CNN of Malaria Cell

用CNN來辨識瘧疾細胞

鄧宏宇

主題說明-問題描述

- 瘧疾介紹

- 檢查方式

  1.血液抹片鏡檢

  2.特定抗原快速篩檢

- 鏡檢的兩大問題：

  1.偏鄉沒有相應器材

  2.準確度跟經驗有關

- 哪些地方最多?

主題說明-5W1H

- WHY-瘧疾細胞在醫療資源落後地區特別猖獗
- WHAT-建立一個辨識系統自動辨識瘧疾細胞
- WHERE-世界各地，尤其是非洲國家
- WHEN-進行診斷時
- WHO-醫生或相關醫事人員
- HOW-收集感染與非感染的細胞照片，利用CNN來訓練辨識這些細胞

執行過程

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

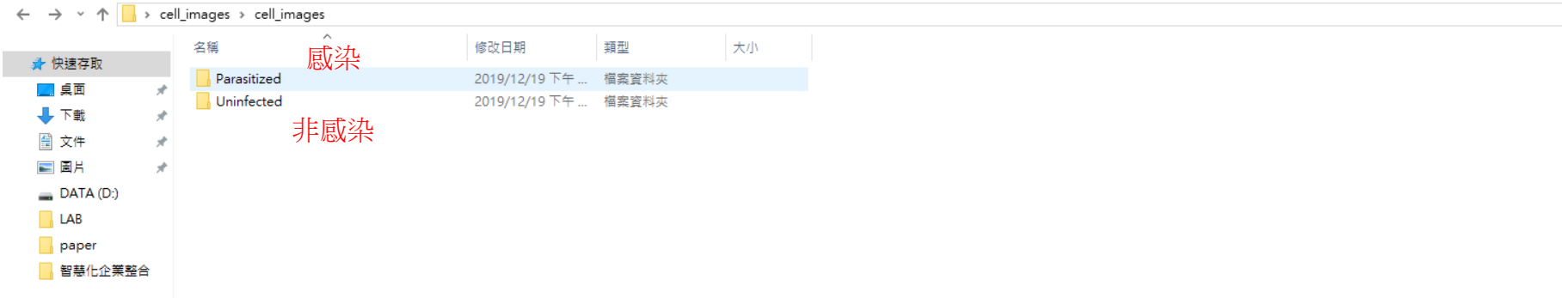
02 **Data Pre-processing**
資料前處理

03 **Classifier Training**
建立模型及改善

04 **Accuracy Testing**
得出改善方案
最終準確率

資料整理-前處理

- 由Kaggle公開數據取得資料。
- 共分成2種類別，0號類代表感染瘧疾細胞的圖片集，1號類別代表未感染瘧疾細胞的圖片集。
- 每一類別包含13780張照片，總共27560張照片。



- 載入資料，將每張圖片大小的統一，使每張圖片大小設為64x64的圖像

```
In [2]: #載入數據，並且查看兩種可看的數據類型，將尺寸設定為64*64
        DATA_DIR = 'C:/Users/kevin/Desktop/cell_images/cell_images/'
        SIZE=64
        dataset = []
        label = []
```

資料整理-前處理

- 將檢視兩個資料集(感染、非感染)分別加上標籤0和1，並且統一轉為RGB模式，同時若出現問題也讓他判斷是哪張圖片發生的
- 檢閱感染細胞資料集，並標籤為”0”

In [3]:
```
#檢閱感染細胞圖像，並且檢查是否為png檔，大小設定為64*64，放入矩陣中並標籤為0
#如果過程有問題可以標示出是哪張出了問題
parasitized_images = os.listdir(DATA_DIR + 'Parasitized/')
for i, image_name in enumerate(parasitized_images):
    try:
        if (image_name.split('.')[1] == 'png'):
            image = cv2.imread(DATA_DIR + 'Parasitized/' + image_name)
            image = Image.fromarray(image, 'RGB')
            image = image.resize((SIZE, SIZE))
            dataset.append(np.array(image))
            label.append(0)
    except Exception:
        print("Could not read image {} with name {}".format(i, image_name))
```

資料整理-前處理

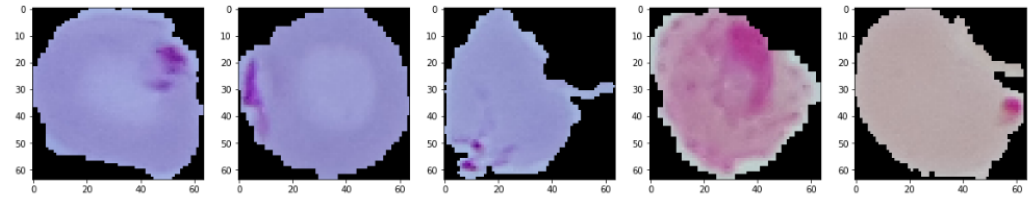
- 將檢視兩個資料集(感染、非感染)分別加上標籤0和1，並且統一轉為RGB模式，同時若出現問題也讓他判斷是哪張圖片發生的
- 檢閱非感染細胞資料集，並標籤為”1”

In [8]:
```
#檢閱未感染細胞圖像，並且檢查是否為png檔，大小設定為64*64，放入矩陣中並標籤為1
#如果過程有問題可以標示出是哪張出了問題
uninfected_images = os.listdir(DATA_DIR + 'Uninfected/')
for i, image_name in enumerate(uninfected_images):
    try:
        if (image_name.split('.')[1] == 'png'):
            image = cv2.imread(DATA_DIR + 'Uninfected/' + image_name)
            image = Image.fromarray(image, 'RGB')
            image = image.resize((SIZE, SIZE))
            dataset.append(np.array(image))
            label.append(1)
    except Exception:
        print("Could not read image {} with name {}".format(i, image_name))
```

資料整理-視覺化資料

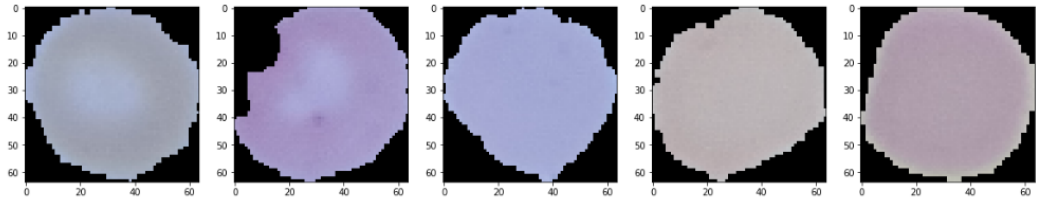
- 隨機抽取5張感染瘧疾細胞的圖片來檢視

```
In [12]: #隨機抽取五張看一下感染的細胞
plt.figure(figsize = (20, 20))
for index, image_index in enumerate(np.random.randint(len(parasitized_images), size = 5)):
    plt.subplot(1, 5, index+1)
    plt.imshow(dataset[image_index])
```



- 隨機抽取5張非感染瘧疾細胞的圖片來檢視

```
In [17]: #隨機抽取五張看一下未感染的細胞
plt.figure(figsize = (20, 20))
for index, image_index in enumerate(np.random.randint(len(uninfected_images), size = 5)):
    plt.subplot(1, 5, index+1)
    plt.imshow(dataset[len(parasitized_images) + image_index])
```



模型設定

損失函數	卷積、池化	激活函數	最後一層激活函數	優化器
binary	1C1P1D	relu	sigmoid	sgd

In [50]:
```
classifier = None
classifier = Sequential()

classifier.add(Convolution2D(32, (3, 3), input_shape = (SIZE, SIZE, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2), data_format="channels_last"))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Flatten())

classifier.add(Dense(activation = 'relu', units=512))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Dense(activation = 'relu', units=256))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Dense(activation = 'sigmoid', units=2))
classifier.compile(optimizer = 'sgd', loss = 'binary_crossentropy', metrics = ['accuracy'])
print(classifier.summary())
```

原本模型結果

In [58]:
```
#訓練集8成，測試集2成
from keras.utils import to_categorical

X_train, X_test, y_train, y_test = train_test_split(dataset, to_categorical(np.array(label)), test_size = 0.20, random_state = 0
```

In [59]:
```
#開始訓練
history = classifier.fit(np.array(X_train),
                         y_train,
                         batch_size = 64,
                         verbose = 2,
                         epochs = 10,
                         validation_split = 0.1,
                         shuffle = False)
```

```
Train on 19841 samples, validate on 2205 samples
Epoch 1/10
 - 98s - loss: 0.6999 - accuracy: 0.6331 - val_loss: 0.7395 - val_accuracy: 0.6417
Epoch 2/10
 - 100s - loss: 0.5229 - accuracy: 0.7407 - val_loss: 0.5151 - val_accuracy: 0.7358
Epoch 3/10
 - 99s - loss: 0.4088 - accuracy: 0.8196 - val_loss: 0.4728 - val_accuracy: 0.7574
Epoch 4/10
 - 97s - loss: 0.3146 - accuracy: 0.8711 - val_loss: 0.3389 - val_accuracy: 0.8565
Epoch 5/10
 - 98s - loss: 0.2547 - accuracy: 0.9019 - val_loss: 0.3100 - val_accuracy: 0.8746
Epoch 6/10
 - 97s - loss: 0.2079 - accuracy: 0.9216 - val_loss: 0.2905 - val_accuracy: 0.8837
Epoch 7/10
 - 98s - loss: 0.2327 - accuracy: 0.9081 - val_loss: 1.0720 - val_accuracy: 0.6946
Epoch 8/10
 - 98s - loss: 0.1829 - accuracy: 0.9317 - val_loss: 0.4363 - val_accuracy: 0.8390
Epoch 9/10
 - 97s - loss: 0.1581 - accuracy: 0.9398 - val_loss: 0.3083 - val_accuracy: 0.8710
Epoch 10/10
 - 108s - loss: 0.1316 - accuracy: 0.9517 - val_loss: 0.3485 - val_accuracy: 0.8535
```

In [60]:
```
#計算測試集的準確率
print("Test_Accuracy: {:.2f}%".format(classifier.evaluate(np.array(X_test), np.array(y_test))[1]*100))
```

```
5512/5512 [==============================] - 8s 1ms/step
Test_Accuracy: 84.00%
```

訓練集8成，測試集2成
Epoch=10
Batch-size=64
訓練花費時間為990秒
測試花費時間為8秒

訓練集準確度：95.17%
驗證集準確度：85.35%

測試集準確度：84.00%

改善方法

- 方法1-Image Data Generator
- 方法2-多增加層數
- 方法3-合併方法1和方法2

改善方法1- Image Data Generator

- 利用隨機縮放最大幅度、隨機水平翻轉、隨機轉動角度來增加照片數量

In [54]:
```
from keras.preprocessing.image import ImageDataGenerator

train_generator = ImageDataGenerator(rescale = 1/255,#標準化
                                     zoom_range = [0.5,1.5],#隨機縮放最大幅度
                                     horizontal_flip = True,#隨機水平翻轉
                                     rotation_range = 30)#隨機轉動角度

test_generator = ImageDataGenerator(rescale = 1/255)#測試集不用增強

train_generator = train_generator.flow(np.array(X_train),
                                       y_train,
                                       batch_size = 64,
                                       shuffle = False)

test_generator = test_generator.flow(np.array(X_test),
                                     y_test,
                                     batch_size = 64,
                                     shuffle = False)
```

改善方法1- Image Data Generator

- 結果

```
In [61]: history = classifier.fit_generator(train_generator,
                                            steps_per_epoch = len(X_train)/64,
                                            epochs = 10,
                                            shuffle = False)

Epoch 1/10
345/344 [==============================] - 132s 382ms/step - loss: 0.4936 - accuracy: 0.7898
Epoch 2/10
345/344 [==============================] - 122s 354ms/step - loss: 0.3743 - accuracy: 0.8495
Epoch 3/10
345/344 [==============================] - 125s 361ms/step - loss: 0.3330 - accuracy: 0.8693
Epoch 4/10
345/344 [==============================] - 116s 336ms/step - loss: 0.3150 - accuracy: 0.8804
Epoch 5/10
345/344 [==============================] - 113s 326ms/step - loss: 0.3010 - accuracy: 0.8847
Epoch 6/10
345/344 [==============================] - 122s 355ms/step - loss: 0.2912 - accuracy: 0.8877
Epoch 7/10
345/344 [==============================] - 131s 379ms/step - loss: 0.2882 - accuracy: 0.8904
Epoch 8/10
345/344 [==============================] - 126s 366ms/step - loss: 0.2814 - accuracy: 0.8947
Epoch 9/10
345/344 [==============================] - 120s 349ms/step - loss: 0.2792 - accuracy: 0.8900
Epoch 10/10
345/344 [==============================] - 126s 364ms/step - loss: 0.2763 - accuracy: 0.8950
```

訓練集8成，測試集2成
Epoch=10
Batch-size=64
訓練花費時間為1233秒
測試花費時間為437秒

訓練集準確度：89.5%

測試集準確度：94.38%
比原本84%增加了10.8%

```
In [62]: print("Test_Accuracy(after augmentation): {:.2f}%".format(classifier.evaluate_generator(test_generator, steps = len(X_test), verb

5512/5512 [==============================] - 437s 79ms/step
Test_Accuracy(after augmentation): 94.38%
```

改善方法2-多增加層數

- 多增加1層Convolution2D、MaxPooling2D、Dropout，變成2C2P2D

In [73]:
```
classifier = None
classifier = Sequential()

classifier.add(Convolution2D(32, (3, 3), input_shape = (SIZE, SIZE, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2), data_format="channels_last"))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Convolution2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2), data_format="channels_last"))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Flatten())

classifier.add(Dense(activation = 'relu', units=512))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Dense(activation = 'relu', units=256))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Dense(activation = 'sigmoid', units=2))
classifier.compile(optimizer = 'sgd', loss = 'binary_crossentropy', metrics = ['accuracy'])
print(classifier.summary())
```

新增的層

改善方法2-多增加層數

- 結果

```
In [79]: #開始訓練
history = classifier.fit(np.array(X_train),
                         y_train,
                         batch_size = 64,
                         verbose = 2,
                         epochs = 10,
                         validation_split = 0.1,
                         shuffle = False)
```

```
Train on 19841 samples, validate on 2205 samples
Epoch 1/10
 - 98s - loss: 0.1545 - accuracy: 0.9437 - val_loss: 0.5410 - val_accuracy: 0.8810
Epoch 2/10
 - 93s - loss: 0.1492 - accuracy: 0.9467 - val_loss: 0.5589 - val_accuracy: 0.8791
Epoch 3/10
 - 100s - loss: 0.1444 - accuracy: 0.9491 - val_loss: 0.6446 - val_accuracy: 0.8612
Epoch 4/10
 - 100s - loss: 0.1402 - accuracy: 0.9502 - val_loss: 0.7563 - val_accuracy: 0.8463
Epoch 5/10
 - 103s - loss: 0.1337 - accuracy: 0.9518 - val_loss: 0.5108 - val_accuracy: 0.8925
Epoch 6/10
 - 100s - loss: 0.1308 - accuracy: 0.9526 - val_loss: 0.3945 - val_accuracy: 0.9002
Epoch 7/10
 - 102s - loss: 0.1285 - accuracy: 0.9535 - val_loss: 0.4826 - val_accuracy: 0.8973
Epoch 8/10
 - 103s - loss: 0.1229 - accuracy: 0.9552 - val_loss: 0.6668 - val_accuracy: 0.8735
Epoch 9/10
 - 95s - loss: 0.1160 - accuracy: 0.9573 - val_loss: 0.4492 - val_accuracy: 0.9036
Epoch 10/10
 - 93s - loss: 0.1125 - accuracy: 0.9588 - val_loss: 0.6164 - val_accuracy: 0.8844
```

訓練集8成，測試集2成
Epoch=10
Batch-size=64
訓練花費時間為987秒
測試花費時間為7秒

訓練集準確度：88.44%

測試集準確度：87.38%
比原本84%增加了3.38%

```
In [80]: #計算測試集的準確率
print("Test_Accuracy: {:.2f}%".format(classifier.evaluate(np.array(X_test), np.array(y_test))[1]*100))
```

```
5512/5512 [==============================] - 7s 1ms/step
Test_Accuracy: 87.38%
```

改善方法3-合併方法1方法2

- 多增加層數也使用Image Data Generator

In [54]:
```
from keras.preprocessing.image import ImageDataGenerator

train_generator = ImageDataGenerator(rescale = 1/255,#標準化
                                     zoom_range = [0.5,1.5],#隨機縮放最大幅度
                                     horizontal_flip = True,#隨機水平翻轉
                                     rotation_range = 30)#隨機轉動角度

test_generator = ImageDataGenerator(rescale = 1/255)#測試集不用增強

train_generator = train_generator.flow(np.array(X_train),
                                       y_train,
                                       batch_size = 64,
                                       shuffle = False)

test_generator = test_generator.flow(np.array(X_test),
                                     y_test,
                                     batch_size = 64,
                                     shuffle = False)
```

In [73]:
```
classifier = None
classifier = Sequential()

classifier.add(Convolution2D(32, (3, 3), input_shape = (SIZE, SIZE, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2), data_format="channels_last"))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Convolution2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2), data_format="channels_last"))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Flatten())

classifier.add(Dense(activation = 'relu', units=512))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Dense(activation = 'relu', units=256))
classifier.add(BatchNormalization(axis = -1))
classifier.add(Dropout(0.2))

classifier.add(Dense(activation = 'sigmoid', units=2))
classifier.compile(optimizer = 'sgd', loss = 'binary_crossentropy', metrics = ['accuracy'])
print(classifier.summary())
```

改善方法3-合併方法1方法2

- 結果

```
In [82]: history = classifier.fit_generator(train_generator,
                                            steps_per_epoch = len(X_train)/64,
                                            epochs = 10,
                                            shuffle = False)

Epoch 1/10
345/344 [==============================] - 119s 344ms/step - loss: 0.3290 - accuracy: 0.8735
Epoch 2/10
345/344 [==============================] - 122s 355ms/step - loss: 0.2960 - accuracy: 0.8863
Epoch 3/10
345/344 [==============================] - 137s 396ms/step - loss: 0.2789 - accuracy: 0.8943
Epoch 4/10
345/344 [==============================] - 137s 396ms/step - loss: 0.2712 - accuracy: 0.8968
Epoch 5/10
345/344 [==============================] - 125s 364ms/step - loss: 0.2668 - accuracy: 0.8983
Epoch 6/10
345/344 [==============================] - 124s 359ms/step - loss: 0.2643 - accuracy: 0.8996
Epoch 7/10
345/344 [==============================] - 124s 360ms/step - loss: 0.2548 - accuracy: 0.9042
Epoch 8/10
345/344 [==============================] - 126s 365ms/step - loss: 0.2563 - accuracy: 0.9051
Epoch 9/10
345/344 [==============================] - 115s 333ms/step - loss: 0.2508 - accuracy: 0.9061
Epoch 10/10
345/344 [==============================] - 110s 319ms/step - loss: 0.2417 - accuracy: 0.9094
```

訓練集8成，測試集2成
Epoch=10
Batch-size=64
訓練花費時間為1239秒
測試花費時間為501秒

訓練集準確度：90.94%

測試集準確度：91.81%
比原本84%增加了7.81%

```
In [83]: print("Test_Accuracy(after augmentation): {:.2f}%".format(classifier.evaluate_generator(test_generator, steps = len(X_test), verb

5512/5512 [==============================] - 501s 91ms/step
Test_Accuracy(after augmentation): 91.81%
```

結論-方法比較

原本模型為84%準確率，以下為改善結果統整

	訓練時間(s)	測試時間(s)	訓練準確率	測試準確率	進步%數
方法1	1233	437	89.5%	94.38	10.38%
方法2	987	7	88.44%	87.38%	3.38%
方法3	1239	501	90.94%	91.81%	7.81%

- 使用Image Data Generator可以達到就好的準確率，但也必須花費更高的時間成本
- 增加層數雖然不會提高時間成本，但準確率提昇效果有限
- 兩種方案合併不但花費很多時間成本，也沒有單純使用Image Data Generator的準確率

結論-未來方向

- 瘧疾病原體分為五種
- 惡性瘧原蟲（*Plasmodium falciparum*）、間日瘧原蟲（*P. vivax*）、三日瘧原蟲（*P. malariae*）、卵形瘧原蟲（*P. ovale*）、諾氏瘧原蟲（*P. knowlesi*）

- 不同的病原體會導致不同的瘧疾種類，有些瘧疾種類同時擁有2種病原體
  (例如：日發虐的病原體為惡性瘧原蟲和間日瘧原蟲)

- 不但可以分辨出是否為瘧疾細胞，還能分辨出是何種瘧原蟲

THANKS FOR YOUR LISTENING