Data Science Survival Skills

Files

Agenda

- What is a file…?
- What kinds of data do we have?
- How is data stored meaningfully?
- What is lossy/lossless compression?
- What are container formats?

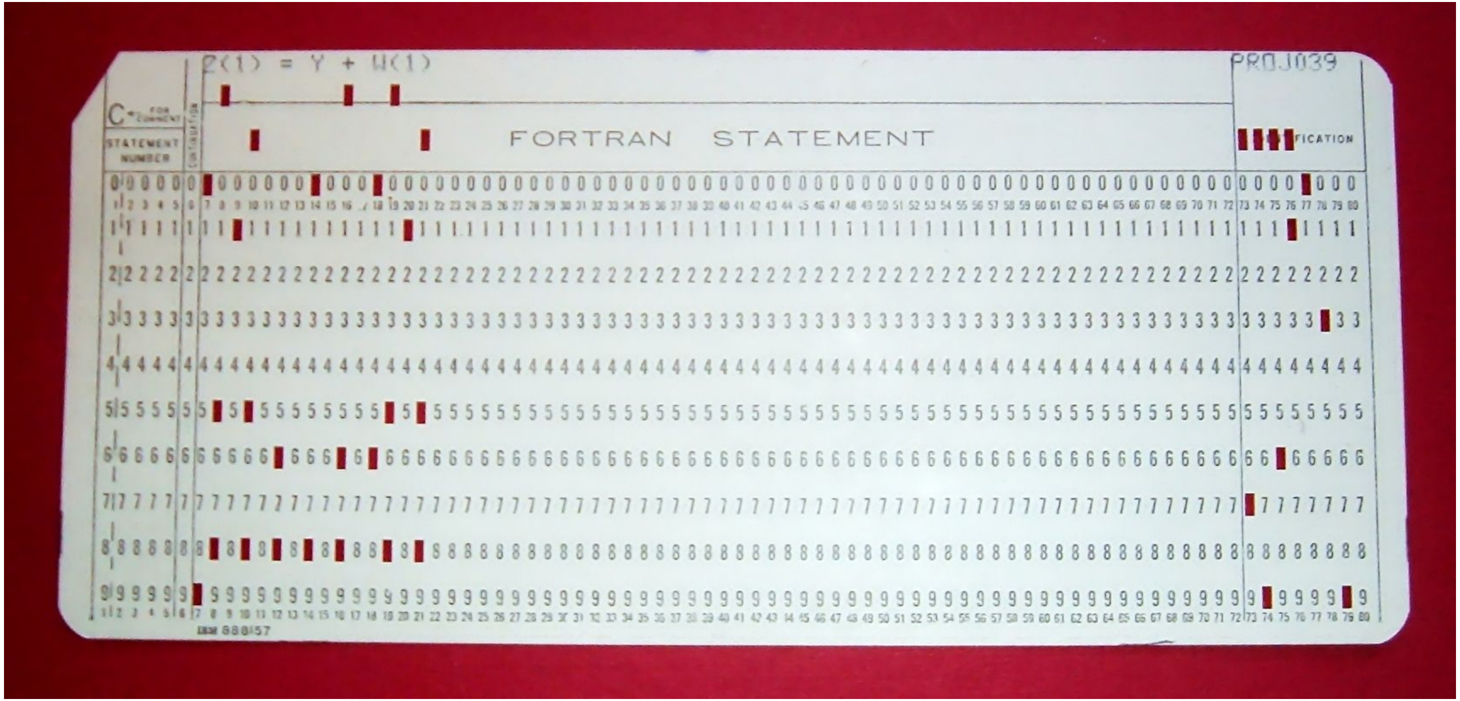
A file or “data”

STORING INFORMATION

What information do we like to store?

A file

- Entity of content
- Back in the days: punch cards



Storing information as bits and bytes

Number:

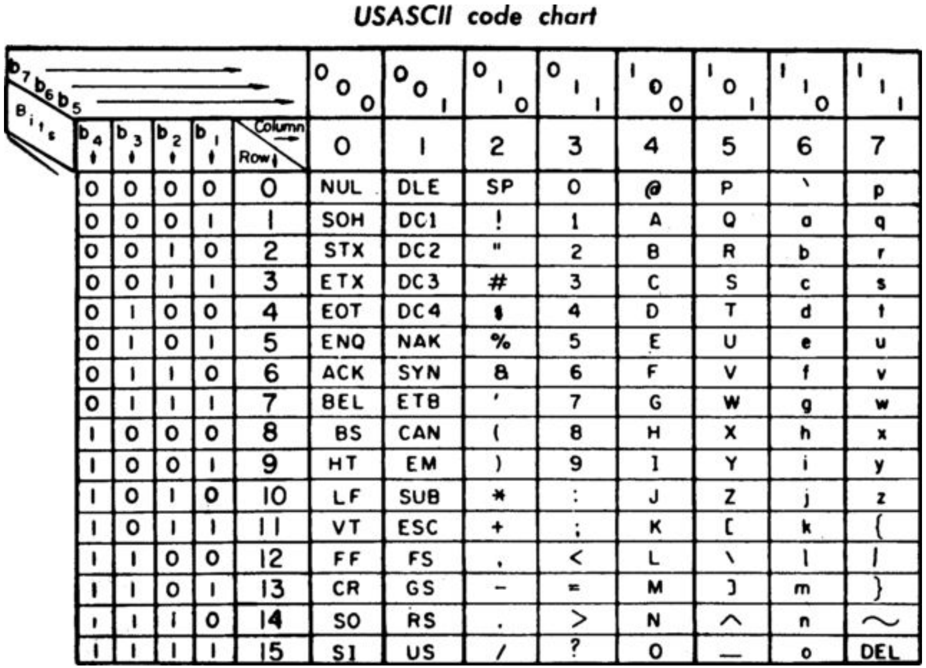
7 Binary: 111

Characters:

DATA Binary →

Pixel values:

Are numbers!!

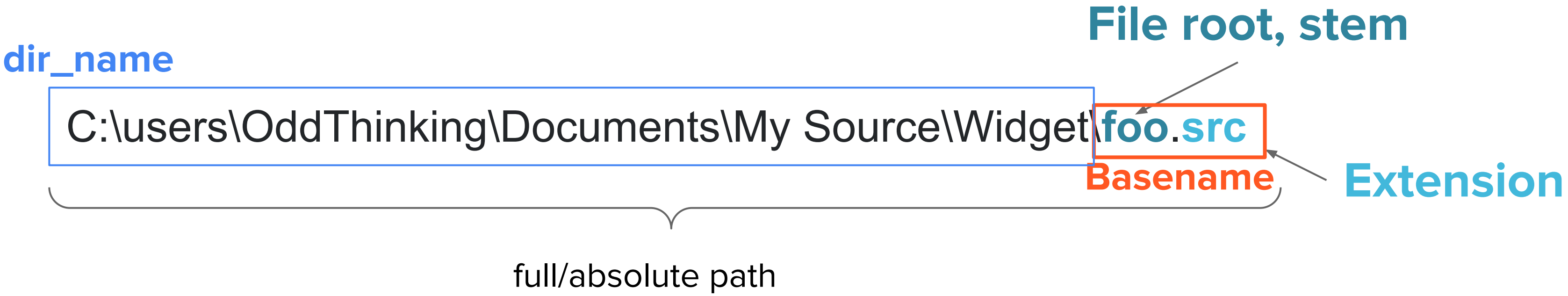


USASCII code chart

b7 b6 b5 → Bits					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4 ↓	b3 ↓	b2 ↓	b1 ↓	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	–	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

File identification

- Root/stem ➜ identifier
- Extension ➜ File type
- Path ➜ Location



dir_name

C:\users\OddThinking\Documents\My Source\Widget\foo.src

File root, stem

Basename

Extension

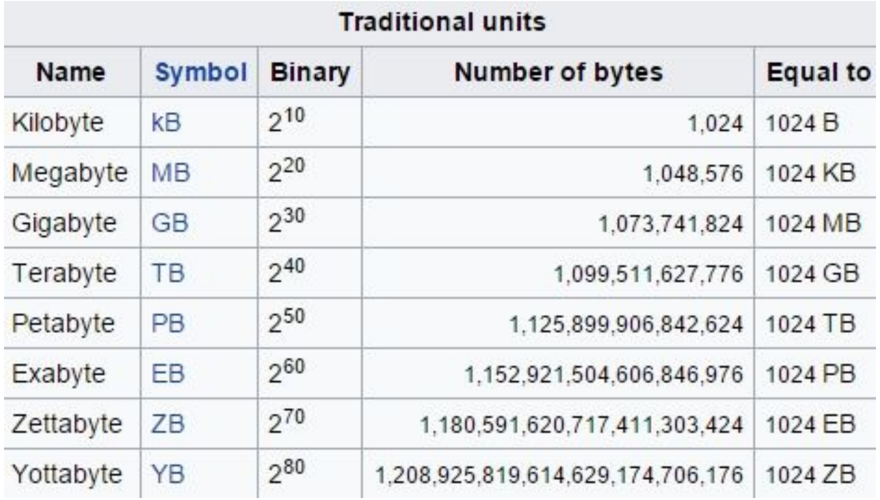
full/absolute path

https://stackoverflow.com/questions/2235173/what-is-the-naming-standard-for-path-components

File size

- Maybe trivial, but it is measured in bytes
- Remember the 4 GB max file size on FAT**32**?

  2^32 - 1 ➔ 4,294,967,295 (2^{32} − 1) bytes, ca 4 GB max

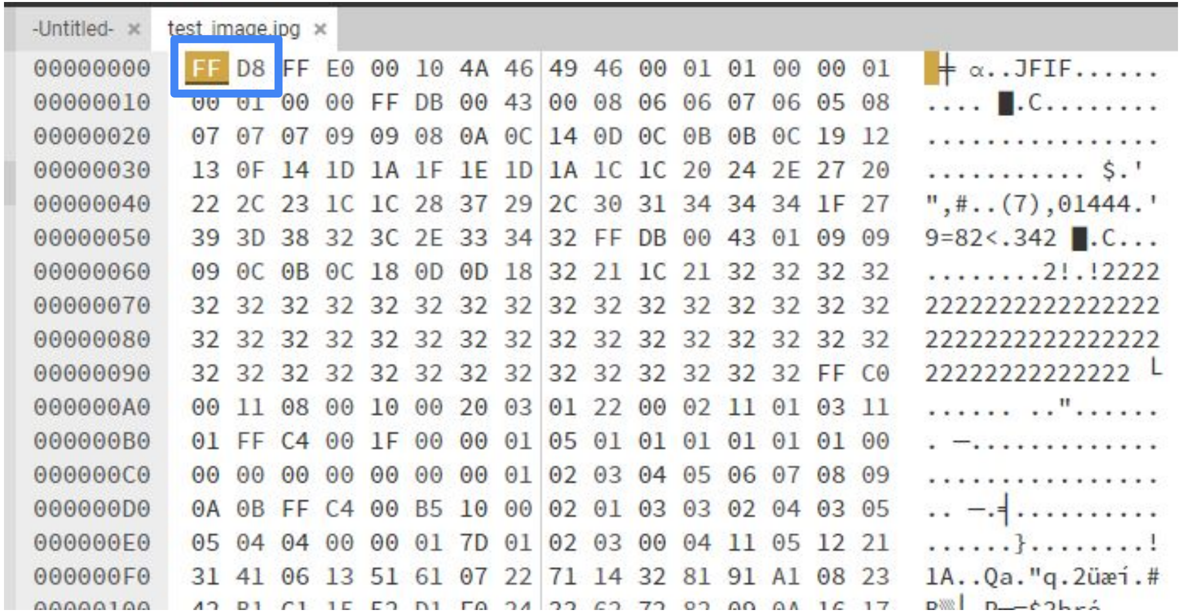


Traditional units				
Name	**Symbol**	**Binary**	**Number of bytes**	**Equal to**
Kilobyte	kB	2^{10}	1,024	1024 B
Megabyte	MB	2^{20}	1,048,576	1024 KB
Gigabyte	GB	2^{30}	1,073,741,824	1024 MB
Terabyte	TB	2^{40}	1,099,511,627,776	1024 GB
Petabyte	PB	2^{50}	1,125,899,906,842,624	1024 TB
Exabyte	EB	2^{60}	1,152,921,504,606,846,976	1024 PB
Zettabyte	ZB	2^{70}	1,180,591,620,717,411,303,424	1024 EB
Yottabyte	YB	2^{80}	1,208,925,819,614,629,174,706,176	1024 ZB

Files' internal metadata

Magic Numbers:

Beginning of file tells you which file type it is!



```
00000000  FF D8 FF E0 00 10 4A 46 49 46 00 01 01 00 00 01   ╪ α..JFIF......
00000010  00 01 00 00 FF DB 00 43 00 08 06 06 07 06 05 08  .... █.C........
00000020  07 07 07 09 09 08 0A 0C 14 0D 0C 0B 0B 0C 19 12  ................
00000030  13 0F 14 1D 1A 1F 1E 1D 1A 1C 1C 20 24 2E 27 20  ............ $.'
00000040  22 2C 23 1C 1C 28 37 29 2C 30 31 34 34 34 1F 27  ",#..(7),01444.'
00000050  39 3D 38 32 3C 2E 33 34 32 FF DB 00 43 01 09 09  9=82<.342 █.C...
00000060  09 0C 0B 0C 18 0D 0D 18 32 21 1C 21 32 32 32 32  ........2!.!2222
00000070  32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32  2222222222222222
00000080  32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32  2222222222222222
00000090  32 32 32 32 32 32 32 32 32 32 32 32 32 32 FF C0  22222222222222 └
000000A0  00 11 08 00 10 00 20 03 01 22 00 02 11 01 03 11  ...... ..".....
000000B0  01 FF C4 00 1F 00 00 01 05 01 01 01 01 01 01 00  . ─.............
000000C0  00 00 00 00 00 00 00 01 02 03 04 05 06 07 08 09  ................
000000D0  0A 0B FF C4 00 B5 10 00 02 01 03 03 02 04 03 05  .. ─.┤..........
000000E0  05 04 04 00 00 01 7D 01 02 03 00 04 11 05 12 21  ......}........!
000000F0  31 41 06 13 51 61 07 22 71 14 32 81 91 A1 08 23  1A..Qa."q.2üæí.#
```

What can I do with files - in general?

- Create a new file
- Change the access permissions and attributes of a file
- Open a file, which makes the file contents available to the program
- Read data from a file
- Write data to a file
- Delete a file
- Close a file, terminating the association between it and the program
- Truncate a file, shortening it to a specified size within the file system without rewriting any content

File extensions are arbitrary

Extensions help to decipher the file content, but the file needs still to follow the file type’s organization.

For example:
Renaming image.**png** to image.**jpg** does not convert the file to the JPG standard.
It has still the SAME CONTENT (--> being a PNG file)

File extensions

Which ones do you know?

File types commonly used in Data Science

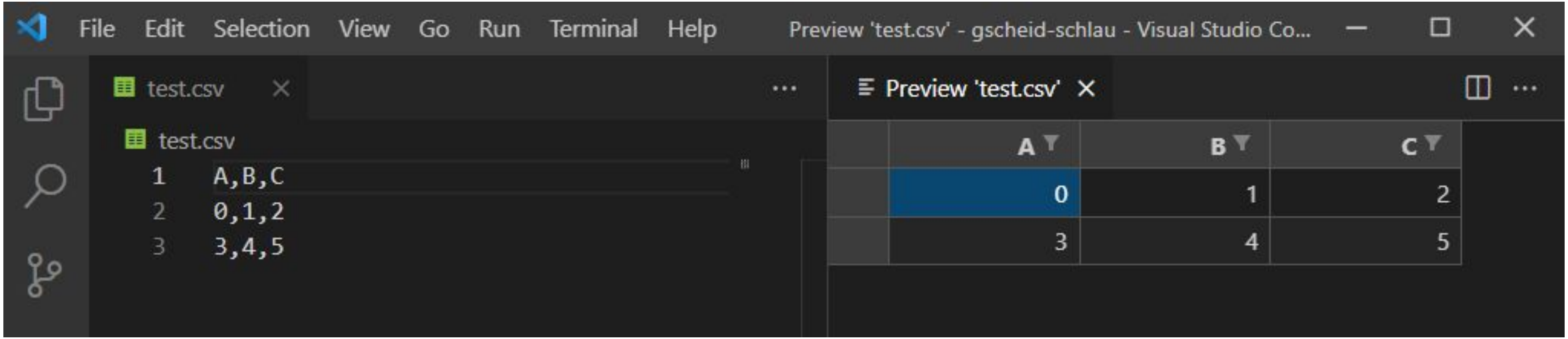
- Plain text (common extensions *.txt, *.csv, *.log, *.json, *.xml) - Python program code!
- Spreadsheets (*.xlsx)
- Word processing files (*.docx)
- Images (*.jpg -> Camera, *.png -> Scientific data, *.tif -> Microscopy)
- Videos (*.avi -> mostly raw data, *.mp4 almost everything, commonly h264 codec)
- Medical imaging data (DICOM, Nifti *.nii and *.nii.gz)
- Vector graphics (*.pdf, *.svg, *.ai)
- Container files (*.hdf5)
- Archives (*.zip, *.tar.gz, *.7z, *.rar)
- Database (*.sqlite)
- Deep Neural Networks (*.pb, *.h5, *.tflite, ...)

Software you should have around

These are EXAMPLES that e.g. work for me. They can be replaced by various other tools. Everything is free except indicated.

- Visual Studio Code (plain text, CSV files, JSON, XML)
- LibreOffice/M$ Office/Google Docs (docx, xlsx, pptx,...)
- FIJI / ImageJ (Microscopy images) and paint.NET (all purpose images)
- VLC (Videos)
- Inkscape (free) or Adobe Illustrator ($$$) (vector graphics)
- 7zip (all kinds of archives)
- HDF5View (HDF5 container files)
- Netron (universal cross-platform deep neural network viewer)

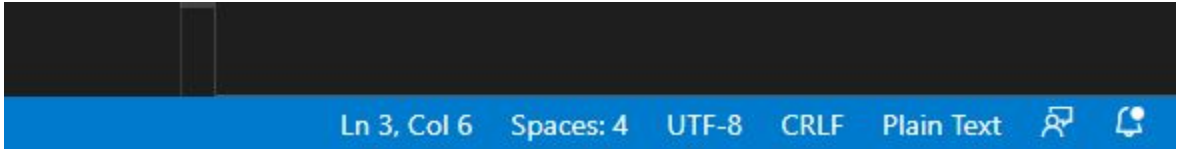
Plain text file



test.csv

```
A,B,C
0,1,2
3,4,5
```

A	B	C
0	1	2
3	4	5



Ln 3, Col 6 Spaces: 4 UTF-8 CRLF Plain Text

Let's deepdive

How is this file stored?

⇒ HEX Editor

Comparison of plain text files

- Older OS did not track how large a file is -
  They used the EOF-tag (end of file)
- Newer OS track how large a file is - no need for EOF
- CR/LF (EOL ➔ \r\n, 0x0D, 0x0A ➔ 13 and 10 in decimal)
  (carriage return, line feed)

  \r ➔ advances to the beginning of the line
  \n ➔ goes to new line

Storing information efficiently

Example: WWII

The war is over	(8 bit * 15 characters = 120 bits)
The war is not over	(8 bit * 19 characters = 152 bits)

Information can be reduced to **1 (!) bit** (either we won or we didn't)

Formalize with Shannon entropy:

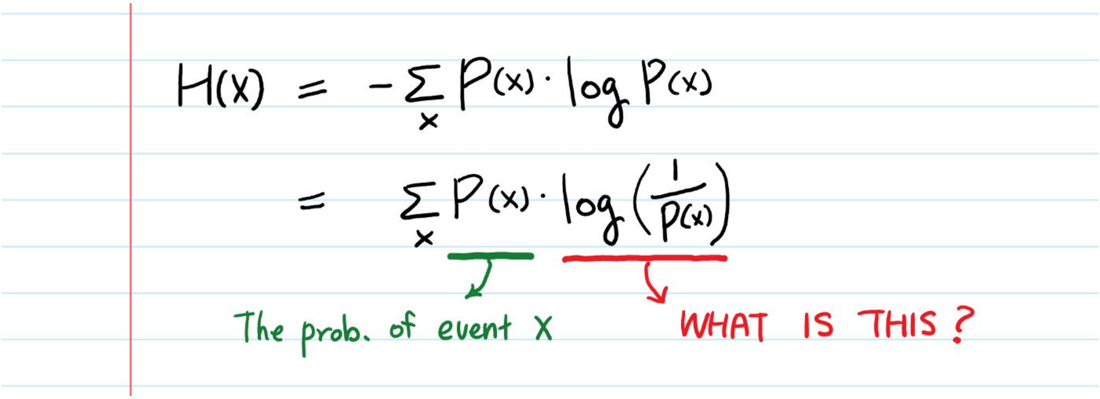
$$H(\mathrm{x}) = \mathbb{E}_{\mathrm{x}\sim P}[I(x)] = -\mathbb{E}_{\mathrm{x}\sim P}[\log P(x)], \qquad (3.49)$$

Deeper...

$$H(\mathrm{x}) = \mathbb{E}_{\mathrm{x}\sim P}[I(x)] = -\mathbb{E}_{\mathrm{x}\sim P}[\log P(x)], \tag{3.49}$$

I(x) is **the information content of X**.

I(x) itself is **a random variable.** In our example, the possible outcomes of the War. Thus, **H(x)** is **the expected value of every possible information.**



$$H(x) = -\sum_{x} P(x) \cdot \log P(x)$$

$$= \sum_{x} \underbrace{P(x)}_{\text{The prob. of event X}} \cdot \underbrace{\log\left(\frac{1}{P(x)}\right)}_{\text{WHAT IS THIS ?}}$$

Nazis surrender 0.75,
Nazis do not surrender 0.25

How much information does the event 'surrender' have?

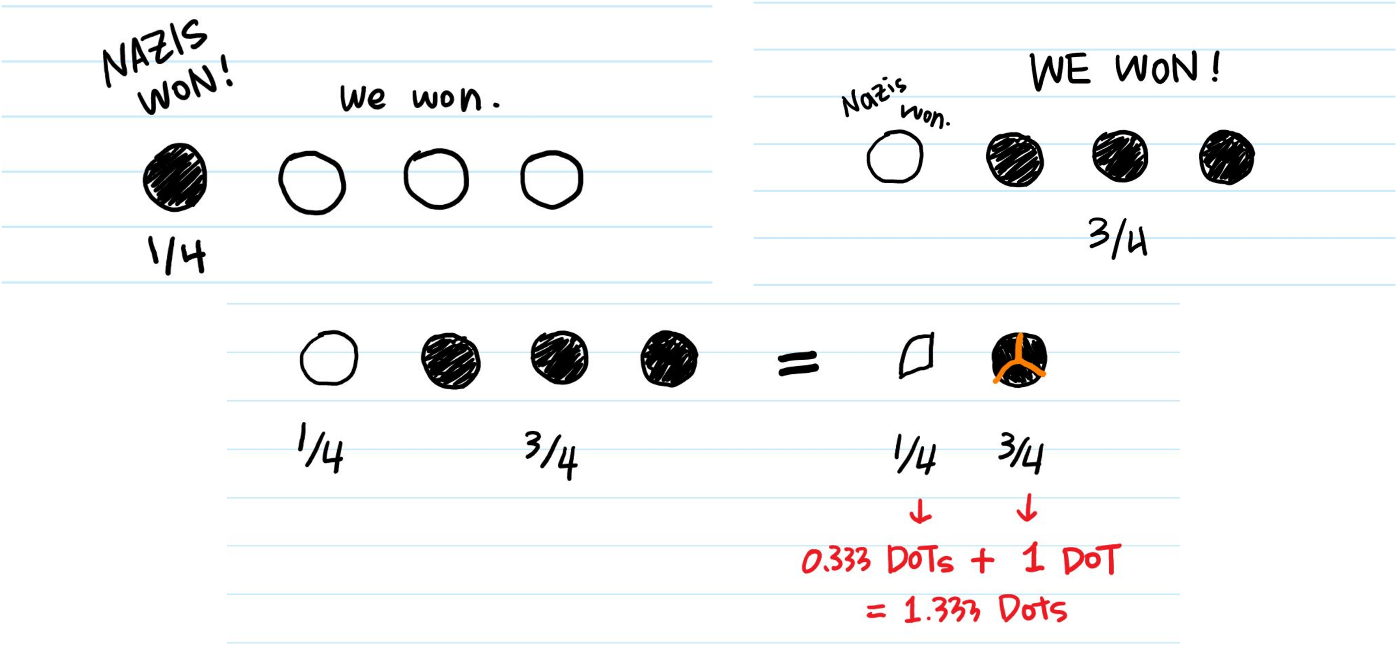
log (1/0.75) = log(1.333) = 0.41 (log base 2 omitted going forward)

How much information does the event 'not surrender' have?

log (1/0.25) = log(4) = 2

⇒ The unlikely event has HIGHER ENTROPY!

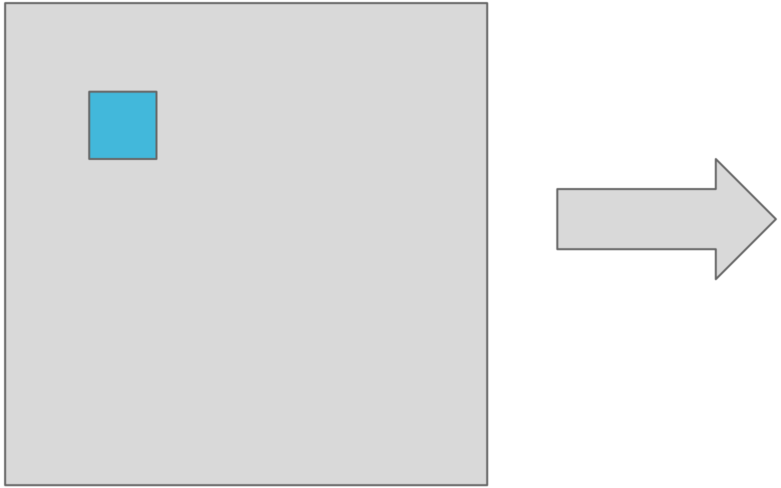
Taken together



Thus, **the information in EVERY possible news** is 0.25 * log(4) + 0.75 * log(1.333)= 0.81 (Shannon's entropy formula.)

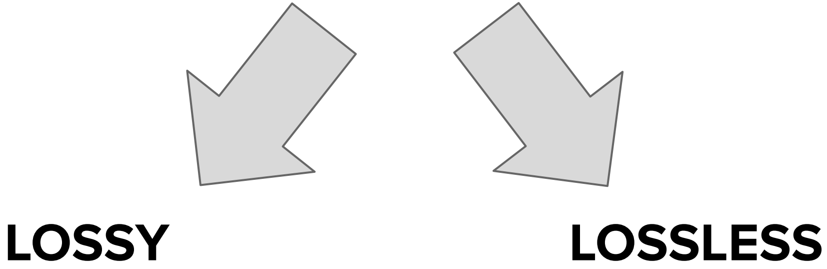
Compression

Increasing entropy! Removing redundant information!



Rectangle size,
Blue rectangle size and location

Compression algorithms



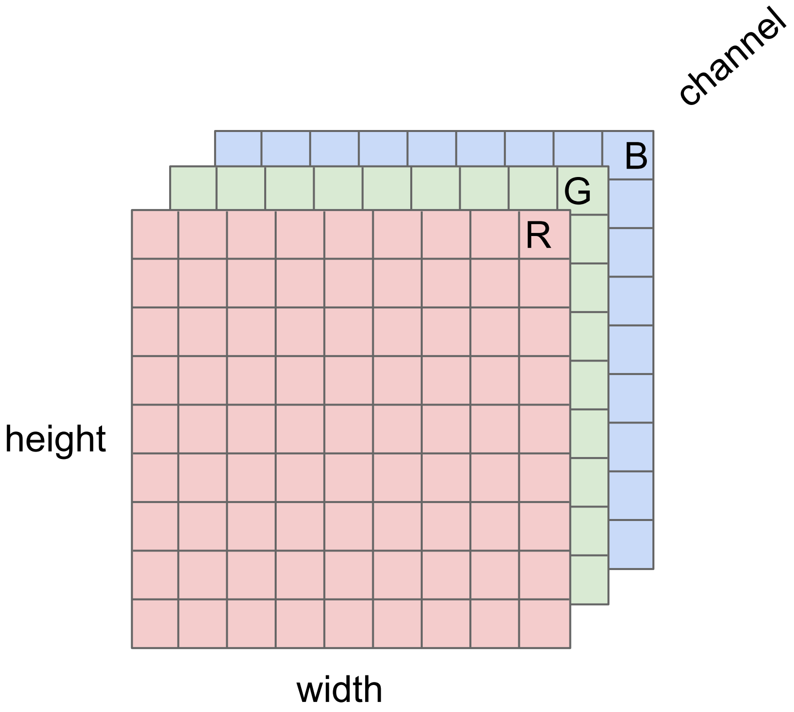
LOSSY

LOSSLESS

E.g. Discrete Cosine Transformations
As in JPEG files or MP3 files

E.g. ZIP/7z files

An image consists of many pixels



Very common:

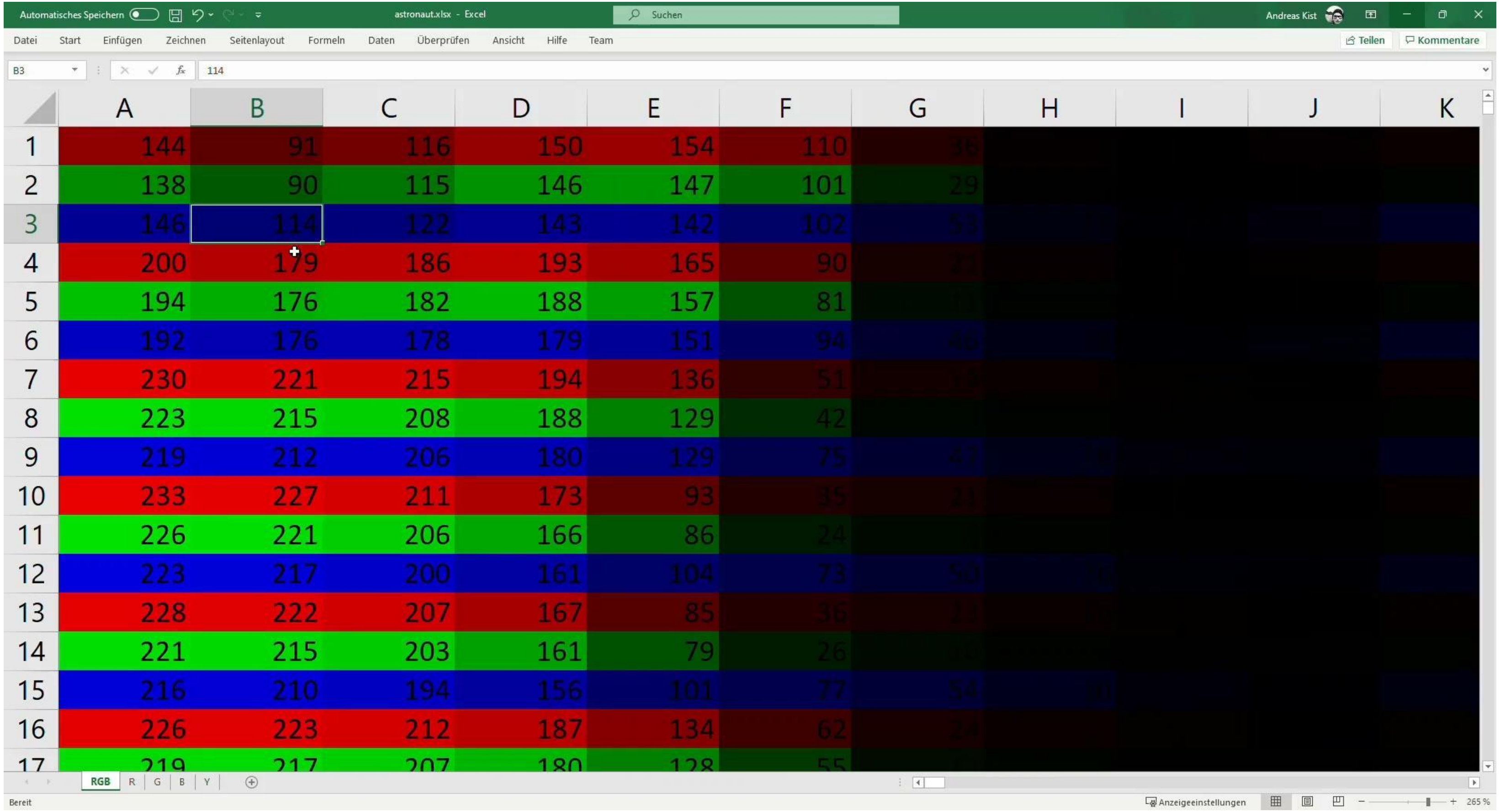
RGB (height x width x channels ⇒ HxWx3)
RGBA (HxWx4, last channel is alpha ⇔ transparency)
Monochrome (HxWx1 ⇒ HxW)

Microscopy data:

HxWxC,
where C is e.g. DAPI, GFP, Alexa488, mCherry, ….

E.g. an image of HxWxC = 256x256x3,
has 256x256x3 = 196,608 units, that we call **pixels**!

Images are just Excel sheets



	A	B	C	D	E	F	G	H	I	J	K
1	144	91	116	150	154	110	36				
2	138	90	115	146	147	101	29				
3	146	114	122	143	142	102	53				
4	200	179	186	193	165	90	21				
5	194	176	182	188	157	81	33				
6	192	176	178	179	151	94	46				
7	230	221	215	194	136	51	19				
8	223	215	208	188	129	42	[illegible]				
9	219	212	206	180	129	75	47	[illegible]			
10	233	227	211	173	93	35	21				
11	226	221	206	166	86	24	[illegible]				
12	223	217	200	161	104	73	50	[illegible]			
13	228	222	207	167	85	36	23	[illegible]			
14	221	215	203	161	79	26	[illegible]				
15	216	210	194	156	101	77	54	[illegible]			
16	226	223	212	187	134	62	24				
17	219	217	207	180	128	55	[illegible]				

Interacting with images in Python

OPENING/SAVING

imageio - Python library for reading and writing image data

scikit-image
image processing in python

OpenCV

PROCESSING

NumPy

Multi-dimensional image processing (scipy.ndimage)¶

scikit-image
image processing in python

OpenCV

PLOTTING

matplotlib

seaborn: statistical data visualization

PyQtGraph
Scientific Graphics and GUI Library for Python

Images in a scientific environment

TIFF

👍

- Saves raw data
- Multiple channels
- Multiple bit depth levels

PNG

👍

- Lossless compression
- Up to 4 channels (RGBA)

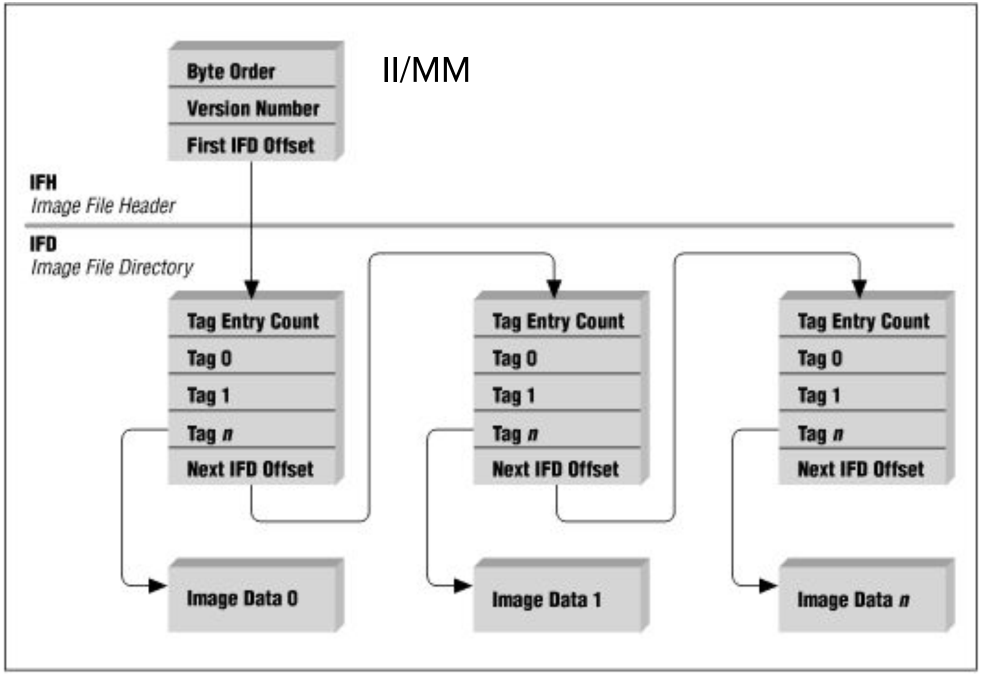
JPG

👎

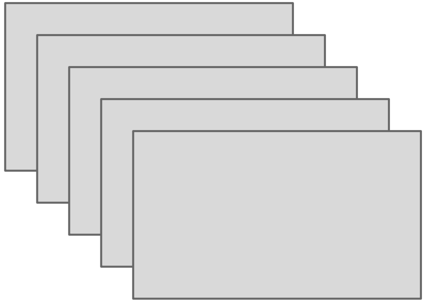
- Lossy compression
- Fine for photography
- Compression artifacts

The TIF file format header

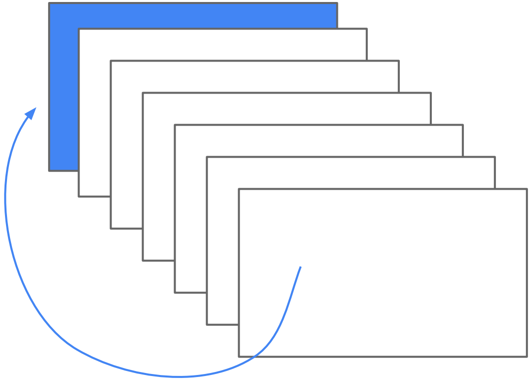
Some files need more information, such as bit depth of an image (8 bit, 16 bit), color or grayscale, size of the image etc.



Videos

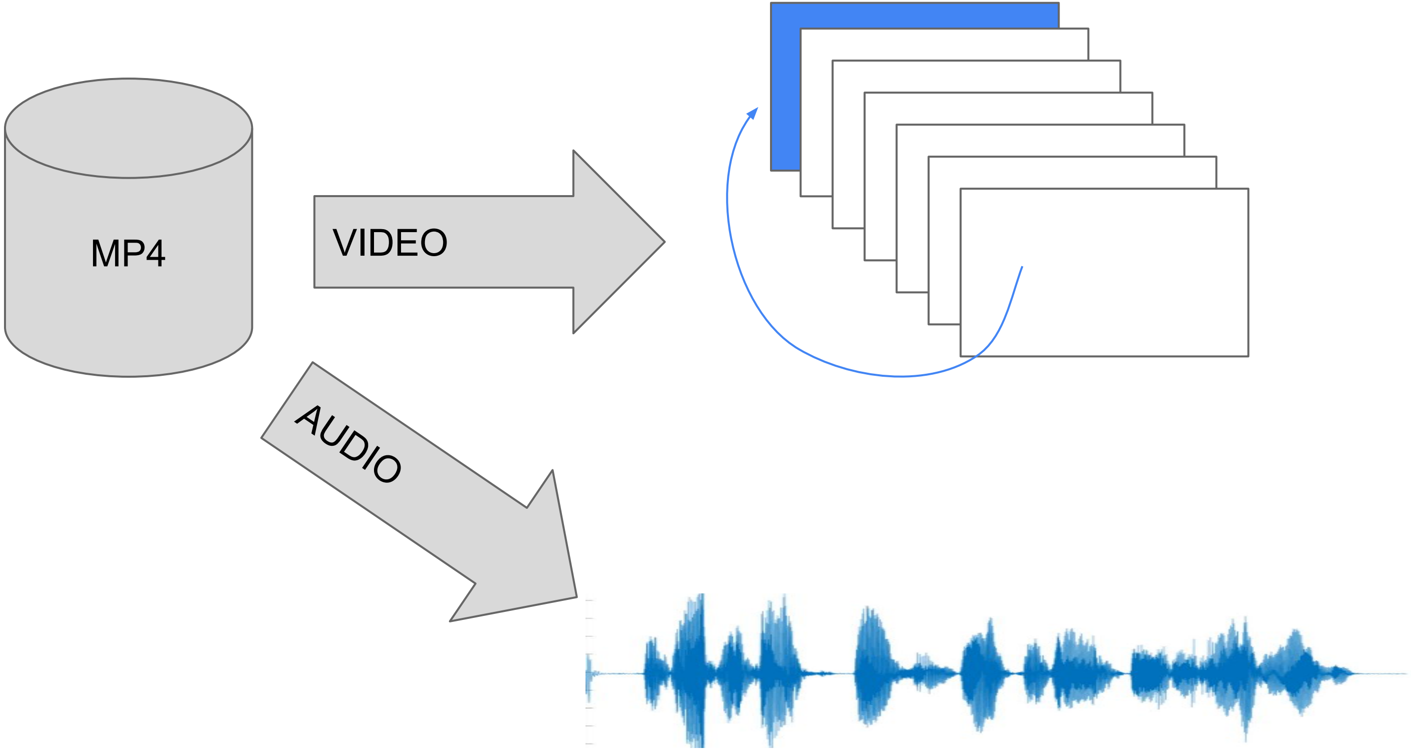


WAY 1: Store each frame one after another, each frame is independent



WAY 2: Store **key frames** and then store only the difference relative to the key frames

H264 codec in MP4 container



H264

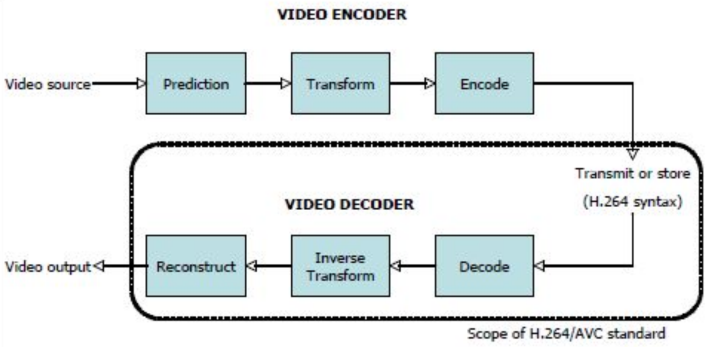


Figure 1 The H.264 video coding and decoding process

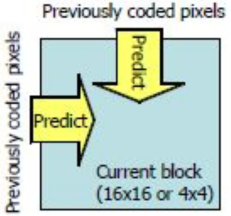


Figure 2 Intra prediction

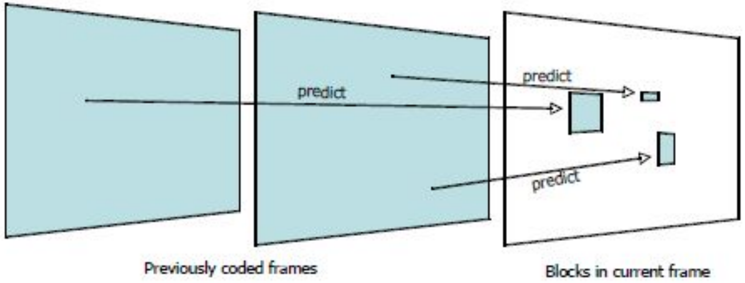


Figure 3 Inter prediction

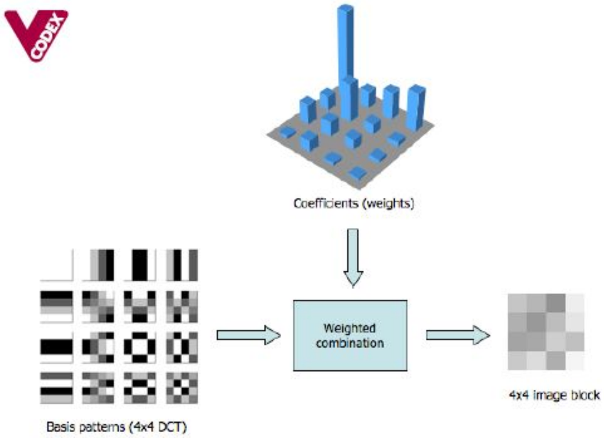


Figure 4 Inverse transform: combining weighted basis patterns to create a 4x4 image block

H264 performance

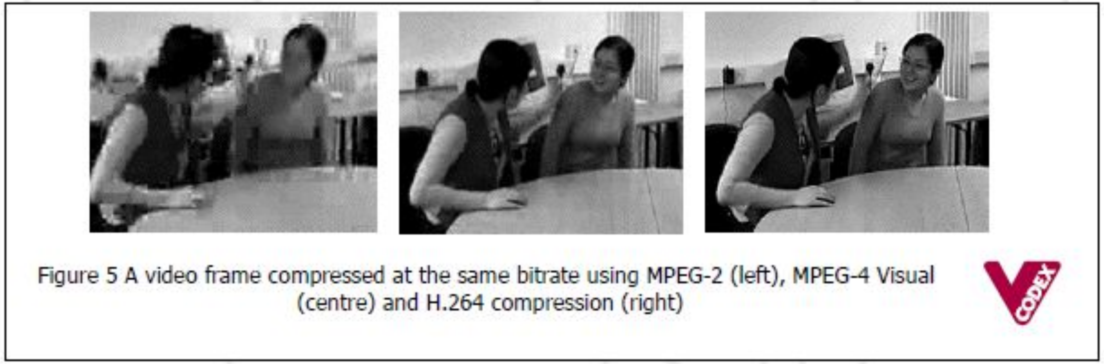
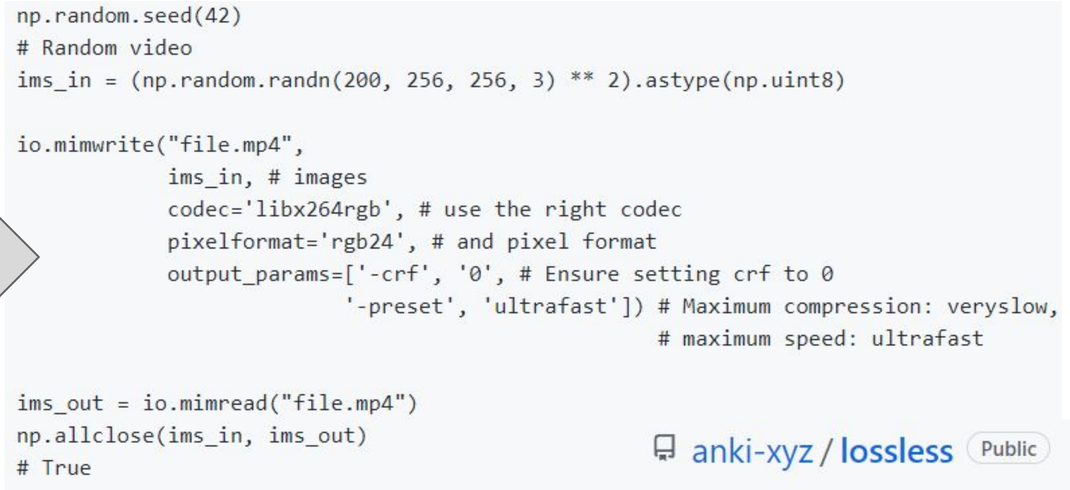


Figure 5 A video frame compressed at the same bitrate using MPEG-2 (left), MPEG-4 Visual (centre) and H.264 compression (right)

H264 is a great encoder, however, with the default settings you encode your data **lossy**!

LOSSLESS!!!



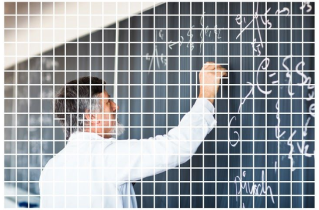
```
np.random.seed(42)
# Random video
ims_in = (np.random.randn(200, 256, 256, 3) ** 2).astype(np.uint8)

io.mimwrite("file.mp4",
            ims_in, # images
            codec='libx264rgb', # use the right codec
            pixelformat='rgb24', # and pixel format
            output_params=['-crf', '0', # Ensure setting crf to 0
                           '-preset', 'ultrafast']) # Maximum compression: veryslow,
                                                    # maximum speed: ultrafast

ims_out = io.mimread("file.mp4")
np.allclose(ims_in, ims_out)
# True
```

anki-xyz / lossless Public

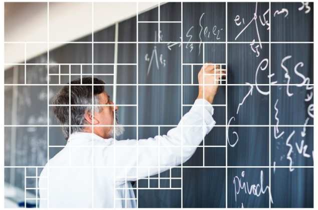
Storing in mp4 is convenient for sharing and inspection using VLC

“New” kids on the block

H.264



HEVC



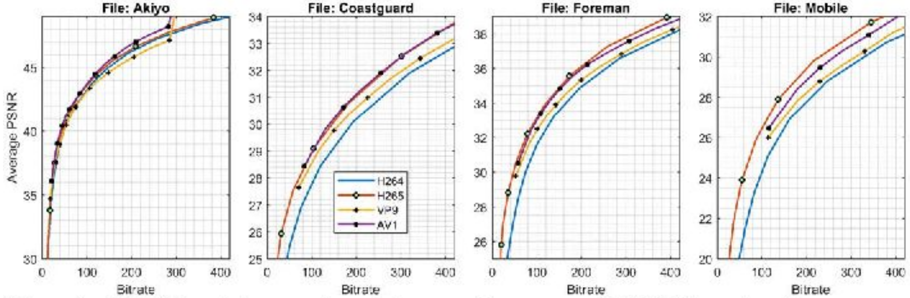


Fig. 8: PSNR with varying bitrates in case of CRF level adjustment (*placebo* presets for H.264 and H.265)

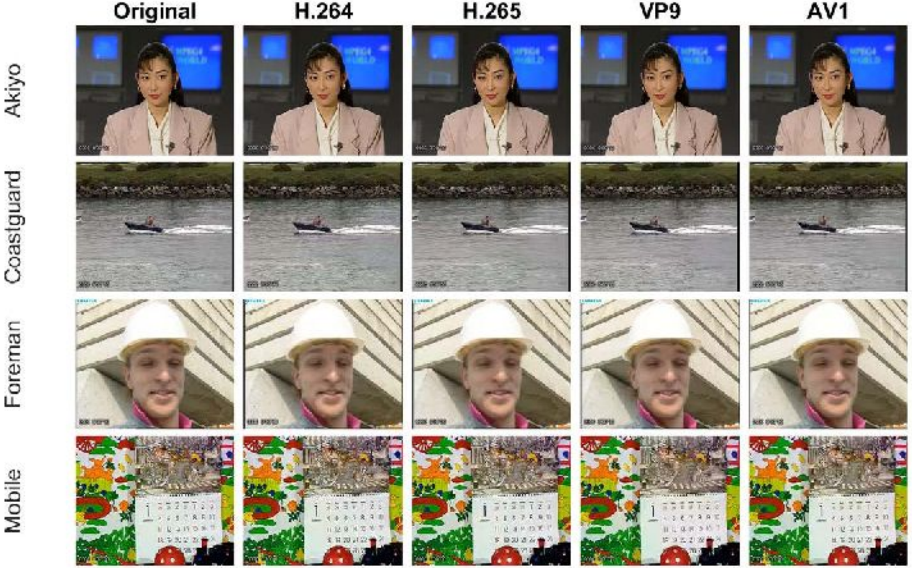
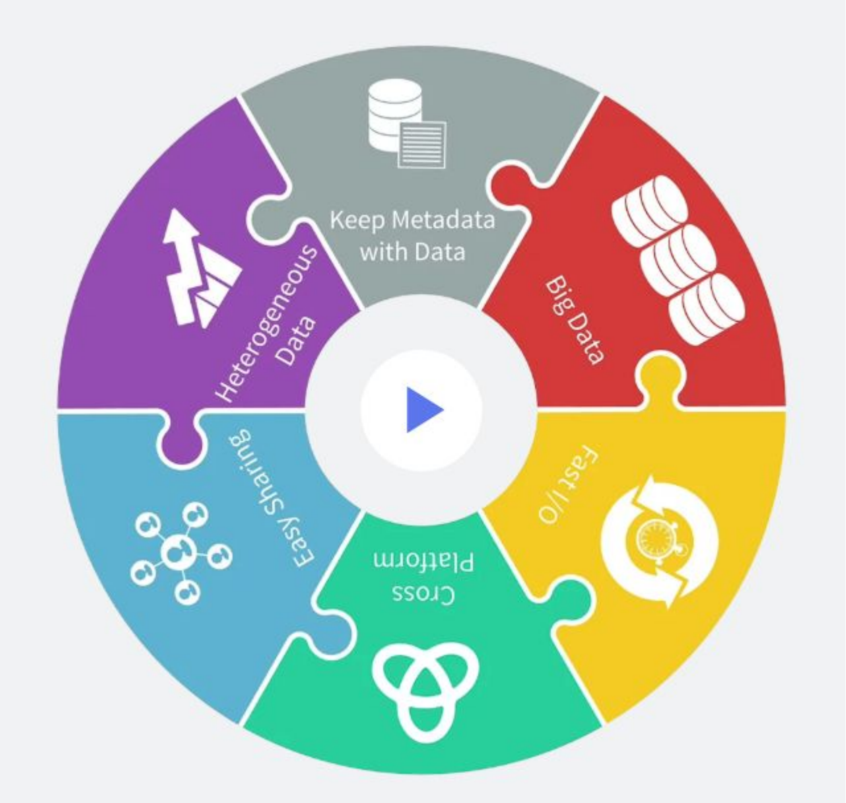


Fig. 9: First frames of the originals and the encoded videos at the

Layek, Md. Abu et al. “Performance analysis of H.264, H.265, VP9 and AV1 video encoders.” *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)* (2017): 322-325.

HDF5 - the universal file container



Keep Metadata with Data

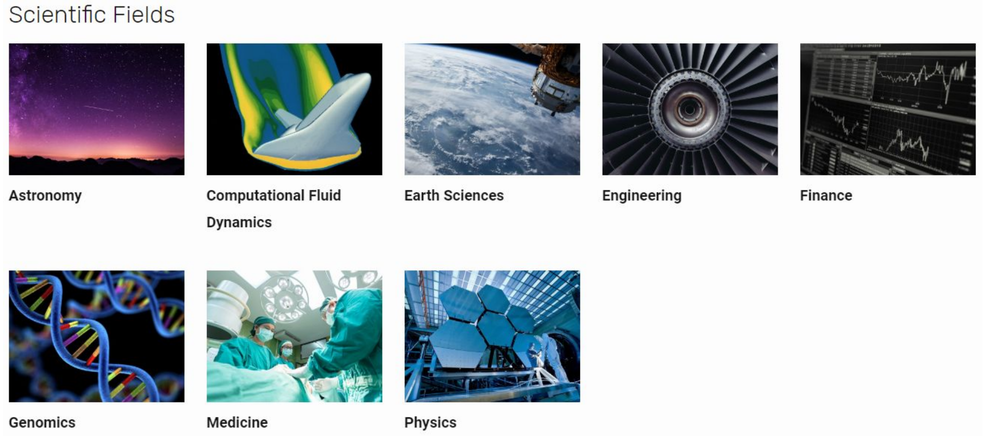
Big Data

Fast I/O

Cross Platform

Easy Sharing

Heterogeneous Data



Scientific Fields

Astronomy

Computational Fluid Dynamics

Earth Sciences

Engineering

Finance

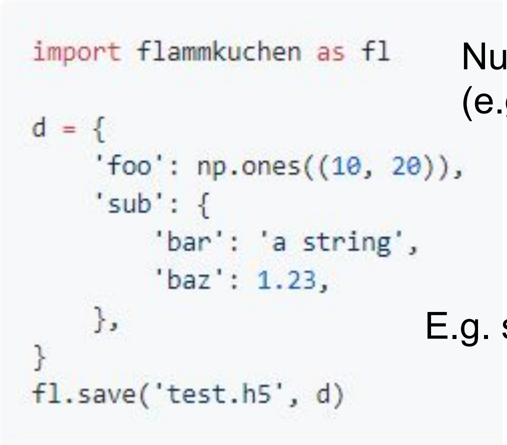
Genomics

Medicine

Physics

How to handle/open/save HDF5?

```
pip install flammkuchen
```

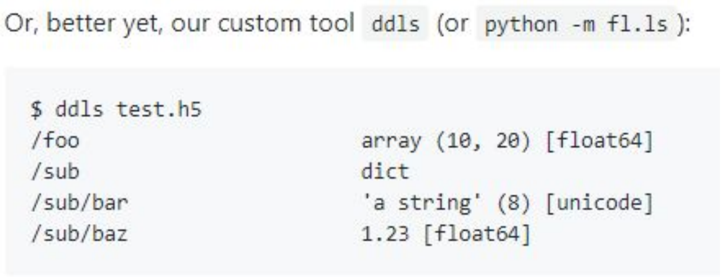


```python
import flammkuchen as fl

d = {
    'foo': np.ones((10, 20)),
    'sub': {
        'bar': 'a string',
        'baz': 1.23,
    },
}
fl.save('test.h5', d)
```

Numpy ndarray
(e.g. multi-channel z-stack…)

E.g. some metadata...

Command line tool



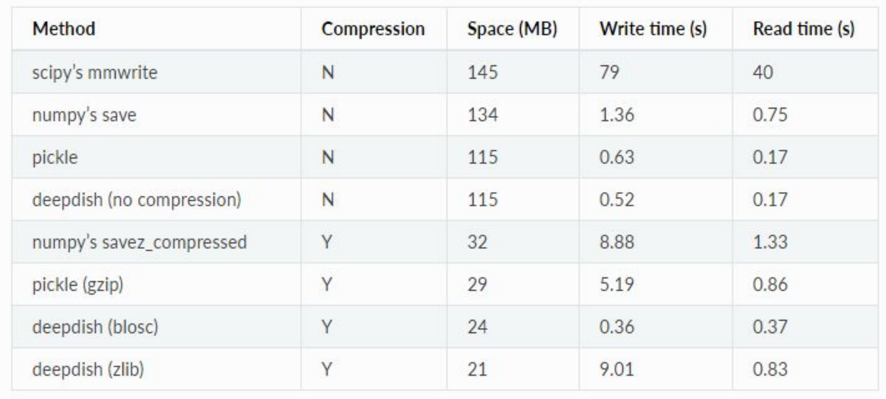
Or, better yet, our custom tool `ddls` (or `python -m fl.ls`):

```
$ ddls test.h5
/foo                        array (10, 20) [float64]
/sub                        dict
/sub/bar                    'a string' (8) [unicode]
/sub/baz                    1.23 [float64]
```

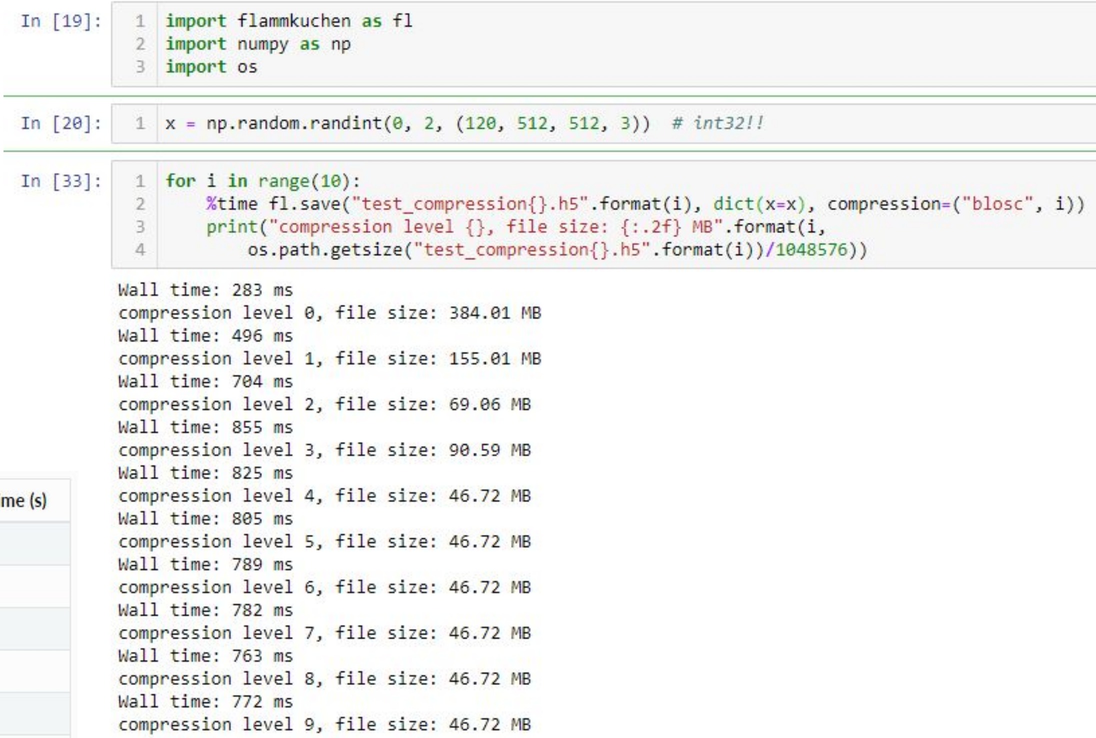
Compression

Intelligent lossless compression,
A general feature of many libraries!

Check your data dtype!
You may save a lot of space!



Method	Compression	Space (MB)	Write time (s)	Read time (s)
scipy's mmwrite	N	145	79	40
numpy's save	N	134	1.36	0.75
pickle	N	115	0.63	0.17
deepdish (no compression)	N	115	0.52	0.17
numpy's savez_compressed	Y	32	8.88	1.33
pickle (gzip)	Y	29	5.19	0.86
deepdish (blosc)	Y	24	0.36	0.37
deepdish (zlib)	Y	21	9.01	0.83



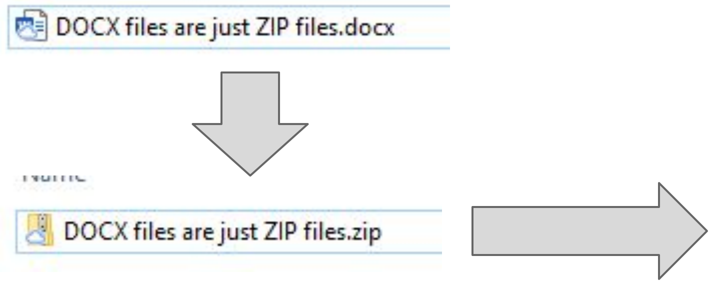
```
In [19]:
import flammkuchen as fl
import numpy as np
import os
```

```
In [20]:
x = np.random.randint(0, 2, (120, 512, 512, 3))  # int32!!
```

```
In [33]:
for i in range(10):
    %time fl.save("test_compression{}.h5".format(i), dict(x=x), compression=("blosc", i))
    print("compression level {}, file size: {:.2f} MB".format(i,
        os.path.getsize("test_compression{}.h5".format(i))/1048576))
```

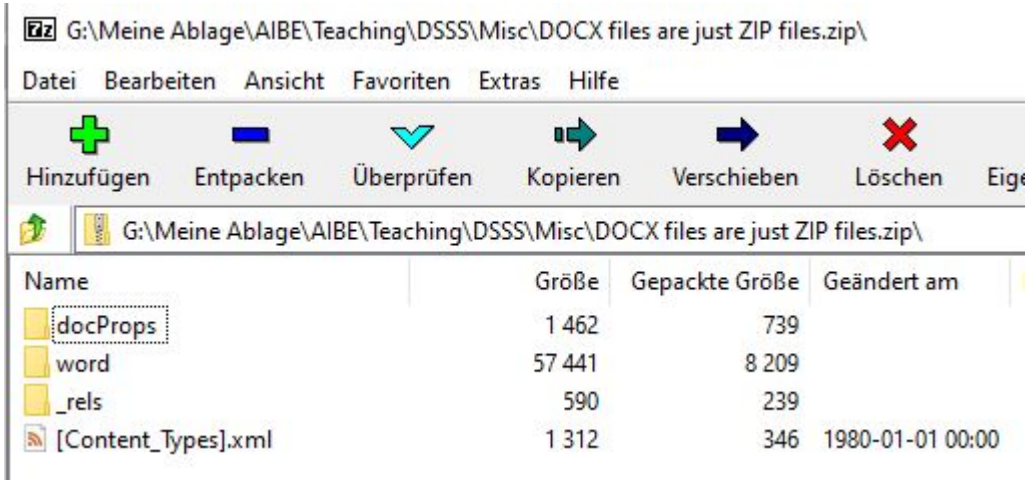
```
Wall time: 283 ms
compression level 0, file size: 384.01 MB
Wall time: 496 ms
compression level 1, file size: 155.01 MB
Wall time: 704 ms
compression level 2, file size: 69.06 MB
Wall time: 855 ms
compression level 3, file size: 90.59 MB
Wall time: 825 ms
compression level 4, file size: 46.72 MB
Wall time: 805 ms
compression level 5, file size: 46.72 MB
Wall time: 789 ms
compression level 6, file size: 46.72 MB
Wall time: 782 ms
compression level 7, file size: 46.72 MB
Wall time: 763 ms
compression level 8, file size: 46.72 MB
Wall time: 772 ms
compression level 9, file size: 46.72 MB
```

Fun fact: DOCX files are just ZIP files...



DOCX files are just ZIP files.docx

DOCX files are just ZIP files.zip



G:\Meine Ablage\AIBE\Teaching\DSSS\Misc\DOCX files are just ZIP files.zip\

G:\Meine Ablage\AIBE\Teaching\DSSS\Misc\DOCX files are just ZIP files.zip\

Name	Größe	Gepackte Größe	Geändert am
docProps	1 462	739	
word	57 441	8 209	
_rels	590	239	
[Content_Types].xml	1 312	346	1980-01-01 00:00