暗視野顯微鏡下細菌影像偵測

指導教授

邱銘傳 博士

組別

第二組

學生

110034587 曾琮祐

目錄

01

Chapter

背景介紹

Introduction

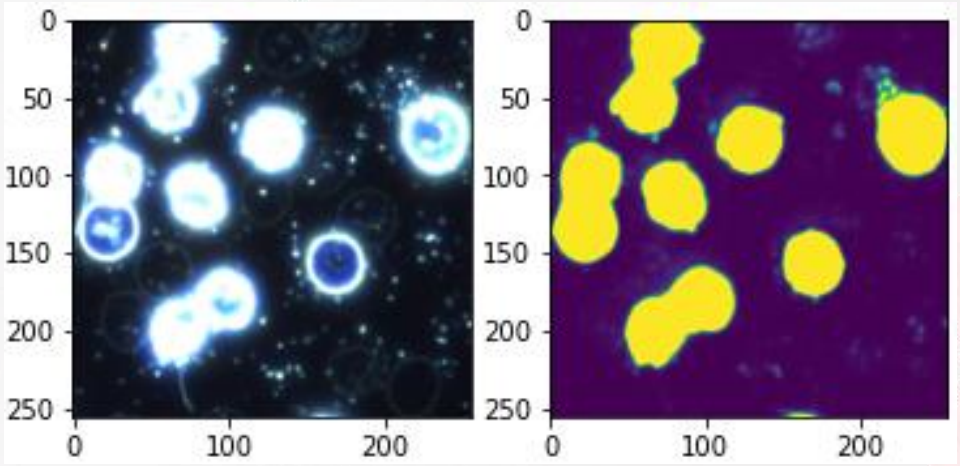
背景

- 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

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



5W1H

What

暗視野顯微鏡圖片會有一些不相干物體，影響判讀

When

暗視野顯微鏡細菌圖片取得後，想快速理解和分析時

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

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

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

How

Unet++模型訓練

02

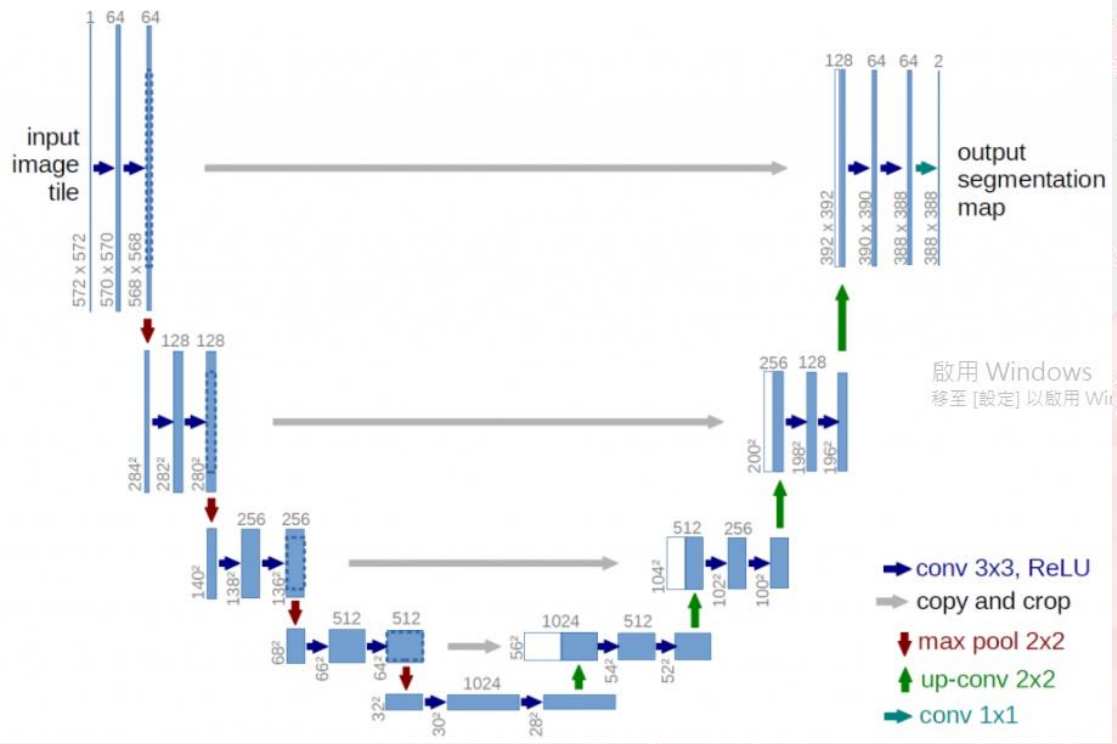
Chapter

研究方法

Method

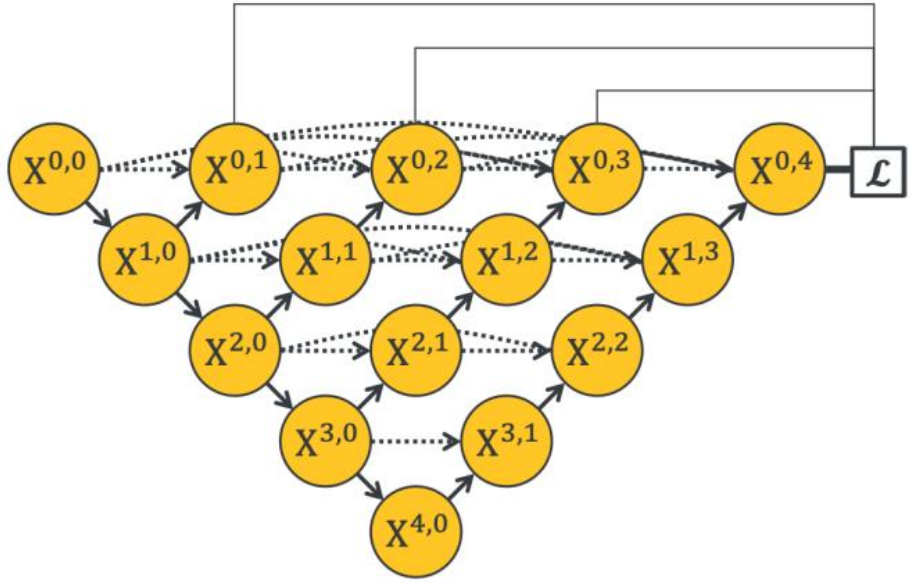
U-net

- 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



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



03

Chapter

模型訓練與積效

Experiment

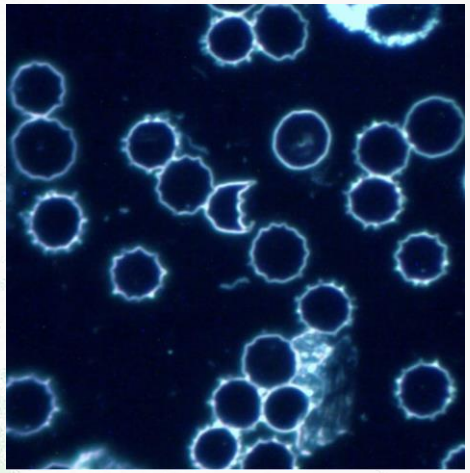
資料介紹

- **資料來源**

  Kaggle Dataset: Bacteria detection with darkfield microscopy

- **資料集介紹**

  顯微鏡下細菌影像: 366張

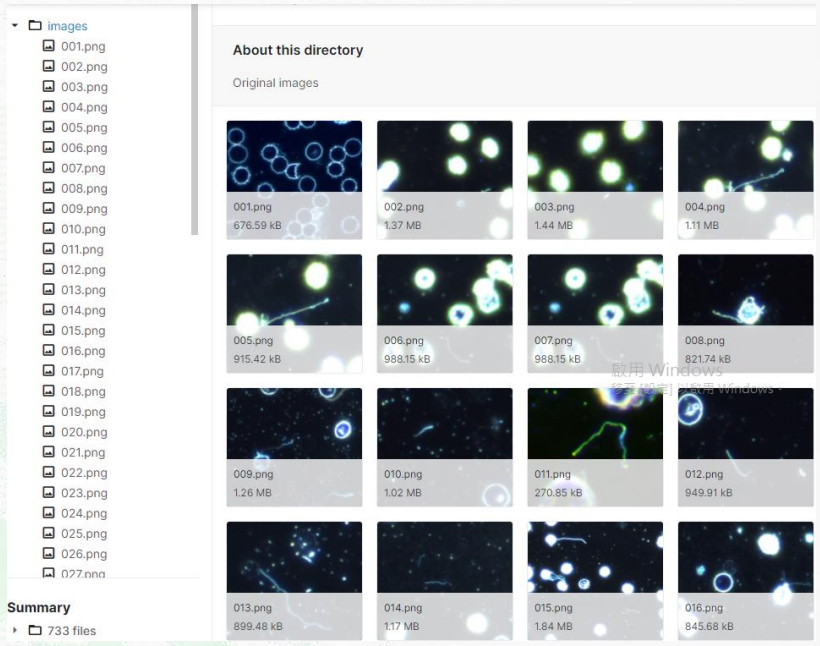
  對應遮罩影像: 366張

資料前處理(1/3)

- **資料檢查**

  檢查有無缺失圖片



- **圖片大小統一**

  圖片大小以256x256為基準:

  ✓ **長、寬皆小於256:** 藉由插值法resize成256

```
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]
```

  ✓ **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

```
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]
```

資料前處理(2/3)

- **數據增強**

利用滑動窗口方式取得多張256x256圖片

```
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))
```


- **增強後資料**

✓ **顯微鏡下細菌影像:** 2441張
✓ **對應遮罩影像:** 2441張

```
images.shape, masks.shape
```

```
((2441, 256, 256, 3), (2441, 256, 256, 1))
```

資料前處理(3/3)

- 資料分割

訓練集、測試集依照85%、15%分割

```
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))
```

模型建立與訓練(1/3)

- 函數定義

利用TensorFlow開源軟體庫取得模型建立相關所需函數

```
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
```

模型建立與訓練(2/3)

- 模型建立

利用相關函數建立Unet++模型

```
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)
```

模型建立與訓練(3/3)

- 模型架構圖

conv2d_transpose_9 (Conv2DTranspose)	(None, 256, 256, 64) 32832	['dropout_17[0][0]']
dropout_13 (Dropout)	(None, 256, 256, 64) 0	['leaky_re_lu_19[0][0]']
concatenate_9 (Concatenate)	(None, 256, 256, 320) 0	['conv2d_transpose_9[0][0]', 'dropout_1[0][0]', 'dropout_4[0][0]', 'dropout_8[0][0]', 'dropout_13[0][0]']
conv2d_28 (Conv2D)	(None, 256, 256, 64) 184384	['concatenate_9[0][0]']
batch_normalization_28 (BatchNormalization)	(None, 256, 256, 64) 256	['conv2d_28[0][0]']
leaky_re_lu_28 (LeakyReLU)	(None, 256, 256, 64) 0	['batch_normalization_28[0][0]']
conv2d_29 (Conv2D)	(None, 256, 256, 64) 36928	['leaky_re_lu_28[0][0]']
batch_normalization_29 (BatchNormalization)	(None, 256, 256, 64) 256	['conv2d_29[0][0]']
leaky_re_lu_29 (LeakyReLU)	(None, 256, 256, 64) 0	['batch_normalization_29[0][0]']
dropout_18 (Dropout)	(None, 256, 256, 64) 0	['leaky_re_lu_29[0][0]']
conv2d_30 (Conv2D)	(None, 256, 256, 3) 195	['dropout_18[0][0]']

Total params: 32,739,011
Trainable params: 32,725,699
Non-trainable params: 13,312

參數優化(1/9)

- 因子選擇

使用田口方法實驗設計提升效率:

✓ **五種因子:** Batch size、Dropout、Number of filter、Optimizer、Learning rate

✓ **四個水準:**

Factor	Item	Level 1	Level 2	Level 3	Level 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

參數優化(2/9)

- 直交表產生

五因子四水準會產生L16直交表:

Experiment	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

參數優化(3/9)

- 實驗結果

使用骰子係數作為圖像分割評估標準:

$$s = \frac{2|X \cap Y|}{|X| + |Y|}$$

Experiment	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

參數優化(4/9)

- 結果分析

使用Minitab分析找出關鍵因子:

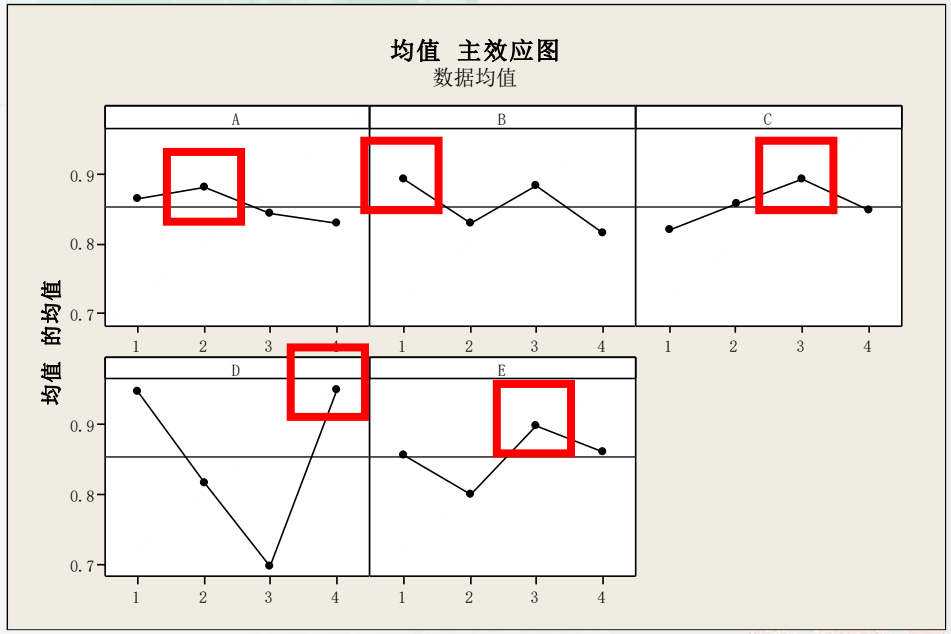
田口分析:test_dice 与 A, B, C, D, E

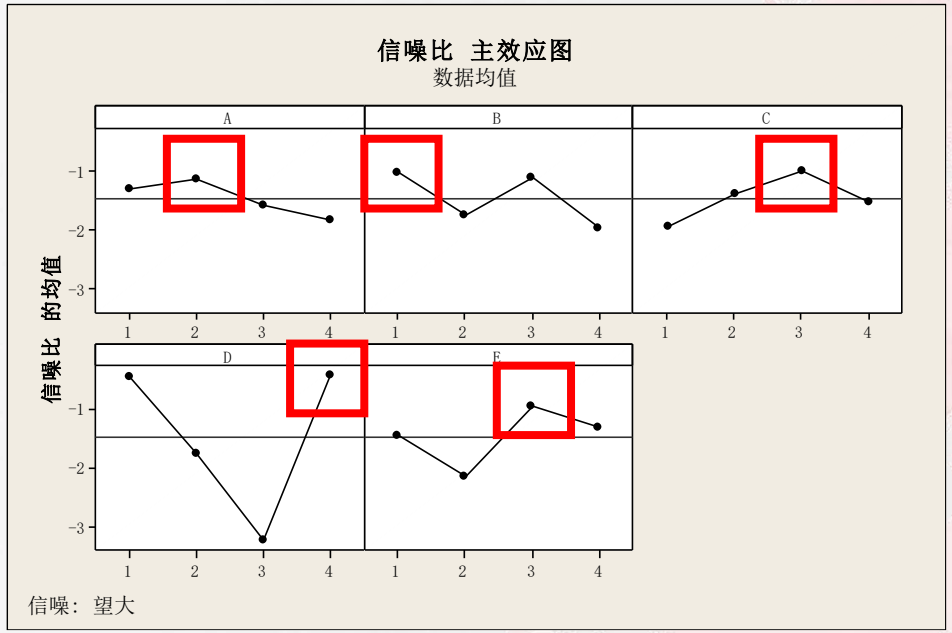
信噪比响应表
望大

水平	A	B	C	D	E
1	-1.3118	-1.0337	-1.9637	-0.4580	-1.4539
2	-1.1448	-1.7592	-1.3906	-1.7655	-2.1674
3	-1.5868	-1.1113	-1.0100	-3.2370	-0.9574
4	-1.8593	-1.9985	-1.5383	-0.4421	-1.3240
Delta	0.7146	0.9647	0.9536	2.7949	1.2100
排秩	5	3	4	1	2

均值响应表

水平	A	B	C	D	E
1	0.8640	0.8914	0.8182	0.9487	0.8559
2	0.8798	0.8270	0.8574	0.8173	0.7991
3	0.8427	0.8826	0.8923	0.6977	0.8975
4	0.8277	0.8132	0.8461	0.9505	0.8617
Delta	0.0521	0.0782	0.0741	0.2529	0.0984
排秩	5	3	4	1	2





參數優化(5/9)

田口分析:test_dice 与 A, B, C, D, E

信噪比响应表
望大

水平	A	B	C	D	E
1	-1.3118	-1.0337	-1.9637	-0.4580	-1.4539
2	-1.1448	-1.7592	-1.3906	-1.7655	-2.1674
3	-1.5868	-1.1113	-1.0100	-3.2370	-0.9574
4	-1.8593	-1.9985	-1.5383	-0.4421	-1.3240
Delta	0.7146	0.9647	0.9536	2.7949	1.2100
排秩	5	3	4	1	2

均值响应表

水平	A	B	C	D	E
1	0.8640	0.8914	0.8182	0.9487	0.8559
2	0.8798	0.8270	0.8574	0.8173	0.7991
3	0.8427	0.8826	0.8923	0.6977	0.8975
4	0.8277	0.8132	0.8461	0.9505	0.8617
Delta	0.0521	0.0782	0.0741	0.2529	0.0984
排秩	5	3	4	1	2

- 最佳參數組合

Experiment	Batch size	Dropout	Filter	Optimizer	Learning rate	Test_dice
17	4	0.2	4	RMSprop	0.005	**0.9635**

- 微調(Optimizer)

確定Optimizer使用RMSprop:

Experiment	Batch size	Dropout	Filter	Optimizer	Learning rate	Test_dice
17	4	0.2	4	RMSprop	0.005	**0.9635**
18	4	0.2	4	Adam	0.005	**0.9342**

參數優化(6/9)

- 微調(Learning rate)

確定Learning rate:

Experiment	Batch size	Dropout	Filter	Optimizer	Learning rate	Test_dice
17	4	0.2	4	RMSprop	0.005	**0.9635**
19	4	0.2	4	RMSprop	0.01	**0.9605**
20	4	0.2	4	RMSprop	0.006	**0.9618**
21	4	0.2	4	RMSprop	0.004	**0.9612**

田口分析:test_dice 与 A, B, C, D, E

信噪比响应表
望大

水平	A	B	C	D	E
1	-1.3118	-1.0337	-1.9637	-0.4580	-1.4539
2	-1.1448	-1.7592	-1.3906	-1.7655	-2.1674
3	-1.5868	-1.1113	-1.0100	-3.2370	-0.9574
4	-1.8593	-1.9985	-1.5383	-0.4421	-1.3240
Delta	0.7146	0.9647	0.9536	2.7949	1.2100
排秩	5	3	4	1	2

均值响应表

水平	A	B	C	D	E
1	0.8640	0.8914	0.8182	0.9487	0.8559
2	0.8798	0.8270	0.8574	0.8173	0.7991
3	0.8427	0.8826	0.8923	0.6977	0.8975
4	0.8277	0.8132	0.8461	0.9505	0.8617
Delta	0.0521	0.0782	0.0741	0.2529	0.0984
排秩	5	3	4	1	2

參數優化(7/9)

- 微調(Dropout)

確定Dropout:

Experiment	Batch size	Dropout	Filter	Optimizer	Learning rate	Test_dice
17	4	0.2	4	RMSprop	0.005	**0.9635**
22	4	0.1	4	RMSprop	0.005	**0.9613**
23	4	0.15	4	RMSprop	0.005	**0.9587**

田口分析:test_dice 与 A, B, C, D, E

信噪比响应表
望大

水平	A	B	C	D	E
1	-1.3118	-1.0337	-1.9637	-0.4580	-1.4539
2	-1.1448	-1.7592	-1.3906	-1.7655	-2.1674
3	-1.5868	-1.1113	-1.0100	-3.2370	-0.9574
4	-1.8593	-1.9985	-1.5383	-0.4421	-1.3240
Delta	0.7146	0.9647	0.9536	2.7949	1.2100
排秩	5	3	4	1	2

均值响应表

水平	A	B	C	D	E
1	0.8640	0.8914	0.8182	0.9487	0.8559
2	0.8798	0.8270	0.8574	0.8173	0.7991
3	0.8427	0.8826	0.8923	0.6977	0.8975
4	0.8277	0.8132	0.8461	0.9505	0.8617
Delta	0.0521	0.0782	0.0741	0.2529	0.0984
排秩	5	3	4	1	2

參數優化(8/9)

- 微調(Number of filter)

確定Number of filter:

Experiment	Batch size	Dropout	Filter	Optimizer	Learning rate	Test_dice
17	4	0.2	4	RMSprop	0.005	0.9635

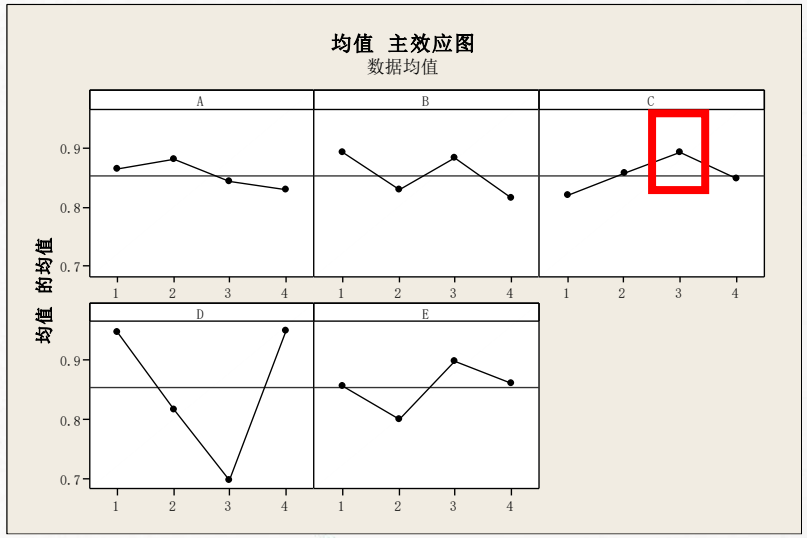
田口分析:test_dice 与 A, B, C, D, E

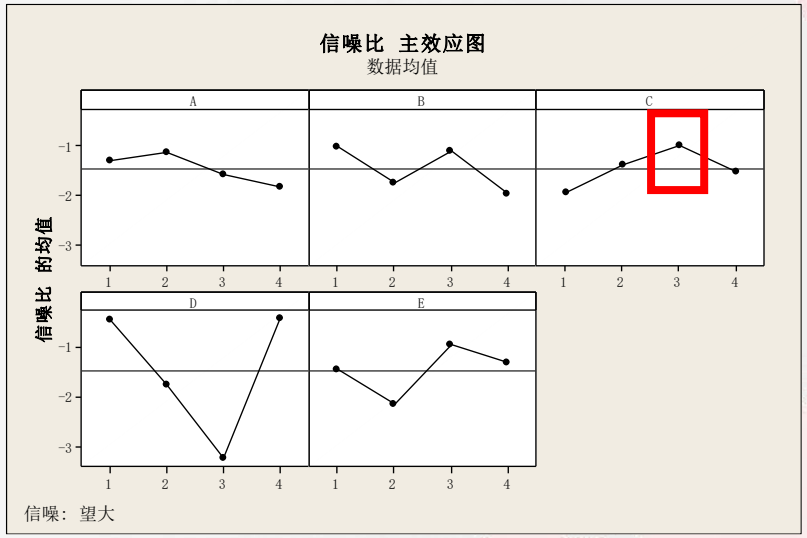
信噪比响应表
望大

水平	A	B	C	D	E
1	-1.3118	-1.0337	-1.9637	-0.4580	-1.4539
2	-1.1448	-1.7592	-1.3906	-1.7655	-2.1674
3	-1.5868	-1.1113	-1.0100	-3.2370	-0.9574
4	-1.8593	-1.9985	-1.5383	-0.4421	-1.3240
Delta	0.7146	0.9647	0.9536	2.7949	1.2100
排秩	5	3	4	1	2

均值响应表

水平	A	B	C	D	E
1	0.8640	0.8914	0.8182	0.9487	0.8559
2	0.8798	0.8270	0.8574	0.8173	0.7991
3	0.8427	0.8826	0.8923	0.6977	0.8975
4	0.8277	0.8132	0.8461	0.9505	0.8617
Delta	0.0521	0.0782	0.0741	0.2529	0.0984
排秩	5	3	4	1	2





參數優化(9/9)

- 微調(Batch size)

確定Batch size:

Experiment	Batch size	Dropout	Filter	Optimizer	Learning rate	Test_dice
17	4	0.2	4	RMSprop	0.005	0.9635

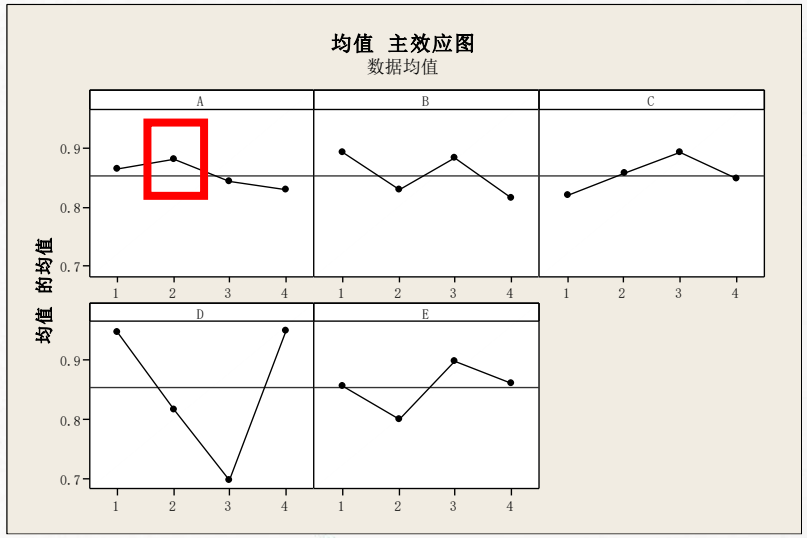
田口分析:test_dice 与 A, B, C, D, E

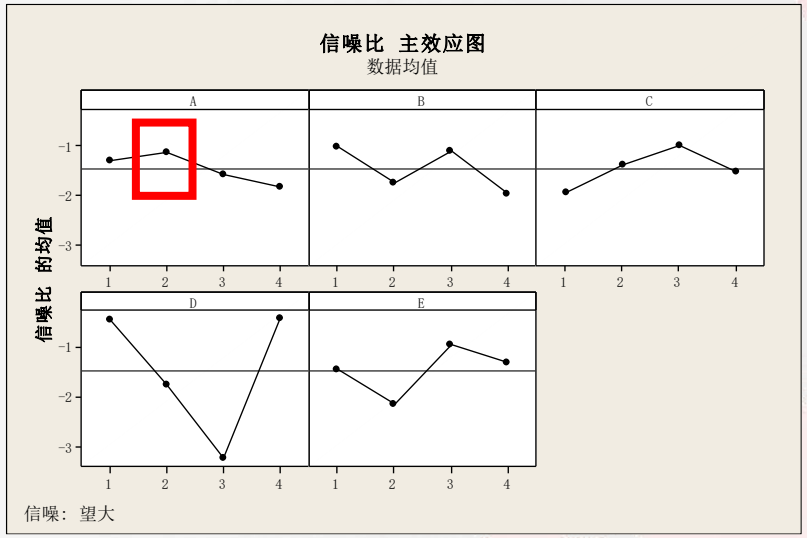
信噪比响应表
望大

水平	A	B	C	D	E
1	-1.3118	-1.0337	-1.9637	-0.4580	-1.4539
2	-1.1448	-1.7592	-1.3906	-1.7655	-2.1674
3	-1.5868	-1.1113	-1.0100	-3.2370	-0.9574
4	-1.8593	-1.9985	-1.5383	-0.4421	-1.3240
Delta	0.7146	0.9647	0.9536	2.7949	1.2100
排秩	5	3	4	1	2

均值响应表

水平	A	B	C	D	E
1	0.8640	0.8914	0.8182	0.9487	0.8559
2	0.8798	0.8270	0.8574	0.8173	0.7991
3	0.8427	0.8826	0.8923	0.6977	0.8975
4	0.8277	0.8132	0.8461	0.9505	0.8617
Delta	0.0521	0.0782	0.0741	0.2529	0.0984
排秩	5	3	4	1	2





04

Chapter

結果與未來展望

Conclusion

結果討論

- 最佳因子組合

Experiment	Batch size	Dropout	Filter	Optimizer	Learning rate	Test_dice
17	4	0.2	4	RMSprop	0.005	**0.9635**

Conclusion

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

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

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

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

Thanks

Thanks for your attention!