Lecture 11: Image processing and analysis

Nils Olovsson
nils.olofsson@igp.uu.se

Introduction to data science and Python programming for medical research
Uppsala University

2025

UPPSALA
UNIVERSITET

Content

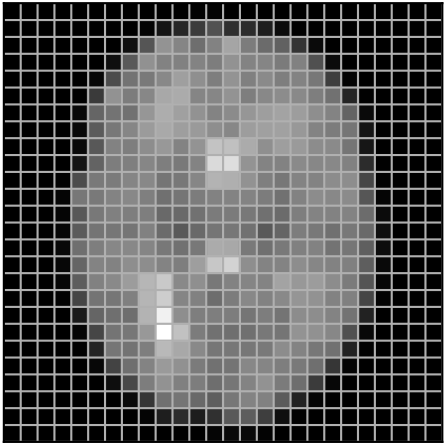
Introduction

Introduction

Introduction: What is an image?

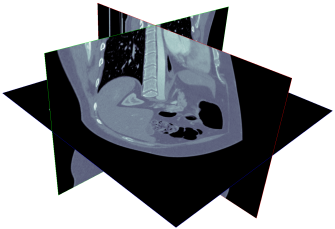
Data sampled on a regular grid (*lattice*).

Some kind of spatial data.
Sometimes in a time sequence.

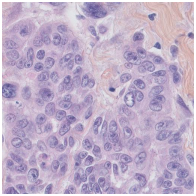


Introduction: Image dimensions, channels, sequence, ...

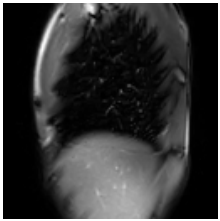
Dimension (2D, 3D)

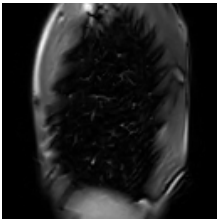


Channels (colors)

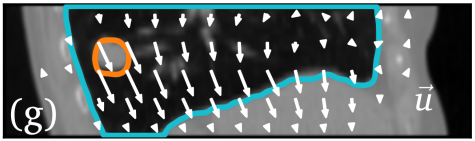


Sequence





Vector valued



Introduction: Biomedical images

Dimension Typically 2D or 3D.

Channels Can often be processed separately or combined into one gray scale image.

Sequence Typically not. (But we will look at some examples)

Vector valued Also not typical. Can be thought of as channels.

We will limit our attention to single or pairs of 2D, scalar valued, images. Represented by a 2D array of values.

Biomedical images

Biomedical images

Biomedical images: Sources

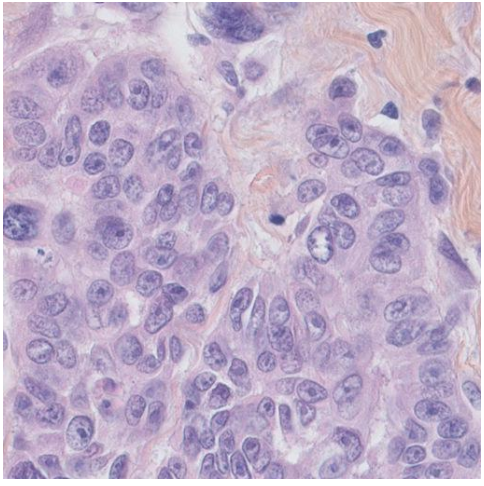
Data sampled on a regular grid (*lattice*).

Some kind of spatial data.
Sometimes in a time sequence.

Biomedical images: Microscopy

Histology images.

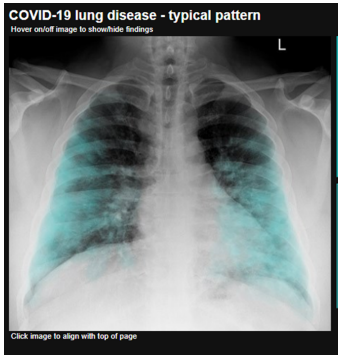
Slices of tissue samples stained using *e.g.* antibodies to differentiate cell type or activity.



Biomedical images: X-ray

Projection x-ray images.

Taken very routinely.
E.g. at the dentist, mammography, suspected fractures etc



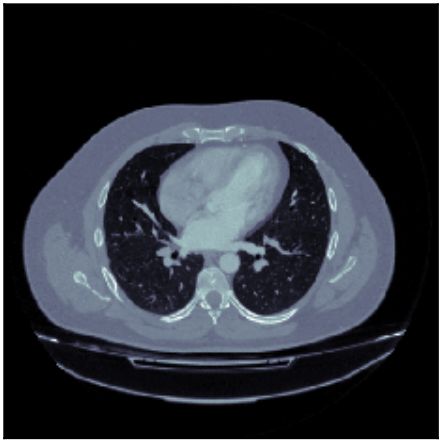
Biomedical images: Medical scans (1/2)

- **Computed tomography (CT)**
- **Magnetic resonance imaging (MRI)**
- **Positron emission tomography (PET)**
- **Gamma camera**
- **Ultra sound**

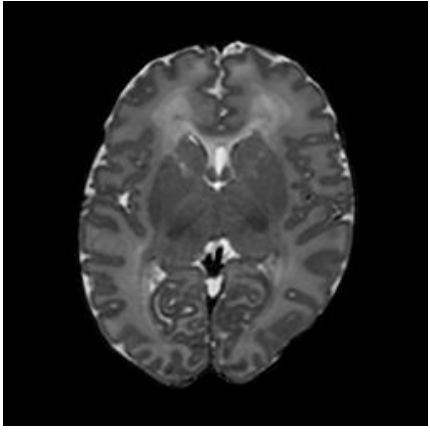
Can be 2D or 3D.

Sequence of 3D usually called 4D or dynamic imaging depending on context.

Computed tomography (CT)

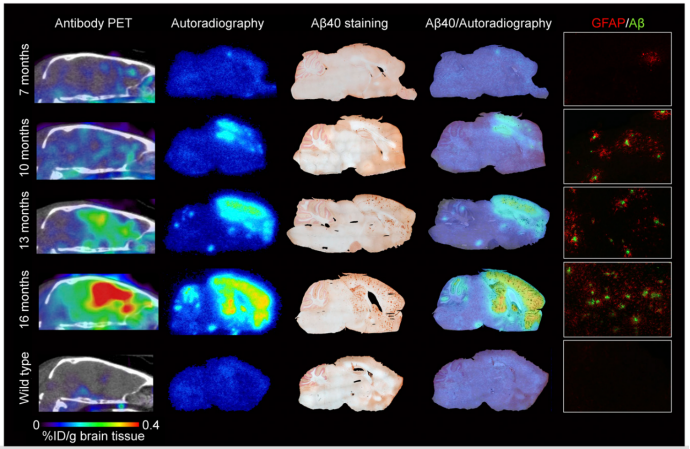


Biomedical images: Medical scans (2/2)



Biomedical images: Autoradiography

Tissue slices with varying uptake of some radioactive compound.



Antibody-Based In Vivo PET Imaging Detects Amyloid-β Reduction in Alzheimer Transgenic Mice After BACE-1 Inhibition
doi:10.2967/jnumed.118.213140

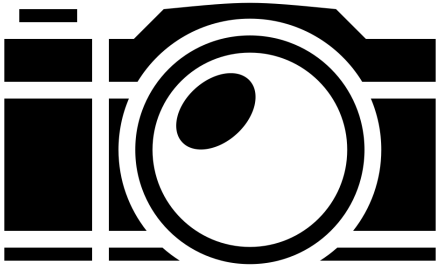
Biomedical images: Photography

Clinical setting

Common for medical doctors to take photographs of patients that are stored on their medical record.

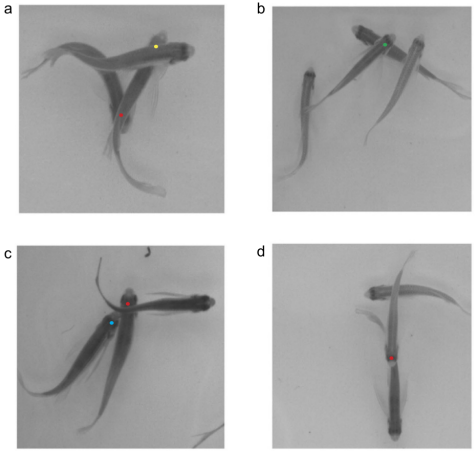
Pre-clinical experimental setting

Photographs of cell cultures, 3D cultures or in animal studies.



Biomedical images: Photography (preclinical)

Experimental settings can be photographed or video can be captured.



Zebrafish tracking using convolutional neural networks
doi:10.1038/srep42815

Biomedical images: Simulations

Computations are often performed or sampled on regular grids that are treated as or used with images.

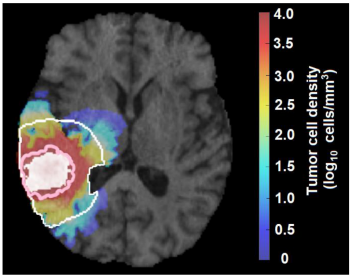
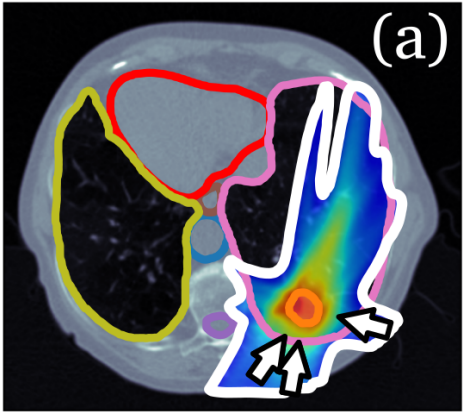


Figure 4: Example of a simulated tumor invasion (colorwash) of normal tissue for a delineated GTV (white mass). The pink contour corresponds to the simulated GTV, and the white contour to the conventionally delineated CTV. There is disagreement with the CTV and the simulated tumor invasion in terms of shape and reach.

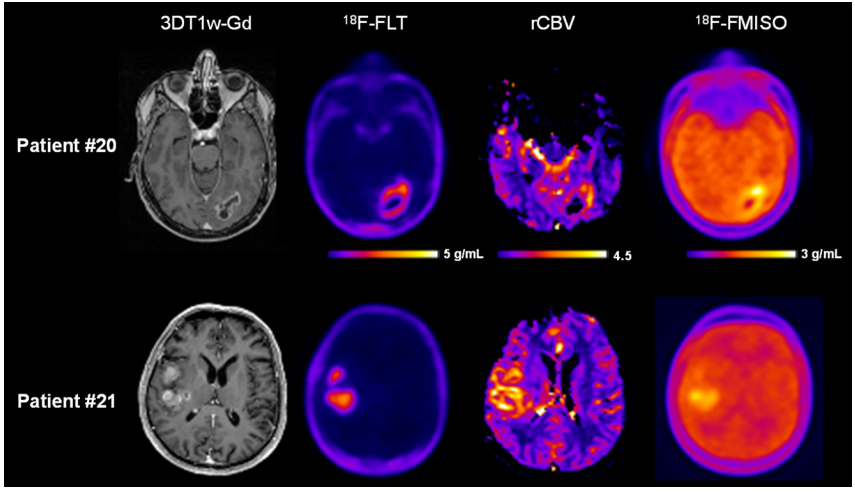


Biomedical images: Quantitative or not?

Values of pixels in an image can represent values that are relative within the context of only that image.

E.g. photography and typical x-ray, ultrasound and MRI examinations.

Biomedical images: Anatomical and functional imaging



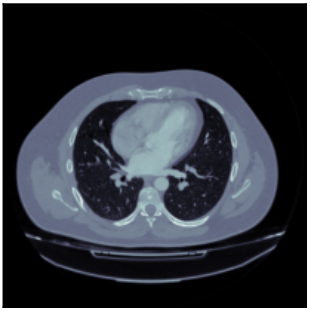
Biomedical images: Processing and analysis

- Visualize and highlight features.
- Segmentation. Identify objects such as cells, organs or a tumor.
- Registration. Align images.
- Quantify.
- Predict. *E.g.* disease or progression.

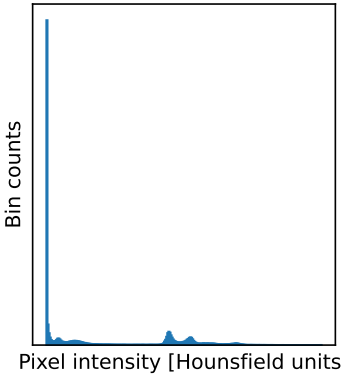
Brightness transforms

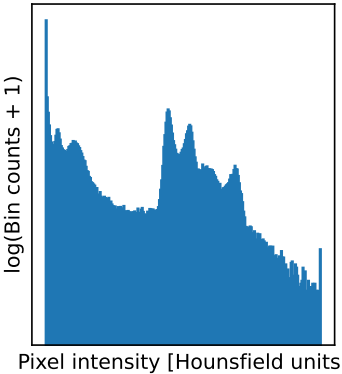
Brightness transforms: Image histogram

Describes distribution of intensity values in the image.



Image





Brightness transforms: Linear transforms and windowing

Add/Subtract

Multiply

Both

Windowing

Common for visualization.
Also known as gray-level mapping, contrast stretching, histogram modification or contrast enhancement.

Brightness transforms: Histogram equalization

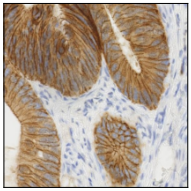
Redistribute intensity values such that they become more equally spread out.
Purpose is to enhance the image contrast.

```
import numpy as np
from skimage.exposure import equalize_hist
import matplotlib.pyplot as plt
# Use the pydicom package to load DICOM image
import pydicom
filename = 'image_160.dcm'
directory = os.path.join('..', '..',
                         'data', '3Dircadb2_1')
filepath = os.path.join(directory, filename)
dcm = pydicom.dcmread(filepath)
img = dcm.pixel_array
img_eq = equalize_hist(img)
# Plot original and smoothed image
fig, axs = plt.subplots(2, 1, figsize=(3, 6))
axs[0].imshow(img, cmap='gray')
axs[1].imshow(img_eq, cmap='gray')
labels = ['Original image', 'Histogram equalized']
for i, ax in enumerate(axs):
    ax.text(0.01, 0.01, labels[i], va='bottom',
            ha='left', color='white',
            transform=ax.transAxes)
```

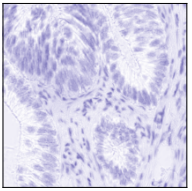
Brightness transforms: Color channel separation

```python
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from skimage.color import rgb2hed, hed2rgb
# Example IHC image, separate stains from IHC image
ihc_rgb = data.immunohistochemistry()
ihc_hed = rgb2hed(ihc_rgb)
# Create an RGB image for each of the stains
null = np.zeros_like(ihc_hed[:, :, 0])
ihc_h = hed2rgb(np.stack((ihc_hed[:, :, 0], null, null),
                         axis=-1))
# Note that there is no eosin stain in this image
#ihc_e = hed2rgb(np.stack((null, ihc_hed[:, :, 1], null),
#                          axis=-1))
ihc_d = hed2rgb(np.stack((null, null, ihc_hed[:, :, 2]),
                         axis=-1))
# Display
fig, axes = plt.subplots(2, 2, figsize=(7, 6),
                         sharex=True, sharey=True)
axs = axes.ravel()
axs[0].imshow(ihc_rgb)
axs[0].set_title("Original image")
axs[1].imshow(ihc_h)
axs[1].set_title("Hematoxylin")
axs[3].imshow(ihc_d)
axs[3].set_title("DAB")
#plt.show()
```

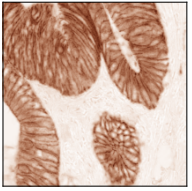
Original image



Hematoxylin



DAB



https://scikit-image.org/docs/stable/auto_examples/color_exposure/plot_ihc_color_separation.html
https://scikit-image.org/docs/stable/api/skimage.color.html#skimage.color.rgb2hed

Convolution filters

Convolution filters: Image smoothing

A very common operation in image processing is to blur or smooth the image.

Mean filter

We could do this by setting the value of each pixel to the mean value of itself and all its neighbors.

Convolution filters: Kernel

This operation can be expressed using a special operation called *convolution* (sv. *faltning*), $*$,

$$I_{\text{Filtered}} = k * I_{\text{Original}}$$

where k is a *kernel*.

The kernel represents the weights of the pixels in the neighborhood.
The new pixel value is computed as a weighted sum of this kernel with the pixels in the neighborhood.

For the mean filter the 3×3 kernel is defined as

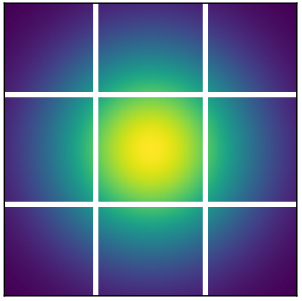
$$k = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Convolution filters: Python example

```
1  import numpy as np
2  from scipy import signal
3  import matplotlib.pyplot as plt
4  # Use the pydicom package to load DICOM image
5  import pydicom
6  filename = 'Mag (0005)'
7  directory = os.path.join('..', '..',
8                           'data', '00-additional',
9                           'cine-MR-thorax-sagittal')
10 filepath = os.path.join(directory, filename)
11 dcm = pydicom.dcmread(filepath)
12 img = dcm.pixel_array
13 # Create and apply mean filter kernel
14 k = (1.0 / 9.0) * np.array([[1, 1, 1],
15                             [1, 1, 1],
16                             [1, 1, 1]], dtype=np.float32)
17 smi = signal.convolve2d(img, k, mode='same', boundary='symm')
18 # Plot original and smoothed image
19 fig, axs = plt.subplots(2, 1, figsize=(3, 6))
20 axs[0].imshow(img, cmap='gray')
21 axs[1].imshow(smi, cmap='gray')
22 labels = ['Original image', 'Mean filtered image']
23 for i, ax in enumerate(axs):
24     ax.text(0.01, 0.01, labels[i], va='bottom',
25             ha='left', color='white',
26             transform=ax.transAxes)
```

Convolution filters: Gaussian filter

A more common smoothing operation than the mean filter is Gaussian filtering.
This is better from a digital signal processing point of view.
It is also intuitive that values further from a pixel should influence its new value less than pixels that are close.
The kernel is defined by a discrete 2D Gaussian distribution.



$\rightarrow$

$$k = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Convolution filters: Gaussian filter with Python

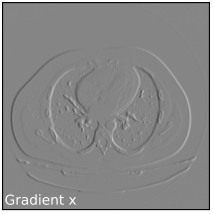
```python
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
# Use the pydicom package to load DICOM image
import pydicom
filename = 'Mag (0005)'
directory = os.path.join('..', '..',
                         'data', '00-additional',
                         'cine-MR-thorax-sagittal')
filepath = os.path.join(directory, filename)
dcm = pydicom.dcmread(filepath)
img = dcm.pixel_array
# Create and apply Gaussian filter kernel
k = (1 / 16) * np.array([[1, 2, 1],
                         [2, 4, 2],
                         [1, 2, 1]])
smi = signal.convolve2d(img, k, mode='same', boundary='symm')
# Plot original and smoothed image
fig, axs = plt.subplots(2, 1, figsize=(3, 6))
axs[0].imshow(img, cmap='gray')
axs[1].imshow(smi, cmap='gray')
labels = ['Original image', 'Gaussian filtered image']
for i, ax in enumerate(axs):
    ax.text(0.01, 0.01, labels[i], va='bottom',
            ha='left', color='white',
            transform=ax.transAxes)
```

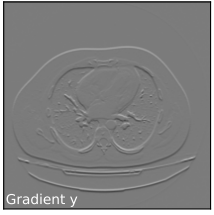
Convolution filters: Other examples

∇_x (Gradient in x): $k = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

∇_y (Gradient in y): $k = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

∇^2 (Laplacian): $k = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$





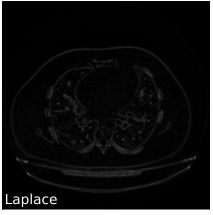


Image segmentation

Image segmentation

Image segmentation: Detect and label

Label all pixels that belong to a certain object in an image. *E.g.* cells, organs, lesions etc.

The problem of separating an object from the background is a form of classification with some similarities to clustering.

Discuss a few classical methods:

- Intensity threshold
- Active contours
- Graph cuts

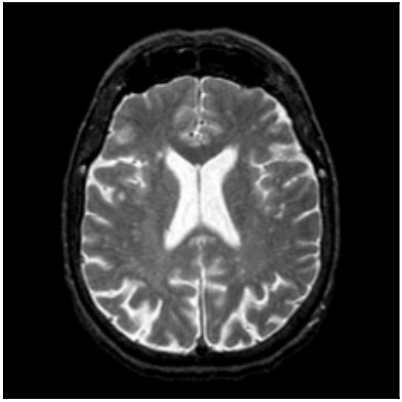


Image segmentation: Detect and label

Label all pixels that belong to a certain object in an image. *E.g.* cells, organs, lesions etc.

The problem of separating an object from the background is a form of classification with some similarities to clustering.

Discuss a few classical methods:

- Intensity threshold
- Active contours
- Graph cuts

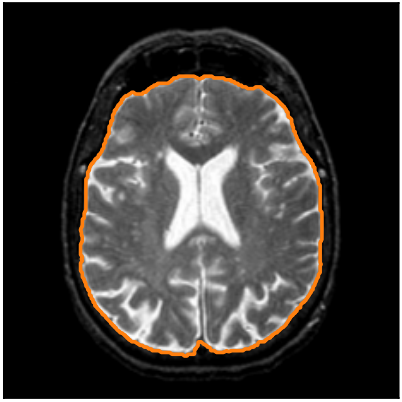
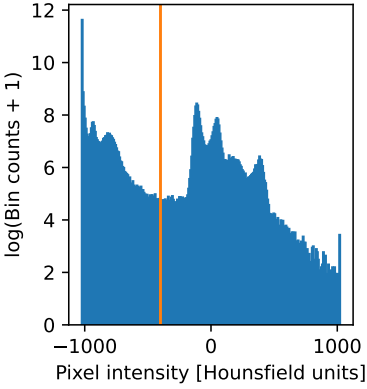


Image segmentation: Manual threshold

Label all pixels above a certain image value threshold as belonging to the object and all below as background.

Simple and limited. May require some knowledge of the imaging modality/method.



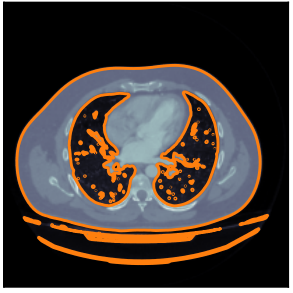
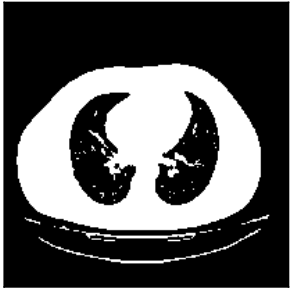


Image with segmentation contours



Segmentation mask

Image segmentation: Automatic threshold

Can find an **optimal threshold** to partition the histogram with **Otsu's method** such that inter-class variance is minimized.

Simple and also limited method.

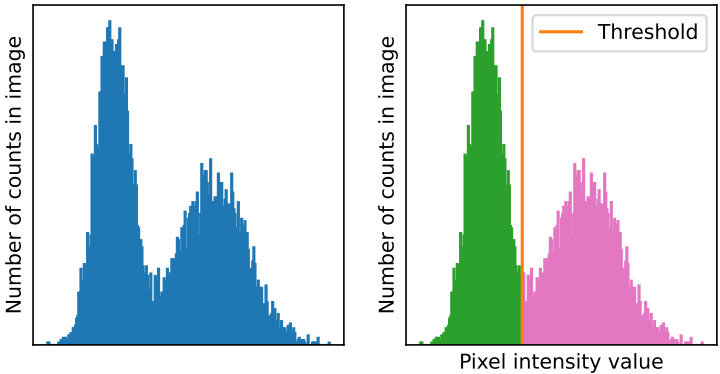


Image segmentation: Otsu threshold with Python

```python
import numpy as np
import matplotlib.pyplot as plt
import pydicom
from skimage.filters import threshold_otsu

# Load sagittal lung cine MR image
filename = 'Mag (0005)'
directory = os.path.join('..', '..',
                         'data',
                         '00-additional',
                         'cine-MR-thorax-sagittal')
filepath = os.path.join(directory, filename)
dcm = pydicom.dcmread(filepath)
img = dcm.pixel_array

# Threshold using Otsu's method
v = threshold_otsu(img)
msk = img > v

# Plot image and contour mask
ax = plt.figure(figsize=(3,3)).gca()
ax.imshow(img, zorder=0, cmap='gray')
ax.contour(msk, levels=[0.5], zorder=1,
           colors=['tab:orange'])
```

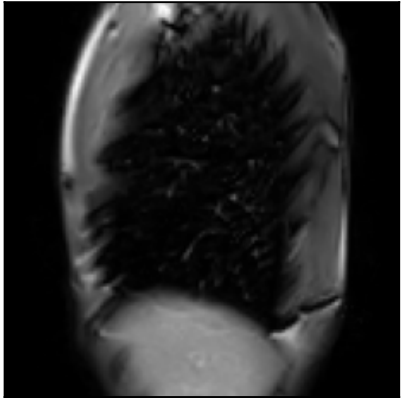


Image segmentation: Otsu threshold with Python

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import pydicom
4  from skimage.filters import threshold_otsu
5
6  # Load sagittal lung cine MR image
7  filename = 'Mag (0005)'
8  directory = os.path.join('..', '..',
9                           'data',
10                          '00-additional',
11                          'cine-MR-thorax-sagittal')
12 filepath = os.path.join(directory, filename)
13 dcm = pydicom.dcmread(filepath)
14 img = dcm.pixel_array
15
16 # Threshold using Otsu's method
17 v = threshold_otsu(img)
18 msk = img > v
19
20 # Plot image and contour mask
21 ax = plt.figure(figsize=(3,3)).gca()
22 ax.imshow(img, zorder=0, cmap='gray')
23 ax.contour(msk, levels=[0.5], zorder=1,
24            colors=['tab:orange'])
```

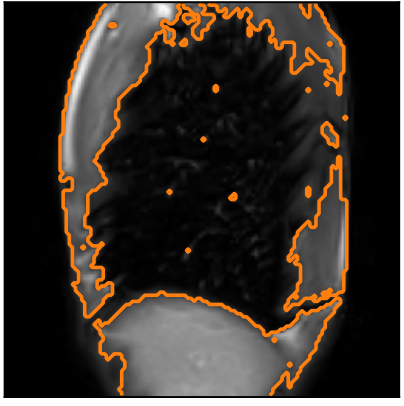
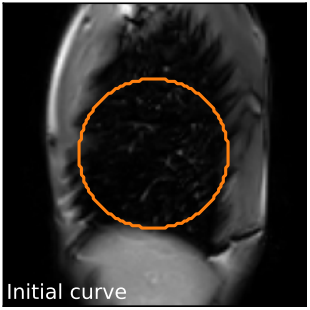
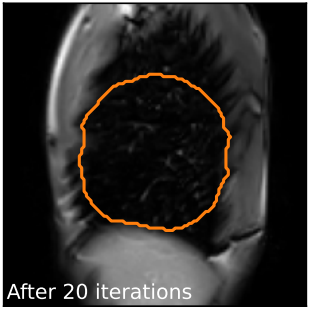


Image segmentation: Active contour / level set

The segmentation problem is formulated as an evolving curve that is attracted to edges in the image.

Modeled as a differential equation that is stepped forward, iterated, in time.





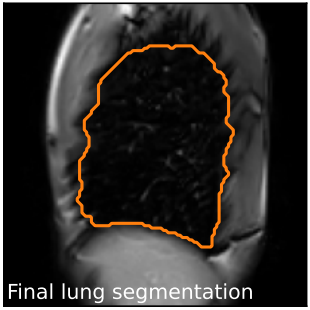


Image segmentation: Active contour in Python

```python
import numpy as np
import matplotlib.pyplot as plt
import skimage.io
from skimage.segmentation import chan_vese
# Load MR brain image
directory = os.path.join('..', '..',
                         'data',
                         '00-additional',
                         'cortex')
filename = 'cortex.bmp'
filepath = os.path.join(directory, filename)
img = skimage.io.imread(filepath)
img = img[:,:,0]
# Segment using active contour
msk = chan_vese(img, mu=0.13,
                lambda1=1, lambda2=1, tol=1e-3,
                max_num_iter=200, dt=0.5,
                init_level_set="checkerboard",
                #init_level_set="disk",
                #init_level_set="small disk",
                )
# Plot image and segmentation contour
ax = plt.figure(figsize=(3,3)).gca()
ax.imshow(img, zorder=0, cmap='gray')
ax.contour(msk, levels=[0.5], zorder=1,
           colors=['tab:orange'])
```

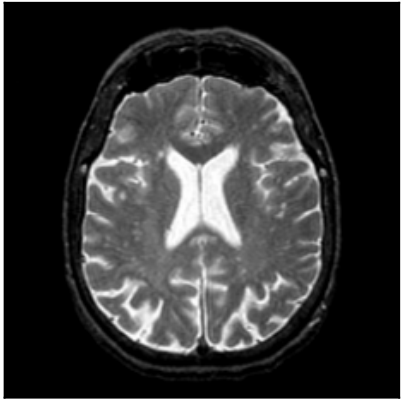


Image segmentation: Active contour in Python

```python
import numpy as np
import matplotlib.pyplot as plt
import skimage.io
from skimage.segmentation import chan_vese
# Load MR brain image
directory = os.path.join('..', '..',
                         'data',
                         '00-additional',
                         'cortex')
filename = 'cortex.bmp'
filepath = os.path.join(directory, filename)
img = skimage.io.imread(filepath)
img = img[:,:,0]
# Segment using active contour
msk = chan_vese(img, mu=0.13,
                lambda1=1, lambda2=1, tol=1e-3,
                max_num_iter=200, dt=0.5,
                init_level_set="checkerboard",
                #init_level_set="disk",
                #init_level_set="small disk",
                )
# Plot image and segmentation contour
ax = plt.figure(figsize=(3,3)).gca()
ax.imshow(img, zorder=0, cmap='gray')
ax.contour(msk, levels=[0.5], zorder=1,
           colors=['tab:orange'])
```

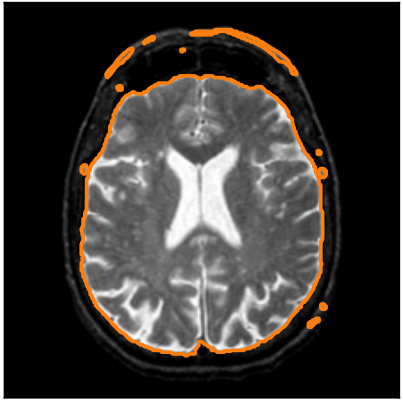


Image segmentation: Post processing with region labels

By selecting the largest segmented **labeled** region in a **post processing** step a good result can be achieved (in this case).

```
from skimage.morphology import label
lbl, N = label(msk, connectivity=1, return_num=True)

# Find the label with the largest region besides
# the label at index 0 which is the background
label_index = -1
label_count = -1
for i in range(1, N+1):
    n = np.count_nonzero(lbl==i)
    if n > label_count:
        label_count = n
        label_index = i
msk = (lbl == label_index).astype(int)

# Plot image and selected segmentation contour
ax = plt.figure(figsize=(3,3)).gca()
ax.imshow(img, zorder=0, cmap='gray')
ax.contour(msk, levels=[0.5], zorder=1,
           colors=['tab:orange'])
```

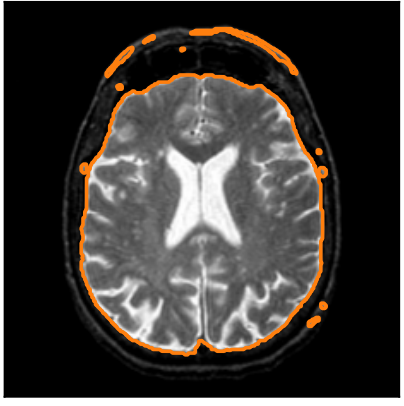


Image segmentation: Post processing with region labels

By selecting the largest segmented **labeled** region in a **post processing** step a good result can be achieved (in this case).

```
from skimage.morphology import label
lbl, N = label(msk, connectivity=1, return_num=True)

# Find the label with the largest region besides
# the label at index 0 which is the background
label_index = -1
label_count = -1
for i in range(1, N+1):
    n = np.count_nonzero(lbl==i)
    if n > label_count:
        label_count = n
        label_index = i
msk = (lbl == label_index).astype(int)

# Plot image and selected segmentation contour
ax = plt.figure(figsize=(3,3)).gca()
ax.imshow(img, zorder=0, cmap='gray')
ax.contour(msk, levels=[0.5], zorder=1,
           colors=['tab:orange'])
```

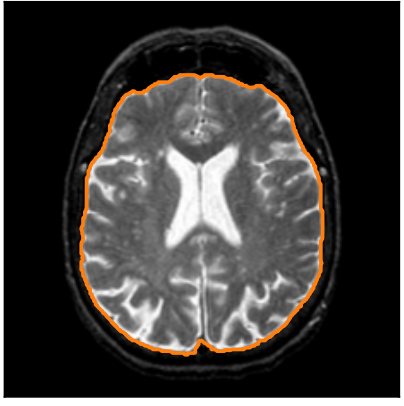


Image segmentation: Graph cuts

Graph based spectral method similar to what we have seen in clustering and manifold learning.

An adjancey matrix is constructed from the graph of pixels and eigen decomposed such that the pixels are divided into foreground and background.

Image segmentation: Summary

Threshold

Active contour

Graph cuts

Deep learning

Image masks

Image masks

Image masks: Digital geometric object

Output from segmentation is an image mask, a digital, binary, geometric object.

Often desirable to be able to process these as well.

Discuss a some types of processing:

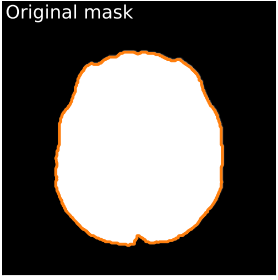
- Morphological operations
- Set operations
- Distance transforms

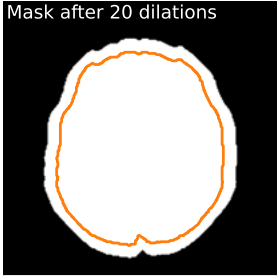
Image masks: Morphological operators

Neighborhood operations similar to the convolution but with a **structure element** instead of a kernel that can **erode** or **dilate** the mask.

The structure element is often some kind of digital ball or disk.

```
1 from skimage.morphology import binary_dilation
2 from skimage.morphology import binary_erosion
3
4 msk_d = msk.copy()
5 msk_e = msk.copy()
6 n_times = 20
7 for i in range(n_times):
8     msk_d = binary_dilation(msk_d)
9     msk_e = binary_erosion(msk_e)
```





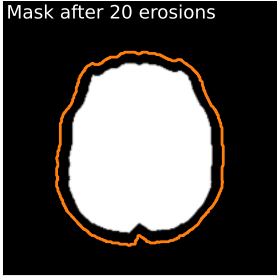


Image masks: Set operations

Set operations can be used to *e.g.* to join masks or take their difference.

- Union
- Intersection
- Difference

```
# Masks can be operatated on using numpys
# bitwise Boolean operators.

# Compute 'union' using the 'or' operator
lung_msk = lung_msk1 | lung_msk2

ax = plt.figure(figsize=(3,3)).gca()
ax.imshow(img, zorder=0, cmap='bone')
ax.contour(lung_msk, levels=[0.5], zorder=1,
           colors=['tab:orange'])
```

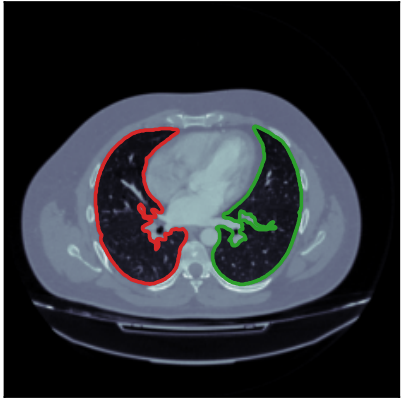


Image masks: Set operations

Set operations can be used to *e.g.* to join masks or take their difference.

- Union
- Intersection
- Difference

```
# Masks can be operatated on using numpys
# bitwise Boolean operators.

# Compute 'union' using the 'or' operator
lung_msk = lung_msk1 | lung_msk2

ax = plt.figure(figsize=(3,3)).gca()
ax.imshow(img, zorder=0, cmap='bone')
ax.contour(lung_msk, levels=[0.5], zorder=1,
           colors=['tab:orange'])
```

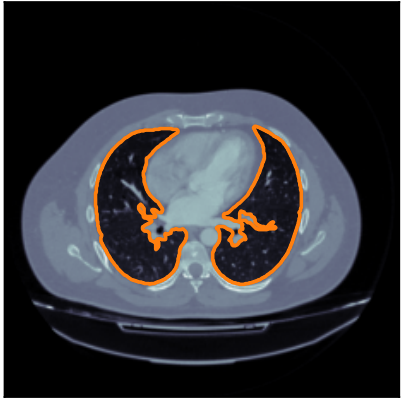


Image masks: Distance transform

Sometimes interesting to know how far a certain pixel is from the image mask.

This can be achieved with a **distance transform**.

Can further be used to compute **distances between image masks** to compare them.

```
# Compute distance transform of the inverse mask
import scipy.ndimage
msk = np.amax(msk) - msk
dst = scipy.ndimage.distance_transform_edt(msk)

# Periodic function to create a wavy pattern
# for visualization only
dst = np.sin(0.3 * dst)

# Plot image and selected segmentation contour
ax = plt.figure(figsize=(3 ,3)).gca()
ax.imshow (dst, zorder=0, cmap = 'twilight')
ax.contour(msk, levels =[0.5], zorder=1 ,
           colors =['tab:orange'])
```

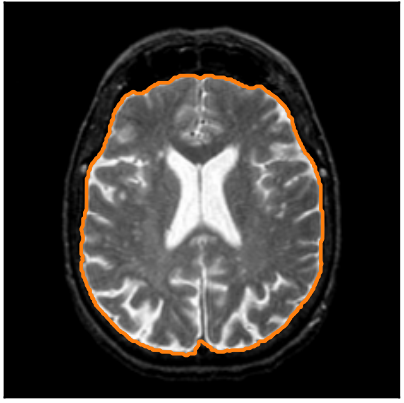


Image masks: Distance transform

Sometimes interesting to know how far a certain pixel is from the image mask.

This can be achieved with a **distance transform**.

Can further be used to compute **distances between image masks** to compare them.

```
# Compute distance transform of the inverse mask
import scipy.ndimage
msk = np.amax(msk) - msk
dst = scipy.ndimage.distance_transform_edt(msk)

# Periodic function to create a wavy pattern
# for visualization only
dst = np.sin(0.3 * dst)

# Plot image and selected segmentation contour
ax = plt.figure(figsize=(3 ,3)).gca()
ax.imshow (dst, zorder=0, cmap = 'twilight')
ax.contour(msk, levels =[0.5], zorder=1 ,
           colors =['tab:orange'])
```

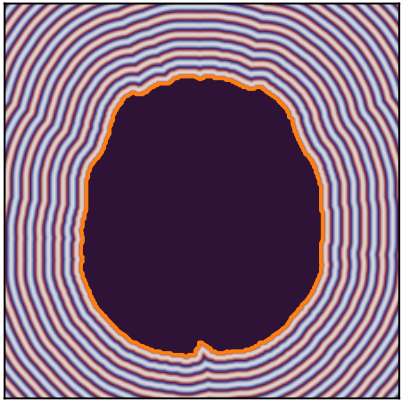


Image masks: Compare different masks

Important to be able to compare masks to a that are the results of different segmentation methods.

Metrics

- Precision
- Recall
- Jacard index

Distance

- Distance between centers of mass.
- Hausdorff distance

Image masks: Summary

Morphological operators

Set operations

Distance transforms

Metrics and distances

Image registration

Image registration

Image registration: Align images

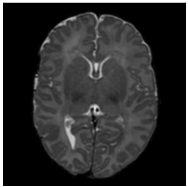
Want to visualize of analyze two or more images in the same **frame of reference**. But with **images** that are **not aligned** due to **shifts, rotations** or some other type of **motion** or **anatomical differences**.

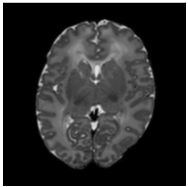
Histology

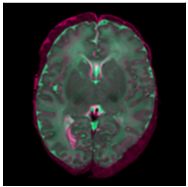
Microscopy photos of same tissue slice needs to be aligned and stitched together.

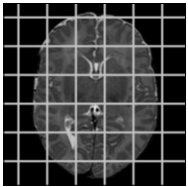
Medical imaging

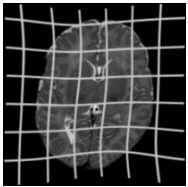
Map to common annotated anatomy for analysis.
In radiotherapy to adapt treatments to motion and anatomical changes.


Moving


Fixed


Comparison
before registration


Moving (Grid)


Moving (deformed)

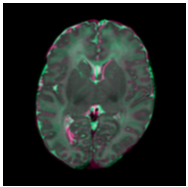

Comparison
after registration

Image registration: Scans of the same subject

Want to track or accommodate for motion or anatomical changes.

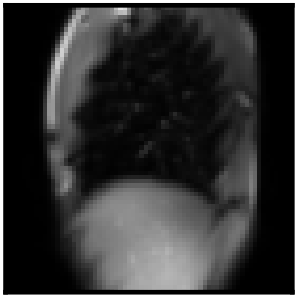
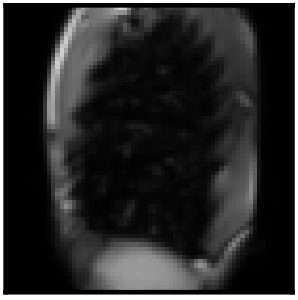
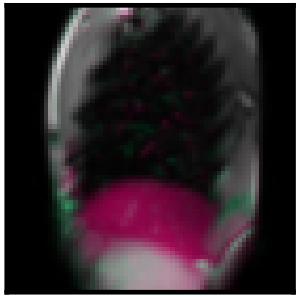
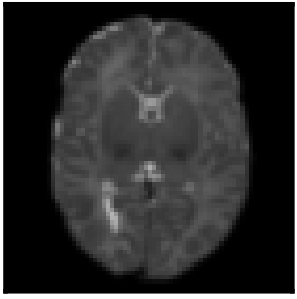
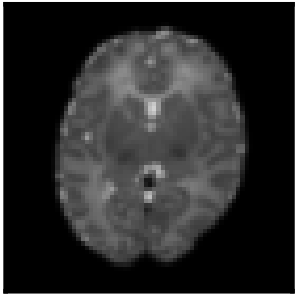




Image registration: Scans of different subjects

Want to map subjects to a common frame of reference anatomy to do analysis.





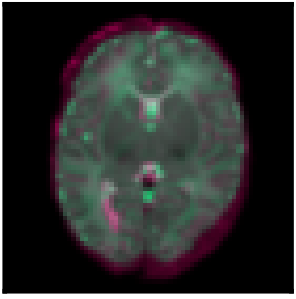


Image registration: Problem and solution

Want to find transform such that the images are aligned such that their content match. Three things are required.

Transform

Type of motion or changes between images.

Similarity metric

How to compare two images.

Optimizer

Find optimal parameters for the transform that maximize the similarity.

Image registration: Spatial objects

It is very helpful to think about images not as arrays of pixels, but as spatial objects that have been sampled on a regular grid.

Also need a way to resample the images (Both in the end and on common geometry during registration).

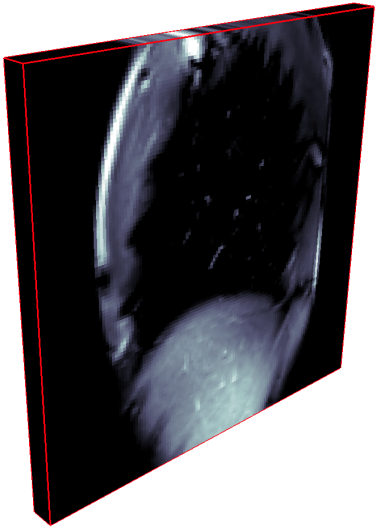


Image registration: Transforms

Translation

Rotation

Rigid transform

Both translation and rotation.

Deformable

- **Bspline**
- **Demons**

Image registration: Similarity metric

How do we know when two images are aligned?
Need way of comparing them.

E.g. compare the pixel values and calculate a mean of the square differences.

$$MSE = \frac{1}{N}\sum_i (I(i) - J(i))^2$$

The mean squared error is sensitive to systematic shifts in intensity between images.

It also typically does not work for images of different modalities such as CT and MRI.

- Mean squared error
- Cross correlation (Normalized)
- Mutual information

Image registration: Similarity metrics

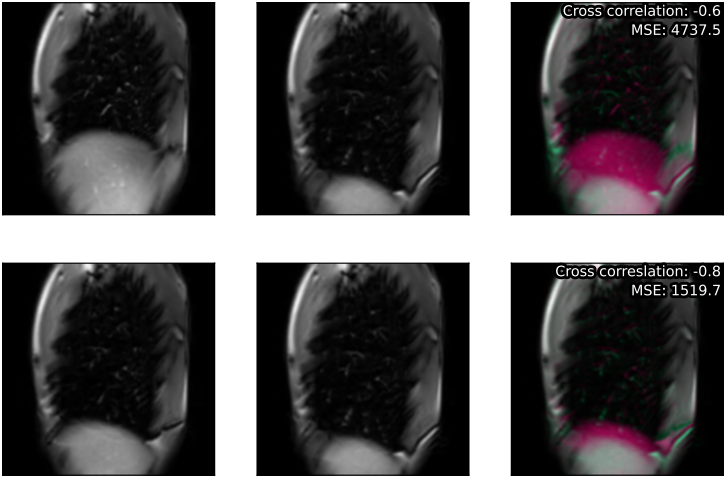


Image registration: Optimizer

We have a transform with parameters we want to find such that the similarity, S, is maximized.

This is an optimization problem. Sometimes also add regularization to prevent *unrealistic* solutions.

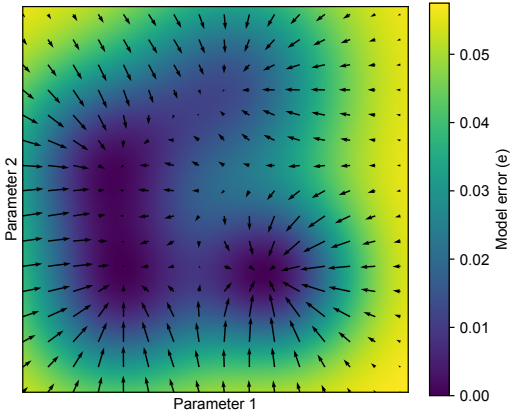
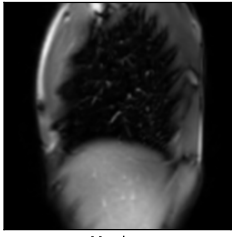


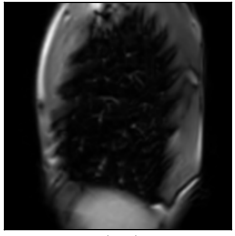
Image registration: Python

```python
import numpy as np
import matplotlib.pyplot as plt
import skimage.io
import SimpleITK as sitk
# Load MR brain images
directory = os.path.join('..', '..',
                         'figures',
                         '13-image-processing',
                         'neonatal-mri')
filenames = ['0_t2w.jpg',
             '64_t2w.jpg']
images = []
for filename in filenames:
    filepath = os.path.join(directory, filename)
    img = skimage.io.imread(filepath)
    img = sitk.GetImageFromArray(img)
    img = sitk.Cast(img, sitk.sitkFloat32)
    images.append(img)
fusion = common.createFusedImage(images[0], images
    [1])

moving = images[0]
fixed = images[1]

# Registration
tr= sitk.BSplineTransformInitializer(moving,
                                     [5, 5])
reg = sitk.ImageRegistrationMethod()
reg.SetInitialTransform(tr, inPlace=True)
reg.SetMetricAsCorrelation()
reg.SetInterpolator(sitk.sitkLinear)
reg.SetOptimizerAsLBFGSB(
    gradientConvergenceTolerance = 1e-3,
                         numberOfIterations = 1000)
reg.SetShrinkFactorsPerLevel([8, 4, 2, 1])
reg.SetSmoothingSigmasPerLevel([9, 7, 3, 0])
tr = reg.Execute(fixed, moving)

# Resampler
resampler = sitk.ResampleImageFilter()
resampler.SetReferenceImage(fixed)
resampler.SetInterpolator(sitk.sitkLinear)
resampler.SetDefaultPixelValue(0)
resampler.SetTransform(tr)

img_grid = common.addGrid2D(images[0], linewidth=3,
                            linespacing=40, offset
    =20)
tr_img_grid = resampler.Execute(img_grid)
tr_img = resampler.Execute(images[0])
fusion_res = common.createFusedImage(tr_img, images
    [1])

# Plot image and segmentation contour
fig, axs = plt.subplots(1, 3, figsize=(3,9))
axs[0].imshow(sitk.GetArrayFromImage(images[0]),
              cmap='gray', interpolation='bicubic')
axs[1].imshow(sitk.GetArrayFromImage(images[1]),
```

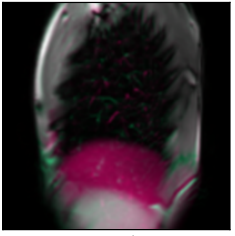
Image registration: Results (Lung)



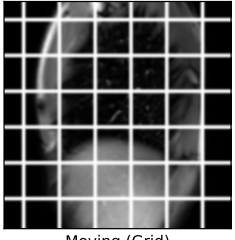
Moving



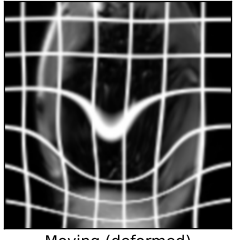
Fixed



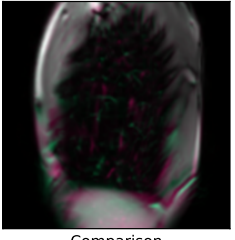
Comparison
before registration



Moving (Grid)

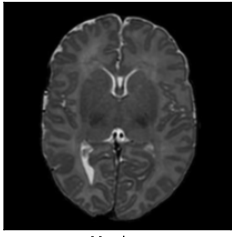


Moving (deformed)

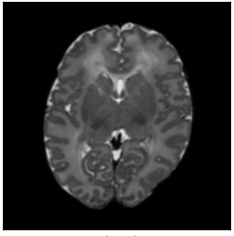


Comparison
after registration

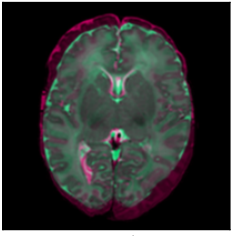
Image registration: Results (Brain)



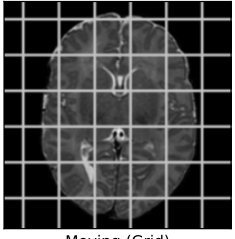
Moving



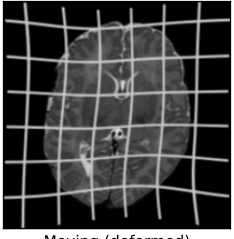
Fixed



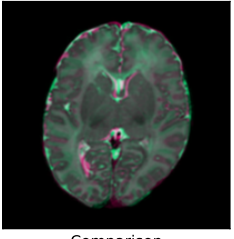
Comparison
before registration



Moving (Grid)

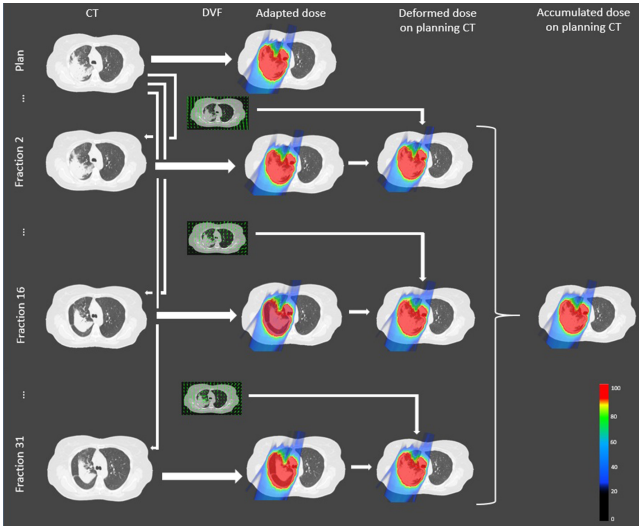


Moving (deformed)



Comparison
after registration

Image registration: Radiotherapy



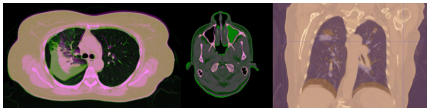


Image registration: Summary

Transform

Similarity metric

Optimizer

Conclusions

Conclusions

Conclusions: Working with images