智慧化企業整合

Project 3

使用 CNN 進行 LEGO 圖像辨識

Individual Research Project

109034553 莊婉琦

目錄

一、緒論

1. 主題介紹

樂高是兒童家中常見的玩具，樂高提供不同層次的任務挑戰，學習形狀、顏色識別、計算數字和圖案等，建構基本的數學技能，模型建構、自由創作能亦能協助訓練空間能力，對於學齡孩童是一個很好的教材。樂高在教育方面，協助家長和孩童可以一起激發創造力，把建構出來的積木當作互動扮演的素材。孩子可以藉由遊戲表達自己的想法、促進各種情境的語言溝通、問題解決的能力，還可以培養孩子的合作性、專注度和責任感。樂高的無限創造空間，深受大眾喜愛。

樂高官方網站收錄了許多電影及動畫的角色，從經典的哈利波特系列到漫威 DC 系列，新增現代電影、動漫角色速度也有目共睹，本篇研究將利用 Kaggle 資料庫所蒐集的樂高圖像資料，建立 CNN 模型辨識 LEGO 人物角色，並探討其辨識結果及正確性。

2. 5W1H

Why: LEGO 新增現代電影、動漫角色速度有目共睹，透過 LEGO 人物的辨識系統，協助樂高玩家更認識各個角色

What: 藉由建立 CNN 辨識樂高圖像，讓樂高玩家認識自己未知的角色

Where: 樂高販賣店

When: 選購樂高時

Who: LEGO 顧客、LEGO 販賣店員工

How: 蒐集 LEGO 角色圖片，利用卷積神經網路訓練照片，判斷出 LEGO 的角色

二、研究方法

研究之架構共分為以下 6 個步驟：(1)資料蒐集、(2)資料前處理、(3)模型建立、(4)訓練模型、(5)驗證集驗證、(6)改善過程，得出分類效果較佳的模型。

1. 資料蒐集

由 Kaggle 公開數據集中取得樂高圖像資料集，其中 3 個資料夾中分別包含不同系列的樂高角色，包含哈利波特系列 2 個角色、漫威電影宇宙系列 10 個角色、星際大戰系列 10 個角色，樣本數總共有 240 張圖片；資料集中一包含兩個 csv 檔，其中一資料集整理圖片路徑、訓練集/測試集標籤，另一資料則標示角色編號、樂高盒組系列名稱以及樂高角色名稱，樣本共包含 22 個樂高角色。

2. 資料前處理

(1) 匯入資料集

將蒐集下來的資料及上傳至雲端硬碟，掛接 Google 雲端允許 Colaboratory 後端存取雲端硬碟中的檔案。

```
#匯入存在雲端的資料集
path = 'drive/MyDrive/Lego/'
```

```
#查看資料 index.csv
index_df = pd.read_csv('drive/MyDrive/Lego/index.csv')
index_df.head()
```

	path	class_id	train-valid
0	marvel/0001/001.jpg	1	train
1	marvel/0001/002.jpg	1	valid
2	marvel/0001/003.jpg	1	train
3	marvel/0001/004.jpg	1	train
4	marvel/0001/005.jpg	1	train

```
#查看 metadata.csv
meta_df = pd.read_csv(path+'metadata.csv')
meta_df.head()
```

	class_id	lego_ids	lego_names	minifigure_name
0	1	[76115]	['Spider Mech vs. Venom']	SPIDER-MAN
1	2	[76115]	['Spider Mech vs. Venom']	VENOM
2	3	[76115]	['Spider Mech vs. Venom']	AUNT MAY
3	4	[76115]	['Spider Mech vs. Venom']	GHOST SPIDER
4	5	[75208]	["Yoda's Hut"]	YODA

(2) 資料合併

組合兩個來源的資料集，確認資料名稱及屬性一致後，進行資料的水平合併，這項操作將兩個不同的資料非常直觀的串聯，整理出一個更乾淨的表格。

```
#將index、metadata兩個資料集合併整理，多對多join(path對class_id/minifigure_name)
data_df = pd.merge(index_df, meta_df[['class_id', 'minifigure_name']], on='class_id')
data_df
```

	path	class_id	train-valid	minifigure_name
0	marvel/0001/001.jpg	1	train	SPIDER-MAN
1	marvel/0001/002.jpg	1	valid	SPIDER-MAN
2	marvel/0001/003.jpg	1	train	SPIDER-MAN
3	marvel/0001/004.jpg	1	train	SPIDER-MAN
4	marvel/0001/005.jpg	1	train	SPIDER-MAN
...	...	...	...	...
235	marvel/0010/010.jpg	22	train	TASKMASTER
236	marvel/0010/011.jpg	22	valid	TASKMASTER
237	marvel/0010/012.jpg	22	valid	TASKMASTER
238	marvel/0010/013.jpg	22	train	TASKMASTER
239	marvel/0010/014.jpg	22	train	TASKMASTER

(3) 檢查資料是否有缺值或遺漏值

結果顯示資料並無缺值，故不用進行資料補值處理。

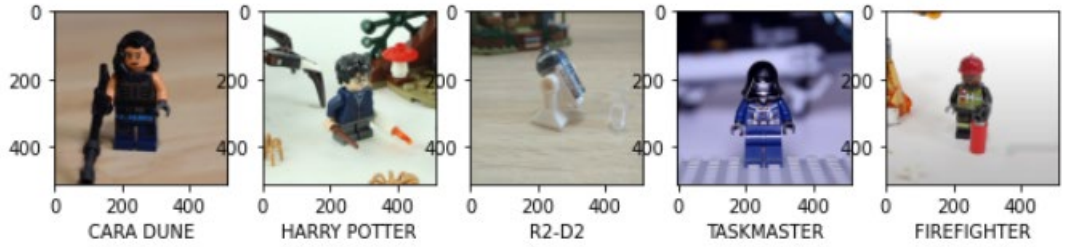
```
#查看資料有無缺值
data_df.isnull().sum()
```

```
path              0
class_id          0
train-valid       0
minifigure_name   0
dtype: int64
```

(4) 資料視覺化

從所有圖片集中隨機抽取 20 張 (為簡單說明，此處只擷取 5 張圖片)來檢視。

```
plt.figure(figsize=(10,10))
i=0
for  i  in  range(20):
        plt.subplot(4,5,i+1)
        plt.grid(False)
        img=load_img('drive/MyDrive/Lego/'+sample_df['path'].values[i])
        plt.imshow(img)
        #plt.axis("off")
        plt.xlabel(sample_df['minifigure_name'].values[i])
        i  +=  1
plt.show()
```



(5) 將資料分為訓練集、驗證集

Index 資料集中的一欄位已隨機將圖片標籤為訓練集、驗證集，其中訓練集佔 52%，驗證集佔 48%。

```
#將資料分為訓練集、驗證集
train_set  =  data_df[data_df["train-valid"]  ==  'train']
validation_set  =  data_df[data_df["train-valid"]  ==  'valid']
```

(6) 使用 OpenCV 讀取圖片時，默認的通道順序是 BGR，此處將圖像轉為 RGB；資料集中的原始圖片大小為(512*512)，後續將用 DenseNet121 演算法作為此研究模型，輸入層圖片大小須為(224*224)，故此處重新定義圖片大小，並將圖像像素標準化到 0 到 1 之間。

```
import  cv2
#Training  Data  Preprocessing
#Converted  the  pixels  of  the  image  data  to  array

train_Data  =  np.zeros((train_set.shape[0],  224,  224,  3))

for  i  in  range(train_set.shape[0]):

        image  =  cv2.imread('drive/MyDrive/Lego/'  +  train_set["path"].values[i])

        #Converting  BGR  to  RGB
        image  =  cv2.cvtColor(image,  cv2.COLOR_BGR2RGB)

        #Resizing  image
        image  =  cv2.resize(image,  (224,224))

        #Normalizing  pixel  values  to  [0,1]
        train_Data[i]  =  image  /  255.0

trainLabel  =  np.array(train_set["class_id"])-1
```

3. 模型建立

(1) 模型選擇

使用 Pre-trained Model DenseNet121 作為模型。

```
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

dense_net = tf.keras.applications.DenseNet121()
dense_net_layer=Dropout(0.5)(dense_net.layers[-2].output)
number_of_classes = len(data_df['class_id'].unique())
last_layer = Dense(number_of_classes, activation="sigmoid")(dense_net_layer)
model = Model(dense_net.input, last_layer)
```

(2) 參數設定

```
#優化器 Adam
model.compile(loss='sparse_categorical_crossentropy',
                            optimizer=Adam(0.0001),
                            metrics=['accuracy'])
```

(3) 儲存點

長時間運行的程序需要能夠中途保存，深度學習的保存點用來存取模型的權重，這樣一來可以再需要時繼續訓練模型，或是直接開始預測。

```
#儲存點
from tensorflow.keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint(filepath='model.h5', monitor="val_accuracy", save_best_only=True, verbose=1)
```

4. 訓練模型

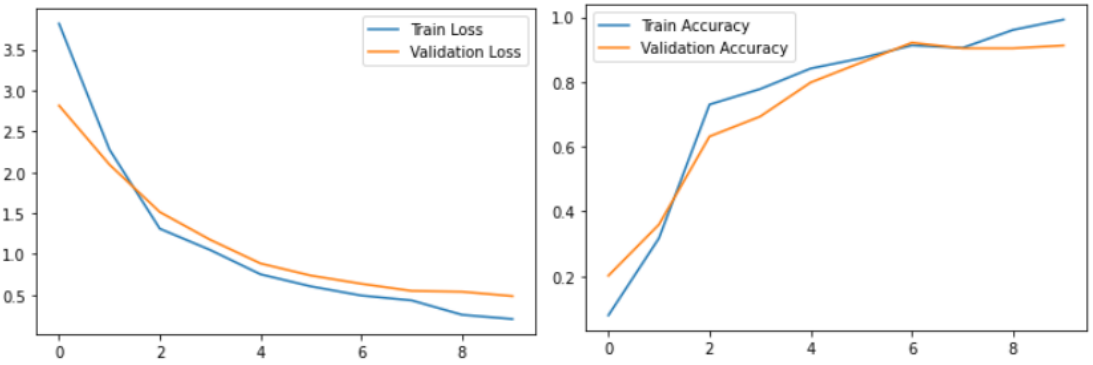
以 epoch 為 10；batch size 為 4 去訓練模型，並儲存訓練結果。

```
#模型訓練
hist=model.fit(
        train_Data,
        trainLabel,
        epochs=10,
        validation_data=(valid_Data, validLabel),
        shuffle=True,
        batch_size=4,
        callbacks=checkpoint
)
```

5. 驗證集驗證

繪製 Loss 及 Accuracy 圖表判斷訓練模型辨識的結果與真實值是否一致，由圖中可看出模型發生過擬合的問題。

```
print(hist.history.keys())
plt.plot(hist.history["loss"], label = "Train Loss")
plt.plot(hist.history["val_loss"], label = "Validation Loss")
plt.legend()
plt.show()

plt.figure()
plt.plot(hist.history["accuracy"], label = "Train Accuracy")
plt.plot(hist.history["val_accuracy"], label = "Validation Accuracy")
plt.legend()
plt.show()
```

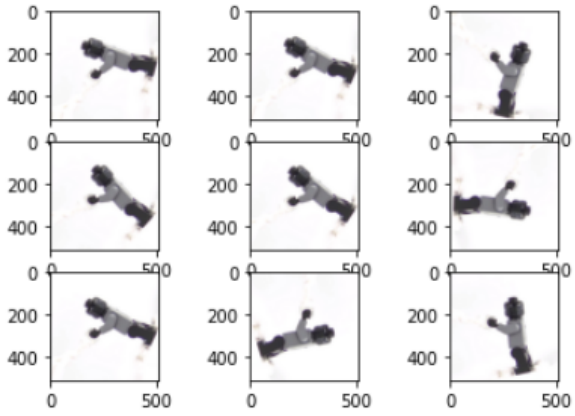


6. 改善過程

模型過擬合的問題可能是由於資料筆數僅有 240 張圖集，資料筆數少所導致，因此利用隨機轉動圖片角度，來增加照片數量。

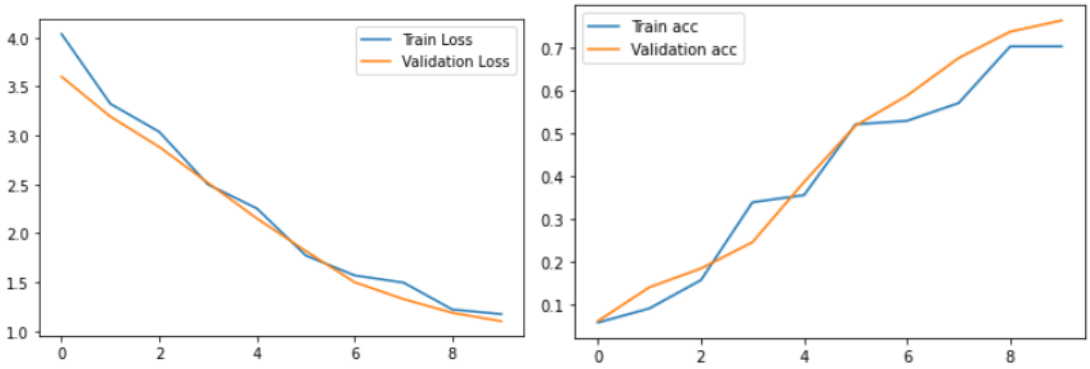
(1) Image Data Generator

```
img = load_img('drive/MyDrive/Lego/marvel/0007/008.jpg')

data = img_to_array(img)
samples = expand_dims(data, 0)
datagen = ImageDataGenerator(rotation_range=120)
it = datagen.flow(samples, batch_size=1)
for i in range(9):
        pyplot.subplot(330 + 1 + i)
        batch = it.next()
        image = batch[0].astype('uint8')
        pyplot.imshow(image)
pyplot.show()
```



改善結果

增加圖片張量後再次訓練模型，發現過擬合問題得到改善，但由於準確率稍嫌不足，於是下一步繼續透過參數的調整，提升模型的準確度。



(2) 調整參數

利用參數調整的方法，找尋最佳的參數組合提升模型的準確率。透過調整激活函數、學習率的方式，來比較原始模型。

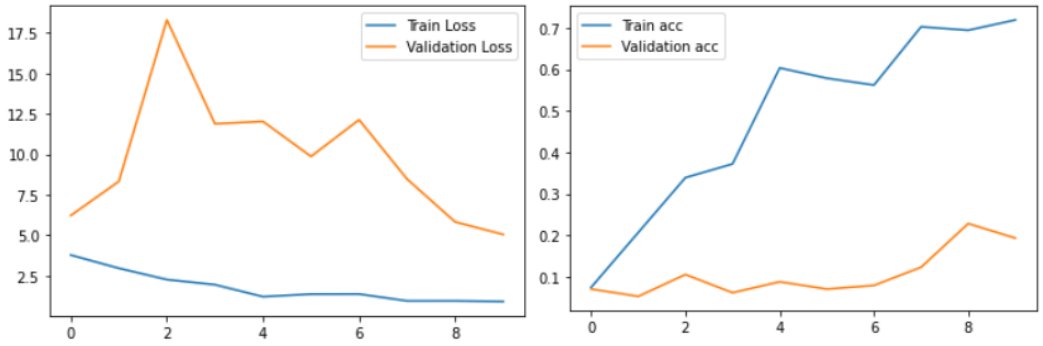
Model 1 的設置如下

	Last layer Dense activation	Learning Rate	Optimizer	Epoch	Accuracy
Model 1	sigmoid	0.0001	Adam	10	0.7632

Model 2 的設置如下

	Last layer Dense activation	Learning Rate	Optimizer	Epoch	Accuracy
Model 2	sigmoid	0.001	Adam	10	0.1930

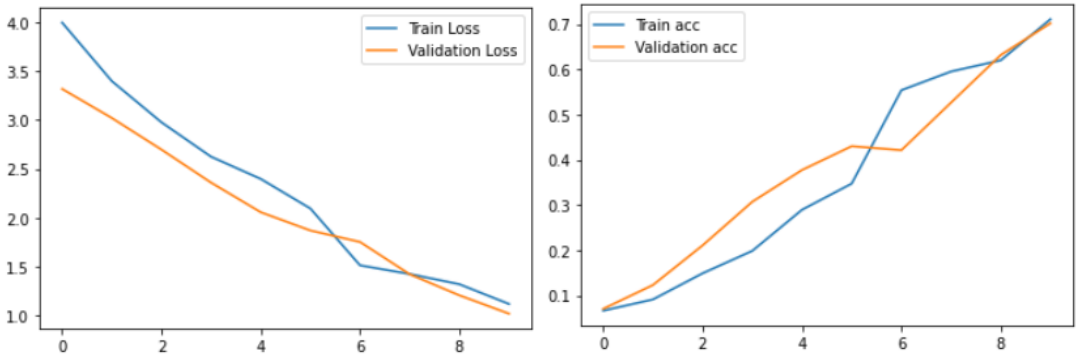
```
Epoch 1/10
9/9 [==============================] - 115s 13s/step - loss: 4.2077 - accuracy: 0.0318 - val_loss: 3.5997 - val_accuracy: 0.0614

Epoch 00001: val_accuracy improved from -inf to 0.06140, saving model to model1.h5
Epoch 2/10
9/9 [==============================] - 102s 11s/step - loss: 3.3584 - accuracy: 0.0841 - val_loss: 3.1919 - val_accuracy: 0.1404

Epoch 00002: val_accuracy improved from 0.06140 to 0.14035, saving model to model1.h5
Epoch 3/10
9/9 [==============================] - 101s 13s/step - loss: 3.2365 - accuracy: 0.1318 - val_loss: 2.8819 - val_accuracy: 0.1842

Epoch 00003: val_accuracy improved from 0.14035 to 0.18421, saving model to model1.h5
Epoch 4/10
9/9 [==============================] - 104s 12s/step - loss: 2.5815 - accuracy: 0.2912 - val_loss: 2.5141 - val_accuracy: 0.2456

Epoch 00004: val_accuracy improved from 0.18421 to 0.24561, saving model to model1.h5
Epoch 5/10
9/9 [==============================] - 106s 13s/step - loss: 2.4448 - accuracy: 0.3239 - val_loss: 2.1510 - val_accuracy: 0.3860

Epoch 00005: val_accuracy improved from 0.24561 to 0.38596, saving model to model1.h5
Epoch 6/10
9/9 [==============================] - 105s 12s/step - loss: 1.7994 - accuracy: 0.5291 - val_loss: 1.8188 - val_accuracy: 0.5175

Epoch 00006: val_accuracy improved from 0.38596 to 0.51754, saving model to model1.h5
Epoch 7/10
9/9 [==============================] - 107s 12s/step - loss: 1.6322 - accuracy: 0.5463 - val_loss: 1.4994 - val_accuracy: 0.5877

Epoch 00007: val_accuracy improved from 0.51754 to 0.58772, saving model to model1.h5
Epoch 8/10
9/9 [==============================] - 103s 11s/step - loss: 1.4440 - accuracy: 0.5980 - val_loss: 1.3273 - val_accuracy: 0.6754

Epoch 00008: val_accuracy improved from 0.58772 to 0.67544, saving model to model1.h5
Epoch 9/10
9/9 [==============================] - 102s 11s/step - loss: 1.1690 - accuracy: 0.7160 - val_loss: 1.1874 - val_accuracy: 0.7368

Epoch 00009: val_accuracy improved from 0.67544 to 0.73684, saving model to model1.h5
Epoch 10/10
9/9 [==============================] - 108s 12s/step - loss: 1.2770 - accuracy: 0.6816 - val_loss: 1.1021 - val_accuracy: 0.7632

Epoch 00010: val_accuracy improved from 0.73684 to 0.76316, saving model to model1.h5
```



Model 3 的設置如下

	Last layer Dense activation	Learning Rate	Optimizer	Epoch	Accuracy
Model 3	softmax	0.0001	Adam	10	0.7018

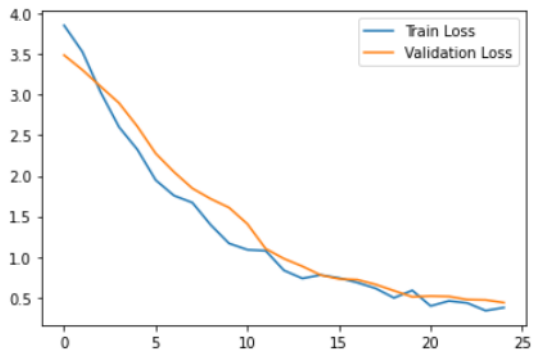
```
Epoch 1/10
9/9 [==============================] - 100s 11s/step - loss: 3.8834 - accuracy: 0.0301 - val_loss: 3.3156 - val_accuracy: 0.0702

Epoch 00001: val_accuracy improved from -inf to 0.07018, saving model to model10.h5
Epoch 2/10
9/9 [==============================] - 88s 10s/step - loss: 3.4830 - accuracy: 0.1155 - val_loss: 3.0194 - val_accuracy: 0.1228

Epoch 00002: val_accuracy improved from 0.07018 to 0.12281, saving model to model10.h5
Epoch 3/10
9/9 [==============================] - 86s 10s/step - loss: 3.0184 - accuracy: 0.1224 - val_loss: 2.6970 - val_accuracy: 0.2105

Epoch 00003: val_accuracy improved from 0.12281 to 0.21053, saving model to model10.h5
Epoch 4/10
9/9 [==============================] - 85s 9s/step - loss: 2.7646 - accuracy: 0.1229 - val_loss: 2.3597 - val_accuracy: 0.3070

Epoch 00004: val_accuracy improved from 0.21053 to 0.30702, saving model to model10.h5
Epoch 5/10
9/9 [==============================] - 85s 9s/step - loss: 2.5371 - accuracy: 0.2823 - val_loss: 2.0578 - val_accuracy: 0.3772

Epoch 00005: val_accuracy improved from 0.30702 to 0.37719, saving model to model10.h5
Epoch 6/10
9/9 [==============================] - 84s 9s/step - loss: 2.0991 - accuracy: 0.3371 - val_loss: 1.8712 - val_accuracy: 0.4298

Epoch 00006: val_accuracy improved from 0.37719 to 0.42982, saving model to model10.h5
Epoch 7/10
9/9 [==============================] - 85s 9s/step - loss: 1.6159 - accuracy: 0.5386 - val_loss: 1.7542 - val_accuracy: 0.4211

Epoch 00007: val_accuracy did not improve from 0.42982
Epoch 8/10
9/9 [==============================] - 84s 9s/step - loss: 1.4979 - accuracy: 0.5697 - val_loss: 1.4247 - val_accuracy: 0.5263

Epoch 00008: val_accuracy improved from 0.42982 to 0.52632, saving model to model10.h5
Epoch 9/10
9/9 [==============================] - 85s 9s/step - loss: 1.3629 - accuracy: 0.5857 - val_loss: 1.2098 - val_accuracy: 0.6316

Epoch 00009: val_accuracy improved from 0.52632 to 0.63158, saving model to model10.h5
Epoch 10/10
9/9 [==============================] - 84s 9s/step - loss: 1.0772 - accuracy: 0.7069 - val_loss: 1.0226 - val_accuracy: 0.7018

Epoch 00010: val_accuracy improved from 0.63158 to 0.70175, saving model to model10.h5
```

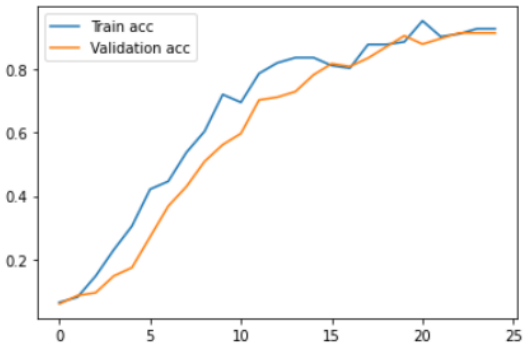


Model 4 的設置如下

	Last layer Dense activation	Learning Rate	Optimizer	Epoch	Accuracy
Model 4	softmax	0.0001	Adam	25	0.9123

```
Epoch 00018: val_accuracy improved from 0.81579 to 0.83333, saving model to model0.h5
Epoch 19/25
9/9 [==============================] - 75s 8s/step - loss: 0.4926 - accuracy: 0.8716 - val_loss: 0.5810 - val_accuracy: 0.8684

Epoch 00019: val_accuracy improved from 0.83333 to 0.86842, saving model to model0.h5
Epoch 20/25
9/9 [==============================] - 76s 8s/step - loss: 0.5940 - accuracy: 0.8772 - val_loss: 0.5053 - val_accuracy: 0.9035

Epoch 00020: val_accuracy improved from 0.86842 to 0.90351, saving model to model0.h5
Epoch 21/25
9/9 [==============================] - 76s 8s/step - loss: 0.4537 - accuracy: 0.9242 - val_loss: 0.5155 - val_accuracy: 0.8772

Epoch 00021: val_accuracy did not improve from 0.90351
Epoch 22/25
9/9 [==============================] - 76s 8s/step - loss: 0.5341 - accuracy: 0.8605 - val_loss: 0.5102 - val_accuracy: 0.8947

Epoch 00022: val_accuracy did not improve from 0.90351
Epoch 23/25
9/9 [==============================] - 75s 8s/step - loss: 0.3640 - accuracy: 0.9263 - val_loss: 0.4715 - val_accuracy: 0.9123

Epoch 00023: val_accuracy improved from 0.90351 to 0.91228, saving model to model0.h5
Epoch 24/25
9/9 [==============================] - 76s 9s/step - loss: 0.7858 - accuracy: 0.8056 - val_loss: 0.4669 - val_accuracy: 0.9123

Epoch 00024: val_accuracy did not improve from 0.91228
Epoch 25/25
9/9 [==============================] - 76s 8s/step - loss: 0.3192 - accuracy: 0.9408 - val_loss: 0.4336 - val_accuracy: 0.9123

Epoch 00025: val_accuracy did not improve from 0.91228
```





三、研究結果

Model	Last layer Dense activation	Learning Rate	Epoch	Accuracy
1	sigmoid	0.0001	10	0.7632
2	sigmoid	0.001	10	0.1930
3	softmax	0.0001	10	0.7018
4	softmax	0.0001	25	0.9123

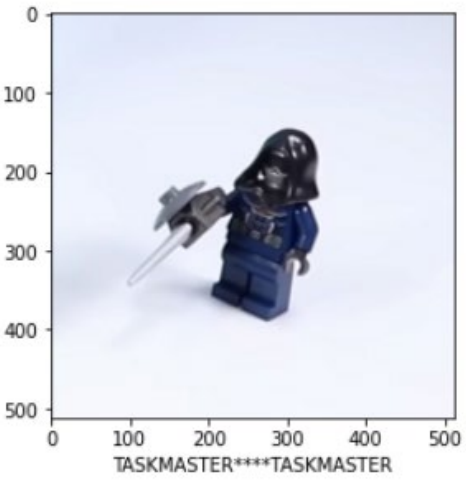
整理出上述結果可以看出，Model 2 更動學習率後，準確度不但沒有提升反而大幅下降，因此提高學習率此方法將不適用於此模型；Model 3 的學習率採用原模型的 0.0001，調整最後一層全連接層激活函數為 softmax 發現準確率提升、接近輸出層的過擬合情形也明顯收斂；Model 4 調整最後一層全連接層激活函數為 softmax、將 Epoch 增加到 25 次，結果顯示增加 Epoch 可以顯著的提升模型的準確率。

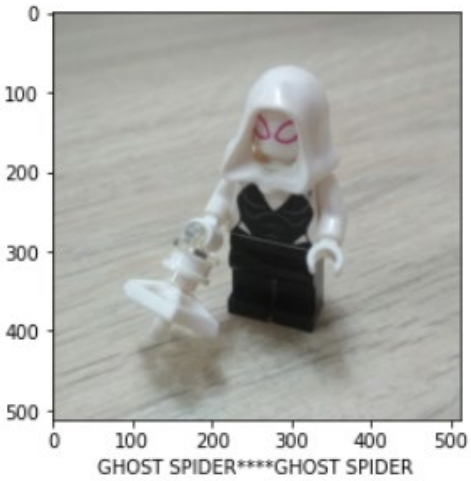
上述參數調整比較模型得知，使用後一層全連接層激活函數為 softmax、學習率為 0.0001 在三個模型中具有最高的準確率為 91.23%。 因此可利用 Model 4 來進行新圖片的辨識，於下圖顯示辨識結果。

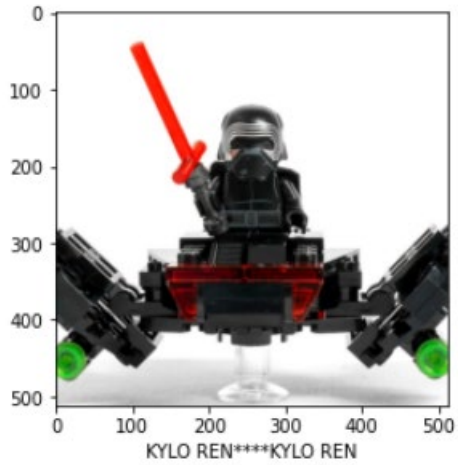
```
final_result = pd.merge(test_df[['minifigure_name','path']], results[['path', 'Predictions']], on='path')
final_result
```

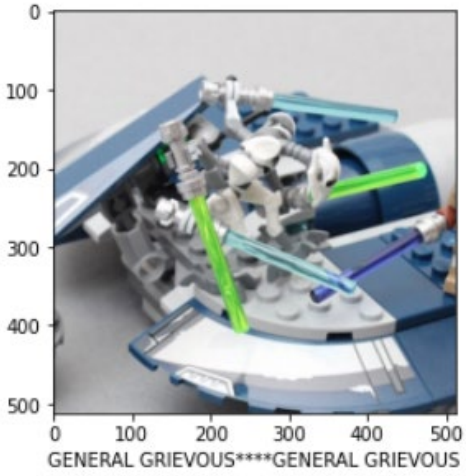
	minifigure_name	path	Predictions
0	TASKMASTER	marvel/0010/014.jpg	TASKMASTER
1	GHOST SPIDER	marvel/0004/007.jpg	GHOST SPIDER
2	KYLO REN	star-wars/0006/003.jpg	KYLO REN
3	GENERAL GRIEVOUS	star-wars/0005/005.jpg	GENERAL GRIEVOUS
4	MACE WINDU	star-wars/0004/012.jpg	MACE WINDU

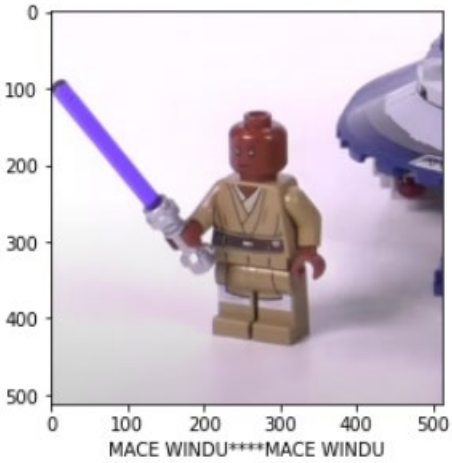
```
for i in range(5):

    image = cv2.imread('drive/MyDrive/Lego/'+final_result['path'].values[i])
    image = cv2.resize(image, dsize=(512,512))
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)/255
    plt.imshow(image)
    plt.xlabel(final_result['minifigure_name'].values[i]+'****'+final_result['Predictions'].values[i])
    plt.show()
```











由測試集預測結果跟真實資料比對，模型皆做出正確的預測。

四、結論

1. 限制

研究過程中遇到的最大瓶頸及困難是程式語言能力的缺乏，需要在更進一步精進，此外這次所採用的數據資料只有 240 筆，雖然已經透過 Image Data Generator 去做改善，但未來仍可增加更多數據集讓模型訓練結果更具可靠性。礙於時間關係，嘗試參數的調整並不夠完善，未來可以利用實驗設計，將所有的參數組合去做訓練，找出最佳的模型。

2. 未來展望

未來可以持續優化辨識模型，結合 LEGO 手機 APP 實現 Real-Time 辨識系統，除了正確辨別出樂高人物角色以外，還能連結 LEGO 官方網站的角色背景來源，在 APP 上快速的連結各角色的故事背景集相關資料。

五、參考文獻

DenseNet-121 https://www.kaggle.com/pytorch/densenet121 、https://medium.com/%E5%AD%B8%E4%BB%A5%E5%BB%A3%E6%89%8D/dense-cnn-%E5%AD%B8%E7%BF%92%E5%BF%83%E5%BE%97-%E6%8C%81%E7%BA%8C%E6%9B%B4%E6%96%B0-8cd8c65a6f3f

Lego Minifigures Dataset https://www.kaggle.com/ihelon/lego-minifigures-classification

Lego Transfer-CNN Classification https://www.kaggle.com/drfrank/lego-transfer-cnn-classification

Tensorflow 儲存模型參數 https://ithelp.ithome.com.tw/articles/10187786

Using Pre-Trained Models https://keras.rstudio.com/articles/applications.html

Geoff Pleiss et al. Memory-Efficient Implementation of DenseNets, Cornell University, 2017 https://arxiv.org/abs/1707.06990