Practical Work 5
Performing Convolution and Correlation Operations on Two-Dimensional Signals
1. Objective

The main objectives of this practical are:

- To understand the concepts of **2D convolution** and **2D correlation**.
- To perform these operations using **MATLAB functions**.
- To analyze how convolution and correlation affect two-dimensional signals (images).
- To compare the results of convolution and correlation visually and numerically.

2. Theoretical Background

Introduction to Two-Dimensional Signals

In the field of signal and image processing, signals can exist in one, two, or more dimensions depending on the nature of the data. A **two-dimensional (2D) signal** is a function that depends on two independent variables, often denoted as f(x,y). The most common example of a 2D signal is a **digital image**, where x and y represent the spatial coordinates of a pixel, and the function value f(x,y) represents the intensity or color value at that position.

A digital image is formed by sampling and quantizing a continuous image. The intensity of each pixel can be represented as an integer value in a given range, for example, 0–255 for an 8-bit grayscale image. In this way, a 2D signal provides a discrete representation of a real-world scene.

Two-dimensional signal processing extends the principles of one-dimensional signal processing (used for time-varying signals such as audio) into the spatial domain. Operations such as **filtering, convolution, correlation, Fourier transform**, and **edge detection** are fundamental tools that help in analyzing, enhancing, and interpreting images.

Understanding convolution and correlation in two dimensions is crucial because almost every image processing technique—from noise removal to pattern recognition—relies on these fundamental operations.

Concept of Convolution

Convolution is one of the most important mathematical operations in signal and image processing. It describes the way one signal modifies or interacts with another. In two dimensions, the convolution between two signals (or images) f(x,y) and h(x,y) is defined as:

$$g(x,y) = f(x,y) * h(x,y) = \sum_m \sum_n f(m,n) \cdot h(x-m, y-n)$$

Here:

- f(x,y) is usually the input image or signal,
- h(x,y) is the kernel or filter,
- g(x,y) is the output (the filtered image),
- and the asterisk (*) denotes the convolution operation.

Conceptually, convolution represents the process of **sliding the kernel across the image**, multiplying overlapping values, summing them up, and storing the result in the corresponding output pixel.

Continuous vs. Discrete Convolution

In continuous systems, convolution is defined by an integral:

$$g(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(a,b) \cdot h(x-a, y-b)\, da\, db$$

In digital image processing, however, images are discrete, so the convolution becomes a **double summation** as shown earlier. The discrete convolution is much more practical because we deal with finite pixel values instead of continuous functions.

Kernel (Filter) in Convolution

The function h(x,y) used in convolution is commonly called a **kernel**, **filter**, or **mask**. A kernel is typically a small matrix such as 3×3 times, 5×5 times, or 7×7 times containing weights that define the operation to be performed.

For example:

- A **smoothing filter** has equal weights that average neighboring pixels.
- A **sharpening filter** emphasizes edges by subtracting blurred information.
- An **edge detection filter** detects boundaries and transitions in intensity.

When the kernel is applied across an image, it locally modifies pixel values depending on the neighboring values and the kernel's weights.

Interpretation of Convolution

Convolution can be understood as **a weighted sum of neighboring pixels**. Each pixel in the output image is obtained by taking a small neighborhood of pixels from the input image, multiplying them by the corresponding weights from the filter, and summing the results.

Mathematically, for a kernel of size 3×3 times

$$g(x,y) = \begin{bmatrix} f(x-1,y-1) & f(x,y-1) & f(x+1,y-1) \\ f(x-1,y) & f(x,y) & f(x+1,y) \\ f(x-1,y+1) & f(x,y+1) & f(x+1,y+1) \end{bmatrix} \cdot \begin{bmatrix} h(-1,-1) & h(0,-1) & h(1,-1) \\ h(-1,0) & h(0,0) & h(1,0) \\ h(-1,1) & h(0,1) & h(1,1) \end{bmatrix}$$

After flipping the kernel both horizontally and vertically, this multiplication and summation yield the new pixel value at (x,y).

Types of Convolution Filters

Averaging (Smoothing) Filter

This filter replaces each pixel value with the average of its neighbors. It reduces noise but also blurs edges.

Example kernel:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Gaussian Filter

A Gaussian kernel applies a weighted average where central pixels contribute more. It provides smoother blurring and is widely used in preprocessing.

Sharpening Filter

Enhances details by highlighting intensity transitions.

Example:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

These filters detect sudden changes in intensity, typically indicating edges. Examples include the **Sobel**, **Prewitt**, and **Laplacian** filters.

Correlation is another fundamental operation in signal and image processing. It measures the **degree of similarity** between two signals or images. For two 2D signals f(x,y) and h(x,y) correlation is given by:

$$r(x,y) = f(x,y) \circ h(x,y) = \sum_m \sum_n f(m,n) \cdot h(x+m, y+n)$$

Unlike convolution, **the kernel is not flipped** during correlation. This makes correlation useful for comparing or matching two patterns directly.

Imagine you have a small image (a template) and you want to find where it appears in a larger image. You can slide the template across the image and compute the correlation at every position. The regions where the correlation value is high

indicate a close match between the template and the corresponding image region. This process is called **template matching**.

Although convolution and correlation look similar mathematically, there are key differences:

Feature	Convolution	Correlation
Kernel flipping	The kernel is flipped horizontally and vertically.	The kernel is not flipped.
Primary use	Filtering, system analysis, feature extraction.	Pattern recognition, template matching.
Symmetry	Sensitive to kernel orientation.	Direct comparison of structures.
Example function	conv2() or imfilter() in MATLAB.	xcorr2() in MATLAB.

In image processing, the practical difference may not always be noticeable if the kernel is symmetric (for example, Gaussian or averaging filters). However, for asymmetric kernels like edge detectors, the difference becomes significant.

Convolution in image processing can be viewed as the process of applying a "mask" over the image, which transforms each pixel according to its neighbors. This can represent physical phenomena such as diffusion, spreading, or response to an impulse.

Correlation, on the other hand, measures **how similar** two patterns are. In machine vision and pattern recognition, correlation is used to identify where a particular object or shape appears within an image.

For example:

- In medical imaging, correlation helps locate tumors by matching known patterns.
- In robotics, convolution filters are used for edge-based obstacle detection.
- In astronomy, convolution enhances celestial images by removing noise.

Both convolution and correlation share several important properties:

1. **Linearity:**
   Convolution and correlation are linear operations:
   $$f*(ah1+bh2)=a(f*h1)+b(f*h2)$$
2. **Commutativity (for convolution only):**

$$f*h=h*f$$

3. **Associativity:**

$$f*(h1*h2)=(f*h1)*h2$$

4. **Distributivity:**

$$f*(h1+h2)=f*h1+f*h2$$

These properties allow multiple filters to be combined or applied sequentially without changing the final result.

In MATLAB, convolution and correlation are implemented with built-in functions that handle boundary conditions and data types efficiently:

- **conv2()** — performs 2D convolution:
  output = conv2(image, kernel, 'same');
- **imfilter()** — another function for convolution with more control options.
- **xcorr2()** — computes 2D correlation:
  result = xcorr2(image, template);

Visualization of the results through functions like imshow() helps analyze the effects of filtering, blurring, or matching.

Convolution in the spatial domain corresponds to **multiplication in the frequency domain**, as described by the **Convolution Theorem**:

$$f(x,y)*h(x,y)\leftrightarrow F(u,v)\cdot H(u,v)$$

This means that we can perform convolution efficiently by taking the **Fourier Transform** of both the image and the filter, multiplying them in the frequency domain, and then taking the **inverse transform**.

This property is extremely useful for large images where direct convolution is computationally expensive.

1. **Noise Reduction:**
   Smooths an image to remove random variations in intensity.
2. **Edge Detection:**
   Identifies boundaries between regions of differing brightness.
3. **Feature Extraction:**
   Highlights important patterns or structures in an image.
4. **Image Sharpening:**
   Enhances details by amplifying high-frequency components.
5. **Template Matching:**
   Detects specific objects or patterns within a larger image.
6. **Motion Detection:**
   Measures correlation between consecutive frames in a video.
7. **Medical Image Analysis:**
   Detects features like cells, bones, or abnormalities in X-rays or MRI scans.

While performing convolution and correlation, several practical issues must be addressed:

- **Boundary Effects:** At the image edges, some pixels lack neighbors. MATLAB handles this using options such as 'same', 'valid', or 'full'.
- **Kernel Size:** Larger kernels produce stronger blurring but increase computational cost.
- **Normalization:** Filters should often be normalized to ensure brightness consistency.
- **Computation Time:** Convolution with large kernels is computationally expensive, and frequency-domain methods (using FFT) are more efficient.

In summary, convolution and correlation are two foundational operations in 2D signal and image processing. Convolution modifies a signal based on a kernel—used for filtering, smoothing, sharpening, and edge detection. Correlation, in contrast, measures the similarity between signals—used in matching and detection tasks.

Understanding these two operations allows engineers and researchers to design powerful image-processing algorithms. Whether in medical diagnostics, remote sensing, robotics, or surveillance, the ability to convolve or correlate two-dimensional data forms the core of modern visual computing.

Task 1. Performing 2D Convolution using a Custom Kernel

To understand how 2D convolution works by applying a custom kernel (e.g., averaging or sharpening filter) to an image.

Google Colab Code

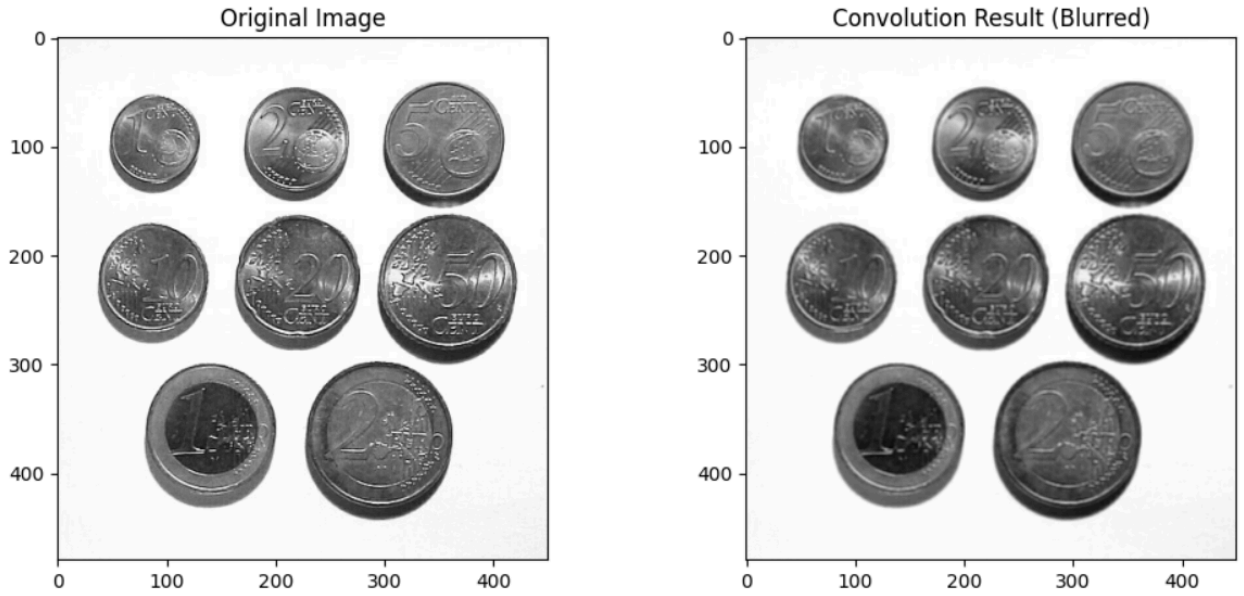
```
# Task 1: 2D Convolution with a custom kernel

import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve2d

# Step 1: Load grayscale image
img = cv2.imread(cv2.samples.findFile("lena.jpg"), cv2.IMREAD_GRAYSCALE)

# Step 2: Define a 3x3 averaging filter
kernel = np.ones((3,3), np.float32) / 9

# Step 3: Perform 2D convolution
conv_result = convolve2d(img, kernel, mode='same', boundary='symm')
```

```
# Step 4: Display results
plt.figure(figsize=(12,5))
plt.subplot(1,2,1); plt.imshow(img, cmap='gray'); plt.title("Original Image")
plt.subplot(1,2,2); plt.imshow(conv_result, cmap='gray'); plt.title("Convolution Result (Blurred)")
plt.show()
```

Explanation

- The convolve2d() function applies convolution between the image and kernel.
- The averaging filter smooths the image and removes noise.

The output image appears **blurred** because high-frequency details (edges) are reduced.



Task 2. Applying Correlation for Template Matching

To perform 2D correlation to locate a small template within a larger image (pattern recognition).

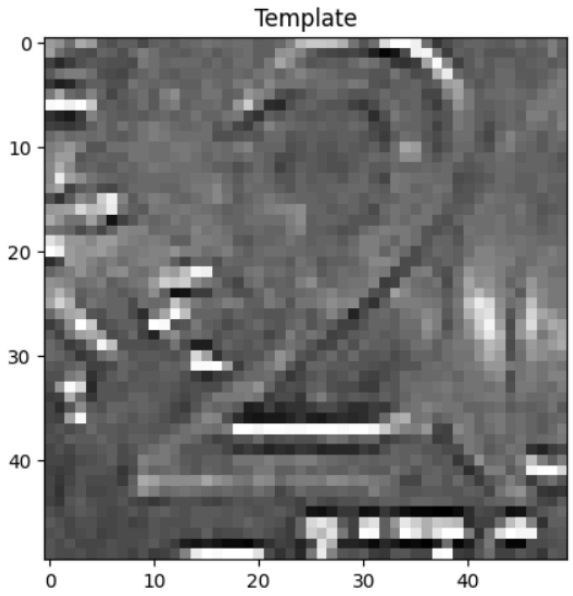
Google Colab Code

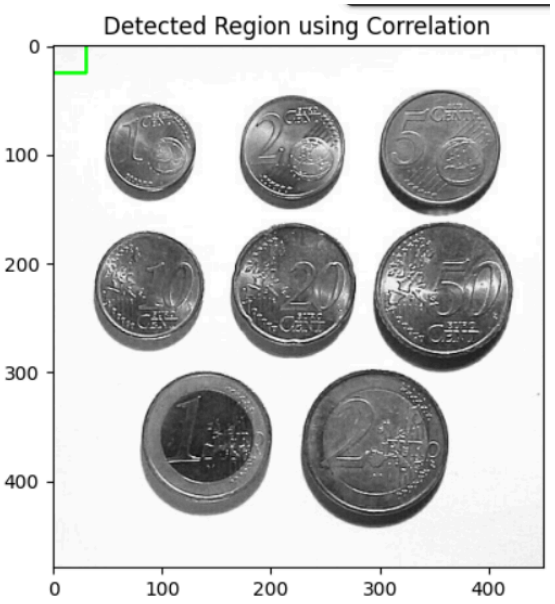
```
# Task 2: Correlation for Template Matching

import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import correlate2d

# Step 1: Load grayscale image
img = cv2.imread(cv2.samples.findFile("lena.jpg"), cv2.IMREAD_GRAYSCALE)

# Step 2: Define a small template (crop a region)
```

```
template = img[200:250, 200:250]  # cropped portion

# Step 3: Perform 2D correlation
corr = correlate2d(img, template, mode='same')

# Step 4: Find peak correlation (best match)
y, x = np.unravel_index(np.argmax(corr), corr.shape)

# Step 5: Draw rectangle around matched region
img_result = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
cv2.rectangle(img_result, (x-25, y-25), (x+25, y+25), (0,255,0), 2)

# Step 6: Display results
plt.figure(figsize=(12,5))
plt.subplot(1,2,1); plt.imshow(template, cmap='gray'); plt.title("Template")
plt.subplot(1,2,2); plt.imshow(img_result[...,::-1]); plt.title("Detected Region using Correlation")
plt.show()
```

Explanation

- Correlation is used to find where a template matches inside an image.
- The correlation peak indicates the **most similar location**.





Task 3. Comparing Convolution and Correlation Outputs

To compare the visual and numerical differences between convolution and correlation when using an asymmetric kernel.

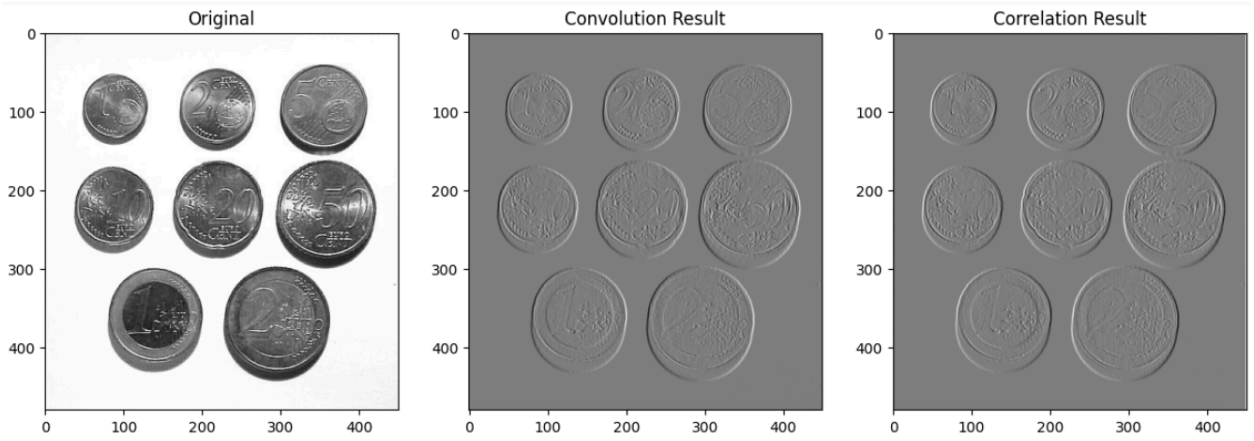
Google Colab Code

```
# Task 3: Compare Convolution and Correlation
```

```
from scipy.signal import convolve2d, correlate2d
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Step 1: Load image
img = cv2.imread(cv2.samples.findFile("lena.jpg"), cv2.IMREAD_GRAYSCALE)

# Step 2: Define an asymmetric kernel (edge detector)
kernel = np.array([[1, 0, -1],
                   [1, 0, -1],
                   [1, 0, -1]])

# Step 3: Perform convolution and correlation
conv_result = convolve2d(img, kernel, mode='same', boundary='symm')
corr_result = correlate2d(img, kernel, mode='same')

# Step 4: Display results
plt.figure(figsize=(15,5))
plt.subplot(1,3,1); plt.imshow(img, cmap='gray'); plt.title("Original")
plt.subplot(1,3,2); plt.imshow(conv_result, cmap='gray'); plt.title("Convolution
Result")
plt.subplot(1,3,3); plt.imshow(corr_result, cmap='gray'); plt.title("Correlation
Result")
plt.show()
```

Explanation

- Convolution flips the kernel before applying, while correlation does not.
- For **symmetric kernels**, results are similar.
- For **asymmetric kernels**, the difference is clearly visible.



Task 4. Edge Detection using Convolution Filters

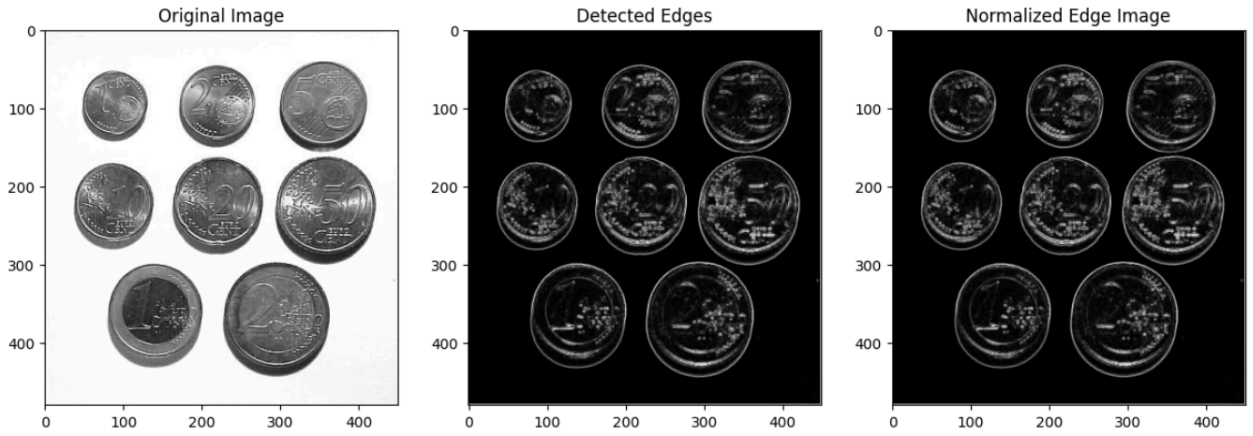
To detect edges in an image using **Sobel** and **Laplacian** convolution filters.

Google Colab Code

```
# Task 4: Edge Detection using Convolution Filters

import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve2d

# Step 1: Load grayscale image
img = cv2.imread(cv2.samples.findFile("lena.jpg"), cv2.IMREAD_GRAYSCALE)

# Step 2: Define Sobel kernels
sobel_x = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])

sobel_y = np.array([[-1, -2, -1],
                    [0,  0,  0],
                    [1,  2,  1]])

# Step 3: Apply convolution
gx = convolve2d(img, sobel_x, mode='same')
gy = convolve2d(img, sobel_y, mode='same')

# Step 4: Compute gradient magnitude
edges = np.sqrt(gx**2 + gy**2)

# Step 5: Display results
plt.figure(figsize=(15,5))
plt.subplot(1,3,1); plt.imshow(img, cmap='gray'); plt.title("Original Image")
plt.subplot(1,3,2); plt.imshow(edges, cmap='gray'); plt.title("Detected Edges")
plt.subplot(1,3,3); plt.imshow(np.uint8(edges / np.max(edges) * 255),
cmap='gray'); plt.title("Normalized Edge Image")
plt.show()
```

Explanation

- Sobel filters compute gradients in horizontal and vertical directions.
- The combination gives the magnitude of edges.



Assignments for students:

Task 1: Load and Convert an Image to Grayscale

- Import necessary libraries: numpy, matplotlib.pyplot, cv2.
- Load any image from your computer or from an online source.
- Convert the image to grayscale using cv2.cvtColor().
- Display both the **original** and **grayscale** images side by side using matplotlib.

Task 2: Apply 2D Convolution Using a Custom Kernel

- Create a **3×3 kernel** manually in NumPy.
  Example: an averaging or edge-detection kernel.
- Use cv2.filter2D() to convolve the image with this kernel.
- Display the original image and the filtered result.
- Print the kernel you used below the output.

Task 3: Perform 2D Correlation and Compare with Convolution

- Use the same image and kernel from Task 2.
- Apply **2D correlation** using scipy.signal.correlate2d().
- Normalize the result to display it properly.
- Show convolution and correlation outputs side by side for comparison.

Task 4: Experiment with Different Filters

- Create and apply the following three filters separately:
    1. **Smoothing filter (averaging)**
    2. **Sharpening filter**
    3. **Edge detection filter**
- Display all three results in one figure (use plt.subplot()).
- Briefly describe the visual effect of each filter in a markdown cell.

Control Questions (12 questions)

1. What is a **two-dimensional signal**, and how is it represented in image processing?
2. Explain the **main difference** between 1D and 2D convolution.
3. Write the **mathematical formula** for 2D convolution and describe each term.
4. What is the **purpose of flipping** the kernel during convolution?
5. How does the **kernel size** (e.g., 3×3 vs. 7×7) affect the resulting image?
6. In which situations is **correlation** used instead of convolution?
7. What is the **difference between convolution and correlation** in terms of kernel orientation?
8. What is the role of **filter kernels** in image processing? Give at least two examples.
9. Describe what happens when a **smoothing filter** is applied to an image.
10. Describe the effect of a **sharpening filter** on an image.
11. Why is **edge detection** important in image analysis?
12. What could happen if you apply convolution repeatedly to the same image?