Computer Vision (CS 419/619)

Image filtering in Spatial Domain

Dr. Puneet Gupta

Image filtering

- Image filtering: compute function of local neighborhood at each position
  - Same function applied at each position
  - Output and input image are typically the same size

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

Image Denoising

Why would images have noise?

- Sensor noise: Sensors count photons: noise in count
- Camera Parameters
- And so on…

Let us assume noise at a pixel is

- independent of other pixels
- distributed according to a Gaussian distribution
  - i.e., low noise values are more likely than high noise values

Noise reduction

- Nearby pixels are likely to belong to same object
  - thus likely to have similar color
- Replace each pixel by average of neighbors

1D Filtering: Moving Average

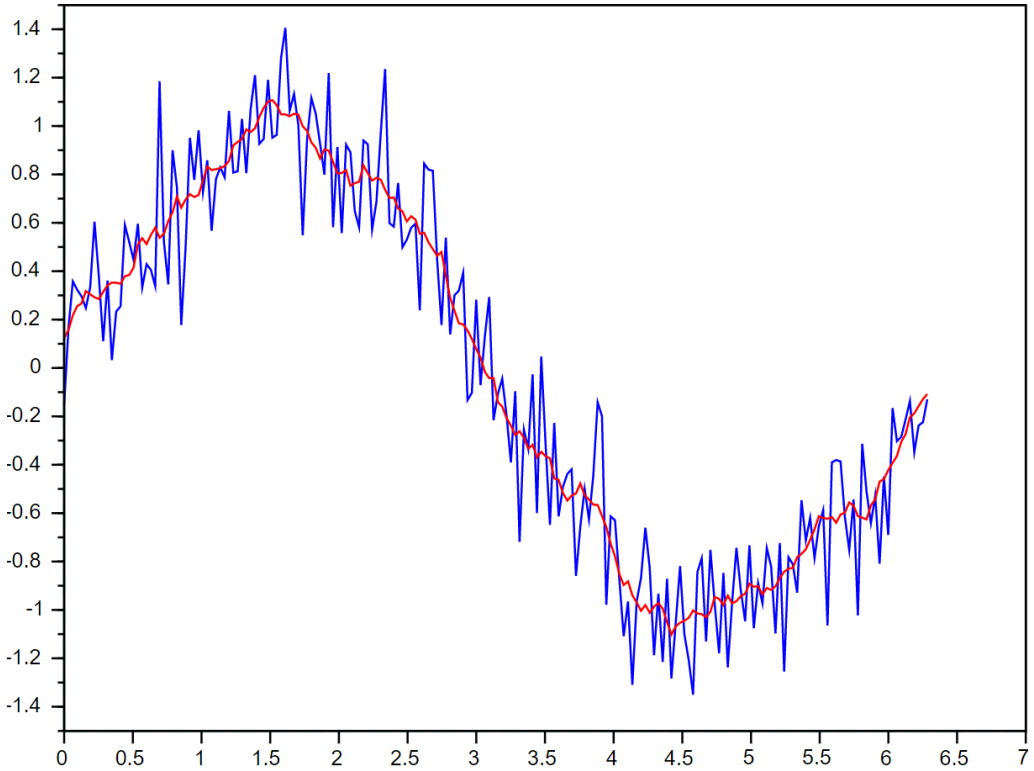
$$I \in \mathcal{R}^{m \times 1}$$

$$h[n] = \frac{1}{k} \sum_{i=n-k+1}^{n} I[i]$$

Ignore boundaries for the moment

Window size ‘k’

$$h[n] = \frac{1}{k} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}^T I[n-k+1:n]$$



$I[n-k/2:n+k/2]$ Different window

https://en.wikipedia.org/wiki/Moving_average

2D Filtering

Compute function of local neighborhood at each position:

h=output f=filter I=image

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

2d coords=k,l 2d coords=m,n

[ ] [ ] [ ]

Note: Filter is often called the 'kernel'

Image Kernels explained visually (setosa.io)

Box/Mean Filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove 'sharp' features)
- Why does it sum to one?

$$f[\cdot,\cdot]$$

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

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

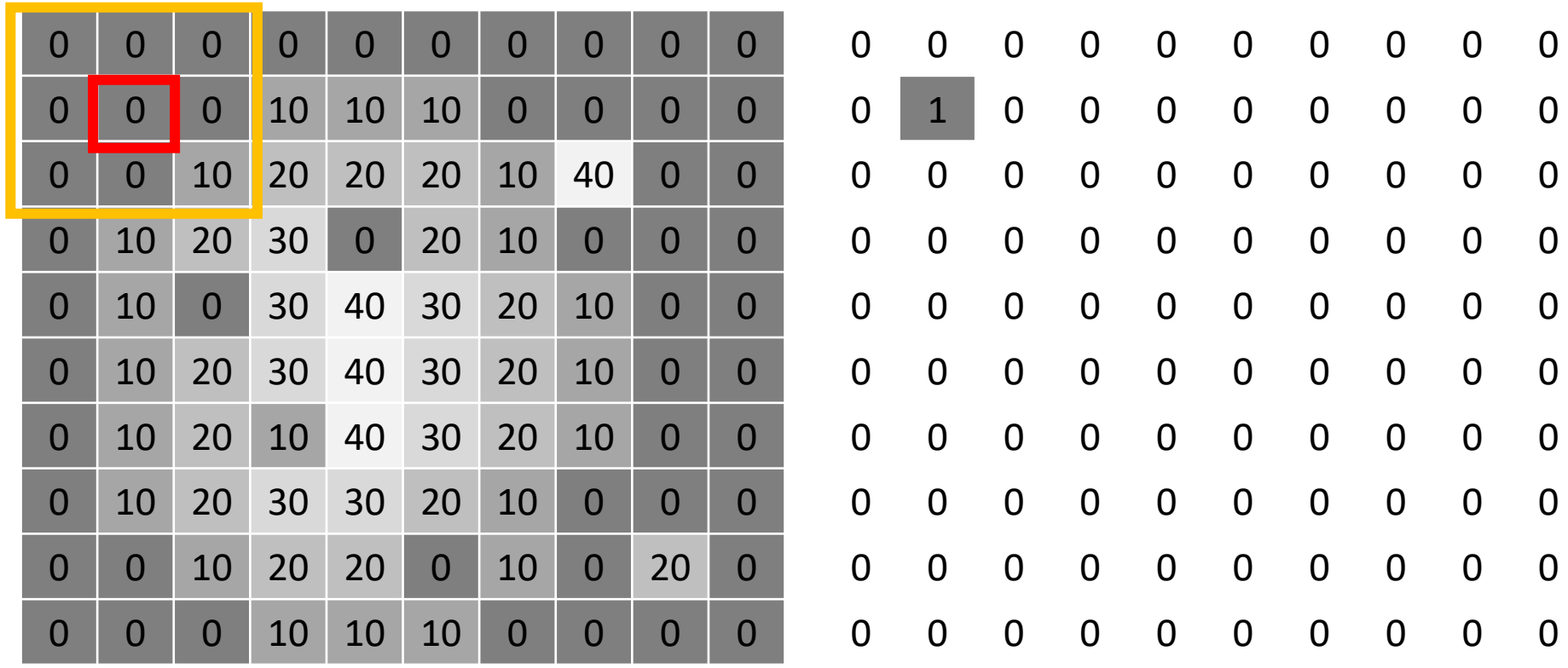
$$(0 + 0 + 0 + 10 + 40 + 0 + 10 + 0 + 0)/9 = 6.66$$

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

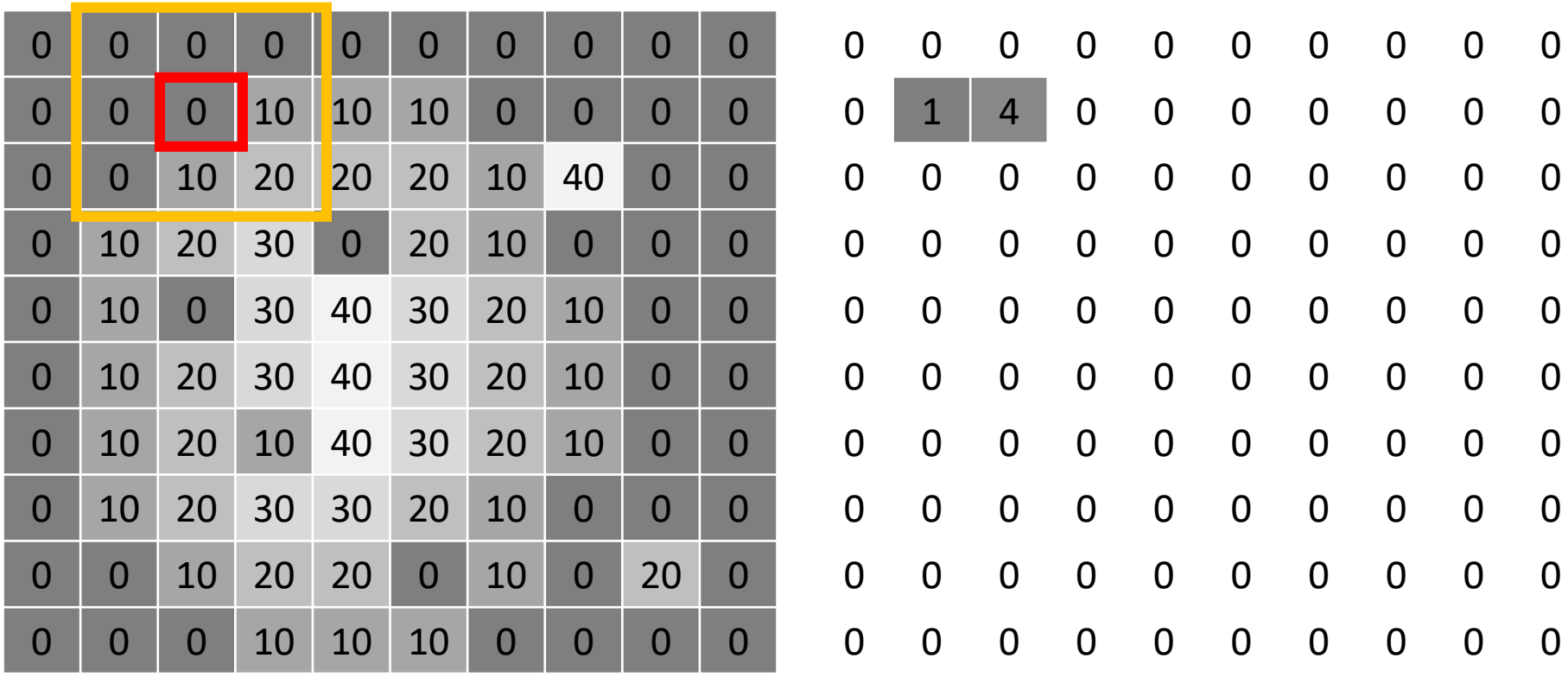
(0 + 0 + 0 + 0 + 0 + 10 + 0 + 0 + 0 + 0 + 20 + 10 + 40 + 0 + 0 + 20 + 10 + 0 + 0 + 0 + 30 + 20 + 10 + 0 + 0)/25 = 6.8

Mean filtering



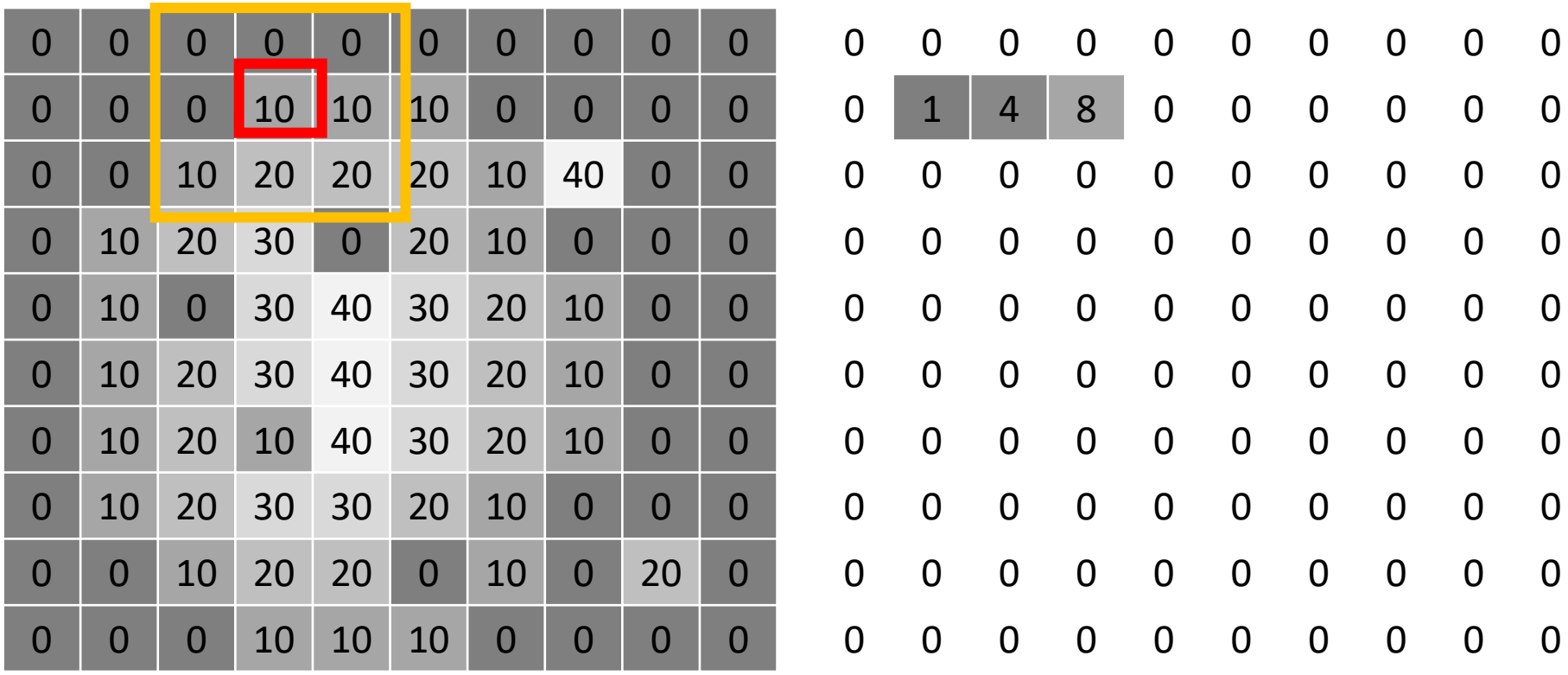
$$(0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 10)/9 = 1.11$$

Mean filtering



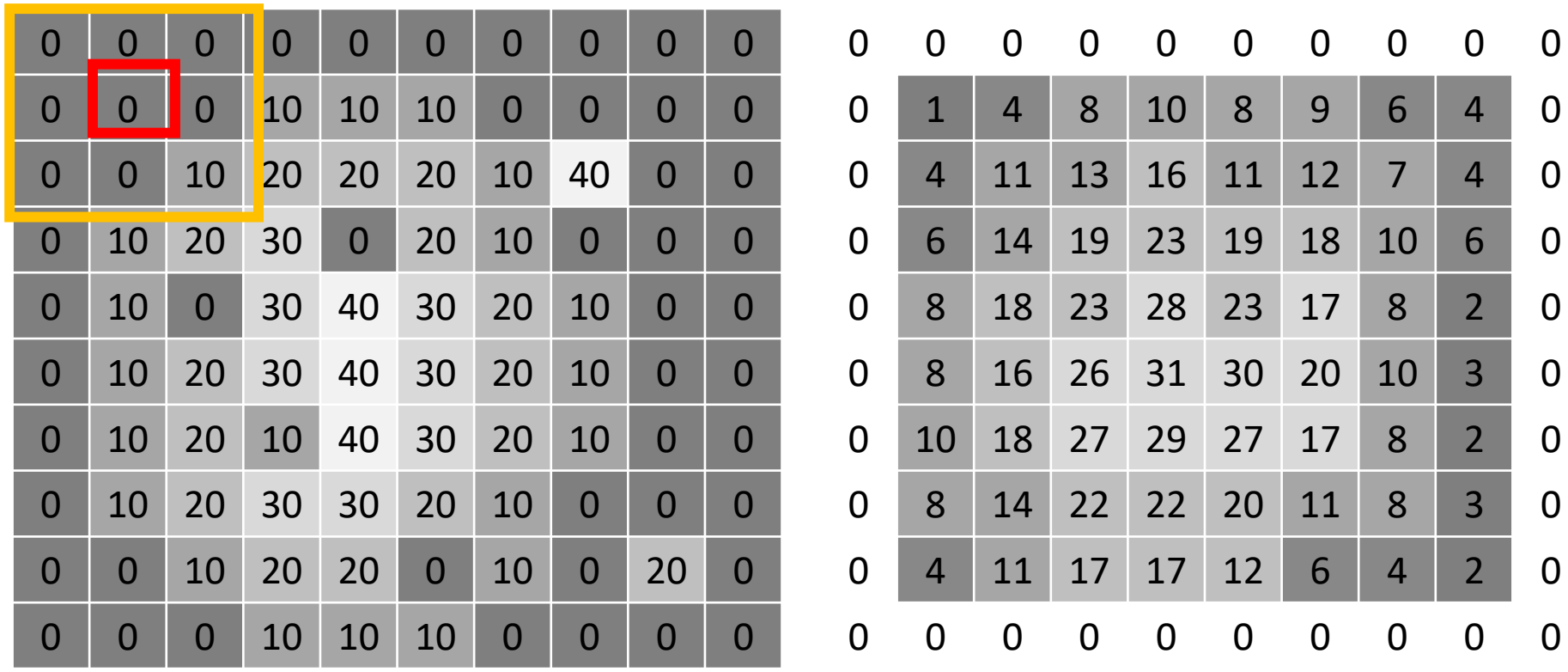
$$(0 + 0 + 0 + 0 + 0 + 10 + 0 + 10 + 20)/9 = 4.44$$

Mean filtering



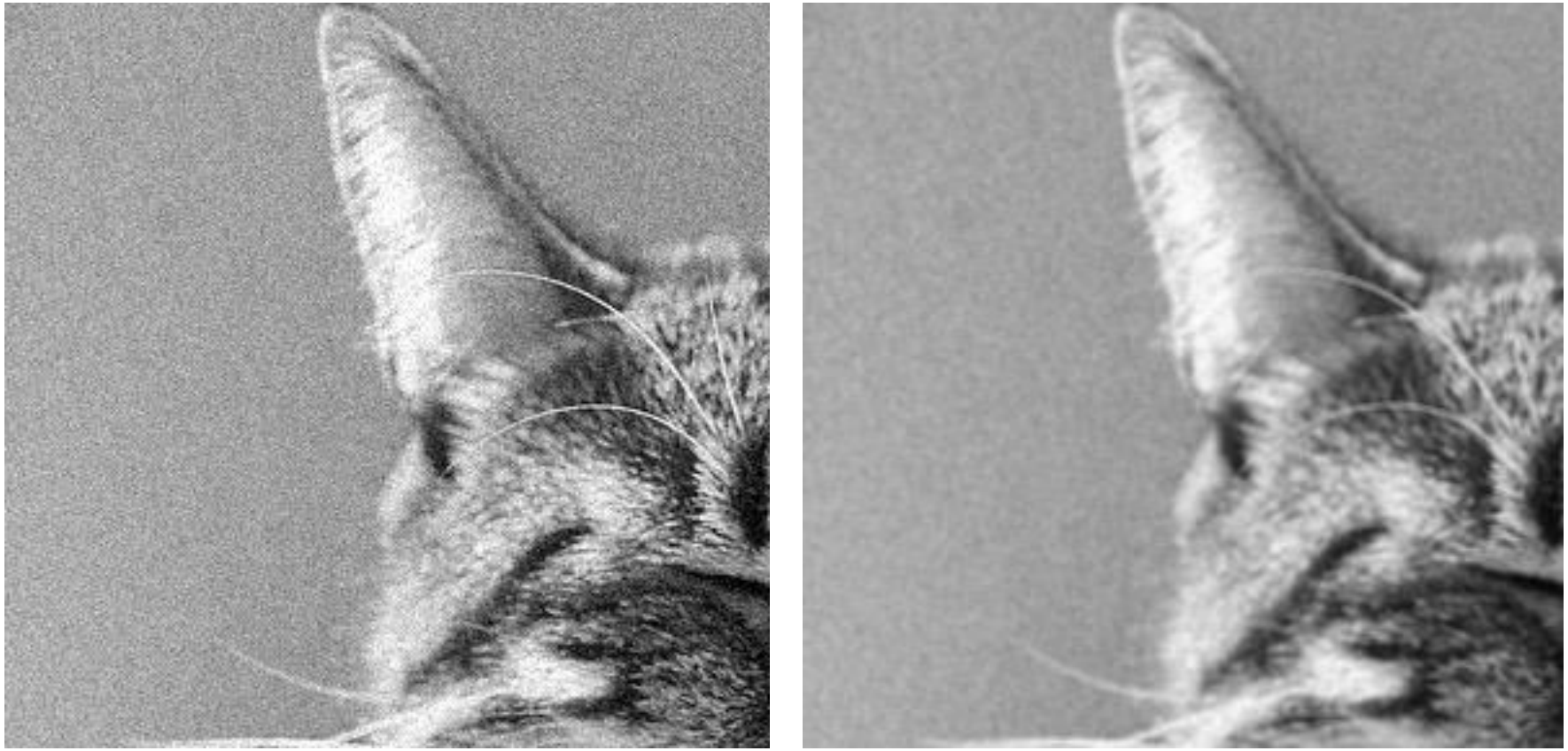
$$(0 + 0 + 0 + 0 + 10 + 10 + 10 + 20 + 20)/9 = 7.77$$

Mean filtering



The box filter blurs the image because for each pixel, the values of its neighbors is averaged with it, smearing out the intensity values. Certain image filtering operations need to preserve mean image intensity. Box filter, a blurring filter, does that (reason why it sums to 1). Blurring only makes sense if the mean image intensity is preserved.

Noise reduction using mean filtering

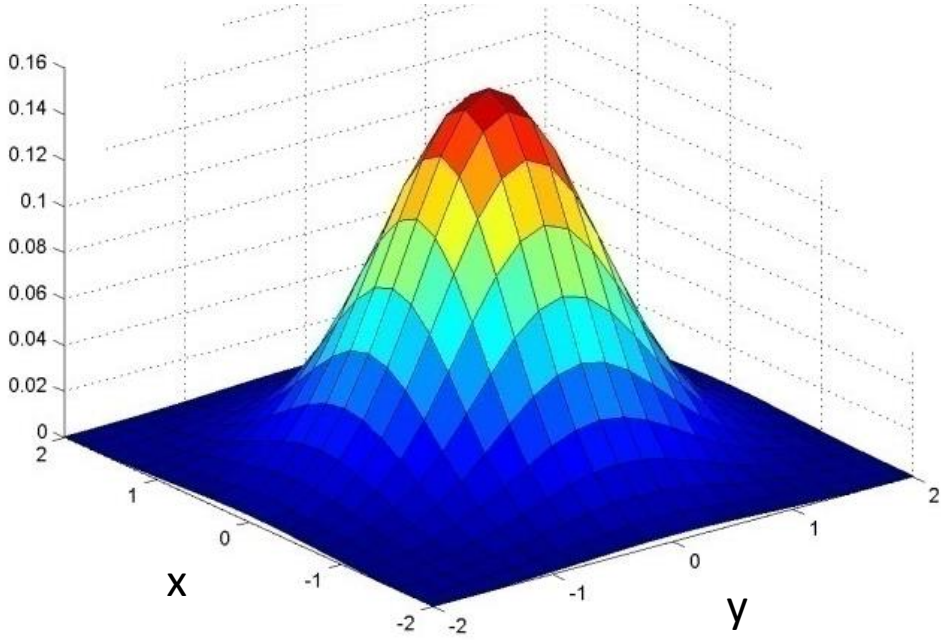


Smoothing with Box Filter

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

1	1	1
1	1	1
1	1	1

Gaussian Filter



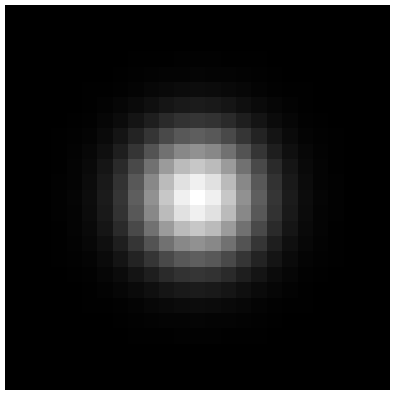
x

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

y

Kernel size 5 x 5,
Standard deviation $\sigma = 1$

Viewed from top

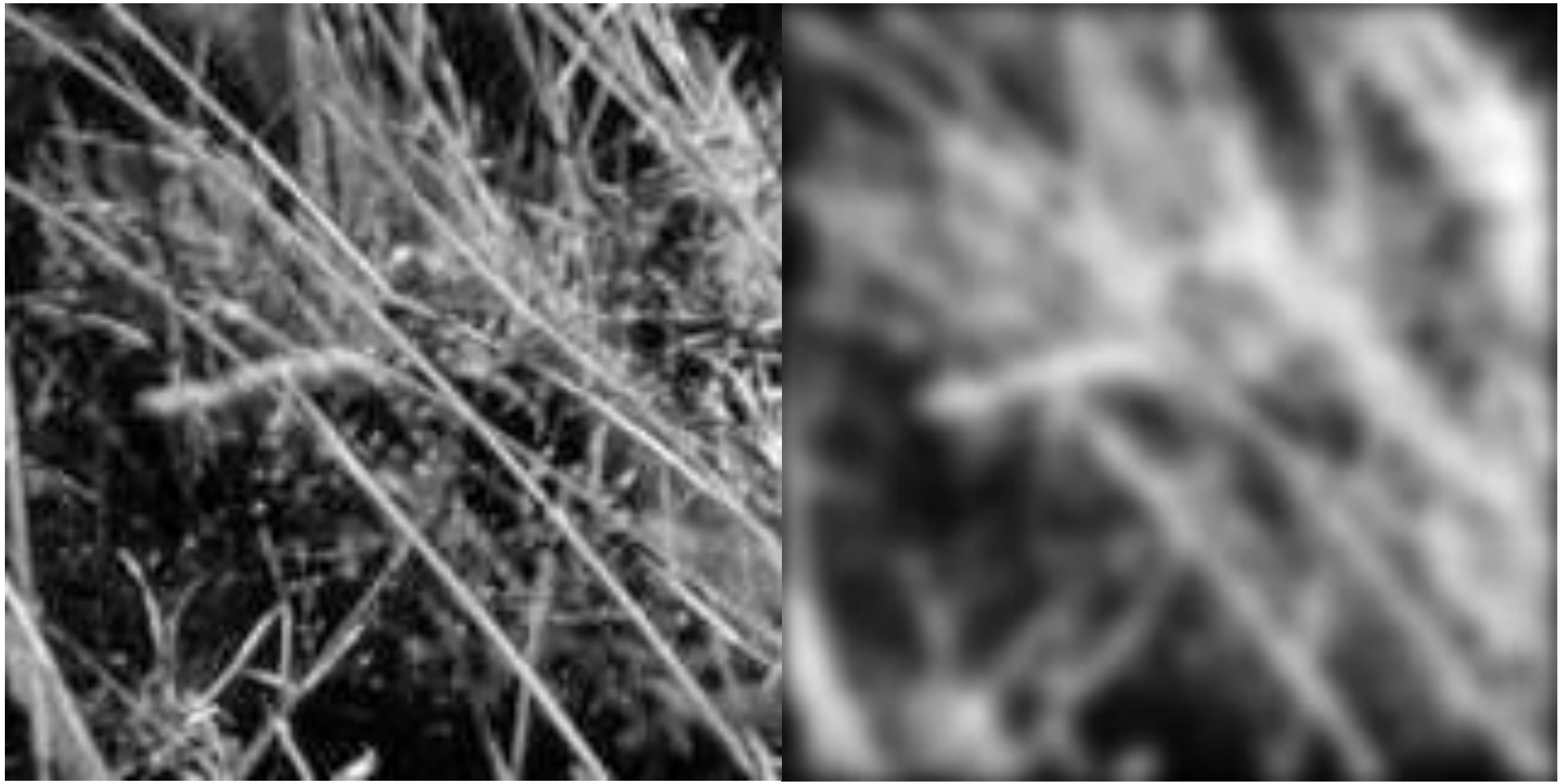


y

x

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

Smoothing with Gaussian Filter



Smoothing with Box Filter

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

1	1	1
1	1	1
1	1	1

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

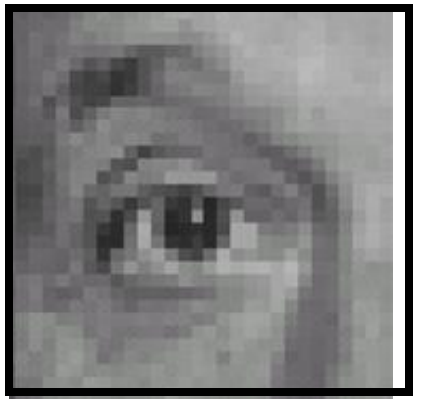
?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Source: D. Lowe

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)

Source: D. Lowe

Practice with linear filters

Original

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

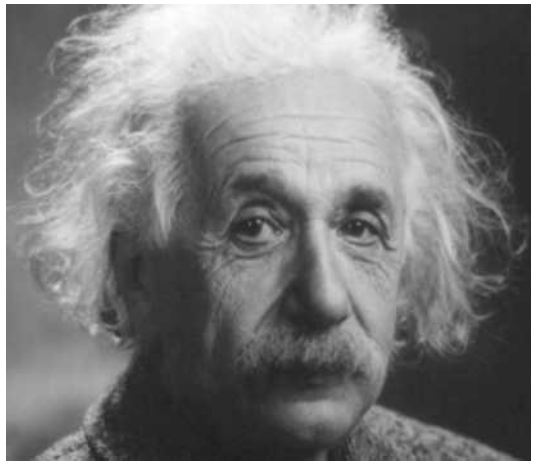
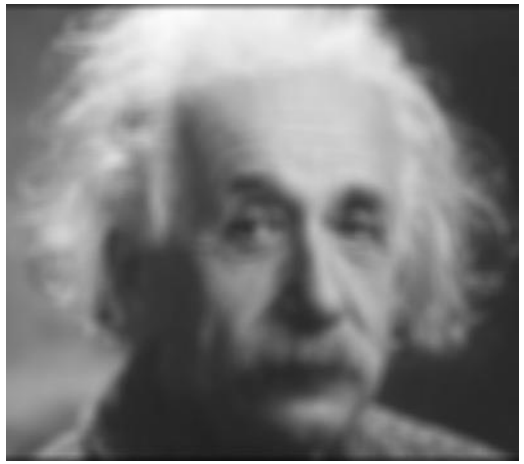
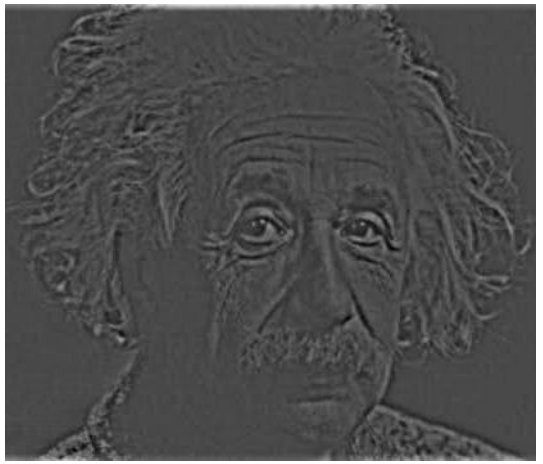
Sharpening filter

- Accentuates differences with local average

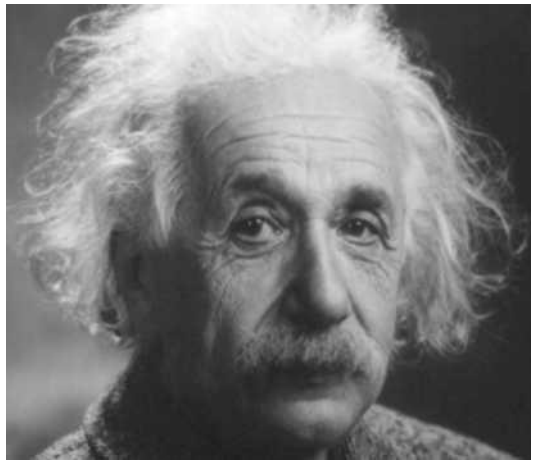
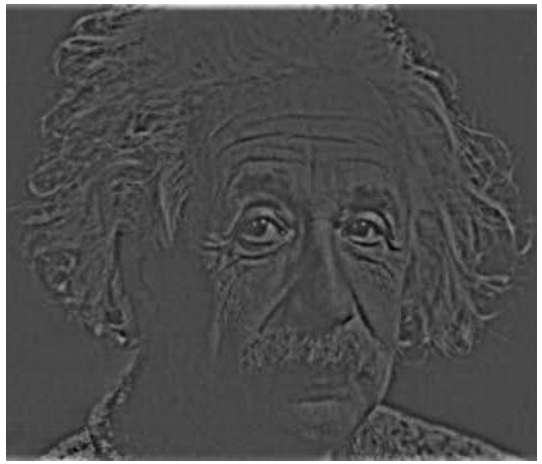
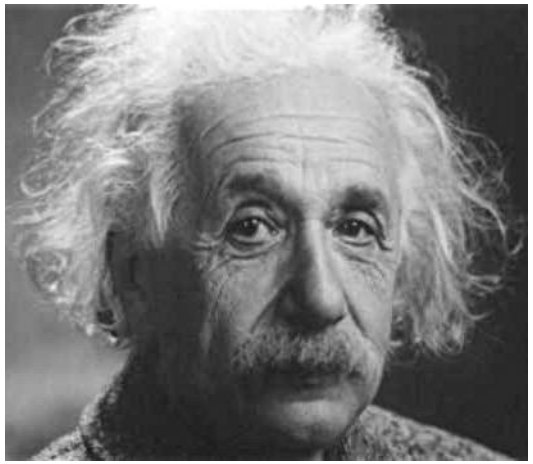
Source: D. Lowe

Sharpening

- What does blurring take away?

Let’s add it back:

Sharpening

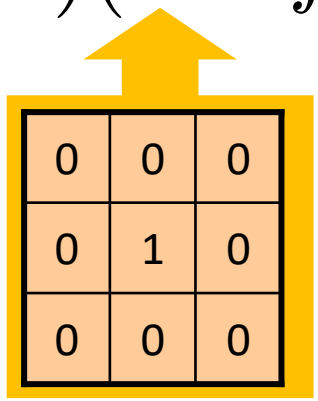
$$f_{sharp} = f + \alpha(f - f_{blur})$$

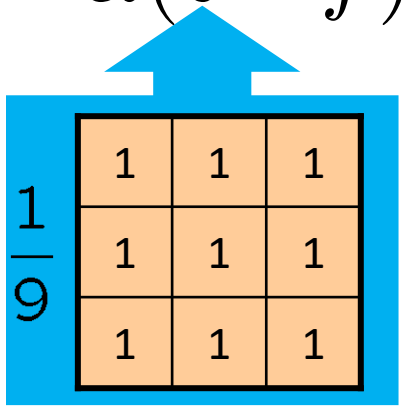
$$= (1+\alpha)f - \alpha f_{blur}$$

$$= (1+\alpha)(w * f) - \alpha(v * f)$$

Our previous filter

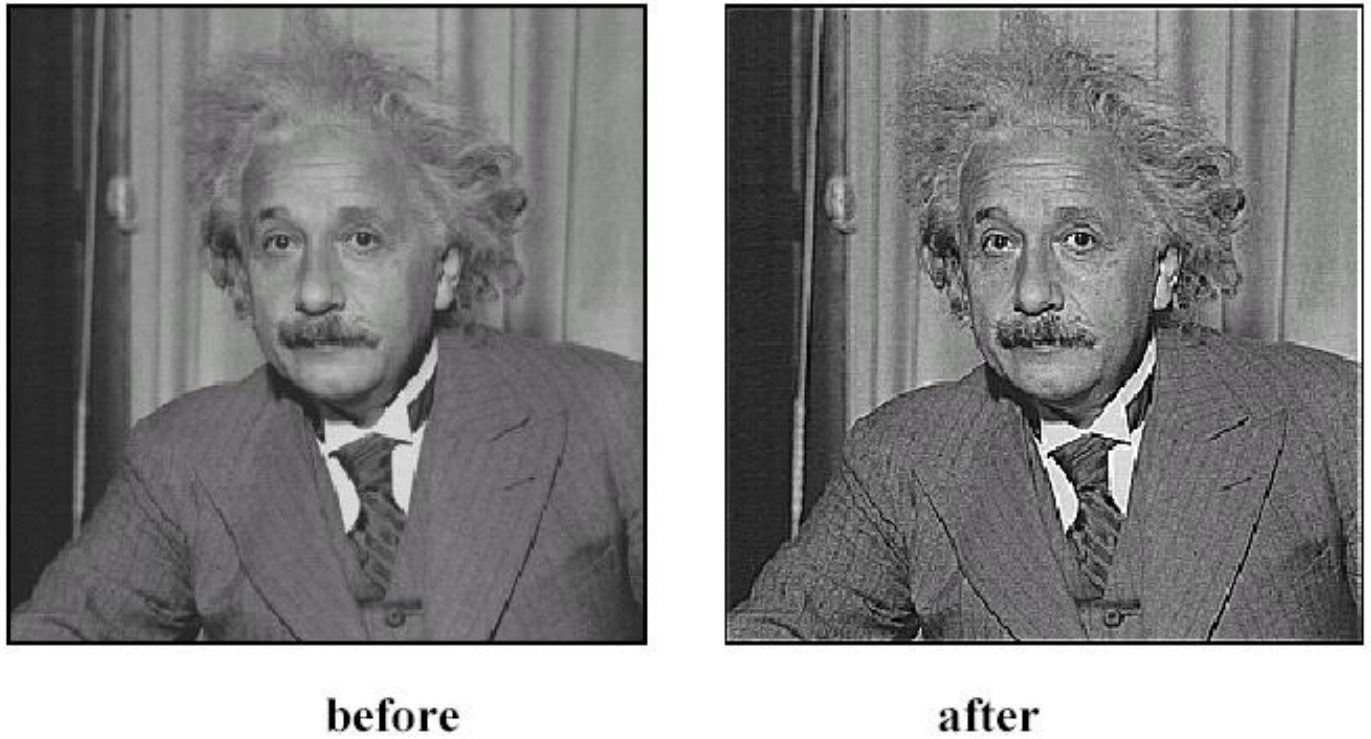
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$





$$= ((1+\alpha)w - \alpha v) * f$$

Sharpening

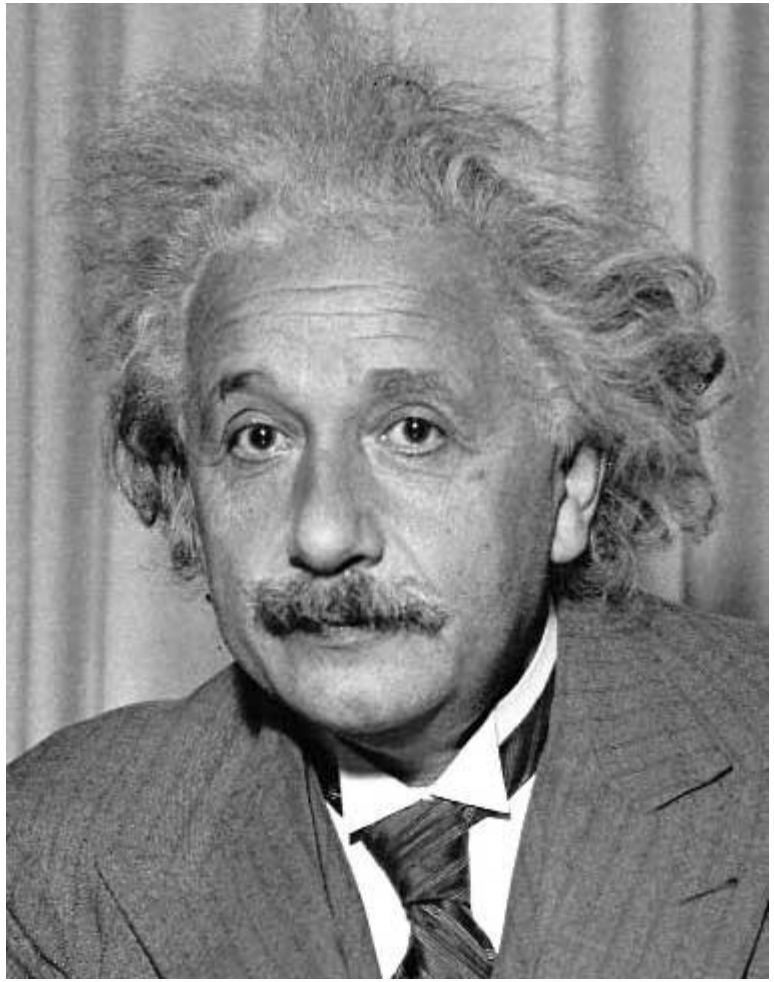


before

after

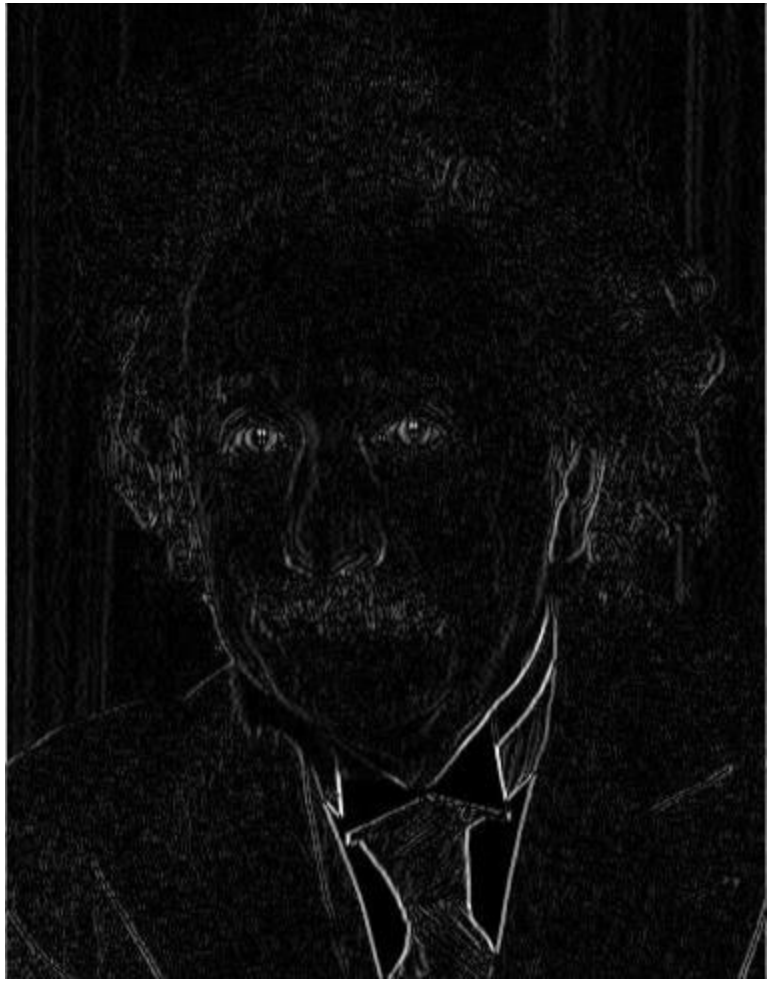
Source: D. Lowe

Edge filters



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

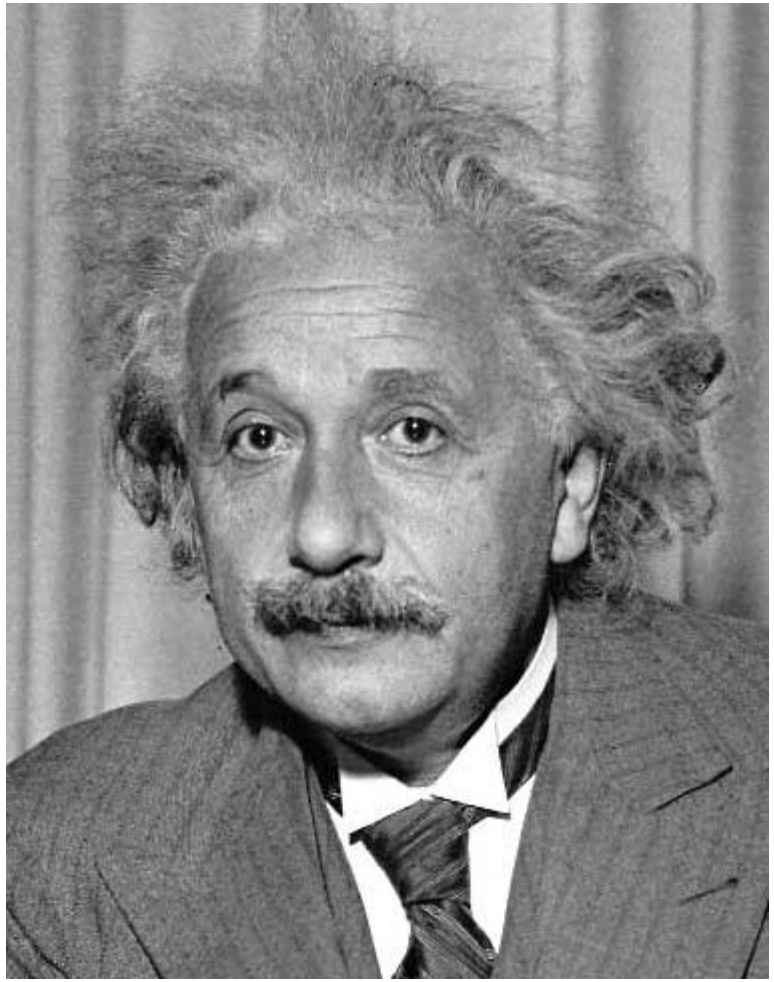
Sobel



Vertical Edge
(absolute value)

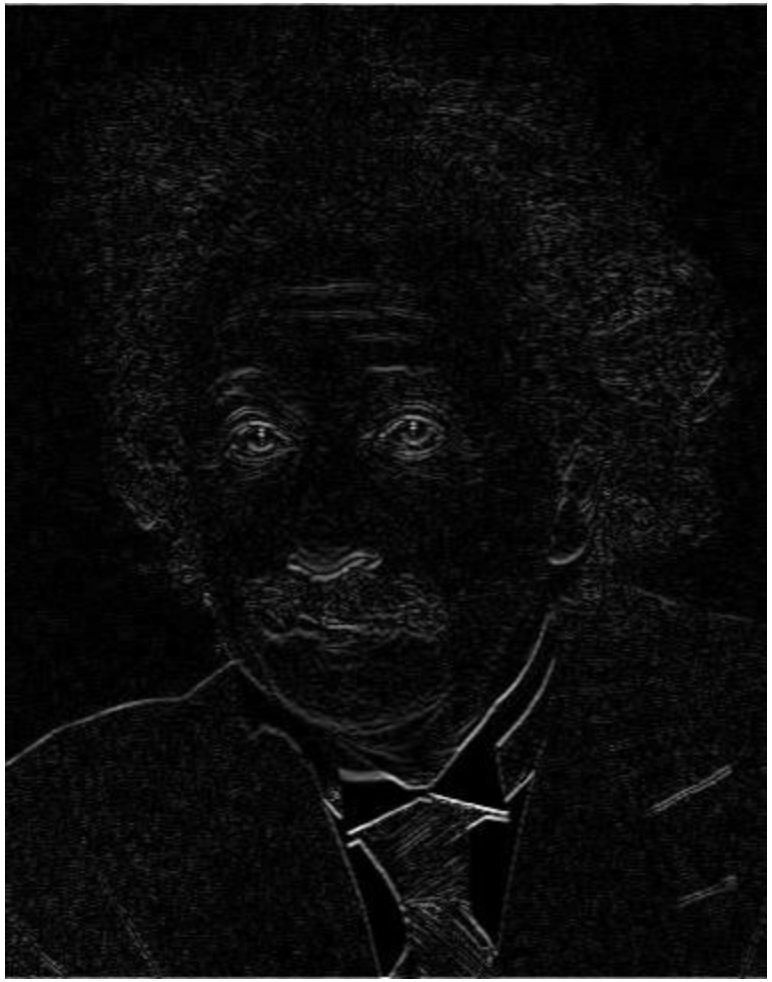
Notice that this filter sums to 0.

Edge filters



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

Sobel



Horizontal Edge
(absolute value)

Sobel Filter

- What is the 1 2 1 pattern?
  - Binomial distribution or “triangle filter”
  - Discrete approximate to Gaussian; fast for integer mathematics
  - Smooths along edge

- What happens to negative numbers?
  - let’s take the absolute value (the magnitude)

- For visualization:
  - Shift image + 0.5
  - If gradients are small, scale edge response

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

Sobel

Another basic gradient/edge 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

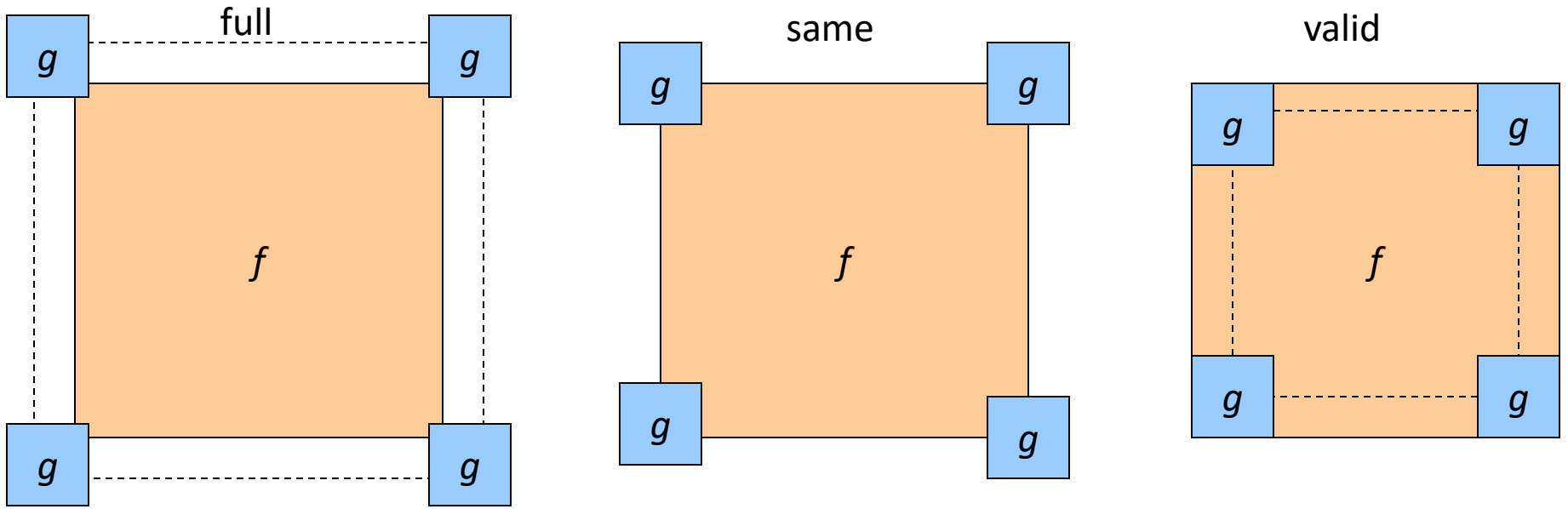
or

-1
0
1

Filters in Practice

What is the size of the output?

- *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

Filters in Practice

What about near the edge?

- The filter window falls off the edge of the image
- Need to extrapolate
- Methods:
  - Clip filter: any value outside of the image is set to 0
  - Wrap around: copy the values near the opposite edge (e.g. copying the bottom portion of the image to the outside of the top boundary)
  - Copy edge: copy the pixel value of the nearest edge
  - Reflect across edge: treat out-of-bounds regions as ‘mirrors’ that reflect near-surface image pixels in reverse order

Filters in Practice




Clip filter



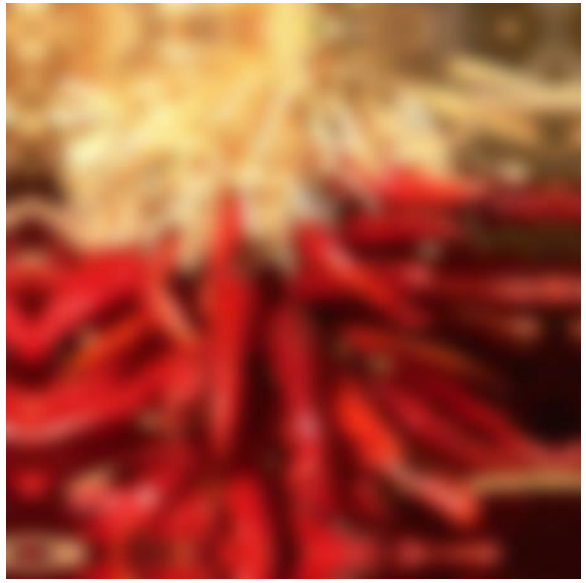

Wrap around

Filters in Practice




Copy edge






Reflect across edge

Correlation and Convolution

2D correlation

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

```
e.g., h = scipy.signal.correlate2d(f,I)
```

2D convolution

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m-k, n-l]$$

```
e.g., h = scipy.signal.convolve2d(f,I)
```

Convolution is the same as correlation with a 180° rotated filter kernel.

Correlation and convolution are identical when the filter kernel is rotationally symmetric*.

* Symmetric in the geometric sense, not in the matrix linear algebra sense.

Correlation and Convolution

- Symmetric in the matrix sense: S**ymmetric matrix** is a square matrix that is equal to its transpose. A = A^T
- However, to rotate the matrix by 180 degrees and still have the same matrix, we need the matrix to also have the same values elsewhere, e.g., at the top left and bottom right.
- Image matrix I: 1 2
  3 4

- Symmetric filter: 1 2 <- this matrix is equal to its transpose, but is not rotationally symmetric
  2 4

- Correlation computation: 1*1+2*2+3*2+4*4 = 27
- Convolution computation: 1*4+2*2+3*2+4*1 = 18 <- Different answers

- Rotationally symmetric: 1 2
  2 1<- Same values with either correlation or convolution

Convolution

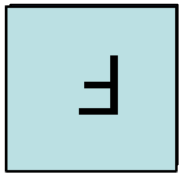
Convolution:

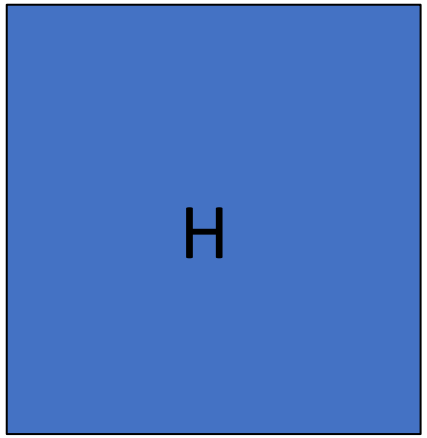
- Flip the filter in both dimensions
  - from bottom to top, and right to left
- Then apply cross-correlation

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

$$G = H \star F$$

Notation for convolution operator





- For a Gaussian, sobel or box filter, how will the outputs differ?
- If the input is an impulse signal, how will the outputs differ?

Slide credit: Kristen Grauman

Linear Filters

Linearity:

$$\texttt{imfilter(I, } f_1 + f_2\texttt{)} = \texttt{imfilter(I,}f_1\texttt{)} + \texttt{imfilter(I,}f_2\texttt{)}$$

Shift/translation invariance:
Same behavior given intensities regardless of pixel location m,n

$$\texttt{imfilter(I,shift(f))} = \texttt{shift(imfilter(I,f))}$$

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

- To be linear, filters must satisfy these two properties. Linearity means that filtering an image with two (different) filters f1 and f2 results in the same output as filtering the images separately first before summing the intermediate outputs.
- Translation invariance means that shifting the original image before filtering results in the same output as filtering then shifting.

Correlation and Convolution: Commutative

F = [1 0 -1]

H = [3 1 4 1 5 9]

- **F convolution H** = [3 1 1 0 1 8 -5 -9]
  - The first term is: -1 * 0 + 0 * 0 + 1 * 3 = 3
  - The second term is: -1 * 0 + 0 * 3 + 1 * 1 = 1
  - The third term is: -1 * 3 + 0 * 0 + 1 * 4 = 1, and so on
- **H convolution F**: [3 1 1 0 1 8 -5 -9]
  - The first term is 9 * 0 + 5 * 0 + 1 * 0 + 4 * 0 + 1 * 0 + 3 * 1 = 3.
  - The second term is 9 * 0 + 5 * 0 + 1 * 0 + 4 * 0 + 1 * 1 + 3 * 0 = 1, and so on
- **F corr H**: [-3 -1 -1 0 -1 -8 5 9]
- **H corr F**: [9 5 -8 -1 0 -1 -1 -3]

So we see that F convolution H is equal to H convolution F, but F corr H is not equal to H corr F. So convolution is commutative but correlation is not.

Convolution calculator online: https://www.rapidtables.com/calc/math/convolution-calculator.html

Correlation and Convolution: Associative

Likewise, the key difference between the two is that convolution is associative.

- If F and G are filters, then F*(G*I) = (F*G)*I.
- Example, use F=G=[-1 0 1], and I = [1 2 3 4]
- Then, F*(G*I) = (F*G)*I = [1 2 1 0 -5 -6 3 4]

Show that it will fail for correlation.

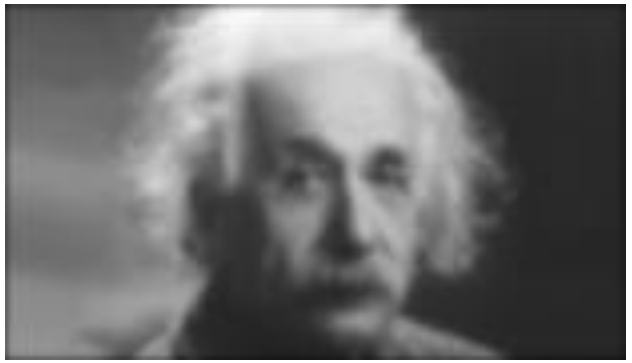
Use in sharpening: Smooth an image and then take its derivative.

- Convolve image with a Gaussian filter, and then convolving it with a derivative filter.
- Convolve the derivative filter with the Gaussian to produce a filter called a Difference of Gaussian (DOG), and then convolve this with our image.
- The DOG filter can be precomputed, and we only have to convolve one filter with our image.

Convolution is usually used for image processing operations such as smoothing, while correlation is used to match a template to an image.

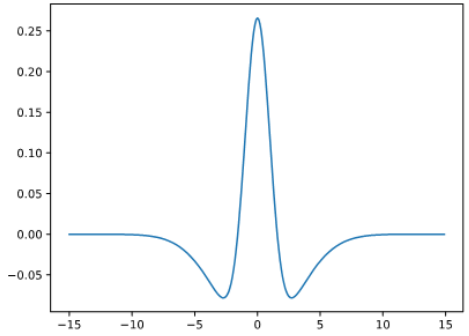
- It doesn’t really make sense to combine two templates into one with correlation.
- We might often want to combine two filter together for convolution.

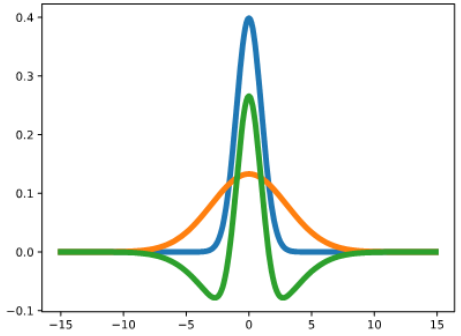
Difference of Gaussians

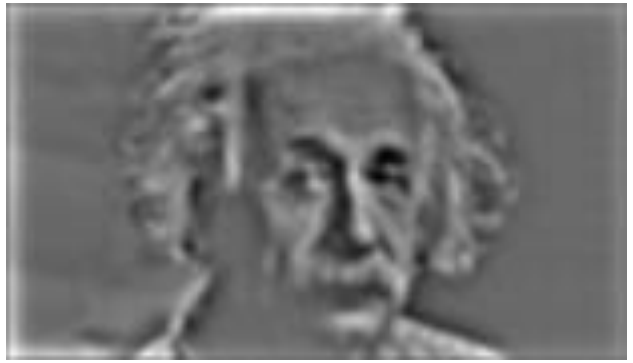


21x21, σ=1

21x21, σ=3







Demo: https://www.olympus-lifescience.com/en/microscope-resource/primer/java/digitalimaging/processing/diffgaussians/

Convolution Properties

Commutative: $a * b = b * a$

- Conceptually no difference between filter and signal but can be different based on implementation (HOW?)
- **Correlation is not commutative**

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: $a * (b_1 * b_2 * b_3)$ -> computationally faster
- **Correlation is not associative**

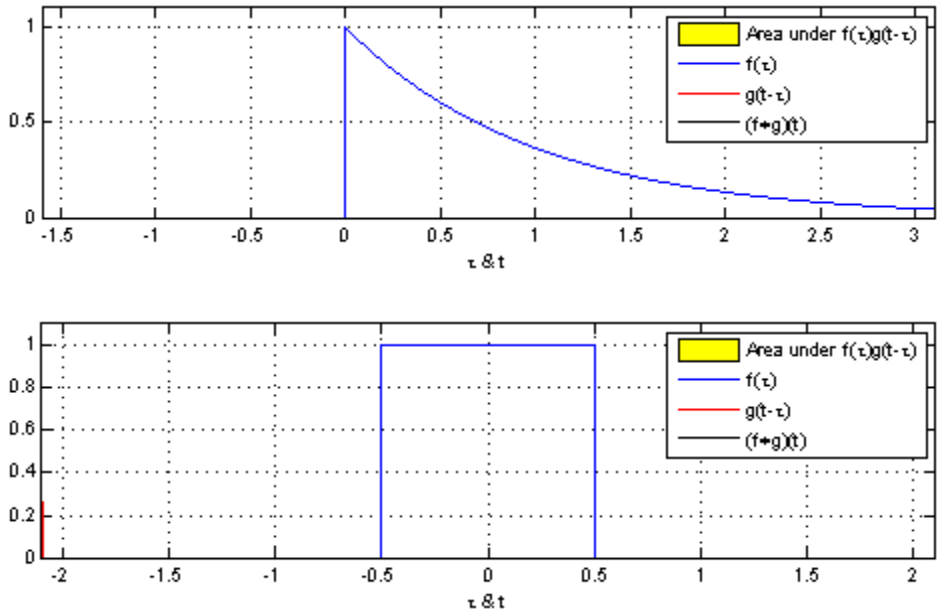
Distributes over addition: $a * (b + c) = (a * b) + (a * c)$

Scalars factor out: $ka * b = a * kb = k (a * b)$

Identity: $a * e = a$ when $e = [0, 0, 1, 0, 0]$,

Convolution is the basic operation in CNNs. Learning convolution kernels allows us to learn which `features' provide useful information in images. (Will discuss later…)

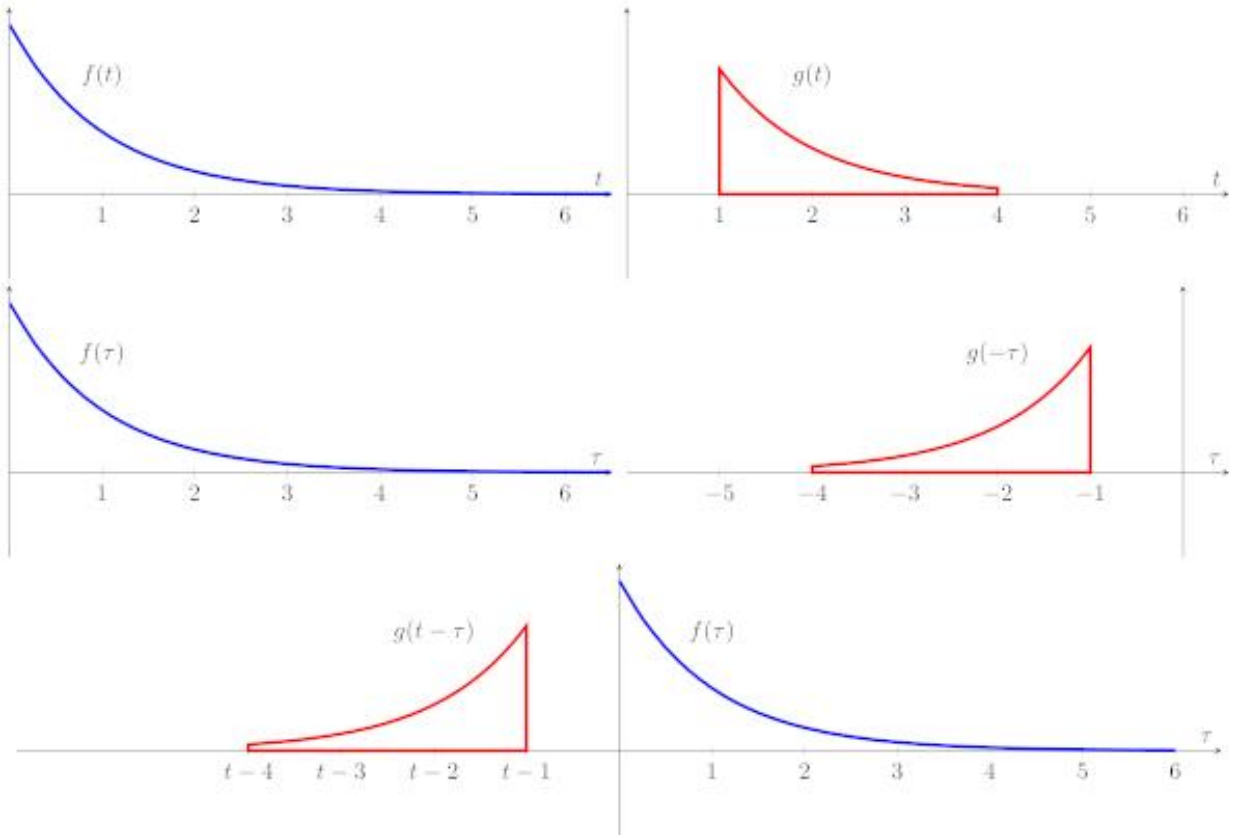
Convolution: Further insights



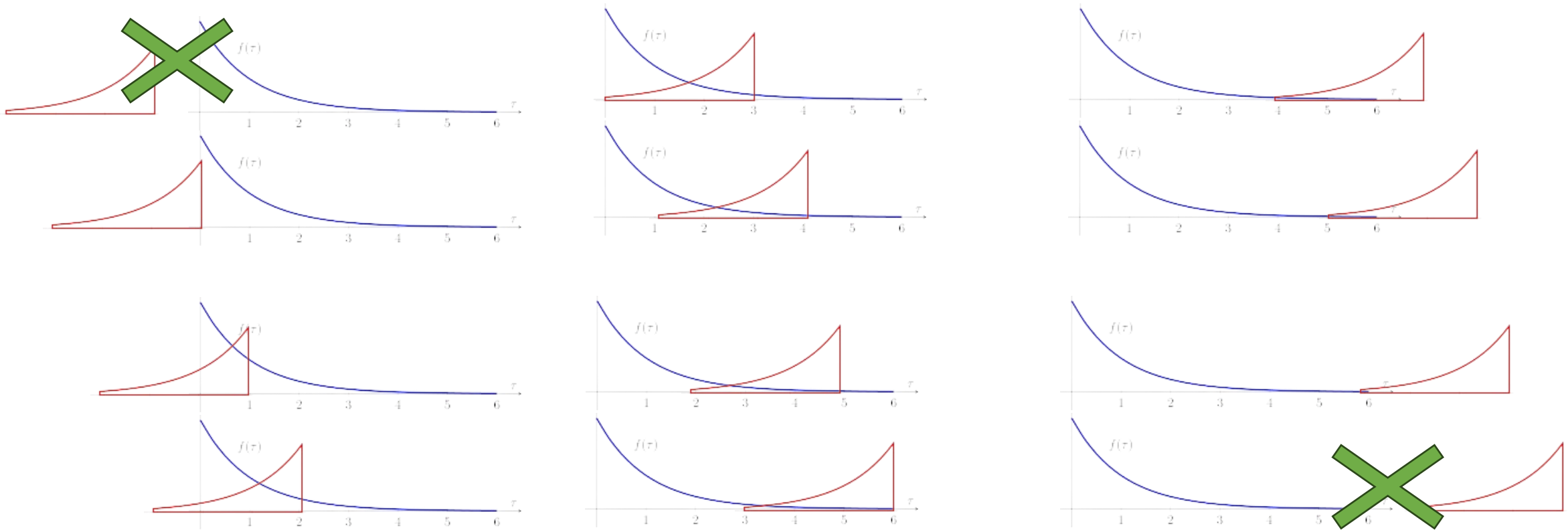
$$(f*g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau$$

To convolve a kernel with an output signal :

1. *flip the signal*
2. *move to the desired time*
3. *accumulate every interaction with the kernel*



Convolution: Further insights



Convolution: Further insights

$$(x * h)(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$

$$\Rightarrow (x * h)(t) = \int_{t+\infty}^{t-\infty} x(t-v)h(v)(-dv) \ (Change\ of\ variables)$$

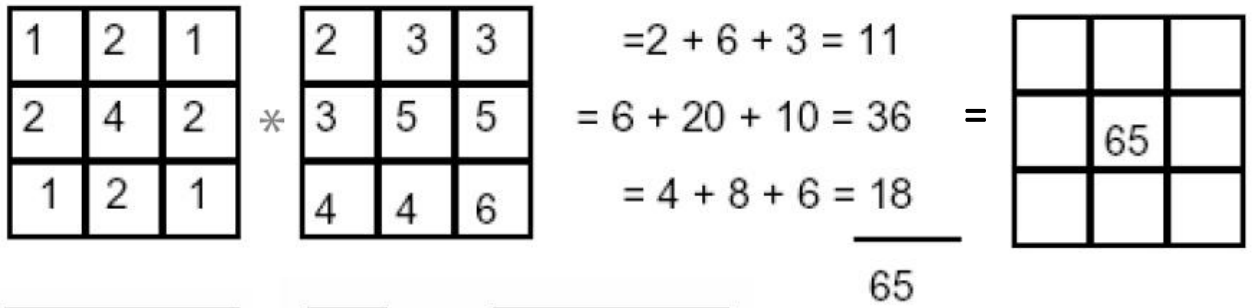
$$= \int_{\infty}^{-\infty} x(t-v)h(v)(-dv) \ (infinity\ dominates\ sum)$$

$$= \int_{-\infty}^{\infty} x(t-v)h(v)dv = \int_{-\infty}^{\infty} h(v)x(t-v)dv = (h * x)(t)$$

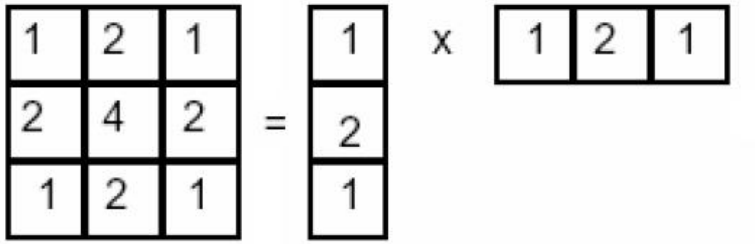
$$\begin{aligned}
((f \star g) \star h)(t) &= \int_0^t (f \star g)(s)h(t-s)\,ds \\
&= \int_{s=0}^{t} \left(\int_{u=0}^{s} f(u)g(s-u)\,du \right) h(t-s)\,ds \\
&= \iint_{0 \le u \le s \le t} f(u)g(s-u)h(t-s)\,du\,ds \\
&= \int_{u=0}^{t} \int_{s=u}^{t} f(u)g(s-u)h(t-s)\,ds\,du \\
&= \int_{u=0}^{t} f(u) \left(\int_{s=0}^{t-u} g(s)h(t-u-s)\,ds \right) du \\
&= \int_{u=0}^{t} f(u)(g \star h)(t-u)\,du \\
&= (f \star (g \star h))(t)
\end{aligned}$$

Separability Example

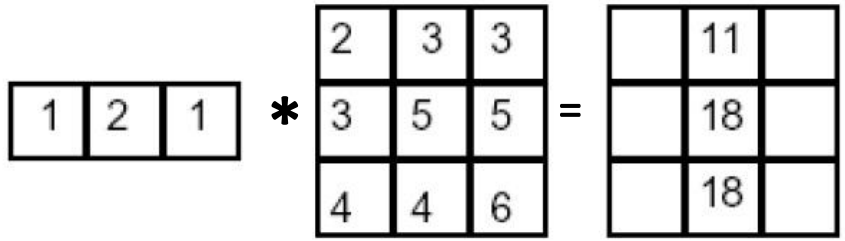
2D convolution
(center location only)



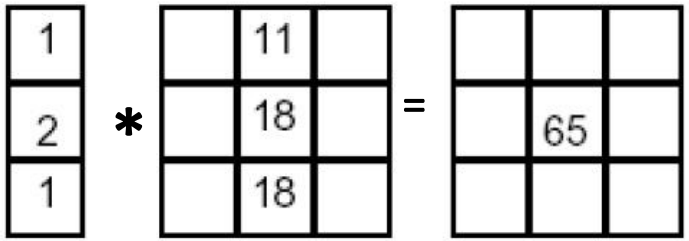
The filter factors
into a product of 1D
filters:



Perform convolution
along rows:



Followed by convolution
along the remaining column:



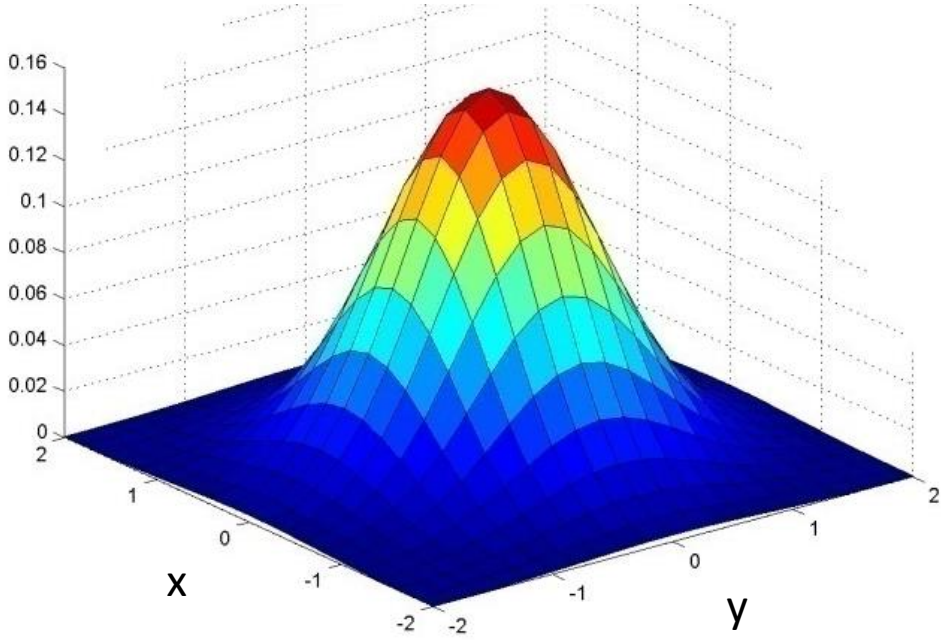
MxN image, PxQ filter

- 2D convolution: ~MNPQ multiply-adds
- Separable 2D: ~MN(P+Q) multiply-adds

Speed up = PQ/(P+Q)

9x9 filter = ~4.5x faster

Gaussian Filter



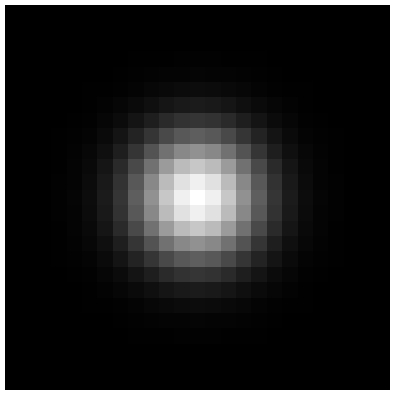
x

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

y

Kernel size 5 x 5,
Standard deviation $\sigma = 1$

Viewed from top



y

x

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

Ignore factor in front, instead, normalize filter to sum to 1

Gaussian Filter Properties

Gaussian convolved with Gaussian is another Gaussian

- So can smooth with small-width kernel, repeat, and get same result as larger-width kernel

Separable kernel: Factors into product of two 1D Gaussians

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

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

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

How big should the Gaussian filter be?

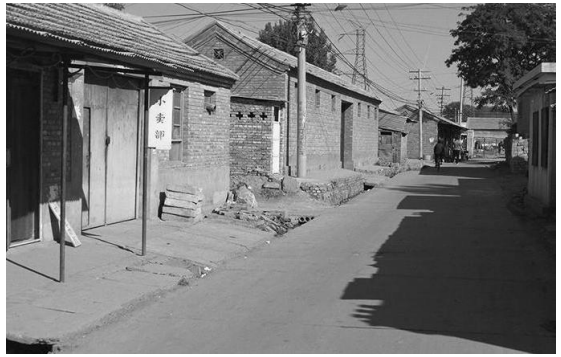
- Values at edges should be near zero
- Gaussians have infinite extent…
- Rule of thumb for Gaussian: set filter half-width to about 3 σ

Review

1) G = D * B
2) A = B * C
3) F = D * E
4) I = D * D

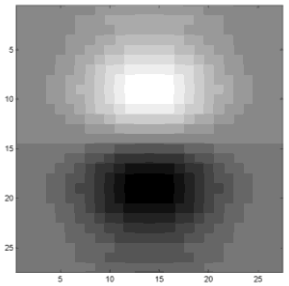
* = Convolution operator

D

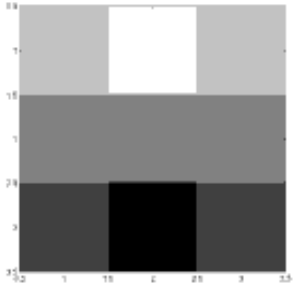


H

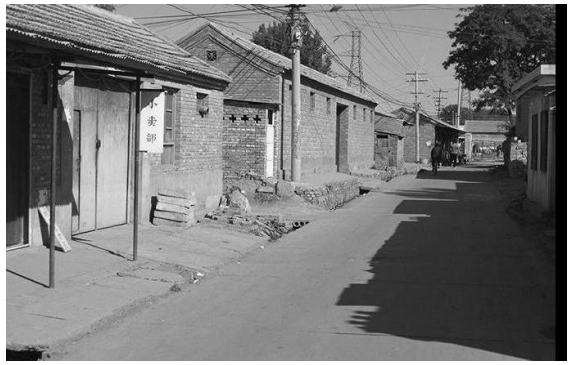
A



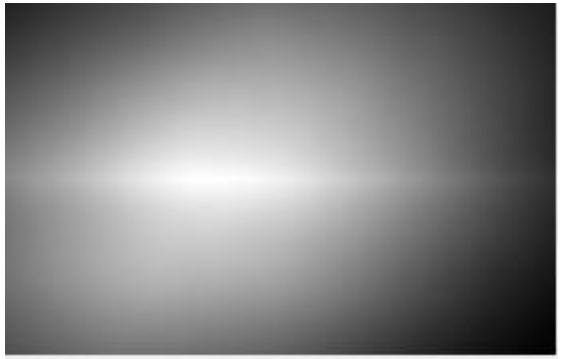
B



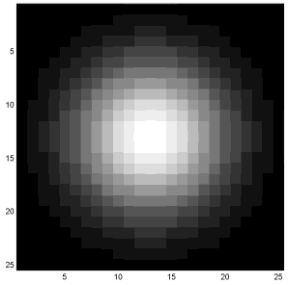
F



I

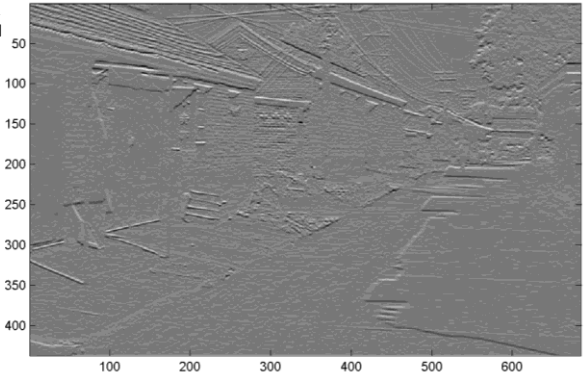


C



E

G



Modified from: Derek Hoiem

Review: questions

1. Write down a 3x3 filter that returns a positive value if the average value of the 4-adjacent neighbors is less than the center and a negative value otherwise

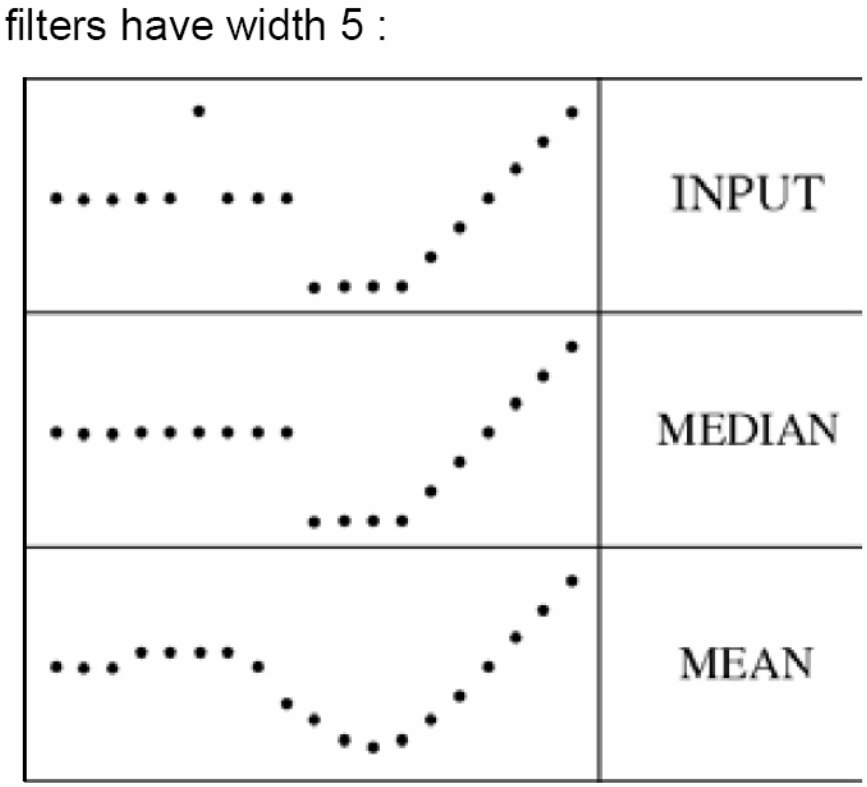
2. Write down a filter that will compute the gradient in the x-direction:

```
gradx(y,x) = im(y,x+1)-im(y,x) for each x, y
```

Slide: Hoiem

Non-linear filter: Median filters

- Operates over a window by selecting the median intensity in the window.
- What advantage does a median filter have over a mean filter?
  - Robustness over outliers
- Is a median filter a kind of convolution?
- Other non-linear filters, aka ‘Rank’ filter as based on ordering of gray levels
  - Like, min, max, range filters

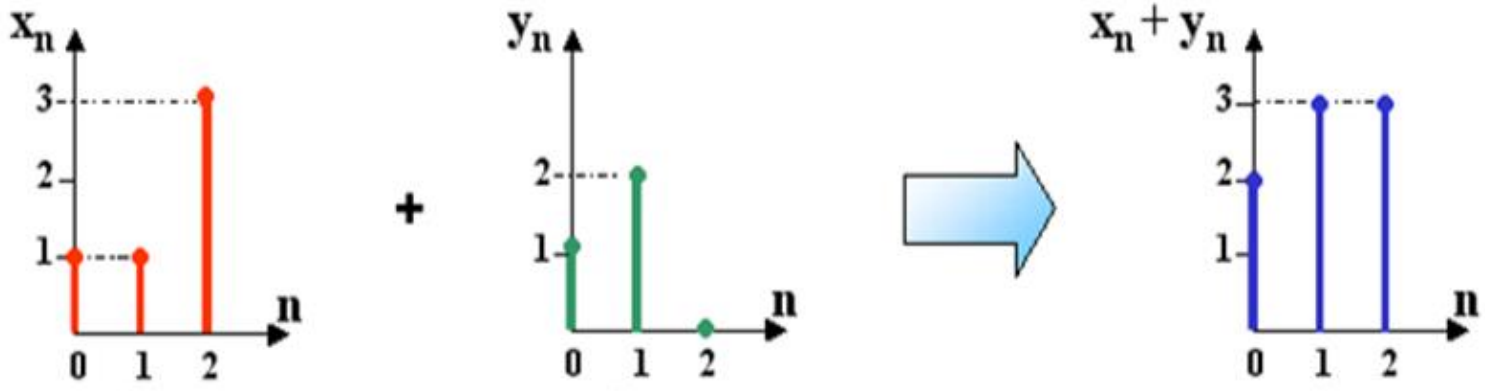


Linear vs non-linear filters

Consider two signals A and B, for linear filter such as mean filter f, we have $f(A+\lambda B)=f(A)+\lambda f(B)$.

It is not satisfied for an nonlinear filter such as the median filter.

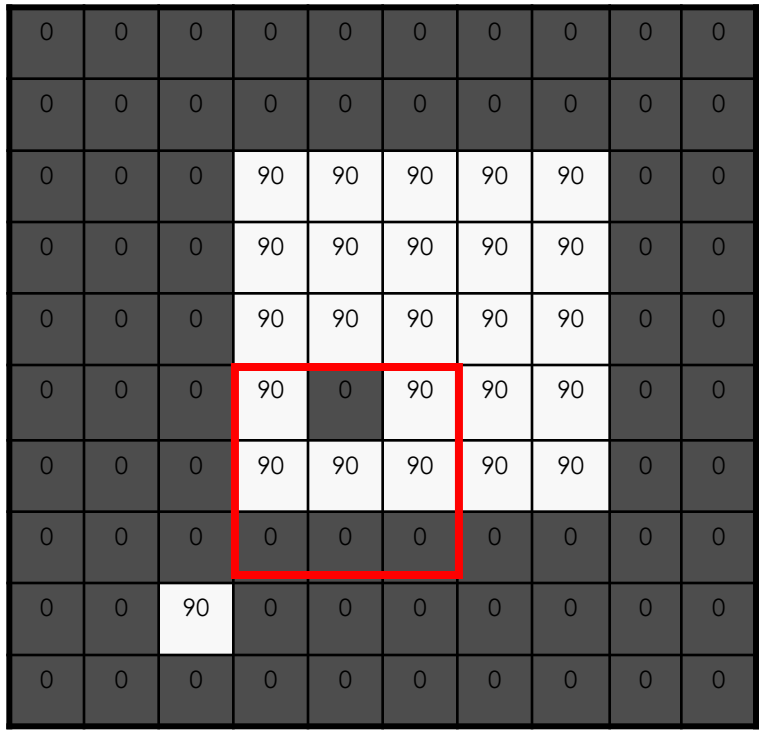
For mean filter:
$\text{mean}(X_n) = 1+1+3/3=5/3$
$\text{mean}(Y_n) = 1+2+0/3=1$
$\text{mean}(X_n+Y_n) = 2+3+3/3=8/3$
$\text{mean}(X_n)+\text{mean}(Y_n)=\text{mean}(X_n+Y_n)$ [5/3+1=8/3]
hence, mean filter is a linear filter.



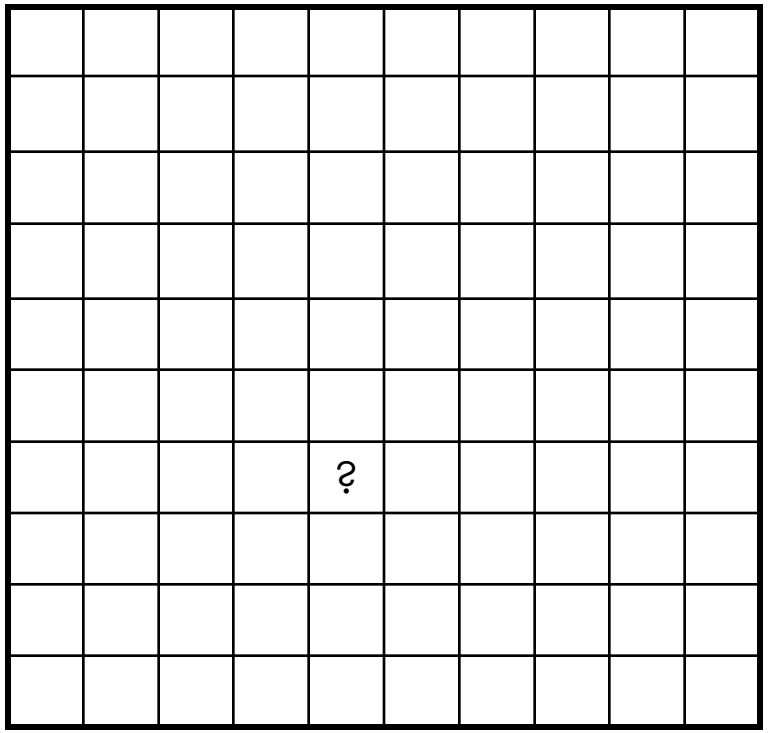
For median filter:
$\text{median}(X_n)= 1$
$\text{median}(Y_n)= 1$
$\text{median}(X_n+ Y_n)= 3$
$\text{median}(X_n)+\text{median}(Y_n)\neq\text{median}(X_n+Y_n)$ [1+1≠3]
hence, median filter is a non-linear filter

Median Filter

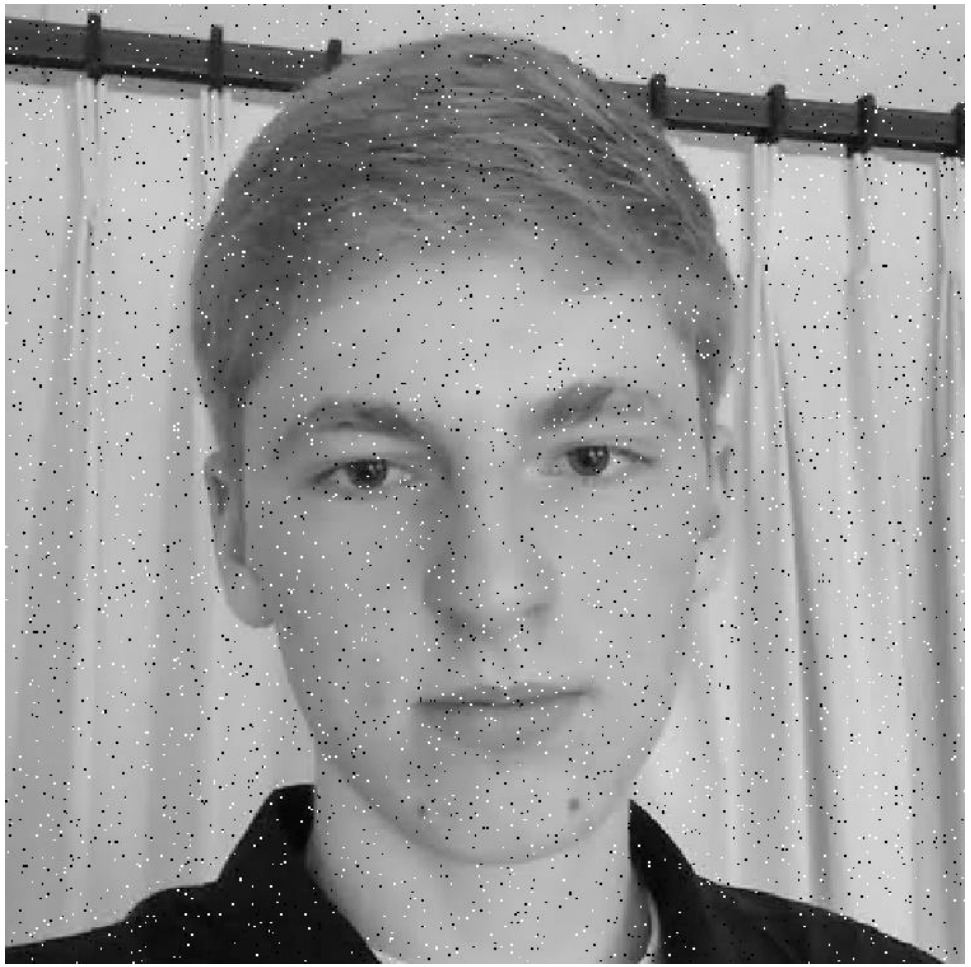
$I[.,.]$



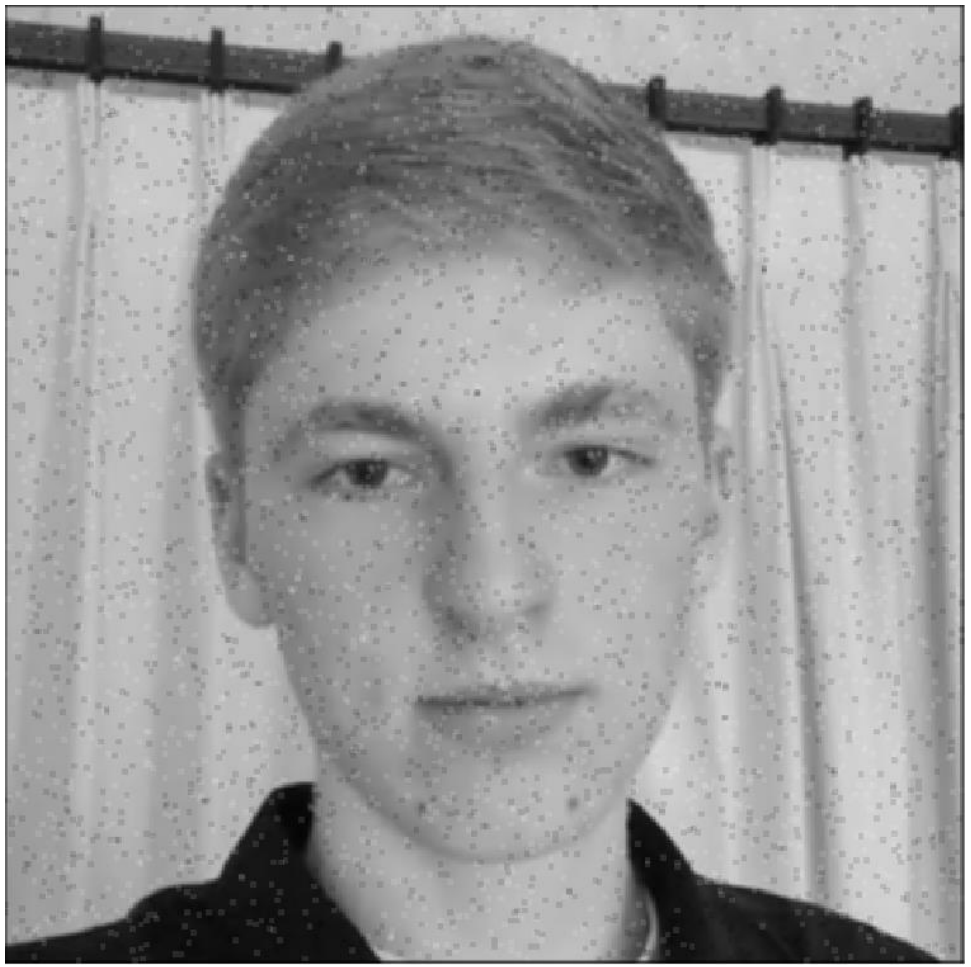
$h[.,.]$



Salt and Pepper Noise

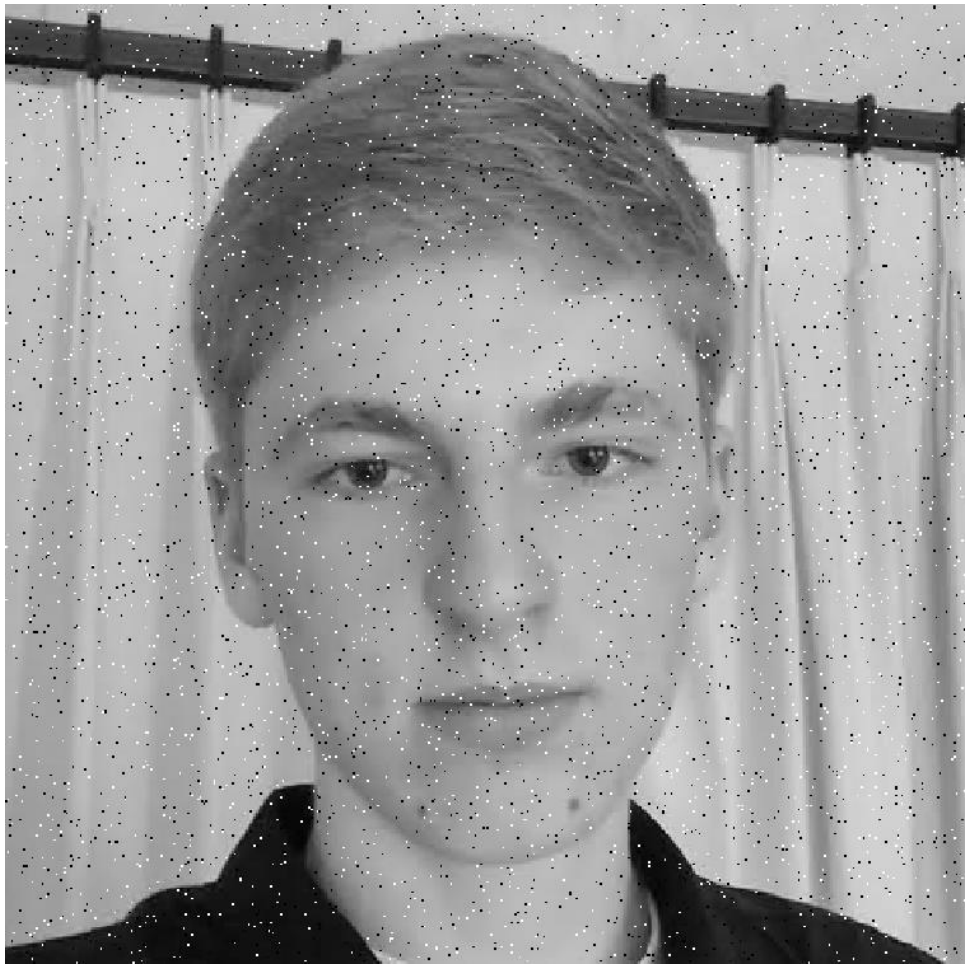


3 x 3 Mean Filter

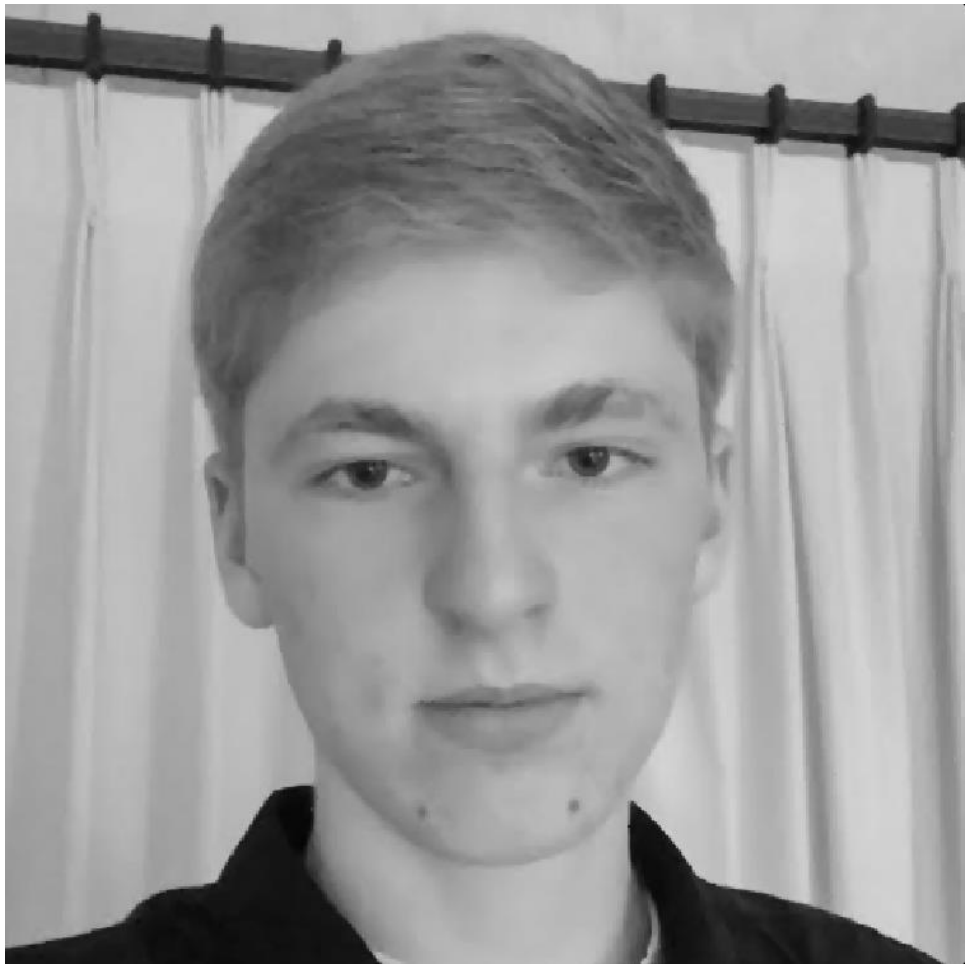


11 x 11 Mean Filter

Salt and Pepper Noise



3 x 3 Median Filter



11 x 11 Median Filter

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

Image: http://vision.ai.uiuc.edu/?p=1455

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.

Other non-linear filters: Thresholding

$$g(m,n) = \begin{cases} 255, & f(m,n) > A \\ 0 & otherwise \end{cases}$$

Other non-linear filters: Rectification

- $g(m,n) = \max(f(m,n), 0)$

- Crucial component of modern convolutional networks

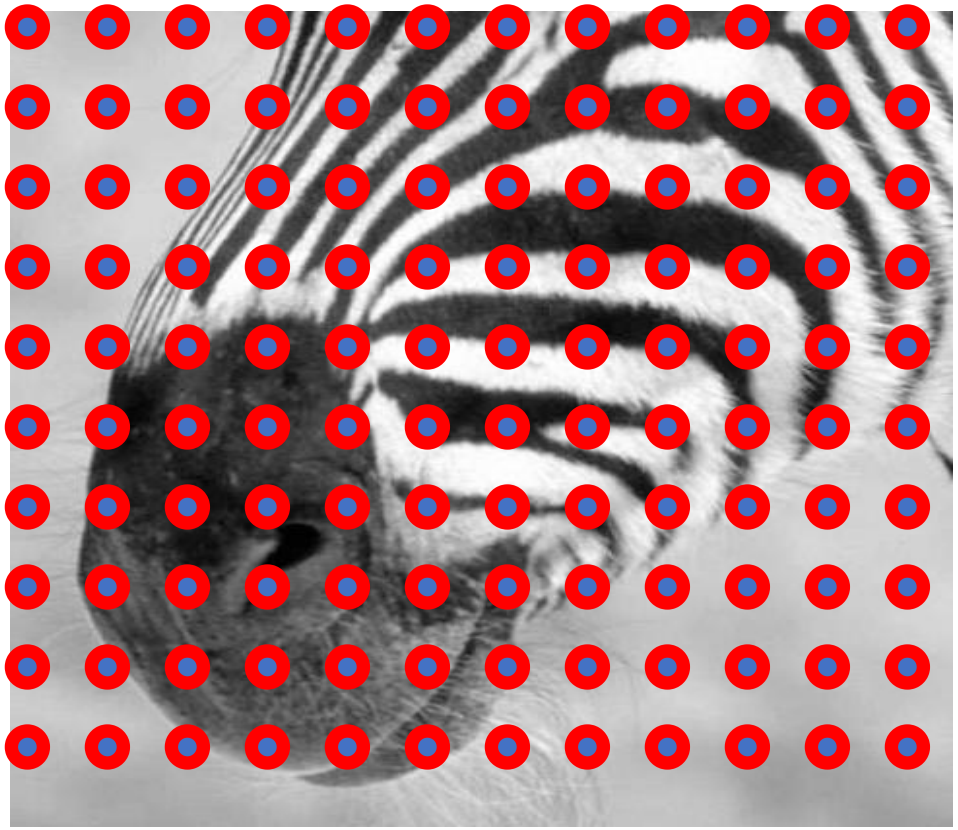
Image Pyramids

Sampling

Why does a lower resolution image still make sense to us? What do we lose?

Image: http://www.flickr.com/photos/igorms/136916757/

Resizing: Making lower resolution images:
Subsampling by a factor of 2



Throw away every other row and column to create a 1/2 size image

Downsampling is useful when we need to reduce storage space.
Lower resolution image still make sense.
What information do we lose?

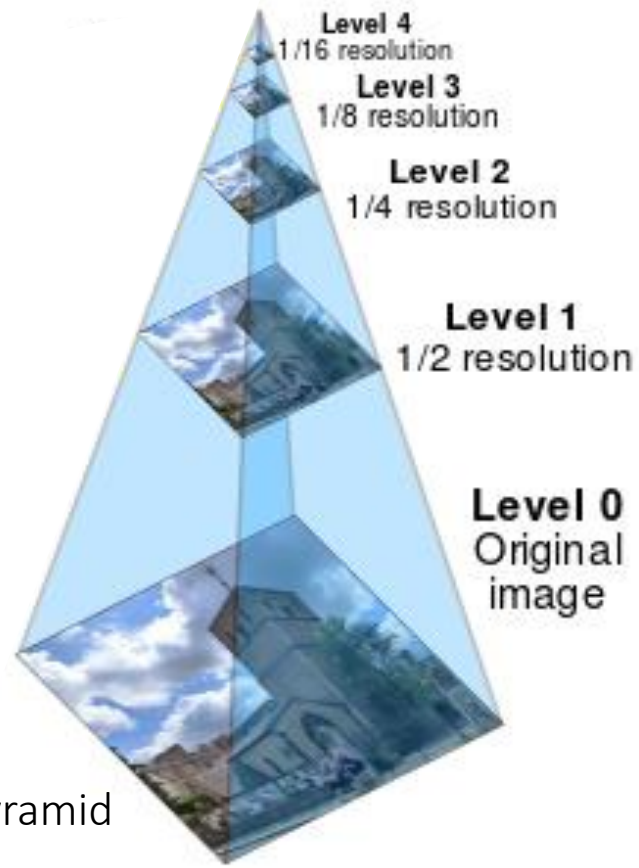


Image Pyramid

Downsampling

Algorithm

1. Start with image of w x h
2. Sample every other pixel
3. `im_small = image[ ::i, ::i ]`
4. To build a pyramid,
   repeat Steps 1 & 2
   until `im_small` is 1 pixel large.

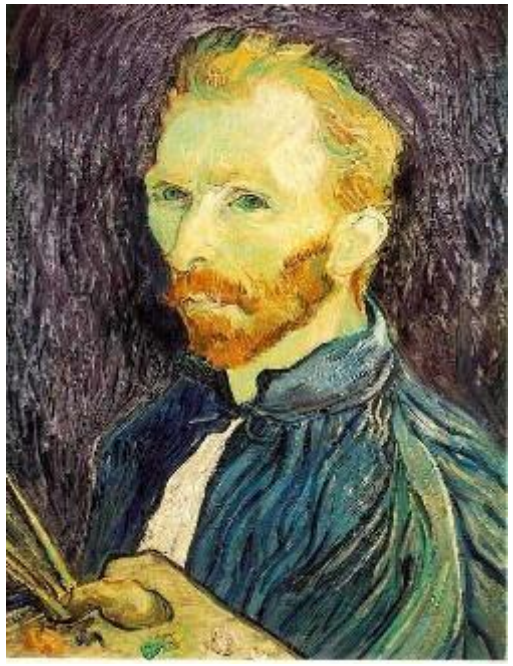
NumPy Syntax

i = 2
::2 -> start at 0,
end at 'end',
increase every 2, until
the end.

e.g.,
0,2,4,6,…,w

(if w is not even, then
this goes to w-1)

Image Subsampling

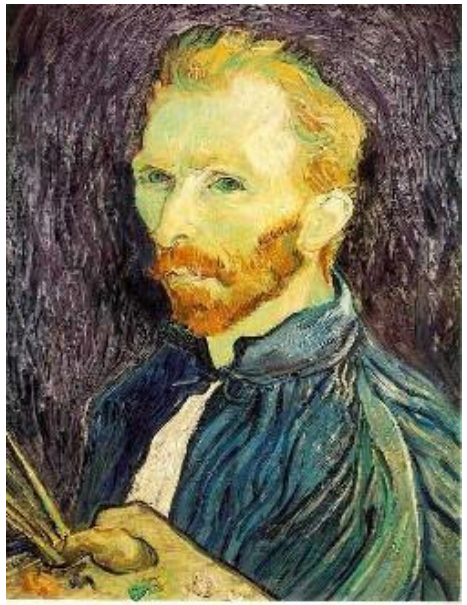


Throw away every other row and column to create a 1/2 size image.

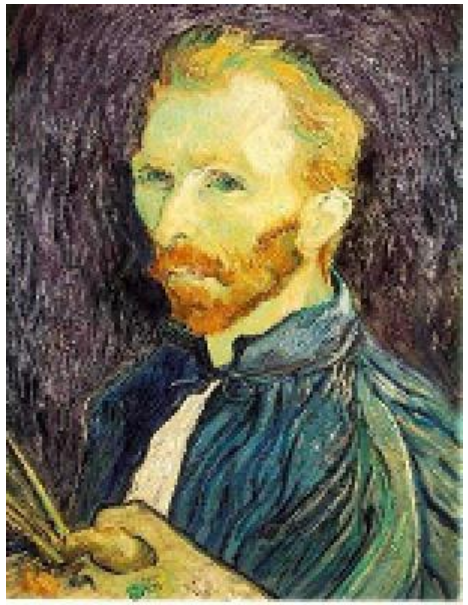
1/4

1/8

Subsampling without Filtering



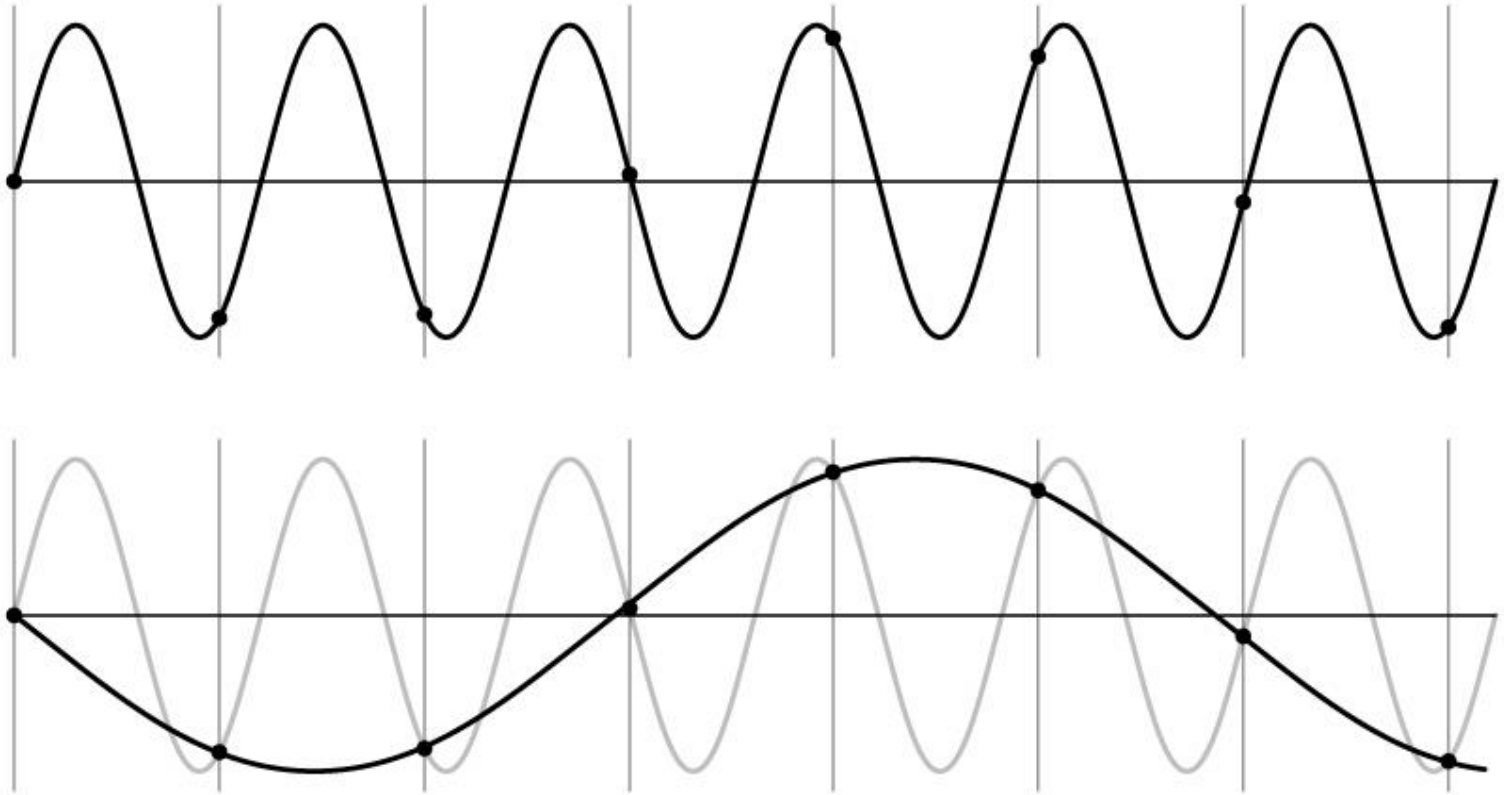
1/2



1/4 (2x subsample)

1/8 (4x subsample)

Aliasing



Aliasing occurs when we sample a signal at a frequency that is too low such that we can't properly reconstruct the original signal.

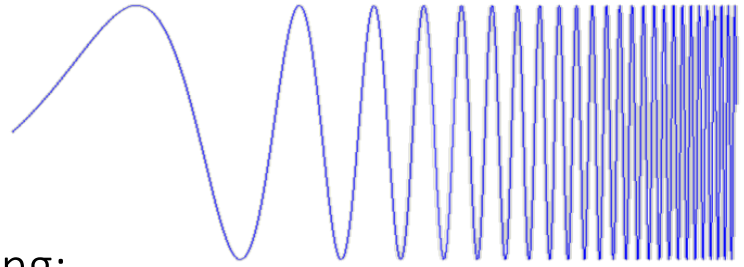
If we reconstructed the signal using just the sampled points (black dots above), we'd get a completely different-looking reconstruction than the original signal.

Intuitively, we don't have enough information about the signal because we sampled it too infrequently.

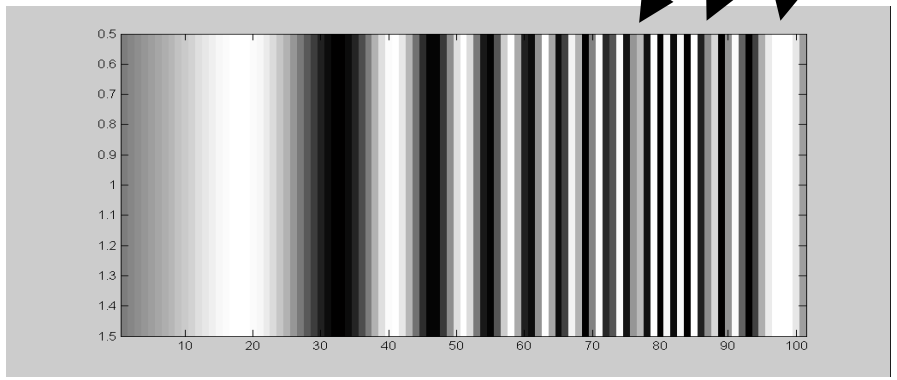
What's happening?

Aliasing!
Not enough samples

Input signal:



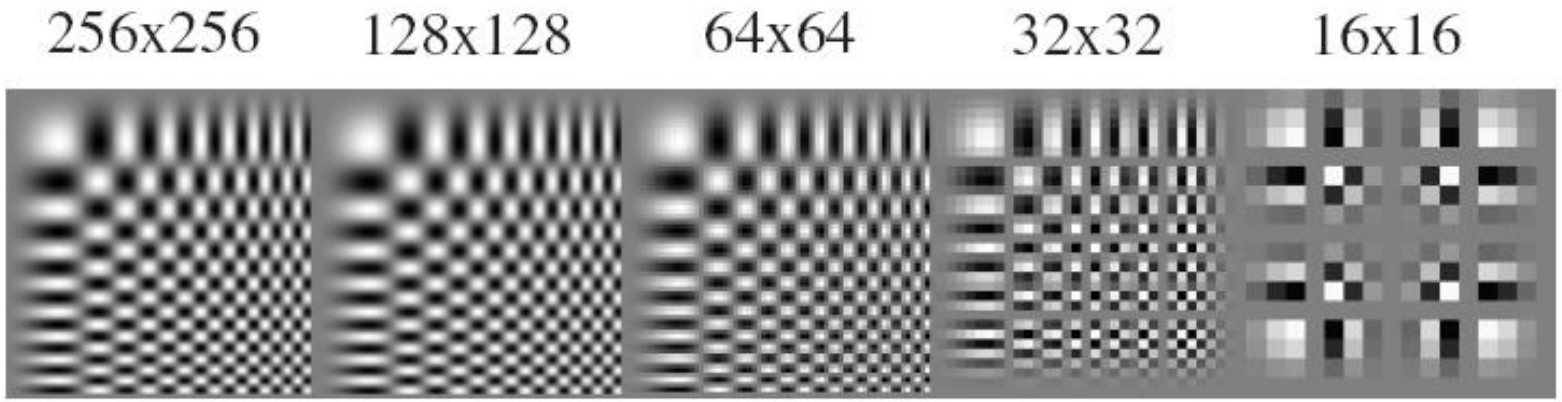
Plot as image:




```
x = 0:.05:5;  plt.imshow(sin((2.^x).*x))
```

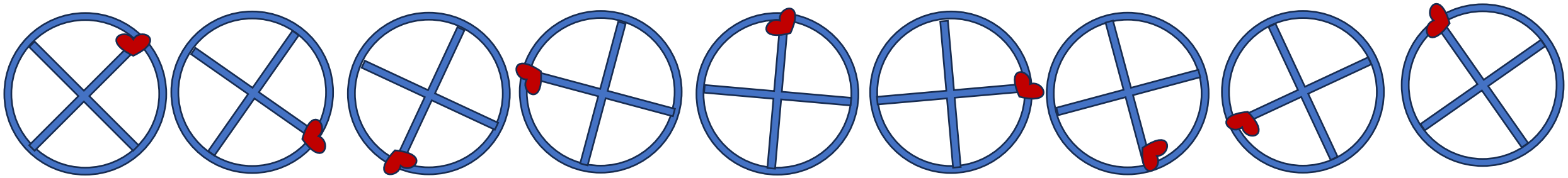
Aliasing:

- When two signals become indistinguishable from one another due to sampling.
- They are 'aliases' of one another.

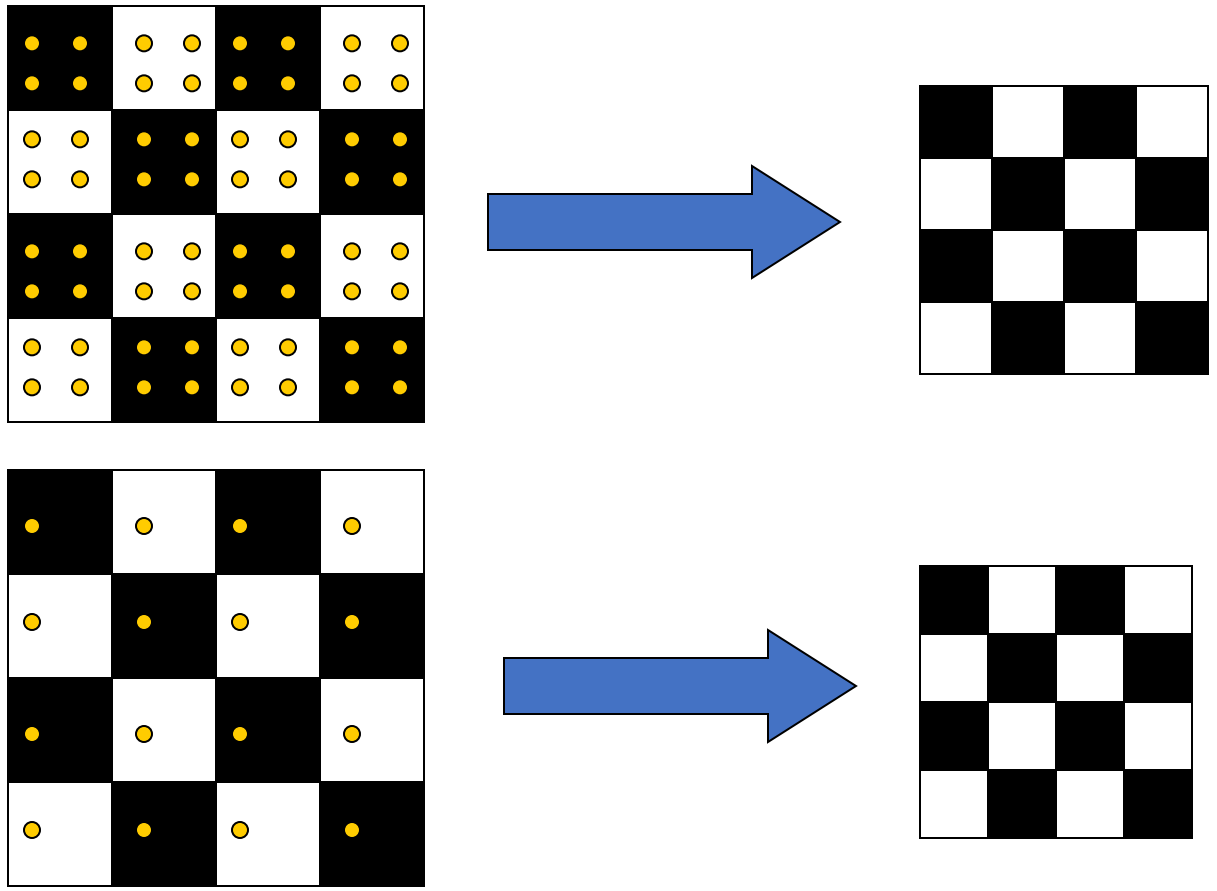


https://www.youtube.com/watch?v=Y1yHMy0-4TM
https://www.youtube.com/watch?v=jQDjJRYmeWg

Aliasing in video

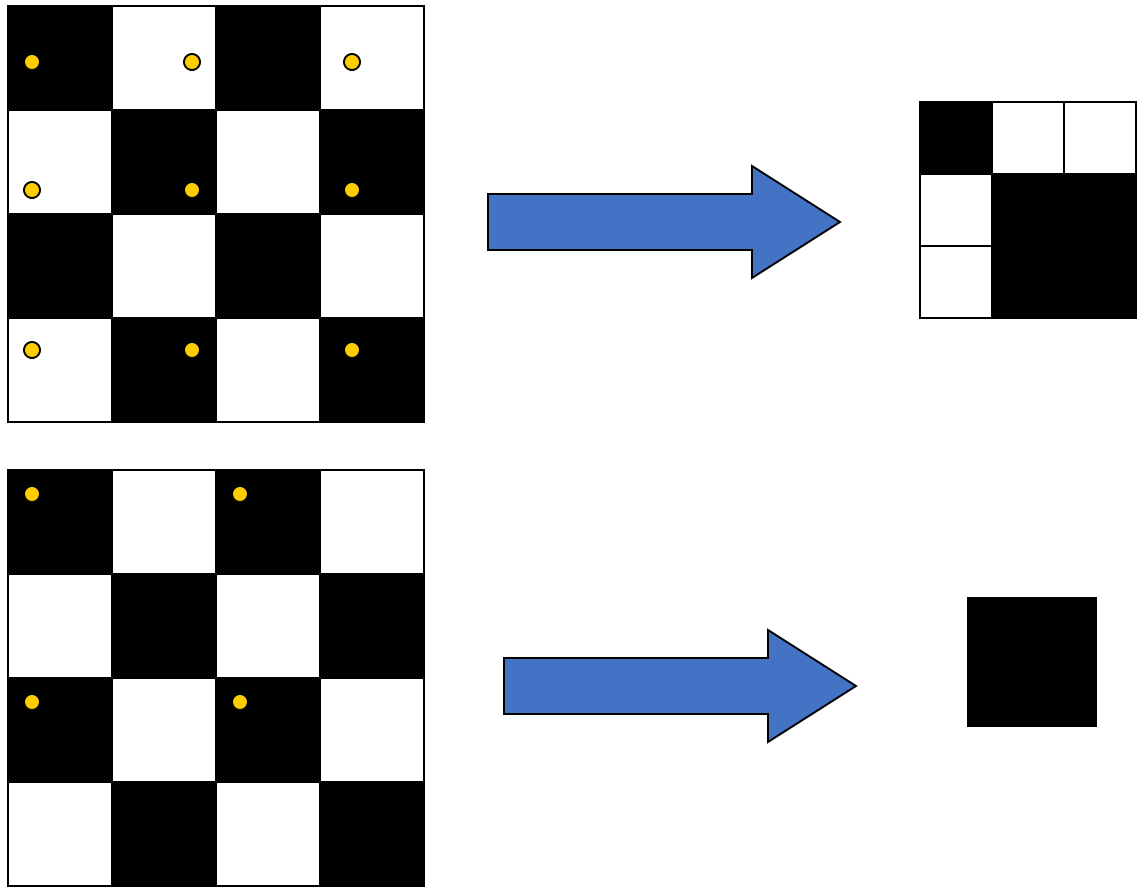


Sampling an image



Examples of GOOD sampling

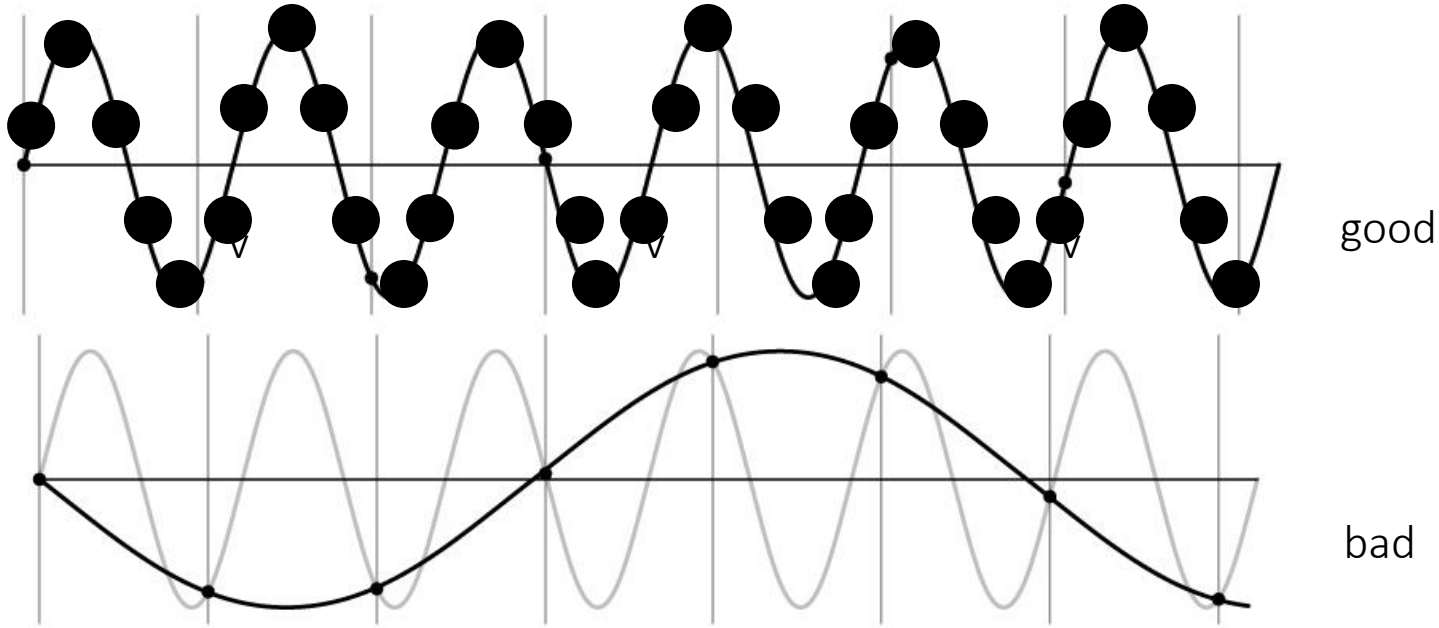
Undersampling



Examples of BAD sampling -> Aliasing

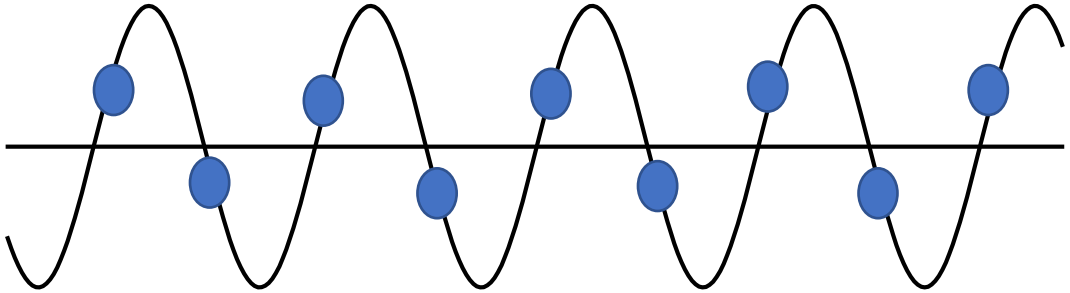
Nyquist-Shannon Sampling Theorem

- When sampling a signal at discrete intervals, the sampling frequency must be $\geq 2 \times f_{max}$
- f_{max} = max frequency of the input signal
- This allows us to reconstruct the original perfectly from the sampled version



How many samples do we need?

- 2 samples per time-period is enough



- Nyquist sampling theorem: Need to sample at least 2 times the frequency
- General signals? Need to sample at least 2 times the maximum frequency

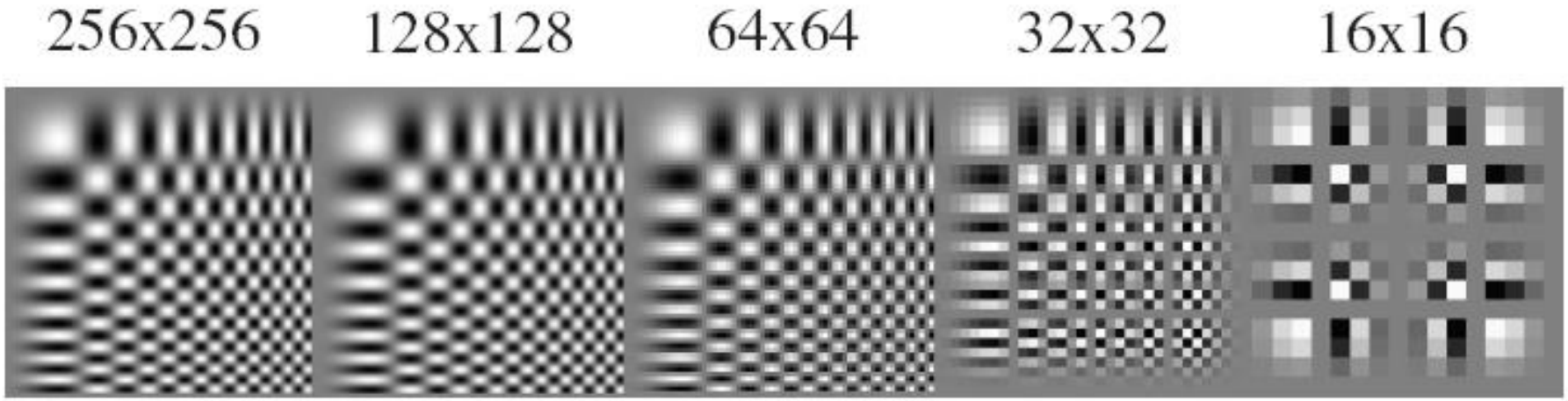
Anti-aliasing

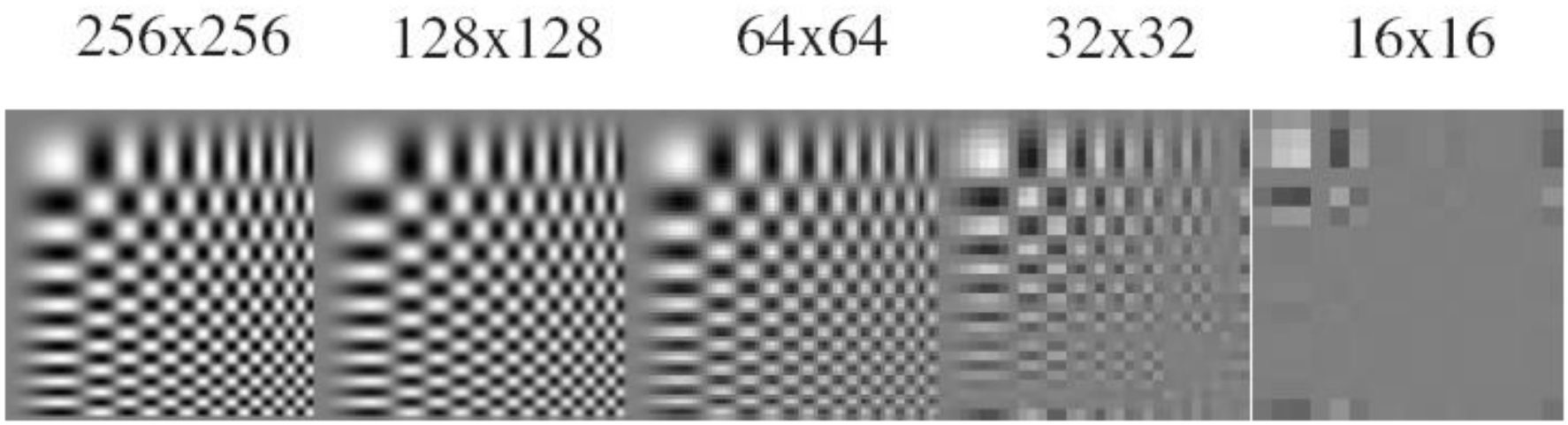
Solutions:

- Sample more often

- Remove all frequencies that are greater than half the new sampling frequency
  - Will lose information
  - But it’s better than aliasing
  - How?

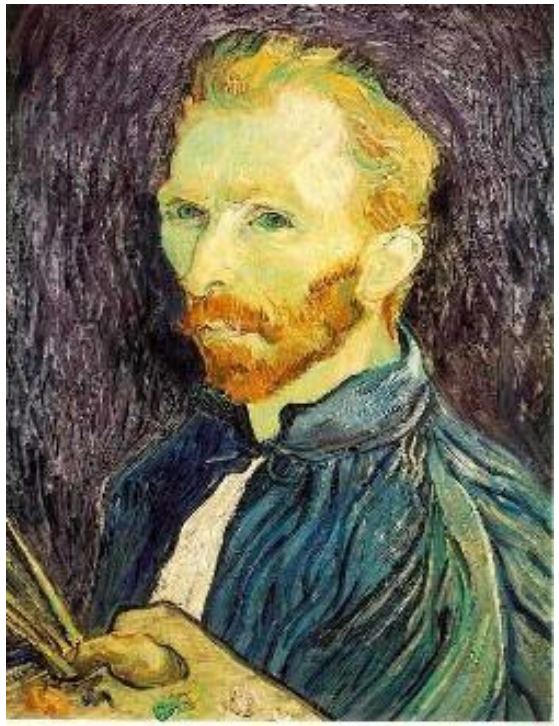
Anti-aliasing



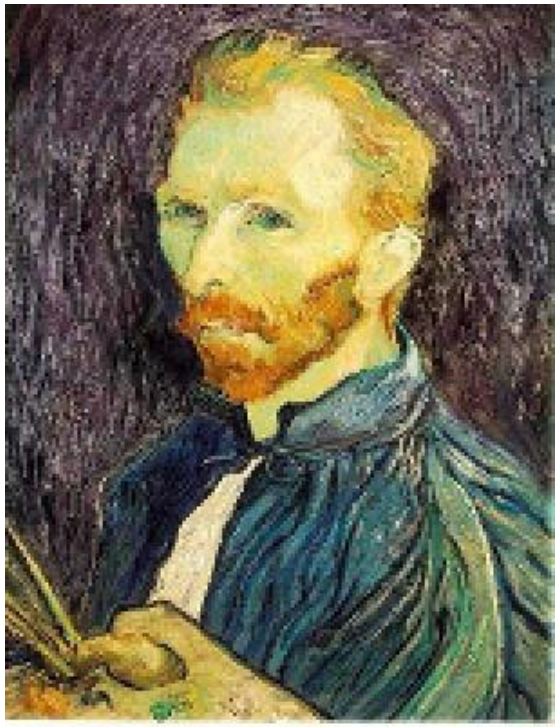


Forsyth and Ponce 2002

Subsampling without Filtering



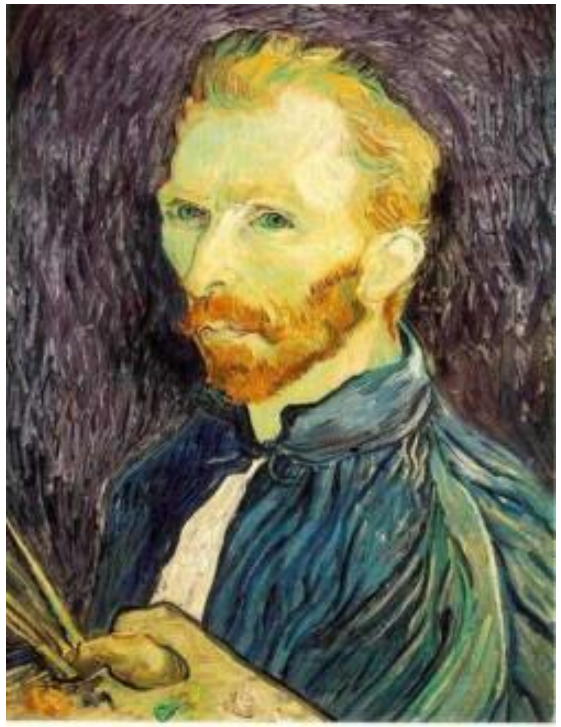
1/2



1/4 (2x subsample)

1/8 (4x subsample)

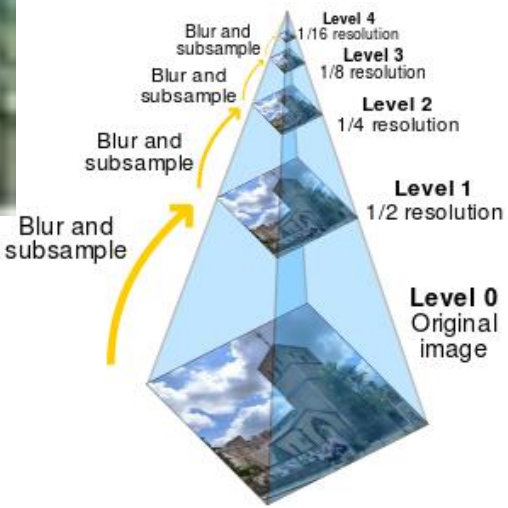
Subsampling with Gaussian Pre-filtering



Gaussian 1/2

G 1/4

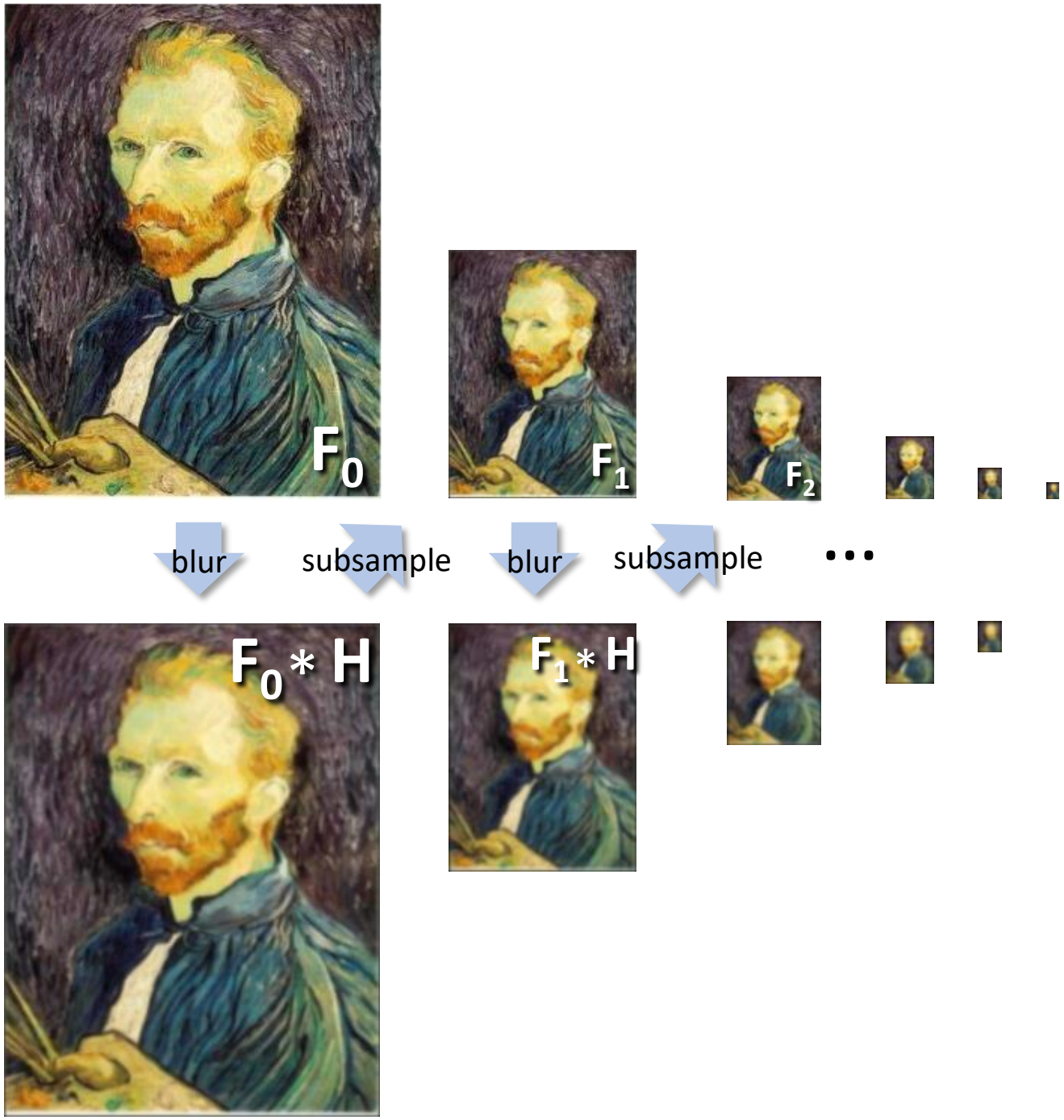
G 1/8



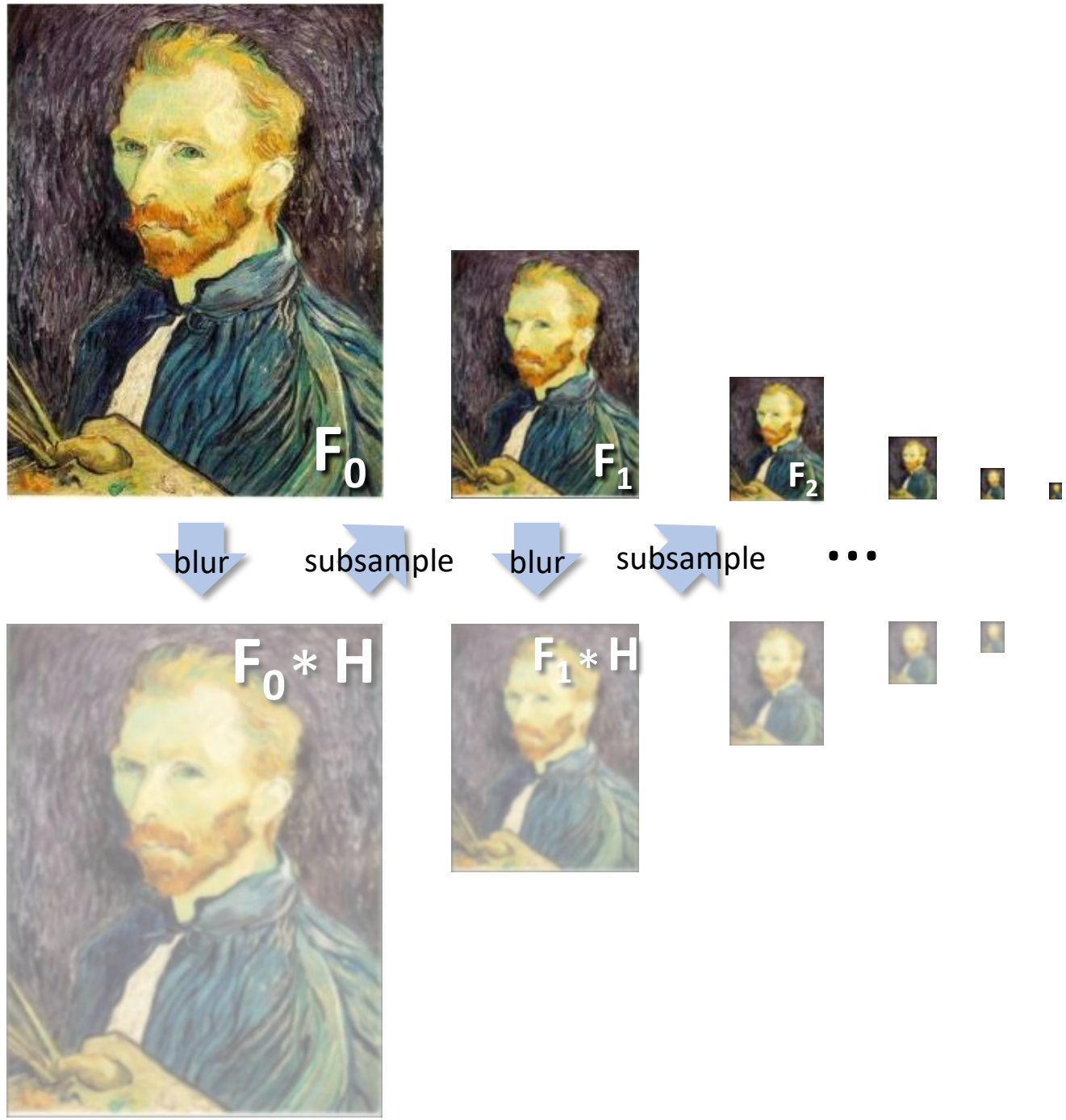
Gaussian Pyramid. Burt and Adelson, 1983

Gaussian pre-filtering

- Solution: filter the image, *then* subsample



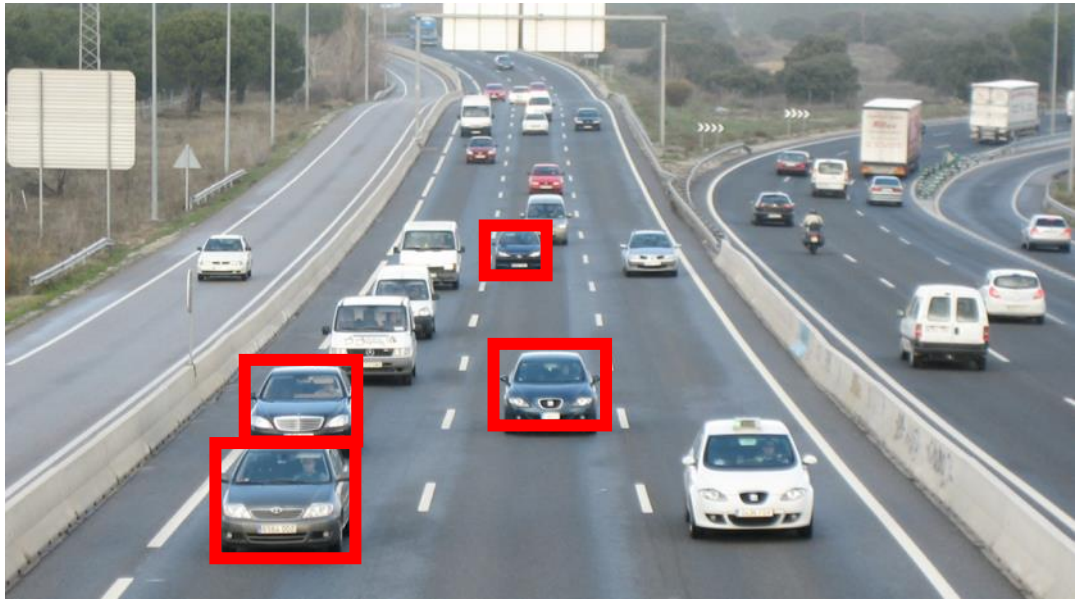
Gaussian pyramid



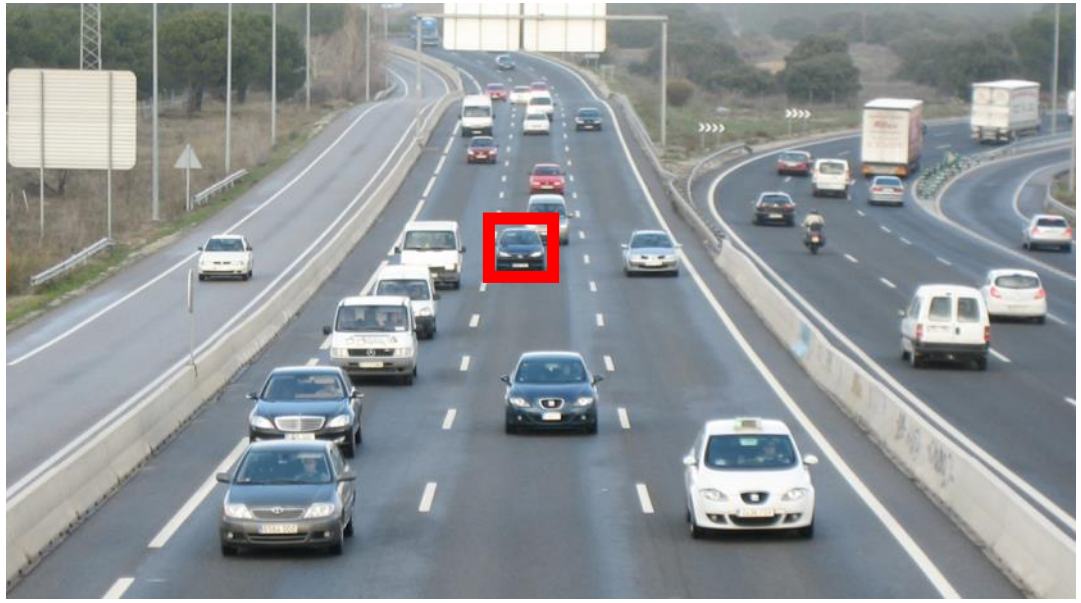
Major uses of image pyramids

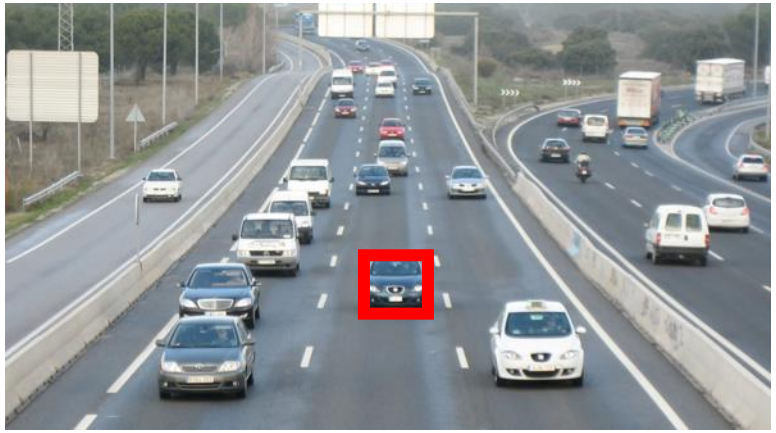
- Object detection
  - Scale search
  - Features
- Detecting stable interest points
- Course-to-fine registration
- Compression

Gaussian pyramids - Searching over scales

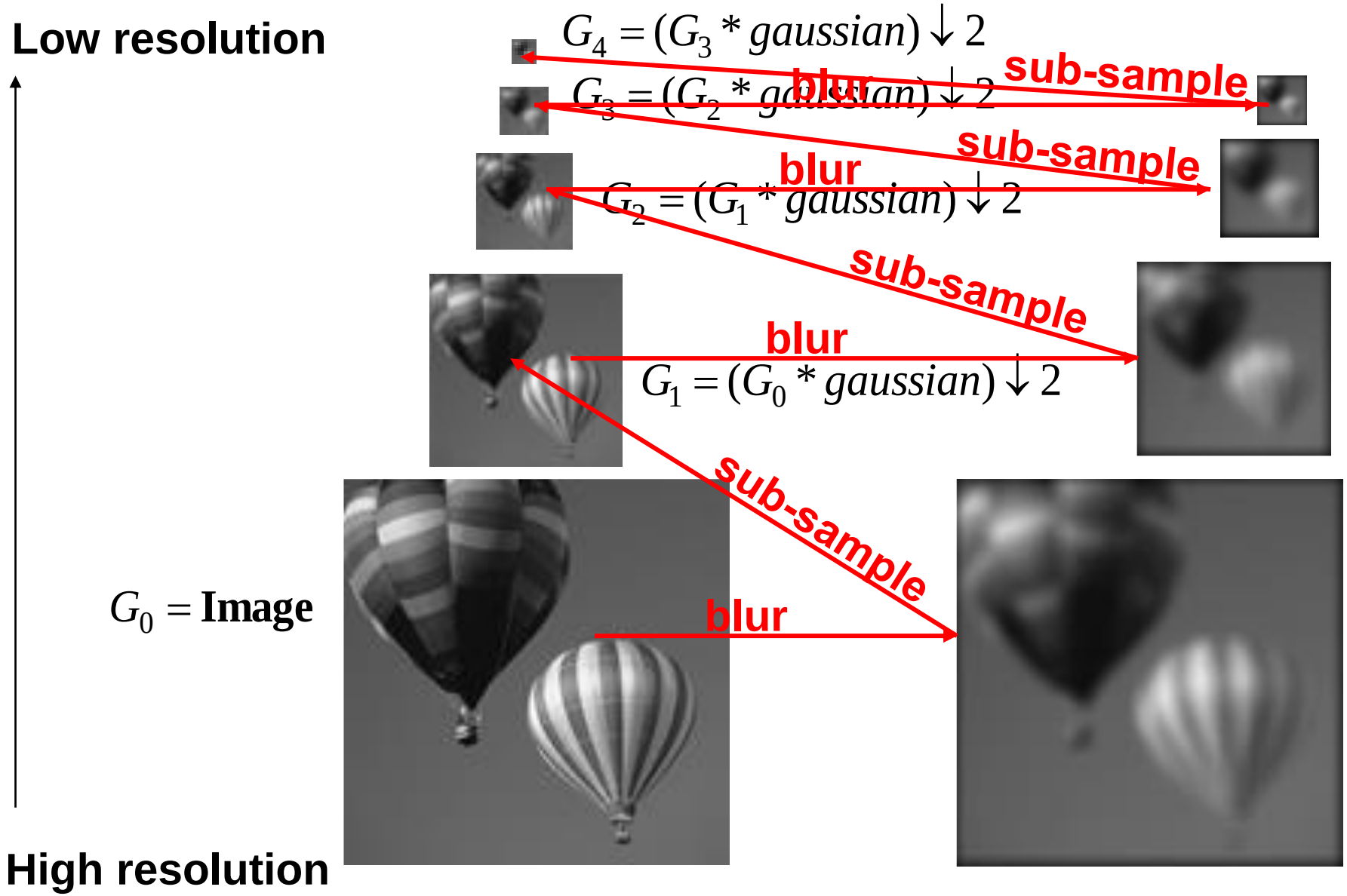


Gaussian pyramids - Searching over scales

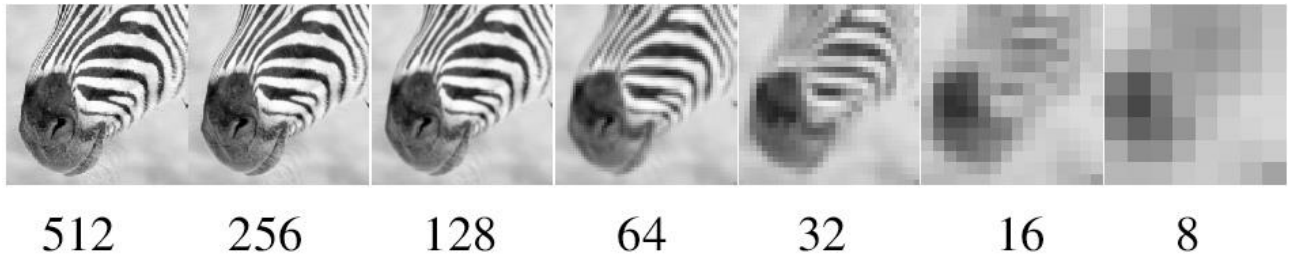




The Gaussian Pyramid



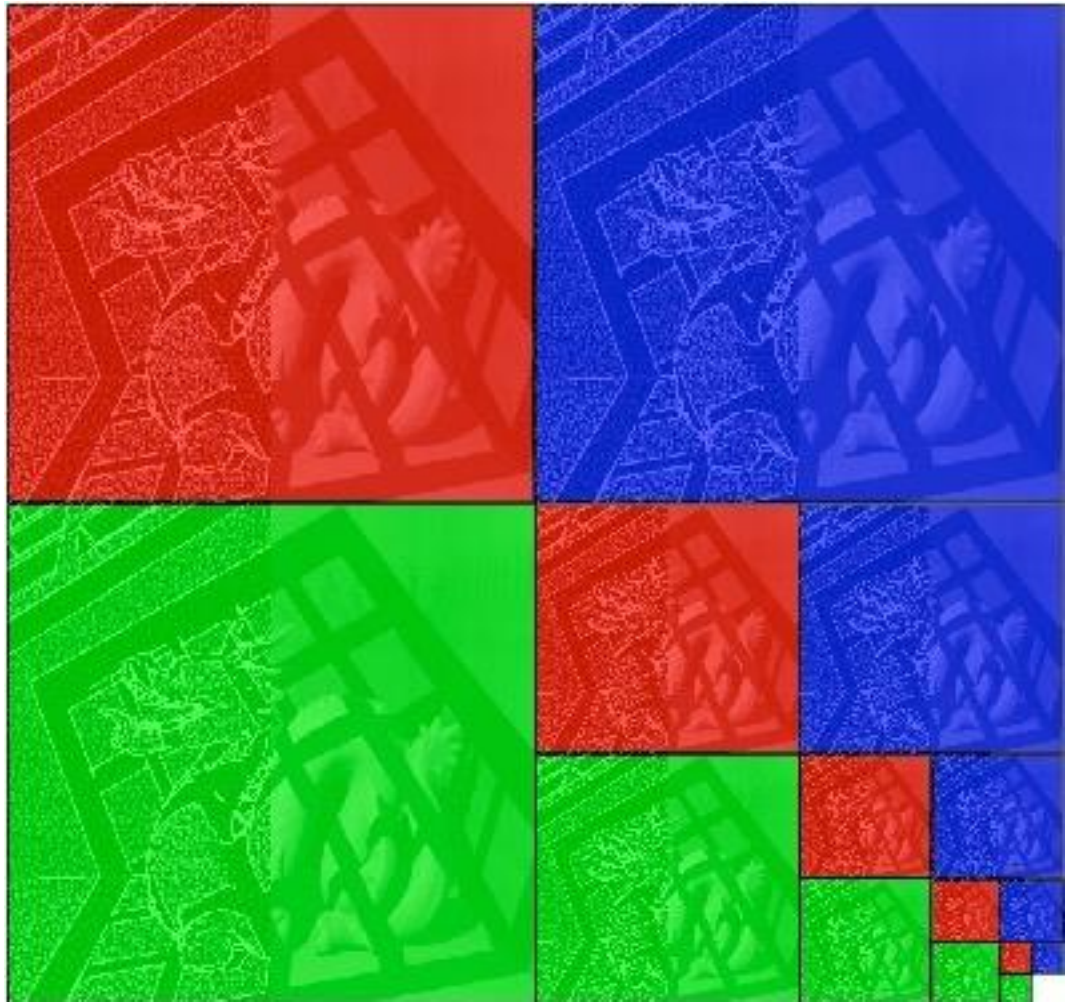
Gaussian pyramid and stack

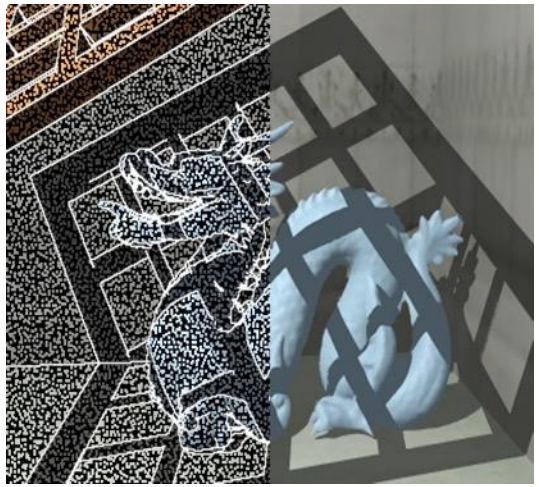


Source: Forsyth

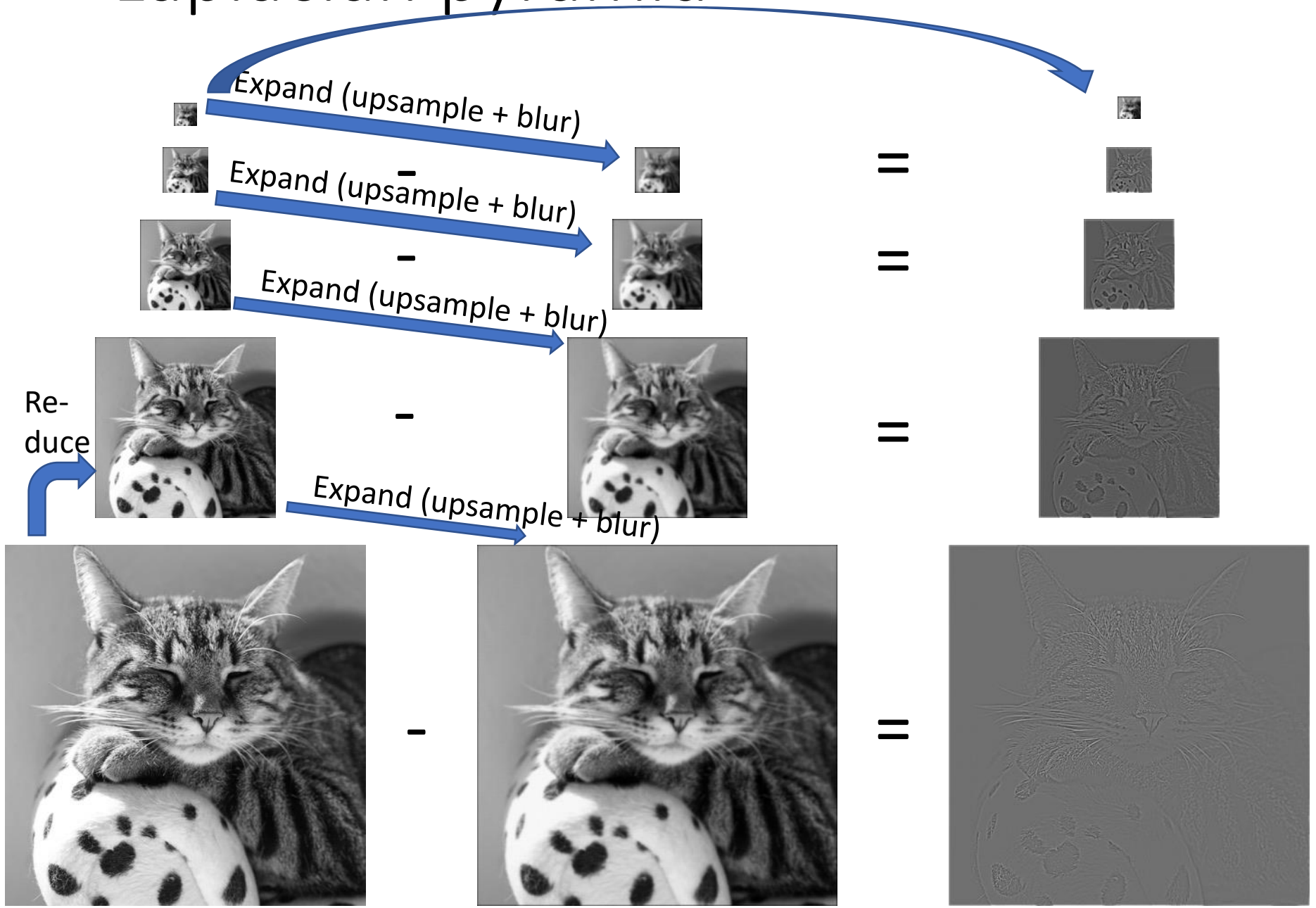
Memory Usage

- What is the size of the pyramid?





Laplacian pyramid



Laplacian pyramid

$L_4 = G_4 =$ 

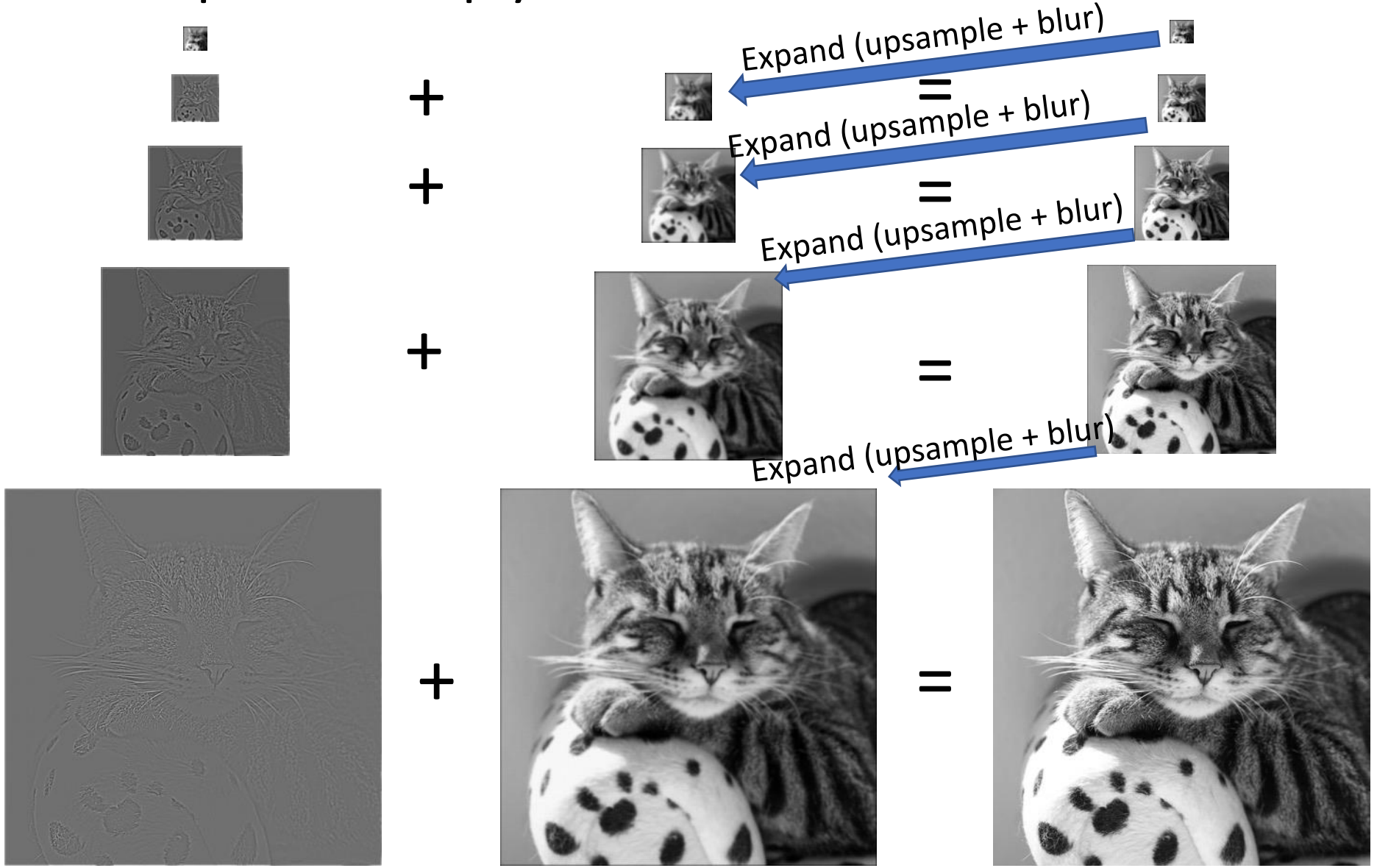
$L_3 = G_3 - \text{expand}(G_4) =$ 

$L_2 = G_2 - \text{expand}(G_3) =$ 

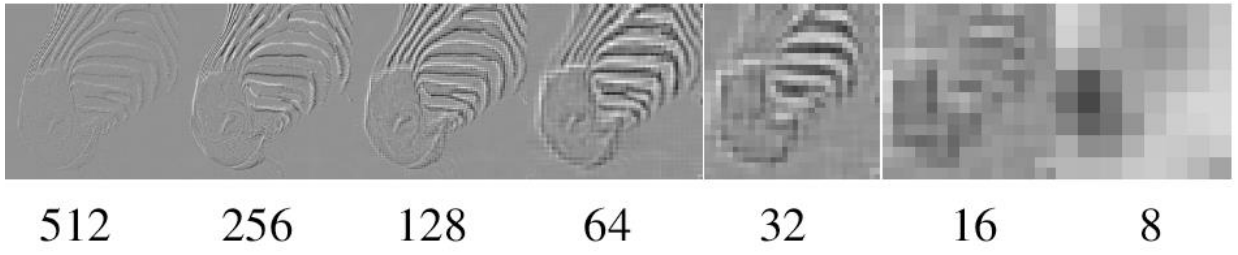
$L_1 = G_1 - \text{expand}(G_2) =$ 

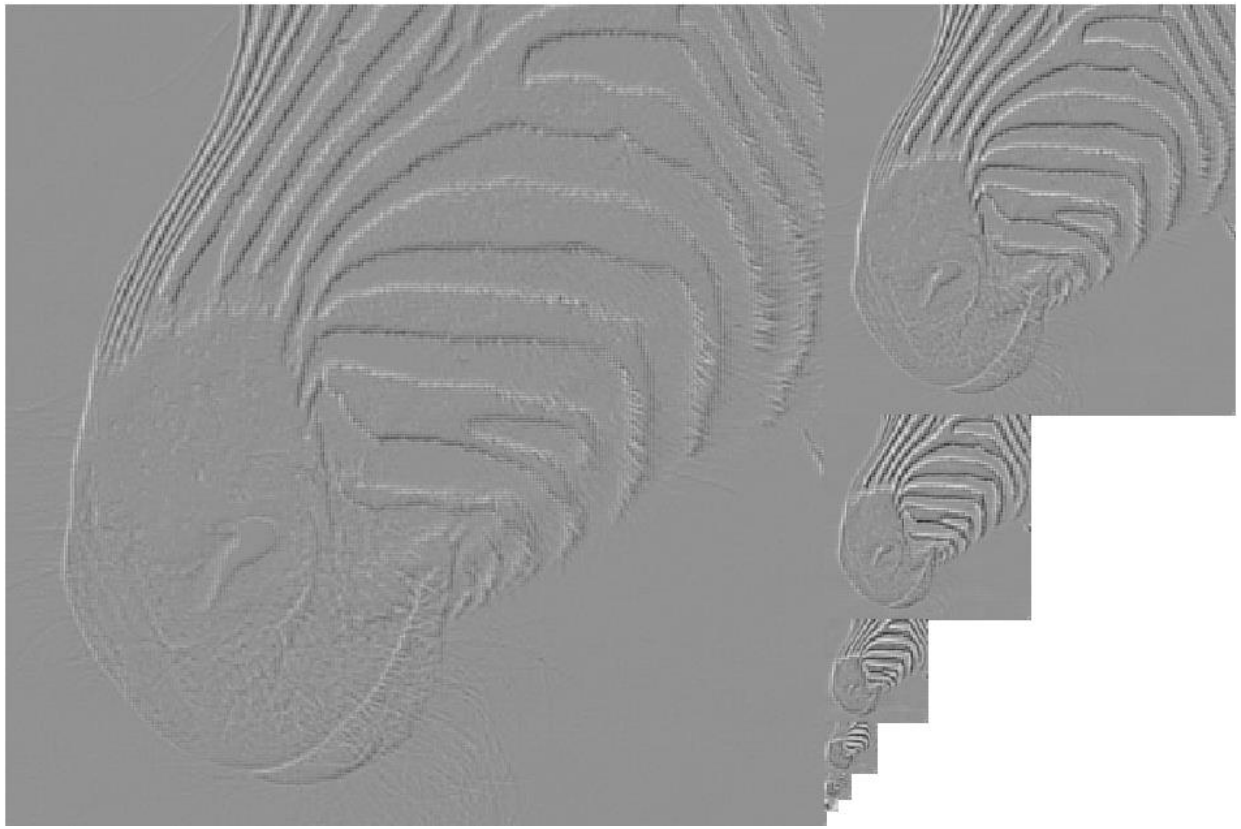
$L_0 = G_0 - \text{expand}(G_1) =$ 

Reconstructing the image from a Laplacian pyramid



Laplacian pyramid





Source: Forsyth

Subsampling images

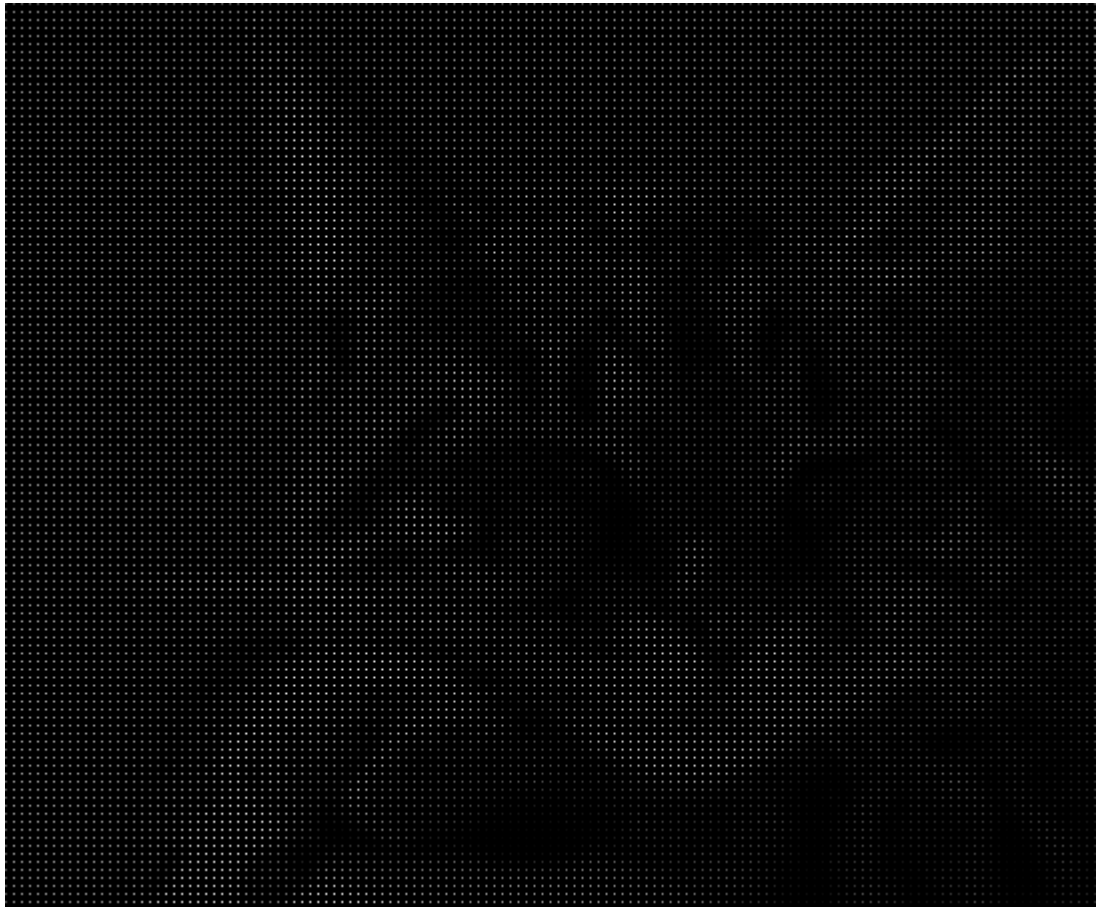
- Step 1: Convolve with Gaussian to eliminate high frequencies
- Step 2: Drop unneeded pixels

Subsampling without removing high frequencies

Subsampling after removing high frequencies

Upsampling images

Step 1: blow up to original size with 0's in between



Upsampling images

Step 2: Convolve with Gaussian

Epitome

- Subsampling causes aliasing
  - High frequencies masquerading as low frequencies
  - Remove high frequencies by blurring!
  - Common: Gaussian
- When upsampling, reconstruct missing values by convolution
  - Common: Gaussian

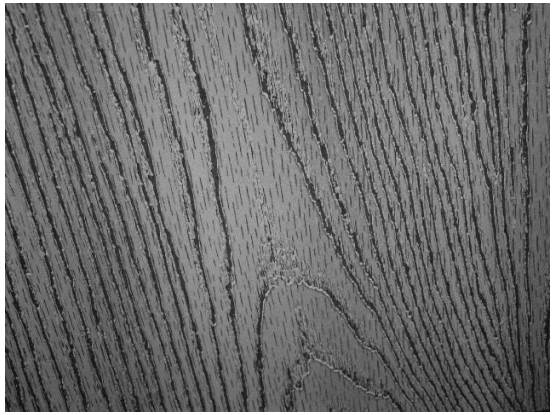
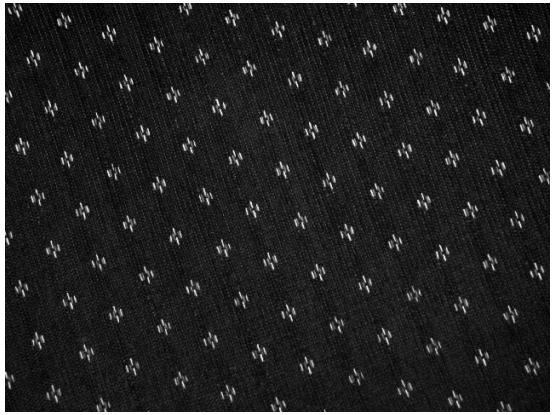
Application of filtering: Representing Texture

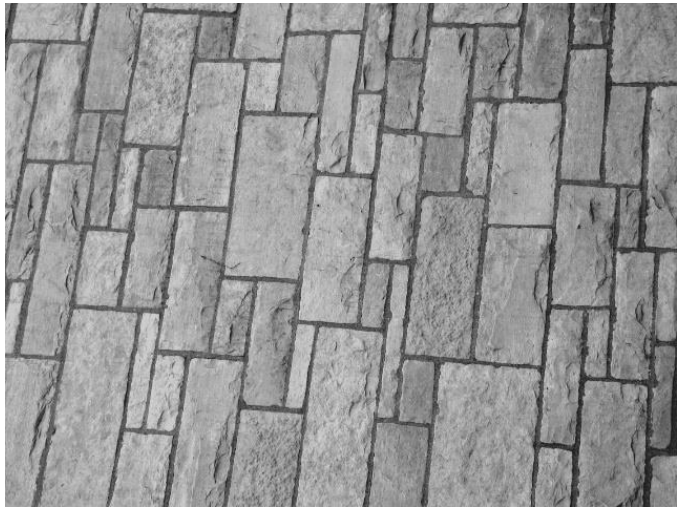
Texture representation: Compute responses of blobs and edges at various orientations and scales

Texture is regular or stochastic patterns caused by bumps, grooves, and/or markings

Source: Forsyth

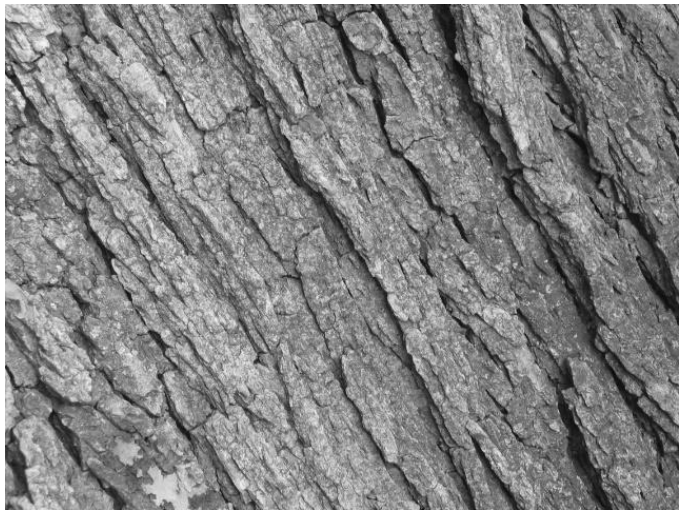
Texture and Material

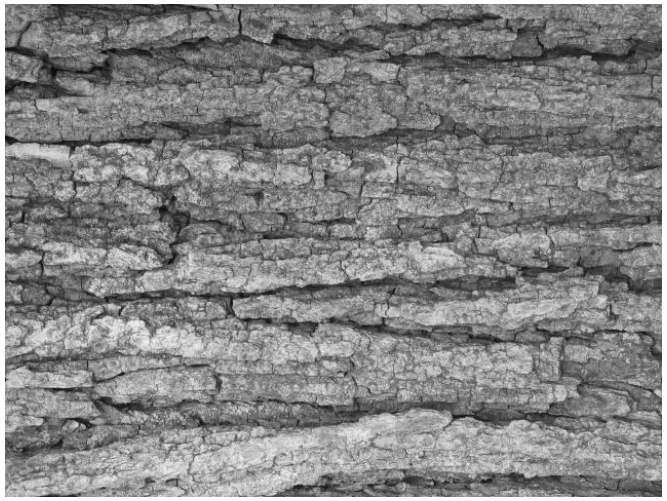





http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/

Texture and Orientation



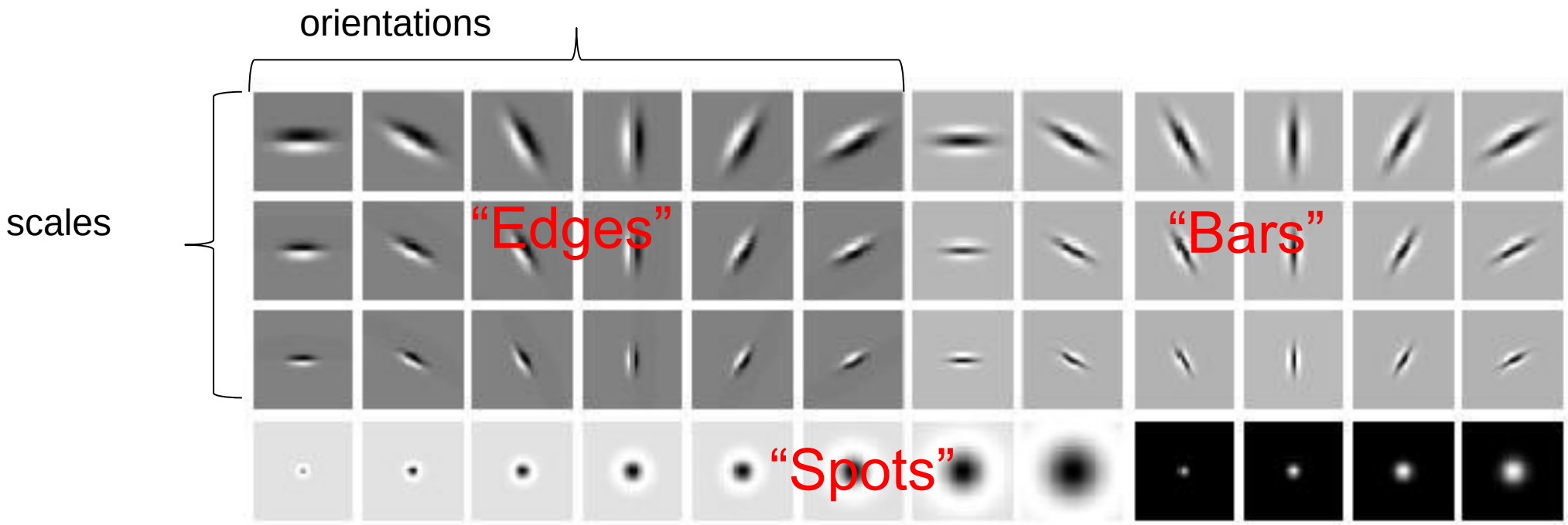


http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/

Texture and Scale

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

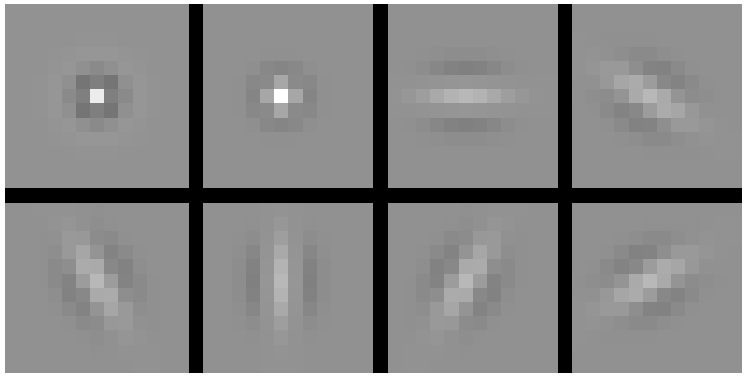
Overcomplete representation: filter banks

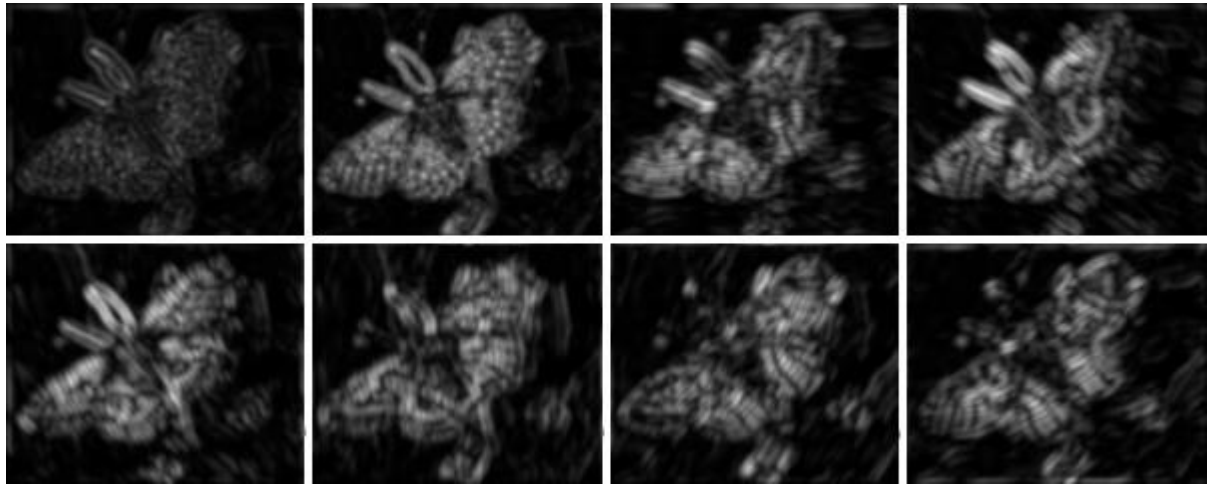


Code for filter banks: *www.robots.ox.ac.uk/~vgg/research/texclass/filters.html*

Filter banks

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





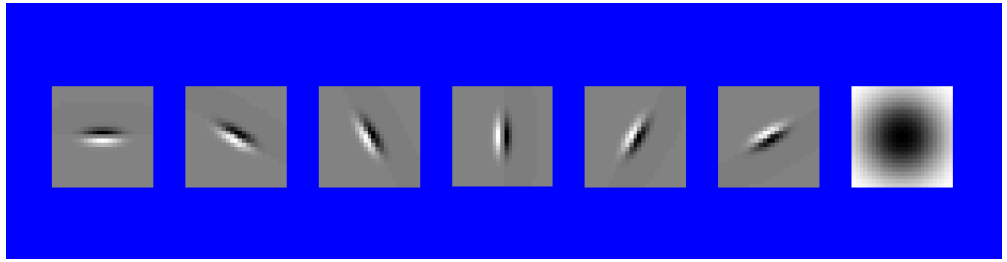
How can we represent texture?

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

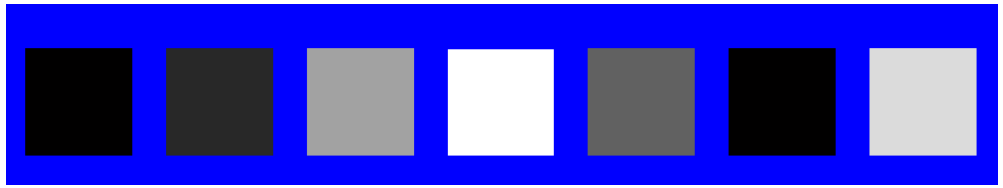
Idea 1: Record simple statistics (e.g., mean, std.) of absolute filter responses

Can you match the texture to the response?

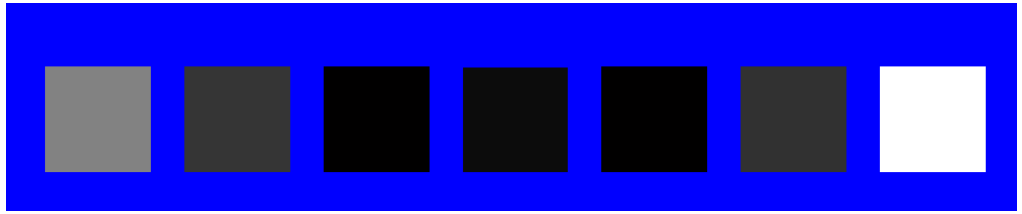
Filters



1



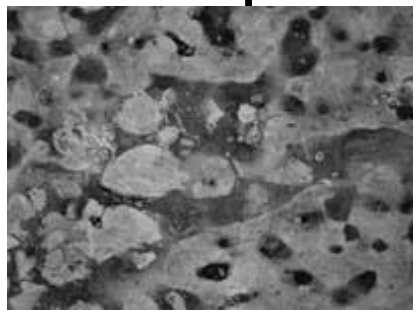
2



3

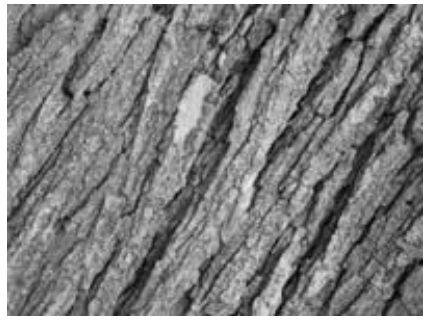
Mean abs responses

A

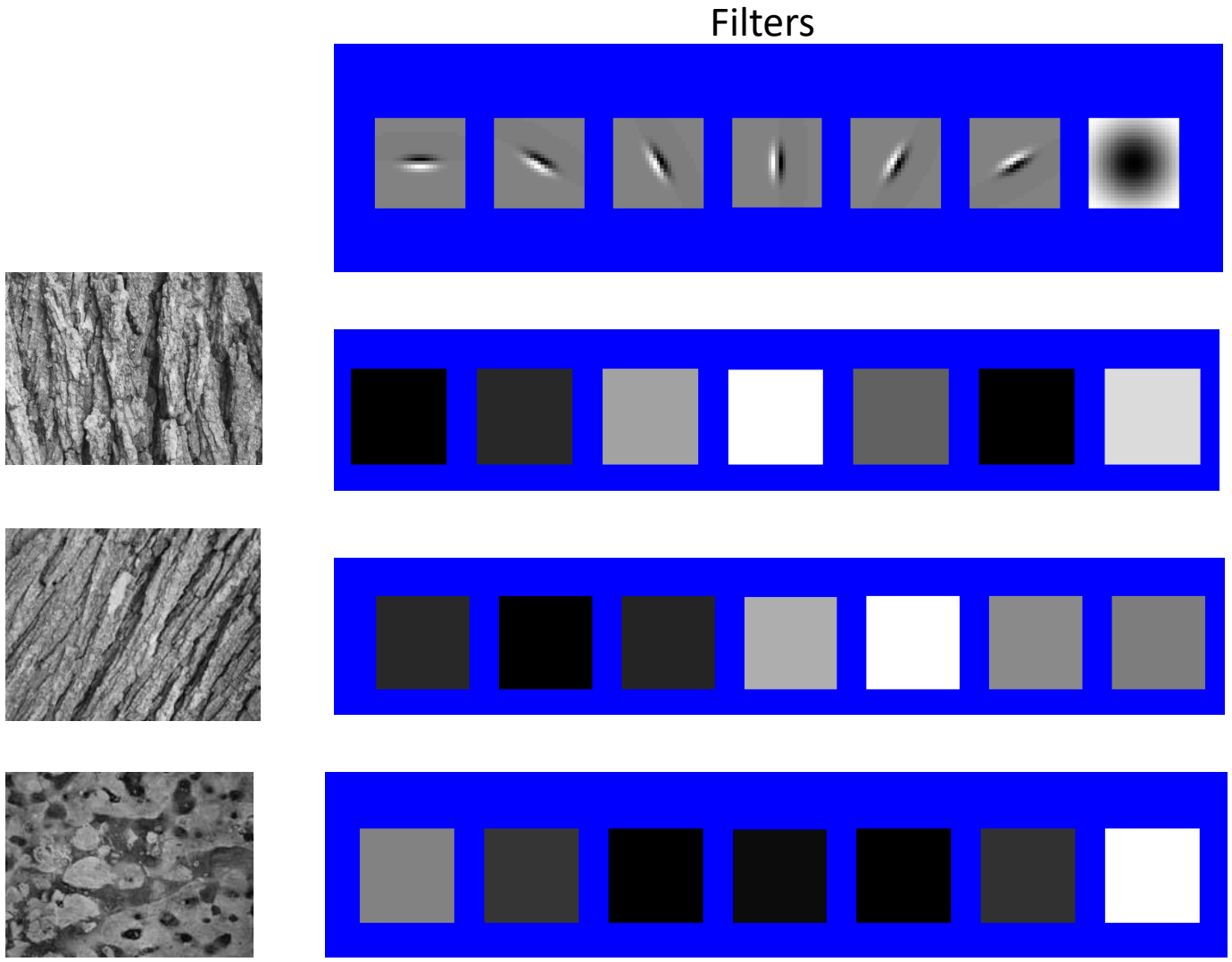


B

C

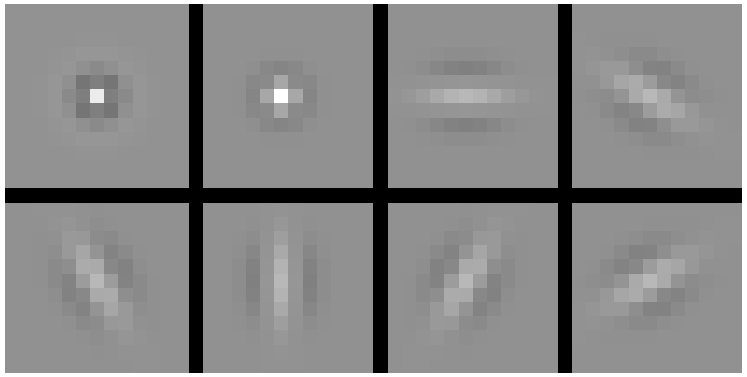


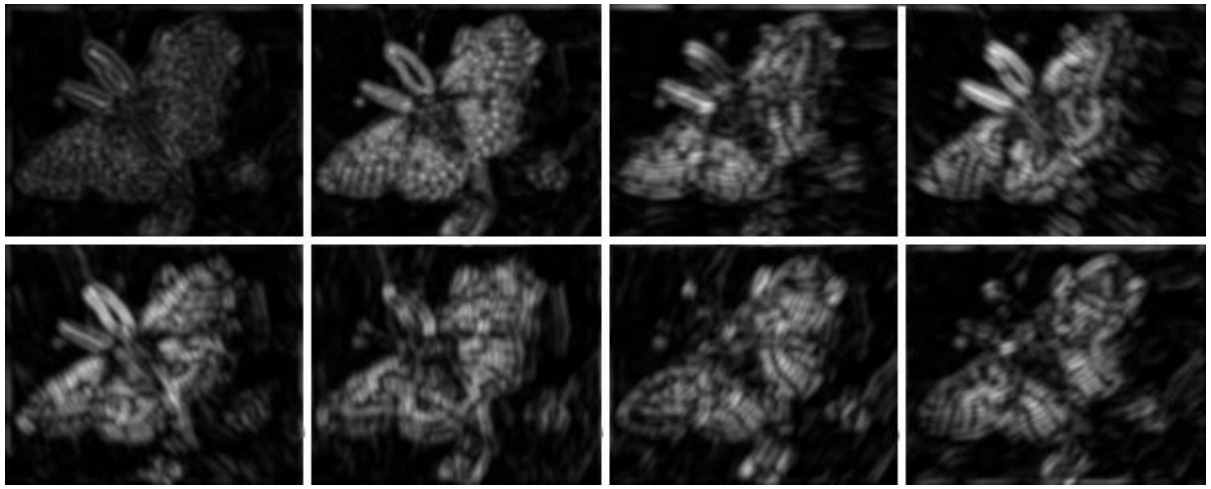
Representing texture by mean abs response



Representing texture

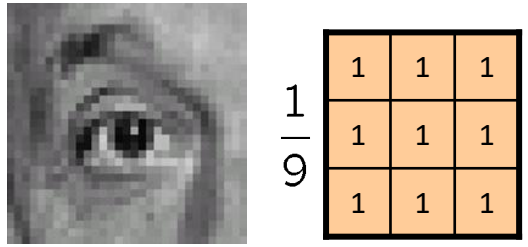
Idea 2: take vectors of filter responses at each pixel and cluster them, then take histograms (later…)

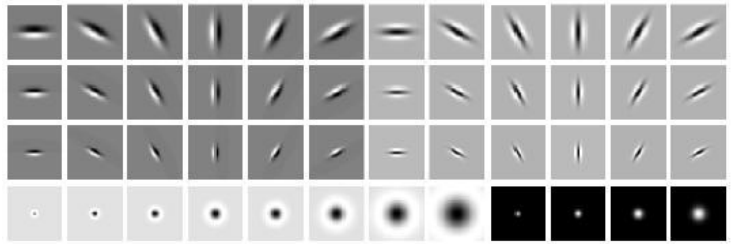




Take-away

- 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

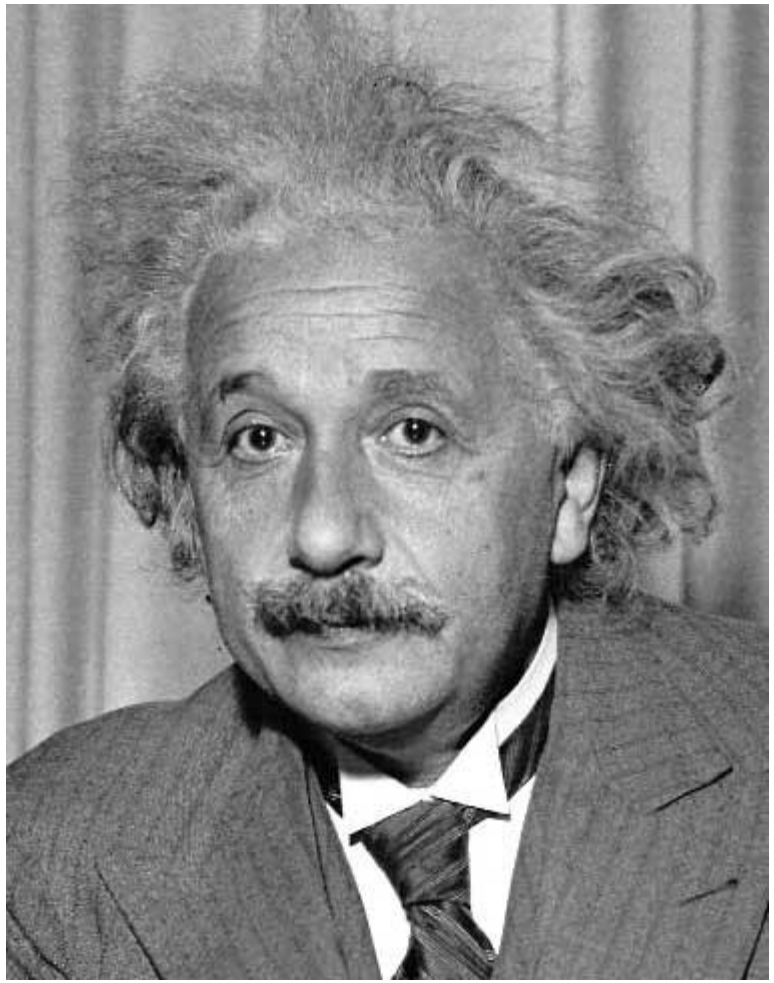




Template matching

- Goal: find  in image

- Main challenge: What is a good similarity or distance measure between two patches? D(,)
  - Correlation
  - Zero-mean correlation
  - Sum Square Difference
  - Normalized Cross Correlation

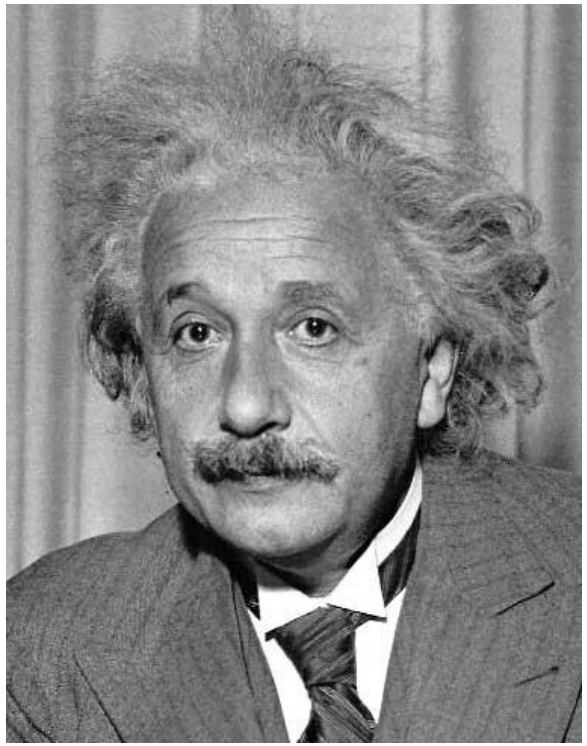


Matching with filters

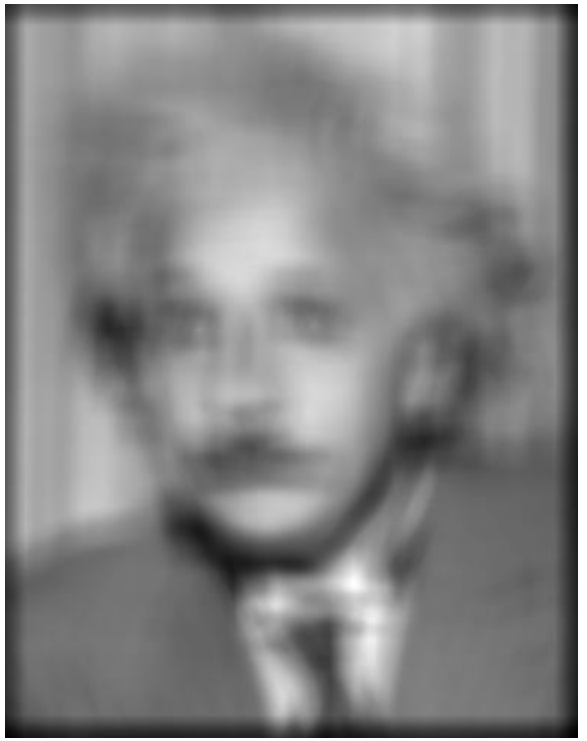
- Goal: find  in image
- Method 0: filter the image with eye patch

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

g = filter f = image



Input



Filtered Image

What went wrong?

Weaknesses of correlation

- Suppose we correlate the filter:

3	7	5

with the image:

3	2	4	1	3	8	4	0	3	8	0	7	7	7	1	2

We get the following result:

40	43	39	34	64	85	52	27	61	65	59	84	105	75	38	27

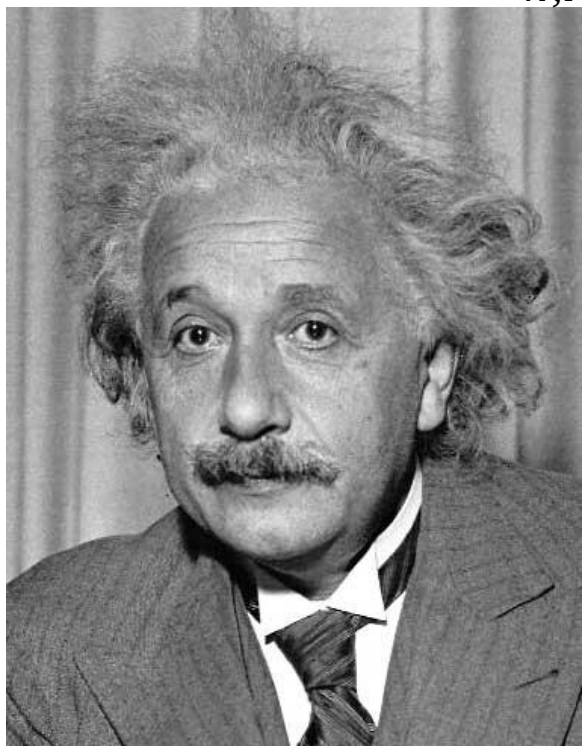
- Notice that we get a high result (85) when we center the filter on the sixth pixel, because (3,7,5) matches very well with (3,8,4). But, the highest correlation occurs at the 13th pixel, where the filter is matched to (7,7,7). Even though this is not as similar to (3,7,5), its magnitude is greater.

Matching with filters

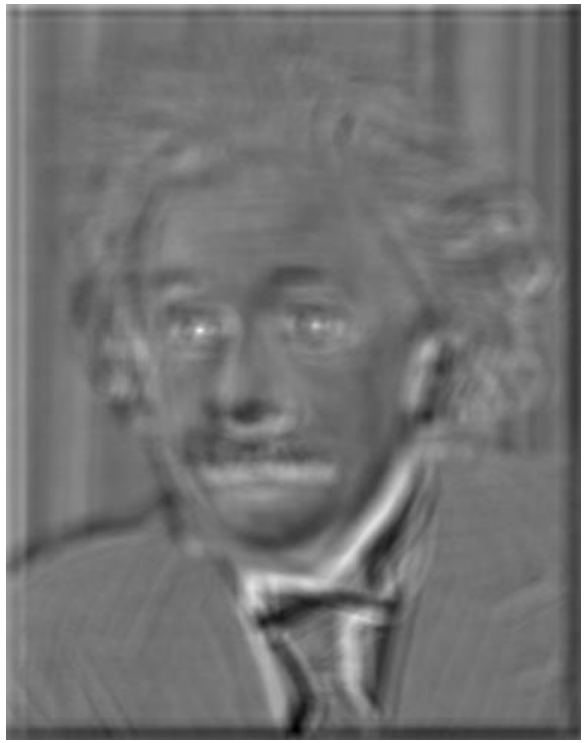
- Goal: find  in image
- Method 1: filter the image with zero-mean eye

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

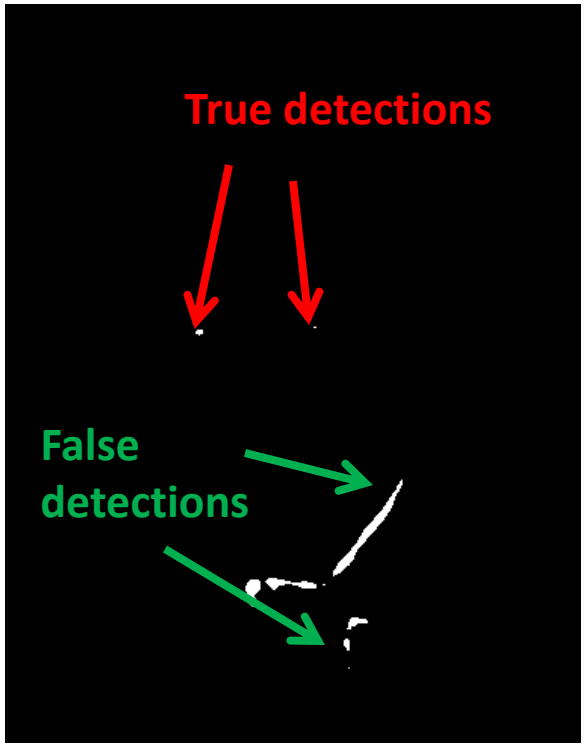
mean of template g



Input



Filtered Image (scaled)

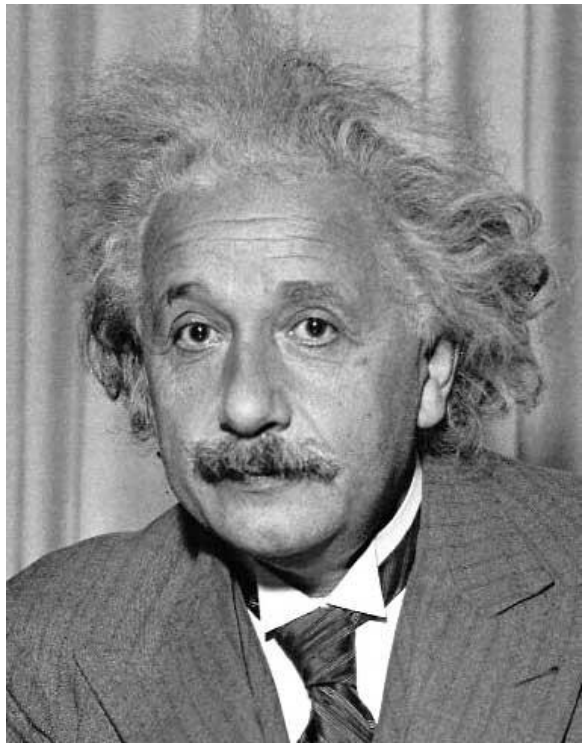


Thresholded Image

Matching with filters

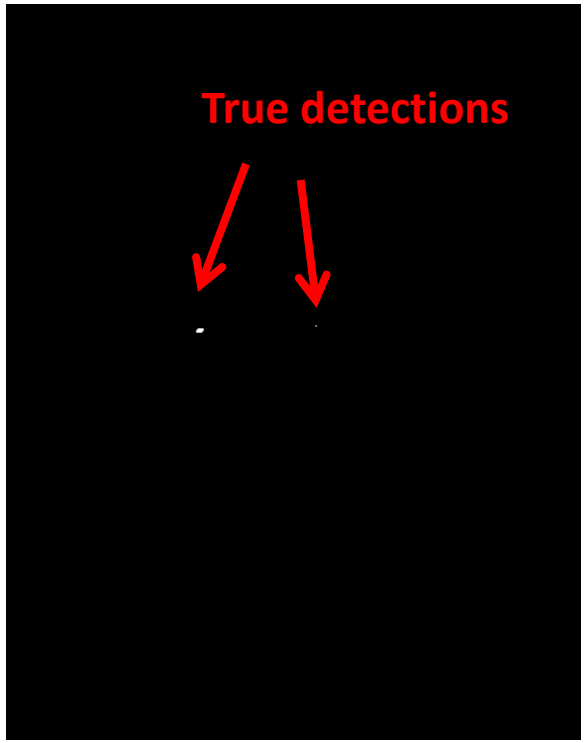
- Goal: find  in image
- Method 2: Sum of squared differences (SSD)

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k,n+l])^2$$



Input

1- sqrt(SSD)



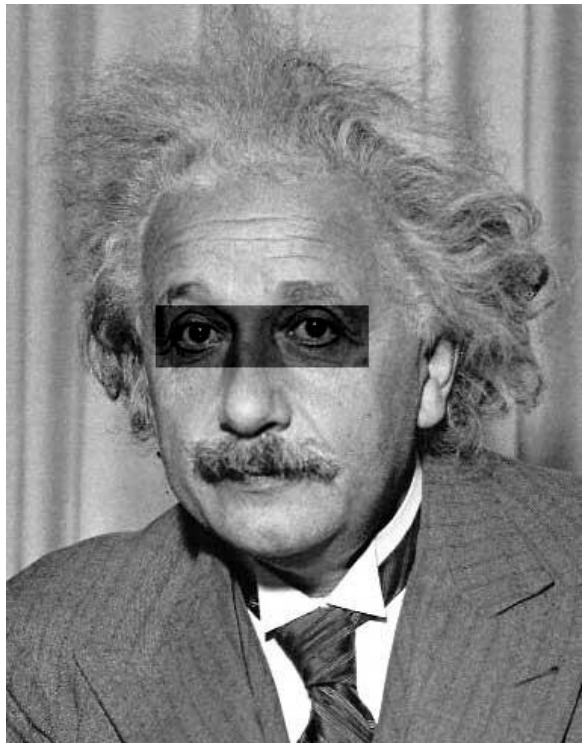
Thresholded Image

Matching with filters

- Goal: find  in image
- Method 2: SSD

What's the potential downside of SSD?

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k,n+l])^2$$

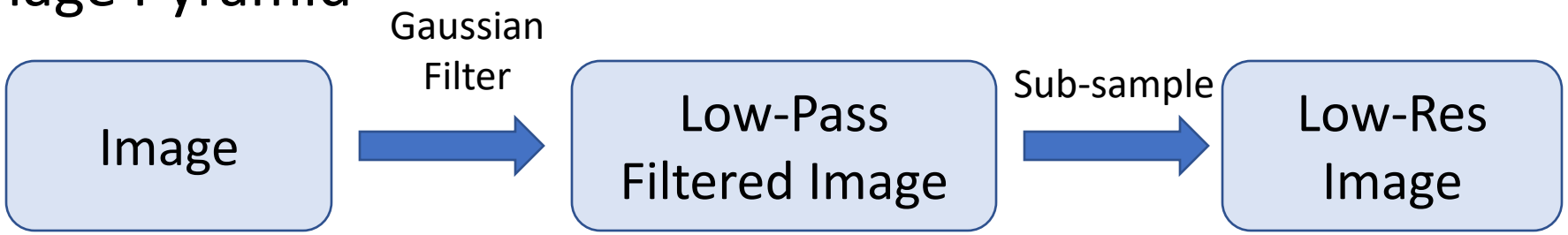


Input

1- sqrt(SSD)

Q: What if we want to find larger or smaller eyes?

A: Image Pyramid



Matching with filters

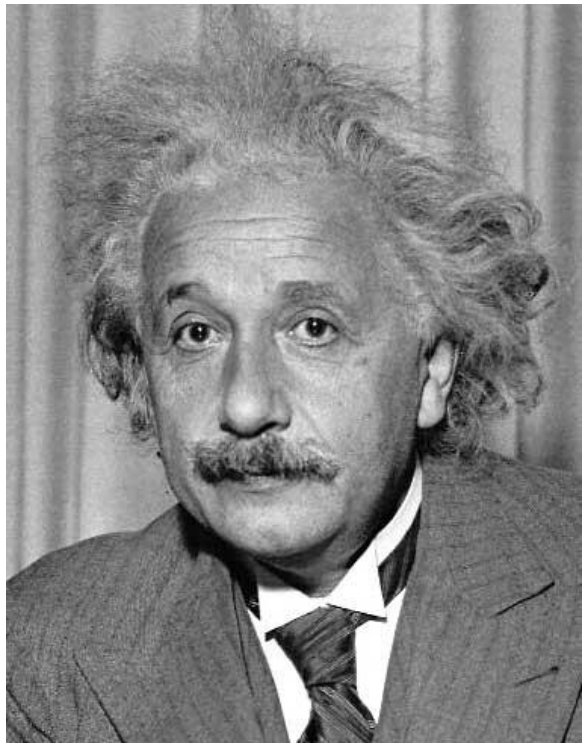
- Goal: find  in image
- Method 3: Normalized cross-correlation

mean template mean image patch

$$h[m,n] = \frac{\sum_{k,l}(g[k,l] - \bar{g})(f[m+k,n+l] - \bar{f}_{m,n})}{\left(\sum_{k,l}(g[k,l] - \bar{g})^2 \sum_{k,l}(f[m+k,n+l] - \bar{f}_{m,n})^2\right)^{0.5}}$$

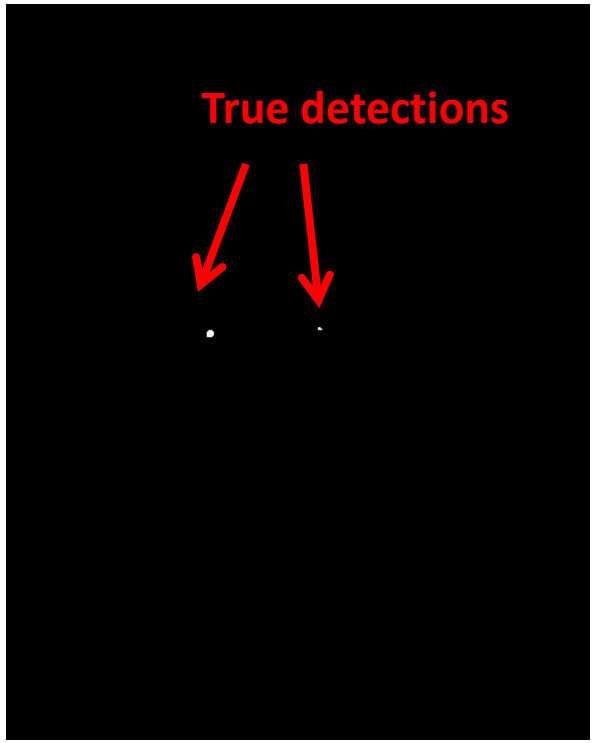
Matching with filters

- Goal: find  in image
- Method 3: Normalized cross-correlation



Input

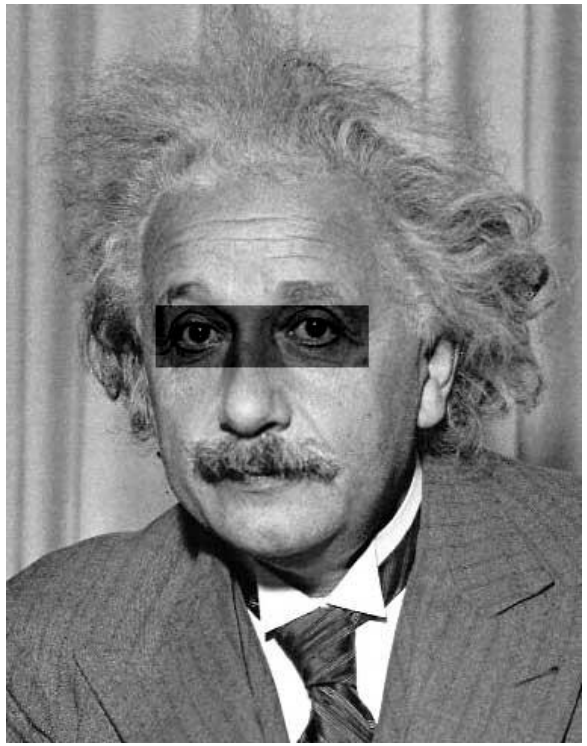
Normalized X-Correlation



Thresholded Image

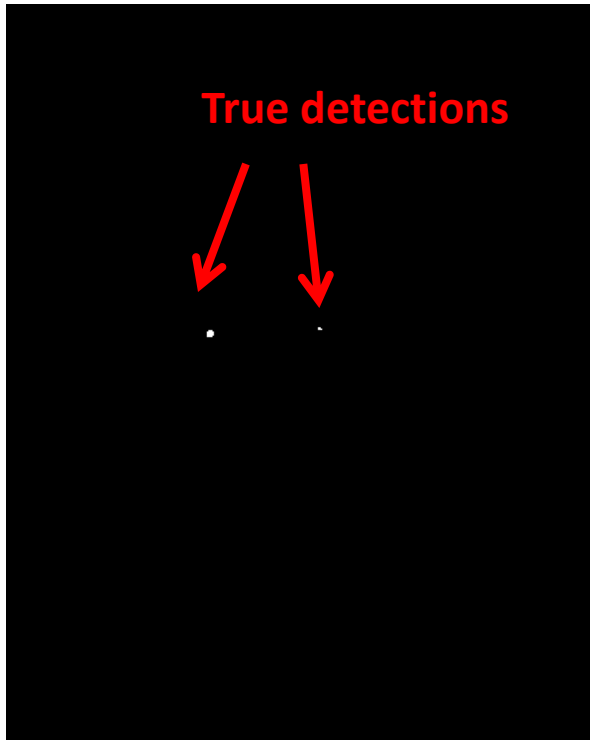
Matching with filters

- Goal: find  in image
- Method 3: Normalized cross-correlation



Input

Normalized X-Correlation



Thresholded Image

Q: What is the best method to use?

A: Depends

- Zero-mean filter: fastest but not a great matcher
- SSD: next fastest, sensitive to overall intensity
- Normalized cross-correlation: slowest, invariant to local average intensity and contrast