CS4670/5670: Intro to Computer Vision

Lecture 2: Images and image filtering

Last time

- Natural images are *not* arbitrary 2D arrays
- They have properties resulting from physics / math of image formation
- Solving computer vision requires using these properties

Last time: Some primitives

- Edge detection: identifying where pixels change color
  - Cue to object boundary
  - Cue to shape
  - More resilient to lighting than pixel color
- Zooming into or out of images
  - Searching for both nearby and far-off objects
- Matching patches from two different images
  - First step in identifying 3D location

Last time: Related problems

- Image Restoration
  - denoising
  - deblurring
- Image Compression
  - JPEG, JPEG2000, MPEG..

- Again, use the same ``priors”

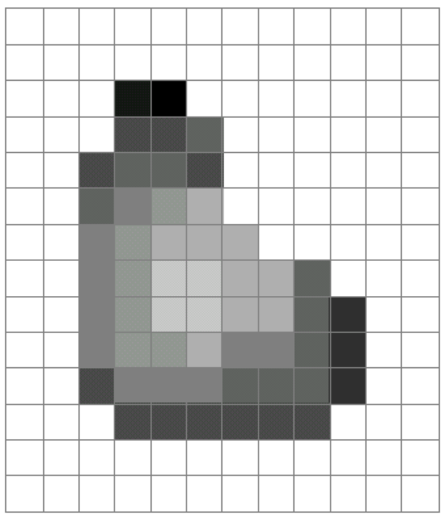
Image denoising

Why would images have noise?

- Sensor noise
  - Sensors count photons: noise in count
- Dead pixels
- Old photographs
- …

What is an image?

- A grid (matrix) of intensity values: 1 color or 3 colors



=

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

(common to use one byte per value: 0 = black, 255 = white)

An assumption about noise

- 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
    - "grainy images"

Noise reduction

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

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	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	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

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

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	0	0	0	0	0
0	1	4	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

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

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	0	0	0	0	0
0	1	4	8	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

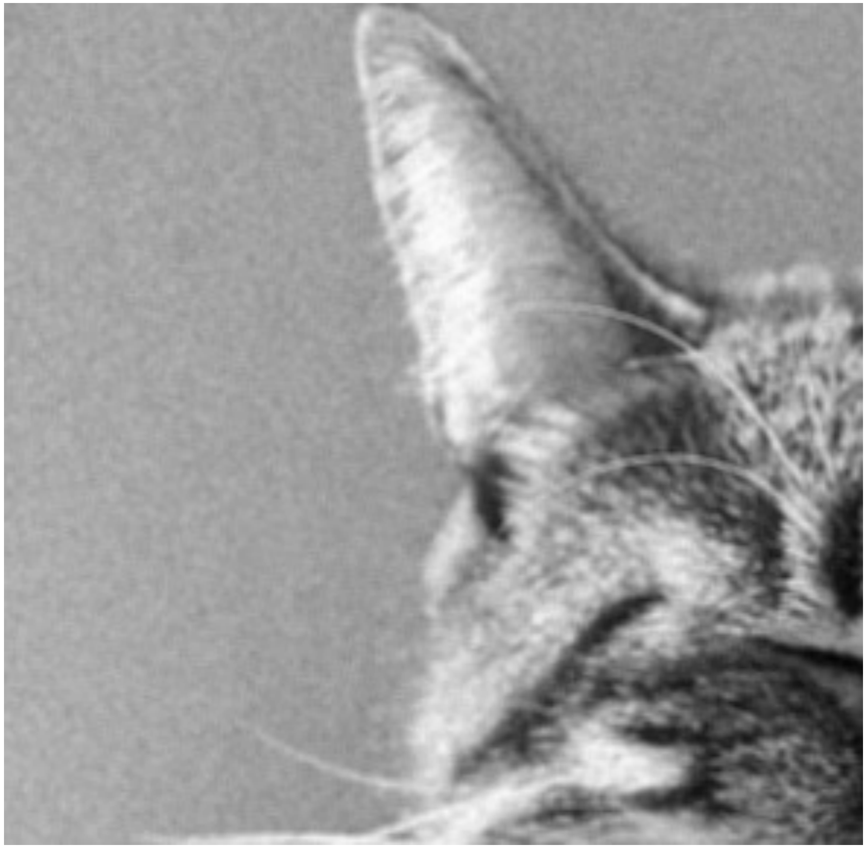
$$(0 + 0 + 0 + 0 + 10 + 10 + 10 + 20 + 20)/9 = 7.77$$

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	0	0	0	0	0
0	1	4	8	10	8	9	6	4	0
0	4	11	13	16	11	12	7	4	0
0	6	14	19	23	19	18	10	6	0
0	8	18	23	28	23	17	8	2	0
0	8	16	26	31	30	20	10	3	0
0	10	18	27	29	27	17	8	2	0
0	8	14	22	22	20	11	8	3	0
0	4	11	17	17	12	6	4	2	0
0	0	0	0	0	0	0	0	0	0

Noise reduction using mean filtering



Mean filtering

- Replace pixel by mean of neighborhood

10	5	3
4	5	1
1	1	7

Local image data

f

	7	

Modified image data

$S[f]$

$$S[f](m,n) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} f(m+i, n+j)/9$$

A more general version

10	5	3
4	5	1
1	1	7

Local image data

	7	

Kernel / filter

$$S[f](m,n) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w(i,j) f(m+i, n+j)$$

A more general version

0	10	5	7	0
5	11	6	8	3
9	22	4	5	1
2	9	14	6	7
3	10	15	12	9

Local image data

		7		

Kernel size = 2k+1

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

Convolution and cross-correlation

- Cross correlation

$$S[f] = w \otimes f$$

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

- Convolution

$$S[f] = w * f$$

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(\color{red}{m-i}, \color{red}{n-j})$$

Cross-correlation

1	2	3
4	5	6
7	8	9

1	2	3
4	5	6
7	8	9

$$1*1 + 2*2 + 3*3 + 4*4 + 5*5 + 6*6 + 7*7 + 8*8 + 9*9$$

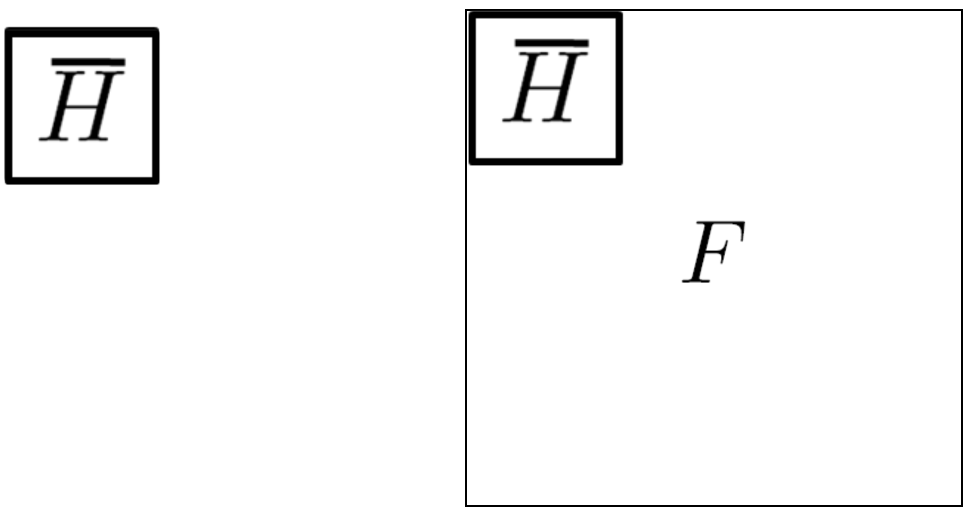
Convolution

1	2	3
4	5	6
7	8	9

1	2	3
4	5	6
7	8	9

$$1*9 + 2*8 + 3*7 + 4*6 + 5*5 + 6*4 + 7*3 + 8*2 + 9*1$$

Convolution



Adapted from F. Durand

Properties: Linearity

$$(w \otimes f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

$$f'(m,n) = af(m,n)$$

$$(w \otimes f')(m,n) = a(w \otimes f)(m,n)$$

Properties: Linearity

$$(w \otimes f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

$$f' = af$$

$$(w \otimes f') = a(w \otimes f)$$

Properties: Linearity

$$(w \otimes f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

$$f' = af + bg$$

$$w \otimes f' = a(w \otimes f) + b(w \otimes g)$$

Properties: Linearity

$$(w \otimes f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

$$w' = aw + bv$$

$$w' \otimes f = a(w \otimes f) + b(v \otimes f)$$

Properties: Shift invariance

$$(w \otimes f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

$$f'(m,n) = f(m-m_0, n-n_0)$$

f

f'

Shift invariance

$$(w \otimes f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

$$f'(m,n) = f(m-m_0, n-n_0)$$

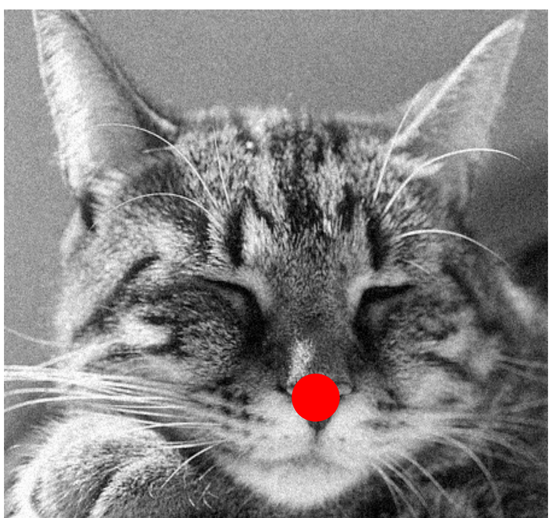
$$\begin{aligned}
(w \otimes f')(m,n) &= \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f'(m+i, n+j) \\
&= \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i-m_0, n+j-n_0) \\
&= (w \otimes f)(m-m_0, n-n_0)
\end{aligned}$$

Shift invariance

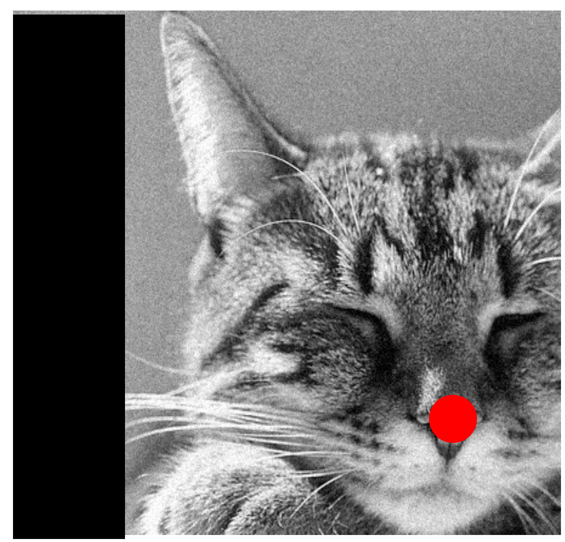
$$f'(m,n) = f(m-m_0, n-n_0)$$

$$(w \otimes f')(m,n) = (w \otimes f)(m-m_0, n-n_0)$$

- Shift, then convolve = convolve, then shift
- Output of convolution does not depend on where the pixel is

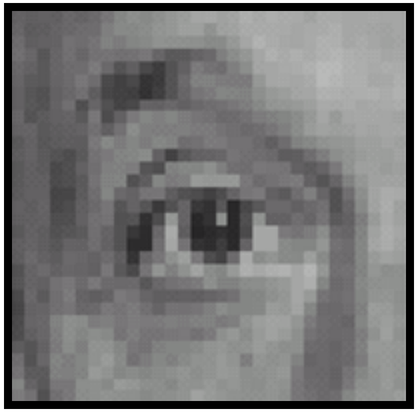


f

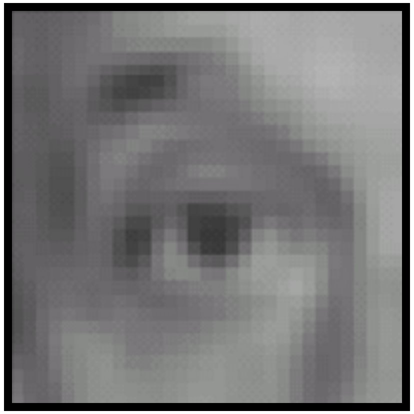


f'

Filters: examples



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



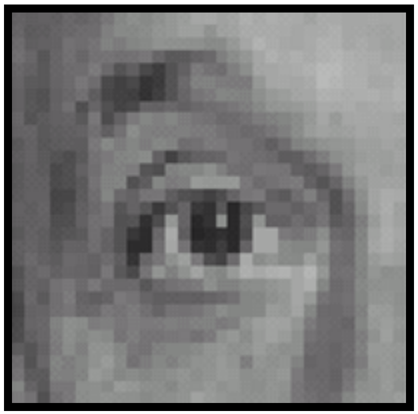
Original (f)

Kernel (k)

Blur (with a mean filter) (g)

Source: D. Lowe

Filters: examples



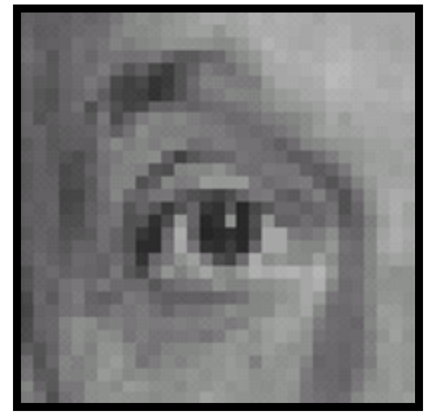
Original (f)

$$*$$

0	0	0
0	1	0
0	0	0

Kernel (k)

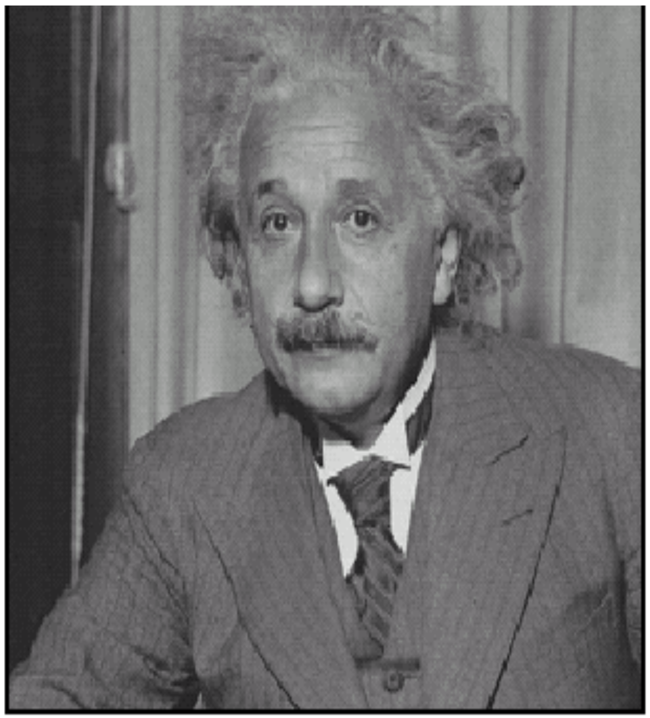
$$=$$



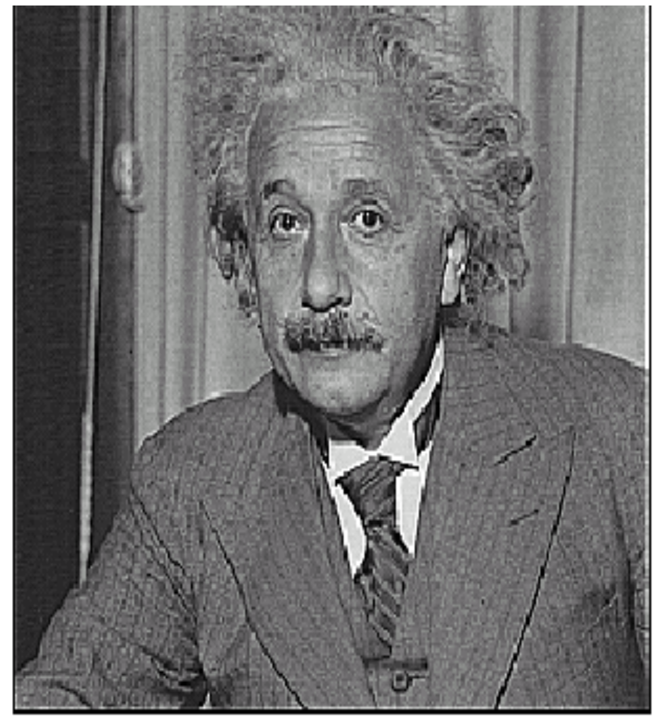
Identical image (g)

Source: D. Lowe

Sharpening



before

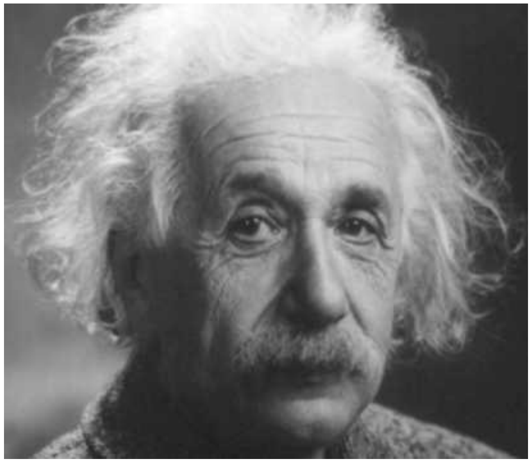


after

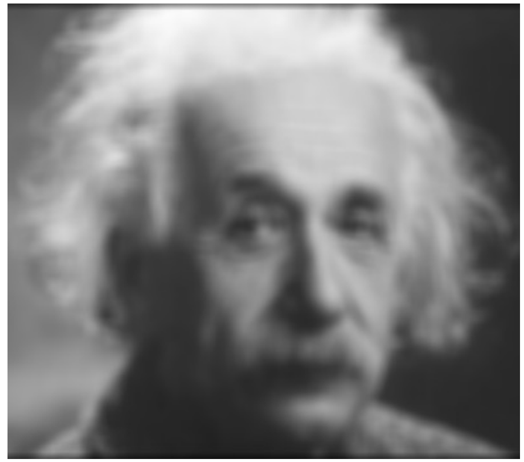
Source: D. Lowe

Sharpening

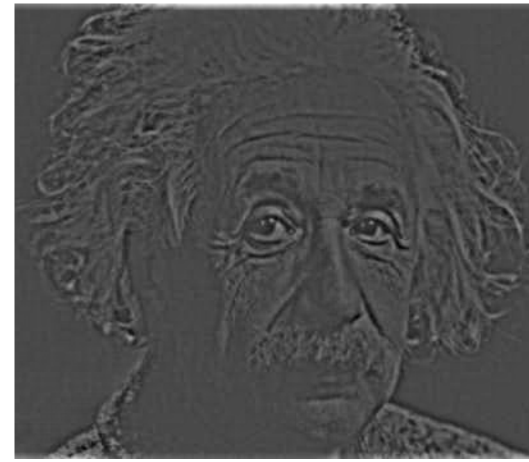
- What does blurring take away?



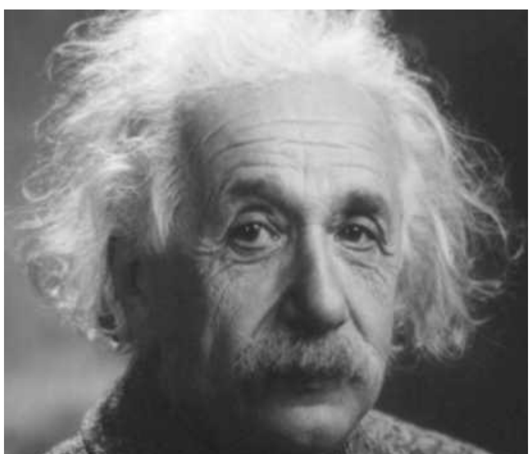
–



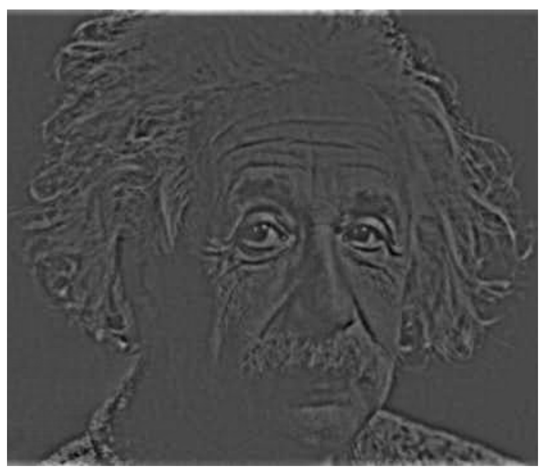
=



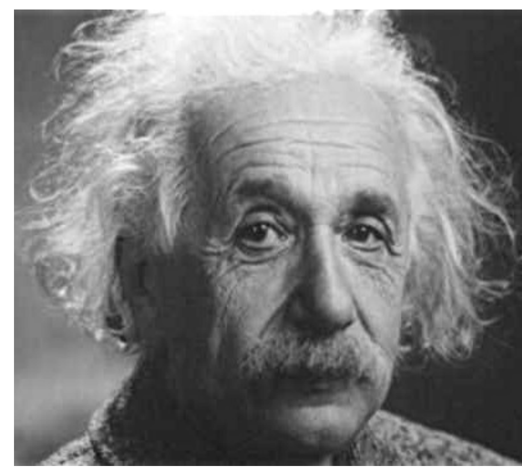
Let's add it back:



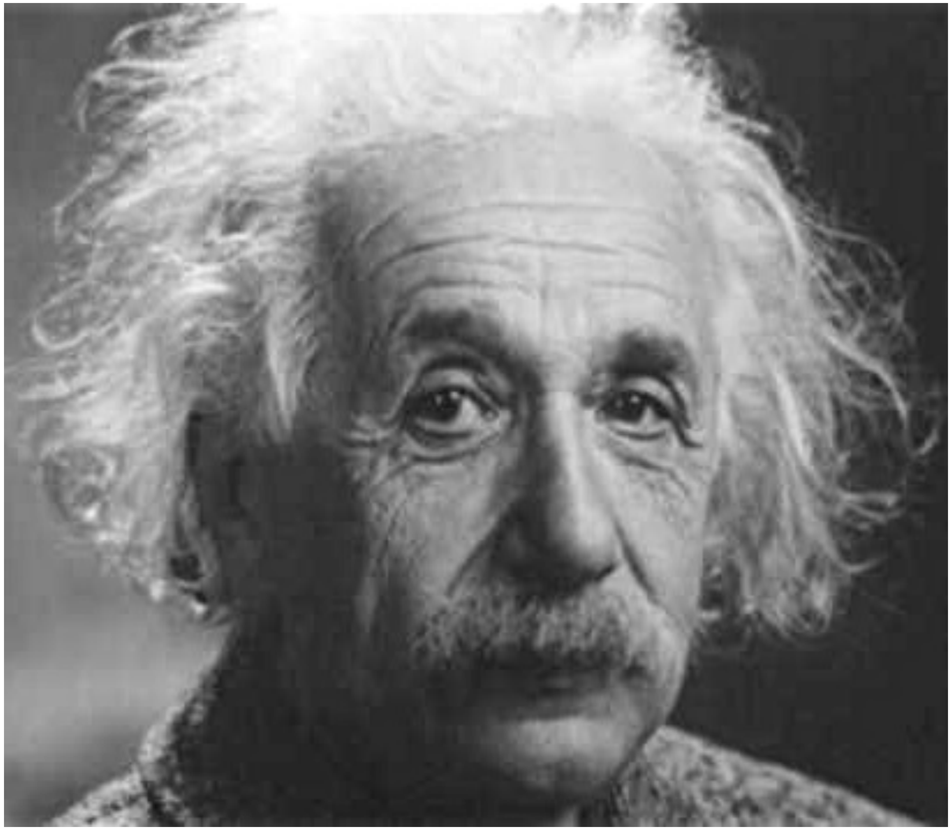
+ α



=



Sharpening



Sharpening

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

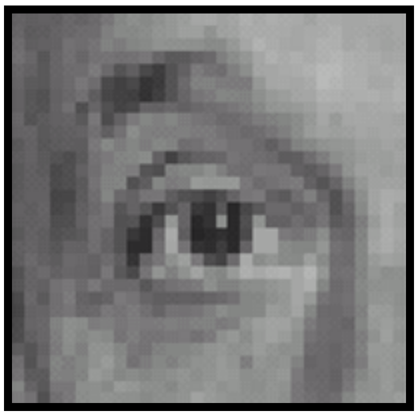
$$= (1 + \alpha)f - \alpha f_{blur}$$

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

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

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

Sharpening filter



Original

$$* \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) =$$

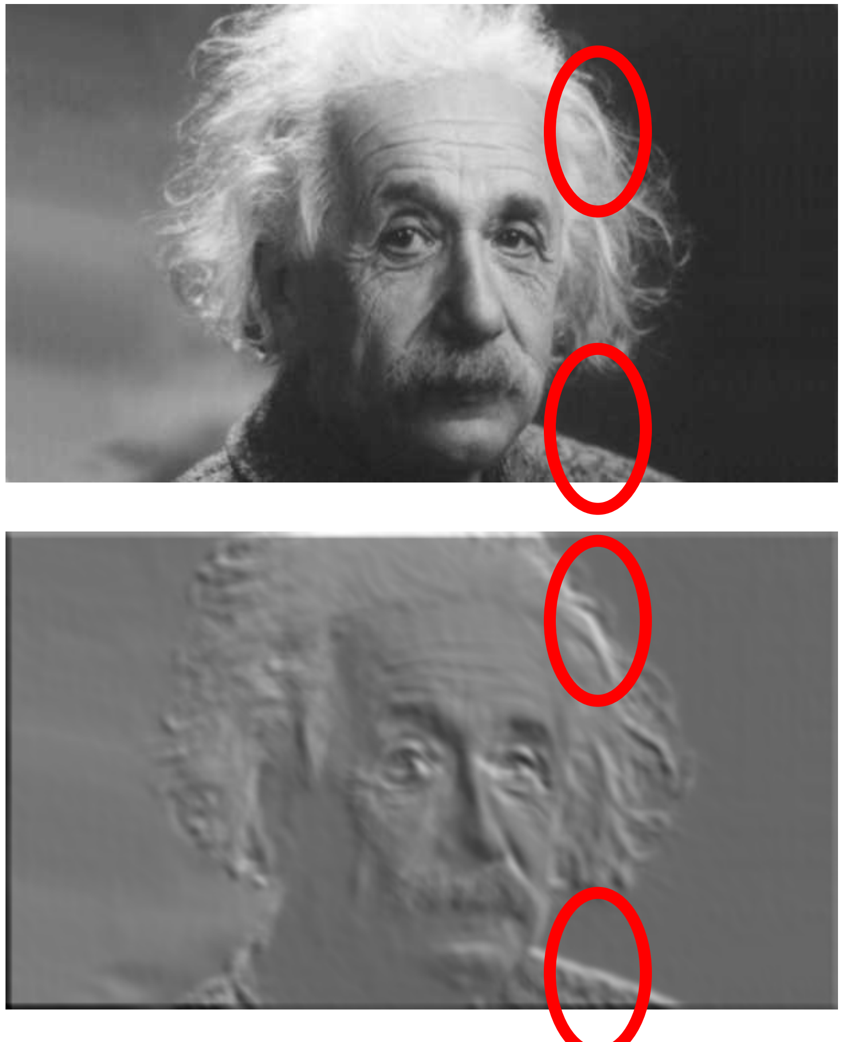
Sharpening filter
(accentuates edges)

Source: D. Lowe

Another example

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	1	-1	-1	-1	-1	-1
1	1	1	1	1	1	1	-1	-1	-1	-1
1	1	1	1	1	1	1	1	-1	-1	-1
1	1	1	1	1	1	1	1	1	-1	-1
1	1	1	1	1	1	1	1	1	1	-1

Another example



Cross-correlation and dot products

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

f_1	f_2	f_3
f_4	f_5	f_6
f_7	f_8	f_9

$$\sum_i w_i f_i = \vec{w} \cdot \vec{f}$$

$\vec{w} =$

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9

$\vec{f} =$

f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9

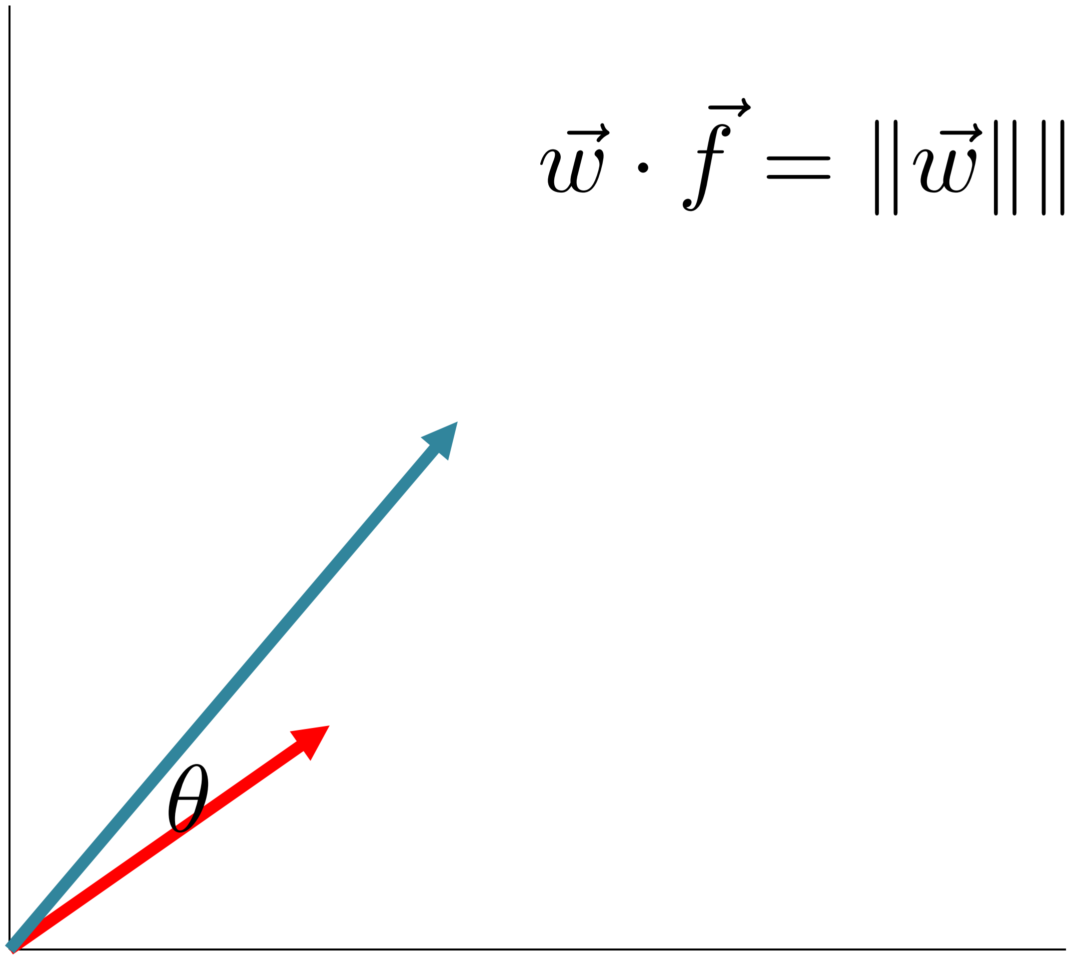
Cross-correlation and dot products

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	0	0	0	0	0
0	1	4	8	10	8	9	6	4	0
0	4	11	13	16	11	12	7	4	0
0	6	14	19	23	19	18	10	6	0
0	8	18	23	28	23	17	8	2	0
0	8	16	26	31	30	20	10	3	0
0	10	18	27	29	27	17	8	2	0
0	8	14	22	22	20	11	8	3	0
0	4	11	17	17	12	6	4	2	0
0	0	0	0	0	0	0	0	0	0

Dot products

$$\vec{w} \cdot \vec{f} = \|\vec{w}\| \|\vec{f}\| \cos\theta$$



Dot products

$$\vec{w} \cdot \vec{f} = \|\vec{w}\| \|\vec{f}\| \cos\theta$$

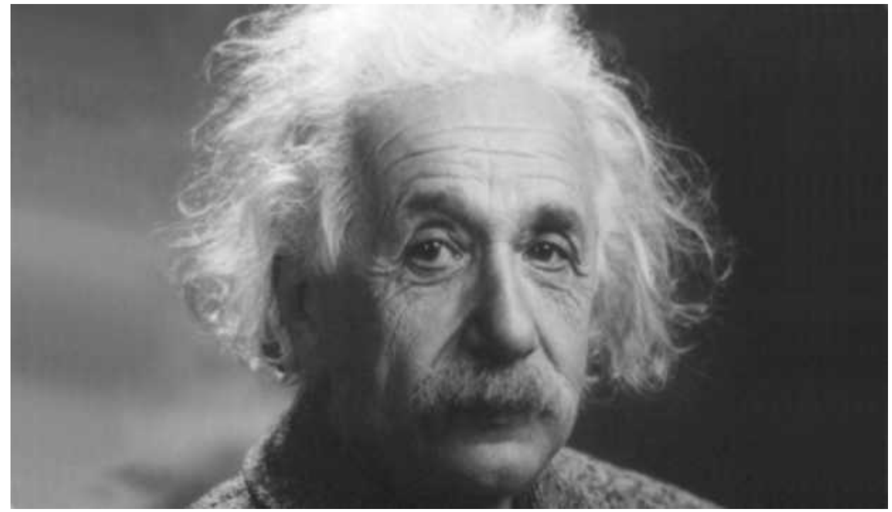
- cos θ indicates similarity
- Can measure how much f “matches” w
  - Central component of “template matching”
  - But might need to divide by magnitude
  - Cosine distance
- Cross-correlation $\approx$ template matching

Dot products

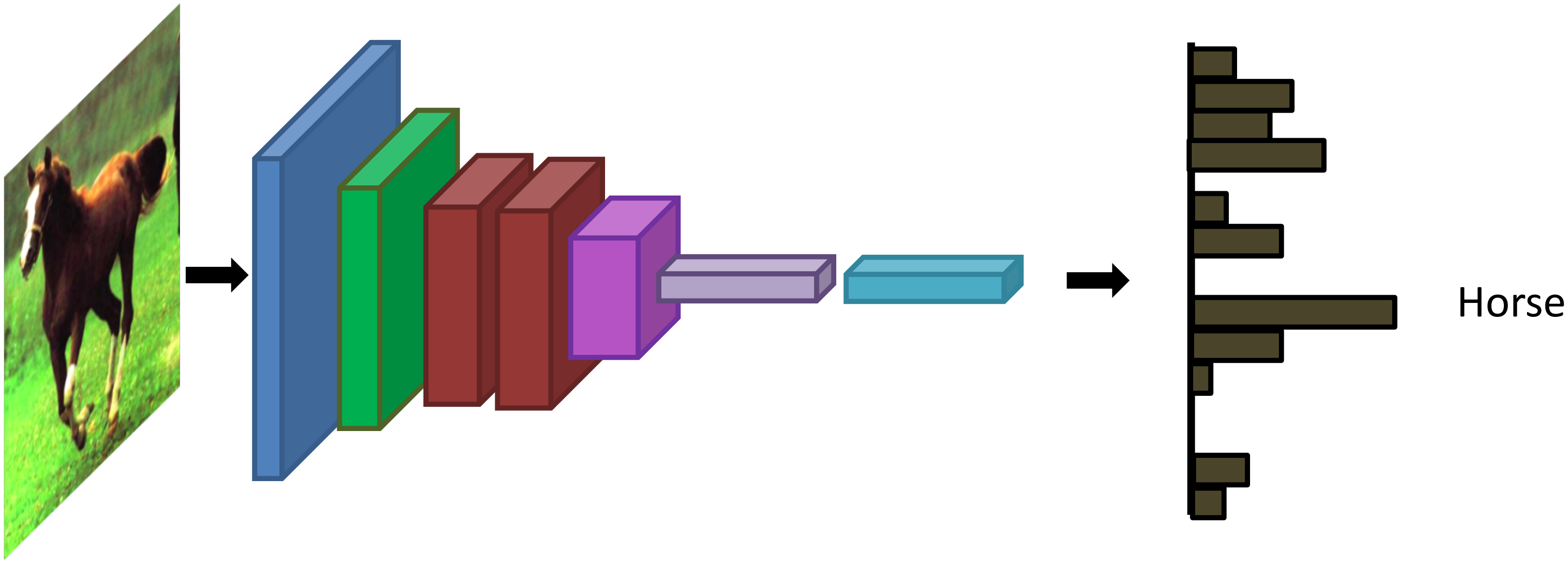
$$\vec{w} \cdot \vec{f} = \|\vec{w}\| \|\vec{f}\| \cos\theta$$

- Usefulness *really* depends on space
- E.g., pixel intensities are often a *bad* space.
- Why?

Cross correlation as template matching

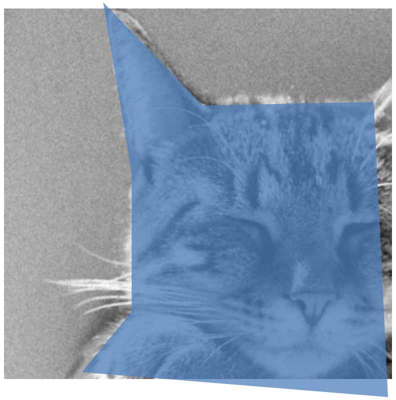
( - b) $\otimes$ 

Convolution is everywhere



Why is convolution important?

- Shift invariance is a crucial property



Why is convolution important?

- We *like* linearity
  - Linear functions behave predictably when input changes
  - Lots of theory just easier with linear functions
- *All linear shift-invariant systems can be expressed as a convolution*

Non-linear filters: Thresholding

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

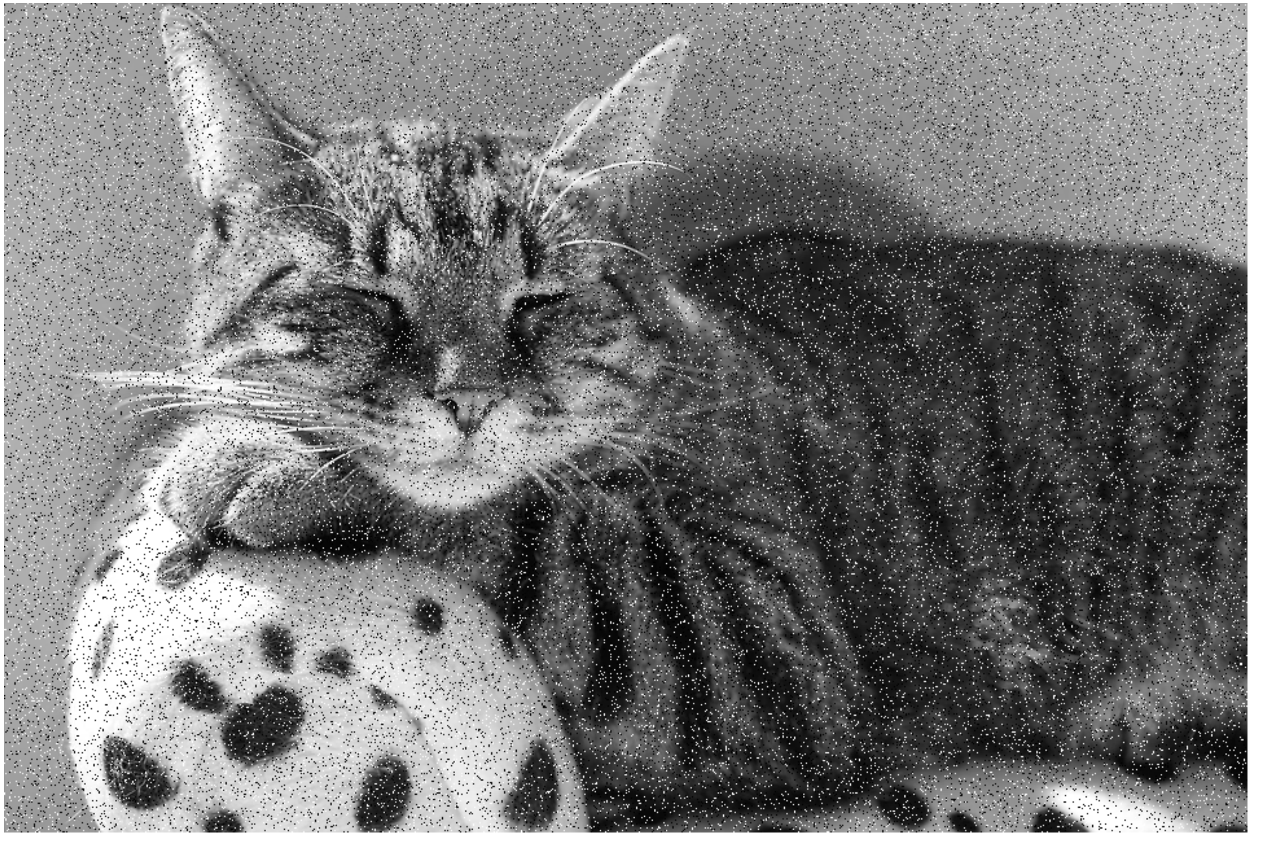
Non-linear filters: Rectification

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

- Crucial component of modern convolutional networks

Non-linear filters

- Sometimes mean filtering does not work



Non-linear filters

- Sometimes mean filtering does not work

Non-linear filters

- Mean is sensitive to outliers
- Median filter: Replace pixel by *median* of neighbors

Non-linear filters

Takeaway

- Two general recipes:
  - convolution
  - cross-correlation
- Properties
  - Shift-invariant: a sensible thing to require
  - Linearity: convenient
- Can be used for smoothing, sharpening
- Also main component of CNNs

Next up

- Back to linear filters
- Signal processing view of filtering
- Filtering for detecting edges etc

Images as functions

- An image contains discrete numbers of pixels
- Pixel value
  - grayscale/intensity
    - [0,255]
  - Color
    - RGB [R, G, B], where [0,255] per channel

Images as functions

- Can think of image as a **function**, f, from R^2 to R or R^M:
  - Grayscale: $f(x,y)$ gives **intensity** at position (x,y)
    - f: [a,b] x [c,d] ➔[0,255]
  - Color: $f(x,y) = [r(x,y), g(x,y), b(x,y)]$
- Most adjacent pixels are correlated => function is *continuous*

What is an image?

A **digital** image is a discrete (**sampled**, **quantized**) version of this function