4TN4 Assignment 3

Adam Bujak (400113347)

February 13, 2022

1 Theory

1.1 Edge Detection, Sobel Operator

Sobel operators are masks used to calculate approximations of the derivative in the x-direction and y-direction. Apply the Sobel operators to the given image. Approximate magnitude and phase of the gradient at each pixel position. (Zero-Padding is NOT required for this question)

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

To calculate the magnitude of the phase of the gradient at each pixel position we can calculate the gradient in the x and y directions and then apply the following formulas to get the phase, and magnitude:

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

$$\phi = atan2(G_y, G_x)$$

(atan2 offers easier application than the simpler phase calculation: $\phi = atan(G_y/G_x)$) when using not doing calculations by hand

To get G_x and G_y at each pixel we convolve the image at the desired pixel with the horizontal (Figure 1) and vertical (Figure **??**) Sobel Filter kernels respectively.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Figure 1: Horizontal Sobel Filter (x-direction)

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figure 2: Vertical Sobel Filter (y-direction)

For example, the calculation of convolution of the Sobel operator in the x-direction and the image at pixel [3,2] ([x,y] with indices starting at 1) would look like this:

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

$$G_x(3,2) = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & -1 \end{bmatrix} * I(3,2)$$

$$G_x(3,2) = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$G_x(3,2) = -1$$

where I is the image.

Doing this for all pixels in the image, we get the following image for G_x:

0	0	0	0	0	0	0	0	0	0	0	0
0	0	-1	-1	0	0	0	0	0	1	1	0
0	0	-2	-3	-1	0	0	0	0	3	3	0
0	0	-1	-3	-3	-1	0	0	0	4	4	0
0	0	0	-1	-3	-3	-1	0	0	4	4	0
0	-1	0	1	-1	-3	-3	-1	0	4	4	0
0	-2	-1	2	1	-1	-3	-3	-1	4	4	0
0	-1	-2	0	2	1	-1	-3	-3	3	4	0
0	0	-1	-2	0	2	1	-1	-3	1	3	0
0	0	0	-1	-2	1	2	0	-1	0	1	0
0	0	0	0	-1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Similarly, doing this for all pixels in the image in the vertical direction, we get the following image for G_y:

0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	3	4	4	4	4	4	3	1	0
0	0	0	1	3	4	4	4	4	3	1	0
0	0	-1	-3	-3	-1	0	0	0	0	0	0
0	0	0	-1	-3	-3	-1	0	0	0	0	0
0	1	2	1	-1	-3	-3	-1	0	0	0	0
0	0	1	2	1	-1	-3	-3	-1	0	0	0
0	-1	-2	0	2	1	-1	-3	-3	-1	0	0
0	0	-1	-2	0	2	1	-1	-3	-3	-1	0
0	0	0	-1	-2	-1	0	0	-1	-2	-1	0
0	0	0	0	-1	-2	-1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Using the formulas above we can find $|\nabla G|$ and ϕ.

0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	1.41	3.16	4.00	4.00	4.00	4.00	4.00	3.16	1.41	0.00
0.00	0.00	2.00	3.16	3.16	4.00	4.00	4.00	4.00	4.24	3.16	0.00
0.00	0.00	1.41	4.24	4.24	1.41	0.00	0.00	0.00	4.00	4.00	0.00
0.00	0.00	0.00	1.41	4.24	4.24	1.41	0.00	0.00	4.00	4.00	0.00
0.00	1.41	2.00	1.41	1.41	4.24	4.24	1.41	0.00	4.00	4.00	0.00
0.00	2.00	1.41	2.83	1.41	1.41	4.24	4.24	1.41	4.00	4.00	0.00
0.00	1.41	2.83	0.00	2.83	1.41	1.41	4.24	4.24	3.16	4.00	0.00
0.00	0.00	1.41	2.83	0.00	2.83	1.41	1.41	4.24	3.16	3.16	0.00
0.00	0.00	0.00	1.41	2.83	1.41	2.00	0.00	1.41	2.00	1.41	0.00
0.00	0.00	0.00	0.00	1.41	2.00	1.41	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 1: $|\nabla G|$

0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	2.36	1.89	1.57	1.57	1.57	1.57	1.57	1.25	0.79	0.00
0.00	0.00	3.14	2.82	1.89	1.57	1.57	1.57	1.57	0.79	0.32	0.00
0.00	0.00	-2.36	-2.36	-2.36	-2.36	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	-2.36	-2.36	-2.36	-2.36	0.00	0.00	0.00	0.00	0.00
0.00	2.36	1.57	0.79	-2.36	-2.36	-2.36	-2.36	0.00	0.00	0.00	0.00
0.00	3.14	2.36	0.79	0.79	-2.36	-2.36	-2.36	-2.36	0.00	0.00	0.00
0.00	-2.36	-2.36	0.00	0.79	0.79	-2.36	-2.36	-2.36	-0.32	0.00	0.00
0.00	0.00	-2.36	-2.36	0.00	0.79	0.79	-2.36	-2.36	-1.25	-0.32	0.00
0.00	0.00	0.00	-2.36	-2.36	-0.79	0.00	0.00	-2.36	-1.57	-0.79	0.00
0.00	0.00	0.00	0.00	-2.36	-1.57	-0.79	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 2: ϕ in radians

1.2 Edge Detection, Canny Operator

As described in the previous section, we can calculate the magnitude and phase of the gradient of the image using the Sobel operators and after normalizing to a max value of 255:

80.6	180.3	228.1	180.3	80.6	0.0	0.0	0.0
114.0	180.3	180.3	255.0	241.9	80.6	0.0	0.0
80.6	241.9	243.7	66.7	191.8	184.7	50.8	0.0
0.0	80.6	214.2	214.2	0.0	214.2	214.2	80.6
0.0	0.0	50.8	184.7	191.8	66.7	247.2	180.3
0.0	0.0	0.0	80.6	241.9	255.0	241.9	180.3
0.0	0.0	0.0	0.0	80.6	180.3	180.3	80.6
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 3: $|\nabla G|$

135	108	90	72	45	180	180	180
180	162	108	63	45	45	180	180
225	225	232	288	27	36	45	180
180	225	233	233	180	53	53	45
180	180	225	216	207	108	33	18
180	180	180	225	225	243	315	342
180	180	180	180	225	252	288	315
180	180	180	180	180	180	180	180

Table 4: ϕ in degrees

Then by observing the phase of the gradient, we can apply non maximum suppression (NMS) to reduce the edges to thinner lines. This works by considering the pixels around each pixel in the magnitude of the gradient along the line defined by the phase of the gradient. If there is a more intense pixel on the line defined by the gradient, the current pixel is set to 0, otherwise it is left unchanged.

Applying this to each pixel in the gradient, we get:

0	0	0	0	0	0	0	0
0	0	0	255	242	0	0	0
0	242	0	0	0	185	0	0
0	0	214	214	0	214	214	0
0	0	0	185	0	0	0	0
0	0	0	0	242	255	242	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Following NMS, we apply thresholding, where any value below 20 is set to 0, any value above 120 is set to 255 and any value between is flagged as weak and set to 50.

Following thresholding, we can apply hysteresis thresholding to detect the edges. Hysteresis thresholding works by removing all weak edges that are not connected to strong edges.

0	0	0	0	0	0	0	0
0	0	0	255	255	0	0	0
0	255	0	0	0	255	0	0
0	0	255	255	0	255	255	0
0	0	0	255	0	0	0	0
0	0	0	0	255	255	255	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

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

The block diagram of the Canny operator can be seen below:

Figure 3: Canny Flowchart

The Canny operator will result in 1 pixel width lines always because of the NMS step which reduces all lines to single pixel with lines, since only the maximum value is taken from each line.

1.3 Convolution Theorem

Convolution Theorem states that convolution of two signals in the spatial domain is equivalent of the multiplication of the two signals' transforms in the frequency domain, and vice versa.

This could be useful in template matching because to find a template quickly, one can take the Fourier transform of the template and the image, and then simply do a multiplication of the two Fourier domain signals, to execute template matching, rather than performing a convolution over an entire image, which is a costly operation.

1.4 Sharpening an Image

A way to attain a high pass filter from a low pass gaussian filter is simply to subtract the low pass filter from an "all-pass" filter (which simply is all 1s, and passes every frequency equally). By simply convolving the image with the high pass filter derived above, one would find that the resulting image is sharper.

2 Implementation

2.1 DoG

Figure 4: Gaussian Edge Detection (lines are faint - it is not pure black)

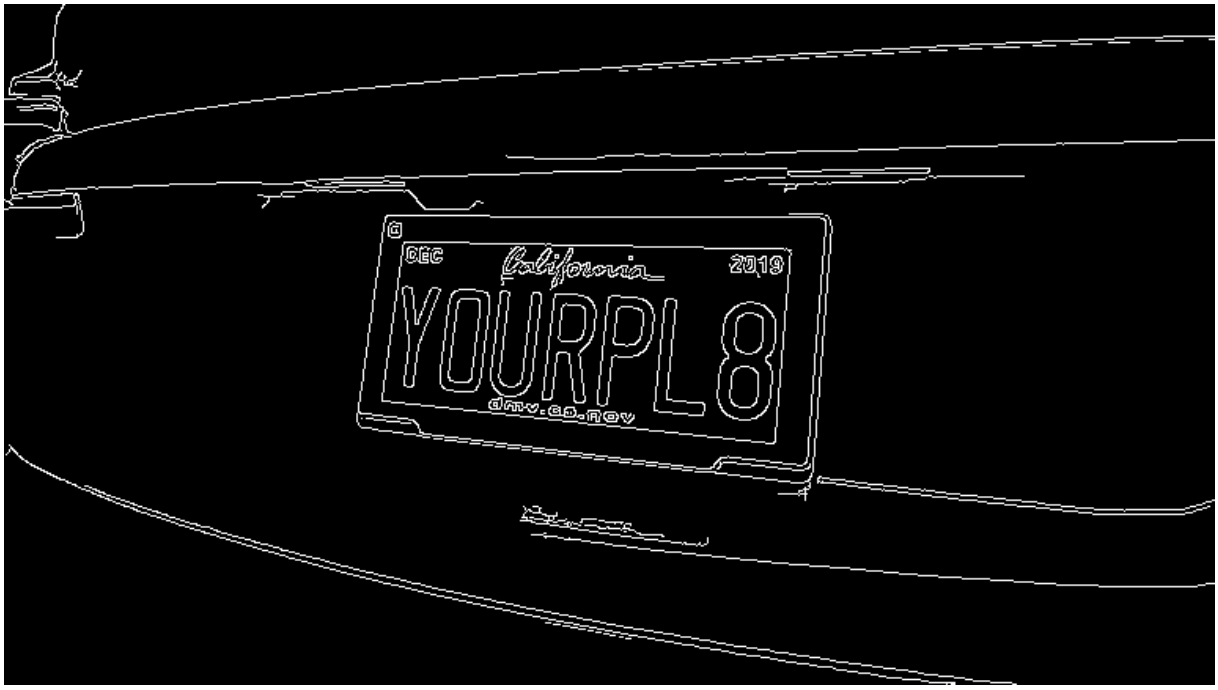

Figure 5: Canny Edge Detection

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

#read image
img = cv2.imread('lp.jpg', 0)

blurred_small = cv2.GaussianBlur(img, (3, 3), 0)
blurred_big = cv2.GaussianBlur(img, (7, 7), 0)

output = (cv2.subtract(blurred_small, blurred_big))
cv2.imwrite('q1_dog.png', output)

output = cv2.Canny(img, 20, 120)
cv2.imwrite('q1_canny.png', output)
```

Figure 6: Code

2.2 2.

If we know the scale of our object the DoG method is better since we can fine tune it to only include objects of a certain size by varying the smaller and bigger variances of the kernel.

With the Canny method all we can specify is the lower and upper bounds during the thresholding stage, which is irrelevant of scale of objects.

2.3 Template Matching

In the heatmap in Figure 7 we see that the cross correlation template matching produced the highest value in the top left of the image, precisely in the middle of the soccer ball. This is the peak of the heatmap and this shows that this is where the template matching matched the most - therefore, is likely what we are looking for, and in this case it is.

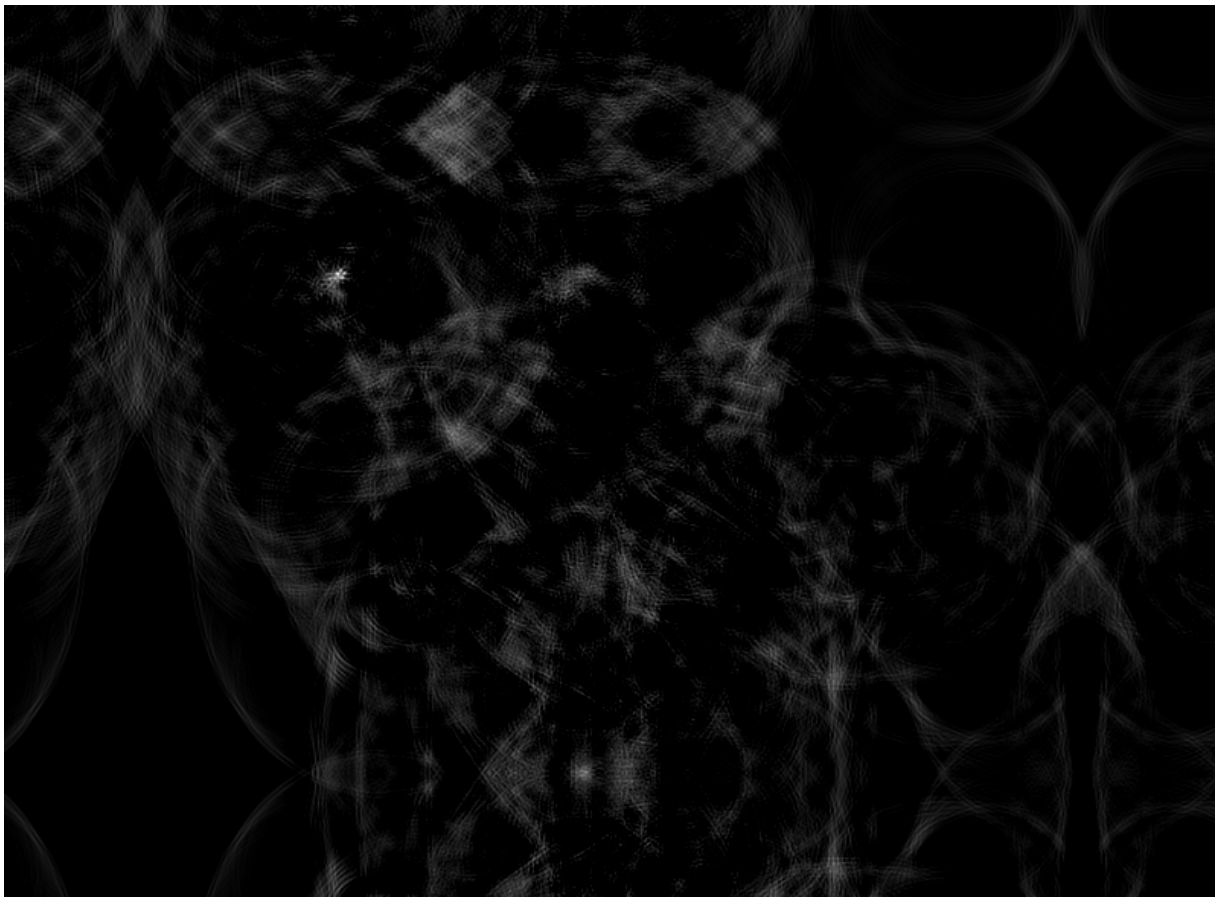

Figure 7: Cross Correlation Heatmap

Figure 8: Cross Correlation Template Matching Results

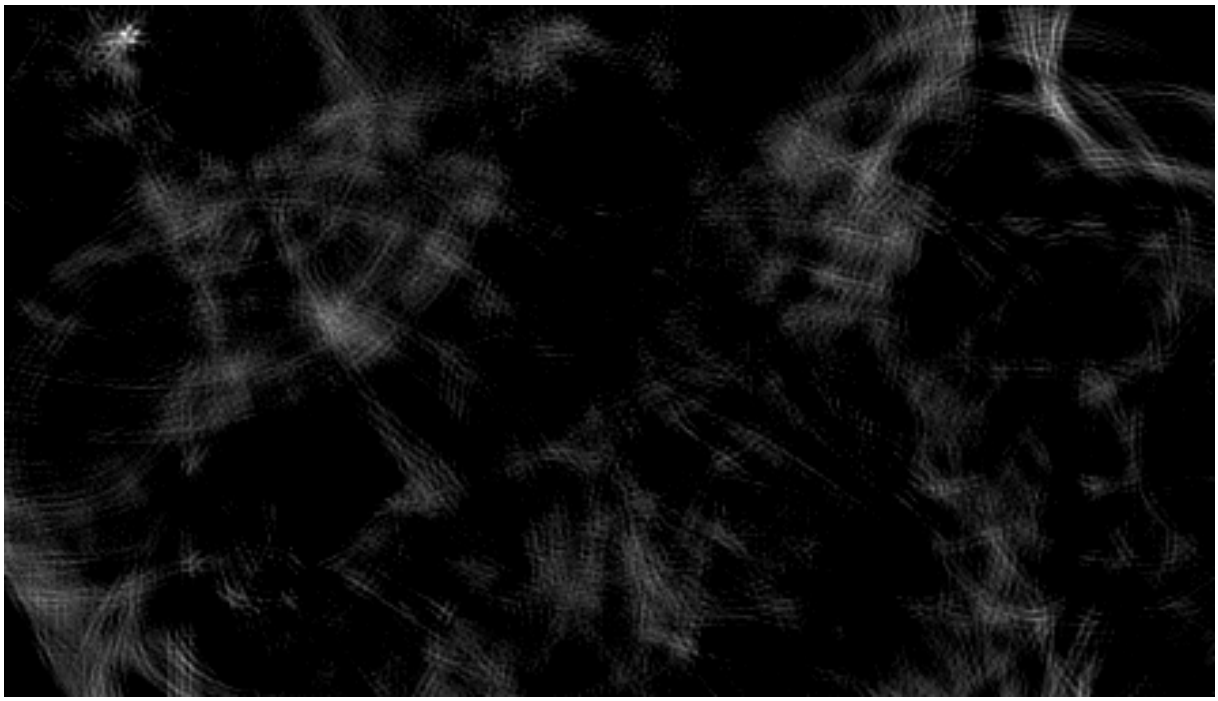

Figure 9: OpenCV Template Matching Heatmap

Figure 10: OpenCV Template Matching Results

The biggest difference between my cross correlation implementation and OpenCV's template matching implementation is that the peak of the heatmap is at the top left corner of the soccer ball rather than the center of the soccer ball as in my implementation.

A naive, and simple solution for template matching when the scale of the template is unknown, is to iterate the scale of the template, and simply try many scales, and eventually, you will likely find the object for which you are searching. See the implementation below. There are many matches at different scale values, but we see that eventually the soccer ball is found when the loop reaches the correct scale.

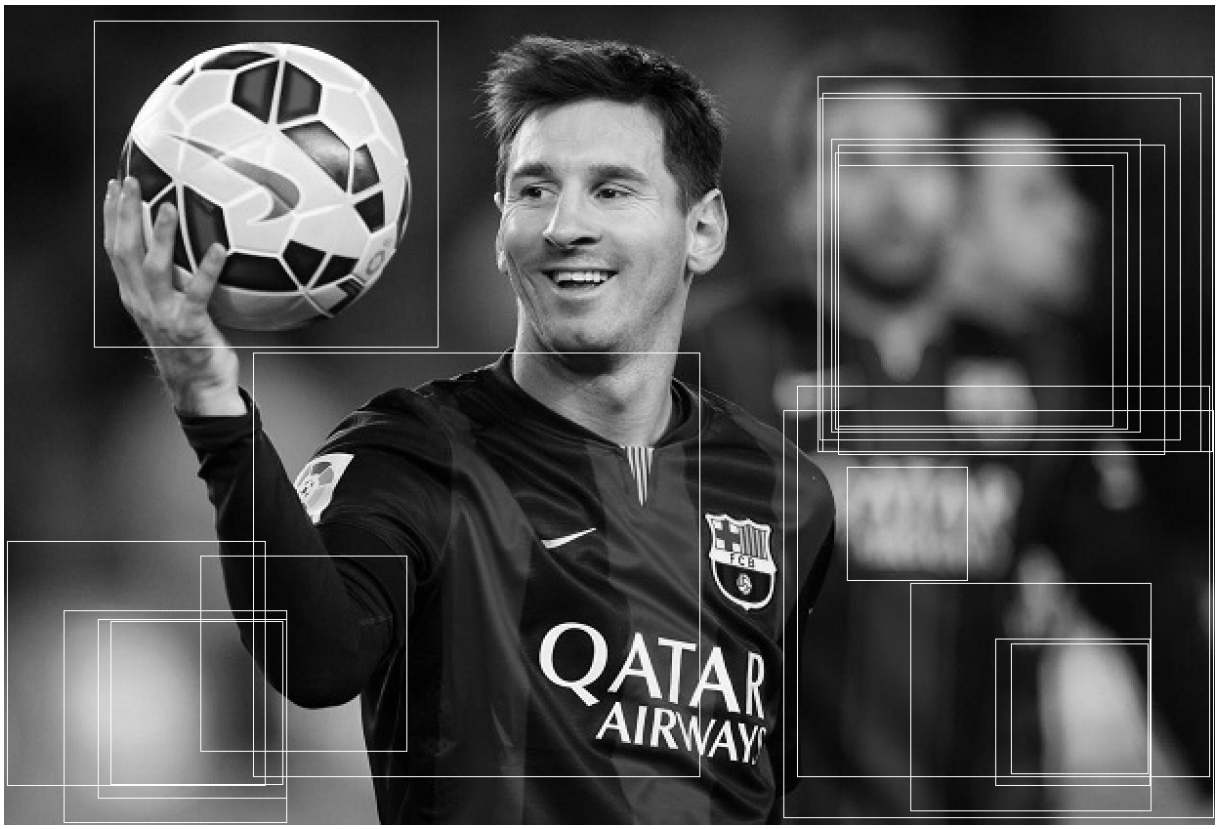

Figure 11: Unknown Scale Template Matching Results

```
from scipy import signal
import numpy as np
import cv2

def my_search(image, template):
    image = cv2.Canny(image, 20, 120)
    image = image - np.mean(image)

    template = cv2.Canny(template, 20, 120)
    template = template - np.mean(template)

    template = np.flipud(np.fliplr(template))
    corr = signal.convolve2d(image, template, boundary='symm', mode='full')

    corr *= (255.0/corr.max())

    cv2.imwrite('cross_correlate.png', corr)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(corr)

    w, h = template.shape[::-1]
    top_left = max_loc[0], max_loc[1]
    bottom_right = (top_left[0] - w, top_left[1] - h)
    cv2.rectangle(image, top_left, bottom_right, 255, 1)
    return image

messi = cv2.imread('messi.jpg', 0)
circle = cv2.imread('circle.bmp', 0)
messi = my_search(messi, circle)
cv2.imwrite('messi2.png', messi)
```

Figure 12: Cross Correlation Code

```
from scipy import signal
import numpy as np
import cv2

messi = cv2.imread('messi.jpg', 0)
messi = cv2.Canny(messi, 20, 120)
circle = cv2.imread('circle.bmp', 0)
circle = cv2.Canny(circle, 20, 120)

w, h = circle.shape[::-1]
img = messi.copy()
res = cv2.matchTemplate(img,circle,cv2.TM_CCOEFF_NORMED)

res *= (255.0/res.max())
cv2.imwrite('cv2_template_match.png', res)

min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
top_left = max_loc[0], max_loc[1]
bottom_right = (top_left[0] + w, top_left[1] + h)
cv2.rectangle(messi, top_left, bottom_right, 255, 1)
cv2.imwrite('messi3.png', messi)
```

Figure 13: Open CV Template Matching Code

```
from scipy import signal
import numpy as np
import cv2

def scale(img, scale_factor):
    width = int(img.shape[1] * scale_factor)
    height = int(img.shape[0] * scale_factor)
    dim = (width, height)
    return cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

messi = cv2.imread('messi.jpg', 0)
messi = scale(messi, 2);
messi_copy = messi.copy()
messi = cv2.Canny(messi, 20, 120)

circle = cv2.imread('circle.bmp', 0)
for i in range(20):
    # start template at 70% scale and increase scale by 10% each time
    scale_factor = 0.7+0.1*i
    template = scale(circle, scale_factor)
    template = cv2.Canny(template, 20, 120)

    w, h = template.shape[::-1]
    img = messi.copy()
    res = cv2.matchTemplate(img,template,cv2.TM_CCOEFF_NORMED)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
    top_left = max_loc[0], max_loc[1]
    bottom_right = (top_left[0] + w, top_left[1] + h)
    cv2.rectangle(messi_copy, top_left, bottom_right, 255, 1)

cv2.imwrite('messi_4.png'.format(scale_factor), messi_copy)
```

Figure 14: Unknown Scale Template Matching Code