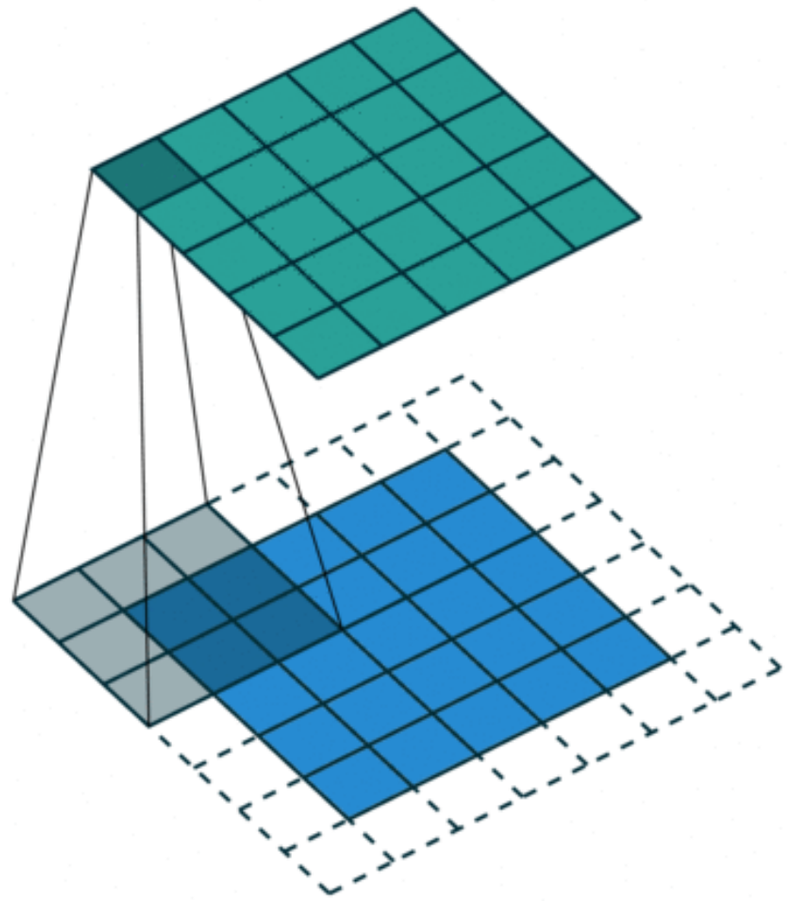
CSCI 497P/597P: Computer Vision



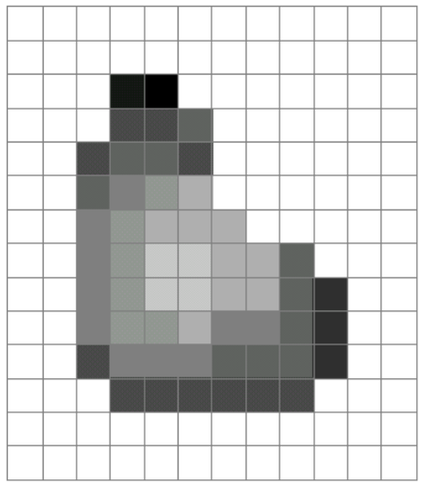
Lecture 2: Image transformations and filtering

Norms

- https://piazza.com/class/k8kojjm2h7n3bg?cid=8

What is an image?

Computationally speaking...



=

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

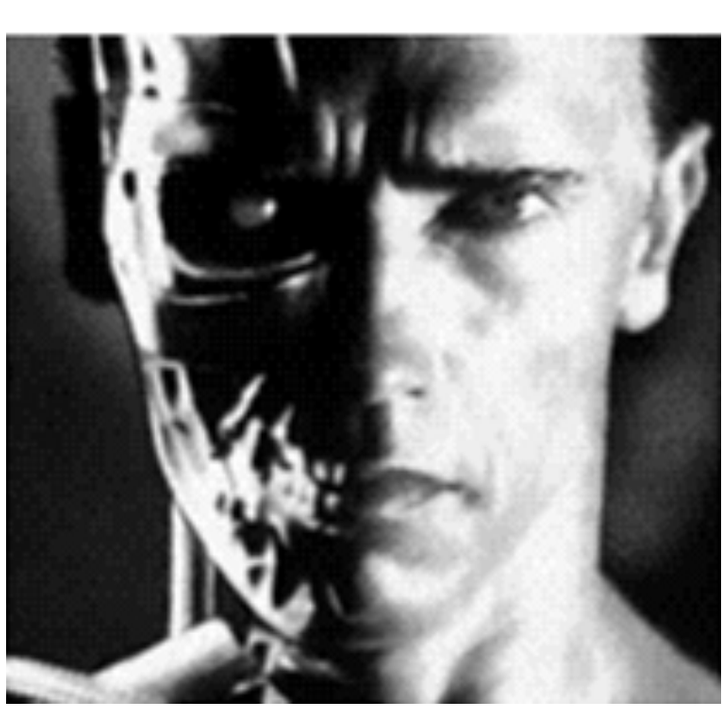
usually one byte per pixel

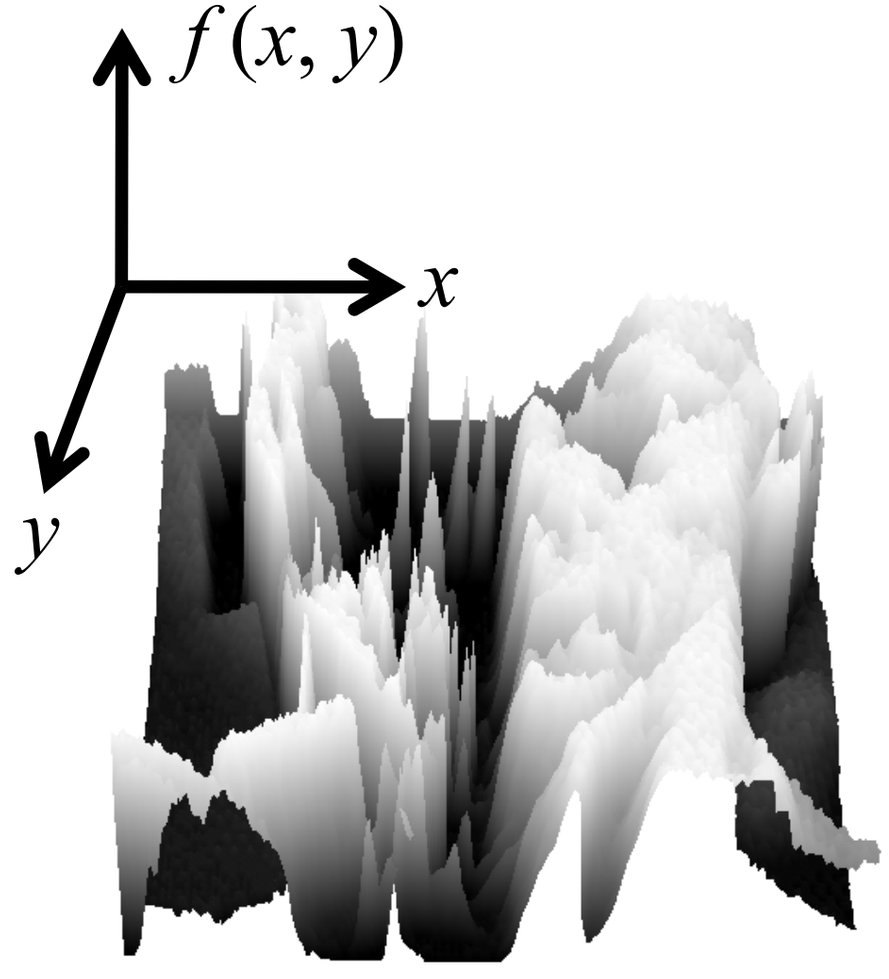
A grayscale (black-and-white) image is a 2D array of numbers.

What is an image?

Mathematically speaking...

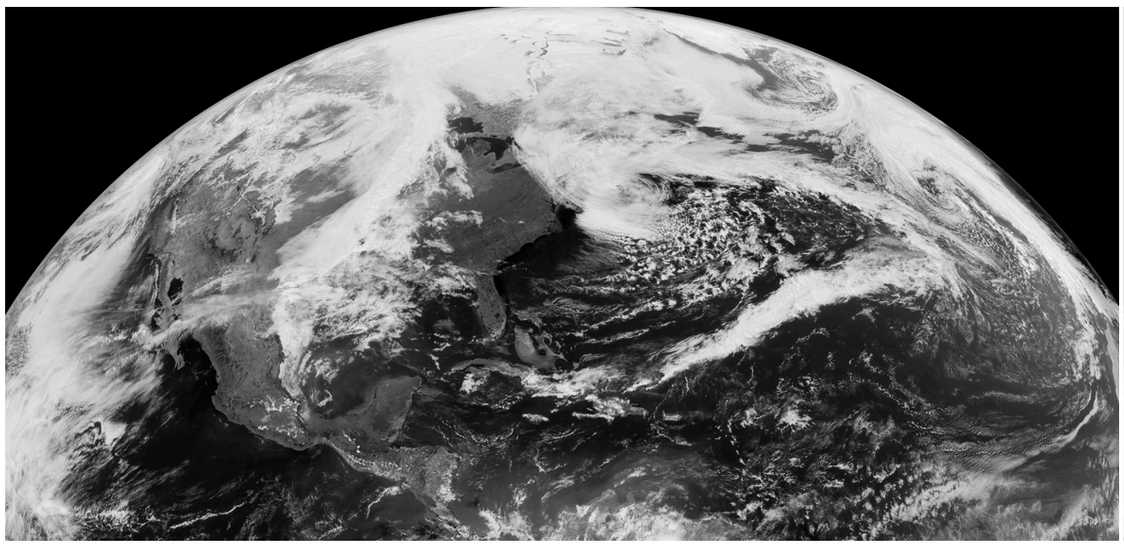
A grayscale (black-and-white) image is a function *f*, from $\boldsymbol{R}^2$ to $\boldsymbol{R}$.



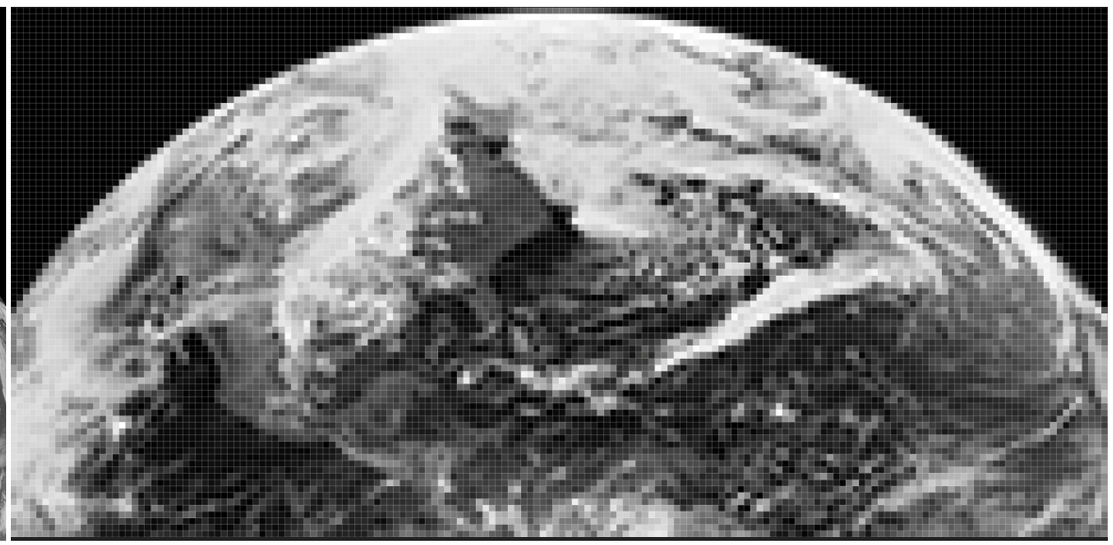


What's the difference?

The computational representation is a **sampled** version of the (ideal) mathematical representation.



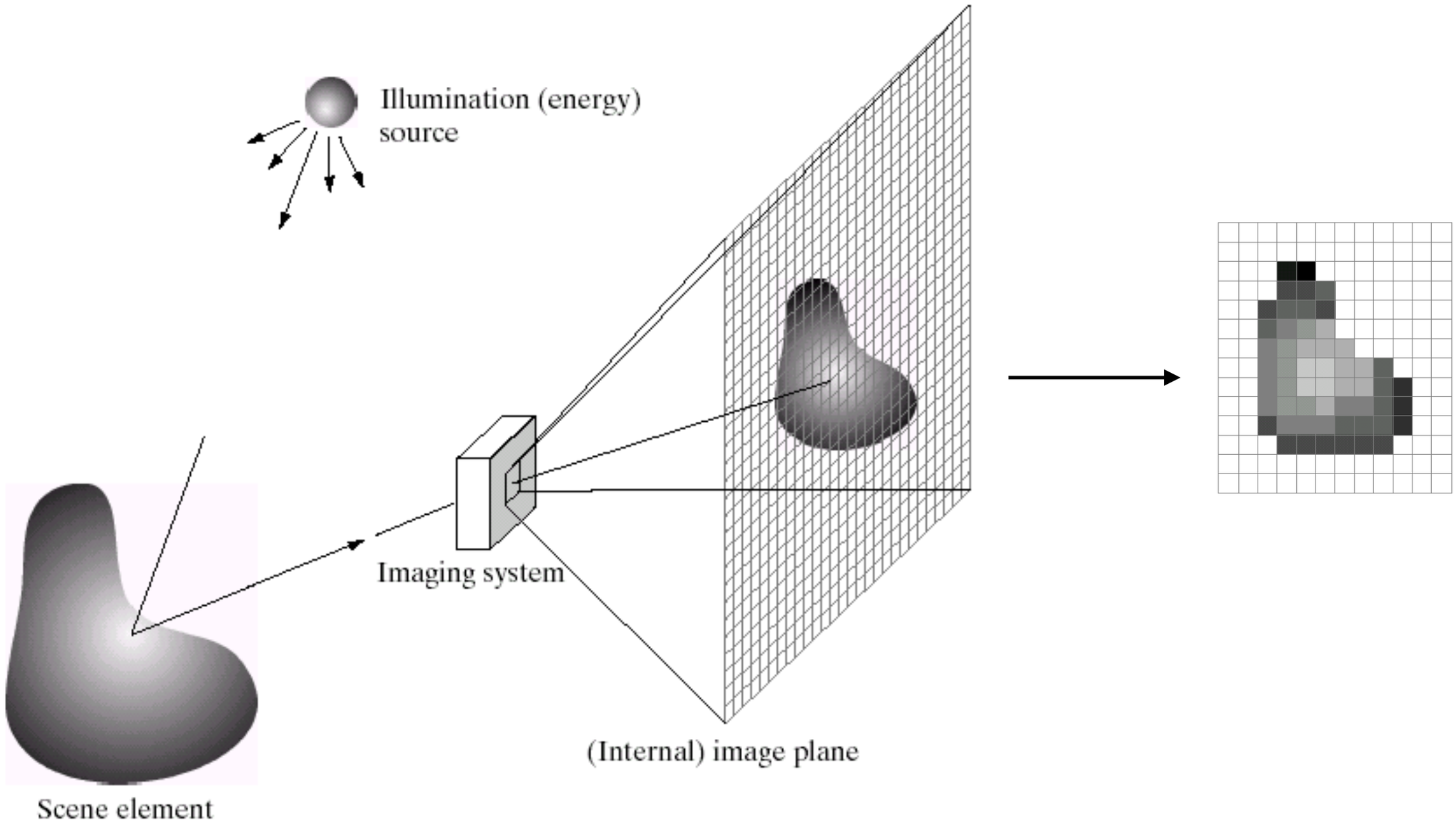
(ideally) continuous



step function

(we can also still write a step function that represents the sampled version)

How do images happen?



more on this process later...

Images are big

Efficiency matters, and not just big-O:

n is large, so even O(n) can be slow

This is a pretty small image.
It's 227 x 336 pixels, grayscale.
That's 95,842 bytes, or 93 kb

A grayscale 1080p frame (1920x1080) is about 2MB.
Want color? 6MB.
Want video? ~30 frames per second; 900MB for a 5s clip

Real images aren't perfect

Real images are not only sampled, but they often have **noise**: unwanted variations in measured intensity value.

f(x, y)

Causes of noise (incomplete list):

- electronic variations in sensor chip
- analog-to-digital quantization
- film grain
- cosmic rays

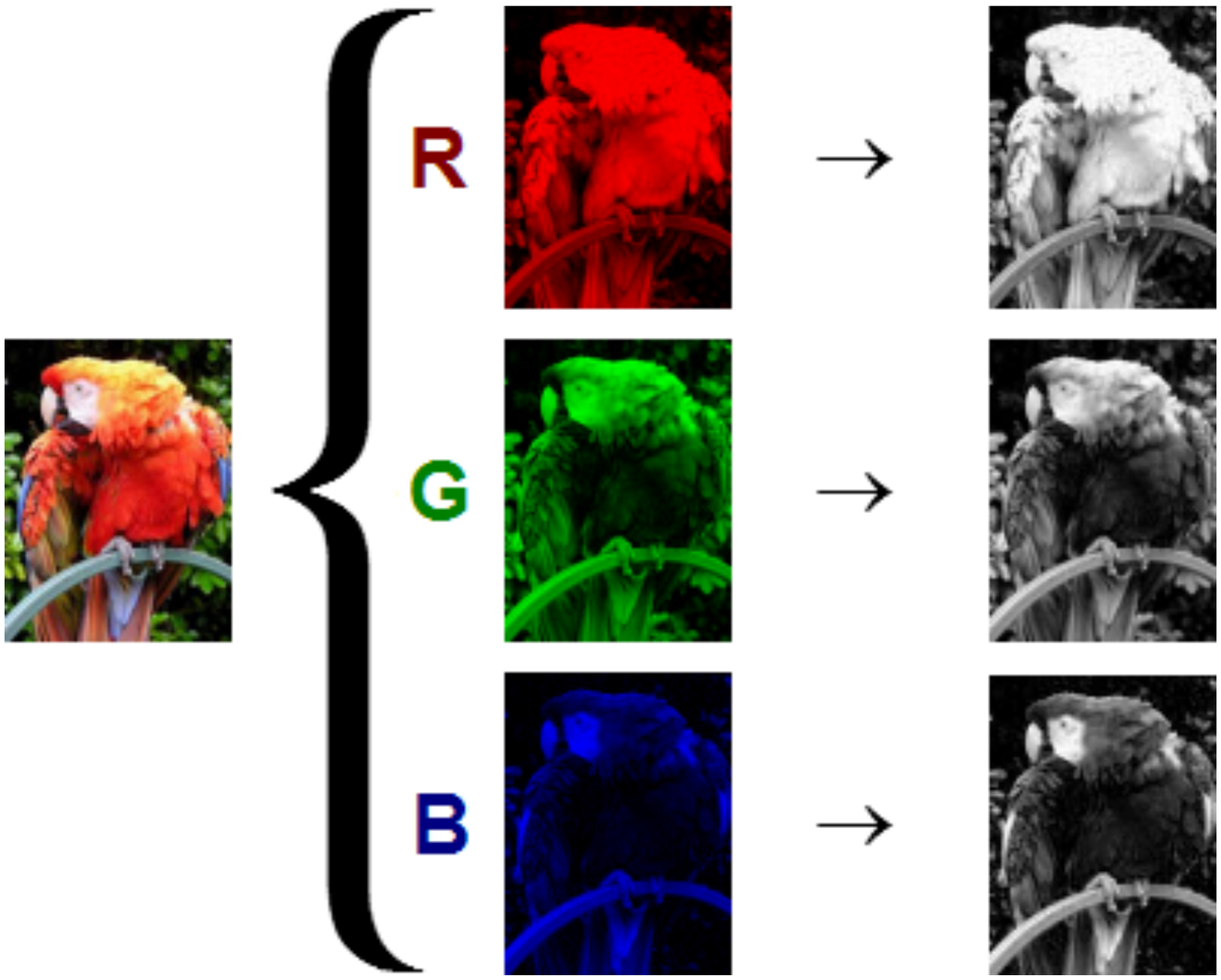
often, we can assume that noise is *random*

(other times, we can't but we do anyway)

What is a color image?

3 grayscale images, each representing one color channel

Often red, green and blue (RGB)



Computationally: HxWx3 array

Mathematically: $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3$

position color

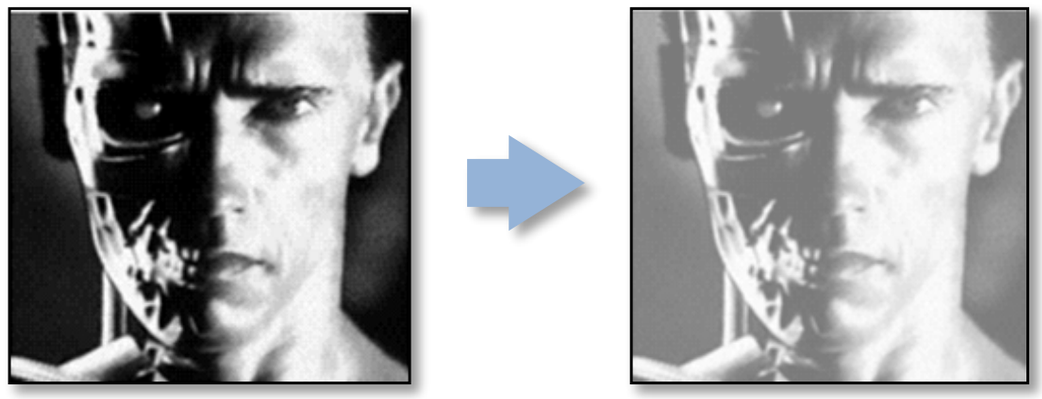
Not always RGB (HSL, Lab, ...)

Not always only 3 channels

Transforming Images

Written as a function, we can *transform* the image function to create altered functions (images):

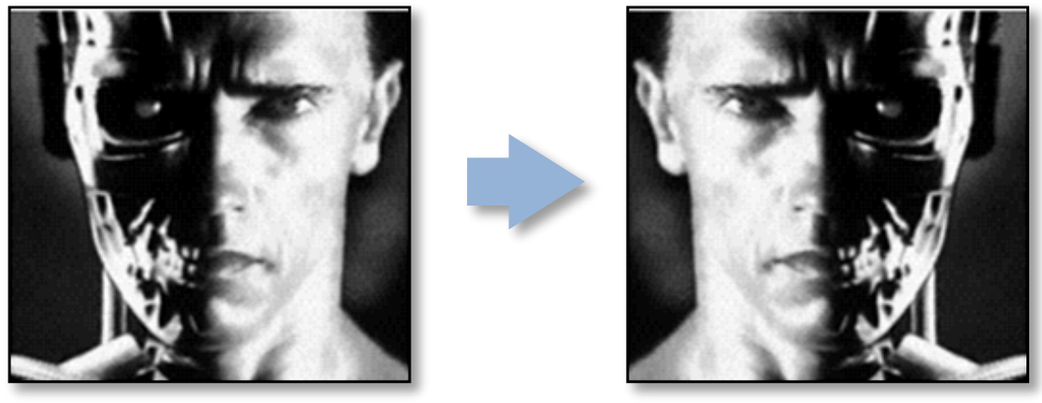
(transforming the *range*)



(increase brightness)

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

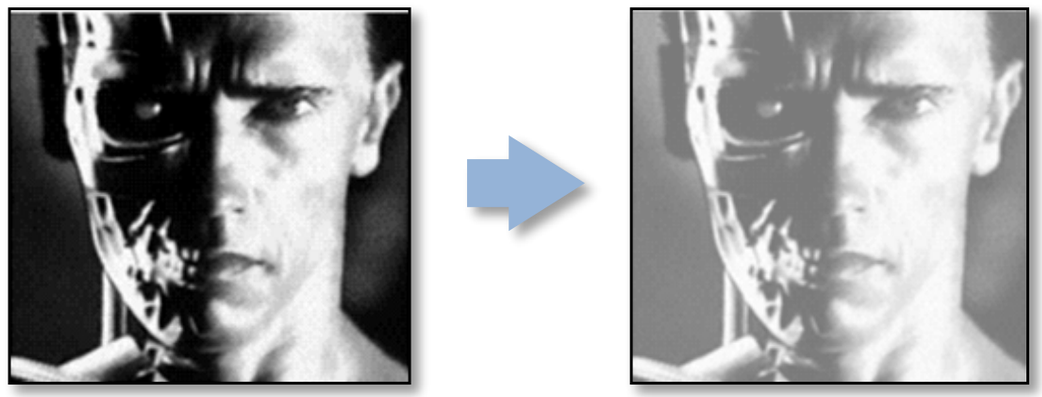
(transforming the *domain*)



(flip horizontally)

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

Transforming Images



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

Here, each output pixel g(x,y) depends only on one input pixel f(x,y)

This doesn't need to be the case.

Example: Denoising

Scenario: you have a camera and this motionless scene.
How can you get a less noisy image?

(let's assume that noise is random)

Example: Denoising

Scenario: you have a camera and this motionless scene.

How can you get a less noisy image?

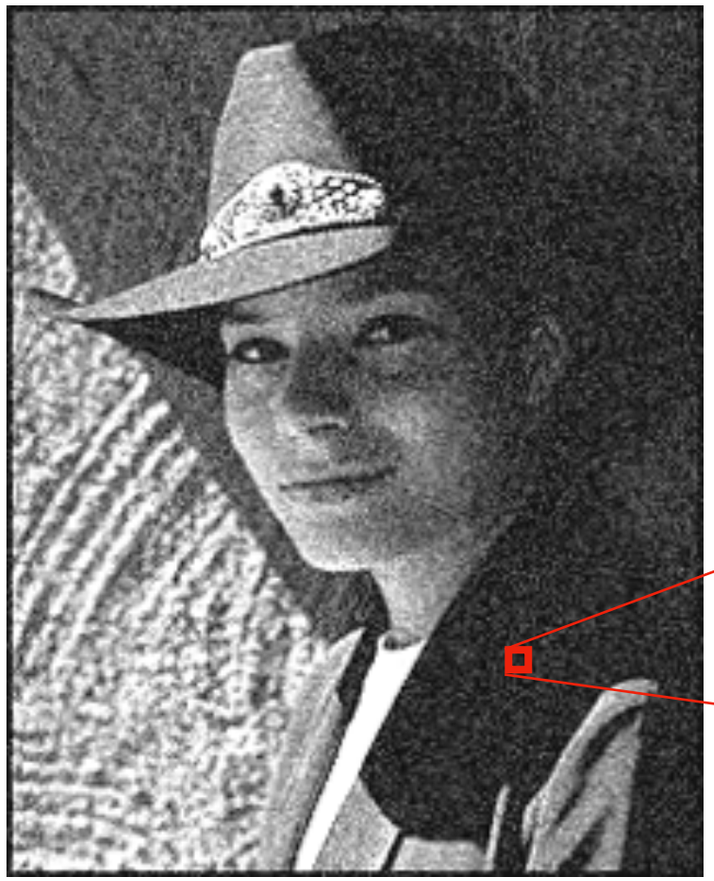
(let's assume that noise is random)

Ideally: average multiple images together

$$g(x,y) = \frac{1}{n}\sum_{i=0}^{n} f_i(x,y)$$

Example: Denoising

Scenario: you're simply given a noisy image. Can you reduce the noise? (still assume that noise is random)



Next best thing, a heuristic:
nearby pixels are often the same (ideal) color,
so average neighboring pixels together.

10	5	3
4	5	1
1	1	7

some function,

e.g., average

	7	

This is an example of **filtering**.
We're taking the mean of the neighborhood, so it's called mean filtering.

Mean filtering: demo

- https://setosa.io/ev/image-kernels/
- At the bottom, enter 0.111 in all 9 boxes (why 0.111?)

Mean filtering: demo

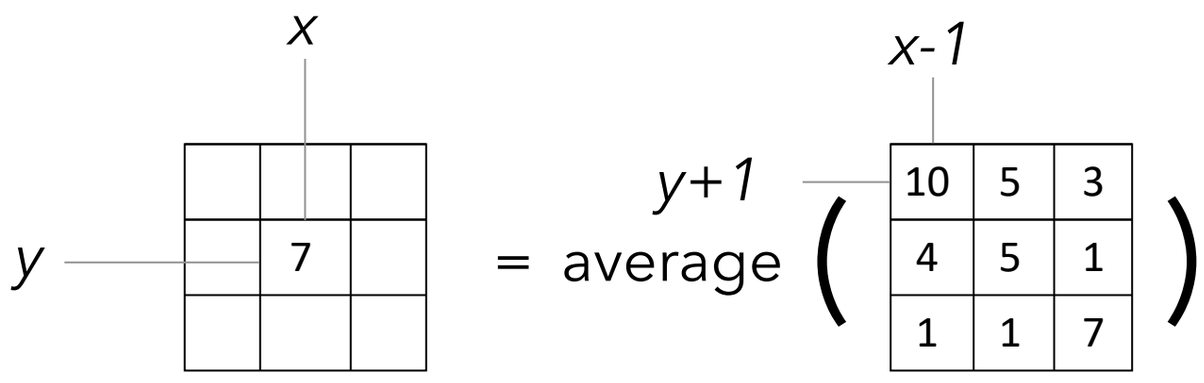
- https://setosa.io/ev/image-kernels/
- At the bottom, enter 0.111 in all 9 boxes (why 0.111?)

Before:

After:

Mean filtering: Mathily

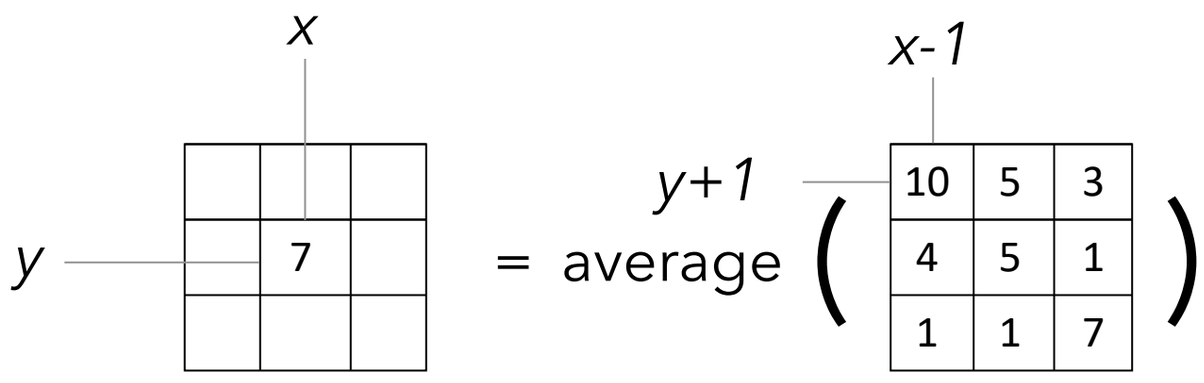
$$g(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} \frac{1}{9} f(x+i, y+j)$$



one output pixel = average (3x3 neighborhood of input pixels)

Let's generalize...

$$g(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} \frac{1}{9} f(x+i, y+j)$$



one output pixel = average (3x3 neighborhood of input pixels)

Let's generalize...

$$g(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} \frac{1}{X} f(x+i, y+j)$$

Socrative poll: What should *X* be?

A: k^2 **B**: $(k+1)^2$ **C**: $(2k-1)^2$ **D**: $(2k+1)^2$

Hint: try an example with with k = 2

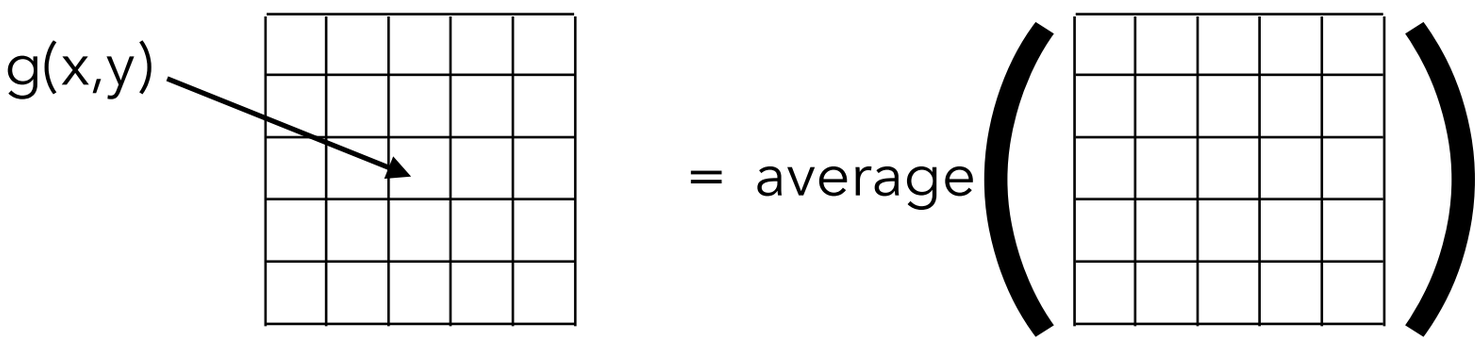
Let's generalize...

$$g(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} \frac{1}{X} f(x+i, y+j)$$

Socrative poll: What should *X* be?

A: k^2 **B**: $(k+1)^2$ **C**: $(2k-1)^2$ **D**: $(2k+1)^2$

Example, with k = 2:



one output pixel = average ((2k+1)x(2k+1) neighborhood of input pixels)

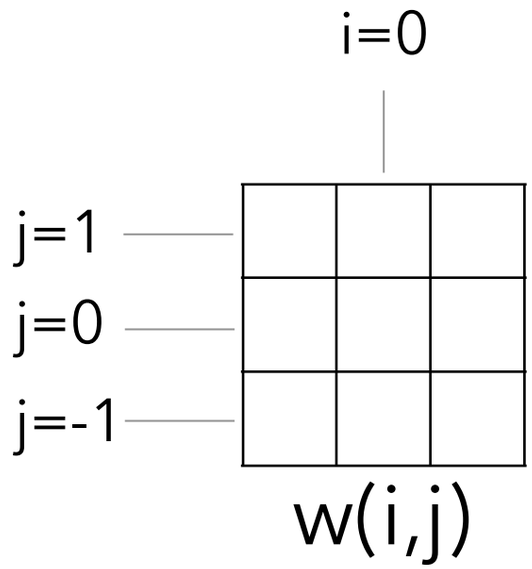
Let's generalize...

$$g(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} \frac{1}{(2k+1)^2} f(x+i, y+j)$$

this makes sure we **average** all values in the neighborhood

Let's generalize to a **weighted average**.

Also store weights in a 2D array (as in the demo): *w(i,j)*



for convenience, (0,0) is at the center

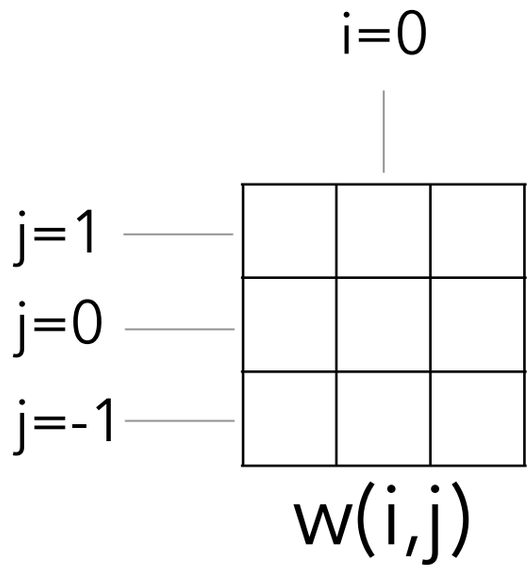
Let's generalize...

$$g(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(x+i, y+j)$$

this makes sure we **average** all values in the neighborhood

Let's generalize to a **weighted average**.

Also store weights in a 2D array (as in the demo): *w(i,j)*



for convenience, (0,0) is at the center

Cross-Correlation

We've just derived the cross-correlation operator.

$$g(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(x+i, y+j)$$

We write this as:

$$g = f \otimes w$$

output image

input image

weights, or filter, or kernel

Cross-Correlation

We've just derived the cross-correlation operator.

$$g(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(x+i, y+j)$$

We write this as:

$$g = f \otimes w$$

output image

input image

weights, or filter, or kernel

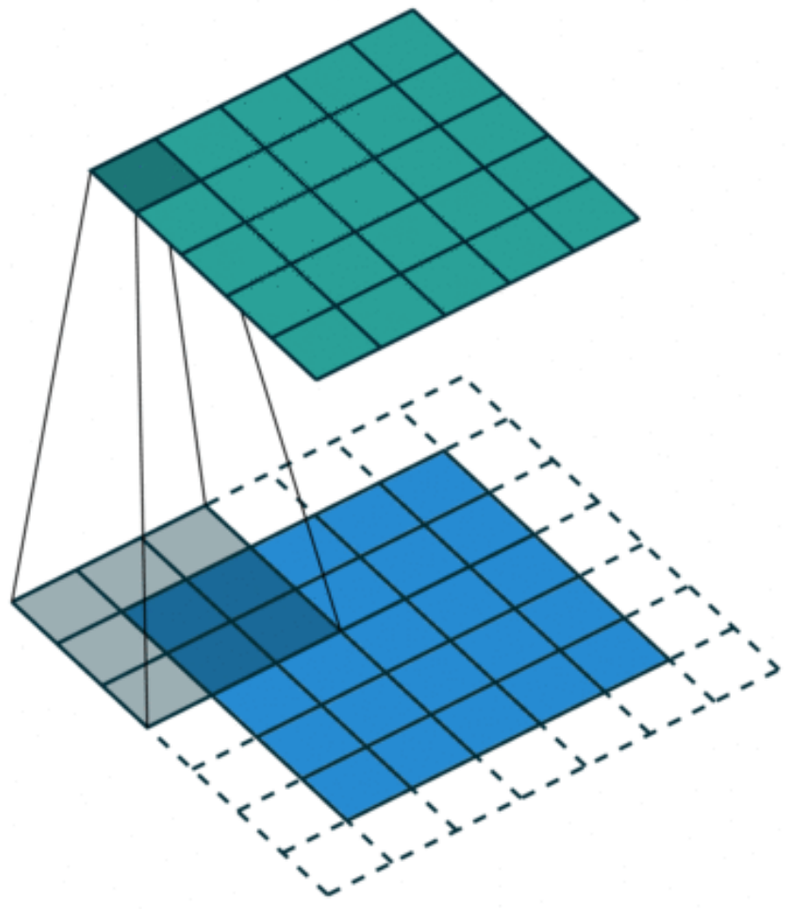
Computing Cross-Correlation

$$g = f \otimes w$$

output image

input image

weights, or filter, or kernel



```
for x = 0 to w:
  for y = 0 to h:
    for i in -k to k:
      for j in -k to k:
        out[x,y] += w[i,j] * in[x+i, y+j]
```

Questions remain

- What happens at the edges?
- What properties does this operator have?
- What can and can't this operator do?