使用CNN辨識LEGO圖像

109034553 莊婉琦

Contents

01

研究主題

主題介紹

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。

5W1H

Why

LEGO更新速度快，透過LEGO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

Who

LEGO顧客、
LEGO店員工

When

選購樂高時

Where

樂高販賣店

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

02

研究方法

研究方法-6步驟

Step 1
資料蒐集

Step 2
資料前處理

Step 3
模型建立

Step 4
訓練模型

Step 5
驗證集驗證

Step 6
改善過程

Step 1 資料蒐集

由Kaggle公開數據集中取得「Lego-Minifigure」資料集
分為三個系列，每個系列中又包含數個角色的資料夾，各角色影像圖片包含不同角度、背景、明亮度
共計240張圖片

Step 1 資料蒐集

Csv檔分別儲存圖片路徑、分類編號、樂高編號、盒組系列名稱、角色名稱

```
#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

Step 2 資料前處理

確認資料名稱及屬性一致後，進行資料的水平合併

```
#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

結果顯示資料並無缺值

```
#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
```

Step 2 資料前處理

將資料分為訓練集(52%)、驗證集(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']
```

```
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
```

Step 3 模型建立

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="softmax")(dense_net_layer)
model = Model(dense_net.input, last_layer)
```

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

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

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

Model: "model_20"

Layer (type)	Output Shape	Param #	Connected to
input_18 (InputLayer)	[(None, 224, 224, 3)	0	
zero_padding2d_34 (ZeroPadding2	(None, 230, 230, 3)	0	input_18[0][0]
conv1/conv (Conv2D)	(None, 112, 112, 64)	9408	zero_padding2d_34[0][0]
conv1/bn (BatchNormalization)	(None, 112, 112, 64)	256	conv1/conv[0][0]
conv1/relu (Activation)	(None, 112, 112, 64)	0	conv1/bn[0][0]
zero_padding2d_35 (ZeroPadding2	(None, 114, 114, 64)	0	conv1/relu[0][0]
pool1 (MaxPooling2D)	(None, 56, 56, 64)	0	zero_padding2d_35[0][0]
conv2_block1_0_bn (BatchNormali	(None, 56, 56, 64)	256	pool1[0][0]
conv2_block1_0_relu (Activation	(None, 56, 56, 64)	0	conv2_block1_0_bn[0][0]
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 128)	8192	conv2_block1_0_relu[0][0]
conv2_block1_1_bn (BatchNormali	(None, 56, 56, 128)	512	conv2_block1_1_conv[0][0]
conv2_block1_1_relu (Activation	(None, 56, 56, 128)	0	conv2_block1_1_bn[0][0]
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 32)	36864	conv2_block1_1_relu[0][0]
conv2_block1_concat (Concatenat	(None, 56, 56, 96)	0	pool1[0][0] conv2_block1_2_conv[0][0]

Step 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

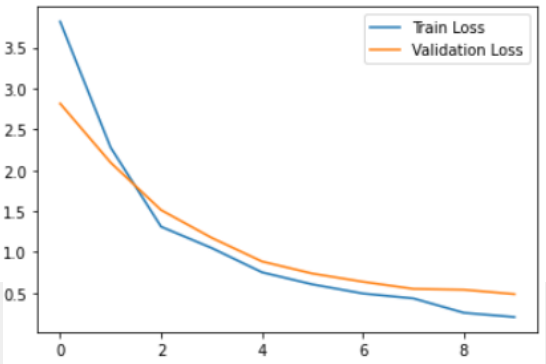
```
#模型訓練
hist=model.fit(
        train_Data,
        trainLabel,
        epochs=10,
        validation_data=(valid_Data,  validLabel),
        shuffle=True,
        batch_size=4,
        callbacks=checkpoint
)
```

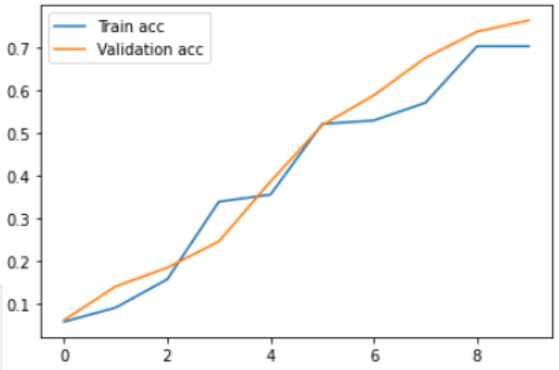
```
Epoch 1/10
32/32 [==============================] - 119s 3s/step - loss: 3.7883 - accuracy: 0.0484 - val_loss: 2.8153 - val_accuracy: 0.2018

Epoch 00001: val_accuracy improved from -inf to 0.20175, saving model to model.h5
Epoch 2/10
32/32 [==============================] - 105s 3s/step - loss: 2.3197 - accuracy: 0.2979 - val_loss: 2.0940 - val_accuracy: 0.3596

Epoch 00002: val_accuracy improved from 0.20175 to 0.35965, saving model to model.h5
Epoch 3/10
32/32 [==============================] - 103s 3s/step - loss: 1.3081 - accuracy: 0.7553 - val_loss: 1.5132 - val_accuracy: 0.6316

Epoch 00003: val_accuracy improved from 0.35965 to 0.63158, saving model to model.h5
Epoch 4/10
32/32 [==============================] - 104s 3s/step - loss: 1.0541 - accuracy: 0.8207 - val_loss: 1.1745 - val_accuracy: 0.6930

Epoch 00004: val_accuracy improved from 0.63158 to 0.69298, saving model to model.h5
Epoch 5/10
32/32 [==============================] - 107s 3s/step - loss: 0.7741 - accuracy: 0.7837 - val_loss: 0.8843 - val_accuracy: 0.7982

Epoch 00005: val_accuracy improved from 0.69298 to 0.79825, saving model to model.h5
Epoch 6/10
32/32 [==============================] - 106s 3s/step - loss: 0.5472 - accuracy: 0.9218 - val_loss: 0.7365 - val_accuracy: 0.8596

Epoch 00006: val_accuracy improved from 0.79825 to 0.85965, saving model to model.h5
Epoch 7/10
32/32 [==============================] - 103s 3s/step - loss: 0.4118 - accuracy: 0.9325 - val_loss: 0.6363 - val_accuracy: 0.9211

Epoch 00007: val_accuracy improved from 0.85965 to 0.92105, saving model to model.h5
Epoch 8/10
32/32 [==============================] - 105s 3s/step - loss: 0.4096 - accuracy: 0.9267 - val_loss: 0.5492 - val_accuracy: 0.9035

Epoch 00008: val_accuracy did not improve from 0.92105
Epoch 9/10
32/32 [==============================] - 102s 3s/step - loss: 0.2434 - accuracy: 0.9639 - val_loss: 0.5389 - val_accuracy: 0.9035
```

Step 5 驗證集驗證

繪製Loss及Accuracy圖表判斷發現模型出現過擬合的情形

```
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()
```

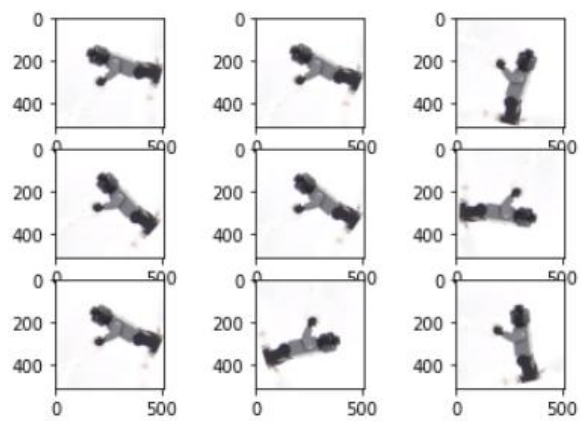




Step 6 改善過程

Image Data Generator-利用隨機轉動圖片角度，來增加照片數量

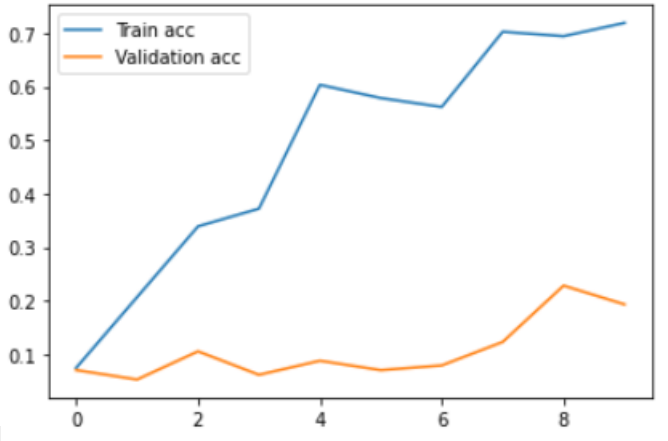
```
img = load_img('drive/MyDrive/Lego/marvel/0007/008.jpg')

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()
```



Step 6 改善過程

調整參數- 激活函數、學習率、Epoch

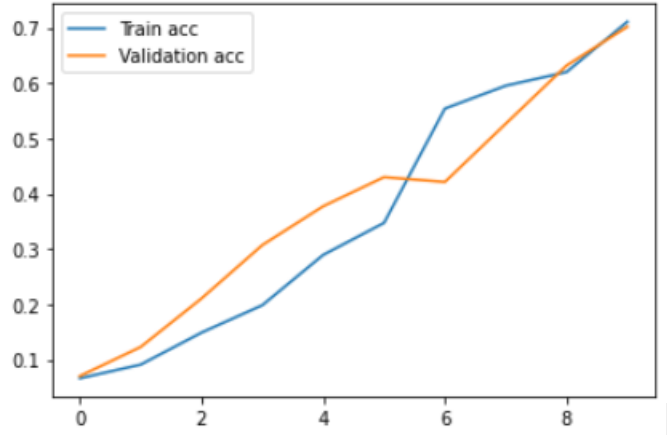
	Last layer Dense activation	Learning Rate	Optimizer	Epoch	Accuracy
Model 2	sigmoid	0.001	Adam	10	0.1930



Step 6 改善過程

調整參數- 激活函數、學習率、Epoch

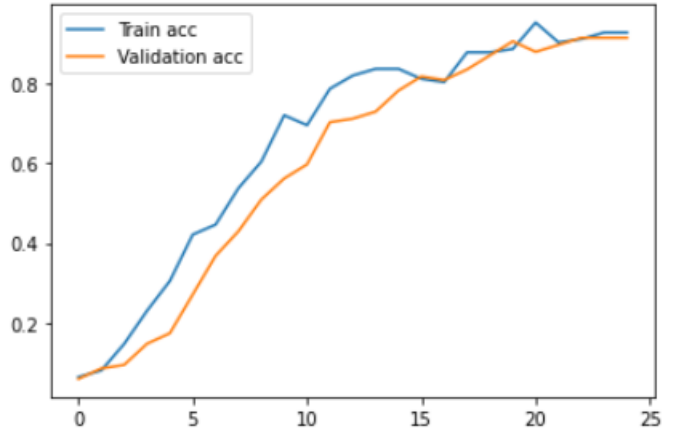
	Last layer Dense activation	Learning Rate	Optimizer	Epoch	Accuracy
Model 3	softmax	0.0001	Adam	10	0.7018



Step 6 改善過程

調整參數- 激活函數、學習率、Epoch

	Last layer Dense activation	Learning Rate	Optimizer	Epoch	Accuracy
Model 4	softmax	0.0001	Adam	25	0.9213



03

研究結果

研究結果

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%。

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

04

結論

結論

研究限制

- 數據資料筆數少，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。
- 參數調整不夠完善，未來可利用實驗設計，將所有的參數組合去做訓練，找出最佳的模型。

未來展望

- 持續優化辨識模型、結合LEGO手機APP實現即時辨識，快速的連結各角色的故事背景集相關資料。

Thank You