TemplateMatching

March 27, 2020

1 Template matching sous Python

Ce rapport a pour but d'expliquer comment mettre en place un algorithme de template matching sous python. Nous utiliserons les packages `numpy`, `scipy` et `skimage` pour le traitement des images.

[1]:
```
import numpy as np
np.seterr(divide='ignore', invalid='ignore')
import matplotlib.pyplot as plt

from skimage import io, data
from skimage.feature import match_template
from skimage.transform import resize, rotate
from scipy import ndimage
```

1.1 Template matching

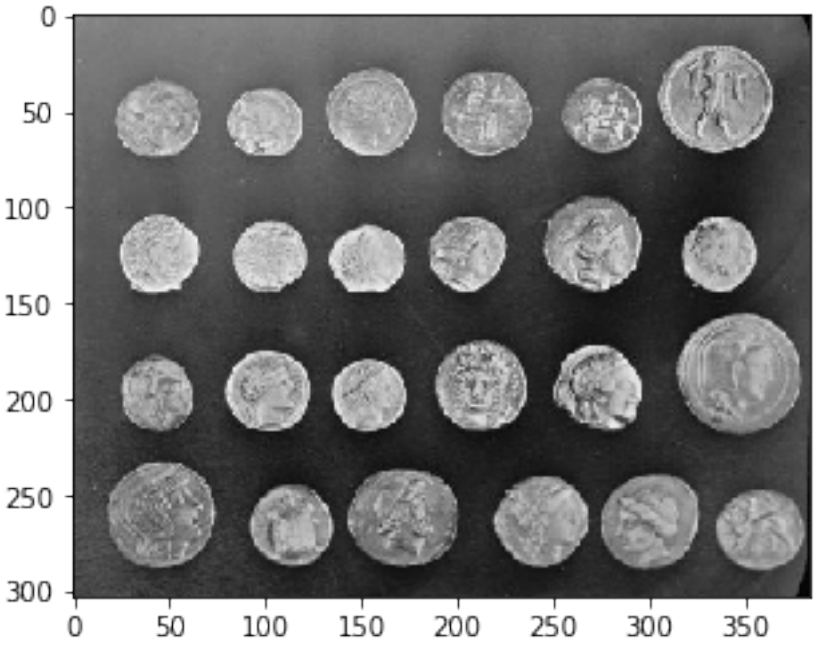
Le template matching est la recherche d'une image ayant valeur de modèle dans une ou plusieurs autres images plus grandes. Elle est utile pour la reconnaissance d'objets et de régions d'intérêt. Le principe est de parcourir l'image avec le modèle. On compare alors le modèle avec la partie de l'image correspondant à chaque fenêtre parcourue, appelée bounding box. On établit un score que l'on attribue au pixel central de cette fenêtre. Ainsi, on obtient une carte de chaleur correspondant à la corrélation entre le modèle et l'endroit de l'image correspondant.

1.2 Application naïve

[2]:
```
image = data.coins()
coin = image[170:220, 75:130] # Une pièce
```

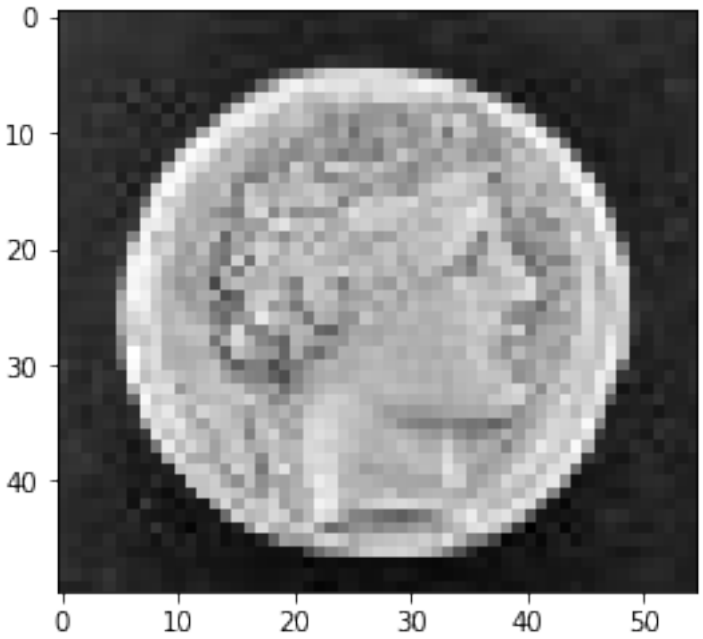
On utilise ici l'image suivante composée de pièces anciennes vues du dessus.

[3]:
```
plt.imshow(image, cmap='gray')
plt.show()
```



On cherche alors à détecter les pièces sur l'image. Pour cela, nous définissons le modèle-image suivant :

[4]:
```
plt.imshow(coin, cmap='gray')
plt.show()
```



Dans cette partie, nous appliquerons directement l'algorithme de template matching sans se préoccuper de la variance du résultat selon la taille et l'angle du modèle.

On définit les dimensions de l'image et du template.

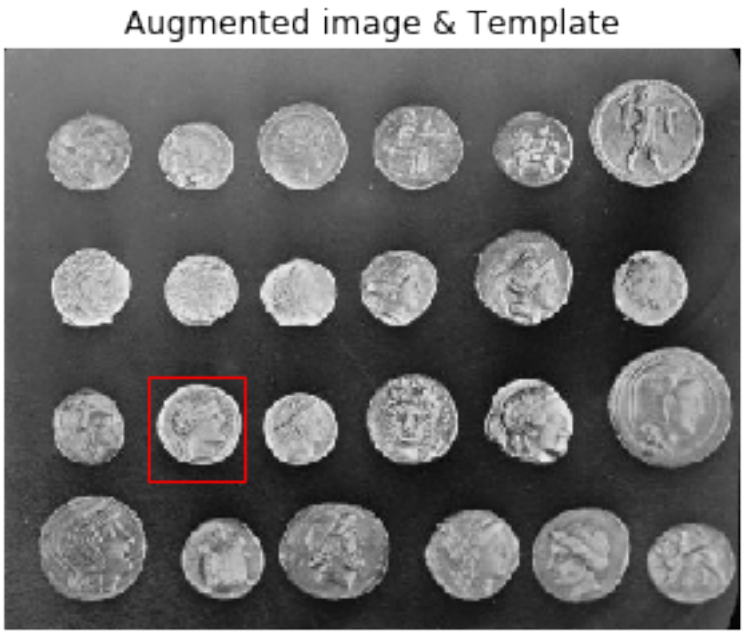
[5]:
```
resolution = np.array(image.shape) # Taille de l'image
window = np.array(coin.shape)      # Taille de la bounding box
```

Nous utilisons la fonction `match_template` de `skimage` pour calculer la carte d'intensités. Les valeurs x et y correspondent à la position où le score est maximum.

[6]:
```
result = match_template(image, coin)
ij = np.unravel_index(np.argmax(result), result.shape)
x, y = ij[::-1]
```

On peut alors représenter l'image avec la bounding box ayant obtenu le score maximal.

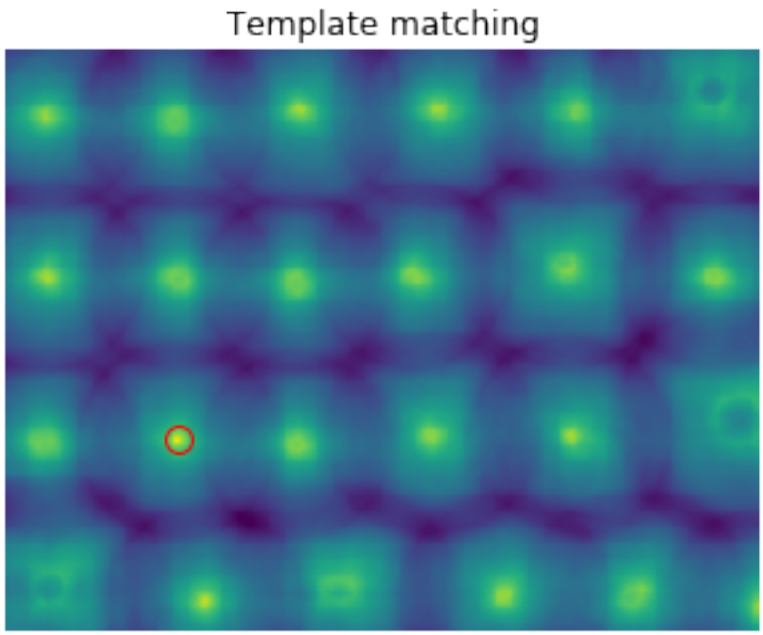
[7]:
```
ax1 = plt.figure().add_subplot(1, 1, 1)
ax1.imshow(image, cmap=plt.cm.gray)
ax1.set_axis_off()
ax1.set_title('Augmented image & Template')
rect = plt.Rectangle((x, y), window[0], window[1], edgecolor='r',␣
 ↪facecolor='none')
ax1.add_patch(rect)
plt.show()
```



Sans surprise c'est la fenêtre que nous avions sélectionnée comme modèle.

Nous pouvons maintenant représenter la carte de chaleur avec la maximum pointé en rouge.

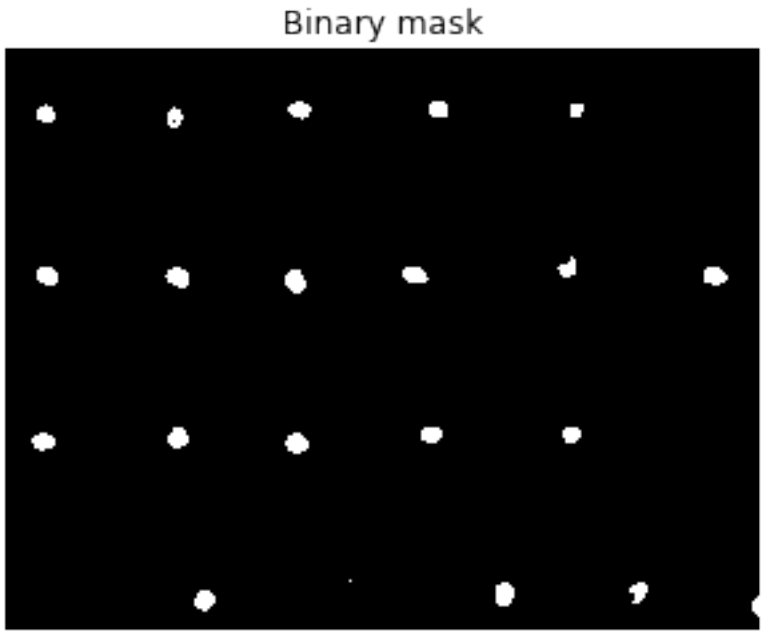
[8]:
```
ax2 = plt.figure().add_subplot(1, 1, 1)
ax2.imshow(result)
ax2.set_axis_off()
ax2.set_title('Template matching')
ax2.autoscale(False)
ax2.plot(x, y, 'o', markeredgecolor='r', markerfacecolor='none', markersize=10)
plt.show()
```



On remarque que les grosses pièces ne sont pas bien détectées par l'algorithme. Cela nous rappelle que si cette méthode est invariante par translation, elle ne l'est pas par contraction/dilatation. De plus, elle ne l'est pas non plus par rotation. Ici, les formes étant rondes, cela ne change rien mais nous verrons ensuite ce cas dans un exemple.

Par seuillage dur, nous cherchons à détecter le maximum d'objets sans que le seuil ne soit trop bas pour ne pas qu'il y ait de fausses détections.

[9]:
```
thresh = result>0.6

ax3 = plt.figure().add_subplot(1, 1, 1)
ax3.imshow(thresh, cmap=plt.cm.gray)
ax3.set_axis_off()
ax3.set_title('Binary mask')
plt.show()
```

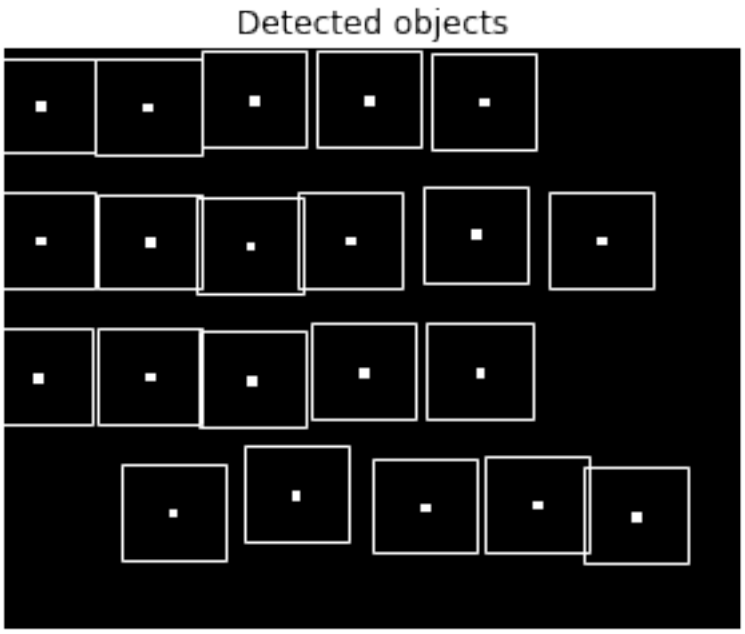


Ici, nous choisissons de seuiller à 0,6.

Nous voulons ensuite récupérer les centres pour chaque objet détecté.

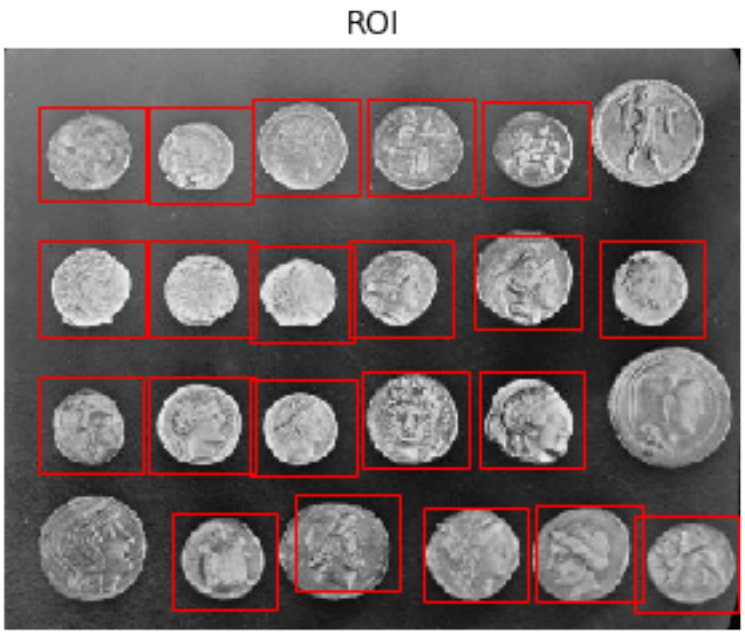
[10]:
```
lbl = ndimage.label(thresh)[0]
pos_tmp = ndimage.measurements.center_of_mass(thresh, lbl, np.arange(100)+1)
pos = []
for i in pos_tmp:
    if not(np.isnan(i[0])):
        pos.append((np.round(i)).astype(int))
M = np.zeros(np.shape(image))
for i in pos:
    for j in range(5):
        for k in range(5):
            M[i[0]-1+j,i[1]-1+k] = 1

ax4 = plt.figure().add_subplot(1, 1, 1)
ax4.imshow(M, cmap=plt.cm.gray)
ax4.set_axis_off()
ax4.set_title('Detected objects')
for i in pos:
    x = i[1]-window[1]//2
    y = i[0]-window[0]//2
    rect = plt.Rectangle((x, y), window[1], window[0], edgecolor='w',␣
 ↪facecolor='none')
    ax4.add_patch(rect)
```

```
plt.show()
```



On obtient alors la carte des objets détectés de l'image. Pour finir, nous souhaitons appliquée cette carte à l'image originale.

[11]:
```
ax5 = plt.figure().add_subplot(1, 1, 1)
ax5.imshow(image, cmap='gray')
ax5.set_axis_off()
ax5.set_title('ROI')
for i in pos:
    rect = plt.Rectangle((i[1], i[0]), window[1], window[0], edgecolor='r',␣
 ↪facecolor='none')
    ax5.add_patch(rect)
plt.show()
```



On remarque alors que si la plupart des pièces sont bien détectées par l'algorithme, les 3 plus grosses pièces, elles, ne le sont pas. Dans la partie suivante, nous allons chercher à optimiser l'algorithme pour qu'il puisse détecter tous les objets de l'image.

1.3 Avec data augmentation

Pour améliorer l'algorithme, nous allons utiliser de la data augmentation sur le modèle. Ainsi, nous souhaitons rendre invariant à certaines transformations la détection des objets. À partir d'un modèle, nous allons donc en obtenir une multitude. La carte de chaleur finale sera obtenue en calculant pour chaque pixel l'intensité maximum des pixels en même position sur les cartes de chaleur obtenues pour chaque modèle utilisé.

1.3.1 Par dilatation/contraction

Dans cette partie, nous conservons les mêmes données que précédemment (image et modèle). Commençons par la data augmentation du modèle. Ici, nous avons des formes rondes donc nous souhaitons rendre la détection invariante à la contraction/dilatation. Les différences de taille entre les pièces ne sont pas énormes. Il s'agira alors d'utiliser des facteurs adaptées à cette échelle.

Si les données sont les mêmes, comme nous allons grossir la taille du modèle, nous rajoutons en bordure d'image, un fond semblable à celui de l'image de la taille du modèle.

[12]:
```
image = data.coins()
coin = image[170:220, 75:130] # Une pièce
resolution = np.array(image.shape)
```

```
window = np.array(coin.shape)
M = np.ones(window+resolution)*70
M[window[0]//2:window[0]//2+resolution[0],window[1]//2:window[1]//
 ↪2+resolution[1]] = image
image = M
resolution = image.shape
```

[13]:
```
templates = []                          # liste de templates
rates = list(np.arange(6)/10+0.8)       # liste des taux de conversion
for t in rates:
    new_size = (np.array(window)*t).astype(int)
    templates.append(resize(coin,new_size))
```

Ici, nous obtenons 6 modèles correspondant au modèle allant de la contraction à 80% de sa taille initiale à la dilatation à 130% de sa taille initiale. Nous pouvons représenter ces 6 modèles :

[14]:
```
N = len(templates)
fig = plt.figure(figsize=(12, N))
axes = []
for i in range(N):
    if i==0:
        axes.append(plt.subplot(1,N,i+1))
    else:
        axes.append(plt.subplot(1,N,i+1,sharex=axes[0],sharey=axes[0]))
    axes[i].imshow(templates[i], cmap='gray')
    axes[i].set_axis_off()
    if i==2:
        axes[i].set_title("Original")
plt.show()
```

Ensuite, nous procédons de la même manière que précédemment, hormis sur la carte de chaleur finale qui est calculée par maximum sur toutes les cartes de chaleur obtenues.

Lors du calcul du template matching, nous obtenons une image dont la taille est inférieure à l'image originale. C'est en fait la taille de l'image originale à laquelle nous soustrayons la taille du modèle. Nous définissons alors la fonction recadrage qui a pour but de bien faire coïncider les images suite à la modification de dimension.

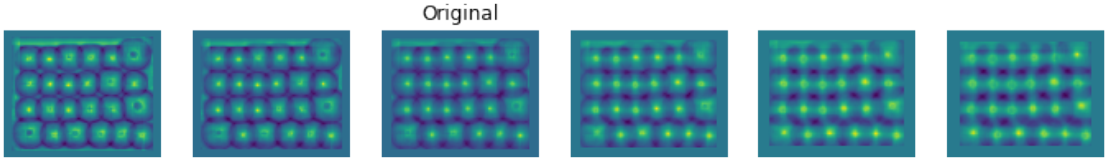
[15]:
```
def recadrage(mat,bigsize):
    smallsize = mat.shape
    M = np.zeros(bigsize)
    x = (bigsize[0]-smallsize[0])//2
    y = (bigsize[1]-smallsize[1])//2
    M[x:x+smallsize[0],y:y+smallsize[1]] = mat
    return M
```

Nous pouvons alors calculer le template matching sur l'image.

[16]:
```
matchs = []
for t in templates :
    m = match_template(image,t)
    matchs.append(recadrage(m,resolution))
matchs = np.array(matchs)
result = np.apply_along_axis(np.max,0,matchs)
```

On peut d'abord regarder la carte de chaleur pour chaque modèle.

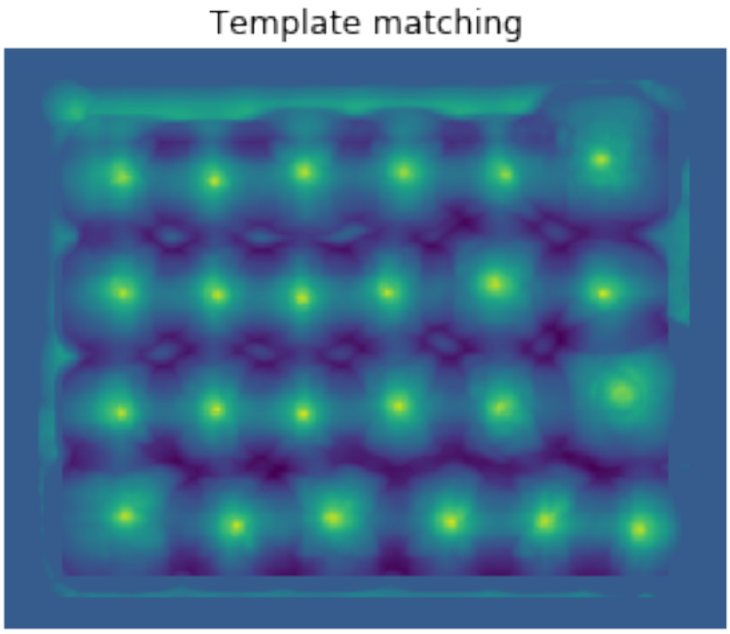
[17]:
```
N = len(templates)
fig = plt.figure(figsize=(12, N))
axes = []
for i in range(N):
    if i==0:
        axes.append(plt.subplot(1,N,i+1))
    else:
        axes.append(plt.subplot(1,N,i+1,sharex=axes[0],sharey=axes[0]))
    axes[i].imshow(matchs[i])
    axes[i].set_axis_off()
    if i==2:
        axes[i].set_title("Original")
plt.show()
```



Par maximum, on obtient la carte de chaleur finale :

[18]:
```
ax2 = plt.figure().add_subplot(1, 1, 1)
ax2.imshow(result)
ax2.set_axis_off()
ax2.set_title('Template matching')
ax2.autoscale(False)
```

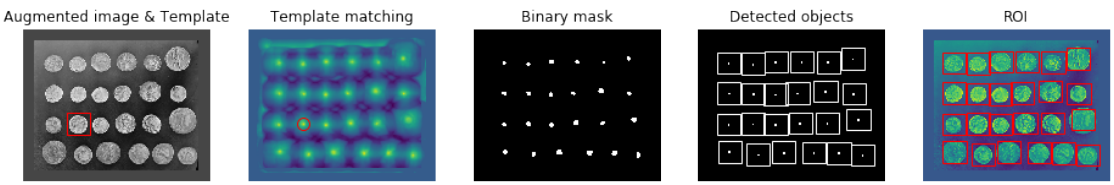
```
plt.show()
```



Ensuite nous procédons de la même manière que précédemment et nous obtenons alors la résultat synthétisé suivant :

[19]:
```
thresh = result>0.6
ij = np.unravel_index(np.argmax(result), result.shape)
x, y = ij[::-1]

lbl = ndimage.label(thresh)[0]
pos_tmp = ndimage.measurements.center_of_mass(thresh, lbl, np.arange(100)+1)
pos = []
for i in pos_tmp:
    if not(np.isnan(i[0])):
        pos.append((np.round(i)).astype(int))
M = np.zeros(np.shape(image))
for i in pos:
    for j in range(5):
        for k in range(5):
            M[i[0]-1+j,i[1]-1+k] = 1

fig = plt.figure(figsize=(16, 5))
ax1 = plt.subplot(1, 5, 1)
ax2 = plt.subplot(1, 5, 2)
ax3 = plt.subplot(1, 5, 3)
ax4 = plt.subplot(1, 5, 4)
```

```
ax5 = plt.subplot(1, 5, 5)

ax1.imshow(image, cmap=plt.cm.gray)
ax1.set_axis_off()
ax1.set_title('Augmented image & Template')
hcoin, wcoin = coin.shape
rect = plt.Rectangle((x-wcoin//2, y-hcoin//2), wcoin, hcoin, edgecolor='r',␣
 ↪facecolor='none')
ax1.add_patch(rect)

ax2.imshow(result)
ax2.set_title('Template matching')
ax2.set_axis_off()
ax2.autoscale(False)
ax2.plot(x, y, 'o', markeredgecolor='r', markerfacecolor='none', markersize=10)

ax3.imshow(thresh, cmap=plt.cm.gray)
ax3.set_axis_off()
ax3.set_title('Binary mask')

ax4.imshow(M, cmap=plt.cm.gray)
ax4.set_axis_off()
ax4.set_title('Detected objects')
for i in pos:
    x = i[1]-wcoin//2
    y = i[0]-hcoin//2
    rect = plt.Rectangle((x, y), wcoin, hcoin, edgecolor='w', facecolor='none')
    ax4.add_patch(rect)

ax5.imshow(image)
ax5.set_axis_off()
ax5.set_title('ROI')
for i in pos:
    x = i[1]-wcoin//2
    y = i[0]-hcoin//2
    rect = plt.Rectangle((x, y), wcoin, hcoin, edgecolor='r', facecolor='none')
    ax5.add_patch(rect)

plt.show()
```



Par cette méthode, nous avons bien obtenu toutes les pièces de l'image.

1.3.2 Par rotation

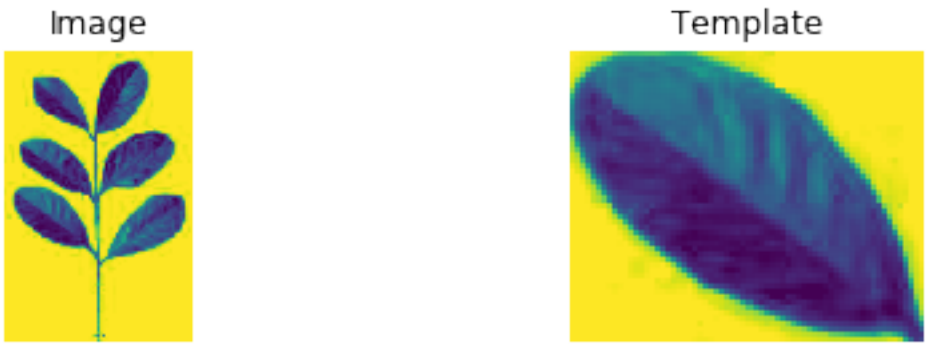
Dans le cas où, la forme n'est pas ronde, nous devons également utiliser différentes rotation du modèle pour qu'il convienne à toutes les position possibles dans l'images.

Nous utiliserons ici l'image et le modèle suivants :

[20]:
```
image_rgb = io.imread("/home/utilisateur/Téléchargements/Seg_leaf/
 ↪TemplateMatching/leafs.jpg")
templ_rgb = io.imread("/home/utilisateur/Téléchargements/Seg_leaf/
 ↪TemplateMatching/leaf.tif")

fig = plt.figure(figsize=(8, 2))
ax1 = plt.subplot(1, 2, 1)
ax2 = plt.subplot(1, 2, 2)

ax1.imshow(image_rgb)
ax1.set_axis_off()
ax1.set_title('Image')

ax2.imshow(templ_rgb)
ax2.set_title('Template')
ax2.set_axis_off()

plt.show()
```

Ces images sont au format RGB, nous devons donc les passer en une dimension. Pour cela, nous conservons simplement la couche couleur Bleue car le fond étant blanc, c'est ce qui différenciera le mieux l'objet du fond.

[21]:
```
image = image_rgb[:,:,2]
templ = templ_rgb[:,:,2]

fig = plt.figure(figsize=(8, 2))
ax1 = plt.subplot(1, 2, 1)
ax2 = plt.subplot(1, 2, 2)

ax1.imshow(image)
ax1.set_axis_off()
ax1.set_title('Image')

ax2.imshow(templ)
ax2.set_title('Template')
ax2.set_axis_off()

plt.show()
```



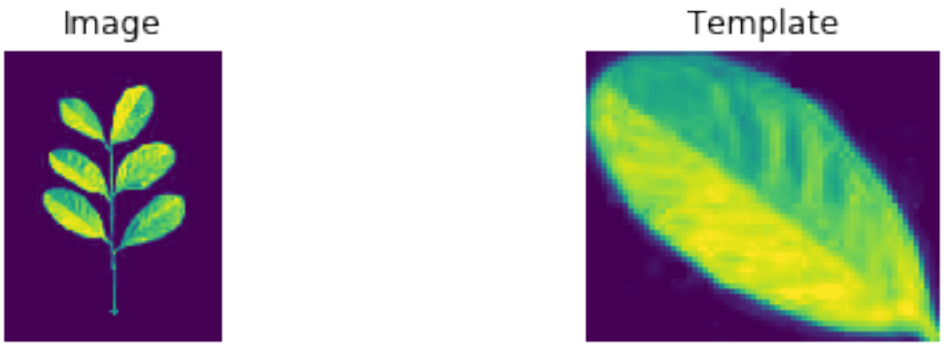
Nous procédons ensuite de la même manière en rajoutant du fond sur les bords.

[22]:
```
resolution = np.array(image.shape)
window = np.array(templ.shape)
M = np.ones(window+resolution)*255
M[window[0]//2:window[0]//2+resolution[0],window[1]//2:window[1]//
 ↪2+resolution[1]] = image
image = M
resolution = image.shape
```

Nous pouvons alors inverser les intensités pour que les fortent intensités correspondent à nos objets.

[23]:
```
image = 255-image
templ = 255-templ

fig = plt.figure(figsize=(8, 2))
ax1 = plt.subplot(1, 2, 1)
```

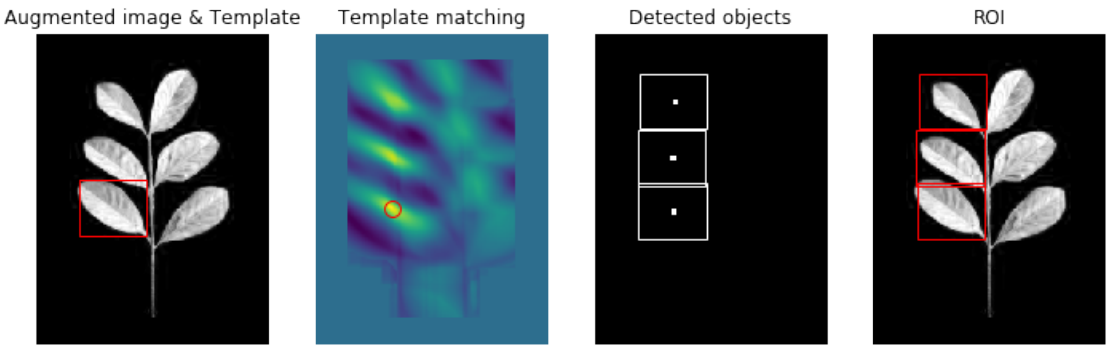
```
ax2 = plt.subplot(1, 2, 2)

ax1.imshow(image)
ax1.set_axis_off()
ax1.set_title('Image')

ax2.imshow(templ)
ax2.set_title('Template')
ax2.set_axis_off()

plt.show()
```



Avec la méthode naïve, nous avons :

[24]:
```
result = match_template(image, templ)
thresh = result>0.6
ij = np.unravel_index(np.argmax(result), result.shape)
x, y = ij[::-1]

lbl = ndimage.label(thresh)[0]
pos_tmp = ndimage.measurements.center_of_mass(thresh, lbl, np.arange(100)+1)
pos = []
for i in pos_tmp:
    if not(np.isnan(i[0])):
        pos.append((np.round(i)).astype(int))
pos = np.array(pos) + np.array(np.shape(templ))//2
M = np.zeros(np.shape(image))
for i in pos:
    for j in range(5):
        for k in range(5):
            M[i[0]-1+j,i[1]-1+k] = 1


fig = plt.figure(figsize=(12, 4))
```

```
ax1 = plt.subplot(1, 4, 1)
ax2 = plt.subplot(1, 4, 2, sharey=ax1)
ax4 = plt.subplot(1, 4, 3)
ax5 = plt.subplot(1, 4, 4)

ax1.imshow(image, cmap=plt.cm.gray)
ax1.set_axis_off()
ax1.set_title('Augmented image & Template')
htempl, wtempl = templ.shape
rect = plt.Rectangle((x, y), wtempl, htempl, edgecolor='r', facecolor='none')
ax1.add_patch(rect)

ax2.imshow(recadrage(result,resolution))
ax2.set_axis_off()
ax2.set_title('Template matching')
ax2.autoscale(False)
ax2.plot(x+wtempl//2, y+htempl//2, 'o', markeredgecolor='r',␣
 ↪markerfacecolor='none', markersize=10)

ax4.imshow(M, cmap=plt.cm.gray)
ax4.set_axis_off()
ax4.set_title('Detected objects')
for i in pos:
    x = i[1]-wtempl//2
    y = i[0]-htempl//2
    rect = plt.Rectangle((x, y), wtempl, htempl, edgecolor='w',␣
 ↪facecolor='none')
    ax4.add_patch(rect)

ax5.imshow(image, cmap=plt.cm.gray)
ax5.set_axis_off()
ax5.set_title('ROI')
for i in pos:
    x = i[1]-wtempl//2
    y = i[0]-htempl//2
    rect = plt.Rectangle((x, y), wtempl, htempl, edgecolor='r',␣
 ↪facecolor='none')
    ax5.add_patch(rect)


plt.show()
```

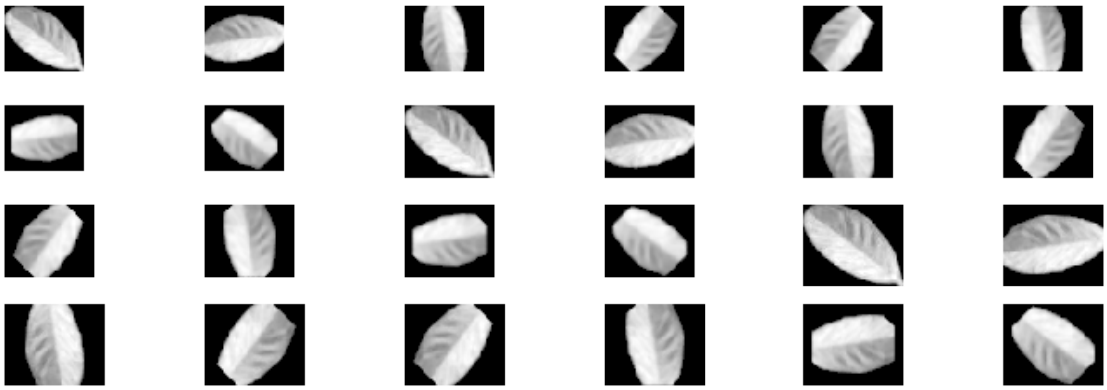


On voit donc que cela ne détecte que les feuilles orientées comme dans le modèle. Nous allons donc faire de la data augmentation, avec de la dilatation/contraction et de la rotation.

[25]:
```
templates = []                          # liste de templates
rates = list(np.arange(3)/10+0.8)       # liste des taux de conversion
angles = list(45*np.arange(8))          # liste des angles de rotation
for t in rates:
    new_size = (np.array(window)*t).astype(int)
    tmp = resize(templ,new_size)
    for theta in angles:
        tmp = rotate(tmp,theta)
        tmp[tmp<0.2] = 0
        templates.append(tmp)
```

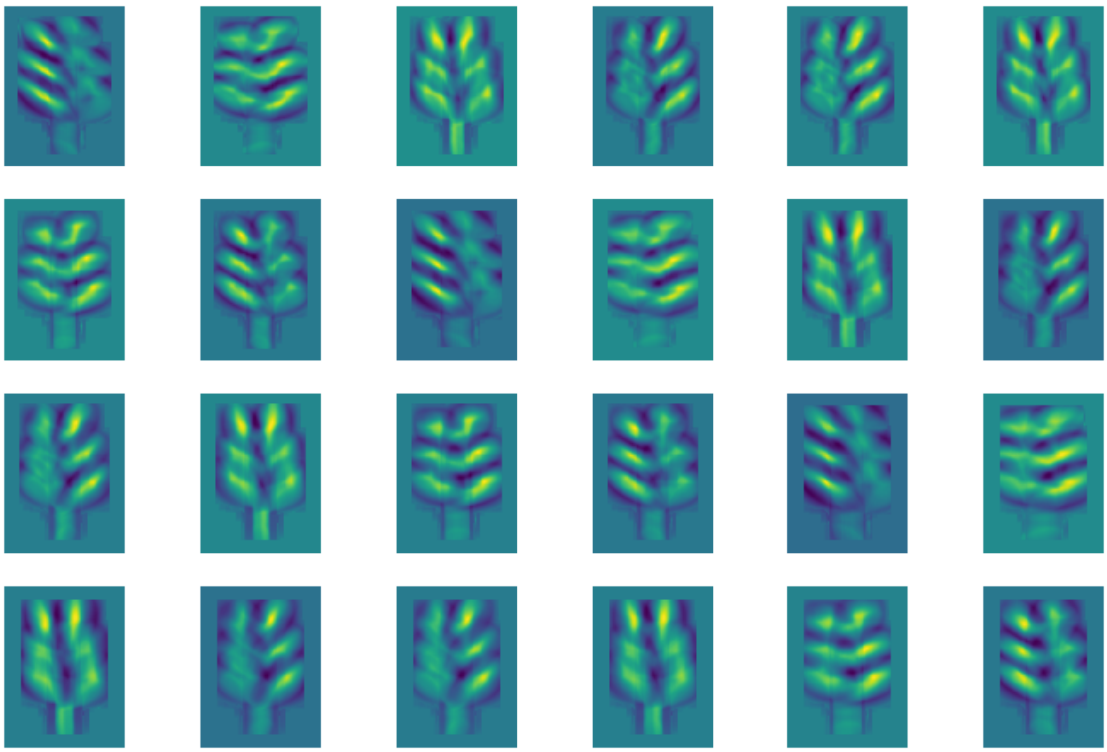
On peut alors représenter tous les modèles obtenus.

[26]:
```
N = len(templates)
fig = plt.figure(figsize=(12, 4))
axes = []
for i in range(N):
    if i==0:
        axes.append(plt.subplot(4,6,i+1))
    else:
        axes.append(plt.subplot(4,6,i+1,sharex=axes[0],sharey=axes[0]))
    axes[i].imshow(templates[i], cmap='gray')
    axes[i].set_axis_off()
plt.show()
```

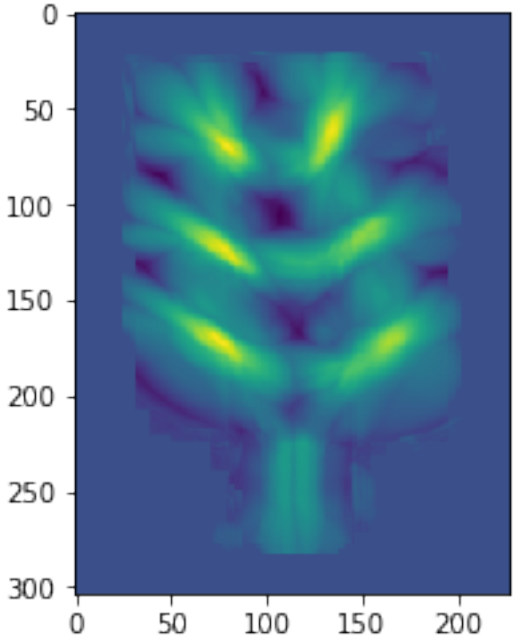


On peut donc appliquer ces modèles pour calculer toutes les cartes de chaleur correspondantes et donc la carte de chaleur finale.

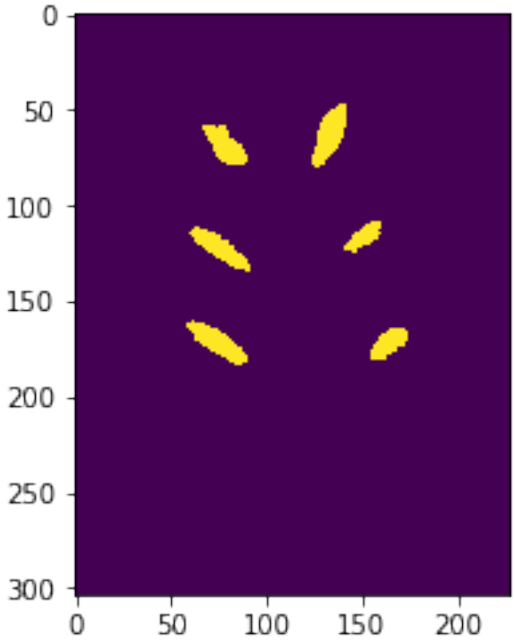
[27]:
```
matchs = []
for t in templates :
    m = match_template(image,t)
    matchs.append(recadrage(m,resolution))
matchs = np.array(matchs)
result = np.apply_along_axis(np.max,0,matchs)

N = len(templates)
plt.figure(figsize=(12,8))
for i in range(N):
    plt.subplot(4,6,i+1)
    plt.imshow(matchs[i])
    plt.axis("off")
plt.show()
```



[28]:
```
plt.imshow(result)
plt.show()
```

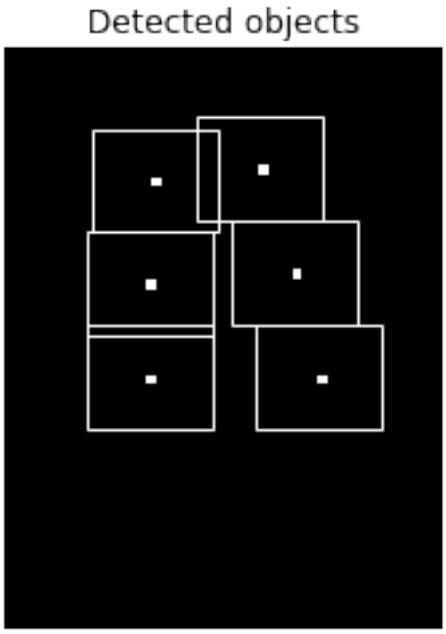


[29]:
```
thresh = result>0.6

plt.imshow(thresh)
plt.show()
```

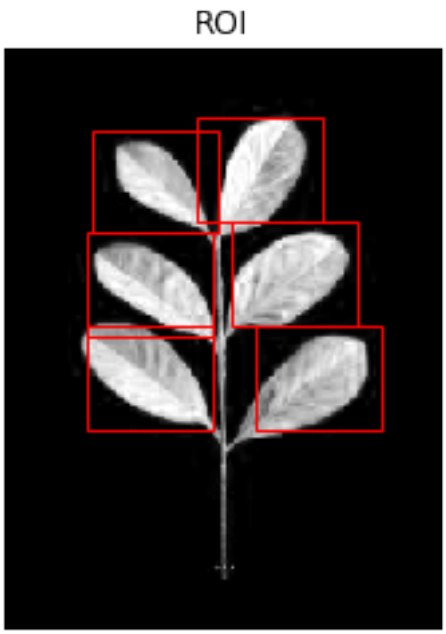


[30]:
```
ij = np.unravel_index(np.argmax(result), result.shape)
x, y = ij[::-1]

lbl = ndimage.label(thresh)[0]
pos_tmp = ndimage.measurements.center_of_mass(thresh, lbl, np.arange(100)+1)
pos = []
for i in pos_tmp:
    if not(np.isnan(i[0])):
        pos.append((np.round(i)).astype(int))
M = np.zeros(np.shape(image))
for i in pos:
    for j in range(5):
        for k in range(5):
            M[i[0]-1+j,i[1]-1+k] = 1

ax4 = plt.figure().add_subplot(1, 1, 1)
ax4.imshow(M, cmap=plt.cm.gray)
ax4.set_axis_off()
ax4.set_title('Detected objects')
```

```
for i in pos:
    x = i[1]-wtempl//2
    y = i[0]-htempl//2
    rect = plt.Rectangle((x, y), wtempl, htempl, edgecolor='w',␣
 ↪facecolor='none')
    ax4.add_patch(rect)
plt.show()
```



[31]:
```
ax5 = plt.figure().add_subplot(1, 1, 1)
ax5.imshow(image, cmap=plt.cm.gray)
ax5.set_axis_off()
ax5.set_title('ROI')
for i in pos:
    x = i[1]-wtempl//2
    y = i[0]-htempl//2
    rect = plt.Rectangle((x, y), wtempl, htempl, edgecolor='r',␣
 ↪facecolor='none')
    ax5.add_patch(rect)
plt.show()
```



Ainsi, nous avons bien réussi à détecter toutes les feuilles de l'image par template matching associée à une étape antérieure de data augmentation.