SSA Linear Algebra Final

Image Convolution in MATLAB (and a bit of Python)

Evan Xiang[1] Vivan Poddar[1] Rohan Khera[1] Beau Brush[1]

[1]Mathematics Department
Shady Side Academy Senior School

December 16th, 2024

Table of Contents

Convolution Overview

Definition

A convolution is done by multiplying a pixel's and its neighboring pixels color value by a matrix.

Generalized Matrix Convolution Formula

$$(F * G)(x, y) = \sum_m \sum_n F(x - m, y - n)G(m, n)$$

Where:

- x and y represent the current position within the output matrix
- m and n are generalized variables representing the shift of the matrix with regard to $G(m, n) \rightarrow$ the kernel matrix

Convolution Cont.

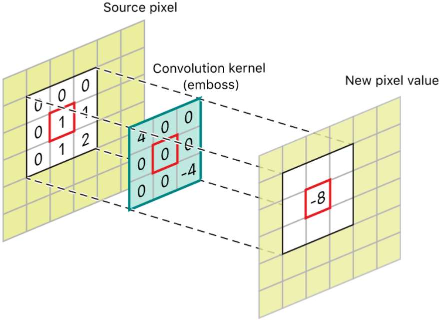


Figure: A general description of the convolution operation. (Photo downloaded from Apple Developer Documentation, CC BY-SA 4.0)

Summary of the Convolution Operation

- Place the kernel $G(m, n)$ over the input matrix $F(x, y)$ such that the kernel's center aligns with the current position (x, y) of the output matrix.
- For every element of the kernel $G(m, n)$, multiply it with the corresponding element of the input matrix $F(x - m, y - n)$.
- Sum all the resulting products to compute the value for the current position (x, y) in the output matrix.

$$(F * G)(x, y) = \sum_m \sum_n F(x - m, y - n) G(m, n)$$

- Slide the kernel over the input matrix to the next position and repeat steps 2 and 3. This involves systematically shifting the kernel across all valid positions in the input matrix.

Weierstrass Transform (Gaussian Filter)

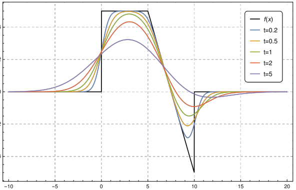


Figure: Weierstrass Transform for 5 parameters of t. The green line represents the standard Weierstrass. (Photo downloaded from Glosser.ca - Own work, CC BY-SA 4.0)

Definition

The Weierstrass transform applies a 2D Gaussian Kernel to an image to reduce noise and create a blurred version of the original function by taking a weighted average of the function's values with the degree of smoothing controlled by the Gaussian's variance.

Gaussian Cont.

Formula

$$O(i,j) = \sum_{x=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \sum_{y=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \left(\frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \right) \cdot I(i-x, j-y)$$

Where:

- N represents the size of the kernel
- $(\frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}})$ represents the Gaussian kernel
- $O(i,j)$ represents the output

Cauchy Kernel

Cauchy Matrix Convolution Formula

$$O(i,j) = \sum_{x=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \sum_{y=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \frac{1}{\pi\gamma\left(1+\frac{x^2+y^2}{\gamma^2}\right)} \cdot I(i-x, j-y)$$

Note that this looks relatively similar to the Weierstrass Transform with the Gaussian! But how does it differ...

Gaussian vs. Cauchy Kernel

The primary difference arises in the difference in the kernel decay rate. The Gaussian, represented by $K_{\text{Gaussian}}(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$, decreases exponentially. The Cauchy, represented by $K_{\text{Cauchy}}(r) = \frac{\gamma}{\pi(\gamma^2+r^2)}$, decreases algebraically, allowing for longer distance influences.

Box Blur Method

Formula

$$O(i,j) = \frac{1}{N^2} \sum_{x=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \sum_{y=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} I(i-x, j-y)$$

See how this is much simpler than all the previous algorithms... There's a reason for this... I mean it's literally just a weighted average of the values covered by the kernel applied to all values in the image matrix.

The Gaussian in Disguise

Proof

The box blur operation is a convolution of the image I with $K(x, y)$, producing a blurred output $O(x, y)$. Repeated application of the box blur corresponds to convolving $K(x, y)$ with itself n times. After n convolutions, the resulting kernel $K_n(x, y)$ is given by:

$$K_n(x, y) = K(x, y) * K(x, y) * \cdots * K(x, y) \quad \text{(n times)}.$$

By the Central Limit Theorem, the sum of n independent random variables converges to a Gaussian distribution as $n \to \infty$ with finite mean and variance. The kernel $K(x, y)$ acts as a probability distribution. Since the uniform box kernel $K(x, y)$ satisfies the conditions of the CLT, the repeated convolution $K_n(x, y)$ converges to a Gaussian kernel $G(x, y)$:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right),$$

where the standard deviation σ grows proportionally to $\sqrt{n}$.
Thus, repeated application of the box blur approximates the Gaussian Kernel as $n \to \infty$.

Kernel Implementation


```
% kernel creation functions
function gaussianKernel = createGaussianKernel(kernelSize, sigma)
    [x, y] = meshgrid(-floor(kernelSize/2):floor(kernelSize/2), -floor(kernelSize/2):floor(kernelSize/2));
    gaussianKernel = exp(-(x.^2 + y.^2) / (2 * sigma^2));
    gaussianKernel = gaussianKernel / sum(gaussianKernel(:));
end

function cauchyKernel = createCauchyKernel(kernelSize, gamma)
    [x, y] = meshgrid(-floor(kernelSize/2):floor(kernelSize/2), -floor(kernelSize/2):floor(kernelSize/2));
    cauchyKernel = 1 ./ (1 + (x.^2 + y.^2) / gamma^2);
    cauchyKernel = cauchyKernel / sum(cauchyKernel(:));
end

function boxKernel = createBoxKernel(kernelSize)
    boxKernel = ones(kernelSize, kernelSize) / (kernelSize^2);
end
```

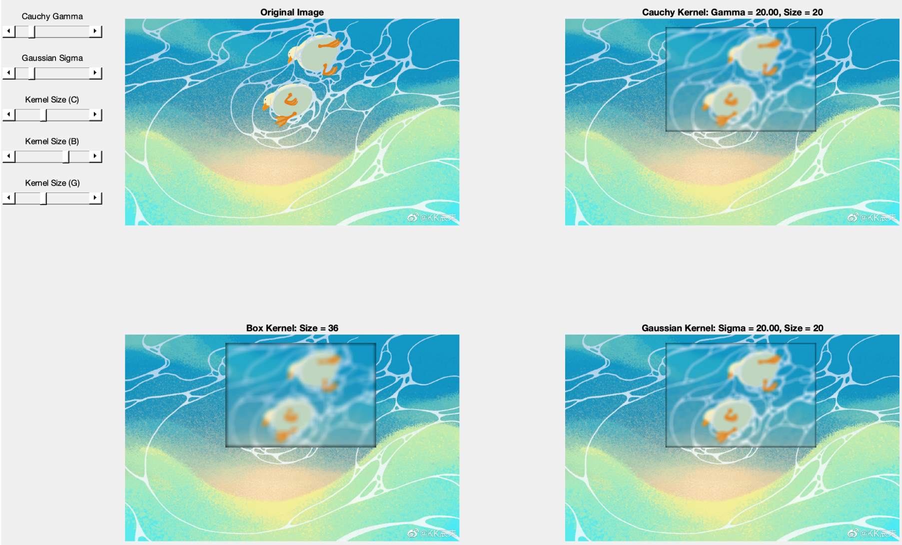
Figure: Implementation of all 3 kernels in MATLAB (.mat) code. Each function defines a separate type of kernel which can be called to slide across the pixels of an image.

Sliding Window

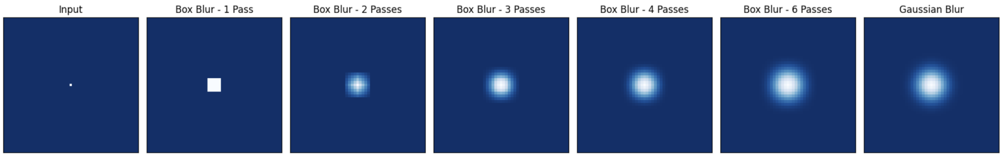
```
for channel = 1:3
    region = double(image(y:y+height-1, x:x+width-1, channel));
    cauchyBlurredRegion = zeros(size(region));
    boxBlurredRegion = zeros(size(region));  % Replaced laplacian with box
    gaussianBlurredRegion = zeros(size(region));
    [kHeight, kWidth] = size(cauchyKernel);
    padHeight = floor(kHeight / 2);
    padWidth = floor(kWidth / 2);
    paddedRegion = padarray(region, [padHeight, padWidth], 0, 'both');
    for i = 1:size(region, 1)
        for j = 1:size(region, 2)
            localWindow = paddedRegion(i:i+kHeight-1, j:j+kWidth-1);
            cauchyBlurredRegion(i, j) = sum(sum(localWindow .* cauchyKernel));
            boxBlurredRegion(i, j) = sum(sum(localWindow .* boxKernel));  % Replaced laplacian with box
            gaussianBlurredRegion(i, j) = sum(sum(localWindow .* gaussianKernel));
        end
    end
    cauchyBlurredImage(y:y+height-1, x:x+width-1, channel) = uint8(cauchyBlurredRegion);
    boxBlurredImage(y:y+height-1, x:x+width-1, channel) = uint8(boxBlurredRegion);  % Replaced laplacian with box
    gaussianBlurredImage(y:y+height-1, x:x+width-1, channel) = uint8(gaussianBlurredRegion);
end
```

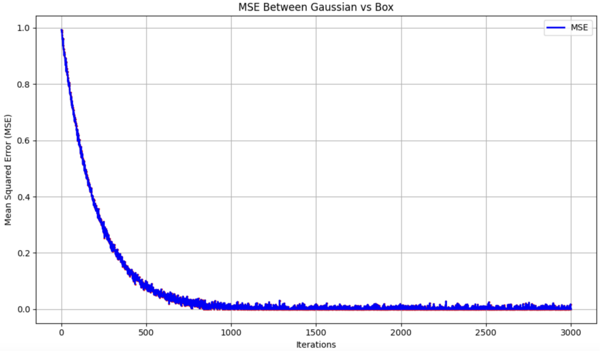
Figure: Implementation of the full convolution operation, using the 3 separate kernels defined in the previous slide.

Final Developed Interface



Quod erat demonstrandum!





Code for Proof

3s

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def gaus2d(x=0, y=0, mx=0, my=0, sx=1, sy=1):
    return 1. / (2. * np.pi * sx * sy) * np.exp(-((x - mx)**2. / (2. * sx**2.) + (y - my)**2. / (2. * sy**2.)))
image = np.ones((49, 49)) * 255
image[24][24] = 0
gblur = cv2.GaussianBlur(image, (31, 31), 3.5, 3.5)
box_blurs = [image]
for i in range(6):
    box_blurs.append(cv2.boxFilter(box_blurs[-1], -1, (5, 5)))
plt.figure(figsize=(18, 3))
titles = ["Input"] + [f"Box Blur - {i} Pass{'es' if i > 1 else ''}" for i in range(1, 7)] + ["Gaussian Blur"]
for i, (title, img) in enumerate(zip(titles, box_blurs + [gblur])):
    plt.subplot(1, len(titles), i + 1)
    plt.title(title)
    plt.imshow(img, cmap='Blues')
    plt.gca().get_xaxis().set_visible(False)
    plt.gca().get_yaxis().set_visible(False)
plt.tight_layout()
plt.savefig("box.jpg", facecolor='white', dpi=300)
plt.show()
```

Figure: Python code that displays the repeated box blurs after each iteration, in addition to the Gaussian blur.

Future Work

Future work would most likely involve the implementation of an algorithm to sharpen images using Laplacian of Gaussian.

Laplacian of Gaussian (LoG)

We apply the Laplacian operator directly to the Gaussian kernel, resulting in the following equation:

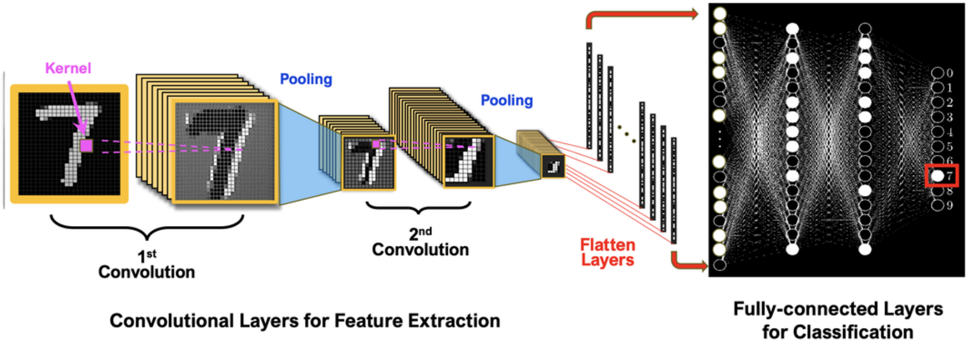
$$LoG(x,y) = \Delta\left(G(x,y) * I(x,y)\right)$$

Then, we expand this with full equations to:

$$\Delta G(x,y) = \left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right) \cdot \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

This equation lends itself to edge detection, as the Laplacian blurring function may selectively blur non-edges, while the Gaussian smooths out noise, resulting in edge-detection with LoG.

Convolutional Neural Networks (ConvNet)



Thanks for Listening

- All information for implementation of our algorithm may be found here: https://github.com/evankxiang/ssalinalgfinal
- We have included CSV files of the image convolution algorithm and all raw code for the algorithm (.mat files). Furthermore, we include our old test files, primarily consisting of non-selective image blurring (no ability to select a bounding box). Finally, we include a preliminary sharpening algorithm using LoG that is a WIP.
- EX, VP completed the programming segment. EX, RK, BB completed the slideshow and testing of the algorithms. Note, we are NOT listed in order of contribution.