Quantitative Big Imaging - Introduction

Anders Kaestner

Feb 20, 2021

CONTENTS

Quantitative Big Imaging ETHZ: 227-0966-00L

CHAPTER

ONE

TODAYS LECTURE

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

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.

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

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
```

CHAPTER TWO

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 Terrestial 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

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 were students came from in previos 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 nescessary 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.

A wide spectrum of backgrounds	A wide range of skills
Biomedical Engineers	
Physicists	I think I've heard of python before
Chemists	.
Art History Researchers	.
Mechanical Engineers	I write template C++ code and hand optimize it afterwards
and Computer Scientists	

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 moduls 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 scinetific 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 chosing 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.

Exercises
Usually 1 set per lecture
Optional (but recommended!)
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 guids 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.

Science project
Optional (but strongly recommended)
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)
Present approach, analysis, and results

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.

2.5 Projects

- 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

2.6 Course Overview

Topic	Date	Title	Description
Introduction	25th February	Introduction and Workflows	Basic overview of the course, introduction to …
Data	4th March	Image Enhancement	Overview of what techniques are available for …
	11th March	Ground Truth: Building and Augmenting Datasets	Examples of large datasets, how they were buil…
Segmentation	18th March	Basic Segmentation, Discrete Binary Structures	How to convert images into structures, startin…
	25th March	Advanced Segmentation	More advanced techniques for extracting struct…
	1st April	Supervised Problems and Segmentation	More advanced techniques for extracting struct…
	8th April	Easter break	Search for eggs
Analysis	15th April	Analyzing Single Objects, Shape, and Texture	The analysis and characterization of single st…
	22th April	Analyzing Complex Objects and Distributions	What techniques are available to analyze more …
	29th April	Dynamic Experiments	Performing tracking and registration in dynami…
Big Imaging	6th May	Imaging with multiple modalities	Combining information from different sources
	13th May	Ascension	Enjoy a lovely early summers day
	20th May	Scaling Up / Big Data	Performing large scale analyses on clusters
Wrapping up	27th May	Project Presentations	You present your projects

2.7 Today's 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

2.8 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

CHAPTER

THREE

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 publiction. The amout 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.

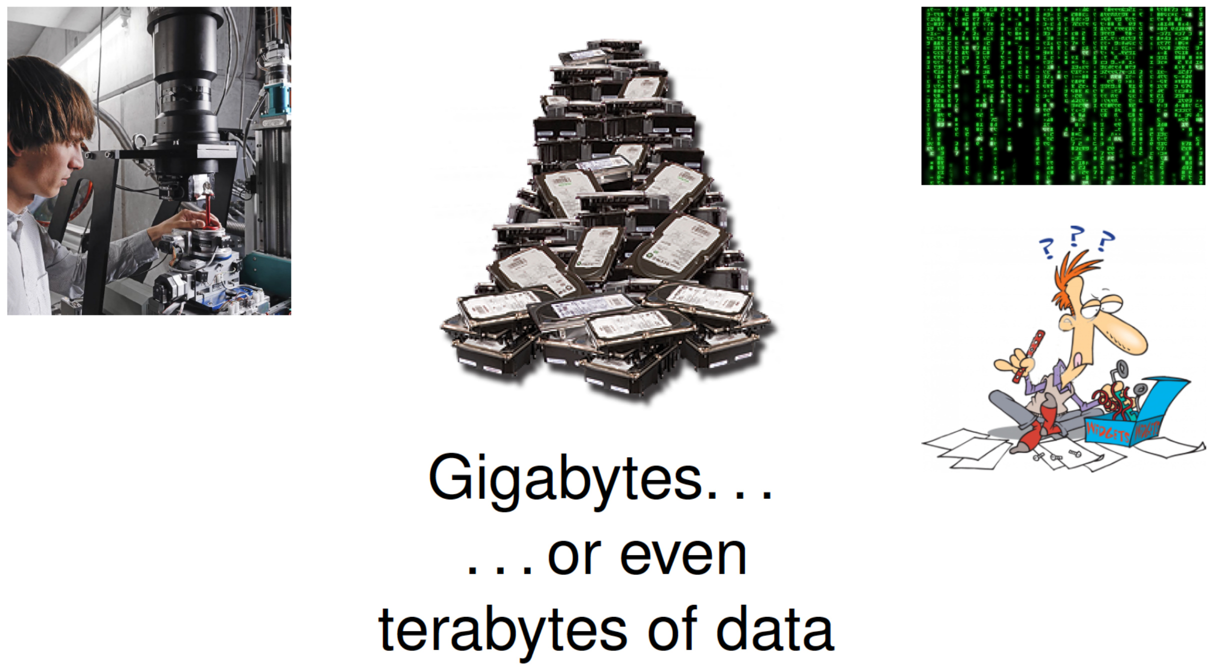


Fig. 3.1: A typical imaging experiment produces large amounts of data.

3.2 Motivation - how to proceed?

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

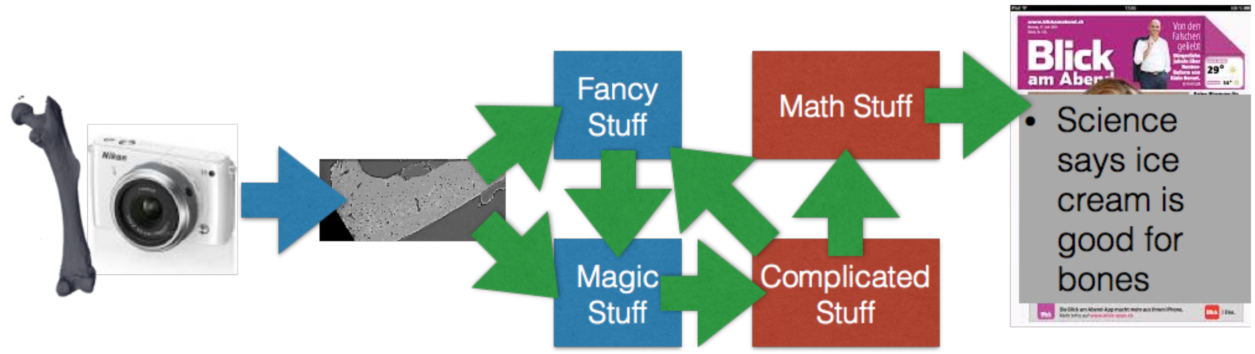


Fig. 3.2: A crazy unstructured and unclear work flow to analyze images from your experiment.

- 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)

3.3 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 acqisition 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
- Todays 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 to 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

3.4 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.

3.4.1 X-Ray

X-ray imaging at syncrotron 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) $\rightarrow$ 8GB/s
- cSAXS diffraction patterns at 30GB/s
- Nanoscopium Beamline, 10TB/day, 10-500GB file sizes

3.4.2 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 $\rightarrow$ 500MB/s
- High-speed confocal images at (>200fps) $\rightarrow$ 78Mb/s

3.4.3 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

3.5 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 remainging 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 posible 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.

3.5.1 How is time used during the experiment life cycle?

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

- Exeperiment 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.

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 beautify volume rendering or a movie of the sample. Meanwhile, it has become a requirement that you provide quantitative results from the images.

3.6 Handling masses of images

3.6.1 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 1000 x 1000 pixels (1 Mpixels)

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

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

plt.matshow(np.random.uniform(size = (1000, 1000)),
            cmap = 'viridis');
```

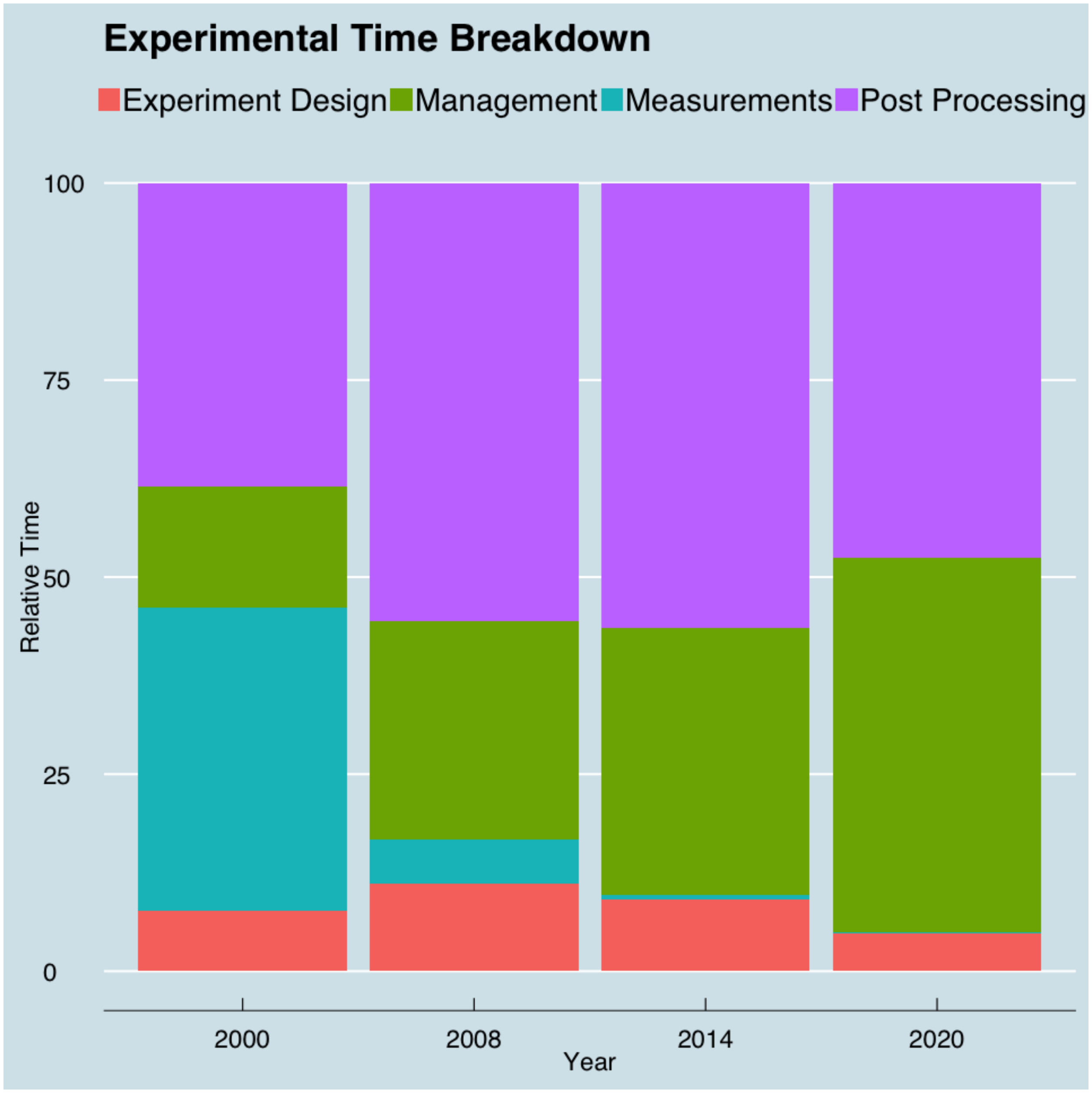
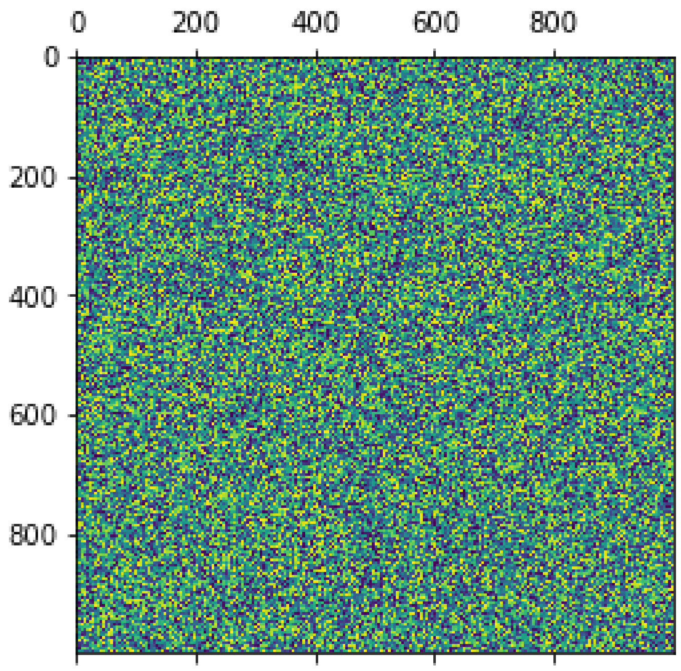


Fig. 3.3: The ratio of how much time is spent on different tasks during the lifecycle of an imaging experiment.



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
```

3.6.2 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.

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?

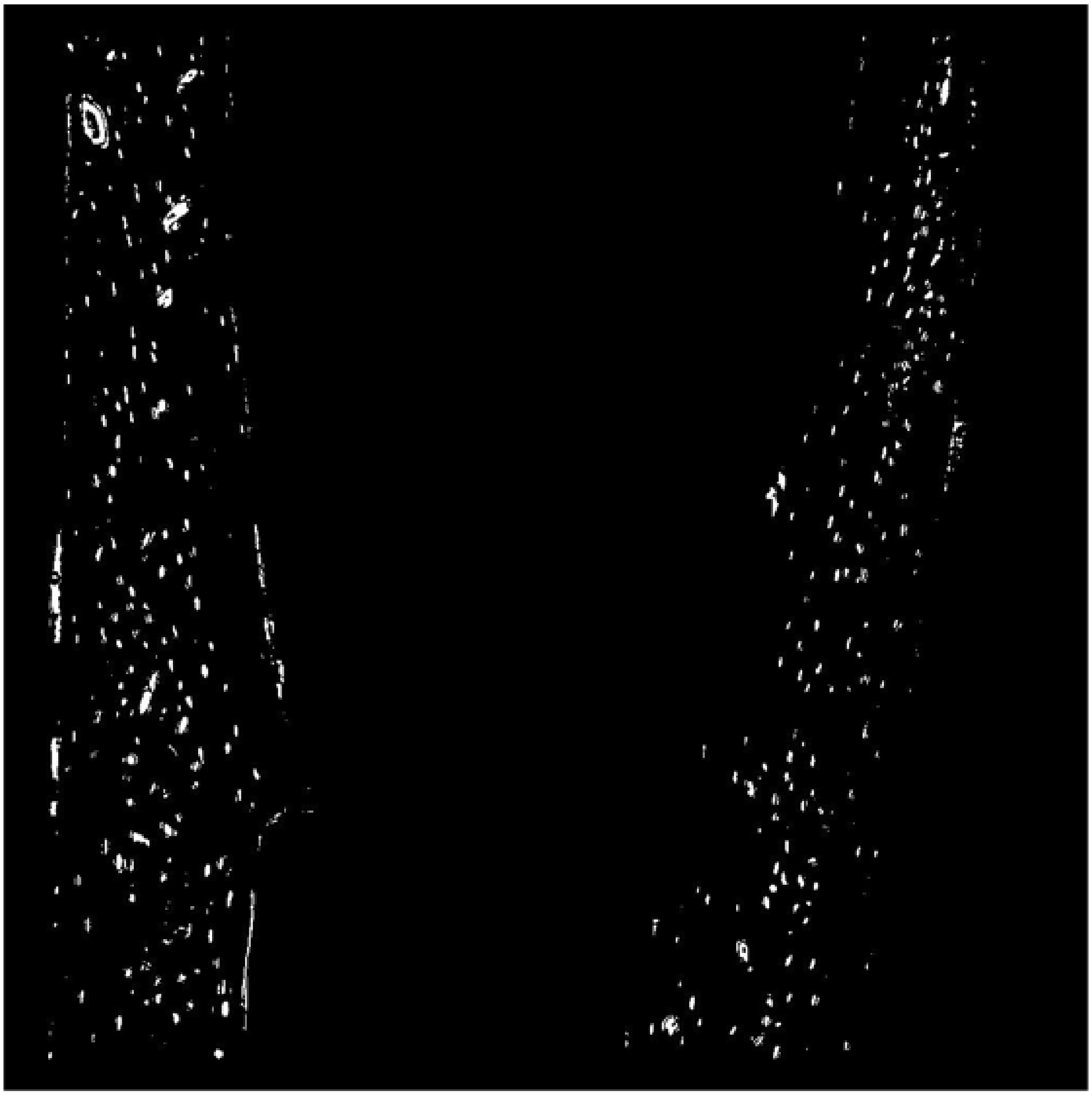


Fig. 3.4: A slice image show bone cells.

3.6.3 More overwhelmed scientist

```
Statistical analysis requires that you study many samples and not just a single one.␣
↪Furthermore, the
```

Many samples are needed:

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

3.6.4 Bring on the pain

Great variations in the population

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

3.7 It gets better

- 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?

3.8 Dynamic Information

- 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?

CHAPTER

FOUR

IMAGES

4.1 An introduction to images

4.1.1 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 (x,y,z coordinates) and a numeric value (intensity)

4.1.2 Image sampling

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

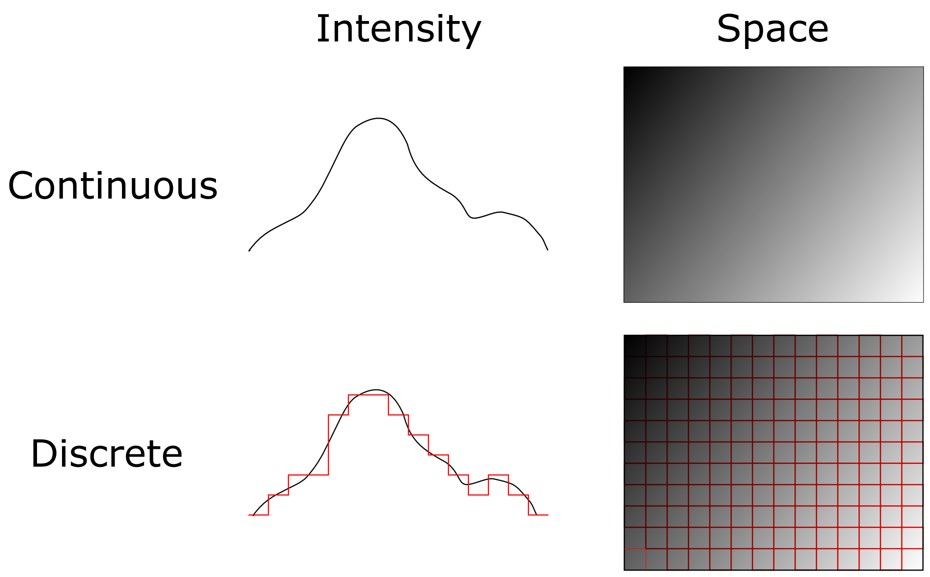
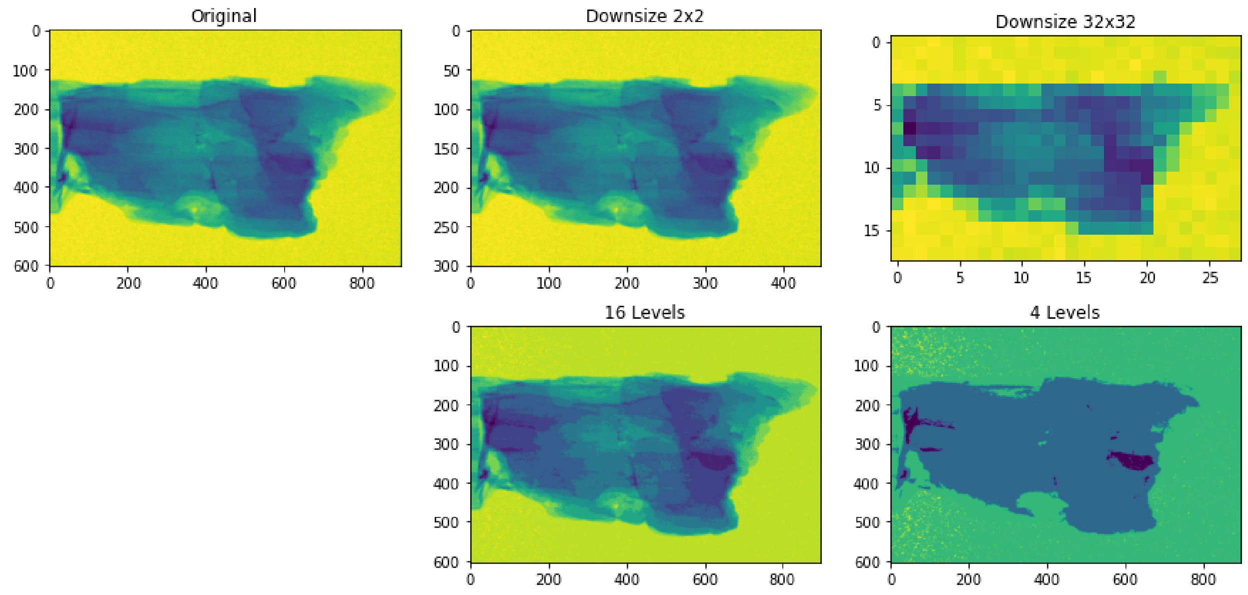


Fig. 4.1: The real world is sampled into discrete images with limited extent.

4.1.3 What does 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));
```



CHAPTER

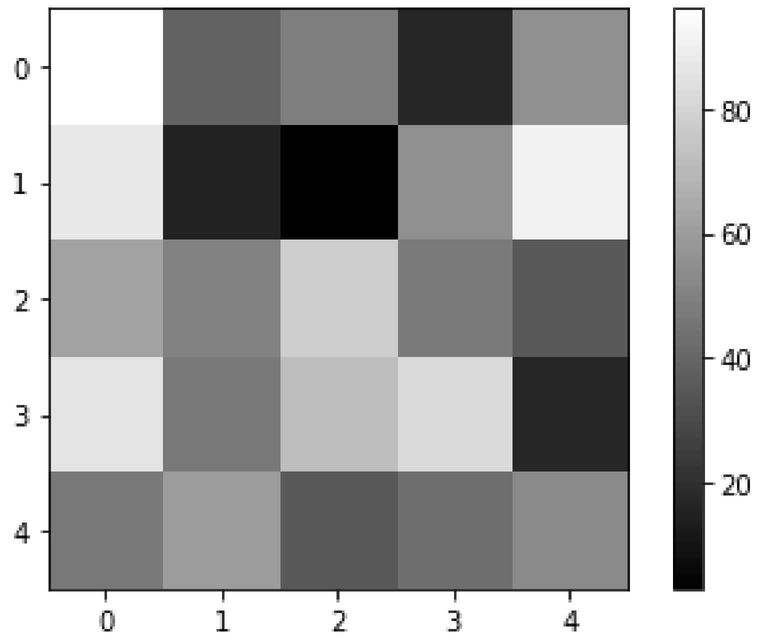
FIVE

LET'S CREATE A SMALL IMAGE

```
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         31
1  1  0         43
2  2  0         44
3  3  0          6
4  4  0         52
```

```
import matplotlib.pyplot as plt
plt.imshow(basic_image, cmap = 'gray')
plt.colorbar();
```

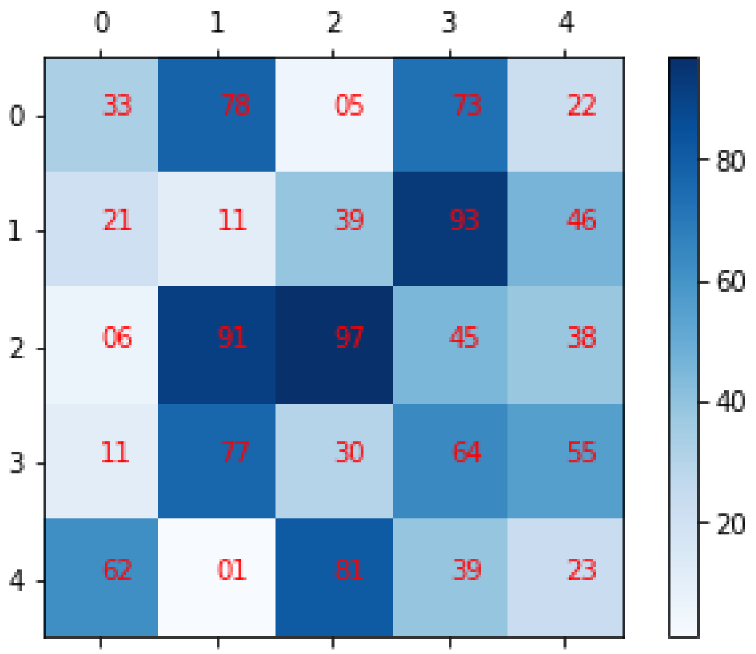


5.1 2D Intensity Images

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

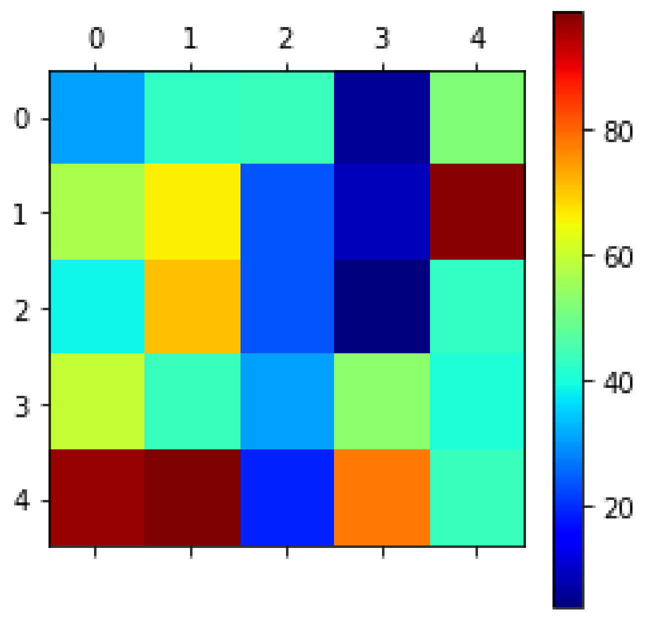
```
fig, ax1 = plt.subplots(1,1)
plot_image = ax1.matshow(basic_image, cmap = 'Blues')
plt.colorbar(plot_image)

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



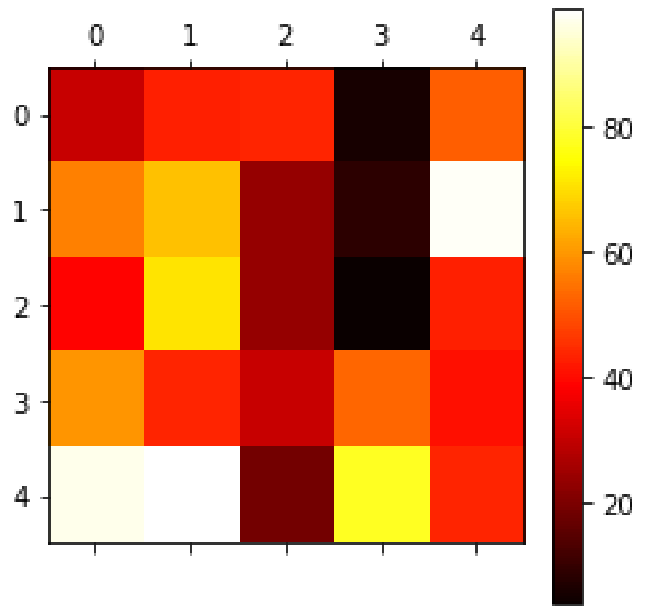
5.1.1 Different colormaps

Color maps can be arbitrarily defined based on how we would like to visualize the information in the image

```
plot_image = plt.matshow(basic_image, cmap = 'jet')
plt.colorbar(plot_image);
```



```
plot_image = plt.matshow(basic_image, cmap = 'hot')
plt.colorbar(plot_image);
```



5.1.2 Lookup Tables

Formally a color map is lookup table or a function which $f(Intensity) → Color$

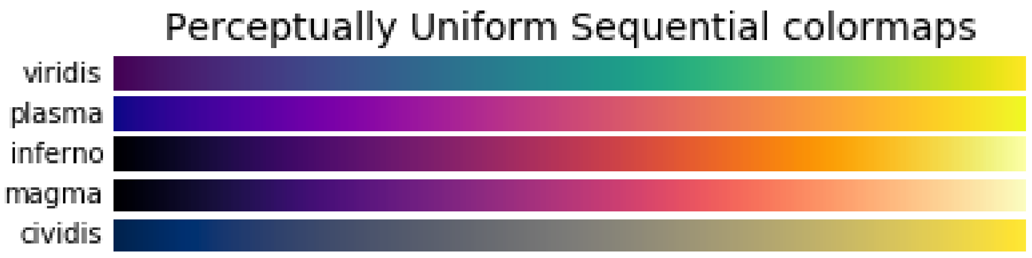
Matplotlib's color maps

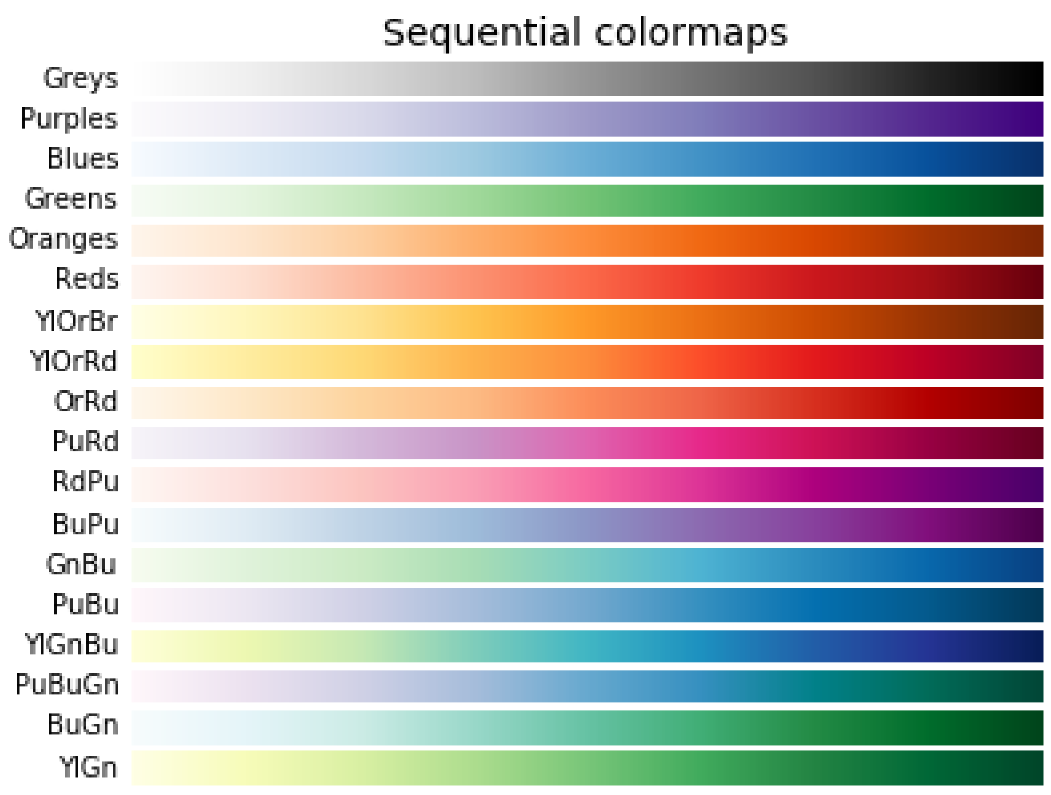
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import cm
#from colorspacious import cspace_converter
from collections import OrderedDict
cmaps = OrderedDict()
cmaps['Perceptually Uniform Sequential'] = [
            'viridis', 'plasma', 'inferno', 'magma', '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))
```

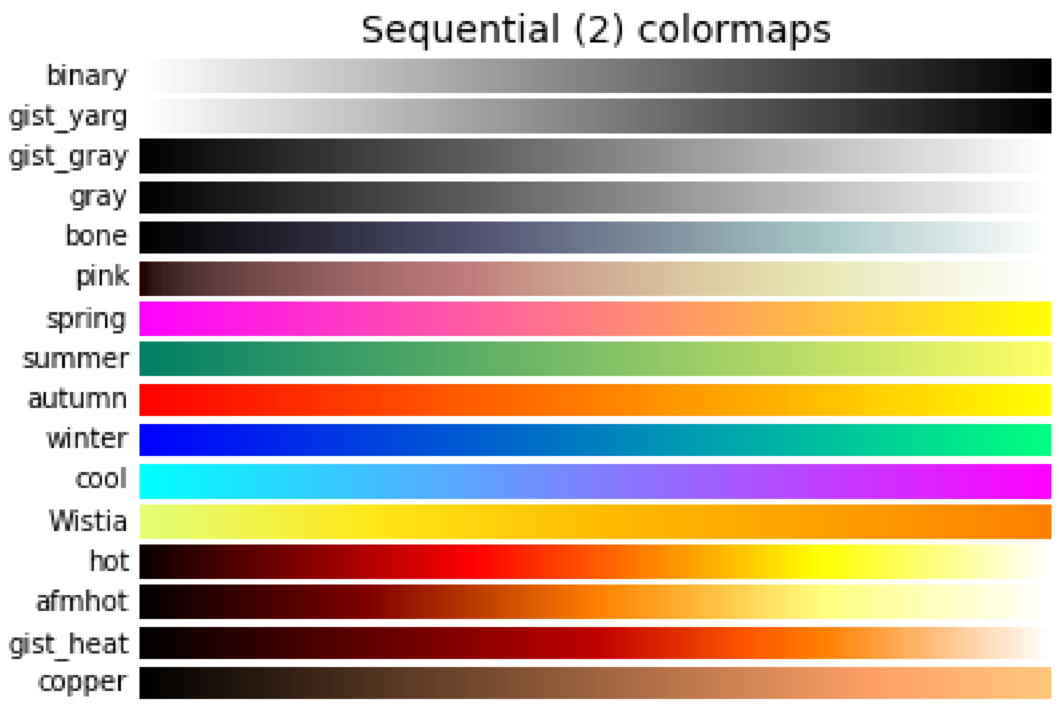
(continues on next page)

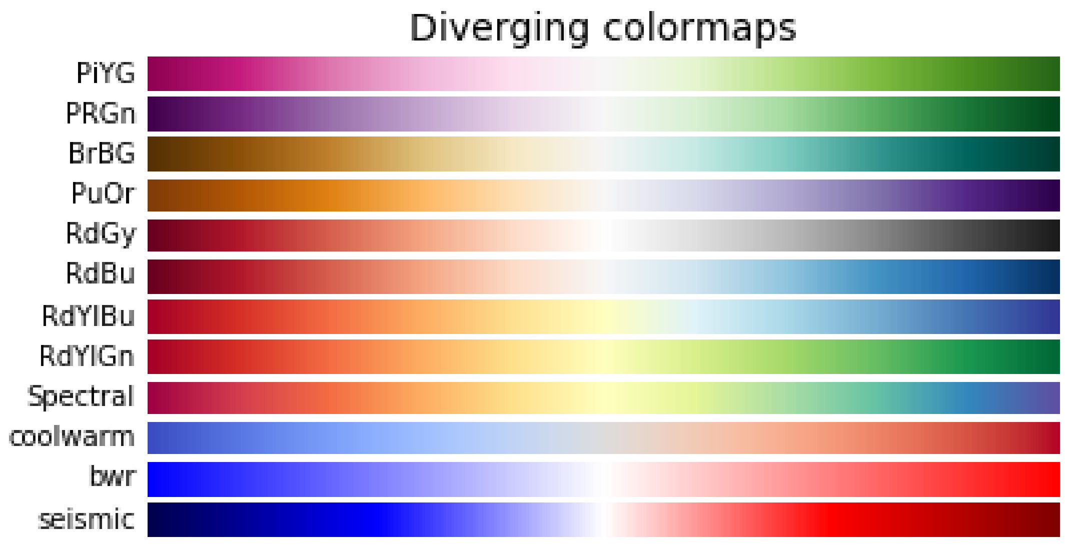
(continued from previous page)

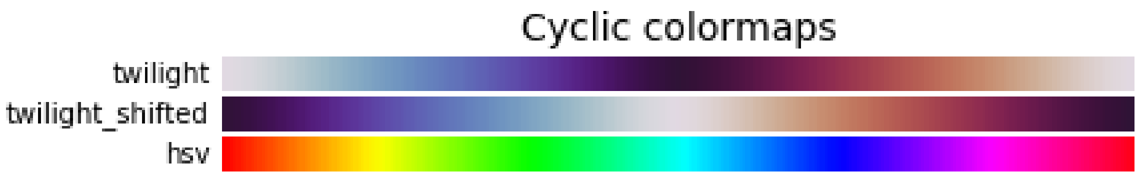
```
        pos = list(ax.get_position().bounds)
        x_text = pos[0] - 0.01
        y_text = pos[1] + pos[3]/2.
        fig.text(x_text, y_text, name, va='center', ha='right', fontsize=10)

    # 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)
```

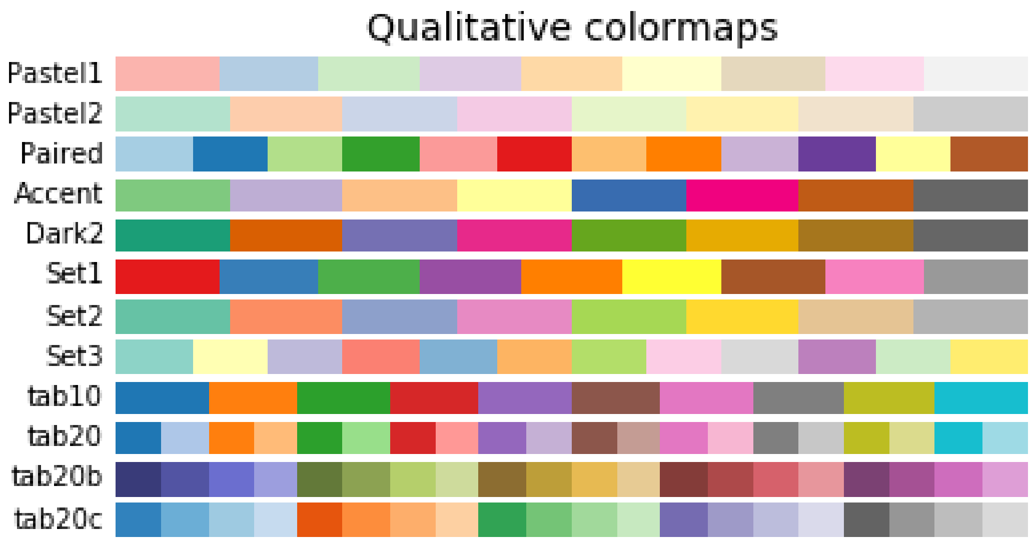


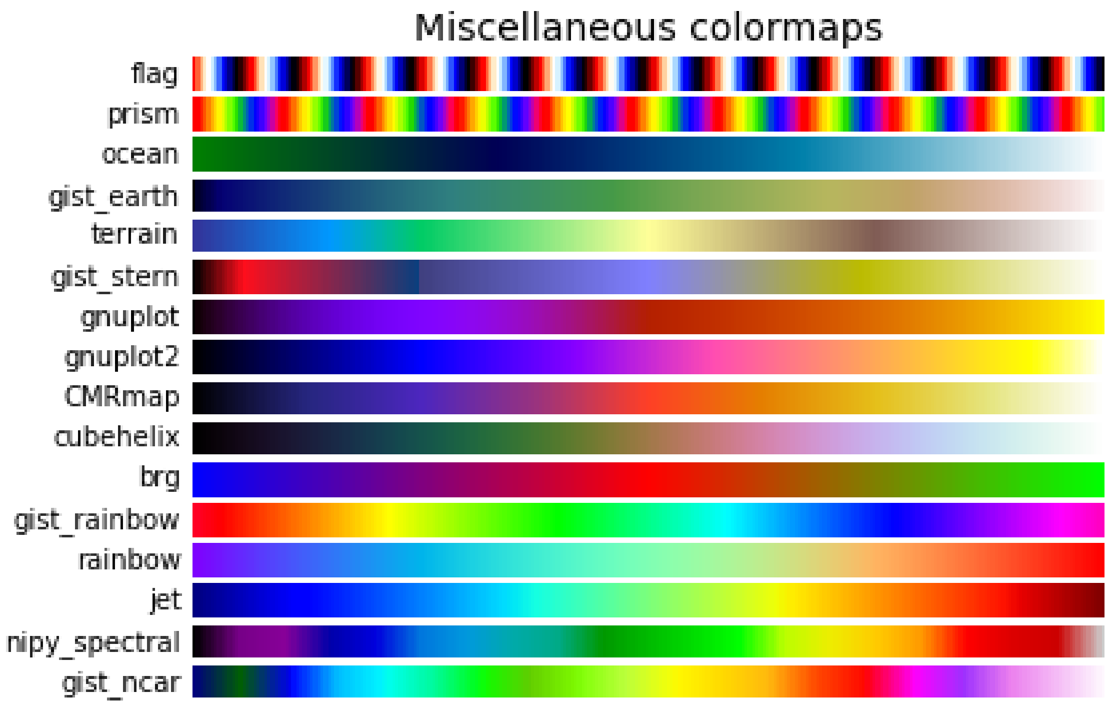

Sequential colormaps
Greys
Purples
Blues
Greens
Oranges
Reds
YlOrBr
YlOrRd
OrRd
PuRd
RdPu
BuPu
GnBu
PuBu
YlGnBu
PuBuGn
BuGn
YlGn


Sequential (2) colormaps
binary
gist_yarg
gist_gray
gray
bone
pink
spring
summer
autumn
winter
cool
Wistia
hot
afmhot
gist_heat
copper


Diverging colormaps
PiYG
PRGn
BrBG
PuOr
RdGy
RdBu
RdYlBu
RdYlGn
Spectral
coolwarm
bwr
seismic

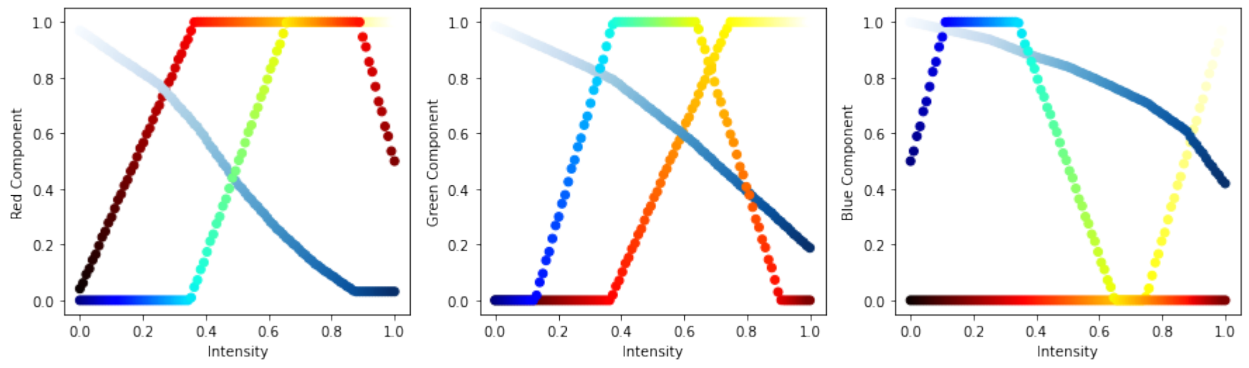

Cyclic colormaps
twilight
twilight_shifted
hsv


Qualitative colormaps
Pastel1
Pastel2
Paired
Accent
Dark2
Set1
Set2
Set3
tab10
tab20
tab20b
tab20c


Miscellaneous colormaps
flag
prism
ocean
gist_earth
terrain
gist_stern
gnuplot
gnuplot2
CMRmap
cubehelix
brg
gist_rainbow
rainbow
jet
nipy_spectral
gist_ncar

5.1.3 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]));
```



5.2 Applied LUTs

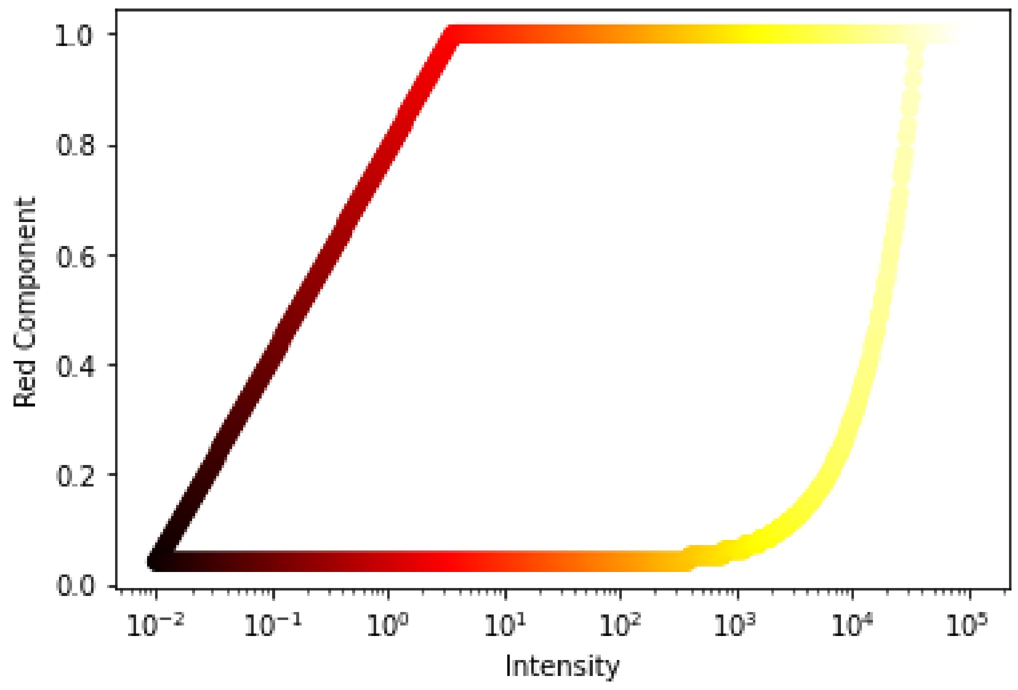
These transformations can also be non-linear as is the case of the graph below where the mapping between the intensity and the color is a log relationship meaning the the difference between the lower values is much clearer than the higher ones

```
%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)

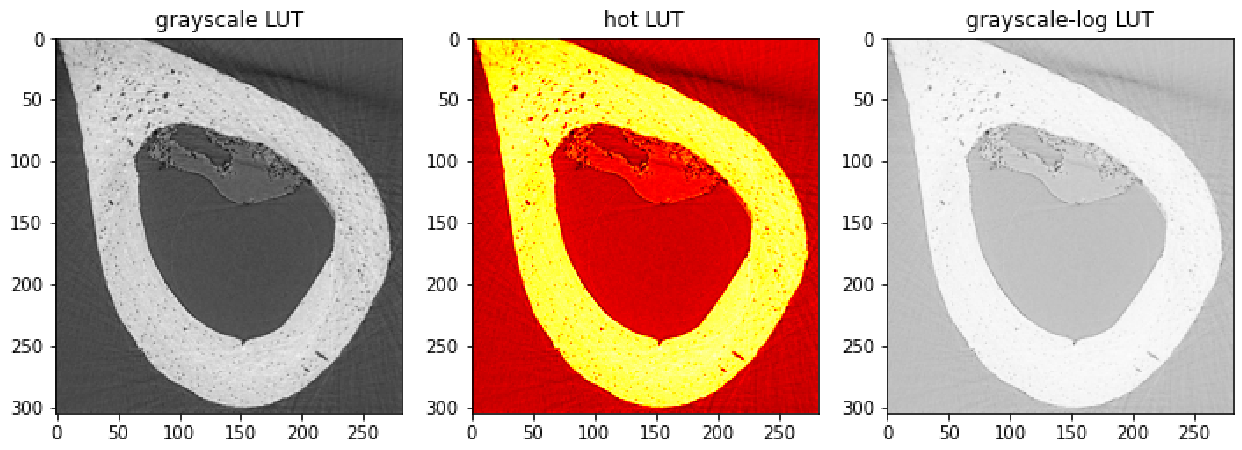
```
ax1.scatter(xlin, plt.cm.hot(xlin/xlin.max())[:,0],
            c = plt.cm.hot(norm_xlin))
ax1.set_xscale('log');ax1.set_xlabel('Intensity');ax1.set_ylabel('Red Component');
```



5.2.1 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');
```



5.3 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)

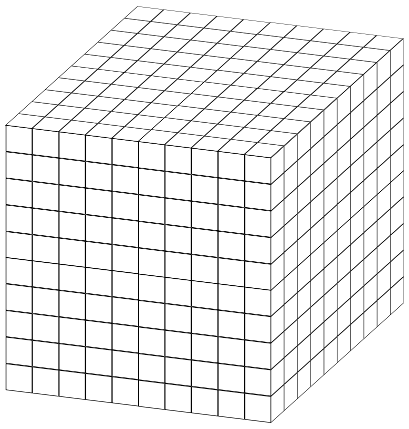


Fig. 5.1: Three-dimensional data can be a volume in space.

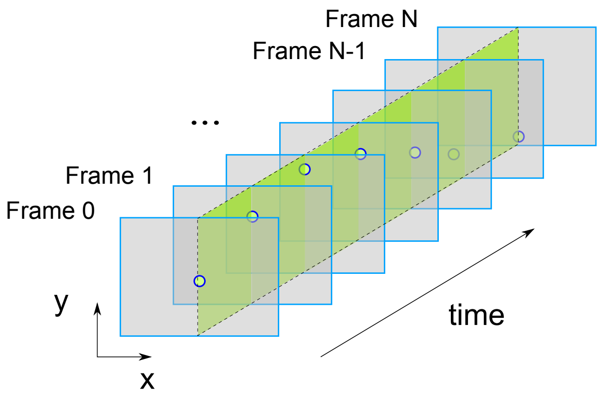


Fig. 5.2: A movie can also be seen as a three-dimensional image.

5.3.1 A 3D image as array

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

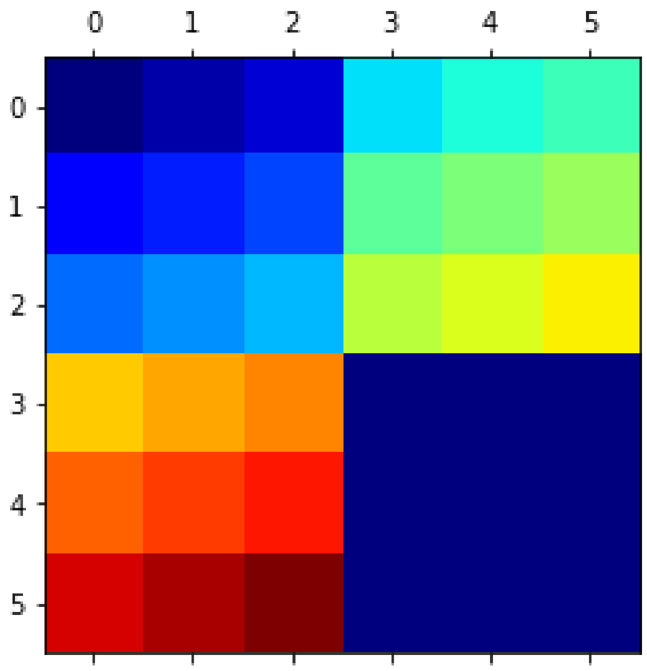
```
[[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]]

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

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

5.3.2 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]]
```



5.4 Multiple Values

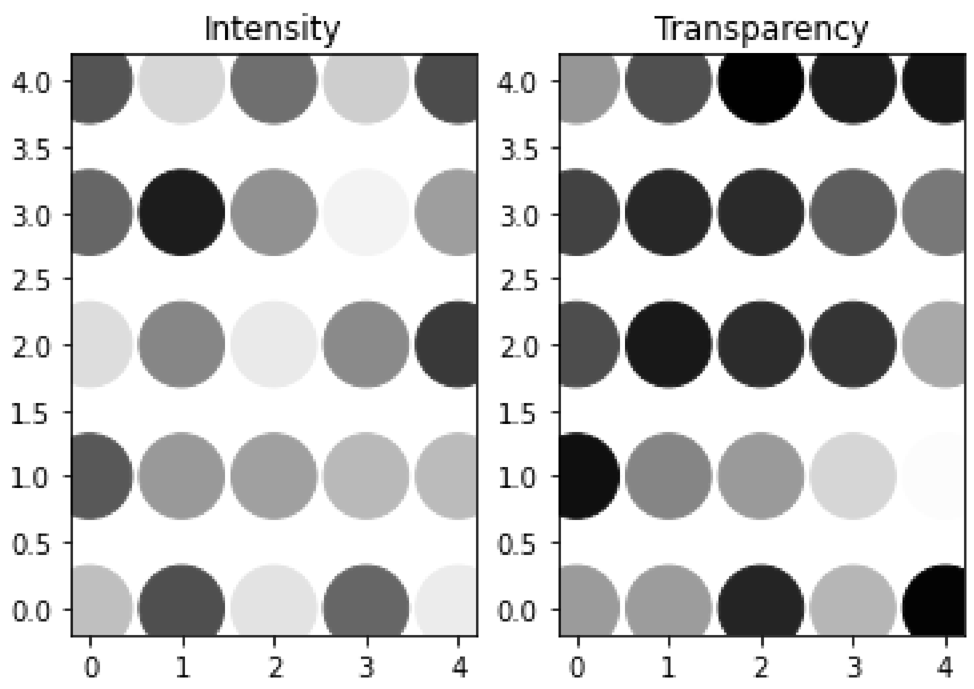
In the images thus far, we have had one value per position, but there is no reason there cannot be multiple values. In fact this is what color images are (red, green, and blue) values and even 4 channels with transparency (alpha) as a different. For clarity we call the **dimensionality** of the image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

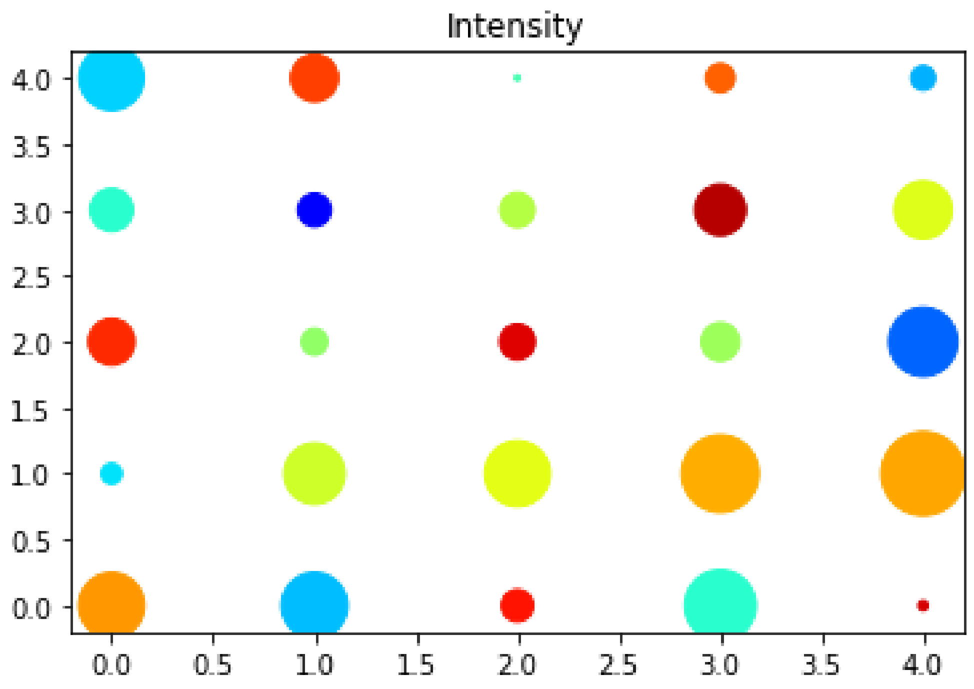
```
   x  y  Intensity  Transparency
0  0  0   0.746783      0.607095
1  0  1   0.347202      0.059656
2  0  2   0.866183      0.301186
3  0  3   0.402008      0.258386
4  0  4   0.328629      0.589196
```

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
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');
```



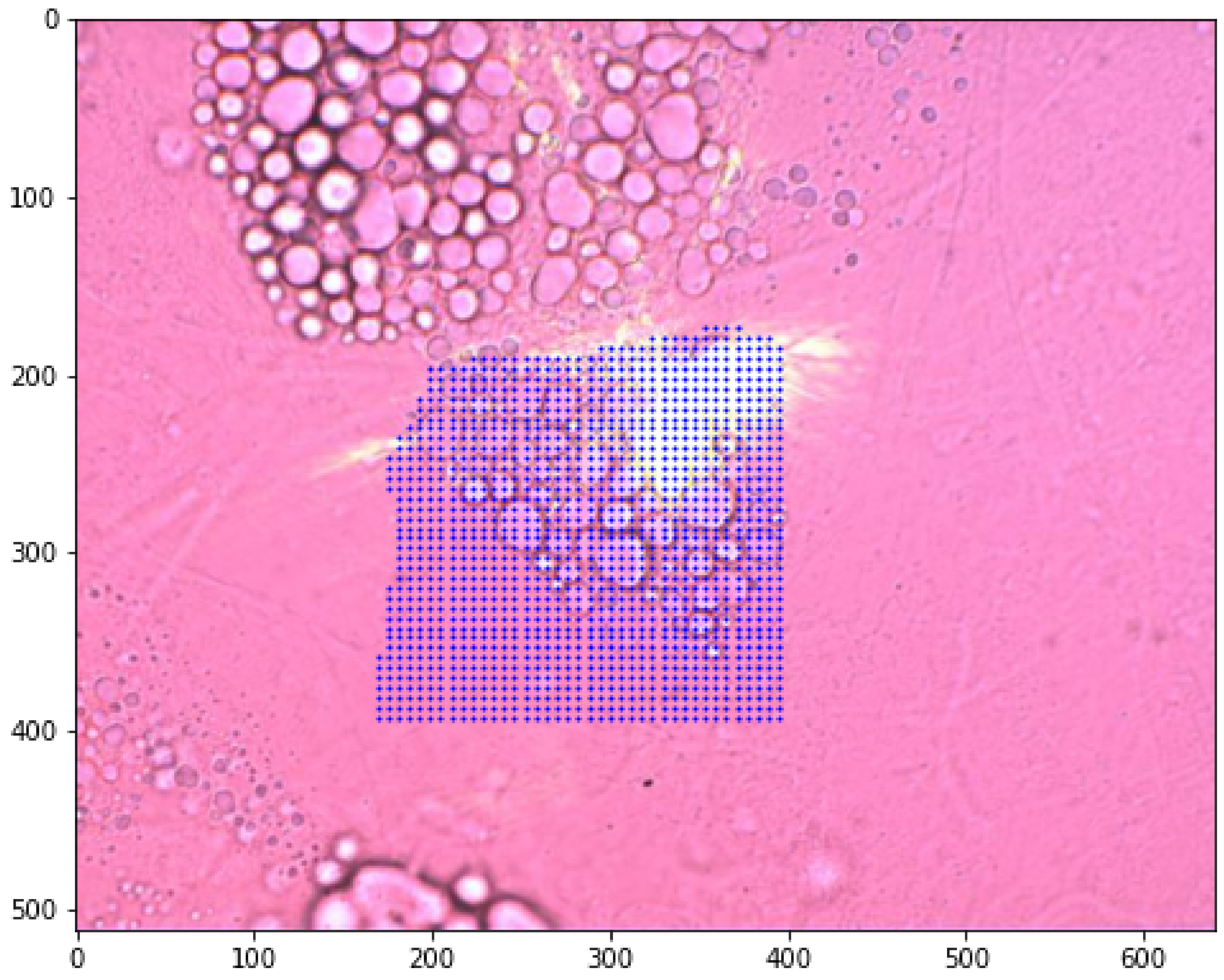
```
fig, (ax1) = plt.subplots(1, 1)
ax1.scatter(base_df['x'], base_df['y'], c = plt.cm.jet(base_df['Intensity']), s =␣
↪1000*base_df['Transparency'])
ax1.set_title('Intensity');
```



5.5 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 and 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');
```

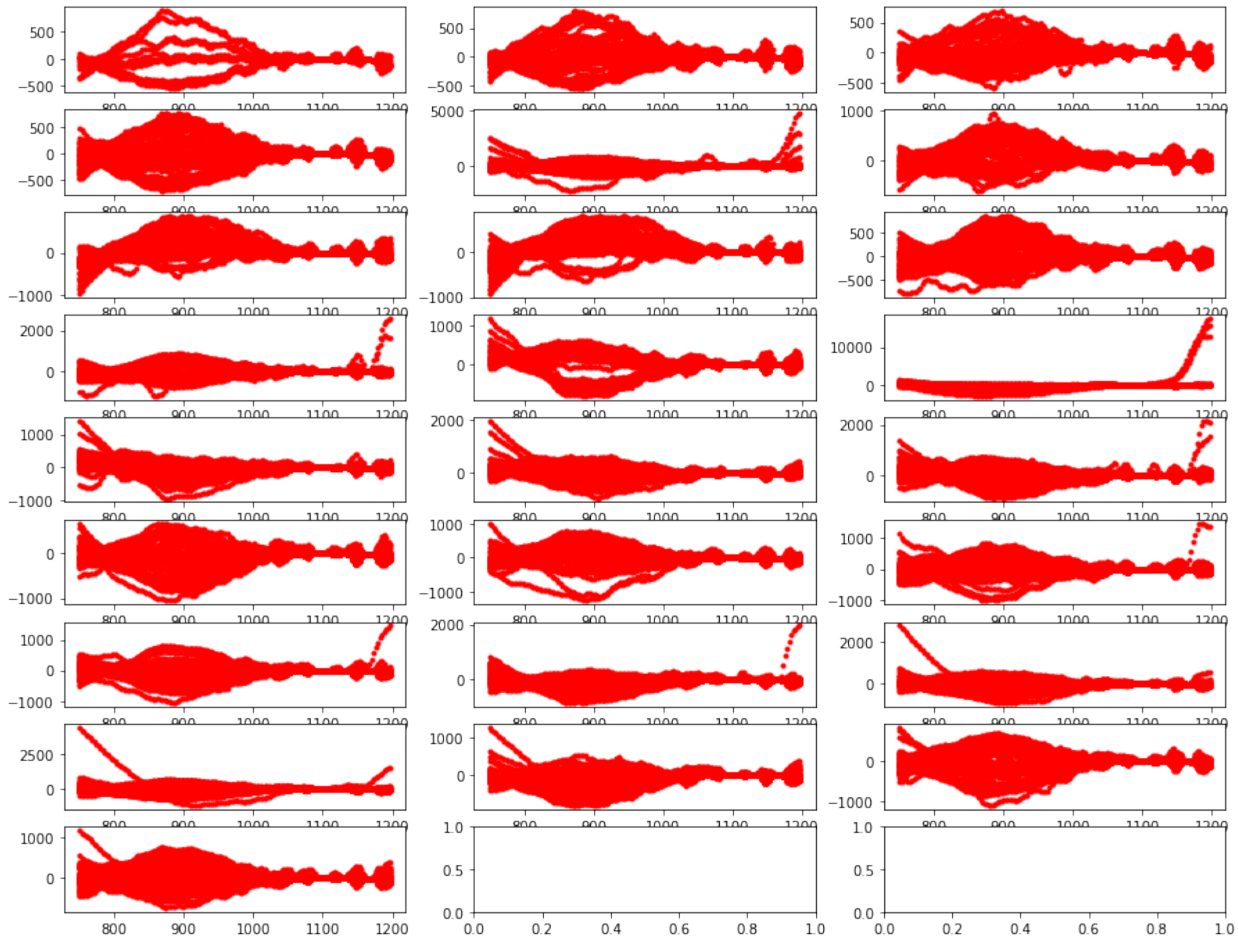


```
full_df = pd.read_csv('../../common/data/full_img.csv').query('wavenum<1200')
print(full_df.shape[0], 'rows')
full_df.head(5)
```

```
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.')
```



CHAPTER

SIX

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.

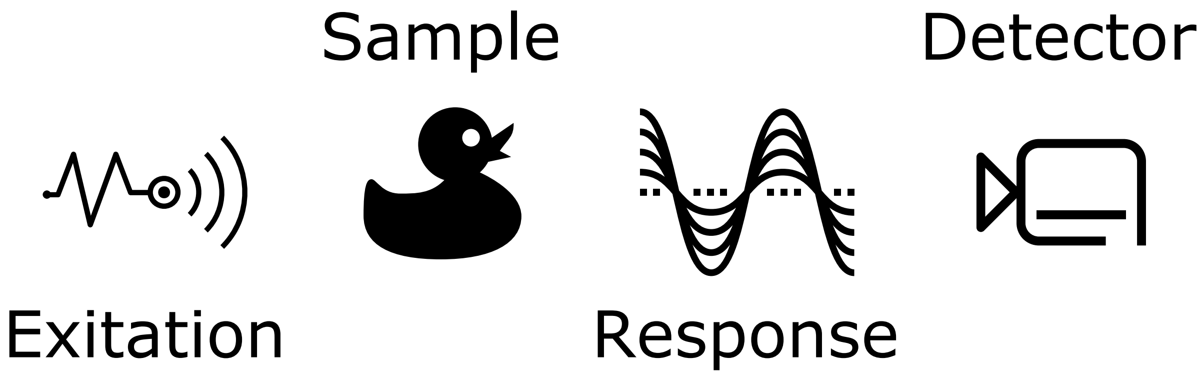


Fig. 6.1: The parts involved in the image formation process probing a sample.

- **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

6.1 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

CHAPTER

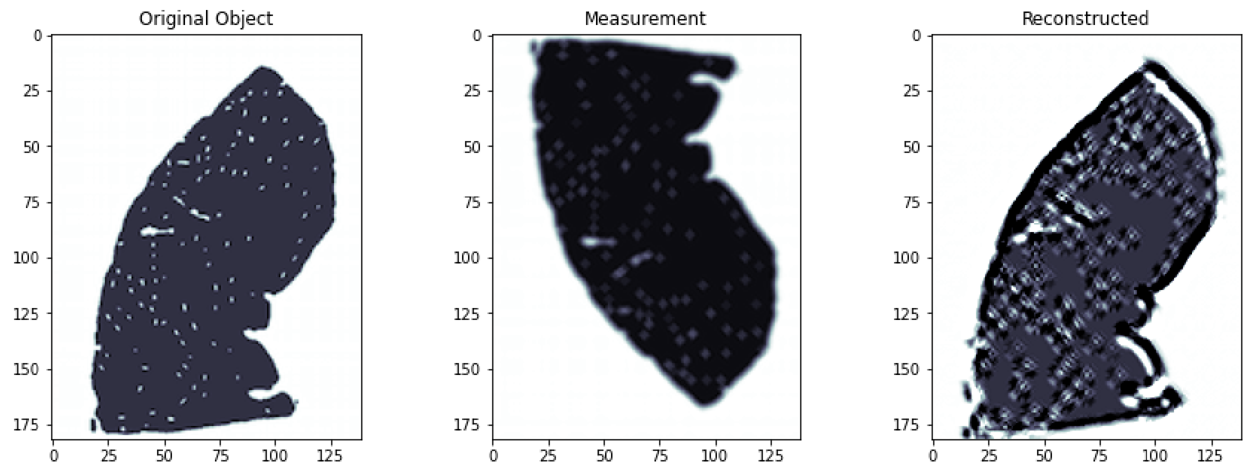
SEVEN

ACQUIRING IMAGES

7.1 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');
```

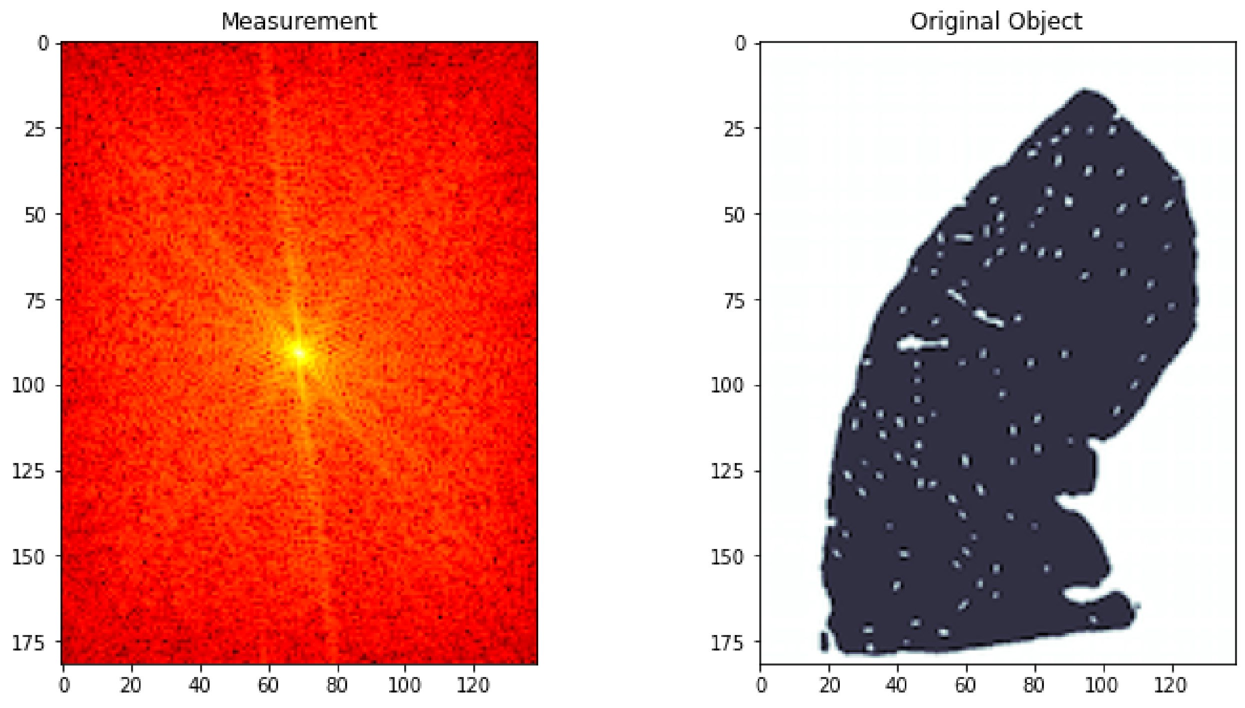


7.2 Indirect / Computational imaging

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

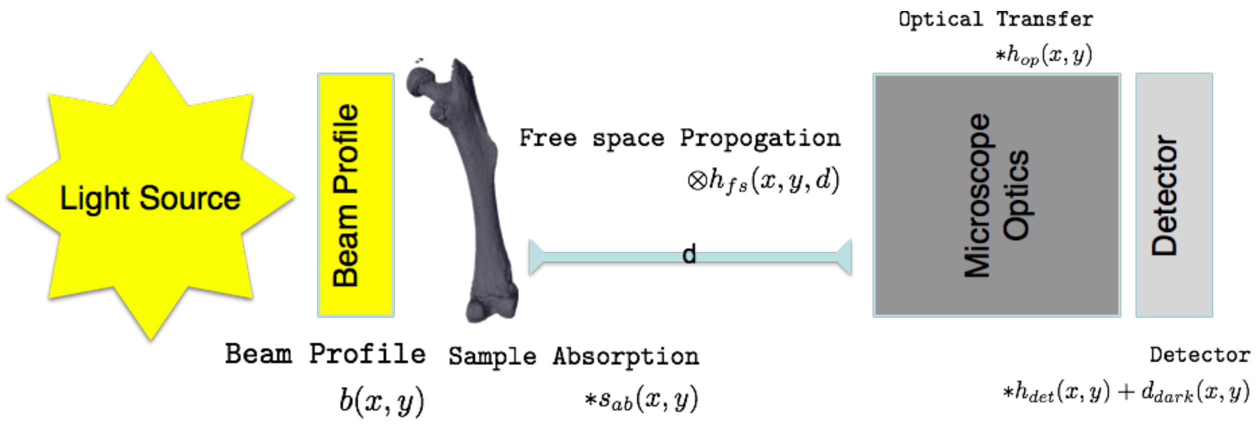
```
bone_img = imread('figures/tiny-bone.png').astype(np.float32)
# simulate measured image
meas_img = np.log10(np.abs(np.fft.fftshift(np.fft.fft2(bone_img))))
print(meas_img.min(), meas_img.max(), meas_img.mean())
fig, (ax1, ax_orig) = plt.subplots(1,2,
                              figsize = (12, 6))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



7.3 Traditional Imaging

7.4 Traditional Imaging: Model



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

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)\mathrm{d}\lambda+d_{dark}(x,y)$\$

7.5 Indirect Imaging (Computational Imaging)

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

7.6 Image Analysis

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

7.7 Image Analysis: Experimentalist

7.7.1 Characteristics

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

7.7.2 Examples

- cell counting
- porosity

7.8 Image Analysis: Computer Vision Approaches

7.8.1 Characteristics

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

7.8.2 Examples

- Edge detection
- Face detection

7.9 Image Analysis: Deep Learning Approach

7.9.1 Characteristics

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

7.9.2 Examples

- Captioning images
- Identifying unusual events

CHAPTER

EIGHT

ON SCIENCE

8.1 What is the purpose?

- Discover and validate new knowledge

8.1.1 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

8.1.2 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

8.2 Science and Imaging

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

8.2.1 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

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

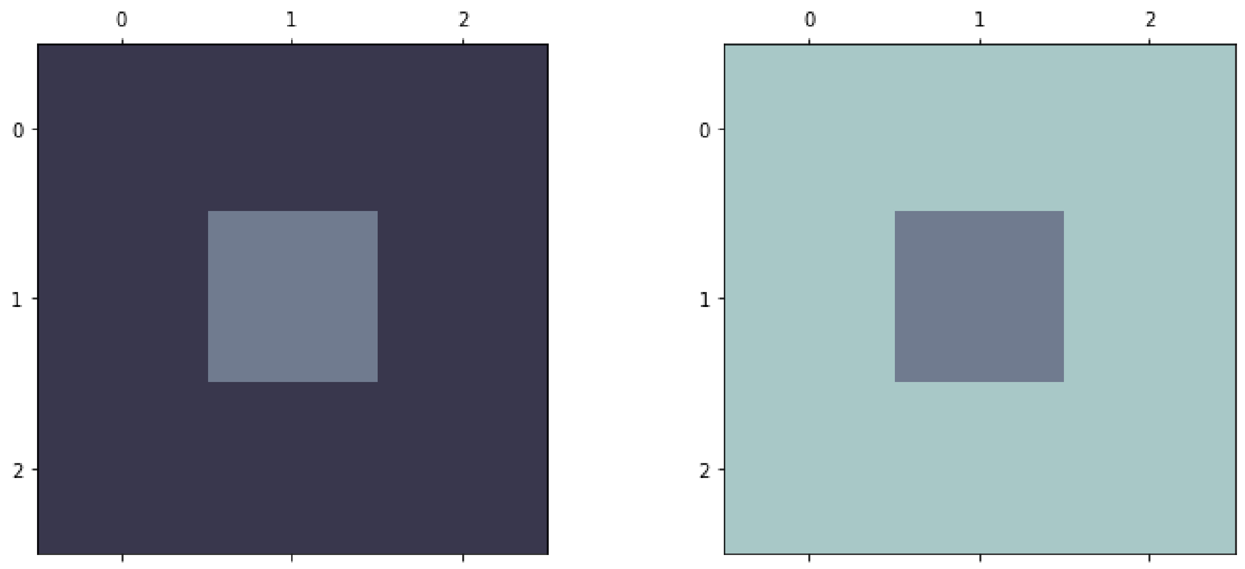
- Thousands of algorithms available
- Thousands of tools
- Many images require multi-step processing
- Experimenting is time-consuming

8.3 Why quantitative?

8.3.1 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');
```

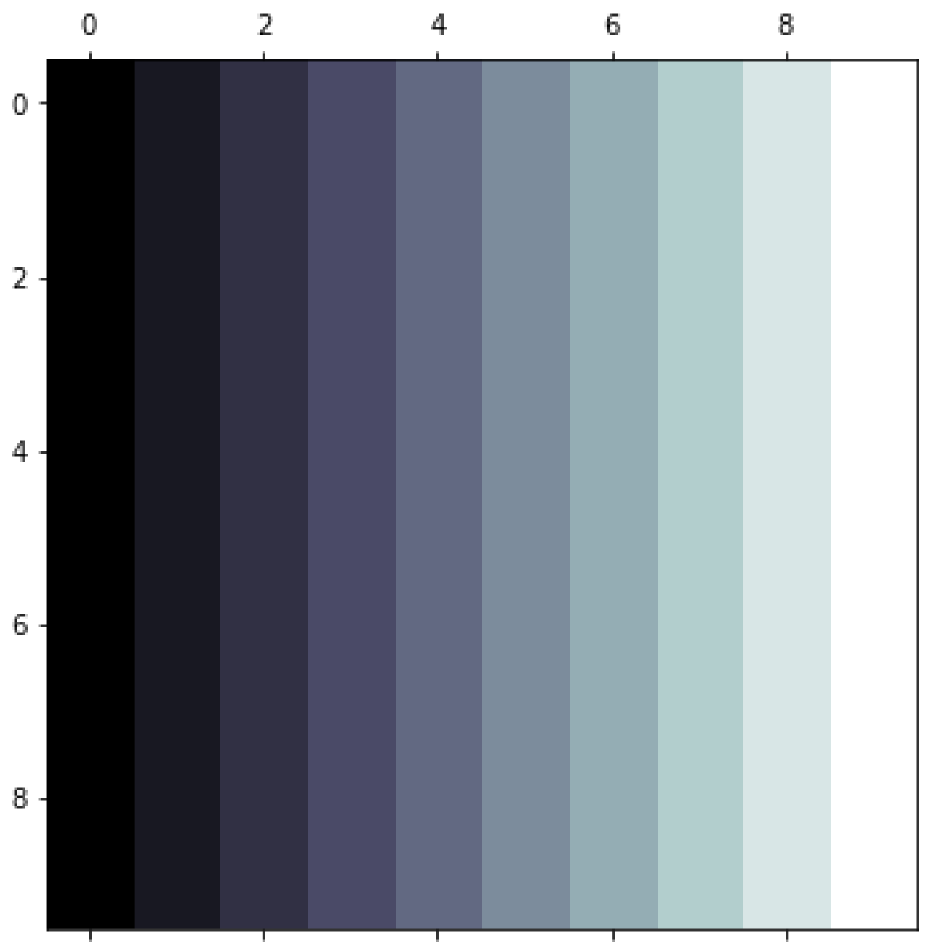


8.4 Intensity gradients

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')
```

```
<matplotlib.image.AxesImage at 0x1c5b4853d0>
```



8.5 Reproducibility vs. Repeatability

8.5.1 Reproducibility

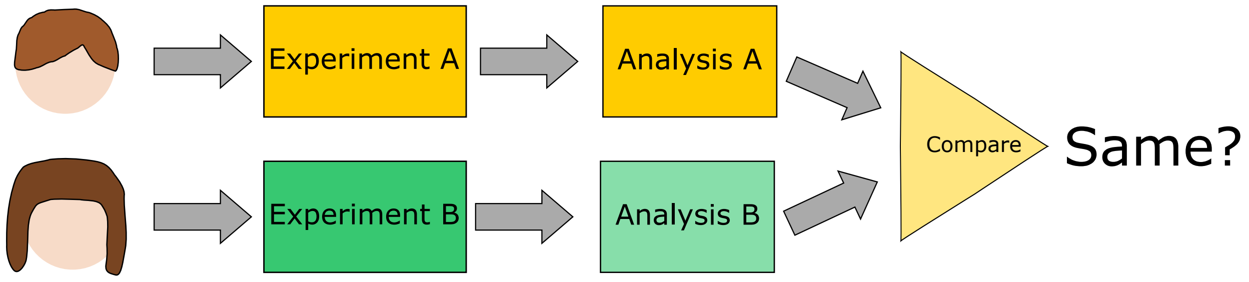


Fig. 8.1: A workflow describing the concept of reproducibility.

8.5.2 Repeatability

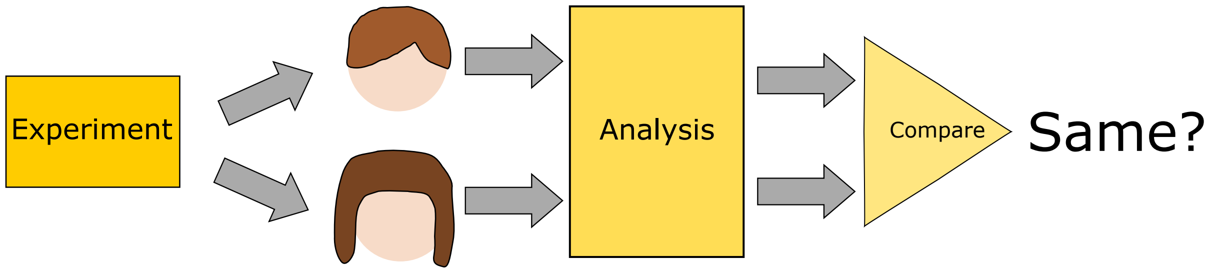


Fig. 8.2: A workflow describing the concept of reproducibility.

8.6 Reproducibility vs. Repeatability

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

- 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

8.7 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

CHAPTER

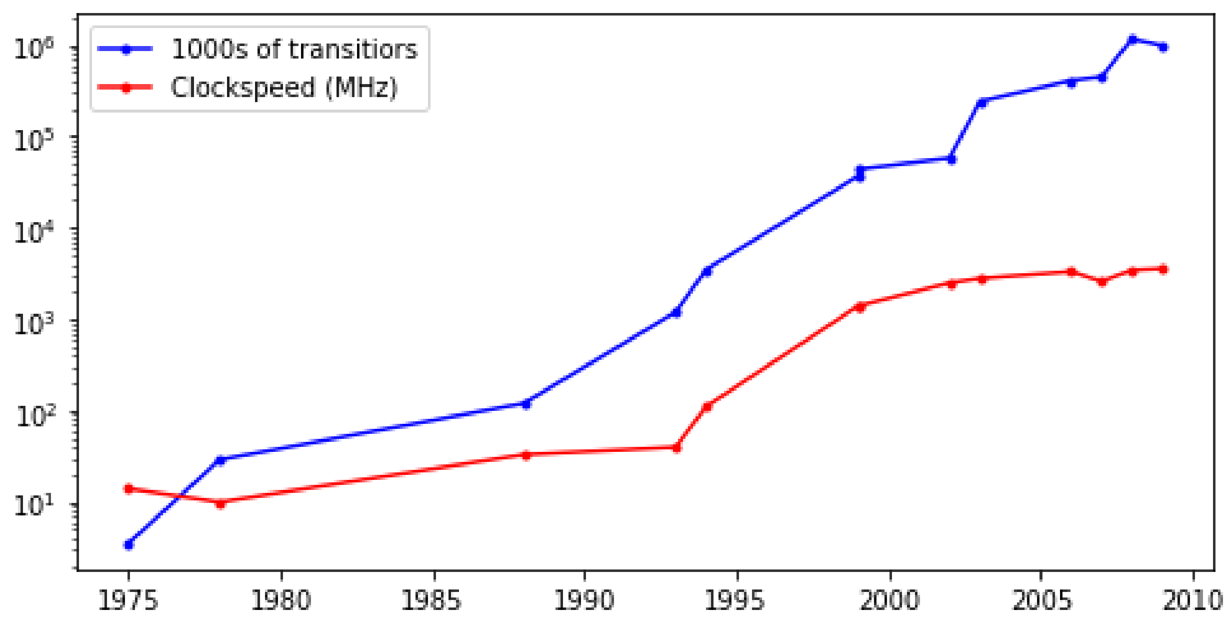
NINE

COMPUTING HAS CHANGED: PARALLEL

9.1 Moores Law

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

```
%matplotlib inline
# stolen from https://gist.github.com/humberto-ortiz/de4b3a621602b78bf90d
import pandas as pd
import matplotlib.pyplot as plt
from io import StringIO
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 transitiors'); 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

---

There are now many more transistors inside a single computer but the processing speed hasn't increased. How can this be?

- Multiple Core
- Many machines have multiple cores for each processor which can perform tasks independently
- Multiple CPUs
- More than one chip is commonly present
- New modalities
  - GPUs provide many cores which operate at slow speed

9.1.1 → Parallel Code is important

9.2 Computing has changed: Cloud

- 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/
  - "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

9.3 Cloud Computing Costs

The figure shows the range of cloud costs (determined by peak usage) compared to a local workstation with utilization shown as the average number of hours the computer is used each week.

9.4 Cloud: Equal Cost Point

Here the equal cost point is shown where the cloud and local workstations have the same cost. The x-axis is the percentage of resources used at peak-time and the y shows the expected usable lifetime of the computer. The color indicates the utilization percentage and the text on the squares shows this as the numbers of hours used in a week.

CHAPTER

TEN

SOUP/RECIPE EXAMPLE

10.1 Simple Soup

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

10.2 More complicated soup

Here it is harder to follow and you need to carefully keep track of what is being performed

10.2.1 Steps 1-4

1. *then* Mix carrots with potatos $\rightarrow mix_1$
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

CHAPTER

ELEVEN

USING FLOW CHARTS / WORKFLOWS

11.1 Simple Soup

```
from IPython.display import SVG
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))

SVG(graph.create_svg())
```

```
<IPython.core.display.SVG object>
```

11.2 Workflows

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 pathsways

---

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 SVG
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)
```

(continues on next page)

(continued from previous page)

```
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')

SVG(graph.create_svg())
```

```
<IPython.core.display.SVG object>
```

CHAPTER

TWELVE

DIRECTED ACYCLICAL GRAPHS (DAG)

We can represent almost any computation without loops as DAG. What this allows us to do is now 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/

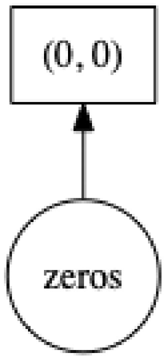
CHAPTER

THIRTEEN

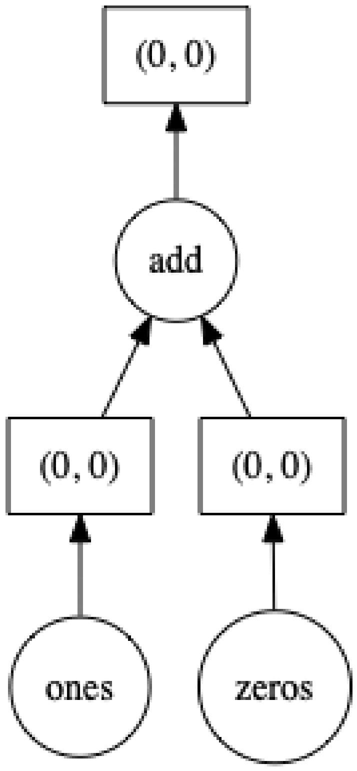
CONCRETE EXAMPLE

What is a DAG good for?

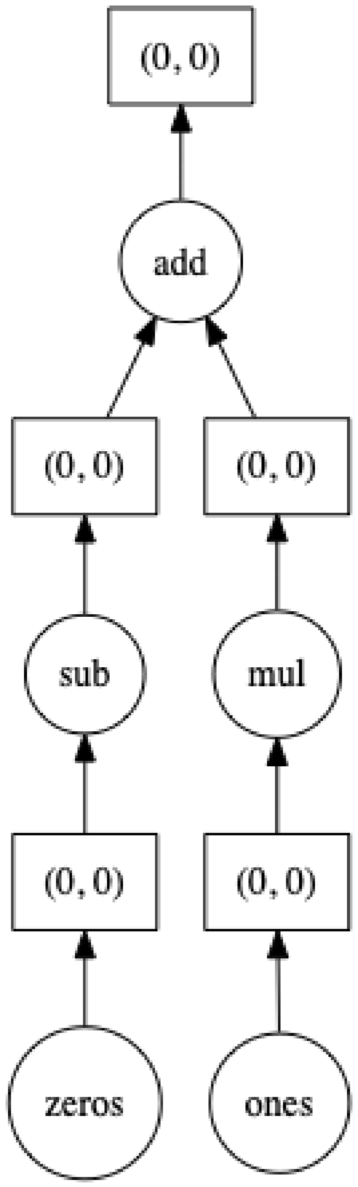
```
import dask.array as da

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)
```



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



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


(0, 0)
add
(0, 0)
(0, 0)
sub
mul
(0, 0)
(0, 0)
zeros
ones

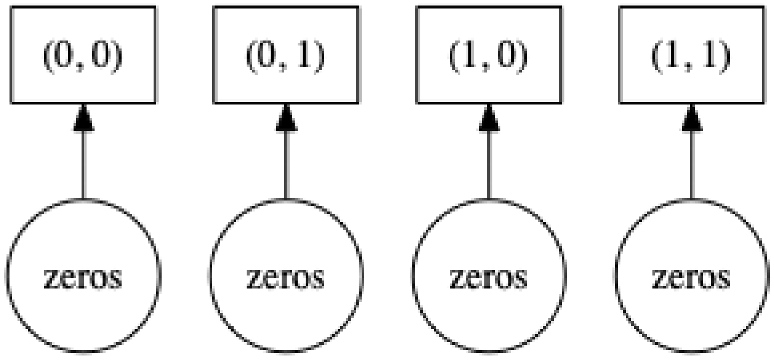
CHAPTER

FOURTEEN

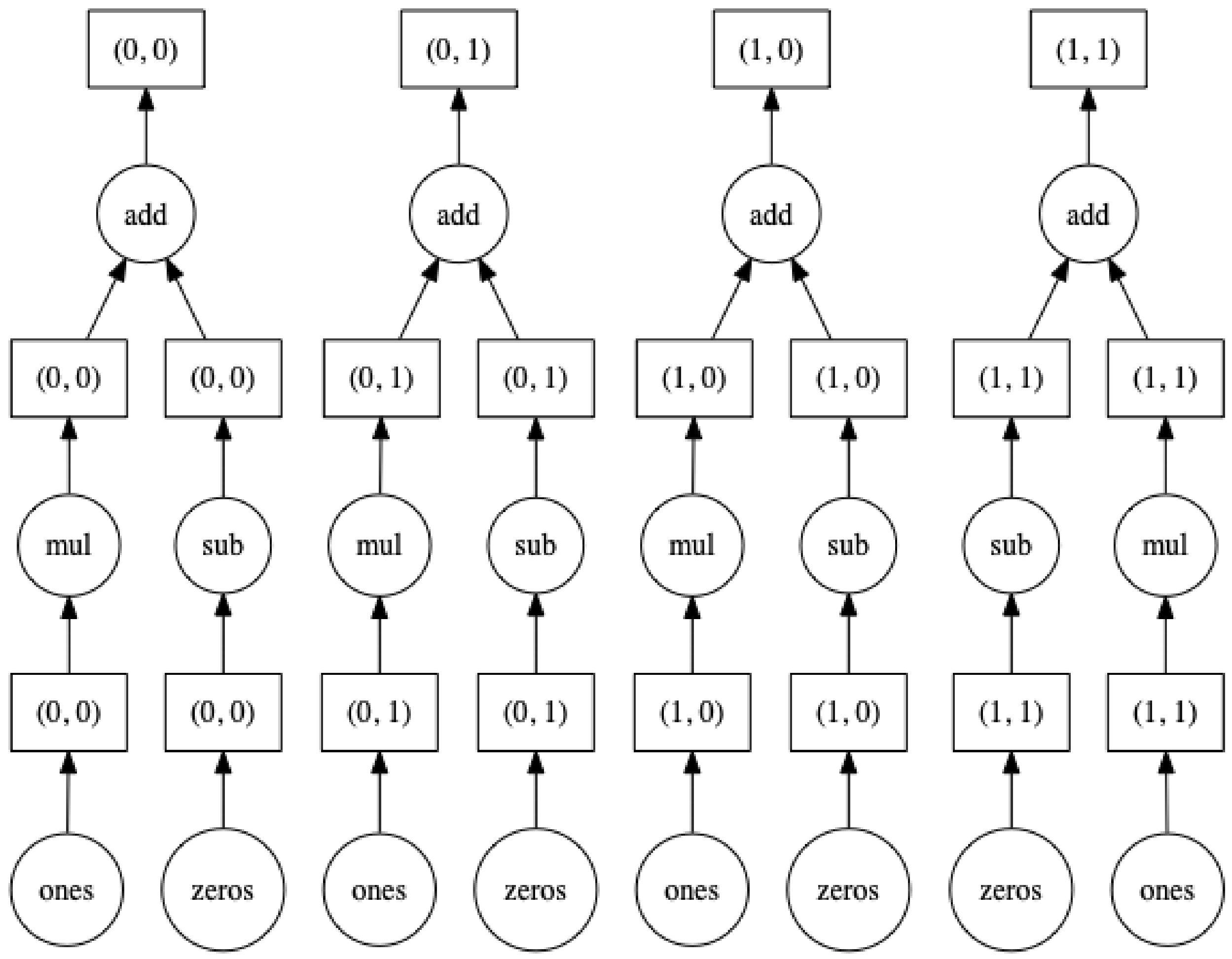
LET'S GO BIG

Now let's see where this can be really useful

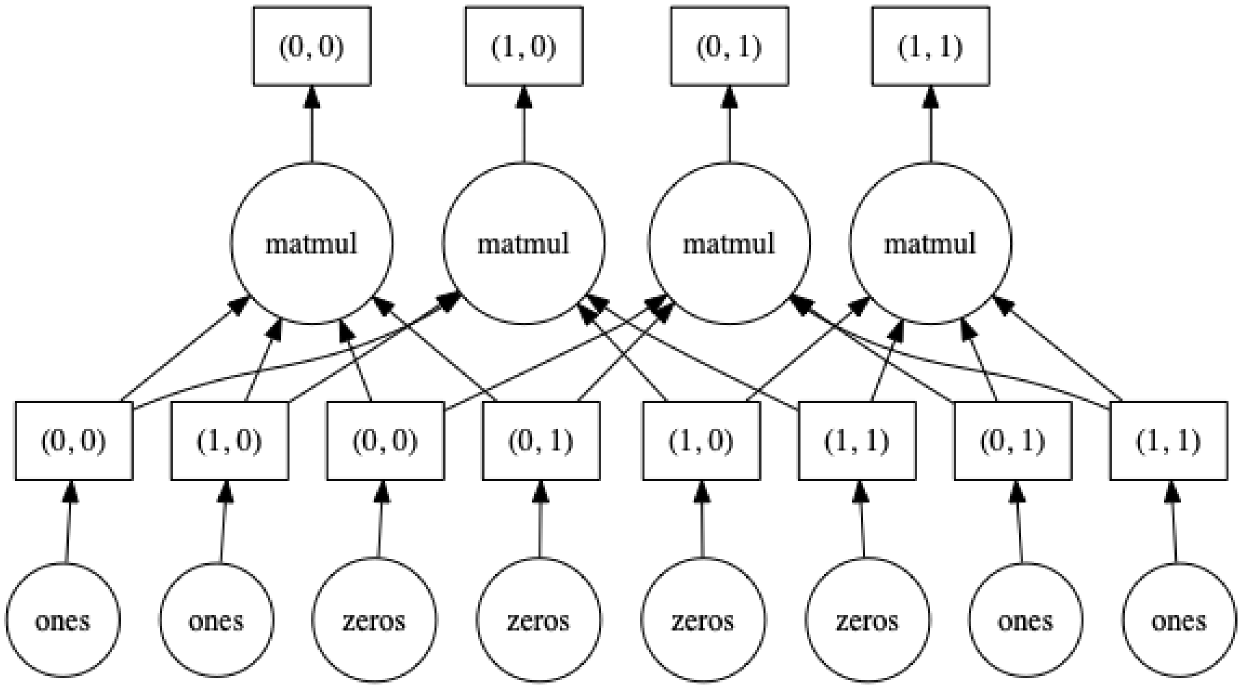
```
import dask.array as da
from dask.dot import dot_graph
image_1 = da.zeros((1024, 1024), chunks = (512, 512))
image_2 = da.ones((1024 ,1024), chunks = (512, 512))
dot_graph(image_1.dask)
```



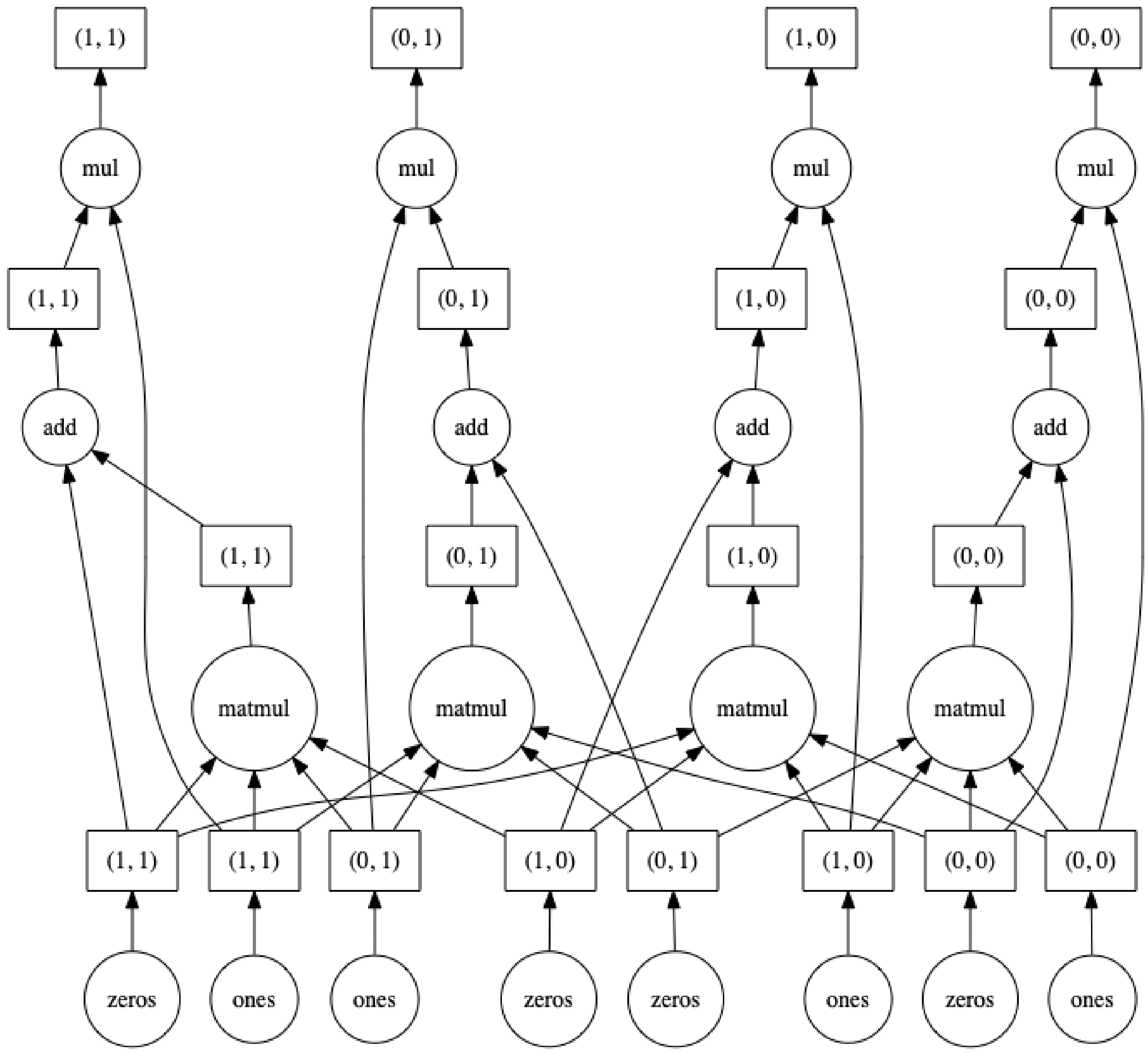
```
image_4 = (image_1-10) + (image_2*50)
dot_graph(image_4.dask)
```



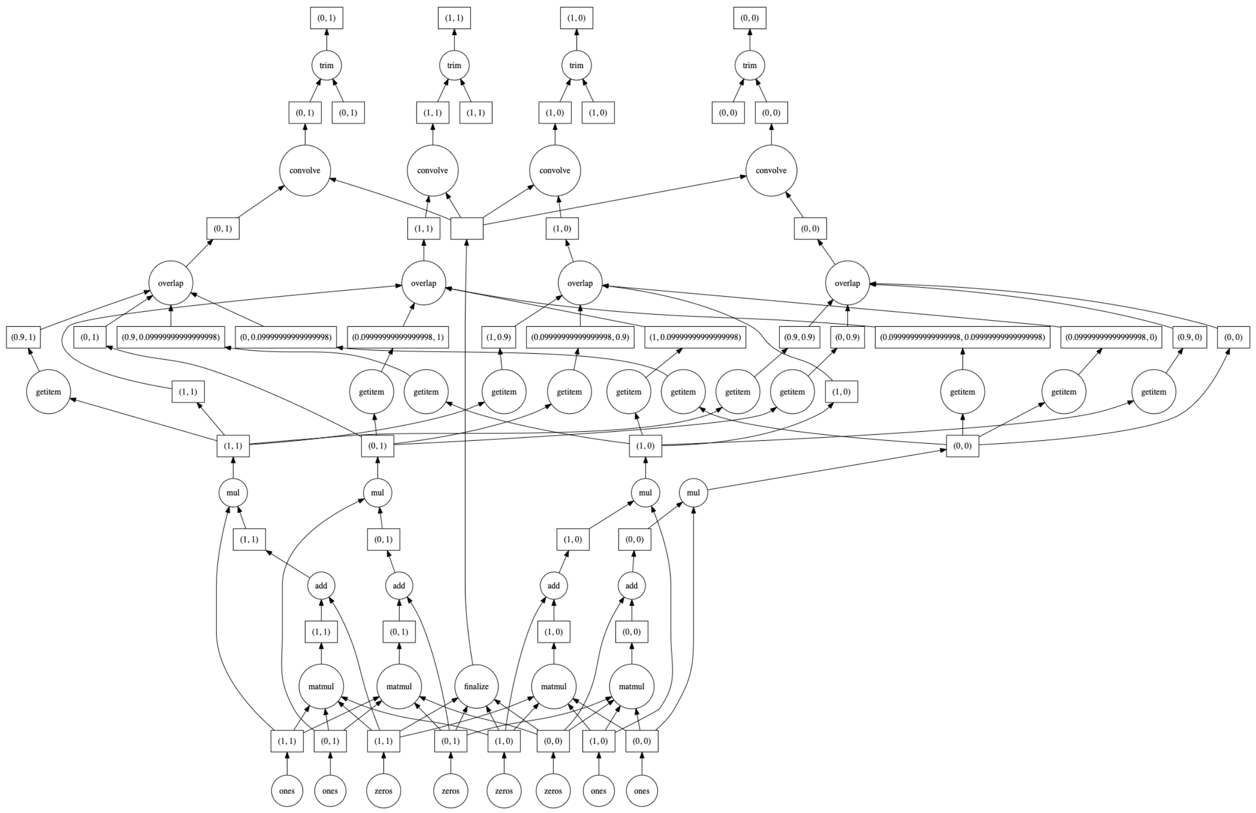
```
image_5 = da.matmul(image_1, image_2)
dot_graph(image_5.dask)
```



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



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


trim
convolve
overlap
getitem
mul
add
matmul
finalize
ones
zeros

CHAPTER

FIFTEEN

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/

```
from IPython.display import SVG
from keras.applications.resnet50 import ResNet50
from keras.utils.vis_utils import model_to_dot
resnet = ResNet50(weights = None)
SVG(model_to_dot(resnet,rankdir="LR",).create_svg())
```

```
<IPython.core.display.SVG object>
```

```
from IPython.display import clear_output, Image, display, HTML
#import keras.backend as K
import tensorflow.python.keras.backend as K
import tensorflow as tf
import numpy as np
def strip_consts(graph_def, max_const_size=32):
    """Strip large constant values from graph_def."""
    strip_def = tf.GraphDef()
    for n0 in graph_def.node:
        n = strip_def.node.add()
        n.MergeFrom(n0)
        if n.op == 'Const':
            tensor = n.attr['value'].tensor
            size = len(tensor.tensor_content)
            if size > max_const_size:
                tensor.tensor_content = ("<stripped %d bytes>"%size).encode('ascii')
    return strip_def

def show_graph(graph_def, max_const_size=32):
    """Visualize TensorFlow graph."""
    if hasattr(graph_def, 'as_graph_def'):
        graph_def = graph_def.as_graph_def()
    strip_def = strip_consts(graph_def, max_const_size=max_const_size)
    code = """
        <script>
          function load() {{
            document.getElementById("{id}").pbtxt = {data};
          }}
        </script>
        <link rel="import" href="https://tensorboard.appspot.com/tf-graph-basic.build.
↪html" onload=load()>
```

(continues on next page)

(continued from previous page)

```
        <div style="height:600px">
          <tf-graph-basic id="{id}"></tf-graph-basic>
        </div>
    """.format(data=repr(str(strip_def)), id='graph'+str(np.random.rand()))

    iframe = """
        <iframe seamless style="width:1200px;height:620px;border:0" srcdoc="{}"></
↪iframe>
    """.format(code.replace('"', '&quot;'))
    display(HTML(iframe))

sess = K.get_session()
show_graph(sess.graph)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-75-f9fa486b2250> in <module>
     40
     41 sess = K.get_session()
---> 42 show_graph(sess.graph)

<ipython-input-75-f9fa486b2250> in show_graph(graph_def, max_const_size)
     21     if hasattr(graph_def, 'as_graph_def'):
     22         graph_def = graph_def.as_graph_def()
---> 23     strip_def = strip_consts(graph_def, max_const_size=max_const_size)
     24     code = """
     25         <script>

<ipython-input-75-f9fa486b2250> in strip_consts(graph_def, max_const_size)
      6 def strip_consts(graph_def, max_const_size=32):
      7     """Strip large constant values from graph_def."""
----> 8     strip_def = tf.GraphDef()
      9     for n0 in graph_def.node:
     10         n = strip_def.node.add()

AttributeError: module 'tensorflow' has no attribute 'GraphDef'
```