


浙江大学爱丁堡大学联合学院

ZJU-UoE Institute

Lecture 1 - Introduction to image analysis

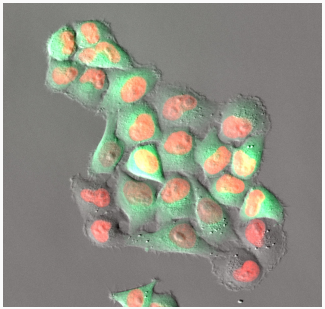
Nicola Romanò - nicola.romano@ed.ac.uk

Outline

Semester 1 only course, 20 UoE credits / 5 ZJU credits.
CO : Nicola Romanò - nicola.romano@ed.ac.uk

This course contributes 3.3% of the final ZJU GPA or 3.6% of the final GPA for international students.

This course contributes 11.1% of the final UoE degree mark used for degree classification.



Zeiss - CC-BY-2.0

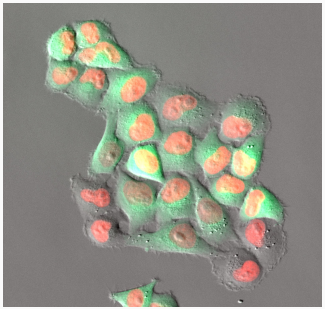
Biomedical image analysis 4

Semester 1 only course, 20 UoE credits / 5 ZJU credits.
CO : Nicola Romanò - nicola.romano@ed.ac.uk

This course contributes 3.3% of the final ZJU GPA or 3.6% of the final GPA for international students.

This course contributes 11.1% of the final UoE degree mark used for degree classification.

This course aims to give you the ability to apply and improve strategies and pipelines for biomedical image analysis.



Zeiss - CC-BY-2.0

Learning outcomes

- Describe and critically discuss strategies and algorithms for analysis of biomedical images
- Implement these strategies using Python
- Work in a team to critically address a real-life problem involving image analysis by developing and applying an appropriate pipeline
- Professionally document their analysis, presenting their pipeline and the results of their analysis, in the context of the biomedical problem it is aiming to solve.

Please see the course handbook for more information

Assessment

ICA 1 – Group work – 40% final mark

You will produce a well-documented piece of Python software to solve a specific problem in bio-imaging.

ICA 2 – Individual work – 60% final mark

You will produce an individual report highlighting the use for your software and reflecting on your group work.

Specific information on the ICAs will be given later during the course.

What will we talk about (roughly)

- *Weeks 1-5* - Basic operations on images, filters and features.
- *Weeks 6-7* - More complex operations (tracking, registration, segmentation)
- *Weeks 8-12* - Traditional and modern machine learning methods for image analysis
- *Weeks 13-14* - Discussion on recent topics in image analysis

What will we talk about (roughly)

- *Weeks 1-5* - Basic operations on images, filters and features.
- *Weeks 6-7* - More complex operations (tracking, registration, segmentation)
- *Weeks 8-12* - Traditional and modern machine learning methods for image analysis
- *Weeks 13-14* - Discussion on recent topics in image analysis

We will cover both the theory and practical aspects (with Python code!) of these topics.
You will have Python workshops to practice these problems first hand.

What will we talk about (roughly)

- *Weeks 1-5* - Basic operations on images, filters and features.
- *Weeks 6-7* - More complex operations (tracking, registration, segmentation)
- *Weeks 8-12* - Traditional and modern machine learning methods for image analysis
- *Weeks 13-14* - Discussion on recent topics in image analysis

We will cover both the theory and practical aspects (with Python code!) of these topics.
You will have Python workshops to practice these problems first hand.

Please note that you will need Python knowledge to make the most of this course.

Asking for help

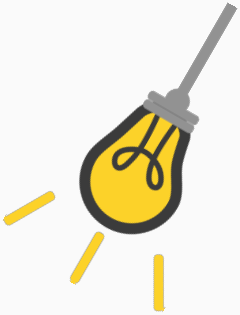
I added you to a Slack group. Please join and introduce yourself, if you have not already!

This is an invaluable opportunity to discuss anything related to the course with your peers and myself.

Introductory material about Python is available on Learn and further material can be provided, if needed - plase ask!

Learning objectives

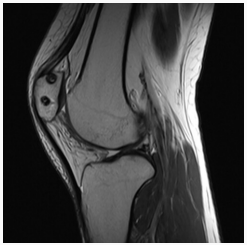
- Give examples of problems in image analysis
- Describe common Python libraries used in image analysis
- Describe images as matrices or tensors
- Display images using Python



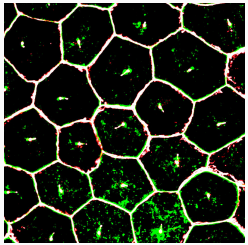
Outline

Examples of biomedical images and associated problems

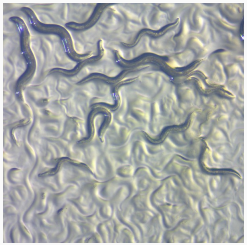
Image processing is a major task in biomedical sciences (and not only). Problems in image analysis include detection of objects; quantification of their properties; improvement of image quality; classification of images…



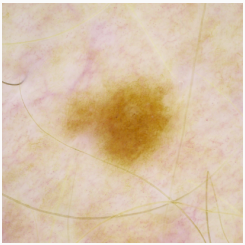
Becky Stern - CC-BY-SA 2.0



NIH - CC-BY-SA 2.0

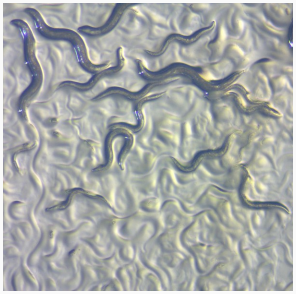


Zeiss - CC-BY 2.0



ISIC melanoma competition - CC-0

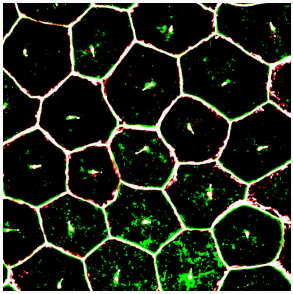
Examples of image analysis problems



Zeiss - CC-BY 2.0

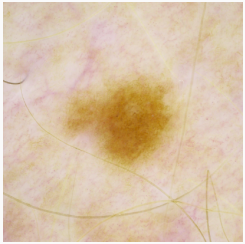
How many worms are in this image?
Where are they located?
Where are they moving to (assuming you have a video!)?

Examples of image analysis problems



Where are the cells located?
How intense is the green staining in each cell?
How uniformly is the staining distributed?

Examples of image analysis problems



ISIC melanoma competition - CC-0

Is this a melanoma?

In this course...

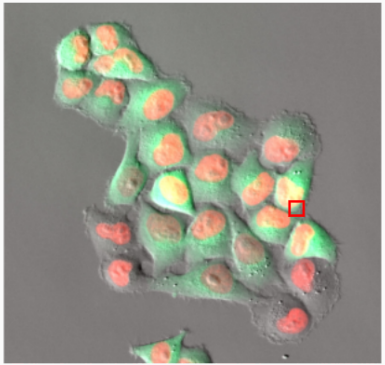
We will learn strategies to analyse images and answer this types of questions.

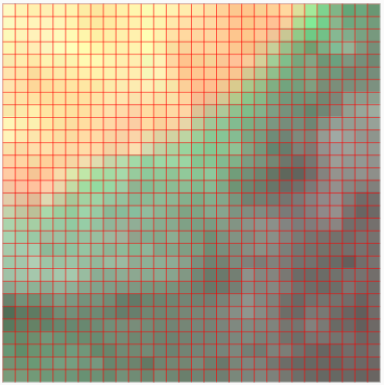
Let's start with something simple!

Outline

Images as rasters

When analysing digital images we are concerned with **rasters**.
A raster is a series of **pixels** arranged in a rectangular grid, which represent an image. Each pixel is associated with a value or a series of values, representing its intensity and colour.

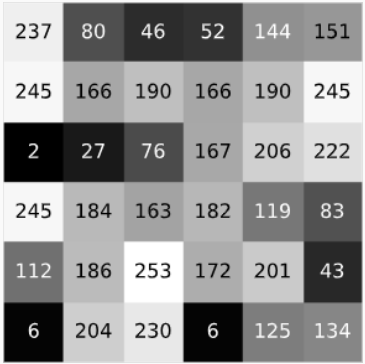




Images as matrices (or tensors)

Because images are grids of values, we can represent them as matrices.
The dimensions of the matrix will be (width, height). For example, if 0 is black and 255 is white (more on this later)

$$\begin{bmatrix} 237 & 80 & 46 & 52 & 144 & 151 \\ 245 & 166 & 190 & 166 & 190 & 245 \\ 2 & 27 & 76 & 167 & 206 & 222 \\ 245 & 184 & 163 & 182 & 119 & 83 \\ 112 & 186 & 253 & 172 & 201 & 43 \\ 6 & 204 & 230 & 6 & 125 & 134 \end{bmatrix}$$

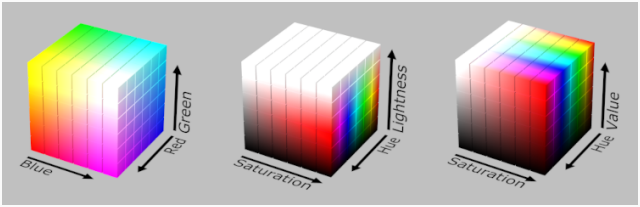


What about colour?

Colour images are generally stored as RGB (red, green, blue).
The color of each pixel depends on its intensity in each of these three **colour channels**.

What about colour?

Colour images are generally stored as RGB (red, green, blue).
The color of each pixel depends on its intensity in each of these three **colour channels**.

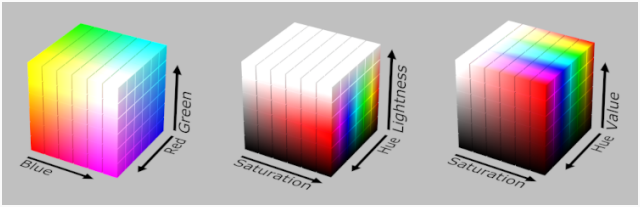


Adapted from Wikipedia

RGB is not the only way to represent colour. If you are interested in knowing more about **colour spaces** see the attached review by Hastings and Rubins, 2012.

What about colour?

Colour images are generally stored as RGB (red, green, blue).
The color of each pixel depends on its intensity in each of these three **colour channels**.

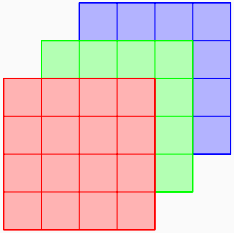


Adapted from Wikipedia

RGB is not the only way to represent colour. If you are interested in knowing more about **colour spaces** see the attached review by Hastings and Rubins, 2012.

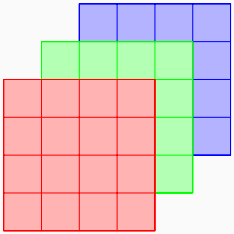
Important: microscope images are often composed of multiple grayscale images, e.g. of different fluorophores. In this case, each pixel has multiple colour channels.

Storing colour images

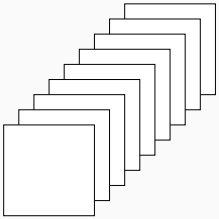


Colour images are stored as a 3-dimensional matrix, also called a **tensor** with dimensions (number of colour channels, width, height).

Storing colour images



Colour images are stored as a 3-dimensional matrix, also called a **tensor** with dimensions (number of colour channels, width, height).

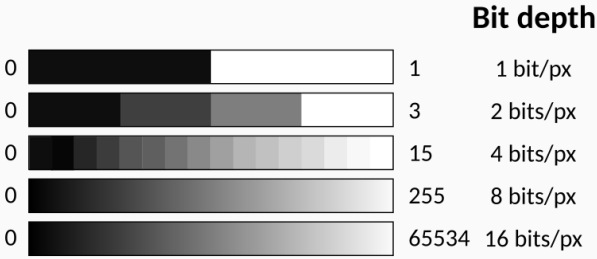


Multi-dimensional matrices can be used to represent videos (number of frames, width, height) or stacks of images, e.g. in confocal images, with dimensions (number of planes, width, height).

Note that the order of the dimensions might vary.

Bit depth

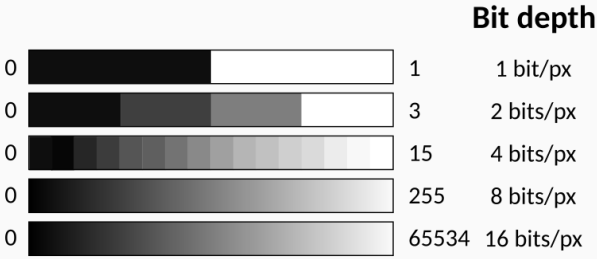
Bit depth is the number of bits necessary to represent each pixel in the image.



Increasing bit depth > Increasing information

Bit depth

Bit depth is the number of bits necessary to represent each pixel in the image.



Increasing bit depth > Increasing information

You can also represent images with floating-point numbers. In that case, you have infinite levels between [0.0, 1.0] or [-1.0, 1.0].

The Numpy library simplifies working with matrices in Python.

NumPy is a Python open-source library for numerical computing, with support for n-dimensional arrays and many functionalities to work with matrices.

Numpy can be installed via `pip`

```
pip install numpy
```

Our first NumPy matrix!

```
# This is the standard way of importing numpy
import numpy as np
a = np.array([1, 2, 3])
```

Multiple dimensions

Numpy supports n-dimensional arrays with any number of dimensions.

```
import numpy as np
b = np.array([[1, 2, 3][4, 5, 6]])
print(b.shape)
```

```
(2, 3)
```

The shape of an array is a vector showing the number of element in each of the dimensions of the array. In this case, we have a matrix of 2 rows by 3 columns.

Matrix operations in Python

The Numpy library allows easy manipulation of matrices (and thus images).

```
import numpy as np
# We can define a matrix
a = np.array([[10, 15, 20], [20, 25, 30], [120, 5, 20]])
# Perform basic operations on it
b = a + 10
c = a * 2
```

Matrix operations in Python

The Numpy library allows easy manipulation of matrices (and thus images).

```
import numpy as np
# We can define a matrix
a = np.array([[10, 15, 20], [20, 25, 30], [120, 5, 20]])
# Perform basic operations on it
b = a + 10
c = a * 2
```

$$a = \begin{bmatrix} 10 & 15 & 20 \\ 20 & 25 & 30 \\ 120 & 5 & 20 \end{bmatrix} \quad b = \begin{bmatrix} 20 & 25 & 30 \\ 30 & 35 & 40 \\ 130 & 15 & 30 \end{bmatrix} \quad c = \begin{bmatrix} 20 & 30 & 40 \\ 40 & 50 & 60 \\ 240 & 10 & 40 \end{bmatrix}$$

Matrix operations in Python

The Numpy library allows easy manipulation of matrices (and thus images).

```
import numpy as np
# We can define a matrix
a = np.array([[10, 15, 20], [20, 25, 30], [120, 5, 20]])
# Perform basic operations on it
b = a + 10
c = a * 2
```

$$a = \begin{bmatrix} 10 & 15 & 20 \\ 20 & 25 & 30 \\ 120 & 5 & 20 \end{bmatrix} \quad b = \begin{bmatrix} 20 & 25 & 30 \\ 30 & 35 & 40 \\ 130 & 15 & 30 \end{bmatrix} \quad c = \begin{bmatrix} 20 & 30 & 40 \\ 40 & 50 & 60 \\ 240 & 10 & 40 \end{bmatrix}$$

More information can be found on the NumPy paper (Harris et al., Nature 2020) or on the NumPy website (www.numpy.org)

Matplotlib and Scikit Image

Two very useful libraries to draw and manipulate images are Matplotlib and Scikit Image.

Install using

```
pip install matplotlib
pip install scikit-image
```

Matplotlib and Scikit Image

Two very useful libraries to draw and manipulate images are Matplotlib and Scikit Image.

Install using

```
pip install matplotlib
pip install scikit-image
```

Matplotlib - very powerful, general-purpose plotting library. You will generally using by importing the pyplot submodule.

```
import matplotlib.pyplot as plt
```

Two very useful libraries to draw and manipulate images are Matplotlib and Scikit Image.

Install using

```
pip install matplotlib
pip install scikit-image
```

Matplotlib - very powerful, general-purpose plotting library. You will generally using by importing the pyplot submodule.

```
import matplotlib.pyplot as plt
```

Scikit Image - built on top of Matplotlib and is specialized for image manipulation and analysis. Contains many submodules to perform specific operations.

Loading images - Matplotlib vs Scikit Image


```
import matplotlib.pyplot as plt
img = plt.imread("cells.jpg")
plt.imshow(img)
plt.show()
```

Loading images - Matplotlib vs Scikit Image

matplotlib

```
import matplotlib.pyplot as plt
img = plt.imread("cells.jpg")
plt.imshow(img)
plt.show()
```


```
from skimage import io
img = io.imread("cells.jpg")
io.imshow(img)
io.show()
```

Loading images - Matplotlib vs Scikit Image

matplotlib

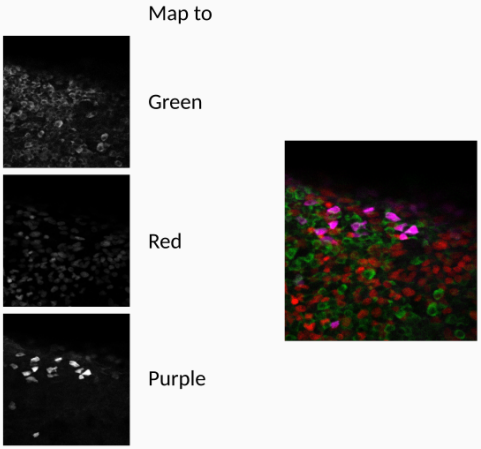
```
import matplotlib.pyplot as plt
img = plt.imread("cells.jpg")
plt.imshow(img)
plt.show()
```


```
from skimage import io
img = io.imread("cells.jpg")
io.imshow(img)
io.show()
```

These are (almost) equivalent and will output `img` as a Numpy array.

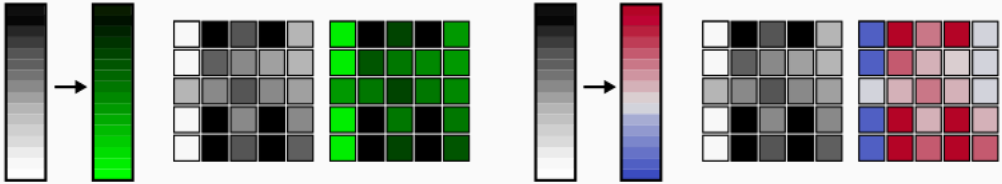
Pseudocoloring

Often colour is applied to microscopic images **after** acquisition to mark different parts of the image. This is called **pseudocolouring** or **false colouring**.



Look-up tables (LUTs)

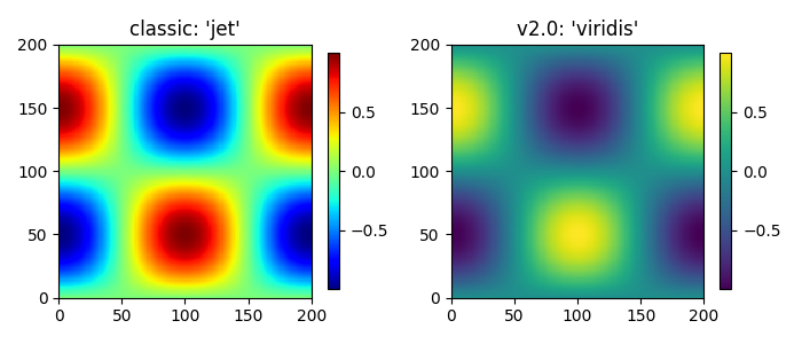
Pseudocoloring is obtained through look-up tables (sometimes referred to as "palettes" or "colourmaps")



Colourmaps can be sequential/linear or non sequential.
Divergent and non-linear colour maps can be good to enhance differences in the image, but should be used with caution!

The Jet palette - when colourmaps go bad...

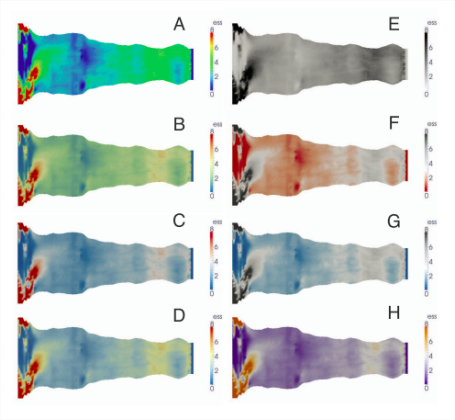
The Jet colourmap has for a long time been the standard palette in Matlab, is used in many pieces of software and is seen in many published articles. Jet is problematic because it can create artefacts in your data.



Jet can create artefacts in the data. [Source: Matplotlib website] See https://www.youtube.com/watch?v=xAoljeRJ3lU

The Jet palette - when colourmaps go bad...

The Jet colourmap has for a long time been the standard palette in Matlab, is used in many pieces of software and is seen in many published articles. Jet is problematic because it can create artefacts in your data.



Rainbow palette resulted in significantly more diagnostic errors - Borkin 2011

Colourmapping in Matplotlib

We can easily colourmap an image loaded in Matplotlib

```
import matplotlib.pyplot as plt
img = imread("cells.tif")
# plt.subplots allows plotting more than one image
# side by side.  It creates and returns a figure and
# a list of axes (the subplots)
fig, ax = plt.subplots(ncols=3, nrows=1, figsize=(10, 6))
```

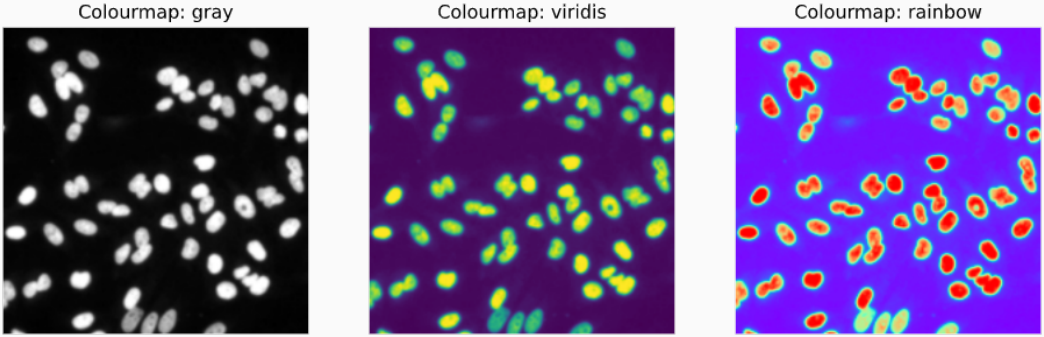
Colourmapping in Matplotlib

We can easily colourmap an image loaded in Matplotlib

```
import matplotlib.pyplot as plt
img = imread("cells.tif")
# plt.subplots allows plotting more than one image
# side by side.  It creates and returns a figure and
# a list of axes (the subplots)
fig, ax = plt.subplots(ncols=3, nrows=1, figsize=(10, 6))
ax[0].imshow(img, cmap="gray")
ax[1].imshow(img, cmap="viridis")
ax[2].imshow(img, cmap="rainbow")
plt.show()
```

Colourmapping in Matplotlib

We can easily colourmap an image loaded in Matplotlib



Check out the Matplotlib website for a list of colourmaps!
`https://matplotlib.org/stable/tutorials/colors/colormaps.html`

Image intensity vs visualized intensity

`imshow` has two parameters called `vmin` and `vmax` to control the **i**ntensity range of the image **on the screen**.

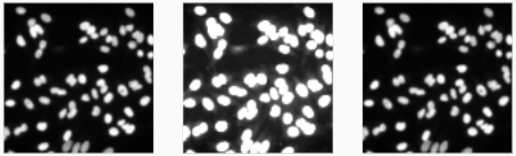
For example, if you set `vmin=0` and `vmax=255`, then the image will be shown with 0 being black and 255 being white. If you set `vmax=100`, then any pixel with an intensity higher than 100 will be shown as white.

Image intensity vs visualized intensity

`imshow` has two parameters called `vmin` and `vmax` to control the **i**ntensity range of the image **on the screen**.

For example, if you set `vmin=0` and `vmax=255`, then the image will be shown with 0 being black and 255 being white. If you set `vmax=100`, then any pixel with an intensity higher than 100 will be shown as white.

```
plt.imshow(img, vmin=0, vmax=255)
plt.imshow(img, vmin=0, vmax=50)
# Automatically uses min and max of the
image
plt.imshow(img)
```



IMPORTANT: the actual pixel intensity of these images are all the same we are just modifying how they are displayed on the screen.

Practice time!

And now some practice on what we saw today!