



COMPUTER VISION

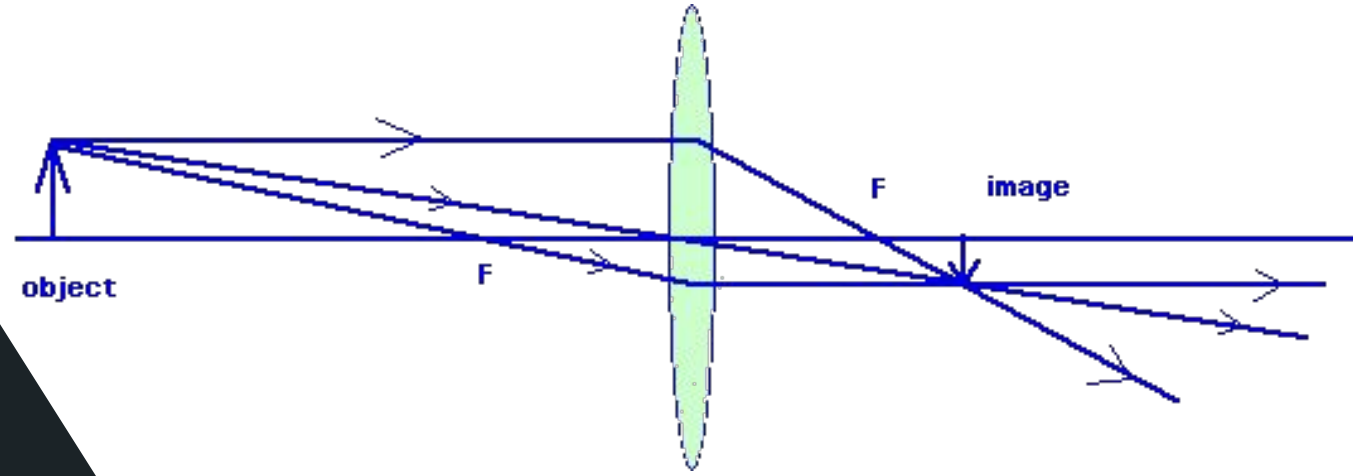
...

2018

Image Forming and Filtering

Image Formation

A Computer Vision Perspective



(Not this stuff xP)

Image Forming

- Silver Halide
- CCD array - Charge-Coupled Device
- CMOS - Complementary metal oxide semiconductor

Image Forming

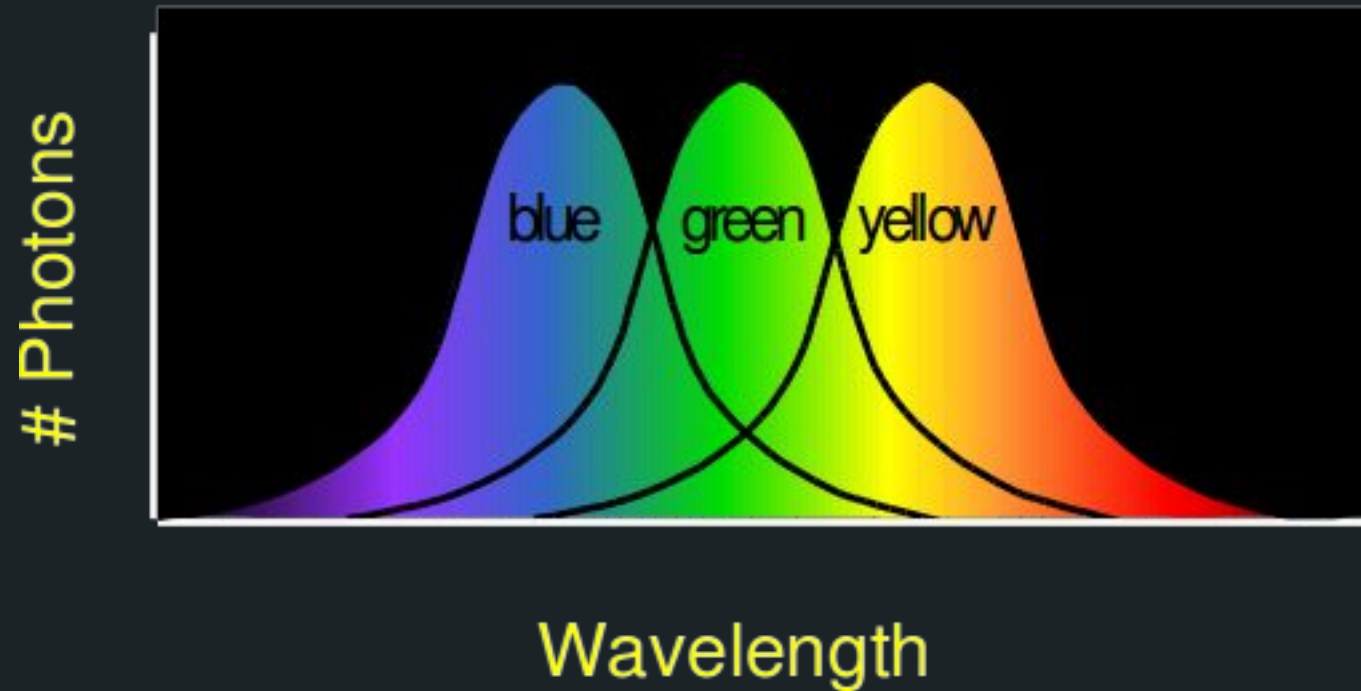
- Hardware
 - Shutter speed
 - Aperture
 - ISO

Projective Geometry

- Projecting 3d to 2d space.
- Lengths are lost
- Angles are lost
- Straight lines remain intact

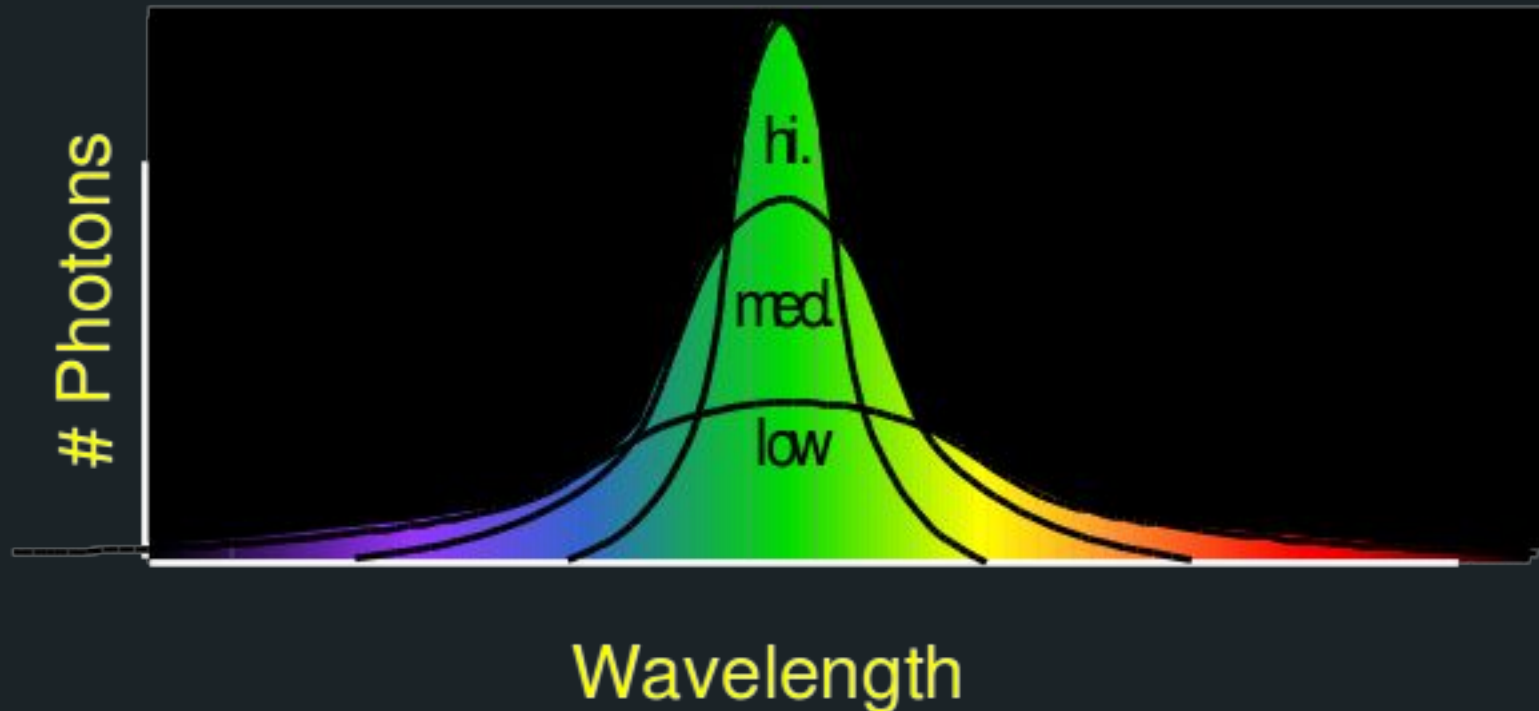
Light - Hue

Mean corresponds to hue



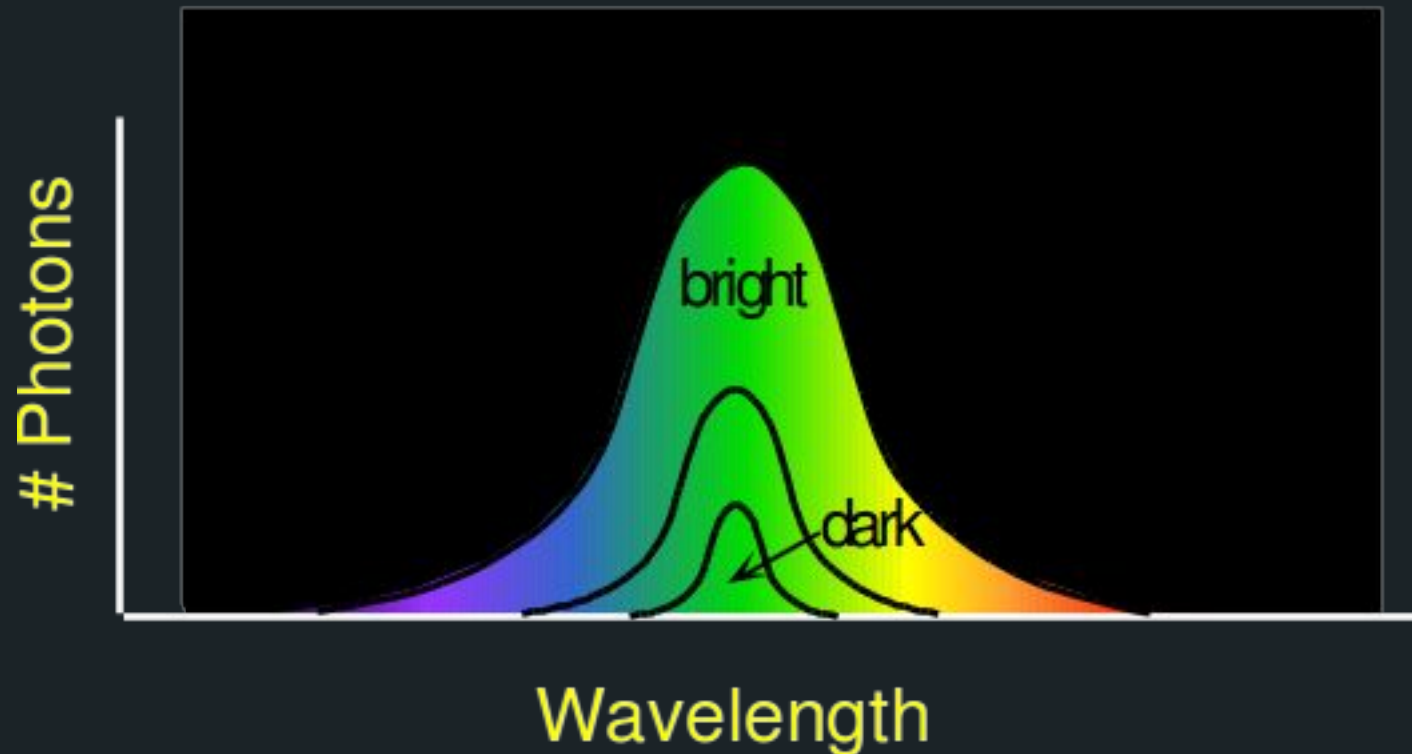
Light - Saturation

Variance corresponds Saturation



Light - Lightness

Area corresponds to brightness



Color Spaces

- CIELAB
- RGB
- YCbCr
- HSV
- HSL
- CMYK

Extract the lanes

Try it in RGB, HSL and Lab
Colour spaces



RGB

Notice that OpenCV reads an image in the order B, G, R and not R, G, B

```
import cv2
```

```
img = cv2.imread('temp.jpg')
```

```
lower_vals = np.array([x,x,x]) #Vary x  
upper_vals = np.array([255,255,255])
```

```
thresh = cv2.inRange(img, lower_vals,  
upper_vals)
```

```
cv2.imshow('ip', img)  
cv2.imshow('op', thresh)  
cv2.waitKey(0)
```

HSV

Have a look at HLS colourspace
too

```
import cv2

img = cv2.imread('temp.jpg')
img = cv2.cvtColor(img,
cv2.COLOR_BGR2HSV)

lower_vals = np.array([0,0,x])
upper_vals = np.array([255,255,255])

thresh = cv2.inRange(img, lower_vals,
upper_vals)

cv2.imshow('ip', img)
cv2.imshow('op', thresh)
cv2.waitKey(0)
```

LAB

```
import cv2
```

```
img = cv2.imread('temp.jpg')  
img = cv2.cvtColor(img,  
cv2.COLOR_BGR2Lab)
```

```
lower_vals = np.array([x,0,0])  
upper_vals = np.array([255,255,255])
```

```
thresh = cv2.inRange(img, lower_vals,  
upper_vals)
```

```
cv2.imshow('ip', img)  
cv2.imshow('op', thresh)  
cv2.waitKey(0)
```

Which method worked best for lane extraction?

Which method would work the best under varying lighting conditions?

Is the L of Lab and V of HSV exactly the same?

What do a and b channels of Lab represent? What is Lab used for?

Image Filtering



Nope. That ain't red wine.

Three views of Filtering

- Image filters in spatial domain
 - Filter is a mathematical operation of a grid of numbers
 - Smoothing, sharpening, measuring texture
- Image filters in the frequency domain
 - Filtering is a way to modify the frequencies of images
 - Denoising, sampling, image compression
- Templates and Image Pyramids
 - Filtering is a way to match a template to the image
 - Detection, coarse-to-fine registration

Image Filtering

- Image filtering: compute function of local neighborhood at each position
- Really important!
 - Enhance images
 - Denoise, resize, increase contrast, etc.
 - Extract information from images
 - Texture, edges, distinctive points, etc.
 - Detect patterns
 - Template matching_a

Example: Box Filter

$g[.,.]$

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

1	1	1
1	1	1
1	1	1

1
—
9

$f[.,.]$

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

$h[.,.]$

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

1	1	1
1	1	1
1	1	1

1
—
9

$f[.,.]$

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							

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

1	1	1
1	1	1
1	1	1

1
—
9

$f[.,.]$

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						

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

1	1	1
1	1	1
1	1	1

1
—
9

$f[.,.]$

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					

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

1	1	1
1	1	1
1	1	1

1
—
9

$f[.,.]$

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				

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

1	1	1
1	1	1
1	1	1

1
—
9

$f[.,.]$

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				

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

1	1	1
1	1	1
1	1	1

1
—
9

$f[.,.]$

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					

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

$f[.,.]$

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	

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

Box Filter

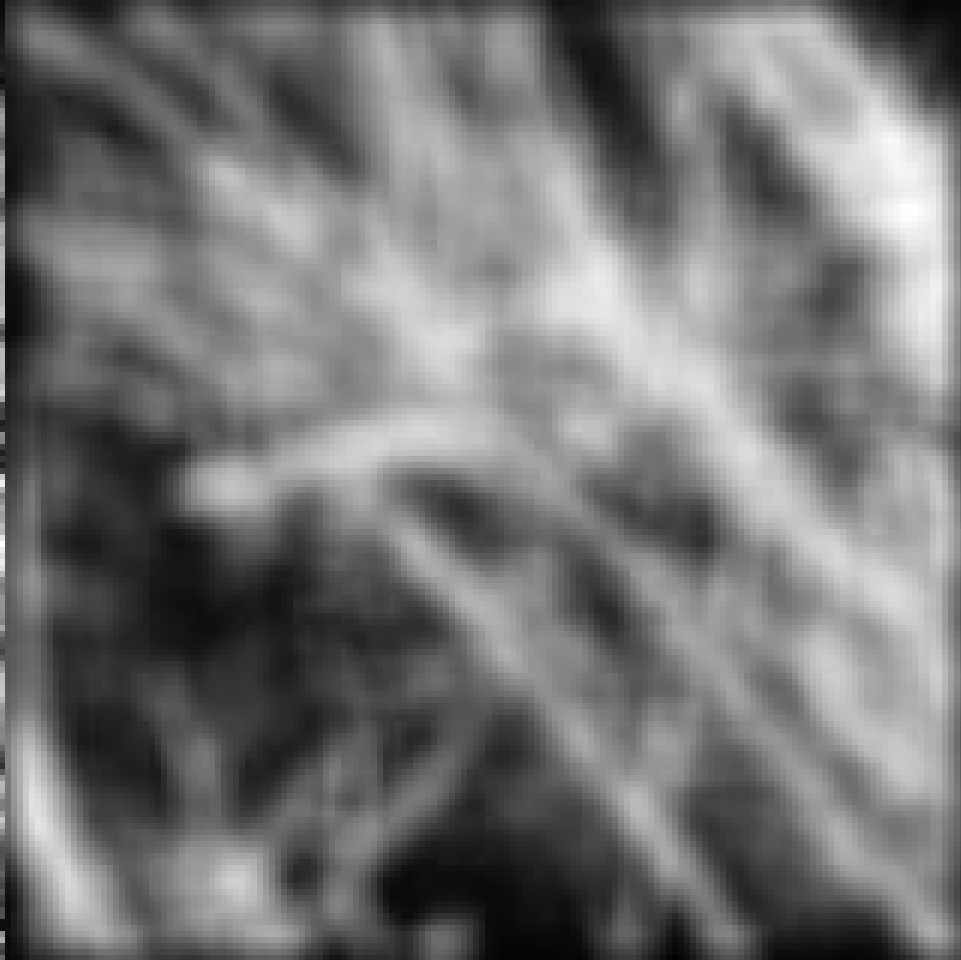
What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

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

1	1	1
1	1	1
1	1	1

Smoothing with Box Filter



Different Linear Filters

(View for Practice)



Original

0	0	0
0	1	0
0	0	0

?

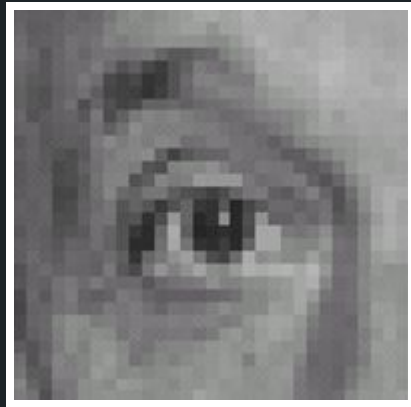


Original

0	0	0
0	1	0
0	0	0



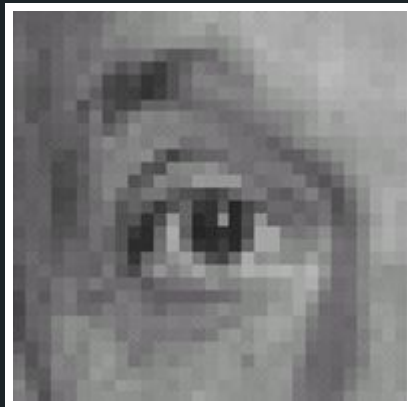
Filtered
(no change)



Original

0	0	0
0	1	0
0	0	0

?



Original

0	0	0
0	2	0
0	0	0

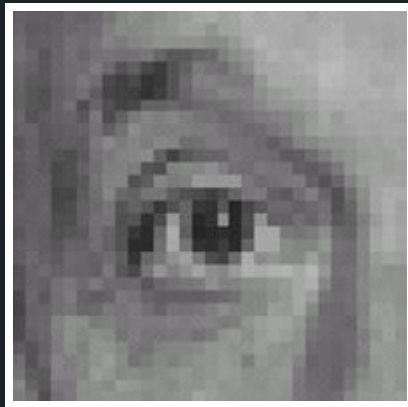
-

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

?

(Note that filter sums to 1)



Original

0	0	0
0	2	0
0	0	0

-

$\frac{1}{9}$

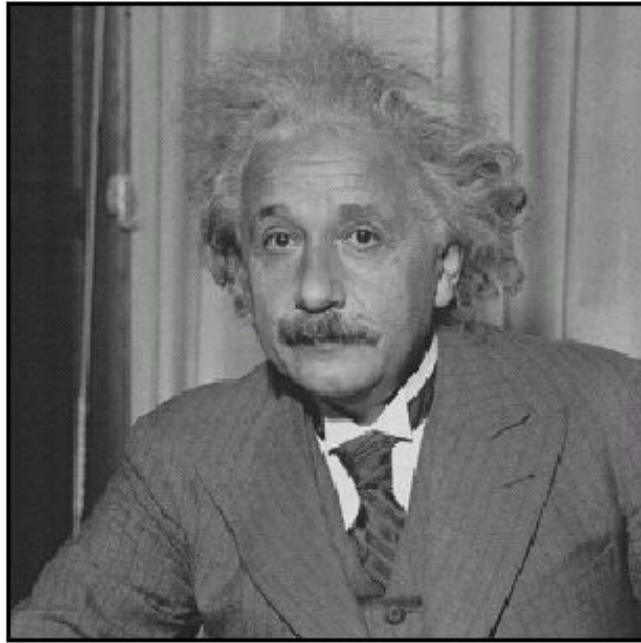
1	1	1
1	1	1
1	1	1



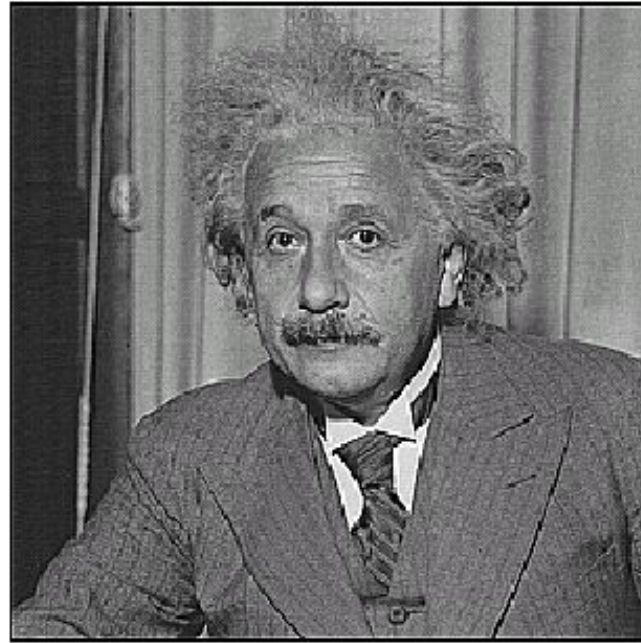
Sharpening filter

- Accentuates differences with local average

Sharpening Filter

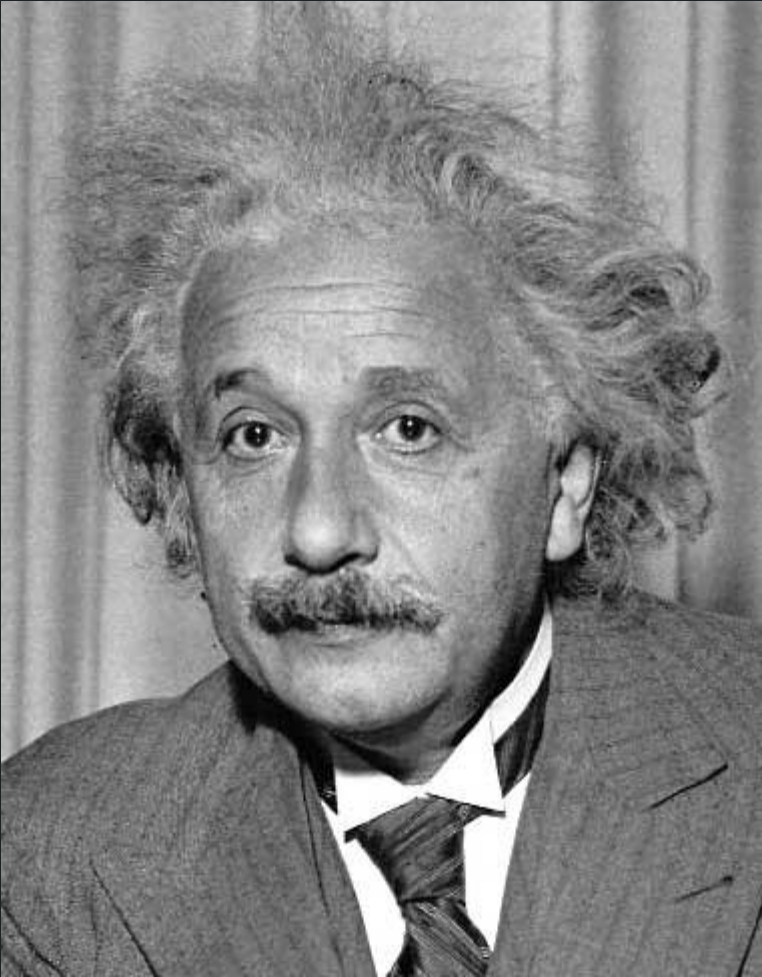


before



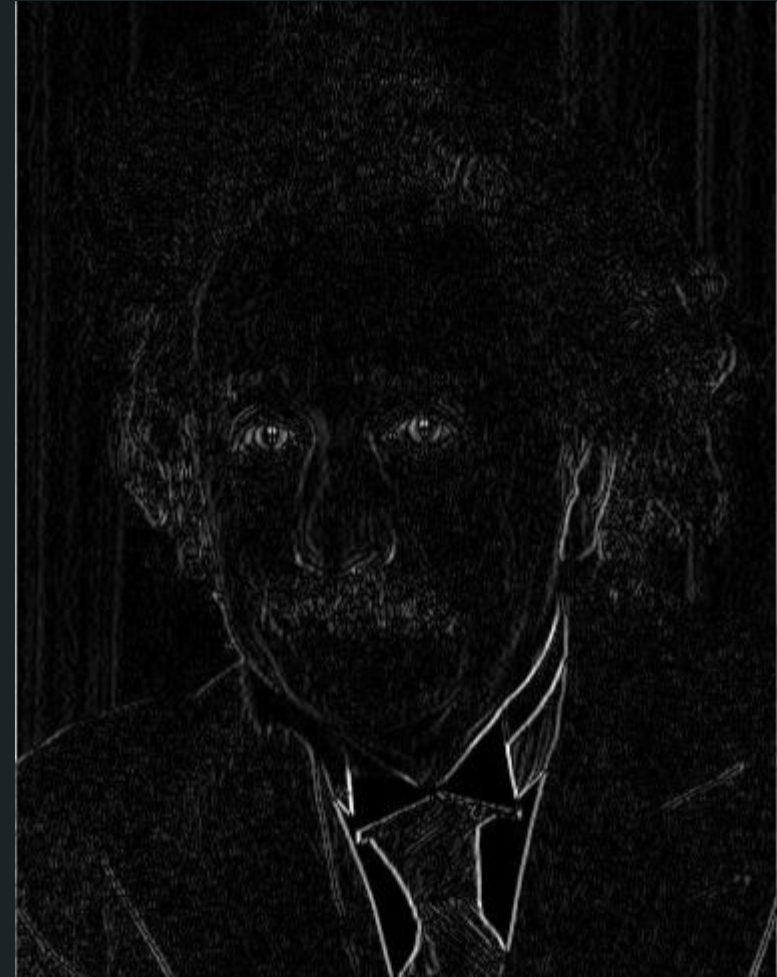
after

Other Filters



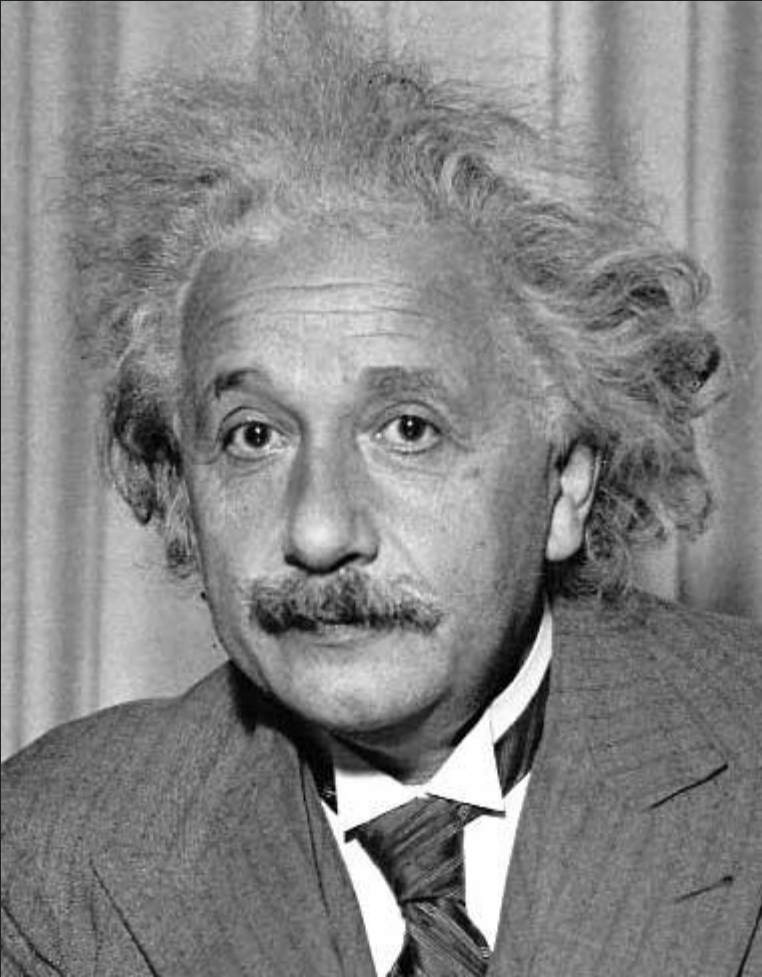
1	0	-1
2	0	-2
1	0	-1

Sobel



Vertical Edge
(absolute value)

Other Filters



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

Sobel



Horizontal Edge
(absolute value)

Synthesising Motion Blur (rotational)

```
#Python. Read some image as img.  
  
angle = 5  
  
img_cntr= tuple(np.array(img.shape)/2)  
rot_mat =  
cv2.getRotationMatrix2D(image_cntr,angle  
,1.0)  
result = cv2.warpAffine(image, rot_mat,  
image.shape,flags=cv2.INTER_LINEAR)  
  
cv2.imshow('temp', 0.8*img+0.2*result)  
  
cv2.waitKey(0)
```

Key Properties of Linear Filters

Linearity:

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

Shift invariance: same behavior regardless of pixel location

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

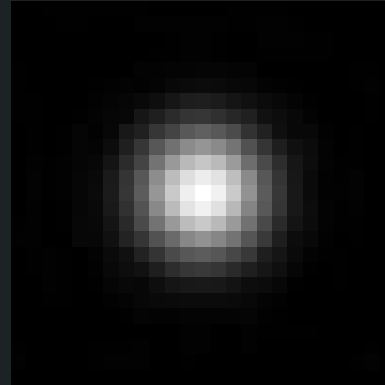
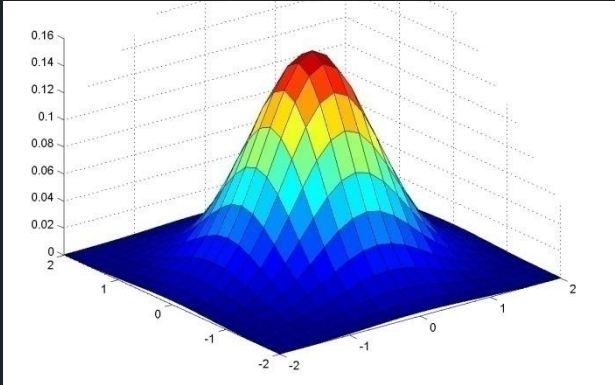
Any linear, shift-invariant operator can be represented as a convolution

More properties

- Commutative: $a * b = b * a$
 - Conceptually no difference between filter and signal
 - But particular filtering implementations might break this equality
- 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 $e = [0, 0, 1, 0, 0]$,
 $a * e = a$

The Gaussian Filter

- Weight contributions of neighboring pixels by nearness

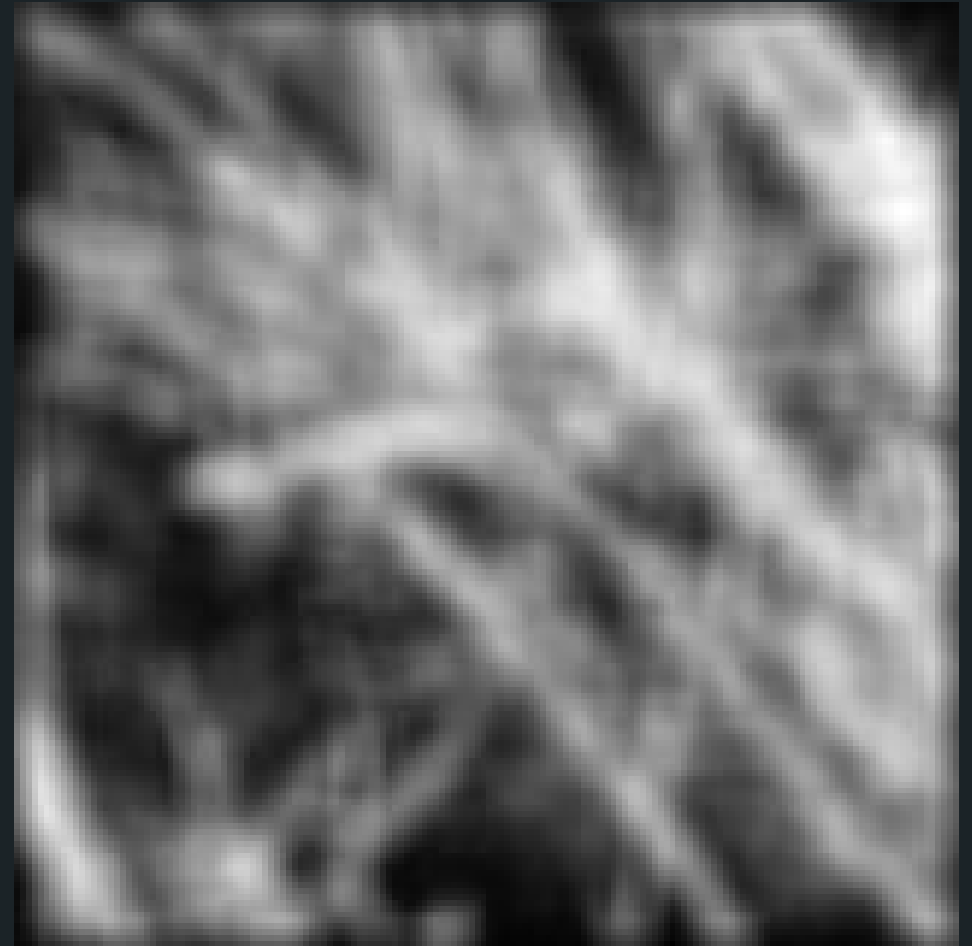
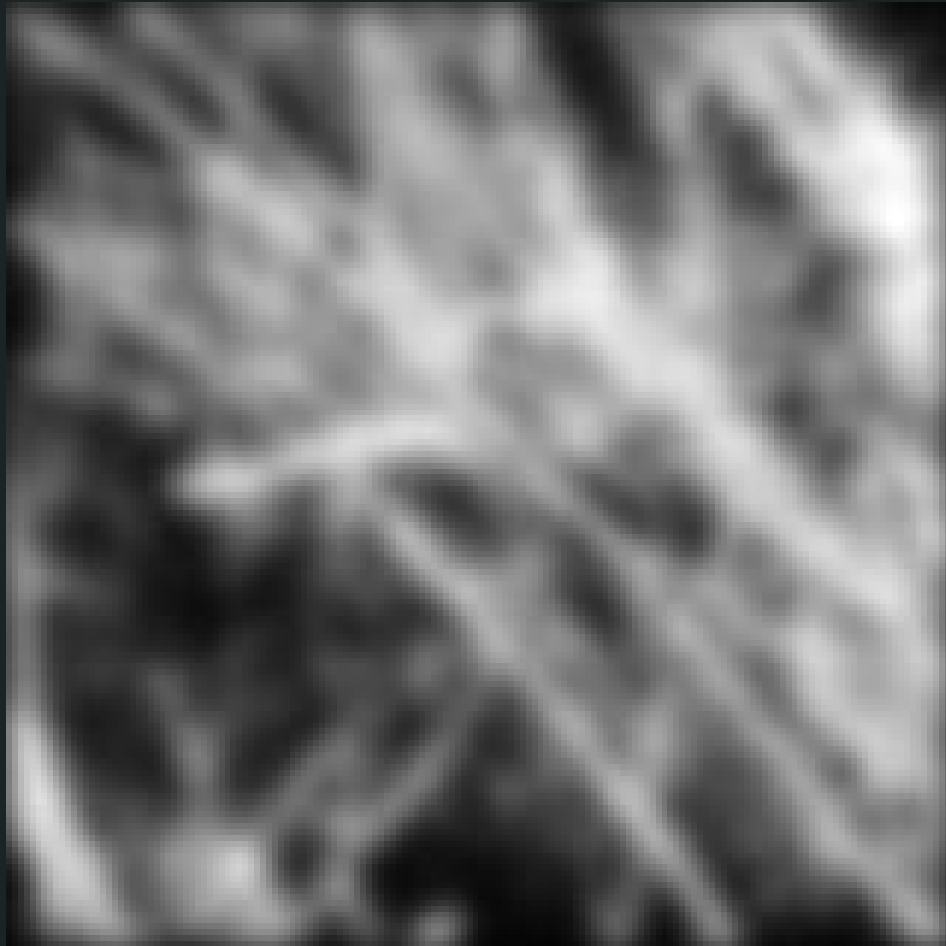


0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

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

Smoothing with Gaussian vs Box Filter



The Gaussian Filter

- 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
 - Convoluting two times with Gaussian kernel of width σ is same as convoluting once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
 - Factors into product of two 1D Gaussians

Separability of the Gaussian Filter

$$\begin{aligned} G_{\sigma}(x, y) &= \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2 + y^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}} \right) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}} \right) \end{aligned}$$

The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

Separability Example

2D convolution
(center location only)

1	2	1
2	4	2
1	2	1

 \ast

2	3	3
3	5	5
4	4	6

$$\begin{aligned} &= 2 + 6 + 3 = 11 \\ &= 6 + 20 + 10 = 36 \\ &= 4 + 8 + 6 = 18 \\ &\hline &65 \end{aligned}$$

Separability Example

The filter factors
into a product of 1D
filters:

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

Separability Example

Perform convolution
along rows:

1	2	1
---	---	---

2	3	3
3	5	5
4	4	6

	11	
	18	
	18	

Followed by convolution
along the remaining column:

1
2
1

	11	
	18	
	18	

	65	

Separability

- Why is separability useful in practice?

Separability

- Why is separability useful in practice?
 - Easier to implement 1D filters
 - Much lesser computation

Practical matters

How big should the filter be?

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

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



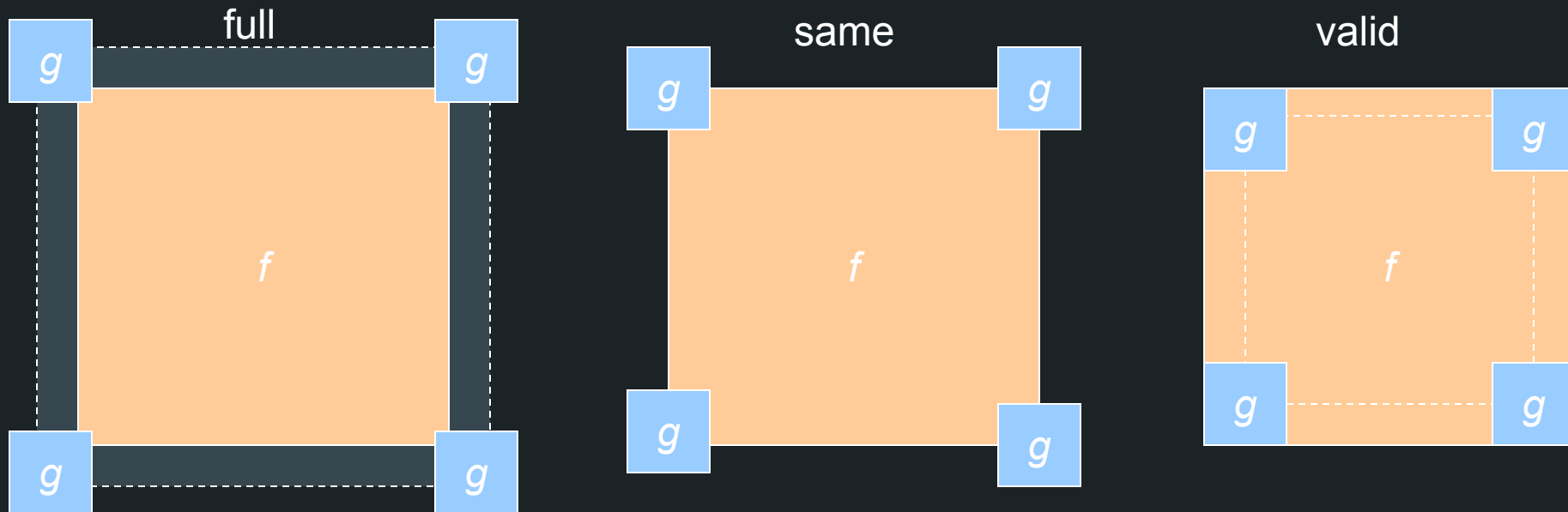
Source: S. Marschner

Practical matters

- `cv2.filter2D(f,-1,g,borderType)`
- `borderType` : Let the image be `|abcdefgh|`
 - `BORDER_REPLICATE` : `aaa|abcdefgh|hhh`
 - `BORDER_REFLECT` : `cba|abcdefgh|hgf`
 - `BORDER_WRAP` : `fgh|abcdefgh|abc`
 - `BORDER_CONSTANT` : `mmm|abcdefgh|mmm`
Where `m` is specified

Practical matters

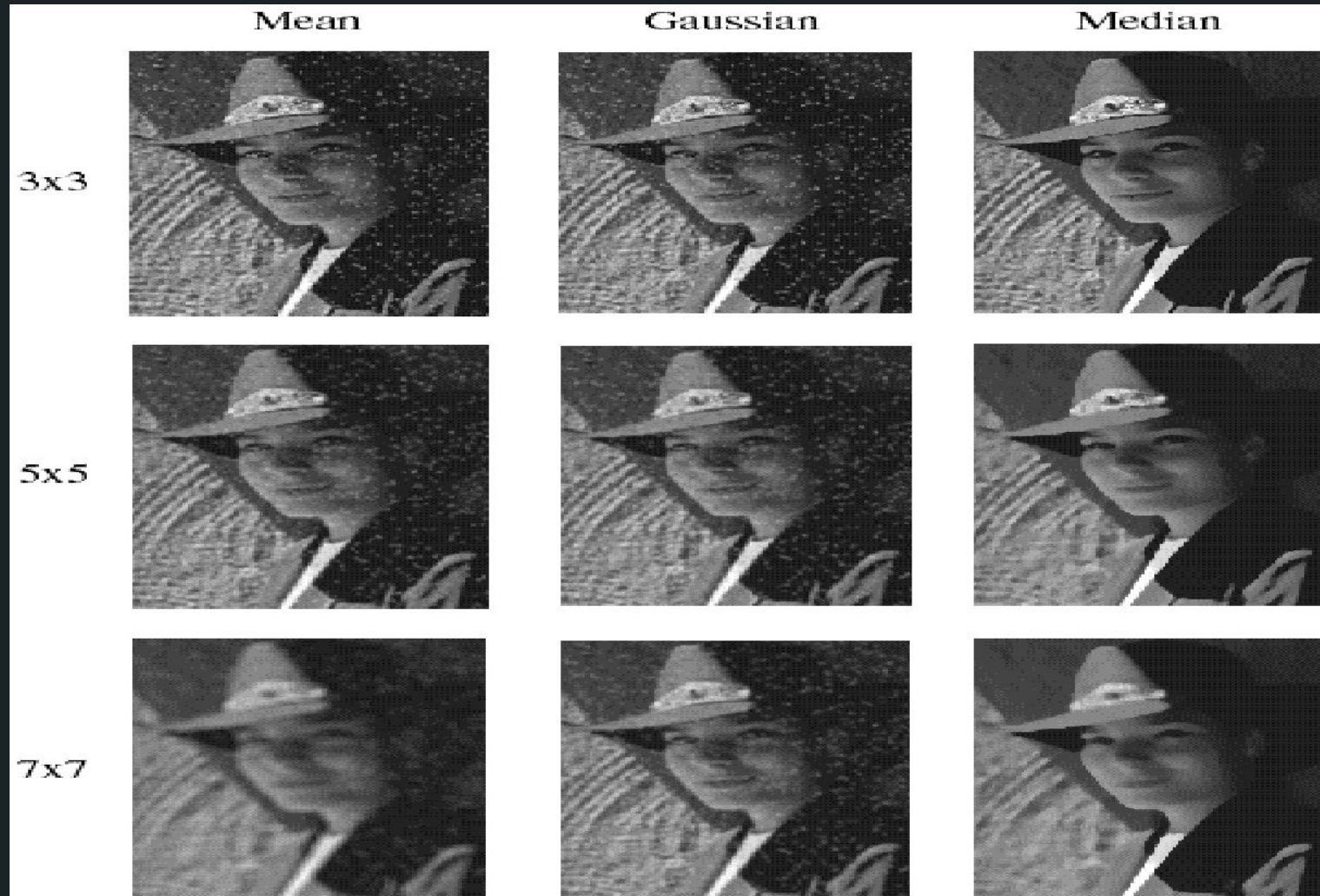
- What is the size of the output?
- MATLAB: `filter2(g, f, shape)`
 - *shape* = 'full': output size is sum of sizes of *f* and *g*
 - *shape* = 'same': output size is same as *f*
 - *shape* = 'valid': output size is difference of sizes of *f* and *g*



Median filters

- A **Median Filter** operates over a window by selecting the median intensity in the window.
- What advantage does a median filter have over a mean filter?
- Is a median filter a kind of convolution?

Comparison: salt and pepper noise



SUMMARY :

- Linear filtering is sum of dot product at each position
 - Can smooth, sharpen, translate (among many other uses)
- Be aware of details for filter size, extrapolation, cropping

