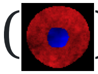
Section 6

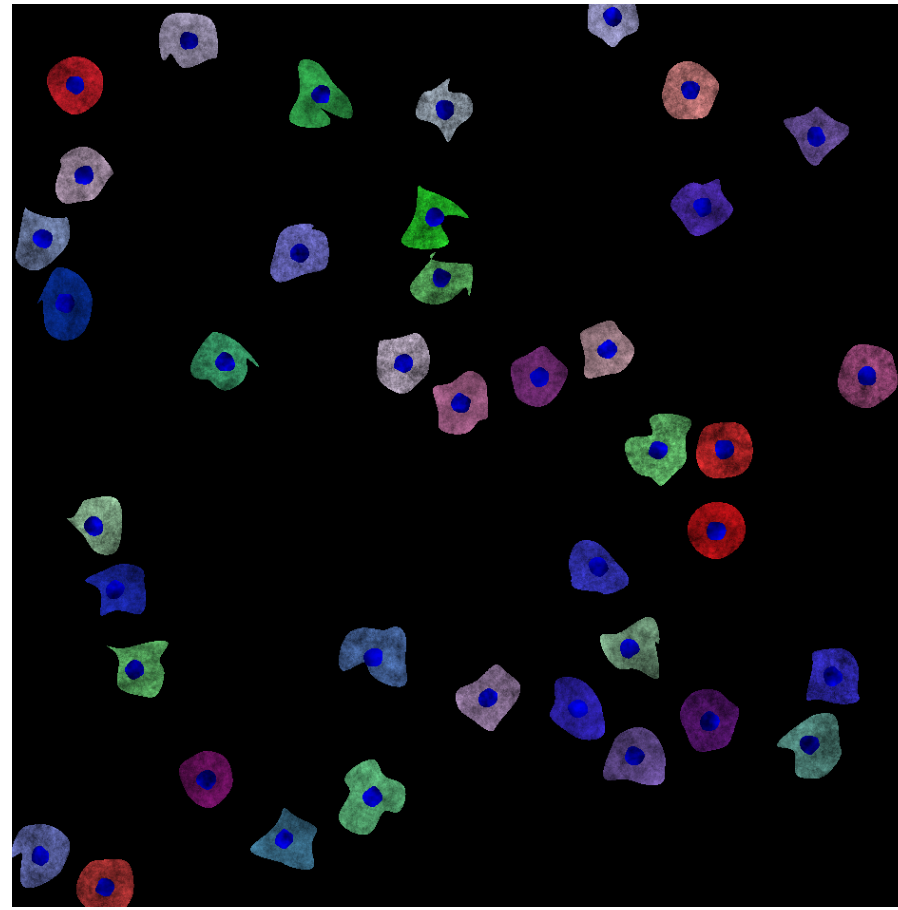
Template Matching and Scale Invariant Feature Descriptors

Presentation by *Asem Alaa*

Template matching

- distance metric
- output: intensity image
- application: crop a single object (e.g cell) and predict where other such similar objects are in the image.
- Not rotation invariant
- good for approximately rounded objects
- good for regular shapes in consistent direction.

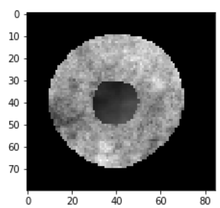
Find () in

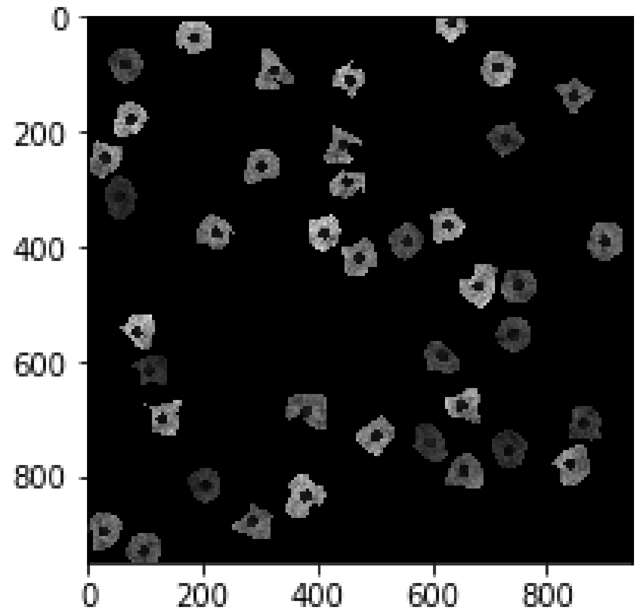


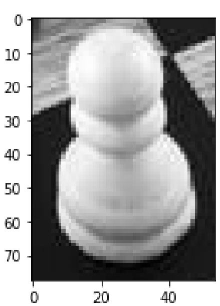
{Dataset: Simulated 24-well plate with synthetic cells (Lehmussola et al., 2007)}

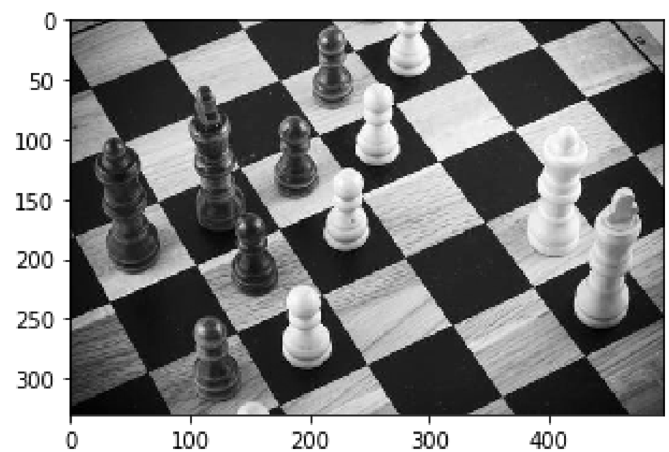
Template matching

Example

$h =$ 

$I =$ 

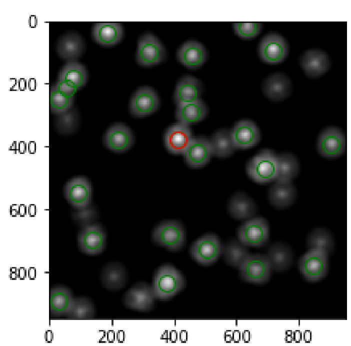
$h =$ 

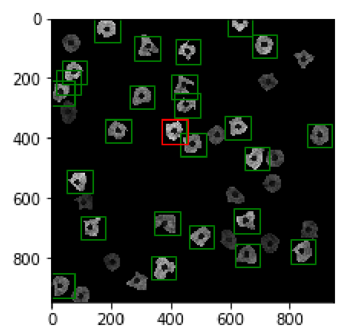
$I =$ 

Method 0: Direct 2D correlation of the image with the template

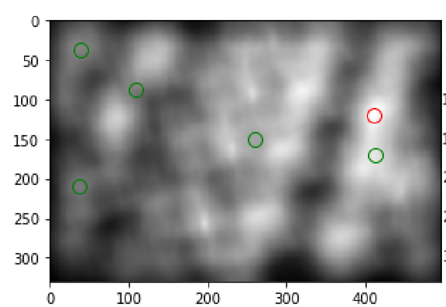
$$y[m,n] = \sum_{k,l} h[k,l]x[m+k,n+l]$$

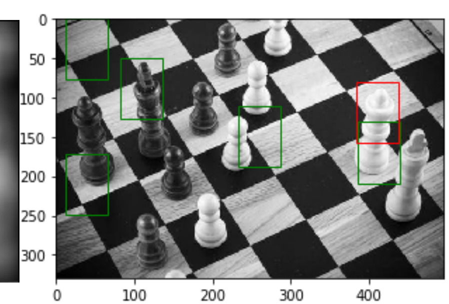
```python
import numpy as np
from scipy.signal import correlate2d
def match_template_corr( x , temp ):
    y = np.empty(x.shape)
    y = correlate2d(x,temp,'same')
    return y
```





$\uparrow\uparrow$ False Positives



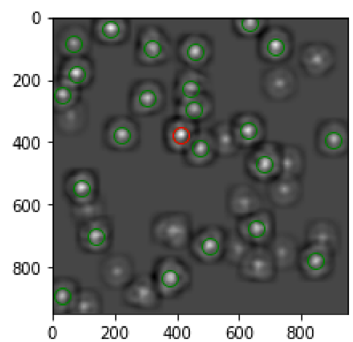


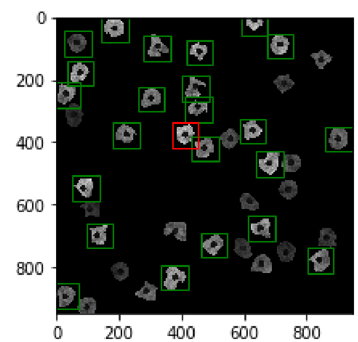
$\uparrow\uparrow$ False Positives

Method 1: Direct 2D correlation of the image with the *zero-mean* template

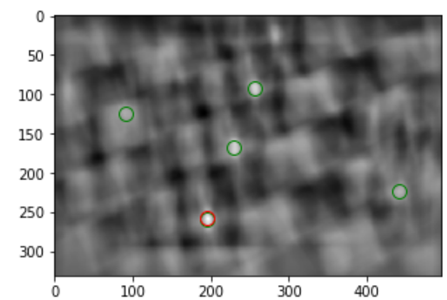
$$y[m,n] = \sum_{k,l} (h[k,l] - \bar{h}) x[m+k, n+l]$$

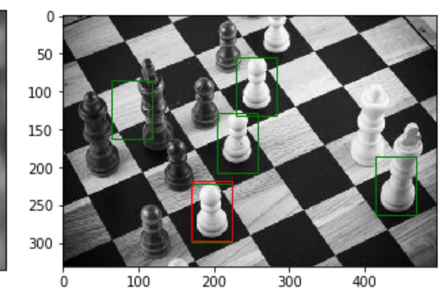
```
def match_template_corr_zmean( x , temp ):
    return match_template_corr(x , temp - temp.mean())
```





fewer FP





$\downarrow$ FP

Method 2: SSD

$$y[m,n] = \sum_{k,l} (h[k,l] - x[m+k, n+l])^2$$

We need to avoid for loops!

$$= \sum_{k,l} h[k,l]^2 - 2\sum_{k,l} h[k,l]x[m+k,n+l] + \sum_{k,l} x[m+k,n+l]^2$$

$\sum_{k,l} h[k,l]^2$	$\sum_{k,l} h[k,l]x[m+k,n+l]$	$\sum_{k,l} x[m+k,n+l]^2$
`np.sum(h*h)`	`correlate2d(x,h)`	`correlate2d(x*x,np.ones(h.shape))`

Method 2: SSD (cont'd)

$$y[m,n] = \sum_{k,l} (h[k,l] - x[m+k, n+l])^2$$

We need to avoid for loops!

$$= \sum_{k,l} h[k,l]^2 - 2\sum_{k,l} h[k,l]x[m+k,n+l] + \sum_{k,l} x[m+k,n+l]^2$$

```
def match_template_ssd( x , temp ):
    term1 = np.sum( np.square( temp ))
    term2 = -2*correlate2d(x, temp,'same')
    term3 = correlate2d( np.square( x ), np.ones(temp.shape),'same' )
    return 1 - np.sqrt(ssd)
```

- Numerical stability?

Method 2: SSD (cont'd)

$$y[m,n] = \sum_{k,l} (h[k,l] - x[m+k, n+l])^2$$

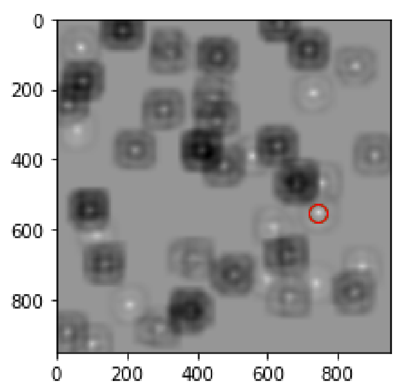
We need to avoid for loops!

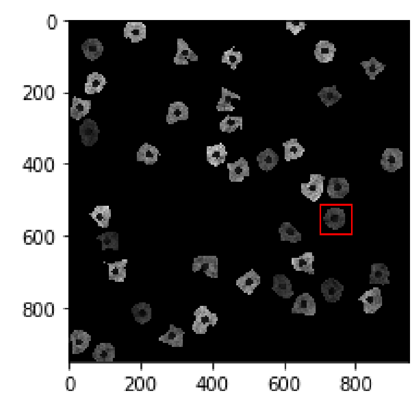
$$= \sum_{k,l} h[k,l]^2 - 2\sum_{k,l} h[k,l]x[m+k,n+l] + \sum_{k,l} x[m+k,n+l]^2$$

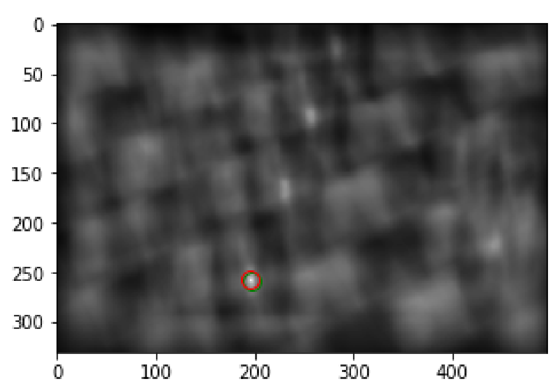
```
def match_template_ssd( x , temp ):
    term1 = np.sum( np.square( temp ))
    term2 = -2*correlate2d(x, temp,'same')
    term3 = correlate2d( np.square( x ), np.ones(temp.shape),'same' )
    ssd = np.maximum( term1 + term2 + term3 , 0 )
    return 1 - np.sqrt(ssd)
```

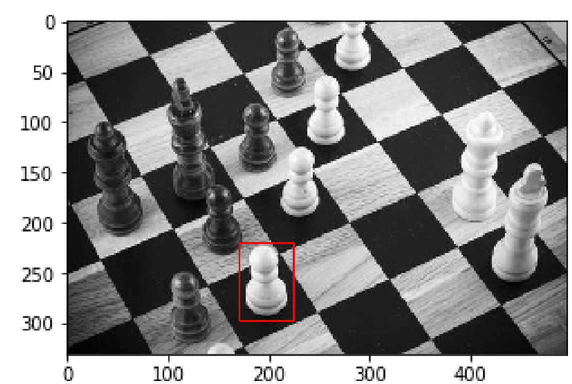
Method 2: SSD (cont'd)

```python
def match_template_ssd( x , temp ):
    term1 = np.sum( np.square( temp ))
    term2 = -2*correlate2d(x, temp,'same')
    term3 = correlate2d( np.square( x ), np.ones(temp.shape),'same' )
    ssd = np.maximum( term1 + term2 + term3 , 0 )
    return 1 - np.sqrt(ssd)
```



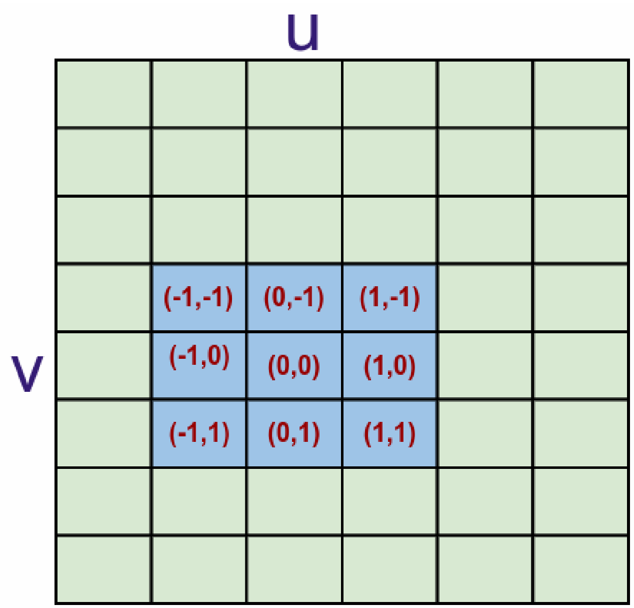






Method 3: Normalized cross-correlation

$$\gamma[u,v] = \frac{\sum_{x,y}(f[x,y]-\bar{f}_{u,v})(t[x-u,y-v]-\bar{t})}{\sqrt{\sum_{x,y}(f[x,y]-\bar{f}_{u,v})^2\sum_{x,y}(t[x-u,y-v]-\bar{t})^2}}$$



In the above image $u=2$, $v=4$, $x \in 1,2,3$, and $y \in 3,4,5$

Method 3: Normalized cross-correlation (cont'd)

$$\gamma[u,v] = \frac{\sum_{x,y}(f[x,y]-\bar{f}_{u,v})(t[x-u,y-v]-\bar{t})}{\sqrt{\sum_{x,y}(f[x,y]-\bar{f}_{u,v})^2\sum_{x,y}(t[x-u,y-v]-\bar{t})^2}}$$

Formula	Python
$f_c = (f[x,y]-\bar{f}_{u,v})$	`f_c=f-correlate2d(f,np.ones(t.shape)/t.size)`
$t_c = (t[x-u,y-v]-\bar{t})$	`t_c=t-t.mean()`
$\sum_{x,y} f_c t_c$	`correlate2d( f_c , t_c , 'same' )`
$\sum_{x,y} f_c^2$	`correlate2d( f_c * f_c , np.ones(t.shape))`
$\sum_{x,y} t_c^2$	`np.sum(t_c * t_c)`

Method 3: Normalized cross-correlation (cont'd)

$$\gamma[u,v] = \frac{\sum_{x,y}(f[x,y]-\bar{f}_{u,v})(t[x-u,y-v]-\bar{t})}{\sqrt{\sum_{x,y}(f[x,y]-\bar{f}_{u,v})^2\sum_{x,y}(t[x-u,y-v]-\bar{t})^2}}$$

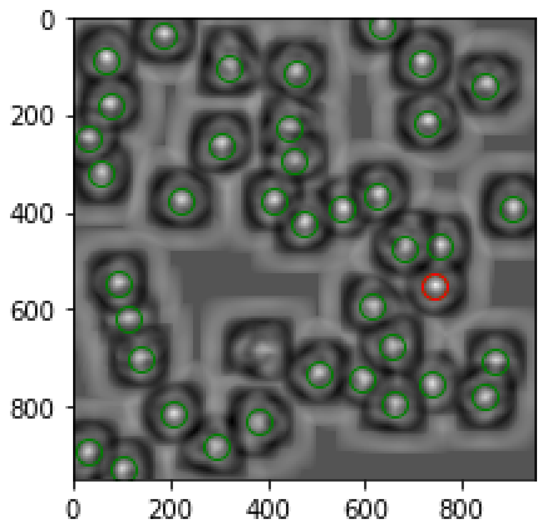
```
def match_template_xcorr( f , t ):
    f_c = f - correlate2d( f , np.ones(t.shape)/np.prod(t.shape), 'same')
    t_c = t - t.mean()
    numerator = correlate2d( f_c , t_c , 'same' )
    d1 = correlate2d( np.square(f_c) , np.ones(t.shape), 'same')
    d2 = np.sum( np.square( t_c ))
    # to avoid sqrt of negative
    denumerator = np.sqrt( np.maximum( d1 * d2 , 0 ))
    return numerator/denumerator
```

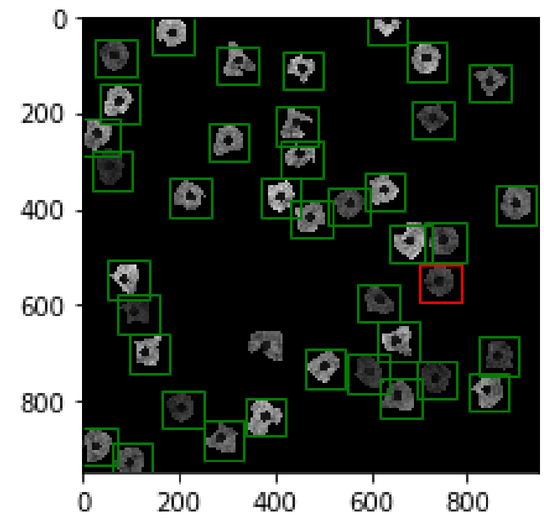
- Division by zero?

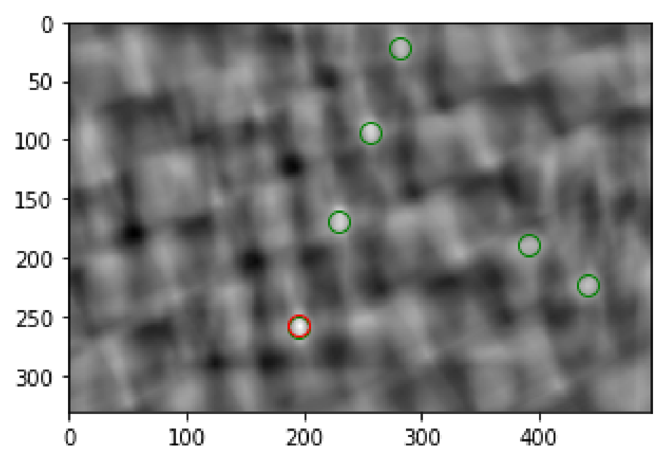
Method 3: Normalized cross-correlation (cont'd)

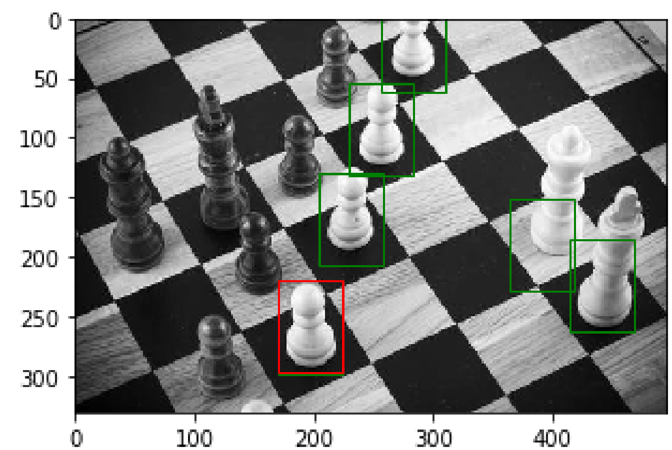
```
def match_template_xcorr( f , t ):
    f_c = f - correlate2d( f , np.ones(t.shape)/np.prod(t.shape), 'same')
    t_c = t - t.mean()
    numerator = correlate2d( f_c , t_c , 'same' )
    d1 = correlate2d( np.square(f_c) , np.ones(t.shape), 'same')
    d2 = np.sum( np.square( t_c ))
    # to avoid sqrt of negative
    denumerator = np.sqrt( np.maximum( d1 * d2 , 0 ))
    response = np.zeros( f.shape )
    # mask to avoid division by zero
    valid = denumerator > np.finfo(np.float32).eps
    response[valid] = numerator[valid]/denumerator[valid]
    return response
```

Method 3: Normalized cross-correlation (cont'd)









{template_matching.ipnyb}