CS419 Digital Image and Video Analysis, Fall 2024 Assignment 2

Melih Kaan Sahinbas

October 30, 2024

1 Question 1: Proving Commutativity of Convolution

Question Statement

We want to prove that for two functions $f(x)$ and $g(x)$, the convolution operation is commutative, meaning:

$$f(x) * g(x) = g(x) * f(x)$$

First Part: Commutativity of Convolution

1. **Convolution Definition**:
The convolution of two functions f and g is defined as:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)\,dy$$

2. **Using a Change of Variables**:
To demonstrate commutativity, change of variables can be used.
- Let $x - y = u$, which equals to $y = x - u$. - Then derivatives are, $dy = -du$.
3. **Applying the Change of Variables**:
Substituting $y = x - u$ and $dy = -du$ into the integral, we get:

$$(f * g)(x) = \int_{\infty}^{-\infty} f(x-u)g(u)(-du) = \int_{-\infty}^{\infty} g(u)f(x-u)\,du$$

Lets calculate:

$$(g * f)(x) = \int_{-\infty}^{\infty} g(u)f(x-u)\,du$$

Thus, we have:

$$(f * g)(x) = \int_{-\infty}^{\infty} g(u)f(x-u)\,du = (g * f)(x)$$

This shows that:

$$f(x) * g(x) = g(x) * f(x)$$

Therefore, **convolution is commutative**.

Second Part: Non-Commutativity of Cross-Correlation

The solution also includes an exploration of **cross-correlation**. Cross-correlation is similar to convolution but does not involve flipping one of the functions. We are asked to prove that **cross-correlation is not commutative**.

1. **Definition of Cross-Correlation**:
The cross-correlation of two functions f and g is defined as:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(y)g(x+y)\,dy$$

2. **Attempting Commutativity**:

To check if $f \star g$ is commutative change of variables can be applied.
- Let $x + y = u$, which equals to $y = u - x$. - Then derivatives are, $dy = du$.
3. **Substituting the Change of Variables**:
Substituting $y = u - x$ and $dy = du$ into the integral, we get:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(u-x)g(u)\,du$$

Lets calculate:

$$(g \star f)(x) = \int_{-\infty}^{\infty} g(y)f(x+y)\,dy$$

However, this expression is **not the same as** $g \star f$, which would be:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(u - x)g(u)\, du \neq \int_{-\infty}^{\infty} g(y)f(x + y)\, dy$$

Therefore, we conclude that:

$$f(x) \star g(x) \neq g(x) \star f(x)$$

So, **cross-correlation is not commutative**.

2 Question 2: Proving Rotation Invariance of the Laplacian Operator

Question Statement

To prove that the Laplacian operator is rotation invariant. This means that if we rotate the coordinates of an image function $f(x, y)$, the Laplacian operator applied to the rotated coordinates should yield the same result as in the original coordinates.

Solution

1. Showing the Provided Rotated Coordinates

Let the original coordinates be (x, y) and the rotated coordinates are given as (x', y'). The rotation by an angle θ is given by:

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

2. Deriving the Partial Derivatives in Terms of Rotated Coordinates

To express the Laplacian in terms of the rotated coordinates, we need to calculate the partial derivatives $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial y}$ with respect to x' and y'.

First Partial Derivative with Respect to x

Using the chain rule, we have:

$$\frac{\partial}{\partial x} = \frac{\partial}{\partial x'}\frac{\partial x'}{\partial x} + \frac{\partial}{\partial y'}\frac{\partial y'}{\partial x}$$

Calculating the partial derivatives:

$$\frac{\partial x'}{\partial x} = \cos(\theta), \quad \frac{\partial y'}{\partial x} = \sin(\theta)$$

Thus,

$$\frac{\partial}{\partial x} = \cos(\theta)\frac{\partial}{\partial x'} + \sin(\theta)\frac{\partial}{\partial y'}$$

Second Partial Derivative with Respect to x

Now we calculate $\frac{\partial^2}{\partial x^2}$:

$$\frac{\partial^2}{\partial x^2} = \frac{\partial}{\partial x}\left(\cos(\theta)\frac{\partial}{\partial x'} + \sin(\theta)\frac{\partial}{\partial y'}\right)$$

Expanding this, the following equation is obtained because $\left(\frac{\partial}{\partial x}\right)^2$ equals to the following equation:

$$\frac{\partial^2}{\partial x^2} = \cos^2(\theta)\frac{\partial^2}{\partial x'^2} + 2\cos(\theta)\sin(\theta)\frac{\partial^2}{\partial x'\partial y'} + \sin^2(\theta)\frac{\partial^2}{\partial y'^2}$$

First Partial Derivative with Respect to y

Similarly, we find:

$$\frac{\partial}{\partial y} = \frac{\partial}{\partial x'}\frac{\partial x'}{\partial y} + \frac{\partial}{\partial y'}\frac{\partial y'}{\partial y}$$

Calculating the partial derivatives:

$$\frac{\partial x'}{\partial y} = -\sin(\theta), \quad \frac{\partial y'}{\partial y} = \cos(\theta)$$

Thus,

$$\frac{\partial}{\partial y} = -\sin(\theta)\frac{\partial}{\partial x'} + \cos(\theta)\frac{\partial}{\partial y'}$$

Second Partial Derivative with Respect to y

Now we calculate $\frac{\partial^2}{\partial y^2}$:

$$\frac{\partial^2}{\partial y^2} = \frac{\partial}{\partial y}\left(-\sin(\theta)\frac{\partial}{\partial x'} + \cos(\theta)\frac{\partial}{\partial y'}\right) = \left(-\sin(\theta)\frac{\partial}{\partial x'} + \cos(\theta)\frac{\partial}{\partial y'}\right)^2$$

Expanding this, the following equation is obtained because $\left(\frac{\partial}{\partial x}\right)^2$ equals to the following equation:

$$\frac{\partial^2}{\partial y^2} = \sin^2(\theta)\frac{\partial^2}{\partial x'^2} + \cos^2(\theta)\frac{\partial^2}{\partial y'^2} - 2\sin(\theta)\cos(\theta)\frac{\partial^2}{\partial x'\partial y'}$$

3. Summation of the Second Derivatives

To compute the Laplacian, we add $\frac{\partial^2}{\partial x^2}$ and $\frac{\partial^2}{\partial y^2}$:

$$\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = \cos^2(\theta)\frac{\partial^2}{\partial x'^2} + \sin^2(\theta)\frac{\partial^2}{\partial x'^2} + \cos^2(\theta)\frac{\partial^2}{\partial y'^2} + \sin^2(\theta)\frac{\partial^2}{\partial y'^2}$$

$$+2\cos(\theta)\sin(\theta)\frac{\partial^2}{\partial x'\partial y'} - 2\sin(\theta)\cos(\theta)\frac{\partial^2}{\partial x'\partial y'}$$

$$\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2} = \left(\cos^2(\theta) + \sin^2(\theta)\right)\frac{\partial^2}{\partial x'^2}+\left(\cos^2(\theta) + \sin^2(\theta)\right)\frac{\partial^2}{\partial y'^2}-(2\cos(\theta)\sin(\theta) - 2\sin(\theta)\cos(\theta))\frac{\partial^2}{\partial x'\partial y'}$$

As you know $\sin^2(\theta) + \cos^2(\theta) = 1$ and $2\cos(\theta)\sin(\theta) - 2\sin(\theta)\cos(\theta) = 0$

simplifying this expression results the following equation:

$$\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = \frac{\partial^2}{\partial x'^2} + \frac{\partial^2}{\partial y'^2}$$

4. Conclusion

We have shown that:

$$\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = \frac{\partial^2}{\partial x'^2} + \frac{\partial^2}{\partial y'^2}$$

Thus, the Laplacian operator is invariant under rotation since as explained in the previous parts x' and y' are rotated versions of x and y, and their Laplacian's are equal to each other:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \frac{\partial^2 f}{\partial x'^2} + \frac{\partial^2 f}{\partial y'^2}$$

This shows that the Laplacian is invariant under rotation.

3 Question 3: Proving Linearity of the Filter and Convolution Mask h

Given a filter h defined by:

$$h(x,y) = 3f(x,y) + 2f(x-1,y) + 2f(x+1,y) - 17f(x,y-1) + 99f(x,y+1)$$

we need to verify if h is a linear filter by proving additivity and homogeneity. If h is a linear filter, convolution matrix should be found.

1. Additivity

To show that the filter h is additive, we start by applying h to the sum of two functions f_1 and f_2:

$$\begin{aligned} h(f_1+f_2)(x,y) = {} & 3(f_1(x,y)+f_2(x,y)) + 2(f_1(x-1,y)+f_2(x-1,y)) \\ & + 2(f_1(x+1,y)+f_2(x+1,y)) - 17(f_1(x,y-1)+f_2(x,y-1)) \\ & + 99(f_1(x,y+1)+f_2(x,y+1)). \end{aligned}$$

Distributing each coefficient:

$$\begin{aligned} h(f_1+f_2)(x,y) = {} & (3f_1(x,y) + 2f_1(x-1,y) + 2f_1(x+1,y) - 17f_1(x,y-1) + 99f_1(x,y+1)) \\ & + (3f_2(x,y) + 2f_2(x-1,y) + 2f_2(x+1,y) - 17f_2(x,y-1) + 99f_2(x,y+1)). \end{aligned}$$

This can be written as:
We know that:

$$\begin{aligned} h(f_1)(x,y) + h(f_2)(x,y) = {} & (3f_1(x,y) + 2f_1(x-1,y) + 2f_1(x+1,y) - 17f_1(x,y-1) + 99f_1(x,y+1)) \\ & + (3f_2(x,y) + 2f_2(x-1,y) + 2f_2(x+1,y) - 17f_2(x,y-1) + 99f_2(x,y+1)). \end{aligned}$$

These equations shows us that:

$$h(f_1+f_2)(x,y) = h(f_1)(x,y) + h(f_2)(x,y).$$

Thus, h satisfies the additivity propert.

2. Homogeneity

To show that the filter h is homogeneous, we apply h to a scaled function af, where a is a constant:

$$\begin{aligned} h(af)(x,y) = {} & 3(af(x,y)) + 2(af(x-1,y)) + 2(af(x+1,y)) \\ & - 17(af(x,y-1)) + 99(af(x,y+1)). \end{aligned}$$

Factoring out a from each term, we get:

$$h(af)(x,y) = a\,(3f(x,y) + 2f(x-1,y) + 2f(x+1,y) - 17f(x,y-1) + 99f(x,y+1)).$$

Thus,

$$h(af)(x,y) = ah(f)(x,y),$$

which shows that h satisfies the homogeneity property.

Since h satisfies both additivity and homogeneity, we conclude that h is a linear filter.

3. Creating the Convolution Matrix (Kernel) for h

The filter $h(x,y)$ can be represented as a 3x3 convolution matrix (or kernel), where each entry corresponds to the coefficient of f at specific locations relative to (x,y):

Convolution Matrix for Filter $h(x, y)$ as a Weighted Sum

Convolution uses a weighted sum of the values of adjacent pixels from the input picture f to calculate the value of $h(x, y)$ at each pixel position (x, y). This can be represented mathematically as:

$$h(x, y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w(i, j) \cdot f(x + i, y + j)$$

where:

- $f(x + i, y + j)$ represents the value of the pixel at position $(x + i, y + j)$ relative to (x, y) in the input image.
- $w(i, j)$ represents the corresponding weight (or coefficient) from the convolution matrix for the neighboring position (i, j).

In this case, the filter $h(x, y)$ can specifically be defined using the 3x3 convolution matrix w:

$$w = \begin{bmatrix} 0 & -17 & 0 \\ 2 & 3 & 2 \\ 0 & 99 & 0 \end{bmatrix}$$

Each $w(i, j)$ in this matrix is a coefficient applied to a relative pixel location in the neighborhood of $f(x, y)$, giving each neighboring pixel a unique weight in the sum. Let's break down the role of each coefficient in this matrix.

Detailed Explanation of Each Coefficient

In this convolution matrix, each element $w(i, j)$ corresponds to a position in the neighborhood of $f(x, y)$ and is used to achieve specific filtering effects:

1. **Center Coefficient $w(0, 0) = 3$:**
   - The center coefficient $w(0, 0)$ corresponds to the pixel $f(x, y)$ itself.
2. **Horizontal Neighbor Coefficients $w(-1, 0) = 2$ and $w(1, 0) = 2$:**
   - These coefficients correspond to the left $f(x - 1, y)$ and right $f(x + 1, y)$ horizontal neighbors of $f(x, y)$.
3. **Vertical Neighbor Coefficients $w(0, 1) = 99$ and $w(0, -1) = -17$:**
   - The coefficient $w(0, 1) = 99$ corresponds to the top neighbor $f(x, y+1)$, while $w(0, -1) = -17$ corresponds to the bottom neighbor $f(x, y - 1)$.
4. **Zero Coefficients for Diagonal Neighbors:**
   - The diagonal neighbors $(w(-1, -1), w(1, -1), w(-1, 1), w(1, 1))$ all have zero coefficients, meaning these pixels do not contribute to the output.

General Formula with Weights

The overall computation at each pixel $h(x, y)$ can thus be summarized as:

$$h(x, y) = 3 \cdot f(x, y) + 2 \cdot f(x - 1, y) + 2 \cdot f(x + 1, y) + 99 \cdot f(x, y + 1) - 17 \cdot f(x, y - 1)$$

This formula shows how each coefficient in w (the convolution matrix) multiplies the corresponding pixel in f and contributes to the output at $h(x, y)$.

$$\begin{bmatrix} 0 & 99 & 0 \\ 2 & 3 & 2 \\ 0 & -17 & 0 \end{bmatrix}$$

In summary, the convolution matrix w is designed to apply specific weights to each neighbor of $f(x, y)$, creating a filtered output $h(x, y)$ that emphasizes, blends, or contrasts regions in the image according to

the filter's purpose. This configuration allows the convolution to detect specific patterns, such as edges or smooth regions, based on the distribution of weights in the matrix.

This convolution matrix can be used to apply the filter h to an image by performing a convolution operation, which involves placing this kernel over each pixel and calculating a weighted sum of the pixel and its neighbors according to the matrix values.

4 Two-Pass Connected Components Algorithm

The goal of the two-pass connected components algorithm is to identify and label all connected regions in a binary image. This algorithm operates by scanning the image twice and utilizing a Union-Find data structure to manage label equivalences. Below, we describe the main algorithm and its helper functions.

4.1 Helper Functions

To implement the two-pass connected components algorithm, we use three helper functions: `union`, `find`, and `initialize`. These functions work together to maintain the Union-Find structure, which helps track equivalent labels during the first pass.

4.1.1 `union(x, y, parent)` Function

This function performs the union of the sets containing labels `x` and `y`. It updates the `parent` array to reflect the union operation by making one label the parent of the other, effectively merging the two sets.

- **Parameters:**
  - `x`: The label of the first set to be united.
  - `y`: The label of the second set to be united.
  - `parent`: An array representing the union-find structure, where each index points to its parent label.
- **Returns:** None. The function modifies `parent` in place to merge the two sets.

4.1.2 `find(x, parent)` Function

This function finds the root of the set containing label `x` with path compression. Using recursion, it updates each node's parent to the root label of the set, optimizing future queries.

- **Parameters:**
  - `x`: The label whose root we want to find.
  - `parent`: An array where each index points to its "parent" label in the union-find structure.
- **Returns:** The root label of the set containing `x`.

4.1.3 `initialize(max_label)` Function

This function initializes the label counter and the `parent` array. Each element in the `parent` array initially points to itself, supporting the union-find structure for connected component labeling.

- **Parameters:**
  - `max_label`: The maximum number of labels that could be created (typically `rows * cols / 2`).
- **Returns:**
  - `label`: The initial label counter, starting from 0.
  - `parent_arr`: An array of size `max_label`, initialized so that each element points to itself.

4.2 Main Algorithm

The two-pass connected components algorithm proceeds as follows:

1. Define the maximum number of rows and columns based on the shape of the image, and compute the maximum number of labels as `max_label = max_row * max_col // 2`.
2. Initialize the label counter and parent array by calling `initialize(max_label)`. This returns `label` (initially 0) and `parent_arr`, where each element points to itself.
3. Traverse each pixel of the image using nested loops (for each row and column) and process foreground pixels (e.g., those with value 255):
   (a) Identify prior neighbors based on the connectivity parameter (either 4 or 8 connectivity). Check whether the neighbors have already been labeled.
   (b) If there are no labeled prior neighbors, assign a new label to the current pixel, increment the label counter, and update `parent_arr`.
   (c) If labeled neighbors exist, assign the current pixel the minimum label from these neighbors and record equivalences using the `union` function. This ensures that all equivalent labels are linked in the union-find structure.
4. After the first pass, perform a second pass over the foreground pixels. Replace each preliminary label with its root label, effectively merging equivalent labels using the `find` function.

4.3 Final Label Mapping

Once the two-pass algorithm is completed, the labels are sorted to ensure they are in ascending order, as required by the coloring algorithm provided in the assignment file. This is done as follows:

- Extract all unique labels from the label matrix `LB` and create a dictionary that maps each unique label to a new, sequential label.
- Update `parent_arr` and `LB` using the new label mapping, so that each connected component is assigned a unique, ordered label.

4.4 Full Function Implementation

Here is the full implementation of the Two-Pass Connected Components algorithm with helper functions `union`, `find`, and `initialize`.

```
def two_pass_connected_components(image, connectivity=4):
    def find(x, parent):
        if parent[x] != x:
            parent[x] = find(parent[x], parent)
        return parent[x]

    def union(x, y, parent):
        temp_x = x
        temp_y = y
        while parent[temp_x] != temp_x:
            temp_x = parent[temp_x]
        while parent[temp_y] != temp_y:
            temp_y = parent[temp_y]
        if temp_x != temp_y:
            parent[temp_y] = temp_x

    def initialize(max_lab):
        label = 0
        parent_arr = np.arange(max_lab, dtype=int)
        return label, parent_arr
```

```
    max_row, max_col = image.shape
    max_lab = max_row * max_col // 2
    label, parent_arr = initialize(max_lab)

    LB = np.zeros_like(image, dtype=int)
    for row in range(max_row):
        for col in range(max_col):
            if image[row, col] == 255:
                prior_neighbours = []
                if row > 0 and LB[row - 1, col] > 0:
                    prior_neighbours.append(LB[row - 1, col])
                if col > 0 and LB[row, col - 1] > 0:
                    prior_neighbours.append(LB[row, col - 1])
                if connectivity == 8:
                    if row > 0 and col > 0 and LB[row - 1, col - 1] > 0:
                        prior_neighbours.append(LB[row - 1, col - 1])
                    if row > 0 and col < max_col - 1 and LB[row - 1, col + 1] > 0:
                        prior_neighbours.append(LB[row - 1, col + 1])

                if not prior_neighbours:
                    LB[row, col] = label
                    parent_arr[label] = label
                    label += 1
                else:
                    temp_label = min(prior_neighbours)
                    LB[row, col] = temp_label
                    for labels in prior_neighbours:
                        union(temp_label, labels, parent_arr)

    for row in range(max_row):
        for col in range(max_col):
            if image[row, col] == 255:
                LB[row, col] = find(LB[row, col], parent_arr)

    unique_labels = np.unique(LB)
    label_mapping = {old_label: new_label for new_label, old_label in enumerate(unique_labels)}
    for old_label, new_label in label_mapping.items():
        parent_arr[old_label] = new_label
    for row in range(max_row):
        for col in range(max_col):
            if LB[row, col] > 0:
                LB[row, col] = label_mapping[LB[row, col]]

    num_labels = len(unique_labels)
    return num_labels, LB
```

4.5 Conclusion

This two-pass connected components algorithm labels each connected region in a binary image. The `union` and `find` functions maintain equivalences among labels, and the final label mapping ensures labels are in ascending order, suitable for visualization or further processing.

Below you can see the sample usage of Two Pass Connected Components Algorithm

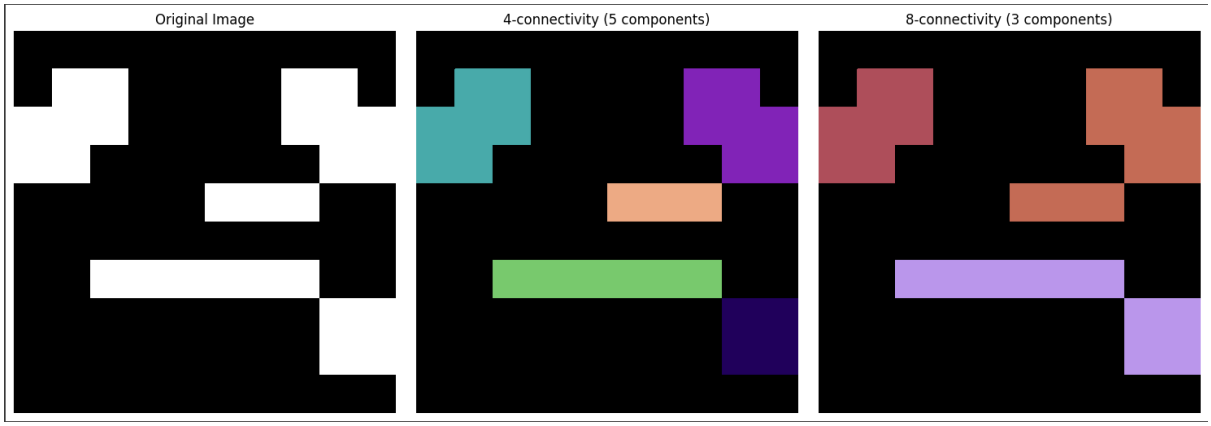

Figure 1: Output of the Two Pass Connected Component Algorithm using 4-connectivity and 8-connectivity

As you can see, connected components labeled correctly for different connectivity values.