

---

# 手把手智能品檢與 預知維修實務

---

主題：表面瑕疵檢測  
講者：廖俊祺

2020/11/28

# 摘要

---

一、前言

二、研究目標與領域知識

三、資料集說明

四、模型設計

五、實驗結果

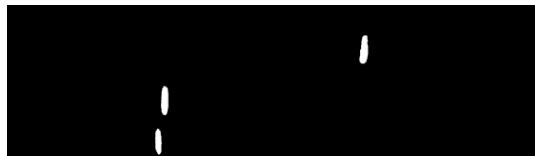
# 一、前言

---

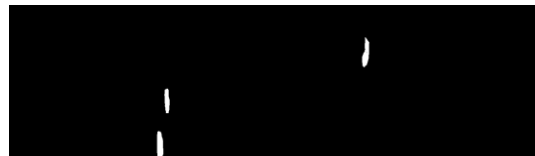
- 台灣的金屬加工業製造之商品行銷全球，品質優良且價格具競爭力，為了保持優質的品質，品質檢測就顯得特別重要。目標為表面瑕疵檢測，以公開資料集 Severstal : Steel Defect Detection 資料集為例子進行說明。該資料集為金屬鐵板之瑕疵樣本(四類)。本次將透過深度學習演算法偵測瑕疵，並比較傳統瑕疵檢測方法以及深度學習方法之偵測效果差異。



原始影像



Mask (Label)



預期偵測結果

## 二、研究目標與領域知識

---

研究目標：

- 研析傳統演算法以及深度學習方法於瑕疵偵測之差異
- 開發金屬鐵板瑕疵偵測模型

方法：

- 鐵板金屬表面經過加熱及冷卻，易在切割及捲取時產生瑕疵。經研析後發現金屬表面紋路過於複雜導致瑕疵難以檢測，人工目檢也十分困難。
- 本研究透過深度學習方法偵測瑕疵，並比較傳統瑕疵檢測方法以及深度學習方法之偵測效果差異。

### 三、資料集說明

---

- 資料來源：Severstal : Steel Defect Detection

Severstal公司提供的金屬鐵板影像，並由Kaggle提供公開資料集

- 瑕疵類型：4類(共7049張)

1. 壓坑(896張)
2. 黑色劃痕(247張)
3. 白色劃痕(5150張)
4. 刮傷(801張)

- 訓練與測試樣本量：總樣本數的八成為訓練樣本，其餘為測試樣本

1. 訓練樣本：5673張影像
2. 測試樣本：1421張影像

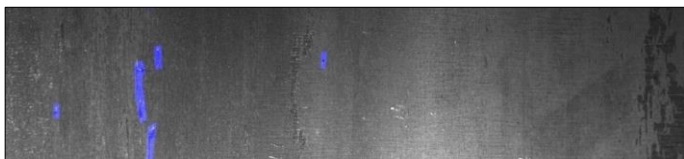
### 三、資料集說明 - 壓坑

---

- 以下為壓坑瑕疵樣本的範例圖以及標示瑕疵示意圖：



(a)壓坑瑕疵原始影像 1



(b)圖(a)之標示瑕疵影像



(c)壓坑瑕疵原始影像 2

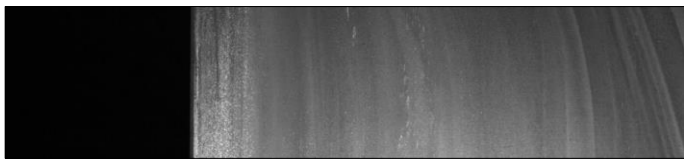


(d)圖(c)之標示瑕疵影像

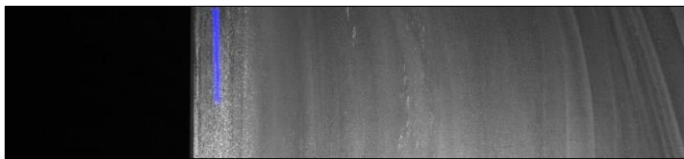
### 三、資料集說明 - 黑色劃痕

---

- 以下為黑色劃痕瑕疵樣本的範例圖以及標示瑕疵示意圖：



(a)黑色劃痕瑕疵原始影像 1



(b)圖(a)之標示瑕疵影像



(c)黑色劃痕瑕疵原始影像 2



(d)圖(c)之標示瑕疵影像

### 三、資料集說明 - 白色劃痕

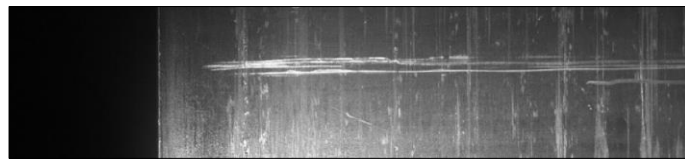
- 以下為白色劃痕瑕疵樣本的範例圖以及標示瑕疵示意圖：



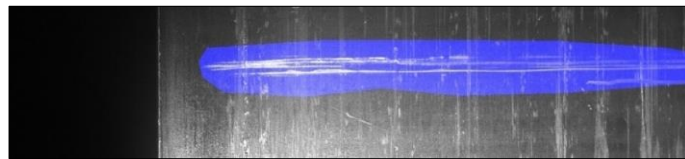
(a)白色劃痕瑕疵原始影像 1



(b)圖(a)之標示瑕疵影像



(c)白色劃痕瑕疵原始影像 2



(d)圖(c)之標示瑕疵影像



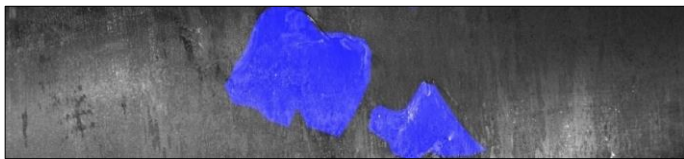
### 三、資料集說明 - 刮傷

---

- 以下為刮傷瑕疵樣本的範例圖以及標示瑕疵示意圖：



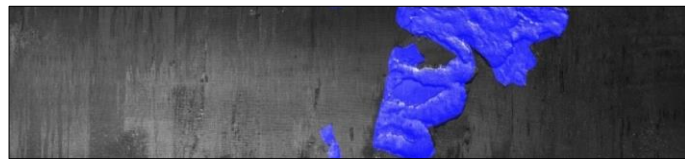
(a)刮傷瑕疵原始影像 1



(b)圖(a)之標示瑕疵影像



(c)刮傷瑕疵原始影像 2



(d)圖(c)之標示瑕疵影像

## 四、模型設計

---

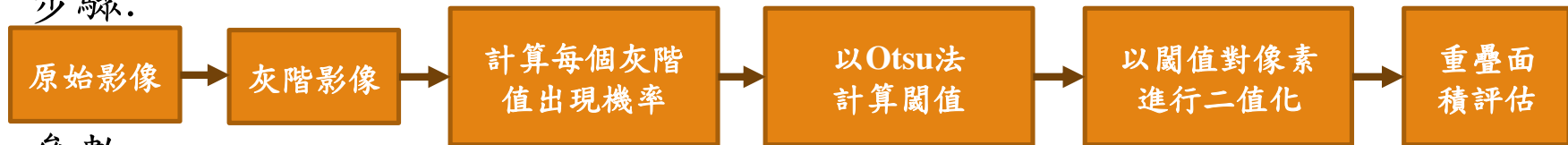
將比較傳統檢測方法與深度學習方法

- 傳統檢測方法:
  1. 二值化(Binarization)
  2. 邊緣檢測法(Edge Detection)
  3. K-平均演算法(K-means)
- 深度學習方法:
  4. U-Net
  5. Mask R-CNN

## 四、模型設計 - 二值化(Binarization)

### 1. 二值化(Binarization)

- 步驟:



- 參數

(1)閾值：以Otsu自動計算

(2)當灰階大於閾值，設定為255；其餘設定為0。

(3)二值化選項：THRESH\_BINARY(黑白二值化顯示)

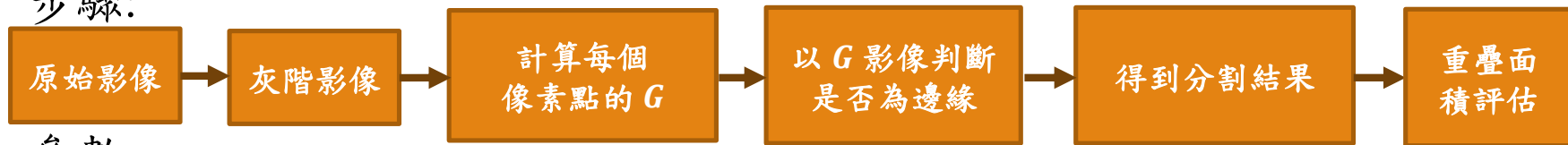
```
#建立二值化模型
```

```
model = cv2.threshold(img, 129, 255, cv2.THRESH_BINARY) # binary (黑白二值)
```

## 四、模型設計 - 邊緣檢測(Edge Detection)

### 2. 邊緣檢測(Edge Detection)-使用濾波器方式計算

- 步驟:



- 參數

#### (1) 邊緣檢測演算法: Sobel

```
#建立邊緣檢測模型  
model = filters.sobel(image)  
re = (edge_sobel, cmap=plt.cm.gray)
```

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

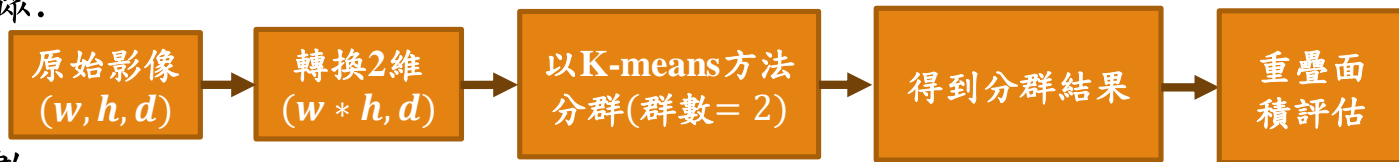
$$G = \sqrt{G_x^2 + G_y^2}$$

## 四、模型設計 - K-means分群法

---

### 3. K-means分群法(K-means Clustering)

- 步驟:



- 參數

(1)分群數量選擇：2 (n\_clusters=2 ，分別為背景與瑕疵兩類)

```
#建立kmeans分割法模型
kmeans_cluster = cluster.KMeans(n_clusters=2)
kmeans_cluster.fit(image_2d)
cluster_labels = kmeans_cluster.labels_
```

## 四、模型設計 - 傳統檢測方法效果

---

- 傳統檢測方法效果



原始影像



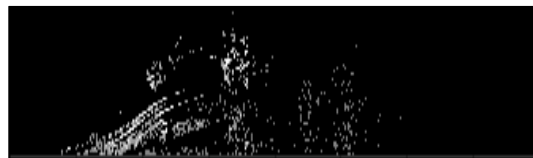
標示瑕疵位置



二值化效果



邊緣檢測效果



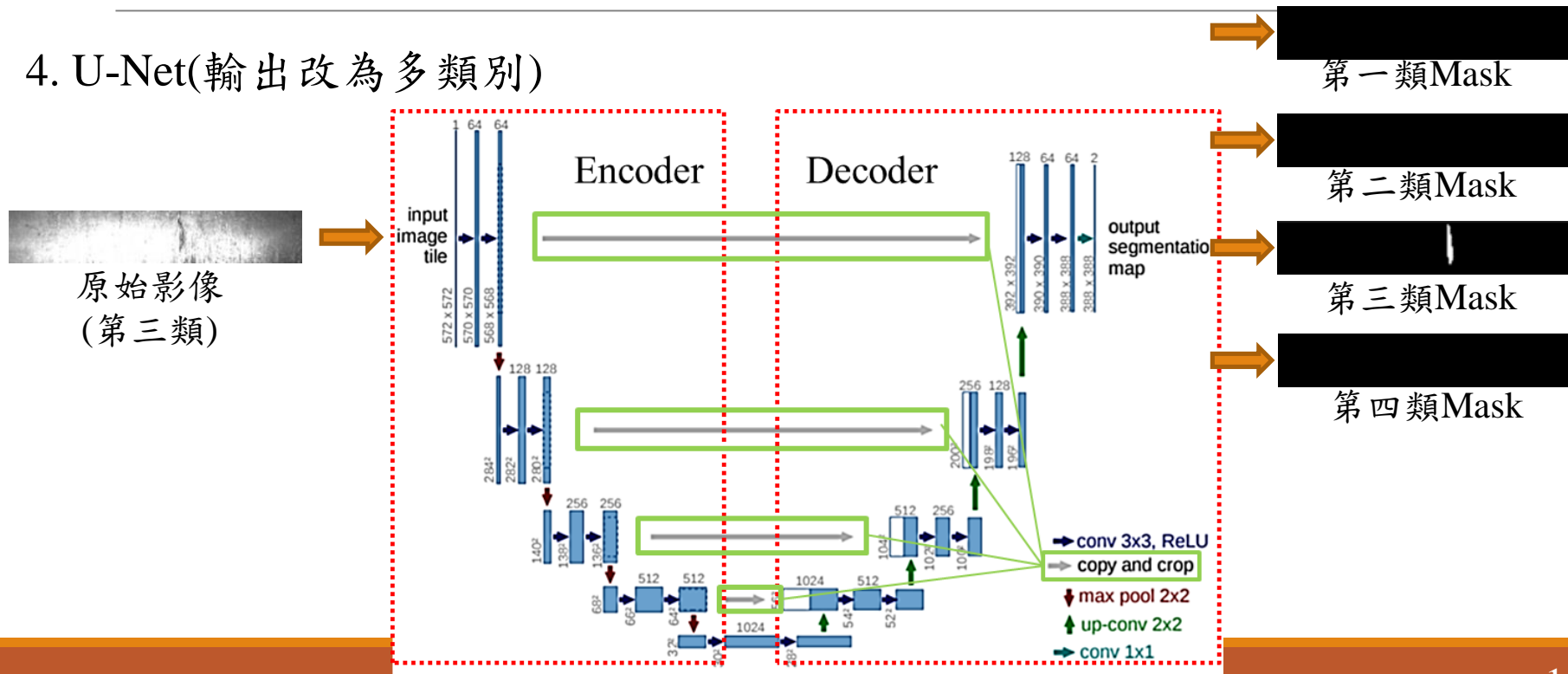
K-means分群法效果

## 四、模型設計 - U-Net

- 採用編碼器(下採樣)-解碼器(上採樣)結構(Encoder-Decoder)
- 使用跳躍連接方法(copy and crop)
- 原始U-Net為二維影像分割神經網絡，本研究修改輸出層，改為多類別。

## 四、模型設計 - U-Net

#### 4. U-Net(輸出改為多類別)





## 四、模型設計 - Mask R-CNN

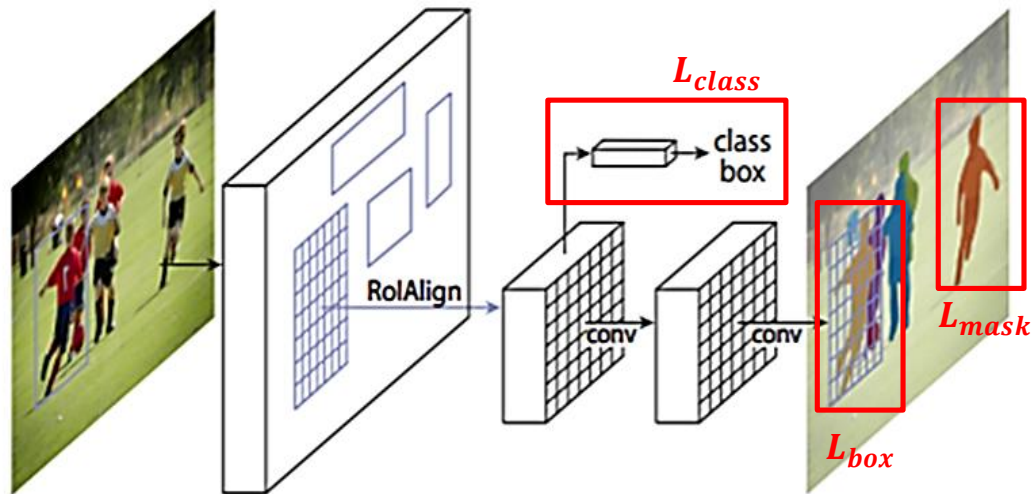
### 5. Mask R-CNN

- 能夠將每個物體進行逐影像切割，於物體重疊處可以區隔清楚並同時達到分類、檢測與分割效果

- 損失函數設計:

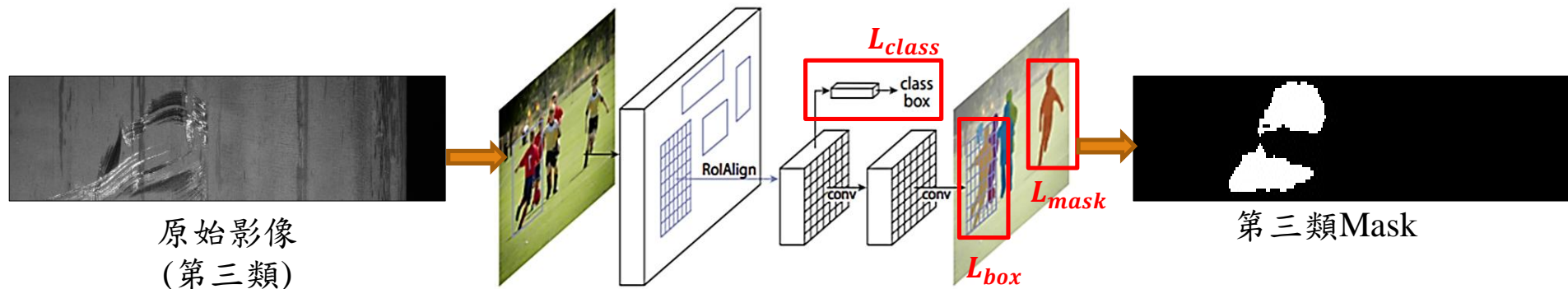
$$L = L_{class} + L_{box} + L_{mask}$$

針對類別( $L_{class}$ )、指示框( $L_{box}$ )、遮罩( $L_{mask}$ )進行損失函數設計



## 四、模型設計 - Mask R-CNN

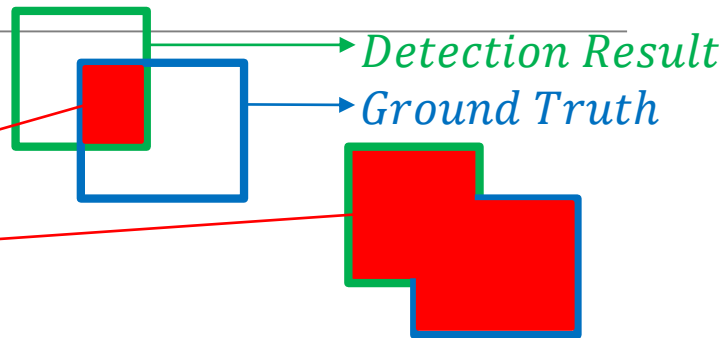
### 5. Mask R-CNN



## 五、實驗結果 - 評估指標說明

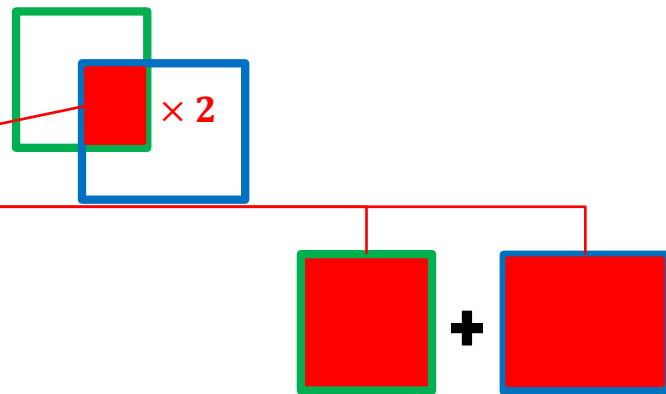
1. IoU (Intersection over Union) :

$$\text{IoU} = \frac{\text{Detection Result} \cap \text{Ground Truth}}{\text{Detection Result} \cup \text{Ground Truth}}$$



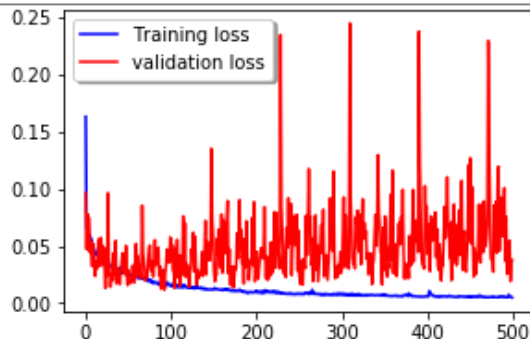
2. Dice coefficient :

$$\text{Dice} = \frac{2 \times (\text{Detection Result} \cap \text{Ground Truth})}{\text{Detection Result} + \text{Ground Truth}}$$



## 五、實驗結果 - U-Net訓練參數與結果

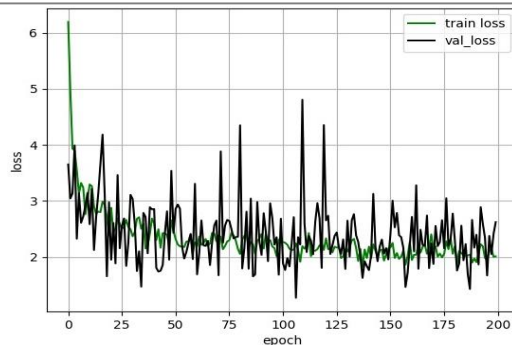
Loss function



U-Net訓練參數	
Optimizer	Adam
Loss function	binary_crossentropy
Epoch	500
Bach size	16
Learning rate	$10^{-4}$
Binarize	0.5

## 五、實驗結果 - Mask R-CNN訓練參數與結果

Loss function



Mask R-CNN訓練參數	
Optimizer	Adam
Loss function	$L = L_{cls} + L_{box} + L_{mask}$
Epochs	200
Bach size	16
Learning rate	$10^{-4}$

## 五、實驗結果 - 演算法結果比較

---

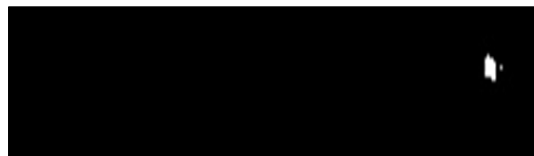
	傳統分割方法			深度學習方法	
	二值化	邊緣檢測	K-means 分群法	U-Net	Mask R-CNN
Mean IoU	0.0505	0.0911	0.0541	0.5513	0.3203
Mean Dice	0.0841	0.1659	0.0827	0.6453	0.4464
FPS	250	340	100	25	5

## 五、實驗結果 - 各瑕疵偵測效果比較 1

- 壓坑(d0328c686.jpg)預測結果:



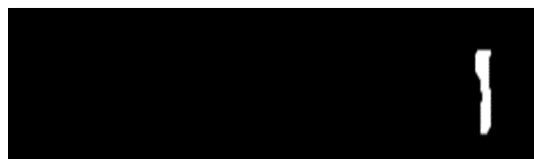
原始影像



Mask R-CNN預測效果



標示瑕疵位置



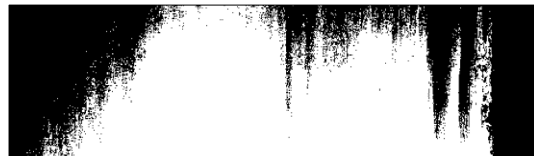
U-Net預測效果



二值化效果



邊緣檢測效果

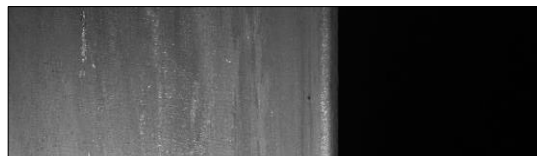


K-means分群法效果

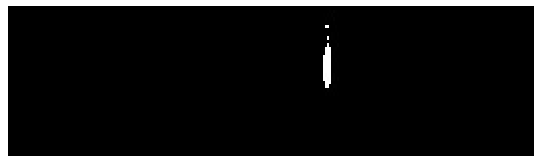
	傳統分割方法			深度學習方法	
	二值化	邊緣檢測	K-means	U-Net	Mask R-CNN
<b>IoU</b>	0.005	0.14	0.01	0.779	0.203
<b>Dice</b>	0.009	0.18	0.02	0.832	0.22

## 五、實驗結果 - 各瑕疵偵測效果比較 2

- 黑色劃痕(e2c4275d6.jpg)預測結果:



原始影像



Mask R-CNN預測效果



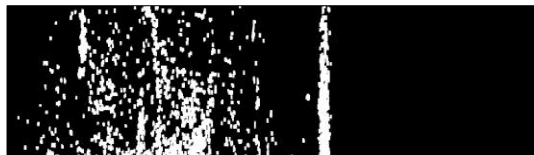
標示瑕疵位置



U-Net預測效果



二值化效果



邊緣檢測效果



K-means分群法效果

	傳統分割方法			深度學習方法	
	二值化	邊緣檢測	K-means	U-Net	Mask R-CNN
<b>IoU</b>	0.005	0.001	0.006	0.286	0.081
<b>Dice</b>	0.005	0.003	0.006	0.311	0.096



## 五、實驗結果 - 各瑕疵偵測效果比較 3

- 白色劃痕(d0328c686.jpg)預測結果:



原始影像



Mask R-CNN預測效果



標示瑕疵位置



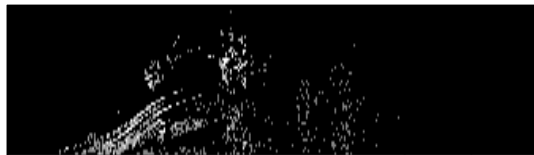
U-Net預測效果



二值化效果



邊緣檢測效果



K-means分群法效果

	傳統分割方法			深度學習方法	
	二值化	邊緣檢測	K-means	U-Net	Mask R-CNN
IoU	0.160	0.163	0.05	0.941	0.721
Dice	0.169	0.17	0.07	0.963	0.828

## 五、實驗結果 - 各瑕疵偵測效果比較 4

- 刮傷(d5a8b7ba4.jpg)預測結果:



原始影像



Mask R-CNN預測效果



標示瑕疵位置



U-Net預測效果



二值化效果



邊緣檢測效果



K-means分群法效果

	傳統分割方法			深度學習方法	
	二值化	邊緣檢測	K-means	U-Net	Mask R-CNN
<b>IoU</b>	0.074	0.061	0.074	0.782	0.573
<b>Dice</b>	0.081	0.069	0.078	0.881	0.601


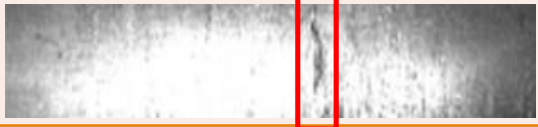

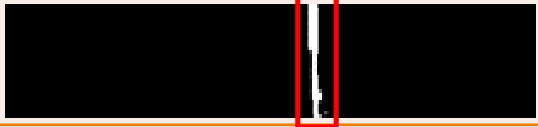


## 五、實驗結果-U-Net與Mask R-CNN偵測效果差異

- U-Net的偵測效果較Mask R-CNN良好且瑕疵偵測較完整。
- 影像中的瑕疵面積較小，U-Net能較有效將瑕疵分割。

原始影像		
瑕疵類別	第一類:壓坑瑕疵	第四類:刮痕瑕疵
Mask R-CNN 偵測效果		
U-Net 偵測效果		

## 五、實驗結果 - 辨識效果不彰與可能原因

- 以效果最佳之U-Net進行示範。
- 成功將瑕疵標示且位置正確，但辨識的類別有誤。由於第三類瑕疵樣本過多(5010張)，其他樣本較少(幾百張)，容易將類別誤判為第三類。

原始影像		
真實瑕疵類別	第一類:壓坑瑕疵	第四類:刮痕瑕疵
真實瑕疵Mask		
U-Net 預測瑕疵類別	第三類:白色劃痕瑕疵	第三類:白色劃痕瑕疵
U-Net 瑕疵偵測結果		

---

# 問題與討論

# 附件1 硬體與環境

---

## 1.訓練用硬體資源(DGX-1)

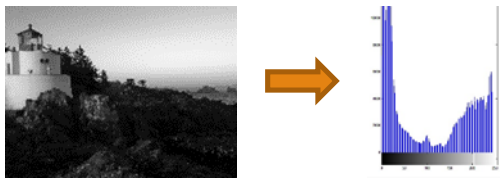
- A.處理器：Dual 20-Core Intel Xeon E5-2698 v4 2.2 GHz
- B.記憶體：512.0 GB
- C.影像處理卡：8 x Tesla V100
- D.作業系統：Ubuntu 18.04

## 2.分析用硬體資源(桌上型)

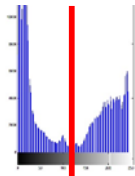
- A.處理器：Intel Core i7-8700 CPU @3.20GHz x 12
- B.記憶體：62.7 GB
- C.影像處理卡：2070
- D.作業系統：Ubuntu 18.04

# 附件2 Otsu公式

Step1. 取得影像直方圖(像素出現機率)



Step 2. 利用Otsu對直方圖找出最佳閾值



Step 3. 利用Otsu找出之閾值將影像二值化



- $\mu_1(T^*) = \frac{\sum_{i=0}^{T^*} h(i) \times i}{\sum_{i=0}^{T^*} h(i)}$
- $\mu_2(T^*) = \frac{\sum_{i=T^*+1}^{T^*} h(i) \times i}{\sum_{i=T^*+1}^{T^*} h(i)}$
- $\sigma_1(T^*) = \sum_{i=0}^{T^*} h(\mu_1 - i)^2 \times h(i)$
- $\sigma_2(T^*) = \sum_{i=T^*+1}^{T^*} h(\mu_2 - i)^2 \times h(i)$
- $\sigma(T^*) = \sigma_1(T^*) + \sigma_2(T^*)$
- $T = \underset{T^*}{\operatorname{argmin}}(\sigma(T^*))$

其中， $T^*$ 為閾值、 $h(i)$ 為每個灰階值出現過的機率

## 附件3 邊緣檢測公式

- 邊緣(Edge)是指周圍灰階急劇變化的那些像素之集合。Sobel運算元使用兩個(3x3)矩陣來進行卷積運算，以計算出兩個方向的灰階差異(水平與垂直)。假定A是原始影像， $G_x$ 和 $G_y$ 分別是在橫向及縱向上對原圖平面卷積的結果， $f(x, y)$ 為影像A中 $(x, y)$ 處的灰階值。

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

$$G_x = [f(x+1, y-1) + 2 * f(x+1, y) + f(x+1, y+1)]$$

$$- [f(x-1, y-1) + 2 * f(x-1, y) + f(x-1, y+1)]$$

$$G_y = [f(x-1, y-1) + 2 * f(x, y-1) + f(x+1, y-1)]$$

$$- [f(x-1, y+1) + 2 * f(x, y+1) + f(x+1, y+1)]$$

- 對於影像中的每個點，其梯度的估計值G便可以透過兩個方向的梯度 $G_x$ 和 $G_y$

$$G = \sqrt{G_x^2 + G_y^2}$$



# 附件3 邊緣檢測公式(續)

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

10	10	10	10	10	10	10
10	10	10	250	10	10	10
10	10	10	250	10	10	10
10	10	10	250	10	10	10
10	10	10	250	10	10	10
10	10	10	250	10	10	10
10	10	10	10	10	10	10

$$G = \sqrt{G_x^2 + G_y^2}$$

$$G_x = (10*1)+(10*2)+(10*1)-(250*1)-(250*2)-(250*1)=-960$$

$$G_y = (10*1)+(10*2)+(250*1)-(10*1)-(10*2)-(250*1)=10$$

$$G = 960$$

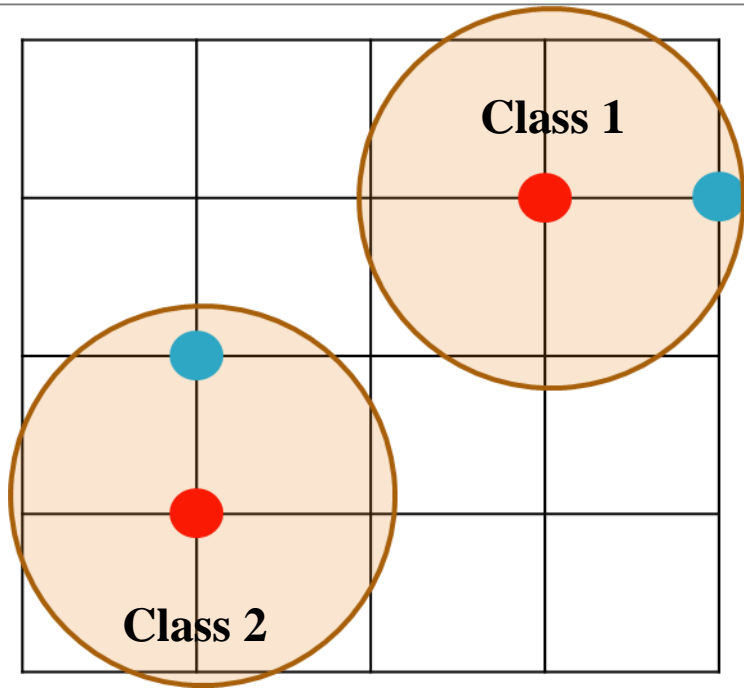
$$G_x = (10*1)+(10*2)+(10*1)-(10*1)-(10*2)-(10*1)=0$$

$$G_y = (10*1)+(10*2)+(10*1)-(10*1)-(10*2)-(10*1)=0$$

$$G = 0$$

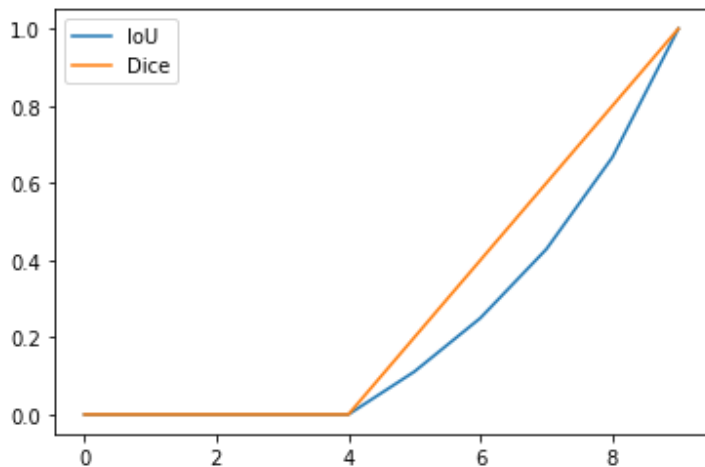
## 附件4 Kmeans範例

- 藍色點為樣本點
- 紅色點為聚類中心點(隨機給定)
- 計算中心點至樣本點的歐式距離
- 將樣本點歸類到最近的聚類中心得到分群結果



## 附件5 評估指標差異

- 藍色區塊代表A；橘色區塊代表B
- 將B區塊向A區塊移動，兩區塊重疊時則呈現綠色
- B區塊每次移動一步計算一次Dice與IoU的數值



1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	1	1	1	1	1

移動0步

1	1	1	1	1	0	0	1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	1	1	1	1	1	0	0	0

移動3步

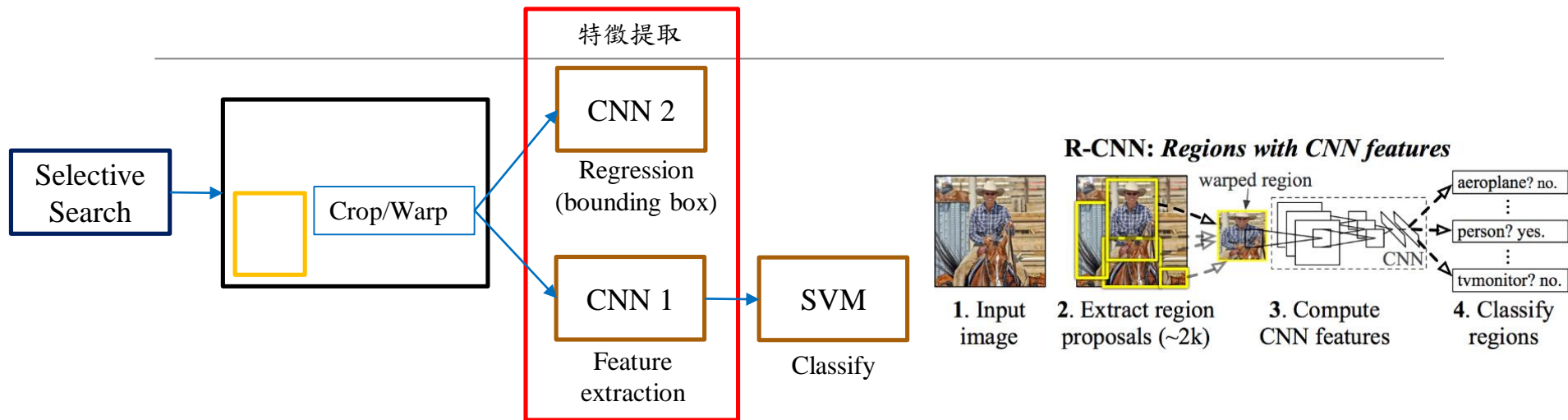
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

移動8步

1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0

移動10步

# 附件6 R-CNN

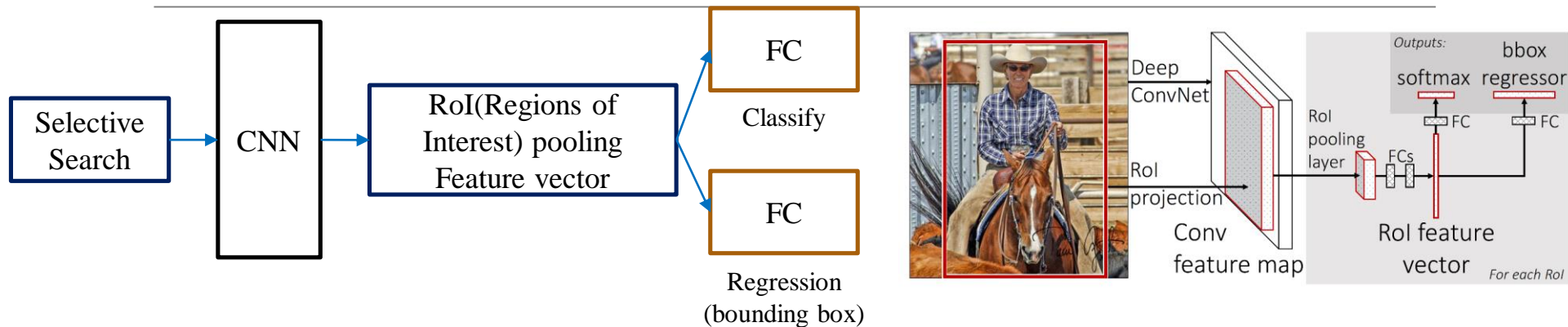


演算法流程:

1. 使用 Selective Search 產生 ~2000 可能區域 (Region proposal)
2. 將每一個 region proposal 經過 Crop/ Warp 後丟入已訓練完成的 CNN 得到固定維度的輸出 (CNN feature)
3. 利用 SVM (Support Vector Machine) 分類器區分屬於哪類
4. 經由線性回歸模型校正 bounding box position

Time per image: 50(s)

# 附件7 Fast R-CNN

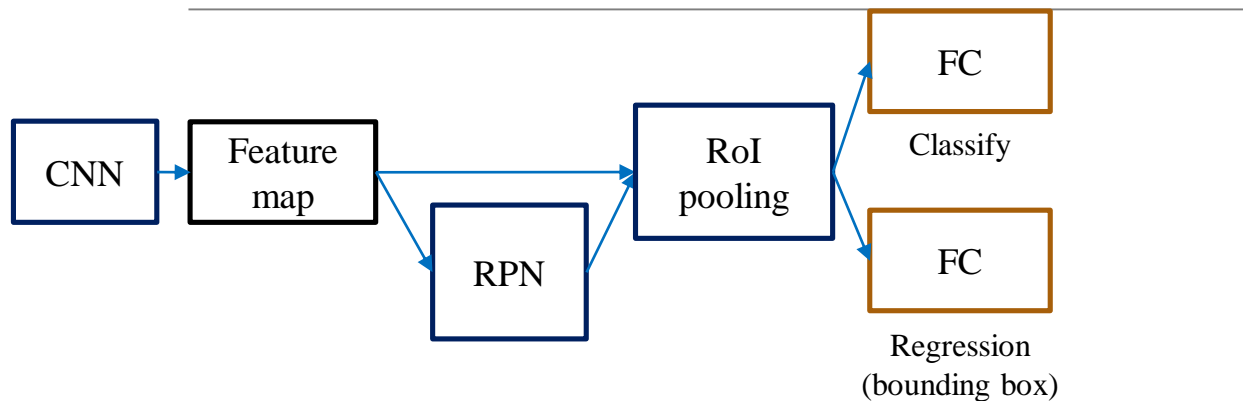


演算法流程:

- 1.使用Selective Search產生~2000可能區域 (Region proposal)
- 2.將每一個region proposal經過CNN得到特徵輸出(CNN feature)
- 3.利用RoI pooling layer統一輸入特徵大小
- 4.使用兩個FC(Fully-connected layer)做分類與邊框計算

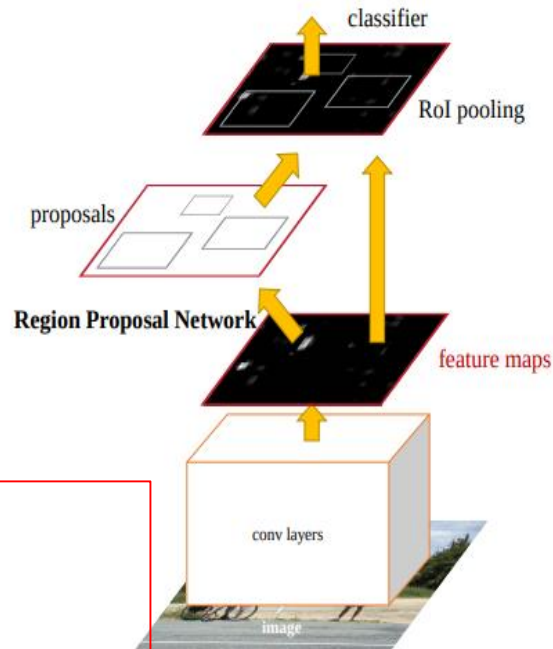
Time per image:2(s)

# 附件7 Faster R-CNN



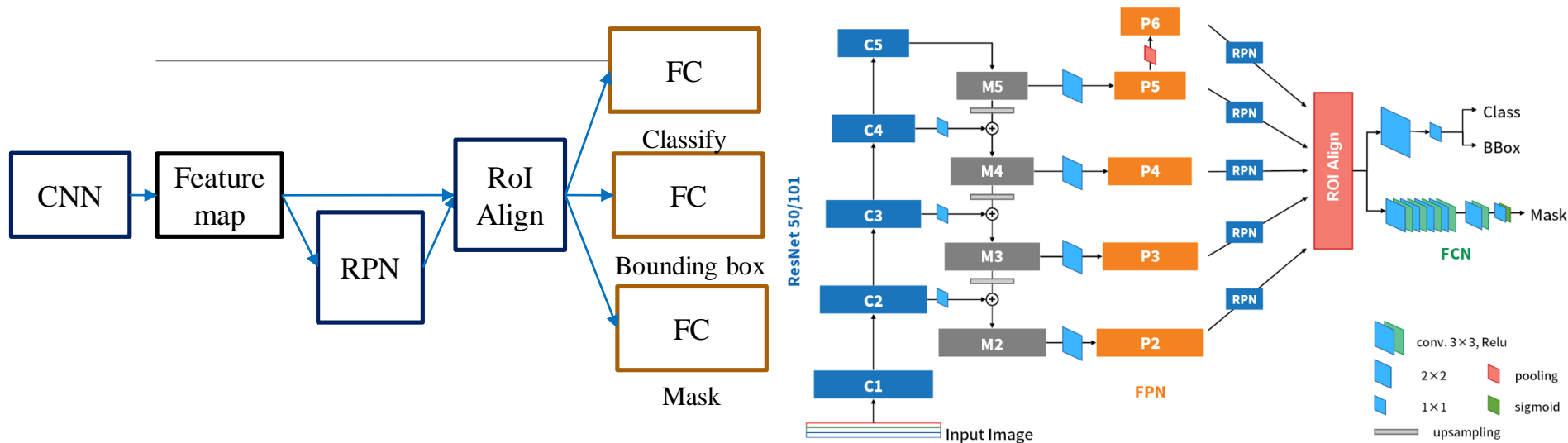
演算法流程:

1. 將輸入影像reshape成固定大小，並以CNN擷取特徵
2. 將特徵透過RPN(Region proposal net work)輸出目標候選框
3. 利用RoI pooling layer統一輸入特徵大小
4. 使用兩個FC(Fully-connected layer)做分類與邊框計算



Time per image:0.2(s)

# 附件8 Mask R-CNN



演算法流程:

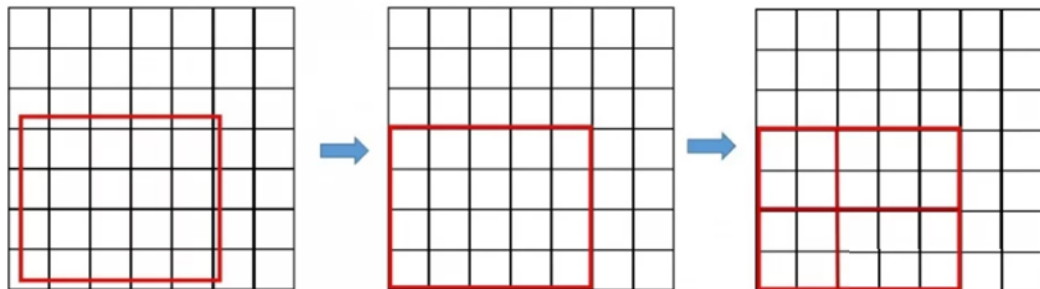
1. 將整張圖片輸入CNN，進行特徵提取
2. 用FPN生成建議視窗(proposals)，每張圖片生成N個建議視窗
3. 把建議視窗對映到CNN的最後一層卷積feature map上
4. 通過RoI Align層使每個RoI生成固定尺寸的feature map
5. 最後利用全連接分類、邊框、mask

Time per image:0.2(s)

# Rol pooling vs Rol Align

---

Rol pooling



Rol Align

