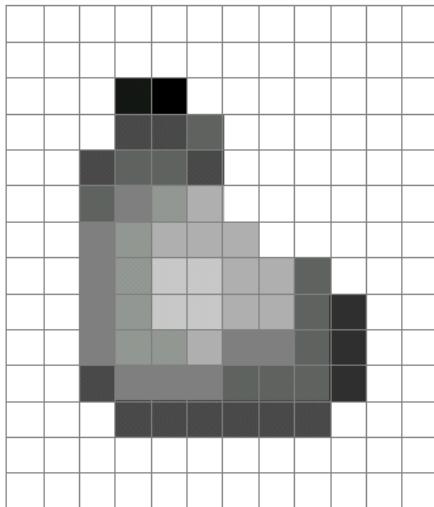


Lecture 2: Image filtering

What is an image?

- A grid (matrix) of intensity values



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

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

Images as functions

- An image contains discrete numbers of pixels

- Pixel value

- grayscale/intensity

- [0,255] or [0,1] or any real

- Color

- **RGB [R, G, B]** most commonly used for computers

- Lab [L, a, b]: Lightness, a and b are color-opponent dimensions

- HSV [H, S, V]: Hue, saturation, value



Images as functions

- Can think of image as a **function**, f , from \mathbb{R}^2 to \mathbb{R} or \mathbb{R}^M :
 - Grayscale: $f(x,y)$ gives **intensity** at position (x,y)
 - $f: [a,b] \times [c,d] \rightarrow \mathbb{R}$ Take pixel location (x,y) and output intensity
 - Color: $f(x,y) = [r(x,y), g(x,y), b(x,y)]$
Take pixel location (x,y) and output r,g, and b values at that location

What is an image?

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

Sampling- part that deals with digitizing of coordinates

Quantization- part that deals with digitizing amplitude

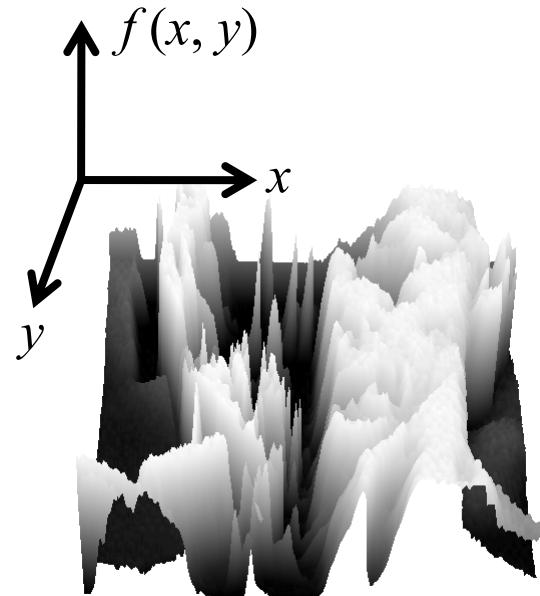


Image transformations

Appearance
transformations



$$g(x,y) = f(x,y) + 20$$

Changes the intensity of each pixel

Examples: making pic lighter, making pic darker, adding a layer of color

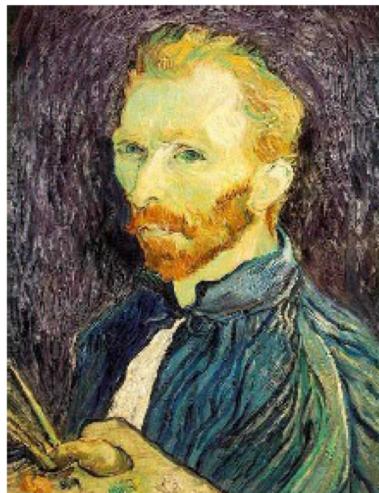
Geometric
transformations



$$g(x,y) = f(-x,y)$$

Changes the location of pixel

Example applications



Super-resolution: Increase image size

Noise reduction: Remove noise



Image denoising

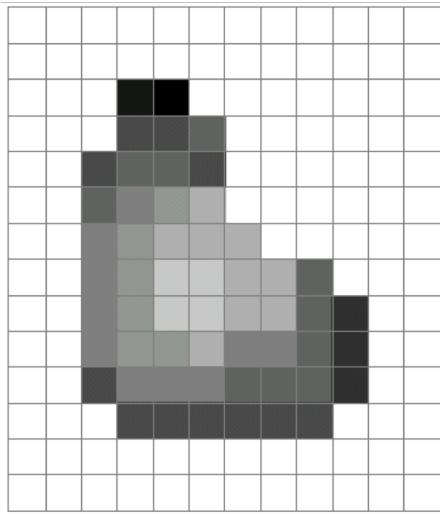


Why would images have noise?

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

0 noise most plausible
Could be slightly more
black/white

low noise images more likely to occur
than high noise 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*

*** Process: Mean filtering ***

Mean filtering: Get neighborhood of pixels, compute mean of neighborhood, replace pixel with this value

Mean filtering

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

$$(0 + 0 + 0 + 0 + 0 + 10 + 0 + 0 + 0 + 20 + 10 + 40 + 0 + 0 + 20 + 10 + 0 + 0 + 0 + 0 + 30 + 14 + 20 + 10 + 0 + 0) / 25 = 6.8$$

Mean filtering

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

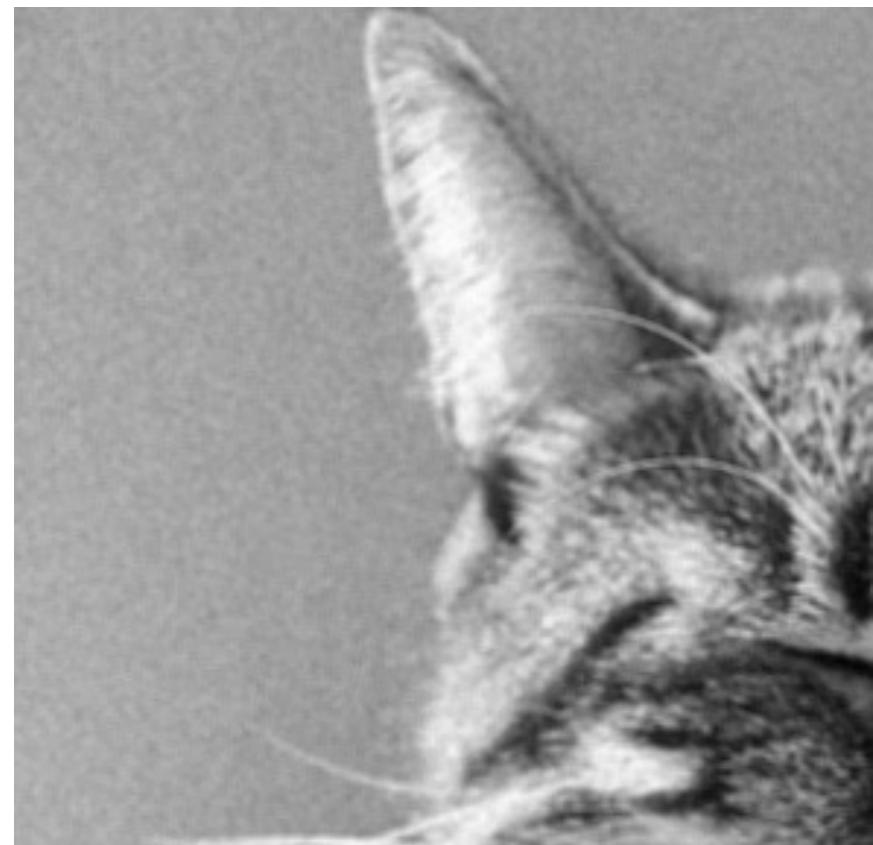
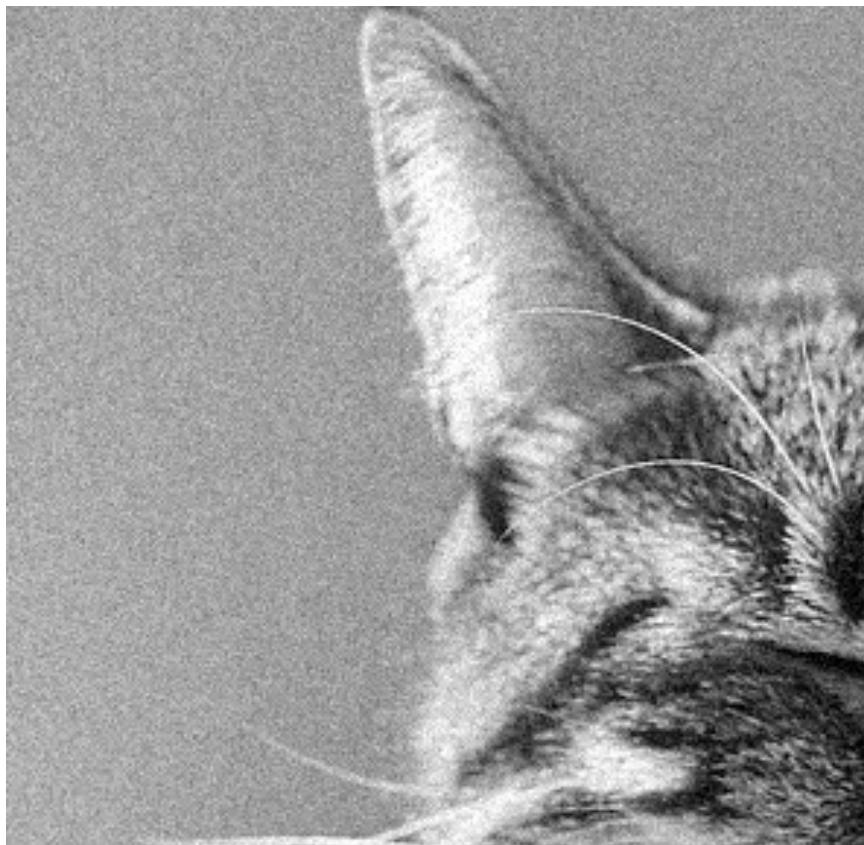
$$(0 + 0 + 0 + 0 + 10 + 10 + 10 + 20 + 20) / 9 = 7.77$$

Mean filtering

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0	0
0	10	20	30	0	20	10	0	0	0	0
0	10	0	30	40	30	20	10	0	0	0
0	10	20	30	40	30	20	10	0	0	0
0	10	20	10	40	30	20	10	0	0	0
0	10	20	30	30	20	10	0	0	0	0
0	0	10	20	20	0	10	0	20	0	0
0	0	0	10	10	10	0	0	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
0	4	11	13	16	11	12	7	4	0	0
0	6	14	19	23	19	18	10	6	0	0
0	8	18	23	28	23	17	8	2	0	0
0	8	16	26	31	30	20	10	3	0	0
0	10	18	27	29	27	17	8	2	0	0
0	8	14	22	22	20	11	8	3	0	0
0	4	11	17	17	12	6	4	2	0	0
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



		4.1

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



	7	

Local image data

Kernel / filter

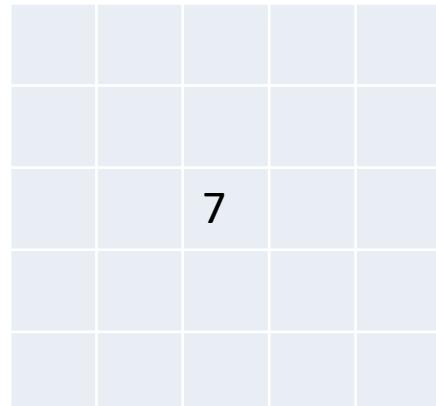


$$S[f](m, n) = \sum_{i=-1}^1 \sum_{j=-1}^1 w(i, j) f(m + i, n + j)$$

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



$k = \#$ of 'levels' used for operation

Kernel size = $2k+1$

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

A more general version

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

- $w(i,j) = 1/(2k+1)^2$ for mean filter
- If $w(i,j) \geq 0$ and sum to 1, *weighted mean*
- But $w(i,j)$ can be *arbitrary real numbers!*
- This operation is called *cross-correlation*

Boundary conditions

Clearly valid

90	0	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0	0	0
0	0	10	20	20	20	10	40	0	0	0	0
0	10	20	30	0	20	10	0	0	0	0	0
0	10	0	30	40	30	20	10	0	0	0	0
0	10	20	30	40	30	20	10	0	0	0	0
0	10	20	10	40	30	20	10	0	0	0	0
0	10	20	30	30	20	10	0	0	0	0	0
0	0	10	20	20	0	10	0	20	0	0	0
0	0	0	10	10	10	0	0	0	0	0	0

Boundary conditions

Can do if we assume empty
space is 0

90	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0	0
0	0	10	20	20	20	10	40	0	0	0
0	10	20	30	0	20	10	0	0	0	0
0	10	0	30	40	30	20	10	0	0	0
0	10	20	30	40	30	20	10	0	0	0
0	10	20	10	40	30	20	10	0	0	0
0	10	20	30	30	20	10	0	0	0	0
0	0	10	20	20	0	10	0	20	0	0
0	0	0	10	10	10	0	0	0	0	0

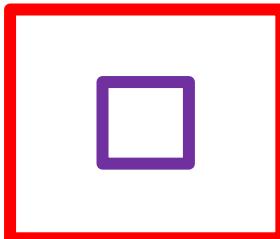
Boundary conditions

Can still potentially compute



90	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0	0
0	0	10	20	20	20	10	40	0	0	0
0	10	20	30	0	20	10	0	0	0	0
0	10	0	30	40	30	20	10	0	0	0
0	10	20	30	40	30	20	10	0	0	0
0	10	20	10	40	30	20	10	0	0	0
0	10	20	30	30	20	10	0	0	0	0
0	0	10	20	20	0	10	0	20	0	0
0	0	0	10	10	10	0	0	0	0	0

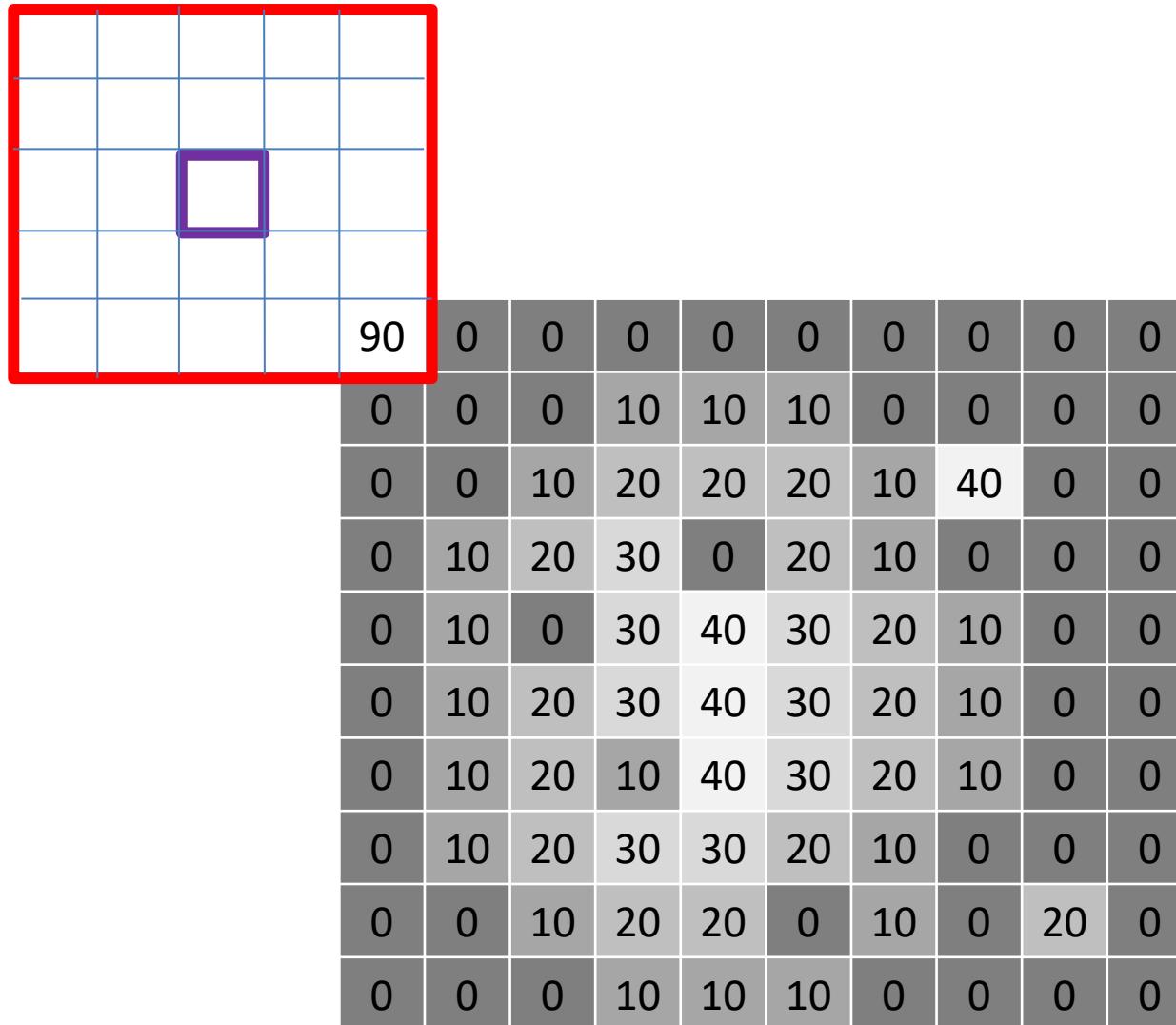
Boundary conditions



Output will be 0 so no point computing

90	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0	0
0	0	10	20	20	20	10	40	0	0	0
0	10	20	30	0	20	10	0	0	0	0
0	10	0	30	40	30	20	10	0	0	0
0	10	20	30	40	30	20	10	0	0	0
0	10	20	10	40	30	20	10	0	0	0
0	10	20	30	30	20	10	0	0	0	0
0	0	10	20	20	0	10	0	20	0	0
0	0	0	10	10	10	0	0	0	0	0

Boundary conditions



Boundary conditions

- Setup
 - $m \times m$ image
 - $k \times k$ kernel
- Output size?

Boundary conditions in practice

- “Full”: compute if *any* part of kernel intersects with image
 - Assumes image is padded with 0’s
 - Output size = $m+k-1$
 - Technically cross-correlation and convolution means this
 - 0 padding can cause artifacts
- “Same convolution”: compute if center of kernel is in image
 - Assumes image is padded with 0’s
 - output size = m
 - Common in practice
- “Valid convolution”: compute only if *all* of kernel is in image
 - no padding
 - output size = $m-k+1$
 - No artifacts

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 equivariance

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

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



f



f'

Shift equivariance

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

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

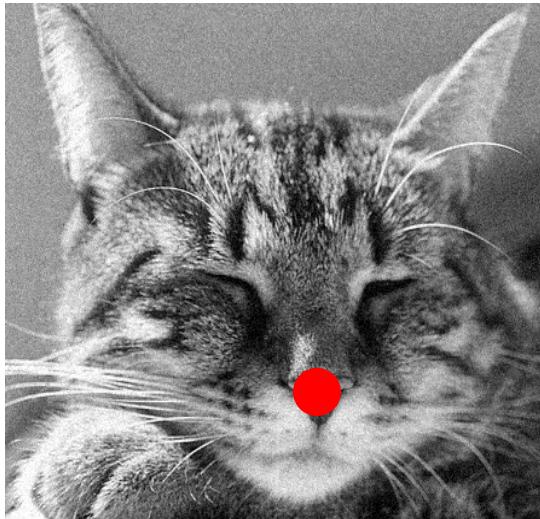
$$\begin{aligned} (w \otimes f')(m, n) &= \sum_{i=-k}^k \sum_{j=-k}^k w(i, j) f'(m + i, n + j) \\ &= \sum_{i=-k}^k \sum_{j=-k}^k w(i, j) f(m + i - m_0, n + j - n_0) \\ &= (w \otimes f)(m - m_0, n - n_0) \end{aligned}$$

Shift equivariance

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

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

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



f



f'

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

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

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

Cross-correlation

1	2	3
4	5	6
7	8	9

w

1	2	3
4	5	6
7	8	9

f

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

Convolution

1	2	3
4	5	6
7	8	9

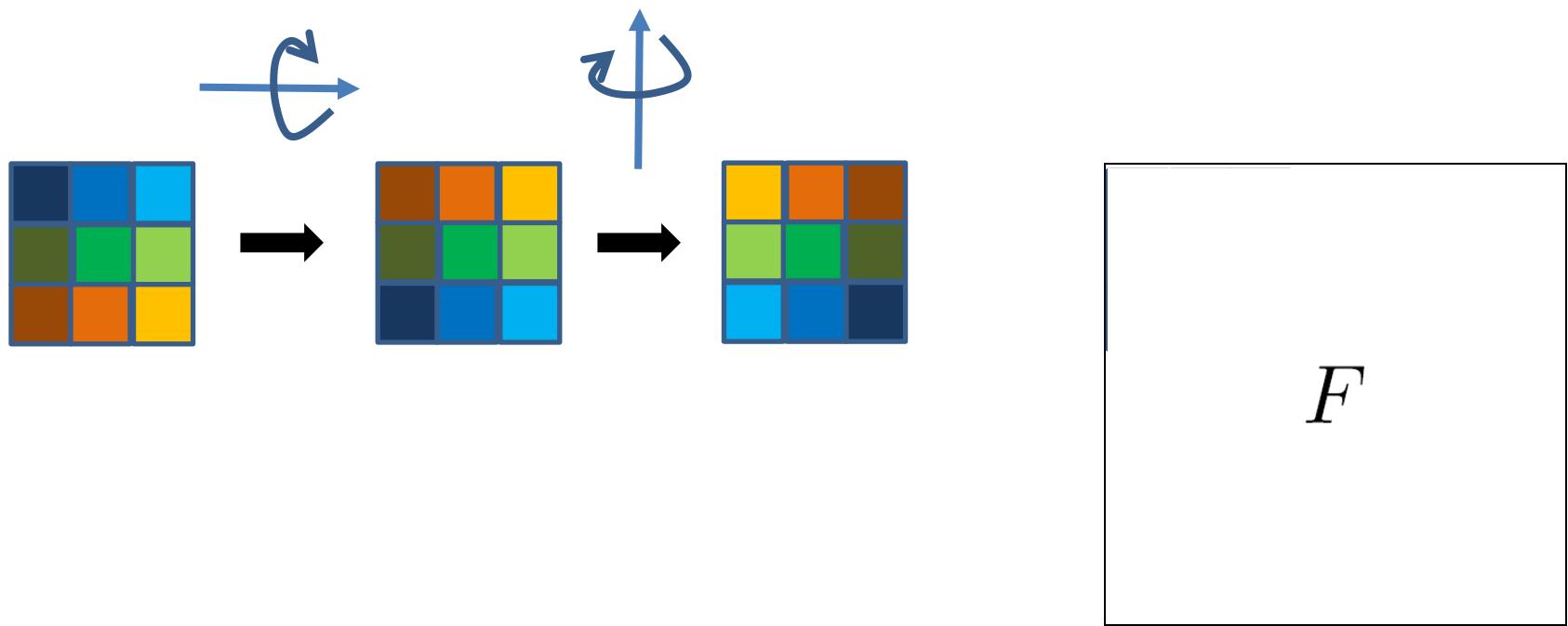
w

1	2	3
4	5	6
7	8	9

f

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

Convolution



Convolution

- Convolution is the more fundamental operation
- “Full” convolution satisfies associative property: commutative
 - $a * (b * c) = (a * b) * c$

Filters: examples



Original (f)

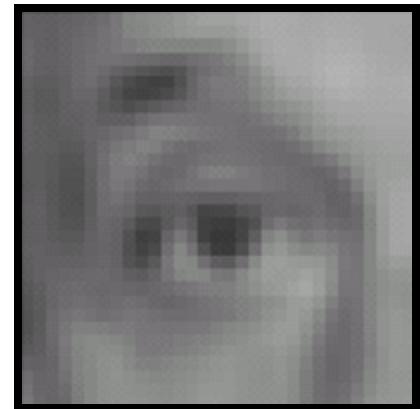
*

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

1	1	1
1	1	1
1	1	1

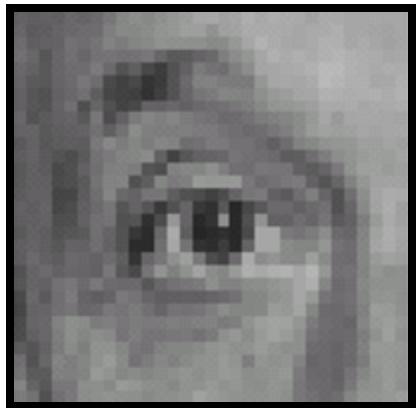
Kernel (k)

=



Blur (with a mean filter) (g)

Filters: examples



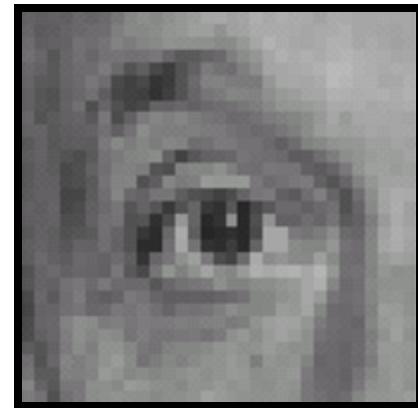
Original (f)

*

0	0	0
0	1	0
0	0	0

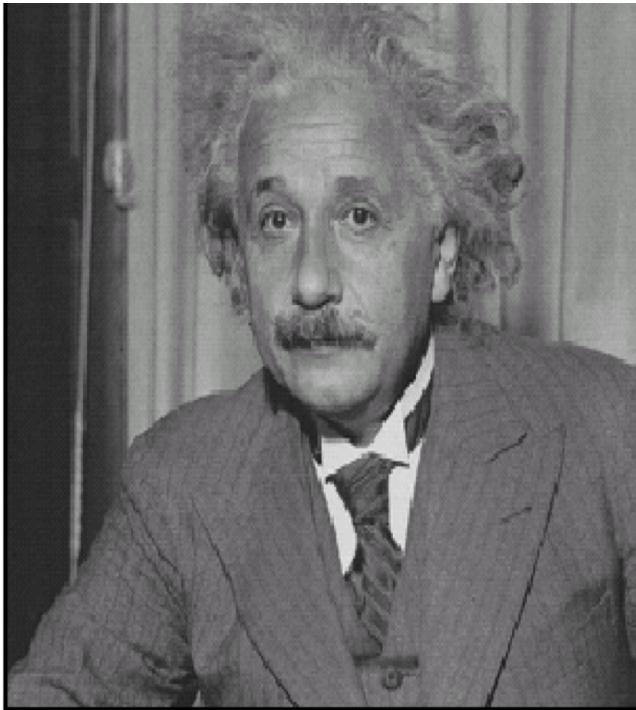
Kernel (k)

=

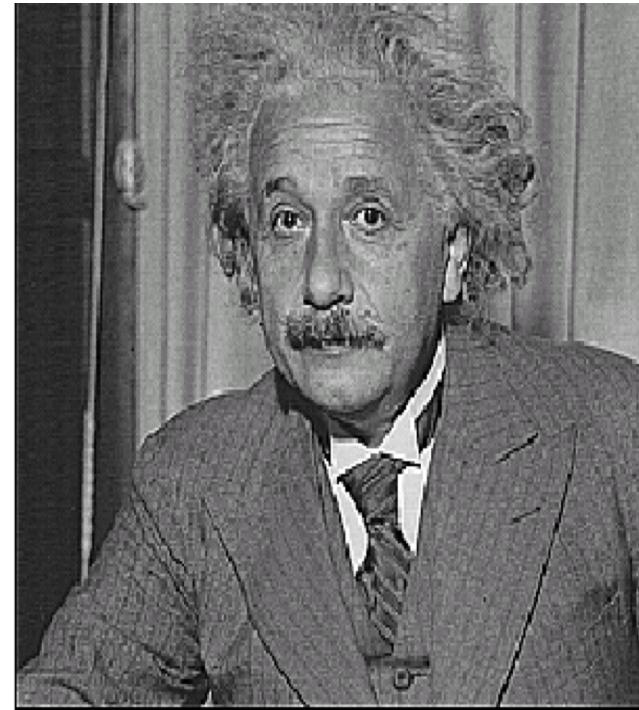


Identical image (g)

Sharpening



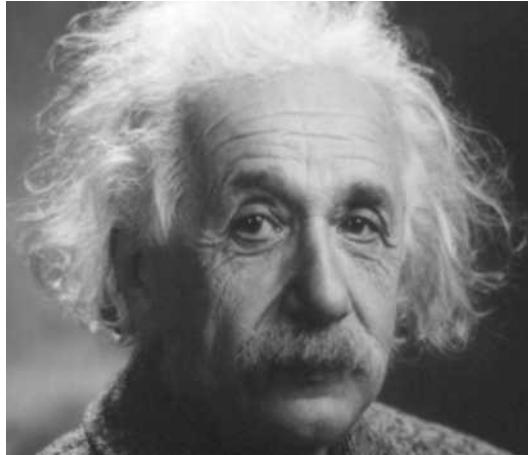
before



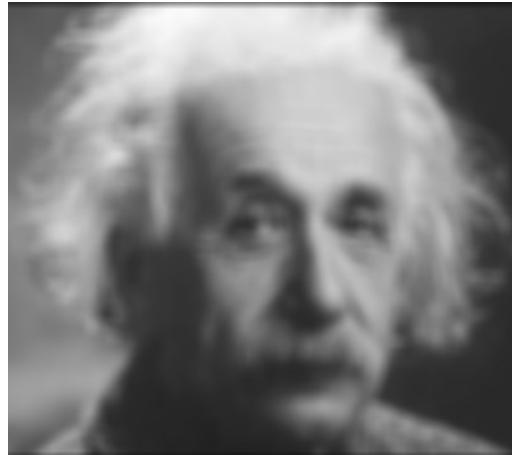
after

Sharpening

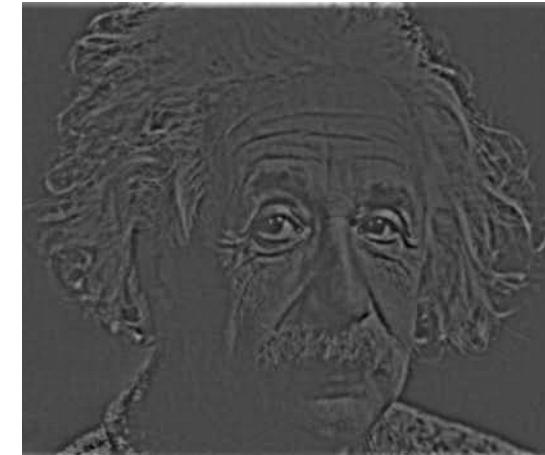
- What does blurring take away?



-



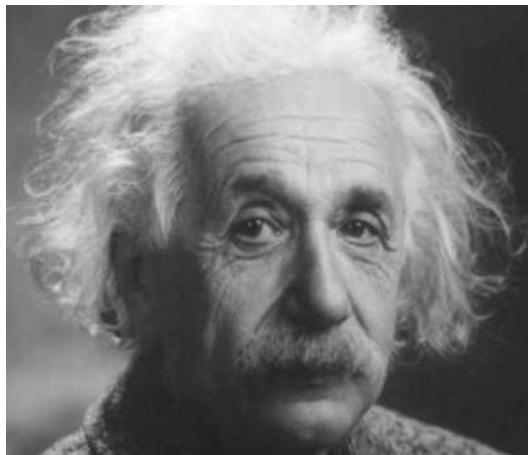
=



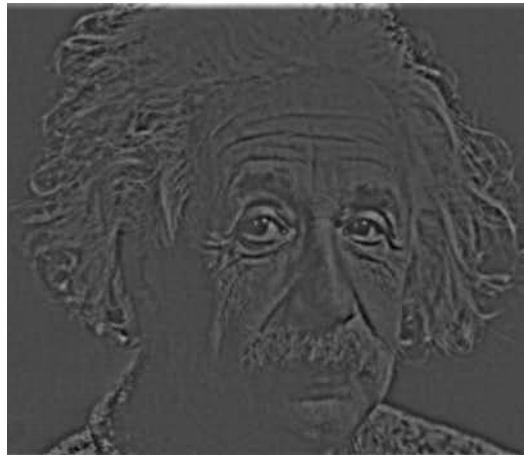
has more details

original image - blurred image

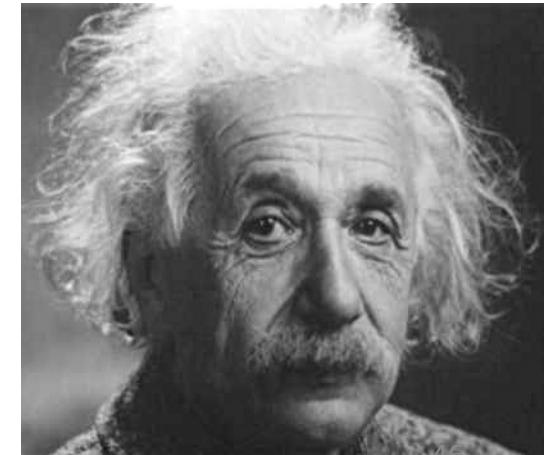
Let's add it back:



$+ \alpha$

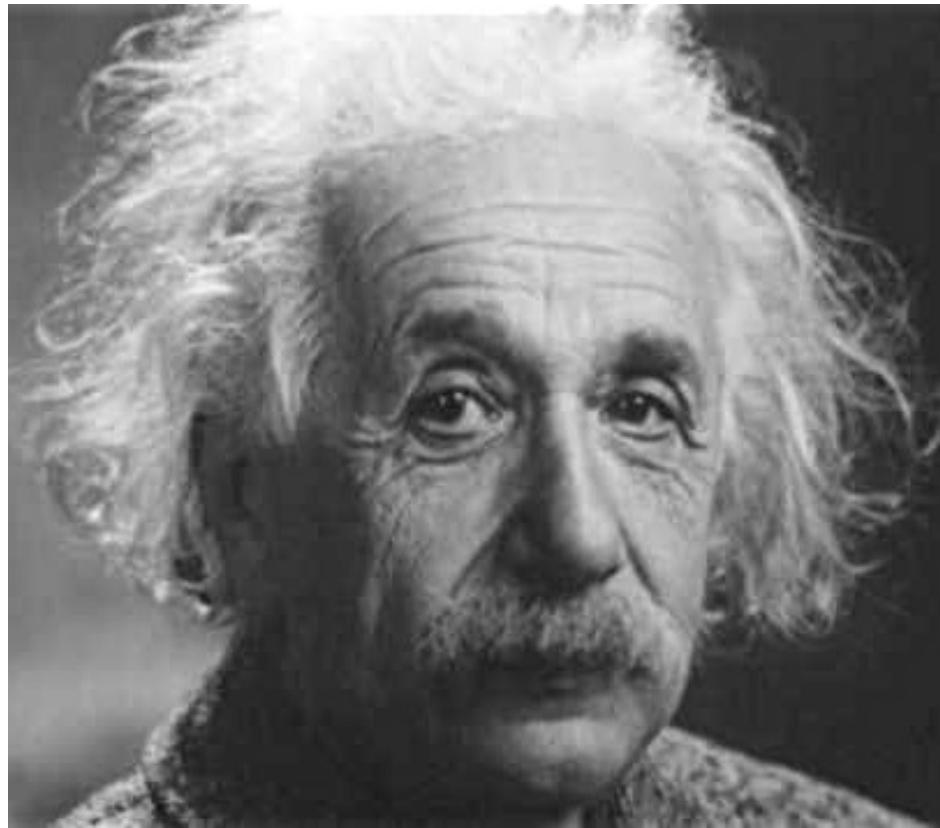


=



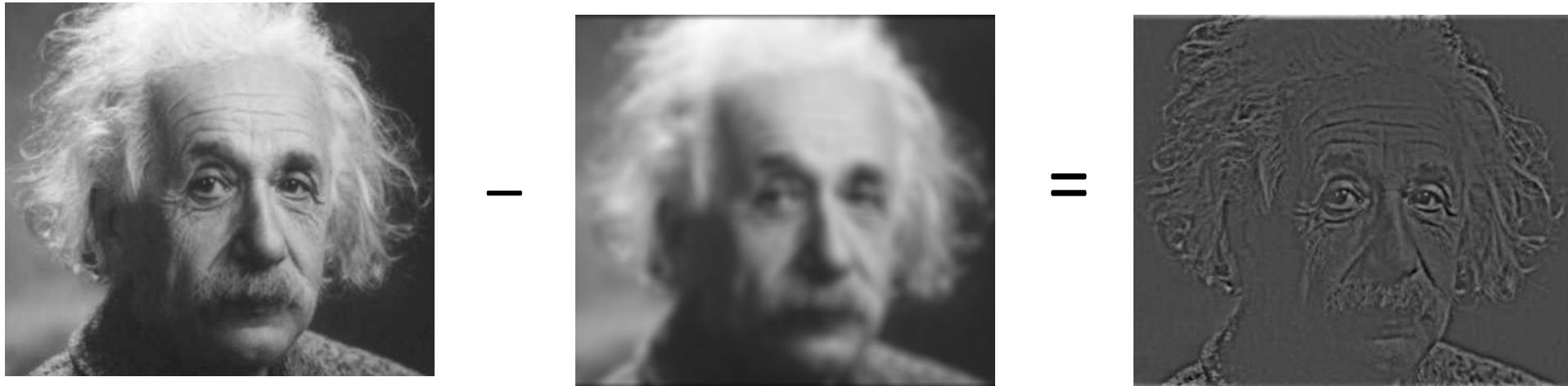
add back to original image but multiply by alpha, making it sharper

Sharpening

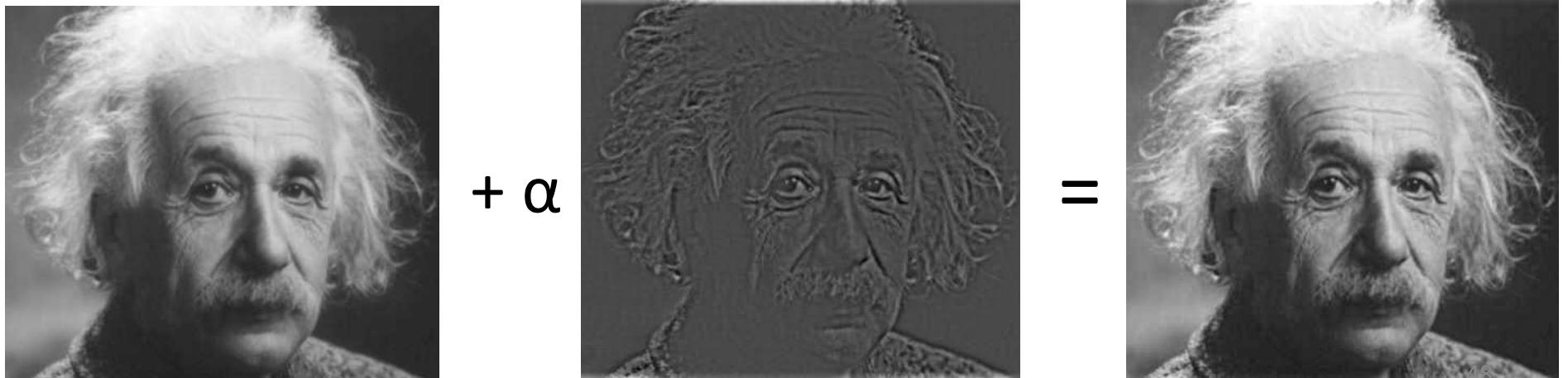


Sharpening

- What does blurring take away?



Let's add it back:



Sharpening

$$\begin{aligned}f_{sharp} &= f + \alpha(f - f_{blur}) \\&= (1 + \alpha)f - \alpha f_{blur} \\&= (1 + \alpha)(w * f) - \alpha(v * f)\end{aligned}$$



0	0	0
0	1	0
0	0	0

original image with
identity filter



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

blurred image is original
with mean filter

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

Sharpening filter



Original

$$\text{Original} * \left(\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{array} - \frac{1}{9} \begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array} \right) = \text{Sharpened Image}$$



Sharpening filter
(accentuates edges)

Another example

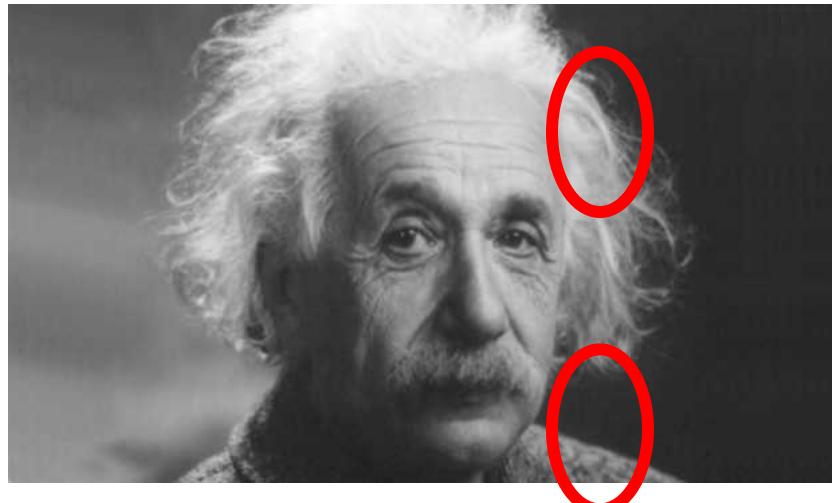
Pixels multiplied by 1 will have high value

Pixels multiplied by -1 will have low value

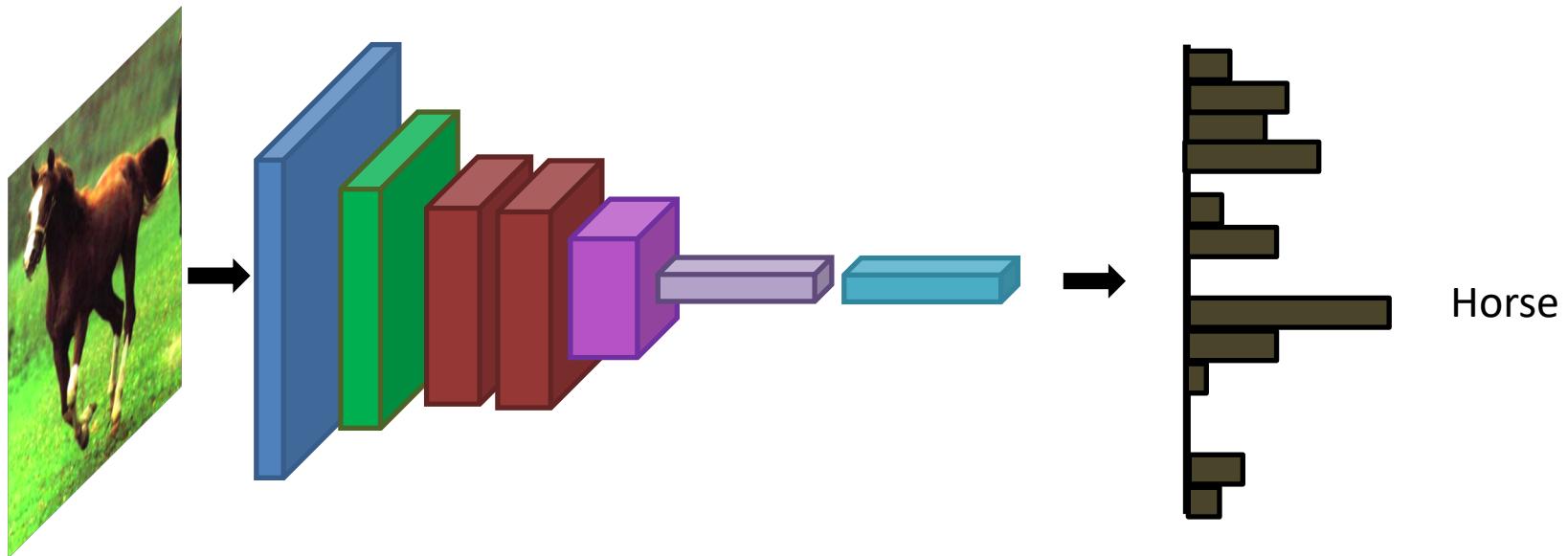
Values along the diagonal with have high response

-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	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

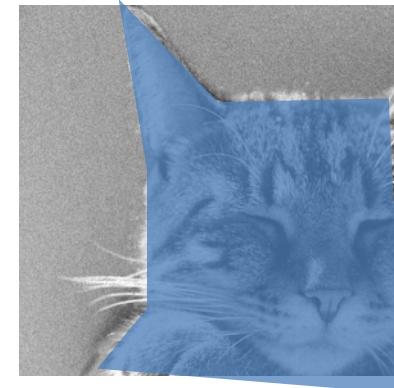
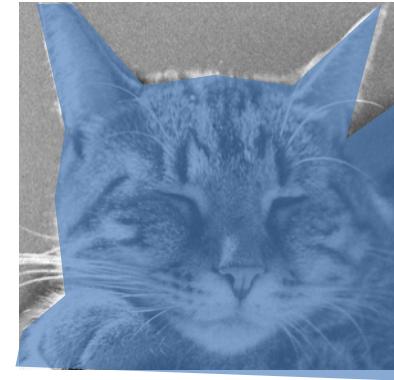


Convolution is everywhere



Why is convolution important?

- Shift equivariance 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-equivariant 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-equivariant: 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