CS195: Computer Vision

Image Filtering

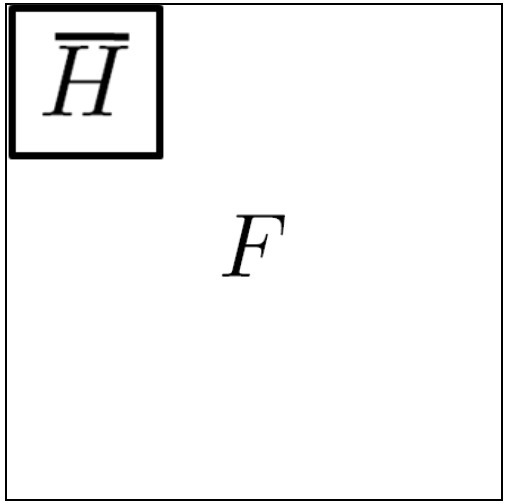
Convolution

September 04, 2024

Md Alimoor Reza
Assistant Professor of Computer Science

Convolution

- Same as cross-correlation, except that the kernel is "flipped" (horizontally and vertically)



Adapted from F. Durand

Convolution

- Same as cross-correlation, except that the kernel is "flipped" (first horizontally and then vertically)

$$G = H * F$$

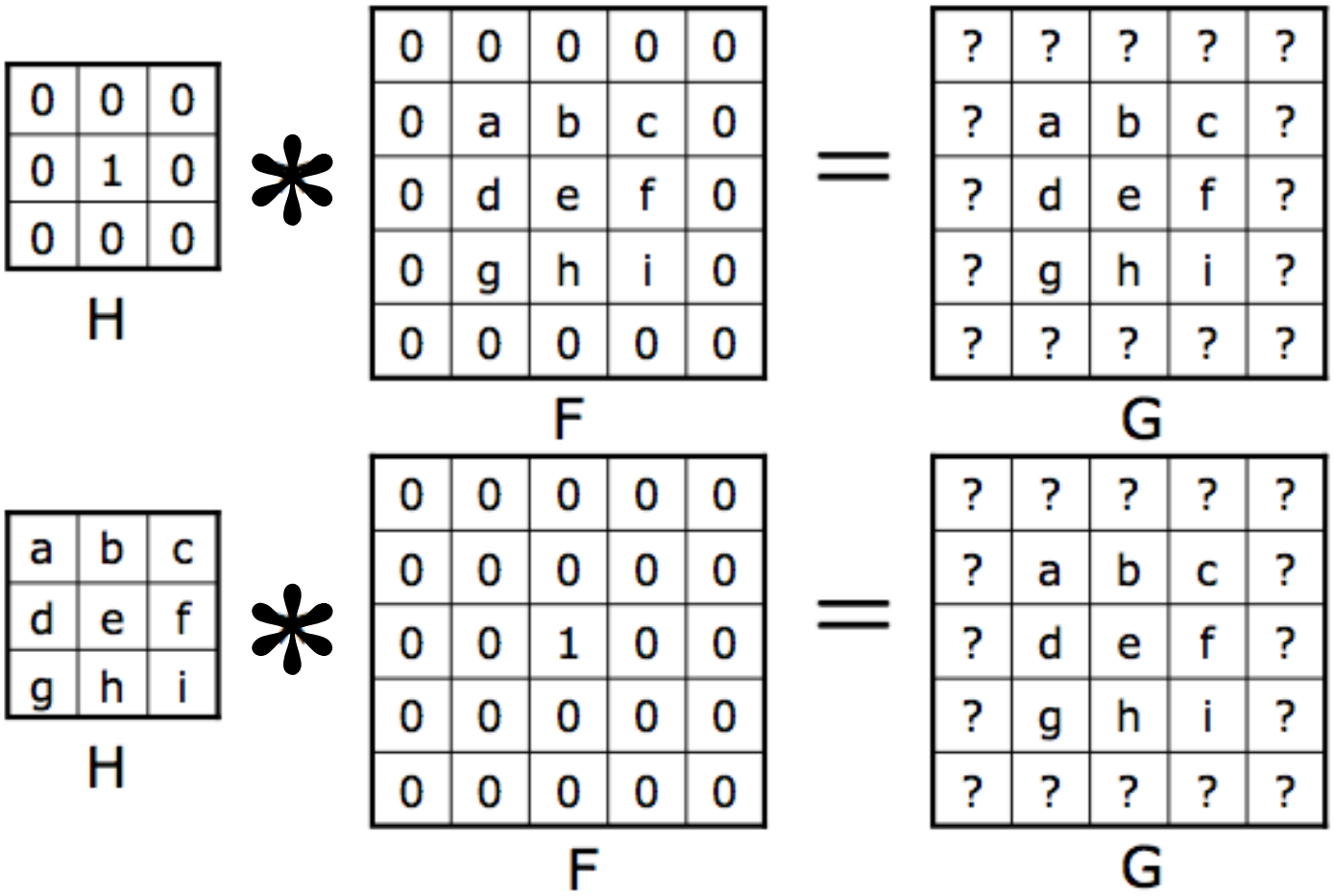
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

Convolution operation

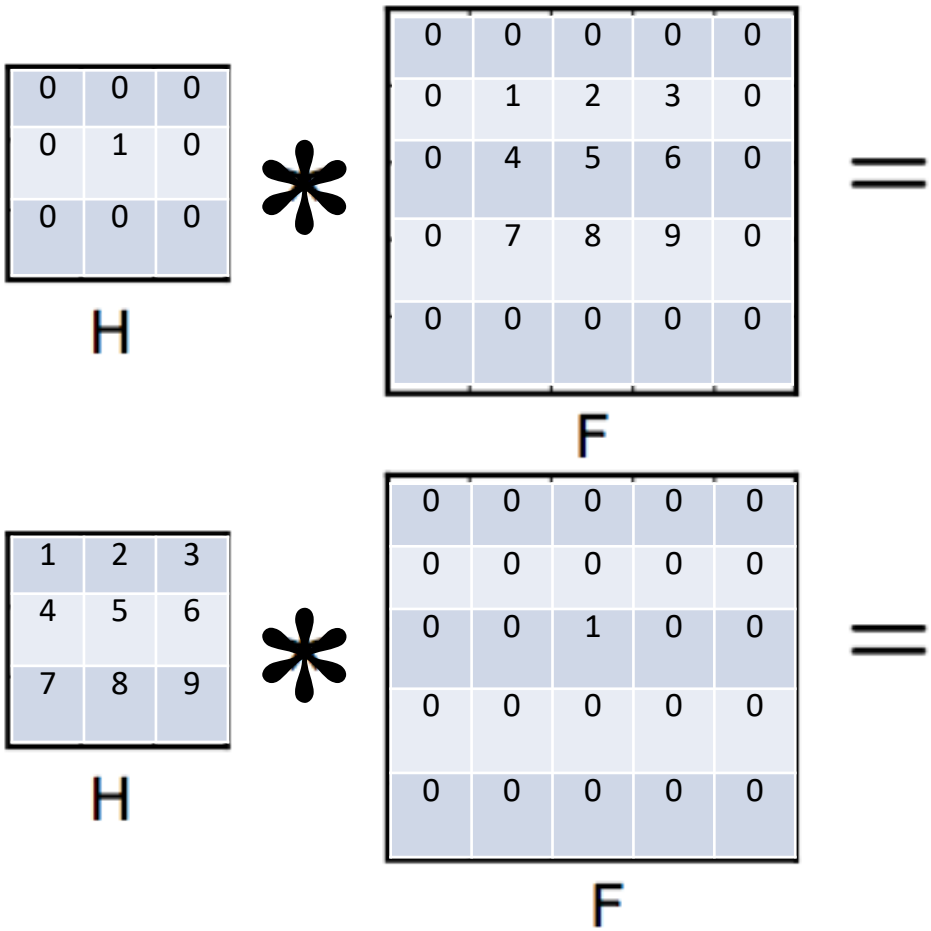
- Why convolution? Because, convolution operation preserves the following mathematical properties

Commutative: A⋆B=B⋆A
Associative: A⋆(B⋆C)=(A⋆B)⋆C
Distributive: A⋆(B+C)=(A⋆B)+(A⋆C)

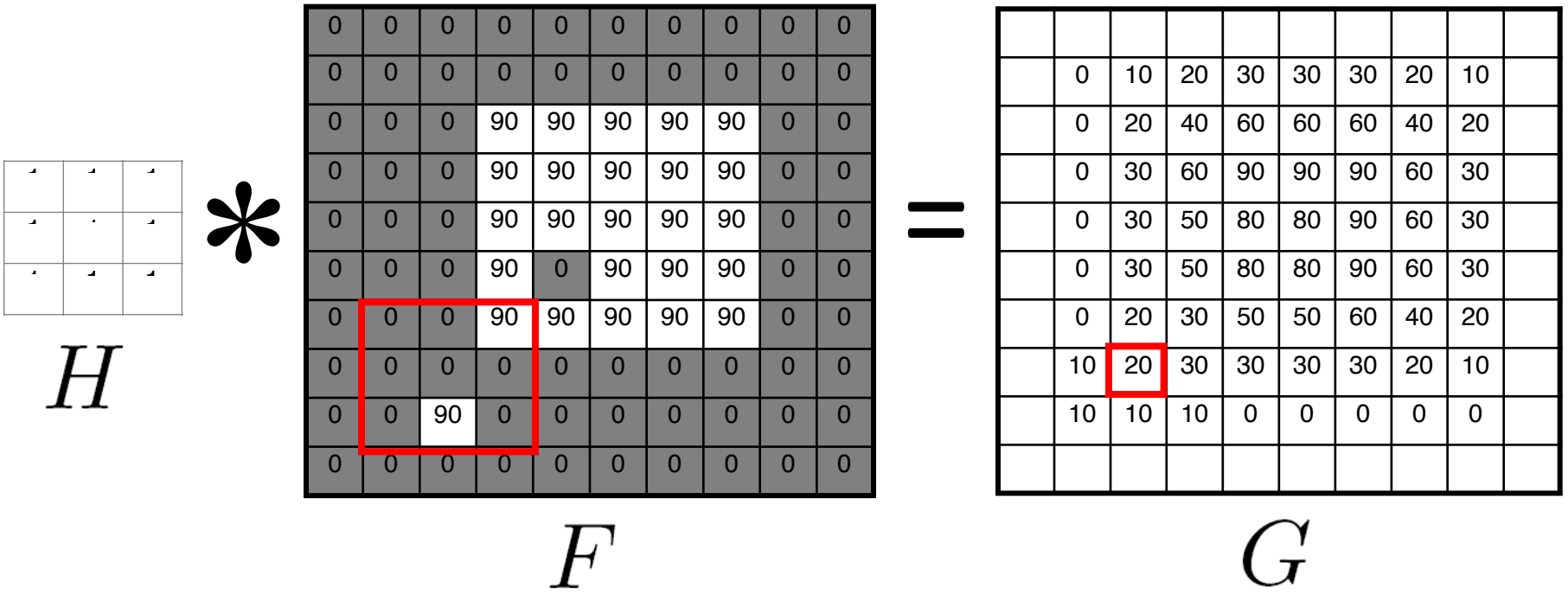
Example: Convolution operation is **Commutative**



Example: Convolution operation is **Commutative**



No need to write separate code for mean filtering



$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

$*$

0	0	0
0	1	0
0	0	0

$=$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$=$

$*$

0	0	0
1	0	0
0	0	0

$=$

Linear filters: examples

Original

*

0	0	0
0	1	0
0	0	0

=

Identical image

Flip horizontally first, then flip vertically during multiplication

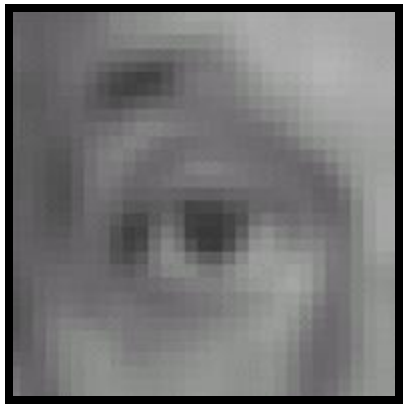
0	0	0
0	1	0
0	0	0

Source: D. Lowe

Linear filters: examples

Original

$$* \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$



Blur (with a mean filter)

Flip horizontally first, then flip vertically during multiplication

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

Source: D. Lowe

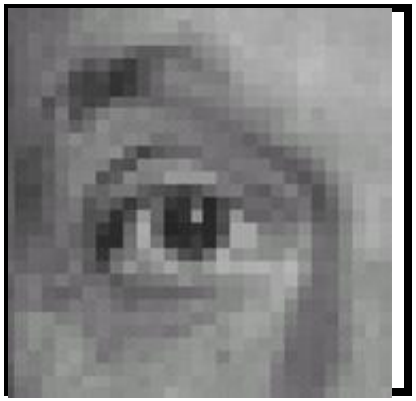
Linear filters: examples

Original

*

0	0	0
1	0	0
0	0	0

=



Shifted left
By 1 pixel

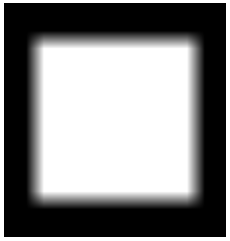
Flip horizontally first, then flip vertically during multiplication

0	0	0
0	0	1
0	0	0

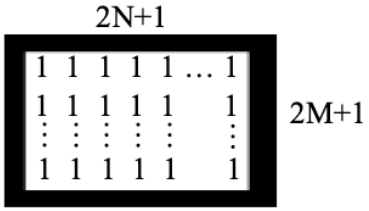
Source: D. Lowe

Box filter

- Weight contributions are equal within the box and zero around the edges



2D visualization

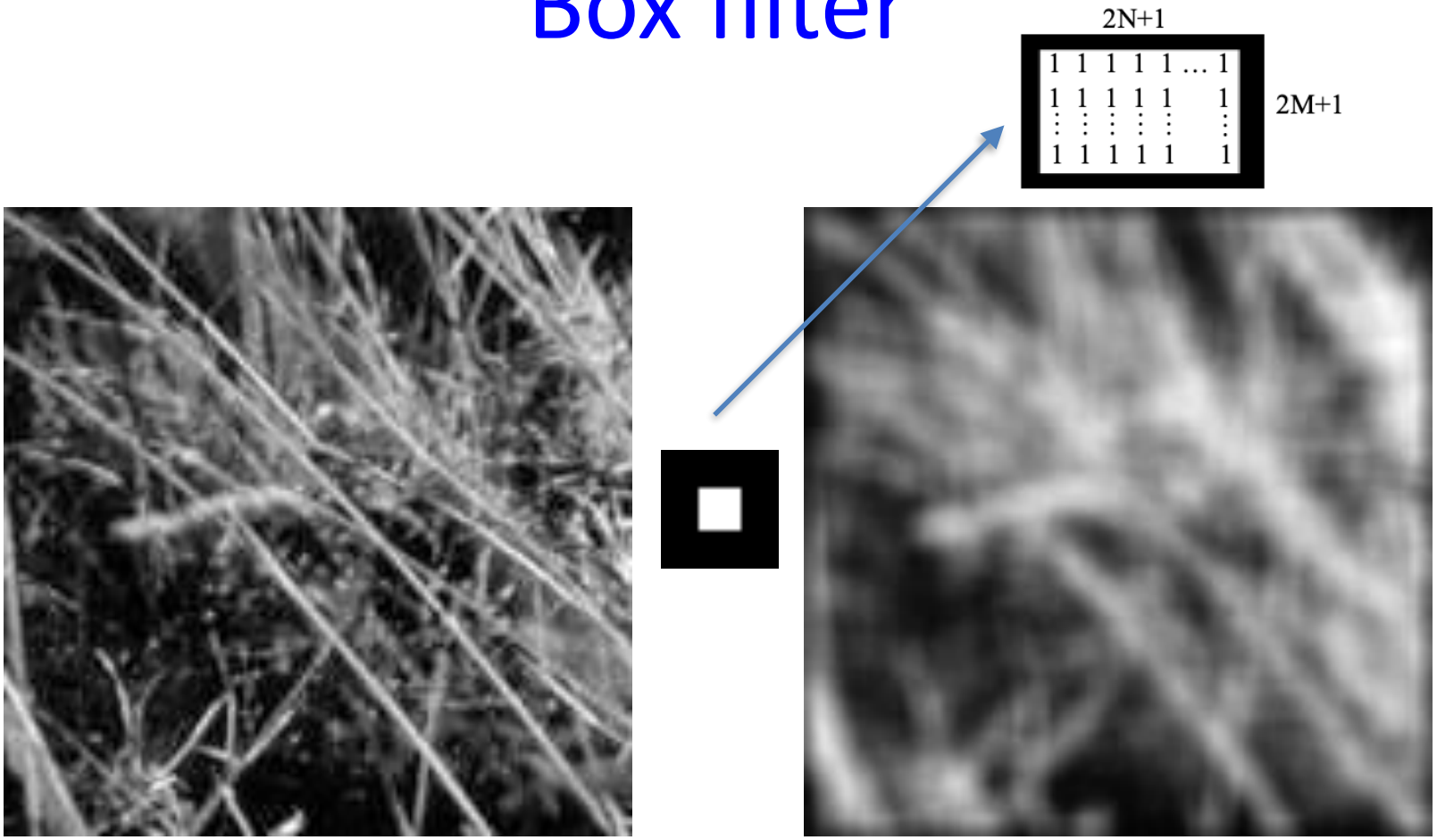


2D visualization

- Mathematical form:

$$h_{N,M}[n,m] = \begin{cases} 1 & : if \ -N \leq n \leq N, and \ -M \leq m \leq M \\ 0 & : otherwise \end{cases}$$

Box filter

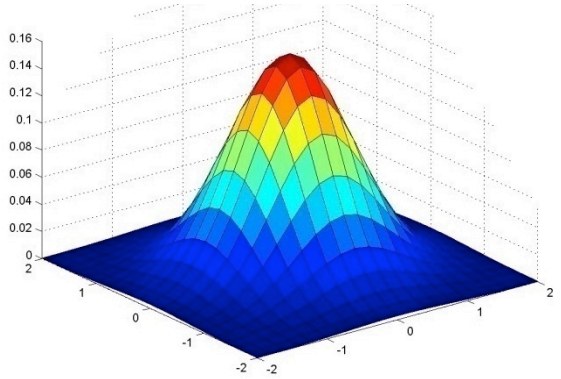


- Performing a convolution operation on the image using a box filter
- The contributions within the box are weighted equally, with no contribution from the surrounding edges.

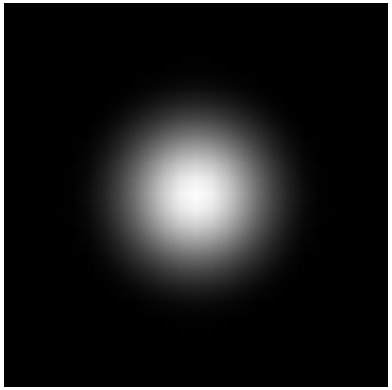
Source: D. Forsyth

Gaussian filter

- Weight contributions of neighboring pixels by proximity



3D visualization



2D visualization
(top-down view)

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Mathematical form

Gaussian filter

- Weight contributions of neighboring pixels by proximity

0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

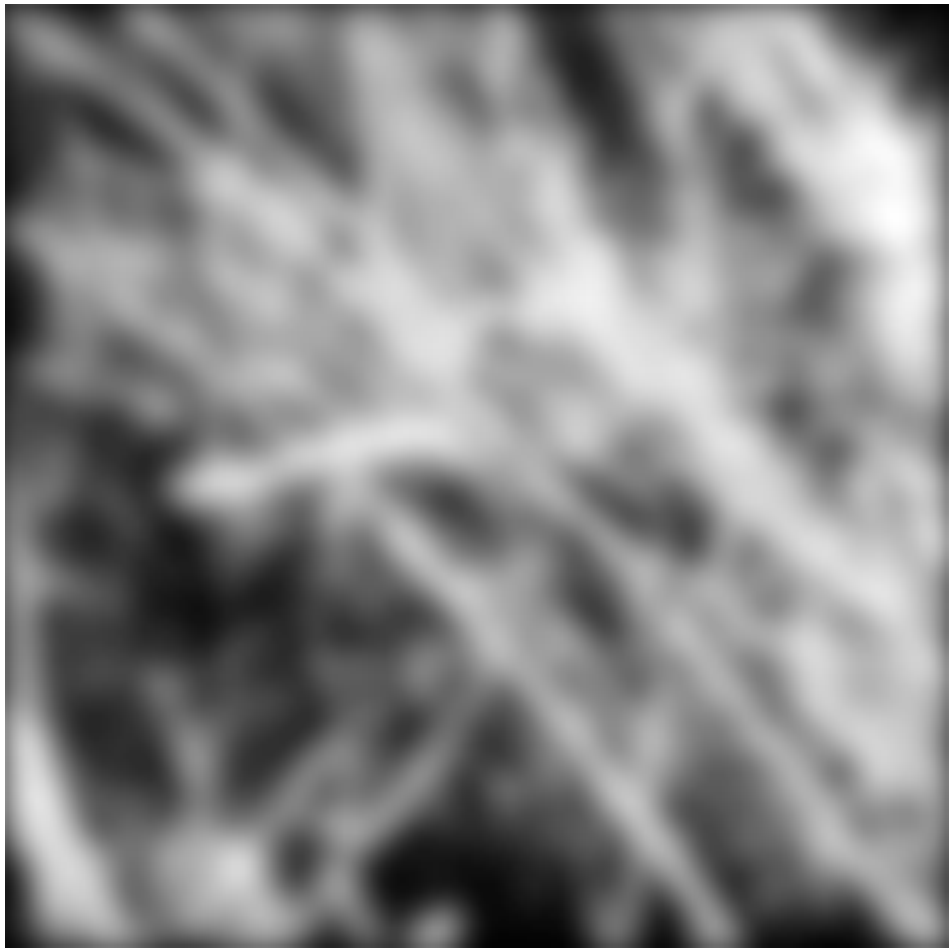
2D visualization for a 5x5 filter with $\sigma = 1$

(top-down view)

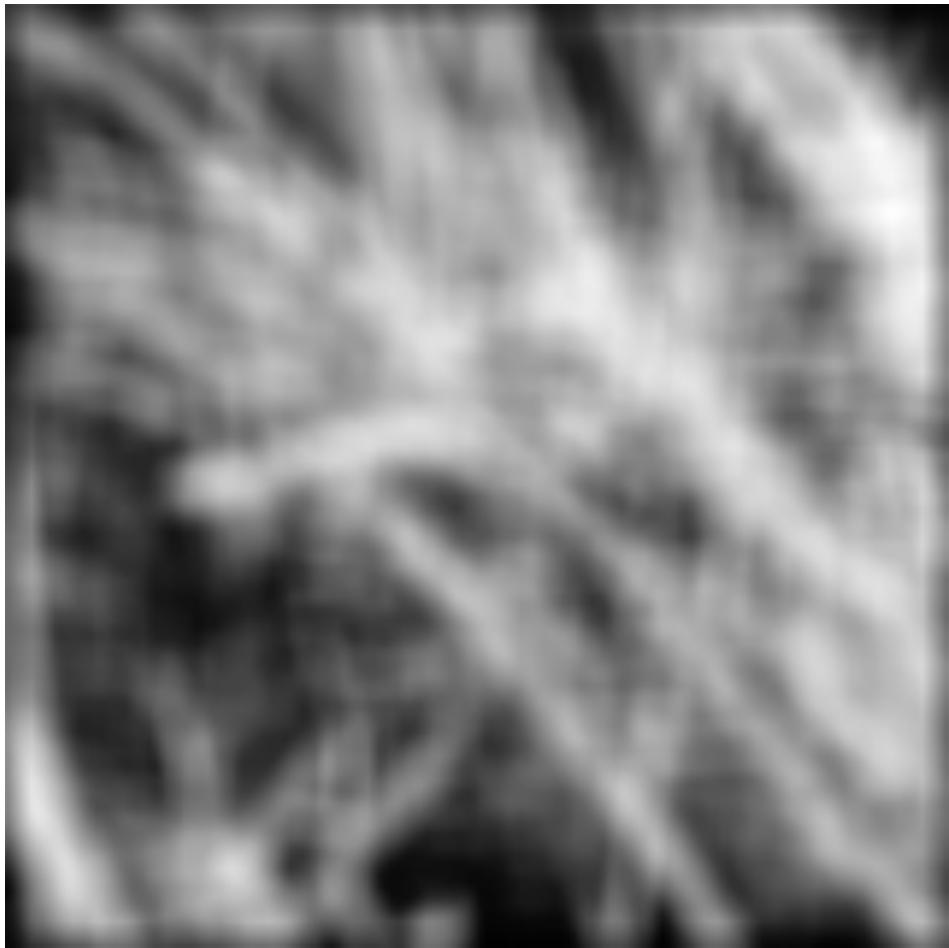
(-2,2)	(-1,2)	(0,2)	(1,2)	(2,2)
(-2,1)	(-1,1)	(0,1)	(1,1)	(2,1)
(-2,0)	(-1,0)	(0,0)	(1,0)	(2,0)
(-2,-1)	(-1,-1)	(0,-1)	(1,-1)	(2,-1)
(-2,-2)	(-1,-2)	(0,-2)	(1,-2)	(2,-2)

Location tuples (x,y) within the 5x5 filter

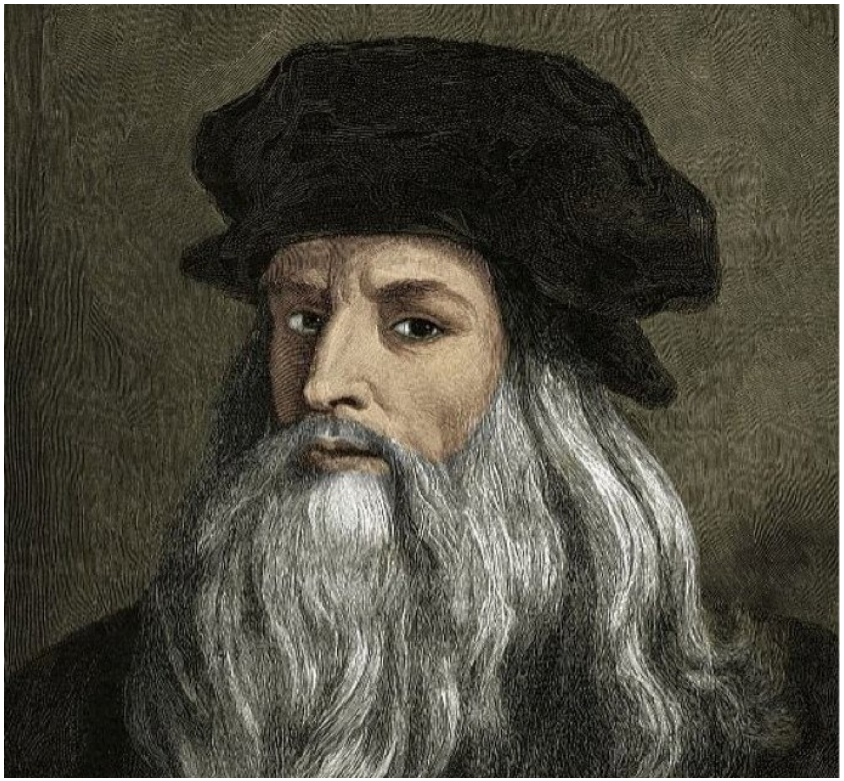
Smoothing with a Gaussian filter



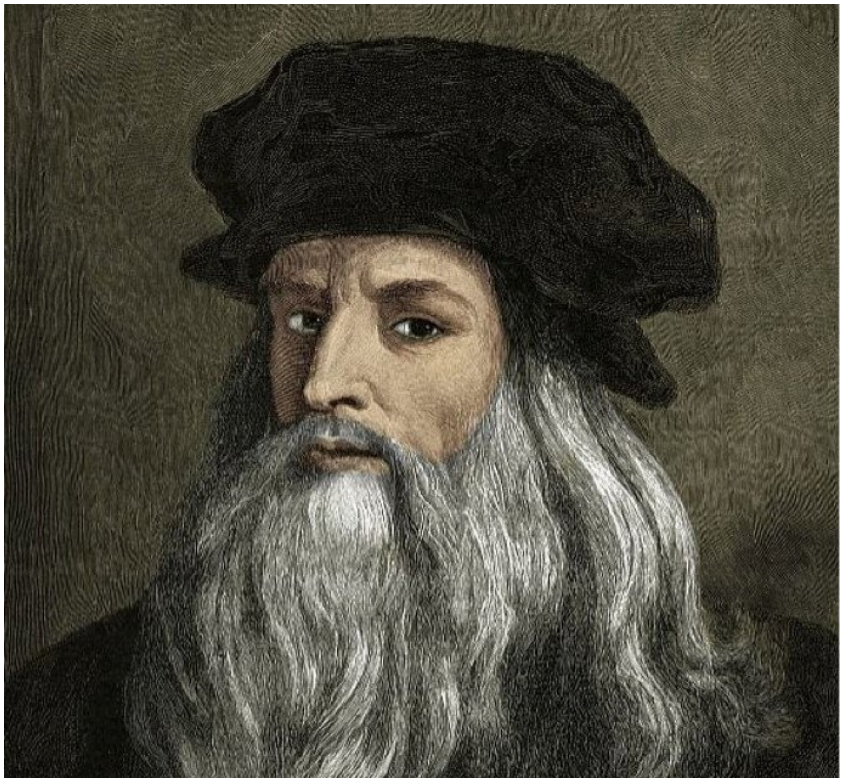
Smoothing with a Box filter

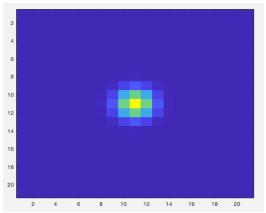


Coding activity: Gaussian Filter



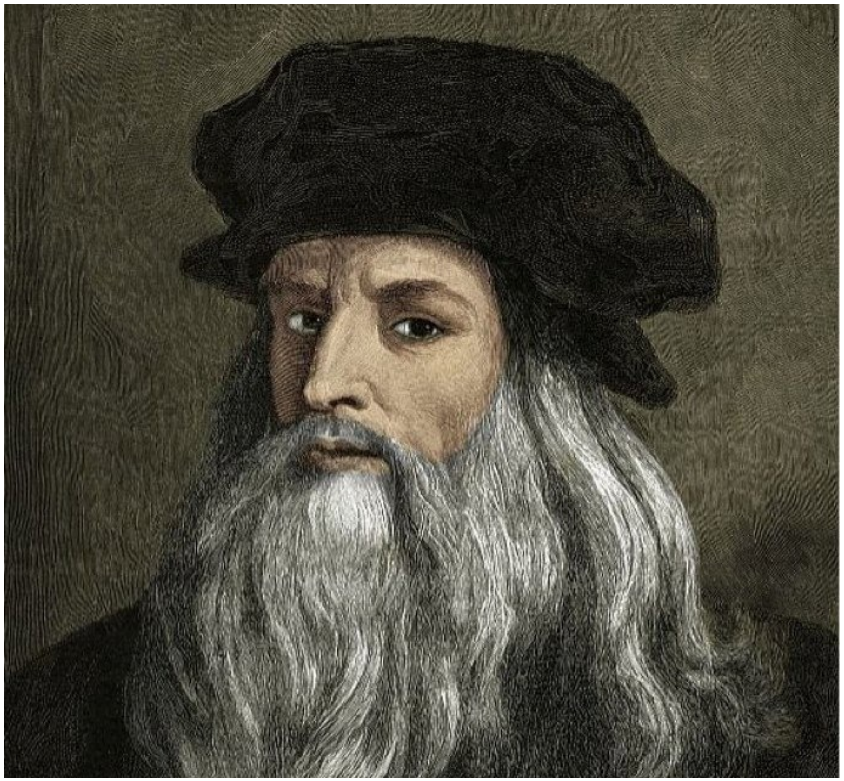
Coding activity: Gaussian Filter

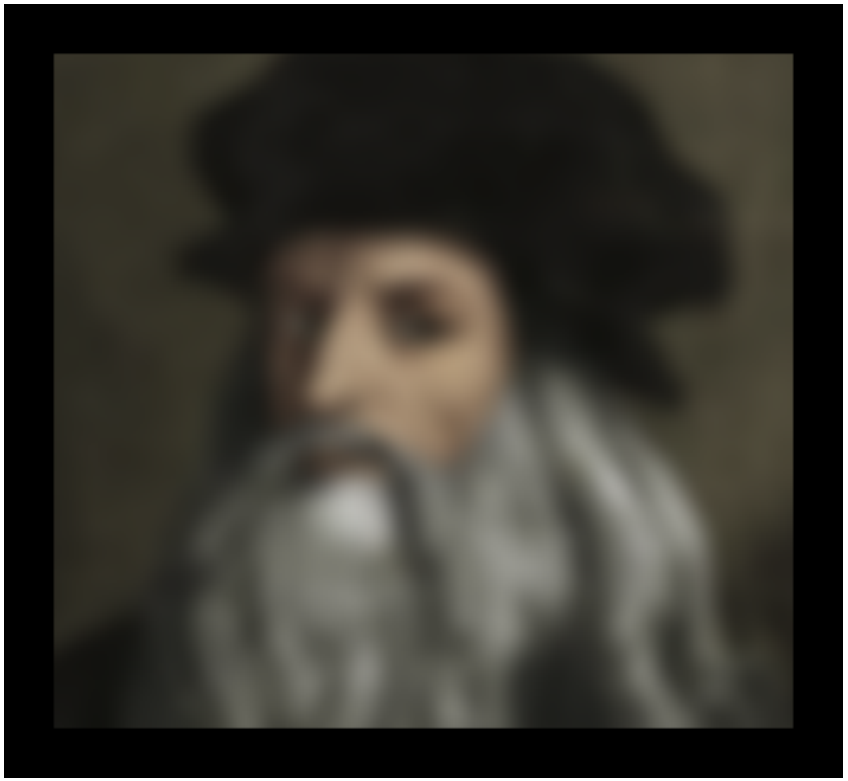


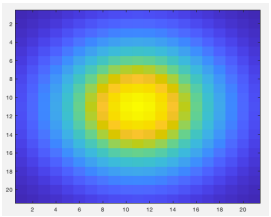


Gaussian kernel (21x21):
mean = 0 and σ = 1

Coding activity: Gaussian Filter

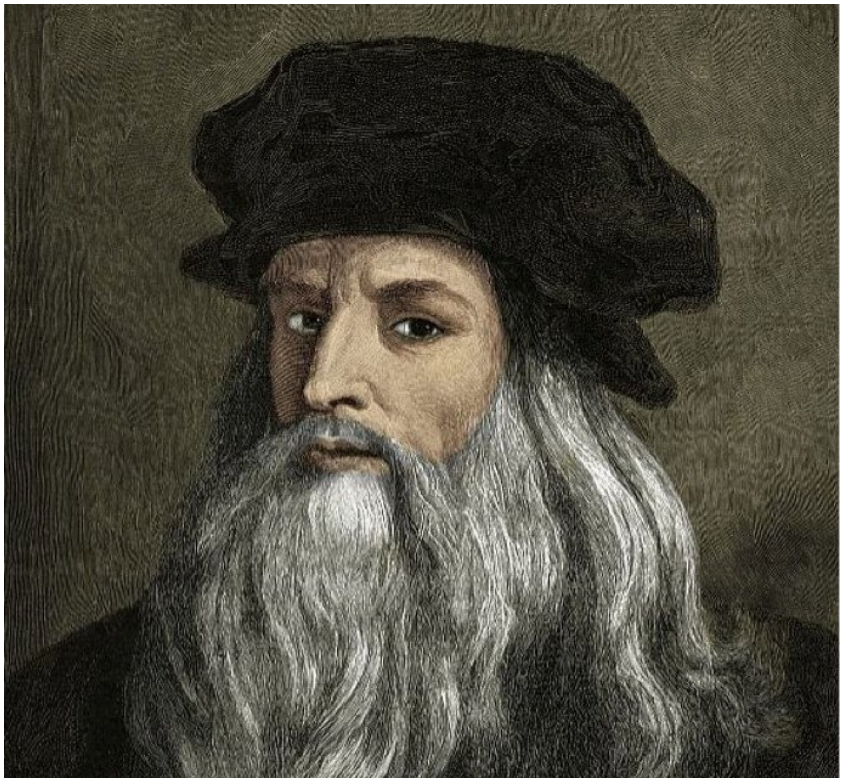


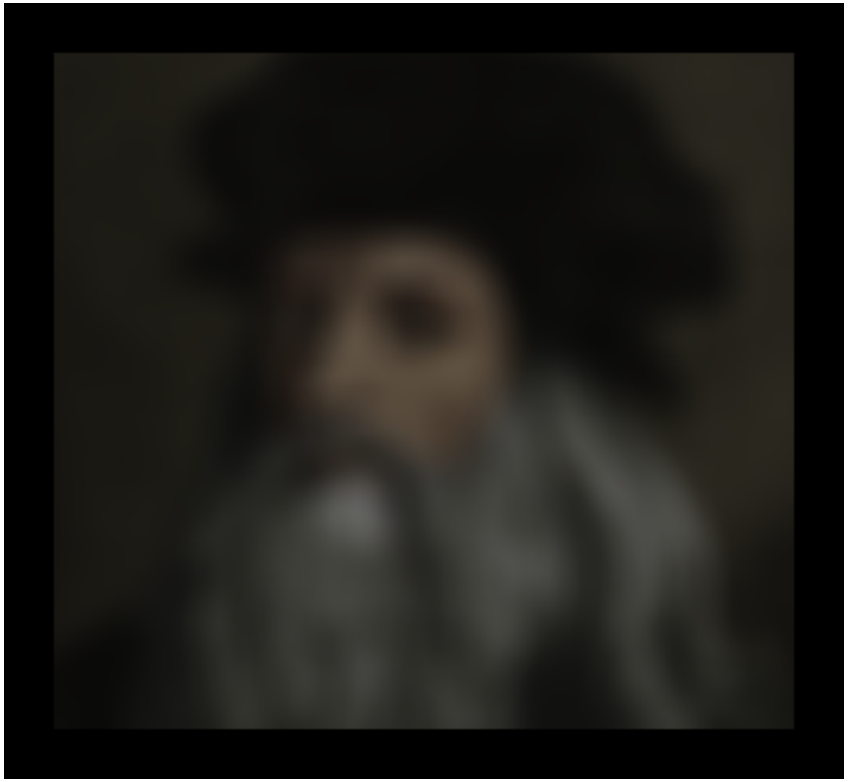




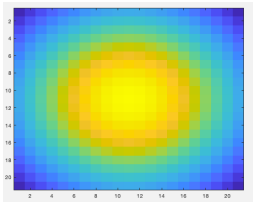
Gaussian kernel (21x21):
mean = 0 and σ = 5

Coding activity: Gaussian Filter

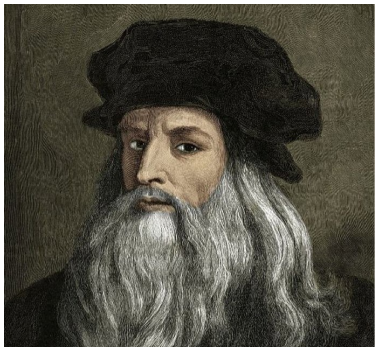


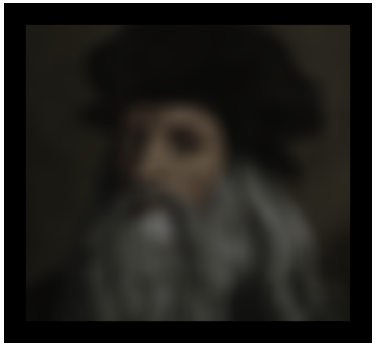


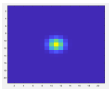
Gaussian kernel (21x21):
mean = 0 and σ = 10



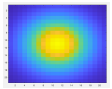
Coding activity: Gaussian Filter



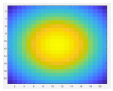




mean = 0 and σ = 1



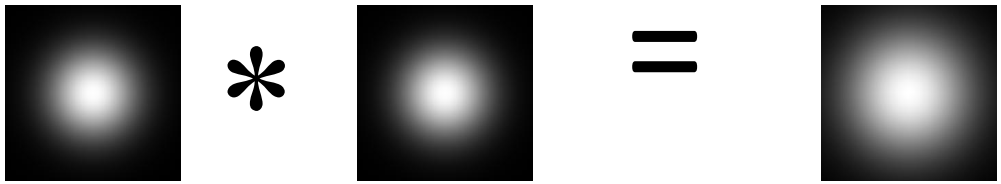
mean = 0 and σ = 5



mean = 0 and σ = 10

Gaussian filter

- Removes “high-frequency” components from the image (low-pass filter)
- Larger σ blurs more
- Convolution with self is another Gaussian



Source: K. Grauman@UT Austin

Coding activity: convolving image with Gaussian filters

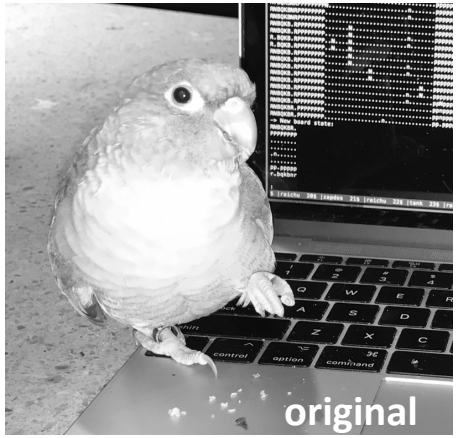
A taxonomy of useful filters

- Blur
  - Box filter
  - Gaussian
  - Bilateral exponential
  - Asymmetric filter: motion blur
- Edges
  - [-1, 1]
  - Derivative filter
  - Derivative of a Gaussian
  - Oriented filters
  - Gabor filter
  - Quadrature filters: phase and amplitude
  - Elongated edges: filling gaps

Source: Antonio Torralba@MIT

Other filtering operations

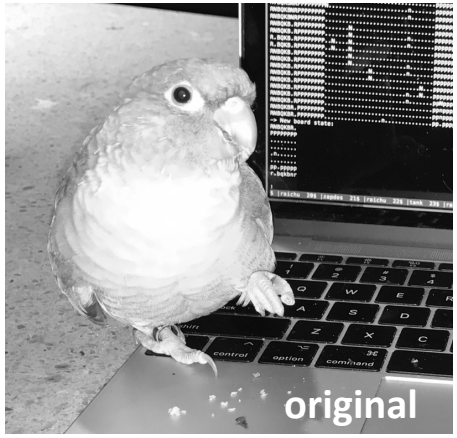
- What does blurring take away?



−

=

Let's add it back:

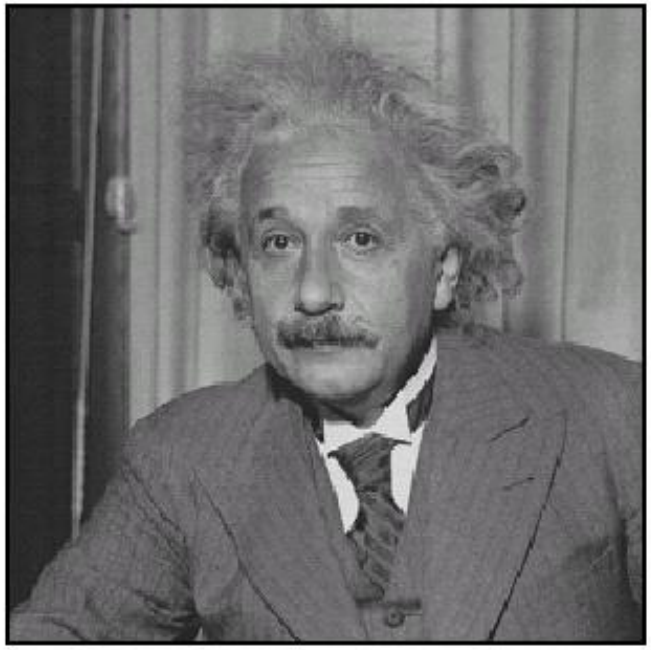


+ α

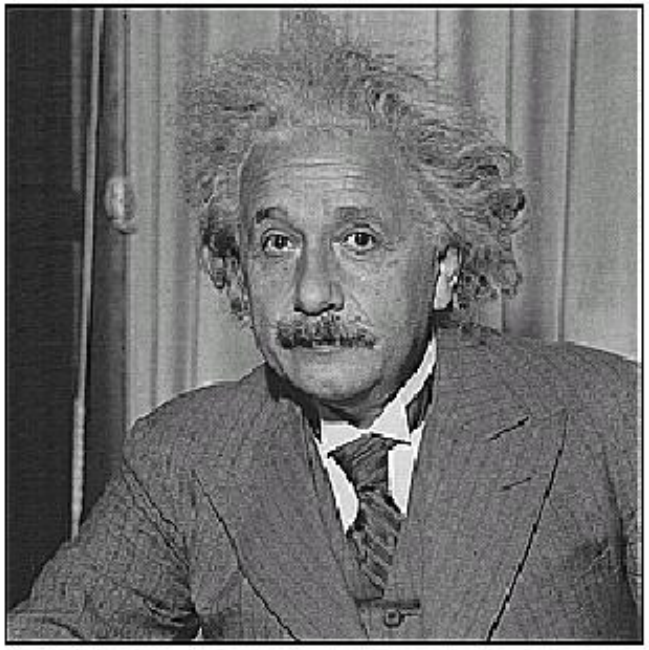
=

Sharpen filter

Sharpen filter



before



after

Source: D. Lowe

Assignment#1 Discussion

Assignment 1: Whitening Transformation

- Task 1: Whitening Transformation
- Task 2: Histogram Equalization

Assignment 1: Whitening Transformation

Task 1: Whitening

You will be adjusting the contrast of the image. Your goal is to transform the image so that the resulting image has a zero mean and unit variance. Denote the image as $I(.)$ which is a 2D array of pixel values. Its width and height are N and M pixels respectively. Also $I(x, y)$ denotes the pixel value at 2D location (x, y). You can compute the mean and variance of the gray-scale image $I(.)$ as follows:

$$\mu = \frac{\sum_{x=1}^{N} \sum_{y=1}^{M} I(x, y)}{N * M} \tag{1}$$

$$\sigma^2 = \frac{\sum_{x=1}^{N} \sum_{y=1}^{M} (I(x, y) - \mu)^2}{N * M} \tag{2}$$

Assignment 1: Whitening Transformation

Now, you can transform each pixel value separately using the above two computed statistics μ and σ as follows:

$$I^{'}(x,y) = \frac{I(x,y) - \mu}{\sigma} \tag{3}$$

Apply this *Whitening Transformation* on the provided input image to see how it affects. The following figures show the effect of applying *Whitening Transformation* on the first-ever photograph – "**View from Window at Le Gras**".

Input

'Whitening' Transformed

Lab 1: Image Transformation

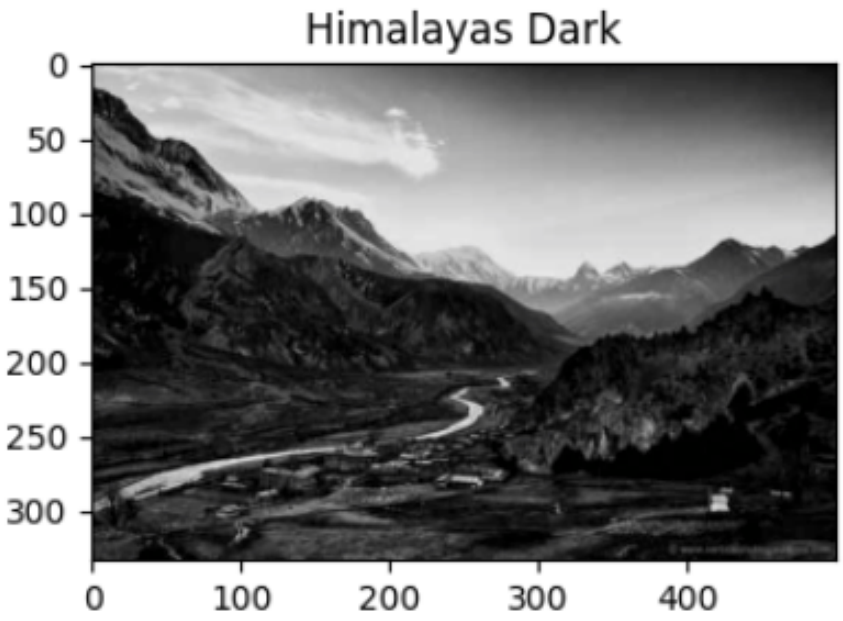
- Task 1: Whitening Transformation

- Task 2: Histogram Equalization

  Before you start applying the per-pixel transformation, it would be a good practice to visualize the histogram of the image first. Write a few lines of python code to visualize the histogram first (I will share the code snippets.)

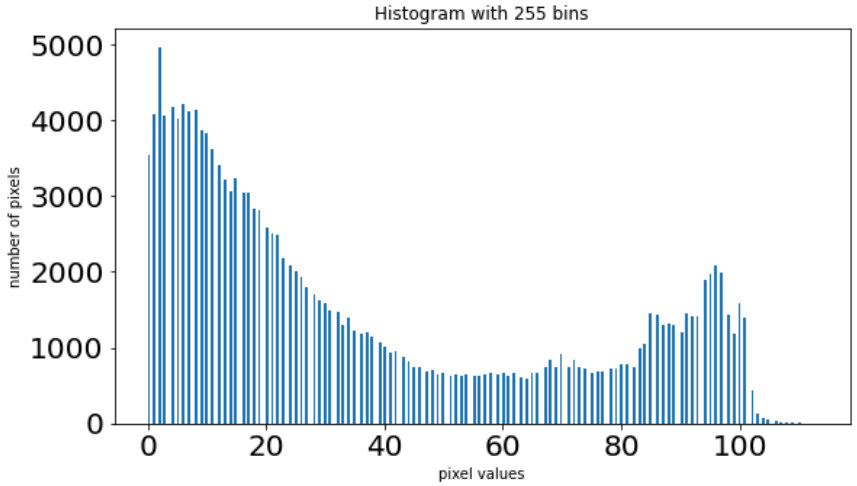
Histogram Visualization

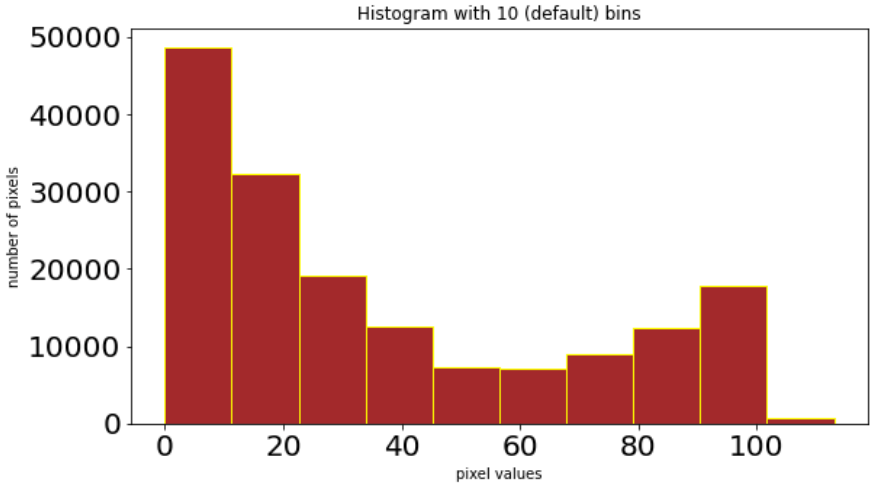
```
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
img_pil = Image.open('/content/drive/MyDrive/cs195_fall24/low_level_vision/assignment1/input_images/himalaya_dark.jpg')
img_pil_array = np.asarray(img_pil)

plt.figure(figsize=(4,4)) # figure size (4 inch, 4 inch)
plt.imshow(img_pil_array, cmap='gray')
plt.title('Himalayas Dark')
img_pil_array.shape
```



Histogram Visualization (v1)

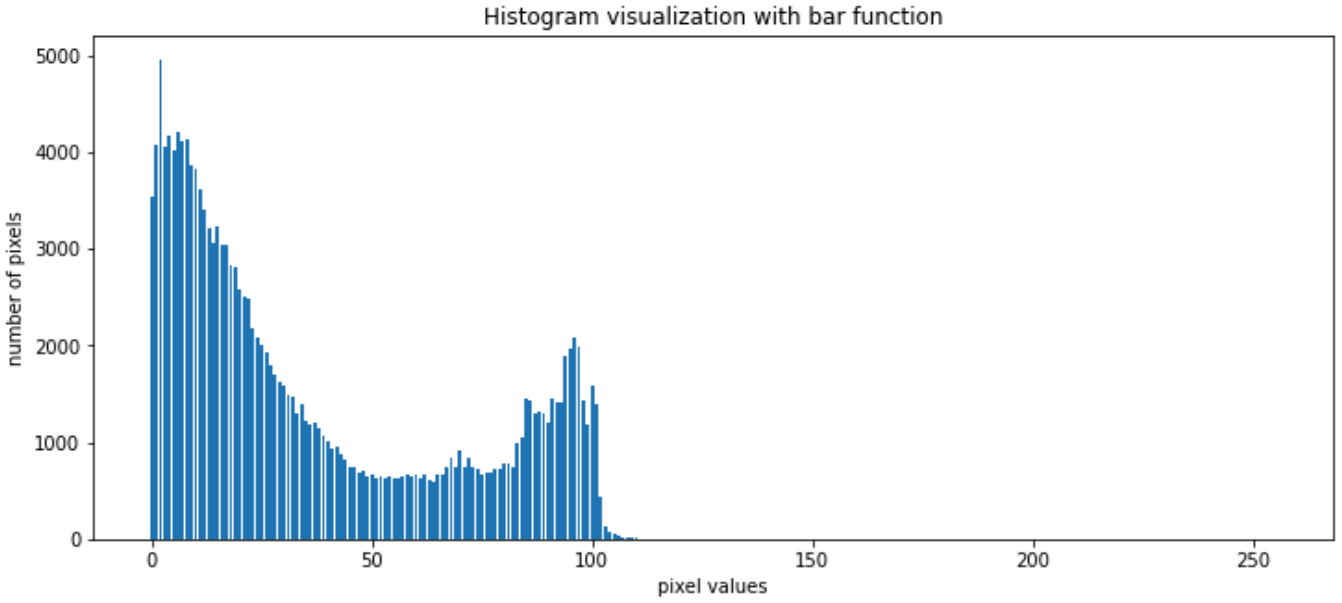
```
img_pixel_vector = img_pil_array.flatten() # convert the matrix into a vector
fig, axes = plt.subplots(1, 2, figsize=(20, 5))
# dense histogram
ax = axes[0]
ax.hist(img_pixel_vector, bins=255)
ax.set_xlabel('pixel values')
ax.set_ylabel('number of pixels')
ax.set_title('Histogram with 255 bins')
ax.tick_params('both', labelsize=20)

# coarse histogram
ax = axes[1]
ax.hist(img_pixel_vector, edgecolor="yellow", color="brown") # default bins=10
ax.set_xlabel('pixel values')
ax.set_ylabel('number of pixels')
ax.set_title('Histogram with 10 (default) bins')
ax.tick_params('both', labelsize=20)
plt.show()
```





Histogram Visualization (v2)

```
# alternate way of visualizing the histogram where we have more control
# histogram() function from Pillow's Image module
# NOTICE: this function returns the histogram, hence we have access to the bins
#         we can compute normalized histogram


# Step 1: generate the histogram
img_hist = img_pil.histogram() # calling Image.histogram() function from PIL

img_hist_array = np.array(img_hist)

bins = np.arange(len(img_hist_array))
fig, axes = plt.subplots(1, 1, figsize=(12, 5))
axes.bar(bins, img_hist_array)
axes.set_title('Histogram visualization with bar function') # explicit bin value
axes.set_xlabel('pixel values')
axes.set_ylabel('number of pixels')
```



Assignment 1: Histogram Equalization

Task 2: Histogram Equalization

Let's try another type of contrast transformation called *Histogram Equalization* on the input image. You may need to do it in multiple steps.

Step 1: you need to generate the histogram of intensity values. The simplest way to find the histogram of intensity values is to make use of the **`Pillow Library`** functionalities (HINT: There is a function called 'histogram()' in Pillow). Alternatively, if you want, you could write few lines of python code and build your own histogram of intensity values. There will be at most 255 intensity values. Let's denote $hist_b$ is 1D vector denoting the histogram of intensity values. b denotes a particular bin, and its value ranges from 0 to 255. Now, count how many pixels in image $I(.)$ have the intensity value equal to b. Put that total count into the respective bin location, i.e., $hist_b$. You can do it for all the intensity values one after another, starting from $b = 0$ up until $b = 255$.

Step 2: Now you need to normalize the histogram $hist$ such that the sum of this new histogram is equal to 255. Denote this new histogram as $hist^{'}$. For each bin b, you can compute it as follows:

$$hist_b^{'} = \frac{hist_b}{N * M} \tag{4}$$

$$hist_b^{'} = hist_b^{'} * 255 \tag{5}$$

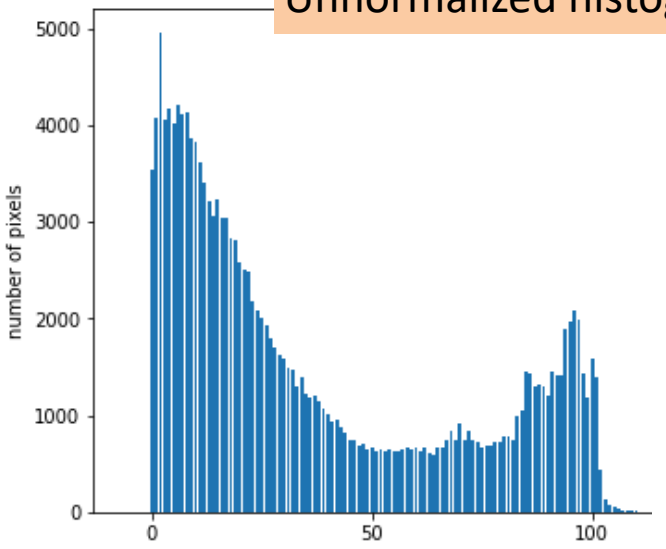
Assignment 1: Histogram Equalization

Step 2: Now you need to normalize the histogram $hist$ such that the sum of this new histogram is equal to 255. Denote this new histogram as $hist^{'}$. For each bin b, you can compute it as follows:

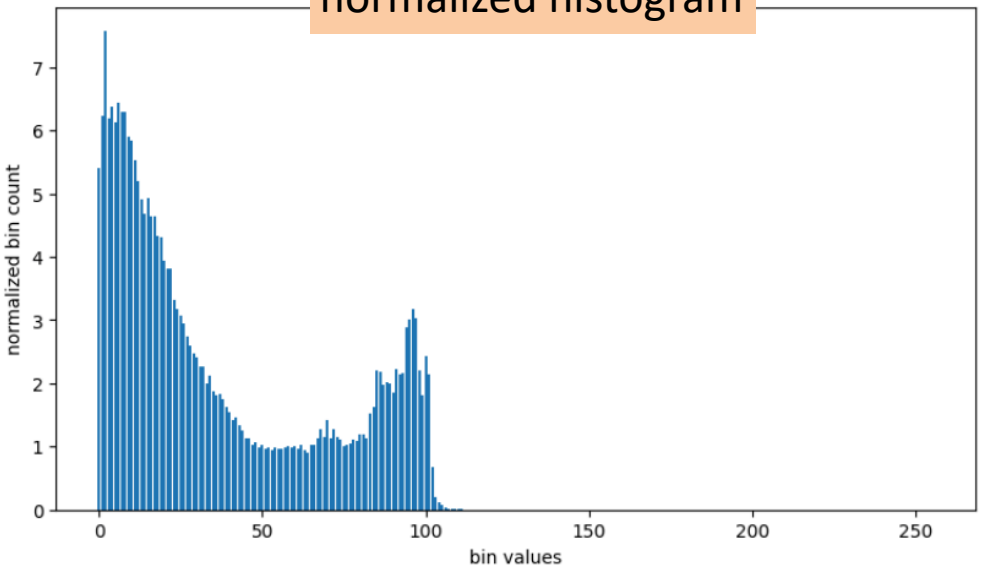
$$hist_b^{'} = \frac{hist_b}{N * M} \tag{4}$$

$$hist_b^{'} = hist_b^{'} * 255 \tag{5}$$

Unnormalized histogram



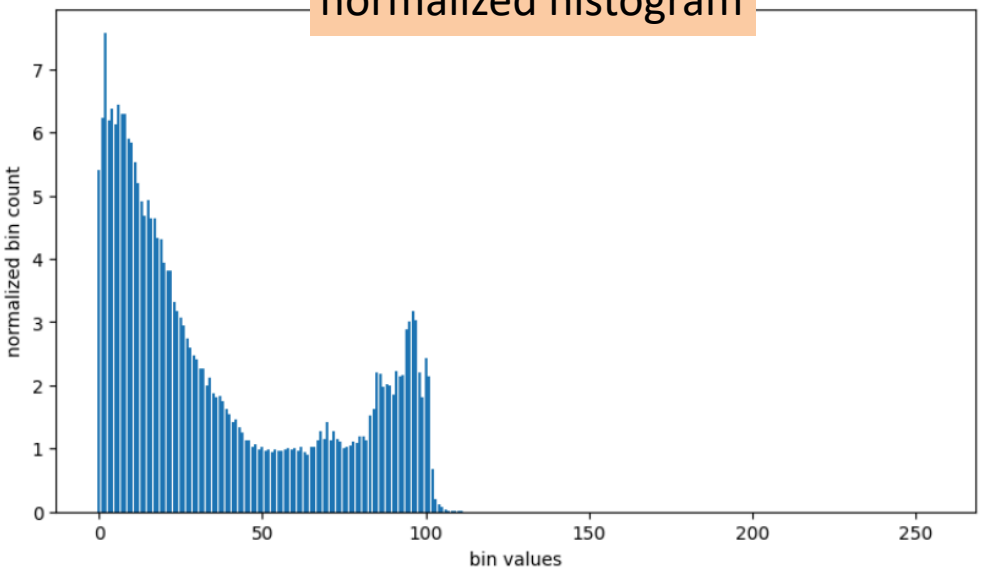
normalized histogram



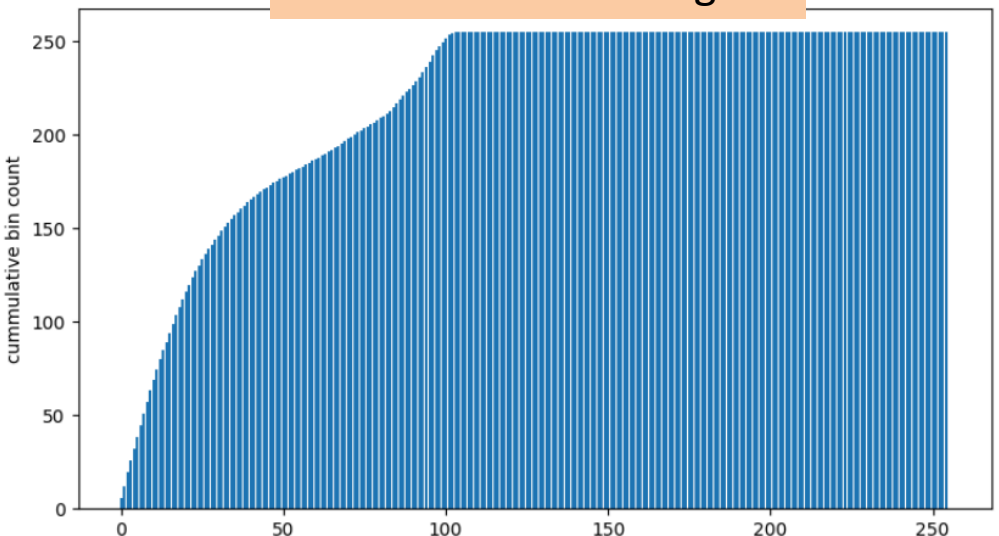
Assignment 1: Histogram Equalization

Step 3: Compute the cumulative sum of the $hist'$. Denote this histogram as $histcum$.

normalized histogram



cumulative sum histogram



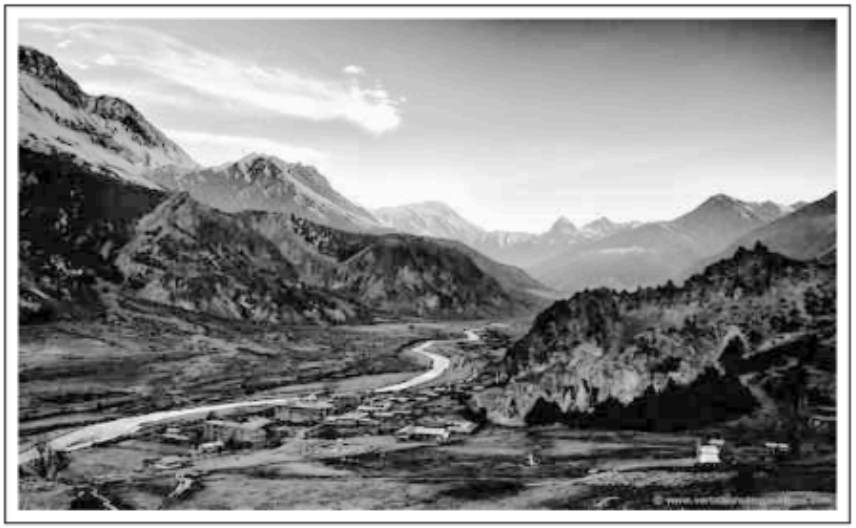
Assignment 1: Histogram Equalization

Step 4: Use the cumulative histogram *histcum* as a lookup table to transform the value of each pixel location as follows:

$$I^{'}(x,y) = histcum(I(x,y)) \tag{6}$$

where $I^{'}(x,y)$ denotes the histogram equalized image value at pixel location (x,y). Apply your newly implemented *Histogram Equalization* on the provided input image to see how it affects. The following figures show the result on another input image called "**Himalaya**".

Input



Histogram Equalized