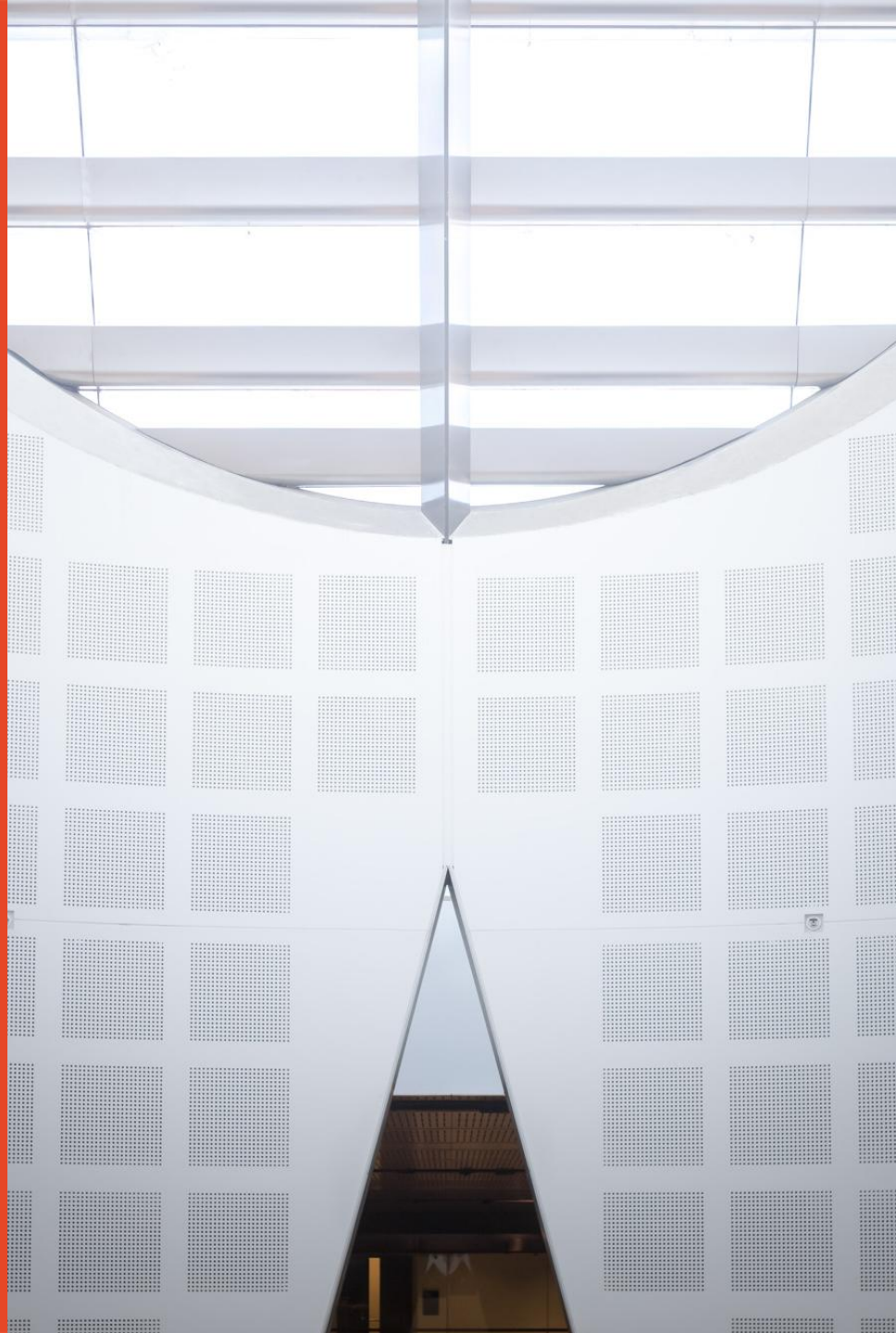
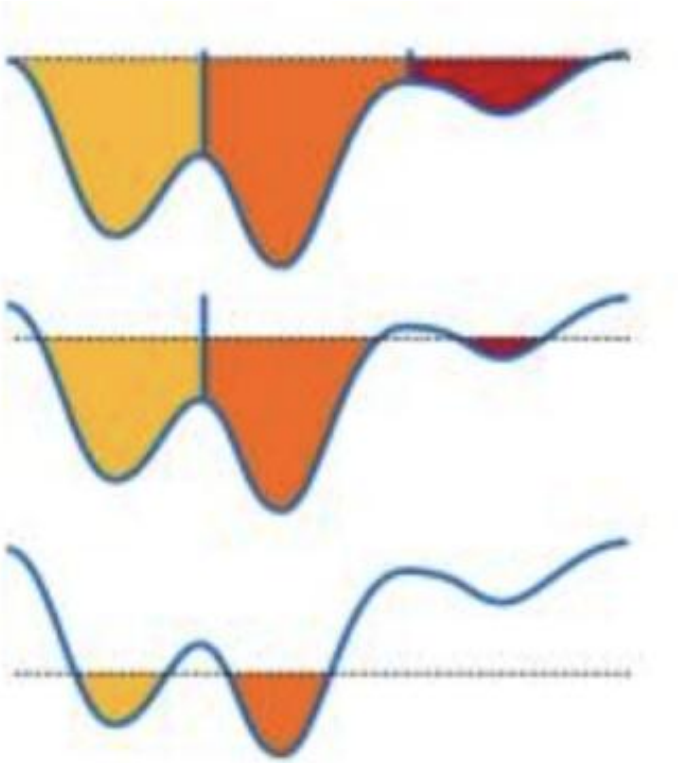
Watershed and DeepReg



THE UNIVERSITY OF SYDNEY

Watershed Segmentation



https://docs.opencv.org/4.x/

Watershed Segmentation with Seeds

- Import packages
- Load the image
- Convert to grayscale
- Thresholding and fill holes
- Find sure background area
- Distance transform
- Find sure foreground areas
- Watershed in the unknown area
- Get the result

https://docs.opencv.org/4.x/

Watershed Segmentation with Seeds

- Import packages
- Load the image
- Convert to grayscale
- Thresholding and fill holes
- Find sure background area
- Distance transform
- Find sure foreground areas
- Watershed in the unknown area
- Get the result

https://docs.opencv.org/4.x/

import cv2 as cv

OpenCV is an open source computer vision library
highly optimized for real-time applications

https://docs.opencv.org/4.x/

cv.threshold

```
cv.threshold(src, thresh, maxval, type) -> retval, dst
```

Applies a threshold to each array element.

`src`: a grayscale image as input
`thresh`: threshold value
`maxval`: max value
`type`: thresholding type. Common options are:

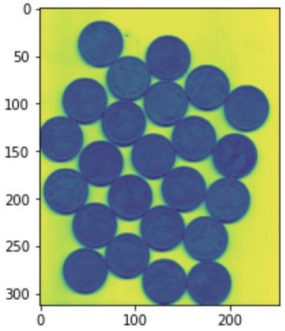
- cv.THRESH_BINARY: basic thresholding type
- cv.THRESH_BINARY_INV: used when the foreground is dark
- cv.THRESH_OTSU: adaptively compute the optimal threshold value

`retval`: threshold value
`dst`: output binary image

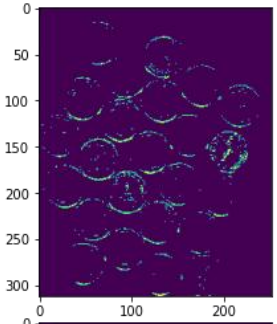
https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57

cv.threshold

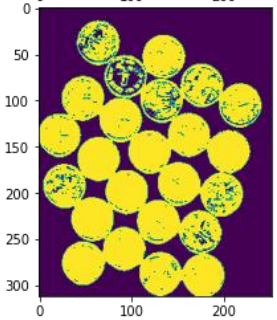
```
cv.threshold(src, thresh, maxval, type) -> retval, dst
```



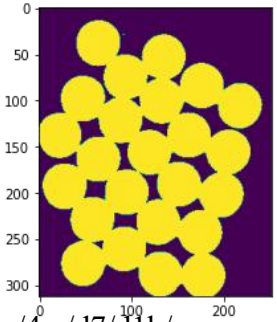
src



cv.threshold(src,75,255,
cv.THRESH_BINARY_INV)



cv.threshold(src,100,255,
cv.THRESH_BINARY_INV)



cv.threshold(src,0,255,
cv.THRESH_BINARY_INV+cv.THRESH_OTSU)

https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57

cv.morphologyEx

```
cv.morphologyEx(src, op, kernel) -> dst
```

Performs morphological transformations. Morphological transformations are some simple operations based on the image shape.

`src`: a binary image as input

`op`: Type of a morphological operation. Common operations are cv.MORPH_ERODE, cv.MORPH_DILATE, cv.MORPH_OPEN, cv.MORPH_CLOSE

`kernel`: Structuring element. Usually larger kernel would change the image more evidently.

`dst`: output image.

https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

cv.morphologyEx

```
cv.morphologyEx(src, op, kernel) -> dst
```

src

dst
(cv.MORPH_ERODE)

dst
(cv.MORPH_ DILATE)

https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

cv.morphologyEx

```
cv.morphologyEx(src, op, kernel) -> dst
```

src

dst

(cv.MORPH_OPEN)

cv.MORPH_OPEN is used for removing noise

https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

cv.morphologyEx

```
cv.morphologyEx(src, op, kernel) -> dst
```

src

dst

(cv.MORPH_CLOSE)

cv.MORPH_CLOSE is used for filling holes

https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

cv.dilate

```
cv.dilate(src, kernel, iteration) -> dst
```

Dilates an image by using a specific structuring element.

`src`: a binary image as input

`kernel`: structuring element used for dilation. (Think it as a filter)
Larger kernel would dilate the image more evidently.

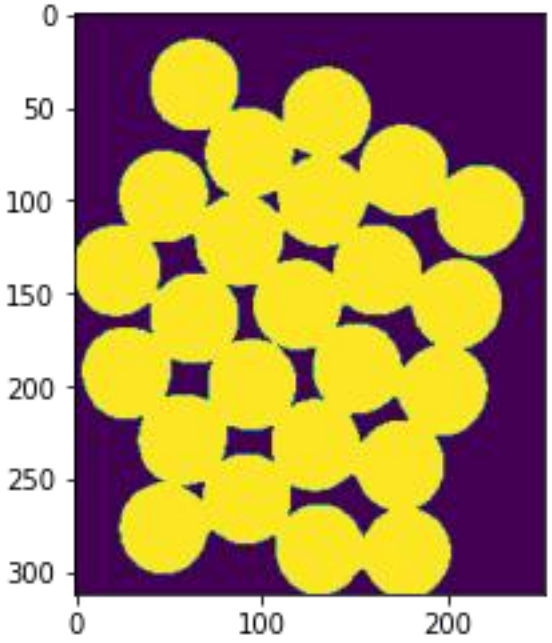
`iteration`: number of times dilation is applied.
Larger iteration would dilate the image more evidently.

`dst`: output image.

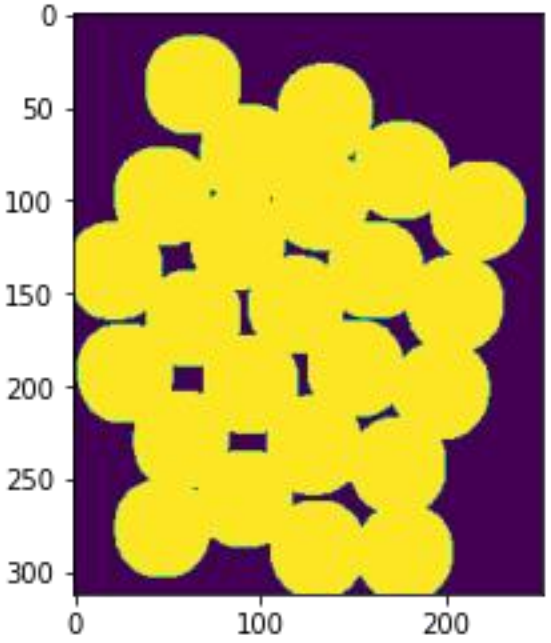
https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga4ff0f3318642c4f469d0e11f242f3b6c

cv.dilate

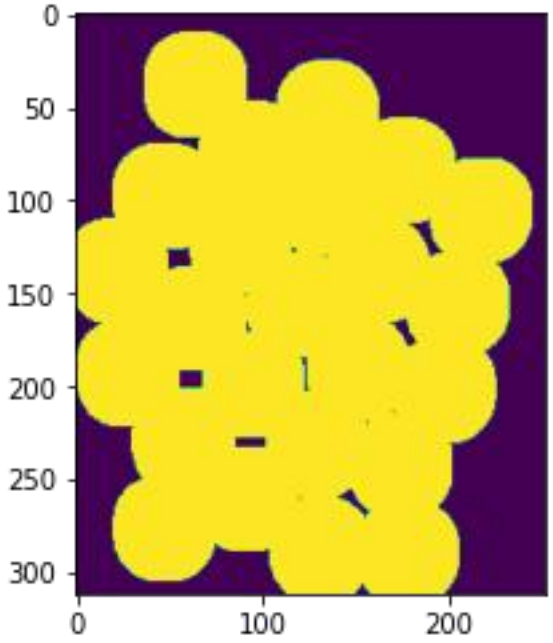
```
cv.dilate(src, kernel, iteration) -> dst
```



src



dst
(kernel 3x3)



dst
(kernel 5x5)

https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga4ff0f3318642c4f469d0e11f242f3b6c

cv.distanceTransform

```
cv.distanceTransform(src, distanceType, maskSize) -> dst
```

Calculates the distance to the closest zero pixel for each pixel.

`src`: a binary image as input

`distanceType`: type of distance.
Common options are cv.DIST_L1, cv.DIST_L2, cv.DIST_C

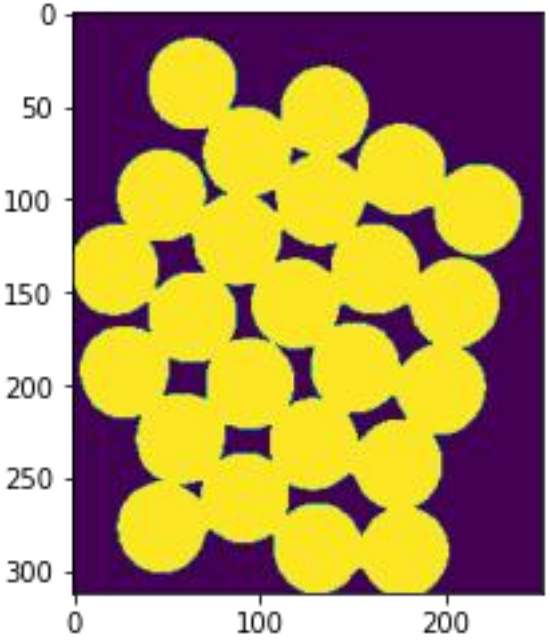
`maskSize`: size of the distance transform mask.
Common options are cv.DIST_MASK_3, cv.DIST_MASK_5

`dst`: output image.

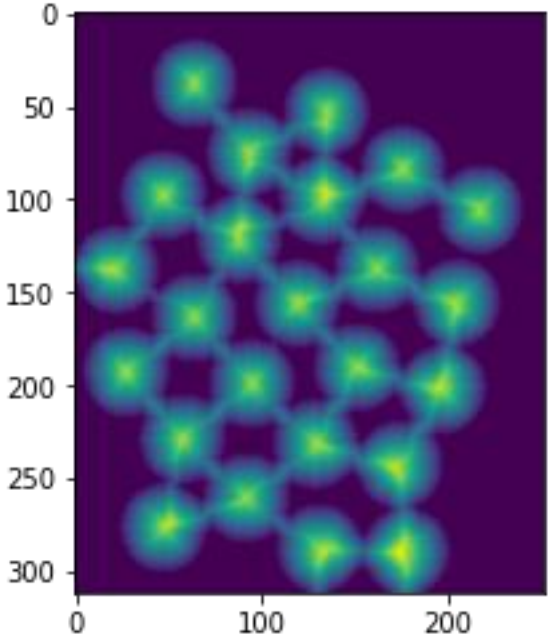
https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#ga25c259e7e2fa2ac70de4606ea800f12f
https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#gaa2bfbebbc5c320526897996aafa1d8eb

cv.distanceTransform

```
cv.distanceTransform(src, distanceType, maskSize) -> dst
```

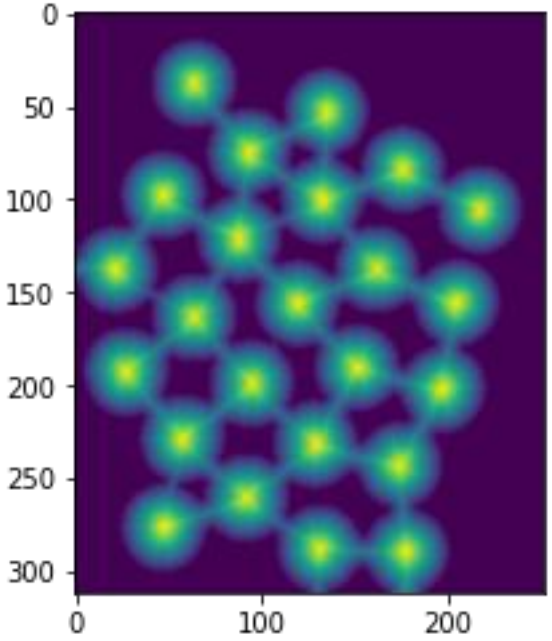


src



dst

(cv.DIST_L1)



dst

(cv.DIST_L2)

https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#ga25c259e7e2fa2ac70de4606ea800f12f
https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#gaa2bfbebbc5c320526897996aafa1d8eb

cv.connectedComponents

```
cv.connectedComponents(image) -> retval, labels
```

Computes the connected components labeled image of boolean image.

`image`: a binary image as input

`retval`: the total number of connected components

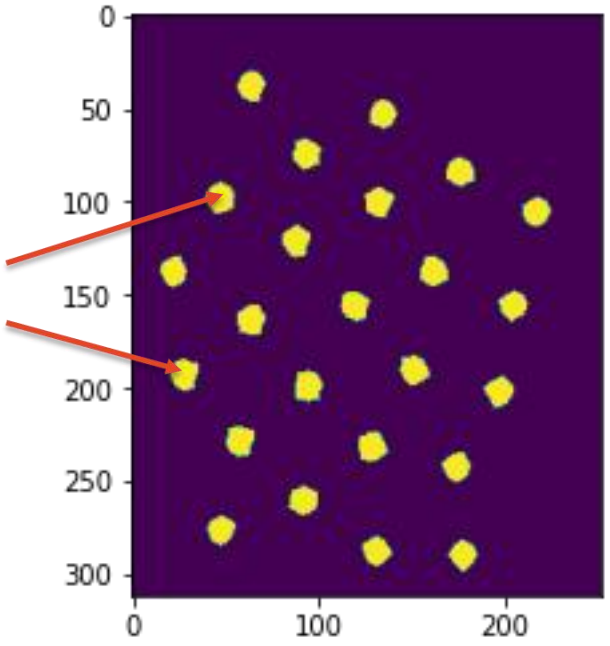
`labels`: a label image in which pixels belong to different components have different values

https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#gac2718a64ade63475425558aa669a943a

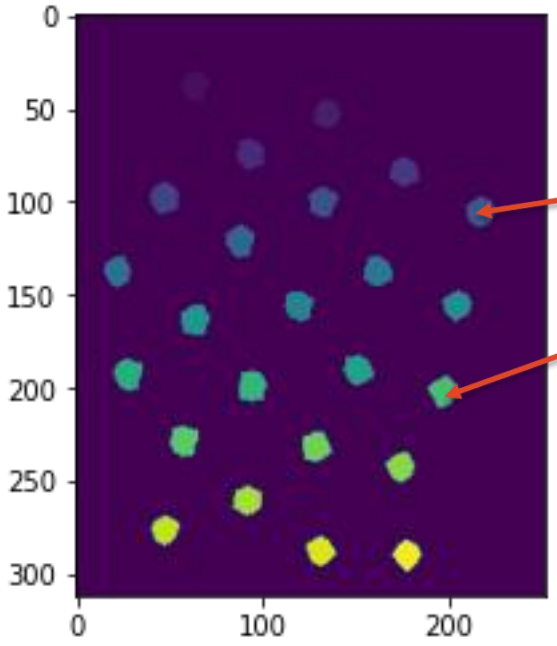
cv.connectedComponents

```
cv.connectedComponents(image) -> retval, labels
```

Different components have the same value



image



labels

Different components have different values

The output could be used as markers for the watershed segmentation

https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#gac2718a64ade63475425558aa669a943a

cv.watershed

```
cv.watershed(image, markers) -> new_markers
```

Performs a marker-based image segmentation using the watershed algorithm. Before passing the image to the function, you have to roughly outline the desired regions in the image markers with positive (>0) indices. So, every region is represented as one or more connected components with the pixel values 1, 2, 3, and so on.

The markers are "seeds" (water sources) of the future image regions. All the other pixels in markers, whose relation to the outlined regions is not known and should be defined by the algorithm, should be set to 0's.

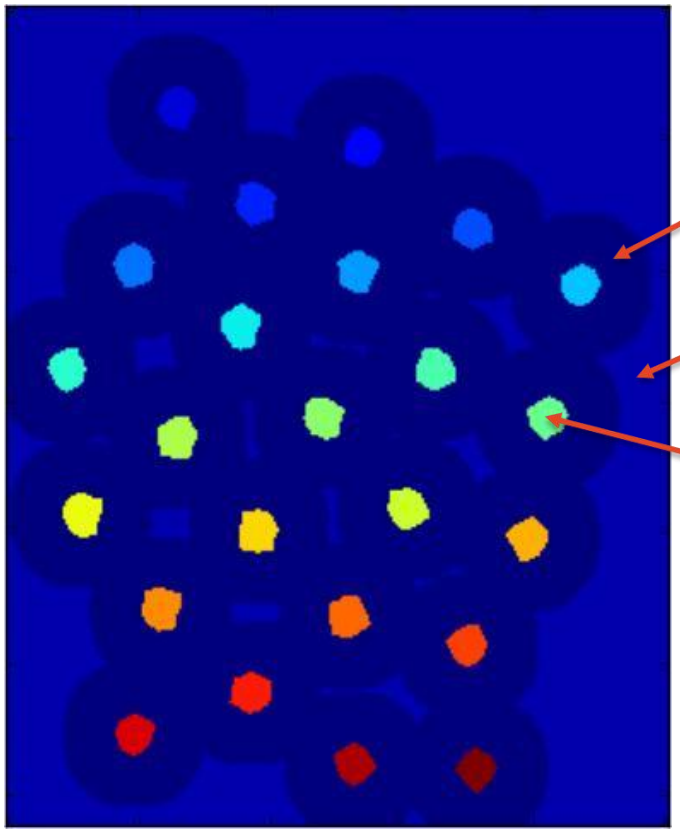
https://docs.opencv.org/4.x/d3/d47/group__imgproc__segmentation.html#ga3267243e4d3f95165d55a618c65ac6e1

cv.watershed

```
cv.watershed(image, markers) -> new_markers
```

image

markers



Unknown marker: 0

Background marker: 1

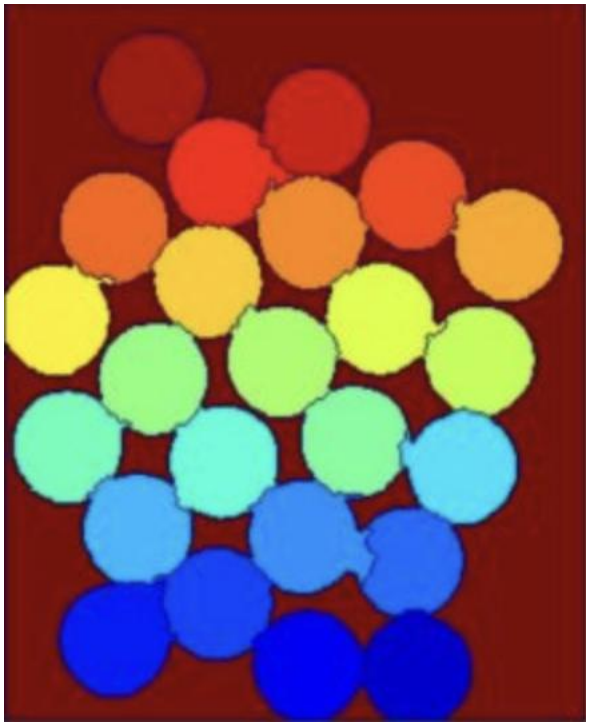
Seed marker: 2,3,4,…

Marker is an image

https://docs.opencv.org/4.x/d3/d47/group__imgproc__segmentation.html#ga3267243e4d3f95165d55a618c65ac6e1

cv.watershed

```
cv.watershed(image, markers) -> new_markers
```



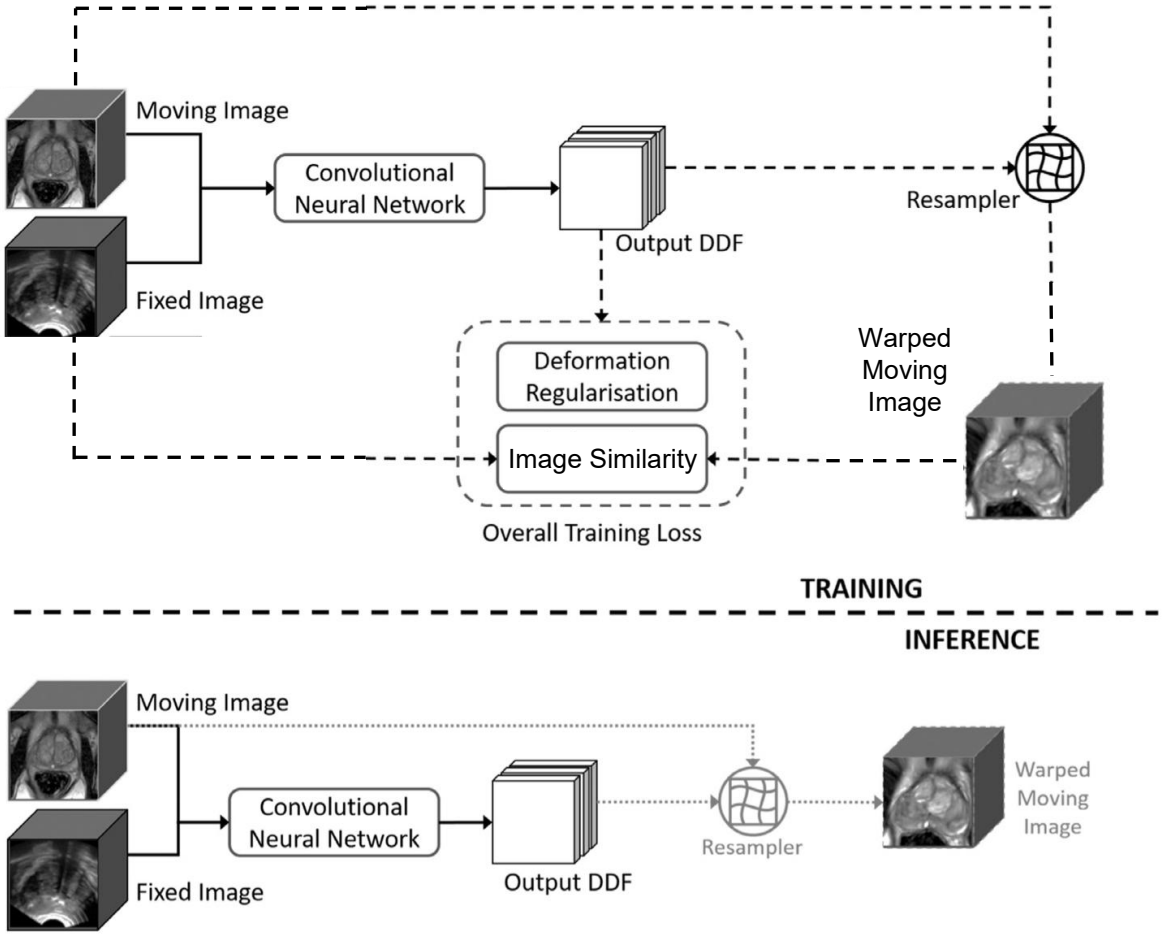
new_markers

In the function output, each pixel in markers is set to a value of the "seed" components or to -1 at boundaries between the regions.

https://docs.opencv.org/4.x/d3/d47/group__imgproc__segmentation.html#ga3267243e4d3f95165d55a618c65ac6e1

DeepReg

Enable GPU in the Colab for this tutorial



import deepreg

DeepReg is a freely available, community-supported open-source toolkit for research and education in medical image registration using deep learning

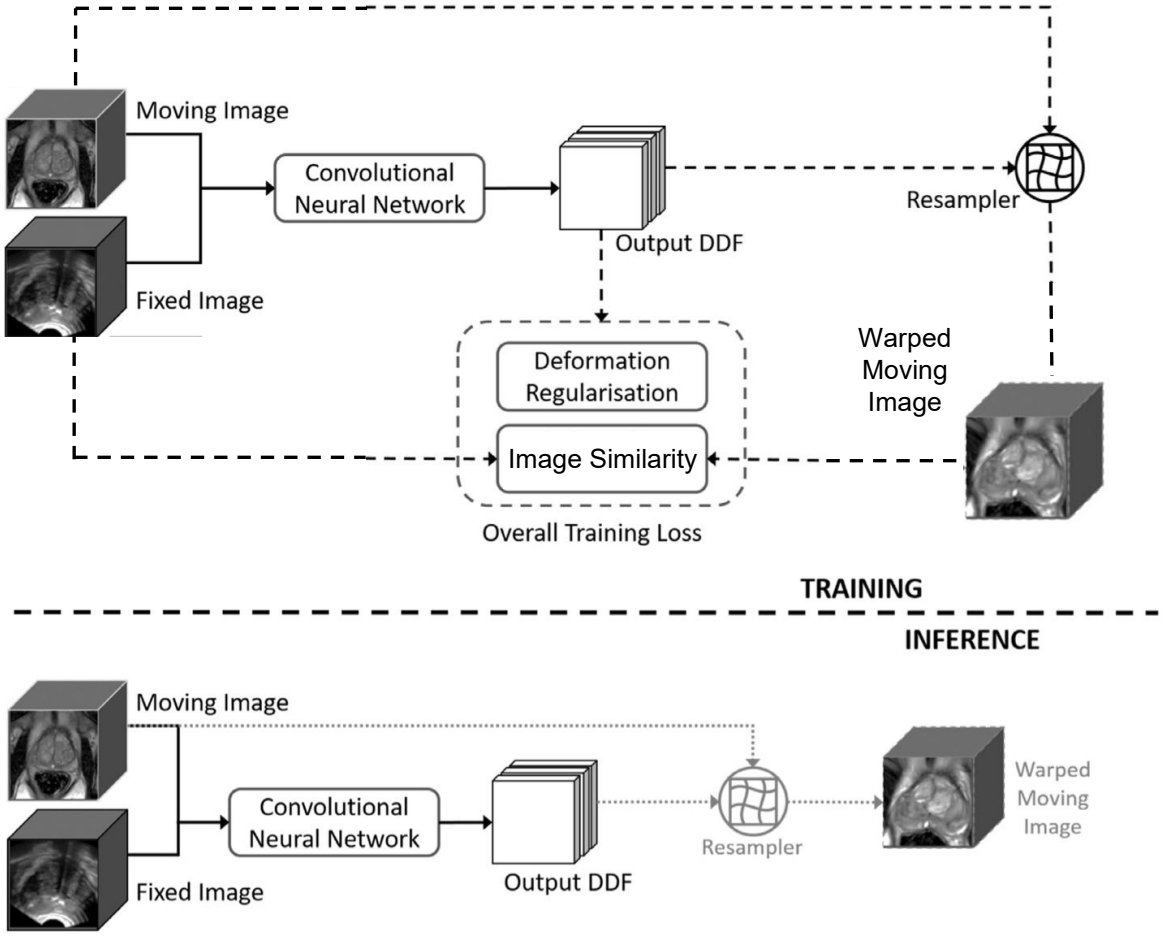
Latest version is v0.1.2
The notebook (MICCAI Tutorial) uses an older version of DeepReg

Check https://github.com/DeepRegNet/DeepReg/tree/miccai2020-challenge/deepreg/model for documentation

https://github.com/DeepRegNet/DeepReg

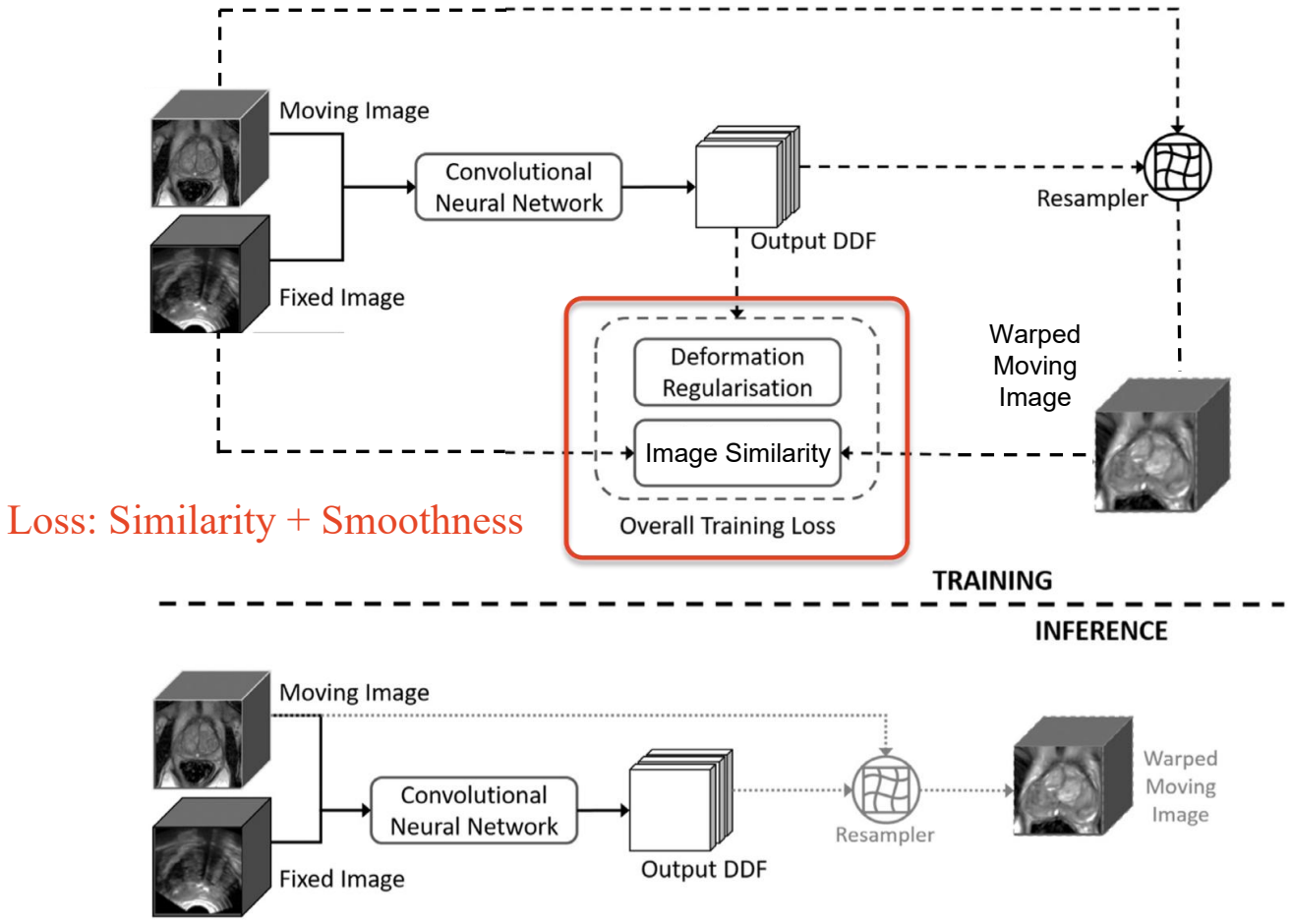
DeepReg

Enable GPU in the Colab for this tutorial



DeepReg

Enable GPU in the Colab for this tutorial



deepreg.model.loss.image.dissimilarity_fn

`dissimilarity_fn(y_true, y_pred, name) -> loss`

Compute the dissimilarity between two medical images.

`y_true`: Fixed image. shape = (batch, f_dim1, f_dim2, f_dim3)

`y_pred`: Warped moving image. shape = (batch, f_dim1, f_dim2, f_dim3)

`name`: type of dissimilarity. Options are:

- "lncc": Local normalized cross-correlation
- "ssd": Sum of squared distance
- "gmi": Global mutual information

`loss`: The dissimilarity to be minimized. shape = (batch,)

https://deepreg.readthedocs.io

deepreg.model.loss.deform.local_displacement_energy

```
local_displacement_energy(ddf, energy_type) -> loss
```

Calculate the displacement energy of the dense displacement field

`ddf`: dense displacement field to be regularized
shape = (batch, m_dim1, m_dim2, m_dim3, 3)

`energy_type`: type of the energy. Options are:

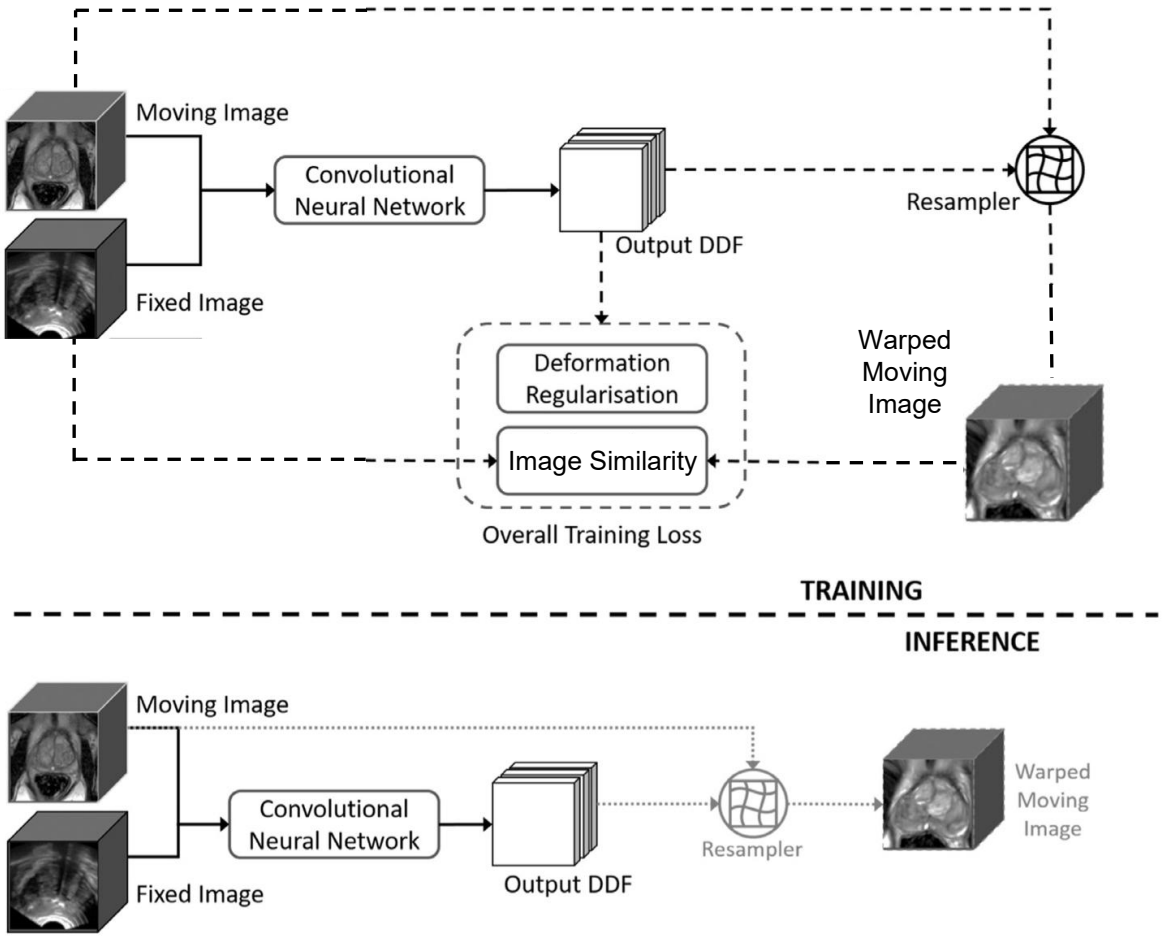
- "bending": bending_energy
- "gradient-l1": ddf gradient l1 norm
- "gradient-l2": ddf gradient l2 norm

`loss`: the loss function to be minimized
shape = (batch,)

https://deepreg.readthedocs.io

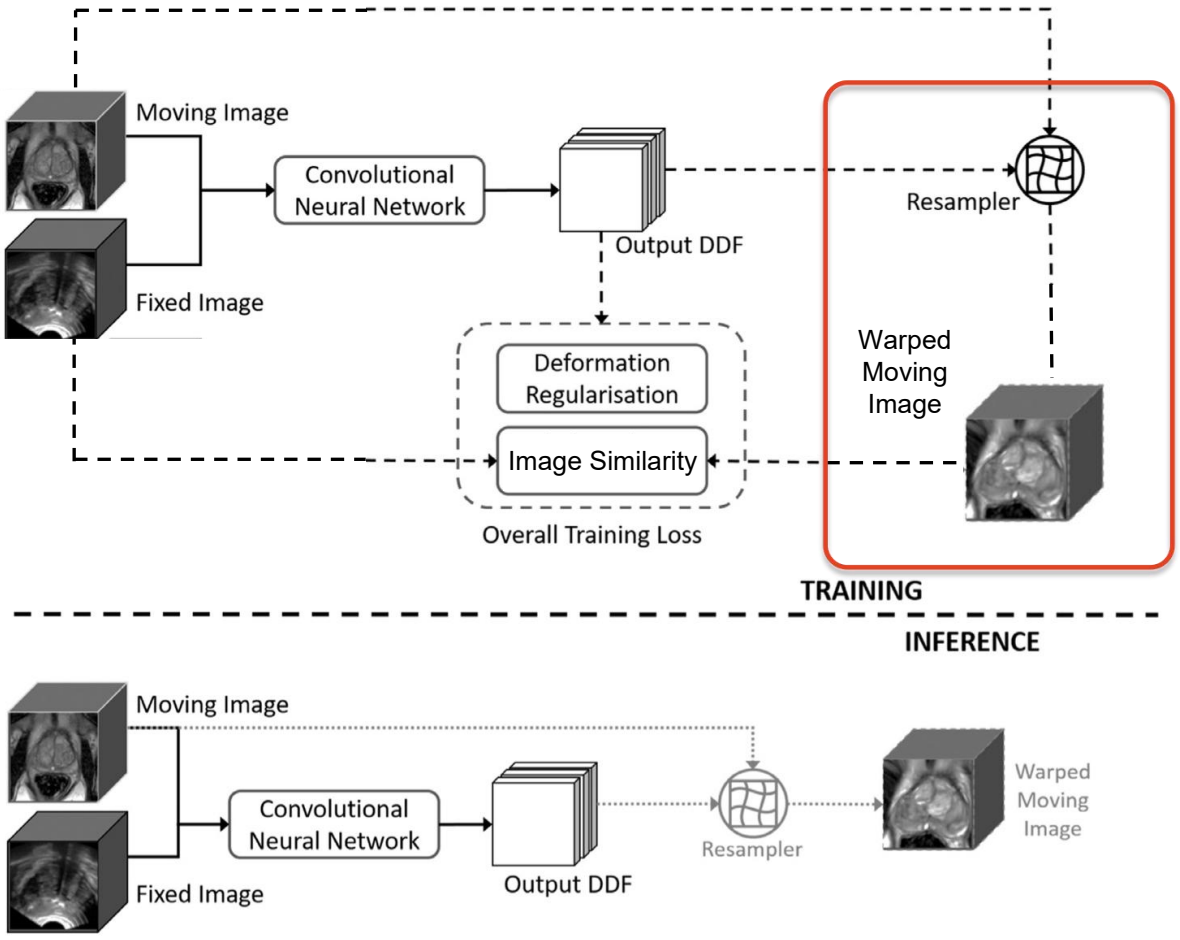
DeepReg

Enable GPU in the Colab for this tutorial



DeepReg

Enable GPU in the Colab for this tutorial



Warping

deepreg.model.layer_util.get_reference_grid

```
get_reference_grid(size) -> grid
```

Generate a grid with the given size

deepreg.model.layer_util.warp_grid

```
warp_grid(grid, transform) -> warped_grid
```

Warp the grid using some affine transform

deepreg.model.layer_util.resample

```
resample(volume, warped_grid) -> warped_volume
```

Sample the volume (image) at grid locations.

https://deepreg.readthedocs.io

deepreg.model.layer.Warping

```
Warping(fixed_image_size) -> warping_object
```

Create an DDF warping layer

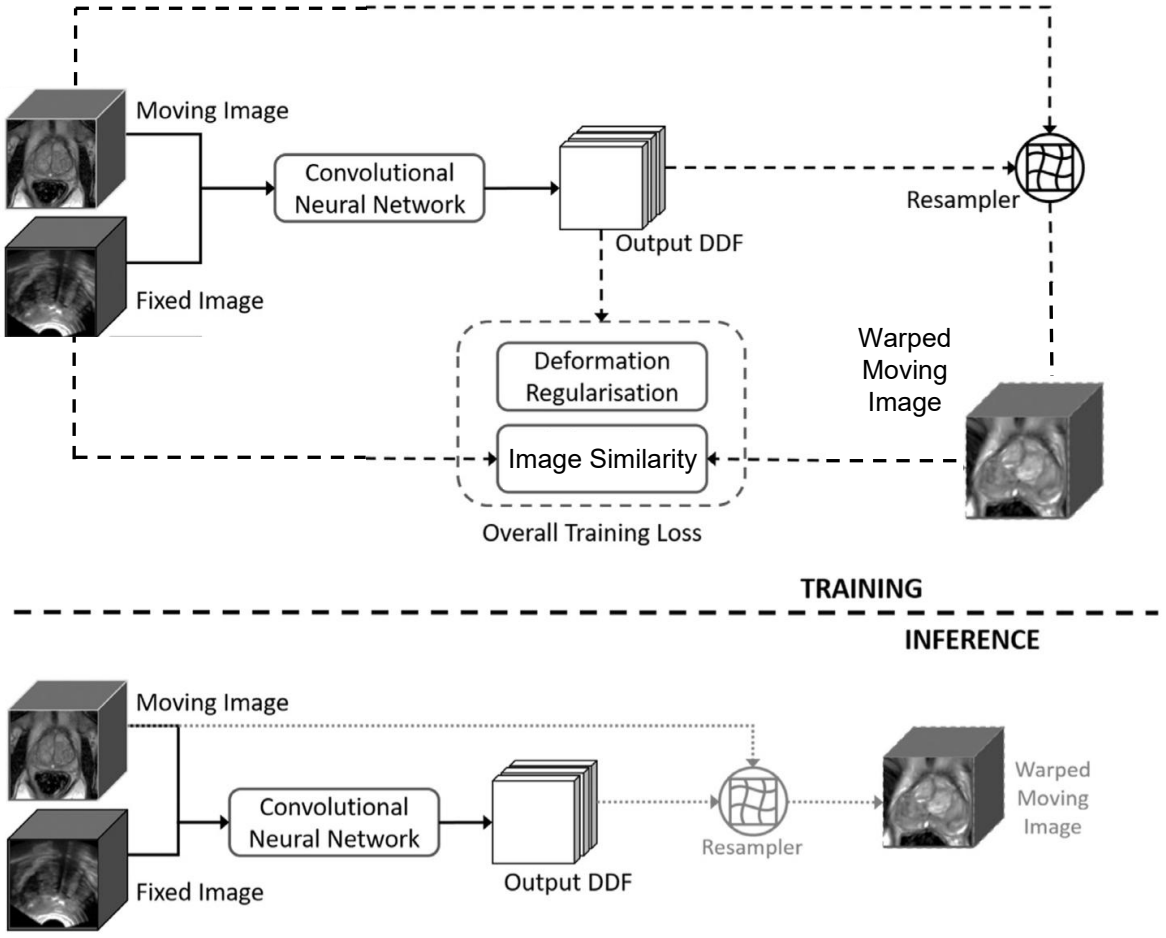
```
warping_object((ddf, moving_image)) -> warped_image
```

Call the warping object to apply the transform on an image

https://deepreg.readthedocs.io

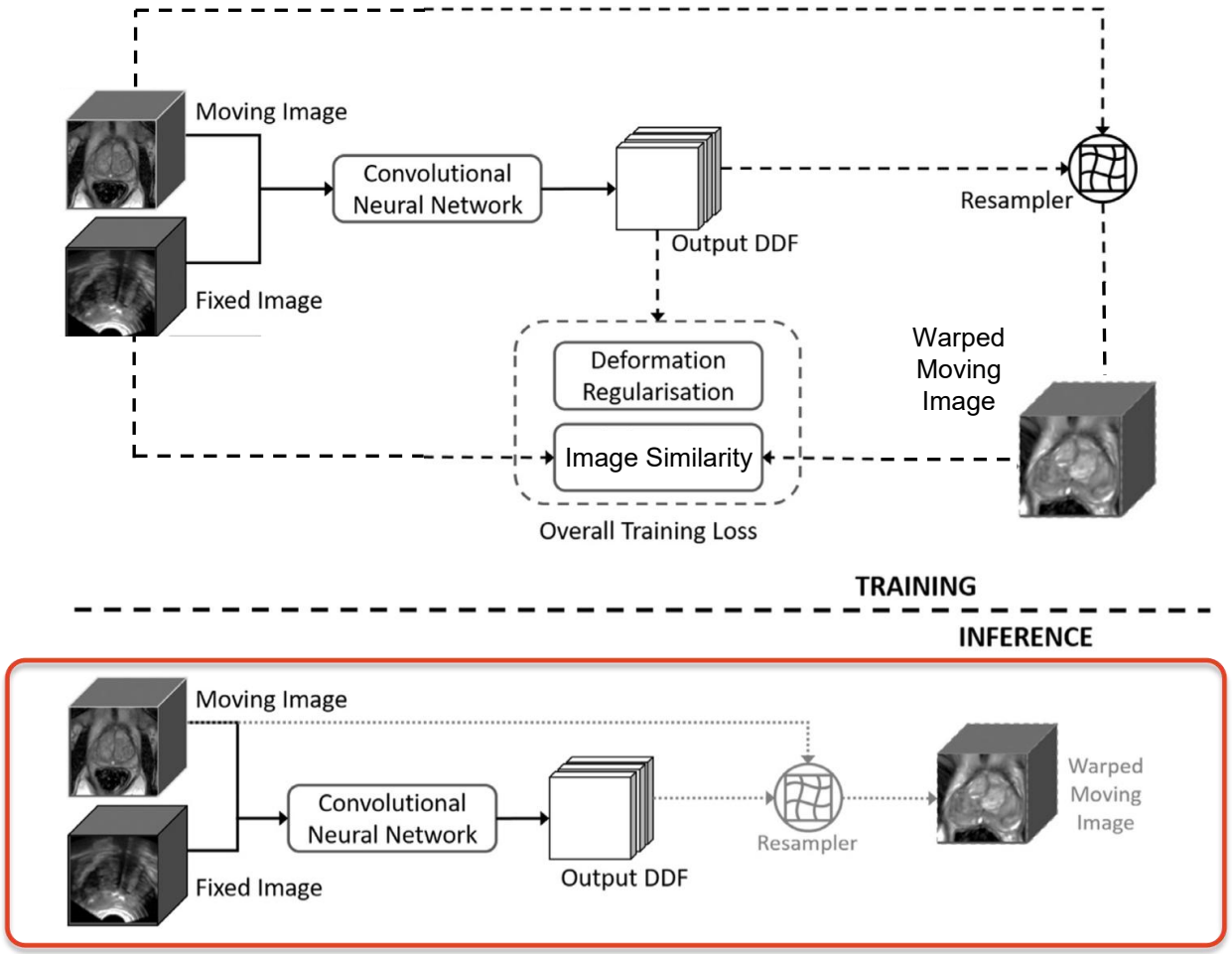
DeepReg

Enable GPU in the Colab for this tutorial



DeepReg

Enable GPU in the Colab for this tutorial



Registration with a pre-trained model

Have fun playing with the notebooks