
Quantitative Big Imaging - Introduction

Anders Kaestner

Mar 31, 2022

CONTENTS

1	Introduction and Overview	3
1.1	Today's lecture	3
2	About the course	5
2.1	Who are we?	5
2.2	Who are you?	6
2.3	So how will this ever work?	6
2.4	Course Expectations	6
2.5	Course Overview	8
2.6	Reading Material	8
2.7	Literature / Useful References	9
3	Motivation - You have data!	11
3.1	Imaging experiments produce a lot of data	11
3.2	Motivation - We want to understand our data	12
3.3	Motivation - how to proceed?	13
3.4	The experiment life cycle	13
3.5	High acquisition rates	15
3.6	Different sources of images	15
3.7	Handling masses of images	16
4	Bringing in the scientific method	25
4.1	What is the purpose?	25
4.2	Science and Imaging	25
4.3	Important questions	26
4.4	Why quantitative?	30
4.5	Intensity gradients	30
4.6	Selecting the right tool for your task	31
4.7	Reproducibility vs. Repeatability	33
5	Different views on image analysis	35
5.1	Image Analysis: Experimentalist	36
5.2	Image Analysis: Computer Vision Approaches	36
5.3	Image Analysis: Deep Learning Approach	36
5.4	Summary analysis approaches	37
5.5	Image Formation	37
5.6	Acquiring Images	38
5.7	Traditional Imaging: Model	40
6	Images	43
6.1	An introduction to images	43

6.2	Pixel size and resolution	44
6.3	Image intensity	47
6.4	Visualizing gray levels	50
6.5	2D Intensity Images	52
6.6	Pixelwise operations	62
6.7	3D Images	63
6.8	4D Images	65
6.9	Multiple Values per pixel	65
6.10	Hyperspectral Imaging	67
7	Workflows for image analysis	71
7.1	Computing has changed: Parallel	71
7.2	Computing has changed: Cloud	72
7.3	Workflow analysis	73
7.4	Using flow charts / workflows	74
7.5	Directed Acyclical Graphs (DAG)	75
7.6	Deep Learning	83
8	Summary	85
8.1	Next weeks lecture	85

This is the lecture notes for the first lecture of the Quantitative big imaging class given during the spring semester 2021 at ETH Zurich, Switzerland.

INTRODUCTION AND OVERVIEW

Quantitative Big Imaging ETHZ: 227-0966-00L

1.1 Today's lecture

- About the course
- Motivating the use of quantitative methods in imaging
- What is an image?
- Where do images come from?
- Science and Reproducibility
- Workflows

1.1.1 We need some python modules

Python is a modular scripting language with limited functionality. Features are added through modules that are imported. These are the modules that are needed for this lecture. Please run this cell before you start using the notebook.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from skimage.io import imread
from scipy.ndimage import convolve
from skimage.morphology import disk
from skimage.transform import resize
from itertools import product
import os
from io import StringIO
```


ABOUT THE COURSE

- Who are we?
- Who are you?
- What is expected?
- **Why does this class exist?**
- Collection
- Changing computing (Parallel / Cloud)
- Course outline

2.1 Who are we?

Anders Kaestner

You will hear me a lot during this course. I am the lecturer and I will also support you with problems during the exercises.

- **Beamline scientist** at the ICON Beamline at the SINQ (Neutron Source) at Paul Scherrer Institute
 - **Lecturer** at ETH Zurich
- **Algorithm developer** Varian Medical Systems, Baden-Daettwil
- **Post Doc** at ETH Zurich, Inst for Terrestrial Ecology
- **PhD** at Chalmers Institute of Technology, Sweden, Signal processing

anders.kaestner@psi.ch

Stefano van Gogh

Will help you during the exercise sessions.

- **PhD Student** in the X-Ray Microscopy Group at ETH Zurich and Swiss Light Source at Paul Scherrer Institute
- Teaching assistant

Will support remotely the first part.

stefano.van-gogh@psi.ch

2.2 Who are you?

This course is targeting a wide range of students with different levels of experience. In the table you'll see where students came from in previous years. Some have a technical background others are merely producing images in the line of their project and have never seen much more than photoshop and similar programs for processing image data. Using some kind of programming is necessary to perform quantitative image analysis on large data sets. A single or a few images can easily be handled with interactive software, but taking it beyond that is hard without writing some lines of code.

Now, some of you have little to no programming experience while others have been programming since they got their first computer in the hand.

2.3 So how will this ever work?

Now you maybe start to get worried! The purpose of this course is not to teach you programming but rather to provide you with a bag full recipes that you can use in your projects. Most of these recipes are just a list of the commands from different python modules that you need to perform your analysis. A side-effect will probably be that you learn one or two programming tricks on the way.

In the lectures, there will be small code pieces on the slides. Some of these are there to illustrate how an operation works, while other parts are there for the nice presentation of the results (this is mostly the second half of the code cell). Presenting the results is important. In the end, you want to show your results to the scientific community. So even though the plotting clutters the slide, there is something to learn there as well.

Adaptive assignments

- Conceptual, graphical assignments with practical examples
 - Emphasis on choosing correct steps and understanding workflow
- Opportunities to create custom implementations, and perform more complicated analysis on larger datasets if interested
 - Emphasis on performance, customizing analysis, and scalability

2.4 Course Expectations

The practical part of the course has two parts. None of these are mandatory, but they will help you to better understand use the material you have learnt in the lectures.

2.4.1 Exercises

- Usually 1 set per lecture
- Optional (but recommended!)
- Level
 - *Easy* - Jupyter notebooks are prepared for the exercises
 - *Advanced* - Writing Python, Java, C++, ...

The exercises are prepared in a way that you learn step by step what you need to do and guides you through the problems. We will be using jupyter notebooks for the lectures. This is a very common way to work with image data these days.

2.4.2 Projects

In the optional science projects you will have to opportunity to test what you have learned during the course on real problems. This is the place for your creativity.

The project is optional (but strongly recommended)

- A small image processing project
- Can be related to you Master or PhD project
- You will get input and ideas for your own projects
- You will get hands on experience on the techniques you learn here
- Can be used as discussion base for your exam

Goal

- Applying Techniques to answer scientific question!
- Ideally use on a topic relevant for your current project, thesis, or personal activities or choose from one of ours (will be online, soon)
- Presentation: (approach, analysis, and results)

2.5 Course Overview

#	Topic	Date	Title	Description
1	Introduction	24th February	Introduction and Workflows	Basic overview of the course, introduction to ...
2	Data	3th March	Image Enhancement	Overview of what techniques are available for ...
3		10th March	Ground Truth: Building and Augmenting Datasets	Examples of large datasets, how they were built...
4	Segmentation	17th March	Basic Segmentation, Discrete Binary Structures	How to convert images into structures, starting...
5		24th March	Advanced Segmentation	More advanced techniques for extracting structures...
6		31st March	Supervised Problems and Segmentation	More advanced techniques for extracting structures...
7	Analysis	7th April	Analyzing Single Objects, Shape, and Texture	The analysis and characterization of single structures...
8		14th April	Statistics, Prediction, and Reproducibility	What techniques are available to analyze more ...
-		21th April	Easter break	Search for eggs
9	Big Imaging	28th April	Dynamic Experiments	Performing tracking and registration in dynamic...
10		5th May	Imaging with multiple modalities	Combining information from different sources
11		12th May	Scaling Up / Big Data	Performing large scale analyses on clusters
12		19th May	Project	Project consultation
-	26th May	Ascension	Enjoy a lovely early summer's day	
13	Wrapping up	2nd June	Project Presentations	You present your projects

2.6 Reading Material

- Some book on image processing with python (to be updated)
- Cloud Computing
- The Case for Energy-Proportional Computing *Luiz André Barroso, Urs Hölzle, IEEE Computer, December 2007*
- Concurrency
- Reproducibility
- Trouble at the lab *Scientists like to think of science as self-correcting. To an alarming degree, it is not*
- Why is reproducible research important? *The Real Reason Reproducible Research is Important*
- Science Code Manifesto

- [Reproducible Research Class @ Johns Hopkins University](#)

2.7 Literature / Useful References

These are books that are useful in many of the lectures. In particular the Image processing hand book by John Russ shows you an overview of typical image processing techniques.

- John C. Russ, “The Image Processing Handbook”,(Boca Raton, CRC Press)
 - Available [online](#) within domain [ethz.ch](#) (or [proxy.ethz.ch](#) / public VPN)
- Jean Claude, Morphometry with R
 - [Online](#) through ETHZ

MOTIVATION - YOU HAVE DATA!

3.1 Imaging experiments produce a lot of data

Working with imaging techniques you will get a lot of images that shows the sample in the eye of the technique you are using. The experiment were you acquire these images is only a small fraction of the complete workflow from idea to the final scientific publication. The amount of data can also be overwhelming for many scientist with the consequence that the data is never analyzed properly, and then also not published in the way it really deserves.

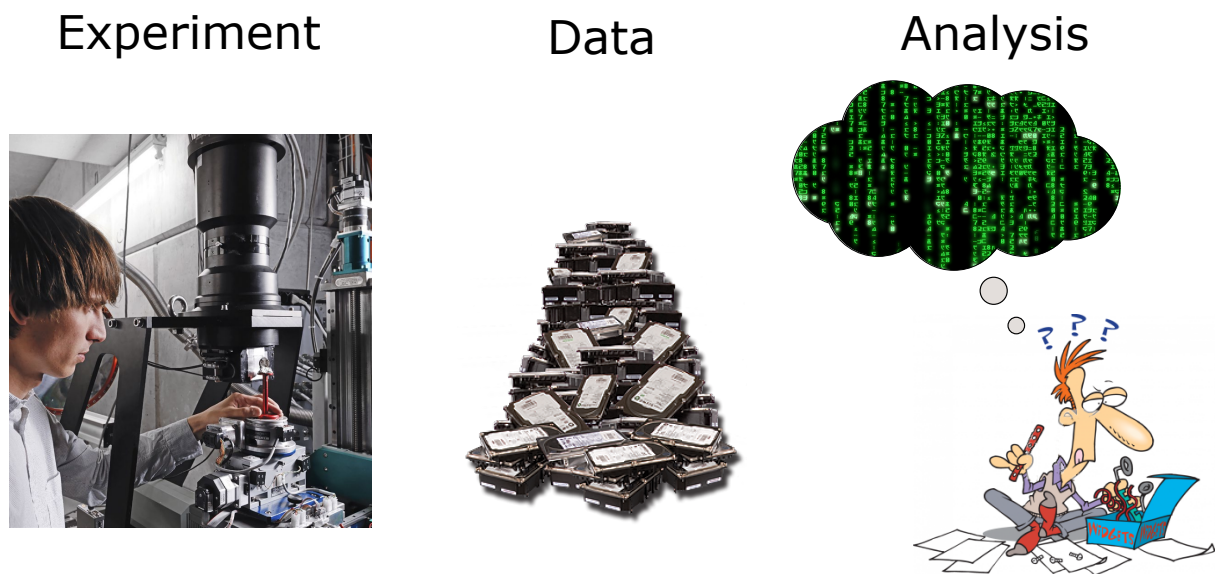


Fig. 3.1: A typical imaging experiment produces large amounts of data.

A typical imaging experiment produce **gigabytes** and even **terabytes** of data!

3.2 Motivation - We want to understand our data

In the beginning you only have data, but you want statistics to support your hypothesis. To achieve this you need to apply a series of operation steps. With each step you come closer to the answer you are seeking from the data. In the plot here you see that the initial processing steps may not provide much more understanding, but they are needed to prepare your data for the analysis.

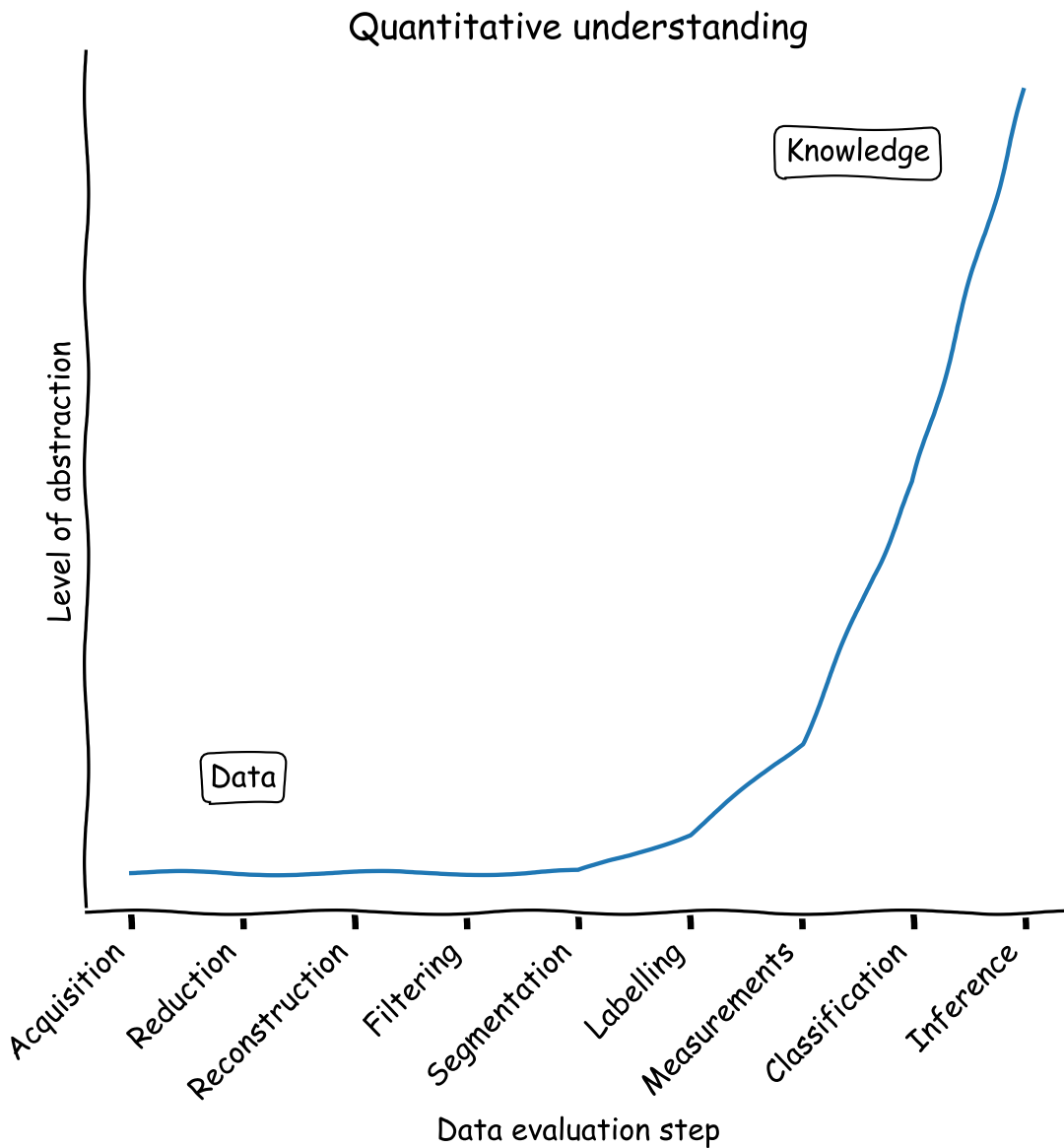


Fig. 3.2: Your understanding of the data increases with each processing step.

3.3 Motivation - how to proceed?

Now is the question how to proceed towards a working analysis workflow that results in repeatable analyses for your data.

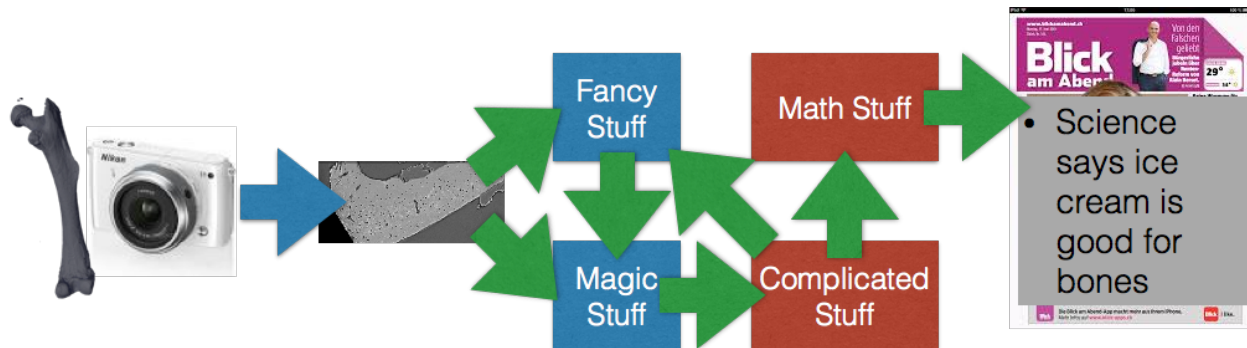


Fig. 3.3: A crazy unstructured and unclear work flow to analyze images from your experiment.

- To understand what, why and how from the moment an image is produced until it is finished (published, used in a report, ...)
- To learn how to go from one analysis on one image to 10, 100, or 1000 images (without working 10, 100, or 1000X harder)

3.4 The experiment life cycle

Now we have seen that there is a wish to obtain data at high rates and that there are technological solutions to provide this. The remaining part to develop is the post processing.

1. **Experimental Design** finding the right technique, picking the right dyes and samples has stayed relatively consistent, better techniques lead to more demanding scientists.
2. **Measurements** the actual acquisition speed of the data has increased wildly due to better detectors, parallel measurement, and new higher intensity sources
3. **Management** storing, backing up, setting up databases, these processes have become easier and more automated as data magnitudes have increased
4. **Post Processing** this portion has is the most time-consuming and difficult and has seen minimal improvements over the last years

The post processing is the least trivial part to generalize. The initial steps are often possible to generalize as these are operations that all types of imaging experiments need to go through. When it comes to the experiment specific analysis the degree of generalization decreases and the scientists are left to develop their own procedure to extract the quantitative information from the images.

3.4.1 How is time used during the experiment life cycle?

With the development of faster acquisition systems there has been a change in the ratio between

- Experiment design and preparation
- Measurements
- Data management
- and post processing

over the years. This in particular the case for X-ray imaging where the flux is high and the acquisition is limited by the detector technology. In other modalities, where the measurement is flux limited we see a different distribution.

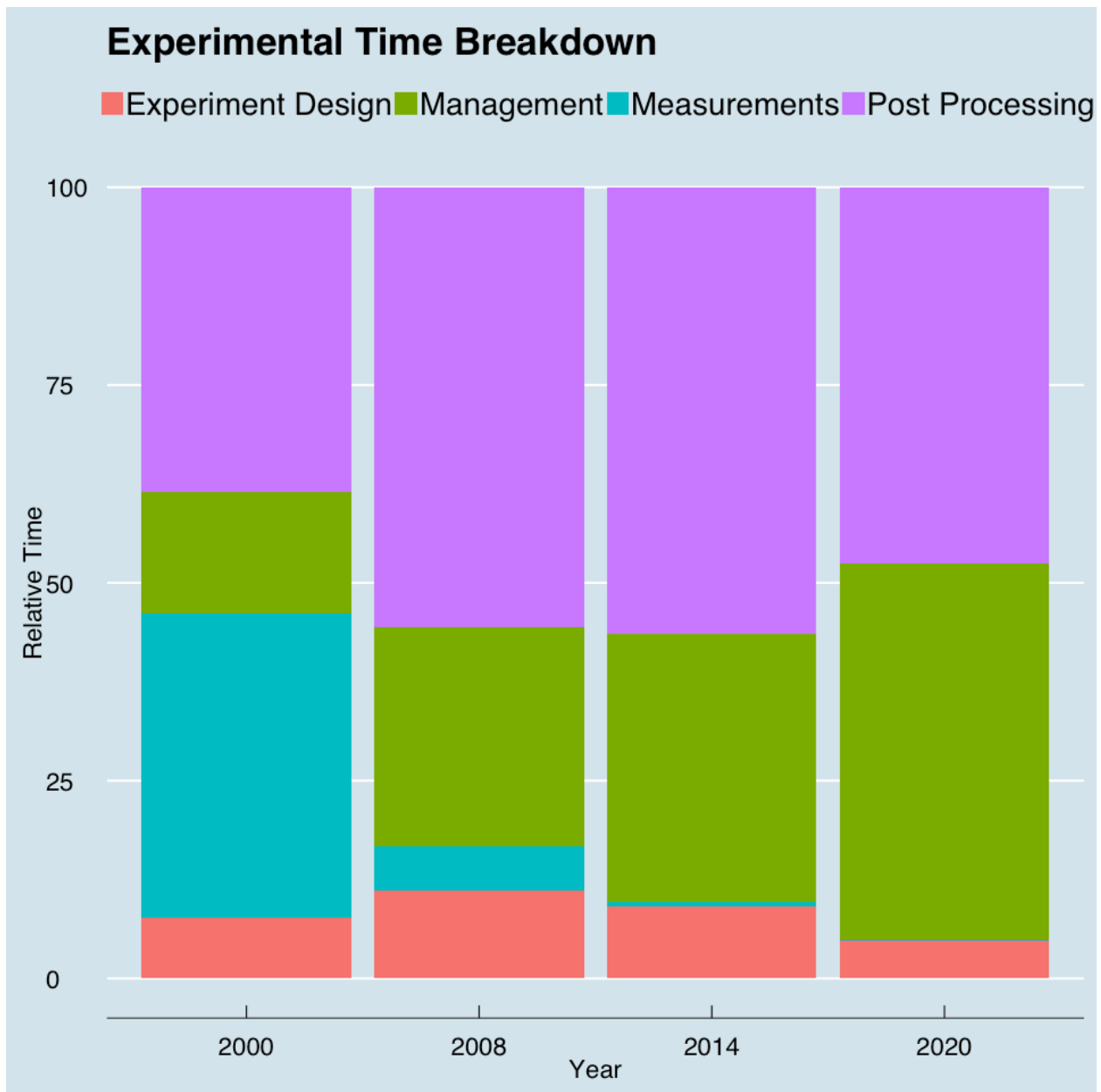


Fig. 3.4: The ratio of how much time is spent on different tasks during the lifecycle of an imaging experiment.

What also increases the post processing time is that the experiments have become more complicated over the years. Twenty years ago, it was sufficient to show qualitative information a beautiful volume rendering or a movie of the sample. Meanwhile, it has become a requirement that you provide quantitative results from the images.

3.5 High acquisition rates

The trend in imaging is that experimentalist want to follow faster and faster processes. This wish can be supported the technical development of new detectors that provide very high acquisition rates. Here, we can also see that some cameras are able to produce more data than is uploaded per day on facebook and instagram!

- Detectors are getting bigger and faster constantly
- Today's detectors are really fast
 - 2560 x 2160 images @ 1500+ times a second = 8GB/s
- Matlab / Avizo / Python / ... are saturated after 60 seconds

Many of the analysis platforms are already overwhelmed with handling the data rates produced by typical detector systems at imaging instrument. This restriction is partly due to hardware limitations. The memory is too small, hard drives are not sufficiently fast. The other side of the problem is that these tools are not prepared to work with large data streams.

- A single camera
 - More information per day than Facebook
 - Three times as many images per second as Instagram

3.6 Different sources of images

Images are produced by many different detectors and in some cases they are even the output from simulations. In the next sections we see some different imaging modalities and the data rates they produce.

3.6.1 X-Ray

X-ray imaging at synchrotron light sources produces very high frame rates thanks to the high brilliance of the source. Here are some examples of data rates from some instruments.

- SRXTM images at (>1000fps) → 8GB/s
- cSAXS diffraction patterns at 30GB/s
- Nanoscopium Beamline, 10TB/day, 10-500GB file sizes

3.6.2 Optical

Optical imaging methods are more modest than the X-ray techniques, but still they produce data in the order of some hundred Mb per second.

- Light-sheet microscopy (see talk of Jeremy Freeman) produces images → 500MB/s
- High-speed confocal images at (>200fps) → 78Mb/s

3.6.3 Personal

Finally, we also take a look at cameras on the consumer market and see that these devices also produce relatively high data rates. This data must mostly be handled by normal household computers, which can be a challenging task...

- GoPro 4 Black - 60MB/s (3840 x 2160 x 30fps) for \$600
- fps1000 - 400MB/s (640 x 480 x 840 fps) for \$400

3.7 Handling masses of images

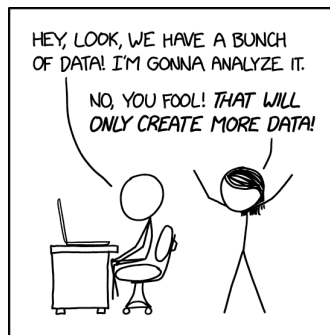


Fig. 3.5: New data will be produced once you start working with it.

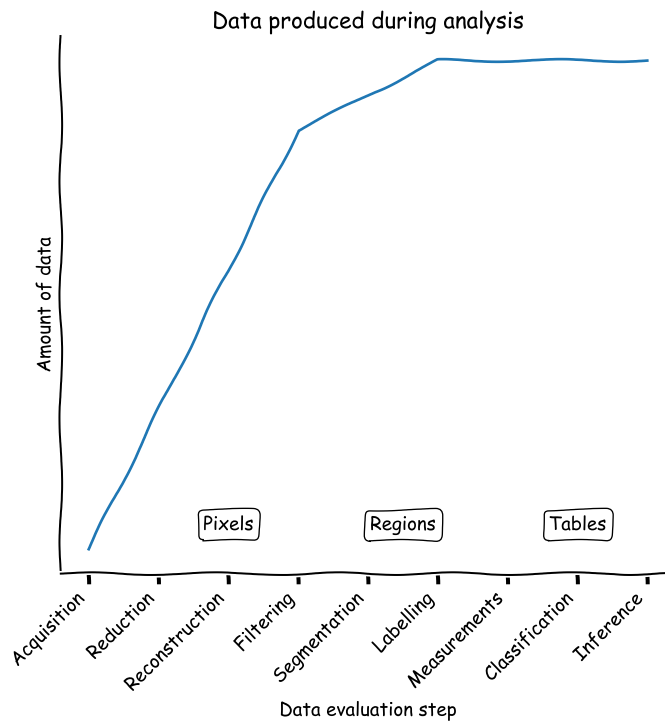


Fig. 3.6: A qualitative plot showing how you data will grow with different processing steps.

3.7.1 So... how much is a TB, really?

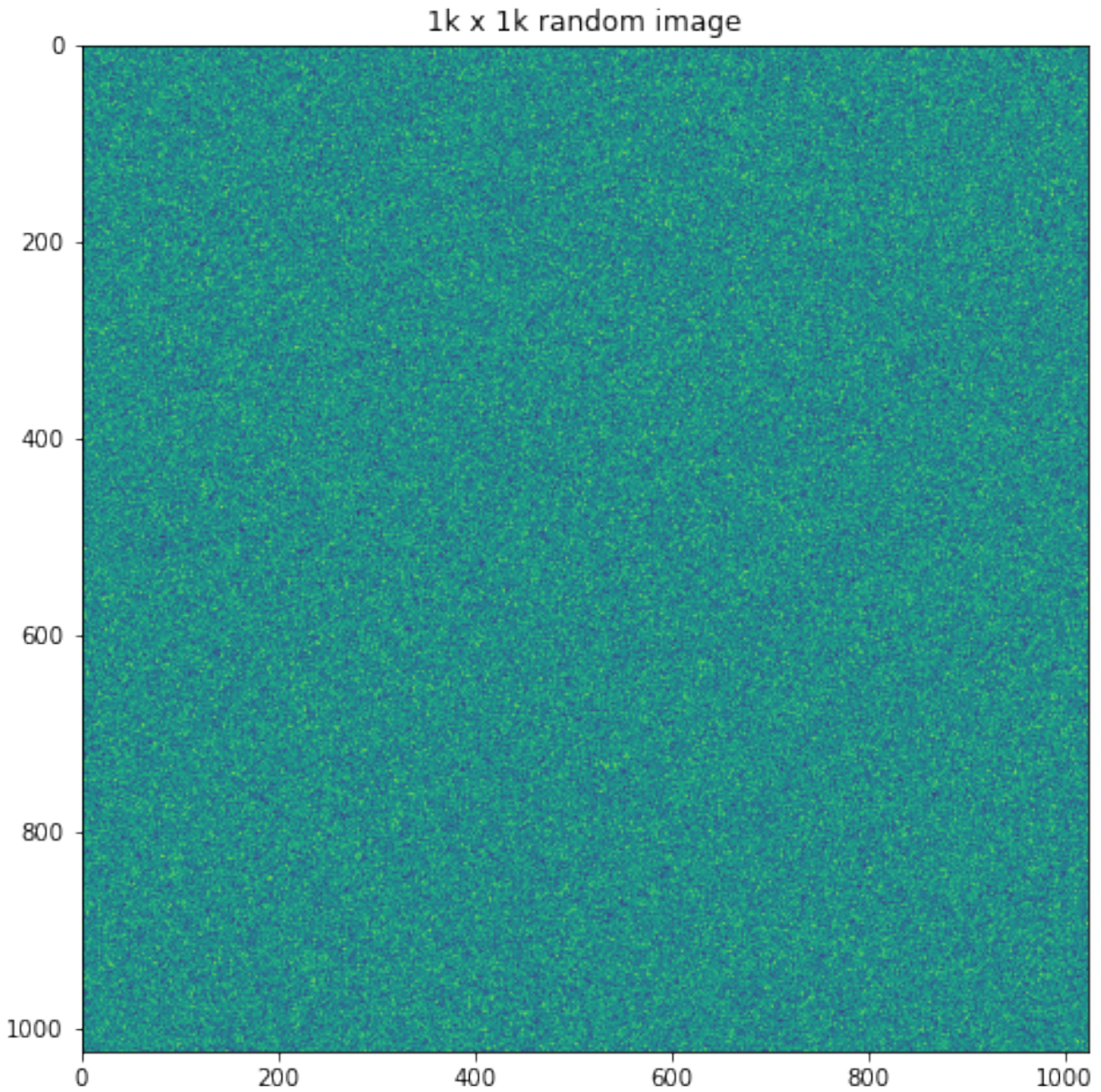
We have been talking about different data amounts of MB, GB, and TB. But, what does that really mean in reality? Let us explore what is a TB.

If **you** looked at one image with 1024 x 1024 pixels (1 Mpixels)

Here we create one image with 1000x1000 pixels with random values from a uniform distribution [0,1] and show it.

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(8,8))
plt.imshow(np.random.uniform(size = (1024, 1024)),
           cmap = 'viridis');
plt.title('1k x 1k random image');
```



every second, it would take you

```
# assuming 16 bit images and a 'metric' terabyte
OneTB      = 1e12
ImageSize  = 1000*1000*16/8
hour       = 60*60

time_per_tb = OneTB/ImageSize/hour
print("{0:0.1f} hours to view a terabyte".format(time_per_tb))
```

```
138.9 hours to view a terabyte
```

3.7.2 Overwhelmed scientist

Providing quantitative statements about image data is often very hard. You can may manage to do it on a single images like the bone image below.

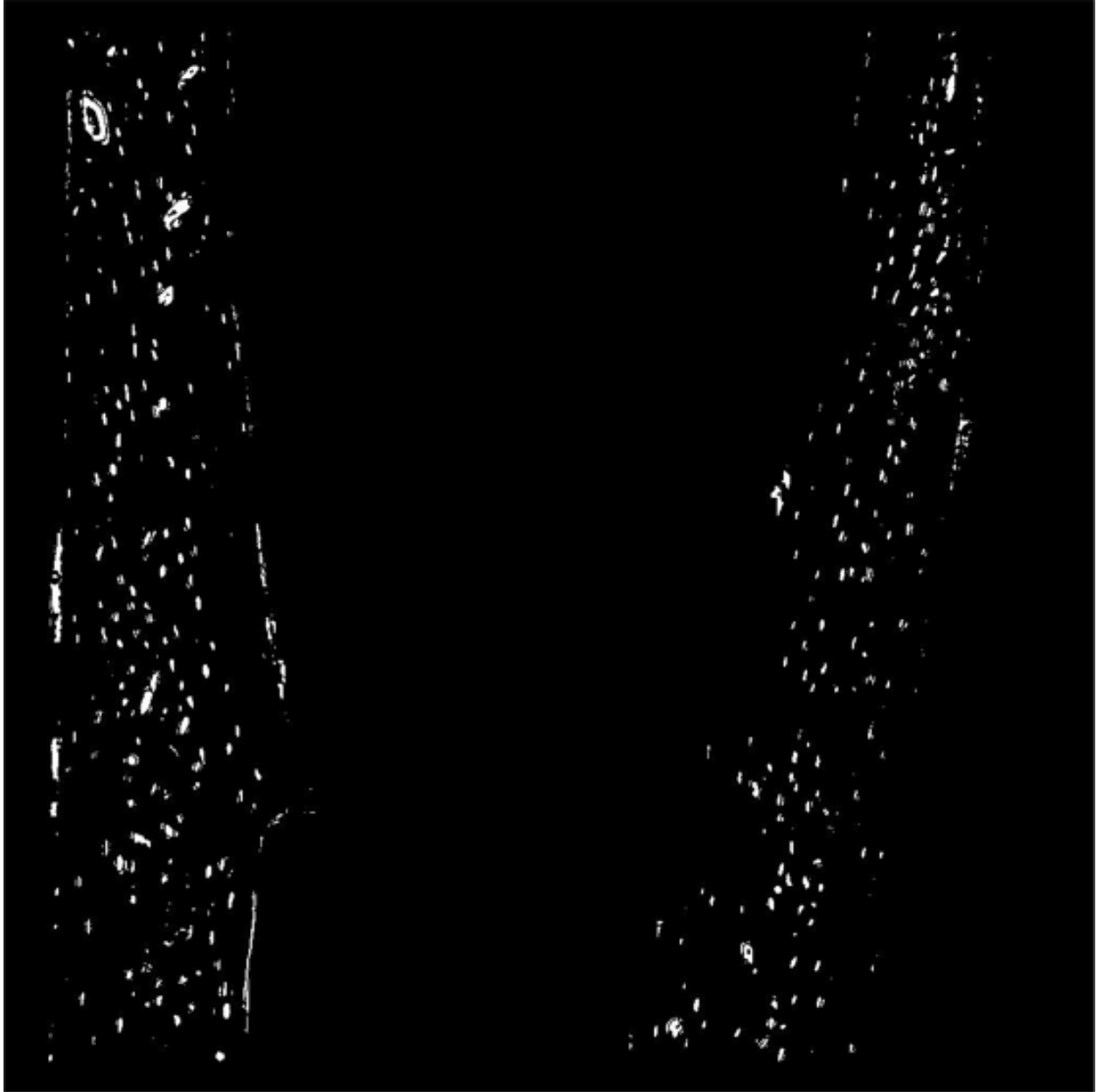


Fig. 3.7: A slice image show bone cells.

You would like to know:

- Count how many cells are in the bone slice.
- Ignore the ones that are 'too big' or shaped 'strangely'.
- Are there more on the right side or left side?
- Are the ones on the right or left bigger, top or bottom?

3.7.3 More overwhelmed scientist

Statistical analysis requires that you study many samples and not just a single one. The samples are also objects which requires 3D data instead of a single 2D slice.

Many samples are needed:

- Do it all over again for 96 more samples
- This time in 3D with 2000 slices instead of just one!



Fig. 3.8: A collection of 96 volume images from different bone samples.

Working with multiple 3D images is not feasible anymore to do manually. We need some kind of automated process to perform the analysis.

3.7.4 Bring on the pain

The 96 samples only represented one of our cases in the study. Now, if we want to study different ages, healthy/diseased, etc, we need to add a sample batch for each case. Maybe we even need to increase the number of samples in each test group. With all these variations, we can easily end up in a thousand samples to analyze.

Great variations in the population

- Now again with 1090 samples!
- How to measure?
- How to analyze?

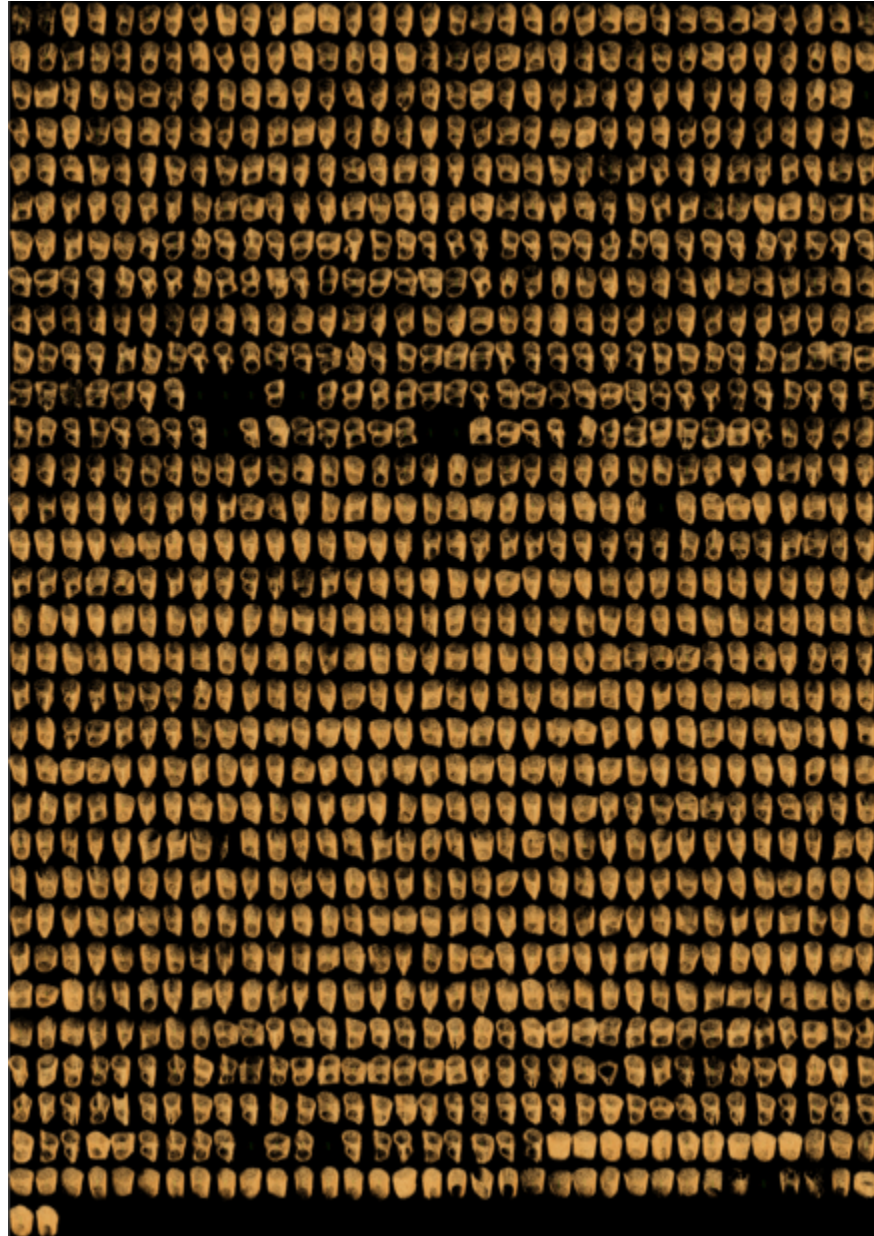


Fig. 3.9: A collection of 1090 bone samples. This is a massive task to analyze!

With so many samples we stand in front of a logistic problem to measure the data and once the data is there we have to analyze it. As a first step, we have to specify how to analyze these images to obtain results that may or may not support a hypothesis.

3.7.5 It gets better

The metrics we specified in the previous example are easy to observe and also to measure. They are direct measurements of pixels and positions. What if we now want to make more complicated inquiries even. Now how do we categorize the images or collections of features using soft metrics?

- Those metrics were quantitative and could be easily visually extracted from the images
- What happens if you have *softer* metrics
 - How aligned are these cells?
 - Is the group on the left more or less aligned than the right?
 - errr?

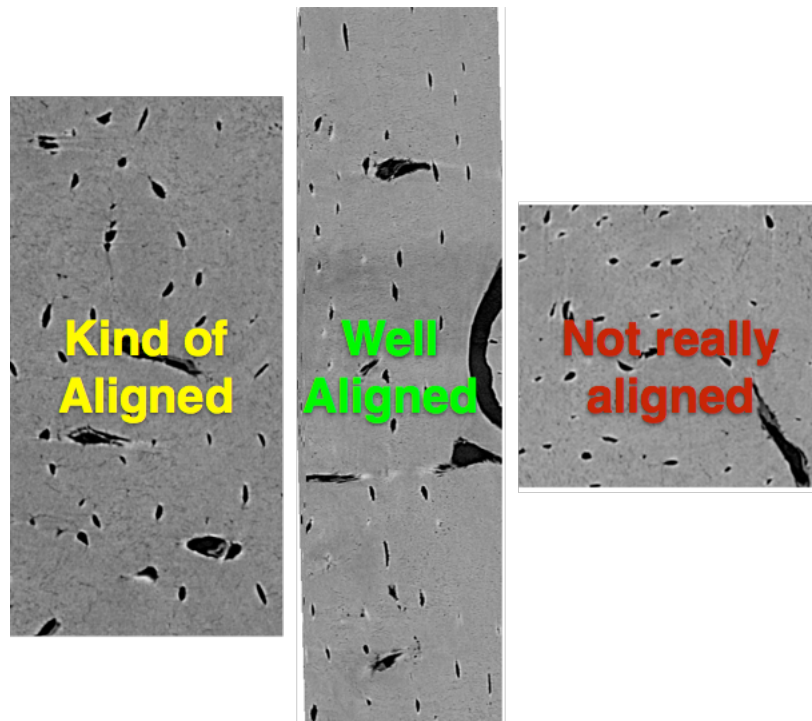


Fig. 3.10: Close-up on different bone segments. How aligned are the cells in these images?

3.7.6 Dynamic Information

Many experiments are on top of the spatial dimensions also studies over time. This brings us 4D data sets to analyze. How are we supposed to handle this? Looking at the movie we

- How many bubbles are here?
- How fast are they moving?
- Do they all move the same speed?
- Do bigger bubbles move faster?
- Do bubbles near the edge move slower?
- Are they rearranging?

BRINGING IN THE SCIENTIFIC METHOD

4.1 What is the purpose?

- Discover and validate new knowledge

4.1.1 How?

- Use the scientific method as an approach to convince other people
- Build on the results of others so we don't start from the beginning

4.1.2 Important Points

- While **qualitative** assessment is important, it is difficult to reliably produce and scale
- **Quantitative** analysis is far from perfect, but provides metrics which can be compared and regenerated by anyone

Inspired by: [imagej-pres](#)

4.2 Science and Imaging

Images are great for qualitative analyses since our brains can quickly interpret them without large *programming* investments.

4.2.1 Proper processing and quantitative analysis is however much more difficult with images.

- If you measure a temperature, quantitative analysis is easy, 50K.
- If you measure an image it is much more difficult and much more prone to mistakes, subtle setup variations, and confusing analyses

4.2.2 Furthermore in image processing there is a plethora of tools available

- Thousands of algorithms available
- Thousands of tools
- Many images require multi-step processing
- Experimenting is time-consuming

4.3 Important questions



Fig. 4.1: A realistic rendering of an ancient gold pendant.

- Nice renderings is mostly not sufficient for a publication
- It is important to define how you want to use your images

4.3.1 Initial questions

You need to ask yourself a number of questions before you start to plan an experiment that involves images. It is good to at least have a plan about how you are going to analyze your images once you have them. It is not always obvious how you get from the experiment idea, to performing the experiment and finally how you perform the analysis to obtain the information you are really interested in for your research project.

- What am I looking for?
- Is my original question specific or too general?
- How can I plan my experiment to make the analysis easier?
- How can I transfer the original question into the vocabulary of image analysis?
- How much a priori information about the sample do I have? Can it be used for the analysis?

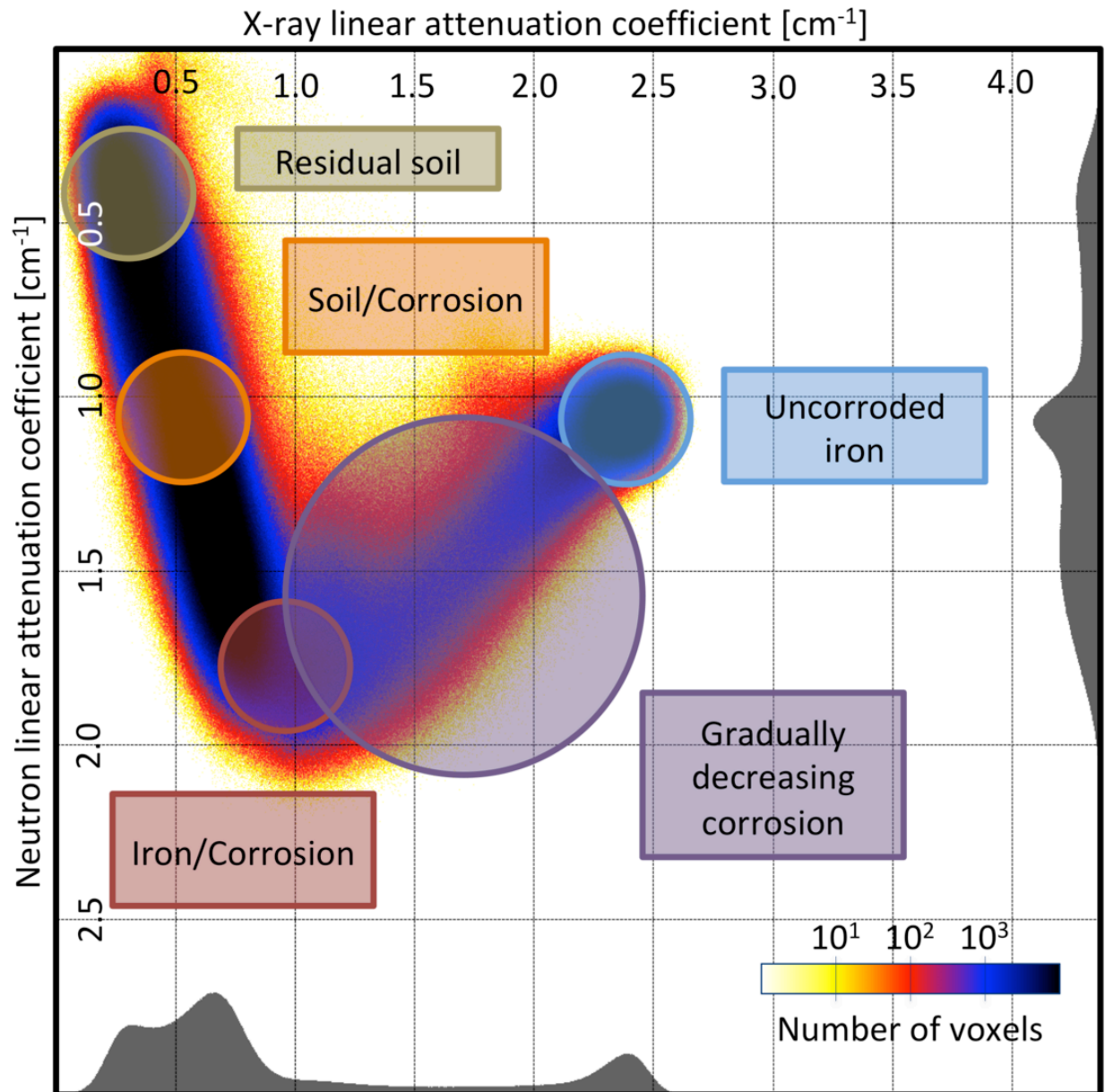


Fig. 4.2: Material analysis of a rusty roman nail.

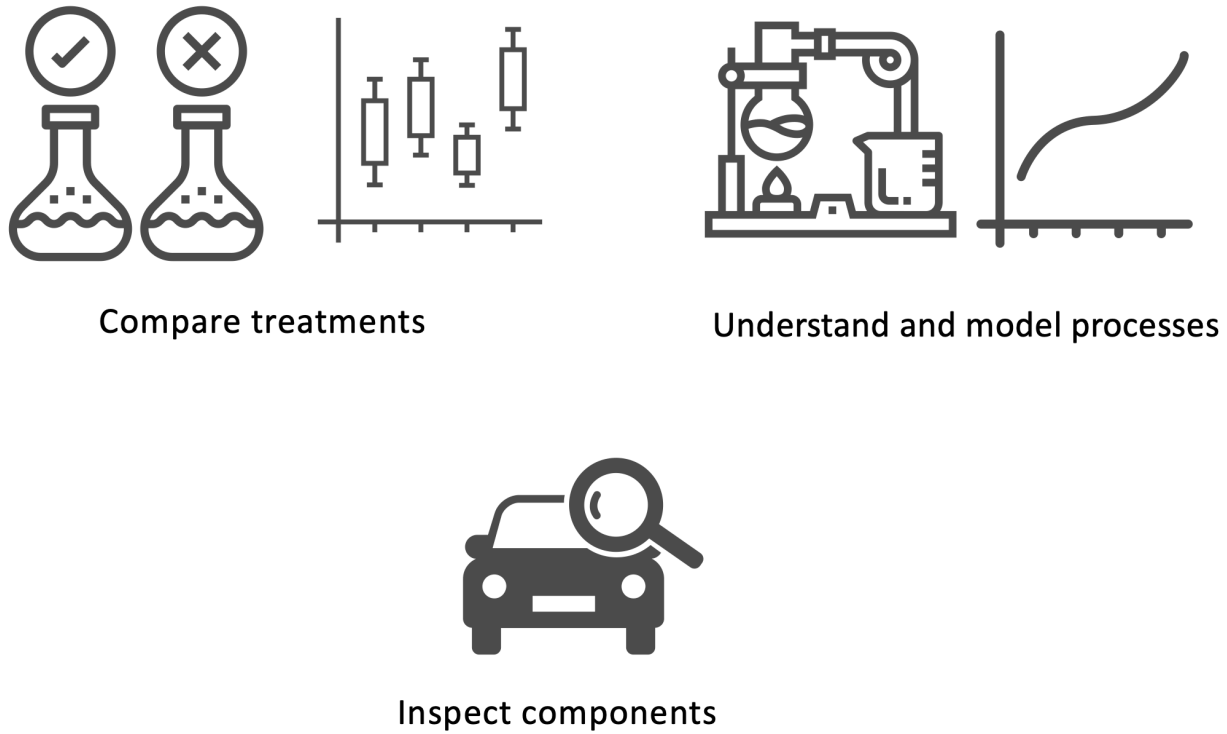


Fig. 4.3: Things a scientist wants to know from an experiment.

4.3.2 What is the purpose of the experiment

- 3D visualization
- Sample characterization
- Model process parameters
- etc.

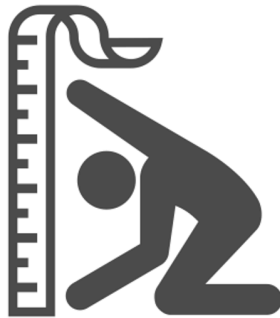
4.3.3 What can we measure?

There is a leap between what you really want to know and what we actually can measure in the images.

Quantification from gray values	Quantification from shapes
Material composition	Identify and count items
Material transport	Measure volume
	Characterize shape



Find hidden objects



Dimensions



Counting objects



Mixing ratios



Track changes

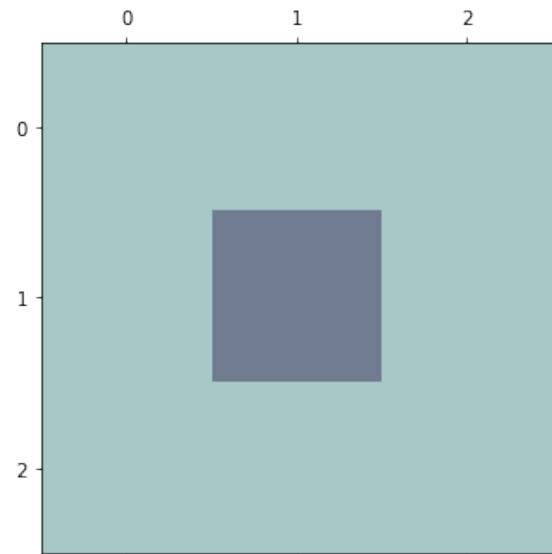
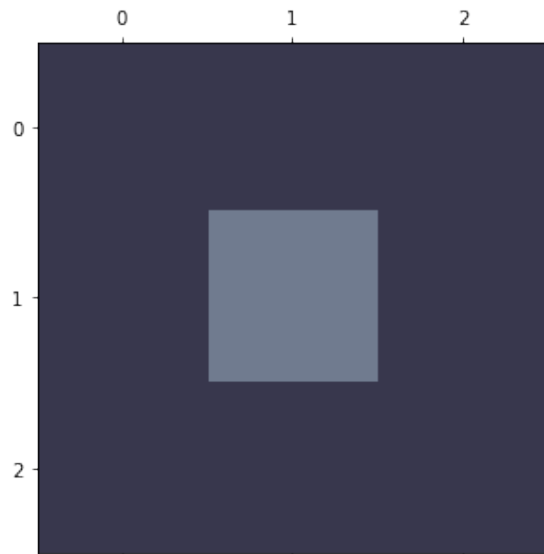
Fig. 4.4: These are things we can measure in images.

4.4 Why quantitative?

4.4.1 Human eyes have issues

Which center square seems brighter?

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
xlin = np.linspace(-1,1, 3)
xx, yy = np.meshgrid(xlin, xlin)
img_a = 25*np.ones((3,3))
img_b = np.ones((3,3))*75
img_a[1,1] = 50
img_b[1,1] = 50
fig, (ax1, ax2) = plt.subplots(1,2, figsize = (12, 5));
ax1.matshow(img_a, vmin = 0, vmax = 100, cmap = 'bone');
ax2.matshow(img_b, vmin = 0, vmax = 100, cmap = 'bone');
```



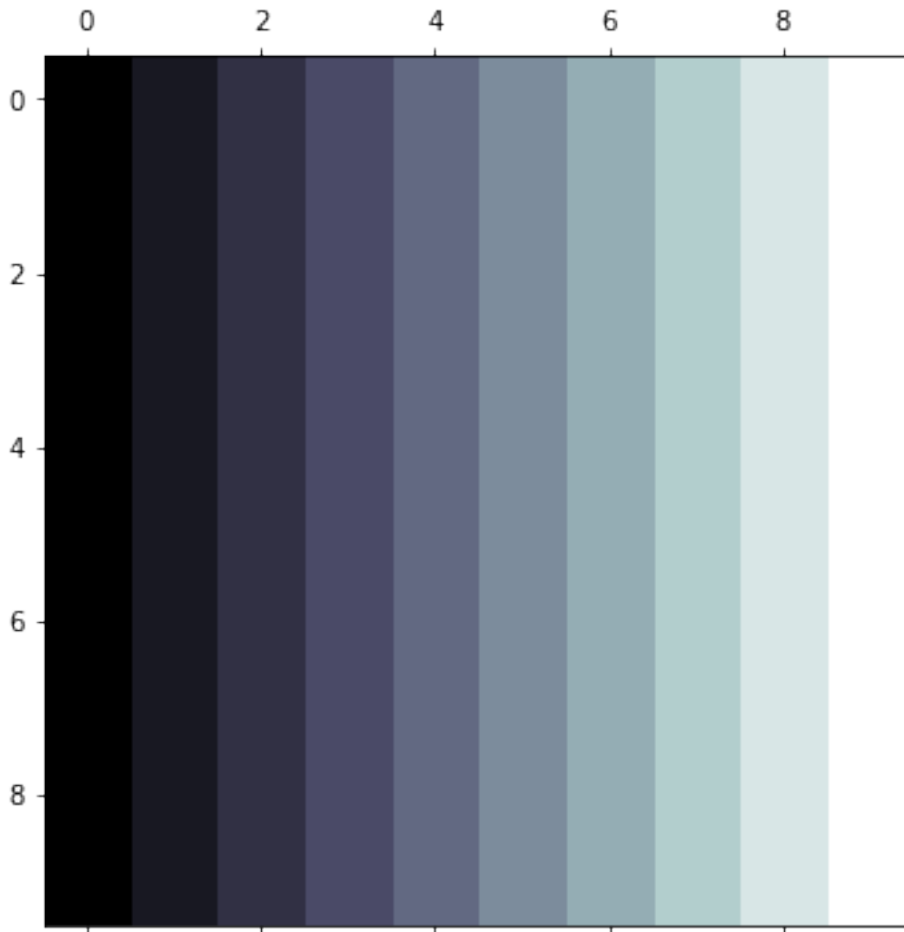
We want to convince with statistics not subjective descriptions.

4.5 Intensity gradients

Are the intensities constant in the image?

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
xlin = np.linspace(-1,1, 10)
xx, yy = np.meshgrid(xlin, xlin)

fig, ax1 = plt.subplots(1,1, figsize = (6, 6))
ax1.matshow(xx, vmin = -1, vmax = 1, cmap = 'bone');
```



Gradients are even harder to describe.

4.6 Selecting the right tool for your task

4.6.1 Criteria

There are many questions you have to ask about the data and analysis task you are facing. Here are some relevant questions that will guide you towards the choice of tool(s) you need to use when you analyse the data.

- How many samples?
- How complex is the sample?
- Is human interpretation needed?
- What is the end product?
- Are there methods tools available?
- Will there be more similar data sets?

4.6.2 The choice

You have different options to choose the tool for your analysis.

- Interactive tools
- Scripting using existing toolboxes
- Development of new algorithms Which one you actually select for your analysis workflow depends on many factors like the criteria listed in the previous section, but also on your experience, preferences in the group you are working with and so on.

In general, there is **no golden recipe!** You have to be flexible from task to task.

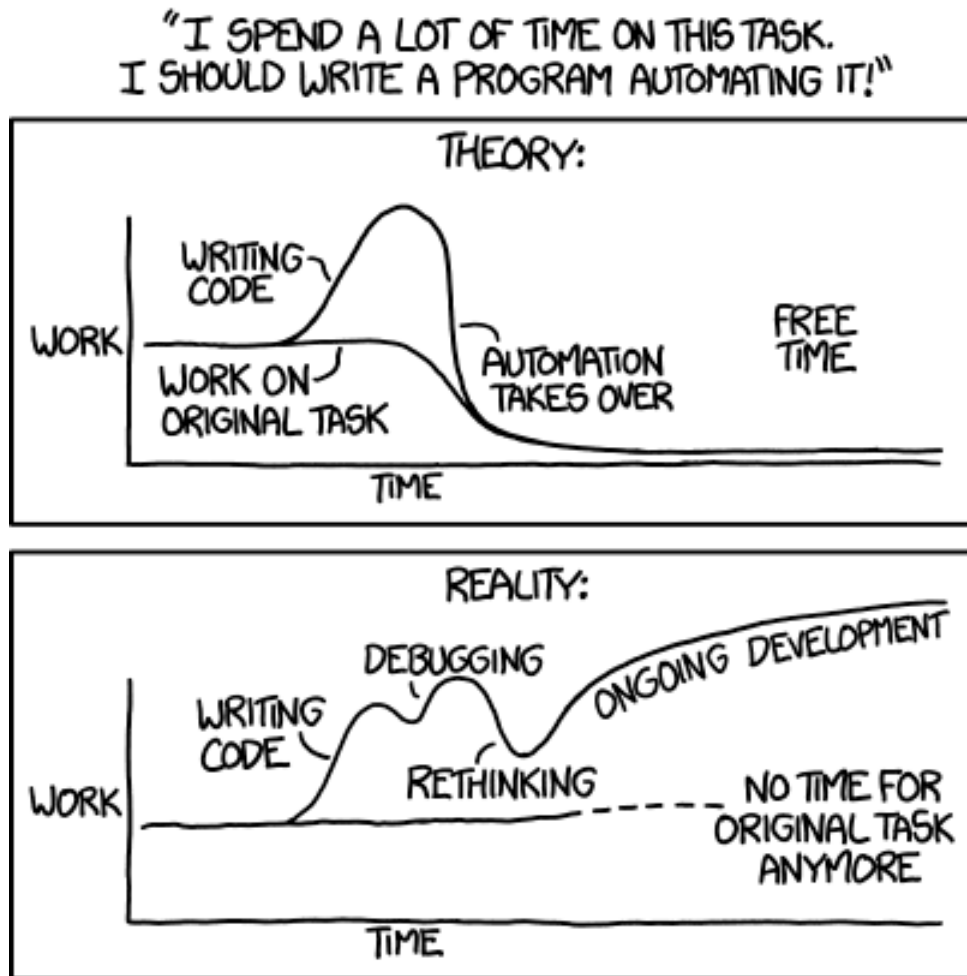


Fig. 4.5: It sometimes feels deceptively simple to implement a script that automates your task... be careful!

Criteria

- How many samples?
- How complex is the sample?
- Is human interpretation needed?
- What is the end product?
- Are there methods tools available?

- Will there be more similar data sets?

The choice

- Interactive tools
- Scripting using existing toolboxes
- Development of new algorithms

4.7 Reproducibility vs. Repeatability

4.7.1 Reproducibility

To reproduce an experiment means that someone else can repeat your entire workflow from sample preparation and experiment to the analysis to obtain the same results you got in your report. This should be possible completely independently for an experiment to be called reproducible.

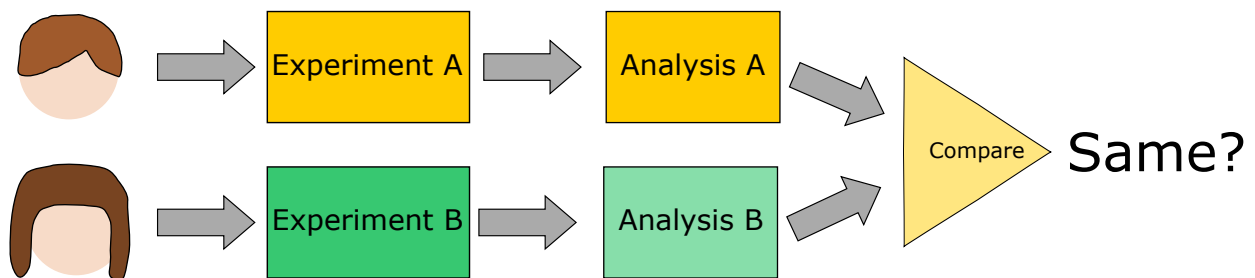


Fig. 4.6: A workflow describing the concept of reproducibility.

4.7.2 Repeatability

Repeatability is a more relaxed concept. It requires that someone else (or even yourself) can obtain the same results multiple time using the same data and analysis workflow. This means that there must very little room left for individual decisions that may have impact on the outcome.

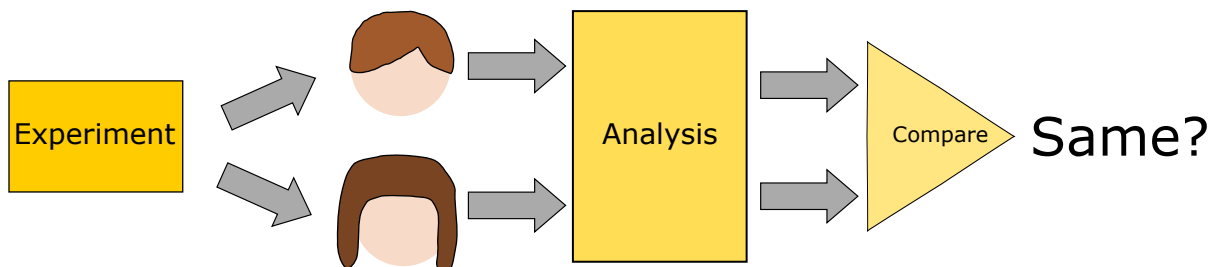


Fig. 4.7: A workflow describing the concept of reproducibility.

4.7.3 Reproducibility vs. Repeatability

Science demands **repeatability!** and really wants **reproducibility**

- Experimental conditions can change rapidly and are difficult to make consistent
 - Animal and human studies are prohibitively time consuming and expensive to reproduce
 - Terabyte datasets cannot be easily passed around many different groups
 - Privacy concerns can also limit sharing and access to data
-

- *Science* is already difficult enough
- Image processing makes it even more complicated
- Many image processing tasks are multistep, have many parameters, use a variety of tools, and consume a very long time

4.7.4 How can we keep track of everything for ourselves and others?

We can make the data analysis easy to repeat by an independent 3rd party

- Document the analysis steps
- Write clear and understandable code

DIFFERENT VIEWS ON IMAGE ANALYSIS

Image analysis is a complex task and there are many ways to reach the quantitative results from the images.

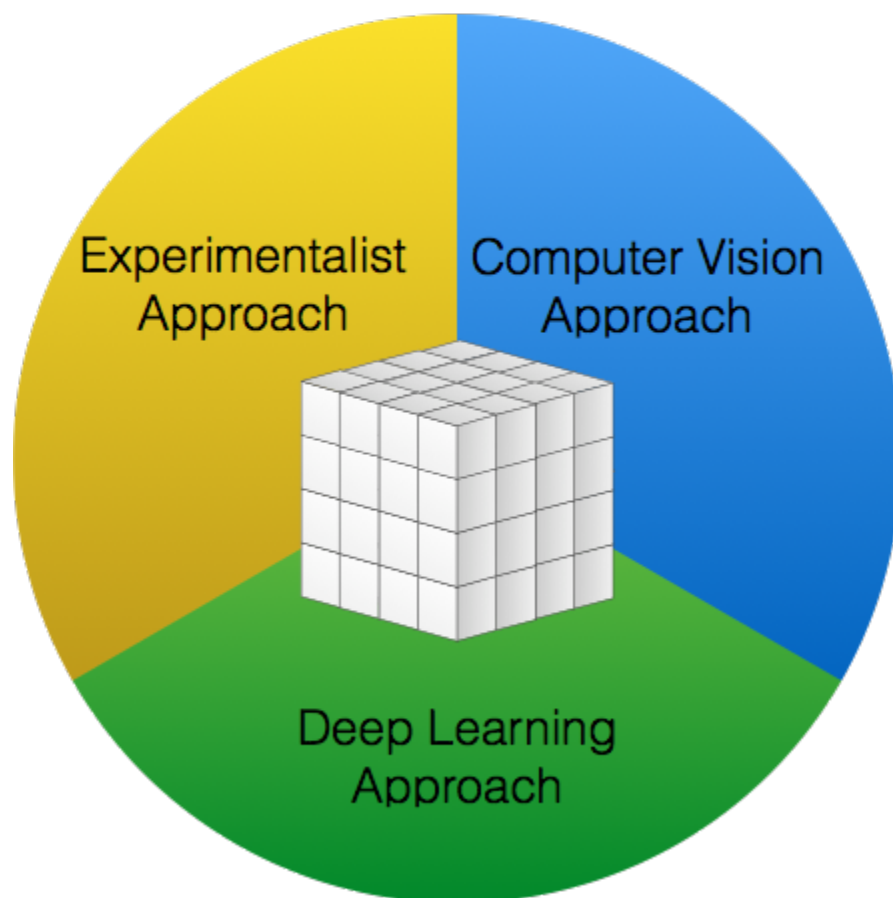


Fig. 5.1: Your background often decides how you approach an image analysis problem.

We can make two initial statements:

- An image is a bucket of pixels.
- How you choose to turn it into useful information is strongly dependent on your background

5.1 Image Analysis: Experimentalist

The experimentalist looks with a problem driven concept on the analysis task. It is often a top down approach aiming at solving the specific problem at hand. The solution is often reality driven and aims at finding models explaining the information presented in the images.

Typical task the experimentalist tries to solve are very practical and specific like counting cells in the image or to measure the porosity of the sample.

Characteristics

- Problem-driven
- Top-down
- *Reality* Model-based

Examples

- cell counting
- porosity

5.2 Image Analysis: Computer Vision Approaches

The computer vision/signal processing scientist works to develop methods to solve a class of image processing problem. The approach is based on abstract features found in the image. The models are based on features and noise found in the images. The systematic approach is even based on engineered image features to better test and evaluate the developed methods.

The computer vision approach is typical looking to detect features like edges, structures, and also complicated features like faces.

Characteristics

- Method-driven
- Feature-based
- *Image* Model-based
- Engineer features for solving problems

Examples

- Edge detection
- Face detection

5.3 Image Analysis: Deep Learning Approach

Finally, the deep learning approach is data driven and inspired by the way nature solves the image analysis problem. This approach rebuilds the way image processing is done from scratch, but at the same time it also based on concepts developed in computer vision. The deep learning approaches doesn't require a specific model describing the images it is meant to analyze, but rather make conclusions based on the previous images it has been exposed to.

The deep learning approach is good handling rare events in the data and when it trained correctly it is also capable of generalizing to detect new features. This may sound like magic, but this is also a well founded and structured approach to

analyzing images. Care must however be taken not to over fit or to generalize too much. The models are never better than the data they have been exposed to.

Examples where deep learning is frequently used are to detect anomalies in the data or to label images based on their contents.

Characteristics

- Results-driven
- Biology ‘inspired’
- Build both image processing and analysis from scratch

Examples

- Captioning images
- Identifying unusual events

5.4 Summary analysis approaches

These three approaches have their own advantages and disadvantages, therefore it is good to know them all to be able to select the adequate method for the task you have to solve. It is not unusual that you will have to use a mix of the approaches. It is important to be open minded and think outside the box. In the end, what matters is that you can provide a reliable analysis of your data.

5.5 Image Formation

The image formation process is the process to use some kind of excitation or impulse probe a sample. This requires the interaction of four parts.

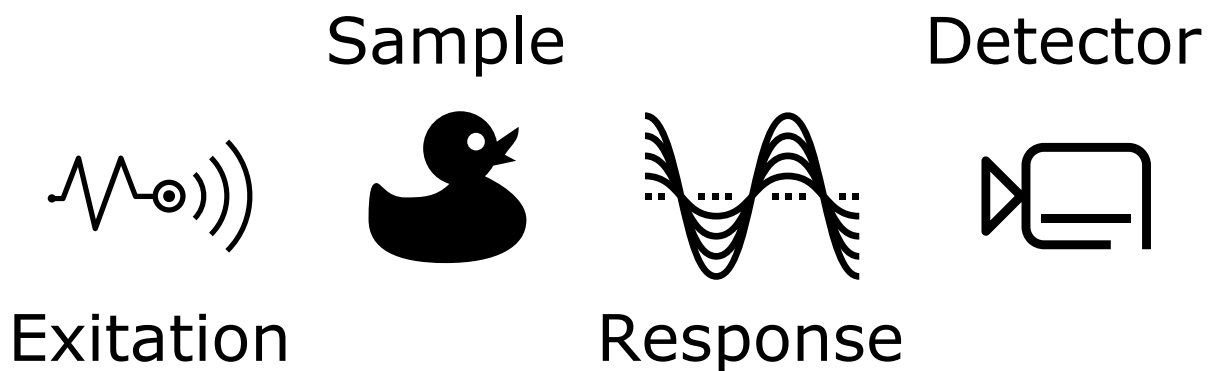


Fig. 5.2: The parts involved in the image formation process probing a sample.

- **Impulses** Light, X-Rays, Electrons, A sharp point, Magnetic field, Sound wave
- **Characteristics** Electron Shell Levels, Electron Density, Phonons energy levels, Electronic, Spins, Molecular mobility
- **Response** Absorption, Reflection, Phase Shift, Scattering, Emission
- **Detection** Your eye, Light sensitive film, CCD / CMOS, Scintillator, Transducer

5.5.1 Where do images come from?

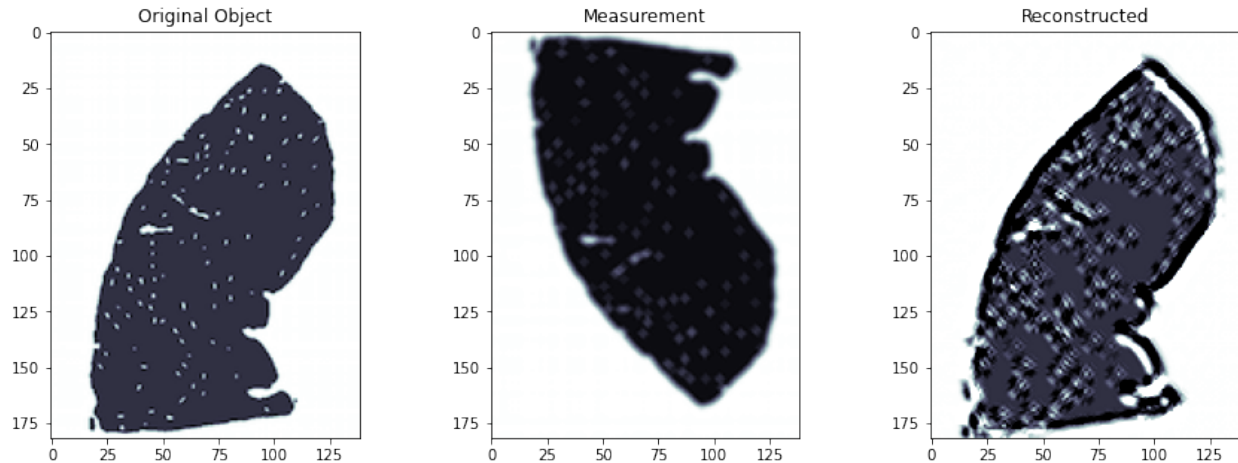
Modality	Impulse	Characteristic	Response	Detection
Light Microscopy	White Light	Electronic interactions	Absorption	Film, Camera
Phase Contrast	Coherent light	Electron Density (Index of Refraction)	Phase Shift	Phase stepping, holography, Zernike
Confocal Microscopy	Laser Light	Electronic Transition in Fluorescence Molecule	Absorption and reemission	Pinhole in focal plane, scanning detection
X-Ray Radiography	X-Ray light	Photo effect and Compton scattering	Absorption and scattering	Scintillator, microscope, camera
Neutron Radiography	Neutrons	Interaction with nucleus	Scattering and absorption	Scintillator, optics, camera
Ultrasound	High frequency sound waves	Molecular mobility	Reflection and Scattering	Transducer
MRI	Radio-frequency EM	Unmatched Hydrogen spins	Absorption and reemission	RF coils to detect
Atomic Force Microscopy	Sharp Point	Surface Contact	Contact, Repulsion	Deflection of a tiny mirror

5.6 Acquiring Images

5.6.1 Traditional / Direct imaging

- Visible images produced or can be easily made visible
- Optical imaging, microscopy

```
bone_img = imread('figures/tiny-bone.png').astype(np.float32)
# simulate measured image
conv_kern = np.pad(disk(2), 1, 'constant', constant_values = 0)
meas_img = convolve(bone_img[:-1], conv_kern)
# run deconvolution
dekern = np.fft.ifft2(1/np.fft.fft2(conv_kern))
rec_img = convolve(meas_img, dekern)[:-1]
# show result
fig, (ax_orig, ax1, ax2) = plt.subplots(1,3, figsize = (15, 5))
ax_orig.imshow(bone_img, cmap = 'bone'); ax_orig.set_title('Original Object')
ax1.imshow(np.real(meas_img), cmap = 'bone'); ax1.set_title('Measurement')
ax2.imshow(np.real(rec_img), cmap = 'bone', vmin = 0, vmax = 255); ax2.set_title(
    'Reconstructed');
```



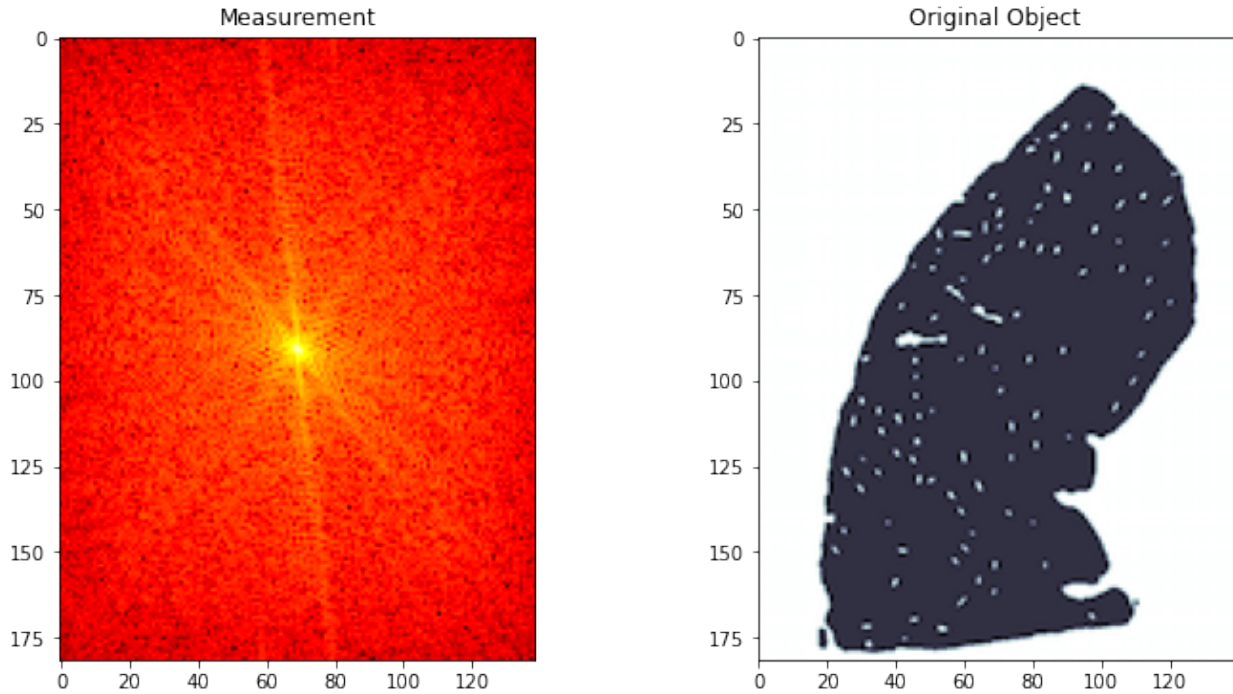
5.6.2 Indirect / Computational imaging

- Recorded information does not resemble object
- Response must be transformed (usually computationally) to produce an image

```
bone_img = imread('figures/tiny-bone.png').astype(np.float32)
# simulate measured image
meas_img = np.log10(np.abs(np.fft.fftshift(np.fft.fft2(bone_img))))
print(meas_img.min(), meas_img.max(), meas_img.mean())
fig, (ax1, ax_orig) = plt.subplots(1,2,
                                   figsize = (12, 6))
ax_orig.imshow(bone_img, cmap = 'bone')
ax_orig.set_title('Original Object')

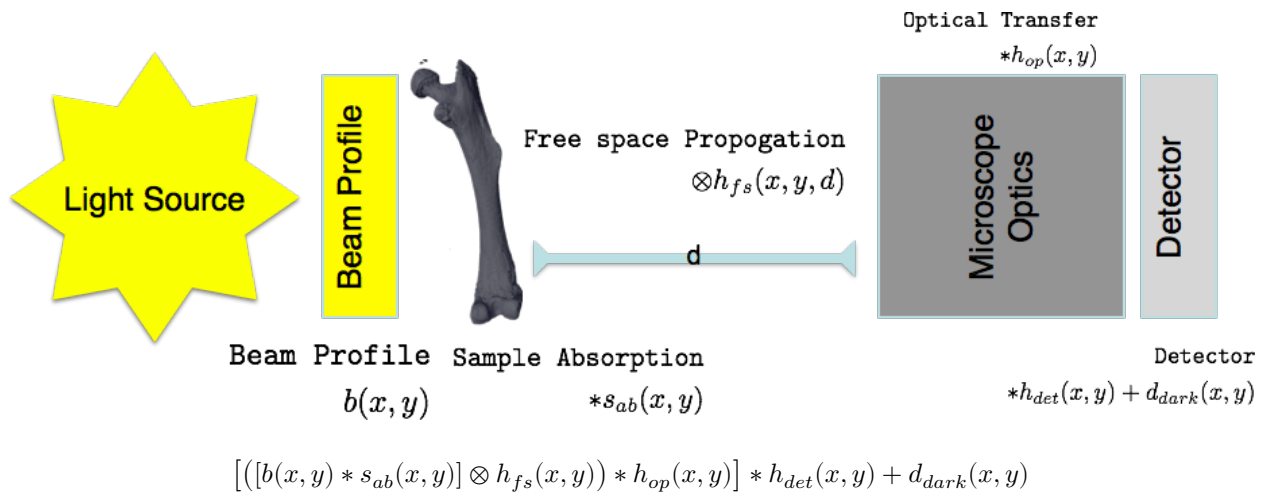
ax1.imshow(meas_img, cmap = 'hot')
ax1.set_title('Measurement');
```

```
1.146423838816545 6.61125552595089 3.3563935033662253
```



5.6.3 Traditional Imaging

5.7 Traditional Imaging: Model



s_{ab} is the only information you are really interested in, so it is important to remove or correct for the other components

For color (non-monochromatic) images the problem becomes even more complicated

$$\int_0^\infty [[[b(x, y, \lambda) * s_{ab}(x, y, \lambda)] \otimes h_{fs}(x, y, \lambda)] * h_{op}(x, y, \lambda)] * h_{det}(x, y, \lambda) d\lambda + d_{dark}(x, y)$$

5.7.1 Indirect Imaging (Computational Imaging)

With indirect imaging you make acquisitions in a form that don't represent the information you want to have. It is needed to perform a numeric transformation to obtain images in observation space.

Some examples are:

- Tomography through projections
- Microlenses [Light-field photography](#)
- Diffraction patterns
- Hyperspectral imaging with Raman, IR, CARS
- Surface Topography with cantilevers (AFM)
- Tomography through projections
- Microlenses [Light-field photography](#)
- Diffraction patterns
- Hyperspectral imaging with Raman, IR, CARS
- Surface Topography with cantilevers (AFM)

6.1 An introduction to images

6.1.1 What is an image?

A very abstract definition:

- **A pairing between spatial information (position)**
- **and some other kind of information (value).**

In most cases this is a 2- or 3-dimensional position with

- x,y,z coordinates on a grid
- and a numeric value (intensity)

6.1.2 Image sampling

The world is	The computer needs
Continuous	Discrete levels
No boundaries	Limited extent

6.1.3 What does spatial sampling mean

```
img=np.load('../..../common/data/wood.npy');
plt.figure(figsize=[15,7])
plt.subplot(2,3,1); plt.imshow(img); plt.title('Original')
downsize = 2; plt.subplot(2,3,2); plt.imshow(resize(img,(img.shape[0] // downsize,
↳img.shape[1] // downsize), anti_aliasing=False)); plt.title('Downsize {0}x{0}'.
↳format(downsize))
downsize = 32; plt.subplot(2,3,3); plt.imshow(resize(img,(img.shape[0] // downsize,
↳img.shape[1] // downsize), anti_aliasing=False)); plt.title('Downsize {0}x{0}'.
↳format(downsize))
levels = 16; plt.subplot(2,3,5); plt.imshow(np.floor(img*levels)); plt.title('{0}↳
↳Levels'.format(levels));
levels = 4 ; plt.subplot(2,3,6); plt.imshow(np.floor(img*levels)); plt.title('{0}↳
↳Levels'.format(levels));
```

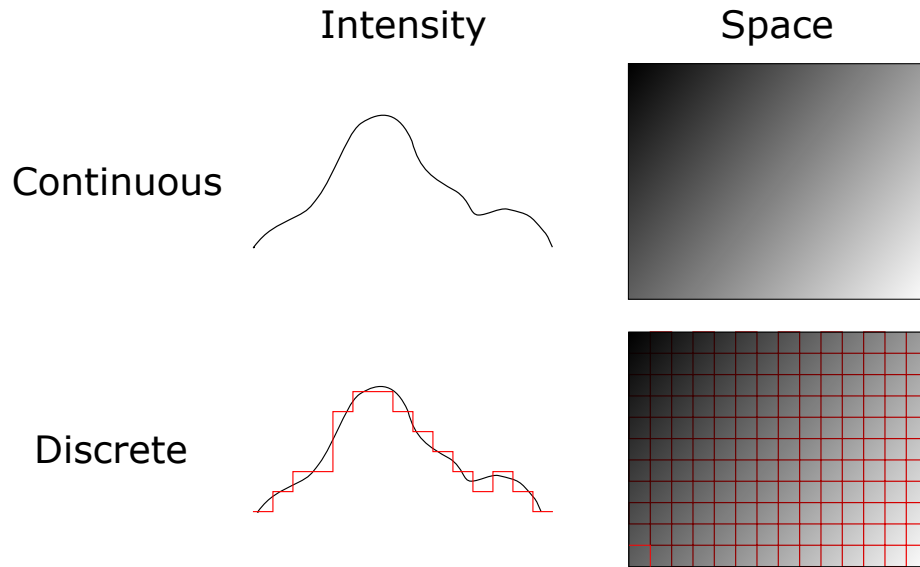
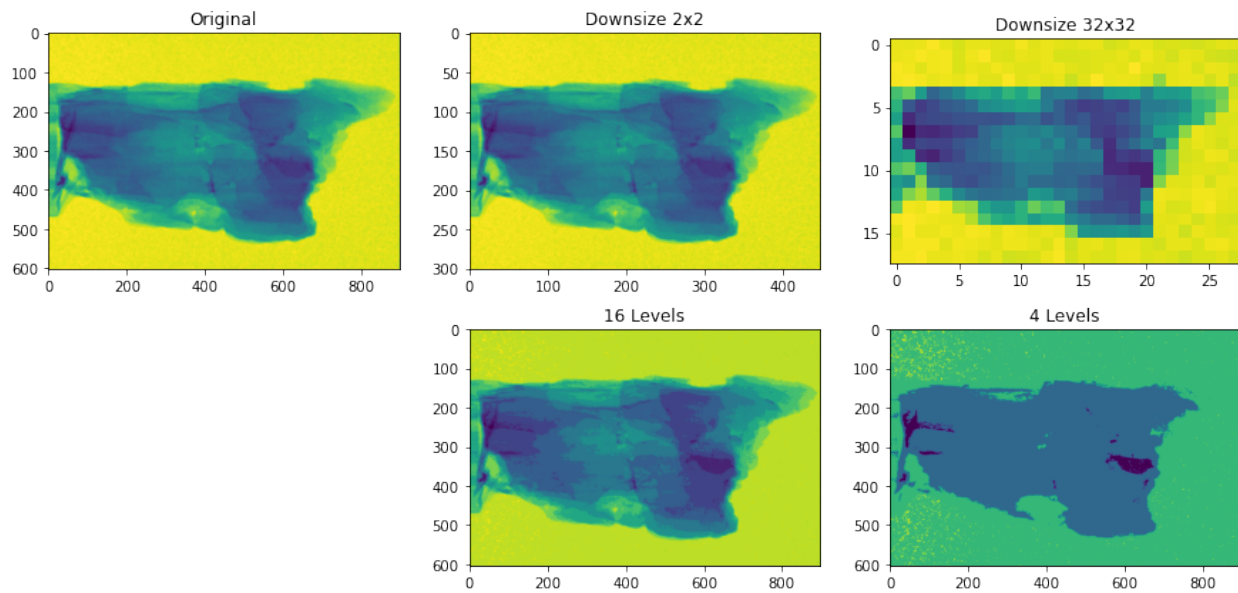


Fig. 6.1: The real world is sampled into discrete images with limited extent.



6.2 Pixel size and resolution

It is important to distinguish between pixel size and resolution

6.2.1 Pixel size

The pixel size is

- The sample pitch between two adjacent pixels
- The smallest area represented in the image

6.2.2 Resolution

The resolution is related to the optical system

- It is the effect of the optical transfer function of the acquisition system.
- Should have a greater value than the pixel size.
- Defines the smallest pixel size when you set up your acquisition conditions.

6.2.3 Demonstrating different pixel sizes

What happens when we represent the same image with less pixels?

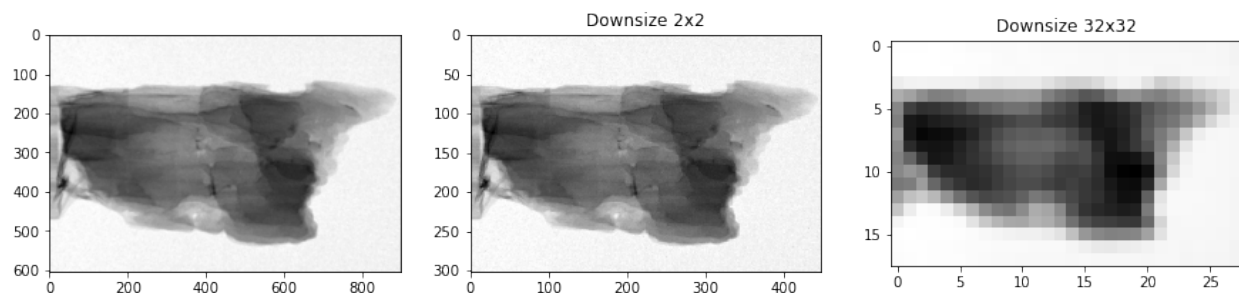
In this example we downsample the image first by a factor two. This change is barely visible when we show the image, but the number of pixels have reduced by a factor four. In the second example the image is downsampled by a factor 32 and you can clearly observe how pixelated the image is. At this level of downscaling, you can only see very coarse features in the sample.

Down scaling is sometimes used as a method to speed up the frame rate as it radically reduces the number of bytes to be transferred from the detector and also the amount of data to write on disk. You should however be careful not to down scale by a too great factor as you will lose spatial information when doing so.

```
img=np.load('../..//common/data/wood.npy');
fig,ax = plt.subplots(1,3, figsize=[15,5])
ax[0].imshow(img,cmap='gray'); plt.title('Original')

downsize = 2;
resized = resize(img,(img.shape[0] // downsize, img.shape[1] // downsize))
ax[1].imshow(resized, interpolation='None',cmap='gray'); ax[1].set_title('Downsize {0}x{0}'.format(downsize))

downsize = 32;
resized = resize(img,(img.shape[0] // downsize, img.shape[1] // downsize))
ax[2].imshow(resized,interpolation='None',cmap='gray'); ax[2].set_title('Downsize {0}x{0}'.format(downsize));
```



6.2.4 Edges at different resolutions and pixel sizes

Finding the correct pixel size is related to the resolution of the imaging system. You can sample low resolved scenes with many pixels but then the edges will appear blurred and will essentially waste a lot data on little added value.

The example below shows what an ideal edge would look like and what it mostly looks like when we acquire our images. As you can see, the “real” edge is represented by a smooth transition spread over several pixels.

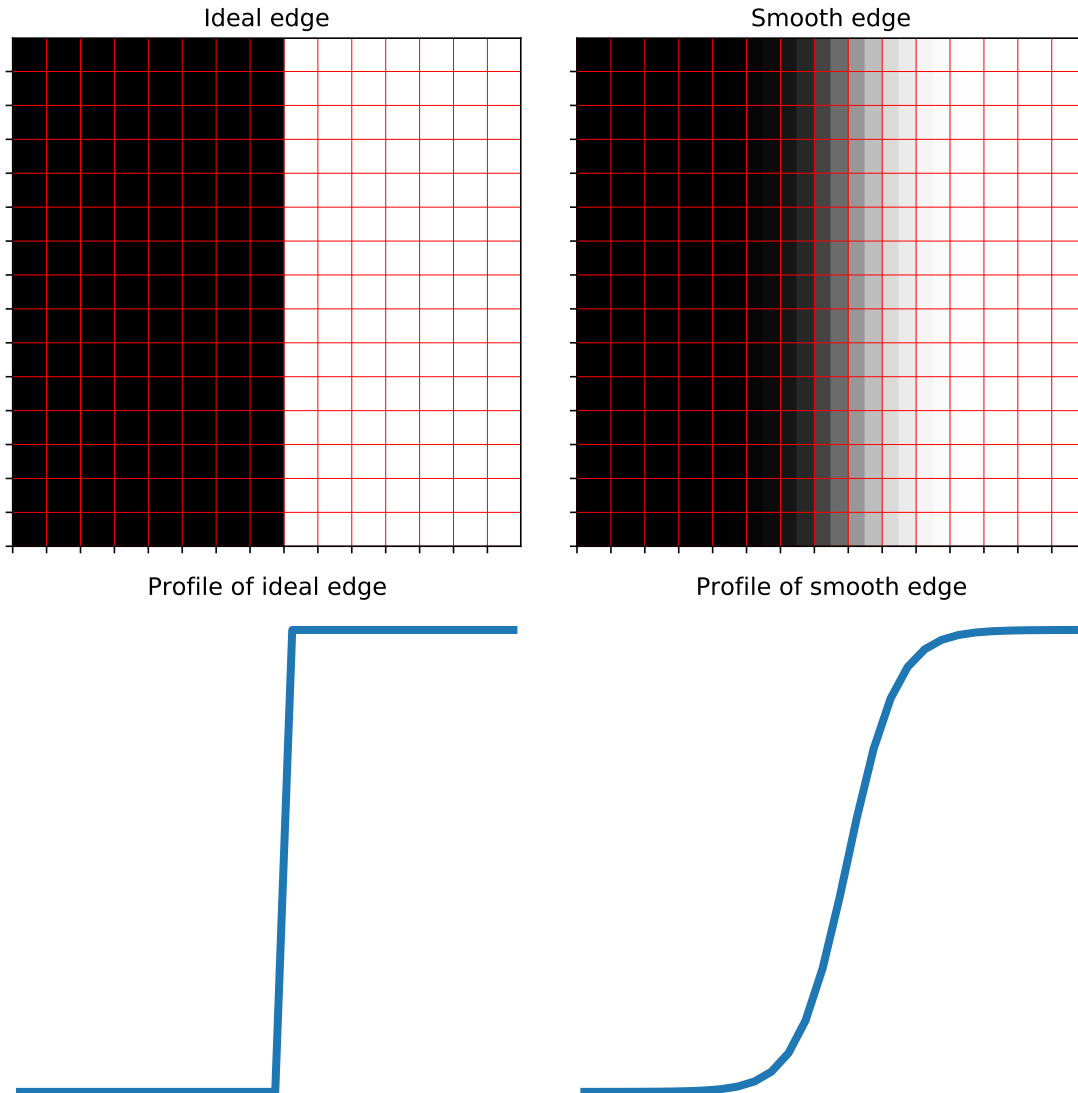


Fig. 6.2: Examples of edges sampled with different pixel sizes.

6.3 Image intensity

What happens when we reduce the number of gray-levels in the image?

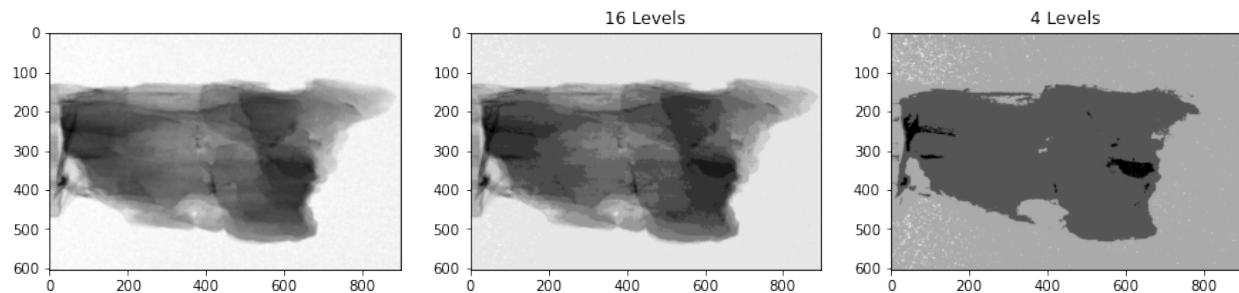
The image intensity is determined by the response function of the imaging system. In the case of neutron imaging we are talking about the transmission of the neutron beam through the sample. The transmission follows Beer-Lambert's law $I(x, y) = I_0(x, y)e^{-\int_L \mu(x) dx}$, this is only a simplified version. More complicated versions including the neutron energy are presented in other parts of this course.

The information captured by the detector is stored in digital form with different gray level dynamics. We are often talking about 8 or 16 bit integer when we store images. This means that each pixel can represent the measured intensity with either 256 or 65536 gray levels respectively. In the example below we demonstrate what happens when only very few gray levels are used.

```
img=np.load('../common/data/wood.npy');
fig,ax = plt.subplots(1,3,figsize=[15,7])
ax[0].imshow(img, cmap='gray'); plt.title('Original')

levels = 16;
lvl = np.floor(img*levels)
ax[1].imshow(lvl, cmap='gray'); ax[1].set_title('{0} Levels'.format(levels));

levels = 4 ;
lvl = np.floor(img*levels)
ax[2].imshow(lvl, cmap='gray'); ax[2].set_title('{0} Levels'.format(levels));
```



It is important to use as many gray levels at possible when you expose your images. The image turns patchy when you use too few levels which you can see in the example above. The patchiness reduces the precision of your evaluation, there is less margin to make estimations and decisions. The number of gray levels depend on many factors like:

- Exposure time
- Source intensity
- Conversion efficiency
- Conversion rate of the detector
- Pixel size

So, it is your task to optimize your acquisition to provide well illuminated images by changing these parameters. Some are easier than others to change and constraint are given by the type of investigation you are doing.

6.3.1 How many bits are needed?

The number of bits you need depends on:

- Contrast difference
- Separate many different sample features
- Sensitivity to rounding errors

The table below gives you an idea how many bits you need to represent your image information. In the extreme you would only need a single bit per pixel (8 pixels per byte) to represent a bi-level image from a segmentation. The other extreme would be to use double precision floating point that requires 64 bits (8 bytes) per pixels. Double precision is rarely needed and single precision is mostly sufficient which saves you memory. Saving memory is in particular important when you work with 3D images.

Few bits	Many bits	Floating point
High contrastClean imagesSegmented data	Low contrastNoisy images-Gradual changes	High intensity dynamicsQuantification to physical propertiesIn algorithms

6.3.2 The histogram

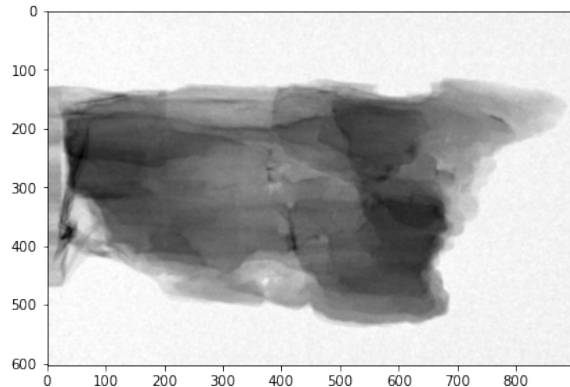
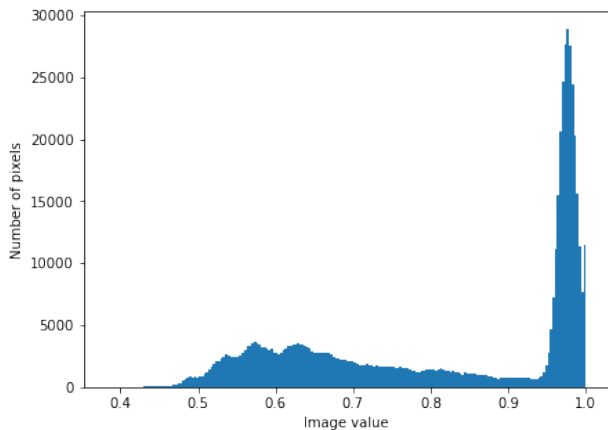
The histogram is a statistical tool to show frequency of each graylevel in the image.

It is essentially a plot where the you count how many times each gray level appears in the image. For a 16-bit image this would result 65565 points. This is far too detailed therefore it is common to use bins of several gray level to reduce the level of detail in the histogram and also improve the readability. In the example below we use 100 histogram bins which look quite reasonable for this image. Chosing the number of bins depends on the image size too. Your histogram doesn't look very useful if you have too many bins compared to the available number of pixels. You can use the piece of code below to explore what happens when you change the number of bins.

```
fig,ax = plt.subplots(1,2,figsize=(15,5))

# Compute and show a histogram
ax[0].hist(img.ravel(),bins=200)

ax[0].set_xlabel('Image value'), ax[0].set_ylabel('Number of pixels')
ax[1].imshow(img,cmap='gray');
```



It can be used for the analysis of the image as it gives you an idea which values are related to different features in the image. The histogram tells you the area covered by a give pixel value and a later section we will see how the histogram can be used to segment the images.

6.3.3 Details of the histogram

Let's look at different regions in the image and their representation in the histogram.

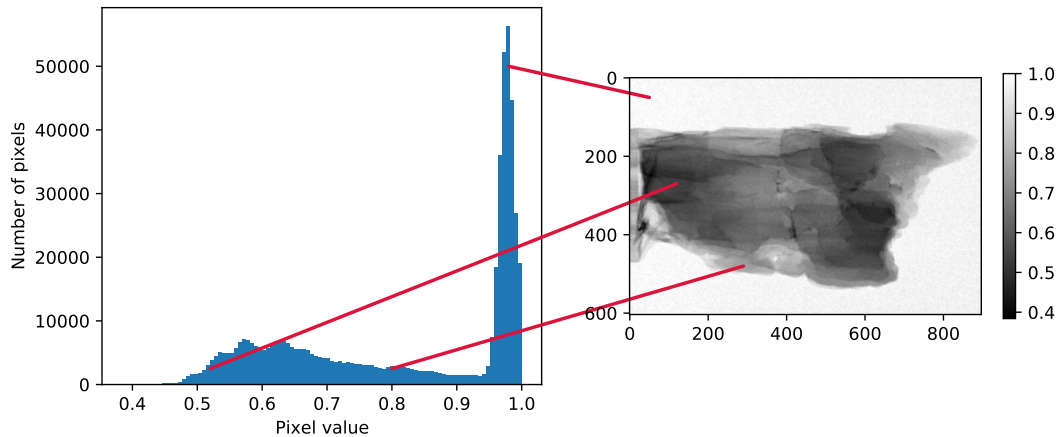


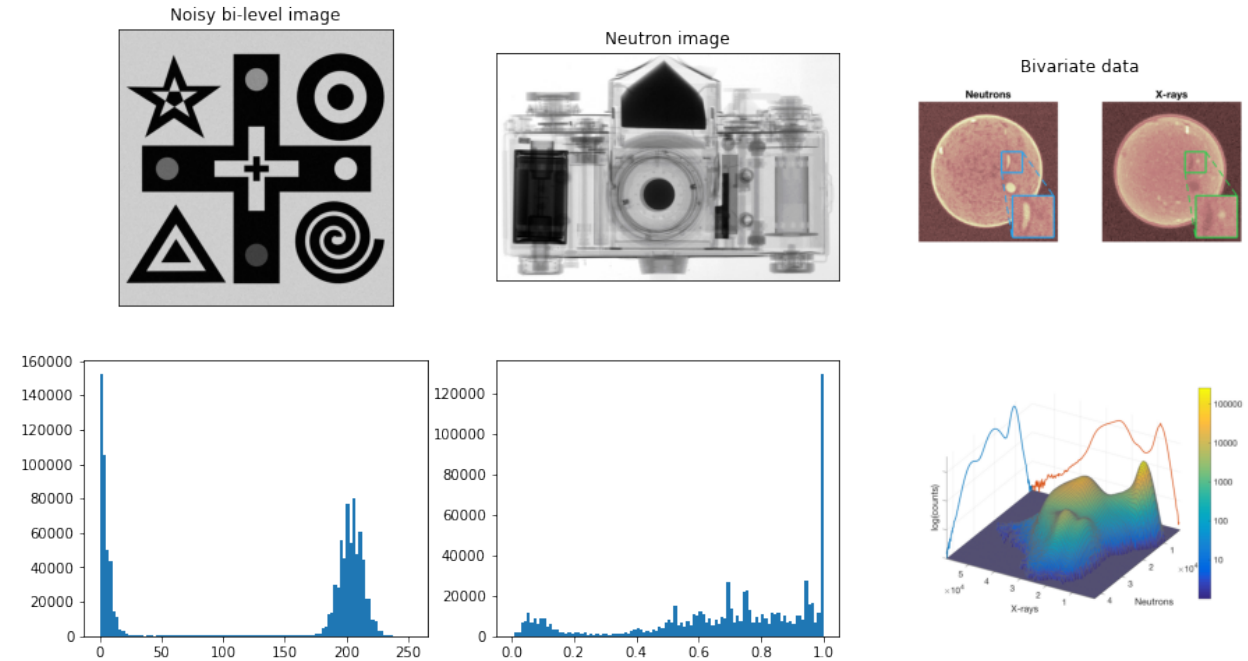
Fig. 6.3: Regions in the image connected to their position in the histogram.

6.3.4 Histogram examples

```
a=plt.imread('figures/testpattern_noisy.jpg')
b=plt.imread('figures/neutron_camera.png')
c0=plt.imread('figures/root_slices.png')
c1=plt.imread('figures/root_histogram.png')

fig,ax=plt.subplots(2,3,figsize=(15,8))
ax=ax.ravel()

ax[0].imshow(a,cmap='gray')
ax[0].set_xticks([]);ax[0].set_yticks([])
ax[0].set_title('Noisy bi-level image')
ax[3].hist(a.ravel(),bins=100)
ax[1].imshow(b,cmap='gray')
ax[1].set_title('Neutron image')
ax[1].set_xticks([]);ax[1].set_yticks([])
ax[4].hist(b.ravel(),bins=100)
ax[2].imshow(c0)
ax[2].set_title('Bivariate data')
ax[2].axis('off')
ax[5].imshow(c1)
ax[5].axis('off');
```



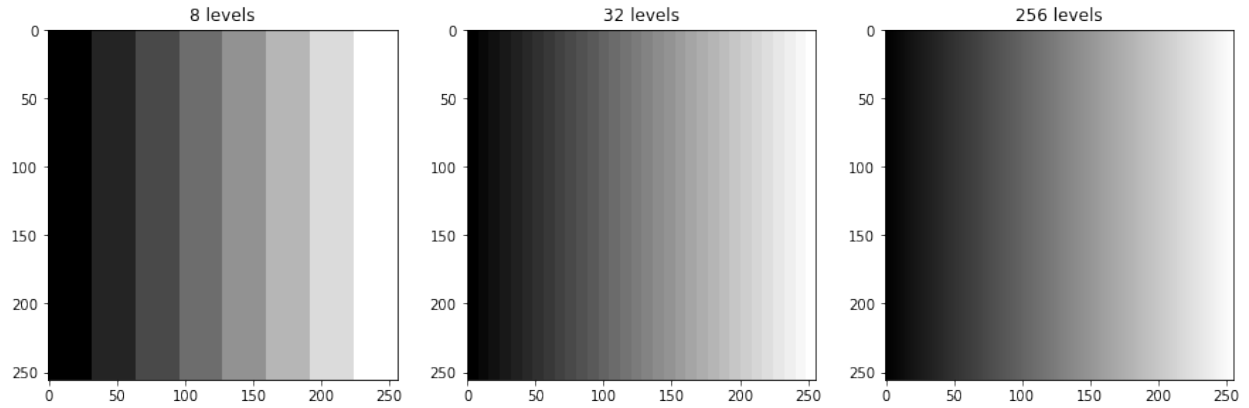
6.4 Visualizing gray levels

The human eye is not able to resolve many intensity levels

In this example we demonstrate how easy or hard it can be to perceive contrast differences in an image with different number of gray levels. How well you can see the differences depends on one hand on how well your eye can resolve the contrast difference, but there are also technical issues related to how well you can see the changes. E.g. how well your screen can display the changes and even the ambient light in the room you are working in.

```
xlin = np.linspace(0,255, 256)
xx, yy = np.meshgrid(xlin, xlin)

fig, ax = plt.subplots(1,3, figsize = (15, 5))
n=32; ax[0].imshow(np.floor(xx/n), interpolation='None', cmap = 'gray'); ax[0].set_
    title('{0} levels'.format(256//n))
n=8; ax[1].imshow(np.floor(xx/n), interpolation='None', cmap = 'gray'); ax[1].set_
    title('{0} levels'.format(256//n))
n=1; ax[2].imshow(np.floor(xx/n), interpolation='None', cmap = 'gray'); ax[2].set_
    title('{0} levels'.format(256//n));
```



6.4.1 Brightness

With image brightness, you can focus on narrow gray level intervals to better visually resolve local details.

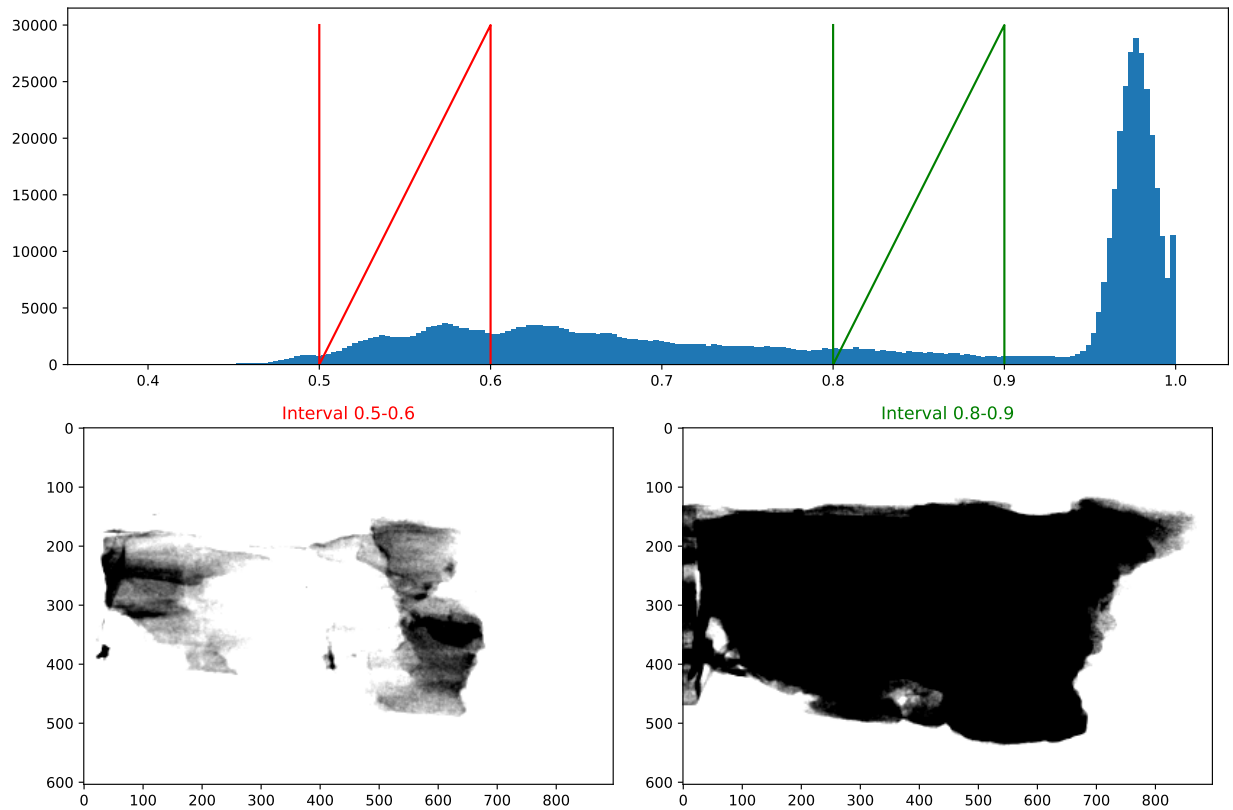


Fig. 6.4: Narrow intensity intervals to highlight low (left) and high (right) graylevel regions.

6.4.2 Contrast

Contrast controls the width of the intensity interval to use.

Contrast control is often used to define which gray levels to include when you save image to file. The image is usually represented in floating point data format after some calculations and you have to limit the interval to resolve the relevant information with many gray levels and reject outliers when you convert to 8- or 16-bit integers.

The example shows a narrow interval that mostly is useful to highlight features with small difference in contrast. The example with wider interval would be set to reject the background while most of the sample is visible. This setting may be useful for presentations and publication where you want to boost the visibility, but is not recommended if you want to use the image in further calculations. In the latter case it is important to keep as many gray levels as possible, i.e. also include the noise fluctuations in the background.

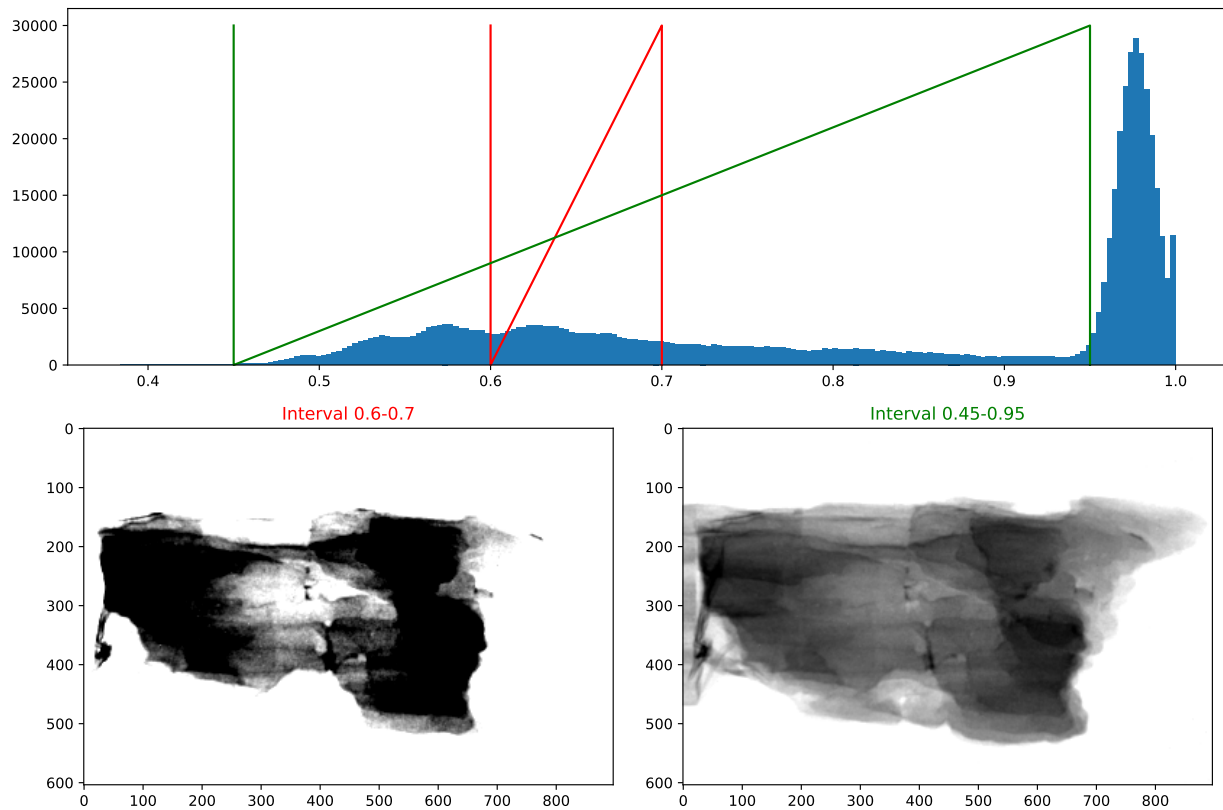


Fig. 6.5: Different image contrasts. Narrow interval to the left and wide interval to the right.

6.5 2D Intensity Images

```
basic_image = np.random.choice(range(100), size = (5,5))

xx, yy = np.meshgrid(range(basic_image.shape[1]), range(basic_image.shape[0]))
image_df = pd.DataFrame(dict(x = xx.ravel(),
                             y = yy.ravel(),
                             Intensity = basic_image.ravel()))
image_df[['x', 'y', 'Intensity']].head(5)
```



```

x  y  Intensity
0  0  0         44
1  1  0         76
2  2  0         96
3  3  0         71
4  4  0         13

```

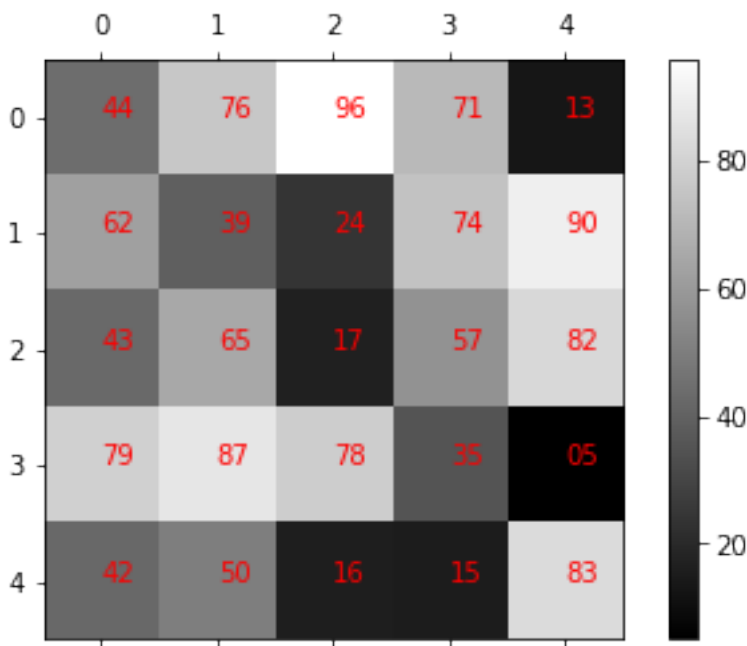
6.5.1 Viewing the data as an image

```

fig, ax1 = plt.subplots(1,1)
plot_image = ax1.matshow(basic_image, cmap = 'gray')
plt.colorbar(plot_image)

for _, c_row in image_df.iterrows():
    ax1.text(c_row['x'], c_row['y'], s = '%02d' % c_row['Intensity'], fontdict = dict(color = 'r'))

```



6.5.2 Let's colorize our image

The next step is to apply a color map (also called lookup table, LUT) to the image

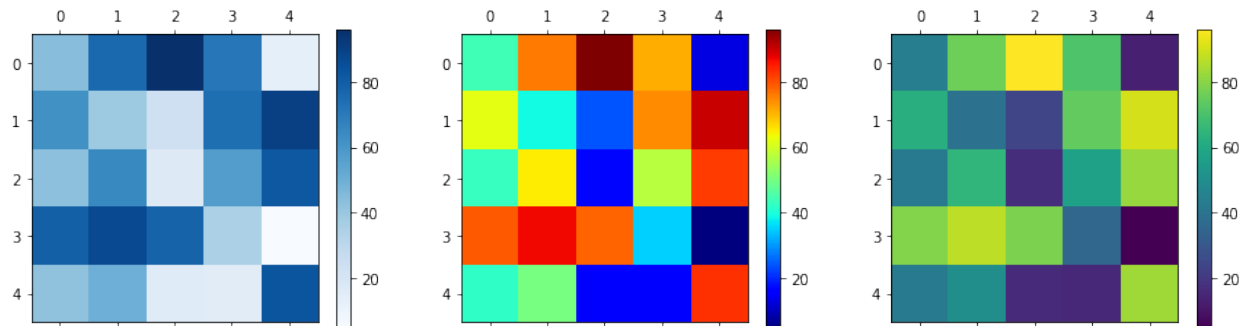
- so it is a bit more exciting
- some features are easier to detect Rogowitz et al. 1996

6.5.3 Using pseudo colormaps

The image intensity is mostly only represented by a scalar by a gray level. Which is makes it hard to see subtle changes in intensity. Colormaps can help here.

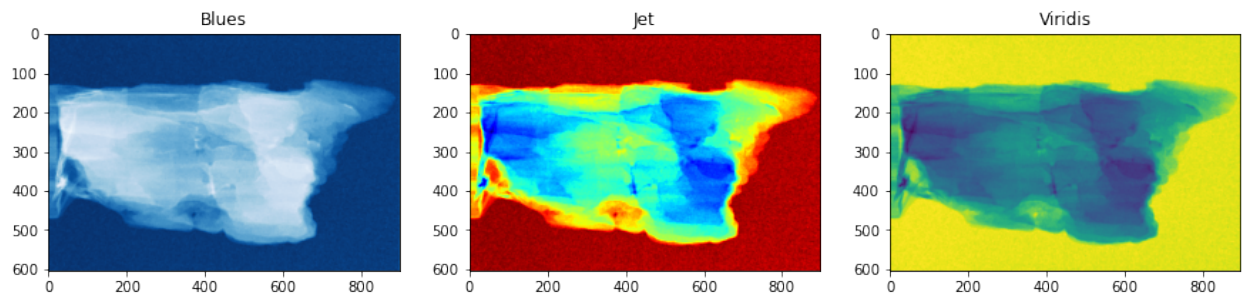
```
fig, ax = plt.subplots(1,3,figsize=(15,5))
ax=ax.ravel()

a0=ax[0].matshow(basic_image, cmap = 'Blues')
fig.colorbar(a0,ax=ax[0],shrink=0.75)
a1=ax[1].matshow(basic_image, cmap = 'jet')
fig.colorbar(a1,ax=ax[1],shrink=0.75)
a2=ax[2].matshow(basic_image, cmap = 'viridis')
fig.colorbar(a2,ax=ax[2],shrink=0.75);
```



6.5.4 Pseudo colors on real images

```
fig,ax=plt.subplots(1,3,figsize=(15,5))
ax[0].imshow(img,cmap='Blues'); ax[0].set_title('Blues')
ax[1].imshow(img,cmap='jet'); ax[1].set_title('Jet')
ax[2].imshow(img,cmap='viridis'); ax[2].set_title('Viridis');
```



6.5.5 Lookup Tables

Formally a color map is lookup table or a function which $f(\text{Intensity}) \rightarrow \text{Color}$

Matplotlib's color maps

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import cm
#from colorspacious import cspace_converter
from collections import OrderedDict

cmaps = OrderedDict()
cmaps['Perceptually Uniform Sequential'] = [
    'viridis', 'plasma', 'inferno', 'magma', 'cividis']

cmaps['Sequential'] = [
    'Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds',
    'YlOrBr', 'YlOrRd', 'OrRd', 'PuRd', 'RdPu', 'BuPu',
    'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn']

cmaps['Sequential (2)'] = [
    'binary', 'gist_yarg', 'gist_gray', 'gray', 'bone', 'pink',
    'spring', 'summer', 'autumn', 'winter', 'cool', 'Wistia',
    'hot', 'afmhot', 'gist_heat', 'copper']

cmaps['Diverging'] = [
    'PiYG', 'PRGn', 'BrBG', 'PuOr', 'RdGy', 'RdBu',
    'RdYlBu', 'RdYlGn', 'Spectral', 'coolwarm', 'bwr', 'seismic']

cmaps['Cyclic'] = ['twilight', 'twilight_shifted', 'hsv']

cmaps['Qualitative'] = ['Pastel1', 'Pastel2', 'Paired', 'Accent',
    'Dark2', 'Set1', 'Set2', 'Set3',
    'tab10', 'tab20', 'tab20b', 'tab20c']

cmaps['Miscellaneous'] = [
    'flag', 'prism', 'ocean', 'gist_earth', 'terrain', 'gist_stern',
    'gnuplot', 'gnuplot2', 'CMRmap', 'cubehelix', 'brg',
    'gist_rainbow', 'rainbow', 'jet', 'nipy_spectral', 'gist_ncar']

nrows = max(len(cmap_list) for cmap_category, cmap_list in cmaps.items())

gradient = np.linspace(0, 1, 256)
gradient = np.vstack((gradient, gradient))

def plot_color_gradients(cmap_category, cmap_list, nrows):
    fig, axes = plt.subplots(nrows=nrows)
    fig.subplots_adjust(top=0.95, bottom=0.01, left=0.2, right=0.99)
    axes[0].set_title(cmap_category + ' colormaps', fontsize=14)

    for ax, name in zip(axes, cmap_list):
        ax.imshow(gradient, aspect='auto', cmap=plt.get_cmap(name))
```

(continues on next page)

(continued from previous page)

```
pos = list(ax.get_position().bounds)
x_text = pos[0] - 0.01
y_text = pos[1] + pos[3]/2.
fig.text(x_text, y_text, name, va='center', ha='right', fontsize=10)

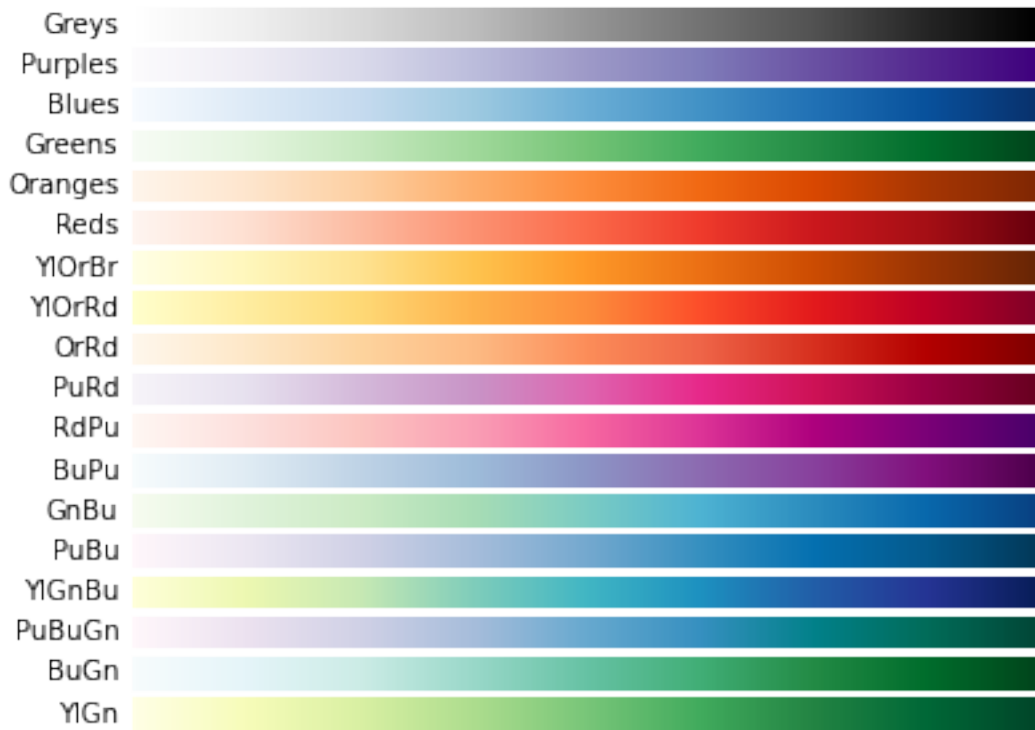
# Turn off all ticks & spines, not just the ones with colormaps.
for ax in axes:
    ax.set_axis_off()

for cmap_category, cmap_list in cmaps.items():
    plot_color_gradients(cmap_category, cmap_list, nrows)
```

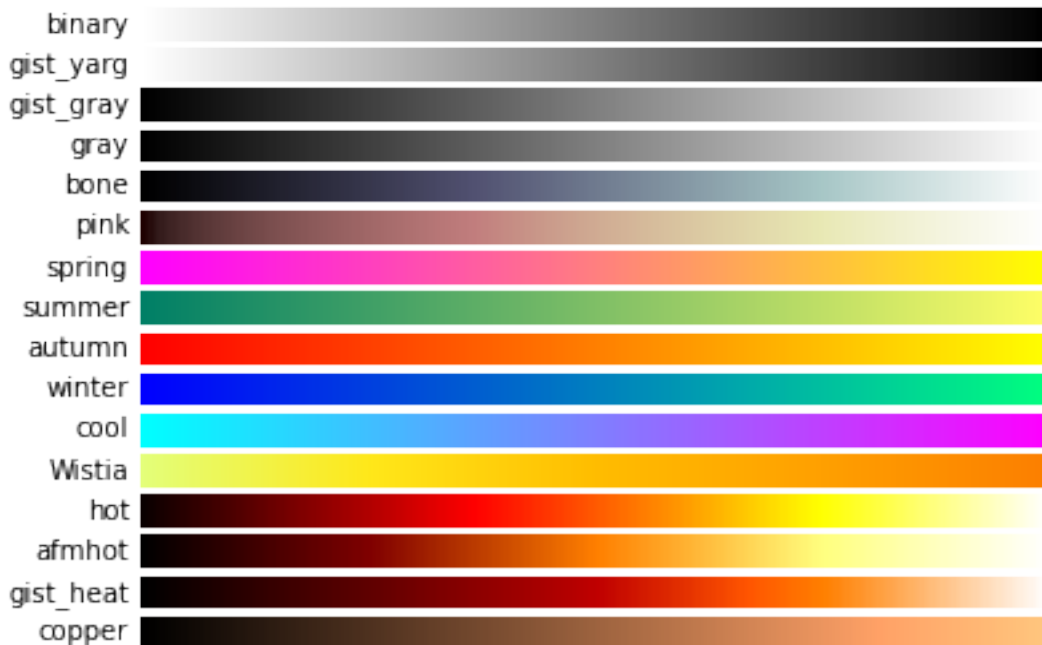
Perceptually Uniform Sequential colormaps



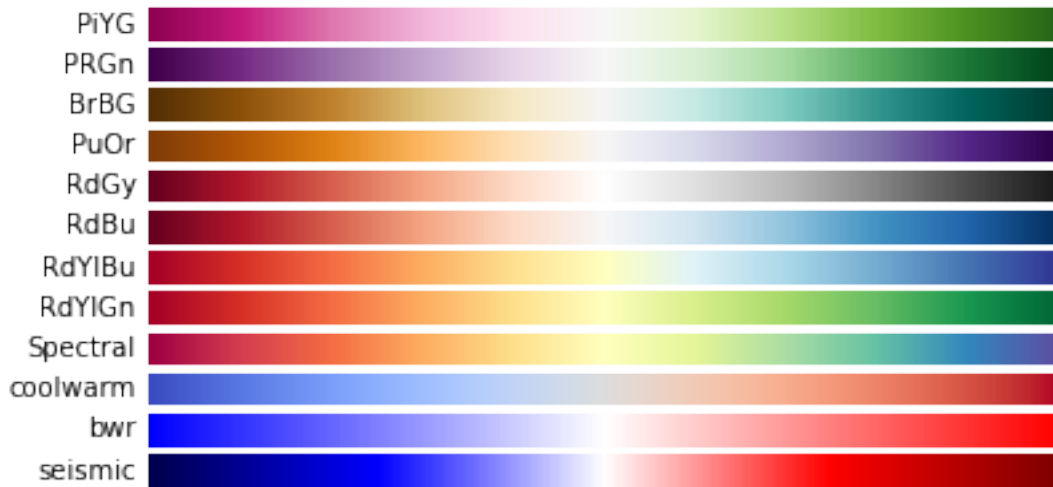
Sequential colormaps



Sequential (2) colormaps



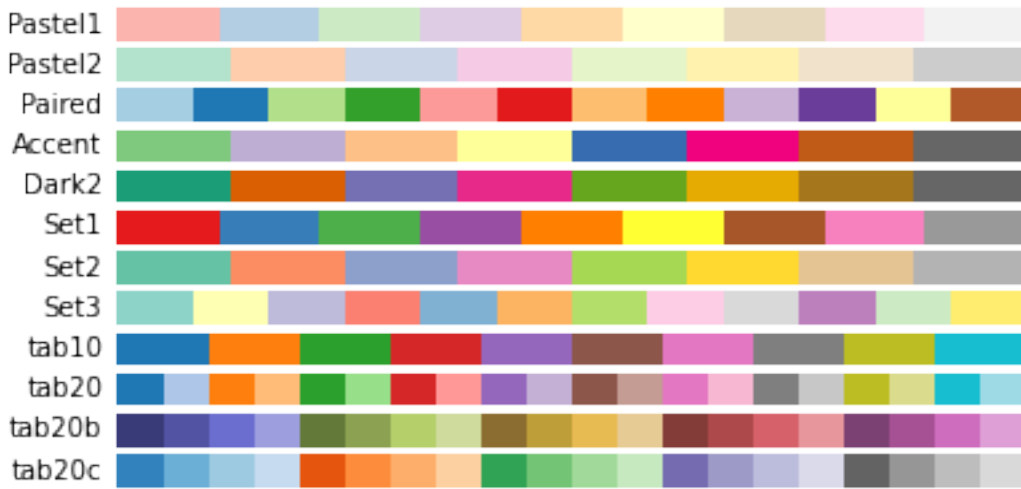
Diverging colormaps



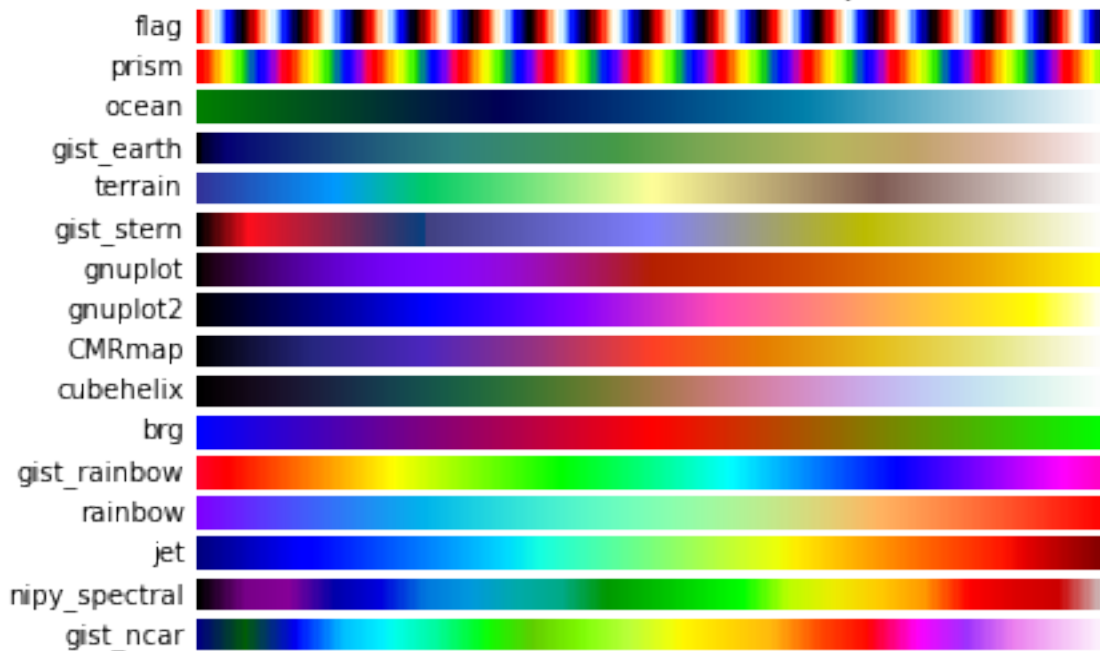
Cyclic colormaps



Qualitative colormaps



Miscellaneous colormaps



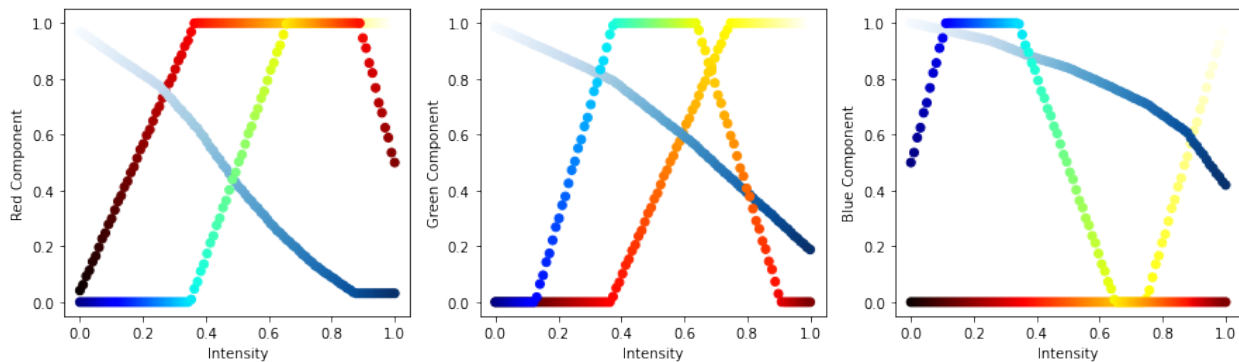
6.5.6 How are the colors combined

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
xlin = np.linspace(0, 1, 100)
colors = ['Red', 'Green', 'Blue']
plt.figure(figsize=[15,4])
for i in np.arange(0,3) :
    plt.subplot(1,3,i+1)

    plt.scatter(xlin,
                plt.cm.hot(xlin)[: ,i],
                c = plt.cm.hot(xlin),label="hot")
    plt.scatter(xlin,
                plt.cm.Blues(xlin)[: ,i],
                c = plt.cm.Blues(xlin),label="blues")

    plt.scatter(xlin,
                plt.cm.jet(xlin)[: ,i],
                c = plt.cm.jet(xlin),label='jet')

plt.xlabel('Intensity');
plt.ylabel('{0} Component'.format(colors[i]));
```



6.5.7 Applied LUTs

These transformations can also be non-linear as is the case of the graph below where the mapping between the intensity and the color is a log relationship meaning the the difference between the lower values is much clearer than the higher ones

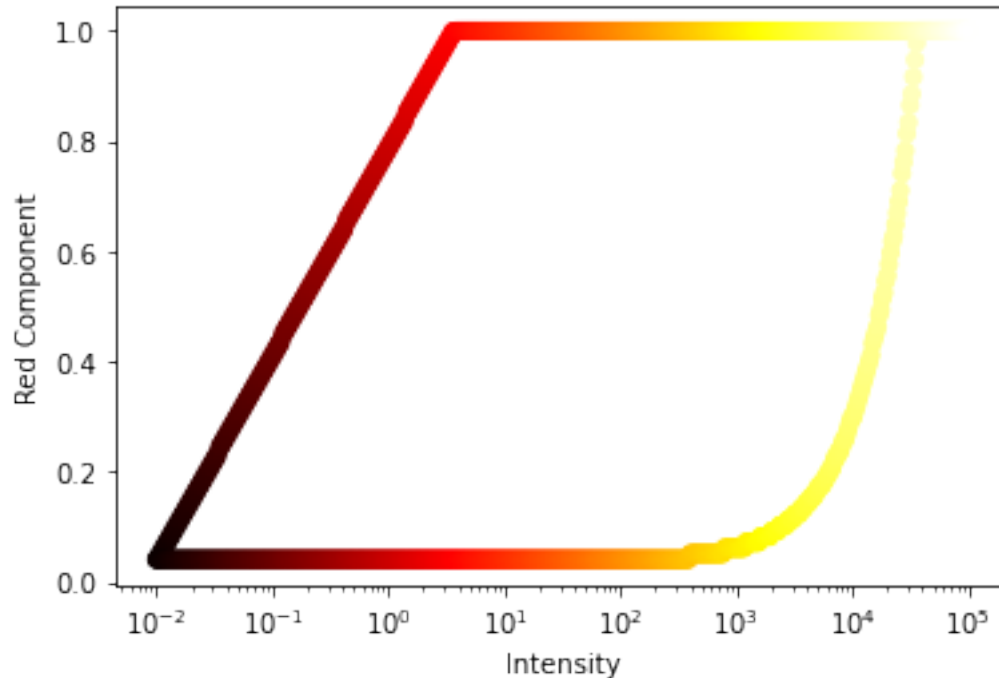
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
xlin = np.logspace(-2, 5, 500)
log_xlin = np.log10(xlin)
norm_xlin = (log_xlin-log_xlin.min())/(log_xlin.max()-log_xlin.min())
fig, ax1 = plt.subplots(1,1)

ax1.scatter(xlin, plt.cm.hot(norm_xlin)[: ,0],
            c = plt.cm.hot(norm_xlin))
```

(continues on next page)

(continued from previous page)

```
ax1.scatter(xlin, plt.cm.hot(xlin/xlin.max())[:,0],
            c = plt.cm.hot(norm_xlin))
ax1.set_xscale('log'); ax1.set_xlabel('Intensity'); ax1.set_ylabel('Red Component');
```



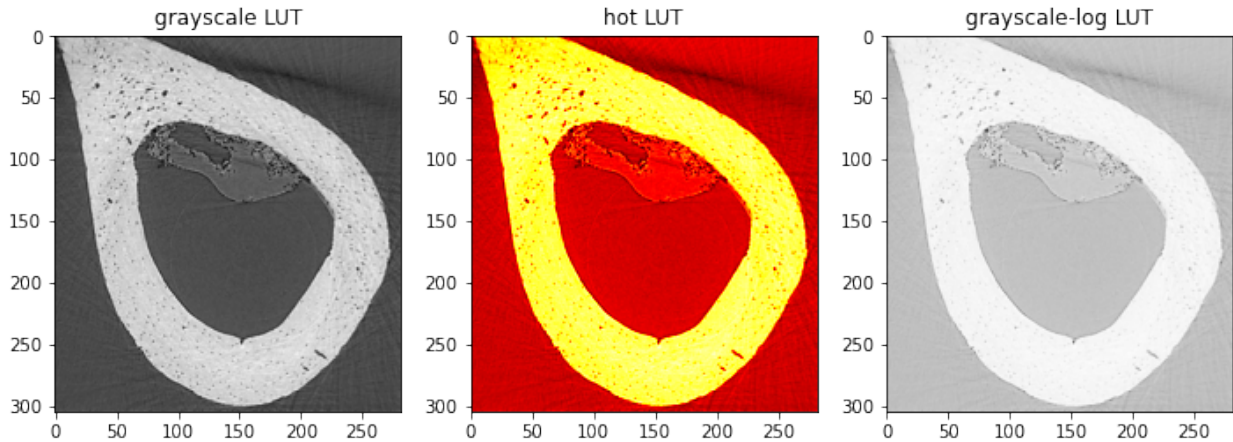
6.5.8 LUTs on real images

On a real image the difference is even clearer

```
%matplotlib inline
import matplotlib.pyplot as plt
from skimage.io import imread
fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize = (12, 4))
in_img = imread('figures/bone-section.png')[:, :, 0].astype(np.float32)
ax1.imshow(in_img, cmap = 'gray');
ax1.set_title('grayscale LUT');

ax2.imshow(in_img, cmap = 'hot');
ax2.set_title('hot LUT');

ax3.imshow(np.log2(in_img+1), cmap = 'gray');
ax3.set_title('grayscale-log LUT');
```



Colormaps are only used for the visualization making it possible to better visualize and highlight features in the image. They can however also be misleading if you chose the wrong colormap. The interpretation of the image is in particular hard when you start manipulating the colormap. In this way it is even possible to “invent” features in the image that are not real. A typical example is that you thanks to the colormap could see a denser skin like structure near the sample boundary. In the reality this “skin” is only the smooth edge which is caused by low resolution of the imaging system.

6.6 Pixelwise operations

Pixelwise operations apply scalar operations to each pixel.

- Arithmetics +, -, *, /
- Functions e.g. $\sin(x)$, $\exp(x)$, $\ln(x)$

Statistic functions

- mean, standard deviation
- min, max

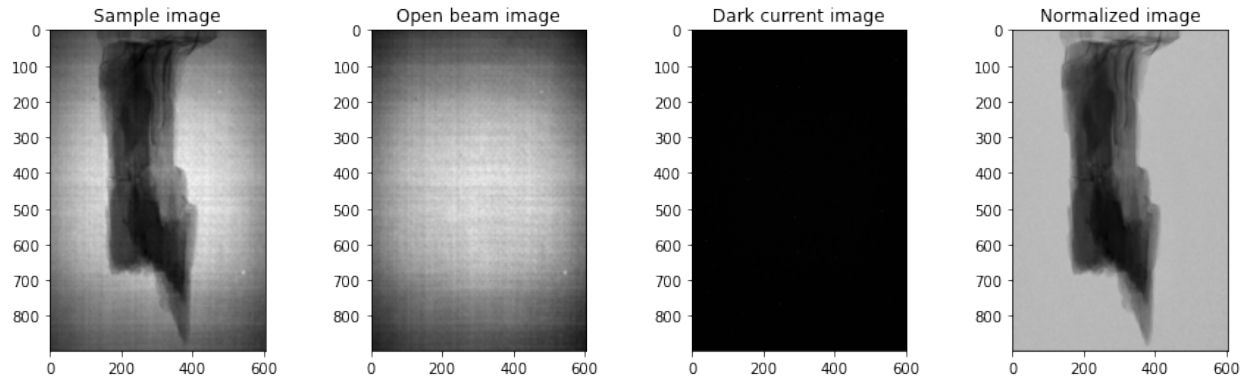
6.6.1 Demonstrating arithmetic functions (flat field normalization)

$$normed = \frac{img - dc}{ob - dc}$$

```
img = plt.imread('data/wood_0000.tif')
ob = plt.imread('data/ob_0000.tif')
dc = plt.imread('data/dc_0000.tif')

normed = (img-dc)/(ob-dc)

# Visualization
fig,ax = plt.subplots(1,4,figsize=(15,4))
ax[0].imshow(img, cmap='gray'); ax[0].set_title('Sample image')
ax[1].imshow(ob, cmap='gray'); ax[1].set_title('Open beam image')
ax[2].imshow(dc, cmap='gray'); ax[2].set_title('Dark current image')
ax[3].imshow(normed, cmap='gray'); ax[3].set_title('Normalized image');
```



6.7 3D Images

For a 3D image, the position or spatial component has a 3rd dimension (z if it is a spatial, or t if it is a movie)

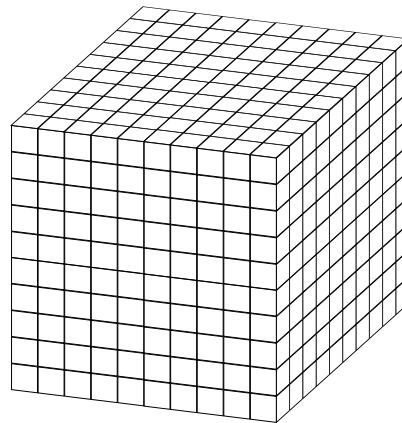


Fig. 6.6: Three-dimensional data can be a volume in space.

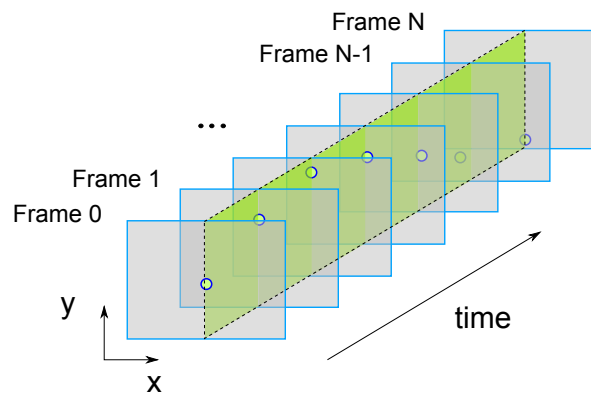


Fig. 6.7: A movie can also be seen as a three-dimensional image.

6.7.1 A 3D image as array

```
import numpy as np
vol_image = np.arange(27).reshape((3,3,3))
print(vol_image)
```

```
[[[ 0  1  2]
   [ 3  4  5]
   [ 6  7  8]]

 [[ 9 10 11]
  [12 13 14]
  [15 16 17]]

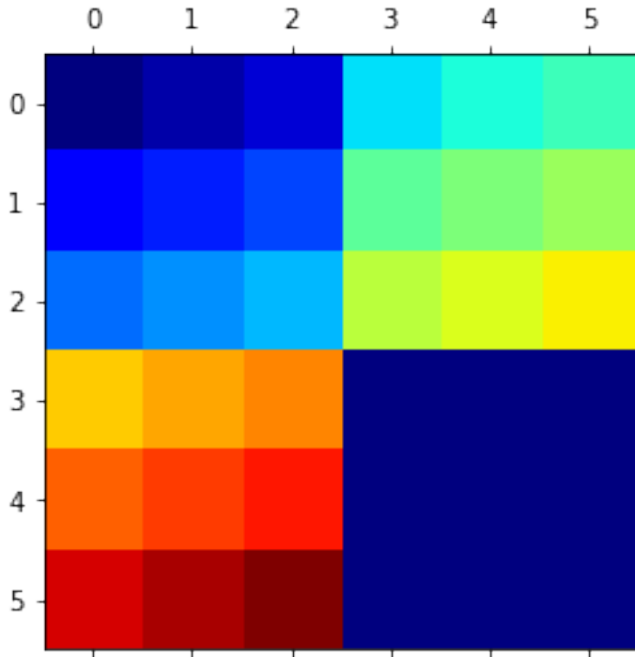
 [[18 19 20]
  [21 22 23]
  [24 25 26]]]
```

6.7.2 Showing 2D slices from volume

This can then be rearranged from a table form into an array form and displayed as a series of slices

```
%matplotlib inline
import matplotlib.pyplot as plt
from skimage.util import montage as montage2d
print(montage2d(vol_image, fill = 0))
plt.matshow(montage2d(vol_image, fill = 0), cmap = 'jet');
```

```
[[ 0  1  2  9 10 11]
 [ 3  4  5 12 13 14]
 [ 6  7  8 15 16 17]
 [18 19 20  0  0  0]
 [21 22 23  0  0  0]
 [24 25 26  0  0  0]]
```



6.8 4D Images

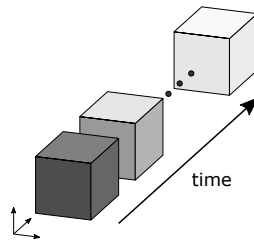


Fig. 6.8: Four-dimensional data are series of volumes.

6.9 Multiple Values per pixel

In the images thus far, we have had one value per position, but there is no reason there cannot be multiple values. In fact this is what color images are (red, green, and blue) values and even 4 channels with transparency (alpha) as a different. For clarity we call the **dimensionality** of the image the number of dimensions in the spatial position, and the **depth** the number in the value.

```
base_df = pd.DataFrame([dict(x = x, y = y) for x,y in product(range(5), range(5))])
base_df['Intensity'] = np.random.uniform(0, 1, 25)
base_df['Transparency'] = np.random.uniform(0, 1, 25)
base_df.head(5)
```

x	y	Intensity	Transparency
0	0	0.495103	0.152570

(continues on next page)

(continued from previous page)

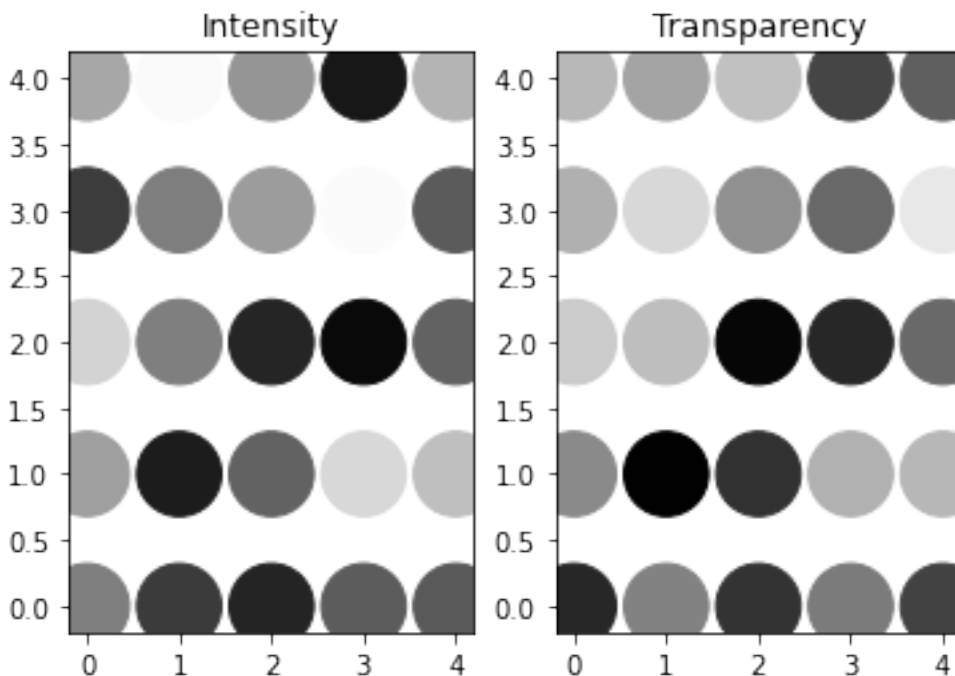
1	0	1	0.622100	0.542207
2	0	2	0.825857	0.794896
3	0	3	0.236108	0.688556
4	0	4	0.655315	0.720574

This can then be rearranged from a table form into an array form and displayed as a series of slices

6.9.1 Display multi-valued pixels separately

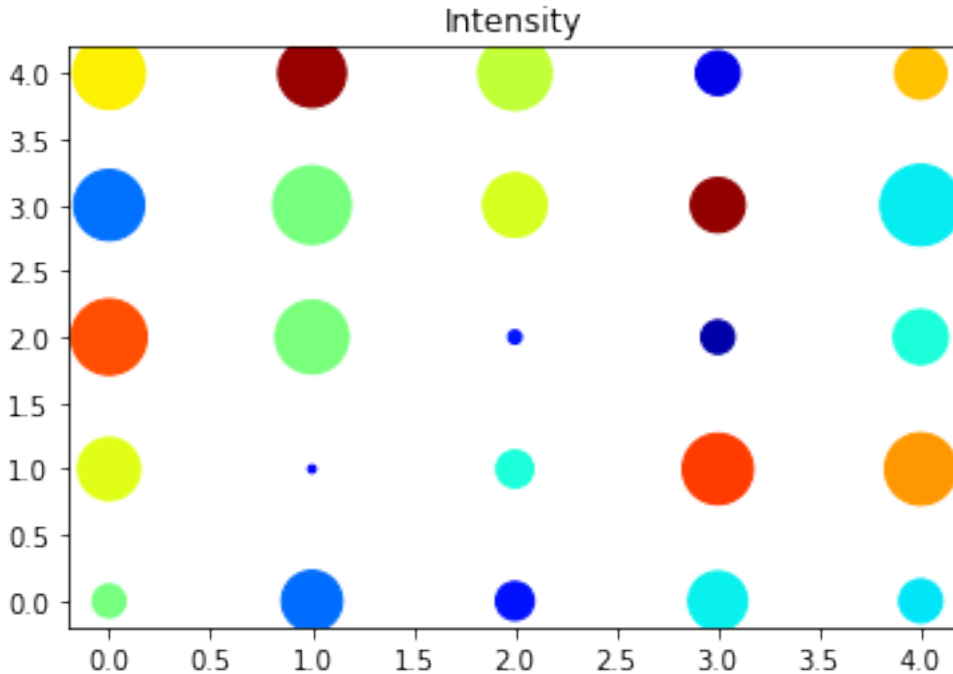
The most straight forward way to display multiple pixel values is to display each value separately. This method is, however, mostly not very suitable as the values often are related in some sense. Therefore it is recommended to combine the values in the same plot.

```
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.scatter(base_df['x'], base_df['y'], c = plt.cm.gray(base_df['Intensity']), s =
    ↪1000)
ax1.set_title('Intensity')
ax2.scatter(base_df['x'], base_df['y'], c = plt.cm.gray(base_df['Transparency']), s =
    ↪1000)
ax2.set_title('Transparency');
```



In this example we combined two values use one value to control the colormap and the other to control the size of the dots. How you combine the data is related to the to of data you want to combine. If the values are components of a vector it makes more sense to show arrows of different length and direction, etc.

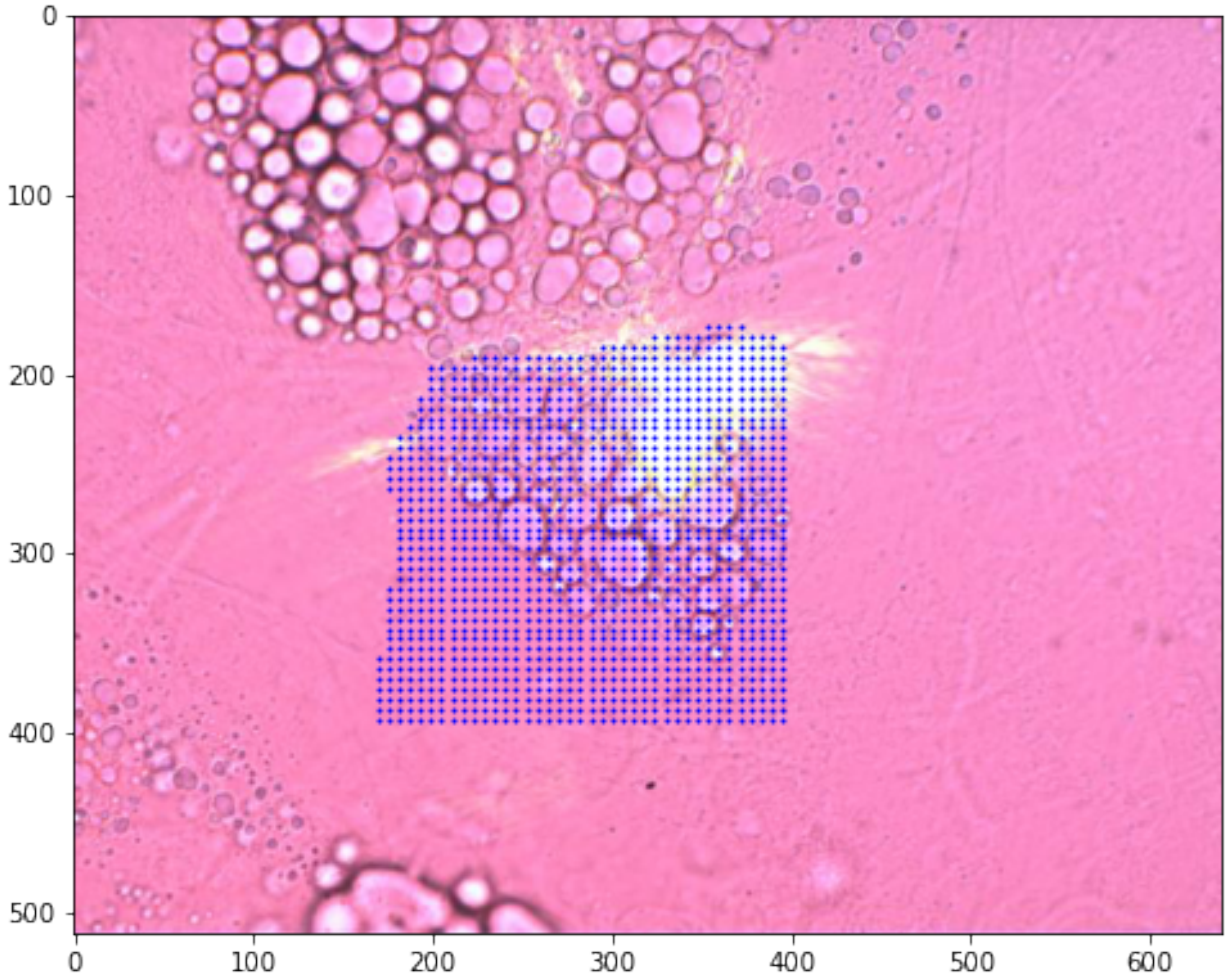
```
fig, (ax1) = plt.subplots(1, 1)
ax1.scatter(base_df['x'], base_df['y'], c = plt.cm.jet(base_df['Intensity']), s =
    ↪1000*base_df['Transparency'])
ax1.set_title('Intensity');
```



6.10 Hyperspectral Imaging

At each point in the image (black dot), instead of having just a single value, there is an entire spectrum. A selected group of these (red dots) are shown to illustrate the variations inside the sample. While certainly much more complicated, this still constitutes an image and requires the same sort of techniques to process correctly.

```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
from skimage.io import imread
import os
raw_img = imread('../common/data/raw.jpg')
im_pos = pd.read_csv('../common/data/impos.csv', header = None)
im_pos.columns = ['x', 'y']
fig, ax1 = plt.subplots(1,1, figsize = (8, 8));
ax1.imshow(raw_img);
ax1.scatter(im_pos['x'], im_pos['y'], s = 1, c = 'blue');
```



6.10.1 Looking at the pixel values

Each pixel of this data set is represented by a full spectrum. This means that we have both wave numbers and intensity values. In many cases, the wavenumbers are the same for all pixels, which makes it possible to reduce the redundancy of the wave number vector using a single 1D array to represent this information.

```
full_df = pd.read_csv('../common/data/full_img.csv').query('wavenum<1200')
print(full_df.shape[0], 'rows')
full_df.head(5)
```

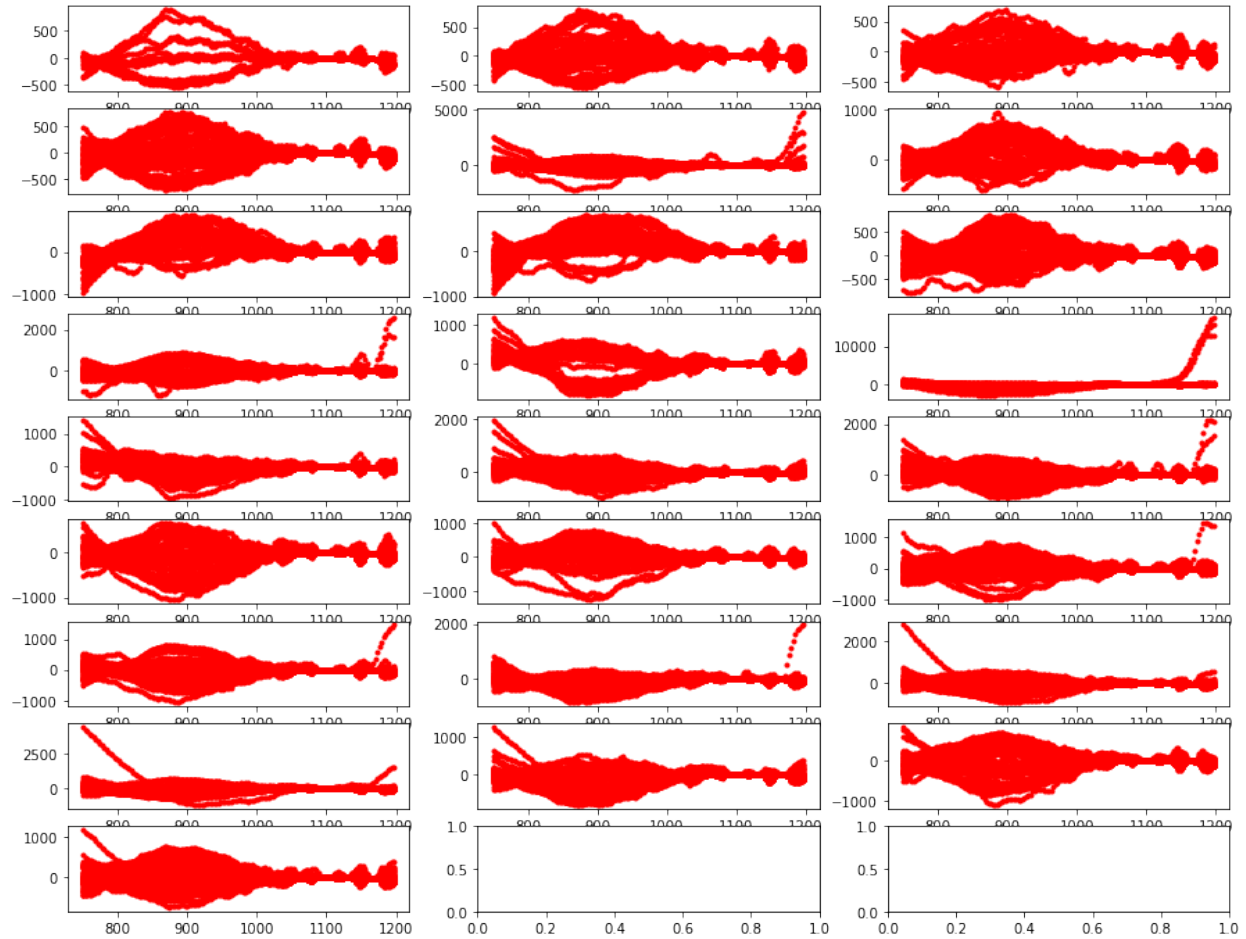
210750 rows

	x	y	wavenum	val
0	168.95	358.8	750	527.571102
1	168.95	358.8	753	459.778584
2	168.95	358.8	756	406.337255
3	168.95	358.8	759	341.858123
4	168.95	358.8	762	246.645673


```

full_df['g_x'] = pd.cut(full_df['x'], 5)
full_df['g_y'] = pd.cut(full_df['y'], 5)
fig, m_axs = plt.subplots(9, 3, figsize = (15, 12)); m_axs=m_axs.ravel()
for ((g_x, g_y), c_rows), c_ax in zip(full_df.sort_values(['x', 'y']).groupby(['g_x',
↪ 'g_y']), m_axs):
    c_ax.plot(c_rows['wavenum'], c_rows['val'], 'r.')

```



WORKFLOWS FOR IMAGE ANALYSIS

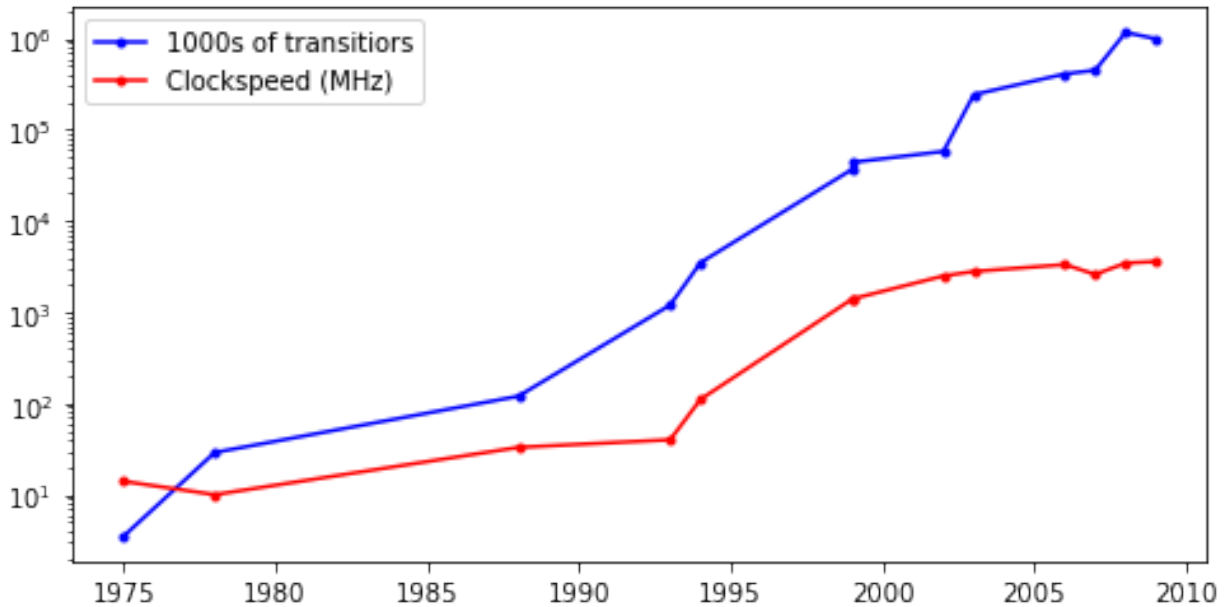
7.1 Computing has changed: Parallel

7.1.1 Moores Law

$$\text{Transistors} \propto 2^{T/(18 \text{ months})}$$

```
# Borrowed from https://gist.github.com/humberto-ortiz/de4b3a621602b78bf90d
moores_txt=["Id Name Year Count(1000s) Clock(MHz)\n",
"0 MOS65XX 1975 3.51 14\n",
"1 Intel8086 1978 29.00 10\n",
"2 MIPSr3000 1988 120.00 33\n",
"3 AMDAm486 1993 1200.00 40\n",
"4 NexGenNx586 1994 3500.00 111\n",
"5 AMDAthlon 1999 37000.00 1400\n",
"6 IntelPentiumIII 1999 44000.00 1400\n",
"7 PowerPC970 2002 58000.00 2500\n",
"8 AMDAthlon64 2003 243000.00 2800\n",
"9 IntelCore2Duo 2006 410000.00 3330\n",
"10 AMDPhenom 2007 450000.00 2600\n",
"11 IntelCorei7 2008 1170000.00 3460\n",
"12 IntelCorei5 2009 995000.00 3600"]

sio_table = StringIO(''.join(moores_txt)); moore_df = pd.read_table(sio_table, sep =
↪'\s+', index_col = 0);
fig, ax1 = plt.subplots(1,1, figsize = (8, 4)); ax1.semilogy(moore_df['Year'], moore_
↪df['Count(1000s)'], 'b.-', label = '1000s of transistors'); ax1.semilogy(moore_df[
↪'Year'], moore_df['Clock(MHz)'], 'r.-', label = 'Clockspeed (MHz)');ax1.legend(loc_
↪= 2);
```



Based on data from <https://gist.github.com/humberto-ortiz/de4b3a621602b78bf90d>

7.1.2 Why doesn't the clock rate follow Moore's law?

There are now many more transistors inside a single computer but the CPU clock rate hasn't increased. It is actually stagnating since the early 2000s. The reason for this is that the number of processing cores has started to increase. Each core is a CPU is an autonomous CPU which makes it possible to perform many individual tasks in parallel.

- Multiple Core
- Multiple CPUs
- New modalities
 - GPUs provide many cores which operate at slow speed

The conclusion is that we need to make the processing on multiple Cores/CPUs possible. This can be done by choosing multithreaded modules for the processing. In the extreme case we need to implement new multi-process code for the analysis. How this is done depends in the target hardware. You will use different approaches for a CPU than you use in a cluster or in cloud computing. In general, we can conclude:

Parallel Code is important

7.2 Computing has changed: Cloud

Cloud computing is a type of service that has evolved from the big data era. Large companies have built up their analysis resources to match the need at peak demand. These resources were often underused which resulted in a new business model. Providing computing resources to external customers. In this way the customers only pay for the computing time they need and let someone else handle investments and system maintenance.

- Computer, servers, workstations are *wildly underused* (majority are <50%)
- Buying a big computer that sits *idle most of the time* is a waste of money
 - <http://www-inst.eecs.berkeley.edu/~cs61c/sp14/>



Fig. 7.1: Different cloud services where it is possible to buy CPU time.

– “The Case for Energy-Proportional Computing,” Luiz André Barroso, Urs Hölzle, IEEE Computer, December 2007

- Traditionally the most important performance criteria was time, how fast can it be done
- With Platform as a service servers can be *rented instead of bought*
- Speed is still important but using cloud computing \$ / Sample is the real metric
- In Switzerland a PhD student is 400x as expensive per hour as an Amazon EC2 Machine
- Many competitors keep prices low and offer flexibility

7.3 Workflow analysis

It sound like a great idea to perform tasks in parallel, but how do we bring the compute to do so. The following example from real life is used to demonstrate how a greater task can be broken down into smaller piecese.

7.3.1 Simple Soup

We want to cook a simple soup. There is already a recipe telling us what to do to produce a soup. It is easy to follow the list, anyone with the right steps can execute and repeat (if not reproduce) the soup

1. Buy {carrots, peas, tomatoes} at market
2. *then* Buy meat at butcher
3. *then* Chop carrots into pieces
4. *then* Chop potatos into pieces
5. *then* Heat water
6. *then* Wait until boiling then add chopped vegetables
7. *then* Wait 5 minutes and add meat

All steps are done in a sequence to produce this soup. This would correspond to performing the task on a single CPU. Now, how can we change this?

7.3.2 More complicated soup

The following recipe is harder to follow and you need to carefully keep track of what is being performed. If you look at it in detail, you will see that some of the steps can be performed independently of the others. This is our opportunity to dispatch parallel tasks that would be running on different CPUs. In the kitchen, you could delegate tasks to other people and thus shorten the time until the soup is ready.

Steps 1-4

1. *then* Mix carrots with potatoes $\rightarrow mix_1$
2. *then* add egg to mix_1 and fry for 20 minutes
3. *then* Tenderize meat for 20 minutes
4. *then* add tomatoes to meat and cook for 10 minutes $\rightarrow mix_2$
5. *then* Wait until boiling then add mix_1
6. *then* Wait 5 minutes and add mix_2

7.4 Using flow charts / workflows

The processing task can be visualize using flow charts to better get an overview of the steps to be done. A sequence has a trivial shape, but it hard to get the big picture as soon as you start creating more complicated schemes .

7.4.1 Workflow of the Simple Soup

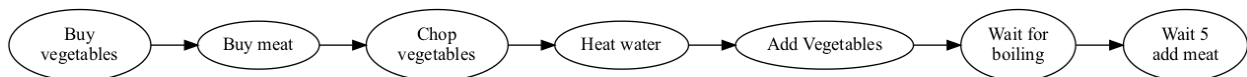
In this example we use the graph plotting module graphviz to draw the flow chart of the soup example. Graphviz is only a plotting tool with no direct processing capacity.

```

from IPython.display import Image, display
import pydot
graph = pydot.Dot(graph_type='digraph', rankdir="LR")
node_names = ["Buy\nvegetables", "Buy meat", "Chop\nvegetables", "Heat water", "Add_
    ↪Vegetables",
    "Wait for\nboiling", "Wait 5\nadd meat"]
nodes = [pydot.Node(name = '%04d' % i, label = c_n)
    for i, c_n in enumerate(node_names)]
for c_n in nodes:
    graph.add_node(c_n)

for (c_n, d_n) in zip(nodes, nodes[1:]):
    graph.add_edge(pydot.Edge(c_n, d_n))

plt = Image(graph.create_png()); display(plt);
    
```



7.4.2 Workflows - the complicated soup

Clearly a linear set of instructions is ill-suited for even a fairly easy soup, it is then even more difficult when there are dozens of steps and different pathways.

Furthermore a clean workflow allows you to better parallelize the task since it is clear which tasks can be performed independently.

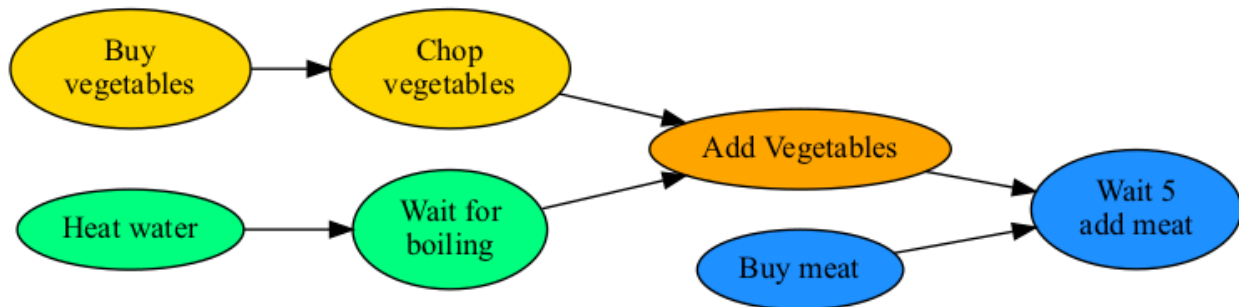
```

from IPython.display import Image, display
import pydot
graph = pydot.Dot(graph_type='digraph', rankdir="LR")
node_names = ["Buy\nvegetables", "Buy meat", "Chop\nvegetables", "Heat water", "Add_
↳Vegetables",
              "Wait for\nboiling", "Wait 5\nadd meat"]
nodes = [pydot.Node(name = '%04d' % i, label = c_n, style = 'filled')
          for i, c_n in enumerate(node_names)]
for c_n in nodes:
    graph.add_node(c_n)

def e(i,j, col = None):
    if col is not None:
        for c in [i,j]:
            if nodes[c].get_fillcolor() is None:
                nodes[c].set_fillcolor(col)
    graph.add_edge(pydot.Edge(nodes[i], nodes[j]))

e(0, 2, 'gold'); e(2, 4); e(3, -2, 'springgreen'); e(-2, 4, 'orange'); e(4, -1); e(1,
↳ -1, 'dodgerblue')
plt = Image(graph.create_png()); display(plt);

```



7.5 Directed Acyclical Graphs (DAG)

- We can represent almost any computation without loops as DAG.
- This allows us to break down a computation into pieces which can be carried out independently.

There are a number of tools which let us handle this issue.

- PyData Dask - <https://dask.pydata.org/en/latest/>
- Apache Spark - <https://spark.apache.org/>
- Spotify Luigi - <https://github.com/spotify/luigi>
- Airflow - <https://airflow.apache.org/>
- KNIME - <https://www.knime.com/>

- Google Tensorflow - <https://www.tensorflow.org/>
- Pytorch / Torch - <http://pytorch.org/>

7.5.1 Concrete example - Creating a DAG

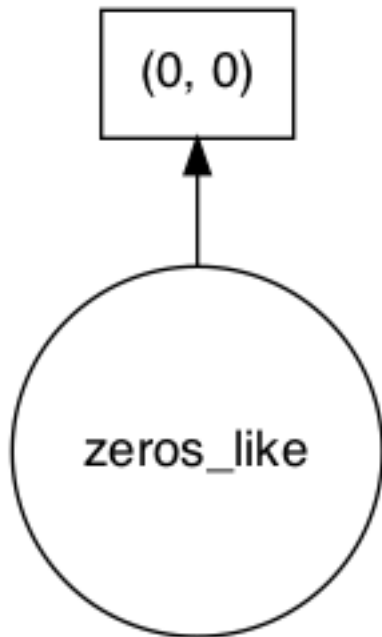
What is a DAG good for?

Create two variables

Let create a DAG to demonstrate how it works. This first piece creates two small images (5x5 pixels) each containing either '0's or '1's. We still haven't started any calculations using this DAG.

```
import dask.array as da

from dask.dot import dot_graph
image_1 = da.zeros((5,5), chunks = (5,5))
image_2 = da.ones((5,5), chunks = (5,5))
dot_graph(image_1.dask)
```

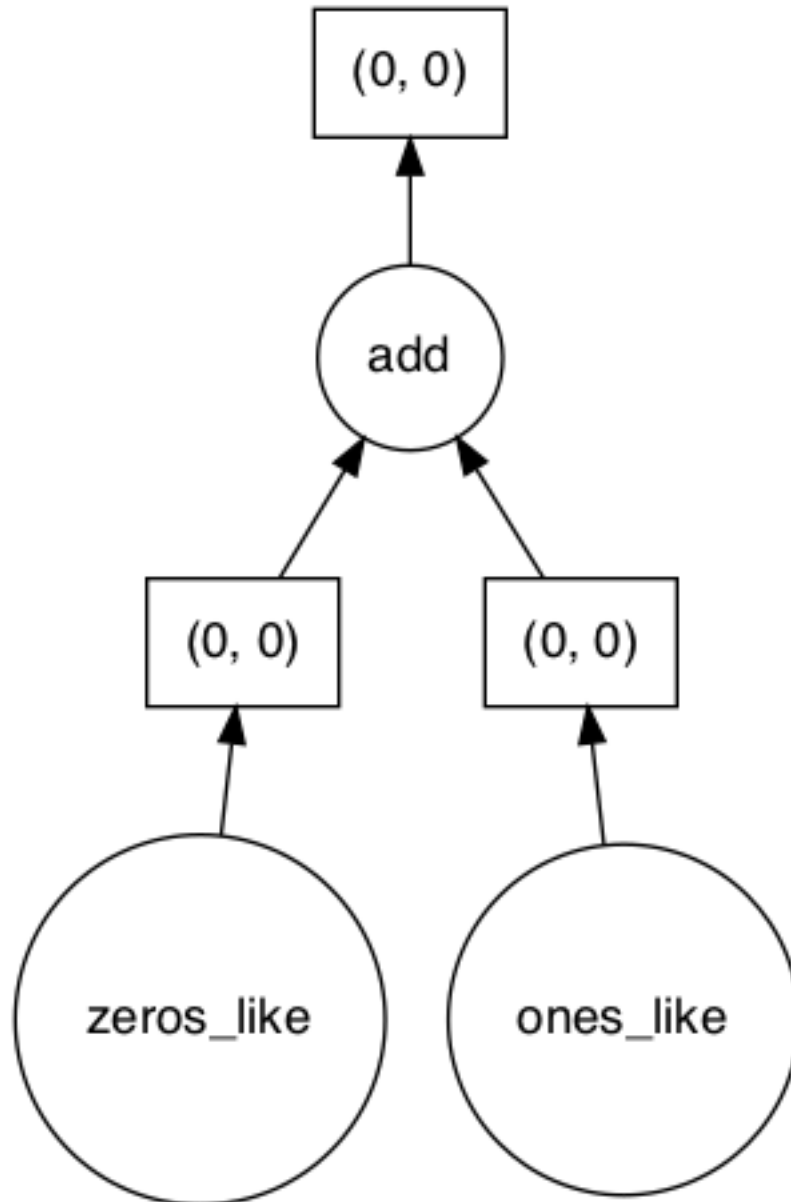


Add two variables

Adding two variables in a DAG looks like the code to add any variables. Now if we look at the graph created by this operation, we that there is a directed flow of the data in this graph.

$$Image_3 = Image_1 + Image_2$$

```
image_3 = image_1 + image_2
dot_graph(image_3.dask)
```

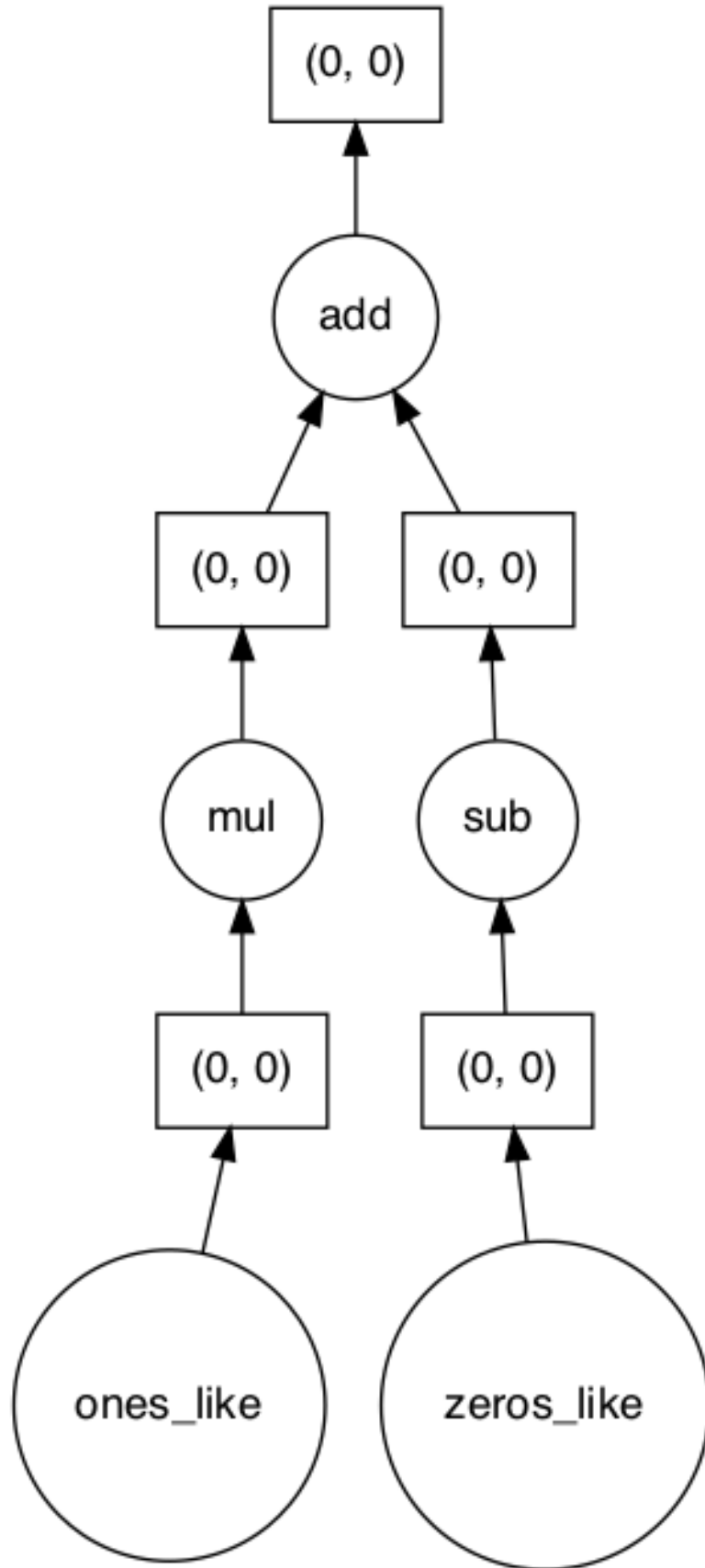



A more complicated operation

Now lets make a more complicated calculation like:

$$Image_4 = (image_1 - 10) + (image_2 * 50)$$

```
image_4 = (image_1-10) + (image_2*50)
dot_graph(image_4.dask)
```



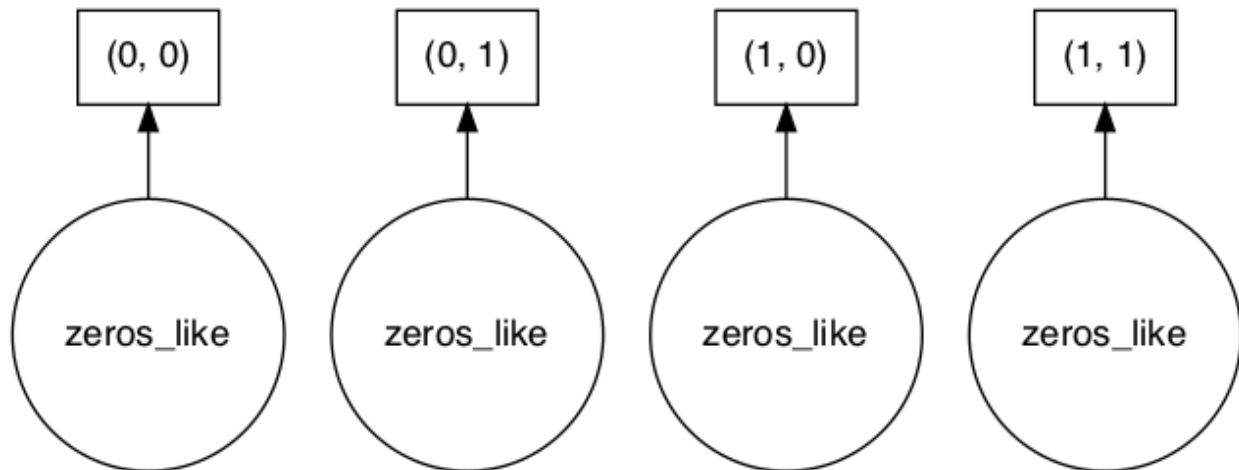
We can here see that there are two clear branches in the flow chart. This indicates that here is an opportunity to perform these tasks in parallel.

7.5.2 Let's go big

Now let's see where this can be really useful. Dask allows you to split the data into smaller pieces (chunks). In the example below we have created a 1024x1024 pixels image and also told dask that this image should be divided into chunks of 512x512 pixels.

Creating a large image with chunks

```
import dask.array as da
from dask.dot import dot_graph
image_1 = da.zeros((1024, 1024), chunks = (512, 512))
image_2 = da.ones((1024, 1024), chunks = (512, 512))
dot_graph(image_1.dask)
```



You see now that there are four graph created. The boxes on the top indicates the location of the chunk in the large image.

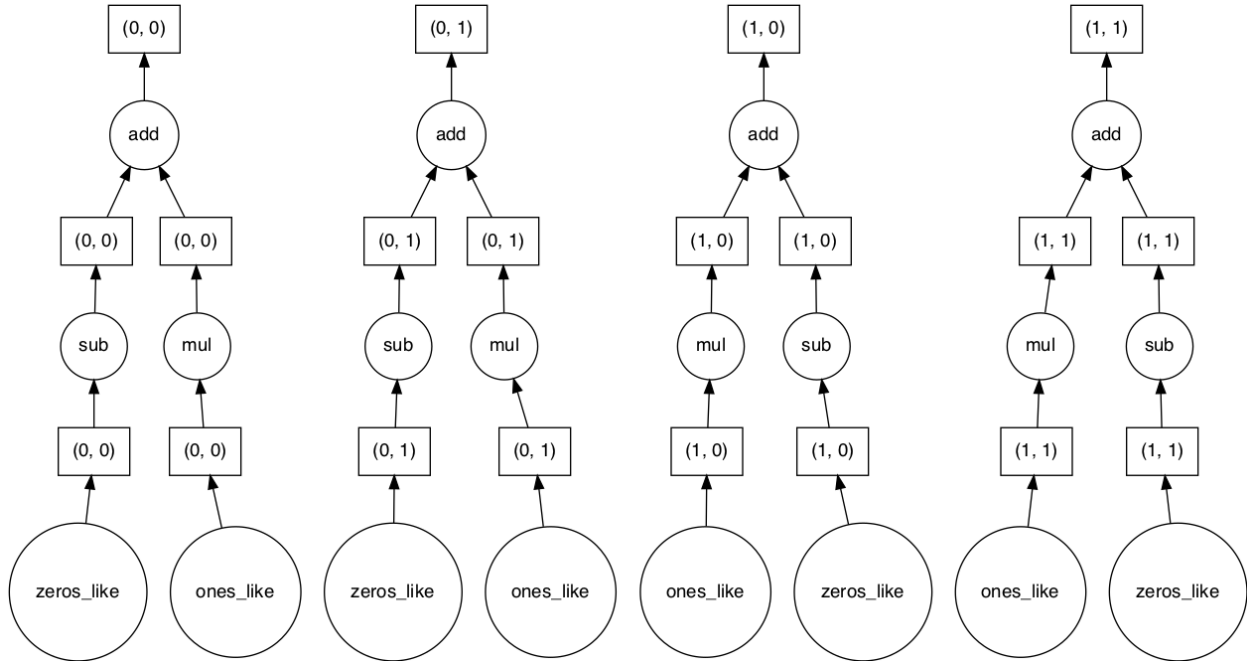
Computing something with a larger image

If we now apply the “complicated” calculation

$$Image_4 = (image_1 - 10) + (image_2 * 50)$$

on the large image with chunks, we see that the code hasn't changed at all. When we look at the graph generated with this data, we see that there are four graphs repeating the same operations. One graph for each chunk we defined in the data.

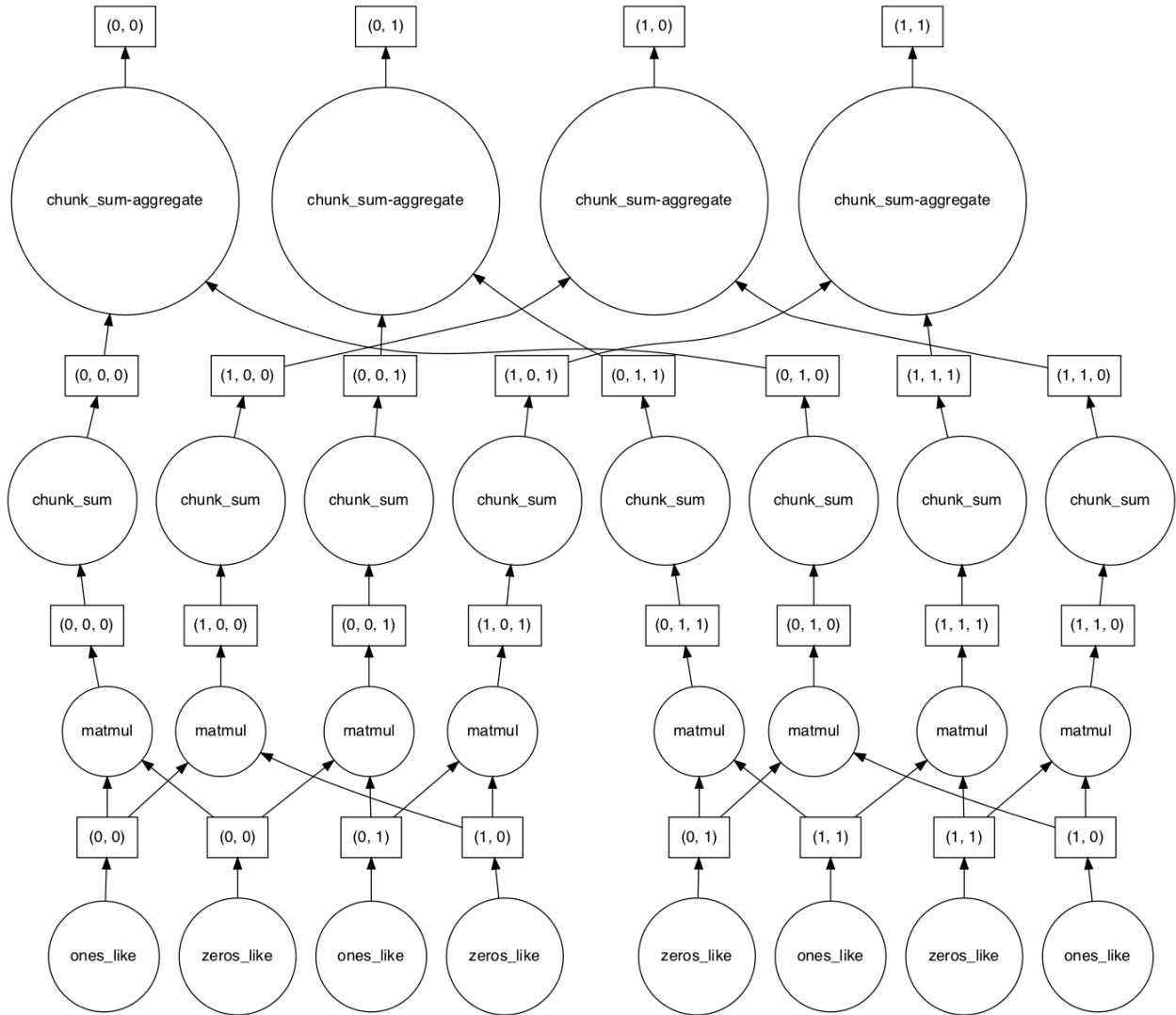
```
image_4 = (image_1-10) + (image_2*50)
dot_graph(image_4.dask)
```



7.5.3 Matrix multiplication in a DAG

A matrix multiplication is a more complicated operation than the previous examples. This operation needs information from all the other chunks to calculate the current chunk. This can also be seen in the wiring of the graph below.

```
image_5 = da.matmul(image_1, image_2)
dot_graph(image_5.dask)
```

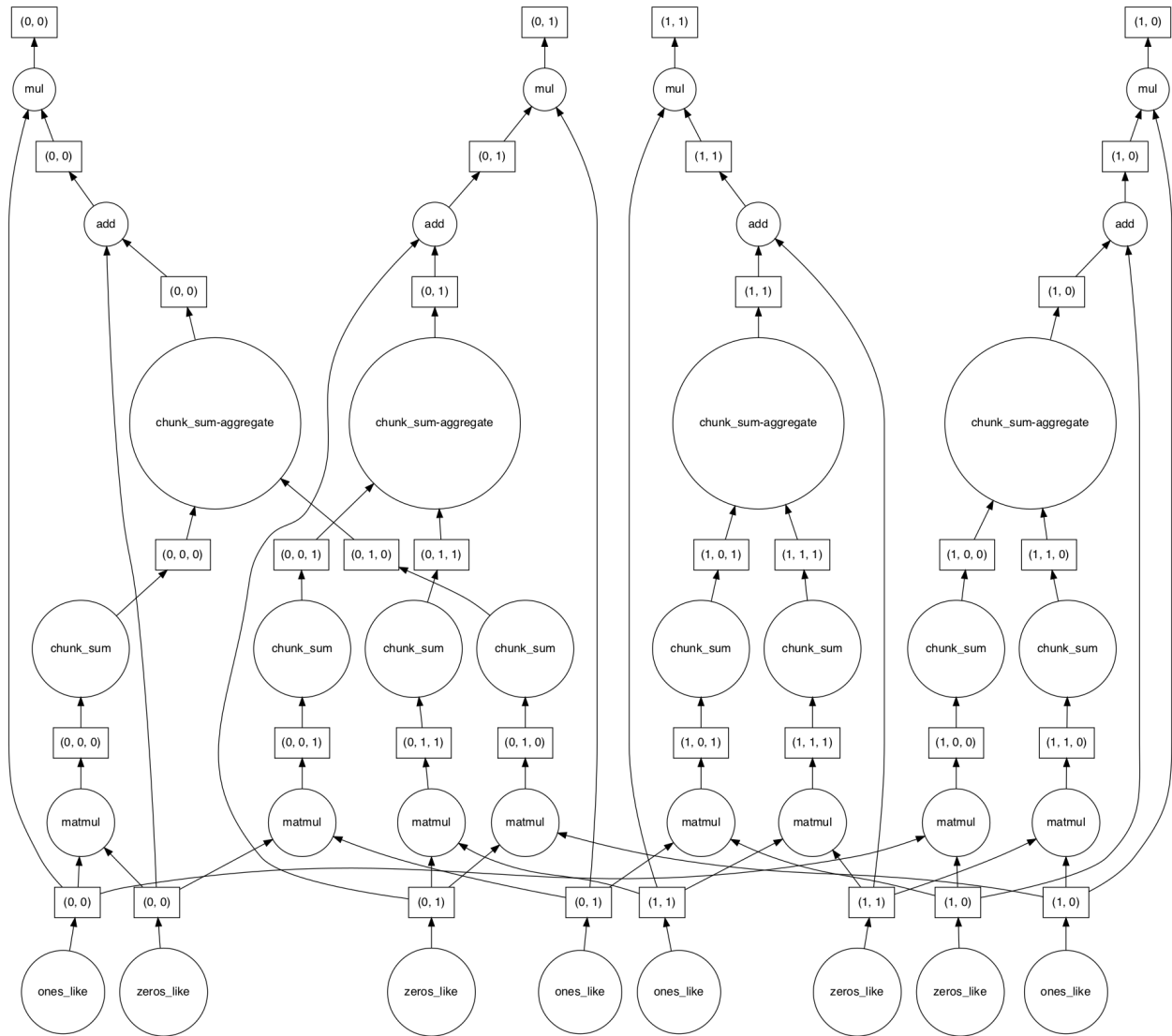


7.5.4 An even more complicated computation

Combining different operations into a final calculation like

$$Image_6 = (Image_1 \cdot Image_2 + Image_1) \circ Image_2$$

```
image_6 = (da.matmul(image_1, image_2)+image_1)*image_2
dot_graph(image_6.dask)
```



We see here that the convolution part connects all branches of the graph which mean all processing must be synchronized here. Afterwards there are no connections and each sub graph can work independently.

7.5.5 Convolution using a DAG

Convolution makes life harder for a DAG. There is a need for an overlap between the chunks to avoid boundary effects. These boundary effects are something we will look into in next week's lecture about filters.

$$Image_7 = Image_6 * Image_1$$

```
import dask_ndfilters as da_ndfilt
image_7 = da_ndfilt.convolve(image_6, image_1)
dot_graph(image_7.dask)
```

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
Input In [41], in <cell line: 1>()
```

(continues on next page)

(continued from previous page)

```

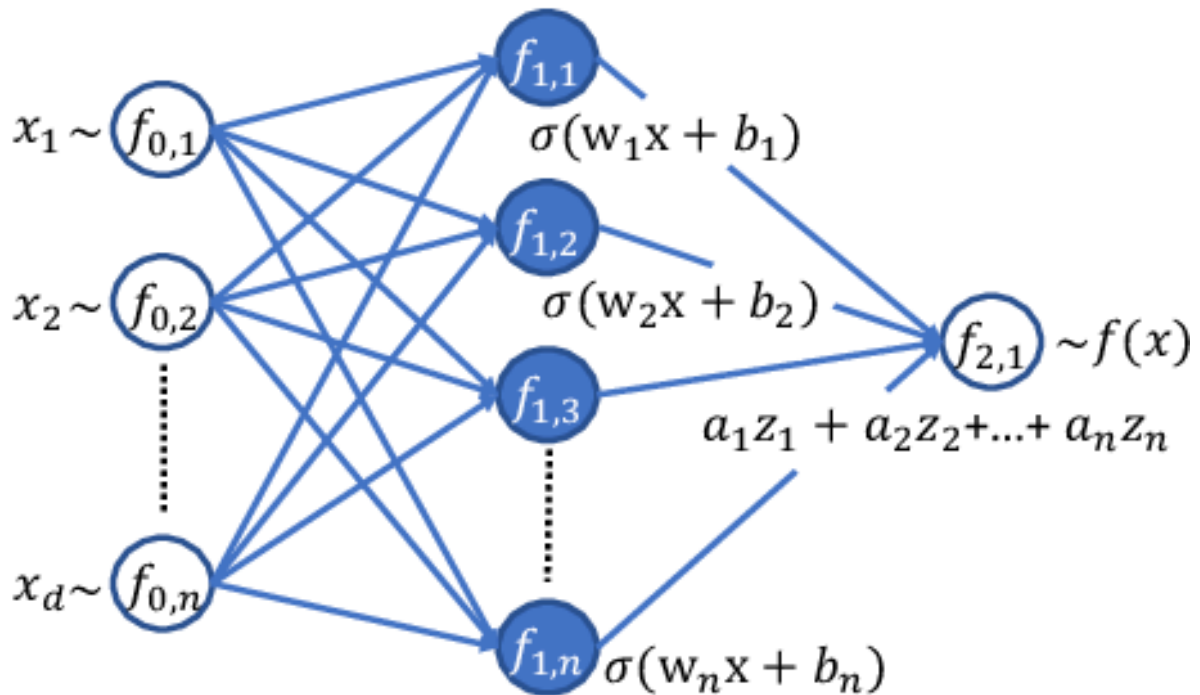
----> 1 import dask_ndfilters as da_ndfilt
      2 image_7 = da_ndfilt.convolve(image_6, image_1)
      3 dot_graph(image_7.dask)

```

```
ModuleNotFoundError: No module named 'dask_ndfilters'
```

7.6 Deep Learning

We won't talk too much about deep learning now, but it certainly shows why DAGs are so important.



The steps above are simple toys compared to what tools are already in use for machine learning

https://keras.io/api/utils/model_plotting_utils/

SUMMARY

In this lecture we saw that:

- Images reveal information about different samples
- Images are a signals that needs to be quantitatively analyzed
- Science with images is a non-trivial task
- Proper workflows are required for efficient analysis repeatable analysis.

8.1 Next weeks lecture

- Data sets
- Data augmentation
- Ground truth