch
1.	categorical	2C1D2C1D2C1P	2	relu	softmax	adam	10
2.	**binary**	2C1D2C1D2C1P	2	relu	softmax	adam	10

```
| MODEL CREATED
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 64)	1792
conv2d_1 (Conv2D)	(None, 32, 32, 32)	18464
dropout (Dropout)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	9248
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18496
dropout_1 (Dropout)	(None, 16, 16, 64)	0
conv2d_4 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_5 (Conv2D)	(None, 8, 8, 256)	295168
max_pooling2d (MaxPooling2D)	(None, 2, 2, 256)	0
batch_normalization (BatchNo	(None, 2, 2, 256)	1024
flatten (Flatten)	(None, 1024)	0
dropout_2 (Dropout)	(None, 1024)	0
dense (Dense)	(None, 512)	524800
dense_1 (Dense)	(None, 29)	14877

```
Train on 70470 samples, validate on 7830 samples
Epoch 1/10
70470/70470 [==============================] - 280s 4ms/sample - loss: 0.0806 - accuracy: 0.9756 - val_loss: 0.1429 - val_accuracy: 0.9646
Epoch 2/10
70470/70470 [==============================] - 279s 4ms/sample - loss: 0.0237 - accuracy: 0.9909 - val_loss: 0.0590 - val_accuracy: 0.9826
Epoch 3/10
70470/70470 [==============================] - 278s 4ms/sample - loss: 0.0139 - accuracy: 0.9947 - val_loss: 0.0236 - val_accuracy: 0.9918
Epoch 4/10
70470/70470 [==============================] - 279s 4ms/sample - loss: 0.0104 - accuracy: 0.9961 - val_loss: 0.0176 - val_accuracy: 0.9938
Epoch 5/10
70470/70470 [==============================] - 279s 4ms/sample - loss: 0.0085 - accuracy: 0.9969 - val_loss: 0.0320 - val_accuracy: 0.9898
Epoch 6/10
70470/70470 [==============================] - 279s 4ms/sample - loss: 0.0072 - accuracy: 0.9974 - val_loss: 0.0284 - val_accuracy: 0.9912
Epoch 7/10
70470/70470 [==============================] - 279s 4ms/sample - loss: 0.0066 - accuracy: 0.9976 - val_loss: 0.0281 - val_accuracy: 0.9907
Epoch 8/10
70470/70470 [==============================] - 279s 4ms/sample - loss: 0.0058 - accuracy: 0.9980 - val_loss: 0.0165 - val_accuracy: 0.9945
Epoch 9/10
70470/70470 [==============================] - 280s 4ms/sample - loss: 0.0056 - accuracy: 0.9981 - val_loss: 0.0250 - val_accuracy: 0.9921
Epoch 10/10
70470/70470 [==============================] - 279s 4ms/sample - loss: 0.0050 - accuracy: 0.9982 - val_loss: 0.0139 - val_accuracy: 0.9954
Final Accuracy: 99.82%
Validation Set Accuracy: 99.54%
```

```
Total params: 957,725
Trainable params: 957,213
Non-trainable params: 512
```

```
=] - 8s 915us/sample - loss: 0.0216 - accuracy: 0.9954

Test loss: 0.014105188155696653
Test accuracy: 0.9953756
```

訓練過程-測試(3)

設立第三個模型：
更改激活函數為sigmoid

	Loss	卷積層、池化層 C: Conv2D D: Dropout(0.5) P: MaxPool2D(3)	全連接層	激活函數	最後一層 全連接層 激活函數	優化器	Epoch
1.	categorical	2C1D2C1D2C1P	2	relu	softmax	adam	10
2.	binary	2C1D2C1D2C1P	2	relu	softmax	adam	10
3.	binary	2C1D2C1D2C1P	2	relu	**sigmoid**	adam	10

```
MODEL CREATED
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 64, 64, 64)        1792
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 32)        18464
_________________________________________________________________
dropout_4 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
dropout_5 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 16, 16, 128)       73856
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 256)         295168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 2, 2, 256)         1024
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dropout_6 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_4 (Dense)              (None, 29)                14877
=================================================================
Total params: 957,725
Trainable params: 957,213
Non-trainable params: 512
```

```
Train on 70470 samples, validate on 7830 samples
Epoch 1/10
70470/70470 [==============================] - 654s 9ms/step - loss: 0.0882 - accuracy: 0.9734 - val_loss: 0.0637 -
val_accuracy: 0.9770
Epoch 2/10
70470/70470 [==============================] - 643s 9ms/step - loss: 0.0239 - accuracy: 0.9908 - val_loss: 0.0529 -
val_accuracy: 0.9816
Epoch 3/10
70470/70470 [==============================] - 641s 9ms/step - loss: 0.0132 - accuracy: 0.9950 - val_loss: 0.0064 -
val_accuracy: 0.9978
Epoch 4/10
70470/70470 [==============================] - 640s 9ms/step - loss: 0.0092 - accuracy: 0.9966 - val_loss: 0.0096 -
val_accuracy: 0.9965
Epoch 5/10
70470/70470 [==============================] - 640s 9ms/step - loss: 0.0073 - accuracy: 0.9974 - val_loss: 0.0029 -
val_accuracy: 0.9990
Epoch 6/10
70470/70470 [==============================] - 663s 9ms/step - loss: 0.0058 - accuracy: 0.9979 - val_loss: 0.0016 -
val_accuracy: 0.9995
Epoch 7/10
70470/70470 [==============================] - 658s 9ms/step - loss: 0.0048 - accuracy: 0.9983 - val_loss: 9.2535e-04 -
val_accuracy: 0.9998
Epoch 8/10
70470/70470 [==============================] - 668s 9ms/step - loss: 0.0046 - accuracy: 0.9984 - val_loss: 0.0012 -
val_accuracy: 0.9996
Epoch 9/10
70470/70470 [==============================] - 667s 9ms/step - loss: 0.0038 - accuracy: 0.9987 - val_loss: 7.1685e-04 -
val_accuracy: 0.9998
Epoch 10/10
70470/70470 [==============================] - 658s 9ms/step - loss: 0.0033 - accuracy: 0.9989 - val_loss: 0.0017 -
val_accuracy: 0.9994
Final Accuracy: 99.89%
Validation Set Accuracy: 99.94%
8700/8700 [==============================] - 19s 2ms/step
Test loss: 0.0017850756841101523
Test accuracy: 0.9992982149124146
```

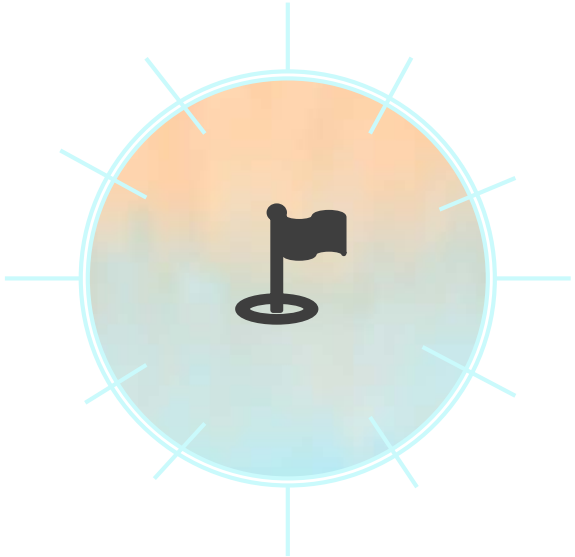
訓練過程-測試(4)

設立第四個模型：
減去兩個卷積層

	Loss	卷積層、池化層 C: Conv2D D: Dropout(0.5) P: MaxPool2D(3)	全連接層	激活函數	最後一層 全連接層 激活函數	優化器	Epoch
1.	categorical	2C1D2C1D2C1P	2	relu	softmax	adam	10
2.	binary	2C1D2C1D2C1P	2	relu	softmax	adam	10
3.	binary	2C1D2C1D2C1P	2	relu	sigmoid	adam	10
4.	binary	**2C1D** **2C1P**	2	relu	sigmoid	adam	10

```
In [32]: model = build_model()
MODEL CREATED
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 64, 64, 64)        1792
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 32, 32, 32)        18464
_________________________________________________________________
dropout_7 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 32, 32, 128)       36992
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 16, 16, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 256)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 5, 5, 256)         1024
_________________________________________________________________
flatten_3 (Flatten)          (None, 6400)              0
_________________________________________________________________
dropout_8 (Dropout)          (None, 6400)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               3277312
_________________________________________________________________
dense_6 (Dense)              (None, 29)                14877
=================================================================
Total params: 3,645,629
Trainable params: 3,645,117
Non-trainable params: 512
_________________________________________________________________
```

```
In [30]: model_history = fit_model()
Train on 70470 samples, validate on 7830 samples
Epoch 1/10
70470/70470 [==============================] - 658s 9ms/step - loss: 0.0043 - accuracy: 0.9986 - val_loss: 0.0021 -
val_accuracy: 0.9992
Epoch 2/10
70470/70470 [==============================] - 657s 9ms/step - loss: 0.0029 - accuracy: 0.9990 - val_loss: 4.9589e-04 -
val_accuracy: 0.9999
Epoch 3/10
70470/70470 [==============================] - 660s 9ms/step - loss: 0.0028 - accuracy: 0.9990 - val_loss: 3.9362e-04 -
val_accuracy: 0.9999
Epoch 4/10
70470/70470 [==============================] - 657s 9ms/step - loss: 0.0025 - accuracy: 0.9991 - val_loss: 1.2871e-04 -
val_accuracy: 1.0000
Epoch 5/10
70470/70470 [==============================] - 656s 9ms/step - loss: 0.0024 - accuracy: 0.9992 - val_loss: 5.5297e-04 -
val_accuracy: 0.9998
Epoch 6/10
70470/70470 [==============================] - 654s 9ms/step - loss: 0.0023 - accuracy: 0.9992 - val_loss: 2.8194e-04 -
val_accuracy: 0.9999
Epoch 7/10
70470/70470 [==============================] - 651s 9ms/step - loss: 0.0020 - accuracy: 0.9993 - val_loss: 2.2002e-04 -
val_accuracy: 0.9999
Epoch 8/10
70470/70470 [==============================] - 652s 9ms/step - loss: 0.0019 - accuracy: 0.9994 - val_loss: 0.0020 -
val_accuracy: 0.9993
Epoch 9/10
70470/70470 [==============================] - 652s 9ms/step - loss: 0.0021 - accuracy: 0.9993 - val_loss: 5.3420e-04 -
val_accuracy: 0.9999
Epoch 10/10
70470/70470 [==============================] - 652s 9ms/step - loss: 0.0017 - accuracy: 0.9995 - val_loss: 1.5109e-04 -
val_accuracy: 0.9999
Final Accuracy: 99.95%
Validation Set Accuracy: 99.99%
8700/8700 [==============================] - 18s 2ms/step
Test loss: 0.0001364278105967091
Test accuracy: 0.9999642968177795
```



結論與未來展望

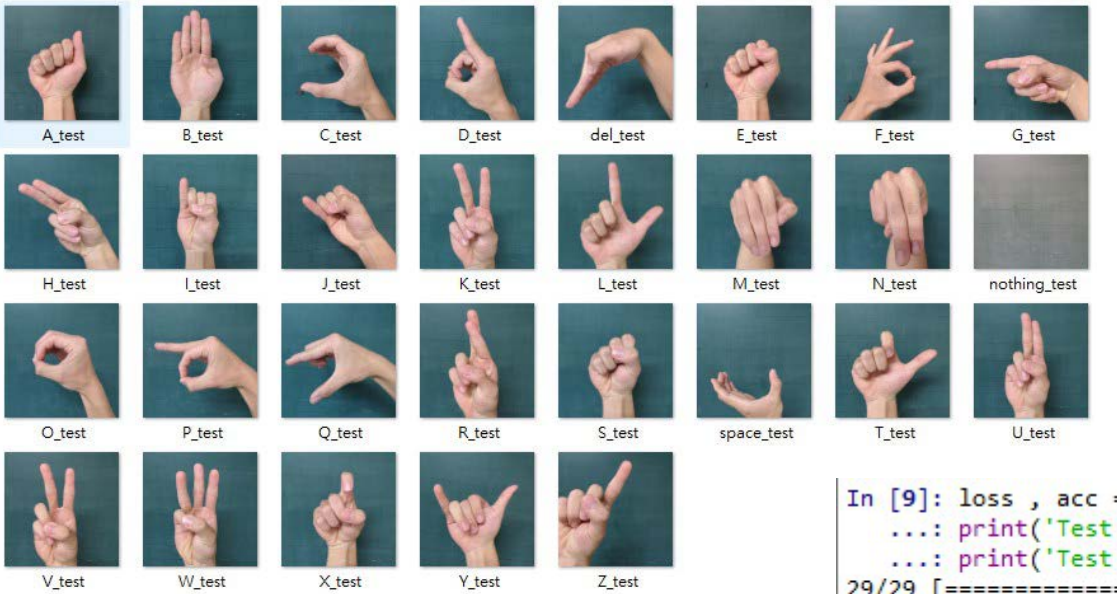
結果統整

從原始模型經過3次之改善測試，可以得到測試集準確率漸漸提升之效果。

	Loss	卷積層、池化層 C: Conv2D D: Dropout(0.5) P: MaxPool2D(3)	全連接層	激活函數	最後一層 全連接層 激活函數	優化器	Epoch	訓練集 accuracy	驗證集 accuracy	測試集 accuracy
1.	categorical	2C1D2C1D2C1P	2	relu	softmax	adam	10	97.70%	98.24%	98.11%
2.	binary	2C1D2C1D2C1P	2	relu	softmax	adam	10	99.82%	99.54%	99.54%
3.	binary	2C1D2C1D2C1P	2	relu	sigmoid	adam	10	99.89%	99.94%	99.92%
4.	binary	2C1D2C1P	2	relu	sigmoid	adam	10	99.95%	99.99%	99.99%

加入新的測試集

引入本小組所私下拍攝之手語照片進行測試，
每個類別各拍攝一張，共29張，其準確率為97.38%




```
In [9]: loss , acc =model.evaluate(X_test1, Y_test1)
   ...: print('Test loss_New:', loss)
   ...: print('Test accuracy_New:', acc)
29/29 [==============================] - 0s 3ms/step
Test loss_New: 0.4344358444213867
Test accuracy_New: 0.9738406538963318
```

結論

- **準確度有所提升**

經過四種不同的模型測試，
準確度從原先98.11%提升至99.99%。

- **新測試集亦有好表現**

在一穩定新測試集下，其準確率亦高達97.38%，顯見此模型能為外界所提供之照片提供可信的辨識能力

- **系統貢獻**

為未來手語自動辨識系統建立基礎

- **提升聽障者社會福利**

使聽障者能有更完善的生活環境。

未來展望

期望發展辨識手語內容系統

未來可收集不僅限於字母手勢，包含字詞的手勢資料，增加系統之實用性。

添加更多複雜度較高的照片，以再提升模型彈性

未來增加訓練集照片的不相關性，豐富訓練集之複雜度，提升模型對未來外界引入的新照片之彈性，而更精準地去判斷實際應用手勢。

完整化字母辨識系統

目前僅做到辨識單一字母，未來可加入時間序列及自然語言之處理，更完整化系統。

Thanks