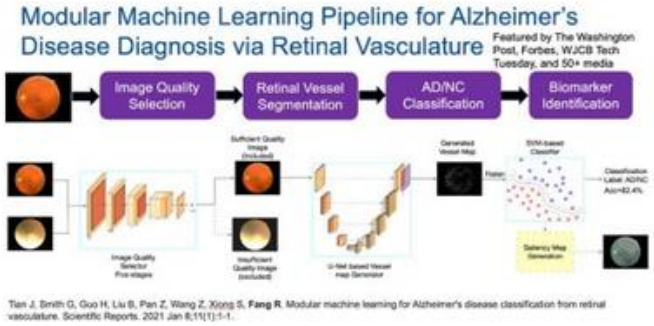
Image Filtering

Dr. Sander Ali Khowaja,
Assistant Professor, Department of Telecommunication Engineering
Faculty of Engineering and Technology, University of Sindh, Pakistan
Senior Member, IEEE – Member, ACM
https://sander-ali.github.io

Computer Vision in the News

robust, reliable, and trustworthy AI model that can segment the T1 MRI volume in only three seconds.

"A challenge we face is that our patient population is over 50 years old, whereas public segmentation tools have been built on younger adults around 20 to 30 years old," Skylar tells us. *"These public tools don't account for the natural processes that occur with aging, such as white matter and gray matter degeneration in older adults. We want a tool that can work better for older adults and better for individualized cases since the human head is highly variable."*

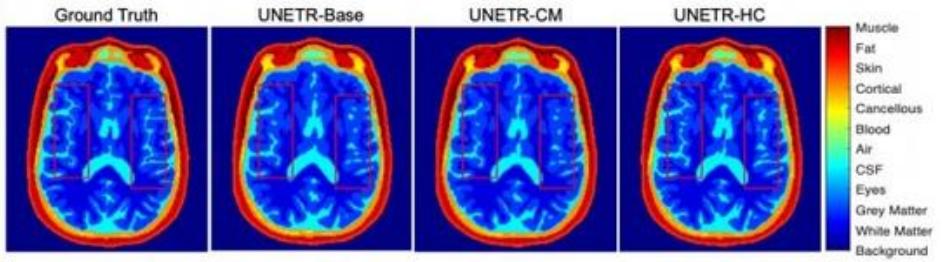


Tian J, Smith G, Guo H, Liu B, Pan Z, Wang Z, Xiong S, **Fang R**. Modular machine learning for Alzheimer's disease classification from retinal vasculature. Scientific Reports. 2021 Jan 8;11(1):1-1.

Skylar points out that the segmentation output of many of the tools available is part of an overall pipeline that can take 11 hours or more to run. These tools are not particularly user-friendly, and someone unfamiliar with them may be unable to separate the specific segmentation approach from the rest and would have to wait the total time for each subject to get the output they need. **Three seconds is clearly a massive improvement on that.**

"The segmentation pipeline is used to get the electrical flow model, which doesn't sound like something that's directly translatable to practice, but it's ***a crucial part of non-invasive brain stimulation****,"* Skylar explains. *"The goal is to have more accurate parameters for non-invasive brain stimulation, so we can use this to treat people with conditions like Alzheimer's. Not all heads are the same, so you want to give someone the best parameters possible to get the treatment that will most help them."*

One of the biggest challenges in head segmentation research matches one of the biggest challenges in deep learning: **a model often struggles with data that is different from the data on which it was trained.** This study uses data from two locations, and where it would train on data from one location, it was challenging to translate the performance to the data from the other location.



"It's fundamental in getting deep learning in medical practice that you have to be ***more translatable to different locations and patient populations****,"* Skylar asserts. *"That is where the idea of the* ***calibration approach*** *comes in. Calibration allows you to get an output in addition to your classification that measures your confidence in your prediction. When we have this additional output that tells us how likely we are to be right or wrong, we can use it to take an output where we think that our model isn't very certain and have manual segmenters help us improve the labeling, feeding that back into our model to train it to handle these harder cases better."*

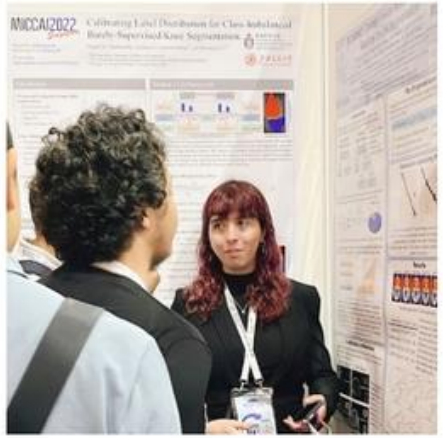
The lab has been using **MONAI (Medical Open Network for AI)** software, which should be familiar to our readers, and was developed from a project originally started by **NVIDIA** and **King's College London.**

"MONAI has been so helpful in my research because it comes with many frameworks already implemented in MONAI," Skylar tells us. *"You can take the U-Net from MONAI and use that in your code instead of a random PyTorch repository. It allows you to* ***take these different frameworks, easily change the parameters, and get your code running without too much effort****.*

A big struggle for coders is repositories that aren't documented well. MONAI is really well documented, and it has all of these neural networks to work with and pre- and post-processing functions that work well on a dictionary of all of your inputs and can handle any problem you have. The libraries available under this framework are also ***GPU-accelerated frameworks****, so we have this faster and more efficient software that allows us to* ***focus on solving future problems*** *rather than struggling with basic challenges."*

When CVN spoke to the guys at NVIDIA about MONAI, they promised something almost revolutionary, and it seems they kept their word! Skylar says that because **MONAI has been so useful for this work,** the lab wants to give back by contributing to progressing the tools available for other researchers worldwide.

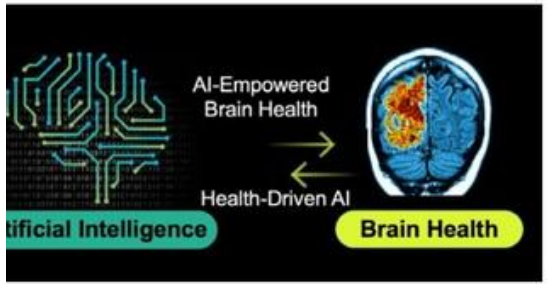
"We presented our loss term in our MICCAI oral talk this year, and we've been talking about the potential for implementing it in MONAI," she reveals. *"This would help researchers because although there are many efficient losses in MONAI, none are calibrated losses. Calibration is a field of deep learning that's not talked about enough."*



(tDCS) to improve their cognitive function and possibly other aspects, such as anxiety and depression, which are risk factors for Alzheimer's disease and related dementia.

Building that pipeline involves **creating a personalized head model,** which requires 20 hours of manual segmentation to segment the whole T1 MRI into 11 tissue types with different conductivity for the electricity to model the current flow in the brain for a specific subject. **Everyone, especially older adults, has distinct brain atrophy, adipose distribution, and skull thickness.**

Skylar's work is an essential part of this, using **deep learning to build a rapid, accurate,**



Some AI related news

Computer Vision News has found great new stories, written somewhere else by somebody else. We share them with you, adding a short comment. **Enjoy!**

Shutterstock Rolls out a Generative AI Toolkit to Create Images Based on Text Prompts

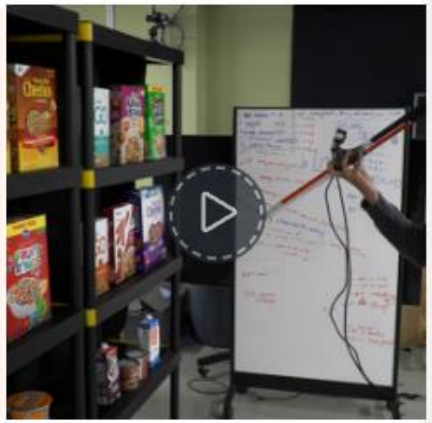
It's going to be interesting in the world of stock images! **Shutterstock** did not lose any time to partner with **OpenAI** to help develop OpenAI's **Dall-E 2** artificial intelligence image-generating platform. And now they bring their own generative AI tools to their users: customers of Shutterstock's Creative Flow online design platform are now able to create images based on text prompts, powered by OpenAI and Dall-E 2. On the other hand, Getty Images seem to stake their bets in the opposite direction: they play it safe, banning all AI-generated content over fears of future copyright claims and legal risks. Read More

How Artificial Intelligence Is Helping us Decode Animal Languages

Curious? You should be. AI and machine learning have been used over years to **analyze and translate human languages** to very impressive state of the art levels, enabling new communications avenues between humans and machines. Ask **Katie Zacarian**, the CEO and co-founder of **Earth Species Project (ESP)**, a non-profit organization that uses AI **to decode animal communication**. The idea is to apply past breakthroughs of AI for human language to animal communication. This is not less impressive! It involves large data sets that contain **visual, oral and physical communications between animals**. Read More

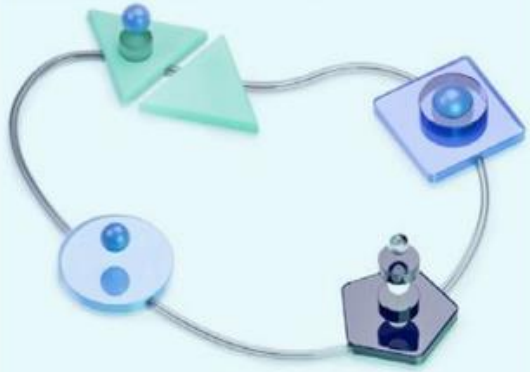
Show Me the Way, AI Stick!

It is not the first time that Spotlight News tells you about AI providing helpful solution to **visually impaired persons:** daily life presents them a variety of challenges that those with normal vision may not be aware of and we are happy to witness great advances in supporting the population affected by visual impairments. Mobility is not the least of these challenges and AI is always one step closer to solving it. This time, a team of researchers from Colorado developed a new kind of **walking stick, that maps and catalogs the world around it** thanks to a camera and computer vision technology. **Watch the Video**



A Robot Was Scheduled to Argue in Court, Then Came the Jail Threats

AI does no always win and not all the stories have a happy end. A British gentleman planned to have a "**robot lawyer**" help a defendant fight a traffic ticket in court. The person challenging a speeding ticket would **wear smart glasses that both record court proceedings and dictate responses** into the defendant's ear from a small speaker. The system relied on the same AI text generators everyone's talking about. To cut a long story short, his idea met a strong opposition from the side of corporations feeling threatened by it. Some of them answered with threats of prosecution and prison time. That's how the story ends this time... Read More



Open-Source Active Learning Toolkit for Computer Vision

What machine learning engineer and data scientist doesn't want to **understand and improve their training data quality and help boost model performance**? That's the promise of an AI-assisted platform provider called **Encord**. They have released a free open-source industry agnostic all-in-one toolkit designed for that purpose. One of the motivations disclosed by the company is that self-driving cars and diagnostic medical models using AI suffer from a "**production gap**": proof-of-concept models perform well in research environments but struggle to make predictions accurately and consistently in real-world scenarios. Read More

MusicLM: Generating Music From Text

Let's end this month's AI Spotlight News with a musical note: why couldn't we use AI to generate the kind of music that best fits our mood and desires? That's what they did at **Google Research**, where a team of researchers is introducing **MusicLM**, a model generating high-fidelity music from text descriptions such as "*a calming violin melody backed by a distorted guitar riff*", why not? Follow the link for examples. They also publicly release MusicCaps, a dataset composed of 5.5k music-text pairs, with rich text descriptions provided by human experts. Read More

Previous Class

- Reflection models
  - diffuse/specular reflectance, albedo
  - Surface orientation and light intensity
- Color vision
  - physics of light, trichromacy, color consistency, color spaces (RGB, HSV, Lab)
- Object cast light and shadows to each other
- Interpret images from local differences

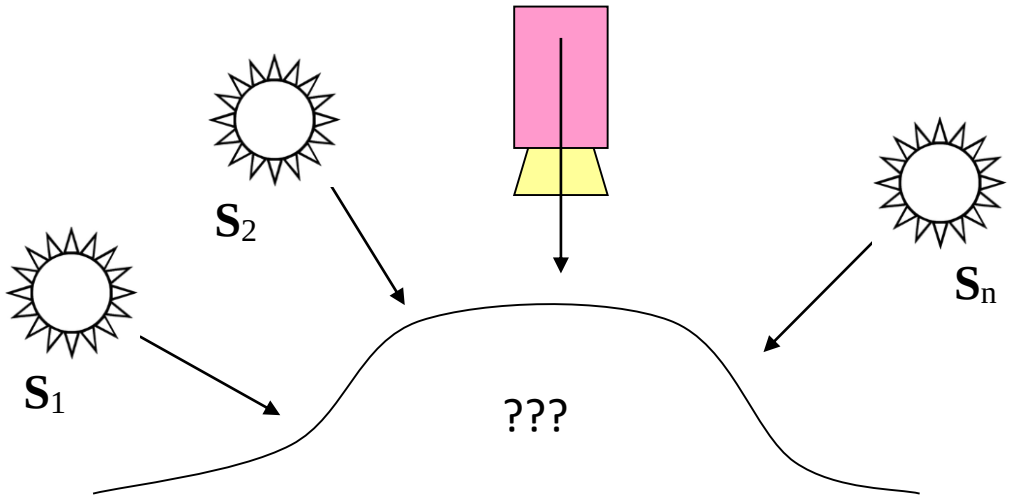
Reflection Models

- Albedo: fraction of light that is reflected
  - Determines color (amount reflected at each wavelength)

Slide credit: Derek Hoiem

Application: Photometric Stereo

- Assume:
  - A Lambertian object
  - A *local shading model* (each point on a surface receives light only from sources visible at that point)
  - A set of *known* light source directions
  - A set of pictures of an object, obtained in exactly the same camera/object configuration but using different sources
  - Orthographic projection
- Goal: reconstruct object shape and albedo



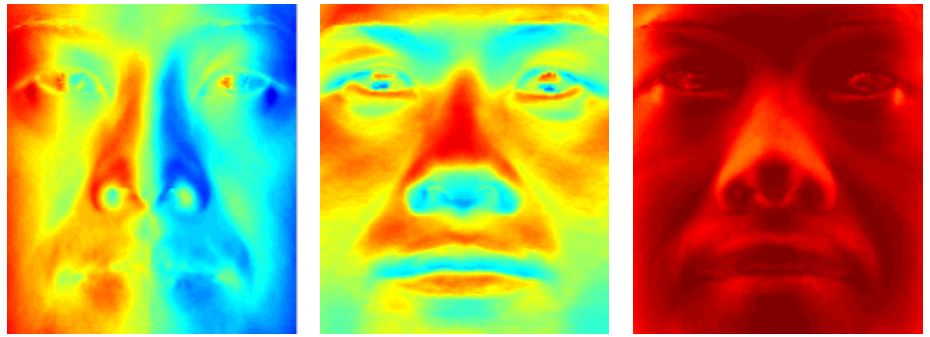
Slide credit: S. Lazebnik

Example

Input

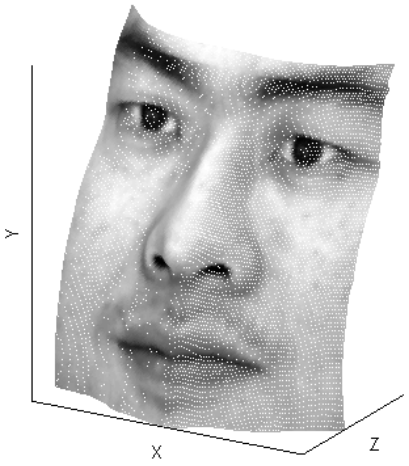
Recovered Albedo

Recovered normal field



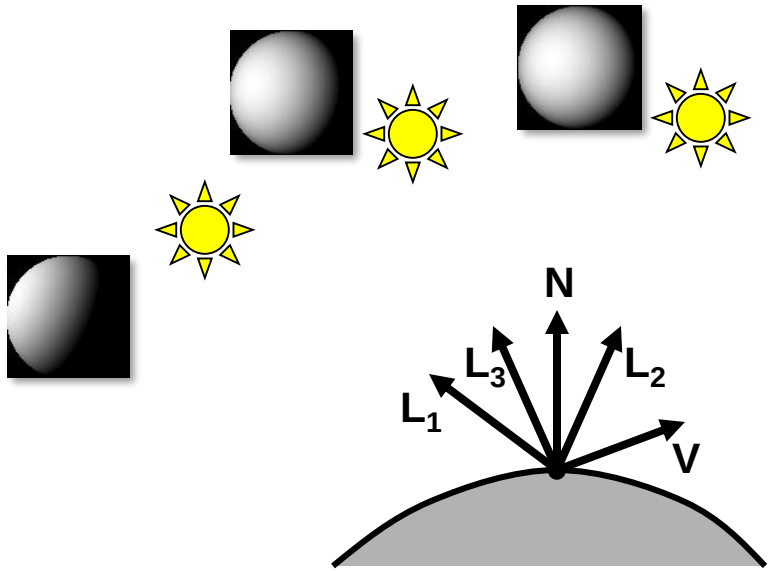
x y z

Recovered surface model



Slide credit: S. Lazebnik

Photometric Stereo



$$
\begin{aligned}
I_1 &= k_d \mathbf{N} \cdot \mathbf{L_1} \\
I_2 &= k_d \mathbf{N} \cdot \mathbf{L_2} \\
I_3 &= k_d \mathbf{N} \cdot \mathbf{L_3}
\end{aligned}
$$

Can write this as a matrix equation:

$$
\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = k_d \begin{bmatrix} \mathbf{L_1}^T \\ \mathbf{L_2}^T \\ \mathbf{L_3}^T \end{bmatrix} \mathbf{N}
$$

Slide credit: N. Snavely

Solving the Equations

$$\underbrace{\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}}_{\mathbf{I}} = \underbrace{\begin{bmatrix} \mathbf{L_1}^T \\ \mathbf{L_2}^T \\ \mathbf{L_3}^T \end{bmatrix}}_{\mathbf{L}} \underbrace{k_d \mathbf{N}}_{\mathbf{G}}$$

$\mathbf{I}$: 3×1 — Intensity (Known)

$\mathbf{L}$: 3×3 — Light direction (Known)

$\mathbf{G}$: 3×1 — Albedo x Surface normal (Unknown)

$$\mathbf{G} = \mathbf{L}^{-1}\mathbf{I}$$

How do we get the albedo and surface normal from G?

$$k_d = \|\mathbf{G}\| \qquad \mathbf{N} = \frac{1}{k_d}\mathbf{G}$$

Slide credit: N. Snavely

More than three lights

- Get better results by using more lights

$$\begin{bmatrix} I_1 \\ \vdots \\ I_n \end{bmatrix} = \begin{bmatrix} \mathbf{L_1} \\ \vdots \\ \mathbf{L_n} \end{bmatrix} k_d \mathbf{N}$$

Least squares solution:

$$\begin{aligned} \mathbf{I} &= \mathbf{LG} \\ \mathbf{L^T I} &= \mathbf{L^T L G} \\ \mathbf{G} &= (\mathbf{L^T L})^{-1}(\mathbf{L^T I}) \end{aligned}$$

Solve for N, k_d as before

What's the size of $\mathbf{L^T L}$?

In MATLAB

```
G = L\I;
```

Slide credit: N. Snavely

Python Code:

```python
import sys
import numpy as np
#from scipy import misc
from matplotlib import pyplot as plt
import imageio

def im_flatten(image):
    return (image / 255).flatten()

def image_regress(images, L):
    """
    A simple implementation of Photometric Stereo using Ordinary Least Squares.
    3 or more light sources, provided the number of images matches the number o
    Images should be an `n` element list of single-channel greyscale images.
    L is a 3xn numpy array of column vectors for the light direction vectors"""

    # We'll need this later...
    original_size = images[0].shape[:2]

    # Normalise all images and convert to row vectors (each image is one row)
    images = np.vstack(map(im_flatten, images))

    # Make sure that lighting vectors are normalised
    L = L / np.linalg.norm(L, ord=2, axis=1, keepdims=True)

    # Solve for G = N'*rho using ordinary least squares
    # (L^T L) \ L^T
    norm_sln = np.linalg.pinv(L.T.dot(L)).dot(L.T)

    # For a single pixel (3x1 column) we can trivially calculate G: norm_sum *
    # norm_sln is 3x3, images is 3xn (where n i num pixels)
    # It's slow to iterate this, but the einsum method lets us broadcast the multiplication over the array
    G = np.einsum("ij,il", norm_sln, images)

    # The albedo is just the column-wise L2 norm (magnitude) of G...
    rho = np.linalg.norm(G, axis=0)

    # The normal map is simply each column of G divided the equivalent column in the albedo
    N = np.divide(G, np.vstack([rho] * 3))

    # Reshape back to image
    rho = rho.reshape(original_size)

    # We need to transpose the normal list before we reshape
    N = N.T.reshape(original_size[0], original_size[1], 3)

    return N, rho
```

```python

if __name__ == "__main__":
    if len(sys.argv) < 5:
        print("USAGE: albedo <lights_filename> <image_1_filename> <image_2_filename> <image_3_filename> ...")
        sys.exit()

    # We take the transpose here because the lighting vectors should be columns and we read as rows.
    L = np.loadtxt(sys.argv[1]).T

    # Sanity check the inputs one last time before we get going.
    if len(sys.argv) - 2 != L.shape[1]:
        raise ValueError('Error: The number of light vectors does not match the number of input images.')

    # Create a list from the input images
    images = []
    for i in range(L.shape[1]):
        # Important: Note that we flatten the images to greyscale here
        images.append(imageio.imread(sys.argv[i + 2], as_gray=True))

    # Run the PS algorithm
    N, rho = image_regress(images, L)

    # Values in N range from -1...1, we need them in 0...1, so we'll quickly remap it
    N_display = (N + 1) / 2

    plt.subplot(1,2,1)
    plt.imshow(N_display)
    plt.subplot(1,2,2)
    plt.imshow(255 * rho, cmap='gray')
    plt.show()
```

Example

- Before we start, there are some import assumptions about the surface we're imaging and the lights.

- Firstly, we assume that the surface exhibits Lambertian reflectance, i.e. the amount of light reflected from a surface is directly proportional to the angle between the light direction and the surface normal.

- The other key assumption is that the lights are point sources – they come from infinitely far away, so the rays are parallel and you don't get differences in illumination across the image.

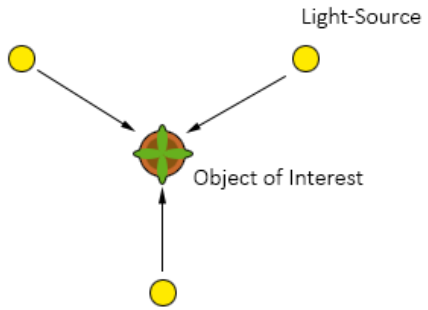
- Realistically, rarely are both these assumptions held completely, and so there are certain steps we can take to improve estimation accuracy, but I'll talk about it later.

Example

The relationship between the light direction, $L = (L_x, L_y, L_z)$, albedo, ρ, and surface normal $N = (N_x, N_y, N_z)$ for a given pixel is provided below:

$$I(x, y) = \rho(x, y)L.N(x, y) = \rho(x, y) \begin{bmatrix} L_x \\ L_y \\ L_z \end{bmatrix}^\top \begin{bmatrix} N_x(x, y) \\ N_y(x, y) \\ N_z(x, y) \end{bmatrix}$$

We assume that L is known, and the $N(x, y)$ is a unit vector, thus we are left with three unknowns, ρ, $N_x(x, y)$ and $N_y(x, y)$. Therefore we only need 3 differently-lit images to estimate the surface normal and albedo pair at each pixel.



By re-arranging the above equation, we can isolate the lighting directions from the surface normals and albedo, such that we first solve for $G(x, y) = \rho(x, y)N(x, y)^\top$, the we'll split $G(x, y)$ back into the ρ and N later.

$$I = G(x, y)^\top L = L^\top G(x, y)$$

Now we simply solve G for a given pixel using the using normal equations as follows:

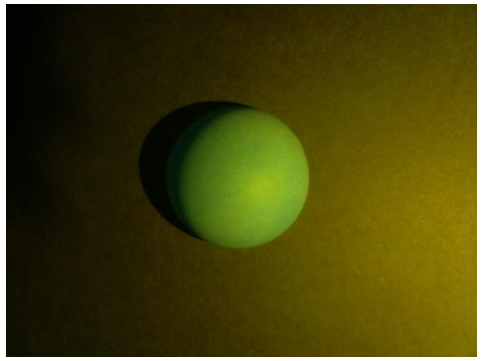
$$G(x, y) = (L^\top L)^{-1} L^\top \cdot I(x, y)$$

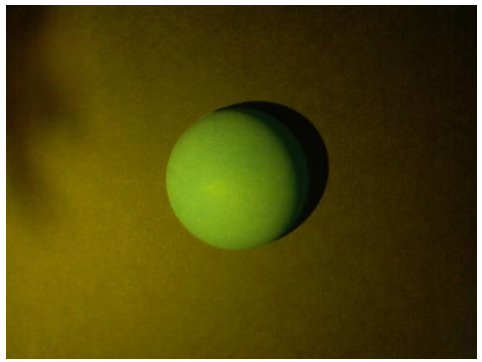
We can work out $N(x, y)$ and $\rho(x, y)$ from $G(x, y)$ trivially:

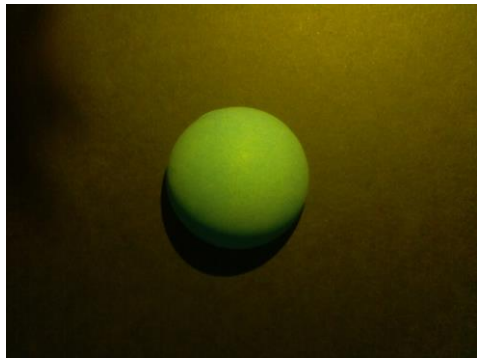
$$\rho(x, y) = ||G(x, y)|| = \sqrt{G_x(x, y)^2 + G_y(x, y)^2 + G_z(x, y)^2}$$

$$N(x, y) = \frac{G(x,y)}{\rho}$$

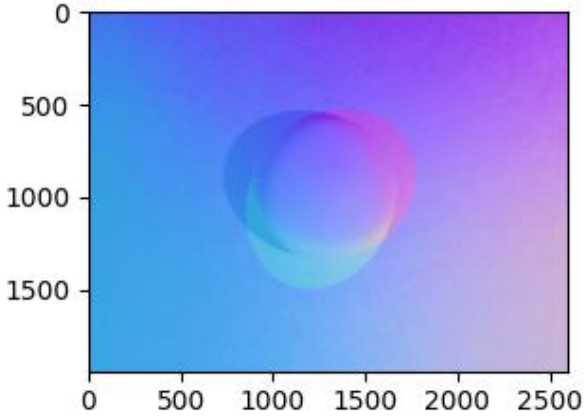
Example

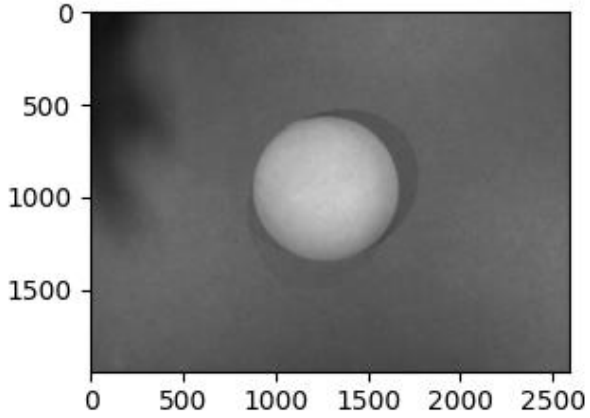






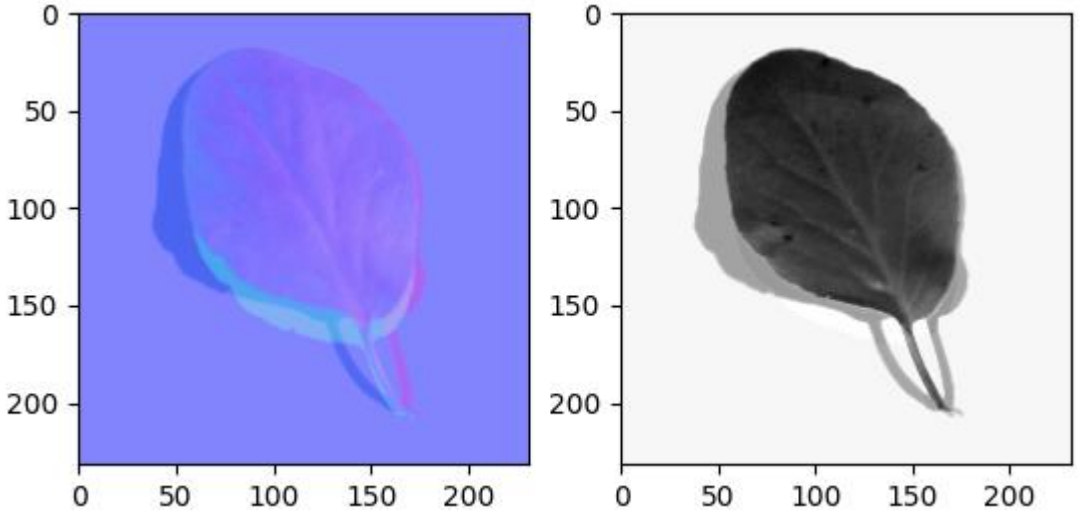
$$+ \begin{bmatrix} 0.6233 & 0.2655 & 0.7355 \\ -0.5397 & 0.3175 & 0.7797 \\ 0.1728 & -0.5480 & 0.8185 \end{bmatrix} =$$





Example

$$+ \begin{bmatrix} 0.5751 & 0.0313 & 0.8175 \\ -0.0343 & -0.4157 & 0.9089 \\ -0.4981 & 0.0409 & 0.8662 \end{bmatrix} =$$

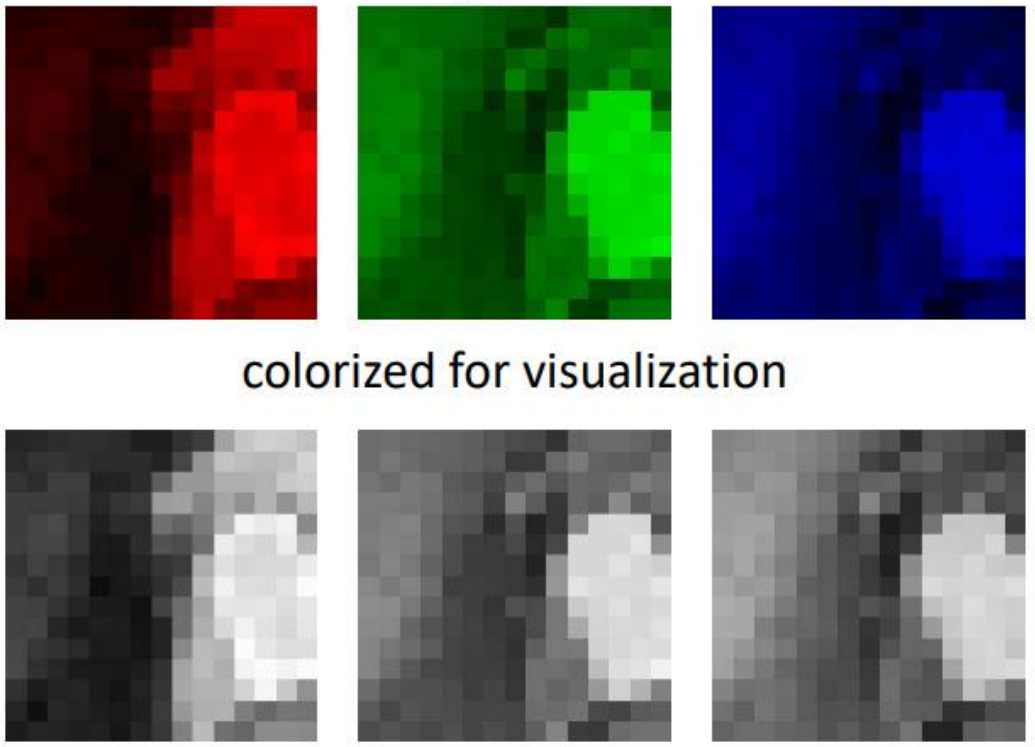


Usage of command. Save the code in the working directory with name albedo.py
Save the images and light variables as a text file in a folder named “leaf”
Run the following command:
python albedo.py leaf/lights.txt leaf/source_00001.png leaf/source_00002.png leaf/source_00003.png

What is an image?

color image patch

How many bits are the intensity values?



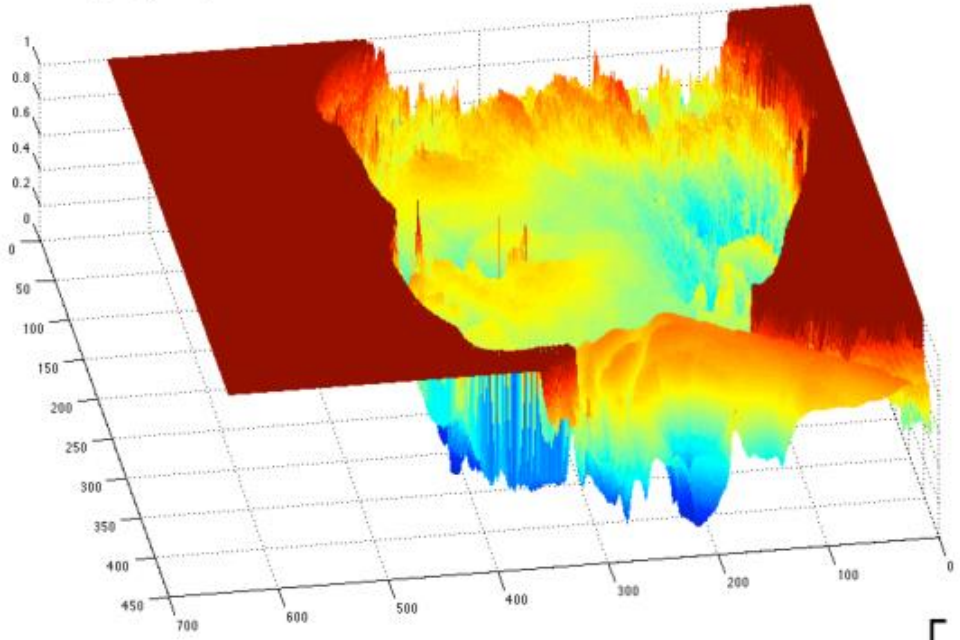
Each channel is a 2D array of numbers.

What is an Image?

grayscale image

What is the range of the image function f?

$f(\boldsymbol{x})$



domain $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

A (grayscale) image is a 2D function.

What types of image transformations can we do?

Filtering

changes pixel *values*

Warping

changes pixel *locations*

What types of image transformations can we do?

F

Filtering

$$G(\boldsymbol{x}) = h\{F(\boldsymbol{x})\}$$

G

changes *range* of image function

F

Warping

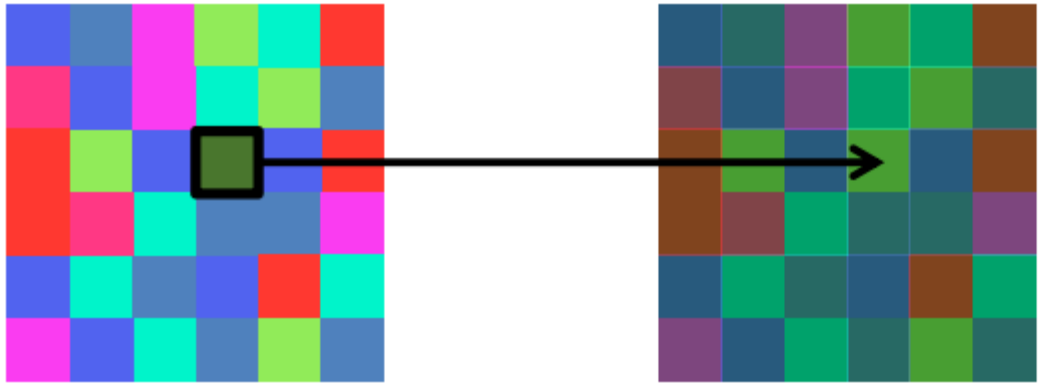
$$G(\boldsymbol{x}) = F(h\{\boldsymbol{x}\})$$

G

changes *domain* of image function

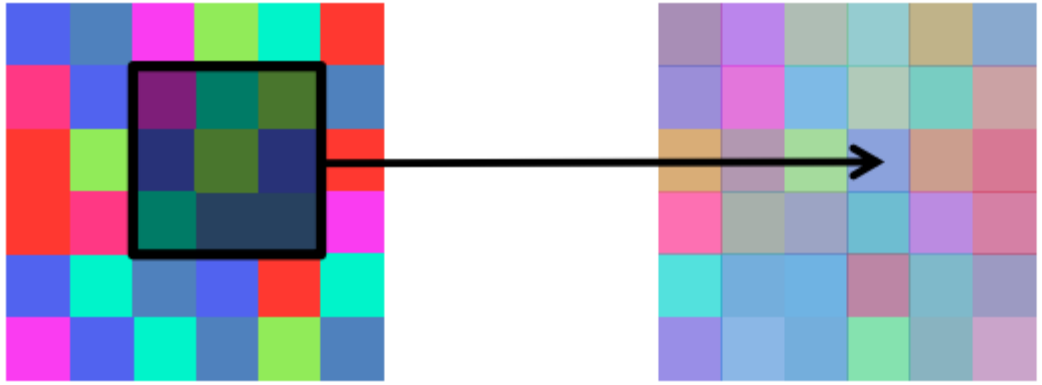
What Determines Pixels' Color?

Point Operation



point processing

Neighborhood Operation



"filtering"

Examples of Point Processing

original

x

darken

$x - 128$

lower contrast

$\frac{x}{2}$

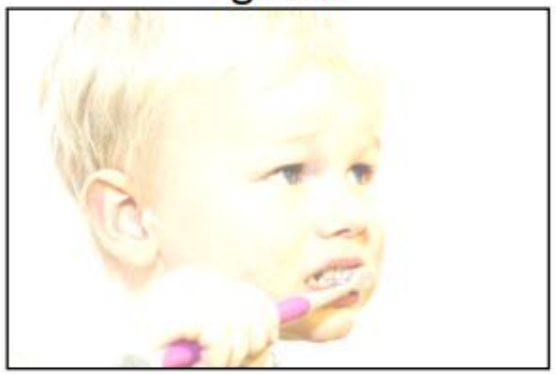
non-linear lower contrast

$\left(\frac{x}{255}\right)^{1/3} \times 255$

invert

$255 - x$

lighten



$x + 128$

raise contrast

$x \times 2$

non-linear raise contrast

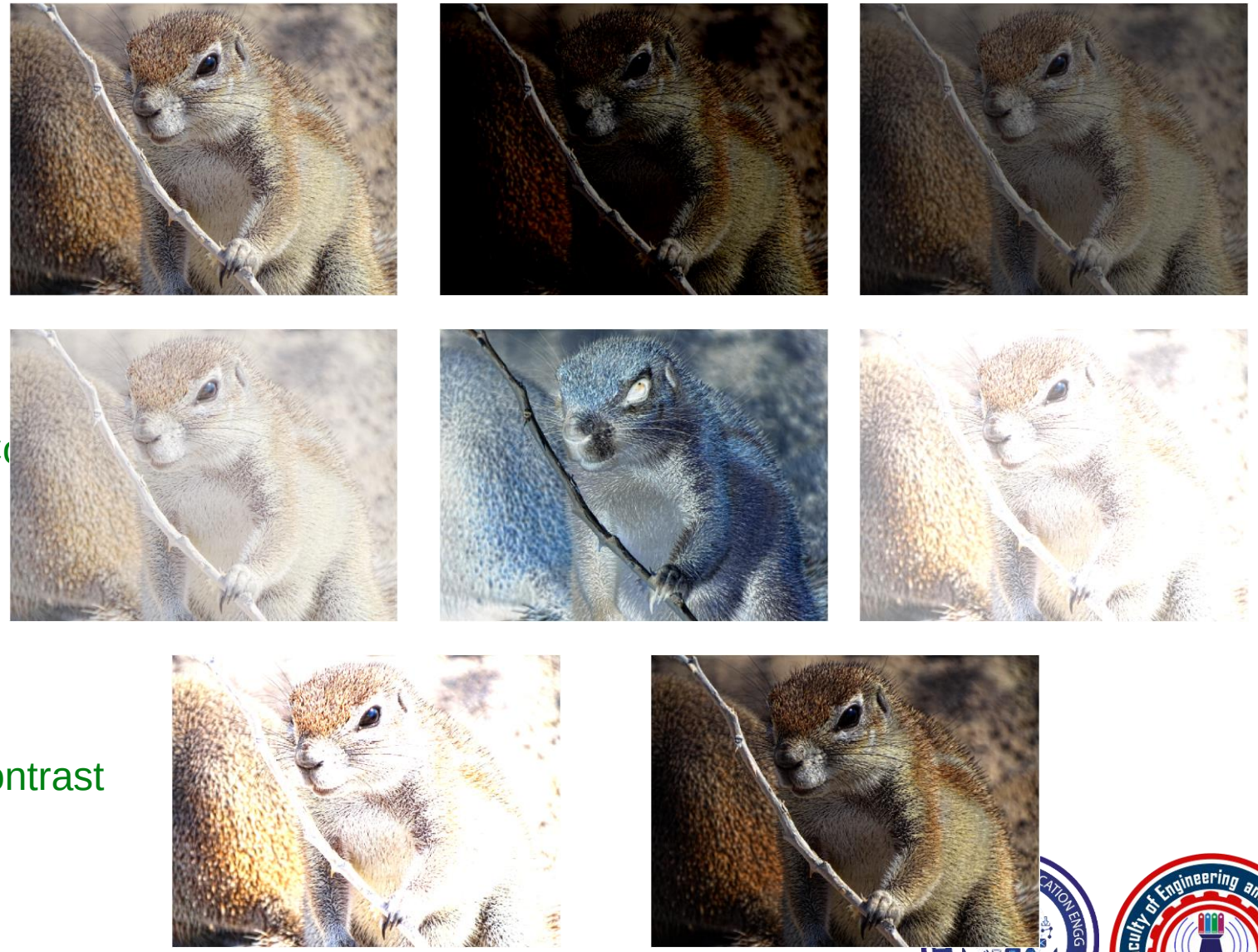
$\left(\frac{x}{255}\right)^{2} \times 255$

How would you implement these?

Example

- MATLAB code

```
I = imread('0776.png'); %Original Image
imshow(I-128) %Darken Image
imshow(I/2) % Lower Contrast Image
I1 = double(I)./255;
I2 = I1.^(1/3);
figure, imshow(I2) %Non-Linear Lower C
imshow(255-I) %Invert
imshow(I+128) %Lighten
imshow(I*2) %Raise Contrast
I3 = I1.^double(2);
figure, imshow(I3) %Non-linear Raise Contrast
```



Many Other types of Point Processing

camera output

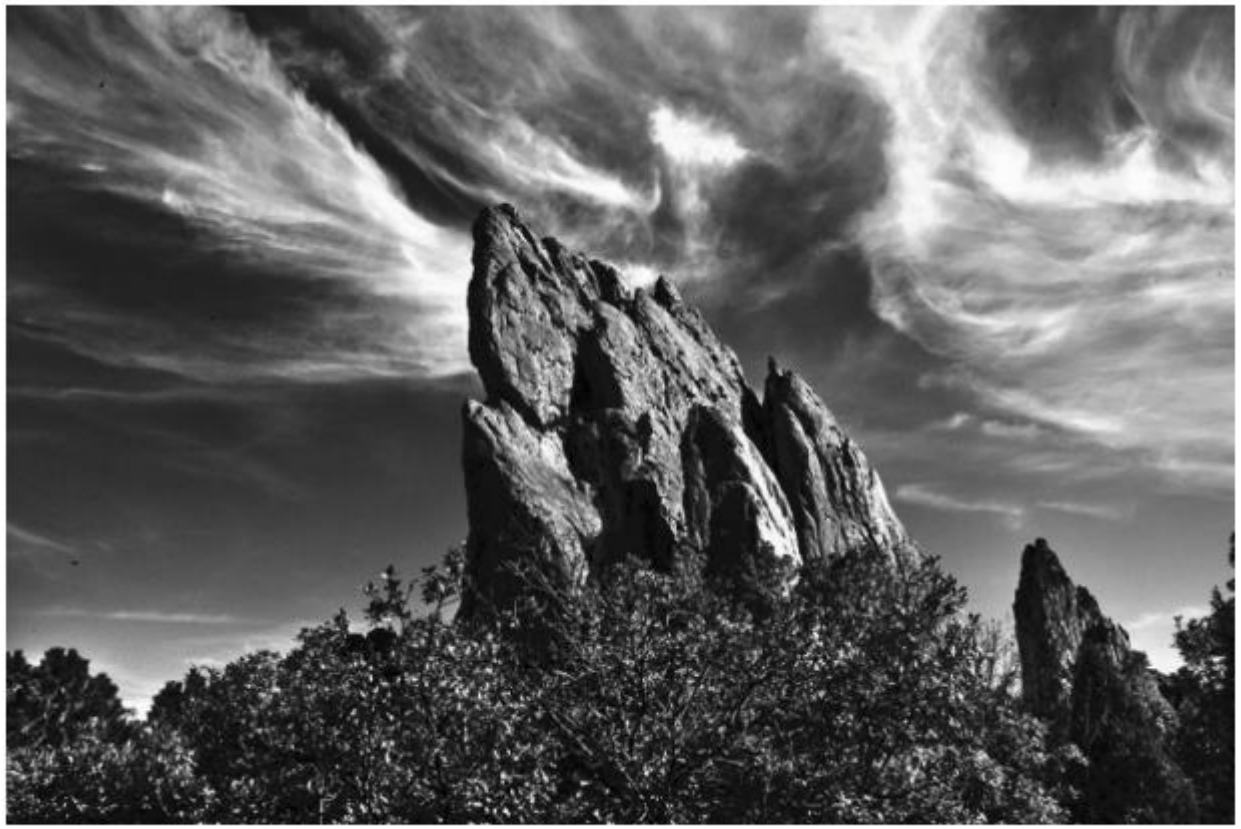
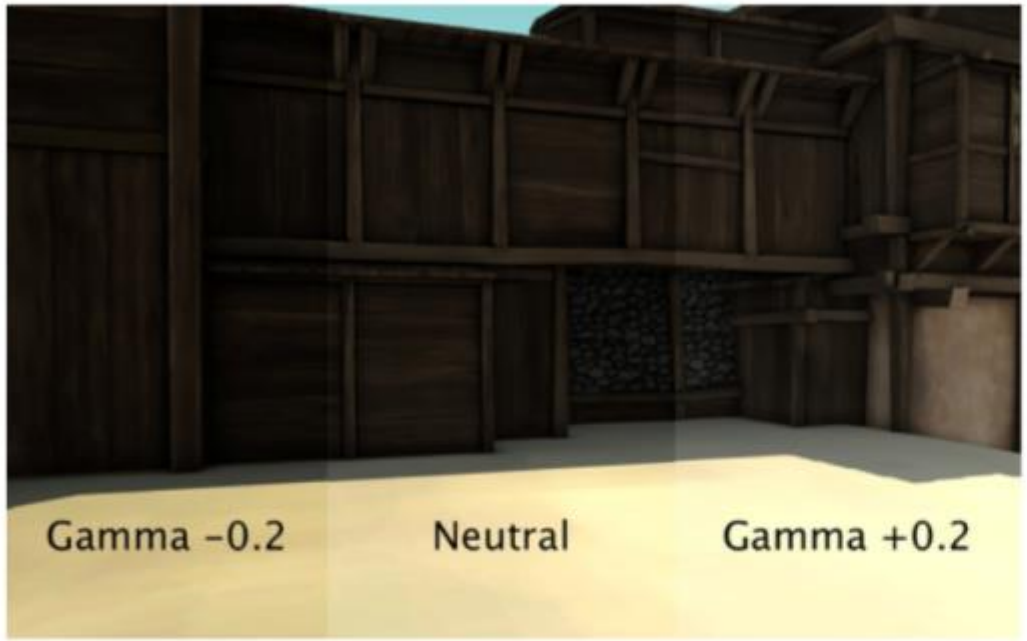


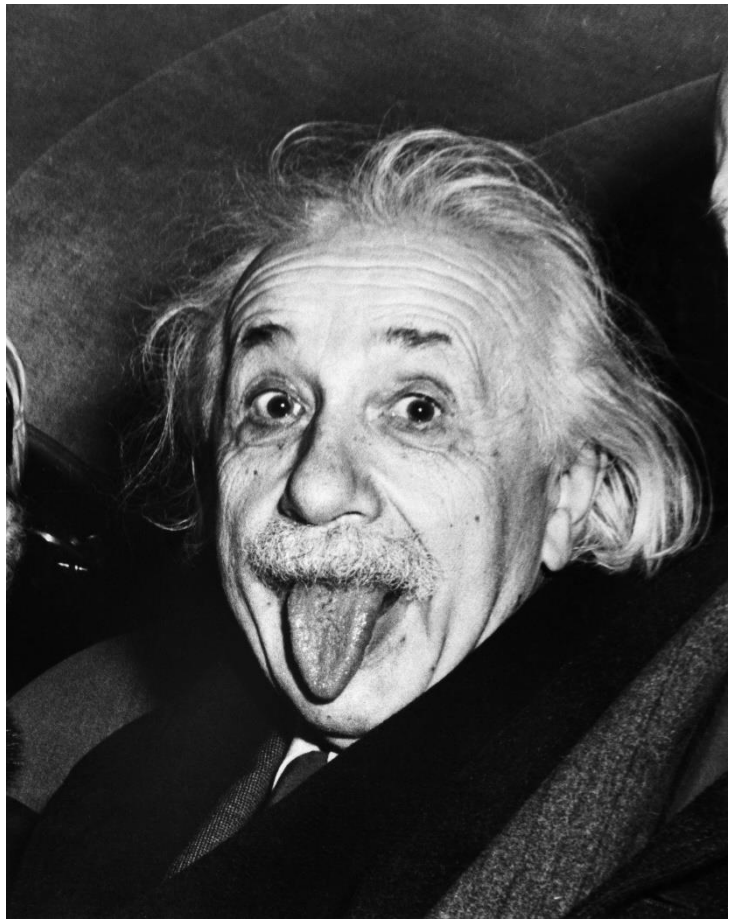
image after stylistic tonemapping

[Bae et al., SIGGRAPH 2006]

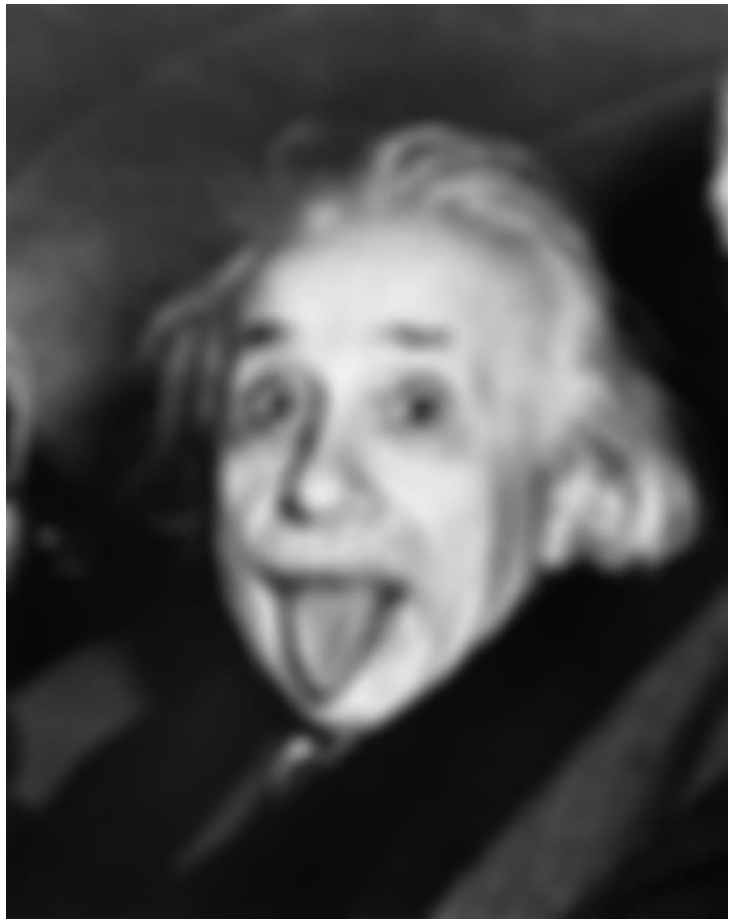
Many other types of point processing



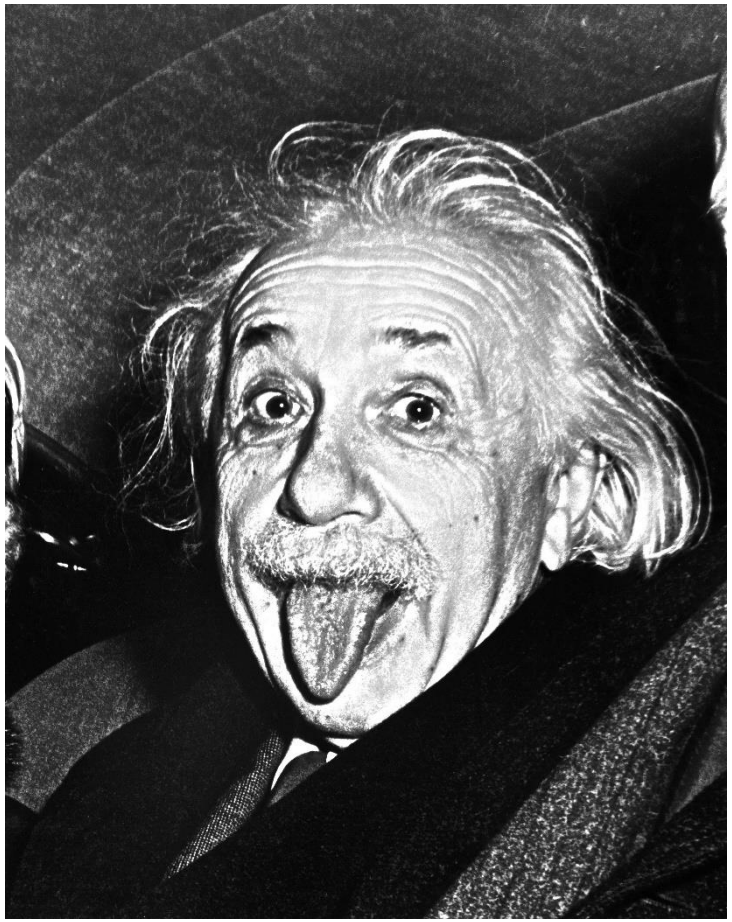
Why should we care?



Input



Smoothing



Sharpening

https://en.wikipedia.org/wiki/Albert_Einstein_in_popular_culture#/media/File:Einstein_tongue.jpg

Why should we care?

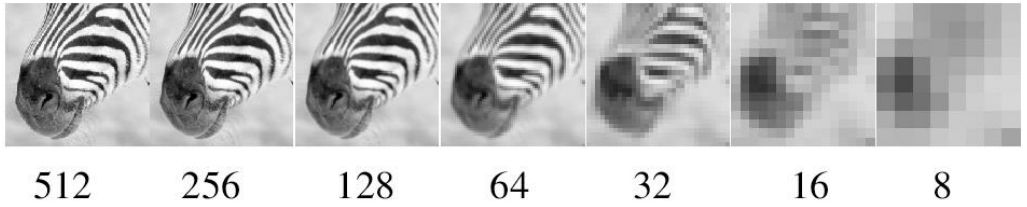


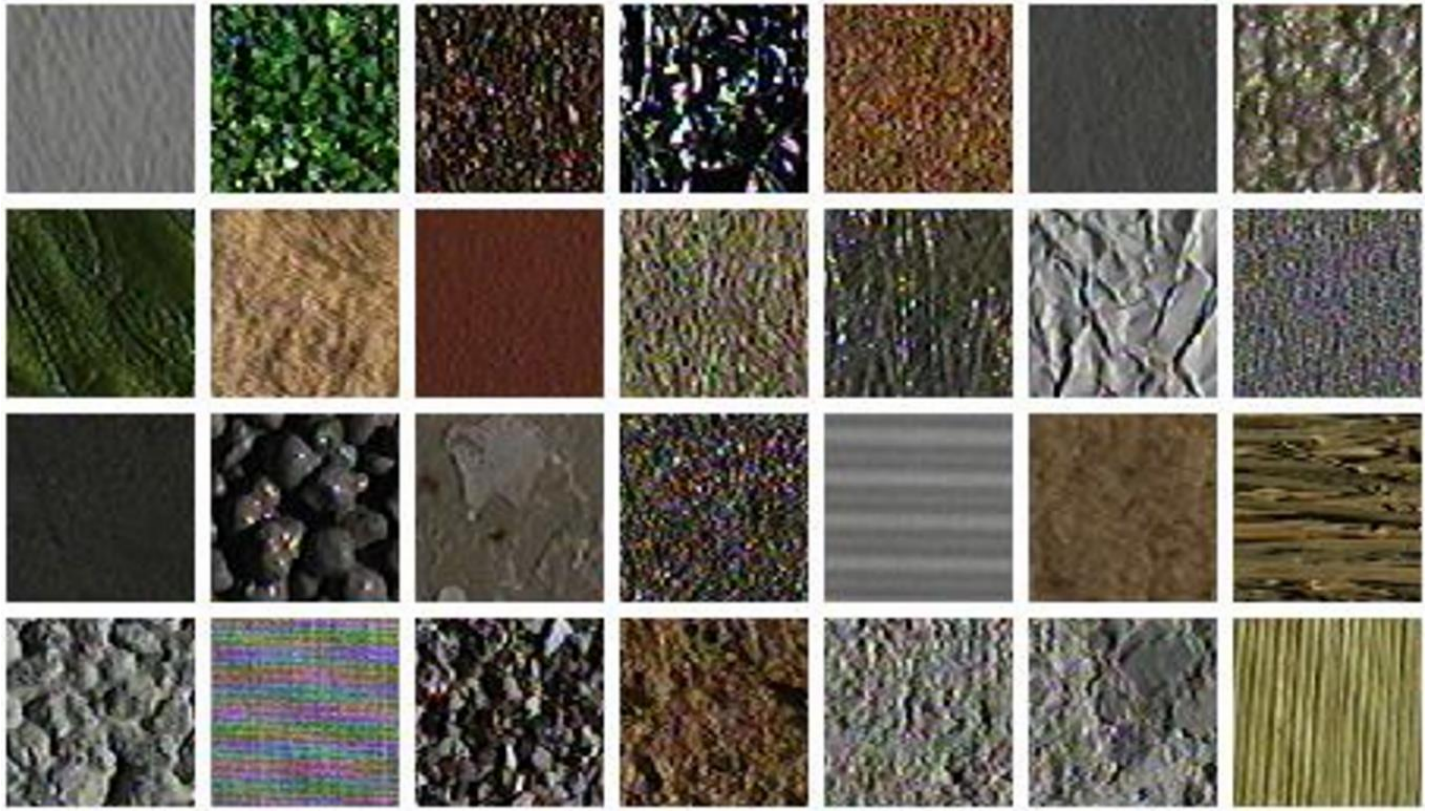
Image Pyramid

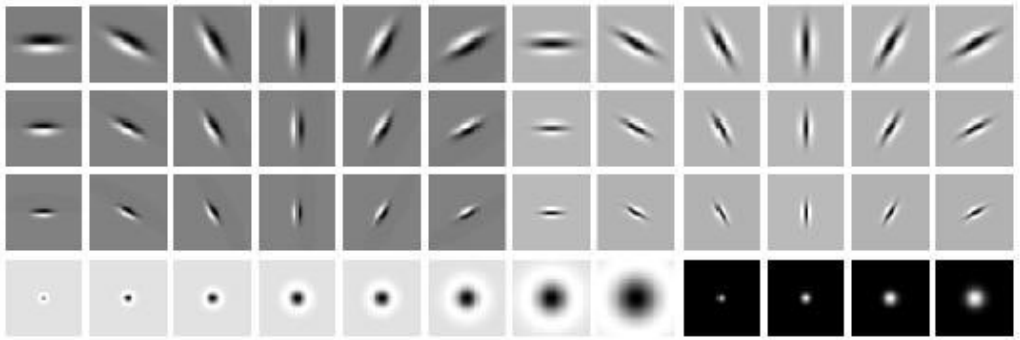
Image interpolation/resampling

Source: D Forsyth

Source: N Snavely

Why should we care?





LM filter bank. Code here

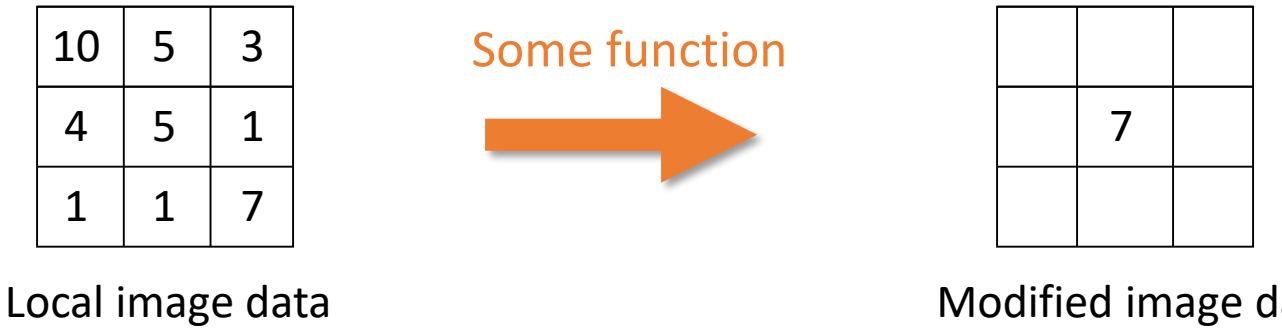
Representing textures with filter banks

The raster image (pixel matrix)

0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

Image Filtering

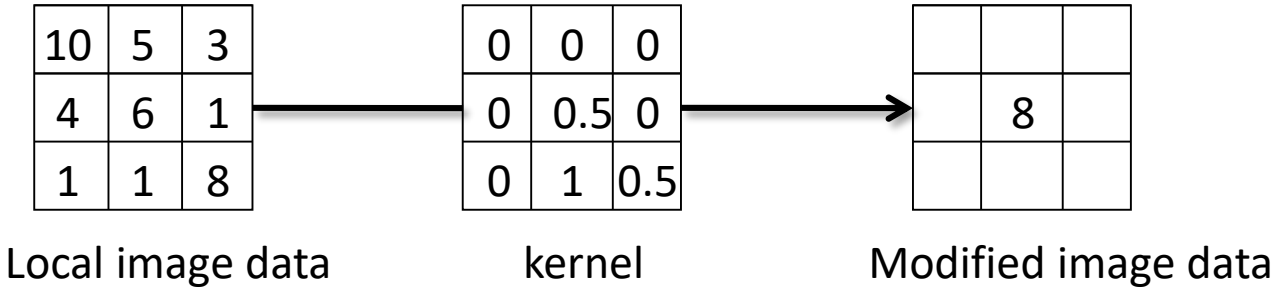
- Image filtering: for each pixel, compute function of local neighborhood and output a new value
  - Same function applied at each position
  - Output and input image are typically the same size



Slide Credit: L. Zhang

Image Filtering

- Linear filtering
  - function is a weighted sum/difference of pixel values



- Really important!
  - Enhance images
    - Denoise, smooth, increase contrast, etc.
  - Extract information from images
    - Texture, edges, distinctive points, etc.
  - Detect patterns
    - Template matching

Slide credit: Derek Hoiem

Question: Noise Reduction

- Given a camera and a still scene, how can you reduce noise?

Take lots of images and average them!
What's the next best thing?

Source: S. Seitz

Linear Shift-Invariant Image Filtering

- Replace each pixel by a linear combination of its neighbors (and possibly itself).
- The combination is determined by the filter's kernel.
- The same kernel is shifted to all pixel locations so that all pixels use the same linear combination of their neighbors.

Example: The Box Filter

- also known as the 2D rect (not rekt) filter
- also known as the square mean filter

kernel $g[\cdot,\cdot] = \frac{1}{9}$

1	1	1
1	1	1
1	1	1

- replaces pixel with local average
- has smoothing (blurring) effect

Example: Box Filter

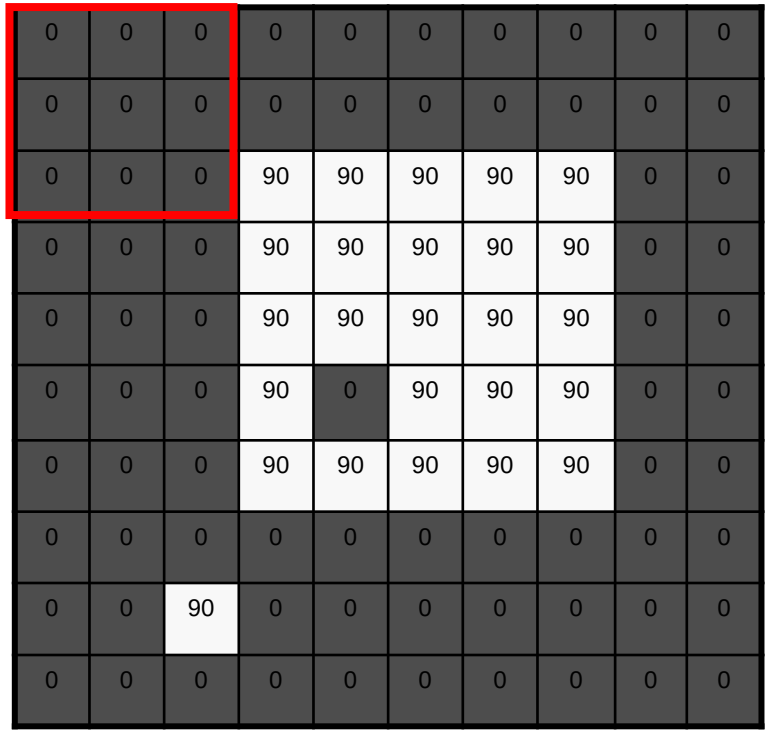
$$g[\cdot,\cdot]$$

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

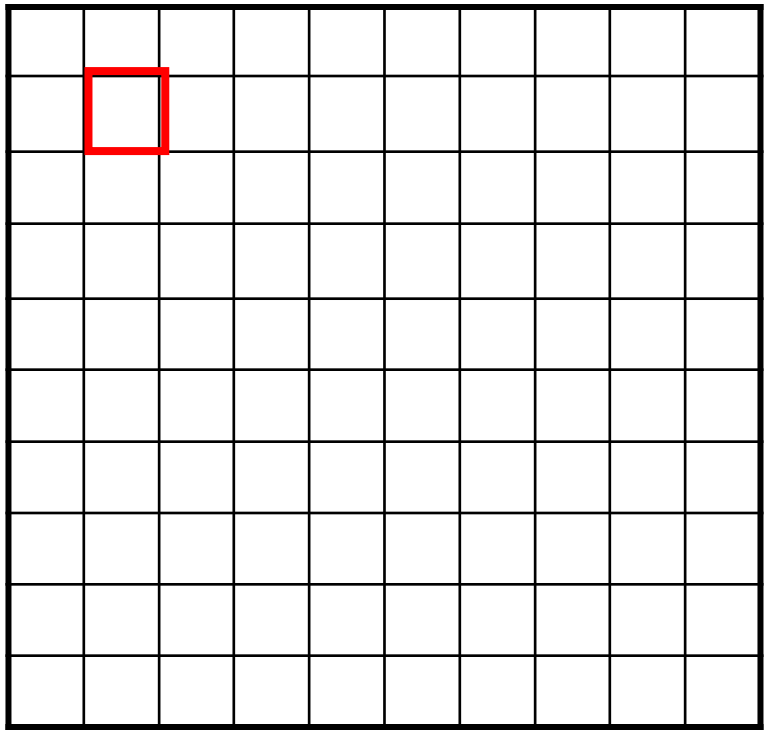
Slide credit: David Lowe

Image Filtering

$f[.,.]$



$h[.,.]$



$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k,n+l]$$

35 Slide credit: S. Seitz

Image Filtering

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10							

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Slide credit: S. Seitz

Image Filtering

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20						

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Slide credit: S. Seitz

Image Filtering

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30					

$g[\cdot,\cdot]$ $\frac{1}{9}$

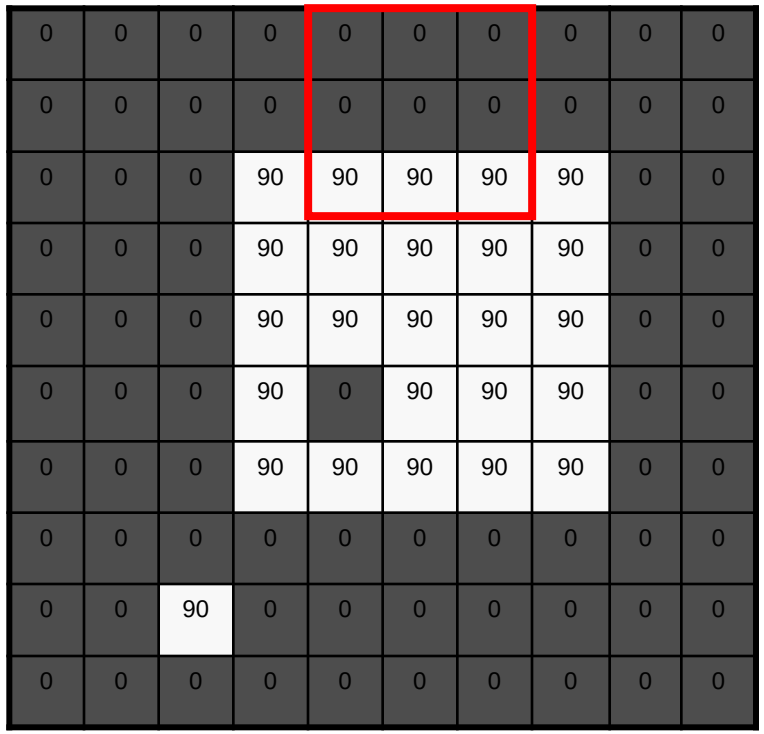
1	1	1
1	1	1
1	1	1

$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k,n+l]$$

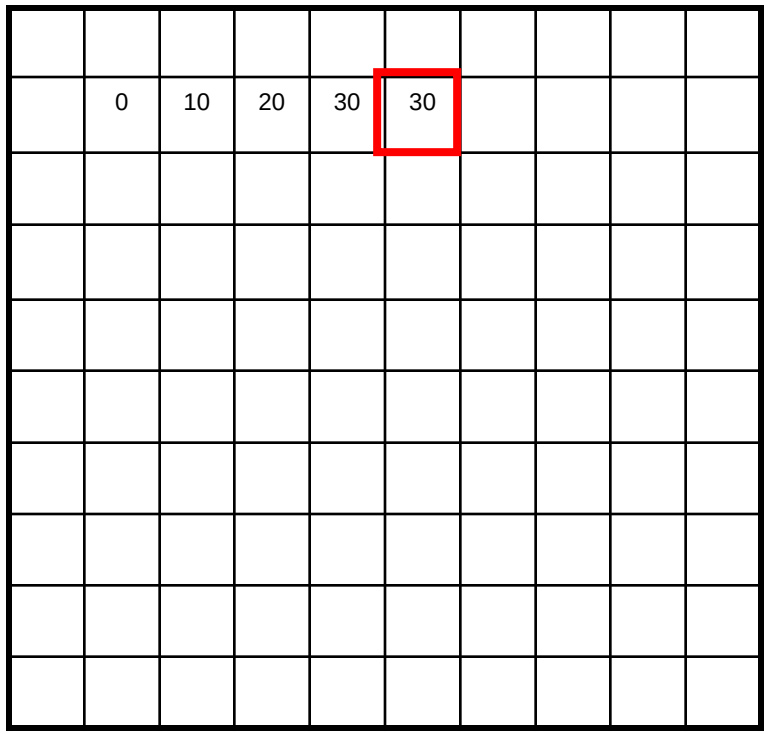
Slide credit: S. Seitz

Image Filtering

$f[.,.]$



$h[.,.]$



$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Slide credit: S. Seitz

Image Filtering

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30				
				?					

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Slide credit: S. Seitz

Image Filtering

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30				
						?			
				50					

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Slide credit: S. Seitz

Image Filtering

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

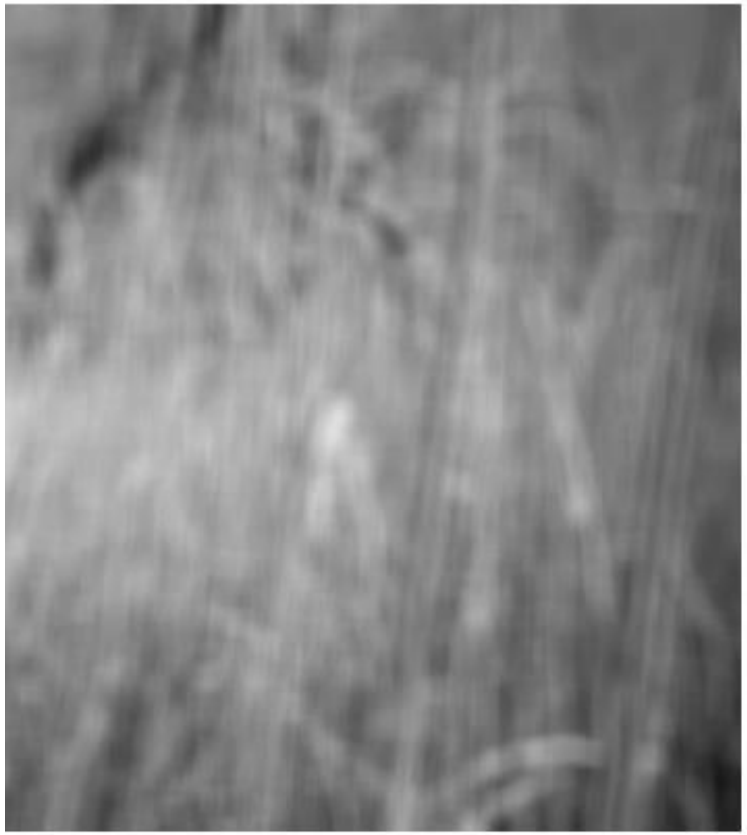
$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Slide credit: S. Seitz

Some Realistic Examples



Box filter python

```python
import cv2
import numpy as np

img = cv2.imread('0420.png')
blur = cv2.boxFilter(img,-1,(3,3), normalize=True)

cv2.imshow("Box Filter 3x3",np.hstack([img,blur]))
cv2.waitKey(0)
```

3x3
5x5
9x9
17x17

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood

- Achieve smoothing effect (remove sharp features)

$$g[\cdot,\cdot]$$

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

Slide credit: David Lowe

Properties of Smoothing Filters

- Smoothing
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter

Slide credit: Kristen Grauman

Correlation Filtering

Say the averaging window size is 2k+1 x 2k+1:

$$G[i,j] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^{k}\sum_{v=-k}^{k} F[i+u, j+v]}_{\text{Loop over all pixels in neighborhood around image pixel F[i,j]}}$$

Now generalize to allow different weights depending on neighboring pixel's relative position:

$$G[i,j] = \sum_{u=-k}^{k}\sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{Non-uniform weights}} F[i+u, j+v]$$

Slide credit: Kristen Grauman

Correlation Filtering

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

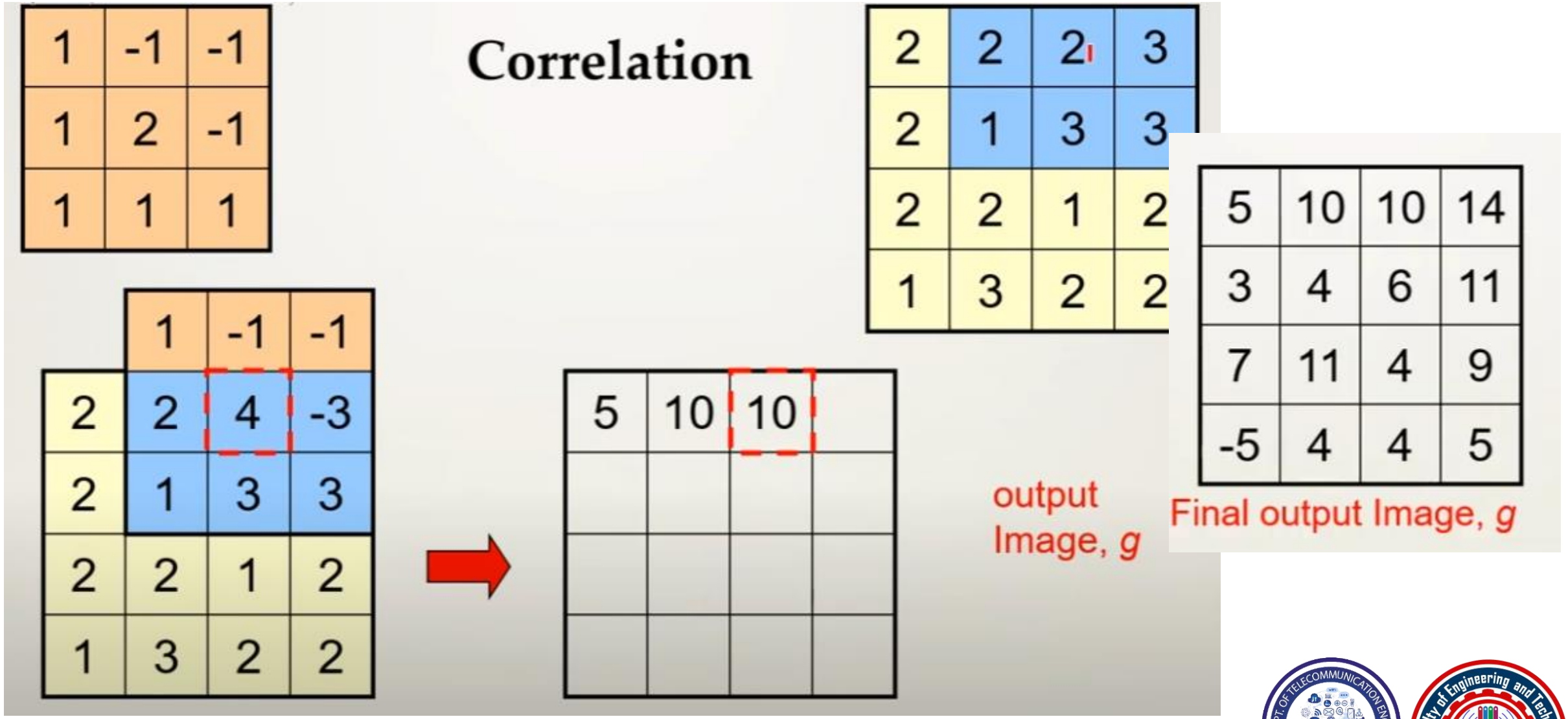
This is called cross-correlation, denoted $G = H \otimes F$

Filtering an image: replace each pixel with a linear combination of its neighbors.

The filter "kernel" or "mask" $H[u,v]$ is the prescription for the weights in the linear combination.

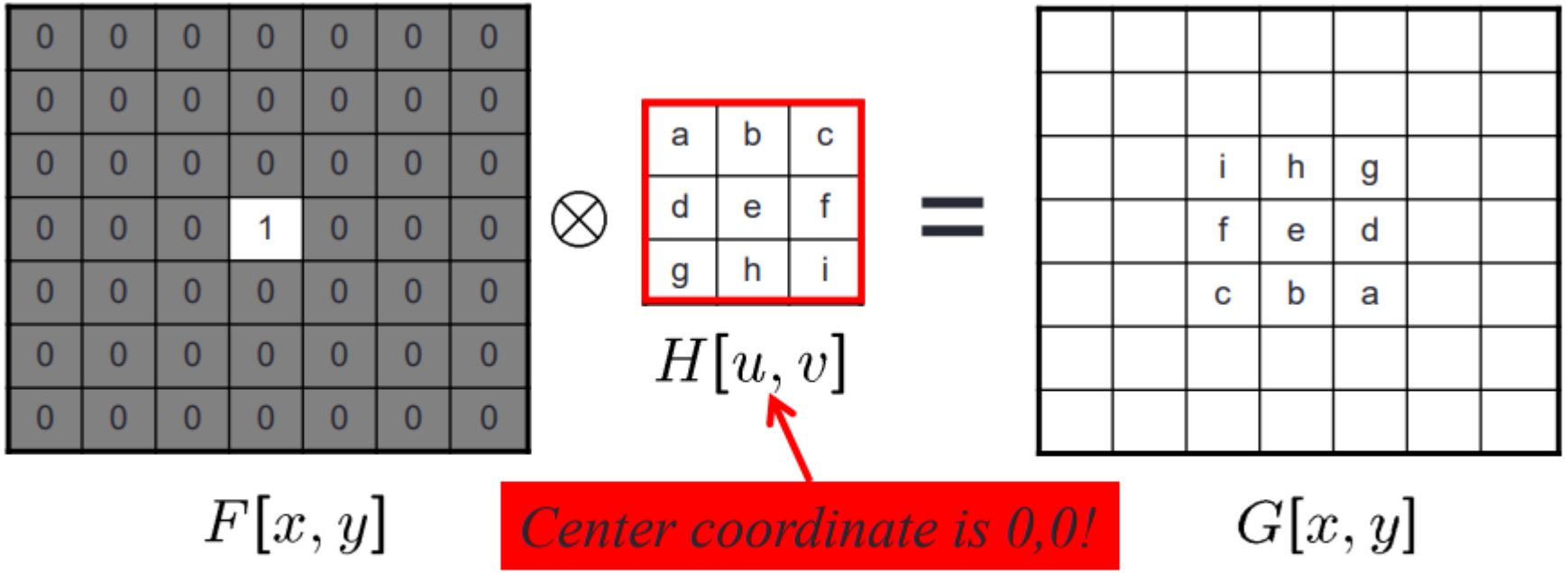
Slide credit: Kristen Grauman

Correlation Filtering



Filtering an Impulse Signal

What is the result of filtering the impulse signal (image) *F* with the arbitrary kernel *H*?



If you just "filter" meaning slide the kernel over the image you get a ***reversed*** response.

Slide credit: Kristen Grauman

Example Python

```python
# read image
src = cv2.imread('0420.png', 0)
# prepare averaging filter
kernel = np.ones((3, 3), dtype=np.float32)
kernel /= 9
# apply kernel to the original image
dst = cv2.filter2D(src, -1, kernel)
# concatenate images horizontally
result = np.concatenate((src, dst), axis=1)
cv2.imshow("Correlation Filter 3x3",np.hstack([src,dst]))
cv2.waitKey(0)
```

```python
# read image
src = cv2.imread('0420.png', 0)
# prepare the filter
kernel = [[0,0,0], [0,1,0], [0,0,0]]
# apply kernel to the original image
dst = cv2.filter2D(src, -1, np.array(kernel))
# concatenate images horizontally
result = np.concatenate((src, dst), axis=1)
cv2.imshow("Correlation Filter 3x3",np.hstack([src,dst]))
cv2.waitKey(0)
```

```python
# read image
src = cv2.imread('0420.png', 0)
# prepare the filter
kernel = [[1,0,0], [0,1,0], [0,0,1]]
# apply kernel to the original image
dst = cv2.filter2D(src, -1, np.array(kernel))
# concatenate images horizontally
result = np.concatenate((src, dst), axis=1)
cv2.imshow("Custom Filter 3x3",np.hstack([src,dst]))
cv2.waitKey(0)
```

Convolution for 1D Continuous Signal

Definition of filtering as convolution:

notice the flip

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

filtered signal

filter

input signal

Convolution for 1D Continuous Signals

Definition of filtering as convolution:

notice the flip

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

filtered signal

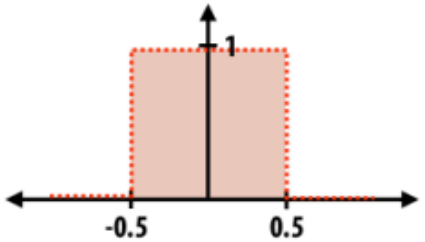
filter

input signal

Consider the box filter example:

1D continuous box filter

$$f(x) = \begin{cases} 1 & |x| \leq 0.5 \\ 0 & otherwise \end{cases}$$



filtering output is a blurred version of g

$$(f * g)(x) = \int_{-0.5}^{0.5} g(x-y)dy$$

Convolution for 2D Discrete Signals

Definition of filtering as convolution:

notice the flip

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j) I(x - i, y - j)$$

filtered image — filter — input image

If the filter $f(i, j)$ is non-zero only within $-1 \leq i, j \leq 1$, then

$$(f * g)(x, y) = \sum_{i,j=-1}^{1} f(i, j) I(x - i, y - j)$$

The kernel we saw earlier is the 3x3 matrix representation of $f(i, j)$.

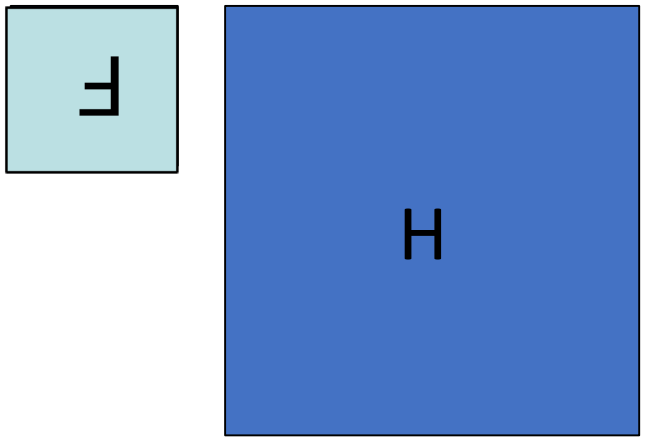
Convolution

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

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

$$G = H \star F$$

Notation for convolution operator



Slide credit: Kristen Grauman

Convolution vs Correlation

Definition of filtering as convolution:

notice the flip

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x - i, y - j)$$

Definition of filtering as correlation:

notice the lack of a flip

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x + i, y + j)$$

- Most of the time won't matter, because our kernels will be symmetric.
- Will be important when we discuss frequency-domain filtering

Convolution vs Correlation

Convolution

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

```
G=conv2(H,F);
```

$$G = H \star F$$

Cross-correlation

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

```
G=filter2(H,F);
```
or
```
G=imfilter(F,H);
```

$$G = H \otimes F$$

For a Gaussian or box filter, how will the outputs differ?

If the input is an impulse signal, how will the outputs differ?

Slide credit: Kristen Grauman

Convolution Filter Example

```
# read image
src = cv2.imread('0420.png', 0)
# prepare the filter
kernel = [[1,-1,-1], [1,2,-1], [1,1,1]]
# apply kernel to the original image
dst = cv2.filter2D(src, -1, np.array(kernel))
# concatenate images horizontally
result = np.concatenate((src, dst), axis=1)
cv2.imshow("Custom Filter 3x3",np.hstack([src,dst]))
cv2.waitKey(0)
```

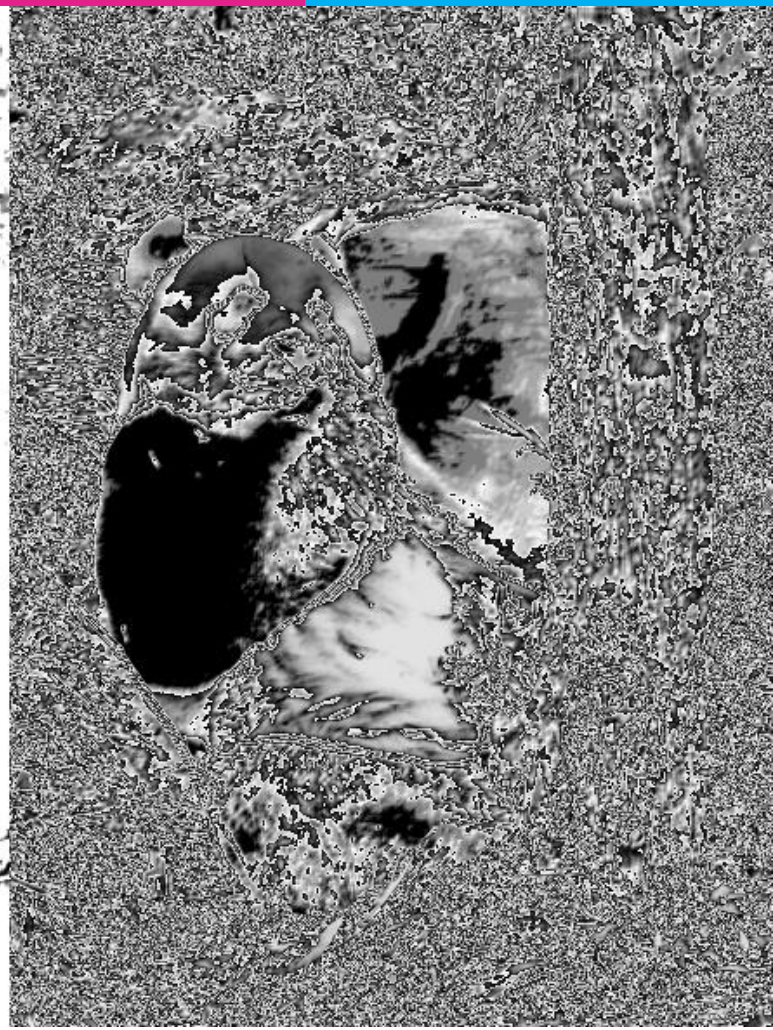
1	2	-1	

9	6	14	5
11	7	6	5
9	12	8	5

Ali Khowaja

Correlation vs Convolution

```
# read image
src = cv2.imread('0420.png',0)
# prepare the filter
kernel = np.array([[1,1,1],[1,1,0],[1,0,0]])
# apply kernel to the original image using convolution filtering
dst_conv  = ndimage.convolve(src, kernel, mode='constant', cval=1.0)
# apply kernel to the original image using correlation filtering
dst_corr = cv2.filter2D(src, -1, kernel)
# concatenate images horizontally
result = np.concatenate((src, dst_corr, dst_conv), axis=1)
cv2.imshow("convolution vs correlation",np.hstack([dst_corr,dst_conv]))
cv2.waitKey(0)
```



Practice with Linear Filters

Original

0	0	0
0	1	0
0	0	0

?

Practice with Linear Filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Slide credit: David Lowe (UBC)

Practice with Linear filters

Original

0	0	0
0	0	1
0	0	0

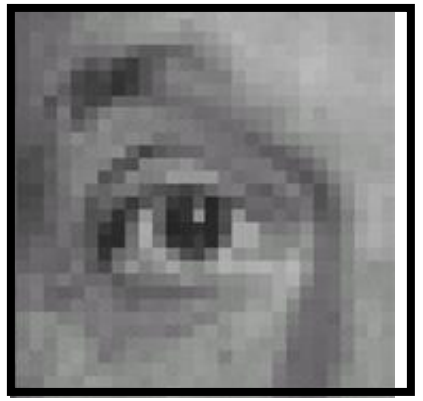
?

Slide credit: David Lowe (UBC)

Practice with Linear filters

Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Slide credit: David Lowe (UBC)

Practice with Linear Filters

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad ?$$

(Note that filter sums to 1)

Slide credit: David Lowe (UBC)

Practice with Linear Filters

Original

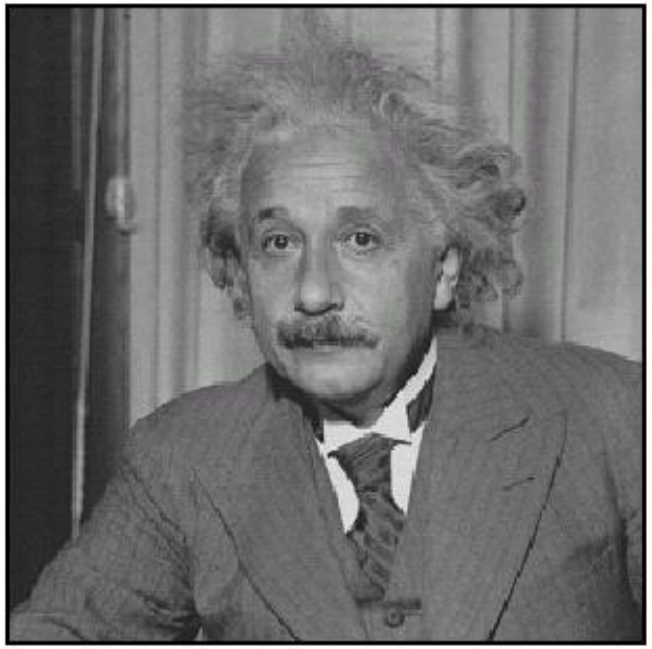
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter

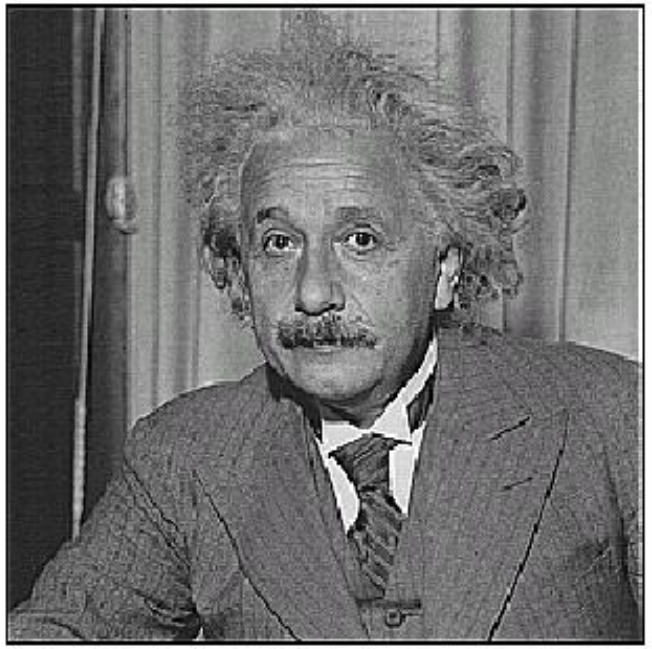
- Accentuates differences with local average

Slide credit: David Lowe (UBC)

Sharpening



before



after

Slide credit: David Lowe (UBC)

Practice with Linear Filters (Example)

```
I = imread('0776.png');

localMean = imboxfilt(I,13);
imshowpair(I,localMean,'montage')

I1 = I - localMean;
imshow(I1)

I2 = I + I1;
imshowpair(I, I2,'montage')
```

Image Sharpening Examples

Image Sharpening Examples

Do not overdo it with sharpening

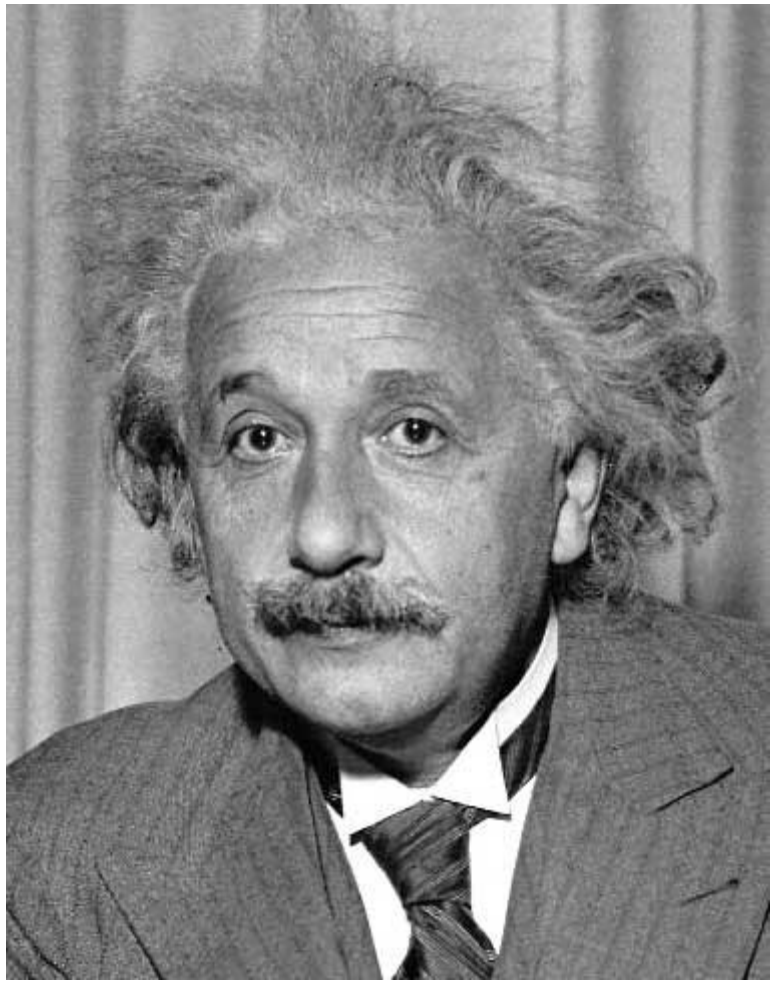
original

sharpened

oversharpened

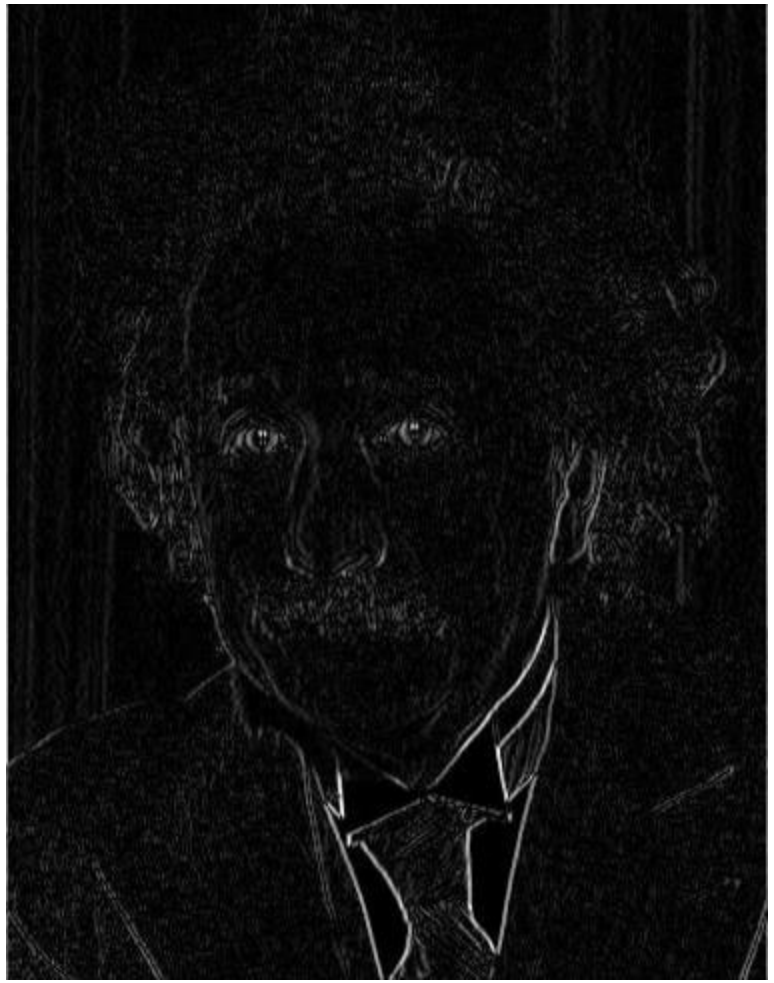
What is wrong in this image?

Other filters



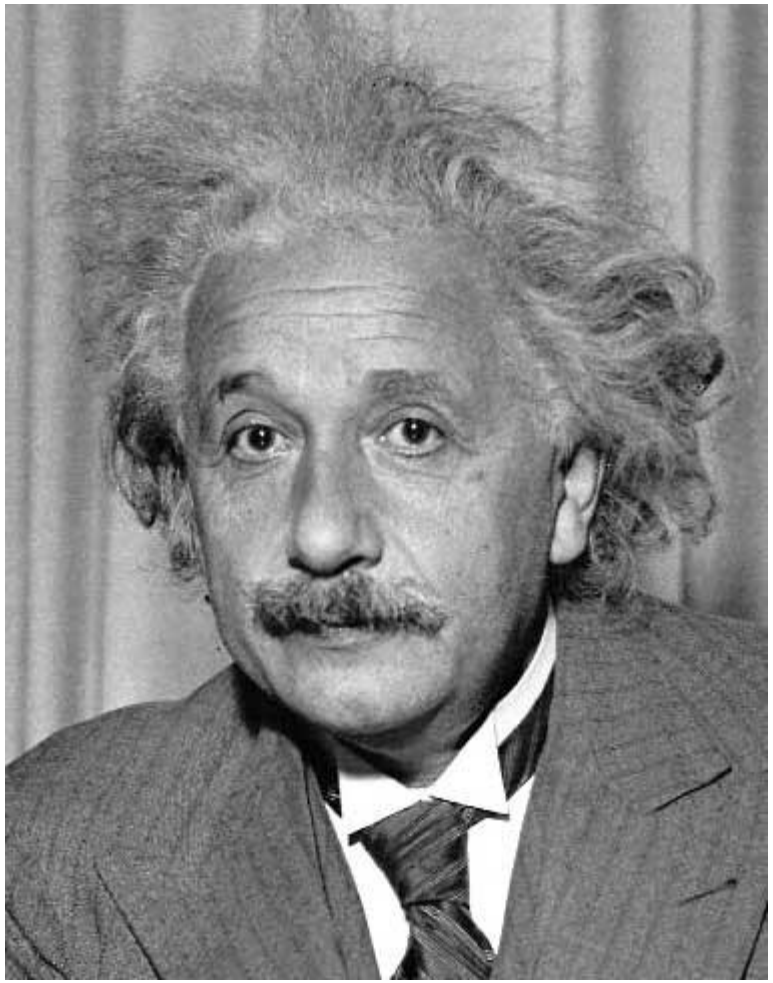
1	0	-1
2	0	-2
1	0	-1

Sobel



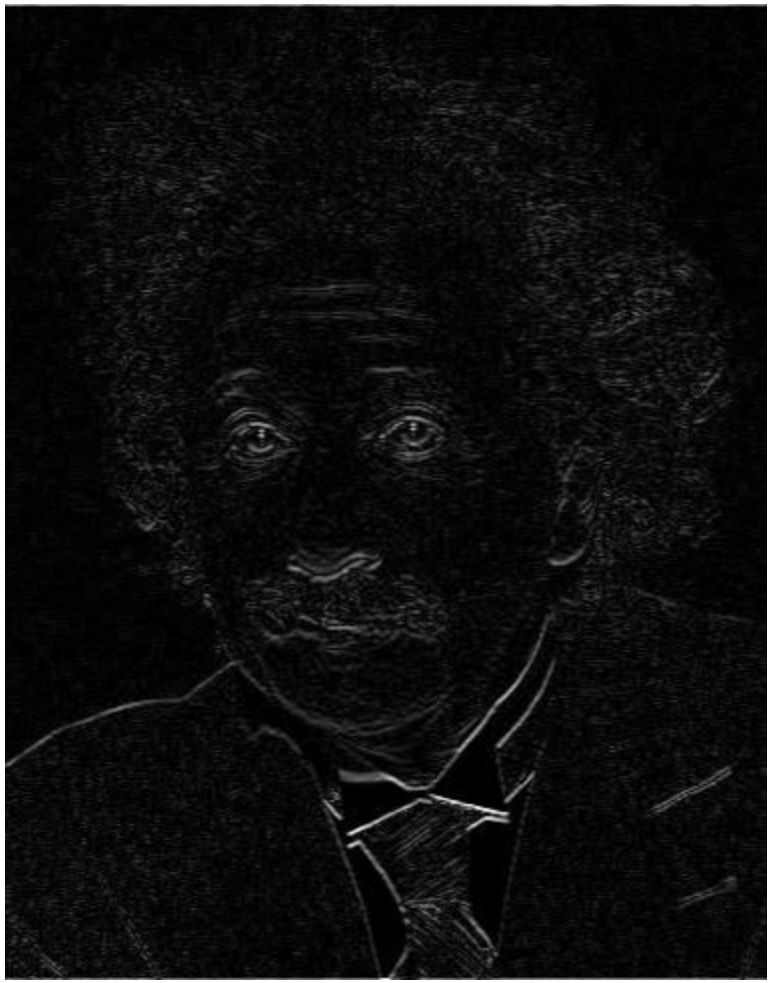
Vertical Edge
(absolute value)

Other Filters



1	2	1
0	0	0
-1	-2	-1

Sobel



Horizontal Edge
(absolute value)

Basic Gradient Filters

Horizontal Gradient

0	0	0
-1	0	1
0	0	0

or

-1	0	1

Vertical Gradient

0	-1	0
0	0	0
0	1	0

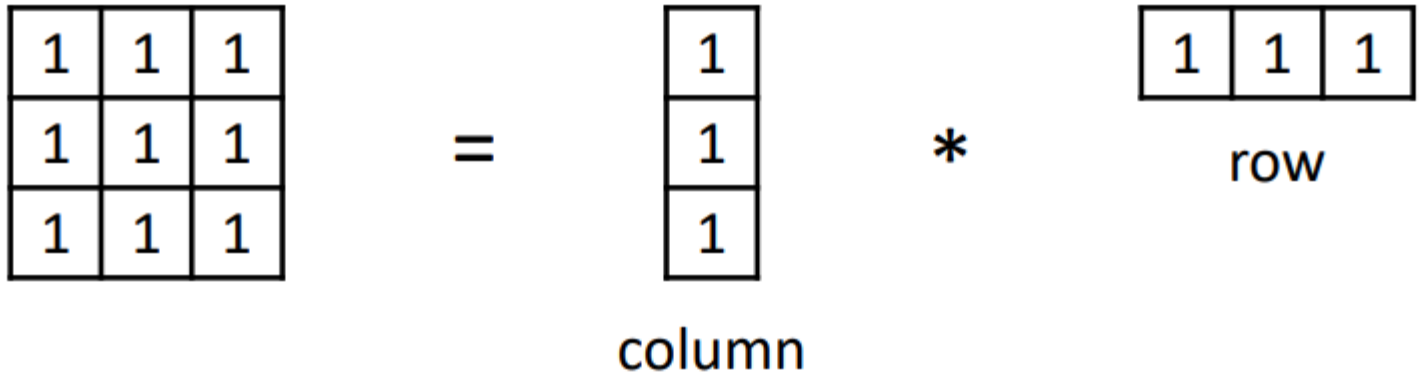
or

-1
0
1

Separable Filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example:
box filter

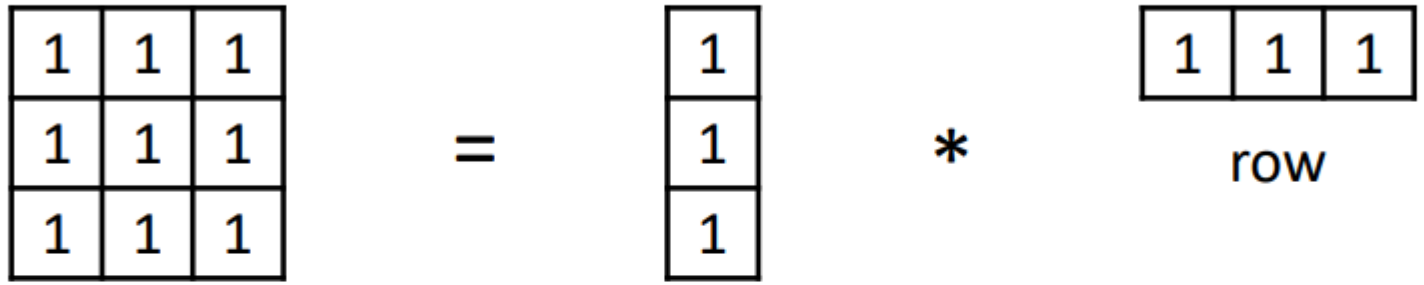


Why is this important?

Separable Filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example:
box filter

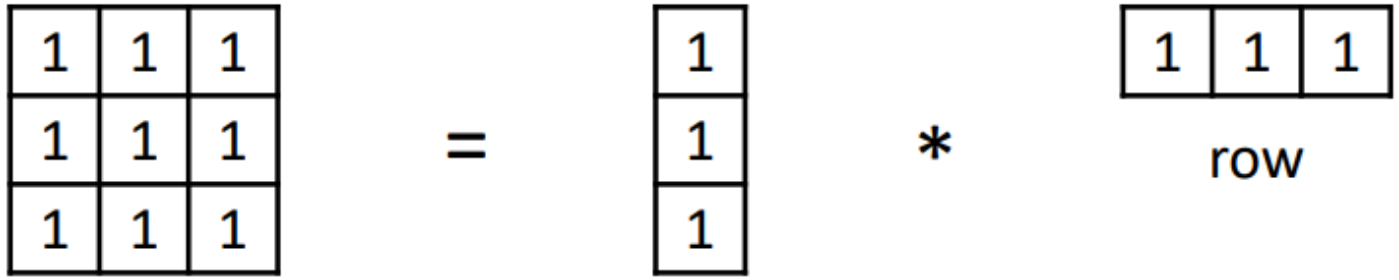


2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

Separable Filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example:
box filter



2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

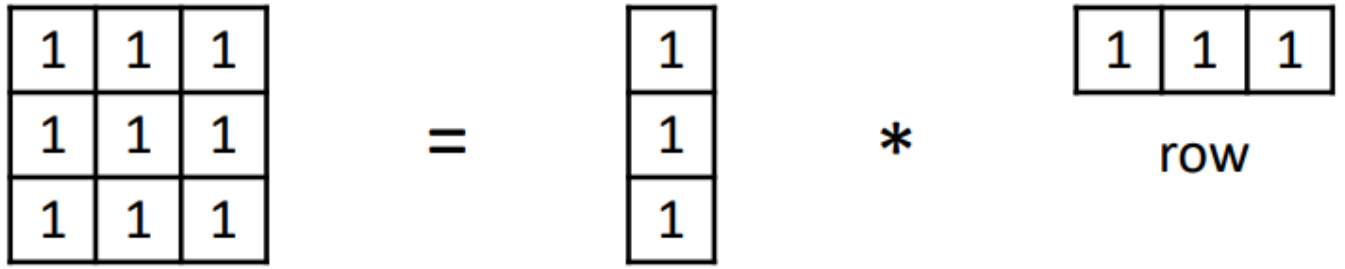
If the image has M x M pixels and the filter kernel has size N x N:

- What is the cost of convolution with a non-separable filter? ⟶ $M^2 \times N^2$

Separable Filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example:
box filter



2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

If the image has M x M pixels and the filter kernel has size N x N:

- What is the cost of convolution with a non-separable filter? ⟶ $M^2 \times N^2$
- What is the cost of convolution with a separable filter? ⟶ $2 \times N \times M^2$

Key Properties of Linear filters

Linearity:

$$\texttt{filter}(f_1 + f_2) = \texttt{filter}(f_1) + \texttt{filter}(f_2)$$

Shift invariance: same behavior regardless of pixel location

$$\texttt{filter}(\texttt{shift}(f)) = \texttt{shift}(\texttt{filter}(f))$$

Any linear, shift-invariant operator can be represented as a convolution

Source: S. Lazebnik

More Properties

- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal
- Associative: $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k (a * b)$
- Identity: unit impulse $e = [0, 0, 1, 0, 0]$,
  $a * e = a$

Source: S. Lazebnik

The Gaussian Filter

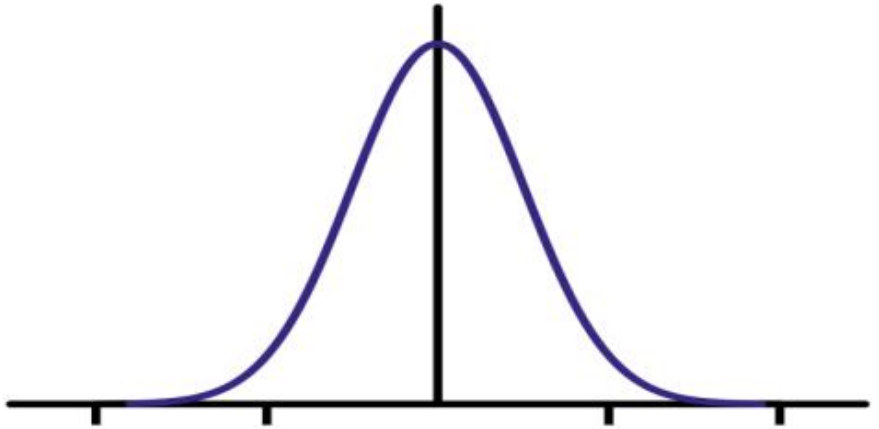
- named (like many other things) after Carl Friedrich Gauss
- kernel values sampled from the 2D Gaussian function:

$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?

usually at 2-3σ



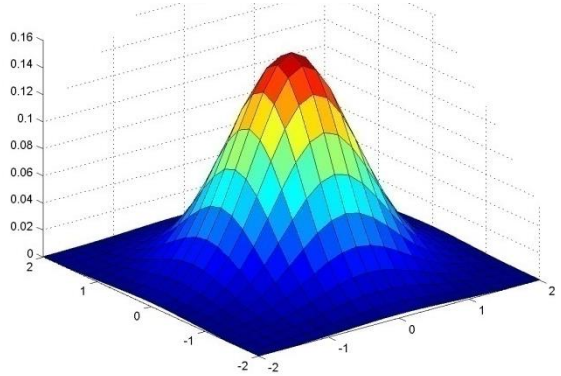
Is this a separable filter? Yes!

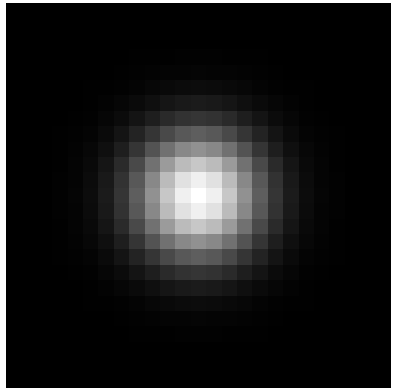
kernel $\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Important Filter: Gaussian

- Spatially-weighted average





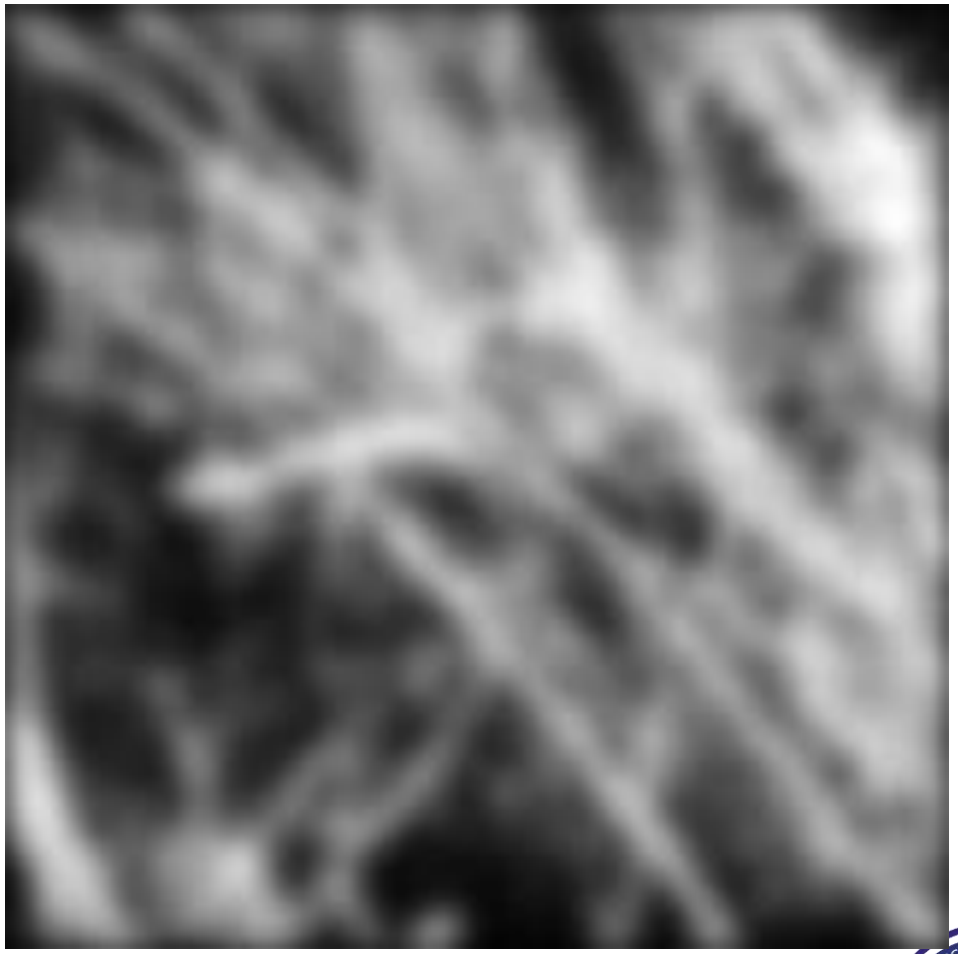
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

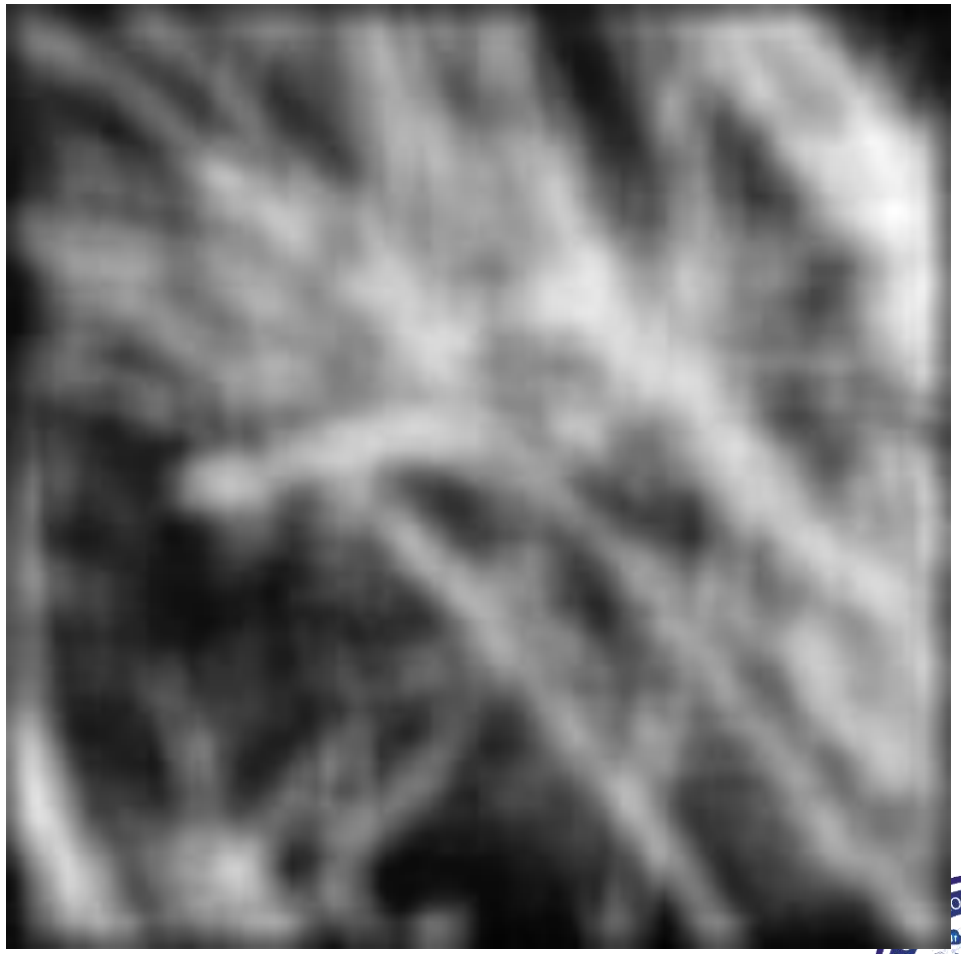
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Slide credit: Christopher Rasmussen

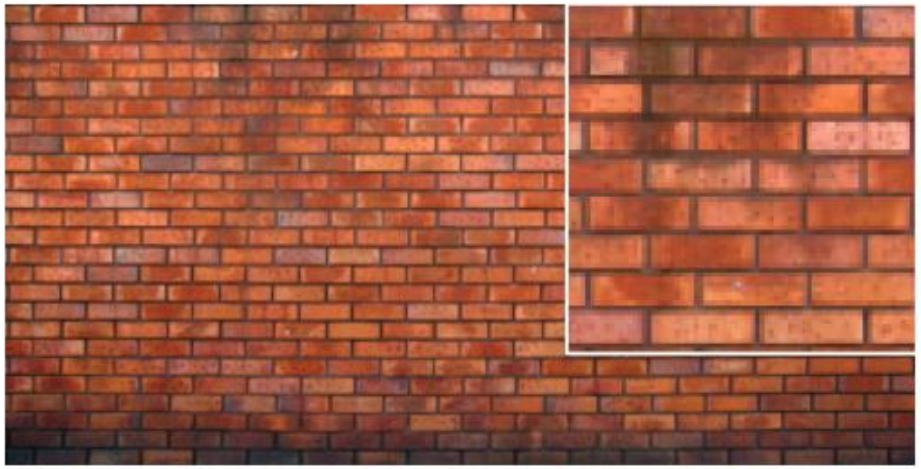
Smoothing with Gaussian Filter



Smoothing with Box Filter

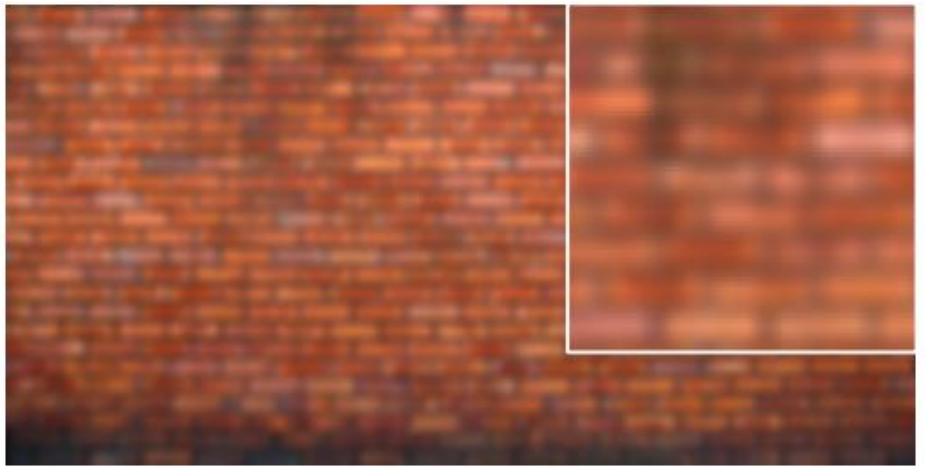


Gaussian vs Box Filtering

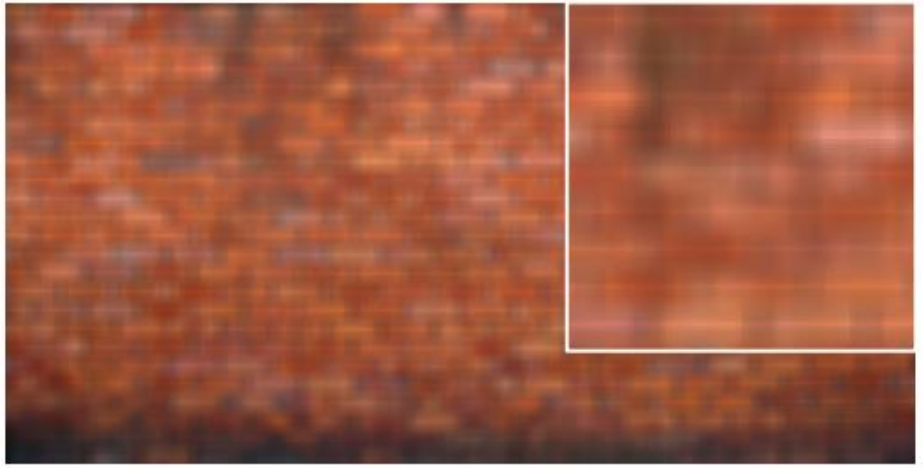


original

Which blur do you like better?



7x7 Gaussian



7x7 box

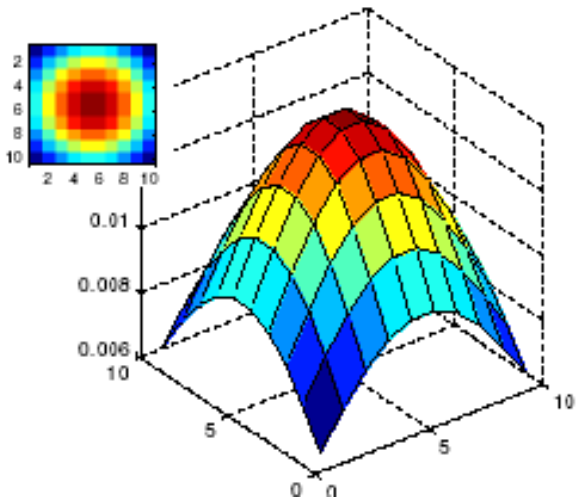
Gaussian Filters

- Remove “high-frequency” components from the image (low-pass filter)
  - Images become more smooth
- Convolution with self is another Gaussian
  - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
  - Factors into product of two 1D Gaussians

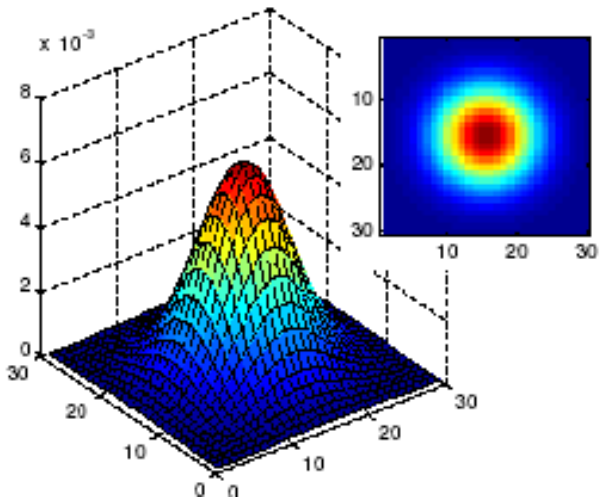
Slide credit: Kristen Grauman

Gaussian Filters

- What parameters matter here?
- **Size** of kernel or mask
  - Note, Gaussian function has infinite support, but discrete filters use finite kernels



σ = 5 with 10 x 10 kernel

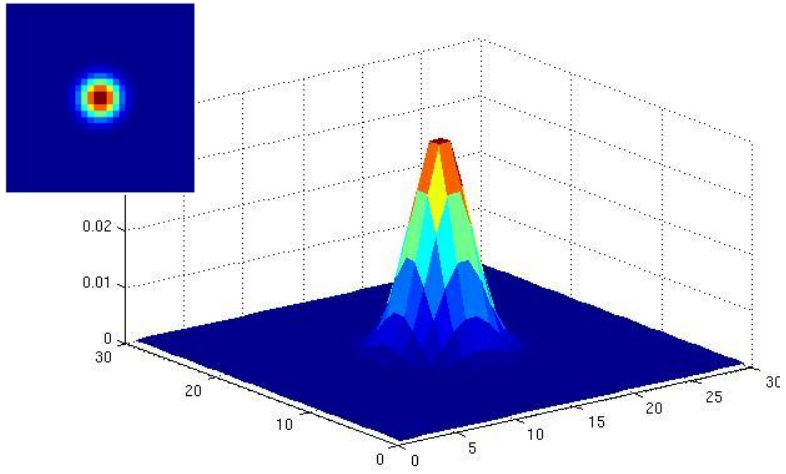


σ = 5 with 30 x 30 kernel

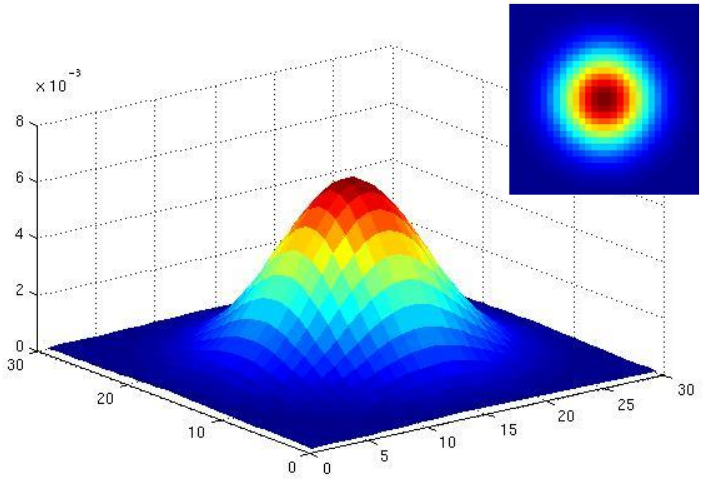
Slide credit: Kristen Grauman

Gaussian Filters

- What parameters matter here?
- **Variance** of Gaussian: determines extent of smoothing



σ = 2 with 30 x 30 kernel



σ = 5 with 30 x 30 kernel

Slide credit: Kristen Grauman

Gaussian Separability Example

2D filtering
(center location only)

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix}$$

The filter factors
into a product of 1D
filters:

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

Perform filtering
along rows:

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix} = \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix}$$

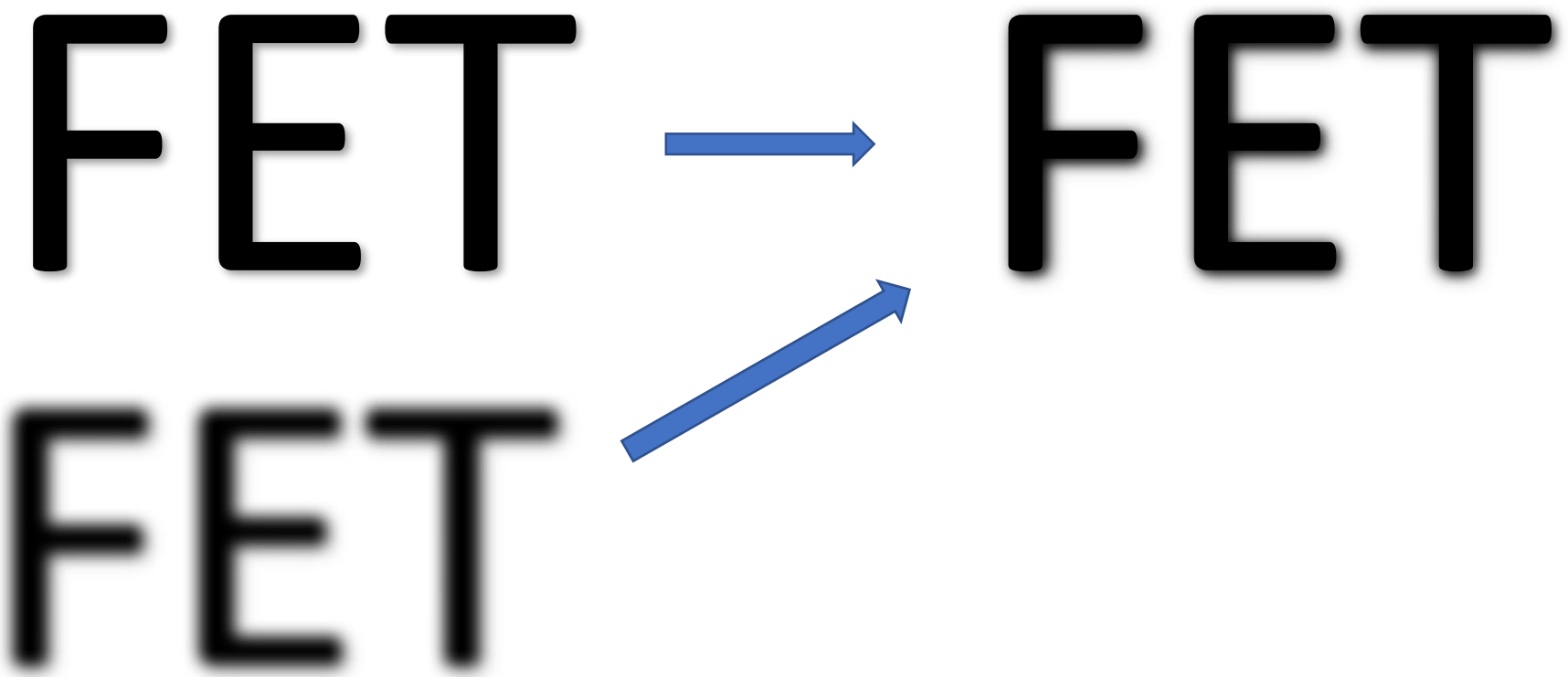
Followed by filtering
along the remaining column:

Source: K. Grauman

Separability

- Why is separability useful in practice?
- Separability means that a 2D convolution can be reduced to two 1D convolutions (one among rows and one among columns)
- What is the complexity of filtering an $n \times n$ image with an $m \times m$ kernel?
  - $O(n^2 m^2)$
- What if the kernel is separable?
  - $O(n^2 m)$

How would you create a soft shadow effect?



Gaussian Blur

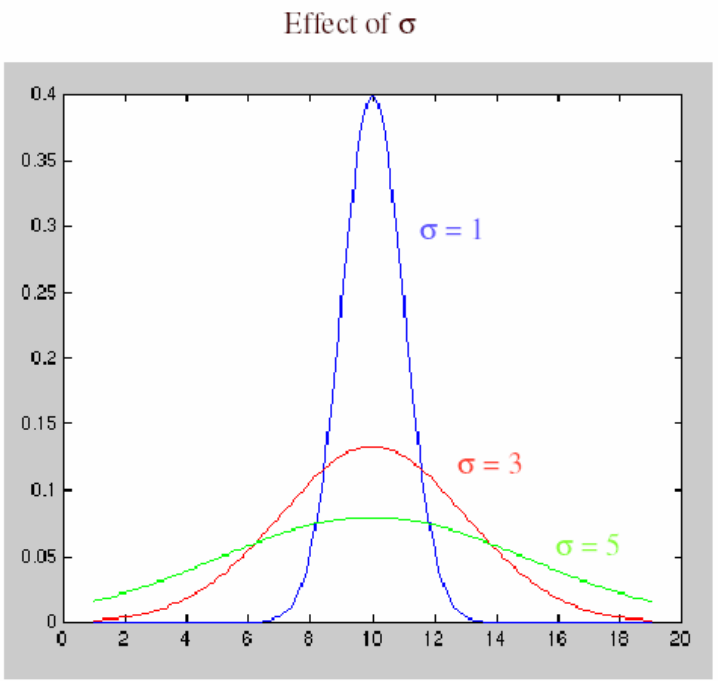
How would you create a soft shadow effect

```
I = imread('FET_text.jpg');
Iblur = imgaussfilt(I,15);
imshow(Iblur)

Ifuse = imfuse(Iblur,I,'blend');
imshowpair(I,Ifuse,'montage')
```

How big should the filter be?

- Values at edges should be near zero ← important!
- Rule of thumb for Gaussian: set filter half-width to about 3 σ



Practical Matters

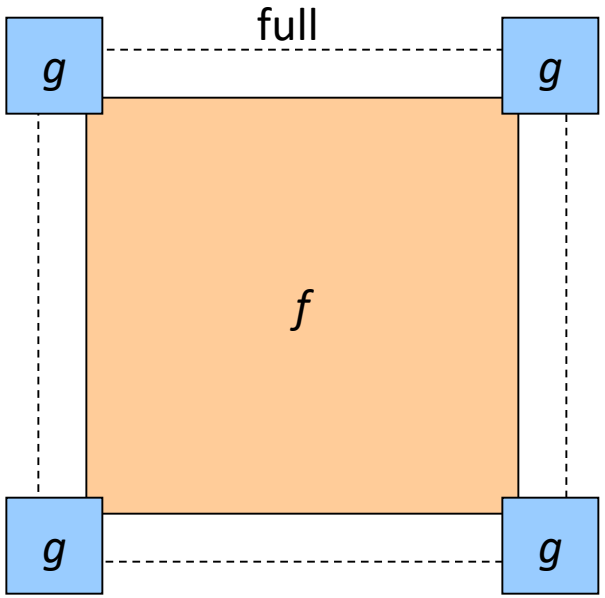
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge

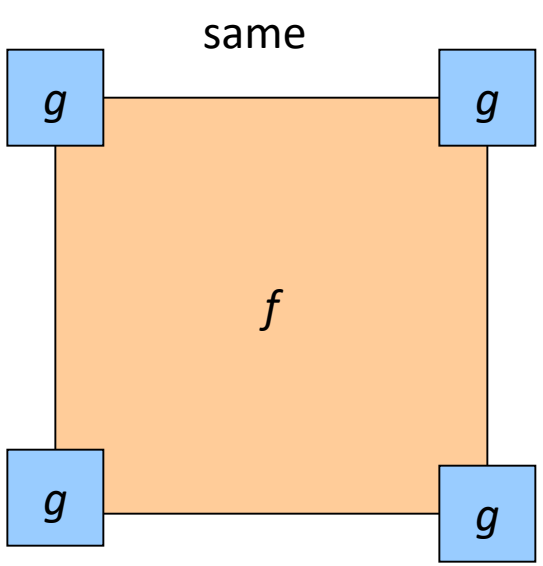
Practical Matters

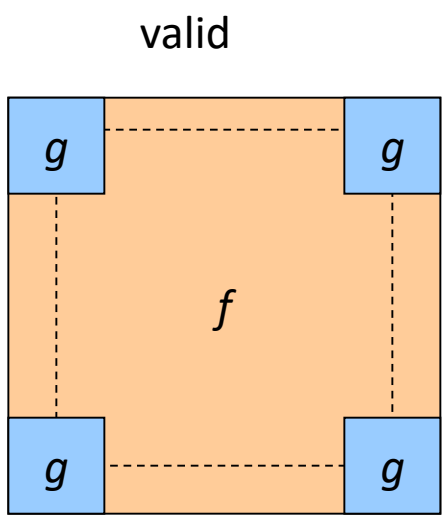
- methods (MATLAB):
  - clip filter (black): `imfilter(f, g, 0)`
  - wrap around: `imfilter(f, g, 'circular')`
  - copy edge: `imfilter(f, g, 'replicate')`
  - reflect across edge: `imfilter(f, g, 'symmetric')`

Practical Matters

- What is the size of the output?
- MATLAB: filter2(g, f, *shape*)
  - *shape* = ‘full’: output size is sum of sizes of f and g
  - *shape* = ‘same’: output size is same as f
  - *shape* = ‘valid’: output size is difference of sizes of f and g





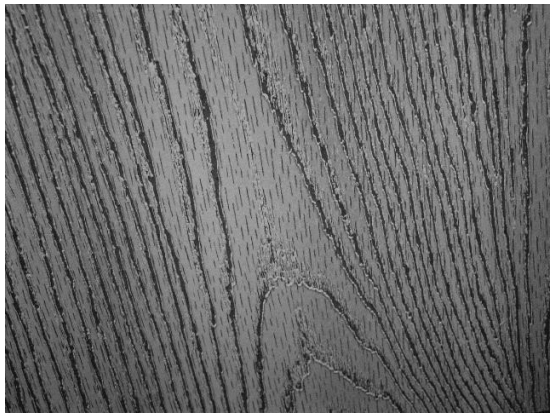
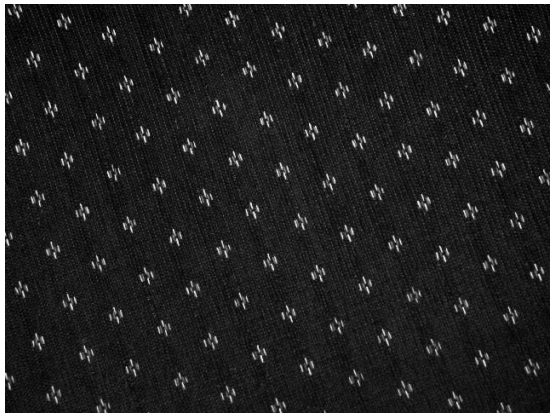


Source: S. Lazebnik

Application: Representing Texture

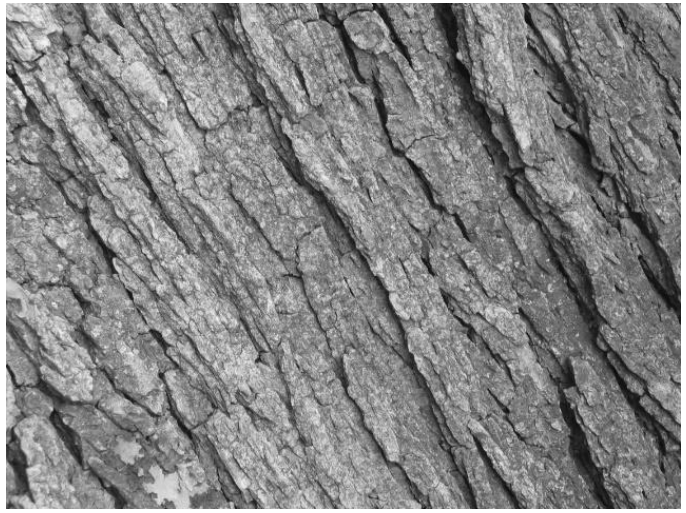
Source: Forsyth

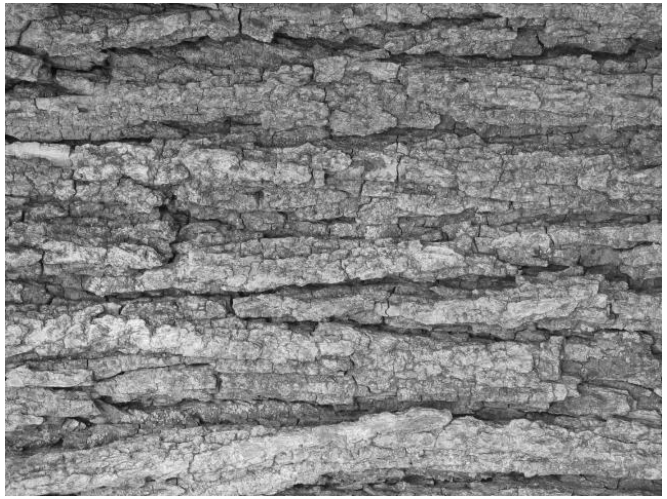
Texture and Material

http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/

Texture and Orientation





http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/

Texture and Scale

http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/

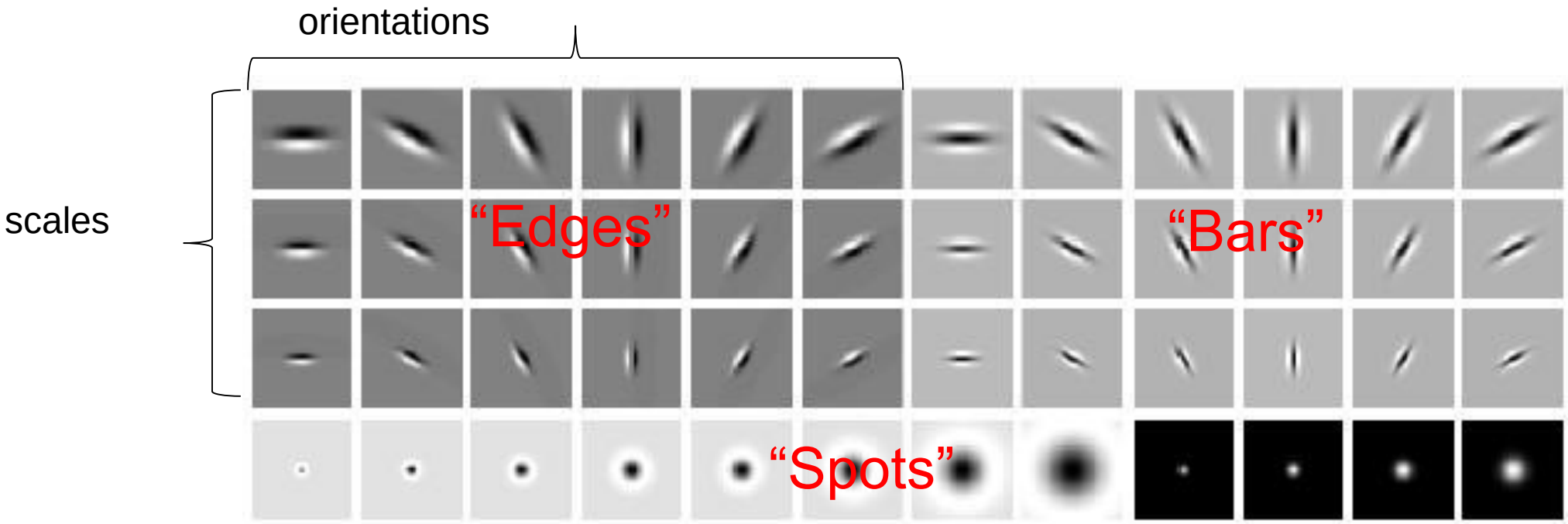
What is texture?

Regular or stochastic patterns caused by bumps, grooves, and/or markings

How can we represent Texture?

- Compute responses of blobs and edges at various orientations and scales

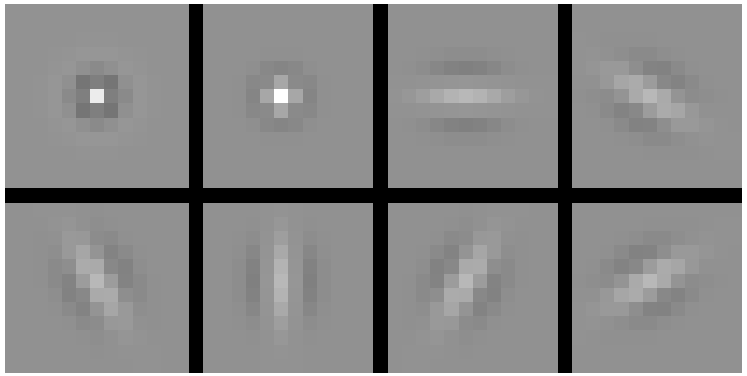
Overcomplete representation: Filter banks

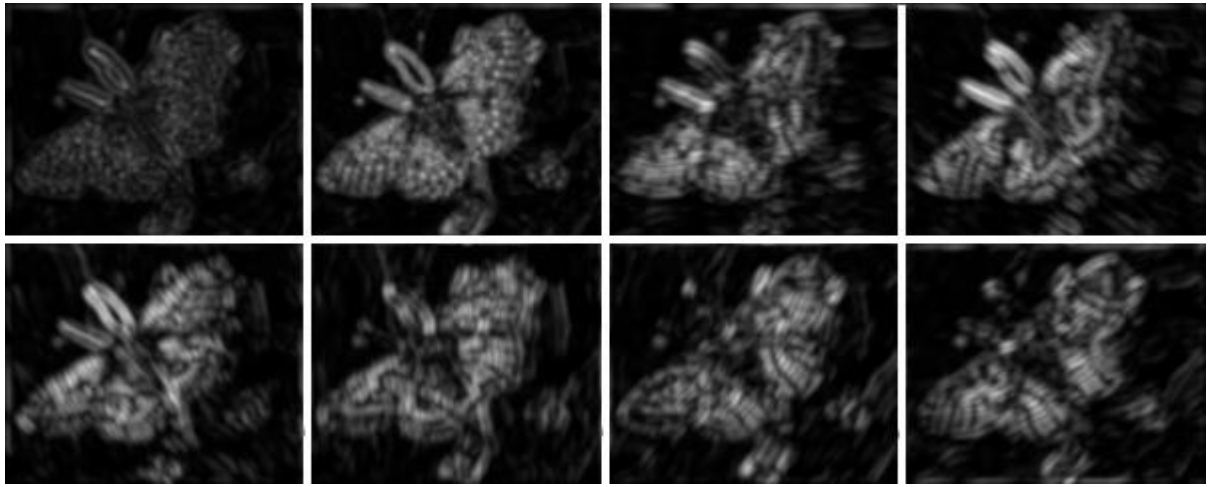


Code for filter banks: *www.robots.ox.ac.uk/~vgg/research/texclass/filters.html*

Filter banks

- Process image with each filter and keep responses (or squared/abs responses)





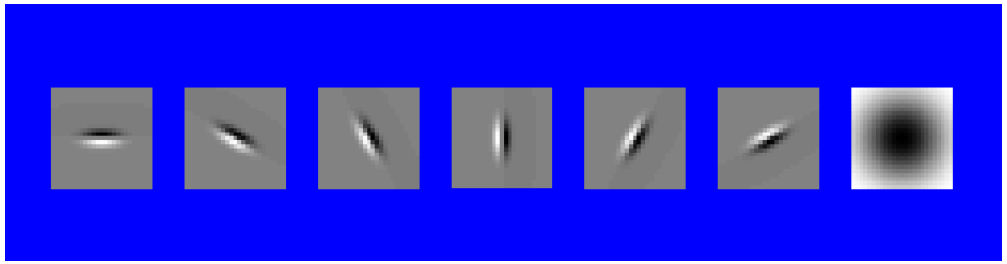
How can we represent Texture?

- Measure responses of blobs and edges at various orientations and scales

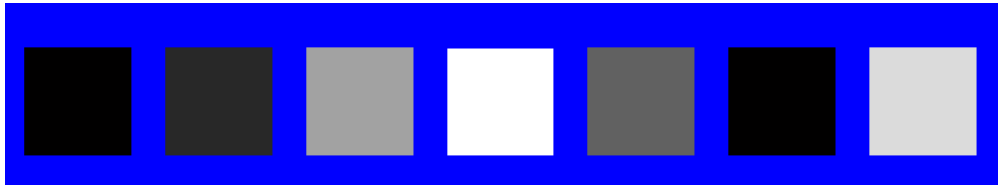
- Idea 1: Record simple statistics (e.g., mean, std.) of absolute filter responses

Can you match the texture to the response?

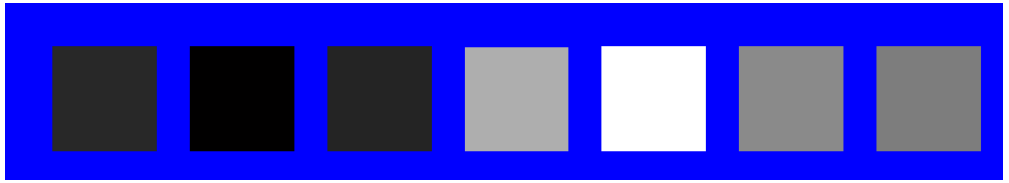
Filters



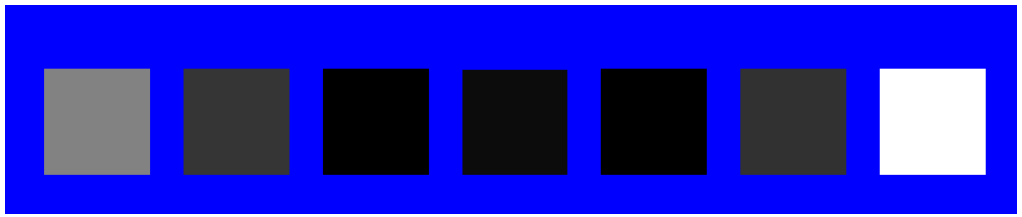
1



2

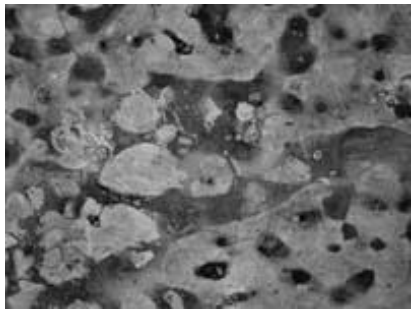


3



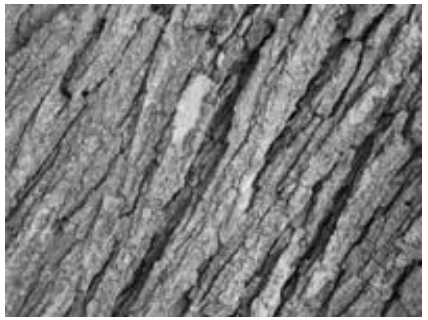
Mean abs responses

A

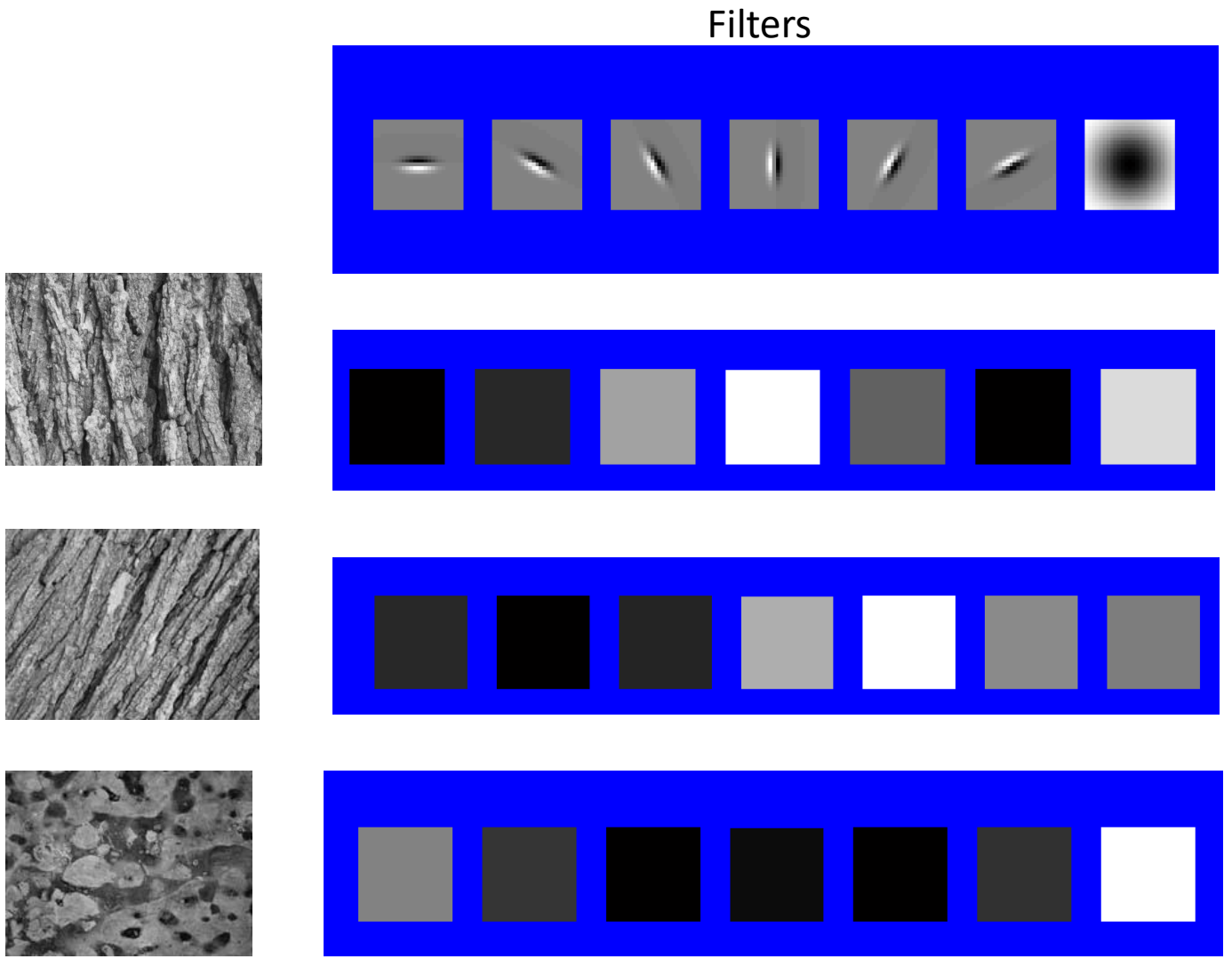


B

C

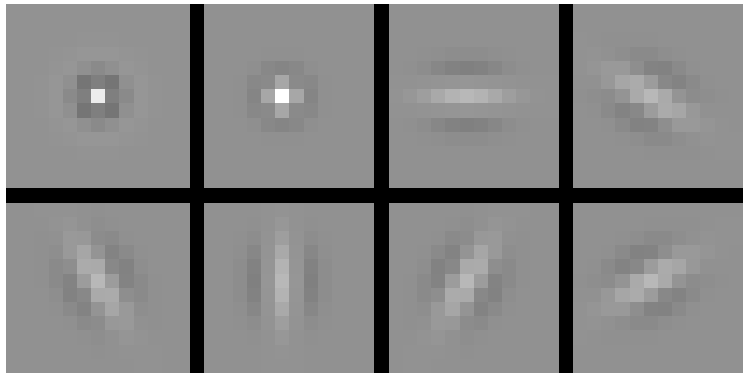


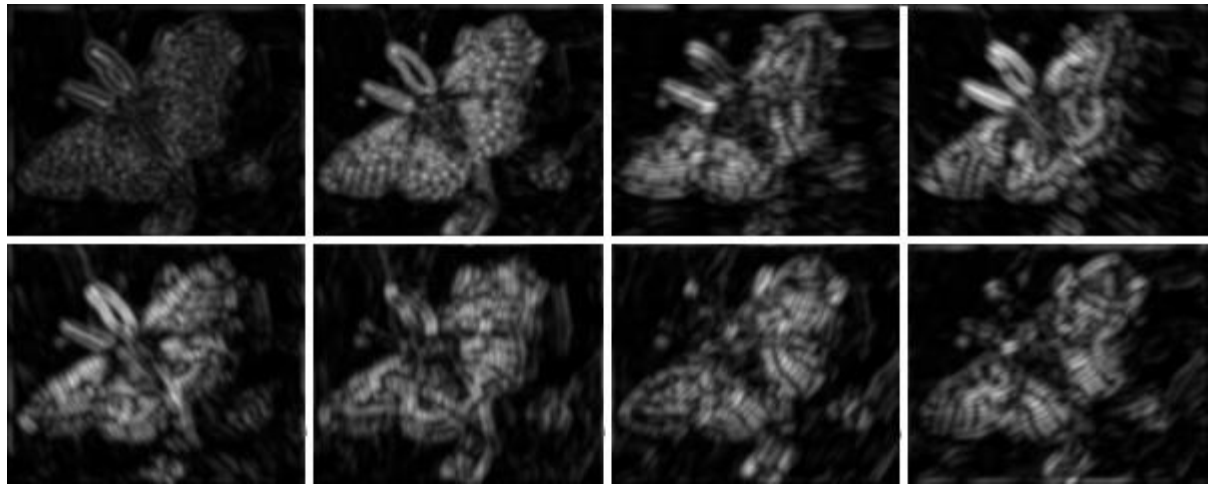
Representing texture by mean absolute response



Representing Texture

- Idea 2: take vectors of filter responses at each pixel and cluster them, then take histograms





Denoising and Nonlinear Image filtering

Original

Salt and pepper noise

Impulse noise

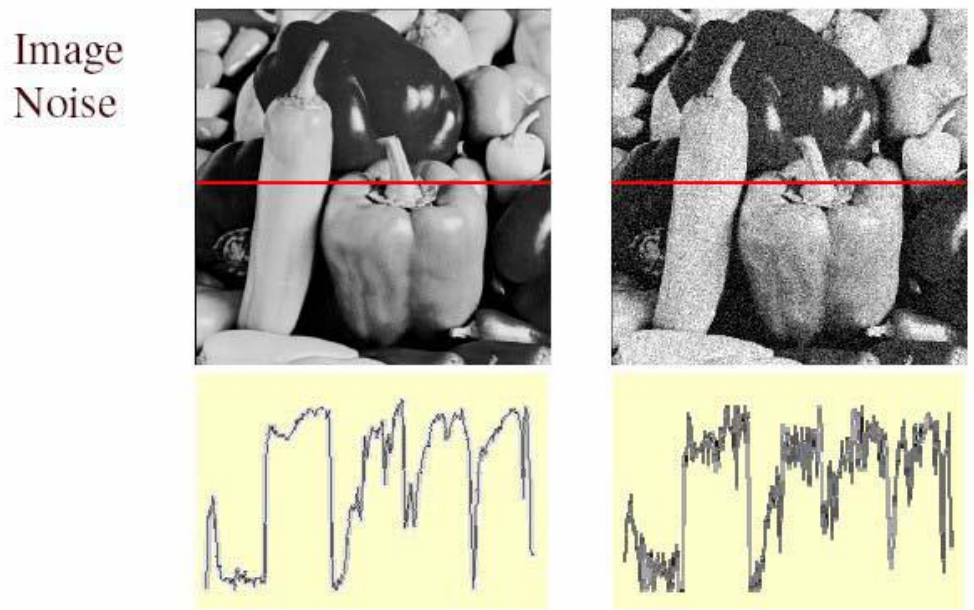
Gaussian noise

- **Salt and pepper noise**: contains random occurrences of black and white pixels
- **Impulse noise:** contains random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Source: S. Seitz

Gaussian Noise

- Mathematical model: sum of many independent factors
- Good for small standard deviations
- Assumption: independent, zero-mean noise

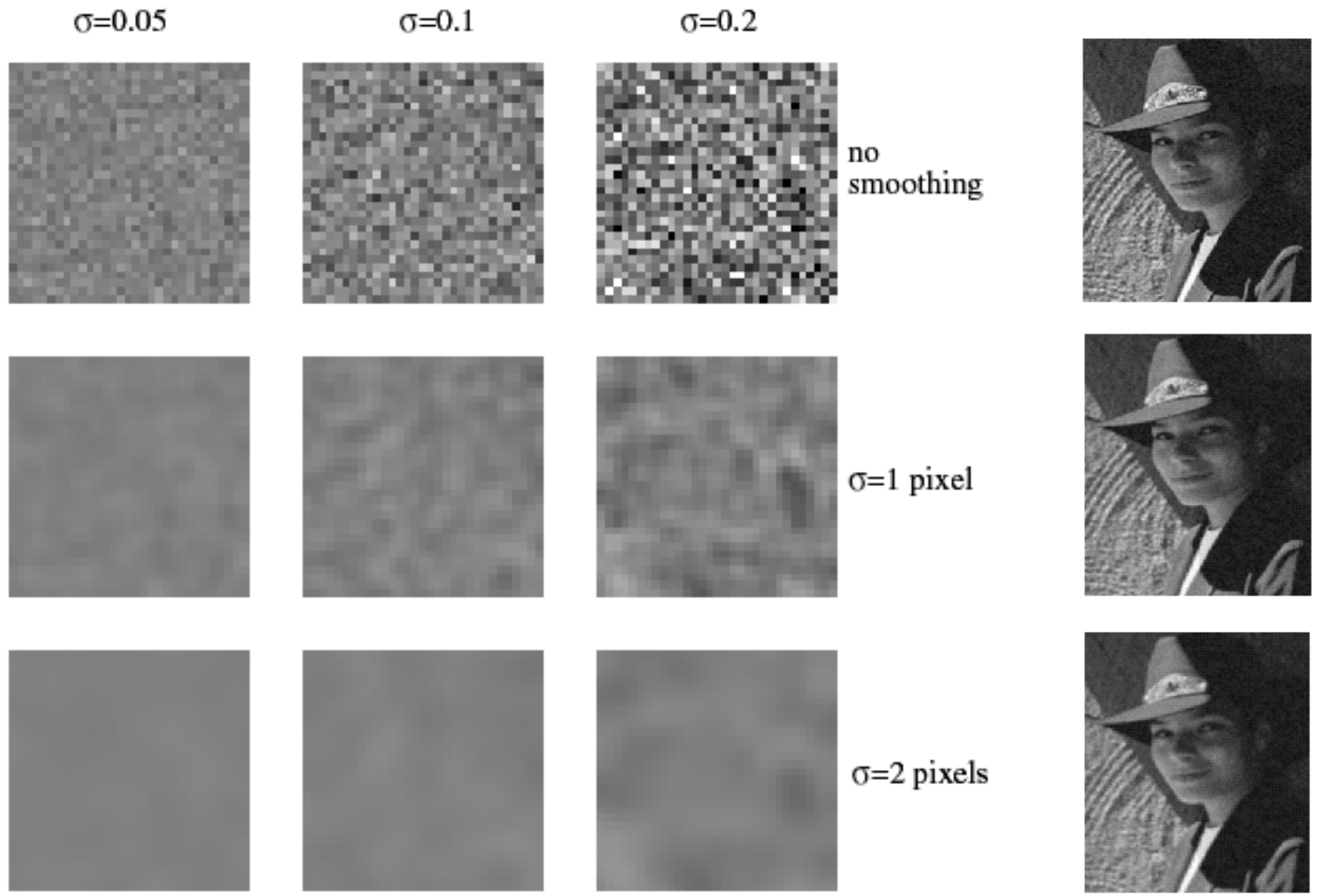


$$f(x,y) = \overbrace{\bar{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$

Source: M. Hebert

Reducing Gaussian Noise



Smoothing with larger standard deviations suppresses noise, but also blurs the image

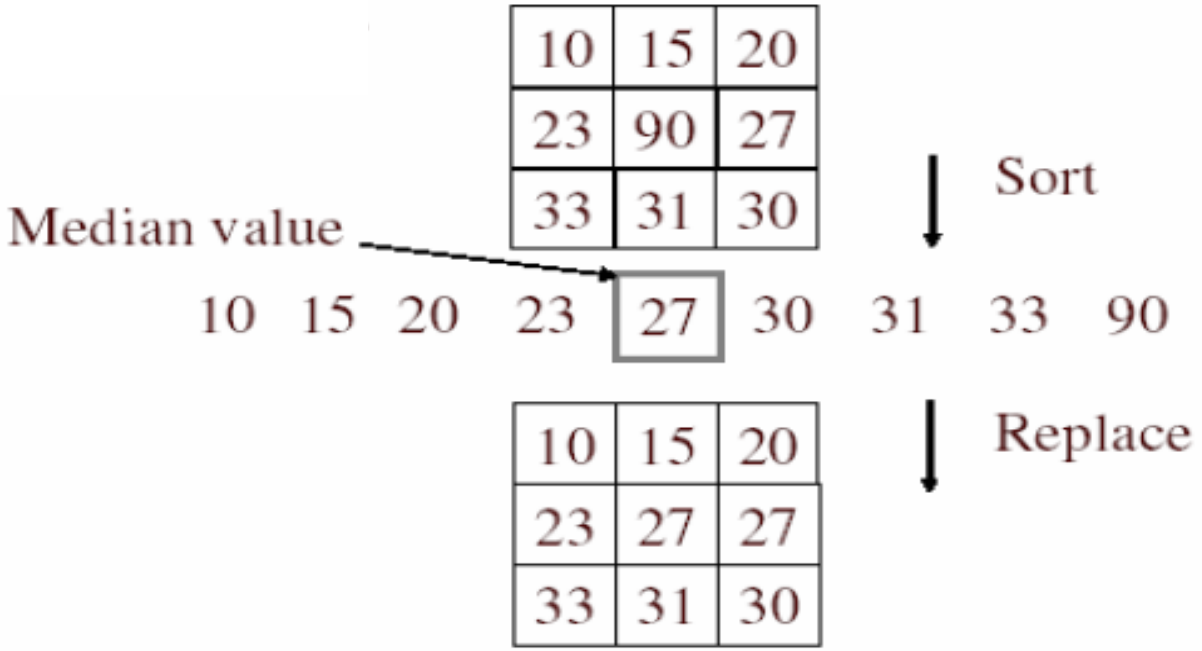
Reducing Salt-and-Pepper noise

3x3 5x5 7x7

- What's wrong with the results?

Alternative Idea: Median Filtering

- A **median filter** operates over a window by selecting the median intensity in the window

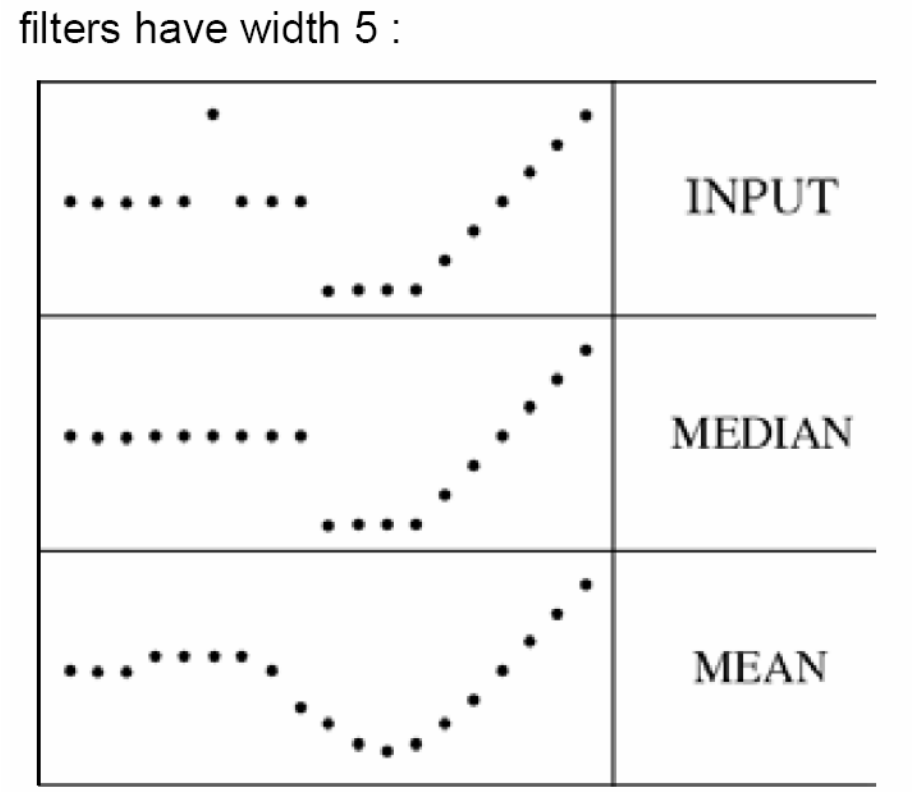


- Is median filtering linear?

Source: K. Grauman

Median Filter

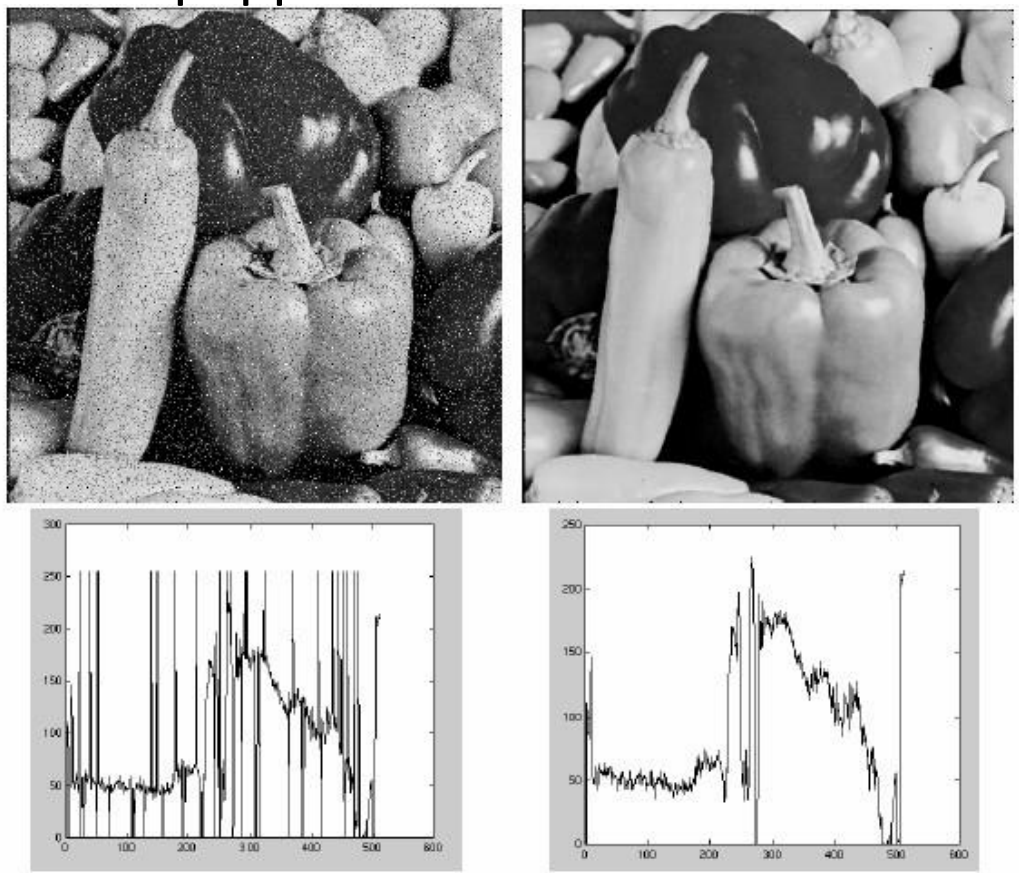
- What advantage does median filtering have over Gaussian filtering?
  - Robustness to outliers



Source: K. Grauman

Median Filter

Salt-and-pepper noise Median filtered



- MATLAB: medfilt2(image, [h w])

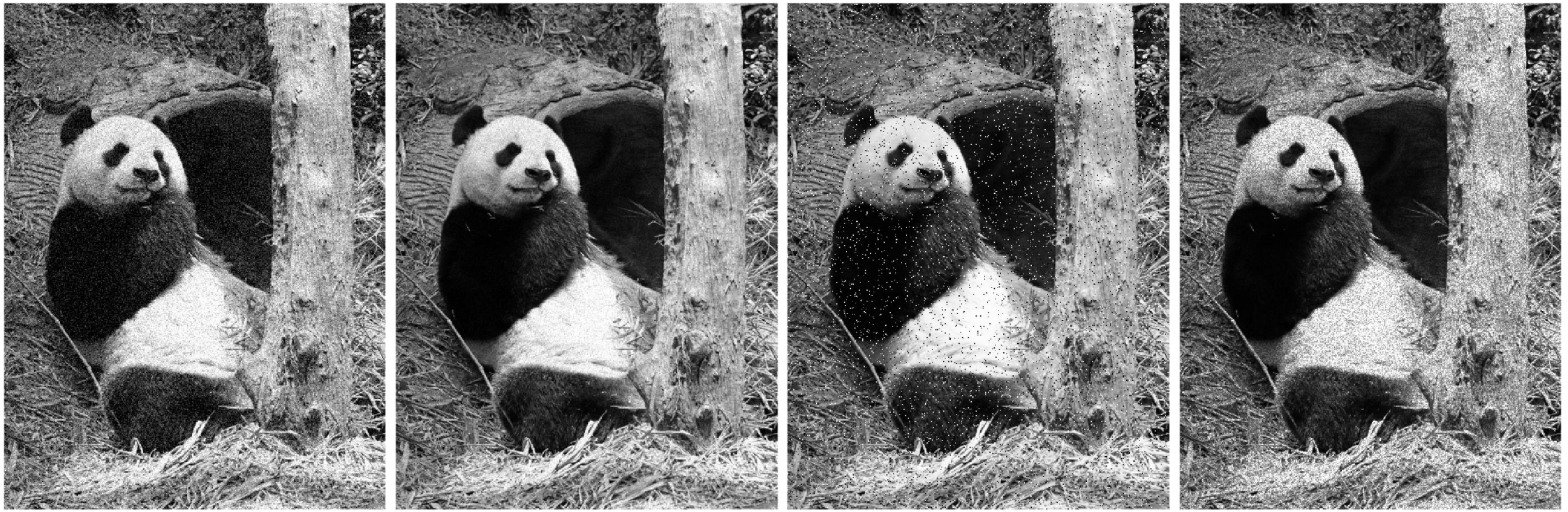
Source: M. Hebert

Gaussian vs Median Filtering

	3x3	5x5	7x7
Gaussian			
Median			

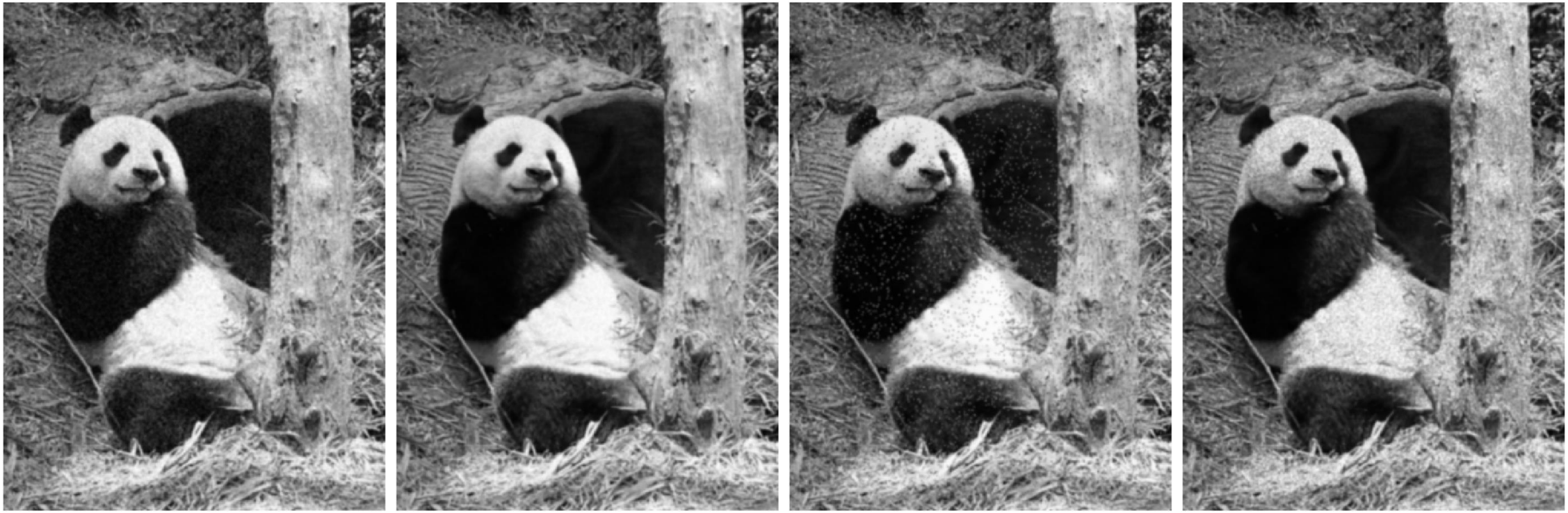
Adding noise in images

```
%Adding Noise in the image
I = imread('0420.png');
imshow(imnoise(I,'gaussian'))
imshow(imnoise(I,'poisson'))
imshow(imnoise(I,'salt & pepper'))
imshow(imnoise(I,'speckle'))
```



Applying Gaussian Filter

```
%Applying Gaussian Filter
I1 = imgaussfilt(imnoise(I,'gaussian'));
I2 = imgaussfilt(imnoise(I,'poisson'));
I3 = imgaussfilt(imnoise(I,'salt & pepper'));
I4 = imgaussfilt(imnoise(I,'speckle'));
```



Applying Median Filter

```
%Applying Median Filtering
I5 = medfilt2(imnoise(I,'gaussian'),[3 3]);
I6 = medfilt2(imnoise(I,'poisson'),[3 3]);
I7 = medfilt2(imnoise(I,'salt & pepper'),[3 3]);
I8 = medfilt2(imnoise(I,'speckle'),[3 3]);
```

Other non-linear filters

- Weighted median (pixels further from center count less)
- Clipped mean (average, ignoring few brightest and darkest pixels)
- Bilateral filtering (weight by spatial distance *and* intensity difference)

Bilateral filtering

Bilateral Filters

- Edge preserving: weights similar pixels more

spatial similarity (e.g., intensity)

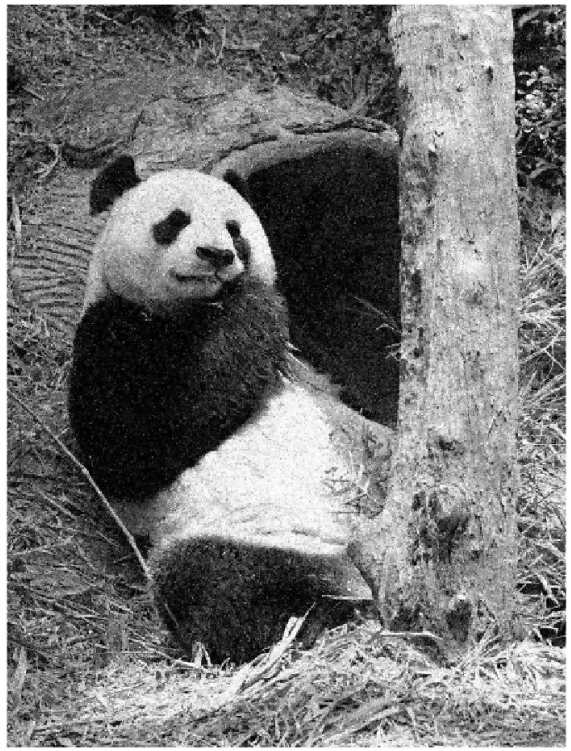
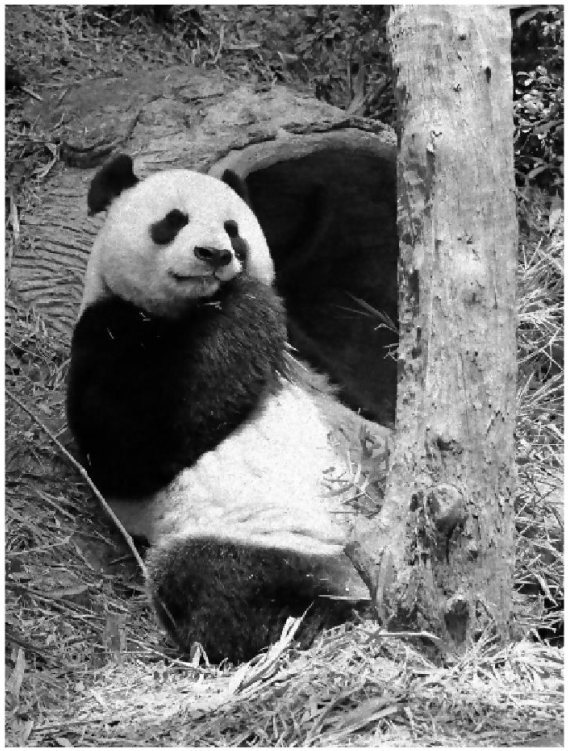
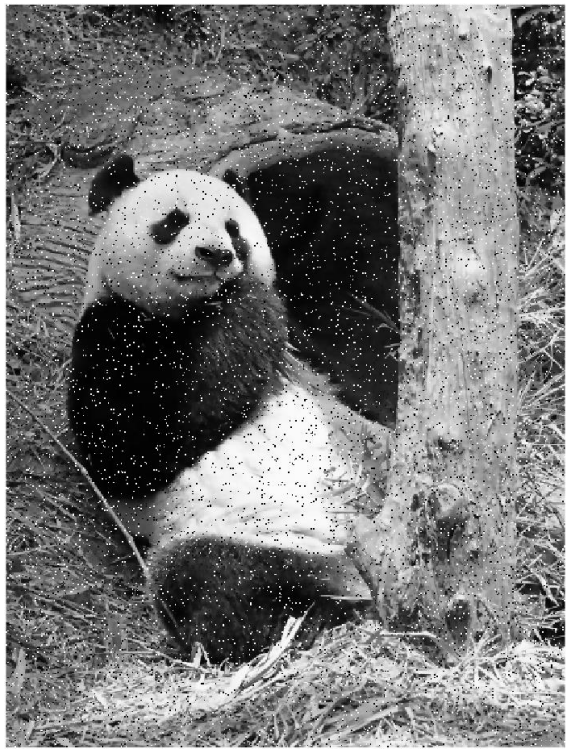
$$I_{\mathbf{p}}^{\mathrm{b}} = \frac{1}{W_{\mathbf{p}}^{\mathrm{b}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

$$\text{with} \quad W_{\mathbf{p}}^{\mathrm{b}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)$$

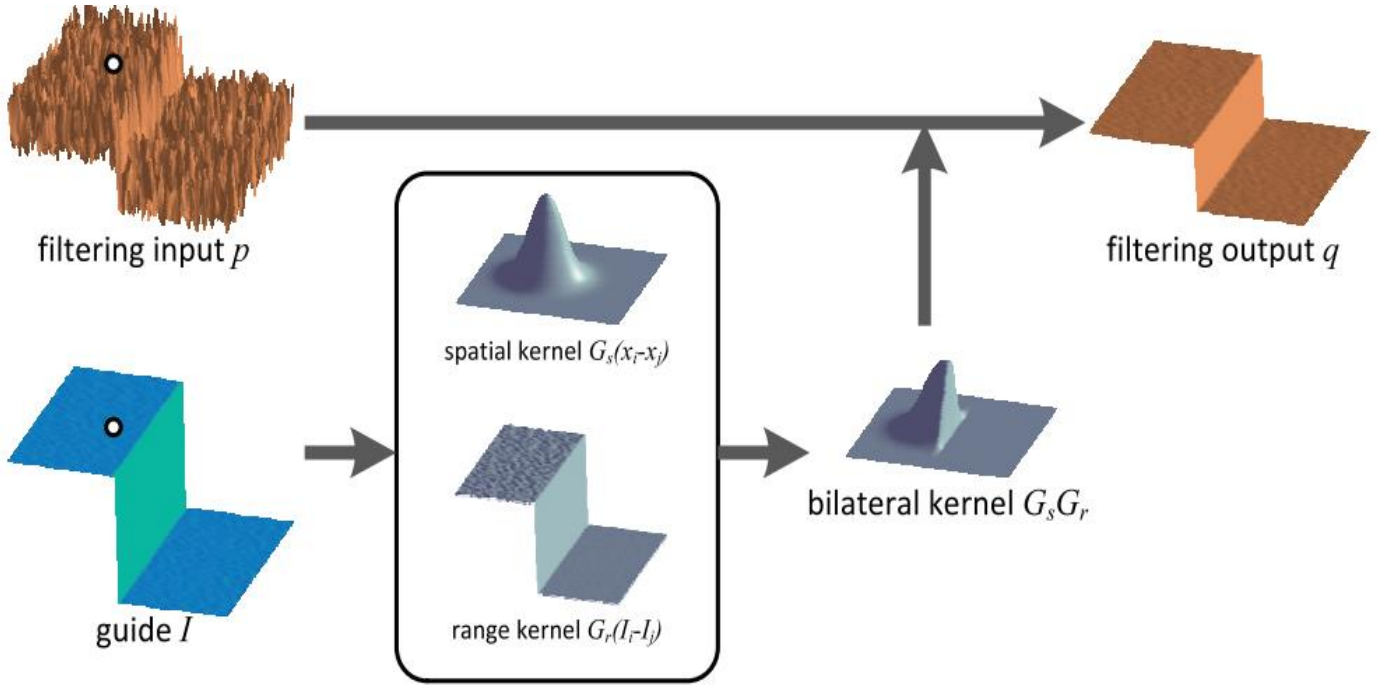
Carlo Tomasi, Roberto Manduchi, Bilateral Filtering for Gray and Color Images, ICCV, 1998.

Applying Bilateral Filter

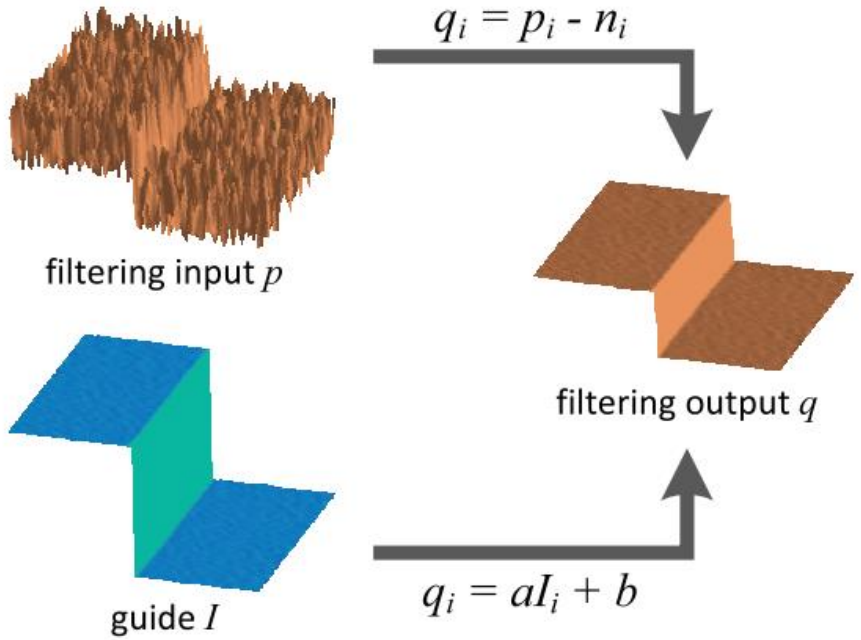
```
%Applying Bilateral Filtering
I9 = imbilatfilt(imnoise(I,'gaussian'),30);
I10 = imbilatfilt(imnoise(I,'poisson'),30);
I11 = imbilatfilt(imnoise(I,'salt & pepper'),30);
I12 = imbilatfilt(imnoise(I,'speckle'),30);
```


Guided Image Filters



Bilateral filters



Guided filters

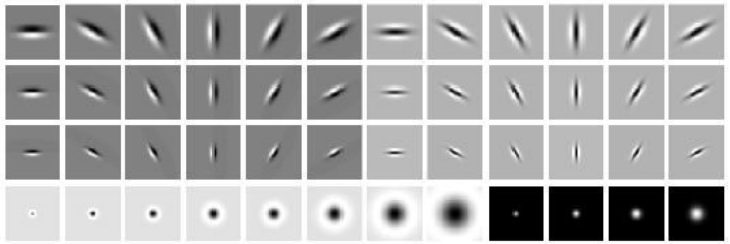
```
B = imguidedfilter(A,G);
```

Kaiming He, Jian Sun, Xiaou Tang, Guided Image Filtering. PAMI 2013

Things to remember

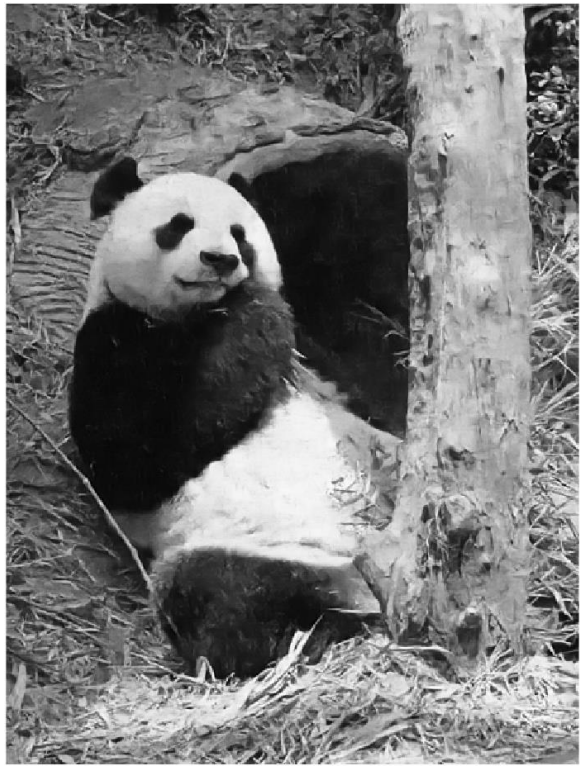
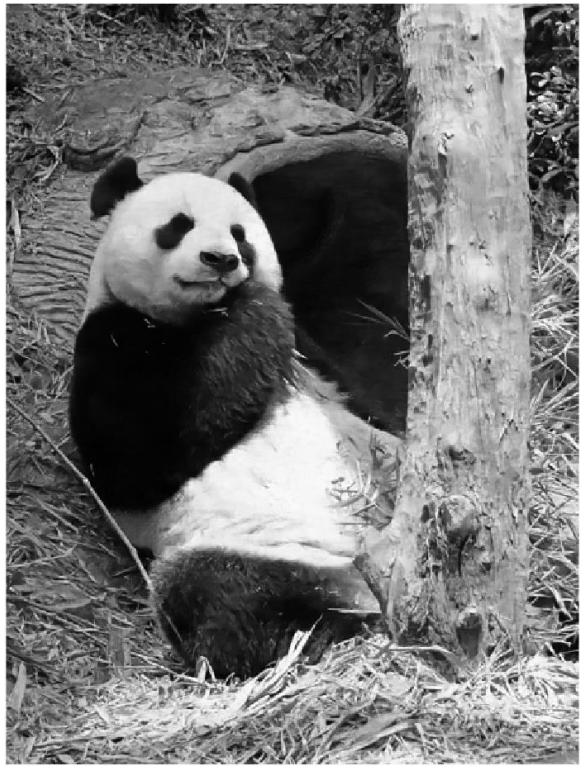
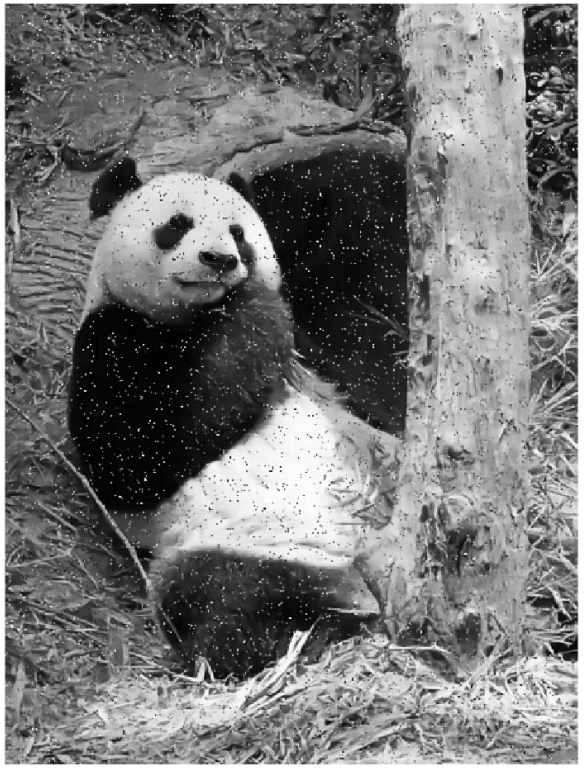
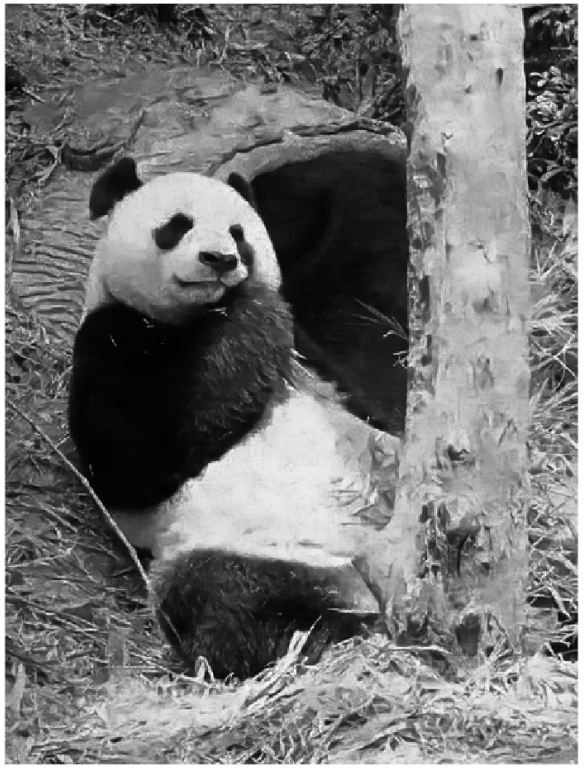
- Linear filtering is sum of dot product at each position
  - Can smooth, sharpen, translate (among many other uses)
- Gaussian filters
  - Low pass filters, separability, variance
- Attend to details:
  - filter size, extrapolation, cropping
- Application: representing textures
- Noise models and nonlinear image filters

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

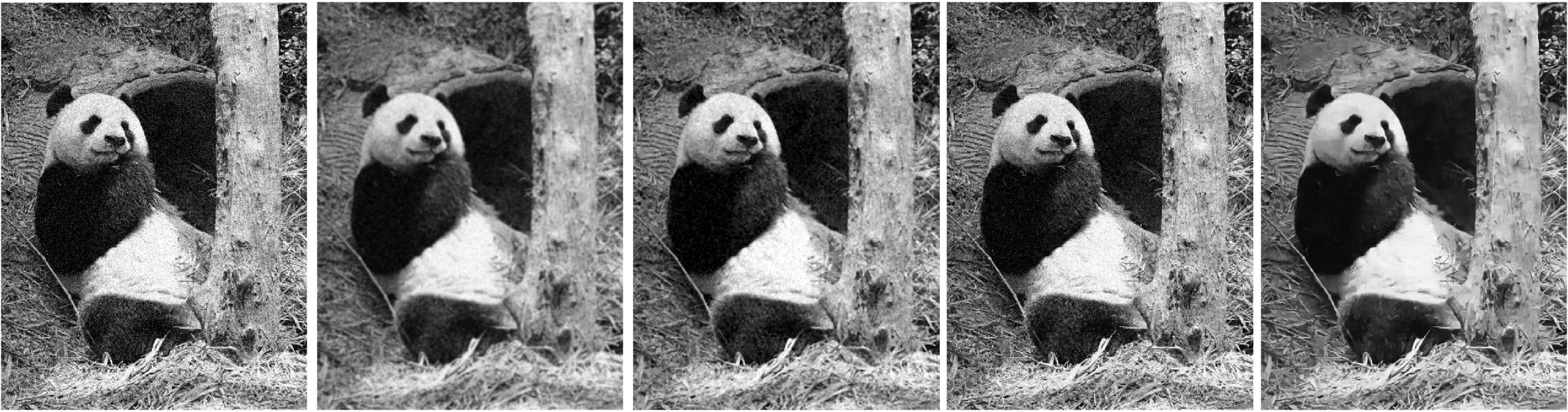


Bonus: Image Denoising using simple Deep learning technique

```
%Denoising using Deep Learning
net = denoisingNetwork('DnCNN');
I13 = denoiseImage(imnoise(I,'gaussian'),net);
I14 = denoiseImage(imnoise(I,'poisson'),net);
I15 = denoiseImage(imnoise(I,'salt & pepper'),net);
I16 = denoiseImage(imnoise(I,'speckle'),net);
```

Comparison of all filters on Gaussian Noise

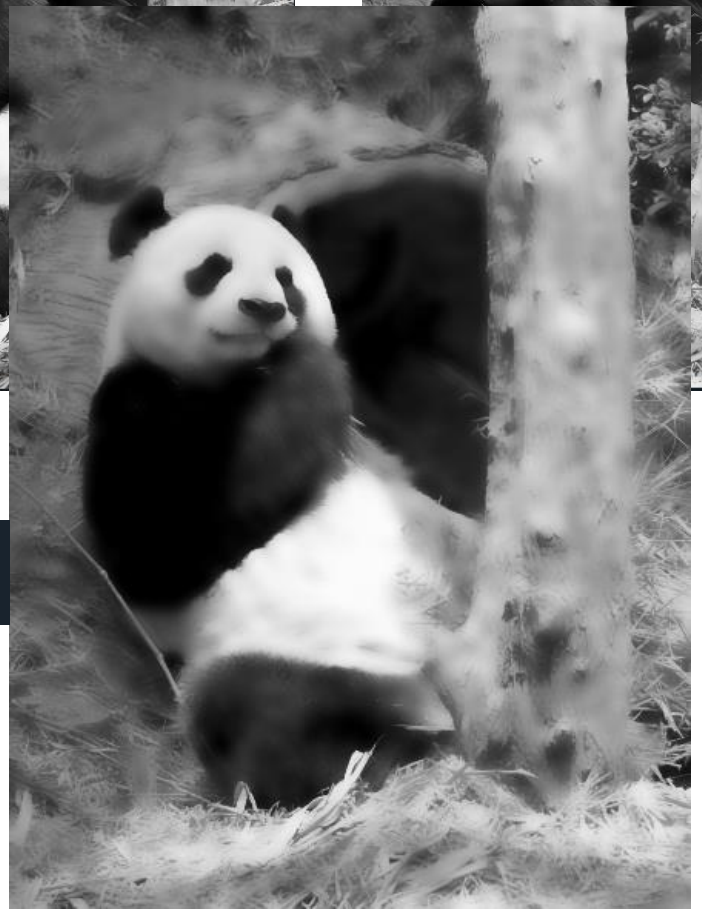


Comparison of all filters on Poisson Noise

Comparison of all filters on Salt and Pepper Noise

Comparison of all filters on Speckle noise

Bonus: Bilateral Filtering is used for image cartoonization (basic)



```
bilateral = cv2.bilateralFilter(src, 15, 75, 75)
cv2.imshow("Bilateral Filtering",bilateral)
cv2.waitKey(0)
```