瘧疾細胞 CNN 辨識系統

108034550 鄧宏宇
工業工程與工程管理所，國立清華大學，台灣

摘要

瘧疾對全球的健康都有很大的威脅，而標準的診斷瘧疾方法是用顯微鏡目視血液細胞看是否有寄生蟲的感染，需要由專業的技術人員，這種方法效率低下，並且檢查成功機率依靠檢查者的本身的技術和經驗，具有較大的不確定性。

而本研究是使用 CNN 機器學習圖像辨識系統來自動辨識血液中細胞，來判斷是否具有較高的可靠性。由 kaggle 圖像資料庫取得所需的圖片資料，判斷的準確率達到 84%，接著使用圖片增強器（IDG）的方法和多增加層數這兩個方法來改善準確率，分別得到 94.38%和 87.38%的準確率，接著合併使用圖片增強器和增加層數的方法，得到 91.91%的準確率，所有方案中是使用圖片增強器會有更好的準確率高達 94.38%，但同時也會耗費更多時間成本。

關鍵字:瘧疾、CNN、機器學習、智慧醫療

目錄

一、主題介紹

在許多國家，蚊子不僅僅是煩人的昆蟲，更是傳播瘧疾的病媒。根據 WHO 的資料，2014 年就有 2.14 億人受瘧疾影響，其中 43 萬人甚至死亡[1]。

而篩檢瘧疾的方法因為很複雜，主要檢查方法有兩種，分別是 血液抹片 鏡檢或特定 抗原快速篩檢 。顯微鏡鏡檢是最常用的篩檢方式，光是 2010 年，全球就篩檢了約 1.65 億片血液抹片[2]。

鏡檢其中有最主要的兩項問題，一是許多偏鄉並沒有篩檢器材；其次，鏡檢的準確率極度依賴檢驗者的技術以及病人血液中瘧原蟲的數量。血液抹片鏡檢的 敏感度 大約在 75－90％之間，最低可能到 50％，而市面上的快速篩檢商品的精準度通常較直接鏡檢高，但其敏感度及精密度與製造廠商相關，且變動幅度極大。

雖然台灣在 50 年前就徹底的根除瘧疾，但在這世界的其他地方還是有許多國家深受虐疾之苦，尤其在非洲瘧疾更是猖獗，在 2012 年，估計發生了 2.07 億例瘧疾病例，造成大約 62.7 萬例瘧疾死亡。估計仍有 34 億人有感染瘧疾的風險，多數人在非洲和東南亞。大約 80%的瘧疾病例發生在非洲[3]。

而判斷瘧疾是醫生的一項靠專業和經驗的技能，在缺乏經驗的依據下容易出錯，而且若在醫療資源有限的地區要辨識是否患有瘧疾是一項耗時耗力的大工程，所以若能發展一套更快、更低成本、更有效的的自動辨識瘧疾細胞的系統， 將可以大大減少一是人員的負擔，也能有效得更快採取相應措施。

二、5W1H 和執行流程

2-1 5W1H

- Why

  辨識瘧疾細胞是項困難且需要有經驗的醫生才能做到，這尤其在醫療資源缺少的地方是一大問題，像是非洲是瘧疾最猖獗同時也是醫療資源最缺少的地方，因此建立一個 CNN 辨識出瘧疾細胞將會大幅度的改善醫療成果。

- What

  我們想建立一個瘧疾細胞辨識系統，讓沒有經驗的醫事人員也能透過這個辨識系統來分辨是否為瘧疾細胞，且比專業醫生評斷也更高的準確度。

- Where

  世界各地都可以使用，尤其是非洲等瘧疾嚴重醫療不足的地區。

- When

  需要辨識是否為瘧疾細胞時。

- Who

  需要進行瘧疾細胞辨識的醫生或相關醫事人員。

- How

  收集感染瘧疾細胞的圖片與非感染的，利用卷積神經網路訓練這些照片，判斷出是否患有瘧疾特徵。

三、資料整理

3-1 資料集照片來源

由 Kaggle 公開數據集中取得瘧疾細胞的圖像資料集，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，依類別分別分成 2 個資料夾。

3-2 資料前處理過程

(1) 將每張圖片大小的統一，將每張圖片大小設為 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 = []
```

(2) 完成圖片大小統一後，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，讓整體訓練能順利辨別每張圖片各自代表的意義。

- 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))
```

- 檢閱非感染細胞資料集，並標籤為”1”。

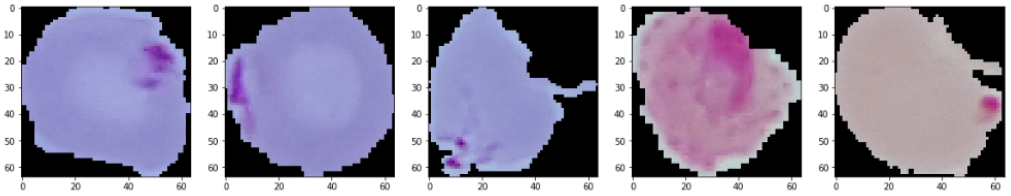
In [8]:
```
#檢閱未感染細胞圖像，並且檢查是否為png檔，大小設定為64*64，放入矩陣中並標籤為1
#如果過程有問題可以標示出是哪張出了問題
uninfected_images = os.listdir(DATA_DIR + 'Uninfected/')
for i, image_name in enumerate(uninfected_images):
    try:
        if (image_name.split('.')[1] == 'png'):
            image = cv2.imread(DATA_DIR + 'Uninfected/' + 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))
```

3-3 視覺化資料

以上動作已經完成資料的前處理了，接下來我們來檢視一下細胞的圖像，讓資料更加視覺化。

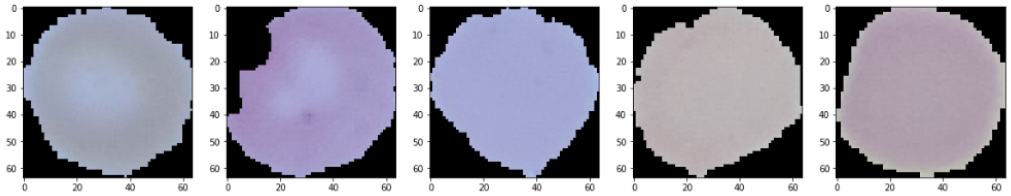
(1) 隨機抽取 5 張感染瘧疾細胞的圖片來檢視

```
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])
```



(2) 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])
```



四、模型設定

(1) 模型架構 model architecture

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())
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
batch_normalization_12 (Batc (None, 31, 31, 32)        128
_________________________________________________________________
dropout_12 (Dropout)         (None, 31, 31, 32)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 30752)             0
_________________________________________________________________
dense_10 (Dense)             (None, 512)               15745536
_________________________________________________________________
batch_normalization_13 (Batc (None, 512)               2048
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_11 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_14 (Batc (None, 256)               1024
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_12 (Dense)             (None, 2)                 514
=================================================================
Total params: 15,881,474
Trainable params: 15,879,874
Non-trainable params: 1,600
_________________________________________________________________
None
```

(2) 執行過程

訓練集 8 成，測試集 2 成。

```
In [22]: #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
```

開始訓練、batch-size 設定為 64，epochs=10。

```
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
```

訓練花費時間為 990 秒，測試花費時間為 8 秒

結果測試集準確路為：84.00%

```
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%
```

(3) 原本模型執行結果表格

損失函數	卷積、池化	激活函數	最後一層激活函數	優化器	訓練集 accuracy	驗證集 accuracy	測試集 accuracy	Epoch
Binary	1C1P1D	relu	sigmoid	sgd	95.17%	85.35%	84.00%	10

五、改善方法及過程

(1) 改善方法 1：利用 Image Data Generator 來產生更多照片。

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

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 的結果：

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
```

訓練花費時間為 1233 秒，測試花費時間為 437 秒

測試集準確率為：94.38%

相較於原本的 84%增加了 10.38%的準確率。

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%
```

(2) 改善方法 2：再多增加一層變成 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())
```

新增層數

改善方法 2 的結果：

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
```

訓練花費時間為 987 秒，測試花費時間為 7 秒

測試集準確率為：87.38%

相較於原本的 84%增加了 3.38%的準確率。

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%
```

(3) 改善方法 3：多加一層加使用 Image Data Generator

（合併方法 1 和方法 2）

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 的結果：

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
```

訓練花費時間為 1239 秒，測試花費時間為 501 秒

測試集準確率為：91.81%

相較於原本的 84%增加了 7.81%的準確率。

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

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

(4) 結果統整表格

方法	訓練時間 (s)	測試時間 (s)	訓練準確率	測驗準確率	改善幅度
方法 1	1233	437	89.5%	**94.38**	**10.38%**
方法 2	987	7	88.44%	87.38%	4.38%
方法 3	1239	501	90.94%	91.81%	7.81%

六、結論

6-1 結果

透過以上的結果可以看出，方案 1-使用 Image Date Generator 來改善是有較佳的結果，但同時也需耗費較高的時間成本，而若使用方案 2-多增加層數雖然不會增加很多的時間成本，但提昇準確度的效果有限，值得注意的是，使用方案 3-合併方案 1 與方案 2 並不會得到比單獨使用方案 1 來得更好的結果，使用方案 3 不但所需時間成本增加，準確率的改進幅度也不比方案 1 要來的好。

6-2 未來改進方向

由結果可以知道其實準確度是不錯的，但這只是最簡單的判斷是否患有瘧疾細胞而已，瘧疾病原體其實還分為 5 種，分別為 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*），最常見的是惡性瘧原蟲，占了 75%，有些瘧疾種類甚至同時包含 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)[2]，

因為不同的病原體有不同的治療措施，所以若未來改進方向為不但可以分辨是否為瘧疾細胞，還能進一步的判斷是何種病原體所導致的，將可以真正減輕醫療人員的負擔，同時也能幫助非洲國家更快速、更低成本、更有效的快速篩檢出是否患有瘧疾。

参考文献

[1]World Malaria Report, World Health Organization, 2015.

[2]”*瘧疾*” *In Wikipedia.* 检至 https://w.wiki/Ej2 (Dec. 29.2019)

[3]World Malaria Report, World Health Organization, 2013.