

# hw2

October 28, 2020

## 1 Computer Vision

## 2 Jacobs University Bremen

## 3 Fall 2020

## 4 Homework 2

*This notebook includes both coding and written questions. Please hand in this notebook file with all the outputs and your answers to the written questions.*

This assignment covers linear filters, convolution and correlation.

```
In [1]: # Setup
        from __future__ import print_function
        import numpy as np
        import matplotlib.pyplot as plt
        from time import time
        from skimage import io

        %matplotlib inline
        plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
        plt.rcParams['image.interpolation'] = 'nearest'
        plt.rcParams['image.cmap'] = 'gray'

        # for auto-reloading external modules
        %load_ext autoreload
        %autoreload 2
```

### 4.1 Part 1: Convolutions

#### 4.1.1 1.1 Commutative Property (10 points)

Recall that the convolution of an image  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  and a kernel  $h : \mathbb{R}^2 \rightarrow \mathbb{R}$  is defined as follows:

$$(f * h)[m, n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j] \cdot h[m - i, n - j]$$

Or equivalently,

$$(f * h)[m, n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h[i, j] \cdot f[m - i, n - j] \quad (1)$$

$$= (h * f)[m, n] \quad (2)$$

Show that this is true (i.e. prove that the convolution operator is commutative:  $f * h = h * f$ ).

**Your Answer:**

Let  $x = m - i$

Let  $y = n - j$

$i = m - x$

$j = n - y$

We'll now substitute these values in the given formulas accordingly.

$$\begin{aligned} (f * h)[m, n] &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j] \cdot h[m - i, n - j] \\ &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[m - x, n - y] \cdot h[x, y] \\ &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h[i, j] \cdot f[m - i, n - j] \\ &= (h * f)[m, n] \end{aligned}$$

Reference: [http://www.songho.ca/dsp/convolution/convolution\\_commutative.html](http://www.songho.ca/dsp/convolution/convolution_commutative.html)

#### 4.1.2 1.2 Implementation (30 points)

In this section, you will implement two versions of convolution: - conv\_nested - conv\_fast

First, run the code cell below to load the image to work with.

```
In [2]: # Open image as grayscale
img = io.imread('dog.jpg', as_gray=True)

# Show image
plt.imshow(img)
plt.axis('off')
plt.title("Isn't he cute?")
plt.show()
```

Isn't he cute?



Now, implement the function `conv_nested` in `filters.py`. This is a naive implementation of convolution which uses 4 nested for-loops. It takes an image  $f$  and a kernel  $h$  as inputs and outputs the convolved image ( $f * h$ ) that has the same shape as the input image. This implementation should take a few seconds to run.

- Hint: It may be easier to implement  $(h * f)$

We'll first test your `conv_nested` function on a simple input.

```
In [3]: from filters import conv_nested
```

```
# Simple convolution kernel.
kernel = np.array(
    [
        [1,0,1],
        [0,0,0],
        [1,0,0]
    ])
```

```

# Create a test image: a white square in the middle
test_img = np.zeros((9, 9))
test_img[3:6, 3:6] = 1

# Run your conv_nested function on the test image
test_output = conv_nested(test_img, kernel)

# Build the expected output
expected_output = np.zeros((9, 9))
expected_output[2:7, 2:7] = 1
expected_output[5:, 5:] = 0
expected_output[4, 2:5] = 2
expected_output[2:5, 4] = 2
expected_output[4, 4] = 3

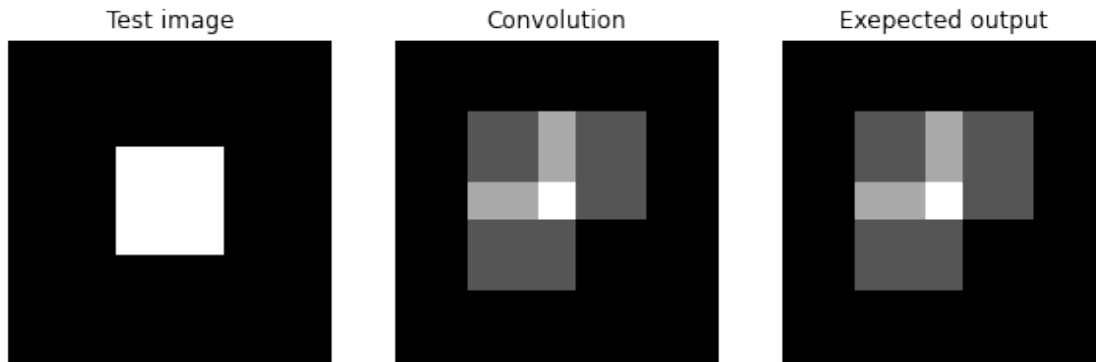
# Plot the test image
plt.subplot(1,3,1)
plt.imshow(test_img)
plt.title('Test image')
plt.axis('off')

# Plot your convolved image
plt.subplot(1,3,2)
plt.imshow(test_output)
plt.title('Convolution')
plt.axis('off')

# Plot the expected output
plt.subplot(1,3,3)
plt.imshow(expected_output)
plt.title('Expected output')
plt.axis('off')
plt.show()

# Test if the output matches expected output
assert np.max(test_output - expected_output) < 1e-10, "Your solution is not correct."

```



Now let's test your `conv_nested` function on a real image.

```
In [4]: from filters import conv_nested

        # Simple convolution kernel.
        # Feel free to change the kernel to see different outputs.
        kernel = np.array(
            [
                [1,0,-1],
                [2,0,-2],
                [1,0,-1]
            ])

        out = conv_nested(img, kernel)

        # Plot original image
        plt.subplot(2,2,1)
        plt.imshow(img)
        plt.title('Original')
        plt.axis('off')

        # Plot your convolved image
        plt.subplot(2,2,3)
        plt.imshow(out)
        plt.title('Convolution')
        plt.axis('off')

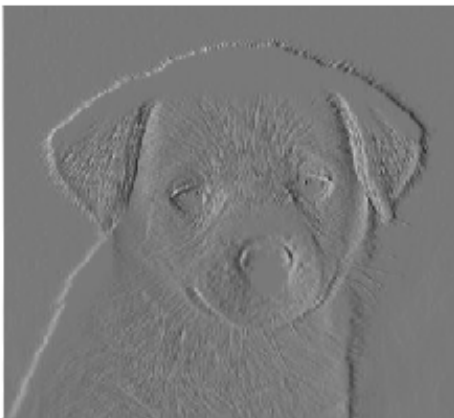
        # Plot what you should get
        solution_img = io.imread('convoluted_dog.jpg', as_gray=True)
        plt.subplot(2,2,4)
        plt.imshow(solution_img)
        plt.title('What you should get')
        plt.axis('off')
```

```
plt.show()
```

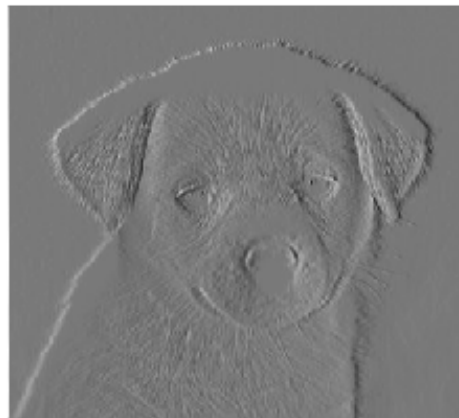
Original



Convolution



What you should get



Let us implement a more efficient version of convolution using array operations in numpy. As shown in the lecture, a convolution can be considered as a sliding window that computes sum of the pixel values weighted by the flipped kernel. The faster version will i) zero-pad an image, ii) flip the kernel horizontally and vertically, and iii) compute weighted sum of the neighborhood at each pixel.

First, implement the function `zero_pad` in `filters.py`.

```
In [5]: from filters import zero_pad
```

```
pad_width = 20 # width of the padding on the left and right  
pad_height = 40 # height of the padding on the top and bottom
```

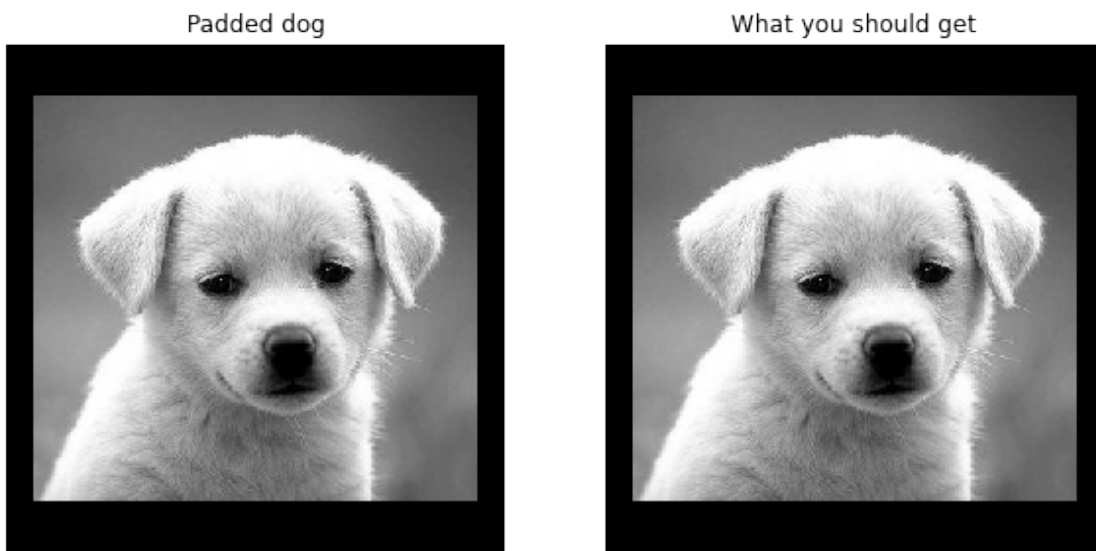
```
padded_img = zero_pad(img, pad_height, pad_width)
```

```
# Plot your padded dog
```

```
plt.subplot(1,2,1)
plt.imshow(padded_img)
plt.title('Padded dog')
plt.axis('off')

# Plot what you should get
solution_img = io.imread('padded_dog.jpg', as_gray=True)
plt.subplot(1,2,2)
plt.imshow(solution_img)
plt.title('What you should get')
plt.axis('off')

plt.show()
```



Next, complete the function `conv_fast` in `filters.py` using `zero_pad`. Run the code below to compare the outputs by the two implementations. `conv_fast` should run significantly faster than `conv_nested`.

Depending on your implementation and computer, `conv_nested` should take a few seconds and `conv_fast` should be around 5 times faster.

```
In [6]: from filters import conv_fast
```

```
t0 = time()
out_fast = conv_fast(img, kernel)
t1 = time()
out_nested = conv_nested(img, kernel)
t2 = time()
```

```
# Compare the running time of the two implementations
```

```

print("conv_nested: took %f seconds." % (t2 - t1))
print("conv_fast: took %f seconds." % (t1 - t0))

# Plot conv_nested output
plt.subplot(1,2,1)
plt.imshow(out_nested)
plt.title('conv_nested')
plt.axis('off')

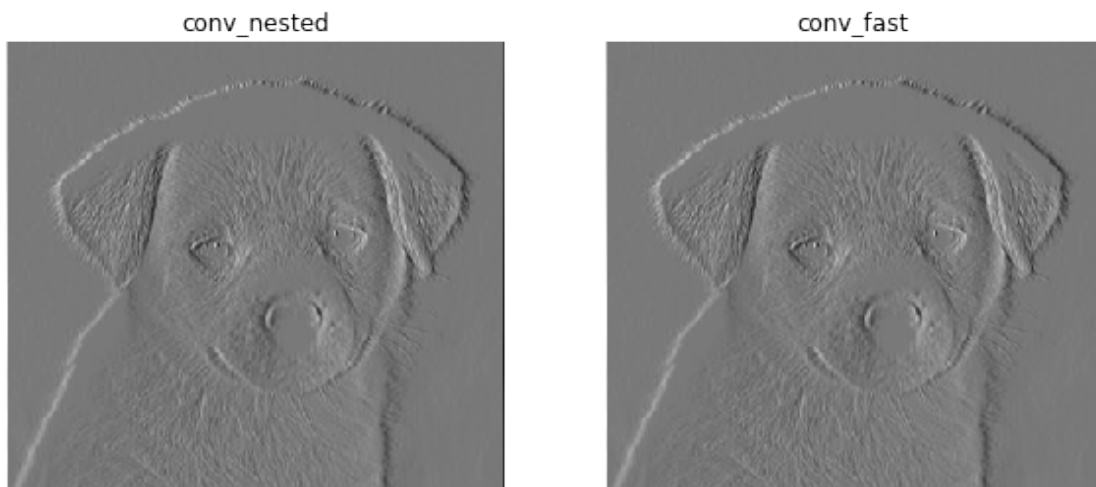
# Plot conv_fast output
plt.subplot(1,2,2)
plt.imshow(out_fast)
plt.title('conv_fast')
plt.axis('off')

# Make sure that the two outputs are the same
if not (np.max(out_fast - out_nested) < 1e-10):
    print("Different outputs! Check your implementation.")

```

conv\_nested: took 4.767129 seconds.

conv\_fast: took 1.112049 seconds.




---

## 4.2 Part 2: Cross-correlation

Cross-correlation of two 2D signals  $f$  and  $g$  is defined as follows:

$$(f \star g)[m, n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j] \cdot g[i - m, j - n]$$



## 4.2.1 2.1 Template Matching with Cross-correlation (12 points)

Suppose that you are a clerk at a grocery store. One of your responsibilities is to check the shelves periodically and stock them up whenever there are sold-out items. You got tired of this laborious task and decided to build a computer vision system that keeps track of the items on the shelf.

Luckily, you have learned in the Computer Vision class at Jacobs University that cross-correlation can be used for template matching: a template  $g$  is multiplied with regions of a larger image  $f$  to measure how similar each region is to the template.

The template of a product (`template.jpg`) and the image of shelf (`shelf.jpg`) is provided. We will use cross-correlation to find the product in the shelf.

Implement `cross_correlation` function in `filters.py` and run the code below.

- Hint: you may use the `conv_fast` function you implemented in the previous question.

```
In [7]: from filters import cross_correlation

# Load template and image in grayscale
img = io.imread('shelf.jpg')
img_gray = io.imread('shelf.jpg', as_gray=True)
temp = io.imread('template.jpg')
temp_gray = io.imread('template.jpg', as_gray=True)

# Perform cross-correlation between the image and the template
out = cross_correlation(img_gray, temp_gray)

# Find the location with maximum similarity
y,x = (np.unravel_index(out.argmax(), out.shape))

# Display product template
plt.figure(figsize=(25,20))
plt.subplot(3, 1, 1)
plt.imshow(temp)
plt.title('Template')
plt.axis('off')

# Display cross-correlation output
plt.subplot(3, 1, 2)
plt.imshow(out)
plt.title('Cross-correlation (white means more correlated)')
plt.axis('off')

# Display image
plt.subplot(3, 1, 3)
plt.imshow(img)
plt.title('Result (blue marker on the detected location)')
plt.axis('off')

# Draw marker at detected location
plt.plot(x, y, 'bx', ms=40, mew=10)
plt.show()
```

Template



Cross-correlation (white means more correlated)



Result (blue marker on the detected location)



**Interpretation** How does the output of cross-correlation filter look? Was it able to detect the product correctly? Explain what problems there might be with using a raw template as a filter.

**Your Answer:** No, unfortunately the detections weren't correct. White is supposed to represent a higher correlation but in the output, we see that the spot where the object is located is shown as black even though it should have been white.

This could be because the sizes of the raw template and the given image differ greatly. The template's shape is way smaller.

---

#### 4.2.2 2.2 Zero-mean cross-correlation (6 points)

A solution to this problem is to subtract the mean value of the template so that it has zero mean.

Implement `zero_mean_cross_correlation` function in `filters.py` and run the code below.

```
In [8]: from filters import zero_mean_cross_correlation

# Perform cross-correlation between the image and the template
out = zero_mean_cross_correlation(img_grey, temp_grey)

# Find the location with maximum similarity
y,x = (np.unravel_index(out.argmax(), out.shape))

# Display product template
plt.figure(figsize=(30,20))
plt.subplot(3, 1, 1)
plt.imshow(temp)
plt.title('Template')
plt.axis('off')

# Display cross-correlation output
plt.subplot(3, 1, 2)
plt.imshow(out)
plt.title('Cross-correlation (white means more correlated)')
plt.axis('off')

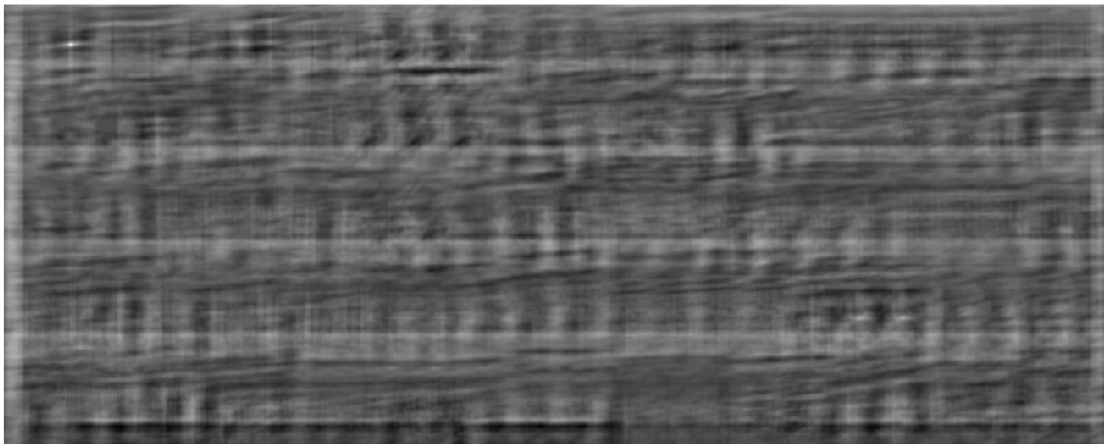
# Display image
plt.subplot(3, 1, 3)
plt.imshow(img)
plt.title('Result (blue marker on the detected location)')
plt.axis('off')

# Draw marker at detected location
plt.plot(x, y, 'bx', ms=40, mew=10)
plt.show()
```

Template



Cross-correlation (white means more correlated)



Result (blue marker on the detected location)



You can also determine whether the product is present with appropriate scaling and thresholding.



```

In [9]: def check_product_on_shelf(shelf, product):
        out = zero_mean_cross_correlation(shelf, product)

        # Scale output by the size of the template
        out = out / float(product.shape[0]*product.shape[1])

        # Threshold output (this is arbitrary, you would need to tune the threshold for a
        out = out > 0.025

        if np.sum(out) > 0:
            print('The product is on the shelf')
        else:
            print('The product is not on the shelf')

        # Load image of the shelf without the product
        img2 = io.imread('shelf_soldout.jpg')
        img2_grey = io.imread('shelf_soldout.jpg', as_gray=True)

        plt.imshow(img)
        plt.axis('off')
        plt.show()
        check_product_on_shelf(img_grey, temp_grey)

        plt.imshow(img2)
        plt.axis('off')
        plt.show()
        check_product_on_shelf(img2_grey, temp_grey)

```



The product is on the shelf



The product is not on the shelf

---

#### 4.2.3 2.3 Normalized Cross-correlation (12 points)

One day the light near the shelf goes out and the product tracker starts to malfunction. The `zero_mean_cross_correlation` is not robust to change in lighting condition. The code below demonstrates this.

```
In [10]: from filters import normalized_cross_correlation

# Load image
img = io.imread('shelf_dark.jpg')
img_grey = io.imread('shelf_dark.jpg', as_gray=True)

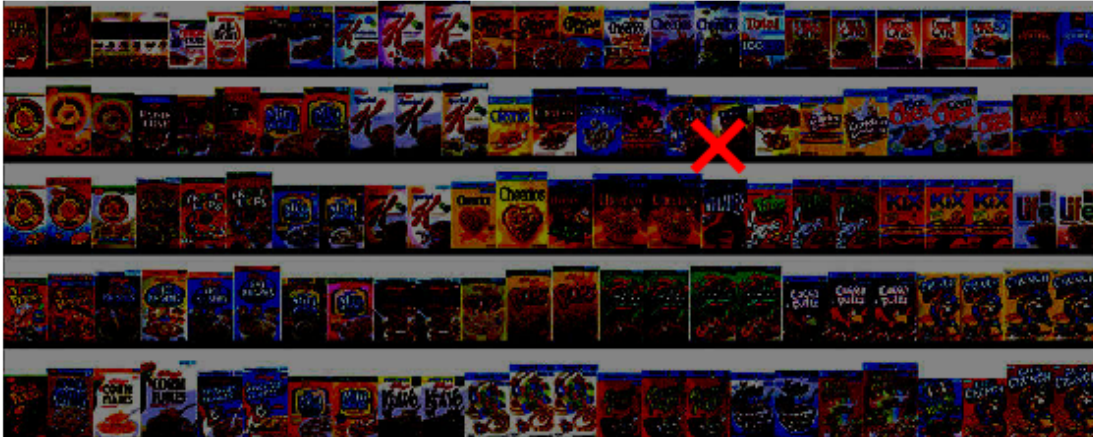
# Perform cross-correlation between the image and the template
out = zero_mean_cross_correlation(img_grey, temp_grey)

# Find the location with maximum similarity
y,x = (np.unravel_index(out.argmax(), out.shape))

# Display image
plt.imshow(img)
plt.title('Result (red marker on the detected location)')
plt.axis('off')

# Draw marker at detected location
plt.plot(x, y, 'rx', ms=25, mew=5)
plt.show()
```

Result (red marker on the detected location)



A solution is to normalize the pixels of the image and template at every step before comparing them. This is called **normalized cross-correlation**.

The mathematical definition for normalized cross-correlation of  $f$  and template  $g$  is:

$$(f \star g)[m, n] = \sum_{i,j} \frac{f[i, j] - \overline{f_{m,n}}}{\sigma_{f_{m,n}}} \cdot \frac{g[i - m, j - n] - \overline{g}}{\sigma_g}$$

where: -  $f_{m,n}$  is the patch image at position  $(m, n)$  -  $\overline{f_{m,n}}$  is the mean of the patch image  $f_{m,n}$  -  $\sigma_{f_{m,n}}$  is the standard deviation of the patch image  $f_{m,n}$  -  $\overline{g}$  is the mean of the template  $g$  -  $\sigma_g$  is the standard deviation of the template  $g$

Implement `normalized_cross_correlation` function in `filters.py` and run the code below.

```
In [11]: from filters import normalized_cross_correlation

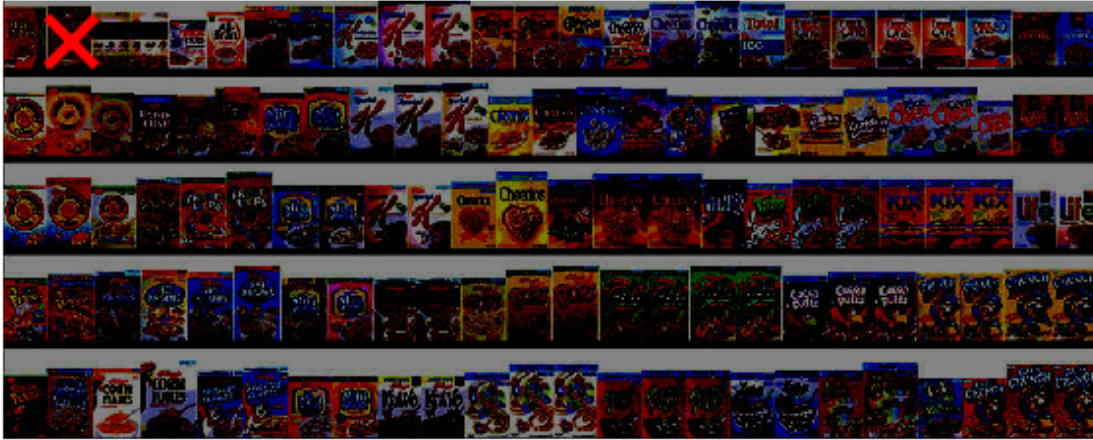
# Perform normalized cross-correlation between the image and the template
out = normalized_cross_correlation(img_grey, temp_grey)

# Find the location with maximum similarity
y,x = (np.unravel_index(out.argmax(), out.shape))

# Display image
plt.imshow(img)
plt.title('Result (red marker on the detected location)')
plt.axis('off')

# Draw marker at detected location
plt.plot(x, y, 'rx', ms=25, mew=5)
plt.show()
```

Result (red marker on the detected location)



### 4.3 Part 3: Separable Filters

#### 4.3.1 3.1 Theory (10 points)

Consider an  $M_1 \times N_1$  image  $I$  and an  $M_2 \times N_2$  filter  $F$ . A filter  $F$  is **separable** if it can be written as a product of two 1D filters:  $F = F_1 F_2$ .

For example,

$$F = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

can be written as a matrix product of

$$F_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, F_2 = \begin{bmatrix} 1 & -1 \end{bmatrix}$$

Therefore  $F$  is a separable filter.

Prove that for any separable filter  $F = F_1 F_2$ ,

$$I * F = (I * F_1) * F_2$$

**Your Answer:**

$$\begin{aligned} (I * F)[m, n] &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I[i, j] \cdot F[m - i, n - j] \\ &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I[i, j] \cdot F_1[m - i] \cdot F_2[n - j] \\ &= \sum_{j=-\infty}^{\infty} \cdot \left( \sum_{i=-\infty}^{\infty} I[i, j] \cdot F_1[m - i] \right) \cdot F_2[n - j] \\ &= (I * F_1) * F_2[m, n] \end{aligned}$$



Alternatively, we can also perform substitutions for better understanding by using  $x, y$  variables:

$$\text{Let } x = m - i$$

$$\text{Let } y = n - j$$

$$i = m - x$$

$$j = n - y$$

as stated in the CS Toronto Reference: <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0BPNcd6GH1-Oa3MU1rA>