國立清華大學

智慧化企業整合
Intelligent Integration of Enterprise

Project 3

暗視野顯微鏡下細菌影像偵測

指導教授： 邱銘傳 教授

組別： 第二組
學生：110034587 曾琮祐

目錄

一、背景介紹..... 5
（一）背景與動機..... 5
（二）研究目的..... 5
（三）問題描述..... 6

二、研究方法..... 6
（一）U-net.....6
（二）Unet++ 6

三、個案研究..... 7
（一）資料介紹..... 7
（二）資料前處理..... 7
（三）模型建立與訓練..... 8
（四）參數優化..... 9
（五）實驗設計結果..... 10

四、結論..... 11
（一）貢獻..... 11
（二）侷限性..... 14
（三）適用性..... 11
（四）未來展望..... 11

五、參考資料..... 11

圖目錄

圖 1 暗視野顯微鏡下影像 5
圖 2 暗視野顯微鏡下影像(左)與預測圖像(右) 5
圖 3 U-net 網路架構 6
圖 4 Unet++網路架構 7
圖 5 暗視野顯微鏡下圖像(左)與對應遮罩影像(右) 7
圖 6 長、寬皆小於 256 時，插值法 resize 的程式碼 7
圖 7 長或寬一項小於 256、一項大於 256 時，插值法 resize 的程式碼 8
圖 8 滑動窗口程式碼 8
圖 9 資料分割 8
圖 10 相關函數程式碼 8
圖 11 Unet++模型程式碼 9

表目錄

表 1 5W1H..... 6
表 2 實驗因子及水準..... 9
表 3 L16 實驗設計參數組合..... 9
表 4 實驗設計結果..... 10
表 5 Unet++最佳組合..... 11

一、 背景介紹

（一）背景與動機

暗視野顯微鏡(Dark field microscope)是光學顯微鏡的一種。照明光線不直接進物鏡，只允許被標本反射和衍射的光線進入物鏡，因而視野的背景是黑的，物體的邊緣是亮的。

暗視野顯微鏡常用來觀察未染色的透明樣品。這些樣品因為具有和周圍環境相似的折射率，不易在一般明視野之下看的清楚，於是利用暗視野提高樣品本身與背景之間的對比。這種顯微鏡能見到小至 4~200nm 的微粒子。

圖 1　暗視野顯微鏡下影像

（二）研究目的

運用 UNet++網路進行圖像分割，所預測出的圖像可以讓人更快辨認細菌所在位置，並且使其更容易理解和分析。

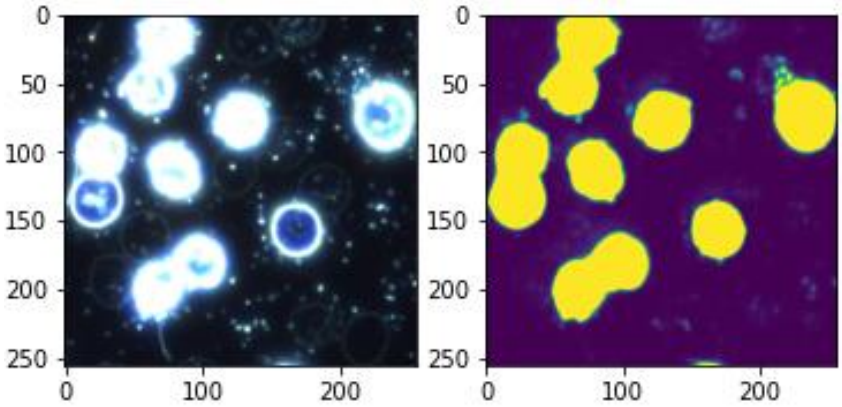

圖 2　暗視野顯微鏡下影像(左)與預測圖像(右)

(三)問題描述

表 1　5W1H

What	暗視野顯微鏡圖片會有一些不相干物體，影響判讀
Why	使相關人員能夠快速理解與分析圖片，並且也方便讓電腦做進一步分析
Where	學校、生技公司等會使用到暗視野顯微鏡觀察細菌的地方
When	暗視野顯微鏡細菌圖片取得後，想快速理解和分析時
Who	教授、學生、生技公司成員等會使用到暗視野顯微鏡觀察細菌的人
How	Unet++模型訓練

二、 研究方法

本研究使用 Unet++模型來進行圖像分割。

(一)　U-net

U-Net 是 Autoencoder 的一種變形(Variant)，因為它的模型結構類似 U 型而得名。

U-Net 在原有的編碼器與解碼器的聯繫上，增加了跳躍連接機制，使編碼器每一層的資訊，額外輸入到一樣大小的解碼器的對應層，在重建的過程就比較不會遺失重要資訊了。

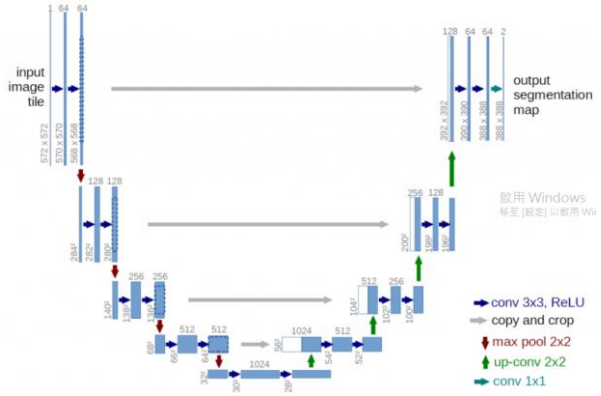

圖 3　U-net 網路架構

(二)　Unet++

在 UNet++中引入不同深度的 U-Net 集成，能夠有效增加對於不同 scale 目標的分割性能

重新設計了跳躍連接這個機制，也就能在解碼器子網路中實現靈活的特徵融合

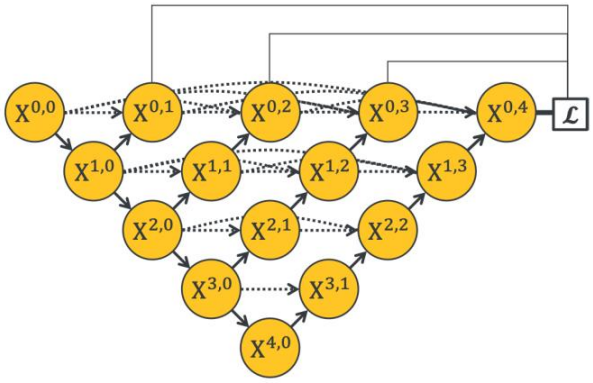

圖 4　Unet++網路架構

三、個案研究

(一) 資料介紹

本研究使用 Kaggle 網站中暗視野顯微鏡下細菌偵測圖像的公開資料集。資料集中包含暗視野顯微鏡下細菌影像與對應遮罩影像兩種圖像資料。兩者各有 366 張圖片，合共 732 張圖片。如圖 5 所示：

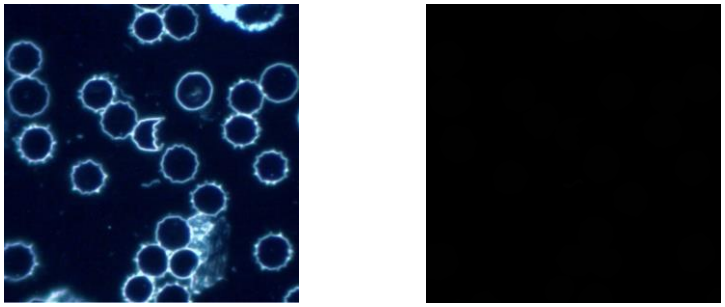

圖 5　暗視野顯微鏡下圖像(左)與對應遮罩影像(右)

(二) 資料前處理

Step 1：

檢查有無缺失圖片。

Step 2：

圖片大小以 256x256 為基準。若長、寬皆小於 256，藉由插值法 resize 成 256，程式碼如圖 6 所示；若長或寬一項大於 256、一項小於 256，將圖片按原本比例藉由插值法放大，使最短邊 resize 成 256，之後再利用滑動窗口方式剪裁，程式碼如圖 7 所示：

```
if image.shape[1] < self.width and image.shape[0] < self.height:
        image = cv2.resize(image, (self.width, self.height), interpolation=cv2.INTER_AREA)
        mask = cv2.resize(mask, (self.width, self.height), interpolation=cv2.INTER_AREA)
        assert image.shape[0] == mask.shape[0] and image.shape[1] == mask.shape[1]
```

圖 6　長、寬皆小於 256 時，插值法 resize 的程式碼

```
elif image.shape[1] > self.width and image.shape[0] < self.height:
        w = int(round(self.height * image.shape[1] / image.shape[0]))
        image = cv2.resize(image, (w, self.height), interpolation=cv2.INTER_AREA)
        mask = cv2.resize(mask, (w, self.height), interpolation=cv2.INTER_AREA)
        assert image.shape[0] == mask.shape[0] and image.shape[1] == mask.shape[1]
```

圖 7　長或寬一項小於 256、一項大於 256 時，插值法 resize 的程式碼

Step 3：

利用滑動窗口方式取得多張 256x256 圖片。其程式碼如圖 8：

```
self.last_image = []
self.last_mask = []
for x in range(0, image.shape[1] - self.stride, self.stride):
    for y in range(0, image.shape[0] - self.stride, self.stride):
        x0 = x
        x1 = x + self.width
        y0 = y
        y1 = y + self.height

        image_slide = image[y0:y1, x0:x1, :]
        mask_slide = mask[y0:y1, x0:x1]

        assert image_slide.shape[0] == mask_slide.shape[0] and image_slide.shape[1] == mask_slide.shape[1]

        if image_slide.shape[1] == self.width and image_slide.shape[0] == self.height:
            pass
        elif image_slide.shape[1] < self.width and image_slide.shape[0] == self.height:
            x1 = image.shape[1]
            x0 = x1 - self.width
        elif image_slide.shape[1] == self.width and image_slide.shape[0] < self.height:
            y1 = image.shape[0]
            y0 = y1 - self.height
        else:
            x1 = image.shape[1]
            x0 = x1 - self.width
            y1 = image.shape[0]
            y0 = y1 - self.height

        image_slide = image[y0:y1, x0:x1, :]
        mask_slide = mask[y0:y1, x0:x1]

        assert image_slide.shape[0] == mask_slide.shape[0] and image_slide.shape[1] == mask_slide.shape[1]
        assert image_slide.shape[0] == self.height and image_slide.shape[1] == self.width

        ret.append((image_slide, mask_slide))
```

圖 8　滑動窗口程式碼

Step 4：

訓練集、測試集依照 85%、15%分割。如圖 9：

```
test_split = 0.15

gc.collect()

indices = np.random.permutation(images.shape[0])
boundary = images.shape[0] - int(images.shape[0] * test_split)
training_idx, test_idx = indices[:boundary], indices[boundary:]
x_train, x_test = images[training_idx, :], images[test_idx, :]
y_train, y_test = masks[training_idx, :], masks[test_idx, :]
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
((2075, 256, 256, 3),
 (2075, 256, 256, 1),
 (366, 256, 256, 3),
 (366, 256, 256, 1))
```

圖 9　資料分割

(三) 模型建立與訓練

利用 TensorFlow 開源軟體庫取得模型建立相關所需函數。相關函數的程式碼如圖 10：

```
import tensorflow as tf
import tensorflow.keras.backend as K
import typing
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, LeakyReLU, Dropout, MaxPooling2D, Conv2DTranspose, concatenate
from tensorflow.keras.optimizers import RMSprop


def weighted_loss(original_loss_function: typing.Callable, weights_list: dict) -> typing.Callable:
    """
    Help function to balance background, bacteria and blood cells.
    """
    def loss_function(true, pred):
        class_selectors = tf.cast(K.argmax(true, axis=-1), tf.int32)
        class_selectors = [K.equal(i, class_selectors) for i in range(len(weights_list))]
        class_selectors = [K.cast(x, K.floatx()) for x in class_selectors]
        weights = [sel * w for sel, w in zip(class_selectors, weights_list)]
        weight_multiplier = weights[0]
        for i in range(1, len(weights)):
            weight_multiplier = weight_multiplier + weights[i]
        loss = original_loss_function(true, pred)
        loss = loss * weight_multiplier
        return loss
    return loss_function
```

圖 10　相關函數程式碼

利用相關函數建立 Unet++模型。相關程式碼如圖 11：

```
class UNetPP:
    def __init__(self):
        model_input = Input((256, 256, 3))
        x00 = conv2d(filters=int(16 * number_of_filters))(model_input)
        x00 = BatchNormalization()(x00)
        x00 = LeakyReLU(0.01)(x00)
        x00 = Dropout(0.2)(x00)
        x00 = conv2d(filters=int(16 * number_of_filters))(x00)
        x00 = BatchNormalization()(x00)
        x00 = LeakyReLU(0.01)(x00)
        x00 = Dropout(0.2)(x00)
        p0 = MaxPooling2D(pool_size=(2, 2))(x00)

        x10 = conv2d(filters=int(32 * number_of_filters))(p0)
        x10 = BatchNormalization()(x10)
        x10 = LeakyReLU(0.01)(x10)
        x10 = Dropout(0.2)(x10)
        x10 = conv2d(filters=int(32 * number_of_filters))(x10)
        x10 = BatchNormalization()(x10)
        x10 = LeakyReLU(0.01)(x10)
        x10 = Dropout(0.2)(x10)
        p1 = MaxPooling2D(pool_size=(2, 2))(x10)

        x01 = conv2dtranspose(int(16 * number_of_filters))(x10)
        x01 = concatenate([x00, x01])
        x01 = conv2d(filters=int(16 * number_of_filters))(x01)
        x01 = BatchNormalization()(x01)
        x01 = LeakyReLU(0.01)(x01)
        x01 = conv2d(filters=int(16 * number_of_filters))(x01)
        x01 = BatchNormalization()(x01)
        x01 = LeakyReLU(0.01)(x01)
        x01 = Dropout(0.2)(x01)
```

圖 11　Unet++模型程式碼

(四) 參數優化

使用田口方法實驗設計提升效率。挑選出的因子以及水準如表 2：

表 2 實驗因子及水準

因子	說明	水準 1	水準 2	水準 3	水準 4
A	Batch Size	2	4	5	3
B	Dropout	0.2	0.3	0.4	0.5
C	Filter	2	3	4	5
D	Optimizer	Adam	Adagrad	Adadelta	RMSprop
E	Learning rate	0.0005	0.0001	0.005	0.001

我們利用了實驗設計中的田口方法，有效減少調整參數的總次數，並獲得相同的結果。我們選擇了上述所提到的五項參數作為五個因子，並使用四個水準，應用 L16 實驗設計參數組合來幫助參數優化。針對每次實驗時都統一使用 25epoch 進行訓練，實驗設計參數組合詳情如表 4：

表 3　L16 實驗設計參數組合

實驗	Batch size	Dropout	Filter	Optimizer	Learning rate
1	2	0.2	2	Adam	0.0005
2	2	0.3	3	Adagrad	0.0001
3	2	0.4	4	Adadelta	0.005
4	2	0.5	5	RMSprop	0.001
5	4	0.2	3	Adadelta	0.001

6	4	0.3	2	RMSprop	0.005
7	4	0.4	5	Adam	0.0001
8	4	0.5	4	Adagrad	0.0005
9	5	0.2	4	RMSprop	0.0001
10	5	0.3	5	Adadelta	0.0005
11	5	0.4	2	Adagrad	0.001
12	5	0.5	3	Adam	0.005
13	3	0.2	5	Adagrad	0.005
14	3	0.3	4	Adam	0.001
15	3	0.4	3	RMSprop	0.0005
16	3	0.5	2	Adadelta	0.0001

(五) 實驗設計結果

表 4　實驗設計結果

實驗	Batch size	Dropout	Filter	Optimizer	Learning rate	Test_dice
1	2	0.2	2	Adam	0.0005	0.9620
2	2	0.3	3	Adagrad	0.0001	0.7506
3	2	0.4	4	Adadelta	0.005	0.8199
4	2	0.5	5	RMSprop	0.001	0.9213
5	4	0.2	3	Adadelta	0.001	0.7738
6	4	0.3	2	RMSprop	0.005	0.9589
7	4	0.4	5	Adam	0.0001	0.9422
8	4	0.5	4	Adagrad	0.0005	0.8443
9	5	0.2	4	RMSprop	0.0001	0.9619
10	5	0.3	5	Adadelta	0.0005	0.6552
11	5	0.4	2	Adagrad	0.001	0.8083
12	5	0.5	3	Adam	0.005	0.9453
13	3	0.2	5	Adagrad	0.005	0.8658
14	3	0.3	4	Adam	0.001	0.9432
15	3	0.4	3	RMSprop	0.0005	0.9600
16	3	0.5	2	Adadelta	0.0001	0.5418

表 4 為實驗組合及結果，準確度最高的為實驗 1，Test_dice 達到了 0.9620，將這 16 次實驗結果透過統計分析後可以得到各個因子對於準確率的影響程度，我們又從中挑選出了各個因子中最好的水準組合來做測試，

希望可以得到更好的結果，最佳組合為表 5，準確度為 0.9635。

表 5　Unet++最佳組合

	Batch size	Dropout	Filter	Optimizer	Learning rate	Test_dice
最佳組合	4	0.2	4	RMSprop	0.005	0.9635

四、結論

Unet++模型在 L16 直交表五種因子十六種實驗組合下分別在 Batch size/ Dropout/ Filter/ Optimizer/ Learning rate 得到最佳準確率分別為 4/ 0.2/ 4/ RMSprop/ 0.005。

總結來說，對於此次研究結果可歸納於以下四點：

(一) 貢獻

對暗視野顯微鏡影像進行有效的圖片分割，幫助分析與判讀。

(二) 侷限性

若用於測試的圖片有經過擾動，會導致圖片的錯誤分割，影響準確度。

(三) 適用性

最終結果有 96%以上，可以應用於實務上判讀的機率很大。

(四) 未來展望

可以做進一步的語義分割判別細菌種類與使用其他種類顯微鏡圖片驗證模型泛化性。

五、參考資料

(一) https://www.kaggle.com/longnguyen2306/bacteria-detection-with-darkfield-microscopy

(二) https://www.kaggle.com/longnguyen2306/unet-for-spirochaeta-bacteria-detection/notebook

(三) Zhou, ZW., Siddiquee, MMR., Tajbakhsh, N., and Liang, JM. (2020) “UNet plus plus : Redesigning Skip Connections to Exploit Multiscale Features in Image Segmentation,” *IEEE TRANSACTIONS ON MEDICAL IMAGING*: 1856-1867 doi:10.1109/TMI.2019.2959609