DEEPHEALTH

Doing CV with ECVL

Costantino Grana - UNIMORE

Winter School 24/01/2022

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Contents

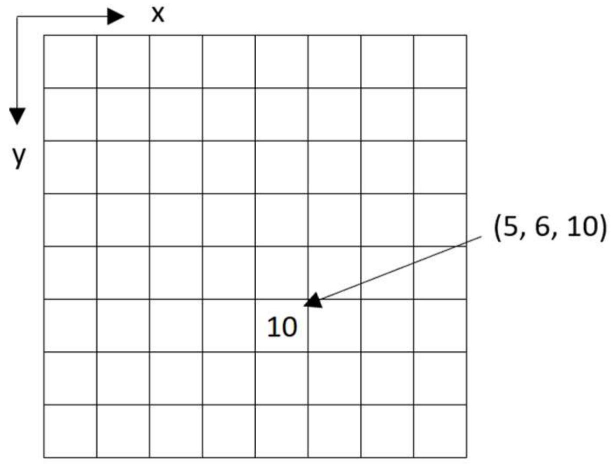
The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Images

- A (digital) image is defined by *integrating* and *sampling* continuous (analog) data in a spatial domain.
- It consists of a rectangular array of *pixels* (x, y, u), each combining a location $(x, y) \in \mathbb{Z}^2$ and a value u, the *sample* at location (x, y).
- An image I with N_{cols} and N_{rows} is defined on a rectangular set

$$\Omega = \{(x, y): 1 \leq x \leq N_{cols} \wedge 1 \leq y \leq N_{rows}\} \subset \mathbb{Z}^2$$

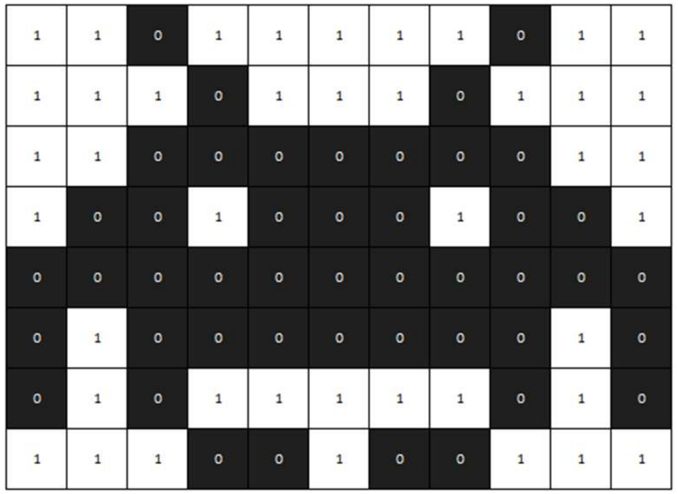
  containing the pixel locations.
- It is common practice to have x increase from left to right and y increase from top to bottom, which is contrary to the classical Cartesian practice.



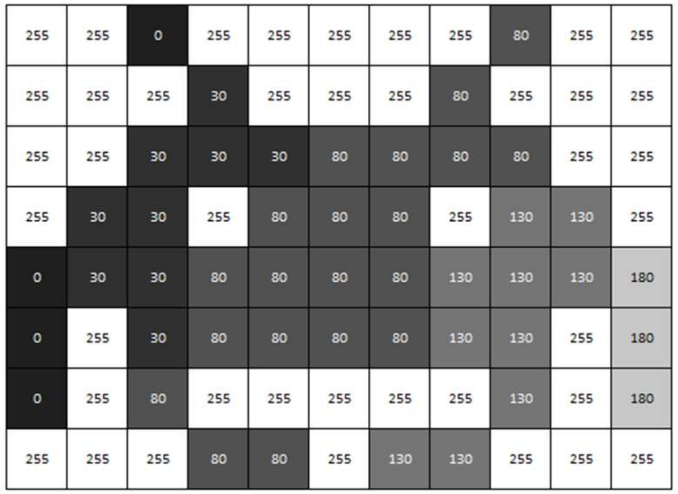
The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Images

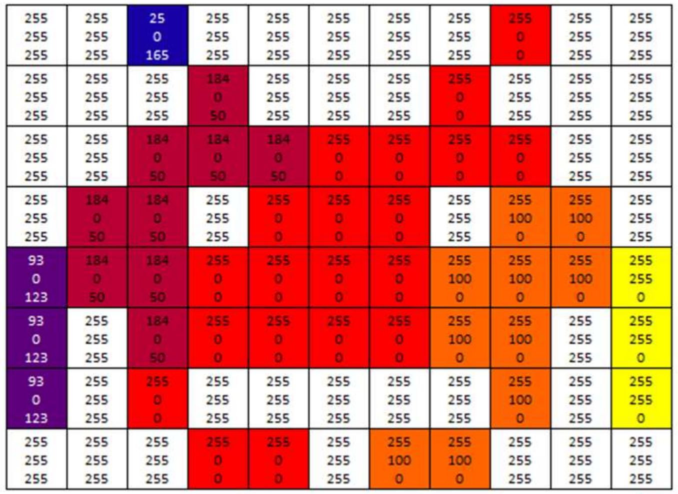
- The sample *u* can be a scalar value, which usually represents light intensity, or a vector value, that is the intensity of different light spectra.
- The value can be binary (0 or 1) in case of black and white images, or an integer value from 0 to 2^n, when images have *n* bits per pixel (bpp). A typical graylevel image has 256 levels (8 bpp), but current digital cameras can deliver values with 12, 14 or even 16 bpp.



1	1	0	1	1	1	1	1	0	1	1
1	1	1	0	1	1	1	0	1	1	1
1	1	0	0	0	0	0	0	0	1	1
1	0	0	1	0	0	0	1	0	0	1
0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	1	0
0	1	0	1	1	1	1	1	0	1	0
1	1	1	0	0	1	0	0	1	1	1



255	255	0	255	255	255	255	255	80	255	255
255	255	255	30	255	255	255	80	255	255	255
255	255	30	30	30	80	80	80	80	255	255
255	30	30	255	80	80	80	255	130	130	255
0	30	30	80	80	80	80	130	130	130	180
0	255	30	80	80	80	80	130	130	255	180
0	255	80	255	255	255	255	255	130	255	180
255	255	255	80	80	255	130	130	255	255	255

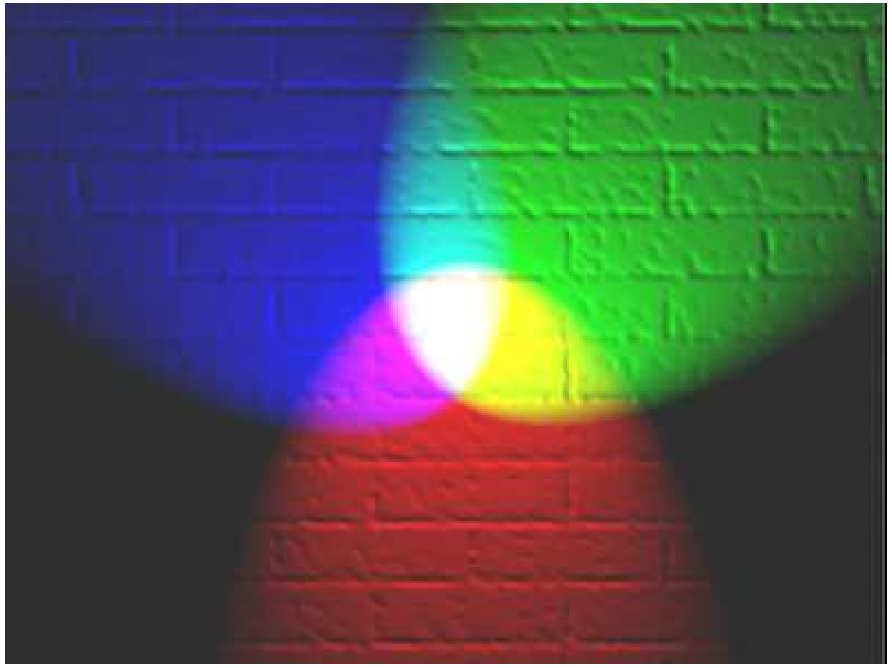


255 255 255	255 255 255	25 0 165	255 255 255	255 255 255	255 255 255	255 255 255	255 255 255	255 0 0	255 255 255	255 255 255
255 255 255	255 255 255	255 255 255	184 0 50	255 255 255	255 255 255	255 255 255	255 0 0	255 255 255	255 255 255	255 255 255
255 255 255	255 255 255	184 0 50	184 0 50	184 0 50	255 0 0	255 0 0	255 0 0	255 0 0	255 255 255	255 255 255
255 255 255	184 0 50	184 0 50	255 255 255	255 0 0	255 0 0	255 0 0	255 255 255	255 100 0	255 100 0	255 255 255
93 0 123	184 0 50	184 0 50	255 0 0	255 0 0	255 0 0	255 0 0	255 100 0	255 100 0	255 100 0	255 255 0
93 0 123	255 255 255	184 0 50	255 0 0	255 0 0	255 0 0	255 0 0	255 100 0	255 100 0	255 255 255	255 255 0
93 0 123	255 255 255	255 0 0	255 255 255	255 255 255	255 255 255	255 255 255	255 255 255	255 100 0	255 255 255	255 255 0
255 255 255	255 255 255	255 255 255	255 0 0	255 0 0	255 255 255	255 100 0	255 100 0	255 255 255	255 255 255	255 255 255

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Color Spaces

- How can we represent color?

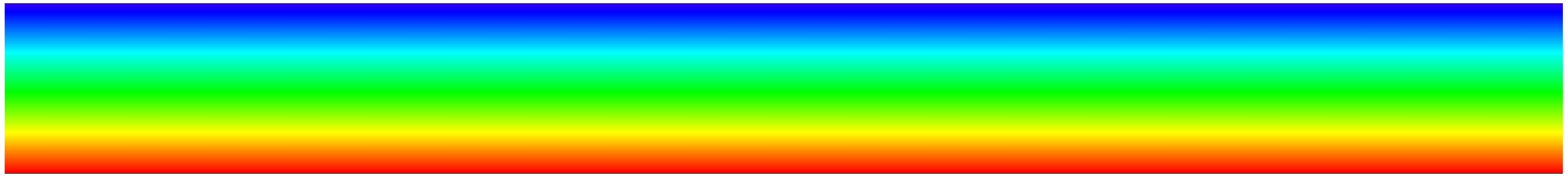


The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

RGB Color Space

- Color representation is usually done with 3 color channels representing Red, Green and Blue wavelengths.

- Typically we use *truecolor* images, which have three 8 bit values for every pixel, used to drive the monitor display.



- Without other information, it's logical to assume that the values have been *gamma corrected*, meaning that a power low transformation has been applied to every value.

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Color Spaces

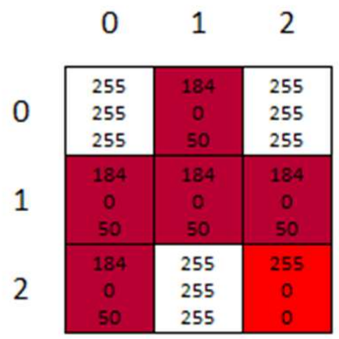
- How do we store color images?
- As mentioned, a digital image is a collection of pixels. To store them in memory we take into account:
  - Number of bytes per pixel (typically 8 bit = 1 byte)
  - Which bytes represent a particular pixel
- In contrast to grayscale images where 1 byte represents each pixel value, an RGB pixel commonly requires 3 bytes. We have two different chances to store the three pixel values:
  - **Interleaved**: Store all the bytes for pixel N before storing the bytes for pixel N + 1
  - **Planar**: Store all the bytes for color C before storing the bytes for color C + 1

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

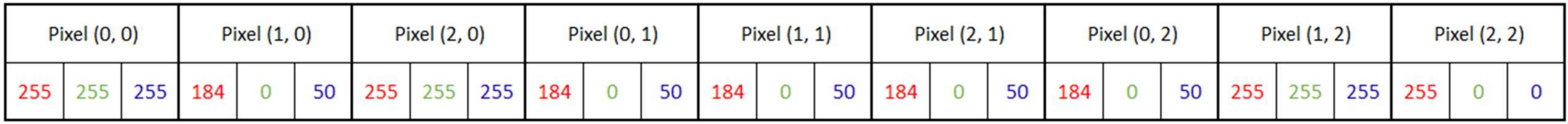
Interleaved Color Spaces

- In the interleaved storage layout, the bytes for a pixel are contiguous. In the 3x3 image on the right, each pixel has RGB components.

- This data would be stored in interleaved, row-major layout like in the diagram below. As you can see, the RGB components for pixel(0, 0) are contiguous. Furthermore, each pixel is still stored in row-major order. This is called interleaved because the data for the colors are mixed in a repeating pattern.



	0	1	2
0	255 255 255	184 0 50	255 255 255
1	184 0 50	184 0 50	184 0 50
2	184 0 50	255 255 255	255 0 0

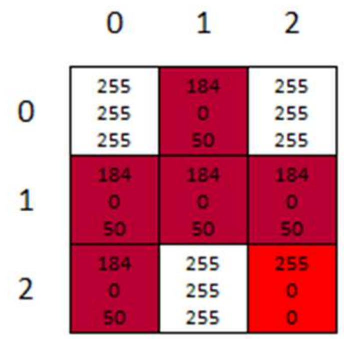


Pixel (0, 0)			Pixel (1, 0)			Pixel (2, 0)			Pixel (0, 1)			Pixel (1, 1)			Pixel (2, 1)			Pixel (0, 2)			Pixel (1, 2)			Pixel (2, 2)		
255	255	255	184	0	50	255	255	255	184	0	50	184	0	50	184	0	50	184	0	50	255	255	255	255	0	0

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Planar Color Spaces

- In the planar storage layout, the bytes for a color, rather than a pixel, are contiguous. For an example we can look at our usual 3×3 image.
- This data would be stored in planar, row-major layout like in the diagram below. As you can see, all the red bytes are stored contiguously in the first 9 elements of the array, all the green bytes are stored contiguously in the next 9 elements, and all the blue bytes are stored contiguously in the last 9 elements. Finally, within each color, the bytes are stored in row-major order.



	0	1	2
0	255 255 255	184 0 50	255 255 255
1	184 0 50	184 0 50	184 0 50
2	184 0 50	255 255 255	255 0 0

(0, 0)	(1, 0)	(2, 0)	(0, 1)	(1, 1)	(2, 1)	(0, 2)	(1, 2)	(2, 2)	(0, 0)	(1, 0)	(2, 0)	(0, 1)	(1, 1)	(2, 1)	(0, 2)	(1, 2)	(2, 2)	(0, 0)	(1, 0)	(2, 0)	(0, 1)	(1, 1)	(2, 1)	(0, 2)	(1, 2)	(2, 2)
255	184	255	184	184	184	184	255	255	255	0	255	0	0	0	0	255	0	255	50	255	50	50	50	50	255	0

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

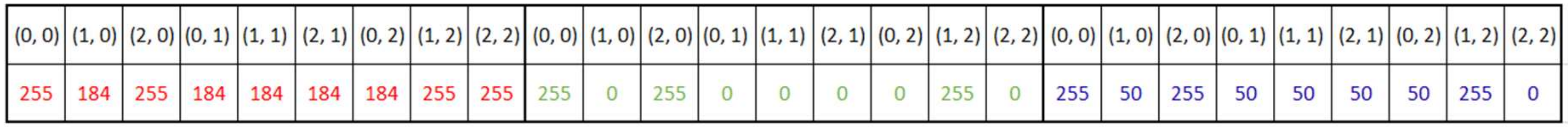
Planar vs Interleaved Spaces

- Modern computers are generally designed to optimize for the common behavior that programs tend to access memory with addresses that are close together. This is known as the principle of locality.
- This is relevant when deciding what memory layout to use for images. The different layouts store the same data.
- Therefore, it's easy to conclude that if the program needs to do a lot of processing for each pixel, interleaved is probably going to be faster. If a program needs to do a lot of processing per color channel, planar is probably going to be faster.
- Performance is typically the main reason to choose planar vs. interleaved layout.

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

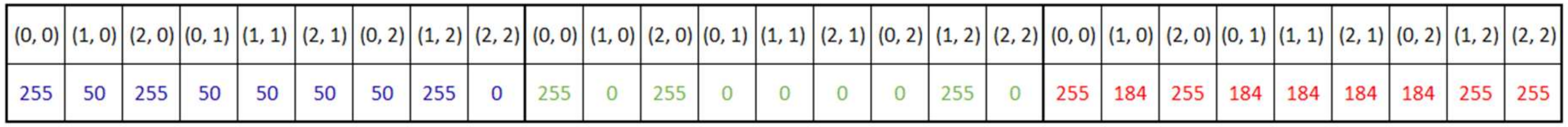
Color Spaces Order

- Actually, we have other configuration options. Indeed, both the Interleaved and Planar layout allows different order than RGB:
  - Planar RGB:



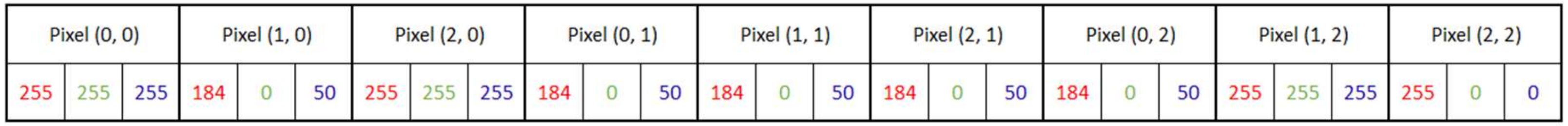
(0, 0)	(1, 0)	(2, 0)	(0, 1)	(1, 1)	(2, 1)	(0, 2)	(1, 2)	(2, 2)	(0, 0)	(1, 0)	(2, 0)	(0, 1)	(1, 1)	(2, 1)	(0, 2)	(1, 2)	(2, 2)	(0, 0)	(1, 0)	(2, 0)	(0, 1)	(1, 1)	(2, 1)	(0, 2)	(1, 2)	(2, 2)
255	184	255	184	184	184	184	255	255	255	0	255	0	0	0	0	255	0	255	50	255	50	50	50	50	255	0

  - Planar BGR:



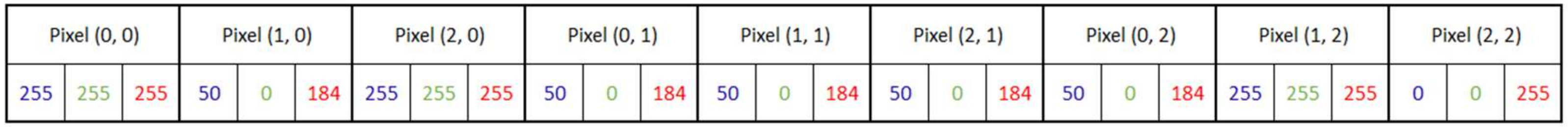
(0, 0)	(1, 0)	(2, 0)	(0, 1)	(1, 1)	(2, 1)	(0, 2)	(1, 2)	(2, 2)	(0, 0)	(1, 0)	(2, 0)	(0, 1)	(1, 1)	(2, 1)	(0, 2)	(1, 2)	(2, 2)	(0, 0)	(1, 0)	(2, 0)	(0, 1)	(1, 1)	(2, 1)	(0, 2)	(1, 2)	(2, 2)
255	50	255	50	50	50	50	255	0	255	0	255	0	0	0	0	255	0	255	184	255	184	184	184	184	255	255

  - Interleaved RGB:



Pixel (0, 0)			Pixel (1, 0)			Pixel (2, 0)			Pixel (0, 1)			Pixel (1, 1)			Pixel (2, 1)			Pixel (0, 2)			Pixel (1, 2)			Pixel (2, 2)		
255	255	255	184	0	50	255	255	255	184	0	50	184	0	50	184	0	50	184	0	50	255	255	255	255	0	0

  - Interleaved BGR:

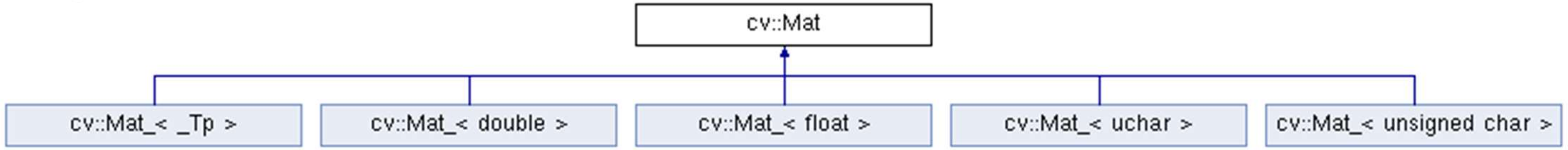


Pixel (0, 0)			Pixel (1, 0)			Pixel (2, 0)			Pixel (0, 1)			Pixel (1, 1)			Pixel (2, 1)			Pixel (0, 2)			Pixel (1, 2)			Pixel (2, 2)		
255	255	255	50	0	184	255	255	255	50	0	184	50	0	184	50	0	184	50	0	184	255	255	255	0	0	255

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Existing CV Libraries

cv::Mat is the object that stores images with BGR interleaved layout, i.e., *BGRBGRBGRBGRBGR…*



PIL Image is the object that stores images by default with the RGB interleaved layout, i.e., *RGBRGBRGBRGBRGB…*

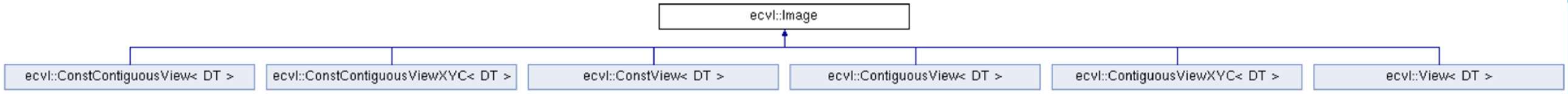
ImageJ allows both interleaved and planar RGB layouts

…

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

What about ECVL?

- ECVL has been designed to hold different kind of images with diverse channels configurations: BGR and RGB byte order are both supported in the interleaved and planar layouts.
- The ECVL object responsible for storing data is the ECVL::Image.
- ECVL::Image.channels contains the information about the current layout.


ecvl::Image
ecvl::ConstContiguousView< DT >
ecvl::ConstContiguousViewXYC< DT >
ecvl::ConstView< DT >
ecvl::ContiguousView< DT >
ecvl::ContiguousViewXYC< DT >
ecvl::View< DT >

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

DEEPHEALTH

But, what is ECVL?

European Computer Vision Library

- C++ open-source library designed to facilitate the integration and exchange of data between existing Computer Vision (CV) and image processing libraries.
- It implements new high-level Computer Vision functionalities commonly used in combination with Deep Learning (DL) algorithms.
- API for C++ and Python

Key benefits

- Support for multiple operating systems (Windows, Unix, and MacOS)
- Hardware transparency support for CPU, GPU and FPGA
- Support for Clusters, Clouds and containerized platforms
- Compliant with existing CV libraries and EDDL

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Reading & Writing Images

- Multiple types of scientific (and non) imaging data and data formats are supported:
  - Windows bitmaps - *.bmp, *.dib (always supported)
  - JPEG files - *.jpeg, *.jpg, *.jpe (see the Note section)
  - JPEG 2000 files - *.jp2 (see the Note section)
  - Portable Network Graphics - *.png (see the Note section)
  - WebP - *.webp (see the Note section)
  - Portable image format - *.pbm, *.pgm, *.ppm *.pxm, *.pnm (always supported)
  - PFM files - *.pfm (see the Note section)
  - Sun rasters - *.sr, *.ras (always supported)
  - TIFF files - *.tiff, *.tif (see the Note section)
  - OpenEXR Image files - *.exr (see the Note section)
  - Radiance HDR - *.hdr, *.pic (always supported)
  - Raster and Vector geospatial data supported by GDAL (see the Note section)

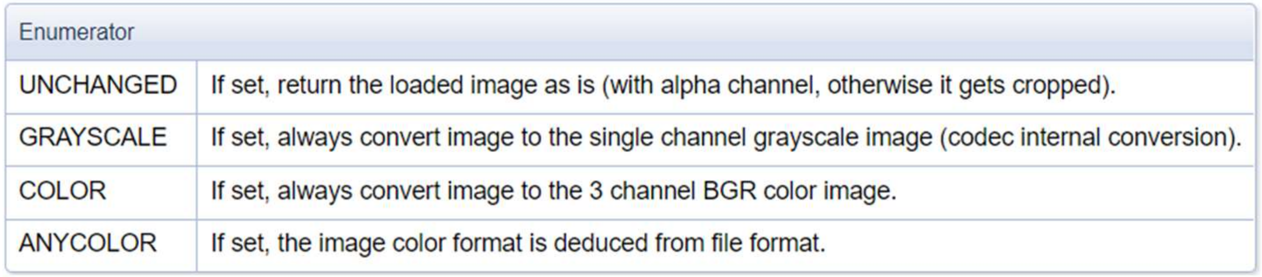
The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Reading & Writing Images

```
import pyecvl.ecvl as ecvl  # library import

image = ecvl.ImRead("/data/lena.png", ecvl.ImReadMode.GRAYSCALE)
```

source image path

optional flags, default is ecvl.ImReadMode.ANYCOLOR



Enumerator	
UNCHANGED	If set, return the loaded image as is (with alpha channel, otherwise it gets cropped).
GRAYSCALE	If set, always convert image to the single channel grayscale image (codec internal conversion).
COLOR	If set, always convert image to the 3 channel BGR color image.
ANYCOLOR	If set, the image color format is deduced from file format.

```
ecvl.ImWrite("/data/lena_2.png", image)
```

destination image path

Image object

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Visualizing Images with PyCharm

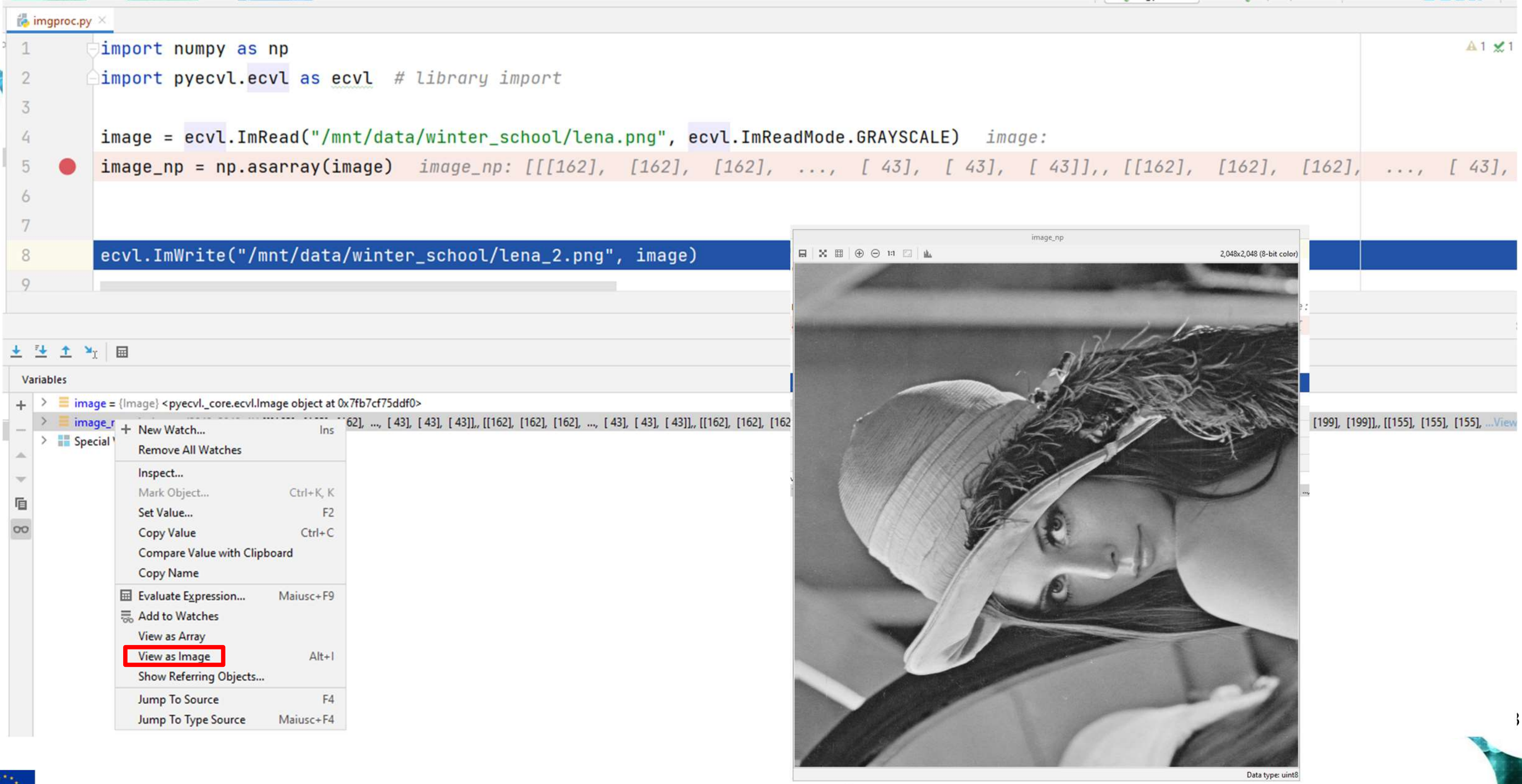
- With Python, ECVL Image object can be wrapped into a NumPy* array.

```
import numpy as np
import pyecvl.ecvl as ecvl

image = ecvl.ImRead("/data/lena.png", ecvl.ImReadMode.GRAYSCALE)
image_np = np.asarray(image)
```

- The *OpenCV Image Viewer* PyCharm IDE plugin allows you to display Image data without writing them into file during debug
- *File → Settings → Plugins* is the menu to install the plugin

*NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.



```
import numpy as np
import pyecvl.ecvl as ecvl  # library import

image = ecvl.ImRead("/mnt/data/winter_school/lena.png", ecvl.ImReadMode.GRAYSCALE)   image:
image_np = np.asarray(image)   image_np: [[[162],  [162],  [162],  ...,  [ 43],  [ 43],  [ 43]],, [[162],  [162],  [162],  ...,  [ 43],


ecvl.ImWrite("/mnt/data/winter_school/lena_2.png", image)
```

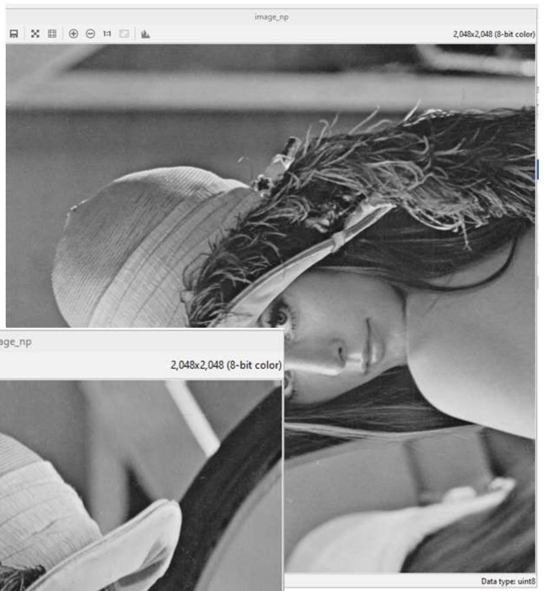
```
image = {Image} <pyecvl._core.ecvl.Image object at 0x7fb7cf75ddf0>
image_n ... 62], ..., [ 43], [ 43], [ 43]],, [[162], [162], [162], ..., [ 43], [ 43], [ 43]],, [[162], [162], [162 ... [199], [199]],, [[155], [155], [155], ...View
Special V
```

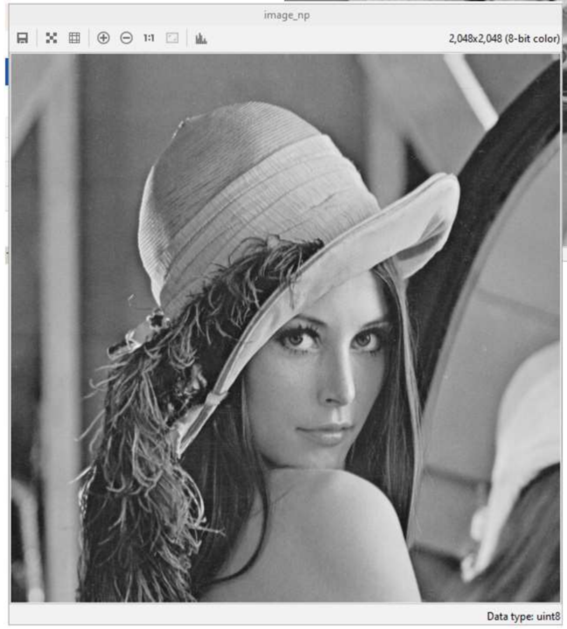
The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Why is the Image Rotated?

- By default, ECVL use “xyc” layout, meaning that pixels are stored inside the ecvl.Image row by row, with a planar layout
- But, the plugin assumes data to be in “yxc” layout
- With ecvl.RearrangeChannels we can change the default order

```
import numpy as np
import pyecvl.ecvl as ecvl

image = ecvl.ImRead("data/lena.png")
disp_image = ecvl.Image.empty()
ecvl.RearrangeChannels(image, disp_image, "yxc")
image_np = np.asarray(disp_image)
```



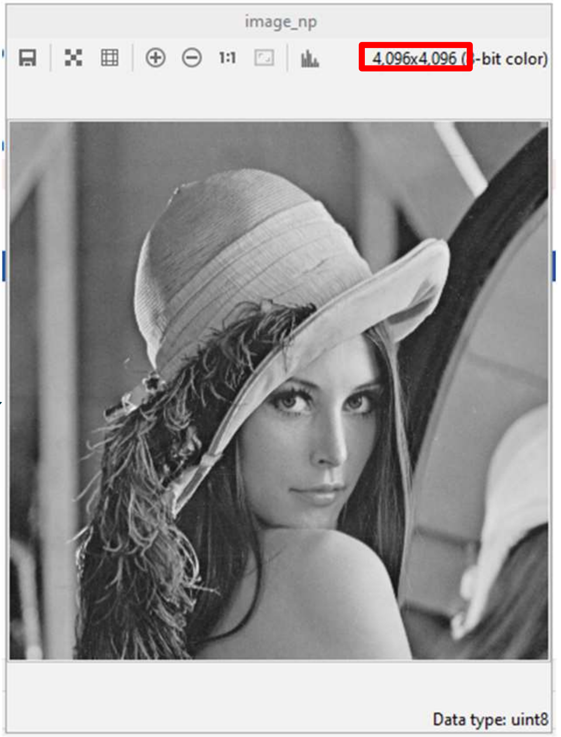


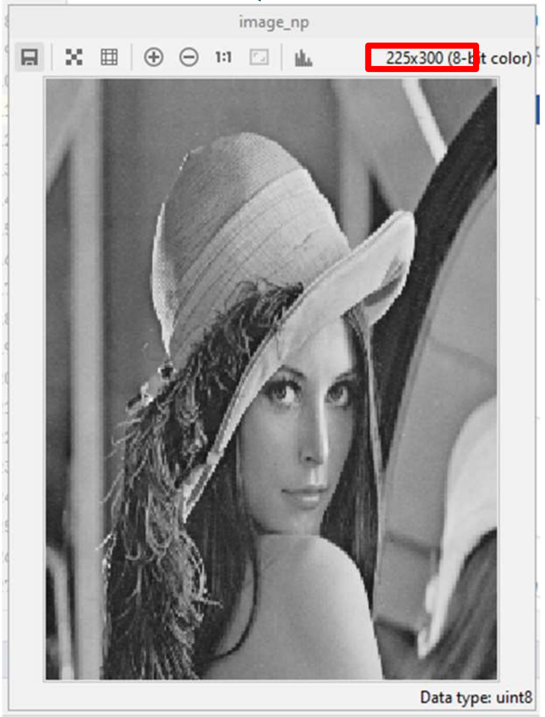
The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Resizing

```python
import numpy as np
import pyecvl.ecvl as ecvl

image = ecvl.ImRead("data/lena.png", ecvl.ImReadMode.GRAYSCALE)
disp_image = ecvl.Image.empty()
ecvl.RearrangeChannels(image, image, "yxc")

ecvl.ResizeDim(image, disp_image, [225, 300],
               ecvl.InterpolationType.nearest)
image_np = np.asarray(disp_image)

ecvl.ResizeScale(image, disp_image, [2, 2],
                 ecvl.InterpolationType.cubic)
image_np = np.asarray(disp_image)
```

Enumerator	
nearest	Nearest neighbor interpolation
linear	Bilinear interpolation
area	Resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire-free results. But when the image is zoomed, it is similar to the nearest method.
cubic	Bicubic interpolation
lanczos4	Lanczos interpolation over 8x8 neighborhood



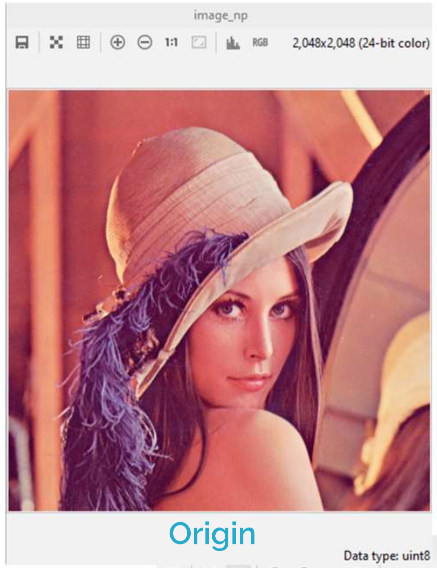


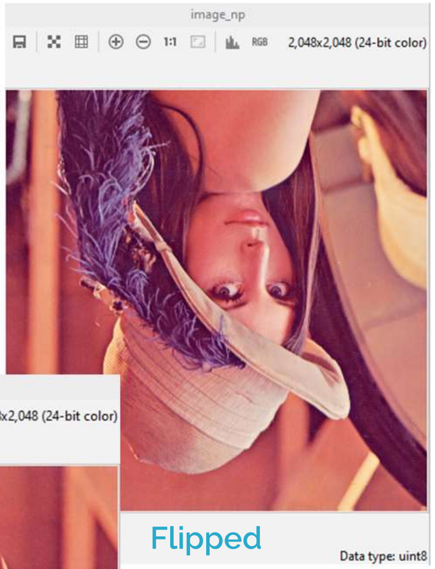
The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

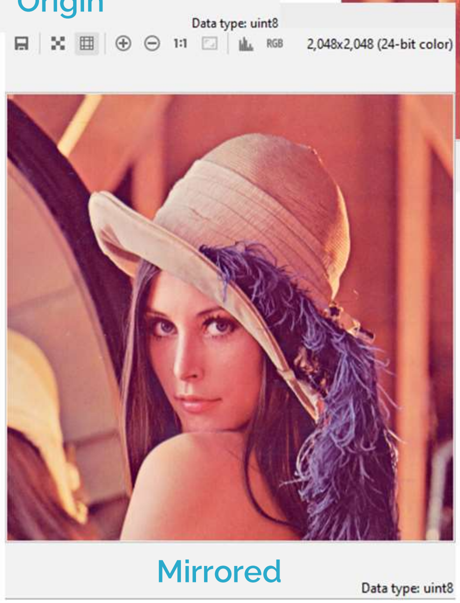
Flip & Mirror

(with colors)

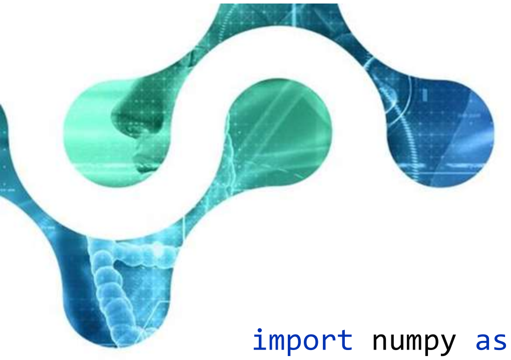
```
import numpy as np
import pyecvl.ecvl as ecvl

image = ecvl.ImRead(“data/lena.png")
disp_image = ecvl.Image.empty()
ecvl.RearrangeChannels(image, image, "yxc")

image_np = np.asarray(image)  # To display the original image

ecvl.Flip2D(image, disp_image)
image_np = np.asarray(disp_image)

ecvl.Mirror2D(image, disp_image)
image_np = np.asarray(disp_image)
```


image_np
2,048x2,048 (24-bit color)
Origin
Data type: uint8


image_np
2,048x2,048 (24-bit color)
Flipped
Data type: uint8

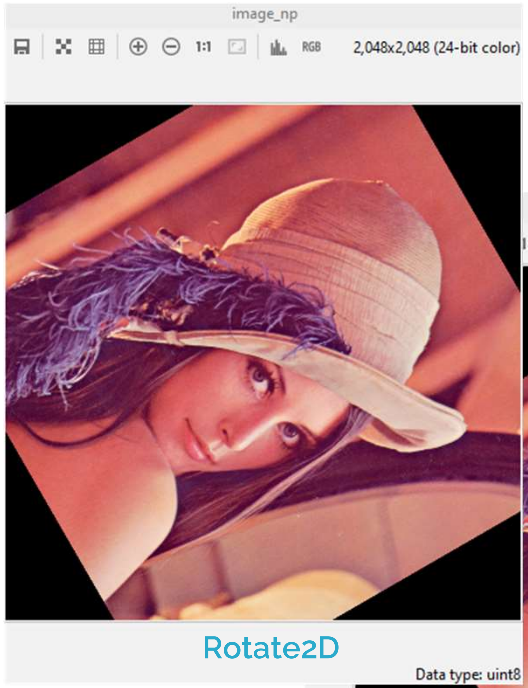

2,048x2,048 (24-bit color)
Mirrored
Data type: uint8

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

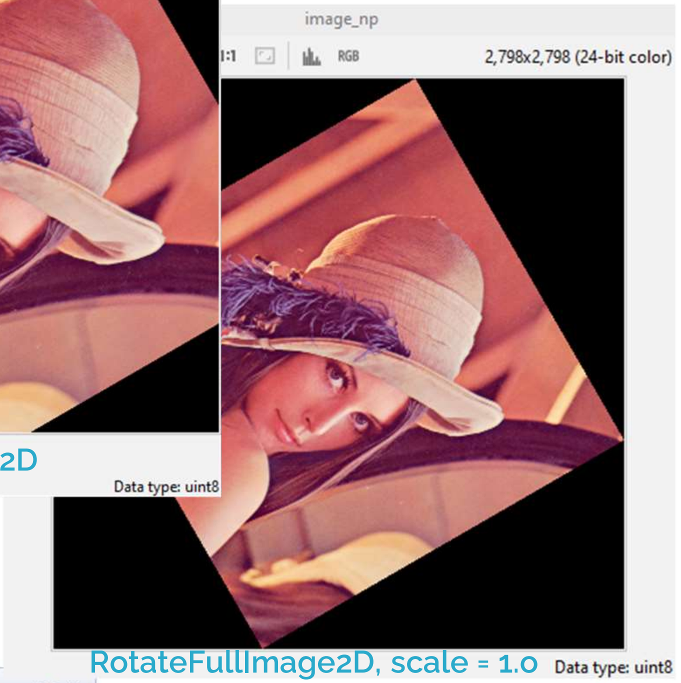


Rotation

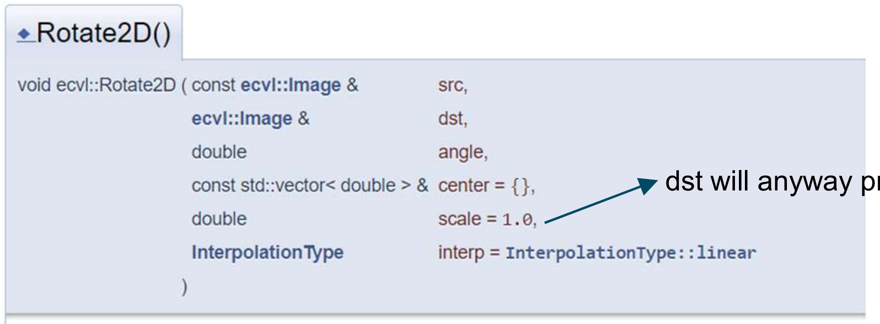
```
import numpy as np
import pyecvl.ecvl as ecvl

image = ecvl.ImRead("data/lena.png")
disp_image = ecvl.Image.empty()
ecvl.RearrangeChannels(image, image, "yxc")

ecvl.Rotate2D(image, disp_image, 60)
image_np = np.asarray(disp_image)

ecvl.RotateFullImage2D(image, disp_image, 60)
image_np = np.asarray(disp_image)
```



Rotate2D



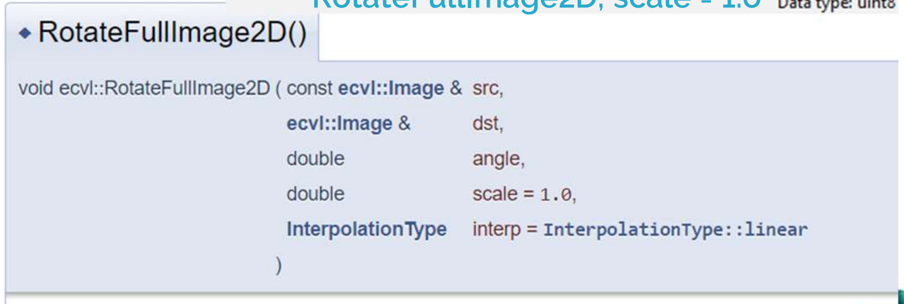
RotateFullImage2D, scale = 1.0



◆ Rotate2D()

```
void ecvl::Rotate2D ( const ecvl::Image &              src,
                      ecvl::Image &                    dst,
                      double                           angle,
                      const std::vector< double > &    center = {},
                      double                           scale = 1.0,
                      InterpolationType                interp = InterpolationType::linear
                    )
```

dst will anyway preserve the input size



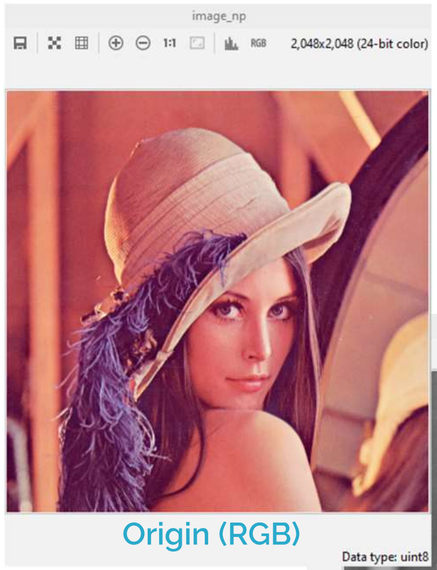
◆ RotateFullImage2D()

```
void ecvl::RotateFullImage2D ( const ecvl::Image &   src,
                               ecvl::Image &         dst,
                               double                angle,
                               double                scale = 1.0,
                               InterpolationType     interp = InterpolationType::linear
                             )
```

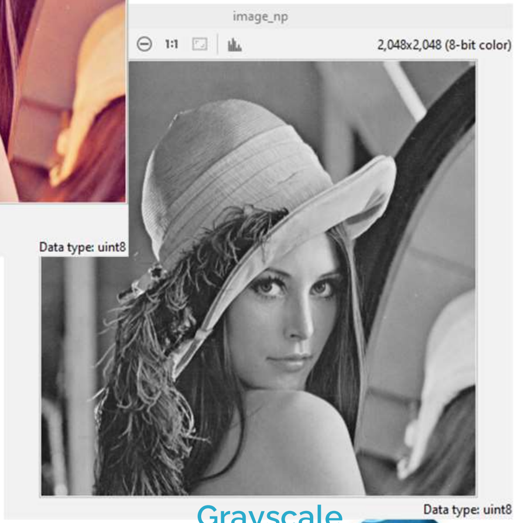
The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Changing the Color Space

```
import numpy as np
import pyecvl.ecvl as ecvl

image = ecvl.ImRead(“data/lena.png")
disp_image = ecvl.Image.empty()
ecvl.RearrangeChannels(image, image, "yxc")

image_np = np.asarray(image)  # To display the original image

ecvl.ChangeColorSpace(image, disp_image, ecvl.ColorType.GRAY)
image_np = np.asarray(disp_image)
```



Origin (RGB)

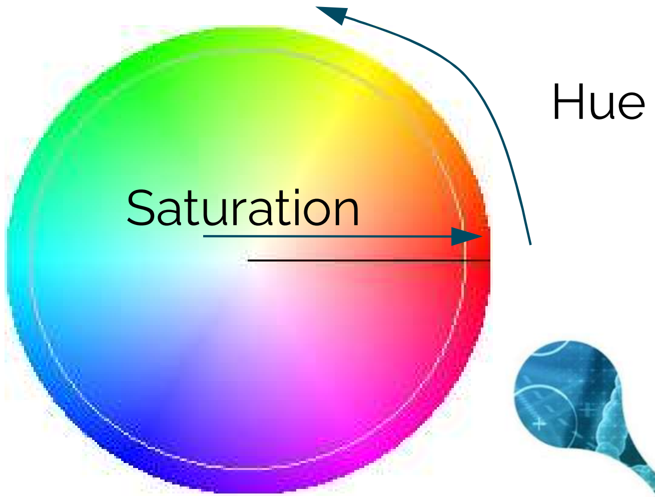


Grayscale

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Graphics Oriented Color Spaces

- In addition to the RGB standard, other color spaces used to introduce some kind of numerical specification of color.
- This kind of transformation is useful when dealing with an interface with the human operator.
- These color spaces have no claim of accuracy and are obtained as a transformation from an undefined RGB color space, contrary to other professional colorimetry standards.
- Their representation is based on the concept of luminance and chrominance (A.H. Munsell)
- All characterized by two basic concepts:
  - H = Hue
  - S = Saturation



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

HSV Color Space

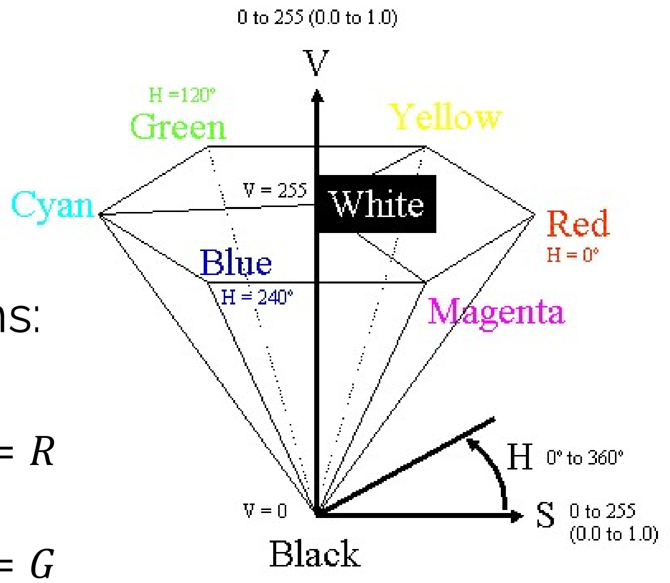
- **HSV** (hue, saturation, value): H is an angle between 0 and 360 degrees, S and V are values in the range [0,1]. This is a transformation of the RGB color space with $0 \le R,G,B \le 1$.
- For every pixel we define:

$$Max = \max(R,G,B)$$
$$Min = \min(R,G,B)$$

- The coordinates are given by the following equations:

$$V = Max$$
$$S = \frac{(Max - Min)}{Max}$$

$$H = \frac{\pi}{3}\begin{cases} \dfrac{G-B}{(Max-Min)} & Max = R \\ 2 + \dfrac{B-R}{(Max-Min)} & Max = G \\ 4 + \dfrac{R-G}{(Max-Min)} & Max = B \end{cases}$$



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

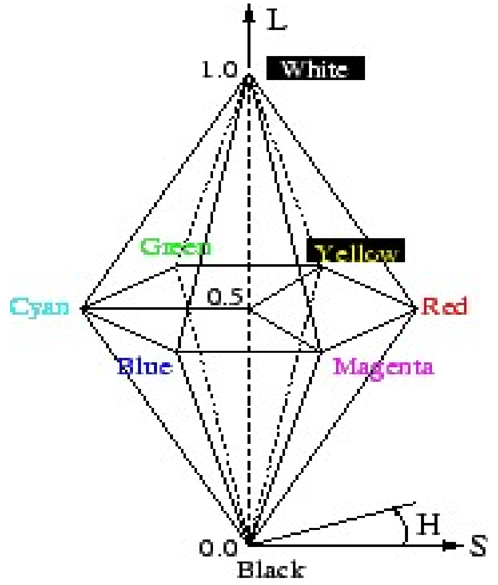
HSL Color Space

- It's just a variation of the HSV color system where "value" is substituted with "lightness":

$$L = \frac{Max + Min}{2}$$

- We need to change the definition of S accordingly:

$$S = \begin{cases} \dfrac{Max - Min}{Max + Min} & L \leq 0.5 \\ \dfrac{Max - Min}{2 - (Max + Min)} & L > 0.5 \end{cases}$$



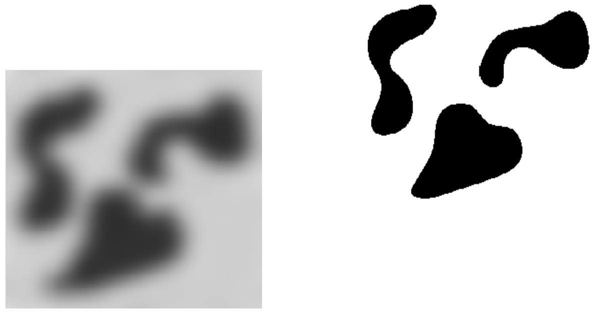
- The pyramidal structure of HSV is doubled as shown here.
- This is the color system used in the standard color selection dialog box on Windows systems.

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Thresholding

- **Thresholding** consists in the selection of a value T of brightness (intensity) capable of dividing the image into 2 regions of pixels with intensity greater or less than T .
- It is an operator which transforms a grey level image into a binary image (thresholding-based binarization)
- Given an image I(x), with x=(i,j), it is transformed to O(x)
- $O(x) = Thresh(I(x), T)$

```
if (I(i,j) >= T)
      O(i,j) = 1;
else
      O(i,j) = 0;
```



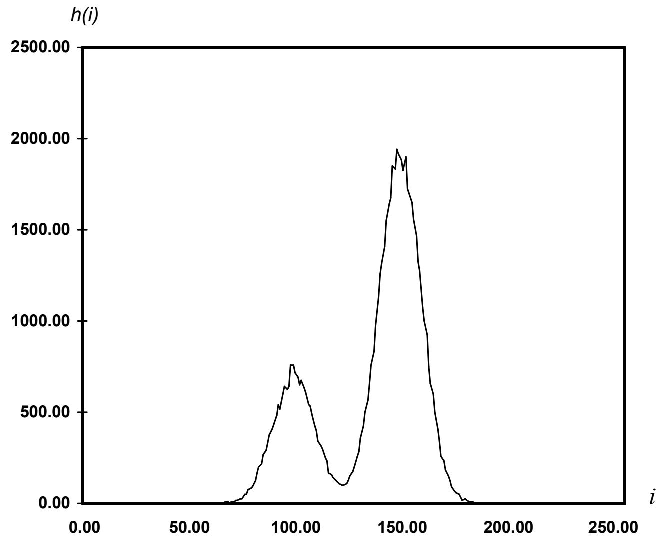
- **Global threshold** if $T = K$ or $T = f(I)$
- **Adaptive threshold** if $T = f(I, \mathrm{x})$, i.e., if it depends on a window $W(\mathrm{x})$ around the current position.

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Automatic Thresholding

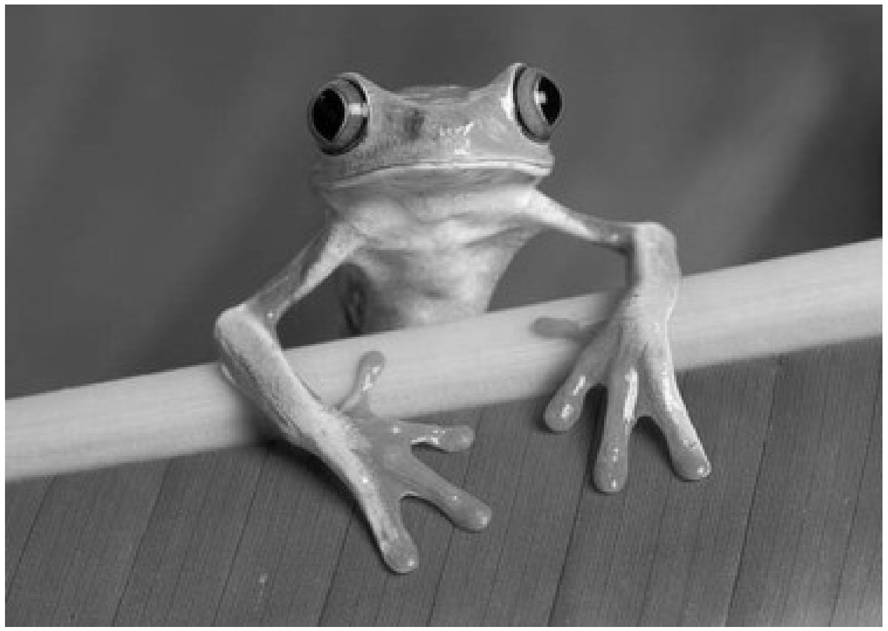
- How to define T automatically?:
- Without knowing the objects of interest let's give the computer the chance of seeing, measuring the statistic proprieties of the histogram
- Hp: the points of the target object have a grey level different from the background gray level (**bimodal histogram**)

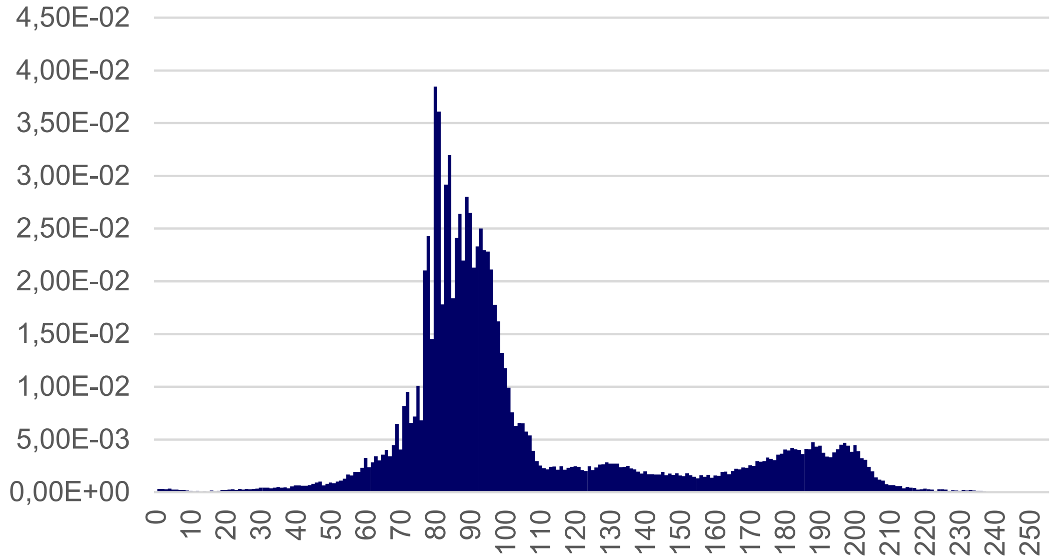
➔ T must divide the two modes



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Example



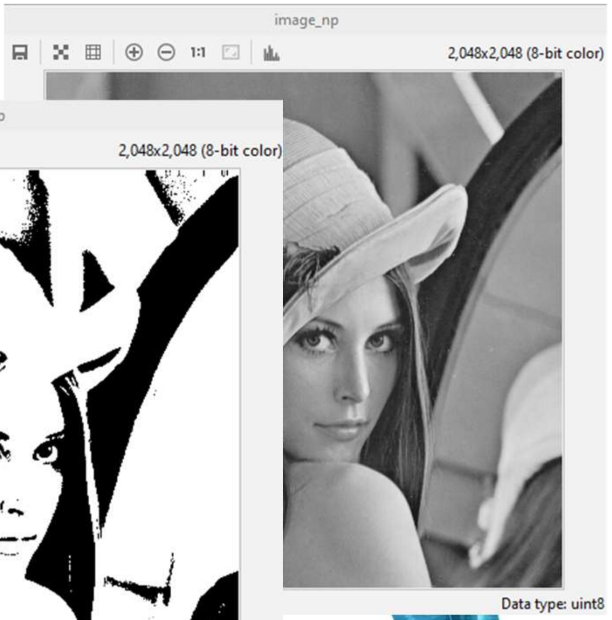


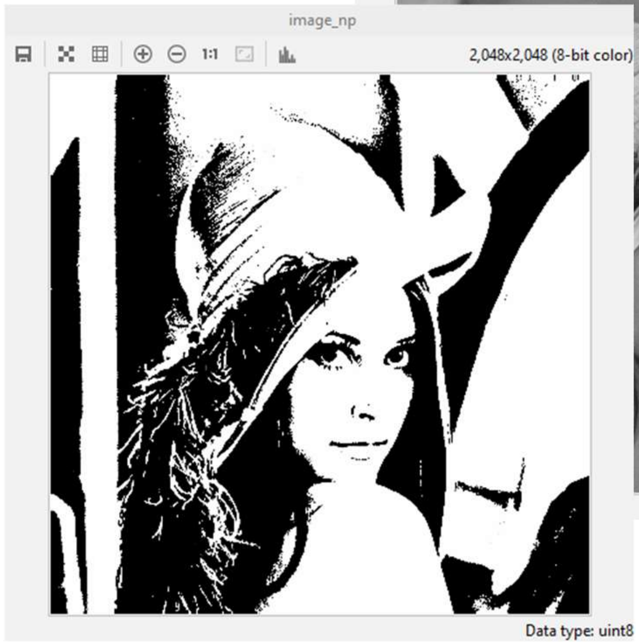
The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Otsu Thresholding

- The algorithm returns a single intensity threshold that separate pixels into two classes, foreground and background.
- This threshold is determined by minimizing intra-class intensity variance, or equivalently, by maximizing inter-class variance.

```
import numpy as np
import pyecvl.ecvl as ecvl

image = ecvl.ImRead("data/lena.png")
disp_image = ecvl.Image.empty()
ecvl.RearrangeChannels(image, image, "yxc")

# The input image must be grayscale
ecvl.ChangeColorSpace(image, disp_image, ecvl.ColorType.GRAY)
image_np = np.asarray(disp_image)

# Calculate threshold that seperates the two classes …
thr = ecvl.OtsuThreshold(disp_image)
# … and binarize the image
ecvl.Threshold(disp_image, disp_image, thr, 255)
image_np = np.asarray(disp_image)
```



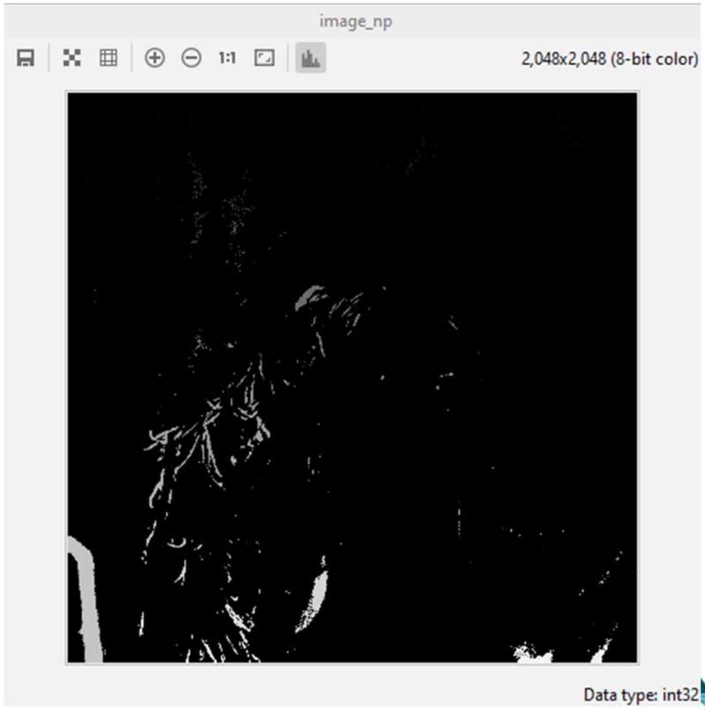


The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Labeling Connected Components

- Connected components labeling consists in the creation of a labeled (symbolic) image in which the positions associated with the same connected component of the binary input image have a unique label.

```
import numpy as np
import pyecvl.ecvl as ecvl

image = ecvl.ImRead("data/lena.png")
disp_image = ecvl.Image.empty()

# First of all, we need a binary image
ecvl.ChangeColorSpace(image, disp_image, ecvl.ColorType.GRAY)
thr = ecvl.OtsuThreshold(disp_image)
ecvl.Threshold(disp_image, disp_image, thr, 255)

# Then we can apply CCL. n_obj is the number of blobs (770 in the
# Lena example) in the original image, labels is the output
# symbolic image.
labels = ecvl.Image.empty()
n_obj = ecvl.ConnectedComponentsLabeling(disp_image, labels)

ecvl.RearrangeChannels(labels, labels, "yxc")
image_np = np.asarray(labels)
```

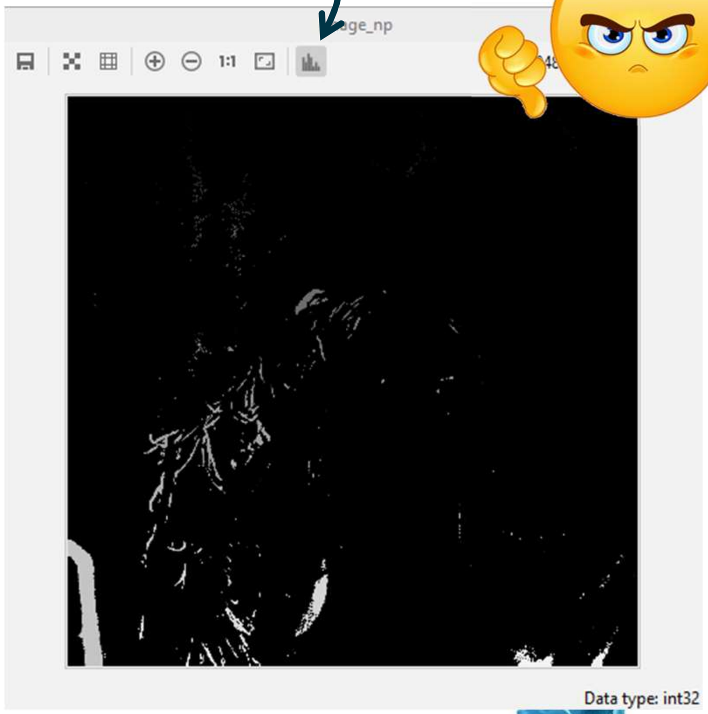


The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Labeling Connected Components

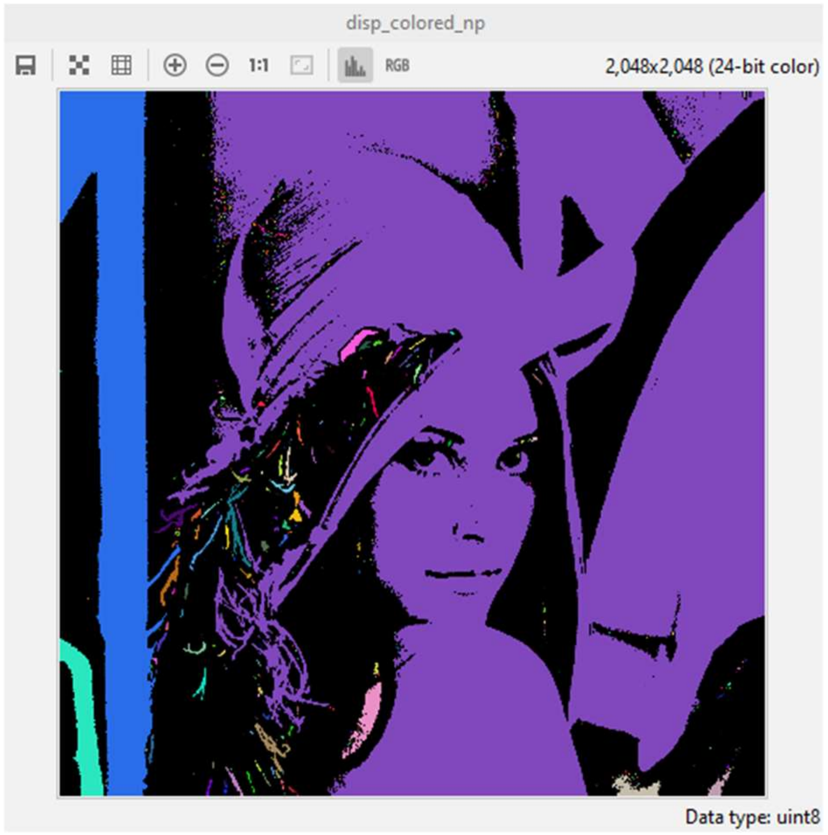
- The output symbolic image is an int32 image (32 bit per pixel). To display it with the view we have to normalize it
- Well, the result is pretty bad. Let's try to assign different color at each label.

```
labels_np = np.array(labels)  # Move to numpy, this time copying the data

r_plane_np = labels_np * 43 % 255
r_plane = ecvl.Image.fromarray(r_plane_np, channels="xyc", colortype=ecvl.ColorType.none)

g_plane_np = labels_np * 109 % 255
g_plane = ecvl.Image.fromarray(g_plane_np, channels="xyc", colortype=ecvl.ColorType.none)

b_plane_np = labels_np * 743 % 255
b_plane = ecvl.Image.fromarray(b_plane_np, channels="xyc", colortype=ecvl.ColorType.none)

colored = ecvl.Image.empty()
ecvl.Stack([b_plane, g_plane, r_plane], colored)  # Always use "o" when stacking channels

disp_colored = ecvl.Image.empty()
ecvl.RearrangeChannels(colored, disp_colored, "yxo", new_type=ecvl.DataType.uint8)
disp_colored_np = np.asarray(disp_colored)
```



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

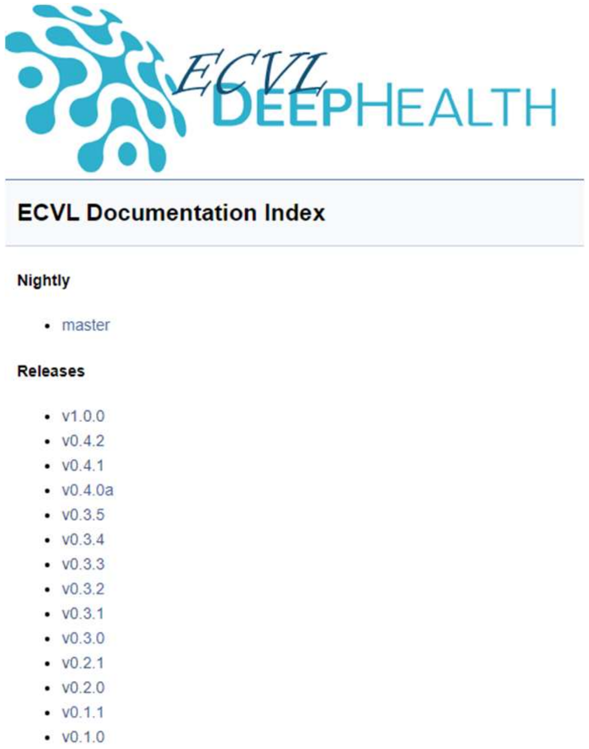
Labeling Connected Components



The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Documentation

ECVL & PyECVL



ECVL DEEPHEALTH

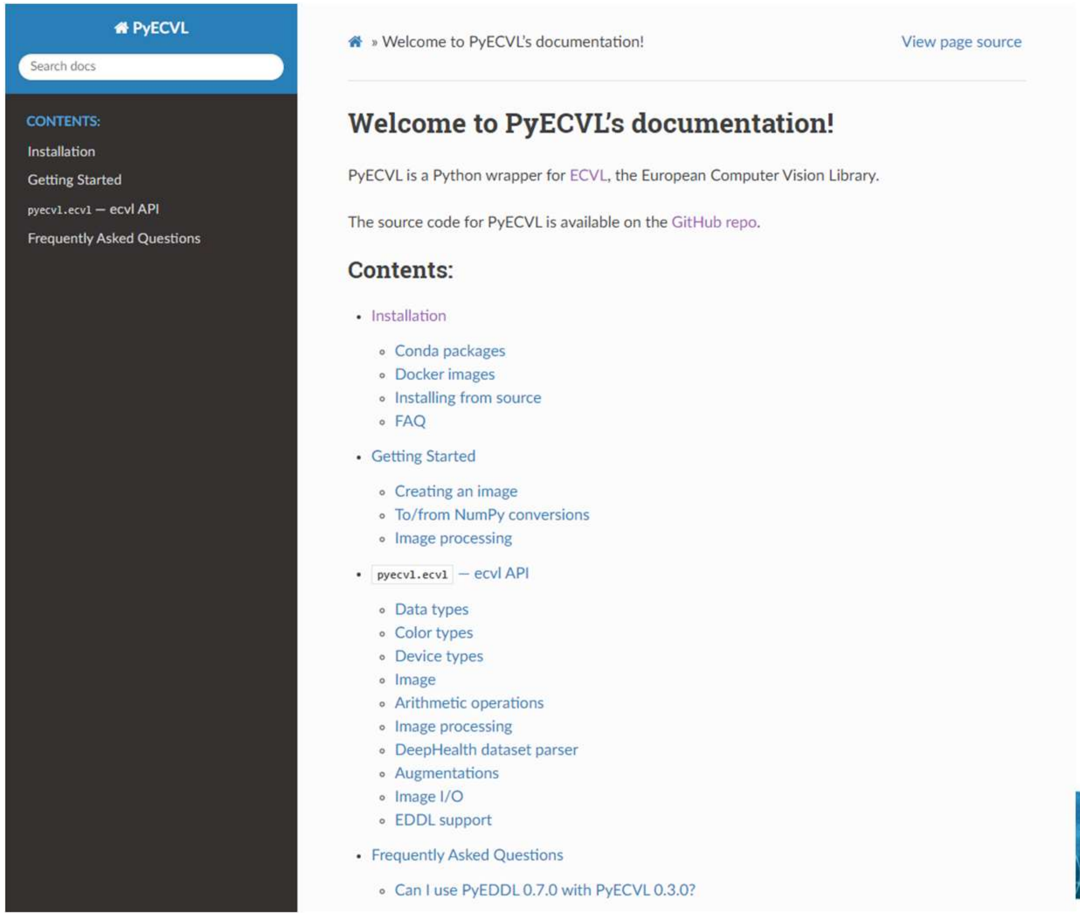
ECVL Documentation Index

Nightly

- master

Releases

- v1.0.0
- v0.4.2
- v0.4.1
- v0.4.0a
- v0.3.5
- v0.3.4
- v0.3.3
- v0.3.2
- v0.3.1
- v0.3.0
- v0.2.1
- v0.2.0
- v0.1.1
- v0.1.0



Welcome to PyECVL's documentation!

PyECVL is a Python wrapper for ECVL, the European Computer Vision Library.

The source code for PyECVL is available on the GitHub repo.

Contents:

- Installation
  - Conda packages
  - Docker images
  - Installing from source
  - FAQ
- Getting Started
  - Creating an image
  - To/from NumPy conversions
  - Image processing
- `pyecvl.ecvl` — ecvl API
  - Data types
  - Color types
  - Device types
  - Image
  - Arithmetic operations
  - Image processing
  - DeepHealth dataset parser
  - Augmentations
  - Image I/O
  - EDDL support
- Frequently Asked Questions
  - Can I use PyEDDL 0.7.0 with PyECVL 0.3.0?

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Resources

Additional links with useful information

ECVL documentation

https://deephealthproject.github.io/ecvl/

GitHub

https://github.com/deephealthproject/ecvl

Python Wrapper (PyECVL)

https://github.com/deephealthproject/pyecvl

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

Conclusion

- ECVL is an open-source C++ library specifically designed for medical imaging, but with general purpose paradigms in mind.
- We need your help to spot bugs and find missing important features thus ensuring the future of the library.
- Also much more popular libraries (e.g. OpenCV, ~50k stars on GitHub) are continuously updated with new feature and bug solved. The difference? ECVL is less than 2-years old library while the OpenCV development started in the far away 2000 from an Intel project.

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.

DEEPHEALTH

Thank you!

Costantino Grana
costantino.grana@unimore.it

The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825111.