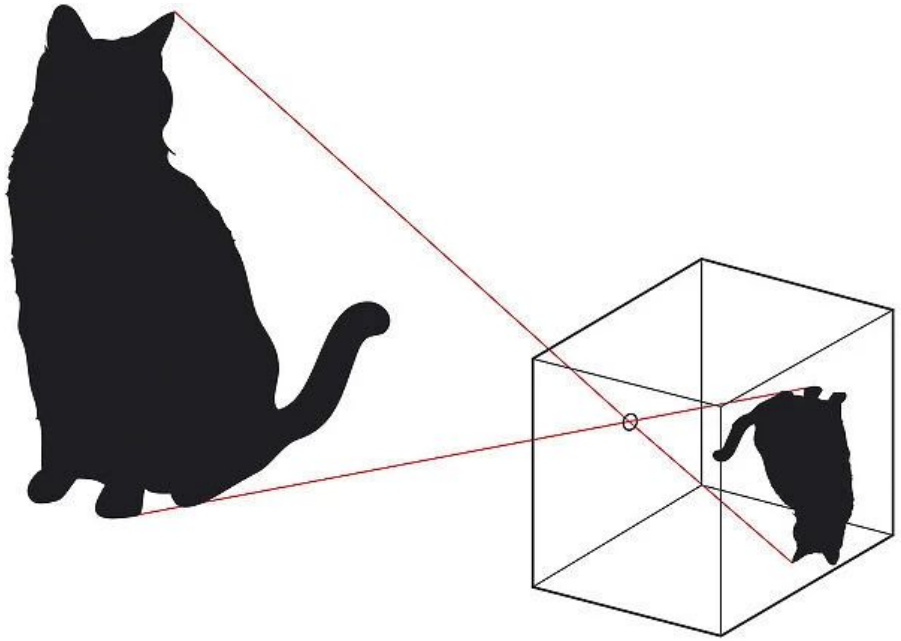
CS-GY 6643
Computer Vision



Lecture 2: Image formation and filtering

Prof. Erdem Varol

Today's menu

Announcements

Image representations (Szeliski 2.2,2.3)

Point operations (Szeliski 3.1)

Intro to image filtering (Szeliski 3.2)

Coding lab

Announcements

Homework 1 out - Due Sep 19 10:59am (1 week)

4 questions (25 + 25 + 25 + 25 = 100 points)

6% of total course grade

Please submit a write up in pdf format on brightspace.

No paper submissions.

You can either write on the assignment sheet and scan

…or provide a separate write up (Latex is preferred).

Must submit individually.

Honor code - OK to work with other students, as long as it's stated.

The whole assignment should take <1.5 hrs.

1 point will be deducted for every hour the assignment is late (rounded down).

CS GY 6643 - Computer Vision, Fall 2024 **Erdem Varol**

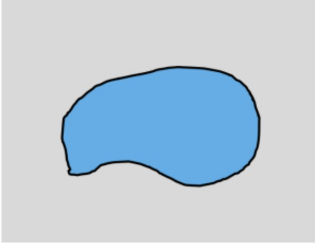
Homework 1

Out: 09/12/2024

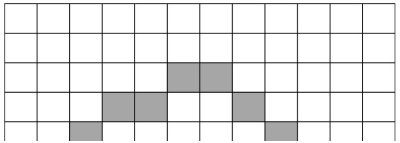
Due: 09/18/2024 11:59 PM

1. Connectivity: Connected component labeling

The famous theorem of Jordan states that every simple closed curve in the plane separates the complement into two connected nonempty sets: an interior region and an exterior. Although this looks simple and intuitive, the proof can get very complex (for those who want to look it up: Link). Elements are a closed curve and the complement, which is the plane, which is divided into 2 regions. Now let us explore how the continuous case transfers to our discrete domain.



Given the following pixel image with white areas and a gray closed pixel curve. How many connected components can you detect?



Project 1 will be out tomorrow

Due October 4, 10:59am (3 weeks)

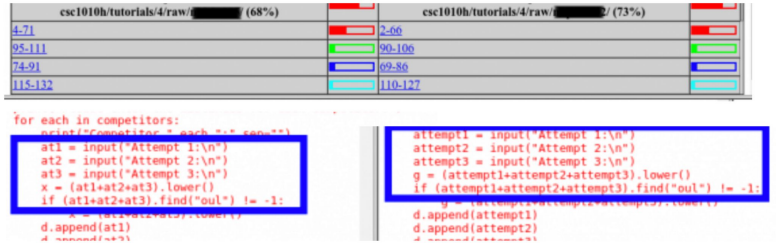
15 % of total course grade.

A python notebook with executed outputs + a pdf write up of each step explained to be submitted on brightspace.

You are encouraged to discuss with other students and look online for examples.

Project must be submitted individually (**We will check for blatant code copying**).

1 point will be deducted for every hour the project is late (rounded down).



Final project team formation as a homework

Due September 22, 10:59am (10 days)

Worth 2% of total course grade

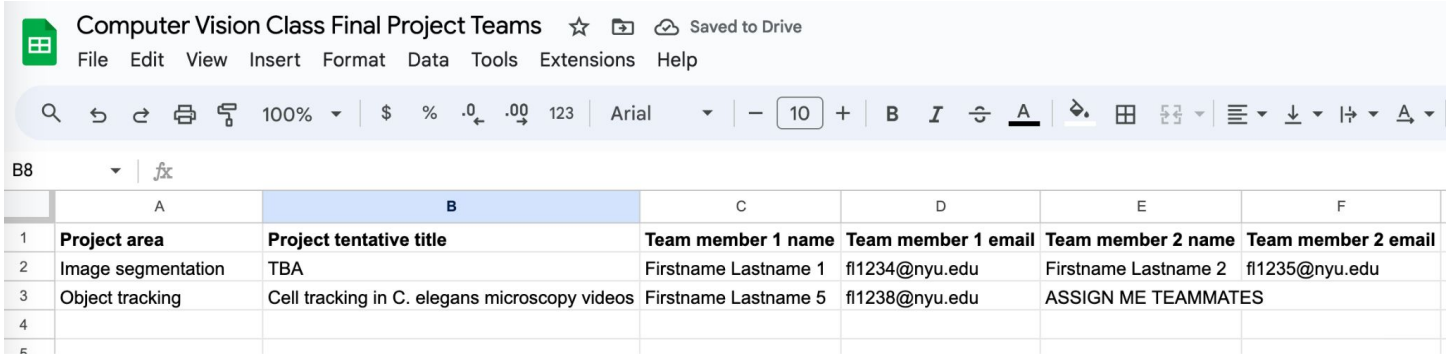
Each student must submit the names and NYU ID's of teammates

Also, each student must submit the tentative final project area.

To be done brightspace.

Use the match-making spreadsheet to declare project teams + ask to join other teams.

https://docs.google.com/spreadsheets/d/1RkjD30T0BsH8VO79zaxYJUE6GJFWTgVNu7nE4SJ1UYw/edit?usp=sharing

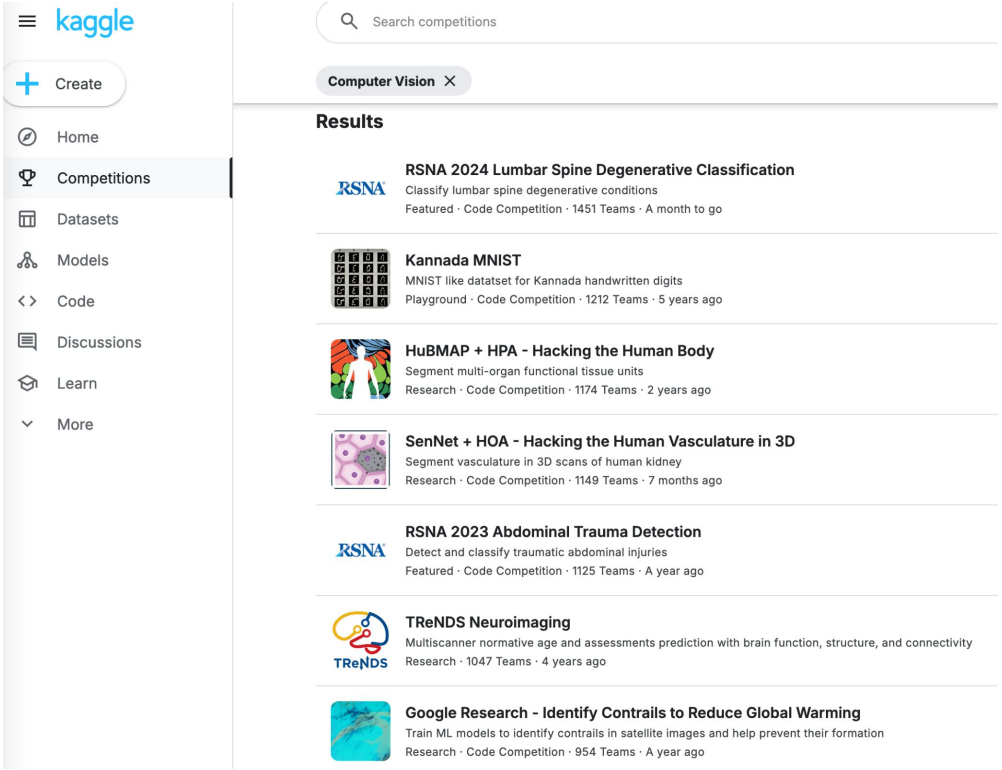


	A	B	C	D	E	F
1	**Project area**	**Project tentative title**	**Team member 1 name**	**Team member 1 email**	**Team member 2 name**	**Team member 2 email**
2	Image segmentation	TBA	Firstname Lastname 1	fl1234@nyu.edu	Firstname Lastname 2	fl1235@nyu.edu
3	Object tracking	Cell tracking in C. elegans microscopy videos	Firstname Lastname 5	fl1238@nyu.edu	ASSIGN ME TEAMMATES	
4						

Project ideas

Check slack channel #project-ideas for suggestions.

Kaggle (and other CV competitions) is a great resource for well benchmarked problems.



Take advantage of office hours to discuss assignments + final project ideas

Instructor: Prof. Erdem Varol

Email: ev2240@nyu.edu

Office Hours: On Zoom,15 min appointments.

TA: Rishabh Raj

Email: rr4574@nyu.edu

Office Hours: Tuesdays 1-2pm on Zoom.

TA: Malhar Patel

Email: mkp6112@nyu.edu

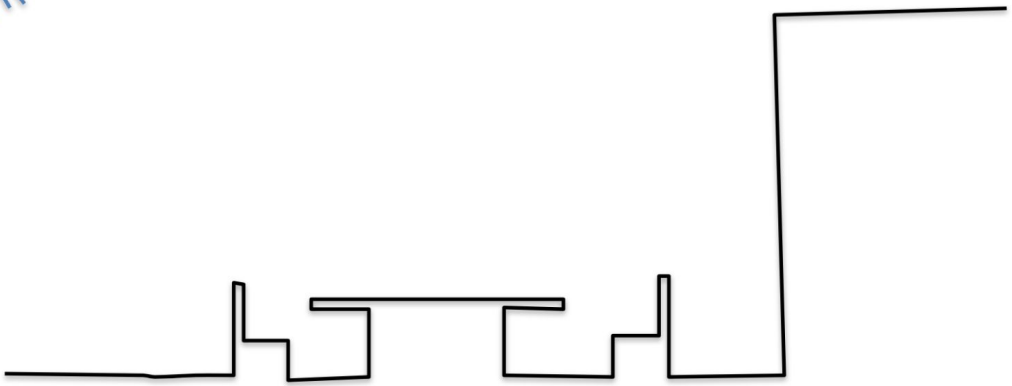
Office Hours: Mondays 1-2pm on Zoom.

Also in person in 370 Jay Street, room 1166

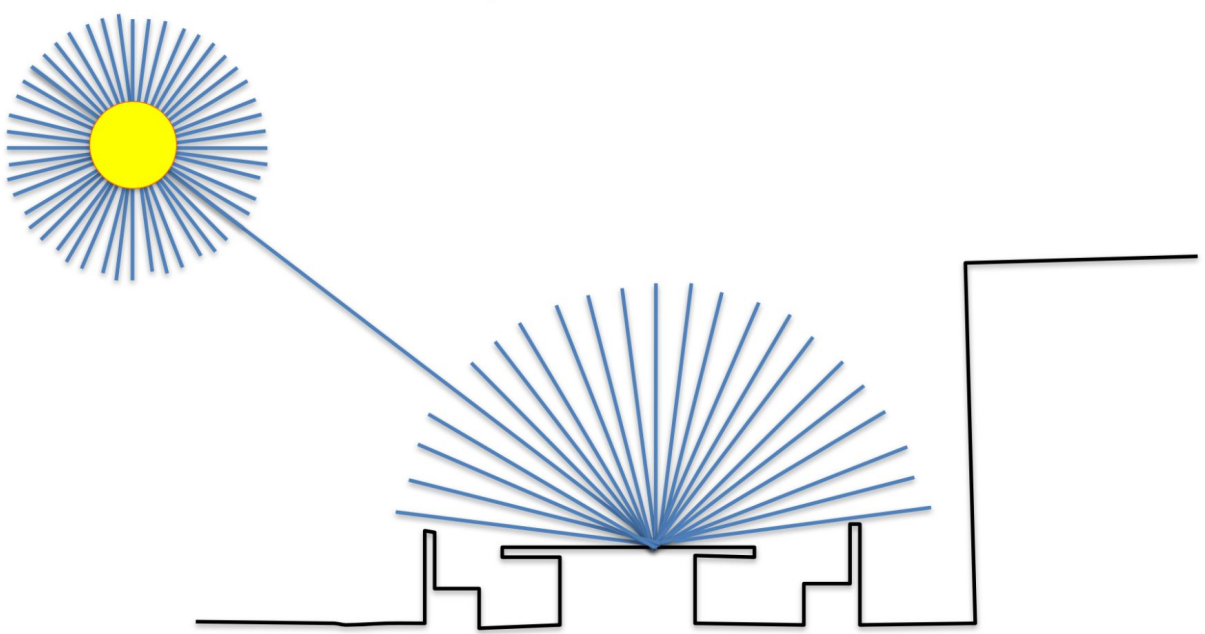
Image formation

Let there be light!

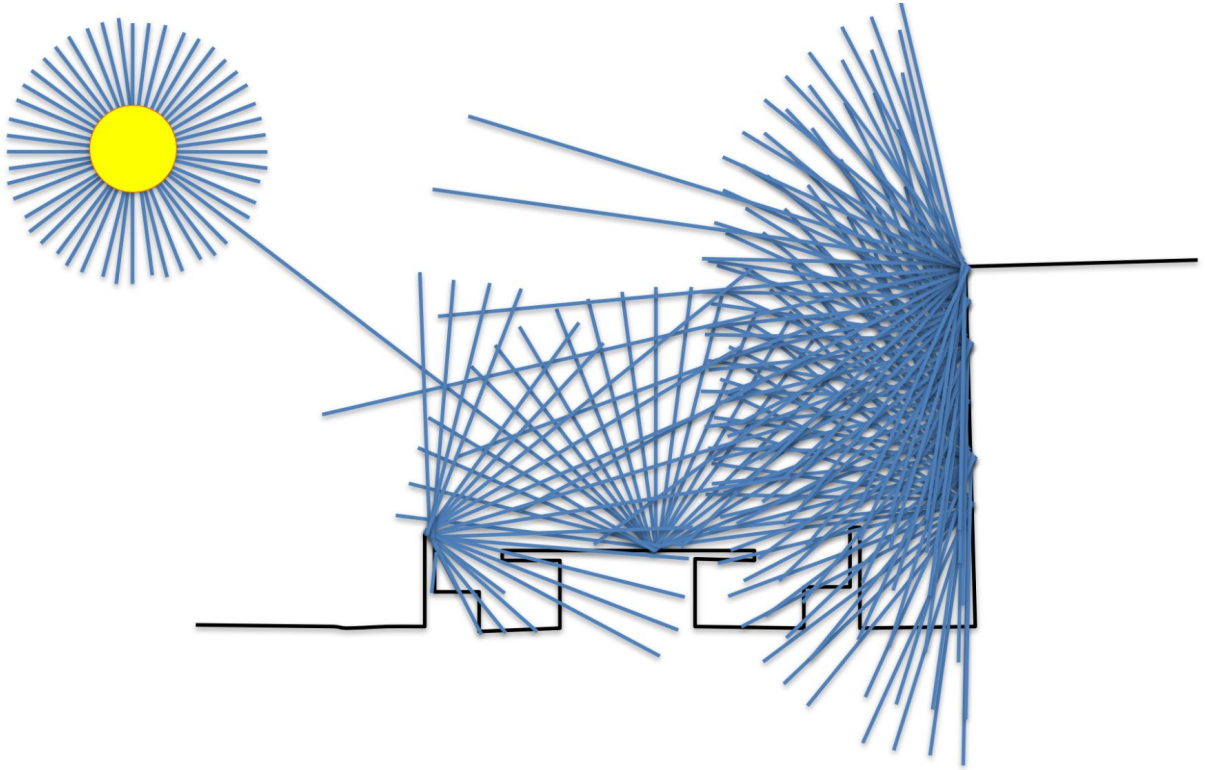
Let there be objects!



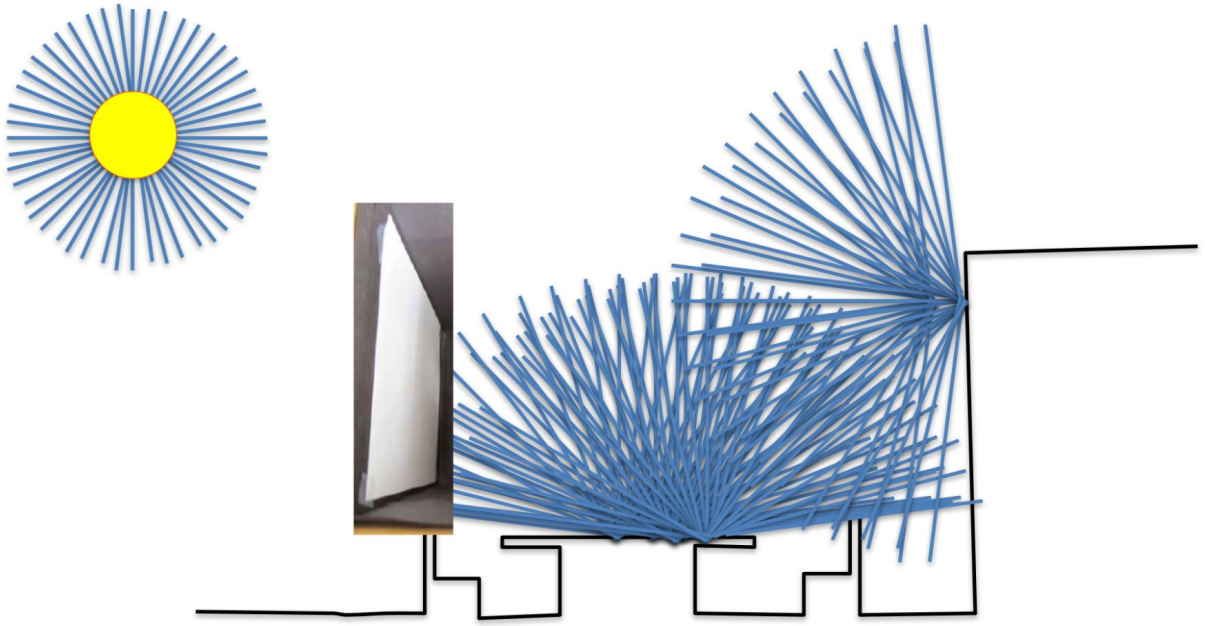
Let there be physics!



Let there be more physics…

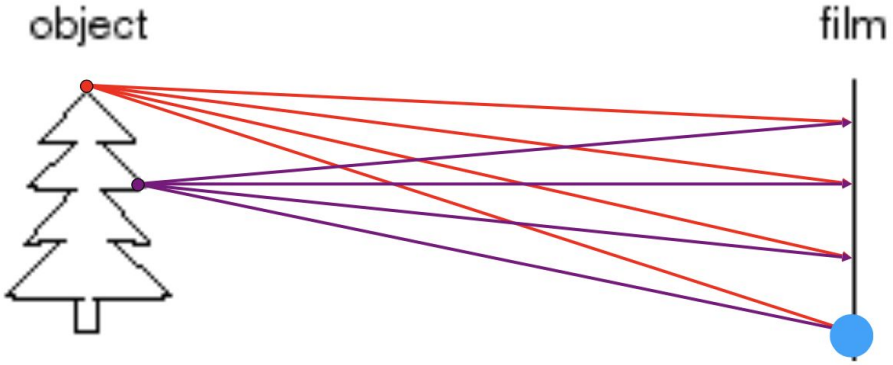


If we put a sensor, what happens?



There will be no image. Why?

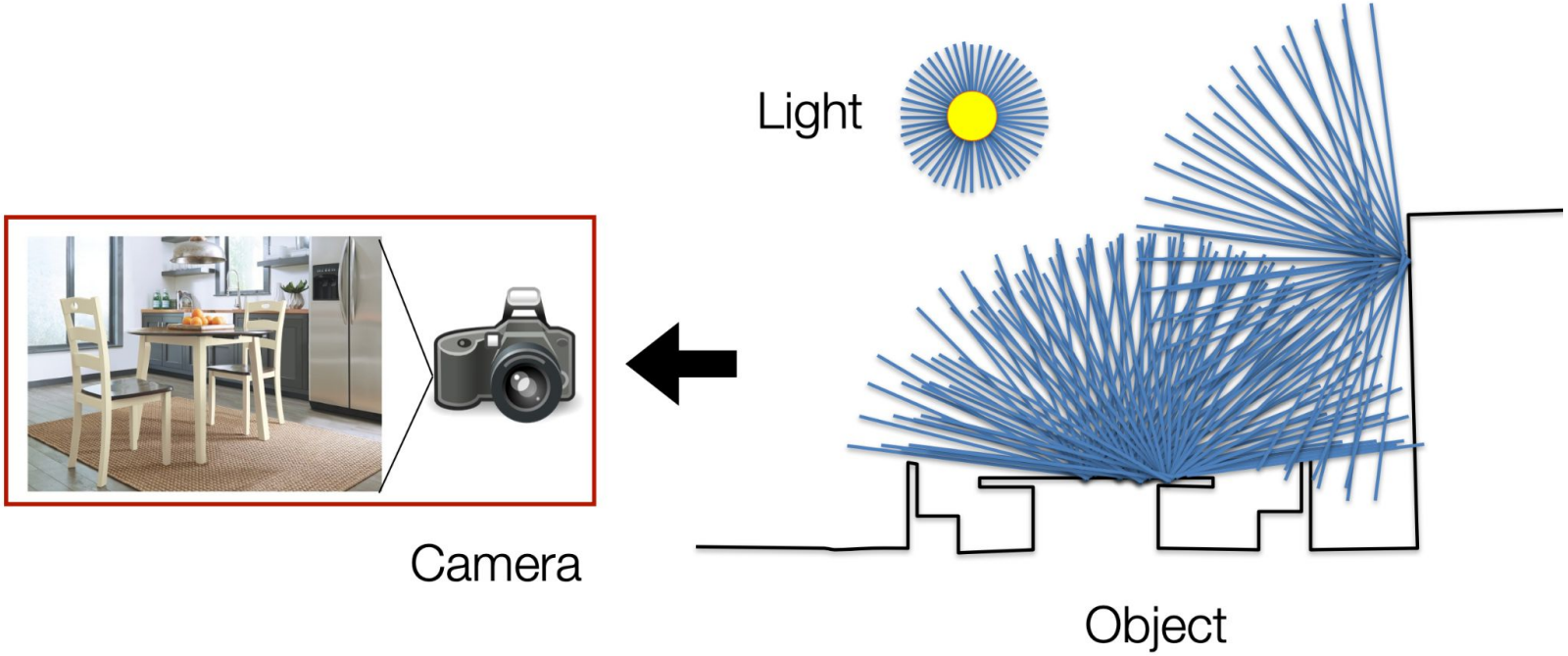
Why doesn’t this generate an image?



For each point on the film

- it receives light from different directions
- in the end, the color is averaged out

What is needed? A tool to restrict light that comes only from where we want to see…



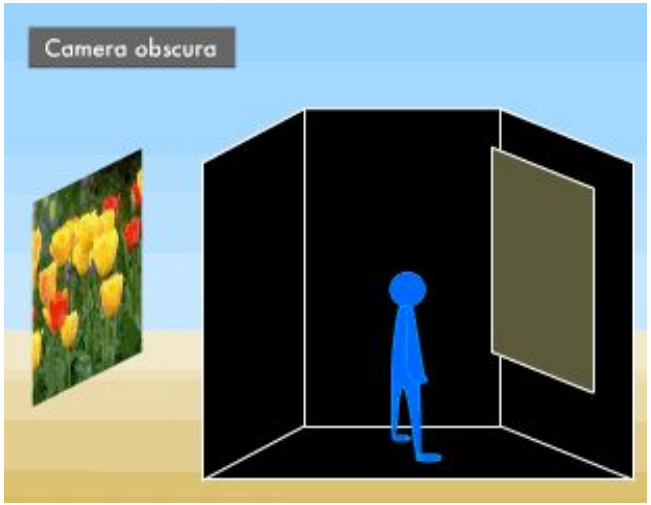
Camera obscura

First camera in history

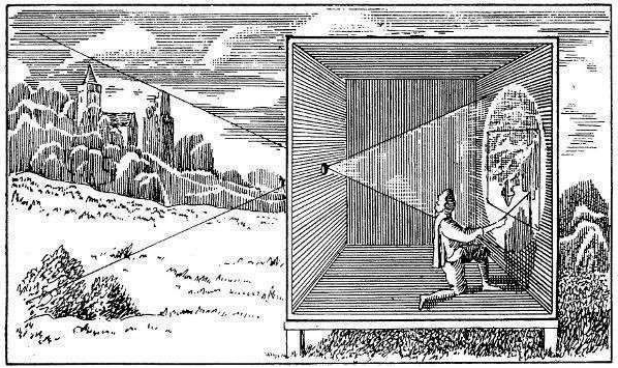
Mentioned in ~500 BC in Chinese writings

Term coined and formalized in 1604 by Kepler

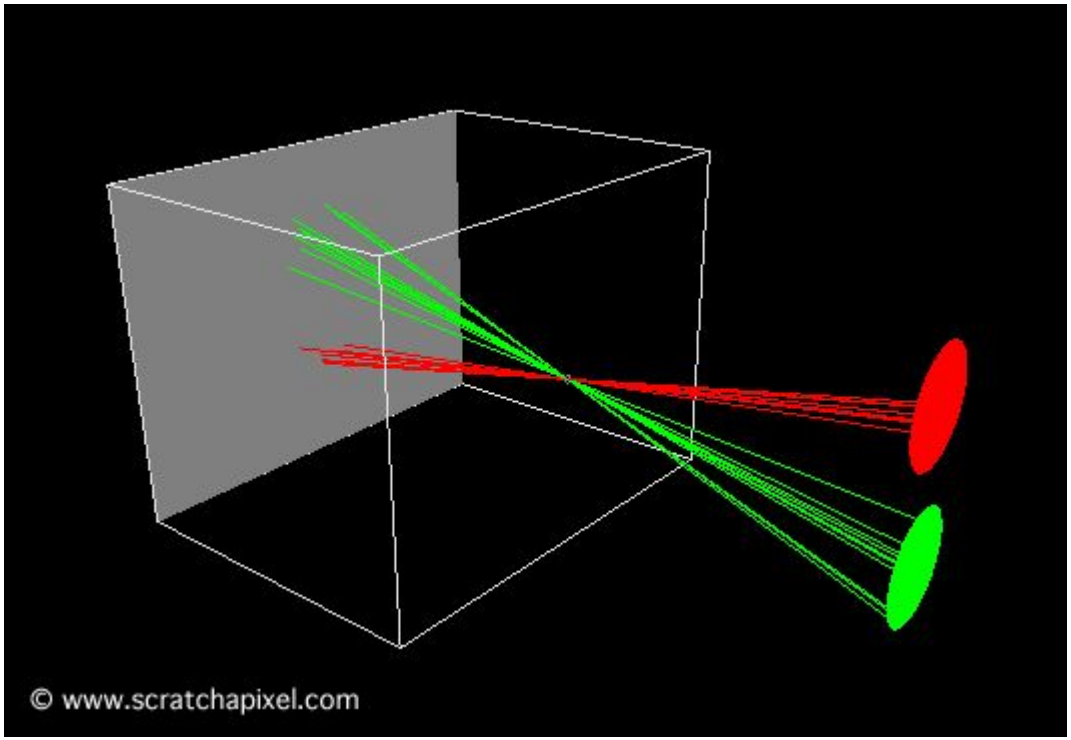
17th century Dutch painters used it for perspective studies.



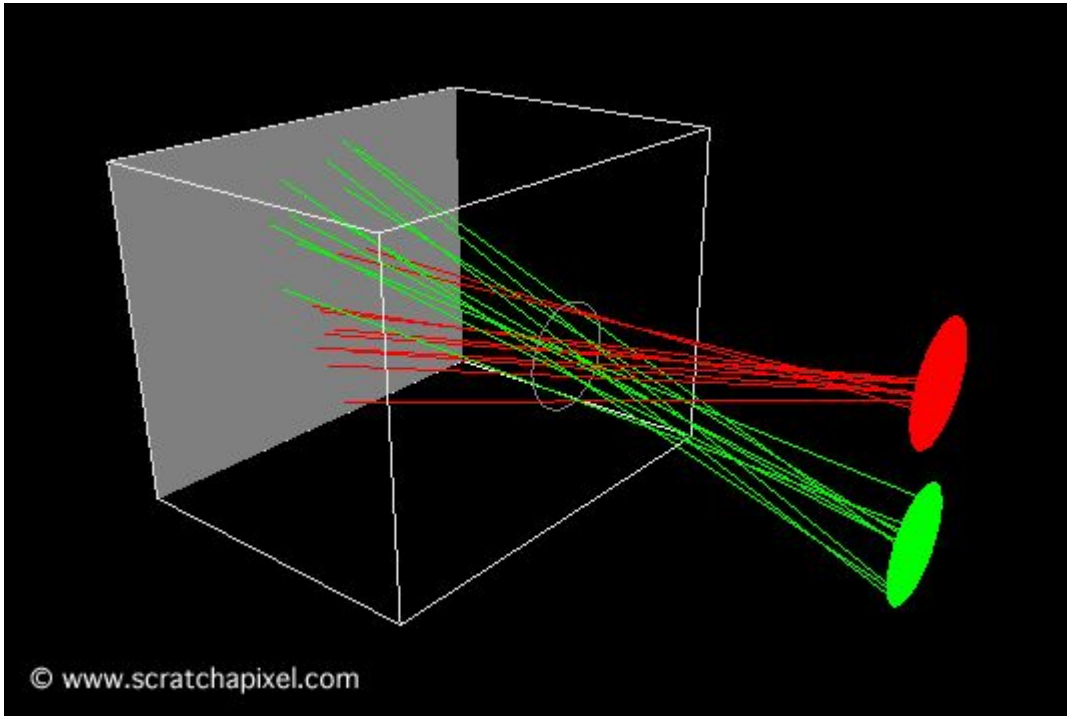

AD VITELLIONEM
PARALIPOMENA,
Quibus
ASTRONOMIÆ
PARS OPTICA
TRADITVR:
Potissimùm
DE ARTIFICIOSA OBSERVATIONE ET ÆSTIMATIONE DIAMETRORVM
deliquiorumq; Solis & Lunæ.
CVM EXEMPLIS INSIGNIVM ECLIPSIVM.
Habes hoc libro, Lector, inter alia multa nova,
Tractatum luculentum de modo visionis, & humorum oculi
usu, contra Opticos & Anatomicos,
AVTHORE
IOANNE KEPLERO, S. C. Mto
Mathematico.
FRANCOFVRTI,
Apud Claudium Marnium & Hæredes Ioannis Aubrii
Anno M. DCIV.
Cum Privilegio S. C. Maiestatis.



How does it work?



What if the pinhole was bigger?



Same issue as having no camera arises - lots of uncontrolled photons hitting the sensor.

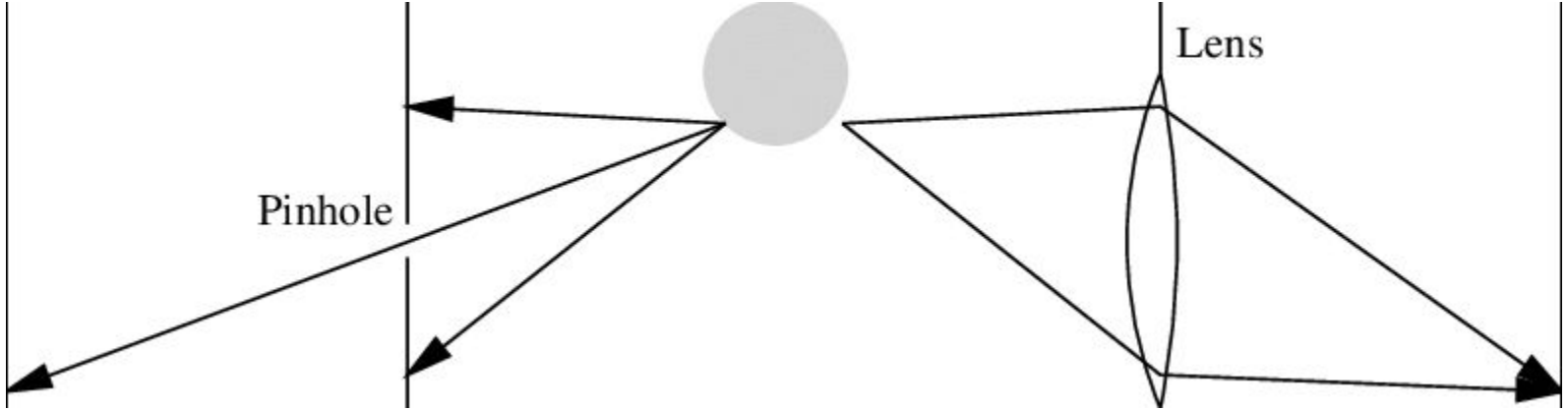
21st century DIY pinhole camera

View outside the class

Camera lens

Camera lens replaced by a very small hole

Pinhole vs. an actual lensed camera

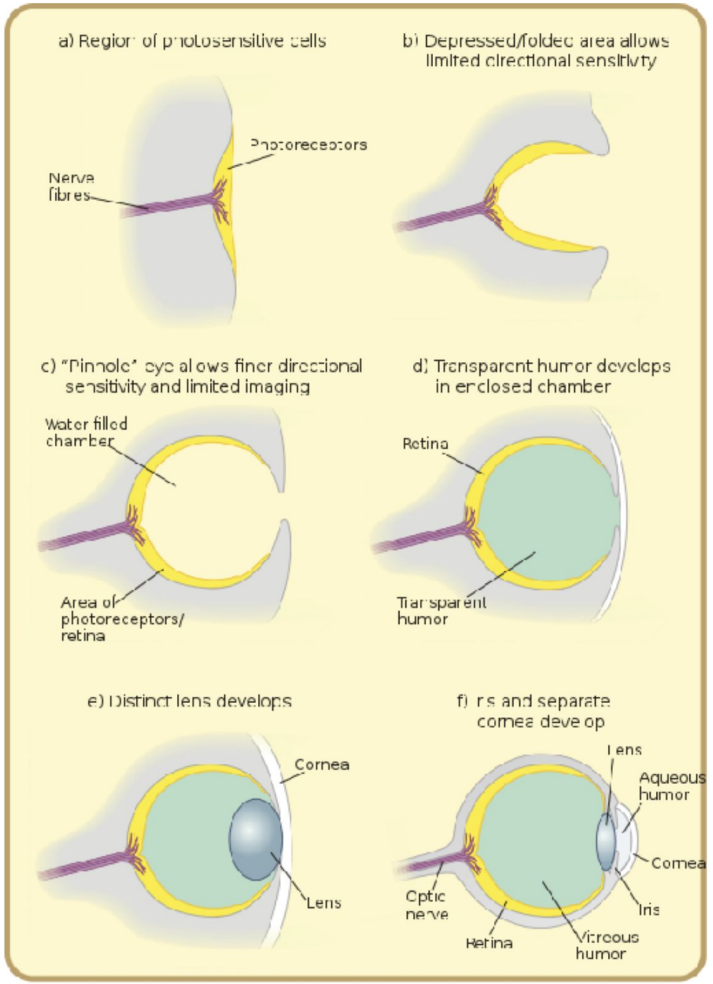


Lens allows for a larger set of photons to be focused on a single point.
Pinhole merely restricts the photons that are scattered in all directions.

Recall first lecture

Primitive “eyes” were pinhole cameras.

Advanced organisms evolved to have lenses in their “cameras”.

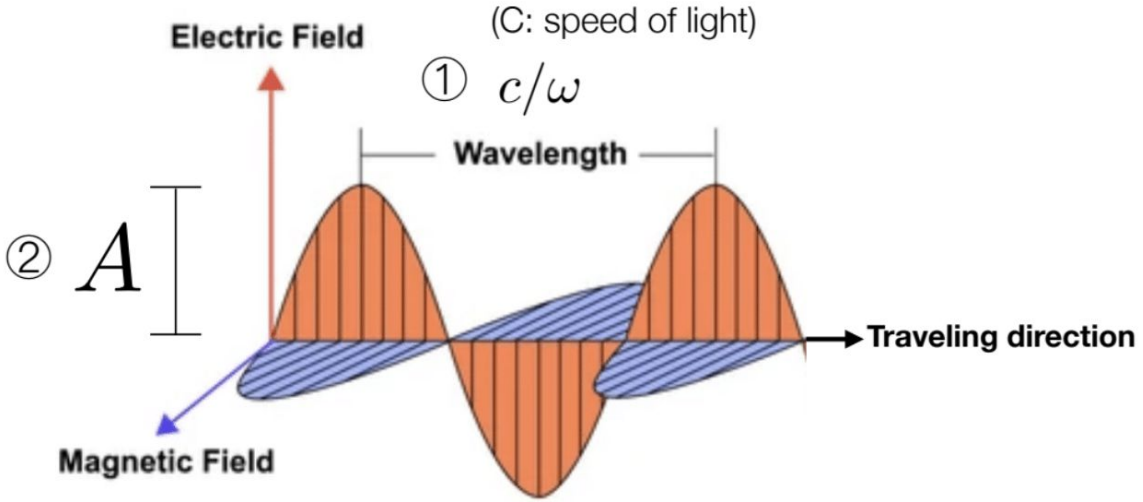


Takeaway:

An image is a focused beam of photons that originate from a specific location in space.

Blur is a collection of photons that come from many random places in space.

What are photons? A type of electromagnetic radiation



① ω : Frequency (how fast it vibrates)

② A : Amplitude (how big it vibrates)

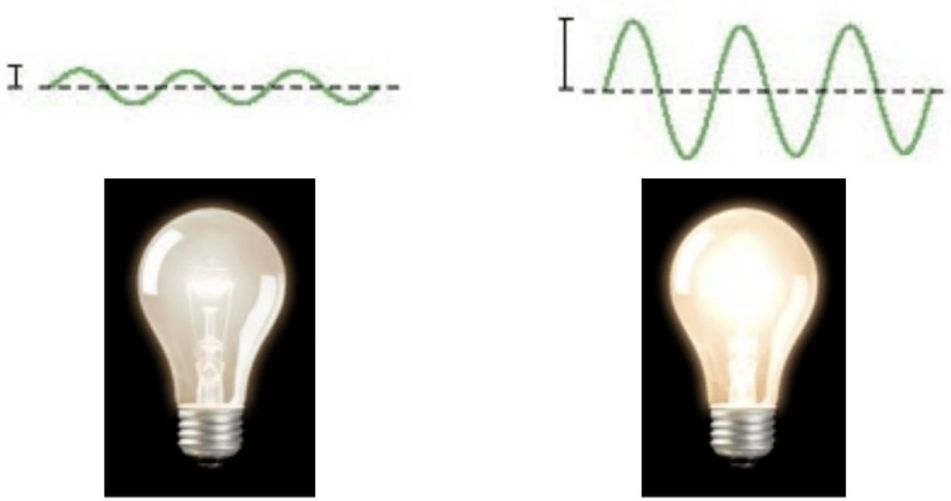
Photons have two qualities

Input image

Brightness

Color

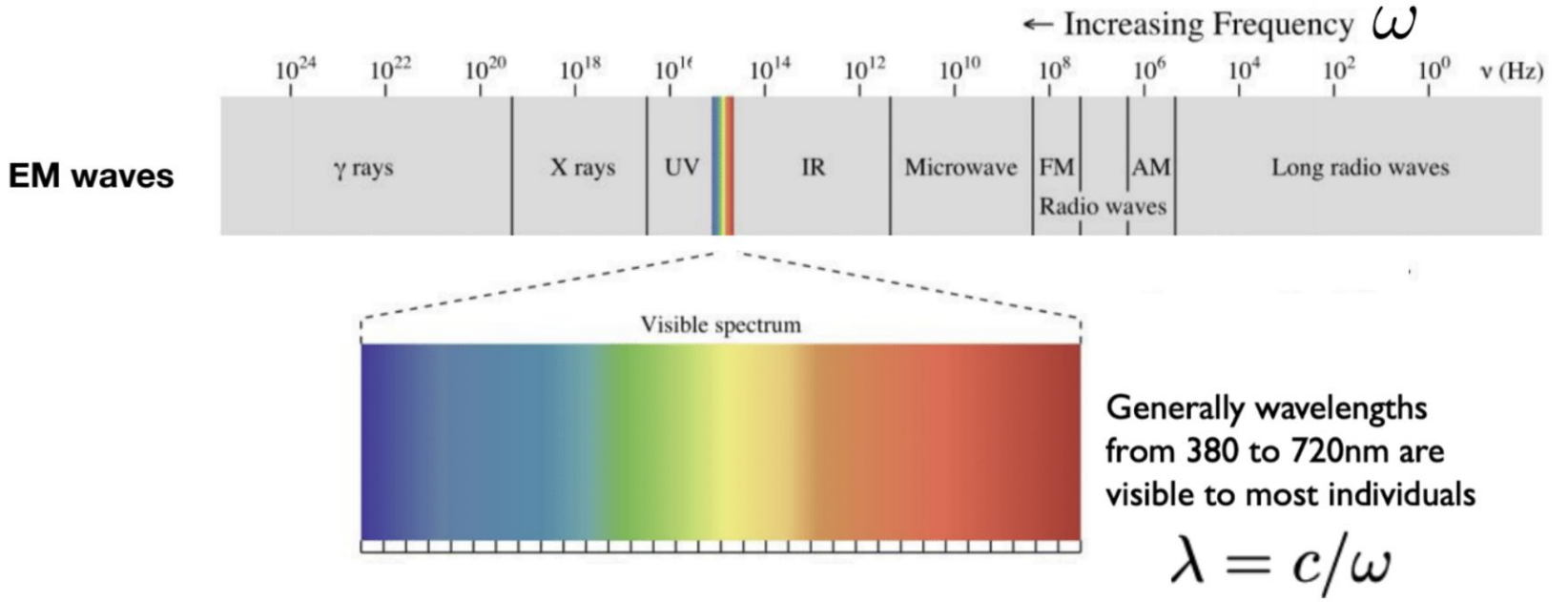
Photon amplitude = brightness



Low brightness

High brightness

Photon frequency = color

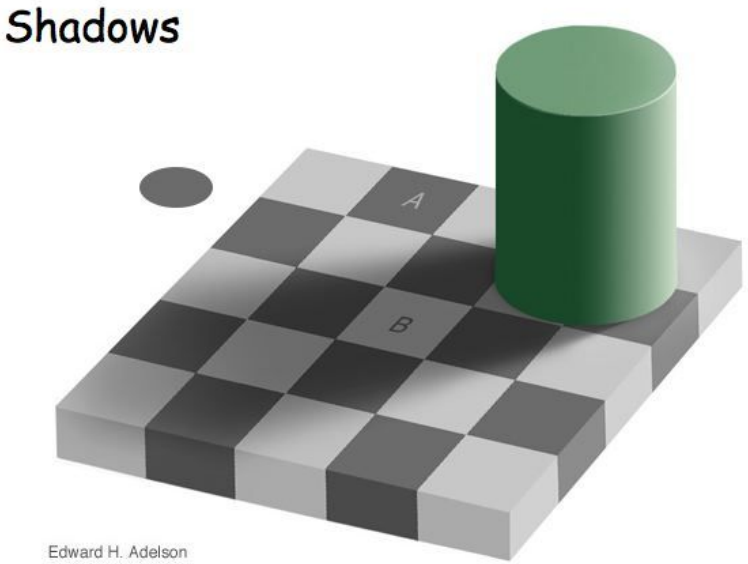


Generally wavelengths from 380 to 720nm are visible to most individuals

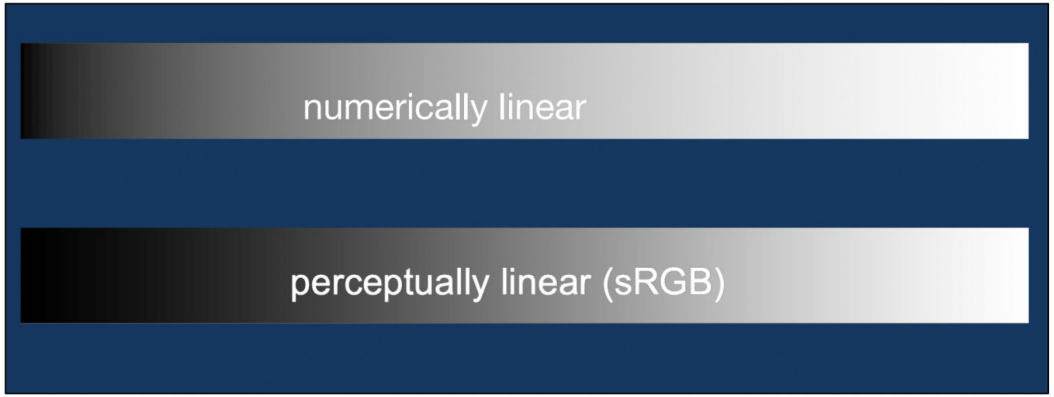
$$\lambda = c/\omega$$

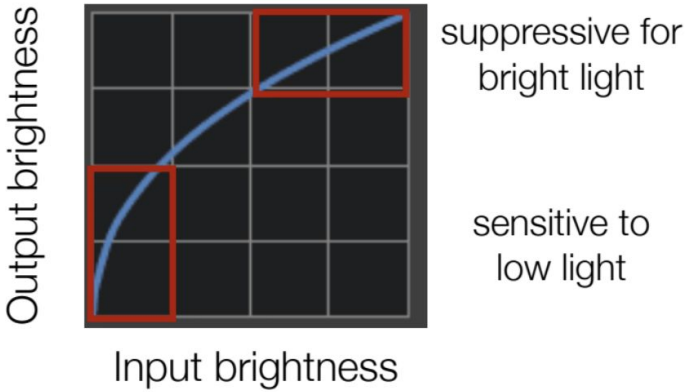
So far, we've talked about vision from a purely objective perspective: photons, brightness, color…

Vision is also subjective = perception



Perceptual brightness is non-linear

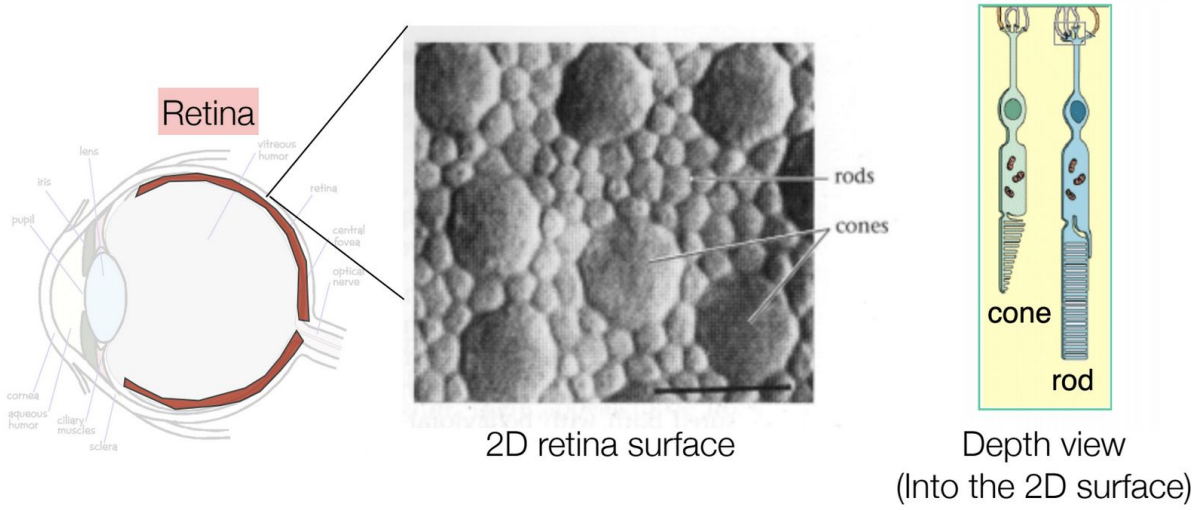




Human perception curve

Why? Answer = evolution

Retina: cones and rods (light sensors)



2D retina surface

Depth view
(Into the 2D surface)

Two types of sensors: one for color, one for brightness

Cone
- Specialized in color

Normal cones

Deuteranopia

Tritanopia

Monochromacy

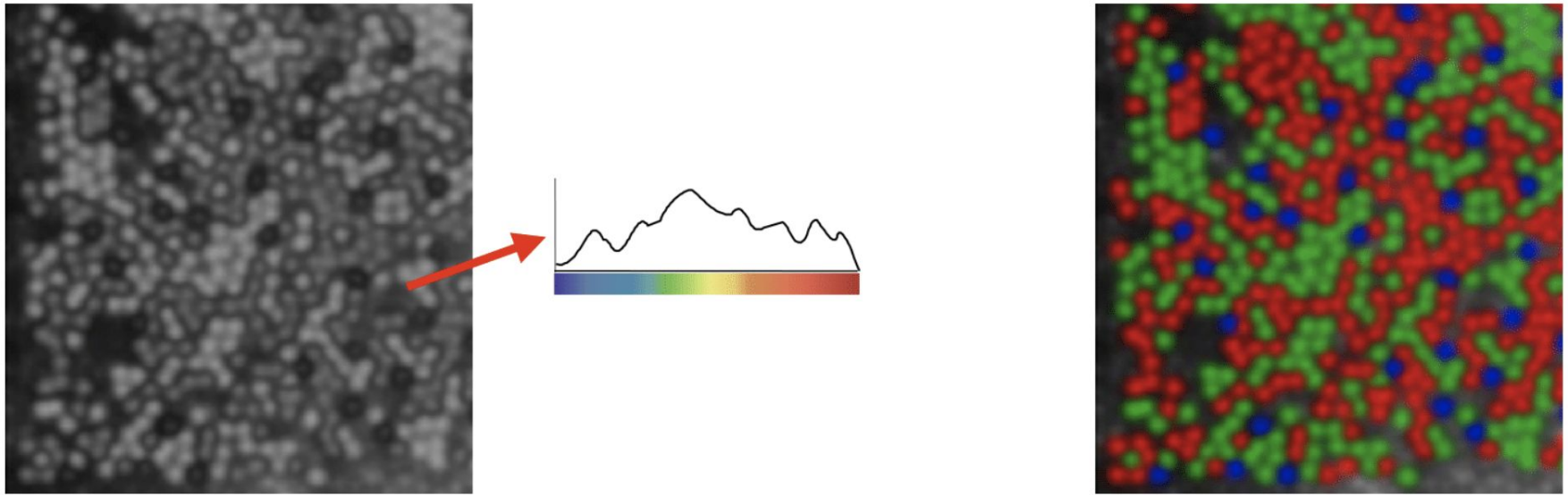
Impaired cones
(Color blind)

Rod
- Sensitive to brightness
- gray-scale

Normal rods

Impaired rods
(Night blindness)

3 types of cones = R, G, B

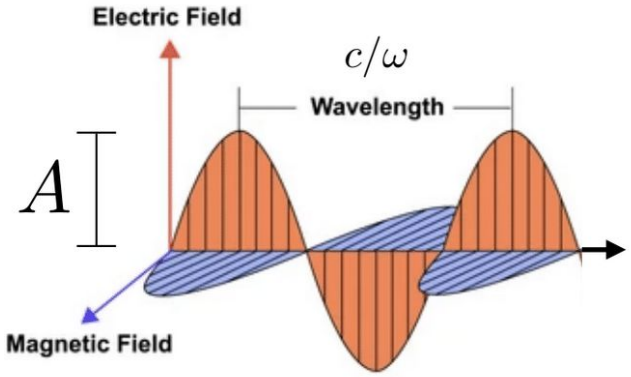


Note: Each cone doesn’t pick a single “color” but a range of colors.

Objective vs. subjective vision

Physics (Optics)

- Wave frequency → Color
- Wave amplitude → Brightness



We can measure these objectively with sensors and cameras.

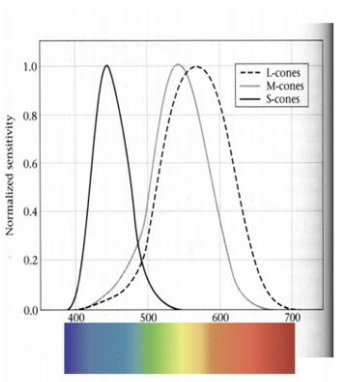
Objective vs. subjective vision

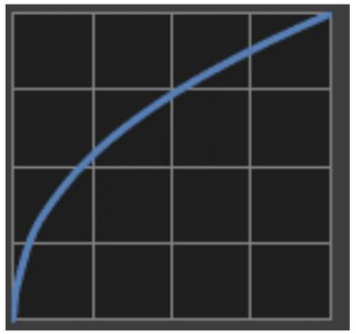
Perception

Color	• 3 types of cones
Brightness	• Non-linear response (**Gamma curve**)

We can't measure these objectively - instead, we build surrogate systems to mimic it.

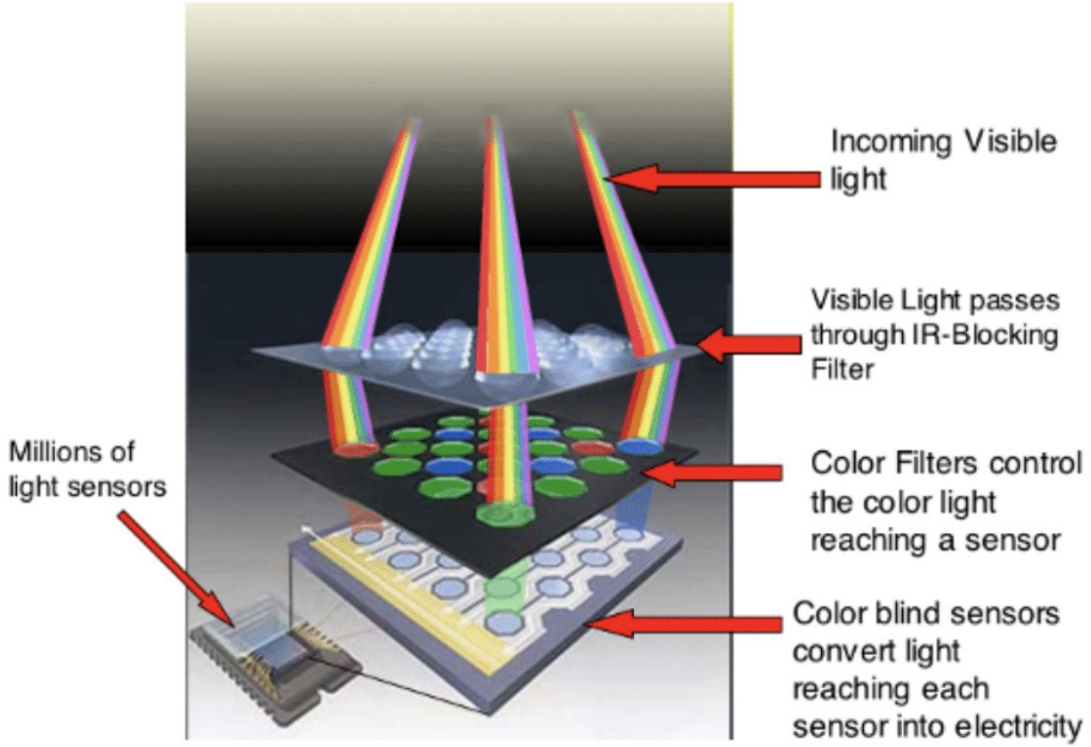
This is computer vision.





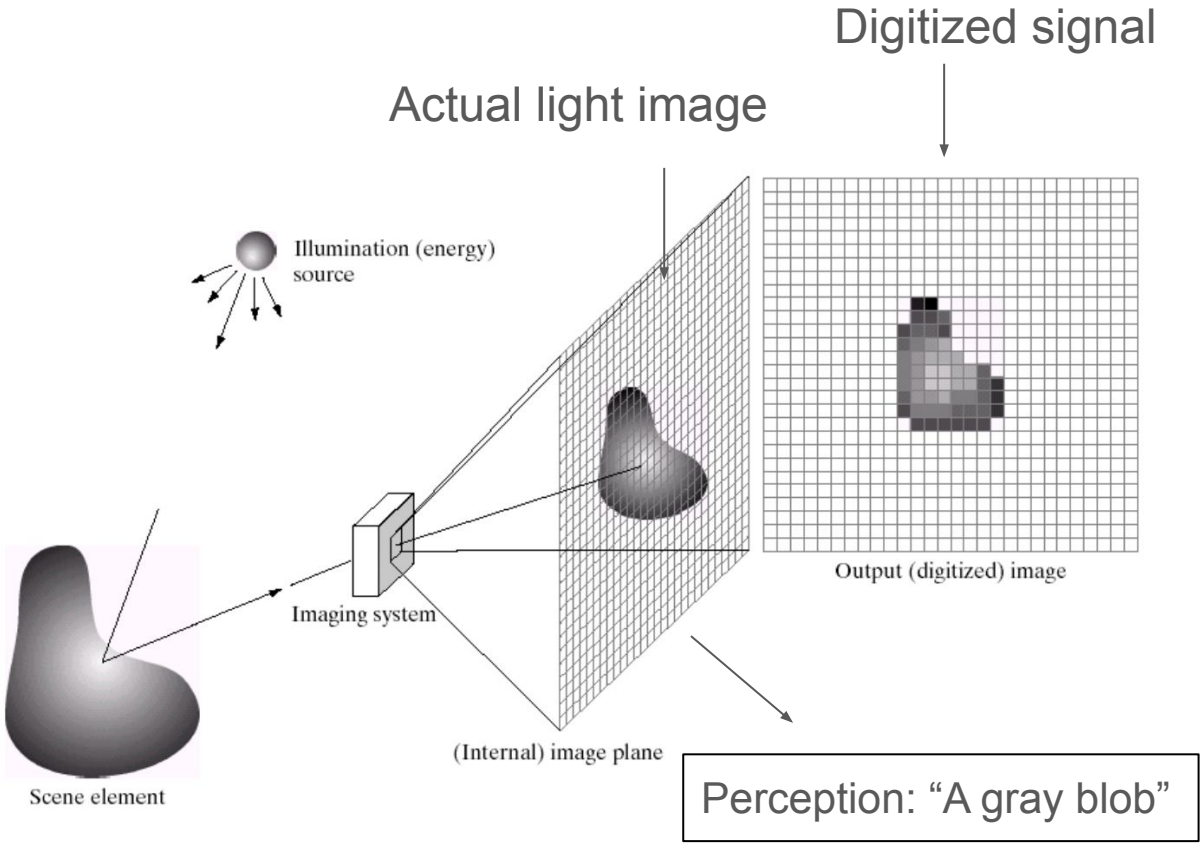
How do we build artificial perception systems?

Camera sensors



Digital images

Each pixel is a sampling of the number of photons in particular spatial bin.

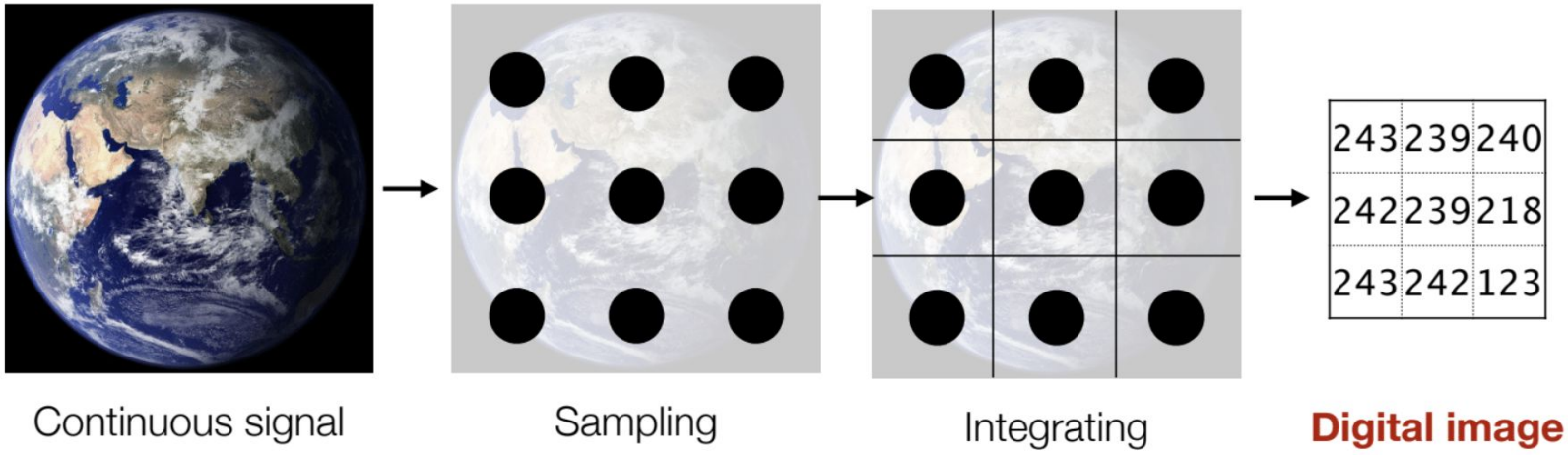


What computers see

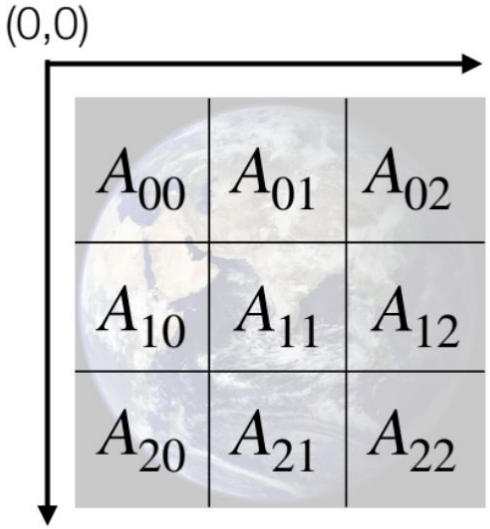
243	239	240	225	206	185	188	218	211	206	216	225
242	239	218	110	67	31	34	152	213	206	208	221
243	242	123	58	94	82	132	77	108	208	208	215
235	217	115	212	243	236	247	139	91	209	208	211
233	208	131	222	219	226	196	114	74	208	213	214
232	217	131	116	77	150	69	56	52	201	228	223
232	232	182	186	184	179	159	123	93	232	235	235
232	236	201	154	216	133	129	81	175	252	241	240
235	238	230	128	172	138	65	63	234	249	241	245
237	236	247	143	59	78	10	94	255	248	247	251
234	237	245	193	55	33	115	144	213	255	253	251
248	245	161	128	149	109	138	65	47	156	239	255
190	107	39	102	94	73	114	58	17	7	51	137
23	32	33	148	168	203	179	43	27	17	12	8
17	26	12	160	255	255	109	22	26	19	35	24

What humans see

An digital image is a matrix of pixel values



Each pixel has a coordinate



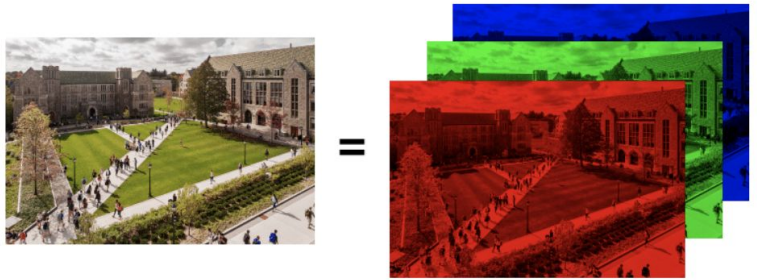
Put matrix on top image

```
A = load_image(..)
print(A[2,1])
```

243	239	240
242	239	218
243	211	123

In Python

Color images are 3D matrices = tensors

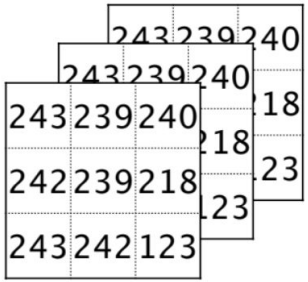


Grayscale image: 2D matrix (M x N)

Color image: 3D tensor (M x N x 3)

243	239	240
242	239	218
243	242	123

A[row,col]



A[row,col,chan]

52

What makes an image informative

Example matrix: **(uint8)**$^{3\times3}$

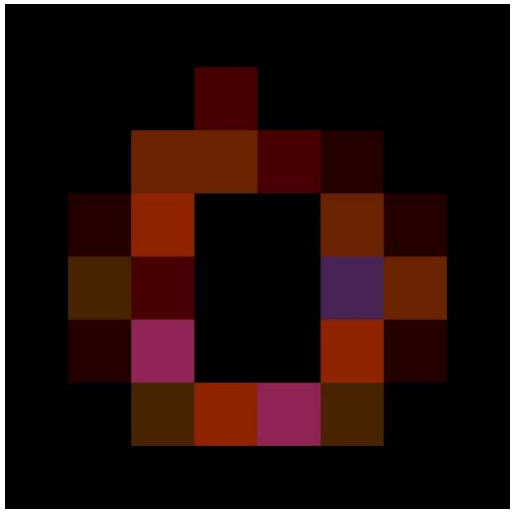
243	239	240
242	239	218
243	242	123

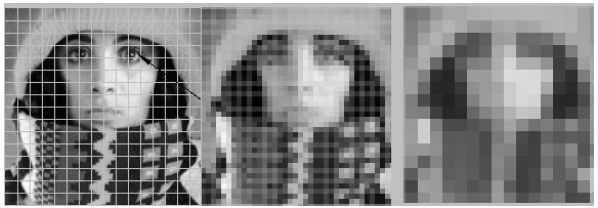
Data type uint8: 0-255

- Bit depth: uint8 $\rightarrow$ uint16
- Resolution: $3 \times 3 \rightarrow 30 \times 30$
- Color: $30 \times 30 \rightarrow 30 \times 30 \times 3$

Quantization

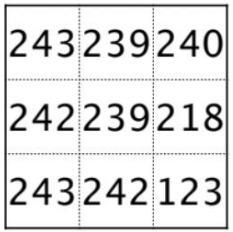
How pixels are quantized affects image resolution.





Summary I: Digital image = $(\text{data type})^{N_1 \times N_2 \times \ldots \times c}$

World

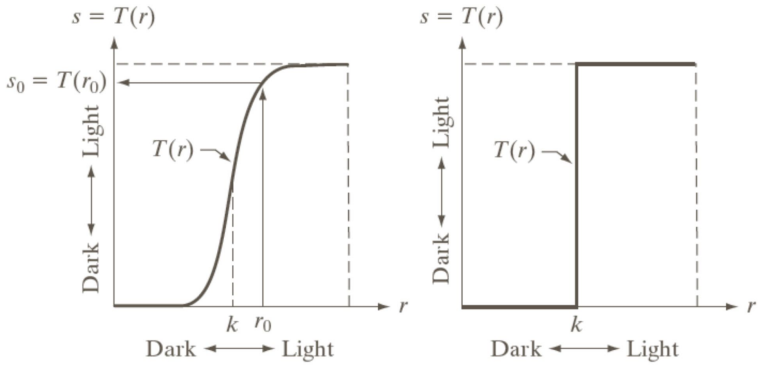


Digital image

- data type : uint8, uint16 (**bit depth**)
- $(N_1 \times N_2 \times \ldots)$: pixel density=**resolution**
- c : RGB channels for any **color**

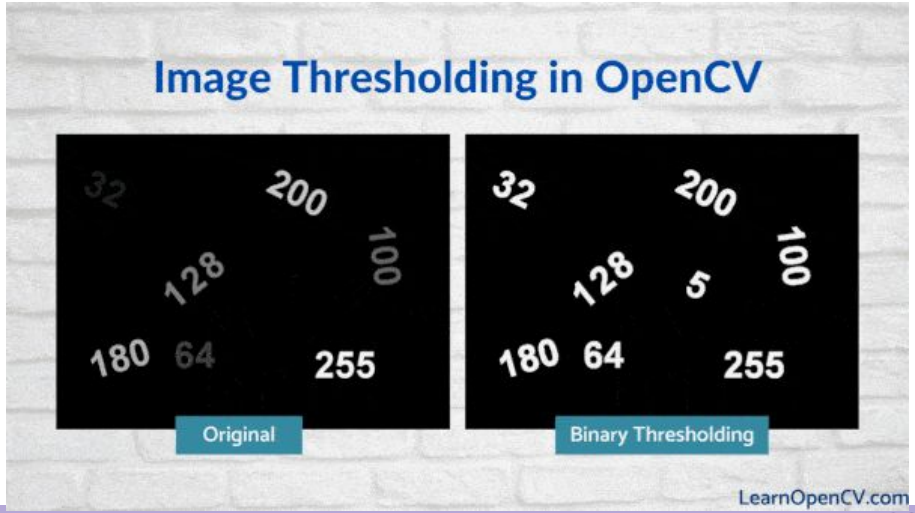
What kind of operations can we do on digital images?

Intensity transformations



$$s = T(r)$$

Transform each pixel value pointwise.



Pixel level operations

90	3	241	65	213
242	143	122	71	8
203	103	45	167	240
29	239	2	93	218
243	242	183	54	123

- 0D linear function (pixel) $ax + b$
- 1D linear function (color) $\begin{bmatrix} 0.19 & 0.77 & 0.39 \\ 0.17 & 0.69 & 0.35 \\ 0.13 & 0.53 & 0.27 \end{bmatrix} \begin{bmatrix} x_R \\ x_G \\ x_B \end{bmatrix}$
- Any function (transfer function)

Input	Output
0	0
…	…
100	50
…	…
255	127

Brightness adjustment = addition / subtraction

Pixel arithmetic

243	222	171
203	145	137
209	132	193

123	102	51
83	25	17
89	12	73

$$p \longrightarrow p - 120$$

Input pixel value

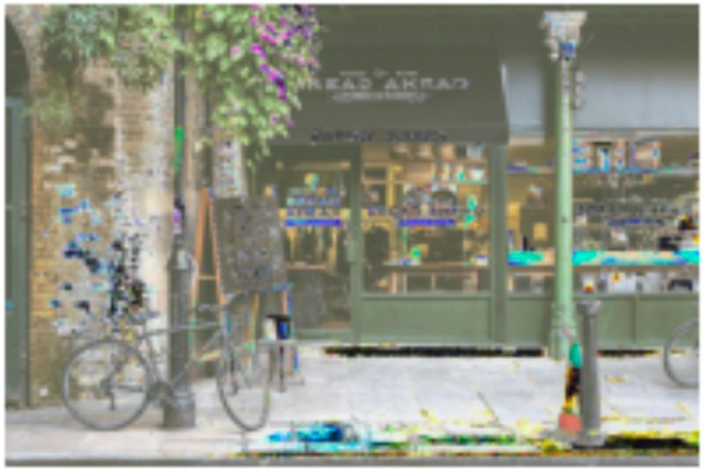
Output pixel value

```
I = imread('image.png')
I = I - 120
```

Caution: numerical overflow

100 + 200 = ?

Overflow: 300-256 = **44**



Uint8 image arithmetic

- convert to “Float”
- do arithmetic
- clip to [0,255]
- convert it back to “uint8”

```
np.clip((im_shop.astype(float)+60),0,255).astype(np.uint8)
```

Contrast adjustment = multiplication/division

We want bright to be brighter, dark to be darker.

If we only multiplied, this wouldn't work.

121	111	85
101	72	68
104	66	96

181	166	127
151	108	102
156	99	144

$p \longrightarrow p * 1.5$

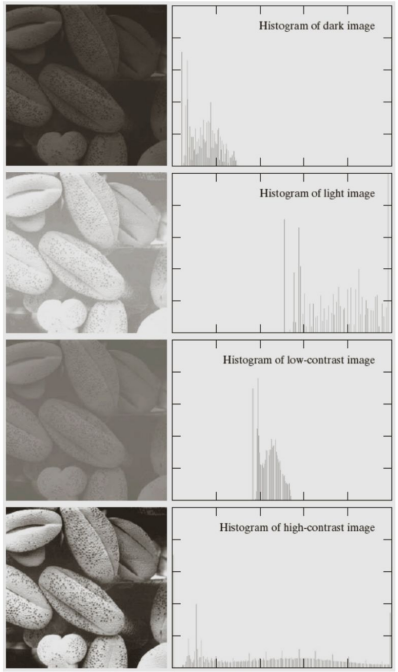
Input pixel value

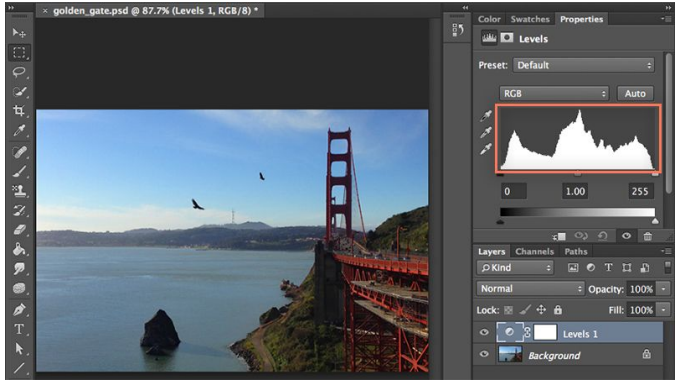
Output pixel value

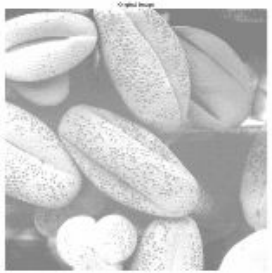
Bright gets brighter, dark **also** gets brighter.

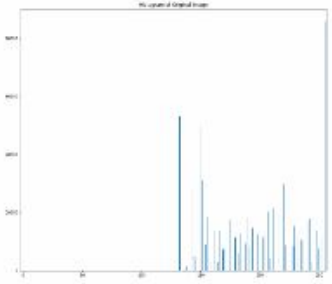
Think of “brightness” vs “darkness” as a distribution of pixel values = a histogram.

- $h(r_i) = n_i$
  - Histogram: number of times intensity level r_i appears in the image
- $p(r_i) = n_i/NM$
  - normalized histogram
  - also a probability of occurrence





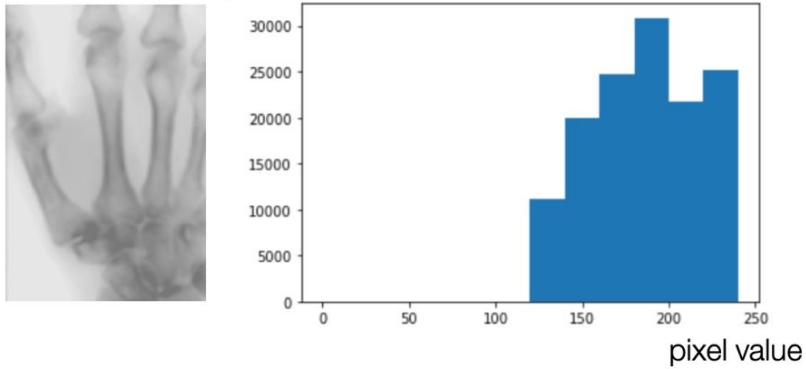




imgflip.com

What are the operations to improve contrast?

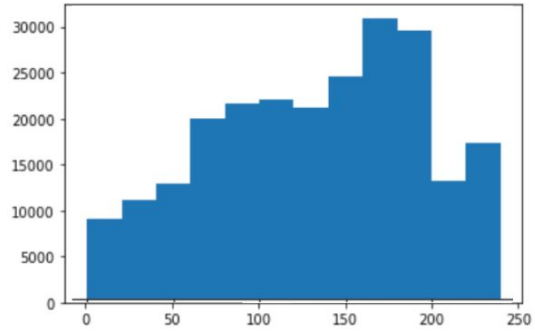
I: (min, max) = 120, 230



Pixel value histogram

?

I: (min, max) = 0, 255

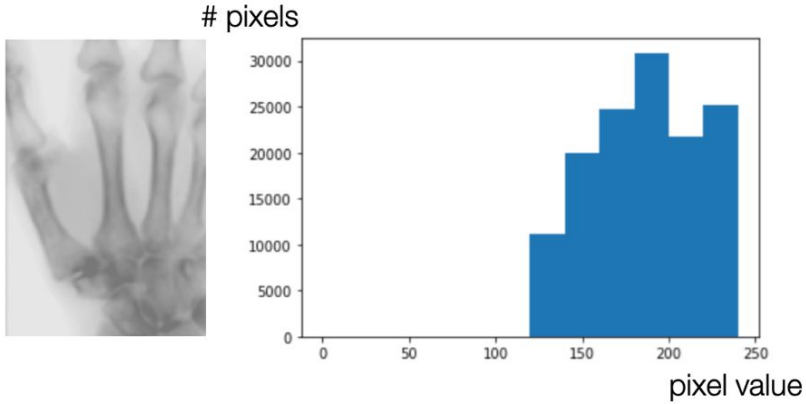


(min, max)

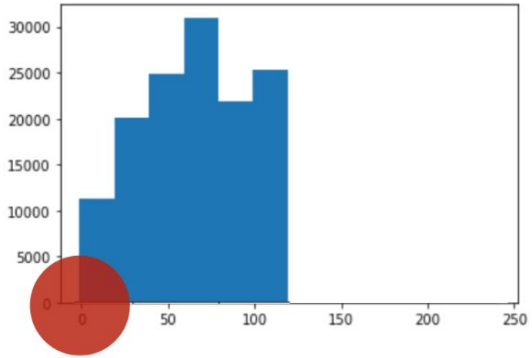
I: (min, max) = 120, 230

```
I = I - I.min()
```

0, 110



Pixel value histogram

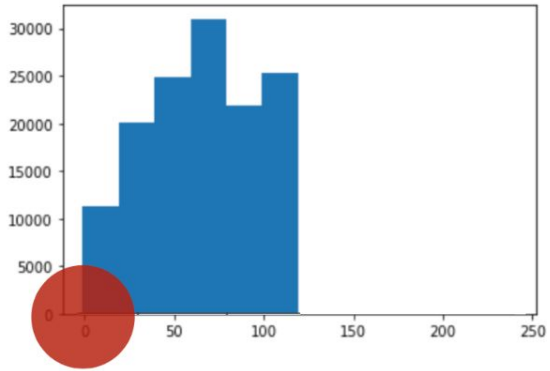


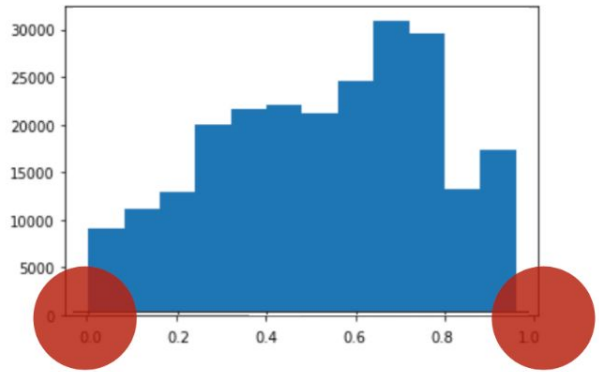
Step 2: scale to right endpoint (min, max)

```
I = I - I.min()
I = I / (I.max() - I.min())
```

0, 110

0, 1

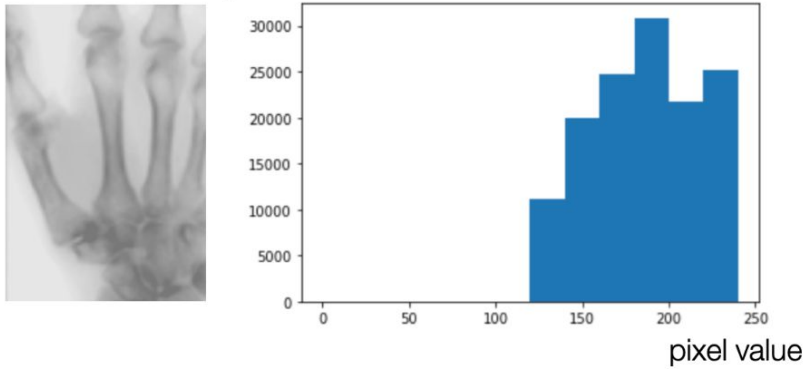




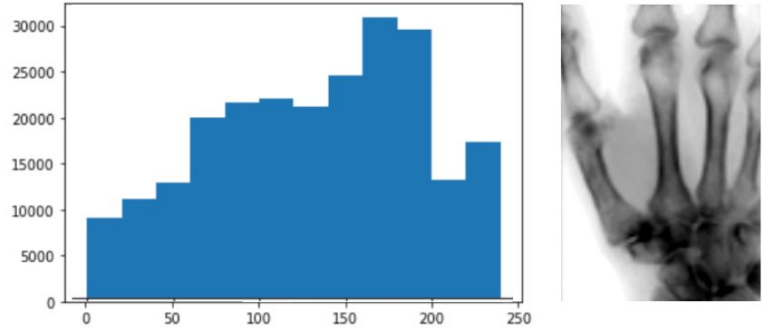
(min, max)

I: (min, max) = 120, 230

```
I = I - 120                          0, 110
I = I / (I.max() - I.min())          0, 1
I = I * 255                          0, 255
```



Pixel value histogram



A combination of multiplicative and additive operations needed.

Intensity vs. color transformations

Intensity operations are arithmetic - pixelwise.

Color transformations require matrix operations since operating on a vector.

1 pixel

Intensity
(scalar: arithmetic)

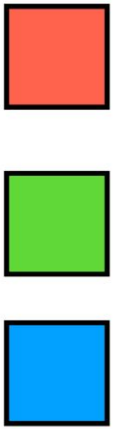
RGB
(vector: linear algebra)

Color filter

input

sepia effect

Matrix operation on each pixel color vector



= 0.19 R + 0.77 G + 0.39 B

= 0.17 R + 0.69 G + 0.35 B

= 0.13 R + 0.53 G + 0.27 B

RGB_new

(Image range: 0-1)

$$= \begin{bmatrix} 0.19 & 0.77 & 0.39 \\ 0.17 & 0.69 & 0.35 \\ 0.13 & 0.53 & 0.27 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Matrix operation on a tensor

Color images are N x M x C tensors.

“Vectorize” each image into a NM x C matrix.

Each entry is a pixel.

Transform each pixel and then reshape it back to the N x M x C form.

```
color_trans = np.array([[0.189, 0.168, 0.131],\
                        [0.769, 0.686, 0.534],\
                        [0.393, 0.349, 0.272]])
```

```
# reshape image into Nx3
im_shop_reshape = im_shop.reshape(-1,3)
```

```
# convert image range into 0-1
im_shop_reshape = im_shop_reshape/ 255.0
```

```
# Sepia color transform
im_shop_sepia = np.matmul(im_shop_reshape, color_trans)
```

```
# reshape it back
im_shop_sepia = im_shop_sepia.reshape(im_shop.shape)
```

Non-linear color transformations

input

summer effect

Lookup tables

Transfer function

$$f : p \rightarrow p'$$

Input pixel	Output pixel
0	0
…	…
100	50
…	…
255	127

Example:

$f_1(p) = p/2$

Key observation:

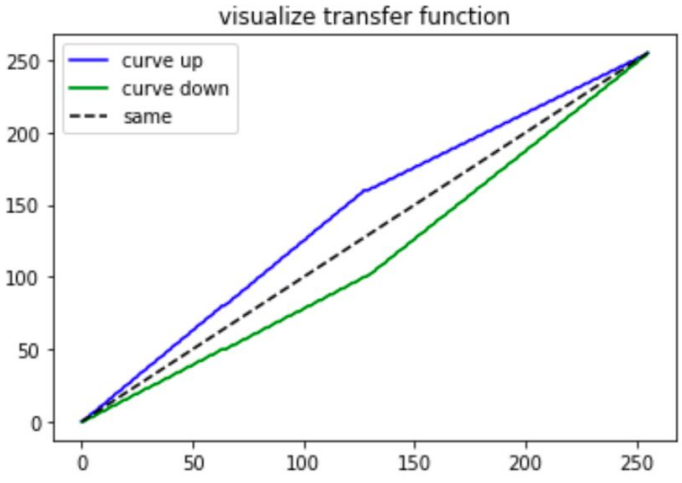
Array index -> value

```
# load in a matrix (uint8: 0-255)
I = imread('image.png')

# transfer function
f1 = np.arange(255) // 2

# apply point-level operation
# (relabel input pixel)
I_out = f1[I]
```

Example of non-linear transfer function



Goal: more red and less blue

1. Create the transfer function

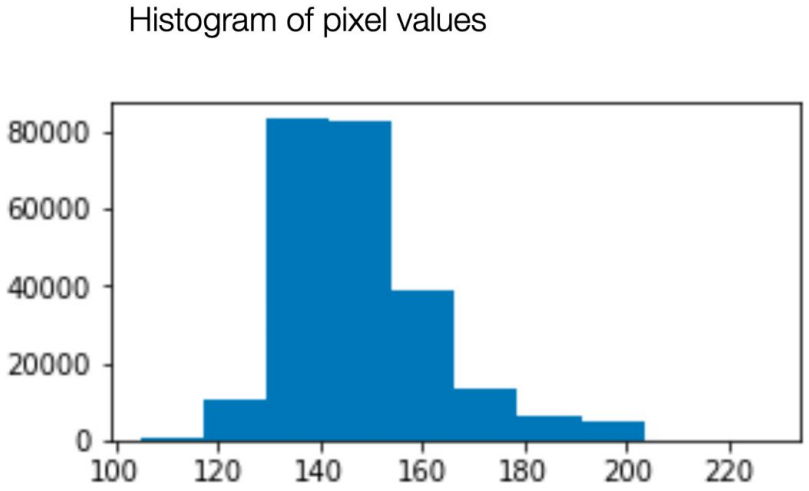
```
transfer_up = np.zeros(256, np.uint8)
transfer_up[:64] = np.linspace(0, 80, 64)
transfer_up[64:128] = np.linspace(80, 160, 64)
transfer_up[128:256] = np.linspace(160, 255, 128)
```

2. Apply the transfer function

```
im_shop_summer = im_shop.copy()
im_shop_summer[:,:,0] = transfer_up[im_shop_summer[:,:,0]]
im_shop_summer[:,:,2] = transfer_down[im_shop_summer[:,:,2]]
```

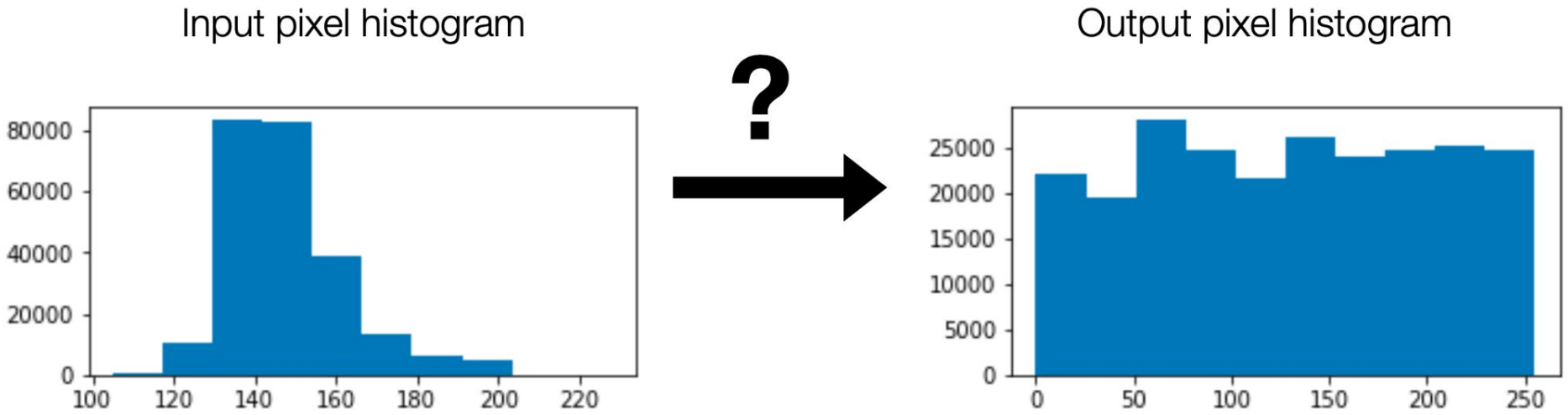
Even a more non-linear transformation needed

- Too many gray pixels → not much contrast



Would additive or subtractive functions work?

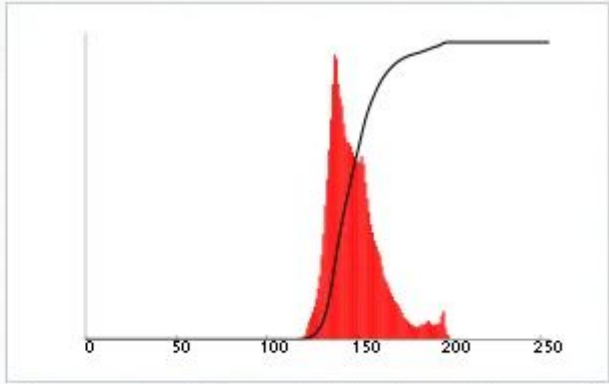
We want to cover the all the brightness levels



We can design a lookup table to achieve this.

Cumulative distribution function (CDF)

Before Histogram Equalization



Corresponding histogram (red) and cumulative histogram (black)

For each value in a set x, CDF computes the probability that x is greater than all other values.
We can use this as a lookup table (with some transformations) to uniformize the histogram.

Image

pixel value	count
50	10
100	20
255	5

(N pixels in total)

?? pixels with value≤ K

pixel value	cumulative
50	10
100	30
255	35

```
unique_id, unique_count = np.unique(im_desert, return_counts=True)
```

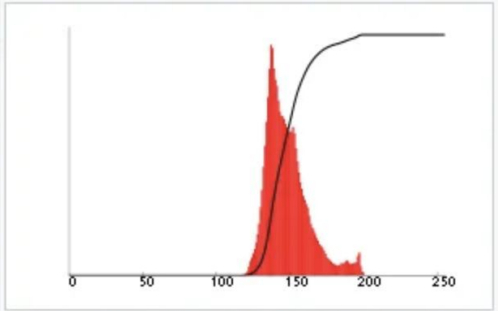
```
im_desert_cdf = np.cumsum(unique_count)
```

$$(x - \text{cdf}_{min}) * \frac{255}{N - \text{cdf}_{min}}$$

pixel value	New value
50	0
100	??
255	255

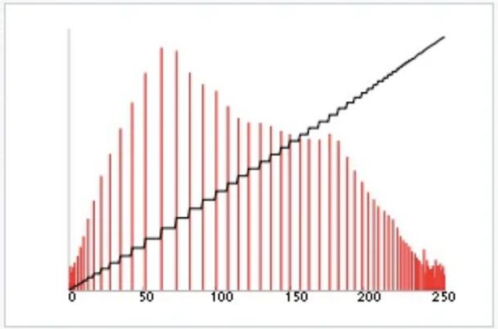
(30-10)/(35-10)*255=219

Before Histogram Equalization



Corresponding histogram (red) and cumulative histogram (black)

After Histogram Equalization



Corresponding histogram (red) and cumulative histogram (black)

We covered pixel level operations - any questions?

90	3	241	65	213
242	143	122	71	8
203	103	45	167	240
29	239	2	93	218
243	242	183	54	123

- 0D linear function (pixel) $ax + b$
- 1D linear function (color) $\begin{bmatrix} 0.19 & 0.77 & 0.39 \\ 0.17 & 0.69 & 0.35 \\ 0.13 & 0.53 & 0.27 \end{bmatrix} \begin{bmatrix} x_R \\ x_G \\ x_B \end{bmatrix}$
- Any function (transfer function)

Input	Output
0	0
…	…
100	50
…	…
255	127

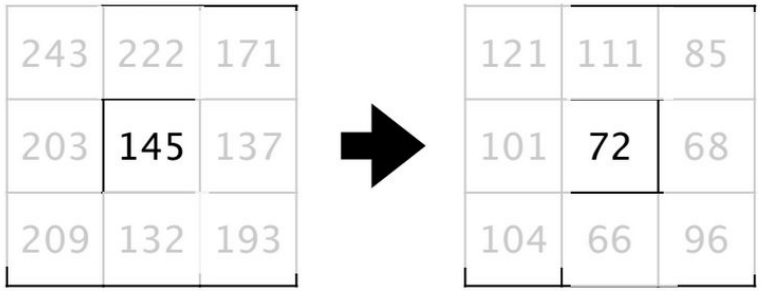
When pixel operations are not enough?

Clean image

Real-world image

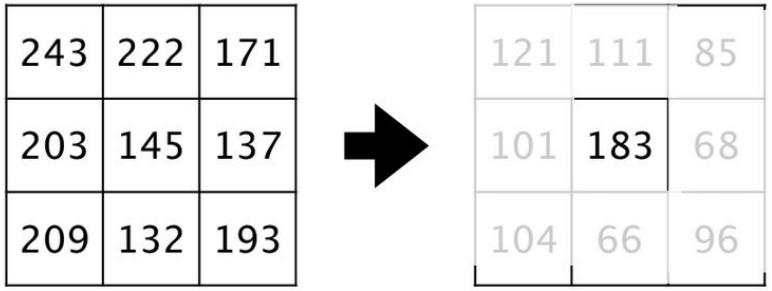
Is there any pixel operation that can “fix” this image?

Need to widen the input of our transformation functions



$$f : p \rightarrow p'$$

Input pixel Output pixel



$$f : pa \rightarrow p'$$

Input **patch** Output pixel

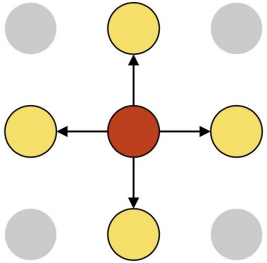
Brief detour on neighborhoods and patches

To denote patches - we need to denote “neighborhoods”:

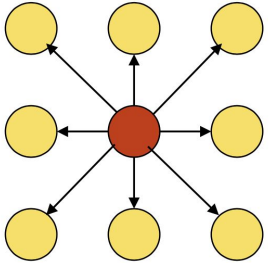
1. Impose topological structure e.g.. “Pixels lie on a grid graph”
   a. 4 or 8 cell topologies.
2. Define adjacency
   a. Adjacency is the set of nodes that are connected.
3. Define a path
   a. Path is a series of pixels that are sequentially adjacent.
4. Define distance
   a. The distance between two pixels is the length of the shortest path.
   b. Ex. D4 or D8 distances.

A patch of radius r around a pixel is the set of all of pixels that are less than a r-distance away.

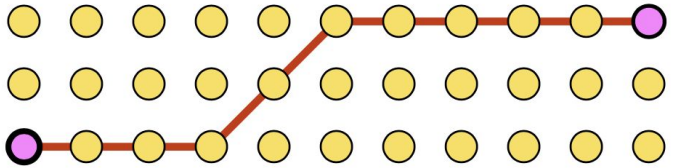
4 connected



8 connected

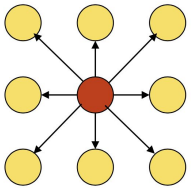


D8 distance



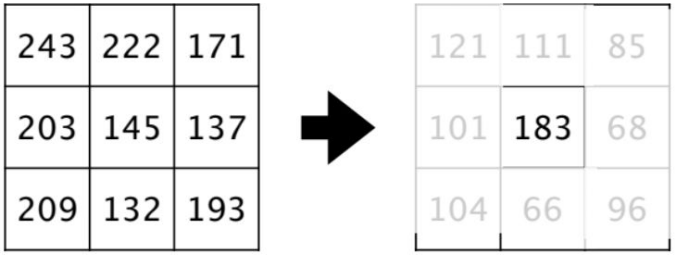
D8 patch of radius 2

8 connected



2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

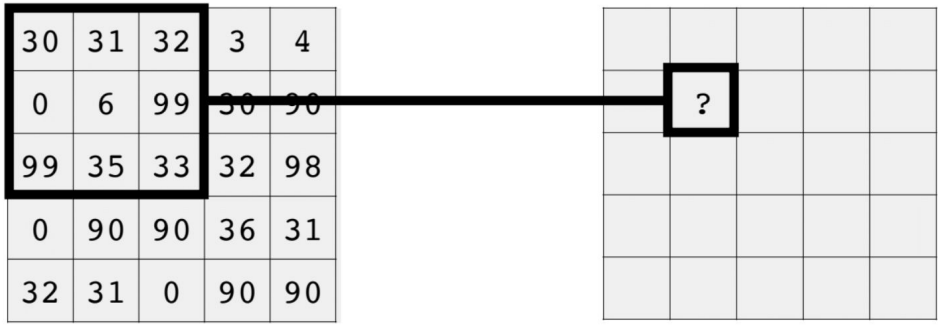
Example: median filter on patch size 1 (in d8)



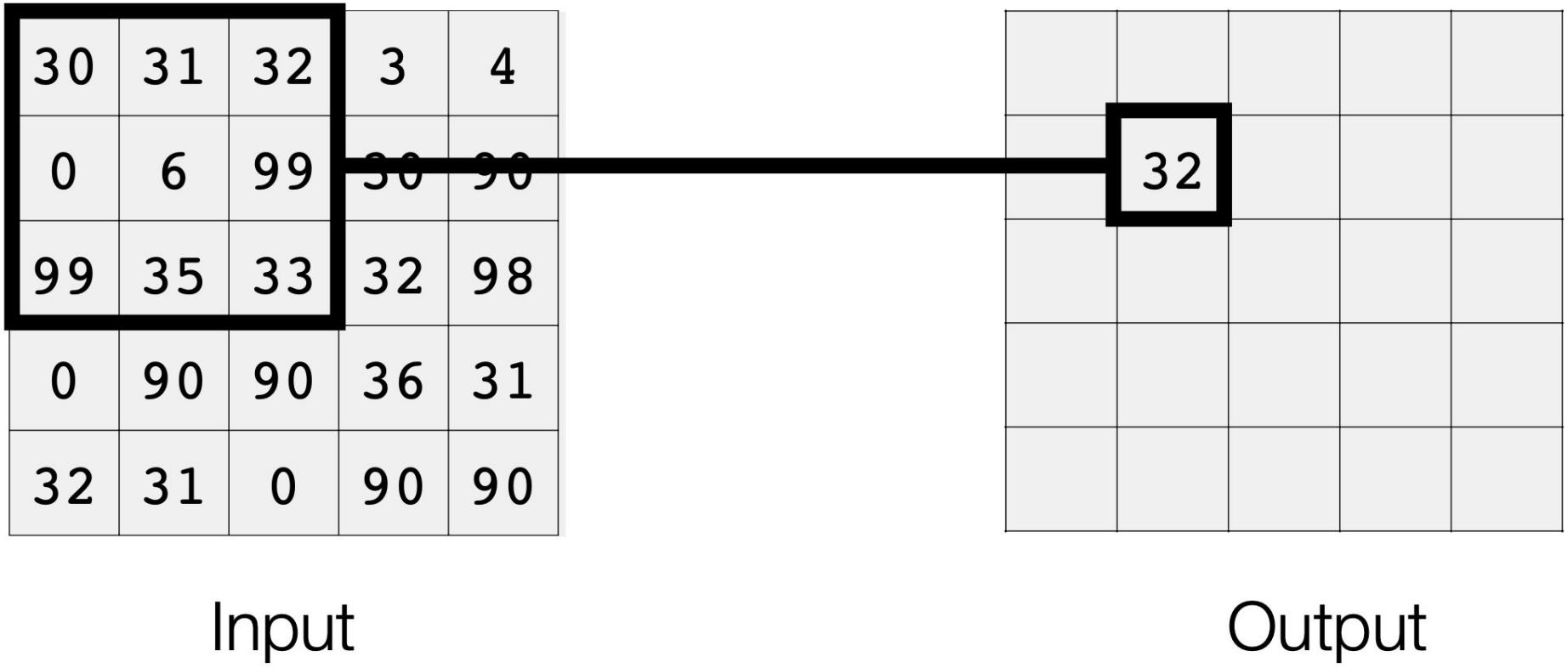
$$f : pa \rightarrow p'$$

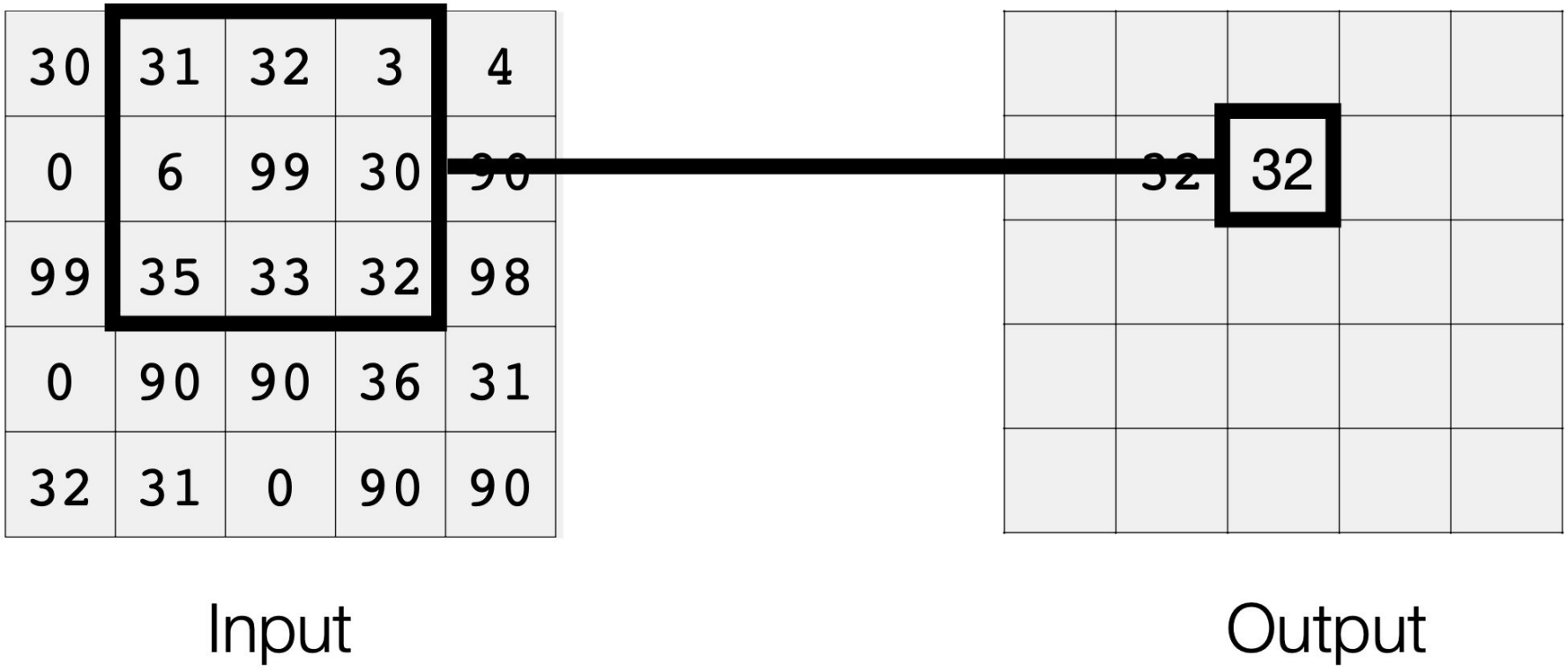
Input **patch** Output pixel

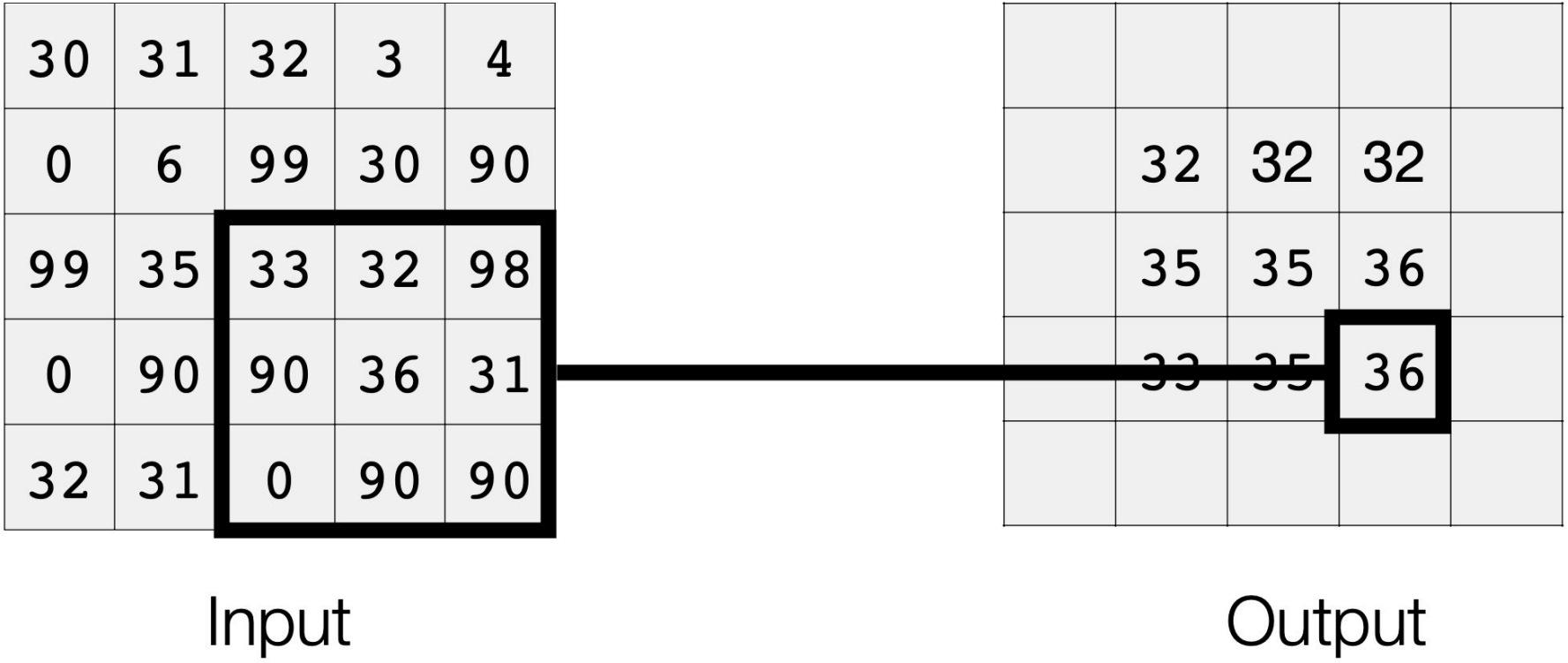
What's the output?



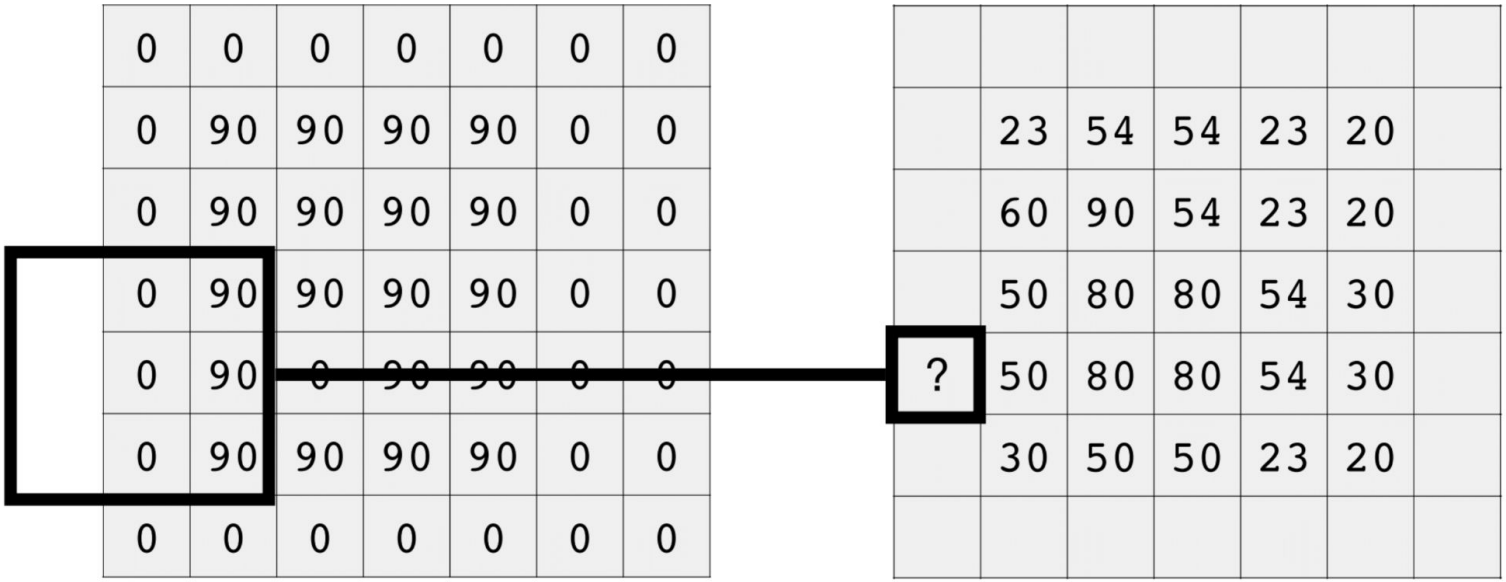
$$p' = \text{median}(p)$$


30 31 32 3 4
0 6 99 30 90
99 35 33 32 98
0 90 90 36 31
32 31 0 90 90
32
Input
Output

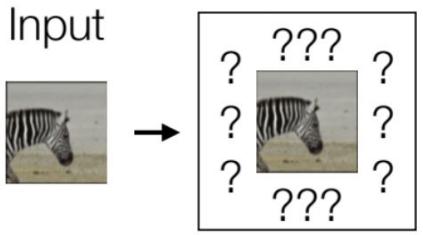

30 31 32 3 4
0 6 99 30 90
99 35 33 32 98
0 90 90 36 31
32 31 0 90 90
32 32
Input
Output

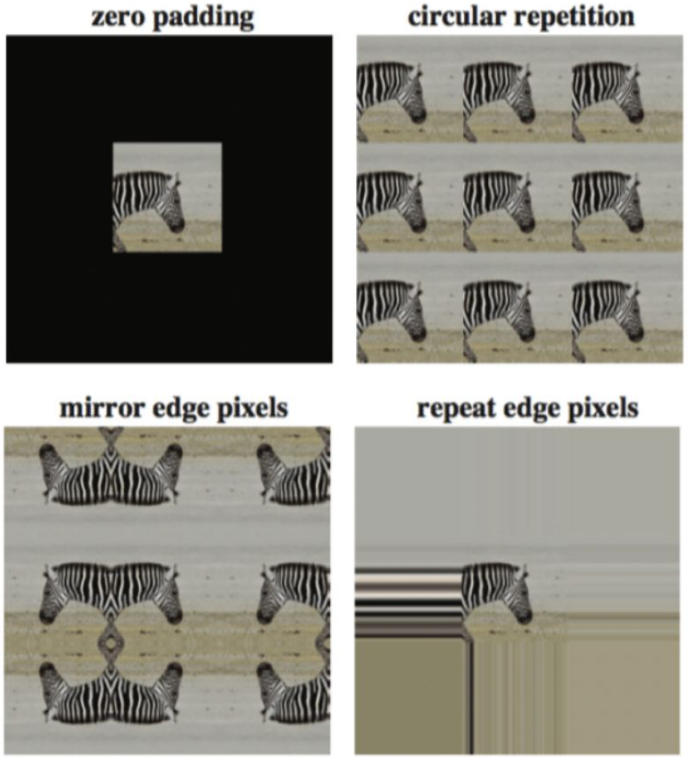

30 31 32 3 4
0 6 99 30 90
99 35 33 32 98
0 90 90 36 31
32 31 0 90 90
32 32 32
35 35 36
33 35 36
Input
Output

Brief issue detour: patch overflows out of image



Solution: padding





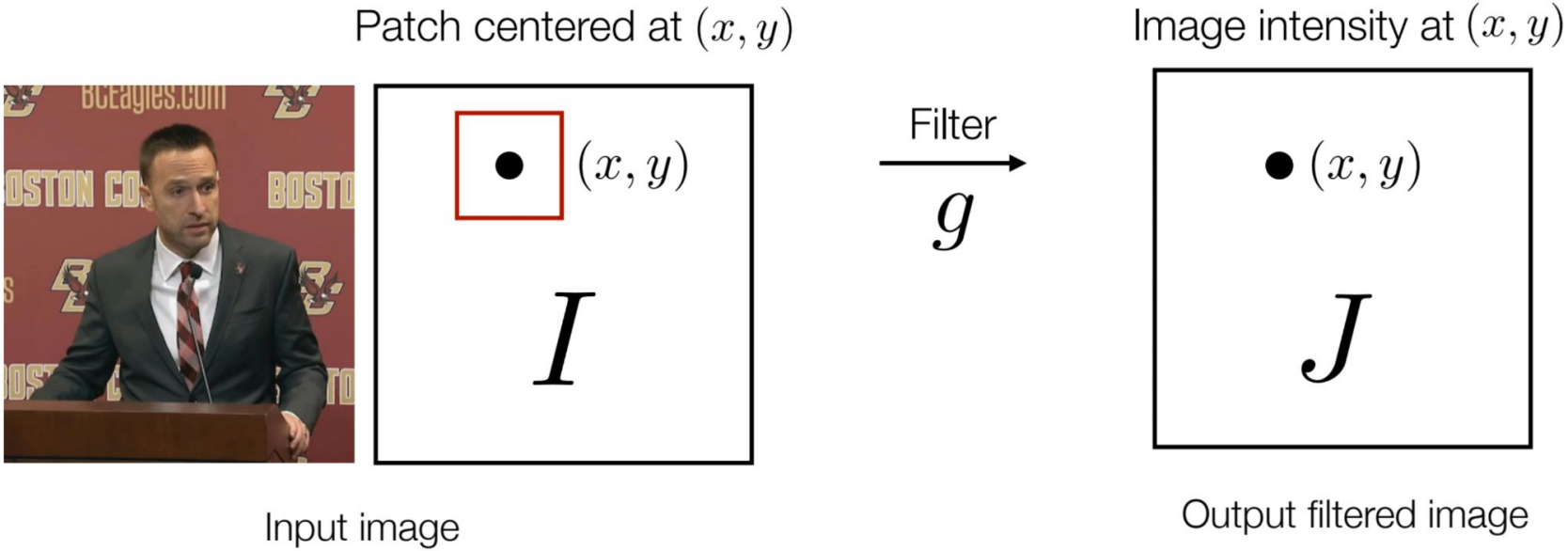
Median filter is a non-parametric filter but it’s non-linear

Ideal image

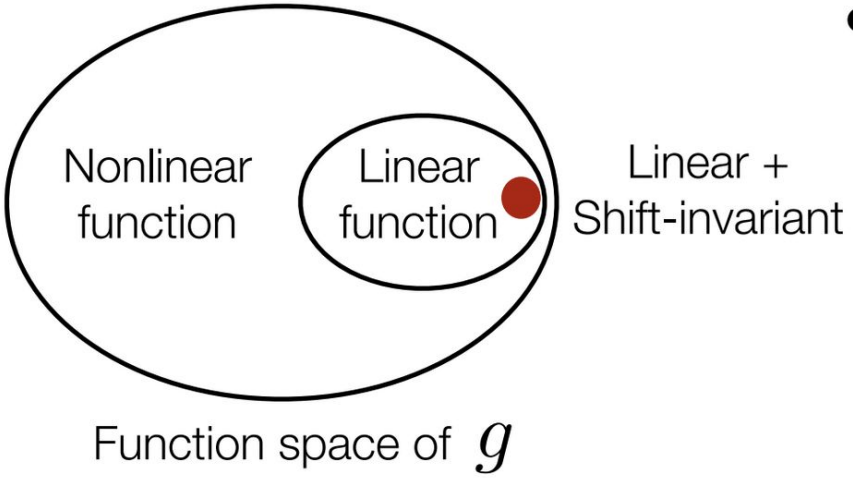
Real-world image

Median filter result

What are some other examples of filters?



Linear vs. non-linear filters



- Linear Function

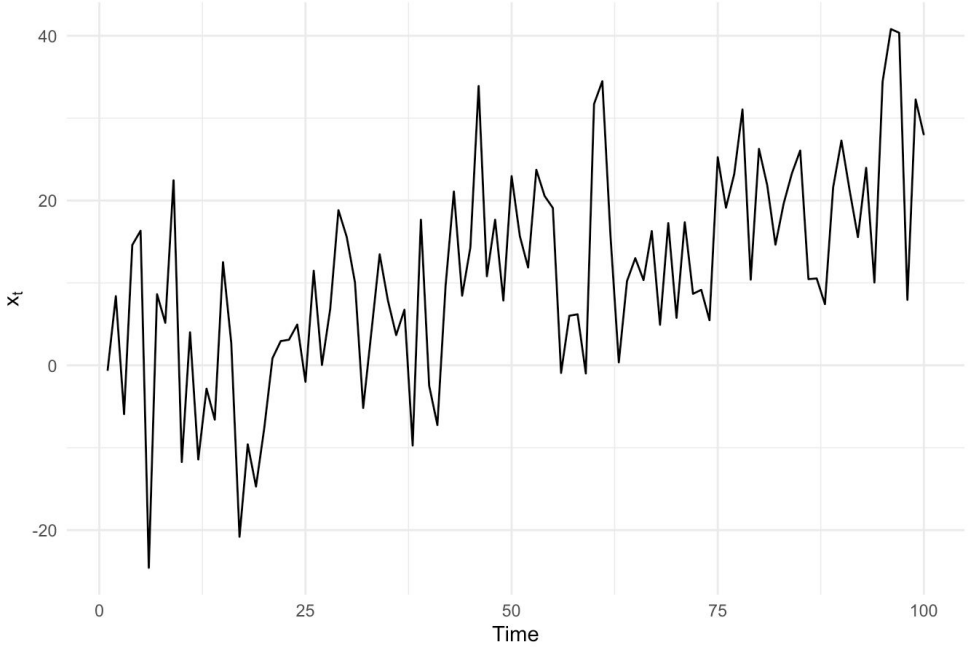
$$g(\alpha I_1 + \beta I_2) = \alpha g(I_1) + \beta g(I_2)$$

- Shift invariance

Same function g for all image patches

1D Linear filtering example

Input x

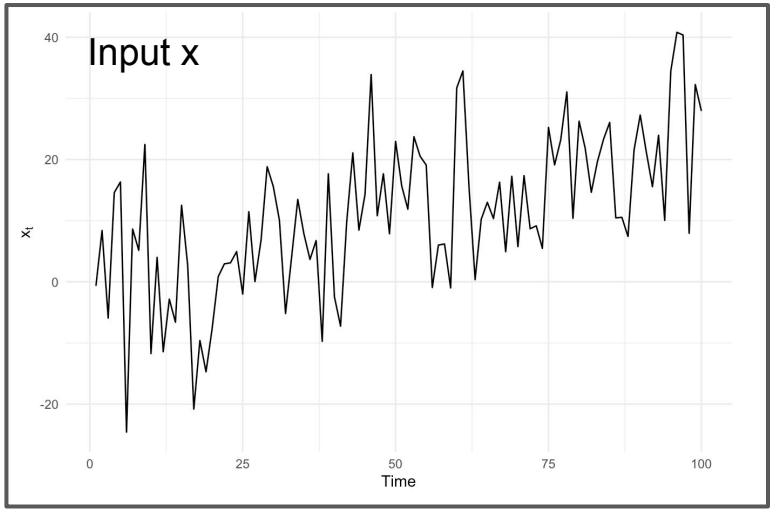


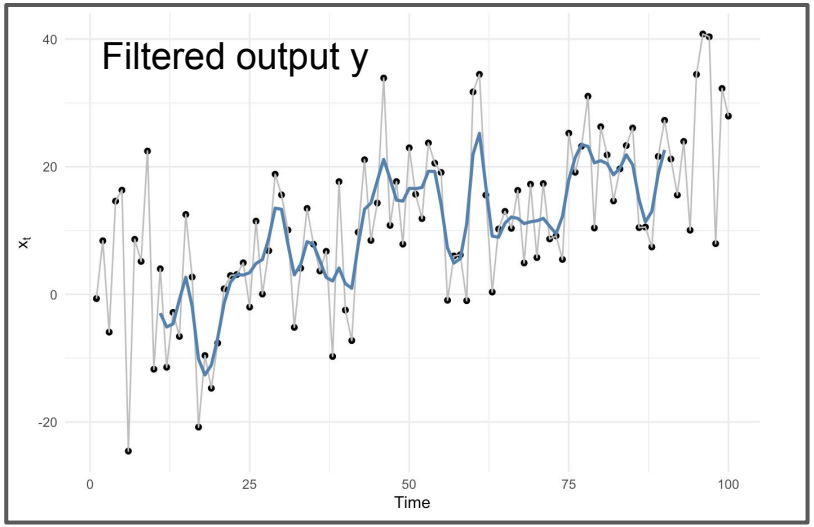
Filtered 1D data

$$y_t = \sum_{j=-\infty}^{\infty} \beta_j x_{t-j}$$

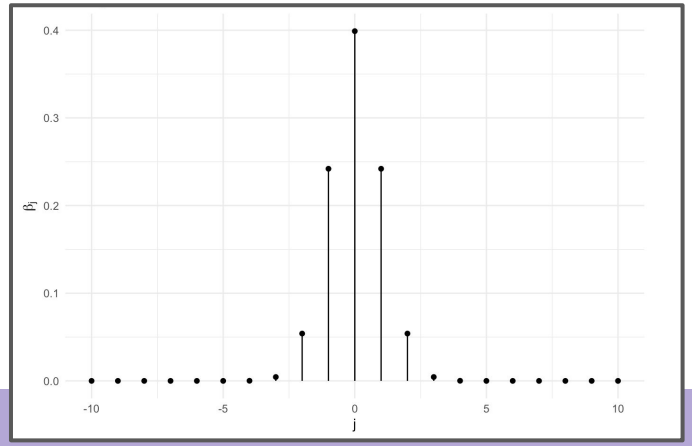
B_j : filter parameters

Example: Low pass filter



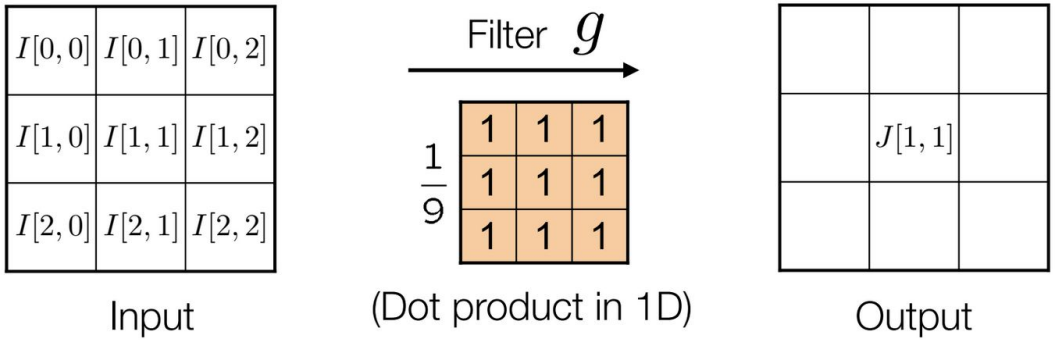


B_j : filter parameters



Simple linear box filter in 2D (moving average)

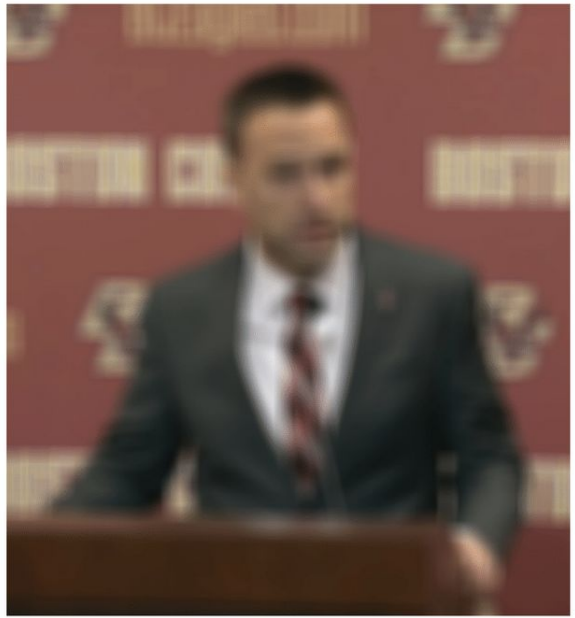
$I[x, y]$: Image intensity at pixel (x,y)



$$J[1,1] = \frac{1}{N} \sum_{x,y} I[x,y]$$

Box filter: blur effect

Input Image



Output: Filtered image
(mimic focus blur)

Basic filters can help computers “perceive” edges, corners and denoise errors

$\frac{1}{K^2}$

1	1	⋯	1
1	1	⋯	1
⋮	⋮	1	⋮
1	1	⋯	1

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$\frac{1}{256}$

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

$\frac{1}{8}$

−1	0	1
−2	0	2
−1	0	1

$\frac{1}{4}$

1	−2	1
−2	4	−2
1	−2	1

$\frac{1}{K}$

1	1	⋯	1

$\frac{1}{4}$

1	2	1

$\frac{1}{16}$

1	4	6	4	1

$\frac{1}{2}$

−1	0	1

$\frac{1}{2}$

1	−2	1

(a) box, $K = 5$

(b) bilinear

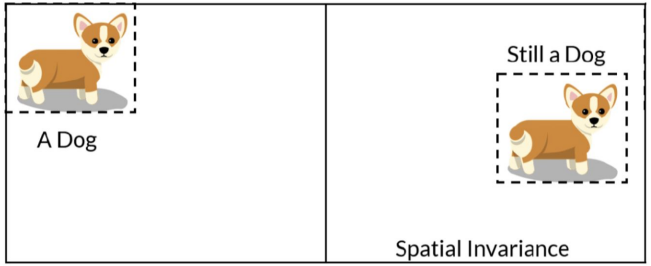
(c) “Gaussian”

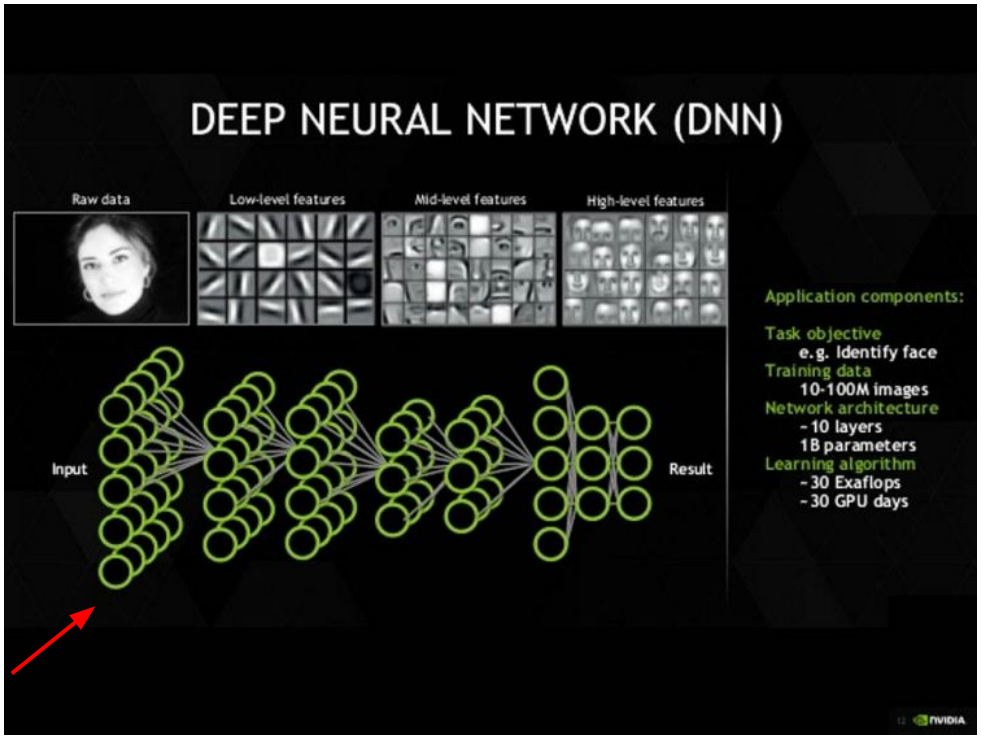
(d) Sobel

(e) corner

More non-linear filters are required for higher order recognition

Deep learning models (built upon filters) need to be shift-invariant.





We are here

Next: code lab

Next lecture - more filtering + convolutions + geometric transformations