Homework 7

due Wed Nov 2 at start of class

`Homework7.pdf`

`HW7.ipynb` or `HW7.py`

download from Brightspace

`Images.zip`

Images and Signals

and some basics of image and signal processing

PIL

Pillow - an updated fork of PIL (Python Imaging Library)

https://pillow.readthedocs.io/en/stable/

opening with PIL

```
from PIL import Image
```

```
im = Image.open('Jordingray.bmp')
```

object with image attributes and methods

```
im.show()
```

opens image in a new window

```
from IPython.display import display
```

```
display(im)
```

opens image in Jupyter Notebook

manipulating images with PIL

```
im.thumbnail((100,100))
```

create a thumbnail
(manipulates image itself)

```
im2 = im.rotate(45)
```

rotates images
(returns new image)

https://pillow.readthedocs.io/en/stable/reference/Image.html

more manipulating images with PIL

```
from PIL import ImageEnhance

im2 = ImageEnhance.Contrast(im)
display(im2.enhance(4))

im2 = ImageEnhance.Brightness(im)
display(im2.enhance(.4))

im2 = ImageEnhance.Sharpness(im)
display(im2.enhance(2.0))
```

https://pillow.readthedocs.io/en/stable/reference/ImageEnhance.html

saving images with PIL

```
im = Image.open('Jordingray.bmp')

im2 = im.rotate(45, expand=True)

im2.save('Jordingray_Rot45.jpg', format='JPEG')
```

https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html

extracting numpy data from images

```
im = Image.open('Jordingray.bmp')

imdata = np.asarray(im)

print(type(imdata))            <class 'numpy.ndarray'>
print(type(imdata[0,0]))       <class 'numpy.uint8'>
print(imdata.shape)            (512, 512)
```

create an image from a numpy array

```
im2 = Image.fromarray(imdata)

im2.save('Jordingray_New.jpg', format='JPEG')
```

manipulating image data in numpy arrays

unsigned int arithmetic

255 = 11111111

decimal binary

$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 255$

$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$

manipulating image data in numpy arrays

`255`	`+`	`1`	`=`	`?`
uint8		uint8		uint8

manipulating image data in numpy arrays

255	+	1	=	0	**Python**
uint8		uint8		uint8	

Matlab

```
Command Window
>> uint8(255) + uint8(1)

ans =

  uint8

   255

>>
```

manipulating image data in numpy arrays

255	+	2	=	1	**Python**
uint8		uint8		uint8	

Matlab

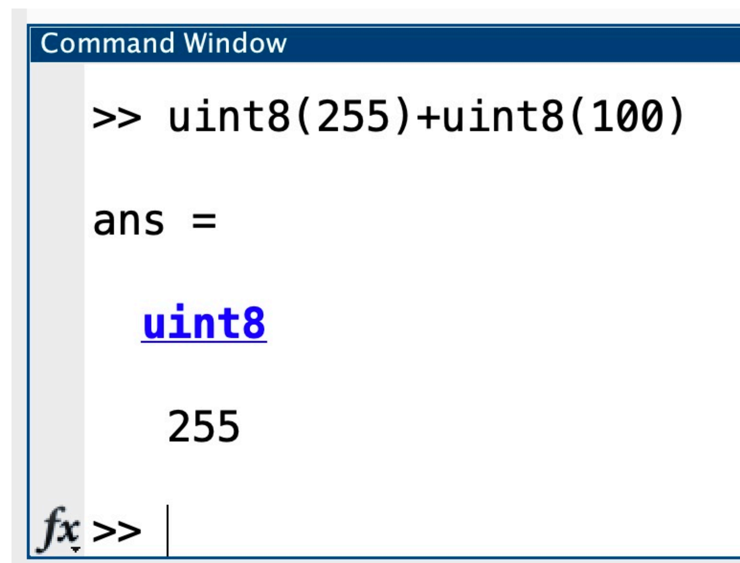
Command Window

```
>> uint8(255)+uint8(2)

ans =

  uint8

   255

>>
```

manipulating image data in numpy arrays

255	+	100	=	99		**Python**
uint8		uint8		uint8		

Matlab



manipulating image data in numpy arrays

```
imdata = np.asarray(im)

imdata2 = imdata + 150
```

manipulating image data in numpy arrays

```
imdata = np.asarray(im)

imdata2 = imdata + 255
```

manipulating image data in numpy arrays

```
imdata = np.asarray(im)

imdata2 = imdata + 1000
```

?

manipulating image data in numpy arrays

```
imdata = np.asarray(im)
```

```
   uint16         uint8       uint16
imdata2 = imdata + 1000
imdata2
```

```
array([[1091, 1092, 1093, ..., 1106, 1109, 1109],
       [1084, 1085, 1087, ..., 1110, 1110, 1110],
       [1085, 1083, 1082, ..., 1108, 1108, 1109],
       ...,
       [1169, 1175, 1179, ..., 1184, 1184, 1187],
       [1171, 1177, 1178, ..., 1184, 1185, 1185],
       [1173, 1177, 1178, ..., 1184, 1184, 1184]], dtype=uint16)
```

manipulating image data in numpy arrays

```
imdata = np.asarray(im)
```

```
   uint16          uint8      uint16
imdata2 = imdata + 1000
imdata2
```

```
array([[1091, 1092, 1093, ..., 1106, 1109, 1109],
       [1084, 1085, 1087, ..., 1110, 1110, 1110],
       [1085, 1083, 1082, ..., 1108, 1108, 1109],
       ...,
       [1169, 1175, 1179, ..., 1184, 1184, 1187],
       [1171, 1177, 1178, ..., 1184, 1185, 1185],
       [1173, 1177, 1178, ..., 1184, 1184, 1184]], dtype=uint16)
```

im2 = Image.fromarray(imdata2)

im2.save('tmp.jpg', format='JPEG') **Error**

converting to/from floats

```
np.uint8(np.float64(np.uint8(255)+np.uint8(100)))
```

```
[140]:  99
```

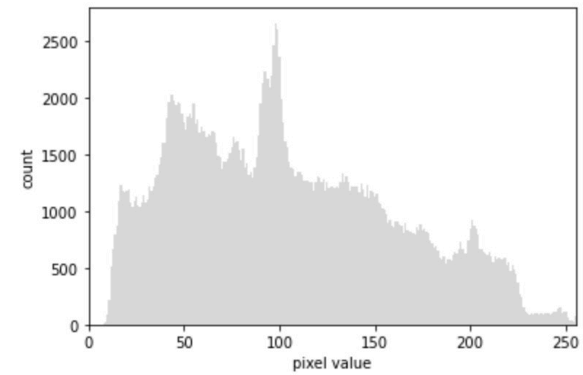
converting to/from floats

```
np.uint8(np.float64(np.uint8(255)+np.uint8(100)))
```

```
[140]: 99
```

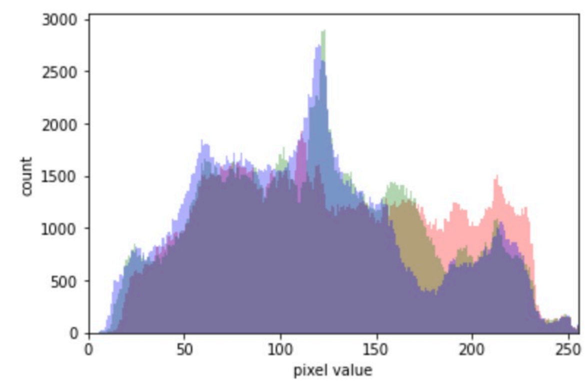
```
imdataf = np.float64(imdata)
imdataf = imdataf*500 + 250
imdata2 = np.uint8((imdataf/np.max(imdataf))*255)
```

image histograms



greyscale image

```
hist = np.array(im.histogram())
pix = np.arange(256)
plt.bar(pix, hist, width=1, alpha=.3, color='grey')
```



color image

```
hist = np.array(im.histogram())
hist = np.resize(hist, (3,256))
pix = np.arange(256)
plt.bar(pix, hist[0,:], width=1, alpha=.3, color='red')
plt.bar(pix, hist[1,:], width=1, alpha=.3, color='green')
plt.bar(pix, hist[2,:], width=1, alpha=.3, color='blue')
```

simple image processing

a very basic operation:

convert an image to a pure black (0) or white (255) image, using a criterion as the cut off

how would we do this?

simple image processing

```
im = Image.open('Jordingray.bmp')
imdata = np.asarray(im)
imdata2 = np.uint8(255*(imdata>C))
```

example of "vectorized" code

```
im2 = Image.fromarray(imdata2)
display(im2)
```

simple image processing

that's a step function ...

what about a more general function?

$$\psi(x) = \gamma + (1-\gamma-\lambda)F(x)$$

$$F(x) = 1 - \exp\left[-\left(\frac{x}{\alpha}\right)^{\beta}\right]$$

should look familiar

how could we modify this?

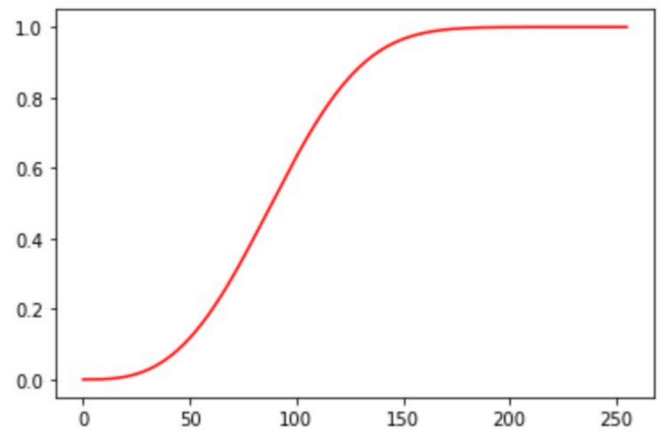
simple image processing

alpha=100

beta=3

gam=0.0

lam=0.0

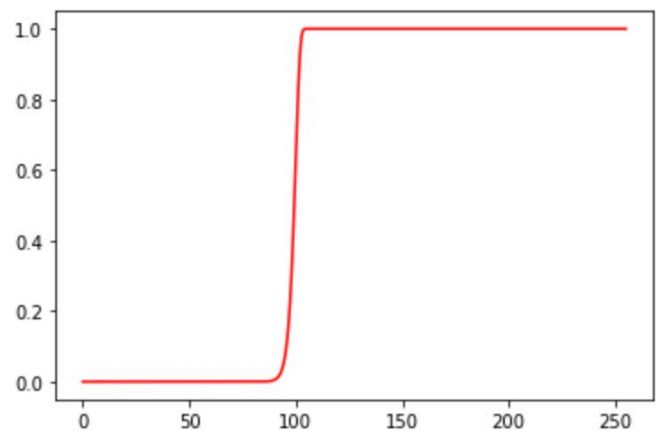


alpha=100

beta=50

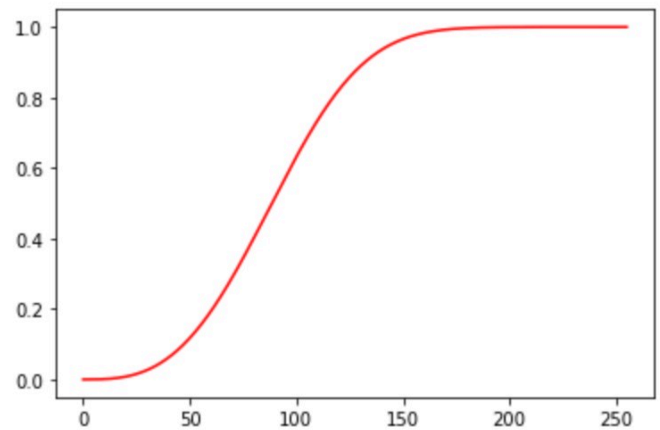
gam=0.0

lam=0.0

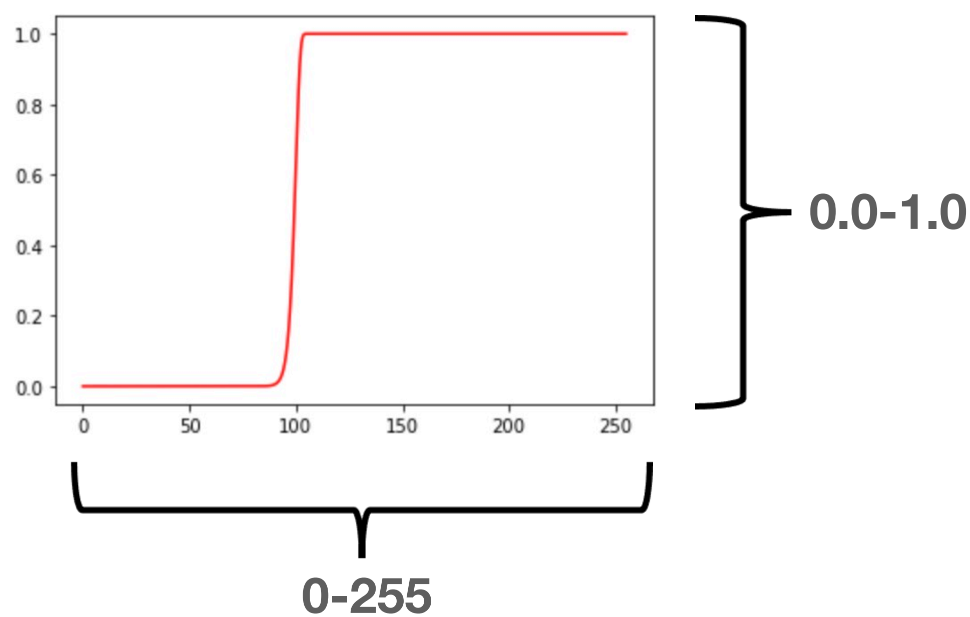


simple image processing

```
alpha=100
beta=3
gam=0.0
lam=0.0
```




```
alpha=100
beta=50
gam=0.0
lam=0.0
```



simple image processing

```
imdata2 = myfilter(imdata, alpha=100, beta=3,
                   gam=0.0, lam=0.0)
```

need to make sure imdata2 is `uint8` **between 0 and 255**

simple image processing

```
imdata2 = myfilter(imdata, alpha=100, beta=3,
                   gam=0.0, lam=0.0)
```

need to make sure imdata2 is `uint8` **between 0 and 255**

```
imdata2 = np.uint8(255*imdata2)
im2 = Image.fromarray(imdata2)
```

original

filtered

simple image processing

```
imdata2 = myfilter(imdata, alpha=150, beta=8,
                   gam=0.4, lam=0.4)
```

need to make sure imdata2 is `uint8` **between 0 and 255**

```
imdata2 = np.uint8(255*imdata2)
im2 = Image.fromarray(imdata2)
```

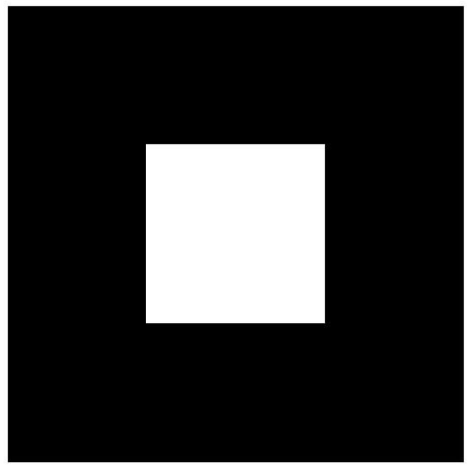
original

filtered

a different transformation applied to different pixels

image masks

x



=

how could we carry out this operation?

image masks

create mask

```
sz = 250
mask = np.zeros((r,c))
mask[int(r/2-sz/2):int(r/2+sz/2),
     int(r/2-sz/2):int(r/2+sz/2)] = 1
```

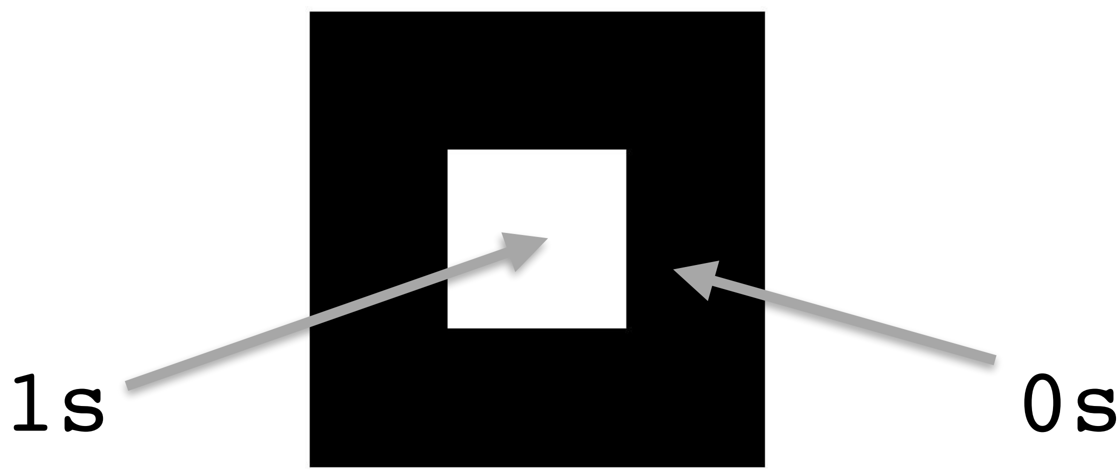


image masks

create mask

```
sz = 250
mask = np.zeros((r,c))
mask[int(r/2-sz/2):int(r/2+sz/2),
     int(r/2-sz/2):int(r/2+sz/2)] = 1

imdata2 = np.uint8(imdata * mask)
```

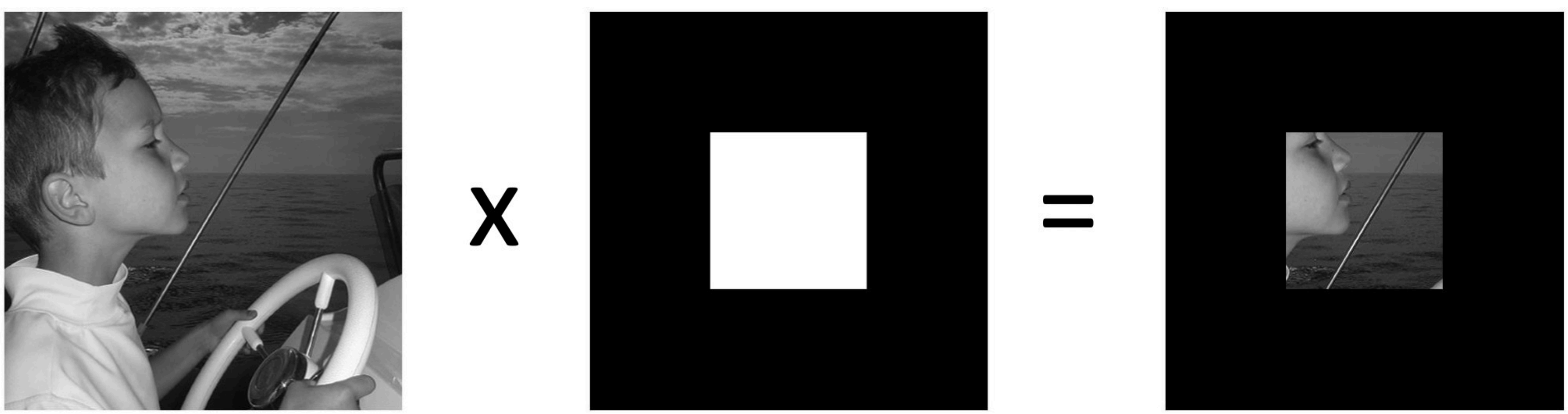
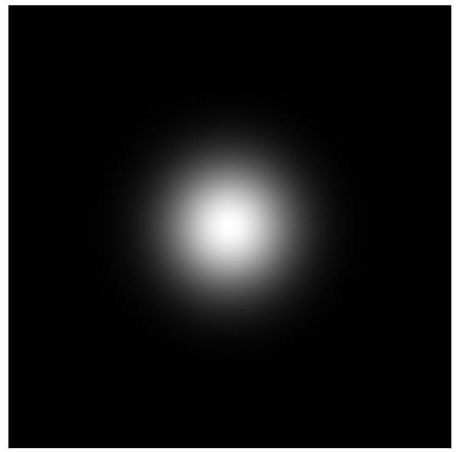
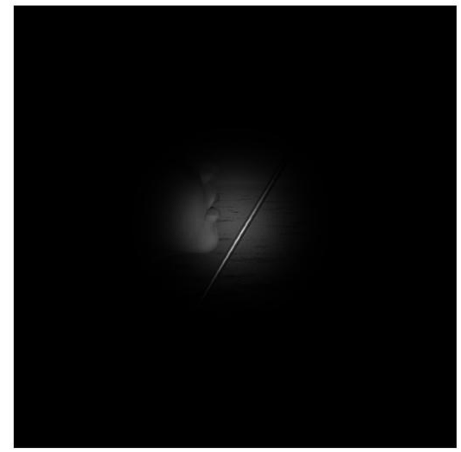


image masks

x



=

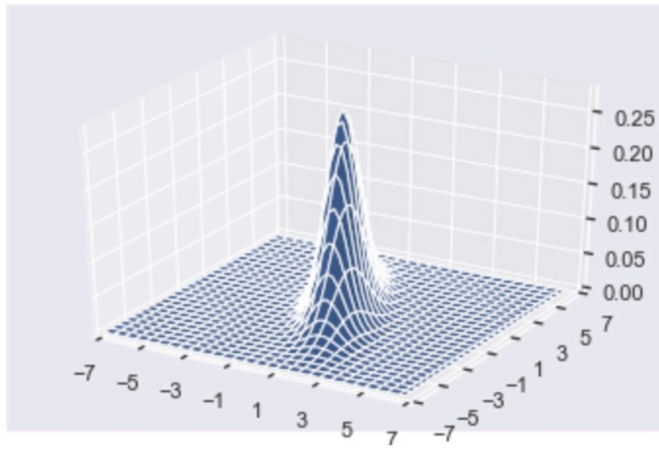


how could we carry out this operation?

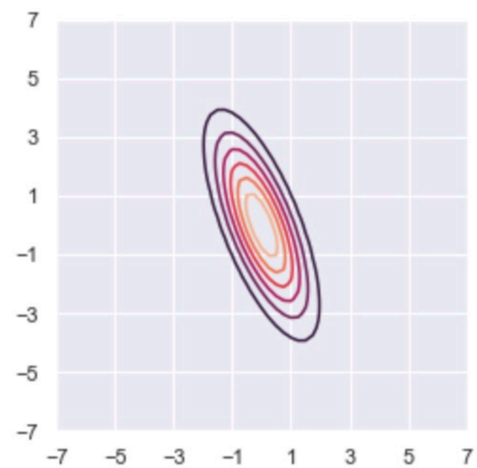
image masks

create mask

```
def my_mvnpdf(x, M, S):
    return (1 / np.sqrt((2 * np.pi * np.linalg.det(S)))) *
    np.exp(-(1/2) * (np.transpose(x-M) @ np.linalg.inv(S) @
    (x-M)))
```




```
mx = r/2; my = c/2
sx = 50; sy = 50; rho = 0.0
mu = np.array([mx, my])
sig = np.array([[sx**2,     rho*sx*sy],
                [rho*sx*sy, sy**2]])
```



basically, same code from before (with 3D plots)

image masks

create mask

```
inc = 1
xsteps = np.arange(0, r, 1)
ysteps = np.arange(0, c, 1)
X, Y = np.meshgrid(xsteps, ysteps)
Z = np.zeros(X.shape)

for i in range(len(xsteps)):
    for j in range(len(ysteps)):
        xy = np.array([X[i,j], Y[i,j]])
        Z[i,j] = my_mvnpdf(xy, mu, sig)

mask = Z/np.max(Z)
```

no longer treating this as a probability distribution,
we want the center point intensity equal to 1.0,
do not care that the area under the curve equals 1.0

image masks

apply mask

```
imdata2 = np.uint8(imdata * mask)

im2 = Image.fromarray(np.uint8(imdata2))
display(im2)
```

image processing scripts

```
import os
import re

pattern = r'.*.((jpg)|(JPG))'
dir = './images'
with os.scandir(dir) as entries:
    for entry in entries:
        if re.search(pattern, entry.name):
            fullpath = dir + '/' + entry.name
            im = Image.open(fullpath)
            im.thumbnail((100,100))
            display(im)
```

of course, this could do something more complication than display the images

More on Images and Signals

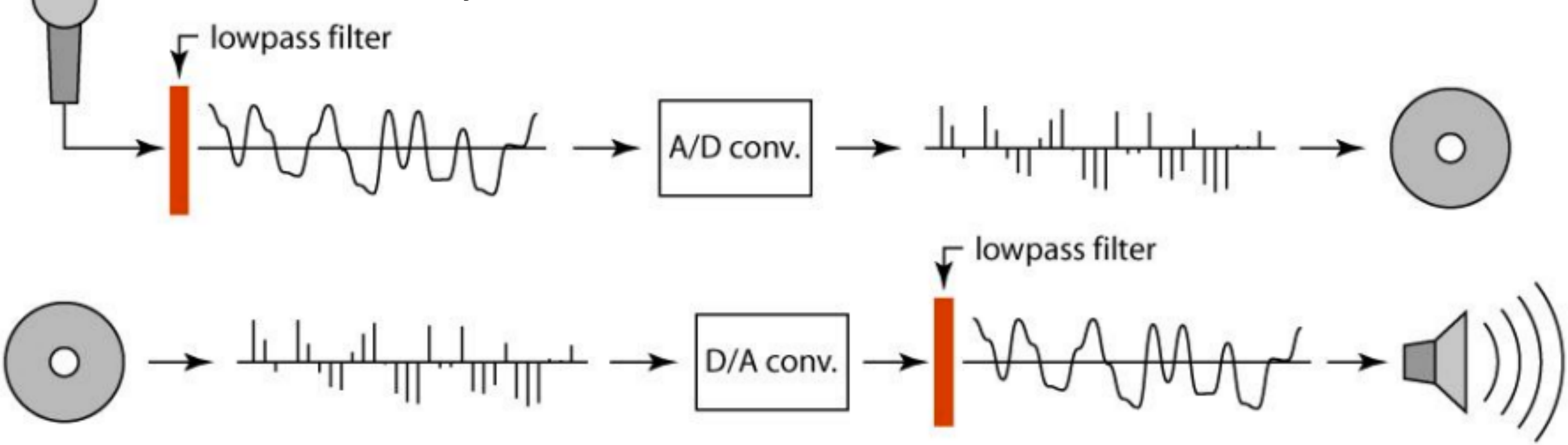
our data structures are discrete in time or space

time = recorded sounds (world), recorded voltages (brain)
space = images (objects, scenes, brains)

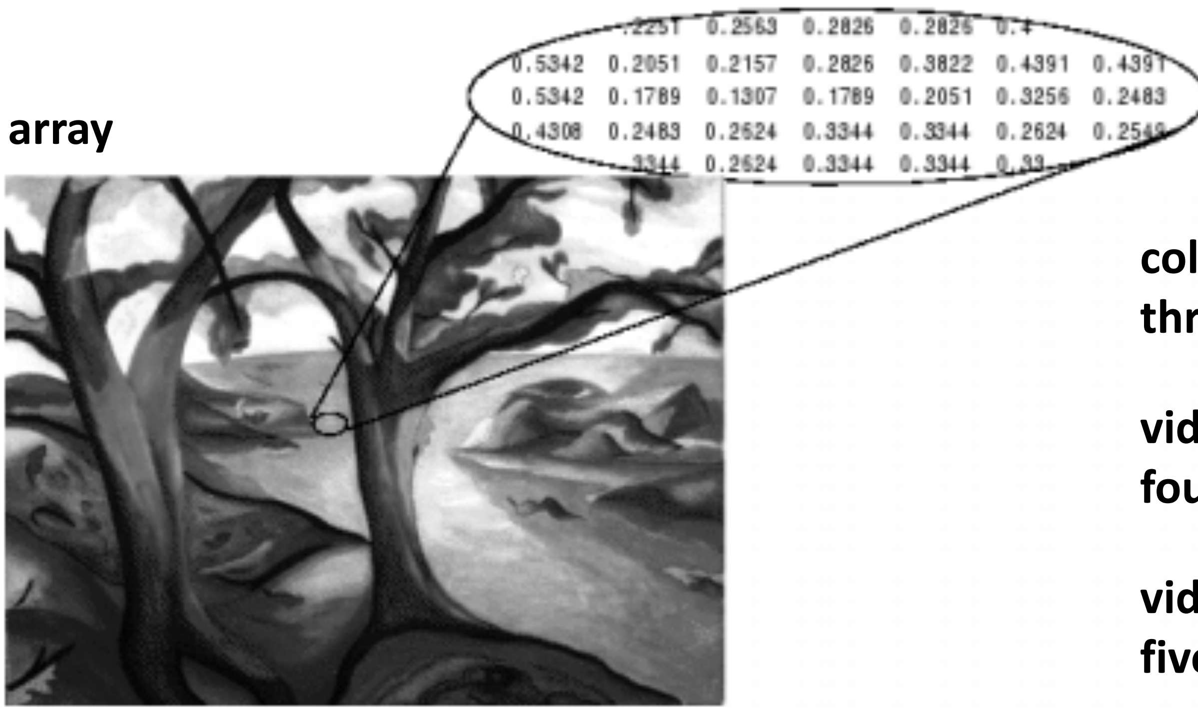
in the real world, time and space are continuous, not discrete

in mathematics, there are continuous and discrete versions

Audio
one-dimensional array



B/W image
two-dimensional array



color image
three-dimensional array

video
four-dimensional array

video + sound
five-dimensional array

key factors that impact quality of digitization

spatial resolution / sampling rate: how close together in space (image/video) or time (audio) samples are taken

bit depth: what range of values (intensities) can be registered depending on the number of bits/bytes devoted to the signal

spatial resolution / sampling rate

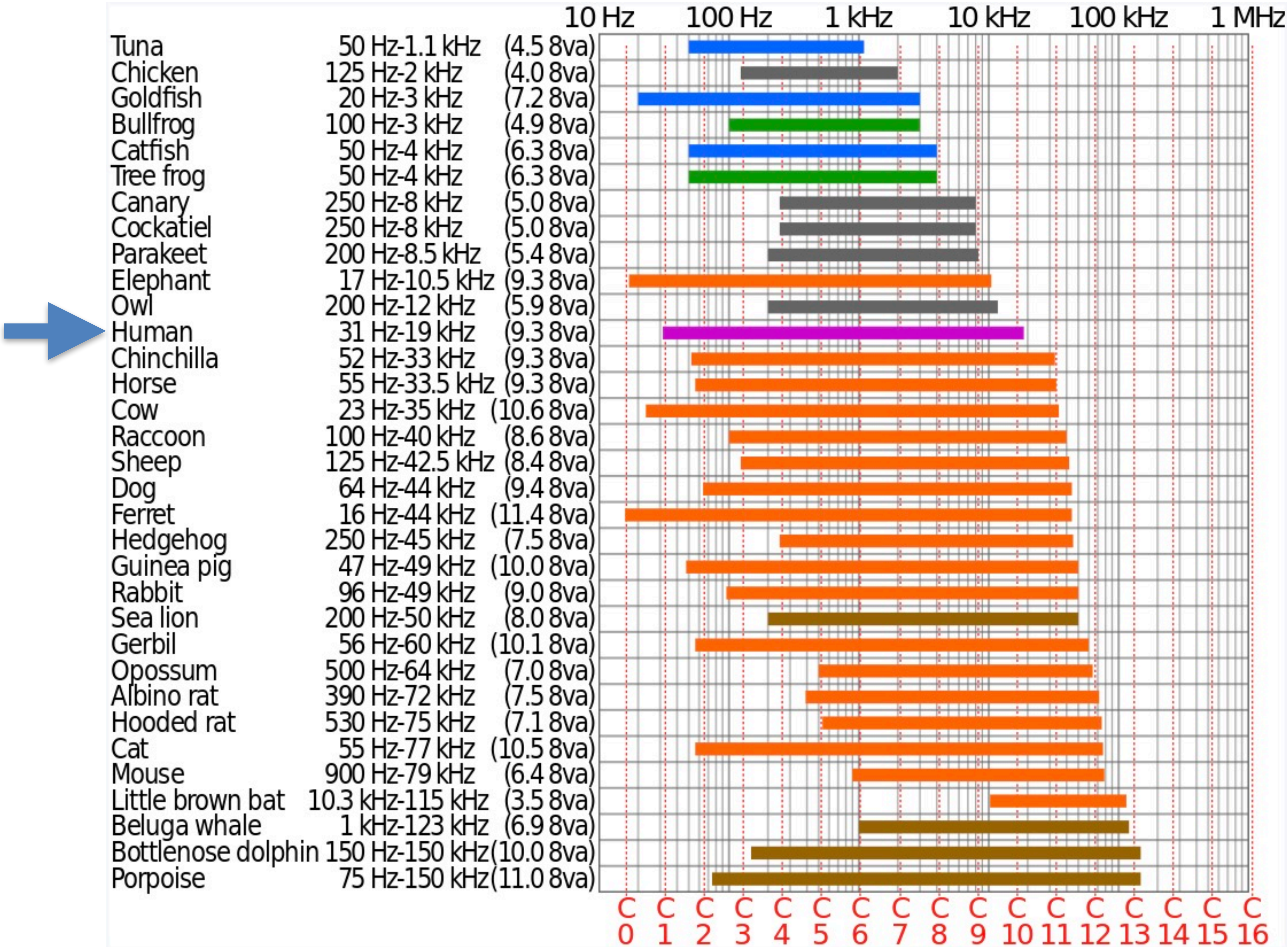
spatial resolution / sampling rate

CD Audio: 44.1kHz* (44,100 samples per second)

HD Audio: 96kHz (96,000 samples per second)

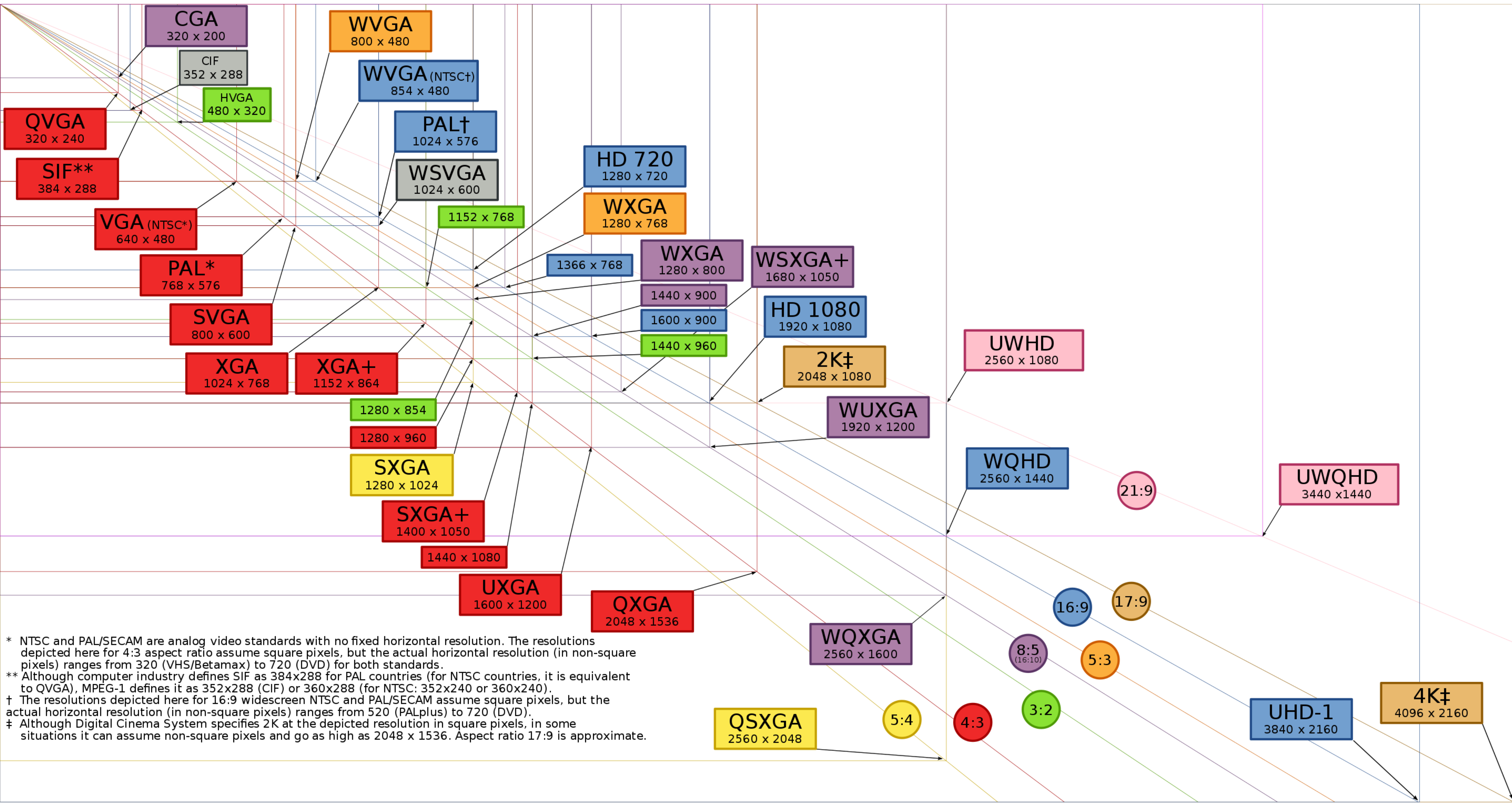
Blu Ray Audio: up to 192kHz

* minimal sampling rate to reproduce 20kHz sound frequencies (general limit of human hearing)



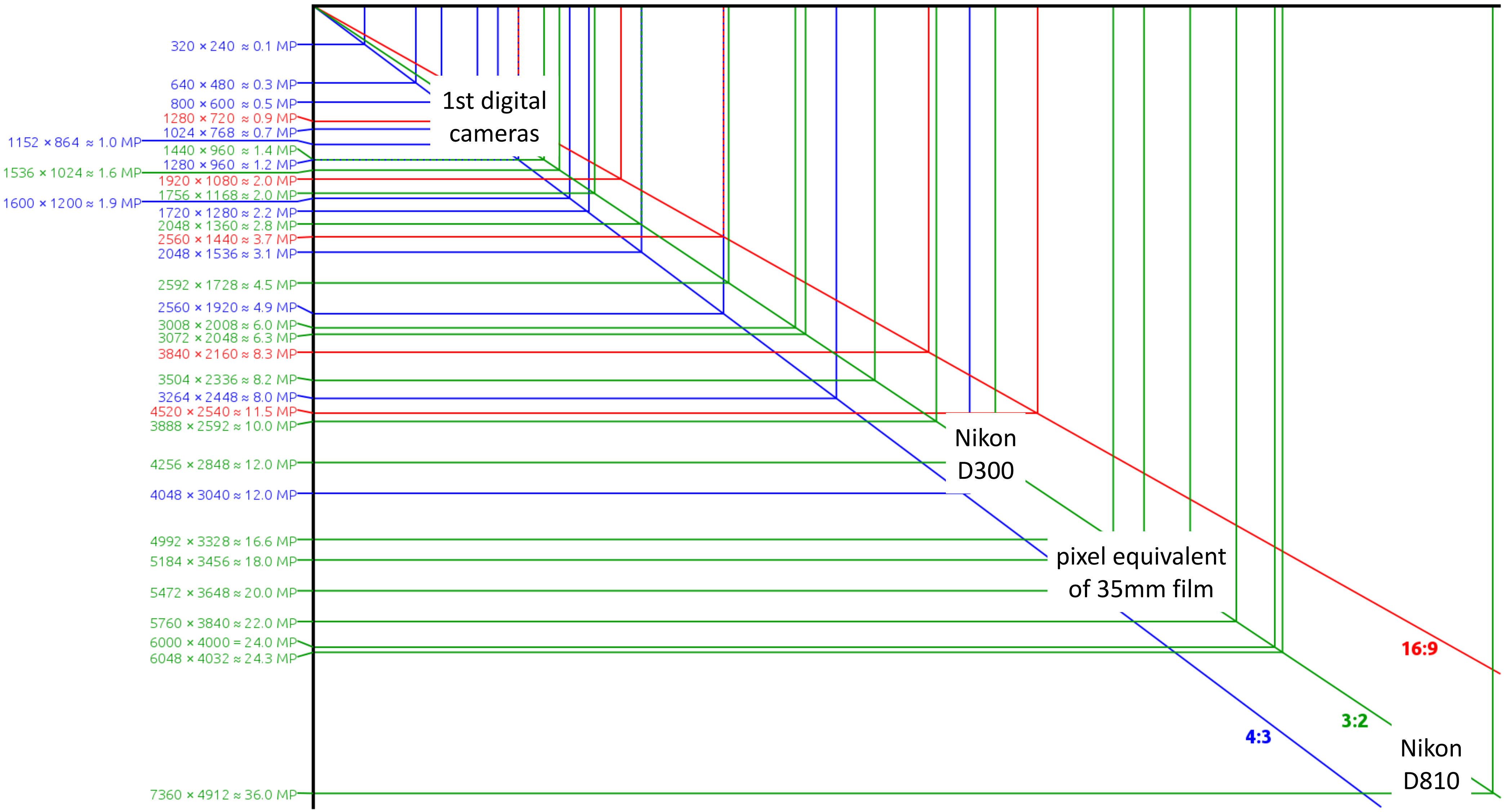
if you were recording or playing sounds for non-human animals
you would want to make sure you had a sufficient sampling rate

spatial resolution / sampling rate



8K is 4x
as large

spatial resolution / sampling rate



bit depth

bit depth

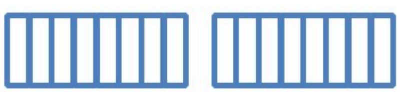
double-precision floating point

single-precision floating point

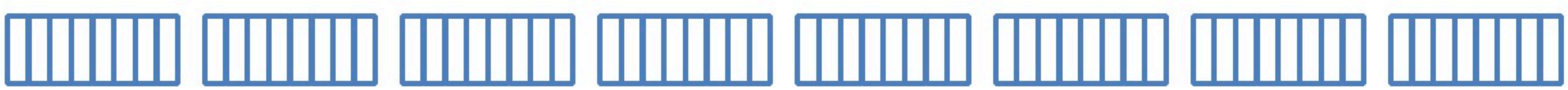
8-bit signed integer

16-bit signed integer

16-bit unsigned integer



64-bit unsigned integer



bit depth (audio)

CD Audio: 16 bits (2 bytes) per sample

DVD/Blu Ray: 24 (3 bytes) per sample

Signal-to-noise ratio and resolution of bit depths

# bits	SNR	Possible integer values (per sample)	Base-ten signed range (per sample)
4	24.08 dB	16	–8 to +7
8	48.16 dB	256	–128 to +127
11	66.22 dB	2048	–1024 to +1023
12	72.24 dB	4096	–2048 to +2047
16	96.33 dB	65,536	–32,768 to +32,767
20	120.41 dB	1,048,576	–524,288 to +524,287
24	144.49 dB	16,777,216	–8,388,608 to +8,388,607
32	192.66 dB	4,294,967,296	–2,147,483,648 to +2,147,483,647
48	288.99 dB	281,474,976,710,656	–140,737,488,355,328 to +140,737,488,355,327
64	385.32 dB	18,446,744,073,709,551,616	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807

bit depth (images)

gif, jpg, tiff	8 bits per channel (R, G, B) - 24 bits total
raw	12 or 14 bits per channel - 36-42 bits total

bit depth (monitors)

1st computers	1 bit (white/black, green/black)	
early NeXT	2 bits	
Atari, Commodore 64	4 bits	used color maps
early VGA	8 bits (total for color)	used color maps
most current monitor	24 bits (R, G, B)	
HDR monitors	30/36/48 bits (R, G, B)	
more than 3 primaries	R, G, B, C, Y, M	