Wensong Hu
*EECS*
*University of Michigan*

**February 13, 2024**

**HW2 — Convolution and Feature Detection**

# 1 Patches

## 1.1 Task 1: Image Patches

### 1.1.1 image_patches() and result

Code submitted to Canvas.
Result see figure1.

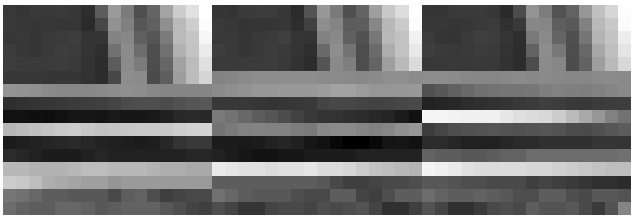

Figure 1: Image patches

### 1.1.2 Why normalized?

Normalization makes the measurement of similarity between image patches more robust to variations in lighting or illumination, focusing the comparison on the intrinsic patterns and textures. It ensures that the dot product similarity measure reflects the actual content similarity, rather than being affected by extrinsic factors like lighting conditions.

### 1.1.3 Discuss patches in early CV

Normalized patches, with zero mean and unit variance, are excellent for enhancing the robustness of matching or recognizing objects under varying conditions such as illumination changes, as they focus on the structure and texture rather than the absolute brightness. However, these patches might not be as effective when dealing with changes in an object's pose, scale, or significant variations in viewpoint, because normalization does not inherently account for geometric transformations, which could lead to significant differences in the appearance of patches extracted from these varied conditions.

# 2 Image Filtering

## 2.1 Task 2: Convolution and Gaussian Filter

### 2.1.1 Prove equivalency

We knows that, for two Gaussian filters $G_y \in \mathbb{R}^{k\times 1}$ and $G_x \in \mathbb{R}^{k\times 1}$:

$$G_x * G_y = G_x G_y = G \tag{1-1}$$

So we have:

$$\begin{aligned} G * X &= (G_x G_y) * X && (1\text{-}2)\\ &= (G_x * G_y) * x && (1\text{-}3)\\ &= G_x * (G_y * X) && (1\text{-}4) \end{aligned}$$

This proves that convolution by a 3D Gaussian filter is equivalent to sequentially applying a vertical and horizontal Gaussian filter.

### 2.1.2 convolve()

Code submitted to Canvas.

### 2.1.3 Result and Discuss

Result see figure 2.

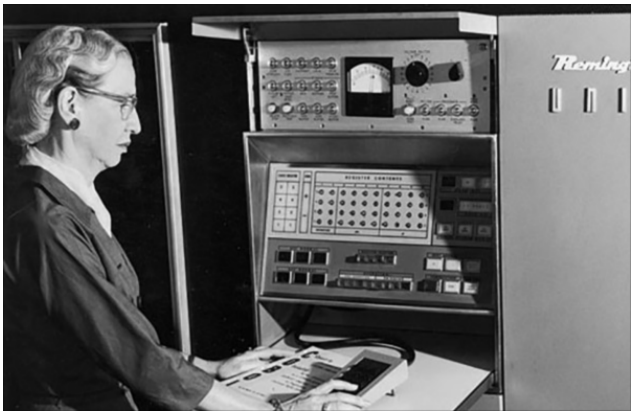

Figure 2: Result of Gaussian Filtering

Gaussian filtering smooths an image by blurring and reducing its high-frequency components, mitigating noise and details.

### 2.1.4 Discussion of normalization of kernel

Ensure the filter sum up to 1 to ensure that the overall brightness of the image is preserved after filtering. If the filter sums to more than 1, it could artificially increase the intensity of

the image (making it brighter), while if it sums to less than 1, it could decrease the intensity (making it darker). This preservation of brightness is important for maintaining the natural appearance of the image.

### 2.1.5 Derive convolution kernels for derivatives

Consider the image as a function $I : \mathbb{R}^2 \to \mathbb{R}$. We define the discrete derivatives in the horizontal (x) and vertical (y) directions as follows:

$$I_x(x, y) = [I(x + 1, y) - I(x - 1, y)] \approx 2\frac{\partial I}{\partial x}$$
$$I_y(x, y) = [I(x, y + 1) - I(x, y - 1)] \approx 2\frac{\partial I}{\partial y}$$

To represent these derivatives as convolutions, we need to define the kernels $k_x$ and $k_y$ that capture the discrete difference operation. The convolution operation for the horizontal derivative can be written as:

$$I_x = I * k_x$$

where $k_x$ is a row vector that subtracts adjacent pixel values along the x-direction. Since convolution is a weighted sum, we assign weights that reflect the discrete difference operation. Thus, we define $k_x$ as:

$$k_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Similarly, for the vertical derivative, the convolution is:

$$I_y = I * k_y$$

where $k_y$ is a column vector that subtracts adjacent pixel values along the y-direction. Accordingly, $k_y$ is defined as:

$$k_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Sometimes a factor of $\frac{1}{2}$ might be included in both kernels to account for the fact that we are approximating the derivative by the difference between the pixels that are two units apart (hence, the derivative is twice the value of the difference). However, the constant factor can be adjusted depending on the specific implementation and scale used in the image processing.

### 2.1.6 edge_detection()

Submitted to Canvas.

### 2.1.7 Result and Discussion

Result of edge detection for original image shown figure3 Result of edge detection for Gaussian filtered image shown figure4

Figure 3: Edge detection for original image

Figure 4: Edge detection for Gaussian filtering image

Although the images having tiny difference, we can still see that the edge detection on Gaussian filtered image has less edges than original one, which complies more with our goal. For example, the wrinkles on the sleeve are less detected so that the edge of human and object is less noisy. However, this may also remove some edges we desire to preserve.

#### 2.1.8 bilateral_filter() and result

Code submitted to Canvas.
Result see figure5.

Figure 5: Bilateral filtered image

## 2.2 Task 3: Sobel Operator

### 2.2.1 Show relateion between Sobel and Gaussian kernel

The Sobel filter for the x-direction is given by:

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

And the Gaussian kernel $G_s$ is given by:

$$G_s = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

We want to prove that:

$$(I * G_s) * k_x = I * (G_s * k_x) = I * S_x \implies (G_s * k_x) = S_x$$

where $k_x$ is the horizontal filter that we need to derive.
We assume $k_x$ to be of the form:

$$k_x = \begin{bmatrix} a & b & c \end{bmatrix}$$

Upon solving, first assume $k_x = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$ based on the structure of the Sobel filter. This gives us:

$$beginalign]G_s * k_x = \begin{bmatrix} 2 & 0 & -2 \\ 4 & 0 & -4 \\ 2 & 0 & -2 \end{bmatrix}$$

where, the padding is zero padding.
After doing some normalization, the result matches the Sobel $S_x$ filter exactly. Therefore, we conclude that applying the Sobel filter $S_x$ to image $I$ after Gaussian filtering with $G_s$ is a analog to taking the horizontal derivative of the Gaussian-filtered image.

### 2.2.2 sobel_operator()

Submitted to Canvas.
Results see figure 6, 7, and 8.

### 2.2.3 Result

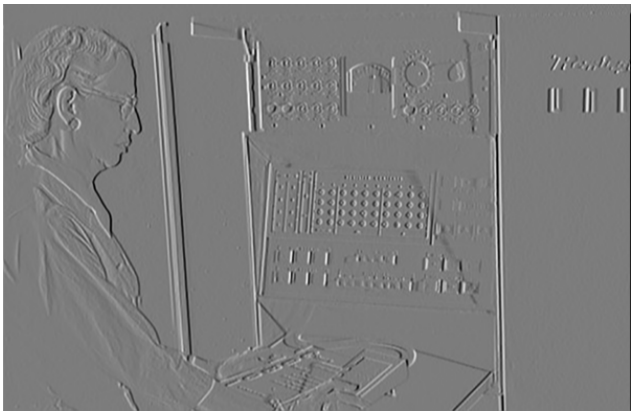

Figure 6: Sobel filter: I * Sx

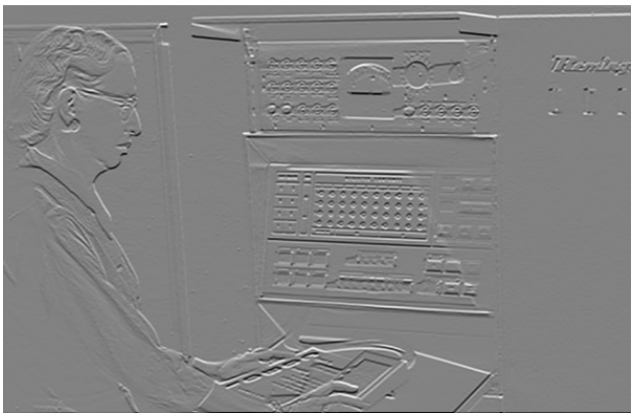

Figure 7: Sobel filter: I * Sy

Figure 8: Sobel filter: gradient magnitude

## 2.3 Task 4: LoG Filter

Result see figure 9, 10.

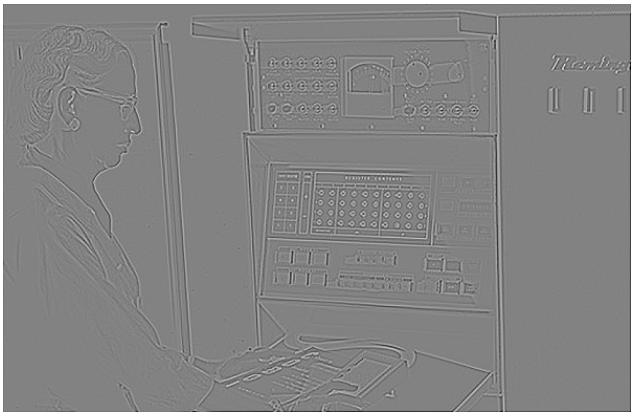

Figure 9: LoG filter 1

Figure 10: LoG filter 2

The figure given by the LoG filter 2 is more clear than LoG filter 1. The reason for this is that the blob size of the original figure have higher alignment with the LoG filter 2 which has a larger covariance value, in other word, the original figure has higher response to LoG filter 2.
Yes, these filters can detect edges, but the filter covariance need to be manually tuned to find the best response. They can also used for blob detection to detect patterns in figure.

### 2.3.1 Approximate LoG

The Laplace of Gaussian (LoG) of image $f$ can be written as

$$\nabla^2(f * g) = f * \nabla^2 g$$

That is, the Laplace of the image smoothed by a Gaussian kernel is identical to the image convolved with the Laplace of the Gaussian kernel. This convolution can be further expanded, in the 2D case, as

$$f * \nabla^2 g = f * \left( \frac{\partial^2}{\partial x^2} g + \frac{\partial^2}{\partial y^2} g \right) = f * \frac{\partial^2}{\partial x^2} g + f * \frac{\partial^2}{\partial y^2} g$$

This gives the similar form of Difference of Gaussian.
See figure 11.

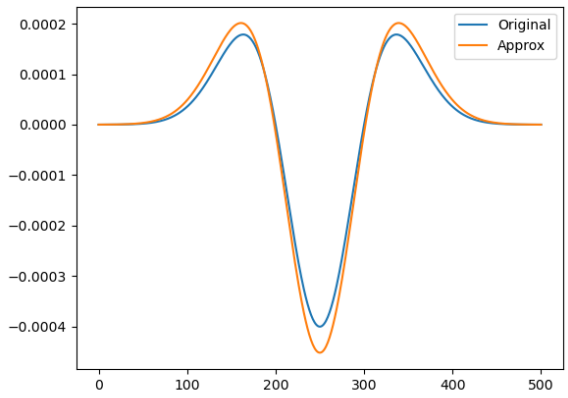


Figure 11: LoG and DoG

## 2.4 Task 5: Who's That Filter?

### 2.4.1 What is the filters?

The result is:

```
    filter1 = np.ones((3, 3))
    filter2 = np.ones((3, 3)) * 1/9
    filter3 = np.array([[0, 0, 0],
                        [1, 0, 0],
                        [0, 0, 0]])
```

```
    filter4 = np.array([[-1, 0, 1],
                        [-1, 0, 1],
                        [-1, 0, 1]])
```

#### 2.4.2 Function of filter 1

Filter 1 does a blurring to the image. It is different from filter 2 since it is not normalized, but kernel filter 2 is normalized.

# 3 Feature Extraction

## 3.1 Task 6: Coner Score

### 3.1.1 corner_score()

Submitted to Canvas.

### 3.1.2 Result

Result see figure 12 to 15

Figure 12: Corner score with (u, v) = (0, 5)

Figure 13: Corner score with (u, v) = (0, -5)

Figure 14: Corner score with (u, v) = (5, 0)

Figure 15: Corner score with (u, v) = (-5, 0)

### 3.1.3 Discuss

If a figure has size $H*W$, and the kernel used is $h*w$, the time complexity will be $O(HWhw)$, which is a large high order term for the computer back to the 80s.

## 3.2 Task 7: Harris Corner Detector

### 3.2.1 harris_detector()

Submitted to Canvas.

### 3.2.2 Result

Result see figure 16

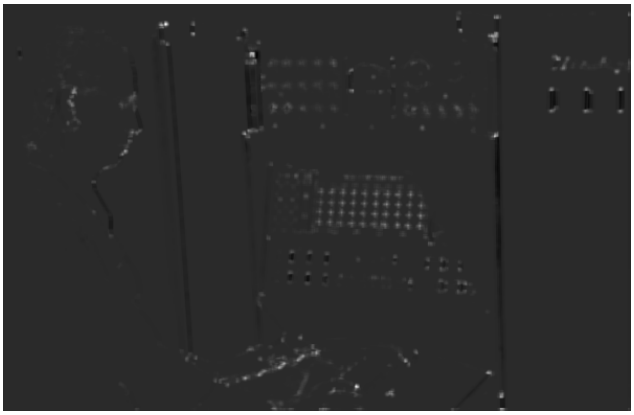

Figure 16: Harris response

# 4 Blob Detection

## 4.1 Task 8: Single-Scale Blob Detection

### 4.1.1 gaussian_filter()

Submitted to Canvas.

### 4.1.2 Result

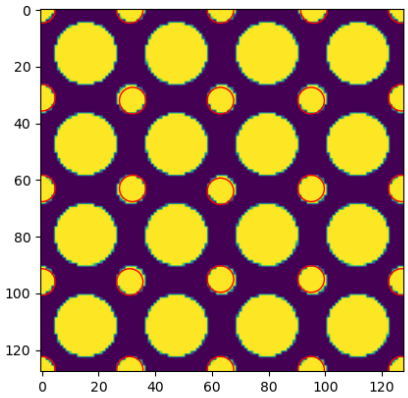


Figure 17: Single-scale blob detection - small

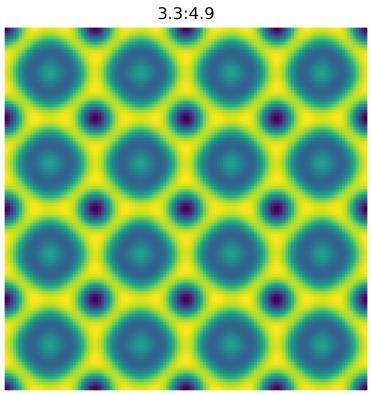


Figure 18: Single-scale blob detection DoG - small

The parameter used for small blob:

```
    k = 5
    sigma_1 = 3.7
    sigma_2 = k * sigma_1
```

There are 25 maxima observed for small blob. No false peak.

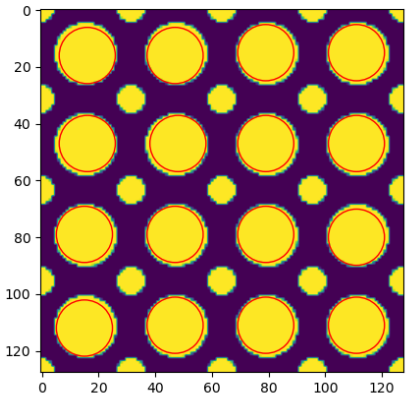


Figure 19: Single-scale blob detection - large

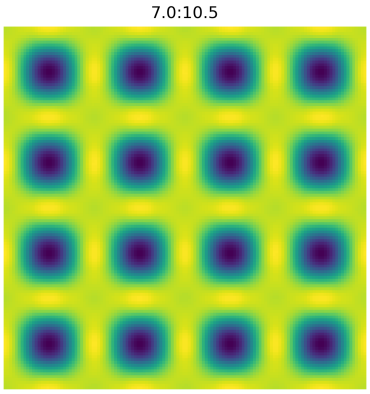


Figure 20: Single-scale blob detection DoG - large

The parameter used for large blob:

```
    k = 1.9
    sigma_1 = 3.7
    sigma_2 = k * sigma_1
```

There are 17 maxima observed for large blob. No false peak.

## 4.2 Task 9: Cell Counting

### 4.2.1 Parameters and Results

Parameters for them are:

```
    #"Detecting cell1 -- 008cell"
    k = 3
    sigma_1 = 3.7
    sigma_2 = k * sigma_1
```

```
    #"Detecting cell2 -- 004cell"
    k = 3
    sigma_1 = 3.4
    sigma_2 = k * sigma_1

    #"Detecting cell3 -- 005cell"
    k = 5
    sigma_1 = 3.7
    sigma_2 = k * sigma_1

    #"Detecting cell4 -- 006cell"
    k = 1.9
    sigma_1 = 3.7
    sigma_2 = k * sigma_1
```

The numbers of blobs are 102, 95, 105 and 158 respectively.

### 4.2.2 Plots and Discussion

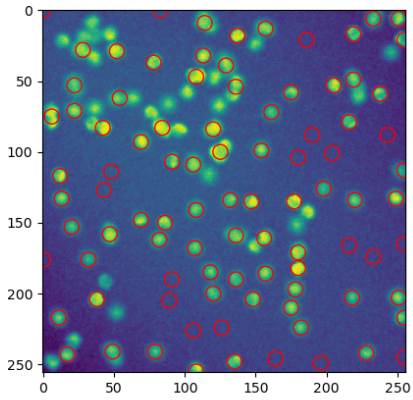


Figure 21: Blob detection - cell1

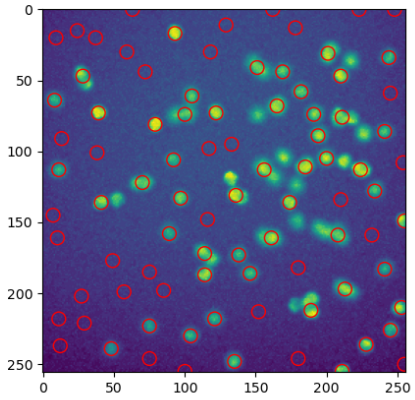


Figure 22: Blob detection - cell2

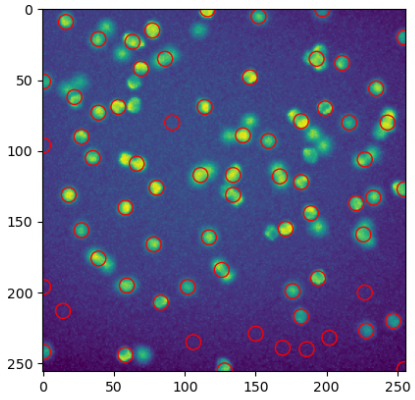


Figure 23: Blob detection - cell3

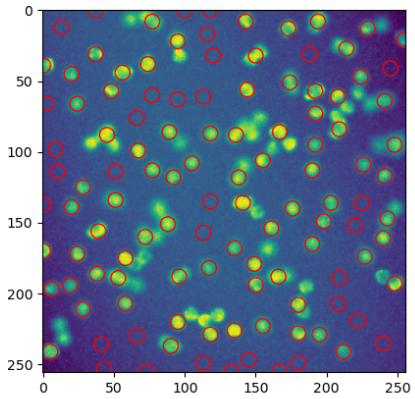


Figure 24: Blob detection - cell4

Increasing sigma will allow more cells be detected, but also increase the probability of false detection. Increasing k will make the false detection less, but correct detection will also be eliminate.
Iterating or using gradient to find the best sigma and k might be a way to make this cell detection more universal applicable.

*Submitted by Wensong Hu on February 13, 2024.*