Image Filtering

EECS 442 – David Fouhey

Winter 2023, University of Michigan

https://web.eecs.umich.edu/~fouhey/teaching/EECS442_W23/

Let's Take An Image

Let's Fix Things

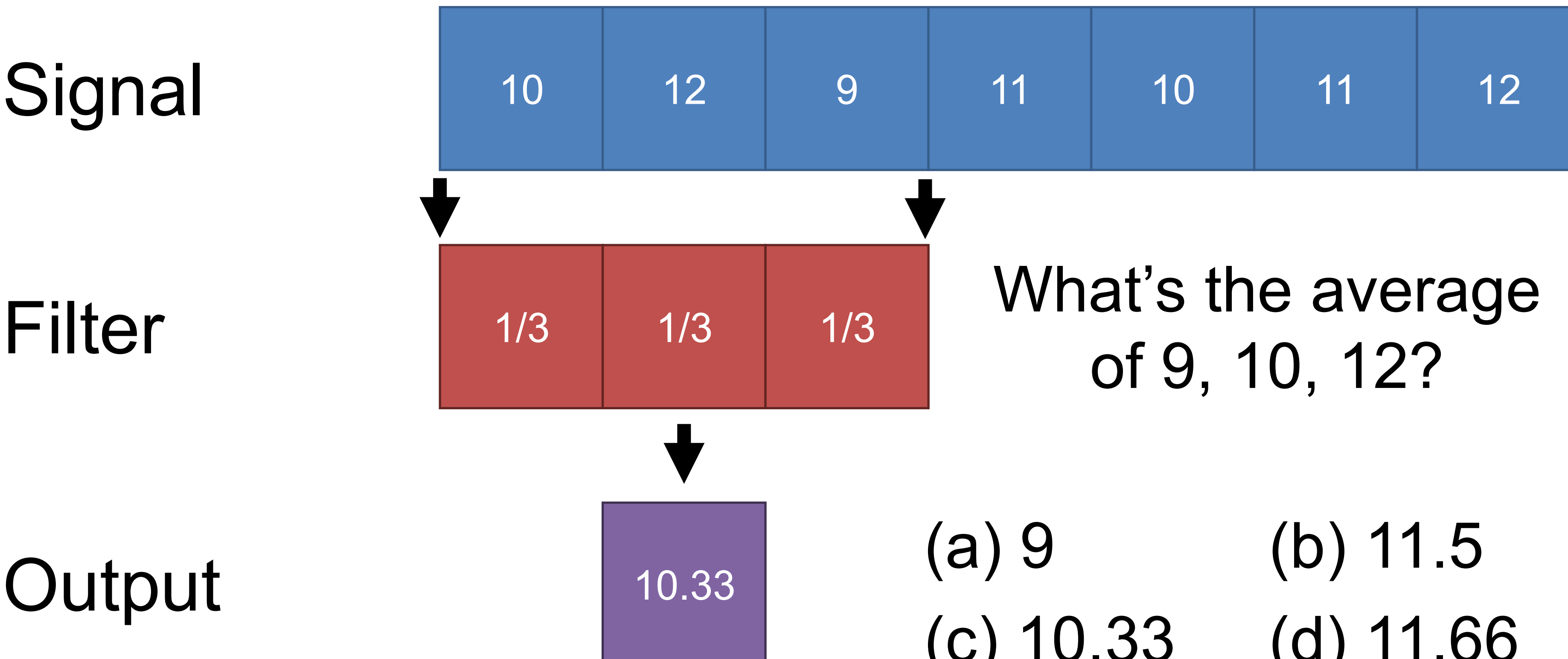
- We have noise in our image
- Let's replace each pixel with a *weighted* average of its neighborhood
- Weights are *filter kernel*

	Out	

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Slide Credit: D. Lowe

1D Case

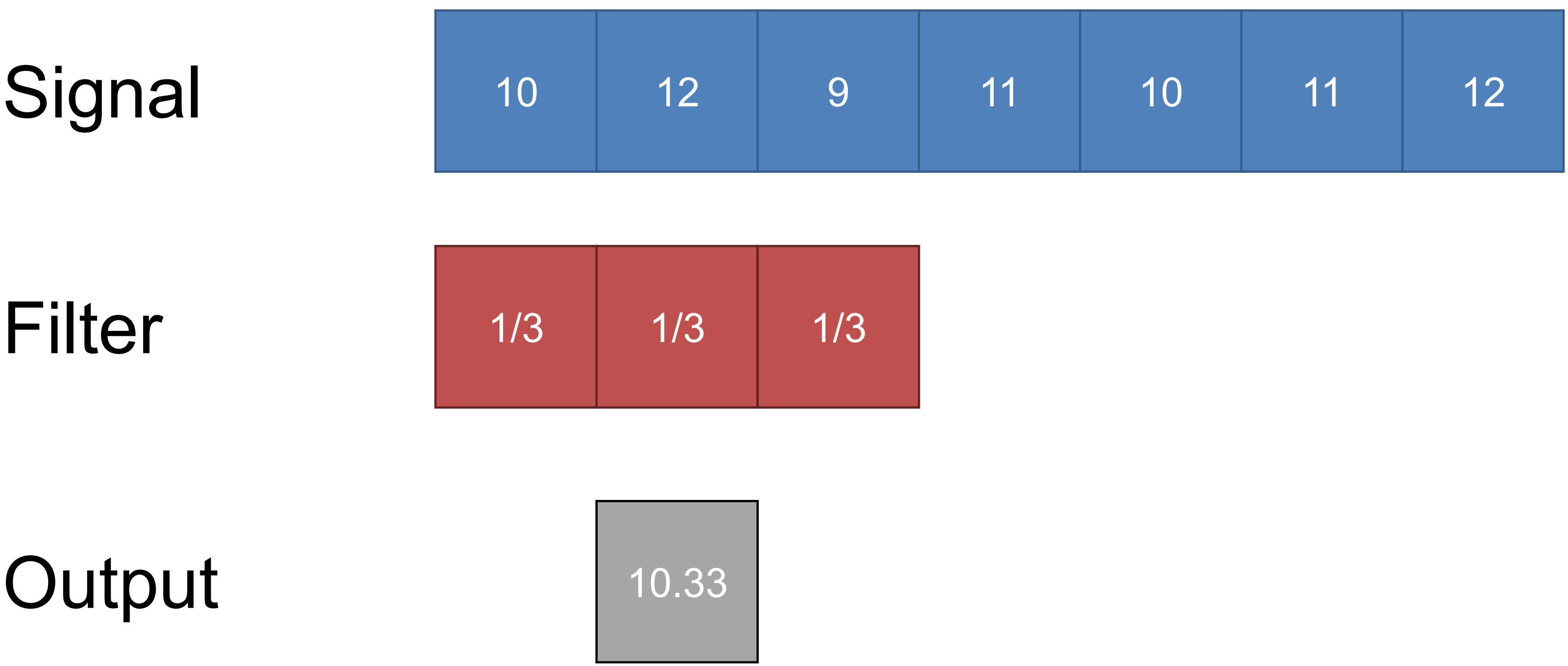


What's the average of 9, 10, 12?

(a) 9 (b) 11.5

(c) 10.33 (d) 11.66

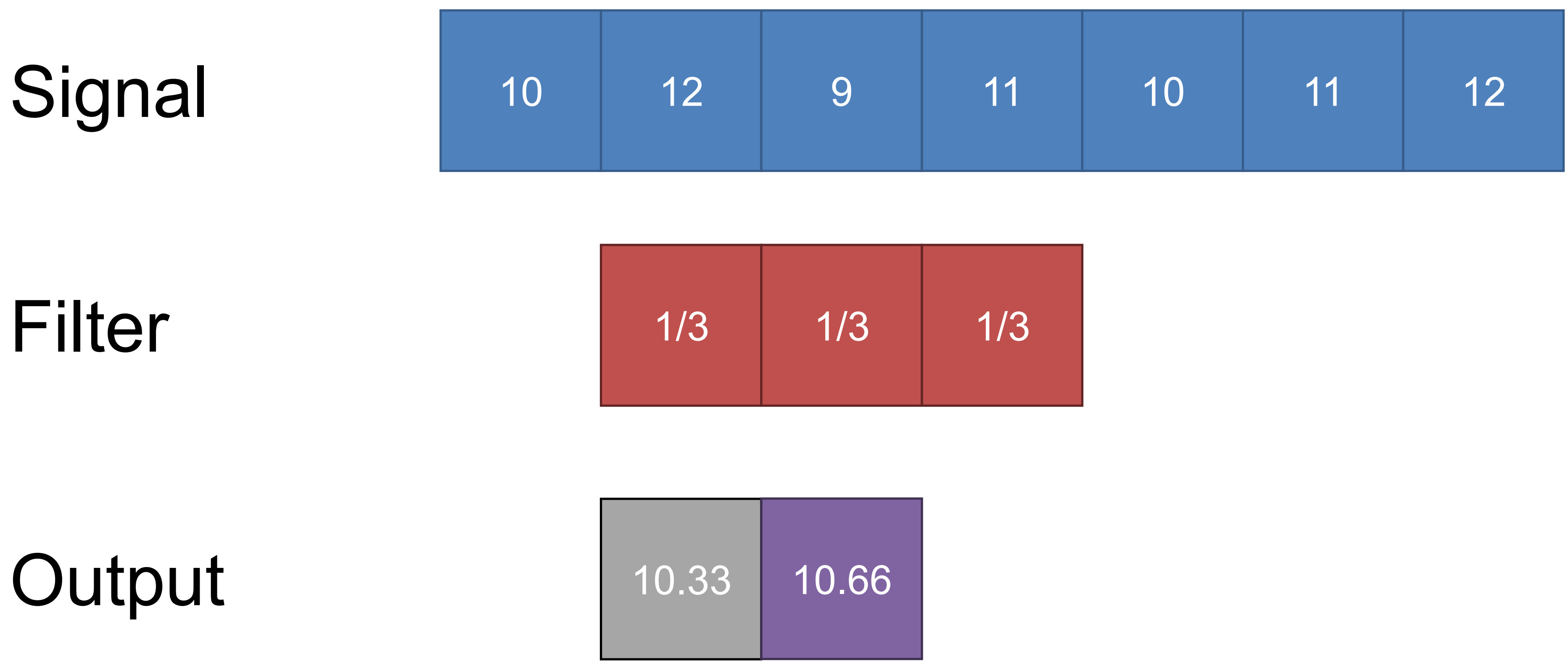
1D Case



Done! Next?

1D Case

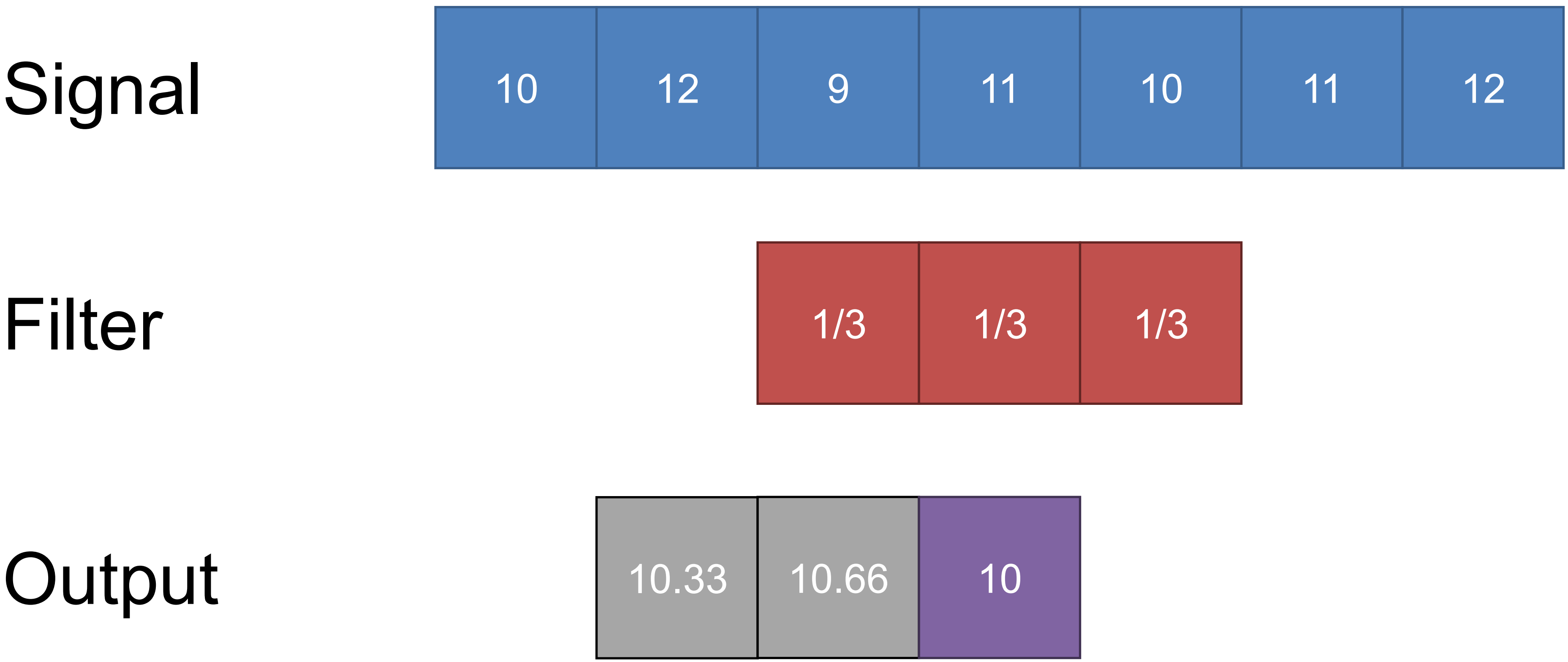
(1) 10.66 (2) 9.33
(3) 14.2 (4) 11.33



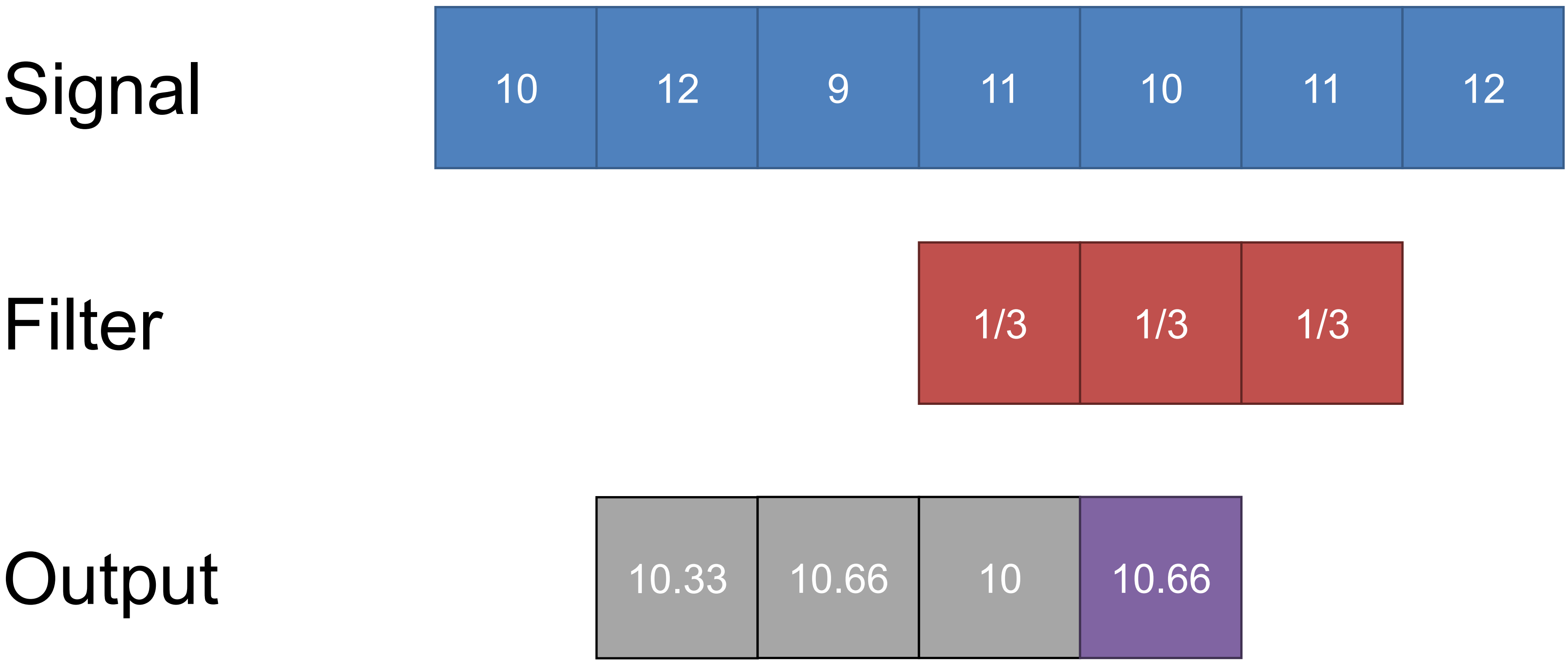
1D Case

(1) 10.33 (2) 11.33

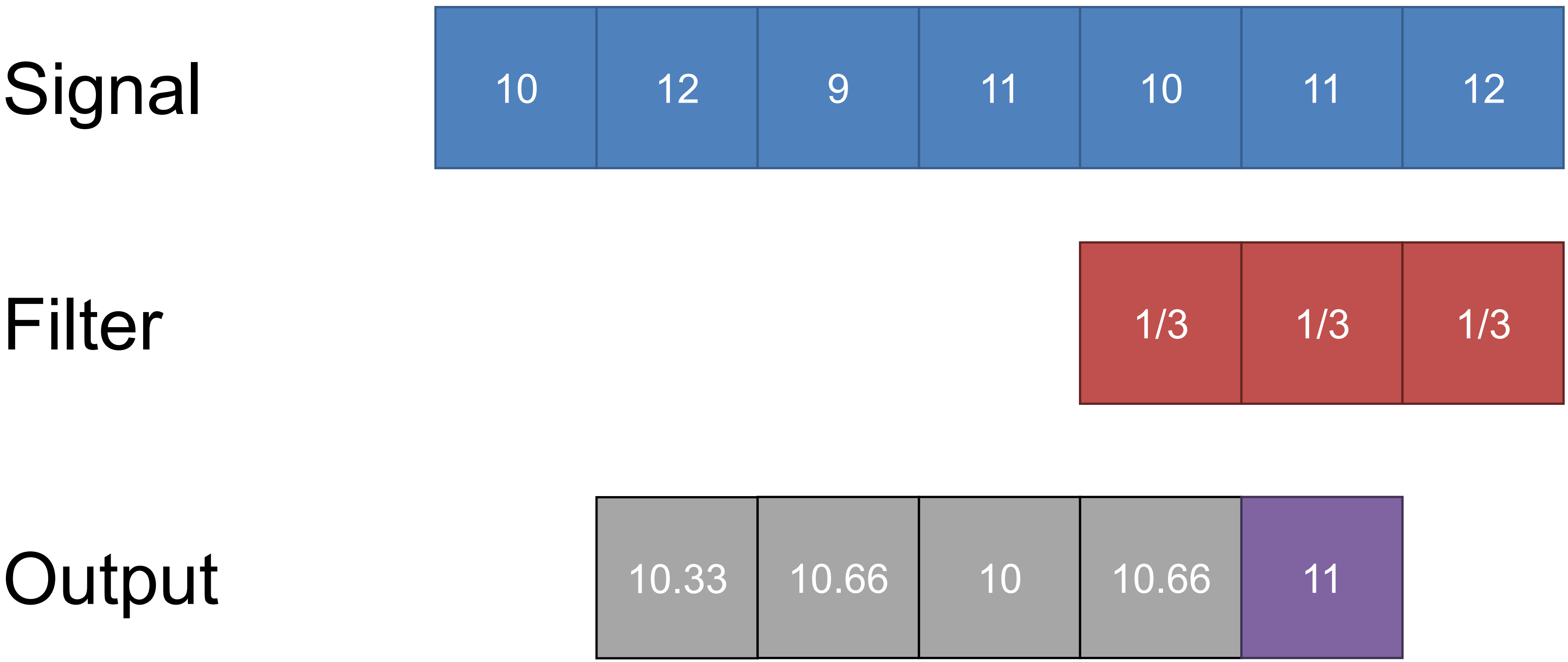
(3) 10 (4) 9.1



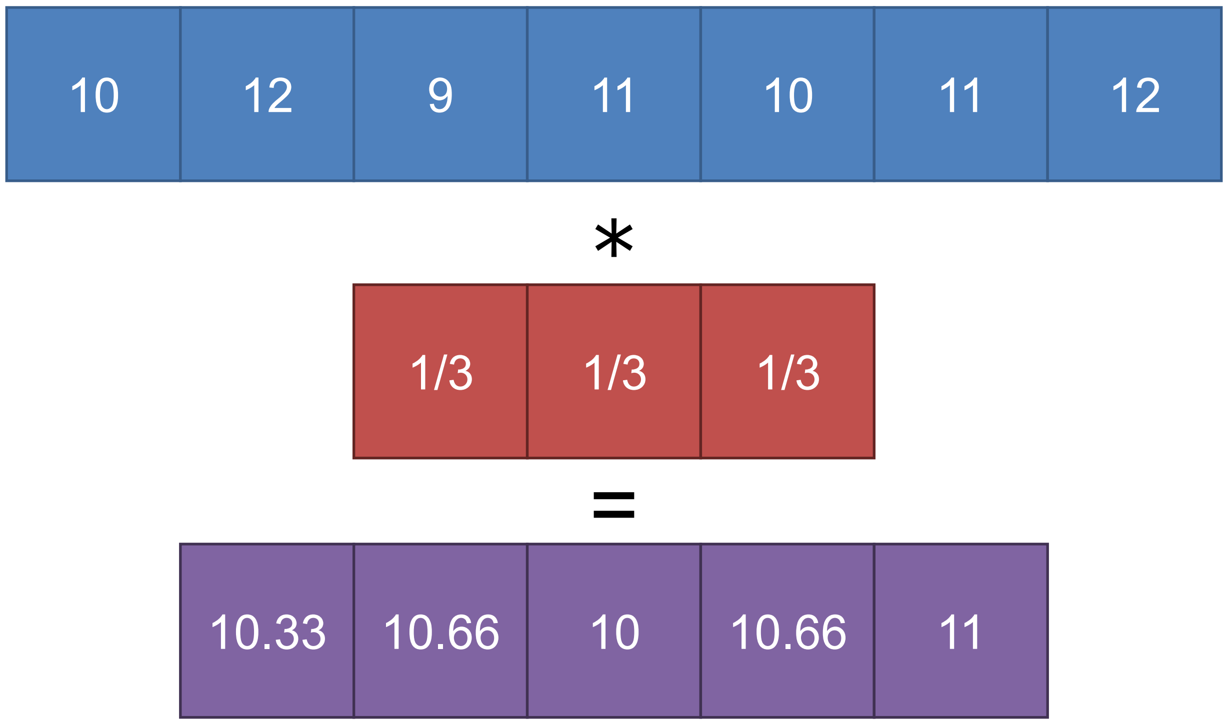
1D Case


Signal
10
12
9
11
10
11
12
Filter
1/3
1/3
1/3
Output
10.33
10.66
10
10.66

1D Case


Signal
10
12
9
11
10
11
12
Filter
1/3
1/3
1/3
Output
10.33
10.66
10
10.66
11

1D Case



You lose pixels (maybe…)

Filter “sees” only a few pixels at a time

Applying a Linear Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

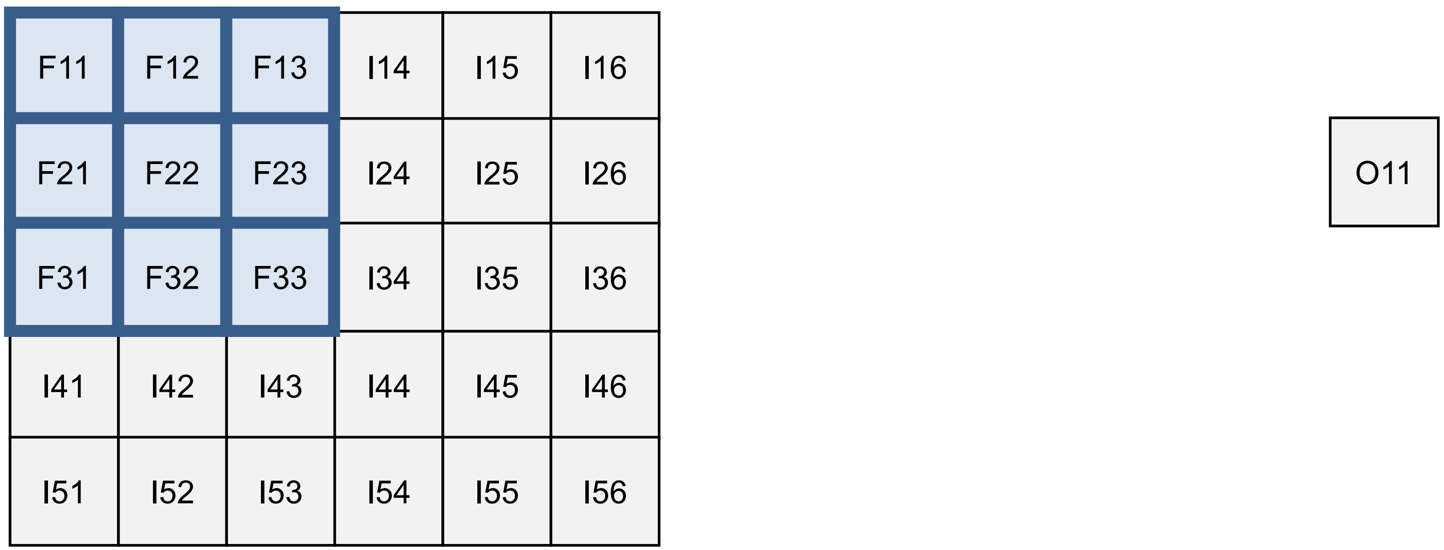
Output

O11	O12	O13	O14
O21	O22	O23	O24
O31	O32	O33	O34

Applying a Linear Filter

Input & Filter

Output

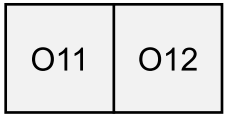


$$O11 = I11*F11 + I12*F12 + \ldots + I33*F33$$

Applying a Linear Filter

Input & Filter

Output



$$O12 = I12*F11 + I13*F12 + \ldots + I34*F33$$

Applying a Linear Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

How many times can we apply a 3x3 filter to a 5x6 image?

Applying a Linear Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

O11	O12	O13	O14
O21	O22	O23	O24
O31	O32	O33	O34

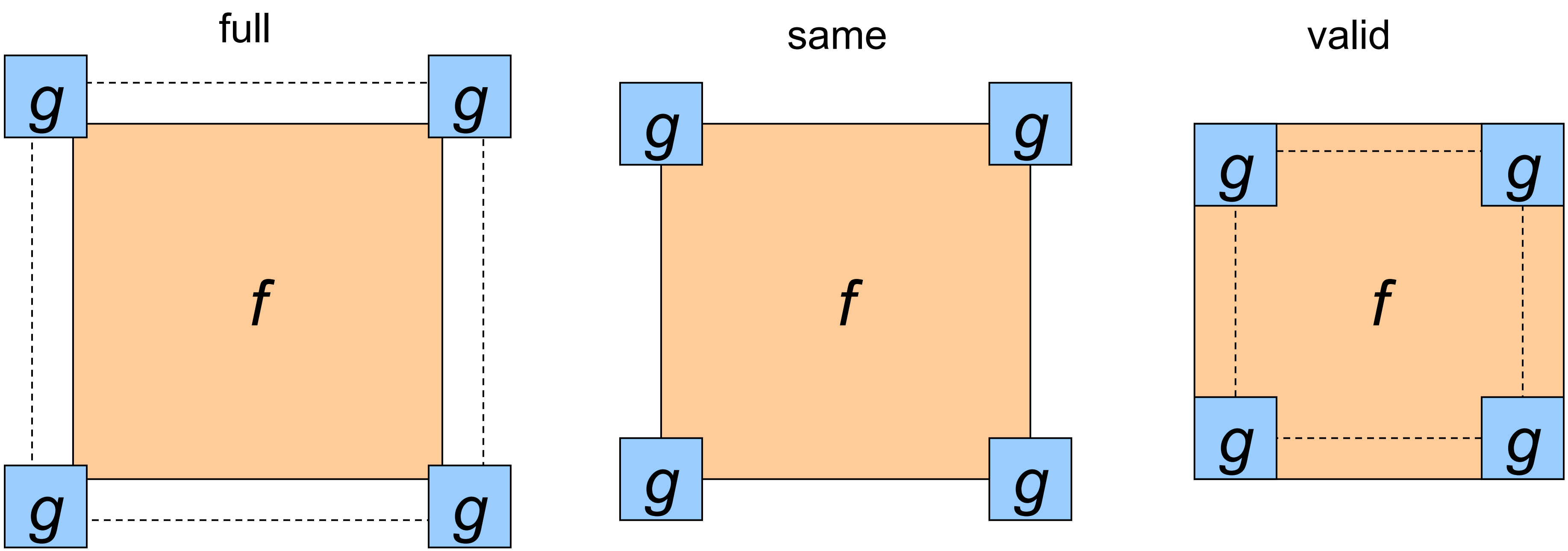
$$Oij = Iij*F11 + Ii(j+1)*F12 + \ldots + I(i+2)(j+2)*F33$$

Painful Details – Edge Cases

Filtering doesn't keep the whole image.

Suppose **f** is the image and **g** the filter.

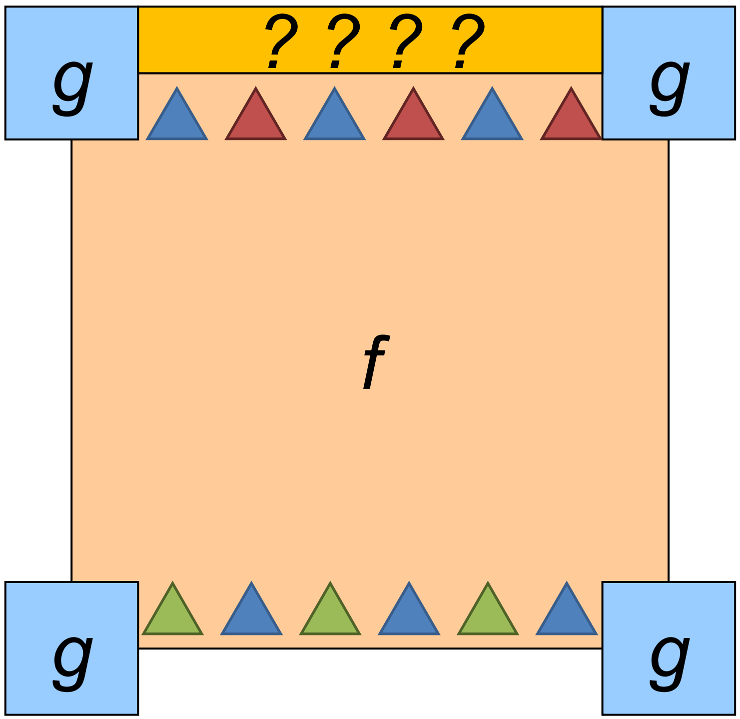
Full – any part of g touches f. **Same** – same size as f;
Valid – only when filter doesn't fall off edge.



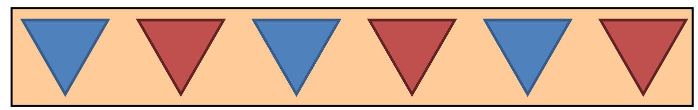
f/g Diagram Credit: D. Lowe

Painful Details – Edge Cases

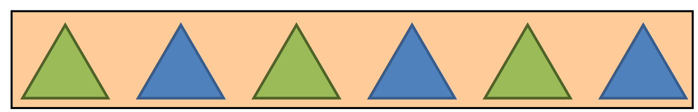
What to about the “?” region?



Symm: fold sides over



Circular/Wrap: wrap around



pad/fill: add value, often 0

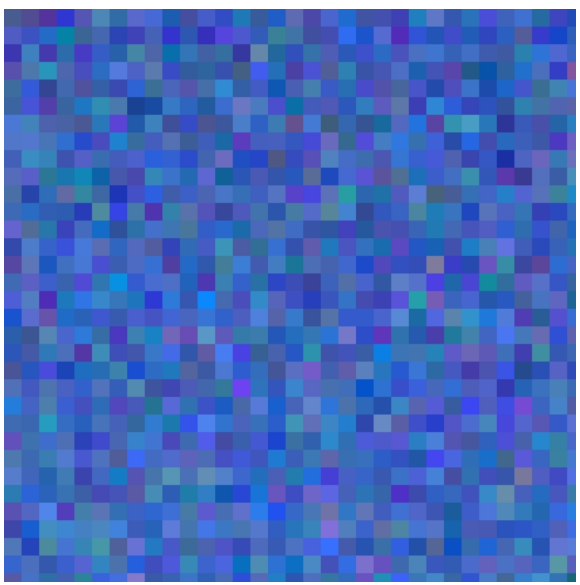
f/g Diagram Credit: D. Lowe

Painful Details – Does it Matter?

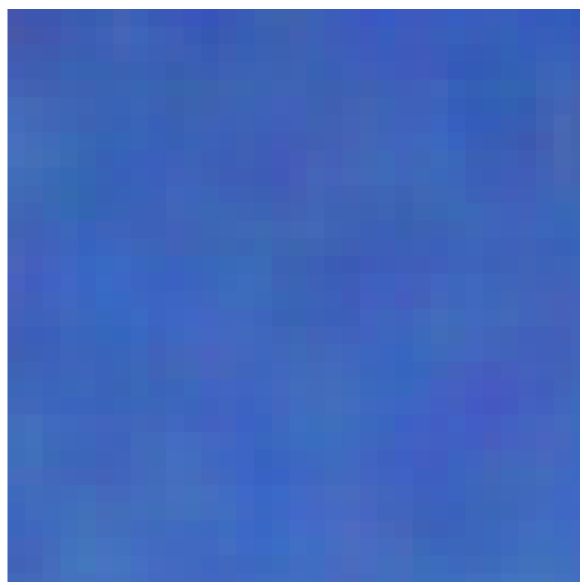
(I've applied the filter per-color channel)

Which padding did I use and *<u>why</u>*?

Input Image



Filtered ???



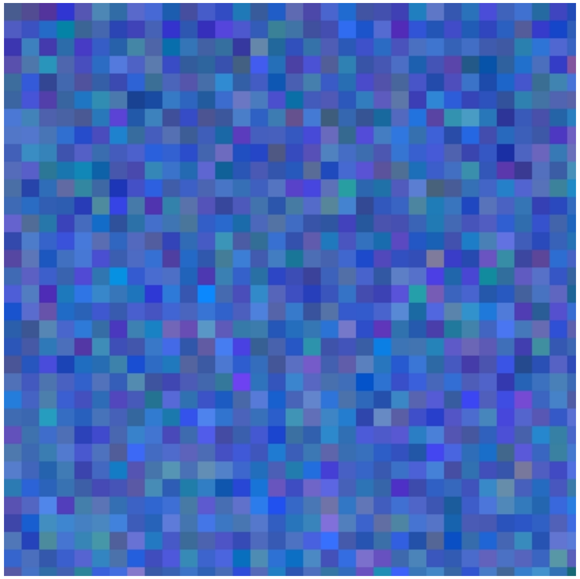
Filtered ???

Note – this is a zoom of the filtered, not a filter of the zoomed

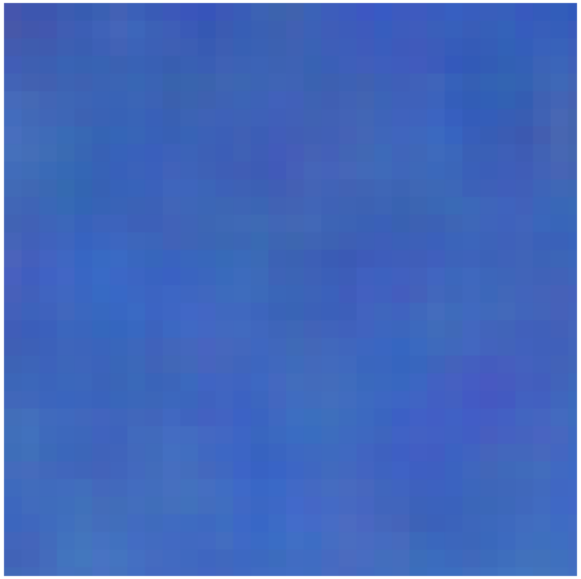
Painful Details – Does it Matter?

(I’ve applied the filter per-color channel)

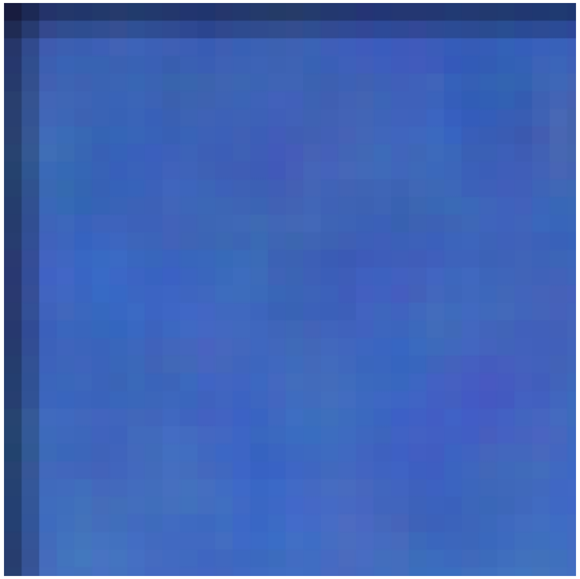
Input Image



Filtered Symm Pad

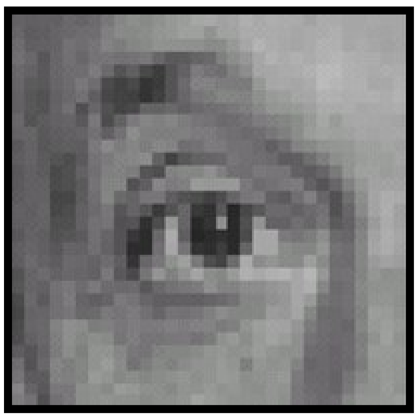


Filtered Zero Pad



Note – this is a zoom of the filtered, not a filter of the zoomed

Practice with Linear Filters



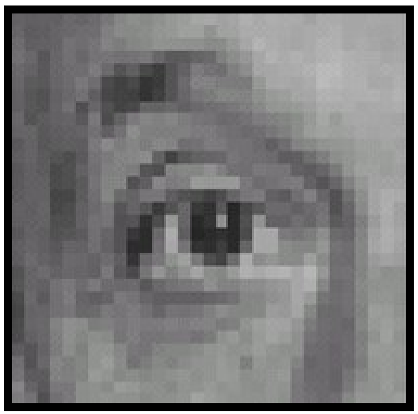
Original

0	0	0
0	1	0
0	0	0

?

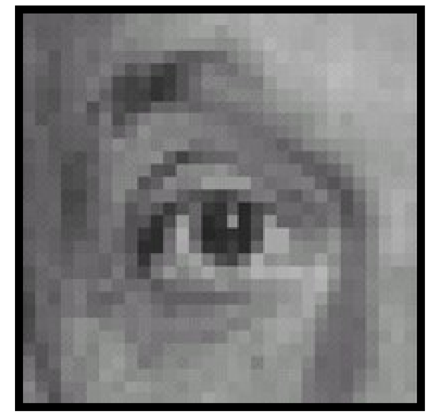
Slide Credit: D. Lowe

Practice with Linear Filters



Original

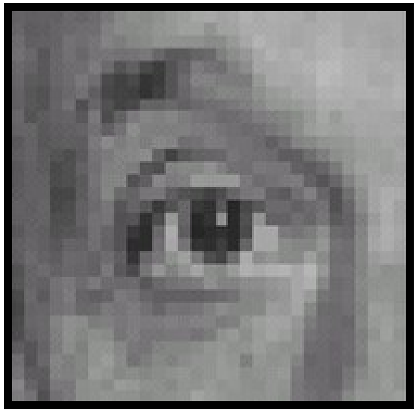
0	0	0
0	1	0
0	0	0



The Same!

Slide Credit: D. Lowe

Practice with Linear Filters



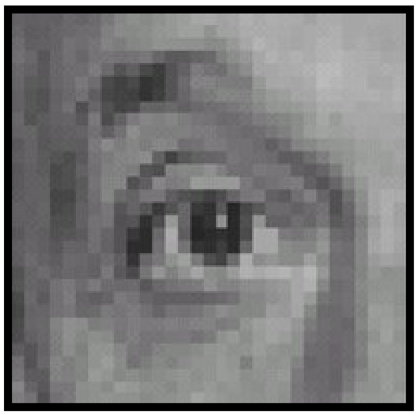
Original

0	0	0
0	0	1
0	0	0

?

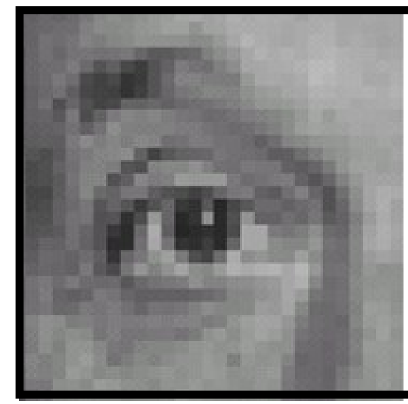
Slide Credit: D. Lowe

Practice with Linear Filters



Original

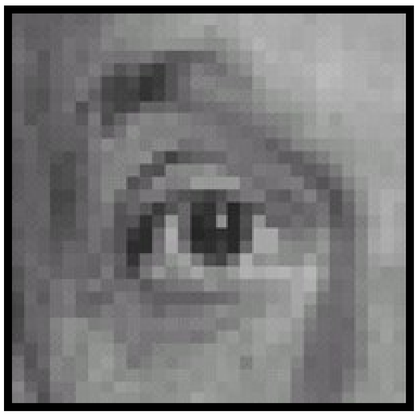
0	0	0
0	0	1
0	0	0



Shifted
LEFT
1 pixel

Slide Credit: D. Lowe

Practice with Linear Filters



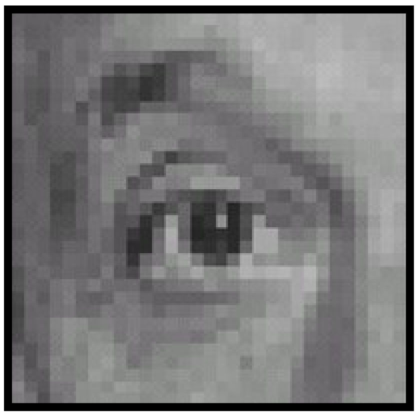
Original

0	1	0
0	0	0
0	0	0

?

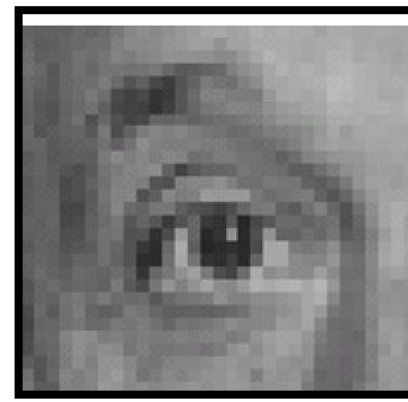
Slide Credit: D. Lowe

Practice with Linear Filters



Original

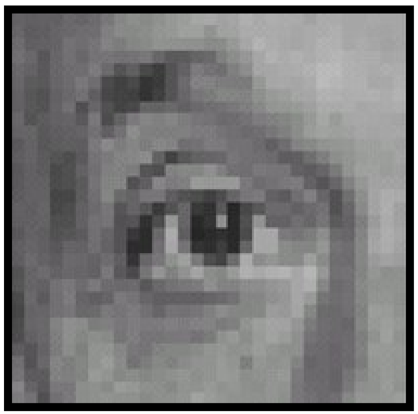
0	1	0
0	0	0
0	0	0



Shifted
<u>DOWN</u>
1 pixel

Slide Credit: D. Lowe

Practice with Linear Filters



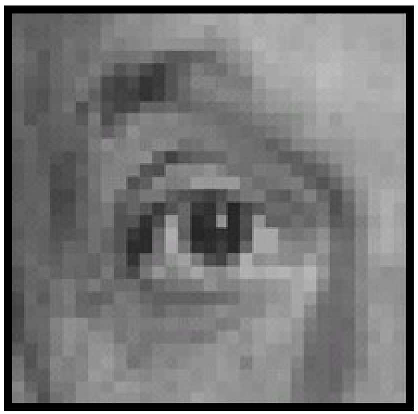
Original

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

?

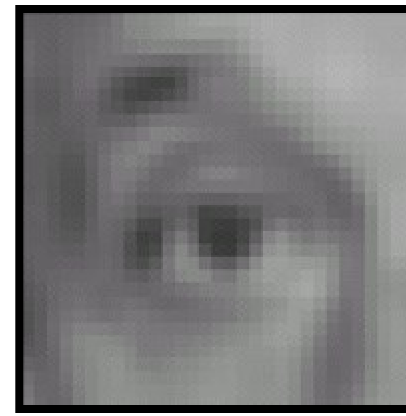
Slide Credit: D. Lowe

Practice with Linear Filters



Original

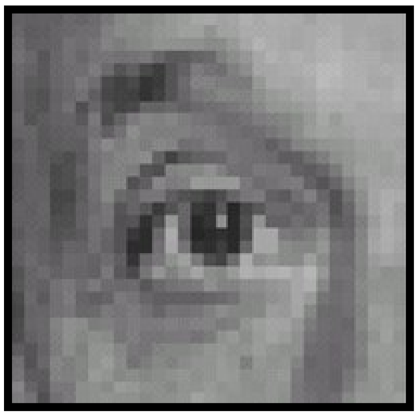
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



Blur
(Box Filter)

Slide Credit: D. Lowe

Practice with Linear Filters



Original

0	0	0
0	2	0
0	0	0

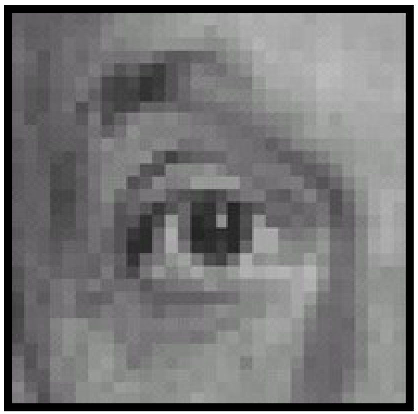
-

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

?

Slide Credit: D. Lowe

Practice with Linear Filters

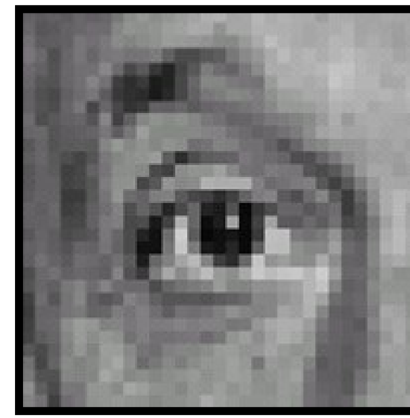


Original

0	0	0
0	2	0
0	0	0

-

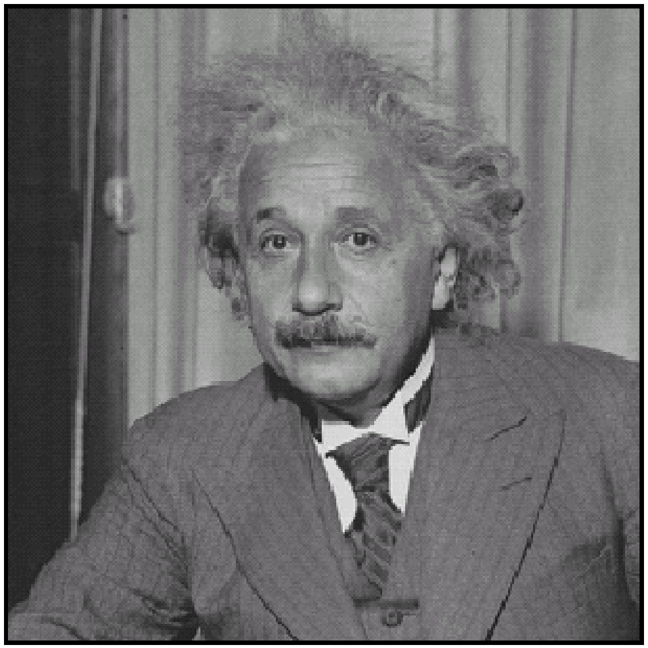
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



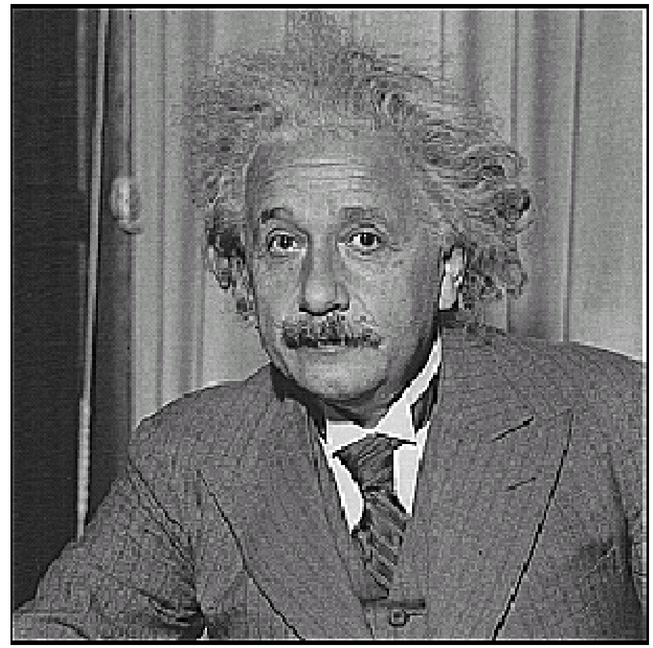
Sharpened
(Acccentuates difference from local average)

Slide Credit: D. Lowe

Sharpening



before



after

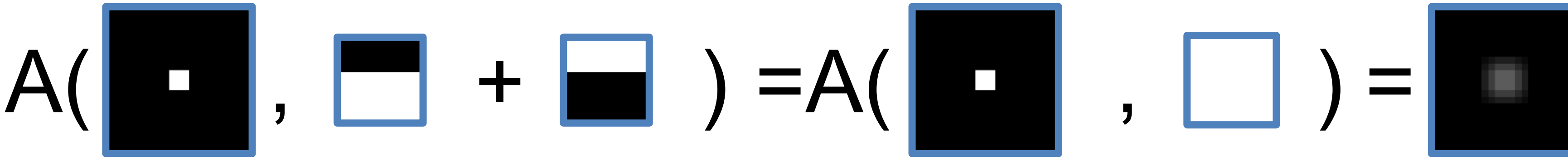
Slide Credit: D. Lowe

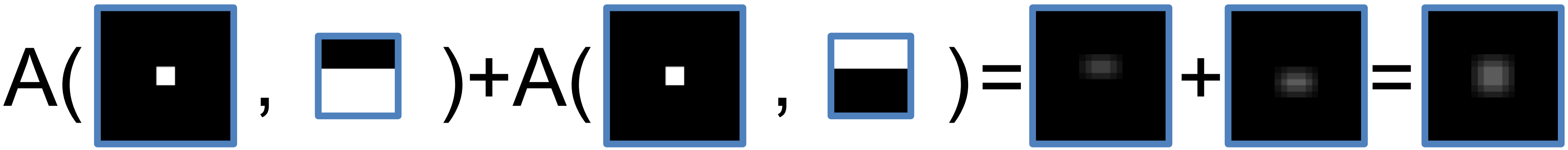
Properties – Linear

Assume: I image f1, f2 filters

Linear: apply(I,f1+f2) = apply(I,f1) + apply(I,f2)

I is a white box on black, and f1, f2 are rectangles





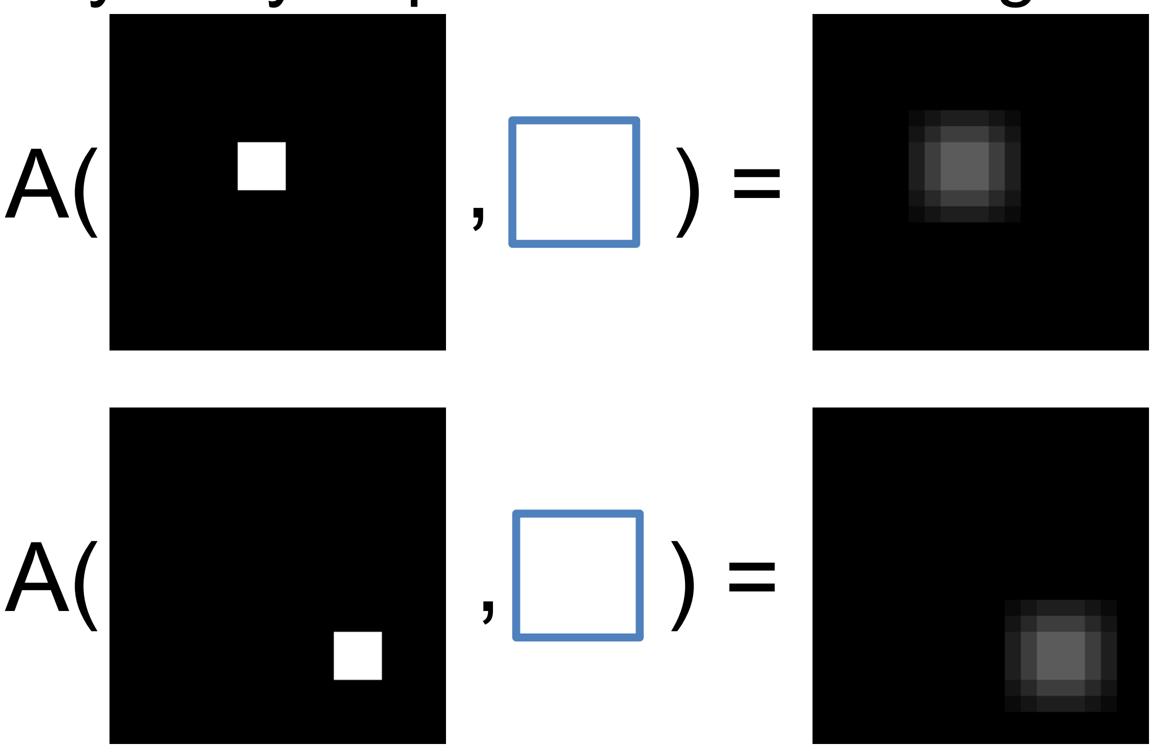
Note: I am showing filters un-normalized and blown up. They're a smaller box filter (i.e., each entry is 1/(size^2))

Properties – Shift-Invariant

Assume: I image, f filter

Shift-invariant: shift(apply(I,f)) = apply(shift(I,f))

Intuitively: only depends on filter neighborhood



Painful Details – Signal Processing

What I just showed is often called “convolution”. *Actually* cross-correlation. Source of ***<u>terrible</u>*** confusion.

Cross-Correlation
(Original Orientation)

Convolution
(Flip filter in x,y first)

Convolution

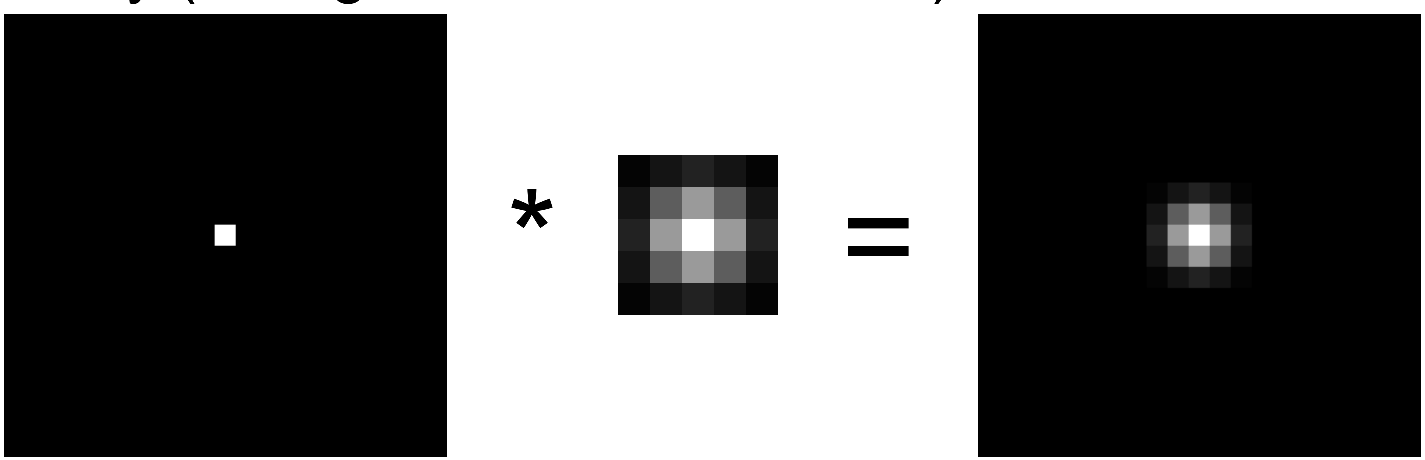
To be more clear:

```
def imageFilter(image, filter):
    for y in range(…): for x in range(…)
    # you'll fill this in
    return filtered

def imageConvolve2D(image, filter):
    # flip the filter left/right and up/down
    filterUse = np.fliplr(np.flipud(filter))
    return imageFilter2D(image, filterUse)
```

Properties of Convolution

- Any shift-invariant, linear operation is a convolution ($*$)
- Commutative: $f * g = g * f$
- Associative: $(f * g) * h = f * (g * h)$
- Distributes over +: $f * (g + h) = f * g + f * h$
- Scalars factor out: $kf * g = f * kg = k (f * g)$
- Identity (a single one with all zeros):



Property List: K. Grauman

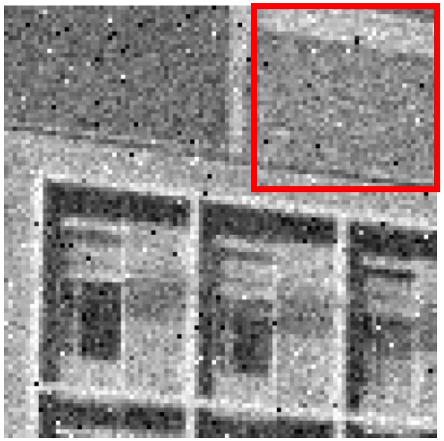
Questions?

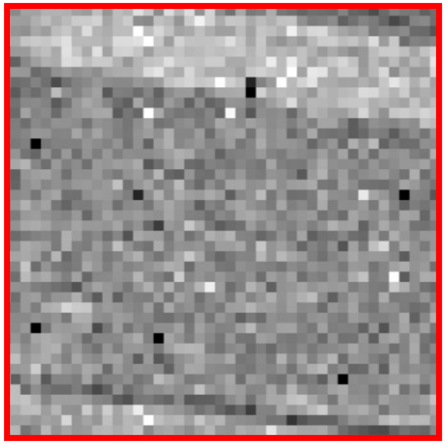
- Nearly everything onwards is a convolution.
- This is important to get right.

Smoothing With A Box

Intuition: if filter touches it, it gets a contribution.

Input

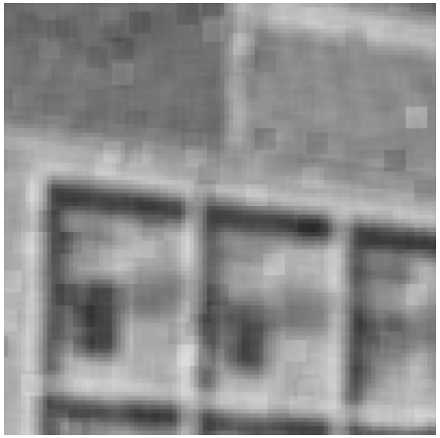


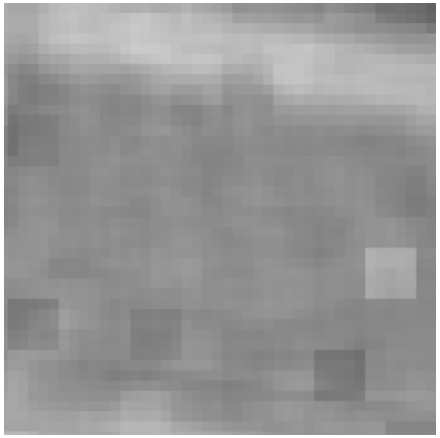


Filter

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Output

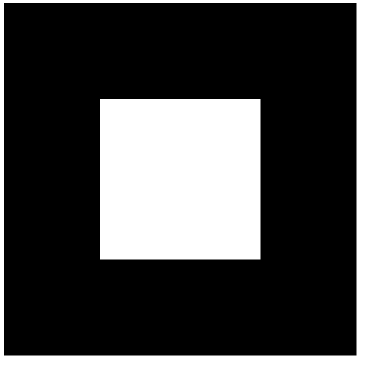




Solution – Weighted Combination

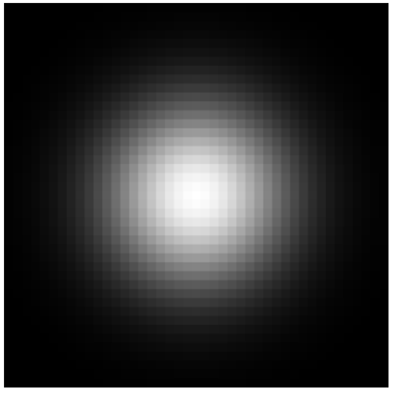
Intuition: weight according to closeness to center. Define 0,0 to be center pixel, then:

$$Filter_{ij} \propto 1$$



What's this?

$$Filter_{ij} \propto \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

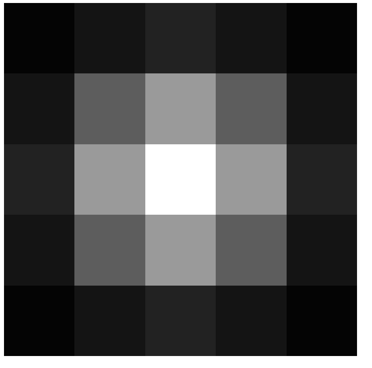


Recognize the Filter?

It's a Gaussian!

$$Filter_{ij} \propto \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

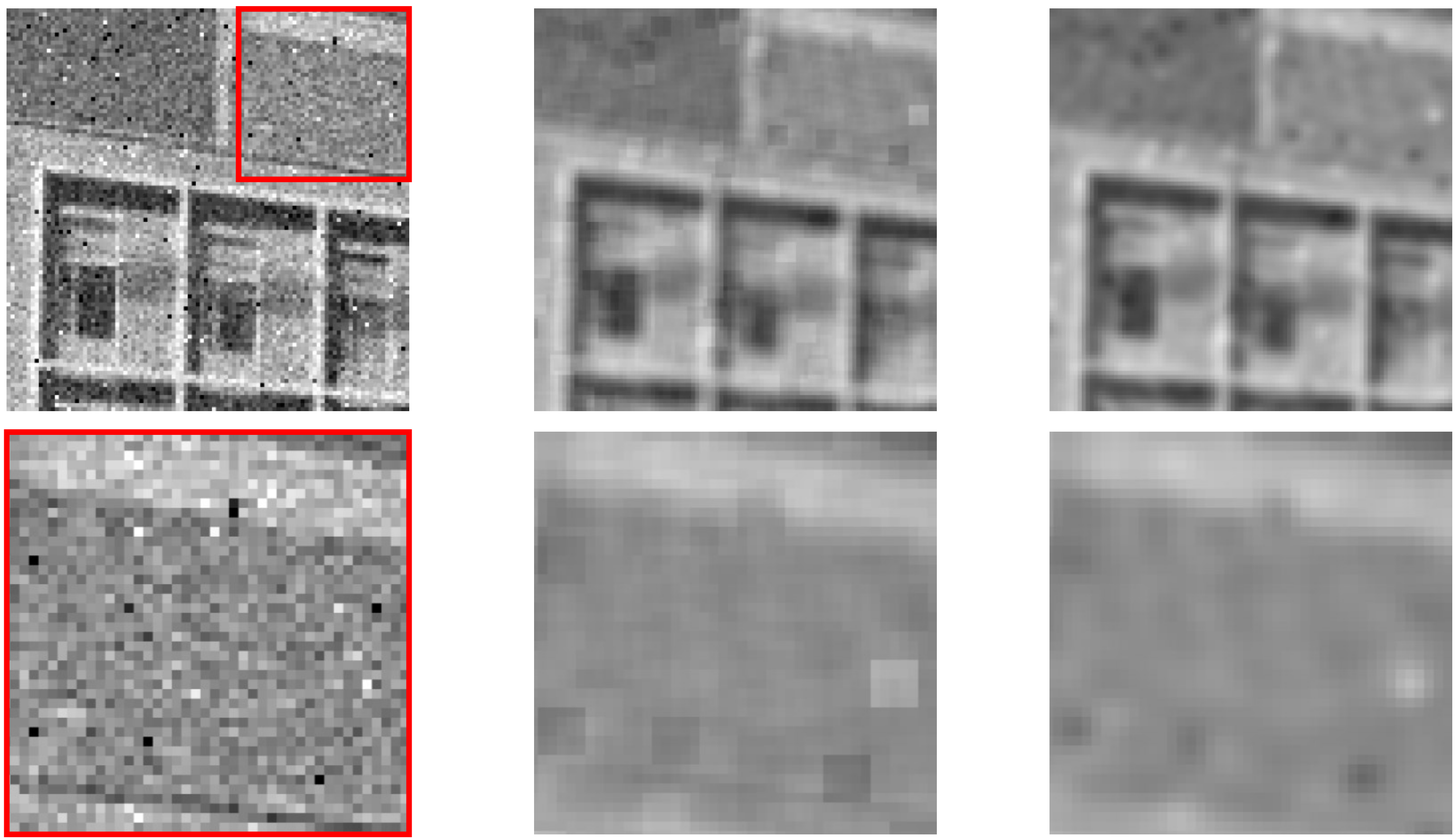
$$\begin{bmatrix} 0.003 & 0.013 & 0.022 & 0.013 & 0.003 \\ 0.013 & 0.060 & 0.098 & 0.060 & 0.013 \\ 0.022 & 0.098 & 0.162 & 0.098 & 0.022 \\ 0.013 & 0.060 & 0.098 & 0.060 & 0.013 \\ 0.003 & 0.013 & 0.022 & 0.013 & 0.003 \end{bmatrix}$$



Smoothing With A Box & Gauss

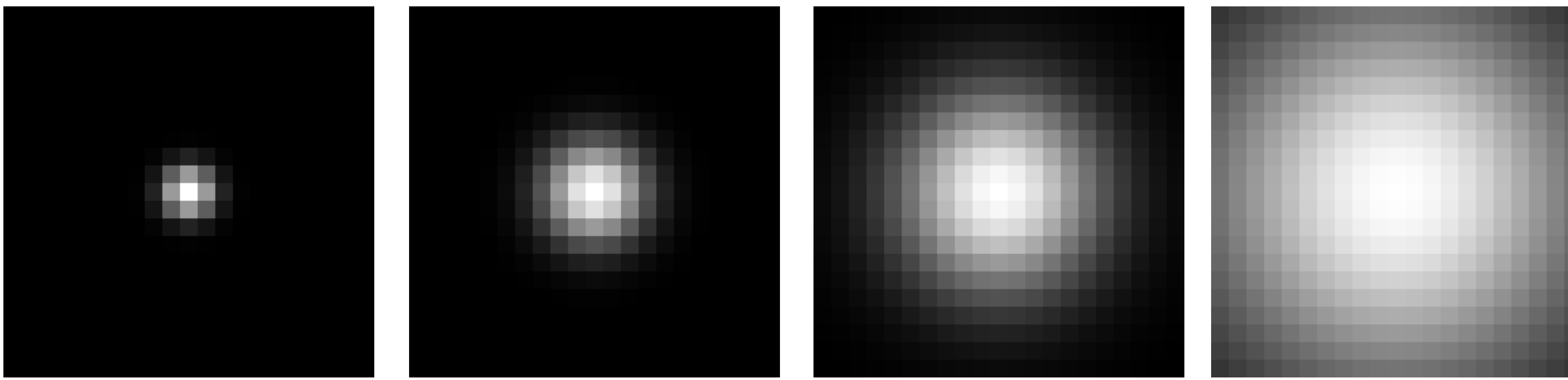
Still have some speckles, but it's not a big box

Input | Box Filter | Gauss. Filter



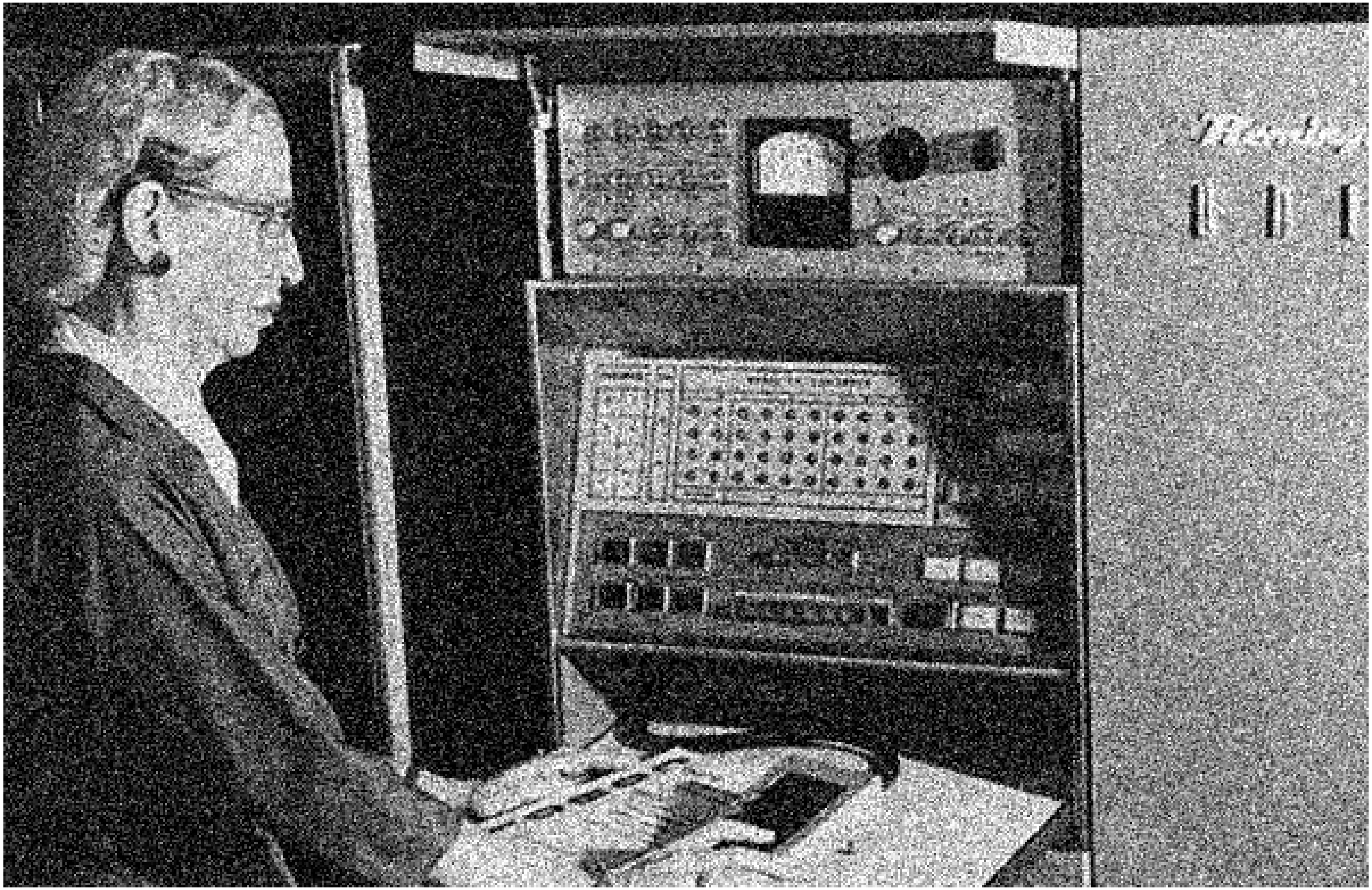
Gaussian Filters

σ = 1, filter = 21x21 | σ = 2, filter = 21x21 | σ = 4, filter = 21x21 | σ = 8, filter = 21x21



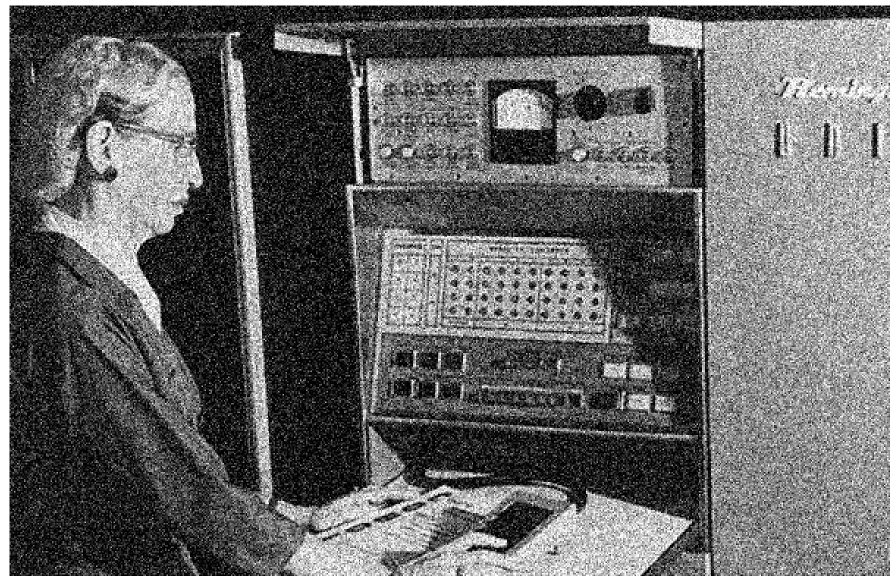
Note: filter visualizations are independently normalized throughout the slides so you can see them better

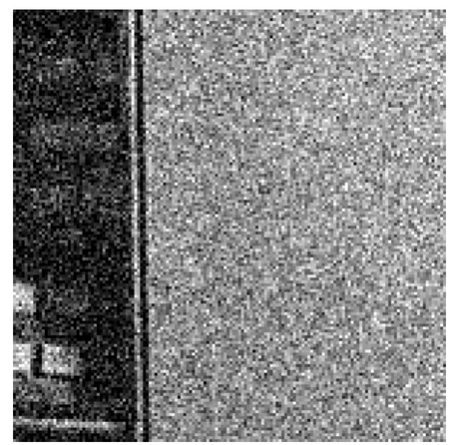
Applying Gaussian Filters



Applying Gaussian Filters

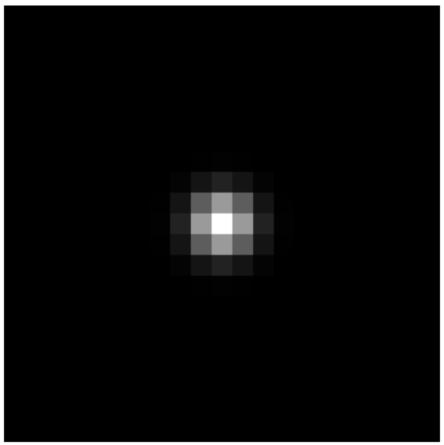
Input Image
(no filter)

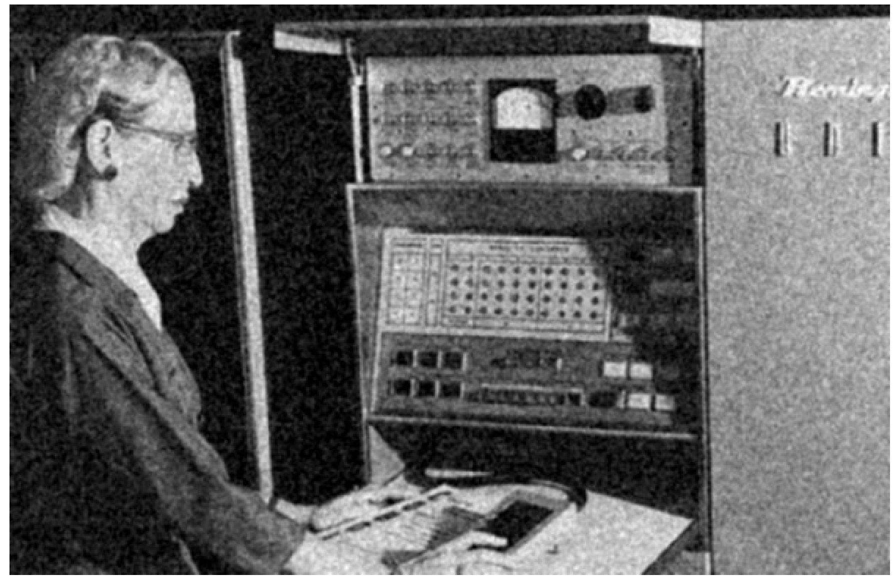


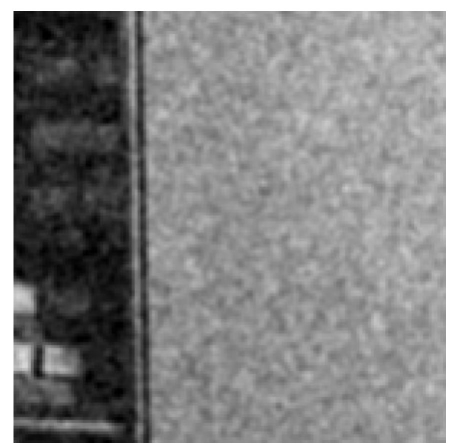


Applying Gaussian Filters

$\sigma = 1$

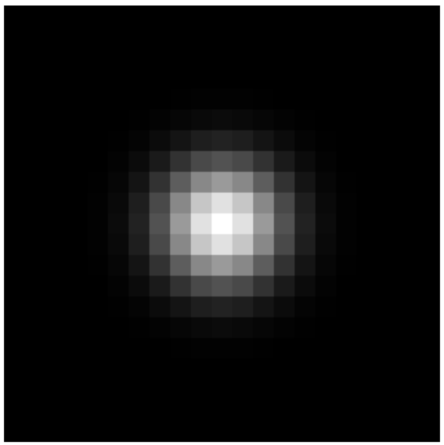


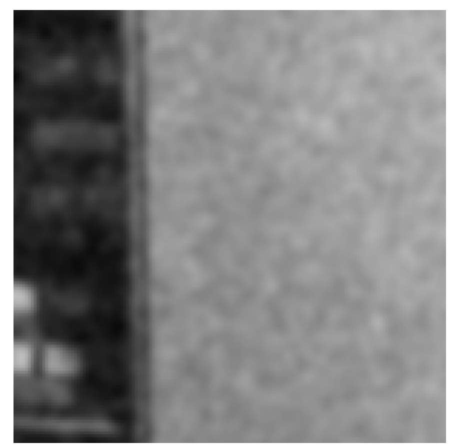




Applying Gaussian Filters

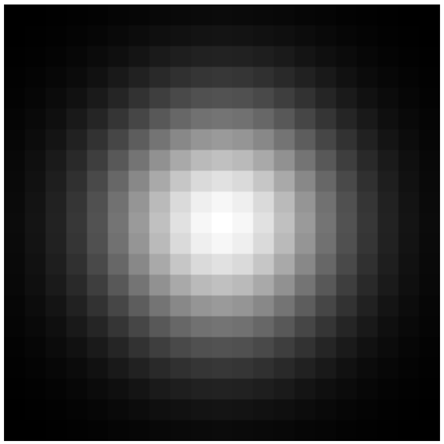
$\sigma = 2$





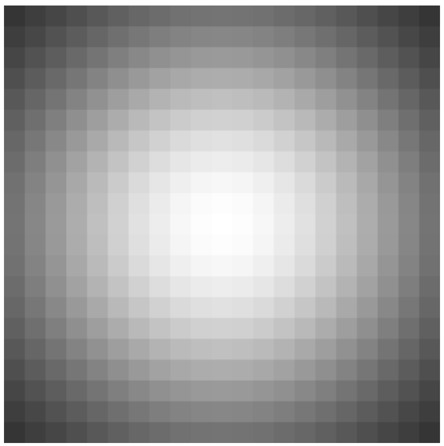
Applying Gaussian Filters

σ = 4



Applying Gaussian Filters

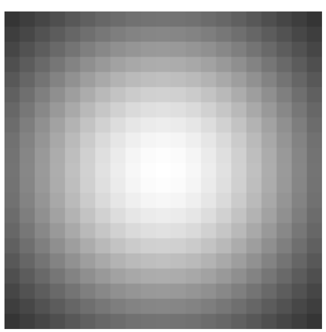
$\sigma = 8$



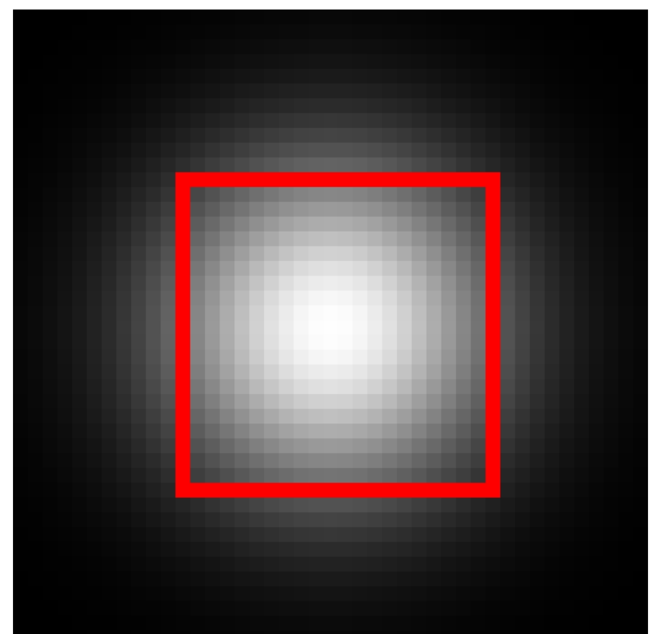
Picking a Filter Size

Too small filter → bad approximation
Want size ≈ 6σ (99.7% of energy)
Left far too small; right slightly too small!

$\sigma = 8$, size = 21



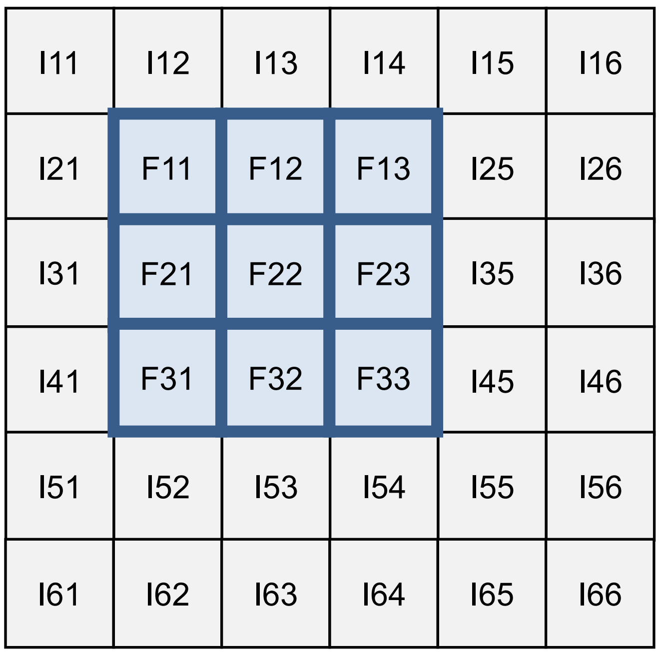
$\sigma = 8$, size = 43



Runtime Complexity

Image size = NxN = 6x6

Filter size = MxM = 3x3




```
for ImageY in range(N):
    for ImageX in range(N):
        for FilterY in range(M):
            for FilterX in range(M):
                ...
```

Time: $O(N^2M^2)$

Separability

Conv(vector, transposed vector) → outer product

$$\begin{bmatrix} Fy1 \\ Fy2 \\ Fy3 \end{bmatrix} * \begin{bmatrix} Fx1 & Fx2 & Fx3 \end{bmatrix} = \begin{bmatrix} Fx1 * Fy1 & Fx2 * Fy1 & Fx3 * Fy1 \\ Fx1 * Fy2 & Fx2 * Fy2 & Fx3 * Fy2 \\ Fx1 * Fy3 & Fx2 * Fy3 & Fx3 * Fy3 \end{bmatrix}$$

Separability

$$Filter_{ij} \propto \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

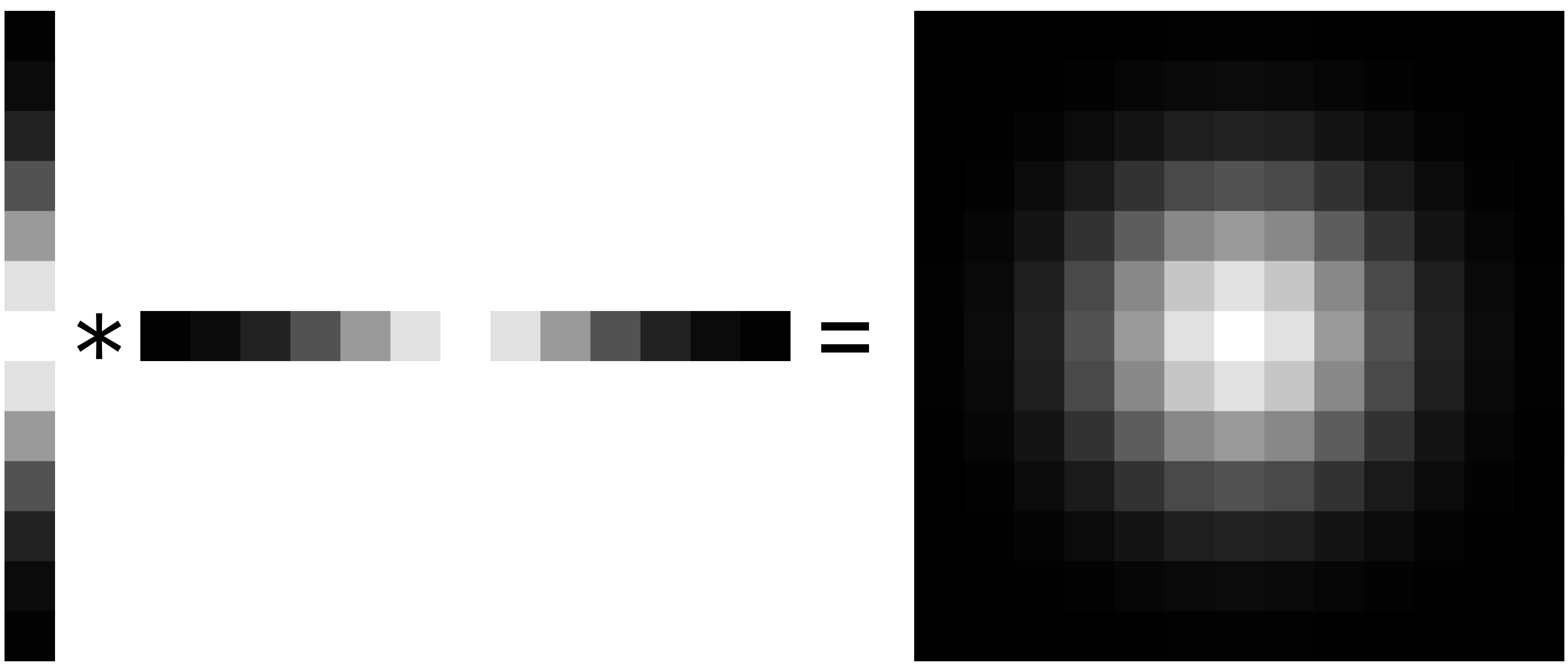
$$\rightarrow$$

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

Separability

1D Gaussian $\ast$ 1D Gaussian = 2D Gaussian

Image $\ast$ 2D Gauss = Image $\ast$ (1D Gauss $\ast$ 1D Gauss)
= (Image $\ast$ 1D Gauss) $\ast$ 1D Gauss



Runtime Complexity

Image size = NxN = 6x6

Filter size = Mx1 = 3x1

I11	I12	I13	I14	I15	I16
I21	F1	I23	I24	I25	I26
I31	F2	I33	I34	I35	I36
I41	F3	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56
I61	I62	I63	I64	I65	I66

What are my compute savings for a 13x13 filter?

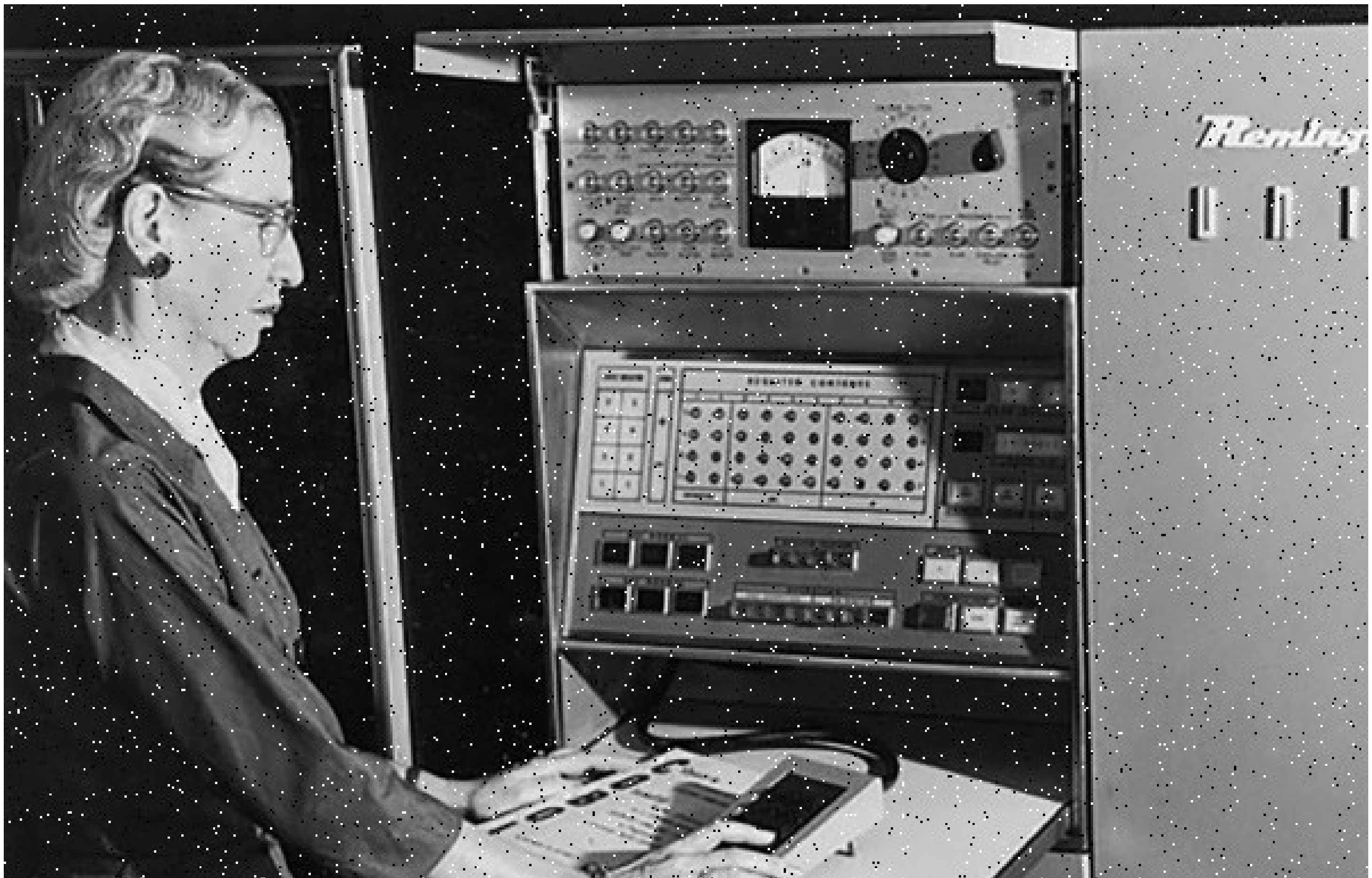
```
for ImageY in range(N):
    for ImageX in range(N):
        for FilterY in range(M):
            …
for ImageY in range(N):
    for ImageX in range(N):
        for FilterX in range(M):
            …
```

Time: $O(N^2M)$

Why Gaussian?

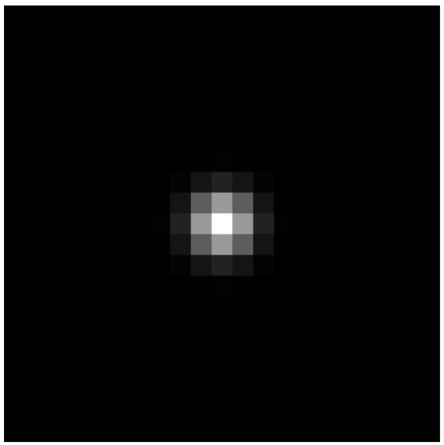
Gaussian filtering removes parts of the signal above a certain frequency. Often noise is high frequency and signal is low frequency.

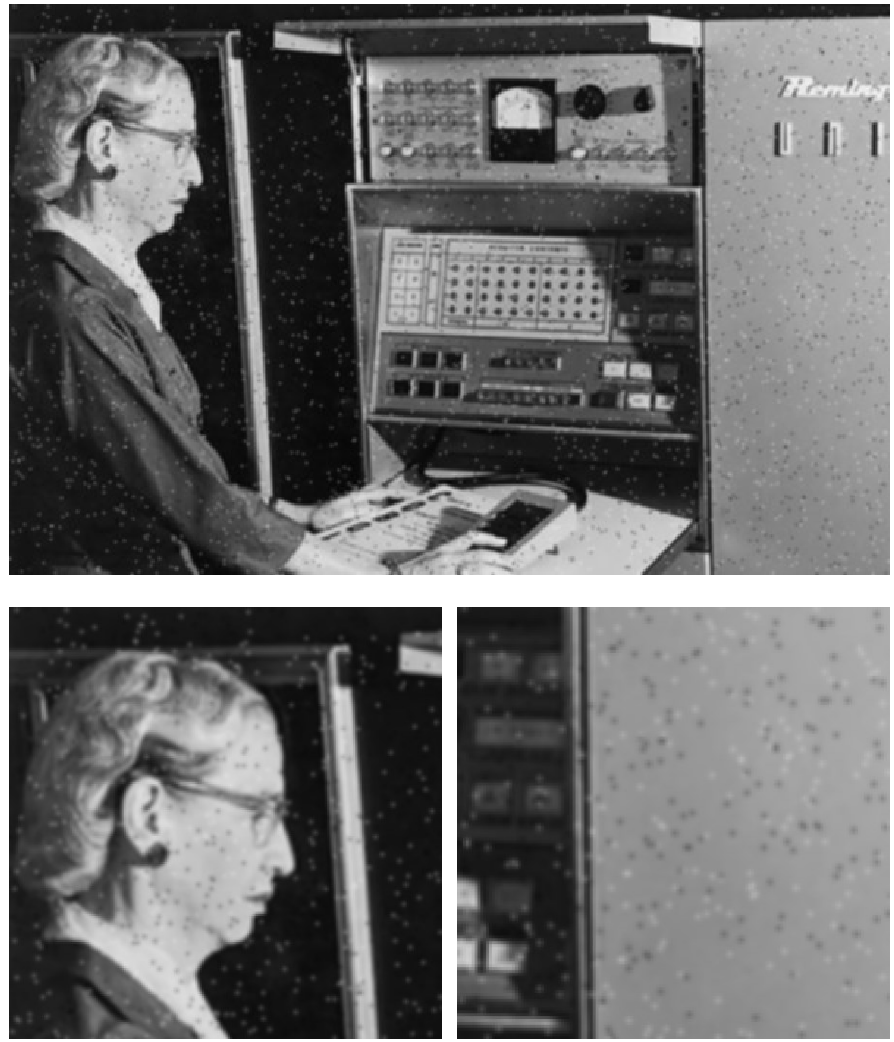
Where Gaussian Fails



Applying Gaussian Filters

$\sigma = 1$





Why Does This Fail?

Means can be arbitrarily distorted by outliers

Signal	10	12	9	8	1000	11	10	12
Filter	0.1	0.8	0.1					
Output	11.5	9.2	107.3	801.9	109.8	10.3		

What else is an “average” other than a mean?

Non-linear Filters (2D)

40	81	13	22
125	830	76	80
144	92	108	95
132	102	106	87

[040, 081, 013, 125, 830, 076, 144, 092, 108]

Sort

[013, 040, 076, 081, 092, 108, 125, 144, 830]

92

[830, 076, 080, 092, 108, 095, 102, 106, 087]

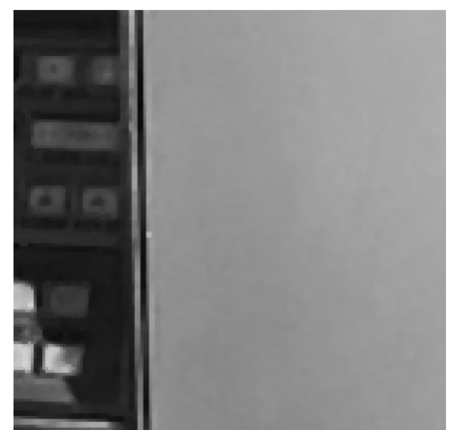
Sort

[076, 080, 087, 092, 095, 102, 106, 108, 830]

95

Applying Median Filter

Median
Filter
(size=3)



Applying Median Filter

Median
Filter
(size = 7)

Is Median Filtering Linear?

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

Median Filter

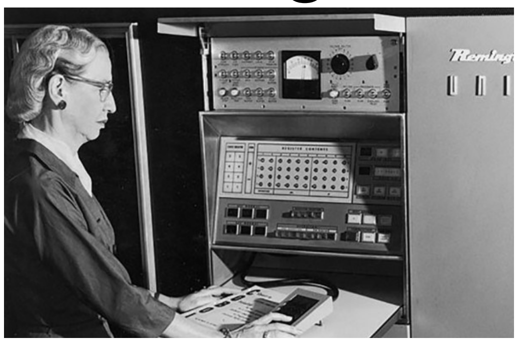
$$1 \quad + \quad 0 \quad = \quad 2$$

Example from (I believe): Kristen Grauman

Some Examples of Filtering

Filtering – Sharpening

Image

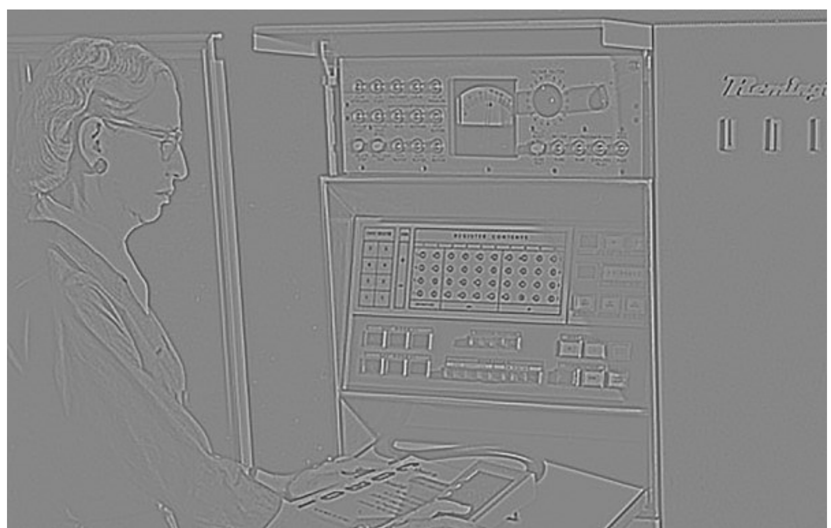


-

Smoothed

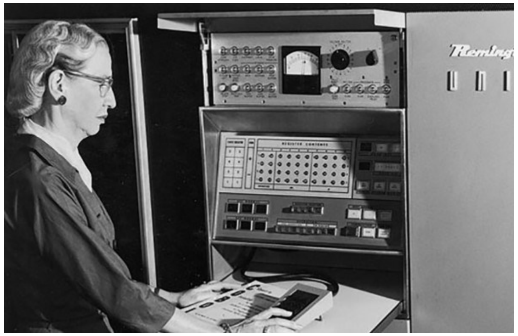
Details

=



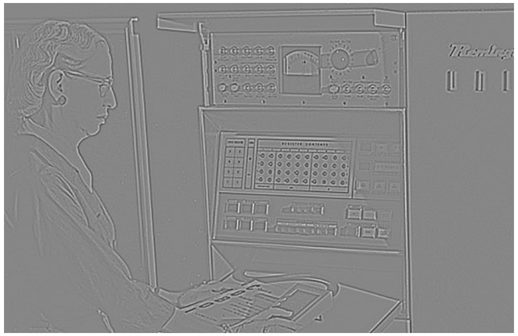
Filtering – Sharpening

Image



+α

Details

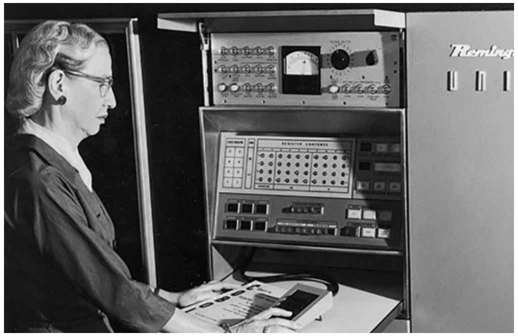


“Sharpened” α=1

=

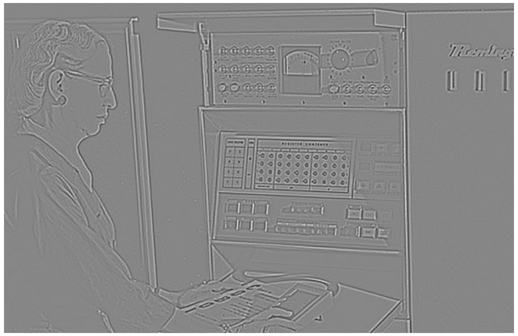
Filtering – Sharpening

Image



+α

Details

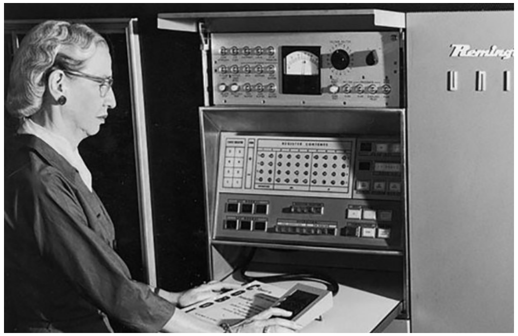


"Sharpened" α=0

=

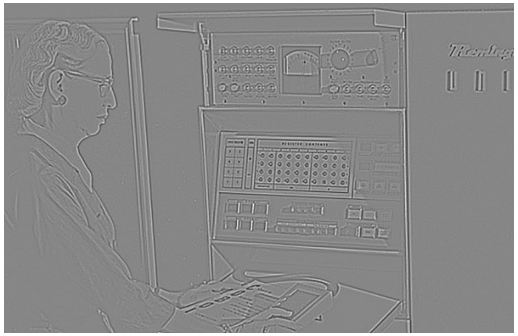
Filtering – Sharpening

Image



+α

Details

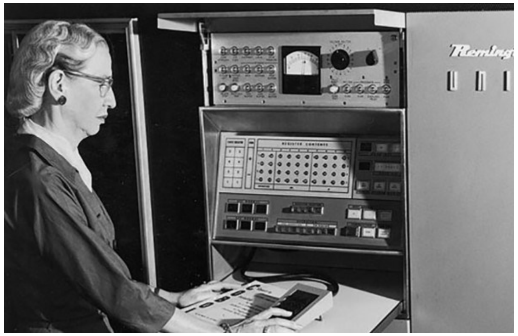


"Sharpened" α=2

=

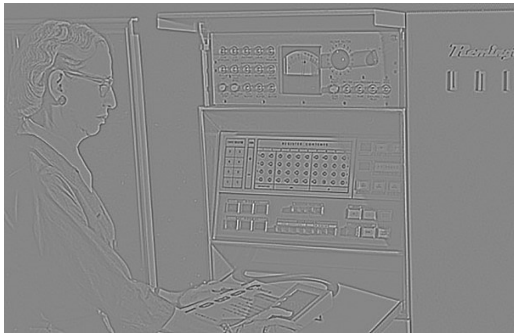
Filtering – Sharpening

Image



+α

Details

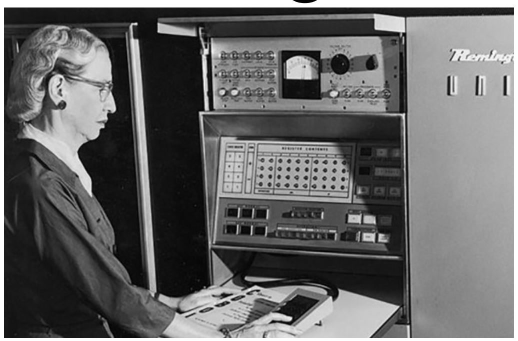


“Sharpened” α=0

=

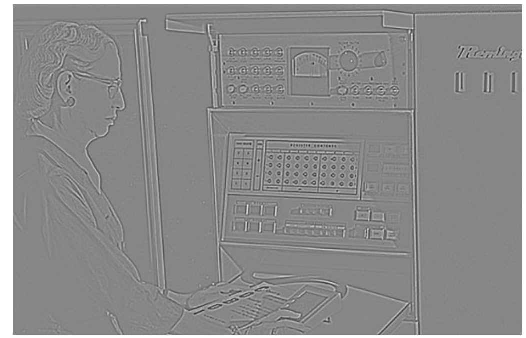
Filtering – Extreme Sharpening

Image



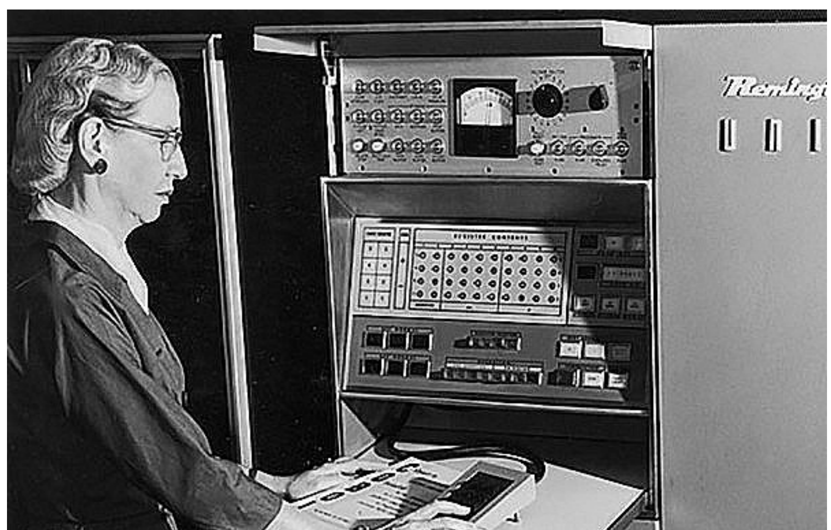
+α

Details



“Sharpened” α=10

=



Filtering

What's this Filter?

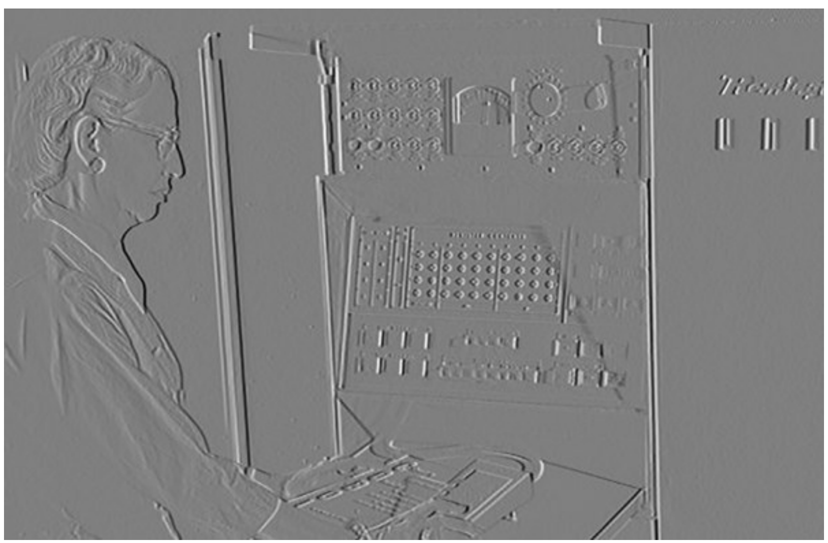
-1	0	1

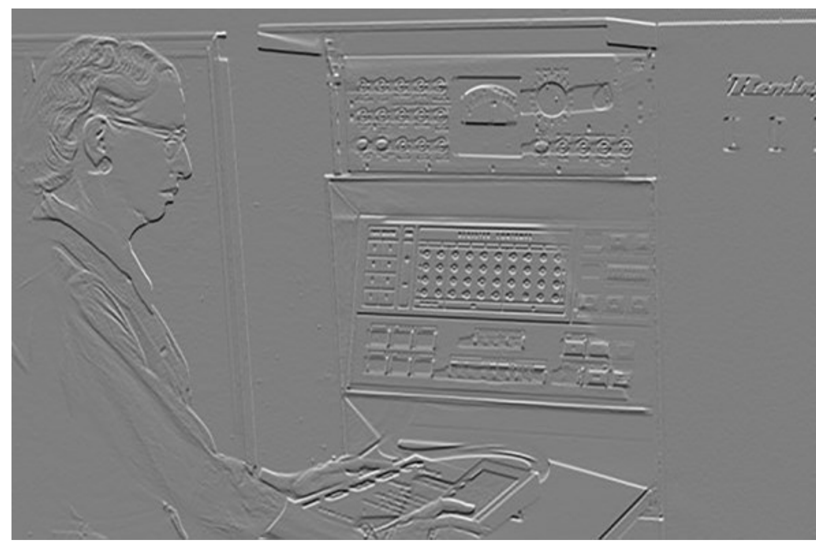
Dx

-1	0	1

T

Dy





Filtering – Derivatives

$$(Dx^2 + Dy^2)^{1/2}$$

Filtering – Bonus

- If you’re curious, you can use filters to accomplish a surprisingly large number of things.

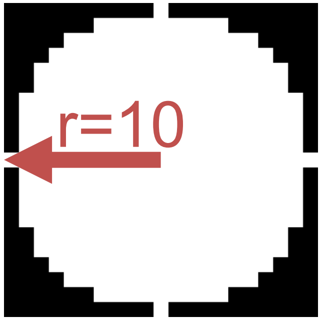
Filtering – Counting

How many “on” pixels have
10+ neighbors within 10 pixels?

Pixels

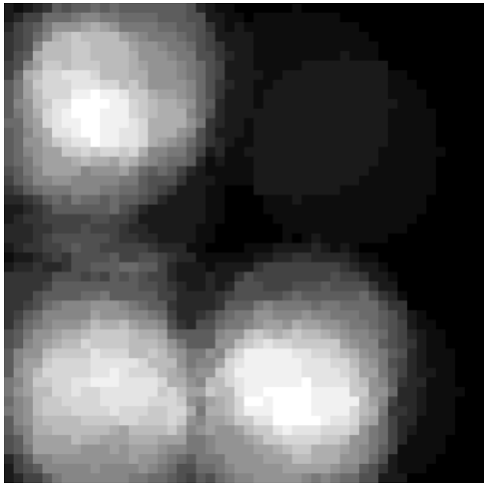
*

Disk



=

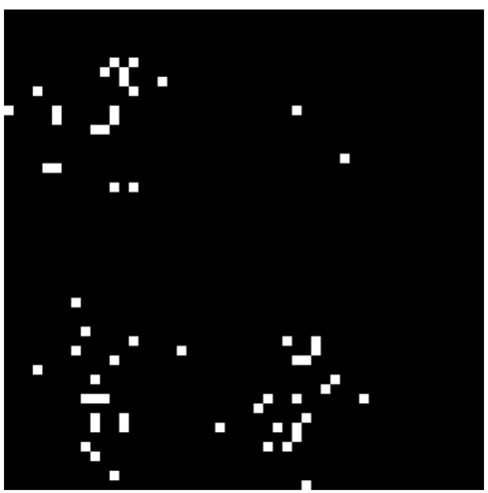
???



Filtering – Counting

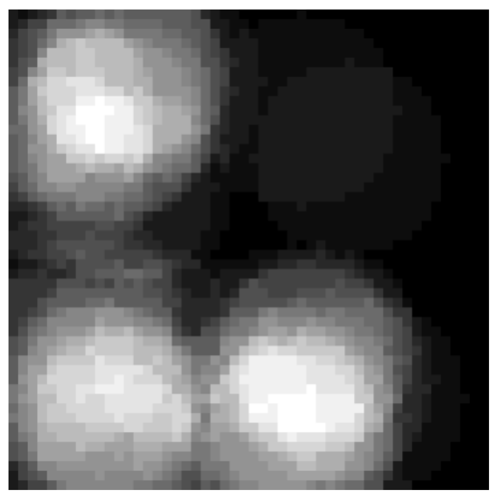
How many “on” pixels have
10+ neighbors within 10 pixels?

Pixels



x

Density

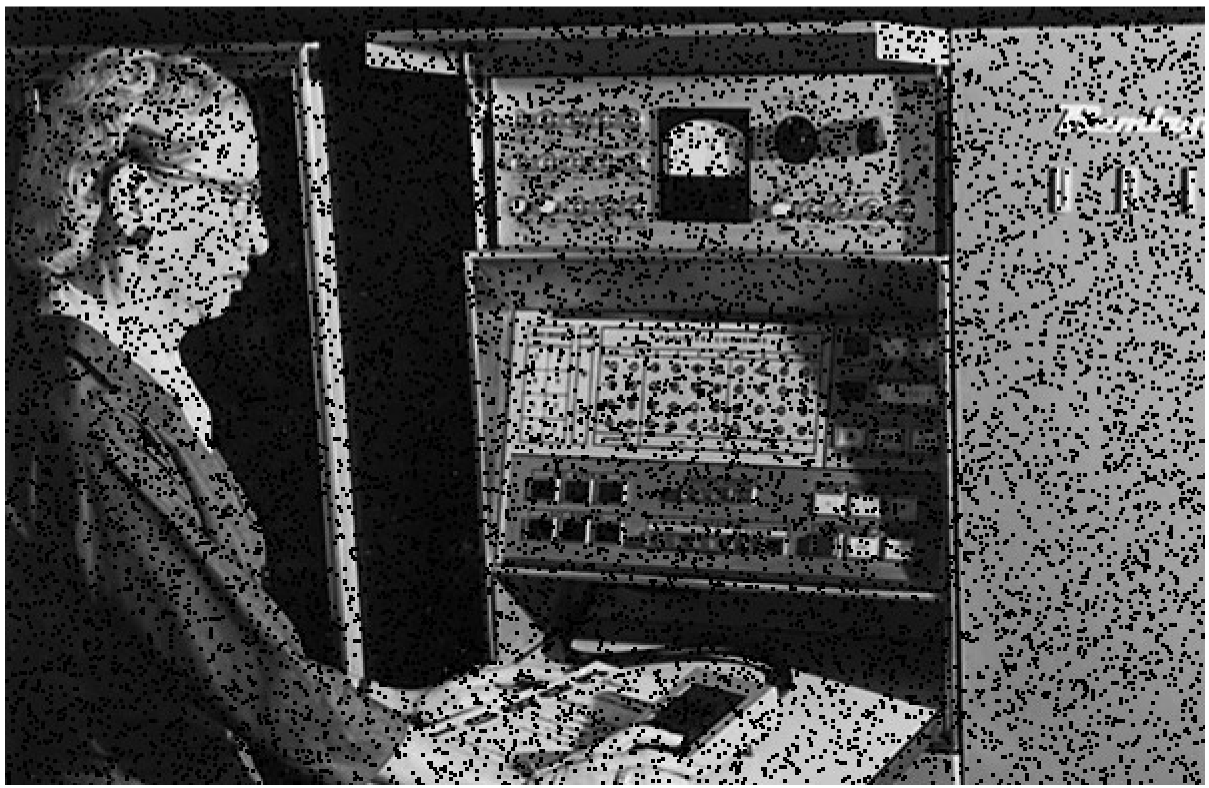


=

Answer

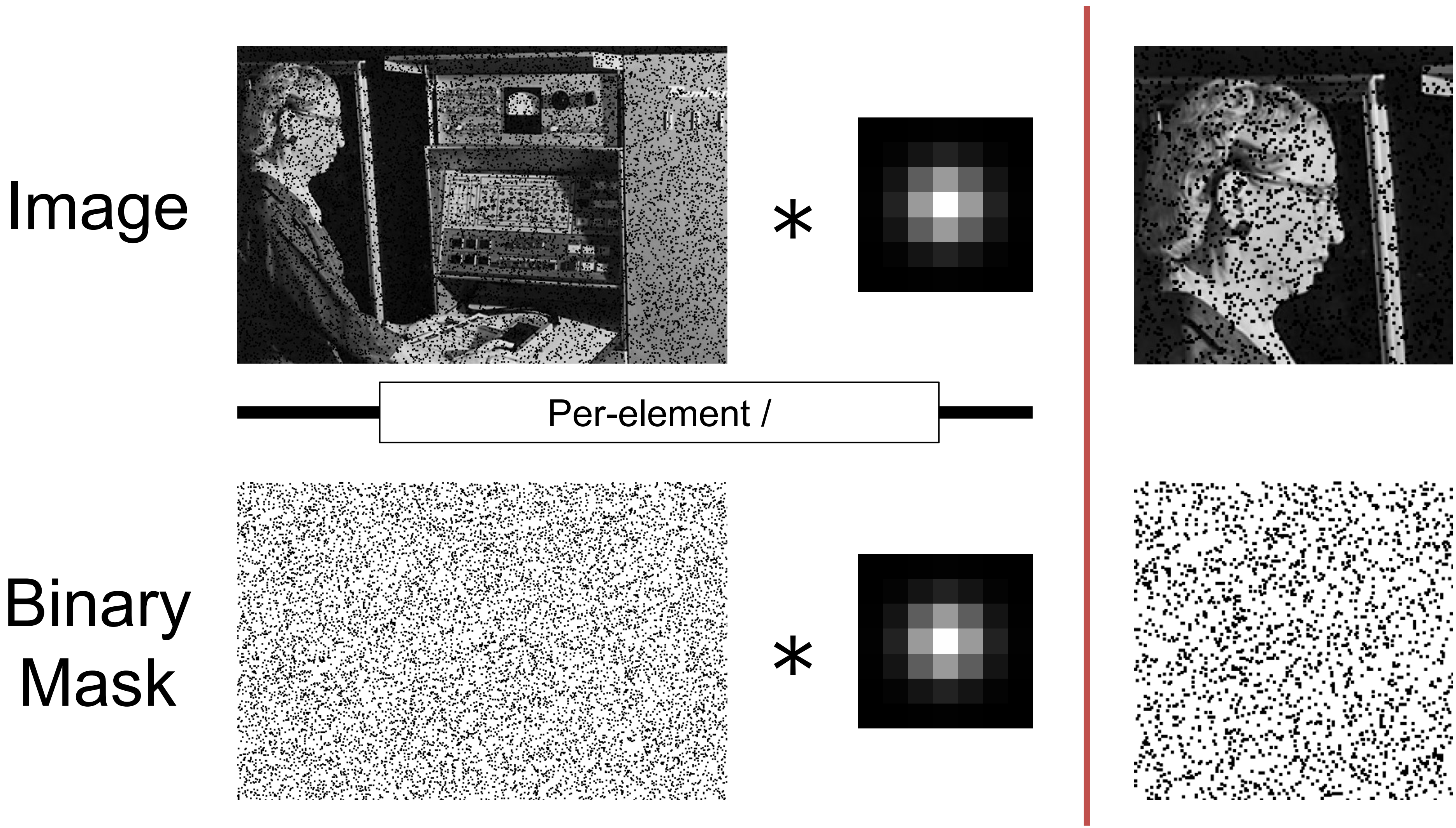
Filtering – Missing Data

Oh no! Missing data!
(and we know where)



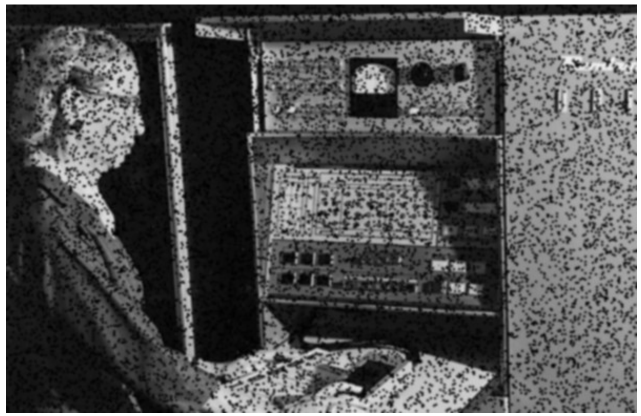
Common with many non-normal cameras (e.g., depth cameras)

Filtering – Missing Data



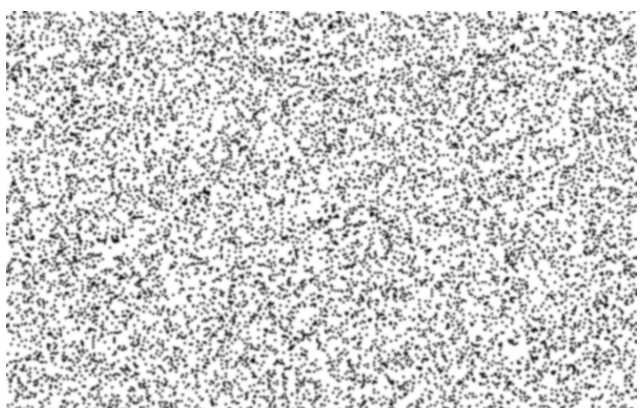
Filtering – Missing Data

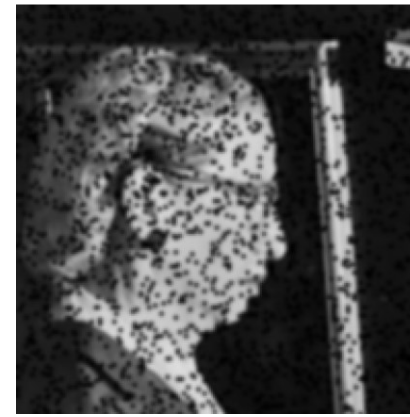
Image

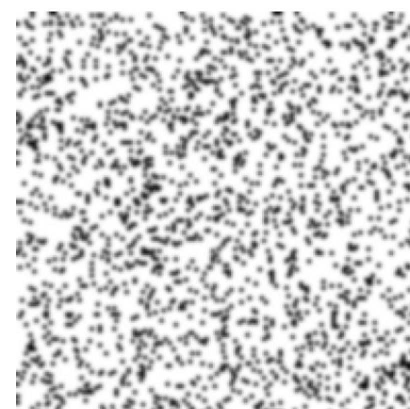


Per-element /

Binary Mask

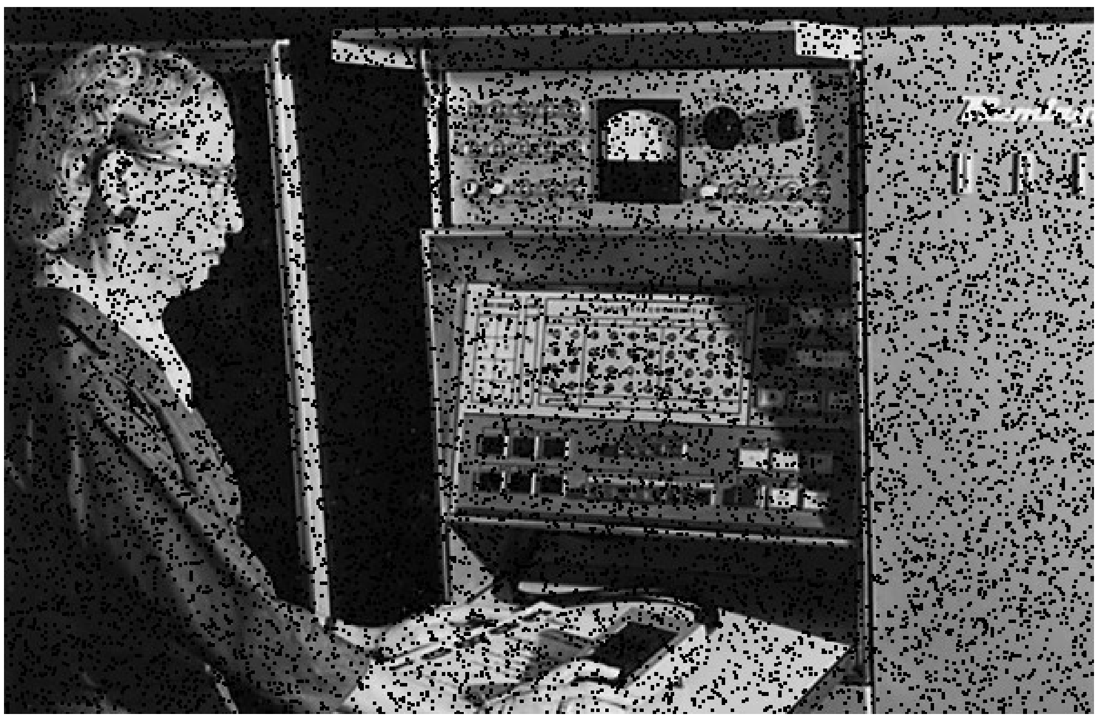






Filtering – Missing Data

Before



Filtering – Missing Data

After

Filtering – Missing Data

After (without missing data)