D1.3 API specifications for EDDLL and ECVL libraries

Project ref. no.	H2020-ICT-11-2018-2019 GA No. 825111
Project title	Deep-Learning and HPC to Boost Biomedical Applications for Health
Duration of the project	1-01-2019 – 31-12-2021 (36 months)
WP/Task:	WP1/ T1.5, T1.6
Dissemination level:	PUBLIC
Document due Date:	30/06/2019 (M6)
Actual date of delivery:	30/06/2019 (M6)
Leader of this deliverable	UPV
Author(s)	Roberto Paredes (UPV), Salvador Carrión (UPV), Mario Parreño (UPV), Federico Bolelli (UNIMORE), Costantino Grana (UNIMORE) and Jon Ander Gómez (UPV) Reviewed by Aimilia Bantouna (WINGS) and Monica Caballero (EVERIS)
Version	1.0

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 82511

Document history

Version	Date	Document history/approvals
0.1	14/06/2019	First draft contents
0.2	24/06/2019	EDDLL part for review
0.3	25/06/2019	ECVL part integrated for review and format adaptation
0.4	26/06/2019	ECVL part upated. ECVL documentation added as an appendix
0.5	28/06/2019	Corrections made from the review done by WINGS as peer-reviewer
0.6	28/06/2019	Corrections made from the review done by Monica Caballero as project manager
1.0	30/06/2019	Definitive

Table of contents

Executive summary

In this document we describe in full detail the Application Programming Interface (API) for the two core libraries which are being developed in the DeepHealth project: the European Distributed Deep Learning Library (EDDLL) and the European Computer Vision Library (ECVL). The definition of the APIs for the EDDLL and ECVL has been carried out in tasks T1.5 and T1.6 respectively, by taking into account the needs arised in tasks T1.1, T1.2 and T1.4.

The functions included in the APIs are the most commonly used in other packages taken as reference, paying special attention to the functions that are necessary in any of the 14 use cases and/or required for the adaptation of each software platform to one or more use cases, as it is described in D1.1.

We would like to highlight that the definition of the APIs is presented as a reference manual of all the functions foreseen to be included so far. This is the most appropriate way for being used by both developers of the libraries and the programmers who will integrate these libraries in other software applications. Hence, the API documentation to be included as part of future deliverables D2.1 and D3.1 will be very close to the content of this deliverable. That is because the API documentation is part of the EDDLL and will appear as one section of D2.1, analogously, the API documentation of the ECVL will appear as one section of D3.1.

1 INTRODUCTION

The DeepHealth framework is based on two new core technology libraries: the European Distributed Deep Learning Library (EDDLL) and the European Computer Vision Library (ECVL). One of the goal of these libraries is to take advantage of the current and coming development of HPC systems deployed over Europe, providing a transparent use of heterogeneous hardware accelerators to optimize the training of predictive models, while considering performance and accuracy trade-offs.

Both EDDLL and ECVL will be available for multiple languages development (*e.g.* C++, Python). Anyway, in order to guarantee optimal performance, the libraries core will be implemented in C++. The aim of this document is to describe the libraries' API, and to present the philosophy at their base. Some examples of C++ and Python APIs will be provided, in order to ease the reading.

In both cases, the API definitions are technical documents presented as reference manuals of the functions to be used by programmers with the proper explanations of the data structures when needed. Some examples are also included for both facilitating the comprehension of the reader and making easy to run the first examples in few steps.

The DeepHelth toolkit is not yet created, at least following our work plan. Note current tasks were only to define the API not to implement it. When ready, it will be available in a public GitHub repository in order to promote its use in any application domain.

1.1 EDDL Library

In order to define the API for the EDDL library, we have studied different deep learning toolkits and libraries as well as common medical use cases where deep-learning is typically applied. Always trying to cover as many main functionalities as the most popular alternatives currently have.

Some of these toolkits used as reference are:

- TensorFlow (https://www.tensorflow.org).
- PyTorch (https://pytorch.org).
- Keras (https://keras.io) A high level wrapper for common machine learning frameworks.

Our aim si to provide the user with an API very similar to Keras in order to ease the learning curve, but with the potential to work with low-level features in a simple way. There is an object-oriented programming class by layer type, so each class has its own set of parameters to create layers and connect them with other layers. Once all the layers for a model have been defined and properly connected, then, we can say the topology has been designed and the following parameters must be provided to build the model:

- An optimizer
- A loss function to be optimized
- A metric to evaluate the performance
- A computing service where the model will be deployed – The computing service is part of the distributed version of the EDDL library and will be detailed in another deliverable. This depends on the adaptation to HPC and Big Data environments –

Once the model has been build, it can be trained by giving some inputs and the corresponding outputs. The provided examples illustrate this.

In the sections 2 to 9, we define the high-level application programming interface for the EDDL library, both in C++ (sections 2–5) and Python (sections 6–9), with which to build a fully working neural network.

It can be observed how the way of programming is practically identical in both languages, C++ and Python, and the way in which a model is created is exactly the same. A model in this context is a neural network with the topology designed by the user that is ready to be trained. And, if it was already trained, the network weights and topology can be loaded from a file on disk and then the model is ready to be used for predicting.

OPEN NEURAL NETWORK EXCHANGE FORMAT (ONNX) (https://onnx.ai) is the format adopted in DeepHealth to save, load and share neural network models. ONNX will be used for sharing models with other deep learning toolkits, in such a way that one can train the model by using this library in a distributed environment and, then, export it for its use in a software platform based on another deep learning library. Or vice versa, the model can be trained by using an existing deep learning toolkit and then loaded in a software platform where the EDDLL is integrated.

The details of the ONNX format are not part of this deliverable. ONNX is described at https://onnx.ai. The functions for saving and loading models will use this format, there are no plans to use other formats.

Additionally, the ONNX format will be also used to serialize models (the weights and the topology of networks) for sending the updated weights from the main program to the worker nodes in a distributed environment, and for reporting the gradients from worker nodes to the main program.

1.2 ECV Library

The main goal of the European Computer Vision Library (ECVL) is to facilitate the integration and exchange of data between existing state-of-the-art Computer Vision (CV) and Image Processing libraries, also providing new high-level Computer Vision functionalities thanks to specialized/accelerated versions of some CV algorithms commonly employed in conjunction with DL algorithms. The algorithms of ECVL will also be adapted to hardware accelerators.

The library will provide multiple operating systems support, and provision for multiple types of scientific imaging data and data formats, with particular reference to medical imaging data formats. The core importance of the library will be the availability of a common infrastructure which will allow the development of distributed image analysis tasks.

The work started with the definition of the computer vision and image processing library to be implemented. An in depth analysis of the most important and most widely used libraries for image processing and analysis was performed, focusing on those which are able to be used in HPC and cloud environments. This was a key step to define a concrete and detailed API, which will enable the application providers to adapt their applications to the new library. This will be detailed in the deliverable D2.1.

The design of ECVL takes into account two aims: the first one is allowing an easy integration and data exchange between existing state of the art libraries and their interconnection with the EDDLL. The development of the ECVL is following the best practices of modern software development, such as a test driven approach, to ensure that novel additions do not break compatibility with already existing libraries and already applied applications of it. The second aim is the setup of performance testing frameworks, which will allow repeatable experiments on large scale datasets to verify the impact of the different modifications. The design of the ECVL started with the inspection of the data model used in mainstream libraries to represent n-dimensional signals and its basic structure has been defined in order to cope with the variability expected in all the models.

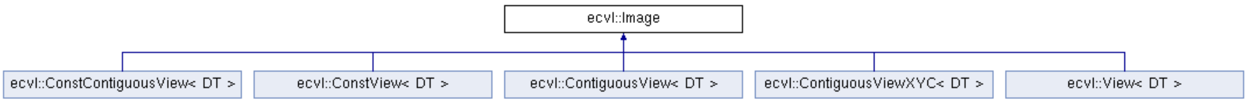
The generic tensor model selected for the ECVL is described in Section and special care has been taken to include all the datatypes currently used.

Another important step, not completed yet, has been the selection of some of the computer vision algorithms which are most commonly employed in conjunction with deep learning algorithms, in order to provide specialized/accelerated versions for use with the EDDLL.

1.2.1 ECVL *Image*

The ECVL library develops around the *Image* object which represents the core of the entire library. For this reason, in order to introduce the ECVL API providing some examples, an exhaustive explanation of how an *Image* works and how it can be used is reported, together with the motivations behind the choices made.

The ECVL *Image* is an object that stores data (either images/videos or raw data) in a multi-dimensional dense numerical single- or multi-channel tensor. The main attributes that constitute an *Image* are the following:



- `elemtype` identifies the type of pixels inside the *Image*. It could be any of the standard types provided by the *C++*. For performance reasons, its value can be reasonably limited to the types commonly used to handle images in Computer Vision algorithms: signed and unsigned integers at 8, 16, 32, 64 bits, and floating point numbers at 32 and 64 bits. Prevision for 16 bits floating point number is being discussed.
- `elemsize` is the size (in bytes) of *Image* pixels.
- `mem` identifies the *Memory Manager* employed by the Image. It is useful to handle *Image* memory (both on CPU and GPU and possibly on FPGA) masking implementation details and thus simplifying the work of the programmer.
- `dims` is a vector of *Image* dimensions. Each dimension is given in number of elements it contains. Faster changing dimensions come before slower changing ones. So, the first dimension is the one that changes faster in memory and the last is the slowest one. Let's consider a 2-dimensional *Image* with `dims={4,3}`: element (0,0) will be the first in memory, followed by (1,0), followed by (2,0), followed by (3,0), followed by (0,1), and so on. The meaning of these dimensions is specified in the `channels` field, explained later.
- `strides` is the data layout of the *Image*. That is a vector whose elements represent the number of bytes the pointer on data has to move to reach the next element on the corresponding dimension. The following equation explains how to use this information to get a pointer to the element at position $(i, j, ..., k)$ of a n-dimensional *Image*.

  $address(I_{i,j,...,k}) = I.stride[0] \cdot i + I.stride[1] \cdot j + ... + I.stride[stride.size() - 1] \cdot k$

  In case of a 2-dimensional *Image*, the above formula is reduced as follows.

  $address(I_{i,j}) = I.stride[0] \cdot i + I.stride[1] \cdot j$

  The introduction of strides allows for an extremely flexible way of accessing image data, because it is possible to obtain a cropped or transposed view without any data copy. This comes at the price of slower generic access, which requires special care during the implementation of core library algorithms.
- `channels` is a string which describes how *Image* planes are organized. A single character provides the information related to the corresponding channel. The possible values are:

  1. `'x'`: horizontal spatial dimension;
  2. `'y'`: vertical spatial dimension;
  3. `'z'`: depth spatial dimension;
  4. `'c'`: color dimension;
  5. `'t'`: temporal dimension;
  6. `'o'`: any other dimension;

  For example, `"xyc"` describes a 2-dimensional *Image* structured in color planes. This could be for example a `GRAY` Image with `dims[2] = 1` or a `RGB` Image with `dims[2] = 3` an so on. The `colortype` (explained later) constrains the value of the dimension corresponding to the color channel. Another example is `"cxy"` with `dims[0] = 3` and `BGR`. In this case the color dimension is the one which changes faster as it is done in many other libraries such as OpenCV. Allowing these different data structures makes the ECVL library compliant with most of the existing CV libraries and, at the same time, allows to reduce the cost of moving data from ECVL *Image* to EDDL *Tensor* and viceversa.
- `colortype` is the *Image* color space. Allowed values for this attribute could be: `none`, a special color type for *Images* that contain only data and do not have any associated color space, `GRAY`, `RGB`, `BGR`, `HSV`, `YCbCr`. If this attribute is different from `none` the channels string must contain a `'c'` and the corresponding dimension must have the appropriate value.

- `data` is the pointer to *Image* data. If the Image is not the owner of data, for example when using *Image* views, this attribute will point to the data of another Image. The possession or not of the data depends on the *Memory Manager*.
- `datasize` is the size (in bytes) of *Image* data.
- `contiguous` whether the image is stored contiguously or not in memory. Allowing non contiguous *Image* data is useful to improve the performance of some Computer Vision algorithms when running on CPU.
- `meta` is a pointer to *Image* metadata.

The fact that the *Image* class is not a template data type simplifies the management of reading and writing data from files, to the detriment of its usability during programming. To solve this problem, a template subclass has been introduced, named *View*, that simplifies the work of the programmer making the coding extremely straightforward. The data handling model is exemplified in Fig. 1.2.1.

To ensure the maximum performace an *Image View* could be either a standard *View* or a *ContiguousView*. When creating a *View* of a contiguous *Image*, *i.e.* an *Image* with contiguous data, it is better to prefer the *ContiguousView*.

Regarding the ECVL, Section 10 describes the API for ECVL including examples in some cases. Section 11 describes the funcions for EDVL–EDDLL exchange of data objects. Mainly images and datasets, in the case of images two functions are provided, one for converting an image into a tensor, and its complementary for converting a tensor into an image. Regarding datasets, a dataset of images, provided as a vector (list) of strings is converted into a tensor including all the images.

Finally, it is provided an appendix including the current version of the documentation of the ECVL. Which has been automatically generated from the comments in the source code by using Doxygen `http://www.doxygen.nl/`. The documentation generated this way is the definition of the API at the technical level. The definitive documentation of both libraries will be provided this way, one of most widely used standards for API documentation. That is why the appendix does not follow the format of deliverables. It is provided as an example.

2 EDDLL – LAYERS in C++

In order to create a Layer the EDDL provides the following API

```
layer l = eddl.capsule.LAYER_TYPE(parent_layer, arguments)
```

Where,

- *capsule* allows us to group some similar.
- *LAYER_TYPE* is one of the layers exposed next.
- *parent_layer*, is the layer which the new layer is connected to (void in the case of input layers).
- *arguments*, are the necessary values depending on the layer to be defined.

2.1 Input

```
static layer Input(const initializer_list<int>& shape, string name)
```

Used as entry point layer to the neural network. Example:

```
int batch = 64; // Num samples
int data_dim = 784 // Images of 28*28 pixels
layer in = eddl.Input({batch, data_dim});
```

Arguments:

- **shape** - Tuple indicating the expected shape for the input data.
- **name** - Layer name so that it can be selected easily. Default: ”input” + number of previous Input layers created.

2.2 Dense

```
static layer Dense(layer parent, int ndim, bool use_bias, string name)
```

Applies a linear transformation to the incoming data: $y = xA^T + b$. Example:

```
layer in = eddl.Input({batch, data_dim});
layer d1 = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
```

Arguments:

- **parent** - Previous layer with which current one is connected.
- **ndim** - Positive integer, dimensionality of the output space.
- **use_bias** - Boolean, whether the layer will learn an additive bias vector. Default: true.
- **name** - Layer name so that it can be selected easily. Default: ”dense” + number of previous Dense layers created.

2.3 Reshape

```
static layer Reshape(layer parent, const initializer_list<int>& shape,
                      string name)
```

Reshapes an output to a certain shape. Example:

```
layer in = eddl.Input({batch,784});
layer l = eddl.Reshape(in, {batch,1,28,28});
```

Arguments:

- **parent** - Previous layer to which we will apply the reshape.
- **shape** - Target shape. Tuple of integers.
- **name** - Layer name so that it can be selected easily. Default: "reshape" + number of previous Reshape layers created.

2.4 Conv

```
static layer Conv(layer parent, int filters,
                  const initializer_list<int>& kernel_size, string padding,
                  const initializer_list<int>& strides, int groups,
                  const initializer_list<int>& dilation_rate,
                  bool use_bias, string name)
```

Applies a convolution over an input signal composed of several input planes. Example:

```
layer in = eddl.Input({batch, {1, 28, 28}}); // Example image data mnist
layer d1 = eddl.Conv(in, 64, {3, 3}); // Conv layer with 64 filters of 3x3
```

Arguments:

- **parent** - Previous layer to which we will apply the convolution.
- **filters** - Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size** - An integer or tuple/list of 2 integers, specifying the height and width of the convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides** - An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Default: {1, 1}.
- **padding** - One of "valid" or "same". Default: "valid".
- **dilation_rate** - An integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Visualization Default: {1, 1}.
- **groups** - Controls the connections between inputs and outputs. Number of groups input channels and output channels are divided into. Default: 1.
- **use_bias** - Boolean, whether the layer uses a bias vector. Default: true.
- **name** - Layer name so that it can be selected easily. Default: "conv" + number of previous Conv layers created.

2.5 ConvT

```
static layer ConvT(layer parent, int filters,
                   const initializer_list<int>& kernel_size, string padding,
                   const initializer_list<int>& output_padding,
                   const initializer_list<int>& dilation_rate,
                   const initializer_list<int>& strides,
                   bool use_bias, string name)
```

Applies a transposed convolution (sometimes called Deconvolution) operator over an input image composed of several input planes. Example:

```
layer in = eddl.Input({batch, {1, 28, 28}}); // Example image data mnist
layer dc1 = eddl.ConvT(in, 64, {3, 3}); // ConvT layer with 64 filters of 3x3
```

Arguments:

- **parent** - Previous layer to which we will apply the deconvolution.
- **filters** - Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size** - An integer or tuple/list of 2 integers, specifying the height and width of the convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides** - An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Default: {1, 1}.
- **padding** - One of "valid" or "same". Default: "valid".
- **output_padding** - An integer or tuple/list of 2 integers, specifying the amount of padding along the height and width of the output tensor. Can be a single integer to specify the same value for all spatial dimensions. The amount of output padding along a given dimension must be lower than the stride along that same dimension. Default: the output shape is inferred.
- **dilation_rate** - Controls the spacing between the kernel points. Visualization.
- **use_bias** - Boolean, whether the layer uses a bias vector. Default: true.
- **name** - Layer name so that it can be selected easily. Default: "convt" + number of previous ConvT layers created.

2.6 UpSampling

```
static layer UpSampling(layer parent, const initializer_list<int>& size,
                        string interpolation, string name)
```

Upsampling layer. Repeats the corresponding dimensions of the data by themselves. Example:

```
layer in = eddl.Input({batch, {1, 28, 28}}); // Example image data mnist
layer up1 = eddl.UpSampling(in, {2, 2}); // Upsampling of size 2x2
```

Arguments:

- **parent** - Previous layer to which we will apply the convolution.
- **size** - The upsampling factors for each dimension.
- **interpolation** - One of nearest or bilinear. Default: nearest.

2.7 Transpose

```
static layer Transpose(layer parent, const initializer_list<int>& dims,
                       string name)
```

Transposes the dimensions of the input according to a given pattern. Example:

```
layer in = eddl.Input({batch,784});
layer l = eddl.Reshape(in, {batch,1,28,28});
layer perm = eddl.Transpose(l, {0,2,3,1}); // Give us l as {batch, 28, 28, 1}
```

Arguments:

- **parent** - Previous layer to which we will apply the reshape.
- **dims** - Tuple of integers. Permutation pattern.
- **name** - Layer name so that it can be selected easily. Default: "transpose" + number of previous Transpose layers created.

2.8 Depthwise and Separable Convolutions

At Conv layer 6.4, if $groups = input\_channels$, then it is Depthwise. If $groups = input\_channels$, and $kernel\_size = (K, 1)$, (and before is a Conv2d layer with groups=1 and kernel_size=(1, K)), then it is Separable.

2.9 Embedding

```
static layer Embedding(int input_dim, int output_dim, string name)
```

Turns positive integers (indexes) into dense vectors of fixed size. eg. [[4], [20]] → [[0.25, 0.1], [0.6, -0.2]]. This layer can only be used as the first layer in a model.

Arguments:

- **input_dim** - Size of the vocabulary.
- **output_dim** - Dimension of the dense embedding.
- **name** - Layer name so that it can be selected easily. Default: "embedding" + number of previous Embedding layers created.

2.10 Activation

```
static layer Activation(layer parent, string activation, string name)
```

Applies an activation function to a layer. We must access them through ***activations***. Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
layer l = eddl.activations.ReLU(eddl.Dense(in,1024));
```

Arguments:

- **parent** - Previous layer to which we will apply the activation.
- **act** - Name of activation function to use. Complete list next.
- **name** - Layer name so that it can be selected easily. Default: "activation" + number of previous Activation layers created.

Available Activation Functions:

- **Sigmoid** - Applies the element-wise function $Sigmoid(x) = \frac{1}{1+\exp(-x)}$.
- **Tanh** - Applies the element-wise function $Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.
- **ReLU** - Applies the element-wise function $ReLU(x) = \max(0, x)$.
- **Softmax** - Applies the Softmax function to an n-dimensional input Tensor rescaling them so that the elements of the n-dimensional output Tensor lie in the range (0,1) and sum to 1. Defined as: $Softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$.

2.11 Dropout

```
static layer Dropout(layer parent, float rate, string name)
```

Applies Dropout to the input. Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting. Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
layer l = eddl.Activation(eddl.Dense(in, 1024), "relu");
layer drop = eddl.Dropout(l, 0.5); // Applies a dropout over l
```

Arguments:

- **parent** - Previous layer to which we will apply the dropout.
- **rate** - Float between 0 and 1. Fraction of the input units to drop.
- **name** - Layer name so that it can be selected easily. Default: "dropout" + number of previous Dropout layers created.

2.12 GaussianNoise

```
static layer GaussianNoise(layer parent, float stdev, string name)
```

Apply additive zero-centered Gaussian noise. This is useful to mitigate overfitting (you could see it as a form of random data augmentation). Gaussian Noise (GS) is a natural choice as corruption process for real valued inputs. As it is a regularization layer, it is only active at training time. Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
// Apply GaussianNoise with stddved 0.1 to Dense Layer with 1024 neurons
layer gs1 = eddl.GaussianNoise(eddl.Dense(in, 1024), 0.1);
```

Arguments:

- **parent** - Previous layer to which we will apply the dropout.
- **stdev** - Standard deviation of the noise distribution.
- **name** - Layer name so that it can be selected easily. Default: "gaussian_noise" + number of previous GaussianNoise layers created.

2.13 BatchNormalization

```
static layer BatchNormalization(layer parent, float momentum, float epsilon,
                                bool affine, string name)
```

This layer type normalizes the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
// Apply BatchNormalization to Dense Layer with 1024 neurons
layer bn1 = eddl.BatchNormalization(eddl.Dense(in, 1024));
```

Arguments:

- **parent** - Previous layer to which we will apply the dropout.
- **momentum** - Momentum for the moving mean and the moving variance. Default: 0.99.
- **epsilon** - Small float added to variance to avoid dividing by zero. Default: 0.001.
- **affine** - A boolean value that when set to true, this module has learnable affine parameters. Default: true.
- **name** - Layer name so that it can be selected easily. Default: "batchnorm" + number of previous BatchNormalization layers created.

2.14 Pooling

A pooling layer is another building block of a CNN. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently.

2.14.1 MaxPool

```
static layer MaxPool(layer parent, const initializer_list<int>& pool_size,
                     const initializer_list<int>& strides, string padding,
                     string name)
```

Max pooling operation for spatial data. Example:

```
layer in = eddl.Input({batch, 1,256,256});
layer conv1 = eddl.Activation(eddl.Conv(in, 32, {3,3}), "relu");
layer pool1 = eddl.MaxPool(conv1, {2,2});
```

Arguments:

- **parent** - Previous layer to which we will apply the Max Pooling.
- **pool_size** - An integer or tuple/list of 2 integers, specifying the height and width of the Max Pooling window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides** - An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Default: Same as pool_size.
- **padding** - One of "valid" or "same". Default: "valid".
- **name** - Layer name so that it can be selected easily. Default: "maxpool" + number of previous MaxPool layers created.

2.14.2 GlobalMaxPool

```
static layer GlobalMaxPool(layer parent, string name)
```

Global max pooling operation for spatial data. Example:

```
layer in = eddl.Input({batch, 1,256,256});
layer conv1 = eddl.Activation(eddl.Conv(in, 32, {3,3}), "relu");
layer pool1 = eddl.GlobalMaxPool(conv1);
```

Arguments:

- **parent** - Previous layer to which we will apply the Global Max Pooling.
- **name** - Layer name so that it can be selected easily. Default: "globalmaxpool" + number of previous GlobalMaxPool layers created.

2.14.3 AveragePool

```
static layer AveragePool(layer parent, const initializer_list<int>& pool_size,
                         const initializer_list<int>& strides, string padding,
                         string name)
```

Average pooling operation for spatial data. Example:

```
layer in = eddl.Input({batch, 1,256,256});
layer conv1 = eddl.Activation(eddl.Conv(in, 32, {3,3}), "relu");
layer pool1 = eddl.AveragePool(conv1, {2,2});
```

Arguments:

- **parent** - Previous layer to which we will apply the Average Pooling.
- **pool_size** - An integer or tuple/list of 2 integers, specifying the height and width of the Average Pooling window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides** - An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions.
- **padding** - One of "valid" or "same".
- **name** - Layer name so that it can be selected easily. Default: "avgpool" + number of previous AveragePool layers created.

2.14.4 GlobalAveragePool

```
static layer GlobalAveragePool(layer parent, string name)
```

Global average pooling operation for spatial data. Example:

```
layer in = eddl.Input({batch, 1,256,256});
layer conv1 = eddl.Activation(eddl.Conv(in, 32, {3,3}), "relu");
layer pool1 = eddl.GlobalAveragePool(conv1);
```

Arguments:

- **parent** - Previous layer to which we will apply the Global Average Pooling.
- **name** - Layer name so that it can be selected easily. Default: "globalavgpool" + number of previous GlobalAveragePool layers created.

2.15 Merge

The function of the Merge layers is to take several input layers and merge them by performing a certain operation so that we obtain a resulting layer.

2.15.1 Add

```
static layer Add(const initializer_list<layer>& layers, string name)
```

Layer that adds a list of inputs. It takes as input a list of layers, all of the same shape, and returns a single layer (also of the same shape). Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
layer l1 = eddl.Dense(in, 512);
layer l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their sum
layer l3 = eddl.Add({l1, l2});
```

Arguments:

- **layers** - List of input layers to perform the add operation.
- **name** - Layer name so that it can be selected easily. Default: ”add” + number of previous Add layers created.

2.15.2 Subtract

```
static layer Subtract(const initializer_list<layer>& layers, string name)
```

Layer that subtract a list of inputs. It takes as input a list of layers, all of the same shape, and returns a single layer (also of the same shape). Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
layer l1 = eddl.Dense(in, 512);
layer l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their subtraction
layer l3 = eddl.Subtract({l1, l2});
```

Arguments:

- **layers** - List of input layers to perform the add operation.
- **name** - Layer name so that it can be selected easily. Default: ”subtract” + number of previous Subtract layers created.

2.15.3 Concat

```
static layer Concat(const initializer_list<layer>& layers, string name)
```

Layer that concatenates a list of inputs. It takes as input a list of layers and returns the concatenation of all inputs. Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
layer l1 = eddl.Dense(in, 512);
layer l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their concatenation
layer l3 = eddl.Concat({l1, l2});
```

Arguments:

- **layers** - List of input layers to perform the concatenation operation.
- **name** - Layer name so that it can be selected easily. Default: ”concat” + number of previous Concat layers created.

2.15.4 MatMul

```
static layer MatMul(const initializer_list<layer>& layers, string name)
```

The output of this layer is the matrix product of the two input layers provided in a list. Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
layer l1 = eddl.Dense(in, 512);
layer l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their matrix multiplication
layer l3 = eddl.MatMul({l1, l2});
```

Arguments:

- **layers** - List of input layers to perform the matrix multiplication operation.
- **name** - Layer name so that it can be selected easily. Default: ”matmul” + number of previous MatMul layers created.

2.15.5 Average

```
static layer Average(const initializer_list<layer>& layers, string name)
```

Layer that averages a list of inputs. It takes as input a list of layers, all of the same shape, and returns a single tensor (also of the same shape). Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
layer l1 = eddl.Dense(in, 512);
layer l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their average
layer l3 = eddl.Average({l1, l2});
```

Arguments:

- **layers** - List of input layers to perform the average operation.
- **name** - Layer name so that it can be selected easily. Default: ”average” + number of previous Average layers created.

2.15.6 Maximum

```
static layer Maximum(const initializer_list<layer>& layers, string name)
```

Layer that computes the maximum (element-wise) a list of inputs. It takes as input a list of layers, all of the same shape, and returns a single tensor (also of the same shape). Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
layer l1 = eddl.Dense(in, 512);
layer l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their maximum
layer l3 = eddl.Maximum({l1, l2});
```

Arguments:

- **layers** - List of input layers to perform the maximum operation.
- **name** - Layer name so that it can be selected easily. Default: ”maximum” + number of previous Maximum layers created.

2.15.7 Minimum

```
static layer Minimum(const initializer_list<layer>& layers, string name)
```

Layer that computes the minimum (element-wise) a list of inputs. It takes as input a list of layers, all of the same shape, and returns a single tensor (also of the same shape). Example:

```
layer in = eddl.Input({batch, 784}); // Example image data mnist
layer l1 = eddl.Dense(in, 512);
layer l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their minimum
layer l3 = eddl.Minimum({l1, l2});
```

Arguments:

- **layers** - List of input layers to perform the minimum operation.
- **name** - Layer name so that it can be selected easily. Default: ”minimum” + number of previous Minimum layers created.

2.16 Recurrent Layers

2.16.1 GRU

```
static layer RNN(layer parent, int units, string activation, bool unroll
                 string recurrent_activation, bool reset_after, bool use_bias,
                 float dropout, float recurrent_dropout, int implementation,
                 bool return_sequences, bool return_state, bool go_backwards,
                 bool stateful)
```

Gated Recurrent Unit - Cho et al. 2014.

Arguments:

- **parent** - Previous layer to which we will apply the GRU.
- **units** - Positive integer, dimensionality of the output space.
- **activation** - Activation function to use. Default: 'linear'.
- **unroll** - If True, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences. Default: false.
- **recurrent_activation** – Activation function to use for the recurrent step. Default: 'linear'.
- **reset_after** – GRU convention (whether to apply reset gate after or before matrix multiplication). False = "before", True = "after" (CuDNN compatible). Default: false.
- **use_bias** - Boolean, whether the layer uses a bias vector. Default: true.
- **dropout** - Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs. Default: 0.
- **recurrent_dropout** - Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state. Default: 0.
- **implementation** - Implementation mode, either 1 or 2. Mode 1 will structure its operations as a larger number of smaller dot products and additions, whereas mode 2 will batch them into fewer, larger operations. These modes will have different performance profiles on different hardware and for different applications. Default: 1.
- **return_sequences** - Boolean. Whether to return the last output in the output sequence, or the full sequence. Default: false.
- **return_state** - Boolean. Whether to return the last state in addition to the output. Default: false.
- **go_backwards** - If True, process the input sequence backwards and return the reversed sequence. Default: false.
- **stateful** - If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch. Default: false.

2.16.2 LSTM

```
static layer LSTM(layer parent, int units, int num_layers, bool use_bias,
                  float dropout, bool bidirectional string name)
```

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

Arguments:

- **units** - Positive integer, dimensionality of the output space.
- **num_layers** - Number of recurrent layers. E.g., setting num_layers=2 would mean stacking two LSTMs together to form a stacked LSTM, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1.
- **use_bias** - Boolean, whether the layer uses a bias vector. Default: true.
- **dropout** – If non-zero, introduces a Dropout layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to dropout. Default: 0
- **bidirectional** – If true, becomes a bidirectional LSTM. Default: false
- **name** - Layer name so that it can be selected easily. Default: ”lstm” + number of previous LSTM layers created.

2.17 Initializers

Initializations define the way to set the initial random weights of some layers. We must access them through ***initializers***. Example:

```
my_layer.SetWeights(eddl.initializers.MYINIT(params))
```

Where *MYINIT* is one of the following.

2.17.1 Constant

```
Constant(float value)
```

Generates tensors initialized to a constant value. Example:

```
layer my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Set to 0 all values of my_dense
my_dense.SetWeights(eddl.initializers.Constant(0));
```

Arguments:

- **value** - The value of the generator tensors.

2.17.2 RandomNormal

```
RandomNormal(float mean, float stdev, int seed)
```

Initializer that generates tensors with a normal distribution. Example:

```
layer my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense with a normal distribution
my_dense.SetWeights(eddl.initializers.RandomNormal(0.0, 0.05));
```

Arguments:

- **mean** - Mean of the random values to generate.
- **stdev** - Standard deviation of the random values to generate.
- **seed** - Used to seed the random generator. Default: 0.

2.17.3 RandomUniform

```
RandomUniform(float minval, float maxval, int seed)
```

Initializer that generates tensors with a uniform distribution. Example:

```
layer my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense with a uniform distribution with seed 42
my_dense.SetWeights(eddl.initializers.RandomUniform(-0.5, 0.05, 42));
```

Arguments:

- **minval** - Lower bound of the range of random values to generate.
- **maxval** - Upper bound of the range of random values to generate. Defaults to 1 for float types.
- **seed** - Used to seed the random generator. Default: 0.

2.17.4 Identity

```
Identity(float gain)
```

Initializer that generates the identity matrix. Only use for 2D matrices. If the desired matrix is not square, it pads with zeros on the additional rows/columns. Example:

```
layer my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense with the identity
my_dense.SetWeights(eddl.initializers.Identity(1.0));
```

Arguments:

- **gain** - Multiplicative factor to apply to the identity matrix.

2.17.5 Orthogonal

```
Orthogonal(float gain, int seed)
```

Initializer that generates a random orthogonal matrix. Example:

```
layer my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense with a random orthogonal matrix
my_dense.SetWeights(eddl.initializers.Orthogonal(1.0));
```

Arguments:

- **gain** - Multiplicative factor to apply to the orthogonal matrix.
- **seed** - Used to seed the random generator. Default: 0.

2.17.6 Glorot normal

```
GlorotNormal(int seed)
```

Glorot normal initializer, also called Xavier normal initializer. Glorot normal initializer, also called Xavier normal initializer, that draws values for the weights from a truncated normal distribution centered on 0 with $stdev = sqrt(2/(fan\_in + fan\_out))$, where fan_in is the number of input units in the weight tensor and fan_out is the number of output units in the weight tensor. Example:

```
layer my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense following the exposed procedure
my_dense.SetWeights(eddl.initializers.GlorotNormal());
```

Arguments:

- **seed** - Used to seed the random generator. Default: 0.

2.17.7 Glorot uniform

```
GlorotUniform(int seed)
```

Glorot uniform initializer, also called Xavier uniform initializer, that draws values for the weights from a uniform distribution within [-limit, limit] where limit is $sqrt(6/(fan\_in + fan\_out))$, fan_in is the number of input units in the weight tensor and fan_out is the number of output units in the weight tensor. Example:

```
layer my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense following the exposed procedure
my_dense.SetWeights(eddl.initializers.GlorotUniform());
```

Arguments:

- **seed** - Used to seed the random generator. Default: 0.

2.18 Additional Methods

All layers have a number of methods in common:

- **layer.GetWeights()**: returns the weights of the layer.
- **layer.GetBias()**: returns the bias of the layer.
- **layer.SetWeights(weights)**: sets the weights of the layer (with the same shapes as the output of get_weights).
- **layer.SetBias(bias)**: sets the bias of the layer (with the same shapes as the output of get_bias).

We can also take intermediate layers of a model as by his name or index:

```
layer intermediate_layer = eddl.GetLayer(my_model, "layer_name")
layer intermediate_layer2 = eddl.GetLayer(my_model, 2)
```

3 EDDLL – MODELS in C++

To define a network we have to specify a list of input and output layers:

```
model net=eddl.Model(list_of_inputs, list_of_outpus)
```

To finally build a network we have to attach the optimizer, list of loss functions and list of metrics:

```
eddl.build(net, optimizer, list_of_loss, list_of_metrics)
```

3.1 Optimizers

The following subsections describe the most common optimizers used in other Deep Learning libraries and that are going to be implemented in the EDDL library. All the classes for the optimizers are subclases of the class **Optimizer** not presented here because is not going to be visible as a class outside the code of the EDDL library.

3.1.1 SGD

```
static optimizer SGD(float lr, float momentum, float decay,
                     bool nesterov)
```

Implements stochastic gradient descent. Example:

```
optimizer sgd = eddl.optimizers.SGD(0.01, 0.9);
eddl.build(net, sgd, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **momentum** - Momentum factor. Default: 0.
- **decay** - Learning rate decay over each update. Default: 0.
- **nesterov** - Learning rate. Default: false.

3.1.2 RMSprop

```
static optimizer RMSprop(float lr, float rho, float epsilon, float decay)
```

RMSProp optimizer. Example:

```
optimizer rmsprop = eddl.optimizers.RMSProp(0.01);
eddl.build(net, rmsprop, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **rho** - Smoothing constant. Default: 0.9.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **decay** - Learning rate decay over each update. Default: 0.

3.1.3 Adam

```
static optimizer Adam(float lr, float beta_1, float beta_2,
                      float epsilon, float decay, bool amsgrad)
```

Implements Adam algorithm. Example:

```
optimizer adam = eddl.optimizers.Adam(0.001);
eddl.build(net, adam, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **beta_1** - $0 < beta < 1$. Generally close to 1. Default: 0.9.
- **beta_2** - $0 < beta < 1$. Generally close to 1. Default: 0.999.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **decay** - Learning rate decay over each update. Default: 0.
- **amsgrad** - Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond". Default: false.

3.1.4 Adagrad

```
static optimizer Adagrad(float lr, float epsilon, float decay)
```

Adagrad is an optimizer with parameter-specific learning rates, which are adapted according to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the learning rate. Example:

```
optimizer adagrad = eddl.optimizers.Adagrad(0.01);
eddl.build(net, adagrad, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **decay** - Learning rate decay over each update. Default: 0.

3.1.5 Adadelta

```
static optimizer Adadelta(float lr, float rho, float epsilon, float decay)
```

Adadelta is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. This way, Adadelta continues learning even when many updates have been done. Compared to Adagrad, in the original version of Adadelta you don't have to set an initial learning rate. Example:

```
optimizer adadelta = eddl.optimizers.Adadelta(0.01, 0.99);
eddl.build(net, adadelta, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **rho** - Adadelta decay factor, corresponding to fraction of gradient to keep at each time step. Default: 0.95.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **decay** - Initial learning rate decay. Default: 0.

3.1.6 Adamax

```
static optimizer Adamax(float lr, float beta_1, float beta_2,
                        float epsilon, float decay)
```

It is a variant of Adam based on the infinity norm. Example:

```
optimizer adamax = eddl.optimizers.Adamax(0.0001);
eddl.build(net, adamax, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **beta_1** - 0 < beta < 1. Generally close to 1. Default: 0.9.
- **beta_2** - 0 < beta < 1. Generally close to 1. Default: 0.999.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **decay** - Learning rate decay over each update. Default: 0.

3.1.7 Nadam

```
static optimizer Nadam(float lr, float beta_1, float beta_2,
                       float epsilon, float schedule_decay)
```

Nesterov Adam optimizer. As Adam is essentially RMSprop with momentum, Nadam is Adam RMSprop with Nesterov momentum. Example:

```
optimizer nadam = eddl.optimizers.Nadam(0.01);
eddl.build(net, nadam, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **beta_1** - $0 < \text{beta} < 1$. Generally close to 1. Default: 0.9.
- **beta_2** - $0 < \text{beta} < 1$. Generally close to 1. Default: 0.999.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **schedule_decay** - $0 < \text{schedule\_decay} < 1$. Default: 0.004.

3.2 Learning Rate Schedulers

There are several methods to adjust the learning rate based on the number of epochs or by choosing some validation measurements. We must access them through ***callbacks***. To apply them, we can attach them as a parameter at the fit function:

```
eddl.fit(..., callbacks={MyCallbacks})
```

3.2.1 StepLR

```
static callback StepLR(int step_size, float gamma, int last_epoch)
```

Sets the learning rate of each parameter group to the initial lr decayed by gamma every step_size epochs. When last_epoch=-1, sets initial lr as lr. Example:

```
callback step_lr = eddl.callbacks.StepLR(20); //Decay every 20 epochs
eddl.fit(..., callbacks={step_lr});
```

Arguments:

- **step_size** - Period of learning rate decay.
- **gamma** - Multiplicative factor of learning rate decay. Default: 0.1.
- **last_epoch** - The index of last epoch. Default: -1.

3.2.2 MultiStepLR

```
static callback MultiStepLR(const initializer_list<int>& milestones,
                            float gamma, int last_epoch)
```

Set the learning rate of each parameter group to the initial lr decayed by gamma once the number of epoch reaches one of the milestones. When last_epoch=-1, sets initial lr as lr. Example:

```
callback multystep_lr = eddlcallbacks..MultiStepLR({10,15,5});
eddl.fit(..., callbacks={multystep_lr});
```

Arguments:

- **milestones** - List of epoch indices. Must be increasing.
- **gamma** - Multiplicative factor of learning rate decay. Default: 0.1.
- **last_epoch** - The index of last epoch. Default: -1.

3.2.3 ExponentialLR

```
static callback ExponentialLR(float gamma, int last_epoch)
```

Set the learning rate of each parameter group to the initial lr decayed by gamma every epoch. When last_epoch=-1, sets initial lr as lr. Example:

```
callback exponential_lr = eddl.callbacks.ExponentialLR(0.87);
eddl.fit(..., callback={exponential_lr});
```

Arguments:

- **gamma** - Multiplicative factor of learning rate decay.
- **last_epoch** - The index of last epoch. Default: -1.

3.2.4 CosineAnnealingLR

```
static callback CosineAnnealingLR(int T_max, float eta_min,
                                  int last_epoch)
```

Set the learning rate of each parameter group using a cosine annealing schedule. Example:

```
callback cosine_lr = eddl.callbacks.CosineAnnealingLR(0.87);
eddl.fit(..., callback={cosine_lr});
```

Arguments:

- **T_max** - Maximum number of iterations.
- **eta_min** - Minimum learning rate. Default: 0.
- **last_epoch** - The index of last epoch. Default: -1.

3.2.5 ReduceLROnPlateau

```
static callback ReduceLROnPlateau(string metric, string mode, float factor,
                                  int patience, float threshold,
                                  string threshold_mode, int cooldown,
                                  float min_lr, float eps)
```

Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This scheduler reads a metrics quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced. Example:

```
callback plateau_lr = eddl.callbacks.ReduceLROnPlateau();
eddl.fit(..., callback={plateau_lr});
```

Arguments:

- **metric** - One of 7.5.
- **mode** - One of min, max. In min mode, lr will be reduced when the quantity monitored has stopped decreasing; in max mode it will be reduced when the quantity monitored has stopped increasing. Default: 'min'.
- **factor** -Factor by which the learning rate will be reduced. new_lr = lr * factor. Default: 0.1.
- **patience** -Number of epochs with no improvement after which learning rate will be reduced. For example, if patience = 2, then we will ignore the first 2 epochs with no improvement, and will only decrease the LR after the 3rd epoch if the loss still hasn't improved then. Default: 10.
- **threshold** - Threshold for measuring the new optimum, to only focus on significant changes. Default: 1e-4.
- **threshold_mode** - One of rel, abs. In rel mode, dynamic_threshold = best * (1 + threshold) in 'max' mode or best * (1 - threshold) in min mode. In abs mode, dynamic_threshold = best + threshold in max mode or best - threshold in min mode. Default: 'rel'.
- **cooldown** - Number of epochs to wait before resuming normal operation after lr has been reduced. Default: 0.
- **min_lr** - A scalar or a list of scalars. A lower bound on the learning rate of all param groups or each group respectively. Default: 0.
- **eps** - Minimal decay applied to lr. If the difference between new and old lr is smaller than eps, the update is ignored. Default: 1e-8.

3.3 General Callbacks

There are another useful callbacks to get a view on internal states and statistics of the model during training.

3.3.1 History

Callback that records events into an object of the class **History**. This callback is automatically applied to every model, it is not necessary the programmer creates any object of this class explicitly. The **History** object is returned by the fit method of models.

3.3.2 Model Checkpoint

```
static callback ModelCheckpoint(string filepath, bool save_best_only,
                                string mode, int period)
```

Save the model after every epoch.

```
callback model_checkpoint = eddl.callbacks.ModelCheckpoint('results/');
eddl.fit(..., callback={model_checkpoint});
```

Arguments:

- **filepath** - Path to save the model file.
- **save_best_only** - If true, the latest best model according to the quantity monitored will not be overwritten. Default: false.
- **mode** - One of auto, min, max. In 'auto' mode, the direction is automatically inferred from the name of the monitored quantity. Default: 'auto'.
- **period** - Interval (number of epochs) between checkpoints. Default: 1.

3.3.3 Lambda Callback

```
static callback LambdaCallback(void on_epoch_begin, void on_epoch_end,
                               void on_batch_begin, void on_batch_end,
                               void on_train_begin, void on_train_end)
```

Callback for creating simple and custom callbacks on-the-fly. This type of callback is constructed with functions that will be called at the appropriate time. Note that the callbacks expects positional arguments, as:

- *on_epoch_begin* and *on_epoch_end* expect two positional arguments: epoch, logs
- *on_batch_begin* and *on_batch_end* expect two positional arguments: batch, logs
- *on_train_begin* and *on_train_end* expect one positional argument: logs

Examples of how to create lambda callbacks will be provided in the final version of the API documentation.

Arguments:

- **on_epoch_begin:** function to be called at the beginning of every epoch.
- **on_epoch_end:** function to be called at the end of every epoch.
- **on_batch_begin:** function to be called at the beginning of every batch.
- **on_batch_end:** function to be called at the end of every batch.
- **on_train_begin:** function to be called at the beginning of model training.
- **on_train_end:** function to be called at the end of model training.

3.4 Loss Functions

A loss function (or objective function, or optimization score function) is one of the parameters required to **build** a model. Available loss functions:

- **mse** - Creates a criterion that measures the mean squared error.
- **cross_entropy** - It is useful when training a classification problem with C classes.
- **bceloss** - measures the Binary Cross Entropy.
- **kullback_leibler_divergence**
- **poisson**

Example using cross_entropy loss:

```
...
eddl.build(__, __, {"cross_entropy"}, __, __);
```

Custom Loss Functions can be defined as follows: *examples of how to define custom loss functions will be provided in D2.1*

3.5 Metrics

A metric is a function that is used to judge the performance of your model. Available metrics:

- **binary_accuracy**
- **categorical_accuracy**
- **sparse_categorical_accuracy**
- **top_k_categorical_accuracy**
- **sparse_top_k_categorical_accuracy**

Example using categorical_accuracy as metric:

```
...
eddl.build(__, __, __, {"categorical_accuracy"}, __);
```

Custom Metrics can be defined as follows: *examples of how to define custom loss functions will be provided in D2.1*

3.6 Computing services

It is important to note that EDDL provides a hardware abstraction. The API is the same independently of the hardware to be used. It is only necessary to define the hardware to use when the network is build. It is also thought to provide the ability to train the different models in a distributed way.

When a model is build, by default it will use the CPU, but it is possible to provide an object specifying the computing services to use. Two options will be available, local computing services and distributed computing services.

3.6.1 Local

For local training we have only to indicate the device as follows:

```
compserv cs = eddl.computing.CS_XXXX();
eddl.build(net, my_optimizer, {"my_loss"}, {"my_metric"}, cs);
```

Where XXXX() is one of ***CPU***, ***GPU*** or ***FPGA***. You can use an integer with the number of threads or GPUs to be used, or a binary list indicating the exact device. For example:

```
// local CPU with 6 threads
compserv cs = eddl.computing.CS_CPU(4);
// local GPU using the first gpu of 4 installed
compserv cs = eddl.computing.CS_GPU({1,0,0,0});
// local GPU using the first gpu of 1 installed
compserv cs = eddl.computing.CS_GPU({1});
```

3.6.2 Distributed

For distributed training we have to provide an object that will be created by loading the configuration from a text file. The definitive version of the file format for specifying the resources will be defined in collaboration with the teams in charge of adapting the library to HPC architectures and Big Data environments.
An example of using a computing service for training in a distributed environment:

```
compserv cs = eddl.computing.CS_Distributed( "cluster.cfg" );
eddl.build(net, my_optimizer, {"my_loss"}, {"my_metric"}, cs);
```

An example of the contents of the text file **cluster.cfg** is:

```
workernode-01  cpu-cores=32 max-cores=16 gpus=0 max-gpus=0 fpga=0 max-fpga=0
workernode-02  cpu-cores=32 max-cores=16 gpus=0 max-gpus=0 fpga=0 max-fpga=0
workernode-03  cpu-cores=32 max-cores=16 gpus=0 max-gpus=0 fpga=0 max-fpga=0
workernode-04  cpu-cores=32 max-cores=16 gpus=0 max-gpus=0 fpga=0 max-fpga=0
gpu-17         cpu-cores=1  max-cores=0  gpus=2 max-gpus=1 fpga=0 max-fpga=0
gpu-18         cpu-cores=1  max-cores=0  gpus=2 max-gpus=1 fpga=0 max-fpga=0
```

The runtime will execute a process in the CPU of those computers with GPUs, but not for running the training procedure if the maximum of cores is set to zero. The computations will be carried out by the GPU(s).

3.7 Training

To train a network we have to provide the input and output data according to the input and output layers defined. This data is loaded using the tensor functionalities:

```
tensor X=eddl.T("trX.bin")
tensor Y=eddl.T("trY.bin")
```

Finally, train the model:

```
eddl.fit(net,{X},{Y},batch,epochs)
```

4 EDDLL – UTILS for C++

On the other hand we have a series of useful functions for different purposes.

4.1 Save Model

Saves the weights and architecture of the network to be able to load them later.

```
eddl.utils.SaveModel(net, "my_model.pt")
```

4.2 Load Model

Load the weights and the architecture of a network.

```
eddl.utils.LoadModel("trained_model.pt")
```

4.3 Get Layer

Gets by reference a layer of a network.

```
layer middle_layer = eddl.utils.GetLayer(model, "layer_name")
```

4.4 Trainable Models and Layers

It is possible to freeze and unfreeze layers and models. By default all layers are trainable.

```
eddl.utils.SetTrainable(model, false)
eddl.utils.SetTrainable(layer, true)
```

4.5 Zoo models

It is possible to load a model architecture as starting point. Avaible models are:

- VGG: vgg11, vgg13, vgg16, vgg19
- ResNet: resnet18, resnet34, resnet50, resnet101, resnet151
- DenseNet: densenet101, densenet161, densenet169, densnet201

We can laod them as:

```
model myModel = eddl.utils.ZooModels("architecture_name")
```

4.6 Datasets

There exist several public datasets we can load directly:

- mnist: The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.
- cifar10: The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- cifar100: This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class.

For using them we need to call corresponding function by substituting the word DATASET by the name of the dataset in the following example:

```
eddl.utils.DownloadDATASET()
```

4.7 Print summary

Prints a summary of a Net.

```
eddl.utils.summary(net)
```

4.8 Plot

Generates a plot of a network and saves it in a file. In the following example the format is PDF.

```
eddl.utils.plot(net, "model.pdf")
```

To get other formats just change the extension of the file:

```
eddl.utils.plot(net, "model.png")
```

5 EDDLL – EXAMPLES in C++

With very few lines of code we are able to define the network topology, read the data and train the model.

5.1 MultiLayer Perceptron (MLP) - MNIST

In the next snippet we create a MLP composed of an input that receives a vetor of size 784 as Input and then iteratively we add 5 hidden layers of 1024 neurons followed by a relu activation. Finally, an output layer of 10 neurons is added (the number of MNIST classes) and we define our model from the input and output layers. We indicate the optimizer, cost function and metric, load the data and train.

```
#include "../eddl.h"
int main(int argc, char **argv)
{
  int batch=1000;
  layer in=eddl.Input({batch,784});
  layer l=in;
  for(int i=0;i<5;i++)
    l=eddl.Activation(eddl.Dense(l,1024),"relu");
  layer out=eddl.Activation(eddl.Dense(l,10),"softmax");

  model net=eddl.Model({in},{out});

  optimizer sgd = eddl.SGD(0.01, 0.9);

  eddl.build(net,sgd,{"cross_entropy"},{"categorical_accuracy"});

  tensor X=eddl.T("trX.bin");
  tensor Y=eddl.T("trY.bin");

  eddl.div(X,255.0);

  eddl.fit(net,{X},{Y},batch,100);
}
```

5.2 ResNet

The following snippet shows how to create a ResNet model. Also introduces how to use a learning rate scheduler.

```
#include "../eddl.h"

layer ResBlock(layer in,int k,int n)
{

  layer l=in;
  for(int i=0;i<n;i++)
    l=eddl.Activation(eddl.Conv(l,{k,3,3},{1,1}),"relu");

  // adapt depth of input
  in=eddl.Conv(in,{k,1,1},{1,1});
  // add input and last
  l=eddl.Add({in,l});

  // reduce size
  l=eddl.Conv(l,{k,3,3},{2,2});
  return l;
}

int main(int argc, char **argv)
{
  // download MNIST data
  eddl.DownloadMnist();

  int batch=128;

  // network
```

```
  layer in=eddl.Input({batch,784});

  l=eddl.Reshape(in, {batch,1,28,28});

  l=eddl.Activation(eddl.Conv(l,{16,3,3},{2,2}),"relu");
  l=eddl.Activation(eddl.Conv(l,{32,3,3},{2,2}),"relu");
  l=eddl.Activation(eddl.Conv(l,{64,3,3},{2,2}),"relu");
  l=eddl.Activation(eddl.Conv(l,{128,3,3},{2,2}),"relu");

  for(int i=0,k=16;i<3;i++,k=k*2)
    l=ResBlock(l,k,2);

  l=eddl.Reshape(l,{batch,-1});

  l=eddl.Activation(eddl.Dense(l,1024),"relu");

  layer out=eddl.Activation(eddl.Dense(l,10),"softmax");

  // net define input and output layers list
  model net=eddl.Model({in},{out});

  // plot the model
  eddl.plot(net,"model.pdf");

  // get some info from the network
  eddl.info(net);

  // Attach an optimizer and a list of error criteria and metrics
  // size of error criteria and metrics list must match with size
  // of list of outputs. Optionally put a DEVICE where the net will run
  optimizer sgd = eddl.SGD(0.01, 0.9);
  scheduler stepScheduler = eddl.StepLR(25);
  // Apply the scheduler to the optimizer
  eddl.asign_scheduler(sgd, stepScheduler);
  eddl.build(net, sgd,{"cross_entropy"},{"categorical_accuracy"}, DEV_CPU);


  // read data
  tensor X=eddl.T("trX.bin");
  tensor Y=eddl.T("trY.bin");

  eddl.div(X,255.0);

  // training, list of input and output tensors, batch, epochs
  eddl.fit(net,{X},{Y},batch,100);

  // Evaluate test
  tensor tX=eddl.T("tsX.bin");
  tensor tY=eddl.T("tsY.bin");
  eddl.div(tX,255.0);

  eddl.evaluate(net,{tX},{tY});
}
```

5.3 Simple U-Net

The following snippet shows how to create a U-Net model.

```
#include "../eddl.h"
int main(int argc, char **argv)
{
  int batch=64;
  layer in=eddl.Input({batch,1,256,256});

  /* DOWN */
  layer down1 = eddl.Activation(eddl.Conv(in, 32, {3,3}, "same"), "relu");
  down1 = eddl.Activation(eddl.Conv(down1, 32, {3,3}, "same"), "relu");
  layer pool1 = eddl.MaxPool(down1, {2,2});
```

```
  layer down2 = eddl.Activation(eddl.Conv(pool1, 64, {3,3}, "same"), "relu");
  down2 = eddl.Activation(eddl.Conv(down2, 64, {3,3}, "same"), "relu");
  layer pool2 = eddl.MaxPool(down2, {2,2});

  layer down3 = eddl.Activation(eddl.Conv(pool2, 128, {3,3}, "same"), "relu");
  down3 = eddl.Activation(eddl.Conv(down3, 128, {3,3}, "same"), "relu");
  layer pool3 = eddl.MaxPool(down3, {2,2});


  /* Middle */
  layer middle = eddl.Activation(eddl.Conv(pool3, 256, {3,3}, "same"), "relu");
  middle = eddl.Dropout(middle, 0.5);

  /* UP */
  layer up1 = eddl.UpSampling(middle, {2,2});
  up1 = eddl.Concat({down3, up1});
  up1 = eddl.Activation(eddl.Conv(up1, 128, {3,3}, "same"), "relu");
  up1 = eddl.Activation(eddl.Conv(up1, 128, {3,3}, "same"), "relu");

  layer up2 = eddl.UpSampling(up1, {2,2});
  up2 = eddl.Concat({down2, up2});
  up2 = eddl.Activation(eddl.Conv(up2, 128, {3,3}, "same"), "relu");
  up2 = eddl.Activation(eddl.Conv(up2, 128, {3,3}, "same"), "relu");

  layer up3 = eddl.UpSampling(up2, {2,2});
  up3 = eddl.Concat({down1, up3});
  up3 = eddl.Activation(eddl.Conv(up3, 128, {3,3}, "same"), "relu");
  up3 = eddl.Activation(eddl.Conv(up3, 128, {3,3}, "same"), "relu");

  layer out = eddl.Activation(eddl.Conv(up3, 1, {3,3}), "sigmoid");

  model net=eddl.Model({in},{out});
  optimizer adam = eddl.Adam(0.001);
  eddl.build(net, adam,{"bceloss"},{"binary_accuracy"});

  /* Load your data and fit */
}
```

5.4 Transfer Learning

The following snippet shows how to load a model architecture and reuse it for a custom task.

```
#include "../eddl.h"
int main(int argc, char **argv)
{
  int batch=64;
  model vgg16 = edd.zoo_models("vgg16");
  /* we can find 'last_pool' with eddl.summary(vgg16) */
  layer lastPool = eddl.GetLayer(vgg16, "last_pool")
  layer newOut = eddl.Activation(eddl.Dense(lastPool, 10), "softmax");

  model net=eddl.Model({vgg16},{newOut});

  optimizer adam = eddl.Adam(0.001);
  eddl.build(net, adam,{"cross_entropy"},{"categorical_accuracy"});

  /* Load your data and fit */
}
```

6 EDDLL – LAYERS in Python

In order to create a Layer the EDDL provides the following API

```
l = eddl.capsule.LAYER_TYPE(parent_layer, arguments)
```

Where,

- *capsule* allows us to group some similar.
- *LAYER_TYPE* is one of the layers exposed next.
- *parent_layer*, is the layer which the new layer is connected to (void in the case of input layers).
- *arguments*, are the necessary values depending on the layer to be defined.

6.1 Input

```
Input(shape, name)
```

Used as entry po layer to the neural network. Example:

```
batch = 64; // Num samples
data_dim = 784 // Images of 28*28 pixels
in = eddl.Input((batch, data_dim));
```

Arguments:

- **shape** - Tuple indicating the expected shape for the input data.
- **name** - Layer name so that it can be selected easily. Default: ”input” + number of previous Input layers created.

6.2 Dense

```
Dense(parent, ndim, use_bias, name)
```

Applies a linear transformation to the incoming data: $y = xA^T + b$. Example:

```
in = eddl.Input((batch, data_dim));
d1 = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
```

Arguments:

- **parent** - Previous layer with which current one is connected.
- **ndim** - Positive integer, dimensionality of the output space.
- **use_bias** - Boolean, whether the layer uses a bias vector. Default: true.
- **name** - Layer name so that it can be selected easily. Default: ”dense” + number of previous Dense layers created.

6.3 Reshape

```
Reshape(parent, shape, name)
```

Reshapes an output to a certain shape. Example:

```
in = eddl.Input((batch,784));
l = eddl.Reshape(in, [batch,1,28,28]);
```

Arguments:

- **parent** - Previous layer to which we will apply the reshape.
- **shape** - Target shape. Tuple of integers.
- **name** - Layer name so that it can be selected easily. Default: ”reshape” + number of previous Reshape layers created.

6.4 Conv

```
Conv(parent, filters,
     kernel_size, padding,
     strides, groups,
     dilation_rate,
     use_bias, name)
```

Applies a convolution over an input signal composed of several input planes. Example:

```
in = eddl.Input([batch, [1, 28, 28]]); // Example image data mnist
d1 = eddl.Conv(in, 64, [3, 3]); // Conv layer with 64 filters of 3x3
```

Arguments:

- **parent** - Previous layer to which we will apply the convolution.
- **filters** - Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size** - An integer or tuple/list of 2 integers, specifying the height and width of the convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides** - An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Default: {1, 1}.
- **padding** - One of ”valid” or ”same”. Default: ”valid”.
- **dilation_rate** - An integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Visualization Default: {1, 1}.
- **groups** - Controls the connections between inputs and outputs. Number of groups input channels and output channels are divided into. Default: 1.
- **use_bias** - Boolean, whether the layer uses a bias vector. Default: true.
- **name** - Layer name so that it can be selected easily. Default: ”conv” + number of previous Conv layers created.

6.5 ConvT

```
ConvT(parent, filters,
      kernel_size, padding,
      output_padding,
      dilation_rate,
      strides,
      use_bias, name)
```

Applies a transposed convolution (sometimes called Deconvolution) operator over an input image composed of several input planes. Example:

```
in = eddl.Input([batch, [1, 28, 28]]); // Example image data mnist
dc1 = eddl.ConvT(in, 64, [3, 3]); // ConvT layer with 64 filters of 3x3
```

Arguments:

- **parent** - Previous layer to which we will apply the deconvolution.
- **filters** - Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size** - An integer or tuple/list of 2 integers, specifying the height and width of the convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides** - An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Default: {1, 1}.
- **padding** - One of "valid" or "same". Default: "valid".
- **output_padding** - An integer or tuple/list of 2 integers, specifying the amount of padding along the height and width of the output tensor. Can be a single integer to specify the same value for all spatial dimensions. The amount of output padding along a given dimension must be lower than the stride along that same dimension. Default: the output shape is inferred.
- **dilation_rate** - Controls the spacing between the kernel points. Visualization.
- **use_bias** - Boolean, whether the layer uses a bias vector. Default: true.
- **name** - Layer name so that it can be selected easily. Default: "convt" + number of previous ConvT layers created.

6.6 UpSampling

```
UpSampling(parent, size, interpolation, name)
```

Upsampling layer. Repeats the corresponding dimensions of the data by themselves. Example:

```
in = eddl.Input([batch, [1, 28, 28]]); // Example image data mnist
up1 = eddl.UpSampling(in, [2, 2]); // Upsampling of size 2x2
```

Arguments:

- **parent** - Previous layer to which we will apply the convolution.
- **size** - The upsampling factors for each dimension.
- **interpolation** - One of nearest or bilinear. Default: nearest.

6.7 Transpose

```
Transpose(parent, dims, name)
```

Transposes the dimensions of the input according to a given pattern. Example:

```
in = eddl.Input([batch,784]);
l = eddl.Reshape(in, [batch,1,28,28]);
perm = eddl.Transpose(l, [0,2,3,1]); // Give us l as [batch, 28, 28, 1}
```

Arguments:

- **parent** - Previous layer to which we will apply the reshape.
- **dims** - Tuple of integers. Permutation pattern.
- **name** - Layer name so that it can be selected easily. Default: "transpose" + number of previous Transpose layers created.

6.8 Depthwise and Separable Convolutions

At Conv layer 6.4, if $groups = input\_channels$, then it is Depthwise. If $groups = input\_channels$, and $kernel\_size = (K, 1)$, (and before is a Conv2d layer with groups=1 and kernel_size=(1, K)), then it is Separable.

6.9 Embedding

```
Embedding (input_dim, output_dim, name)
```

Turns positive integers (indexes) into dense vectors of fixed size. eg. [[4], [20]] → [[0.25, 0.1], [0.6, -0.2]]. This layer can only be used as the first layer in a model.

Arguments:

- **input_dim** - Size of the vocabulary.
- **output_dim** - Dimension of the dense embedding.
- **name** - Layer name so that it can be selected easily. Default: "embedding" + number of previous Embedding layers created.

6.10 Activation

```
Activation(parent, activation, name)
```

Applies an activation function to a layer. We must access them through ***activations***. Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
l = eddl.activations.ReLU(eddl.Dense(in,1024));
```

Arguments:

- **parent** - Previous layer to which we will apply the activation.
- **act** - Name of activation function to use. Complete list next.
- **name** - Layer name so that it can be selected easily. Default: "activation" + number of previous Activation layers created.

Available Activation Functions:

- **Sigmoid** - Applies the element-wise function $Sigmoid(x) = \frac{1}{1+\exp(-x)}$.
- **Tanh** - Applies the element-wise function $Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.
- **ReLU** - Applies the element-wise function $ReLU(x) = \max(0, x)$.
- **Softmax** - Applies the Softmax function to an n-dimensional input Tensor rescaling them so that the elements of the n-dimensional output Tensor lie in the range (0,1) and sum to 1. Defined as: $Softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$.

6.11 Dropout

```
Dropout(parent, rate, name)
```

Applies Dropout to the input. Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting. Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
l = eddl.Activation(eddl.Dense(in, 1024), "relu");
drop = eddl.Dropout(l, 0.5); // Applies a dropout over l
```

Arguments:

- **parent** - Previous to which we will apply the dropout.
- **rate** - between 0 and 1. Fraction of the input units to drop.
- **name** - Layer name so that it can be selected easily. Default: "dropout" + number of previous Dropout layers created.

6.12 GaussianNoise

```
GaussianNoise(parent, stdev, name)
```

Apply additive zero-centered Gaussian noise. This is useful to mitigate overfitting (you could see it as a form of random data augmentation). Gaussian Noise (GS) is a natural choice as corruption process for real valued inputs. As it is a regularization layer, it is only active at training time. Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
// Apply GaussianNoise with stddved 0.1 to Dense Layer with 1024 neurons
gs1 = eddl.GaussianNoise(eddl.Dense(in, 1024), 0.1);
```

Arguments:

- **parent** - Previous layer to which we will apply the dropout.
- **stdev** - Standard deviation of the noise distribution.
- **name** - Layer name so that it can be selected easily. Default: "gaussian_noise" + number of previous GaussianNoise layers created.

6.13 BatchNormalization

```
BatchNormalization(parent, momentum, epsilon, affine, name)
```

This layer type normalizes the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
// Apply BatchNormalization to Dense Layer with 1024 neurons
bn1 = eddl.BatchNormalization(eddl.Dense(in, 1024));
```

Arguments:

- **parent** - Previous layer to which we will apply the dropout.
- **momentum** - Momentum for the moving mean and the moving variance. Default: 0.99.
- **epsilon** - Small float added to variance to avoid dividing by zero. Default: 0.001.
- **affine** - A boolean value that when set to true, this module has learnable affine parameters. Default: true.
- **name** - Layer name so that it can be selected easily. Default: "batchnorm" + number of previous BatchNormalization layers created.

6.14 Pooling

A pooling layer is another building block of a CNN. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently.

6.14.1 MaxPool

```
MaxPool(parent, pool_size,
        strides, padding, name)
```

Max pooling operation for spatial data. Example:

```
in = eddl.Input([batch, 1,256,256]);
conv1 = eddl.Activation(eddl.Conv(in, 32, [3,3]), "relu");
pool1 = eddl.MaxPool(conv1, [2,2]);
```

Arguments:

- **parent** - Previous layer to which we will apply the Max Pooling.
- **pool_size** - An integer or tuple/list of 2 integers, specifying the height and width of the Max Pooling window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides** - An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Default: Same as pool_size.
- **padding** - One of "valid" or "same". Default: "valid".
- **name** - Layer name so that it can be selected easily. Default: "maxpool" + number of previous MaxPool layers created.

6.14.2 GlobalMaxPool

```
GlobalMaxPool(parent, name)
```

Global max pooling operation for spatial data. Example:

```
in = eddl.Input([batch, 1,256,256]);
conv1 = eddl.Activation(eddl.Conv(in, 32, [3,3]), "relu");
pool1 = eddl.GlobalMaxPool(conv1);
```

Arguments:

- **parent** - Previous layer to which we will apply the Global Max Pooling.
- **name** - Layer name so that it can be selected easily. Default: "globalmaxpool" + number of previous GlobalMaxPool layers created.

6.14.3 AveragePool

```
AveragePool(parent, pool_size,
            strides, padding,
            name)
```

Average pooling operation for spatial data. Example:

```
in = eddl.Input([batch, 1,256,256]);
conv1 = eddl.Activation(eddl.Conv(in, 32, [3,3]), "relu");
pool1 = eddl.AveragePool(conv1, [2,2]);
```

Arguments:

- **parent** - Previous layer to which we will apply the Average Pooling.
- **pool_size** - An integer or tuple/list of 2 integers, specifying the height and width of the Average Pooling window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides** - An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions.
- **padding** - One of "valid" or "same".
- **name** - Layer name so that it can be selected easily. Default: "avgpool" + number of previous AveragePool layers created.

6.14.4 GlobalAveragePool

```
GlobalAveragePool(parent, name)
```

Global average pooling operation for spatial data. Example:

```
in = eddl.Input([batch, 1,256,256]);
conv1 = eddl.Activation(eddl.Conv(in, 32, [3,3]), "relu");
pool1 = eddl.GlobalAveragePool(conv1);
```

Arguments:

- **parent** - Previous layer to which we will apply the Global Average Pooling.
- **name** - Layer name so that it can be selected easily. Default: "globalavgpool" + number of previous GlobalAveragePool layers created.

6.15 Merge

The function of the Merge layers is to take several input layers and merge them by performing a certain operation so that we obtain a resulting layer.

6.15.1 Add

```
Add(layers, name)
```

Layer that adds a list of inputs. It takes as input a list of layers, all of the same shape, and returns a single layer (also of the same shape). Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
l1 = eddl.Dense(in, 512);
l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their sum
l3 = eddl.Add([l1, l2]);
```

Arguments:

- **layers** - List of input layers to perform the add operation.
- **name** - Layer name so that it can be selected easily. Default: ”add” + number of previous Add layers created.

6.15.2 Subtract

```
Subtract(layers, name)
```

Layer that subtract a list of inputs. It takes as input a list of layers, all of the same shape, and returns a single layer (also of the same shape). Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
l1 = eddl.Dense(in, 512);
l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their subtraction
l3 = eddl.Subtract([l1, l2]);
```

Arguments:

- **layers** - List of input layers to perform the add operation.
- **name** - Layer name so that it can be selected easily. Default: ”subtract” + number of previous Subtract layers created.

6.15.3 Concat

```
Concat(layers, name)
```

Layer that concatenates a list of inputs. It takes as input a list of layers and returns the concatenation of all inputs. Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
l1 = eddl.Dense(in, 512);
l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their concatenation
l3 = eddl.Concat([l1, l2]);
```

Arguments:

- **layers** - List of input layers to perform the concatenation operation.
- **name** - Layer name so that it can be selected easily. Default: ”concat” + number of previous Concat layers created.

6.15.4 MatMul

```
MatMul(layers, name)
```

The output of this layer is the matrix product of the two input layers provided in a Python list. Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
l1 = eddl.Dense(in, 512);
l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their matrix multiplication
l3 = eddl.MatMul([l1, l2]);
```

Arguments:

- **layers** - List of input layers to perform the matrix multiplication operation.
- **name** - Layer name so that it can be selected easily. Default: ”matmul” + number of previous MatMul layers created.

6.15.5 Average

```
Average(layers, name)
```

Layer that averages a list of inputs. It takes as input a list of layers, all of the same shape, and returns a single tensor (also of the same shape). Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
l1 = eddl.Dense(in, 512);
l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their average
l3 = eddl.Average([l1, l2]);
```

Arguments:

- **layers** - List of input layers to perform the average operation.
- **name** - Layer name so that it can be selected easily. Default: ”average” + number of previous Average layers created.

6.15.6 Maximum

```
Maximum(layers, name)
```

Layer that computes the maximum (element-wise) a list of inputs. It takes as input a list of layers, all of the same shape, and returns a single tensor (also of the same shape). Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
l1 = eddl.Dense(in, 512);
l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their maximum
l3 = eddl.Maximum([l1, l2]);
```

Arguments:

- **layers** - List of input layers to perform the maximum operation.
- **name** - Layer name so that it can be selected easily. Default: ”maximum” + number of previous Maximum layers created.

6.15.7 Minimum

```
Minimum(layers, name)
```

Layer that computes the minimum (element-wise) a list of inputs. It takes as input a list of layers, all of the same shape, and returns a single tensor (also of the same shape). Example:

```
in = eddl.Input([batch, 784]); // Example image data mnist
l1 = eddl.Dense(in, 512);
l2 = eddl.Dense(in, 512);
// Take l1 and l2 outputs and get their minimum
l3 = eddl.Minimum([l1, l2]);
```

Arguments:

- **layers** - List of input layers to perform the minimum operation.
- **name** - Layer name so that it can be selected easily. Default: ”minimum” + number of previous Minimum layers created.

6.16 Recurrent Layers

6.16.1 GRU

```
RNN(parent, units, activation, unroll
    recurrent_activation, reset_after, use_bias,
    dropout, recurrent_dropout, implementation,
    return_sequences, return_state, go_backwards,
    stateful)
```

Gated Recurrent Unit - Cho et al. 2014.

Arguments:

- **parent** - Previous layer to which we will apply the GRU.
- **units** - Positive integer, dimensionality of the output space.
- **activation** - Activation function to use. Default: 'linear'.
- **unroll** - If True, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences. Default: false.
- **recurrent_activation** – Activation function to use for the recurrent step. Default: 'linear'.
- **reset_after** – GRU convention (whether to apply reset gate after or before matrix multiplication). False = "before", True = "after" (CuDNN compatible). Default: false.
- **use_bias** - Boolean, whether the layer uses a bias vector. Default: true.
- **dropout** - Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs. Default: 0.
- **recurrent_dropout** - Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state. Default: 0.
- **implementation** - Implementation mode, either 1 or 2. Mode 1 will structure its operations as a larger number of smaller dot products and additions, whereas mode 2 will batch them into fewer, larger operations. These modes will have different performance profiles on different hardware and for different applications. Default: 1.
- **return_sequences** - Boolean. Whether to return the last output in the output sequence, or the full sequence. Default: false.
- **return_state** - Boolean. Whether to return the last state in addition to the output. Default: false.
- **go_backwards** - If True, process the input sequence backwards and return the reversed sequence. Default: false.
- **stateful** - If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch. Default: false.

6.16.2 LSTM

```
LSTM(parent, units, num_layers, use_bias, dropout, bidirectional name)
```

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

Arguments:

- **units** - Positive integer, dimensionality of the output space.
- **num_layers** - Number of recurrent layers. E.g., setting num_layers=2 would mean stacking two LSTMs together to form a stacked LSTM, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1.
- **use_bias** - Boolean, whether the layer uses a bias vector. Default: true.
- **dropout** – If non-zero, introduces a Dropout layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to dropout. Default: 0
- **bidirectional** – If true, becomes a bidirectional LSTM. Default: false
- **name** - Layer name so that it can be selected easily. Default: ”lstm” + number of previous LSTM layers created.

6.17 Initializers

Initializations define the way to set the initial random weights of some layers. We must access them through ***initializers***. Example:

```
my_layer.SetWeights(eddl.initializers.MYINIT(params))
```

Where *MYINIT* is one of the following.

6.17.1 Constant

```
Constant (value)
```

Generates tensors initialized to a constant value. Example:

```
my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Set to 0 all values of my_dense
my_dense.SetWeights(eddl.initializers.Constant(0));
```

Arguments:

- **value** - The value of the generator tensors.

6.17.2 RandomNormal

```
RandomNormal (mean, stdev, seed)
```

Initializer that generates tensors with a normal distribution. Example:

```
my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense with a normal distribution
my_dense.SetWeights(eddl.initializers.RandomNormal(0.0, 0.05));
```

Arguments:

- **mean** - Mean of the random values to generate.
- **stdev** - Standard deviation of the random values to generate.
- **seed** - Used to seed the random generator. Default: 0.

6.17.3 RandomUniform

```
RandomUniform (minval, maxval, seed)
```

Initializer that generates tensors with a uniform distribution. Example:

```
my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense with a uniform distribution with seed 42
my_dense.SetWeights(eddl.initializers.RandomUniform(-0.5, 0.05, 42));
```

Arguments:

- **minval** - Lower bound of the range of random values to generate.
- **maxval** - Upper bound of the range of random values to generate. Defaults to 1 for float types.
- **seed** - Used to seed the random generator. Default: 0.

6.17.4 Identity

```
Identity (gain)
```

Initializer that generates the identity matrix. Only use for 2D matrices. If the desired matrix is not square, it pads with zeros on the additional rows/columns. Example:

```
my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense with the identity
my_dense.SetWeights(eddl.initializers.Identity(1.0));
```

Arguments:

- **gain** - Multiplicative factor to apply to the identity matrix.

6.17.5 Orthogonal

```
Orthogonal (gain, seed)
```

Initializer that generates a random orthogonal matrix. Example:

```
my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense with a random orthogonal matrix
my_dense.SetWeights(eddl.initializers.Orthogonal(1.0));
```

Arguments:

- **gain** - Multiplicative factor to apply to the orthogonal matrix.
- **seed** - Used to seed the random generator. Default: 0.

6.17.6 Glorot normal

```
GlorotNormal (seed)
```

Glorot normal initializer, also called Xavier normal initializer, that draws values for the weights from a truncated normal distribution centered on 0 with $stdev = sqrt(2/(fan\_in + fan\_out))$, where fan_in is the number of input units in the weight tensor and fan_out is the number of output units in the weight tensor. Example:

```
my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense following the exposed procedure
my_dense.SetWeights(eddl.initializers.GlorotNormal());
```

Arguments:

- **seed** - Used to seed the random generator. Default: 0.

6.17.7 Glorot uniform

```
GlorotUniform (seed)
```

Glorot uniform initializer, also called Xavier uniform initializer, that draws values for the weights from a uniform distribution within [-limit, limit] where limit is $sqrt(6/(fan\_in + fan\_out))$, fan_in is the number of input units in the weight tensor and fan_out is the number of output units in the weight tensor. Example:

```
my_dense = eddl.Dense(in, 1024); // Dense layer with 1024 neurons
// Initializes all values of my_dense following the exposed procedure
my_dense.SetWeights(eddl.initializers.GlorotUniform());
```

Arguments:

- **seed** - Used to seed the random generator. Default: 0.

6.18 Additional Methods

All layers have a number of methods in common:

- **layer.GetWeights()**: returns the weights of the layer.
- **layer.GetBias()**: returns the bias of the layer.
- **layer.SetWeights(weights)**: sets the weights of the layer (with the same shapes as the output of get_weights).
- **layer.SetBias(bias)**: sets the bias of the layer (with the same shapes as the output of get_bias).

We can also take intermediate layers of a model as by his name or index:

```
intermediate_layer = eddl.GetLayer(my_model, "layer_name")
intermediate_layer2 = eddl.GetLayer(my_model, 2)
```

7 EDDLL – MODELS in Python

To define a network we have to specify a list of input and output layers:

```
net=eddl.Model(list_of_inputs, list_of_outpus)
```

To finally build a network we have to attach the optimizer, list of loss functions and list of metrics:

```
eddl.build(net, optimizer, list_of_loss, list_of_metrics)
```

7.1 Optimizers

The following subsections describe the most common optimizers used in other Deep Learning libraries and that are going to be implemented in the EDDL library. The classes for the optimizers are subclases of the class ***Optimizer***. Nevertheless, in the Python API this is hidden and each optimizer has its own class to wrap the same class in the C++ implementation.

7.1.1 SGD

```
SGD (lr, momentum, decay, nesterov)
```

Implements stochastic gradient descent. Example:

```
sgd = eddl.optimizers.SGD(0.01, 0.9);
eddl.build(net, sgd, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **momentum** - Momentum factor. Default: 0.
- **decay** - Learning rate decay over each update. Default: 0.
- **nesterov** - Learning rate. Default: false.

7.1.2 RMSprop

```
RMSprop (lr, rho, epsilon, decay)
```

RMSProp optimizer. Example:

```
rmsprop = eddl.optimizers.RMSProp(0.01);
eddl.build(net, rmsprop, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **rho** - Smoothing constant. Default: 0.9.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **decay** - Learning rate decay over each update. Default: 0.

7.1.3 Adam

```
Adam (lr, beta_1, beta_2, epsilon, decay, amsgrad)
```

Implements Adam algorithm. Example:

```
adam = eddl.optimizers.Adam(0.001);
eddl.build(net, adam, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **beta_1** - $0 < beta < 1$. Generally close to 1. Default: 0.9.
- **beta_2** - $0 < beta < 1$. Generally close to 1. Default: 0.999.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **decay** - Learning rate decay over each update. Default: 0.
- **amsgrad** - Whether to apply the AMSGrad variant of this algorithm from the paper ”On the Convergence of Adam and Beyond”. Default: false.

7.1.4 Adagrad

```
Adagrad (lr, epsilon, decay)
```

Adagrad is an optimizer with parameter-specific learning rates, which are adapted according to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the learning rate. Example:

```
adagrad = eddl.optimizers.Adagrad(0.01);
eddl.build(net, adagrad, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **decay** - Learning rate decay over each update. Default: 0.

7.1.5 Adadelta

```
Adadelta(lr, rho, epsilon, decay)
```

Adadelta is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. This way, Adadelta continues learning even when many updates have been done. Compared to Adagrad, in the original version of Adadelta you don't have to set an initial learning rate. Example:

```
adadelta = eddl.optimizers.Adadelta(0.01, 0.99);
eddl.build(net, adadelta, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **rho** - Adadelta decay factor, corresponding to fraction of gradient to keep at each time step. Default: 0.95.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **decay** - Initial learning rate decay. Default: 0.

7.1.6 Adamax

```
Adamax (lr, beta_1, beta_2, epsilon, decay)
```

It is a variant of Adam based on the infinity norm. Example:

```
adamax = eddl.optimizers.Adamax(0.0001);
eddl.build(net, adamax, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **beta_1** - $0 < beta < 1$. Generally close to 1. Default: 0.9.
- **beta_2** - $0 < beta < 1$. Generally close to 1. Default: 0.999.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **decay** - Learning rate decay over each update. Default: 0.

7.1.7 Nadam

```
Nadam (lr, beta_1, beta_2, epsilon, schedule_decay)
```

Nesterov Adam optimizer. As Adam is essentially RMSprop with momentum, Nadam is Adam RMSprop with Nesterov momentum. Example:

```
nadam = eddl.optimizers.Nadam(0.01);
eddl.build(net, nadam, my_loss, my_metric);
```

Arguments:

- **lr** - Learning rate.
- **beta_1** - 0 < beta < 1. Generally close to 1. Default: 0.9.
- **beta_2** - 0 < beta < 1. Generally close to 1. Default: 0.999.
- **epsilon** - Term added to the denominator to improve numerical stability. Default: 1e-8.
- **schedule_decay** - 0 < schedule_decay < 1. Default: 0.004.

7.2 Learning Rate Schedulers

There are several methods to adjust the learning rate based on the number of epochs or by choosing some validation measurements. We must access them through ***callbacks***. To apply them, we can attach them as a parameter at the fit function:

```
eddl.fit(..., callbacks=[MyCallbacks])
```

7.2.1 StepLR

```
StepLR (step_size, gamma, last_epoch)
```

Sets the learning rate of each parameter group to the initial lr decayed by gamma every step_size epochs. When last_epoch=-1, sets initial lr as lr. Example:

```
callback step_lr = eddl.callbacks.StepLR(20); //Decay every 20 epochs
eddl.fit(..., callbacks=[step_lr]);
```

Arguments:

- **step_size** - Period of learning rate decay.
- **gamma** - Multiplicative factor of learning rate decay. Default: 0.1.
- **last_epoch** - The index of last epoch. Default: -1.

7.2.2 MultiStepLR

```
MultiStepLR(milestones, gamma, last_epoch)
```

Set the learning rate of each parameter group to the initial lr decayed by gamma once the number of epoch reaches one of the milestones. When last_epoch=-1, sets initial lr as lr. Example:

```
callback multystep_lr = eddlcallbacks..MultiStepLR([10,15,5]);
eddl.fit(..., callbacks=[multystep_lr]);
```

Arguments:

- **milestones** - List of epoch indices. Must be increasing.
- **gamma** - Multiplicative factor of learning rate decay. Default: 0.1.
- **last_epoch** - The index of last epoch. Default: -1.

7.2.3 ExponentialLR

```
ExponentialLR (gamma, last_epoch)
```

Set the learning rate of each parameter group to the initial lr decayed by gamma every epoch. When last_epoch=-1, sets initial lr as lr. Example:

```
callback exponential_lr = eddl.callbacks.ExponentialLR(0.87);
eddl.fit(..., callback=[exponential_lr]);
```

Arguments:

- **gamma** - Multiplicative factor of learning rate decay.
- **last_epoch** - The index of last epoch. Default: -1.

7.2.4 CosineAnnealingLR

```
CosineAnnealingLR (T_max, eta_min, last_epoch)
```

Set the learning rate of each parameter group using a cosine annealing schedule. Example:

```
callback cosine_lr = eddl.callbacks.CosineAnnealingLR(0.87);
eddl.fit(..., callback=[cosine_lr]);
```

Arguments:

- **T_max** - Maximum number of iterations.
- **eta_min** - Minimum learning rate. Default: 0.
- **last_epoch** - The index of last epoch. Default: -1.

7.2.5 ReduceLROnPlateau

```
ReduceLROnPlateau(metric, mode, factor,
                  patience, threshold,
                  threshold_mode, cooldown,
                  min_lr, eps)
```

Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This scheduler reads a metrics quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced. Example:

```
callback plateau_lr = eddl.callbacks.ReduceLROnPlateau();
eddl.fit(..., callback=[plateau_lr]);
```

Arguments:

- **metric** - One of 7.5.
- **mode** - One of min, max. In min mode, lr will be reduced when the quantity monitored has stopped decreasing; in max mode it will be reduced when the quantity monitored has stopped increasing. Default: 'min'.
- **factor** -Factor by which the learning rate will be reduced. new_lr = lr * factor. Default: 0.1.
- **patience** -Number of epochs with no improvement after which learning rate will be reduced. For example, if patience = 2, then we will ignore the first 2 epochs with no improvement, and will only decrease the LR after the 3rd epoch if the loss still hasn't improved then. Default: 10.
- **threshold** - Threshold for measuring the new optimum, to only focus on significant changes. Default: 1e-4.
- **threshold_mode** - One of rel, abs. In rel mode, dynamic_threshold = best * (1 + threshold) in 'max' mode or best * (1 - threshold) in min mode. In abs mode, dynamic_threshold = best + threshold in max mode or best - threshold in min mode. Default: 'rel'.
- **cooldown** - Number of epochs to wait before resuming normal operation after lr has been reduced. Default: 0.
- **min_lr** - A scalar or a list of scalars. A lower bound on the learning rate of all param groups or each group respectively. Default: 0.
- **eps** - Minimal decay applied to lr. If the difference between new and old lr is smaller than eps, the update is ignored. Default: 1e-8.

7.3 General Callbacks

There are another useful callbacks to get a view on internal states and statistics of the model during training.

7.3.1 History

Callback that records events into a History object. This callback is automatically applied to every model. The History object gets returned by the fit method of models.
Callback that records events into an object of the class **History**. This callback is automatically applied to every model, it is not necessary the programmer creates any object of this class explicitly. The **History** object is returned by the fit method of models.

7.3.2 Model Checkpoint

```
ModelCheckpoint(filepath, save_best_only, mode, period)
```

Save the model after every epoch.

```
callback model_checkpoint = eddl.callbacks.ModelCheckpoint('results/');
eddl.fit(..., callback=[model_checkpoint]);
```

Arguments:

- **filepath** - Path to save the model file.
- **save_best_only** - If true, the latest best model according to the quantity monitored will not be overwritten. Default: false.
- **mode** - One of auto, min, max. In 'auto' mode, the direction is automatically inferred from the name of the monitored quantity. Default: 'auto'.
- **period** - Interval (number of epochs) between checkpoints. Default: 1.

7.3.3 Lambda Callback

```
LambdaCallback(on_epoch_begin, on_epoch_end,
               on_batch_begin, on_batch_end,
               on_train_begin, on_train_end)
```

Callback for creating simple and custom callbacks on-the-fly. This type of callback is constructed with functions that will be called at the appropriate time. Note that the callbacks expects positional arguments, as:

- *on_epoch_begin* and *on_epoch_end* expect two positional arguments: epoch, logs
- *on_batch_begin* and *on_batch_end* expect two positional arguments: batch, logs
- *on_train_begin* and *on_train_end* expect one positional argument: logs

Examples of how to create lambda callbacks will be provided in the final version of the API documentation.

Arguments:

- **on_epoch_begin:** function to be called at the beginning of every epoch.
- **on_epoch_end:** function to be called at the end of every epoch.
- **on_batch_begin:** function to be called at the beginning of every batch.
- **on_batch_end:** function to be called at the end of every batch.
- **on_train_begin:** function to be called at the beginning of model training.
- **on_train_end:** function to be called at the end of model training.

7.4 Loss Functions

A loss function (or objective function, or optimization score function) is one of the parameters required to **build** a model. Available loss functions:

- **mse** - Creates a criterion that measures the mean squared error.
- **cross_entropy** - It is useful when training a classification problem with C classes.
- **bceloss** - measures the Binary Cross Entropy.

- **kullback_leibler_divergence**
- **poisson**

Example using cross_entropy loss:

```
...
eddl.build(__, __, [eddl.loss.cross_entropy], __, __);
```

Custom Loss Functions can be defined as follows: *examples of how to define custom loss functions will be provided in D2.1*

7.5 Metrics

A metric is a function that is used to judge the performance of your model. Available metrics:

- **binary_accuracy**
- **categorical_accuracy**
- **sparse_categorical_accuracy**
- **top_k_categorical_accuracy**
- **sparse_top_k_categorical_accuracy**

Example using categorical_accuracy as metric:

```
...
eddl.build(__, __, __, [eddl.metric.categorical_accuracy], __);
```

Custom Metrics can be defined as follows: *examples of how to define custom loss functions will be provided in D2.1*

7.6 Computing services

It is important to note that EDDL provides a hardware abstraction. The API is the same independently of the hardware to be used. It is only necessary to define the hardware to use when the network is build. It is also thought to provide the ability to train the different models in a distributed way.

When a model is build, by default it will use the CPU, but it is possible to provide an object specifying the computing services to use. Two options will be available, local computing services and distributed computing services.

7.6.1 Local

For local training we have only to indicate the device as follows:

```
cs = eddl.computing.CS_XXXX();
eddl.build(net, my_optimizer, [eddl.loss.MYLOSS], [eddl.metric.MYMETRIC], cs)
```

Where XXXX() is one of ***CPU***, ***GPU*** or ***FPGA***. You can use an integer with the number of threads or gpus to be used, or a binary list indicating the exact device. For example:

```
// local CPU with 6 threads
cs = eddl.computing.CS_CPU(4);
// local GPU using the first gpu of 4 installed
cs = eddl.computing.CS_GPU([1,0,0,0]);
// local GPU using the first gpu of 1 installed
cs = eddl.computing.CS_GPU([1]);
```

7.6.2 Distributed

For distributed training we have to provide an object that will be created by loading the configuration from a text file. The definitive version of the file format for specifying the resources will be defined in collaboration with the teams in charge of adapting the library to HPC architectures and Big Data environments.
An example of using a computing service for training in a distributed environment:

```
cs = eddl.computing.CS_Distributed( "cluster.cfg" )
eddl.build(net, my_optimizer, [eddl.loss.MYLOSS], [eddl.metricMYMETRIC], cs)
```

An example of the contents of the text file **cluster.cfg** is:

```
workernode-01  cpu-cores=32 max-cores=16 gpus=0 max-gpus=0 fpga=0 max-fpga=0
workernode-02  cpu-cores=32 max-cores=16 gpus=0 max-gpus=0 fpga=0 max-fpga=0
workernode-03  cpu-cores=32 max-cores=16 gpus=0 max-gpus=0 fpga=0 max-fpga=0
workernode-04  cpu-cores=32 max-cores=16 gpus=0 max-gpus=0 fpga=0 max-fpga=0
gpu-17         cpu-cores=1  max-cores=0  gpus=2 max-gpus=1 fpga=0 max-fpga=0
gpu-18         cpu-cores=1  max-cores=0  gpus=2 max-gpus=1 fpga=0 max-fpga=0
```

The runtime will execute a process in the CPU of those computers with GPUs, but not for running the training procedure if the maximum of cores is set to zero. The computations will be carried out by the GPU(s).

7.7 Training

To train a network we have to provide the input and output data according to the input and output layers defined. This data is loaded using the tensor functionalities:

```
X=eddl.T("trX.bin")
Y=eddl.T("trY.bin")
```

Finally, train the model:

```
eddl.fit(net,[X],[Y],batch,epochs)
```

8 EDDLL – UTILS for Python

On the other hand we have a series of useful functions for different purposes.

8.1 Save Model

Saves the weights and architecture of the network to be able to load them later.

```
eddl.utils.SaveModel(net, "my_model.pt")
```

8.2 Load Model

Load the weights and the architecture of a network.

```
eddl.utils.LoadModel("trained_model.pt")
```

8.3 Get Layer

Gets by reference a layer of a network.

```
middle_layer = eddl.utils.GetLayer(model, "layer_name")
```

8.4 Trainable Models and Layers

It is possible to freeze and unfreeze layers and models. By default all layers are trainable.

```
eddl.utils.SetTrainable(model, false)
eddl.utils.SetTrainable(layer, true)
```

8.5 Zoo models

It is possible to load a model architecture as starting point. Avaible models are:

- VGG: vgg11, vgg13, vgg16, vgg19
- ResNet: resnet18, resnet34, resnet50, resnet101, resnet151
- DenseNet: densenet101, densenet161, densenet169, densnet201

We can laod them as:

```
myModel = eddl.utils.ZooModels("architecture_name")
```

8.6 Datasets

There exist several public datasets we can load directly:

- mnist: The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.
- cifar10: The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- cifar100: This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class.

For using them we need to call corresponding function by substituting the word DATASET by the name of the dataset in the following example:

```
eddl.utils.DownloadDATASET()
```

8.7 Print summary

Prints a summary of a Net.

```
eddl.utils.summary(net)
```

8.8 Plot

Generates a plot of a network and saves it in a file. In the following example the format is PDF.

```
eddl.utils.plot(net, "model.pdf")
```

To get other formats just change the extension of the file:

```
eddl.utils.plot(net, "model.png")
```

9 EDDLL – EXAMPLES in Python

With very few lines of code we are able to define the network topology, read the data and train the model.

9.1 MultiLayer Perceptron (MLP) - MNIST

In the next snippet we create a MLP composed of an input that receives a vetor of size 784 as Input and then iteratively we add 5 hidden layers of 1024 neurons followed by a relu activation. Finally, an output layer of 10 neurons is added (the number of MNIST classes) and we define our model from the input and output layers. We indicate the optimizer, cost function and metric, load the data and train.

```
main():
    batch=1000;
    in=eddl.Input((batch,784));
    l=in;
    for i in range(5):
        l=eddl.Activation(eddl.Dense(l,1024),"relu");
    out=eddl.Activation(eddl.Dense(l,10),"softmax");

    net=eddl.Model([in],[out]);

    sgd = eddl.SGD(0.01, 0.9);

    eddl.build(net,sgd,["cross_entropy"},[eddl.metric.categorical_accuracy]);

    X=eddl.T("trX.bin");
    Y=eddl.T("trY.bin");

    eddl.div(X,255.0);

    eddl.fit(net,[X],[Y],batch,100);
```

9.2 ResNet

The following snippet shows how to create a ResNet model. Also introduces how to use a learning rate scheduler.

```
ResBlock(in, k, n):
    l=in;
    for i in range(n):
        l=eddl.Activation(eddl.Conv(l,[k,3,3],[1,1]),"relu");

    // adapt depth of input
    in=eddl.Conv(in,[k,1,1],[1,1]);
    // add input and last
    l=eddl.Add([in,l]);

    // reduce size
    l=eddl.Conv(l,[k,3,3],[2,2]);
    return l;


main():
    // download MNIST data
    eddl.DownloadMnist();

    batch=128;

    // network
    in=eddl.Input([batch,784]);

    l=eddl.Reshape(in, {batch,1,28,28]);

    l=eddl.Activation(eddl.Conv(l,[16,3,3],[2,2]),"relu");
    l=eddl.Activation(eddl.Conv(l,[32,3,3],[2,2]),"relu");
    l=eddl.Activation(eddl.Conv(l,[64,3,3],[2,2]),"relu");
    l=eddl.Activation(eddl.Conv(l,[128,3,3],[2,2]),"relu");
```

```
k=16
for i in range(3):
     k*=2
     l=ResBlock(l,k,2);

l=eddl.Reshape(l,[batch,-1]);

l=eddl.Activation(eddl.Dense(l,1024),"relu");

out=eddl.Activation(eddl.Dense(l,10),"softmax");

// net define input and output layers list
net=eddl.Model([in],[out]);

// plot the model
eddl.plot(net,"model.pdf");

// get some info from the network
eddl.info(net);

// Attach an optimizer and a list of error criteria and metrics
// size of error criteria and metrics list must match with size
// of list of outputs. Optionally put a DEVICE where the net will run
sgd = eddl.SGD(0.01, 0.9);
stepScheduler = eddl.StepLR(25);
// Apply the scheduler to the optimizer
eddl.asign_scheduler(sgd, stepScheduler);
eddl.build(net, sgd,["cross_entropy"},["categorical_accuracy"}, DEV_CPU);


// read data
X=eddl.T("trX.bin");
Y=eddl.T("trY.bin");

eddl.div(X,255.0);

// training, list of input and output tensors, batch, epochs
eddl.fit(net,[X],[Y],batch,100);

// Evaluate test
tX=eddl.T("tsX.bin");
tY=eddl.T("tsY.bin");
eddl.div(tX,255.0);

eddl.evaluate(net,[tX],[tY]);
```

9.3 Simple U-Net

The following snippet shows how to create a U-Net model.

```
main():
   batch=64;
   in=eddl.Input([batch,1,256,256]);

   /* DOWN */
   down1 = eddl.Activation(eddl.Conv(in, 32, [3,3], "same"), "relu");
   down1 = eddl.Activation(eddl.Conv(down1, 32, [3,3], "same"), "relu");
   pool1 = eddl.MaxPool(down1, [2,2]);

   down2 = eddl.Activation(eddl.Conv(pool1, 64, [3,3], "same"), "relu");
   down2 = eddl.Activation(eddl.Conv(down2, 64, [3,3], "same"), "relu");
   pool2 = eddl.MaxPool(down2, [2,2]);

   down3 = eddl.Activation(eddl.Conv(pool2, 128, [3,3], "same"), "relu");
   down3 = eddl.Activation(eddl.Conv(down3, 128, [3,3], "same"), "relu");
   pool3 = eddl.MaxPool(down3, [2,2]);
```

```
    /* Middle */
    middle = eddl.Activation(eddl.Conv(pool3, 256, [3,3], "same"), "relu");
    middle = eddl.Dropout(middle, 0.5);

    /* UP */
    up1 = eddl.UpSampling(middle, [2,2]);
    up1 = eddl.Concat([down3, up1]);
    up1 = eddl.Activation(eddl.Conv(up1, 128, [3,3], "same"), "relu");
    up1 = eddl.Activation(eddl.Conv(up1, 128, [3,3], "same"), "relu");

    up2 = eddl.UpSampling(up1, [2,2]);
    up2 = eddl.Concat([down2, up2]);
    up2 = eddl.Activation(eddl.Conv(up2, 128, [3,3], "same"), "relu");
    up2 = eddl.Activation(eddl.Conv(up2, 128, [3,3], "same"), "relu");

    up3 = eddl.UpSampling(up2, [2,2]);
    up3 = eddl.Concat([down1, up3]);
    up3 = eddl.Activation(eddl.Conv(up3, 128, [3,3], "same"), "relu");
    up3 = eddl.Activation(eddl.Conv(up3, 128, [3,3], "same"), "relu");

    out = eddl.Activation(eddl.Conv(up3, 1, [3,3]), "sigmoid");

    net=eddl.Model([in],[out]);
    adam = eddl.Adam(0.001);
    eddl.build(net, adam, [eddl.loss.bce],[eddl.metric.binary_accuracy]);
```

9.4 Transfer Learning

The following snippet shows how to load a model architecture and reuse it for a custom task.

```
main():
    batch=64;
    vgg16 = edd.zoo_models("vgg16");
    /* we can find 'last_pool' with eddl.summary(vgg16) */
    lastPool = eddl.GetLayer(vgg16, "last_pool")
    newOut = eddl.Activation(eddl.Dense(lastPool, 10), "softmax");

    net=eddl.Model([vgg16],[newOut]);

    adam = eddl.Adam(0.001);
    eddl.build(net, adam,[eddl.loss.cross_entropy],[eddl.metric.categorical_accuracy]);
```

10 ECVL – API Examples

In this section a list of ECVL *C++* API examples is reported.

10.1 Read and Write

10.1.1 Read

```
bool ecvl::ImRead(const filesystem::path &filename,
                  Image &dst);
```

Loads an image from the specified file. If the image cannot be read for any reason, the function creates an empty Image and returns false.

Arguments:

- **filename** - A filesystem::path (platform indipendent) identifying the file name.
- **dst** - Image in which data will be stored.
- **return** - true if the Image is correctly read, false otherwise.

10.1.2 Write

```
bool ecvl::ImWrite(const filesystem::path &filename,
                   const Image &src);
```

Stores the input *Image* into a specified file. The *Image* format is chosen based on the filename extension. The following sample shows how to create a BGR *Image* and save it to the PNG file "test.png":

```
#include "ecvl/core.h"

using namespace std;
using namespace ecvl;
using namespace filesystem;

int main()
{
    // Create BGR Image
    Image img({ 500, 500, 3 }, DataType::uint8, "xyc", ColorType::BGR);

    // Populate Image with pseudo-random data
    for (int r = 0; r < img.dims_[1]; ++r) {
        for (int c = 0; c < img.dims_[0]; ++c) {
            *img.Ptr({ c, r, 0 }) = 255;
            *img.Ptr({ c, r, 1 }) = (r / 2) % 255;
            *img.Ptr({ c, r, 2 }) = (r / 2) % 255;
        }
    }

    ImWrite(path("./test.png"), img);

    return EXIT_SUCCESS;
}
```

Arguments:

- **filename** - A filesystem::path identifying the output file name.
- **dst** - Image to be saved.
- **return** - true if the Image is correctly written, false otherwise.

10.2 Basic Image Processing

10.2.1 Mirroring

```
void ecvl::Mirror2D(const ecvl::Image &src,
                    ecvl::Image &dst);
```

Horizontally flips an *Image*.

Arguments:

- **src** - The input Image.
- **dst** - The output mirrored Image.

10.2.2 Flip

```
void ecvl::Flip2D(const ecvl::Image &src,
                  ecvl::Image &dst);
```

Vertically flips an *Image*.

Arguments:

- **src** - The input Image.
- **dst** - The output flipped Image.

10.2.3 Resize

```
void ResizeDim(const ecvl::Image& src,
          ecvl::Image& dst,
          const std::vector<int>& newdims,
          InterpolationType interp);
```

Resizes an *Image* to the given dimension.

Arguments:

- **src** - The input Image.
- **dst** - The output resized Image.
- **newdims** - `std::vector<int>` that specifies the new size of each dimension. The vector size must match the src Image dimensions, excluding the color channel.
- **interp** - Interpolation type to be used.
  Default is `InterpolationType::linear`.

10.2.4 Rescale

```
void ResizeScale(const ecvl::Image& src,
                 ecvl::Image& dst,
                 const std::vector<double>& scales,
                 InterpolationType interp = InterpolationType::linear);
```

Scales *Image* dimensions by a given factor.

Arguments:

- **src** - The input Image.
- **dst** - The output rescaled Image.
- **scales** - std::vector<double> that specifies the scale to apply to each dimension. The vector size must match the src Image dimensions, excluding the color channel.
- **interp** - Interpolation type to be used.
  Default is InterpolationType::linear.

10.2.5 Rotate

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

Rotates a *Image* of a given angle without changing its dimensions.

Arguments:

- **src** - The input Image.
- **dst** - The output rotated Image.
- **angle** - The rotation angle in degrees.
- **center** - A std::vector<double> representing the coordinates of the rotation center. If empty, the center of the image is used.
- **scale** - Optional scaling factor.
- **interp** - Interpolation type to be used.
  Default is InterpolationType::linear.

10.2.6 Rotate Full

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

Rotates an *Image* of a given angle ensuring to maintain all the pixels of the rotated image. *Image* dimensions could change.

Arguments:

- **src** - The input Image.
- **dst** - The output rotated Image.
- **angle** - The rotation angle in degrees.
- **scale** - Optional scaling factor.
- **interp** - Interpolation type to be used.
  Default is InterpolationType::linear.

10.2.7 Threshold

```
void Threshold(const Image& src,
               Image& dst,
               double thresh,
               double maxval,
               ThresholdingType thresh_type = ThresholdingType::BINARY);
```

Applies a fixed threshold to an input *Image*.

Arguments:

- **src** - Input Image on which to apply the threshold.
- **dst** - The output thresholded Image.
- **thresh** - Threshold value.
- **maxval** - The maximum values in the thresholded Image.
- **thresh_type** - Type of threshold to be applied. The default value is ThresholdingType::BINARY.

10.2.8 To Change Color Space

```
void ChangeColorSpace(const Image& src,
                      Image& dst,
                      ColorType new_type);
```

Changes the color space of the source *Image*.

Arguments:

- **src** - The input Image to convert in the new color space.
- **dst** - The output Image in the ”new_type” color space.
- **new_type** - The new color space in which the src Image must be converted.

10.3 Image Arithmetic

10.3.1 Saturation

```
template<typename ODT, typename IDT>
ODT saturate_cast(const IDT& v);
```

Converts a value of type IDT into a value of type ODT applying saturation.

Arguments:

- **v** - Input value (of any type).
- **return** - Input value after cast and saturation.

10.3.2 Negation

```
Image& Neg(Image& img);
```

Negates every value of an *Image*, and stores the result in the same *Image*.

Arguments:

- **img** - Image to be negated (in-place).
- **return** - Reference to the Image containing the result of the negation.

10.3.3 Addition

```
template<typename ST1, typename ST2>
void Add(const ST1& src1,
         const ST2& src2,
         Image& dst,
         bool saturate = true);
```

Takes two input values (that could be either scalars or *Image*(s)) and adds them together, storing the result into the destination *Image*.

Arguments:

- **src1** - Augend (first addend) Image.
- **src2** - Addend (second addend) Image.
- **dst** - Image into which save the result of the addition.
- **dst_type** - DataType that destination Image must have at the end of the operation.
- **saturate** - Whether to apply saturation or not. Default is true.

10.3.4 Subtraction

```
template<typename ST1, typename ST2>
void Sub(const ST1& src1,
         const ST2& src2,
         Image& dst,
         bool saturate = true);
```

Takes two input values (that could be either scalars or *Image*(s)) and subtracts the second from the first, storing the result into the destination *Image*.

Arguments:

- **src1** - Minuend Image.
- **src2** - Subtrahend Image.
- **dst** - Image into which save the result of the subtraction.
- **dst_type** - DataType that destination Image must have at the end of the operation.
- **saturate** - Whether to apply saturation or not. Default is true.

10.3.5 Multiplication

```
template<typename ST1, typename ST2>
void Mul(const ST1& src1,
         const ST2& src2,
         Image& dst,
         bool saturate = true)
```

Takes two input values (that could be either scalars or *Image*(s)) and multiplies them together, storing the result into the destination *Image*.

Arguments:

- **src1** - Multiplier (first factor) Image.
- **src2** - Multiplicand (second factor) Image.
- **dst** - Image into which save the result of the multiplication.
- **dst_type** - DataType that destination Image must have at the end of the operation.
- **saturate** - Whether to apply saturation or not. Default is true.

10.3.6 Division

```
template<typename ST1, typename ST2, typename ET = double>
void Div(const ST1& src1,
         const ST2& src2,
         Image& dst,
         bool saturate = true,
         ET epsilon = std::numeric_limits<double>::min())
```

Takes two input values (that could be either scalars or *Image*(s)) and divides the first by the second, storing the result in the destination *Image*.

Arguments:

- **src1** - Dividend (numerator) operand. Could be either a scalar or an Image.
- **src2** - Divisor (denominator) operand. Could be either a scalar or an Image.
- **dst** - Destination Image. It will store the final result. If dst is not empty, its DataType will be preserved. Otherwise, it will have the same DataType as src1 if it is an Image, src2 otherwise.
- **saturate** - Whether to apply saturation or not. Default is true.
- **epsilon** - Small value to be added to divisor pixel values before performing the division. If not specified by default it is the minimum positive number representable in a double. It is ignored if src2 is a scalar value.

10.4 GUI

10.4.1 Visualize

```
void ImShow(const Image& img);
```

Displays an *Image* in a *wxWidgets* frame.

Arguments:

- **src** - The input Image to be displayed.

The first version of the documentation of the ECVL library will be released in the deliverable 3.1 (D13). Anyway a draft version is already available and hosted at `http://imagelab.ing.unimore.it/ecvl`. A pdf version of the documentation is also attached to this document as Appendix A.

11 ECVL – EDDL Interfacing

The ECVL - EDDLL interfacing is based on two main functions that are responsible for the conversion between *Image*(s) and *Tensor*(s). They are described in this section.

11.1 Image to Tensor

```
tensor ImageToTensor(const Image& img);
```

Transforms an ECVL *Image* in a EDDL *Tensor*, converting its data to `float` and rearranging its channels to `"xyc"`, which is how EDDLL *Tensor* handles data in color images. Finally, *Image* data are copied into *Tensor* data.

Arguments:

- **img** - Image to be converted into Tensor.
- **return** - Output Tensor.

11.2 Tensor To Image

```
Image TensorToImage(const tensor& t)
```

Creates a `float` raw *Image* with `none` color space, and copies *Tensor* data into the *Image* data.

Arguments:

- **t** - Tensor to be converted into Image.
- **return** - Output Image.

11.3 Dataset To Tensor

```
tensor DatasetToTensor(vector<string> dataset, const std::vector<int>& dims);
```

Simplifies the usage of EDDLL with more than one ECVL *Image*. One Image at a time is read and resized accordingly. Then it is transformed into a *Tensor* with the *ImageToTensor* function.

Arguments:

- **dataset** - Vector of images path.
- **dims** - Dimensions of the dataset in the form: number of total images, number of channels of the images, and the width and height at which all the images has to be resized. $\{number\_of\_samples, number\_of\_channels, width, height\}$
- **return** - Resulting Tensor.

Appendix A

This appendix includes the documentation of the ECVL library as it is generated by Doxygen `http://www.doxygen.nl/` from the comments included in the code.

This does not mean the ECVL is developed, but at the programming level the profile and the comments describing each function have been prepared. This modus operandi follows the common way of developing software, defining the API in order that all the programmers involved in such development or that are going to use the library agree with the names of the functions, with the data structures used, and with the types of the parameters of each funtion.

Note that the contents of this appendix has its own index as it has been generated automatically.

Chapter 1

Documentation

ECVL is the European Computer Vision Library, under development within the European project DeepHealth. Here you can find the provisional doxygen documentation. Checkout the GitHub project `here`.

Chapter 2

Bug List

Member ecvl::Threshold (p. **34**) **(const Image** (p. **64**) **&src, Image** (p. **64**) **&dst, double thresh, double maxval, ThresholdingType thresh_type=ThresholdingType::BINARY** (p. **18**)**)**

Input and output Images may have different color spaces.

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 ecvl Namespace Reference

Classes

- struct **AddImpl**
- struct **AddImpl< Image, Image >**
- struct **AddImpl< Image, ST2 >**
- struct **AddImpl< ST1, Image >**
- struct **arithmetic_superior_type**
- struct **ConstContiguousIterator**
- class **ConstContiguousView**
- struct **ConstIterator**
- class **ConstView**
- struct **ContiguousIterator**
- class **ContiguousView**
- class **ContiguousViewXYC**
- struct **DivImpl**
- struct **DivImpl< Image, Image, ET >**
- struct **DivImpl< Image, ST2, ET >**
- struct **DivImpl< ST1, Image, ET >**
- class **Image**

  Image *(p. 64) class.*
- struct **ImageScalarAddImpl**
- struct **ImageScalarDivImpl**
- struct **ImageScalarMulImpl**
- struct **ImageScalarSubImpl**
- struct **Iterator**
- struct **larger_arithmetic_type**
- class **MetaData**
- struct **MulImpl**
- struct **MulImpl< Image, Image >**
- struct **MulImpl< Image, ST2 >**
- struct **MulImpl< ST1, Image >**
- struct **promote_superior_type**
- struct **ScalarImageDivImpl**
- struct **ScalarImageSubImpl**
- class **ShowApp**

***ShowApp** (p. 93) is a custom wxApp which allows you to visualize an ECVL **Image** (p. 64).*

- struct **SignedTable1D**
- struct **SignedTable2D**
- struct **StructAdd**
- struct **StructCopyImage**
- struct **StructDiv**
- struct **StructMul**
- struct **StructScalarNeg**
- struct **StructSub**
- struct **SubImpl**
- struct **SubImpl< Image, Image >**
- struct **SubImpl< Image, ST2 >**
- struct **SubImpl< ST1, Image >**
- struct **Table1D**
- struct **Table2D**
- class **View**
- class **wxImagePanel**

  ***wxImagePanel** (p. 113) creates a wxPanel to contain an **Image** (p. 64).*

Typedefs

- template<typename T , typename U >
  using **larger_arithmetic_type_t** = typename **larger_arithmetic_type**< T, U >::type
- template<typename T , typename U >
  using **arithmetic_superior_type_t** = typename **arithmetic_superior_type**< T, U >::type
- template<typename T , typename U >
  using **promote_superior_type_t** = typename **promote_superior_type**< T, U >::type
- template<DataType DT, DataType DU>
  using **promote_superior_type_dt** = **promote_superior_type_t**< TypeInfo_t< DT >, TypeInfo_t< DU > >

Enumerations

- enum **ColorType** {
  ColorType::none, **ColorType::GRAY**, **ColorType::RGB**, **ColorType::BGR**,
  ColorType::HSV, **ColorType::YCbCr** }

  Enum class representing the ECVL supported color spaces.
- enum **ThresholdingType** { **ThresholdingType::BINARY**, **ThresholdingType::BINARY_INV** }

  Enum class representing the ECVL threhsolding types.
- enum **InterpolationType** {
  InterpolationType::nearest, **InterpolationType::linear**, **InterpolationType::area**, **InterpolationType::cubic**,
  InterpolationType::lanczos4 }

  Enum class representing the ECVL interpolation types.

Functions

- template<DataType ODT, typename IDT >
  TypeInfo< ODT >::basetype **saturate_cast** (IDT v)

  Saturate a value (of any type) to the specified type.
- template<typename ODT , typename IDT >
  ODT **saturate_cast** (const IDT &v)

  Saturate a value (of any type) to the specified type.
- **Image** & **Neg** (**Image** &img)

  In-place negation of an ***Image*** *(p. 64).*
- void **Mul** (const **Image** &src1, const **Image** &src2, **Image** &dst, DataType dst_type, bool saturate=true)

  Multiplies two Image(s) and stores the result in a third ***Image*** *(p. 64).*
- void **Sub** (const **Image** &src1, const **Image** &src2, **Image** &dst, DataType dst_type, bool saturate=true)

  Subtracts two Image(s) and stores the result in a third ***Image*** *(p. 64).*
- void **Add** (const **Image** &src1, const **Image** &src2, **Image** &dst, DataType dst_type, bool saturate=true)

  Adds two Image(s) and stores the result in a third ***Image*** *(p. 64).*
- template<typename ST1 , typename ST2 >
  void **Add** (const ST1 &src1, const ST2 &src2, **Image** &dst, bool saturate=true)

  Adds two objects that could be either a scalar value or an ***Image*** *(p. 64), storing the result into a destination* ***Image*** *(p. 64). The procedure does not perform any type promotion.*
- template<typename ST1 , typename ST2 >
  void **Sub** (const ST1 &src1, const ST2 &src2, **Image** &dst, bool saturate=true)

  Subtracts two objects that could be either a scalar value or an ***Image*** *(p. 64), storing the result into a destination* ***Image*** *(p. 64). The procedure does not perform any type promotion.*
- template<typename ST1 , typename ST2 >
  void **Mul** (const ST1 &src1, const ST2 &src2, **Image** &dst, bool saturate=true)

  Multiplies two objects that could be either a scalar value or an ***Image*** *(p. 64), storing the result into a destination* ***Image*** *(p. 64). The procedure does not perform any type promotion.*
- template<typename ST1 , typename ST2 , typename ET = double>
  void **Div** (const ST1 &src1, const ST2 &src2, **Image** &dst, bool saturate=true, ET epsilon=std::numeric_↩
  limits< double >::min())

  Divides two objects that could be either a scalar value or an ***Image*** *(p. 64), storing the result into a destination* ***Image*** *(p. 64). The procedure does not perform any type promotion.*
- void **RearrangeChannels** (const **Image** &src, **Image** &dst, const std::string &channels)

  Changes the order of the ***Image*** *(p. 64) dimensions.*
- void **CopyImage** (const **Image** &src, **Image** &dst, DataType new_type= **DataType::none**)

  Copies the source ***Image*** *(p. 64) into the destination* ***Image*** *(p. 64).*
- bool **ImRead** (const std::string &filename, **Image** &dst)

  Loads an image from a file.
- bool **ImRead** (const **filesystem::path** &filename, **Image** &dst)
- bool **ImWrite** (const std::string &filename, const **Image** &src)

  Saves an image into a specified file.
- bool **ImWrite** (const **filesystem::path** &filename, const **Image** &src)
- void **ResizeDim** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< int > &newdims, **InterpolationType** interp= **InterpolationType::linear**)

  Resizes an ***Image*** *(p. 64) to the specified dimensions.*
- void **ResizeScale** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< double > &scales, **InterpolationType** interp= **InterpolationType::linear**)

  Resizes an ***Image*** *(p. 64) by scaling the dimentions to a given scale factor.*
- void **Flip2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

  Flips an ***Image*** *(p. 64).*
- void **Mirror2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

  Mirrors an ***Image*** *(p. 64).*

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

  *Rotates an **Image** (p. 64).*
- void **RotateFullImage2D** (const **ecvl::Image** &src, **ecvl::Image** &dst, double angle, double scale=1.0, **InterpolationType** interp= **InterpolationType::linear**)

  *Rotates an **Image** (p. 64) resizing the output accordingly.*
- void **ChangeColorSpace** (const **Image** &src, **Image** &dst, **ColorType** new_type)

  *Copies the source **Image** (p. 64) into destination **Image** (p. 64) changing the color space.*
- void **Threshold** (const **Image** &src, **Image** &dst, double thresh, double maxval, **ThresholdingType** thresh_type= **ThresholdingType::BINARY**)

  *Applies a fixed threshold to an input **Image** (p. 64).*
- double **OtsuThreshold** (const **Image** &src)

  Calculates the Otsu thresholding value.
- **ecvl::Image** **MatToImage** (const cv::Mat &m)

  *Convert a cv::Mat into an **ecvl::Image** (p. 64).*
- cv::Mat **ImageToMat** (const **Image** &img)

  *Convert an ECVL **Image** (p. 64) into OpenCV Mat.*
- template<typename T , typename U >
  promote_superior_type_t< T, U > **PromoteAdd** (T rhs, U lhs)
- template<typename T , typename U >
  promote_superior_type_t< T, U > **PromoteSub** (T rhs, U lhs)
- template<typename T , typename U >
  promote_superior_type_t< T, U > **PromoteMul** (T rhs, U lhs)
- template<typename T , typename U >
  promote_superior_type_t< T, U > **PromoteDiv** (T rhs, U lhs)
- **Image** **TensorToImage** (tensor &t)

  *Convert a EDDLL Tensor into an ECVL **Image** (p. 64).*
- tensor **ImageToTensor** (const **Image** &img)

  *Convert an ECVL **Image** (p. 64) into EDDLL Tensor.*
- tensor **DatasetToTensor** (vector< string > dataset, const std::vector< int > &dims)

  Convert a set of images into a single EDDLL Tensor.
- void **ImShow** (const **Image** &img)

  *Displays an **Image** (p. 64).*
- wxImage **WxFromImg** (**Image** &img)

  *Convert an ECVL **Image** (p. 64) into a wxImage.*
- **Image** **ImgFromWx** (const wxImage &wx)

  *Convert a wxImage into an ECVL **Image** (p. 64).*

7.1.1 Typedef Documentation

7.1.1.1 arithmetic_superior_type_t

```
template<typename T , typename U >
using ecvl::arithmetic_superior_type_t = typedef typename arithmetic_superior_type<T, U>↩
::type
```

Definition at line 32 of file type_promotion.h.

7.1.1.2 larger_arithmetic_type_t

```
template<typename T , typename U >
using ecvl::larger_arithmetic_type_t = typedef typename larger_arithmetic_type<T, U>::type
```

Definition at line 19 of file type_promotion.h.

7.1.1.3 promote_superior_type_dt

```
template<DataType DT, DataType DU>
using ecvl::promote_superior_type_dt = typedef promote_superior_type_t<TypeInfo_t<DT>,
TypeInfo_t<DU> >
```

Definition at line 51 of file type_promotion.h.

7.1.1.4 promote_superior_type_t

```
template<typename T , typename U >
using ecvl::promote_superior_type_t = typedef typename promote_superior_type<T, U>::type
```

Definition at line 48 of file type_promotion.h.

7.1.2 Enumeration Type Documentation

7.1.2.1 ColorType

```
enum ecvl::ColorType [strong]
```

Enum class representing the ECVL supported color spaces.

Enumerator

Enumerator

none	Special ColorType for Images that contain only data and do not have any ColorType
GRAY	Gray-scale ColorType
RGB	RGB ColorType
BGR	BGR ColorType
HSV	HSV ColorType
YCbCr	YCbCr ColorType

Definition at line 27 of file image.h.

7.1.2.2 InterpolationType

`enum ecvl::InterpolationType [strong]`

Enum class representing the ECVL interpolation types.

Enumerator

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

Definition at line 23 of file imgproc.h.

7.1.2.3 ThresholdingType

`enum ecvl::ThresholdingType [strong]`

Enum class representing the ECVL threhsolding types.

Enumerator

Enumerator

BINARY	$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$
BINARY_INV	$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

Definition at line 14 of file imgproc.h.

7.1.3 Function Documentation

7.1.3.1 Add() [1/2]

```
void ecvl::Add (
            const Image & src1,
            const Image & src2,
            Image & dst,
            DataType dst_type,
            bool saturate = true )
```

Adds two Image(s) and stores the result in a third **Image** (p. 64).

This procedure adds src1 and src2 Image(s) (src1 + src2) and stores the result in the dst **Image** (p. 64) that will have the specified DataType. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in	*src1*	Augend (first addend) **Image** (p. 64).
in	*src2*	Addend (second addend) **Image** (p. 64).
out	*dst*	**Image** (p. 64) into which save the result of the division.
in	*dst_type*	DataType that destination **Image** (p. 64) must have at the end of the operation.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

7.1.3.2 Add() [2/2]

```
template<typename ST1 , typename ST2 >
void ecvl::Add (
            const ST1 & src1,
            const ST2 & src2,
            Image & dst,
            bool saturate = true )
```

Adds two objects that could be either a scalar value or an **Image** (p. 64), storing the result into a destination **Image** (p. 64). The procedure does not perform any type promotion.

The procedure takes two input values (src1 and src2) and adds them together, storing the result into the destination image. If one of the operands is an **Image** (p. 64) and the other one is a scalar value, each pixel of the **Image** (p. 64) is increased by the scalar value, and the result is stored into dst. If src1 and src2 are both Image(s) the pixel-wise addition is applied and, again, the result is stored into dst.

Saturation is applied by default. If it is not the desired behavior change the saturate parameter to false.

In any case, the operation performed is dst = src1 + src2.

Parameters

Parameters

in	*src1*	Augend operand. Could be either a scalar or an **Image** (p. 64).
in	*src2*	Addend operand. Could be either a scalar or an **Image** (p. 64).
out	*dst*	Destination **Image** (p. 64). It will store the final result. If dst is not empty, its DataType will be preserved. Otherwise, it will have the same Data↩ Type as src1 if it is an **Image** (p. 64), src2 otherwise.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

.

Definition at line 252 of file arithmetic.h.

7.1.3.3 ChangeColorSpace()

```
void ecvl::ChangeColorSpace (
            const Image & src,
            Image & dst,
            ColorType new_type )
```

Copies the source **Image** (p. 64) into destination **Image** (p. 64) changing the color space.

The ChangeColorSpace procedure convert the color space of the source **Image** (p. 64) into the specified color space. New data are copied into destination **Image** (p. 64). Source and destination can be contiguous or not and can also be the same **Image** (p. 64).

Parameters

Parameters

in	*src*	The input **Image** (p. 64) to convert in the new color space.
out	*dst*	The output **Image** (p. 64) in the "new_type" color space.
in	*new_type*	The new color space in which the src **Image** (p. 64) must be converted.

Definition at line 168 of file imgproc.cpp.

7.1.3.4 CopyImage()

```
void ecvl::CopyImage (
            const Image & src,
            Image & dst,
            DataType new_type = DataType::none )
```

Copies the source **Image** (p. 64) into the destination **Image** (p. 64).

The **CopyImage()** (p. 20) procedure takes an **Image** (p. 64) and copies its data into the destination **Image** (p. 64). Source and destination cannot be the same **Image** (p. 64). Source cannot be a **Image** (p. 64) with **DataType::none** (p. 17). The optional new_type parameter can be used to change the DataType of the destination **Image** (p. 64). This function is mainly designed to change the DataType of an **Image** (p. 64), copying its data into a new **Image** (p. 64) or to copy an **Image** (p. 64) into a **View** (p. 111) as a patch. So if you just want to copy an **Image** (p. 64) as it is, use the copy constructor or = instead. Anyway, the procedure will handle all the possible situations that may happen trying to avoid unnecessary allocations. When the DataType is not specified the function will have the following behaviors:

- if the destination **Image** (p. 64) is empty the source will be directly copied into the destination.
- if source and destination have different size in memory or different channels and the destination is the owner of data, the procedure will overwrite the destination **Image** (p. 64) creating a new **Image** (p. 64) (channels and dimensions will be the same of the source **Image** (p. 64), pixels type (DataType) will be the same of the destination **Image** (p. 64) if they are not none or the same of the source otherwise).

- if source and destination have different size in memory or different channels and the destination is not the owner of data, the procedure will throw an exception.
- if source and destination have different color types and the destination is the owner of data, the procedure produces a destination **Image** (p. 64) with the same color type of the source.
- if source and destination have different color types and the destination is not the owner of data, the procedure will throw an exception.
- if source and destination are the same **Image** (p. 64), there are two options. If new_type is the same of the two Image(s) or it is **DataType::none** (p. 17), nothing happens. Otherwise, an exception is thrown. When the DataType is specified the function will have the same behavior, but the destination **Image** (p. 64) will have the specified DataType.

Parameters

Parameters

in	*src*	Source **Image** (p. 64) to be copied into destination **Image** (p. 64).
out	*dst*	Destination **Image** (p. 64) that will hold a copy of the source **Image** (p. 64). Cannot be the source **Image** (p. 64).
in	*new_type*	Desired type for the destination **Image** (p. 64) after the copy. If none (default) the destination **Image** (p. 64) will preserve its type if it is not empty, otherwise it will have the same type of the source **Image** (p. 64).

Definition at line 94 of file image.cpp.

7.1.3.5 DatasetToTensor()

```
tensor ecvl::DatasetToTensor (
            vector< string > dataset,
            const std::vector< int > & dims )
```

Convert a set of images into a single EDDLL Tensor.

Parameters

Parameters

in	*dataset*	Vector of all images path.
in	*dims*	Dimensions of the dataset in the form: number of total images, number of channels of the images, and the width and height at which all the images has to be resized. $\{number\_of\_samples, number\_of\_channels, width, height\}$

Returns

EDDLL Tensor.

Definition at line 28 of file support_eddll.cpp.

7.1.3.6 Div()

```
template<typename ST1 , typename ST2 , typename ET = double>
void ecvl::Div (
            const ST1 & src1,
            const ST2 & src2,
             Image & dst,
            bool saturate = true,
            ET epsilon = std::numeric_limits<double>::min() )
```

Divides two objects that could be either a scalar value or an **Image** (p. 64), storing the result into a destination **Image** (p. 64). The procedure does not perform any type promotion.

The procedure takes two input values (src1 and src2) and divides the first by the second, storing the result in the destination image. If src1 is an **Image** (p. 64) and src2 a scalar value all the pixels inside src1 are divided by src2 and the result is stored into dst. If src1 is a scalar value and src2 is an **Image** (p. 64) the opposite happens: all the pixel values of src2 divide the scalar value src1 and the result is stored into dst. If src1 and sr2 are both Image(s) the pixel-wise division is applied and, again, the result is stored into dst.

Saturation is applied by default. If it is not the desired behavior change the saturate parameter to false.

In the cases in which the divisor (denominator) is an **Image** (p. 64) an epsilon value is summed to each divisor pixel value before the division in order to avoid divisions by zero.

In any case, the operation performed is dst = src1 / src2.

Parameters

Parameters

in	*src1*	Dividend (numerator) operand. Could be either a scalar or an **Image** (p. 64).
in	*src2*	Divisor (denominator) operand. Could be either a scalar or an **Image** (p. 64).
out	*dst*	Destination **Image** (p. 64). It will store the final result. If dst is not empty, its DataType will be preserved. Otherwise, it will have the same Data↩ Type as src1 if it is an **Image** (p. 64), src2 otherwise.
in	*saturate*	Whether to apply saturation or not. Default is true.
in	*epsilon*	Small value to be added to divisor pixel values before performing the division. If not specified by default it is the minimum positive number representable in a double. It is ignored if src2 is a scalar value.

Returns

.

Definition at line 678 of file arithmetic.h.

7.1.3.7 Flip2D()

```
void ecvl::Flip2D (
            const ecvl::Image & src,
            ecvl::Image & dst )
```

Flips an **Image** (p. 64).

The Flip2D procedure vertically flips an **Image** (p. 64).

Parameters

Parameters

in	*src*	The input **Image** (p. 64).
out	*dst*	The output flipped **Image** (p. 64).

Definition at line 73 of file imgproc.cpp.

7.1.3.8 ImageToMat()

```
cv::Mat ecvl::ImageToMat (
            const Image & img )
```

Convert an ECVL **Image** (p. 64) into OpenCV Mat.

Parameters

Parameters

in	*img*	Input ECVL **Image** (p. 64).

Returns

Output OpenCV Mat.

Definition at line 98 of file support_opencv.cpp.

7.1.3.9 ImageToTensor()

```
tensor ecvl::ImageToTensor (
            const Image & img )
```

Convert an ECVL **Image** (p. 64) into EDDLL Tensor.

Parameters

Parameters

in	*img*	Input ECVL **Image** (p. 64).

Returns

EDDLL Tensor.

Definition at line 15 of file support_eddll.cpp.

7.1.3.10 ImgFromWx()

```
Image ecvl::ImgFromWx (
            const wxImage & wx )
```

Convert a wxImage into an ECVL **Image** (p. 64).

Parameters

Parameters

in	*wx*	Input wxImage.

Returns

ECVL **Image** (p. 64).

Definition at line 92 of file gui.cpp.

7.1.3.11 ImRead() [1/2]

```
bool ecvl::ImRead (
            const std::string & filename,
             Image & dst )
```

Loads an image from a file.

The function ImRead loads an image from the specified file. If the image cannot be read for any reason, the function creates an empty **Image** (p. 64) and returns false.

Parameters

Parameters

in	*filename*	A std::string identifying the file name. In order to be platform independent consider to use **ImRead(const filesystem::path& filename, Image& dst)** (p. 25) .
out	*dst*	**Image** (p. 64) in which data will be stored.

Returns

true if the image is correctly read, false otherwise.

Definition at line 10 of file imgcodecs.cpp.

7.1.3.12 ImRead() [2/2]

```
bool ecvl::ImRead (
            const filesystem::path & filename,
            Image & dst )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This variant of ImRead is platform independent.

Parameters

Parameters

in	*filename*	A **filesystem::path** (p. 87) identifying the file name.
out	*dst*	**Image** (p. 64) in which data will be stored.

Returns

true if the image is correctly read, false otherwise.

Definition at line 16 of file imgcodecs.cpp.

7.1.3.13 ImShow()

```
void ecvl::ImShow (
            const Image & img )
```

Displays an **Image** (p. 64).

The ImShow function instantiates a **ShowApp** (p. 93) and starts it with a wxEntry() call. The image is shown with its original size.

Parameters

Parameters

in	*img*	**Image** (p. 64) to be shown.

Definition at line 62 of file gui.cpp.

7.1.3.14 ImWrite() [1/2]

```
bool ecvl::ImWrite (
            const std::string & filename,
            const Image & src )
```

Saves an image into a specified file.

The function ImWrite saves the input image into a specified file. The image format is chosen based on the filename extension. The following sample shows how to create a BGR image and save it to the PNG file "test.png":

```
#include "ecvl/core.h"
using namespace std;
using namespace ecvl;
using namespace filesystem;
int main()
{
    // Create BGR Image
    Image img({ 500, 500, 3 }, DataType::uint8, "xyc", ColorType::BGR);

    // Populate Image with pseudo-random data
    for (int r = 0; r < img.dims_[1]; ++r) {
        for (int c = 0; c < img.dims_[0]; ++c) {
            *img.Ptr({ c, r, 0 }) = 255;
            *img.Ptr({ c, r, 1 }) = (r / 2) % 255;
            *img.Ptr({ c, r, 2 }) = (r / 2) % 255;
        }
    }
    ImWrite(path("./test.png"), img);
    return EXIT_SUCCESS;
}
```

Parameters

Parameters

in	*filename*	A std::string identifying the output file name. In order to be platform independent consider to use **ImWrite(const filesystem::path& filename, const Image& src)** (p. 26).
in	*src*	**Image** (p. 64) to be saved.

Returns

true if the image is correctly write, false otherwise.

Definition at line 21 of file imgcodecs.cpp.

7.1.3.15 ImWrite() [2/2]

```
bool ecvl::ImWrite (
            const filesystem::path & filename,
            const Image & src )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This variant of ImWrite is platform independent.

Parameters

Parameters

in	*filename*	A **filesystem::path** (p. 87) identifying the output file name.
in	*src*	**Image** (p. 64) to be saved.

Returns

true if the image is correctly write, false otherwise.

Definition at line 26 of file imgcodecs.cpp.

7.1.3.16 MatToImage()

```
Image ecvl::MatToImage (
            const cv::Mat & m )
```

Convert a cv::Mat into an **ecvl::Image** (p. 64).

Parameters

Parameters

in	*m*	Input OpenCV Mat.

Returns

ECVL image.

Definition at line 7 of file support_opencv.cpp.

7.1.3.17 Mirror2D()

```
void ecvl::Mirror2D (
            const ecvl::Image & src,
            ecvl::Image & dst )
```

Mirrors an **Image** (p. 64).

The Mirror2D procedure horizontally flips an **Image** (p. 64).

Parameters

Parameters

in	*src*	The input **Image** (p. 64).
out	*dst*	The output mirrored **Image** (p. 64).

Definition at line 89 of file imgproc.cpp.

7.1.3.18 Mul() [1/2]

```
void ecvl::Mul (
            const Image & src1,
            const Image & src2,
            Image & dst,
            DataType dst_type,
            bool saturate = true )
```

Multiplies two Image(s) and stores the result in a third **Image** (p. 64).

This procedure multiplies two Image(s) together and stores the result in a third **Image** (p. 64) that will have the specified DataType. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in	*src1*	Multiplier (first factor) **Image** (p. 64).
in	*src2*	Multiplicand (second factor) **Image** (p. 64).
out	*dst*	**Image** (p. 64) into which save the result of the multiplication.
in	*dst_type*	DataType that destination **Image** (p. 64) must have at the end of the operation.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

7.1.3.19 Mul() [2/2]

```
template<typename ST1 , typename ST2 >
void ecvl::Mul (
            const ST1 & src1,
            const ST2 & src2,
            Image & dst,
            bool saturate = true )
```

Multiplies two objects that could be either a scalar value or an **Image** (p. 64), storing the result into a destination **Image** (p. 64). The procedure does not perform any type promotion.

The procedure takes two input values (src1 and src2) and multiplies them together, storing the result into the destination image. If one of the operands is an **Image** (p. 64) and the other one is a scalar value, each pixel of the **Image** (p. 64) is multiplied by the scalar value, and the result is stored into dst. If src1 and src2 are both Image(s) the pixel-wise multiplication is applied and, again, the result is stored into dst.

Saturation is applied by default. If it is not the desired behavior change the saturate parameter to false.

In any case, the operation performed is dst = src1 $*$ src2.

Parameters

Parameters

in	*src1*	Multiplier operand. Could be either a scalar or an **Image** (p. 64).
in	*src2*	Multiplicand operand. Could be either a scalar or an **Image** (p. 64).
out	*dst*	Destination **Image** (p. 64). It will store the final result. If dst is not empty, its DataType will be preserved. Otherwise, it will have the same Data↵ Type as src1 if it is an **Image** (p. 64), src2 otherwise.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

.

Definition at line 518 of file arithmetic.h.

7.1.3.20 Neg()

```
Image & ecvl::Neg (
            Image & img )
```

In-place negation of an **Image** (p. 64).

The **Neg()** (p. 28) function negates every value of an **Image** (p. 64), and stores the the result in the same image. The type of the image will not change.

Parameters

Parameters

in,out	*img*	**Image** (p. 64) to be negated (in-place).

Returns

Reference to the **Image** (p. 64) containing the result of the negation.

Definition at line 41 of file arithmetic.cpp.

7.1.3.21 OtsuThreshold()

```
double ecvl::OtsuThreshold (
            const Image & src )
```

Calculates the Otsu thresholding value.

The OtsuThreshold function calculates the Otsu threshold value over a given input **Image** (p. 64). the **Image** (p. 64) must by **ColorType::GRAY** (p. 17).

Parameters

Parameters

in	*src*	Input **Image** (p. 64) on which to calculato the Otsu threshold value.

Returns

Otsu threshold value.

Definition at line 309 of file imgproc.cpp.

7.1.3.22 PromoteAdd()

```
template<typename T , typename U >
promote_superior_type_t<T, U> ecvl::PromoteAdd (
            T rhs,
            U lhs )
```

Definition at line 60 of file type_promotion.h.

7.1.3.23 PromoteDiv()

```
template<typename T , typename U >
promote_superior_type_t<T, U> ecvl::PromoteDiv (
            T rhs,
            U lhs )
```

Definition at line 63 of file type_promotion.h.

7.1.3.24 PromoteMul()

```
template<typename T , typename U >
promote_superior_type_t<T, U> ecvl::PromoteMul (
            T rhs,
            U lhs )
```

Definition at line 62 of file type_promotion.h.

7.1.3.25 PromoteSub()

```
template<typename T , typename U >
promote_superior_type_t<T, U> ecvl::PromoteSub (
            T rhs,
            U lhs )
```

Definition at line 61 of file type_promotion.h.

7.1.3.26 RearrangeChannels()

```
void ecvl::RearrangeChannels (
            const Image & src,
            Image & dst,
            const std::string & channels )
```

Changes the order of the **Image** (p. 64) dimensions.

The RearrangeChannels procedure changes the order of the input **Image** (p. 64) dimensions saving the result into the output **Image** (p. 64). The new order of dimensions can be specified as a string through the "channels" parameter. Input and output Images can be the same. The number of channels of the input **Image** (p. 64) must be the same of required channels.

Parameters

Parameters

in	*src*	Input **Image** (p. 64) on which to rearrange dimensions.
out	*dst*	The output rearranged **Image** (p. 64). Can be the src **Image** (p. 64).
in	*channels*	Desired order of **Image** (p. 64) channels.

Definition at line 48 of file image.cpp.

7.1.3.27 ResizeDim()

```
void ecvl::ResizeDim (
            const ecvl::Image & src,
            ecvl::Image & dst,
            const std::vector< int > & newdims,
            InterpolationType interp = InterpolationType::linear )
```

Resizes an **Image** (p. 64) to the specified dimensions.

The function resizes **Image** (p. 64) src and outputs the result in dst.

Parameters

Parameters

in	*src*	The input **Image** (p. 64).
out	*dst*	The output resized **Image** (p. 64).
in	*newdims*	std::vector<int> that specifies the new size of each dimension. The vector size must match the src **Image** (p. 64) dimentions, excluding the color channel
in	*interp*	InterpolationType to be used. See **InterpolationType** (p. 18).

Definition at line 30 of file imgproc.cpp.

7.1.3.28 ResizeScale()

```
void ecvl::ResizeScale (
            const ecvl::Image & src,
            ecvl::Image & dst,
            const std::vector< double > & scales,
            InterpolationType interp = InterpolationType::linear )
```

Resizes an **Image** (p. 64) by scaling the dimentions to a given scale factor.

The function resizes **Image** (p. 64) src and outputs the result in dst.

Parameters

Parameters

in	*src*	The input **Image** (p. 64).
out	*dst*	The output resized **Image** (p. 64).
in	*scales*	std::vector<double> that specifies the scale to apply to each dimension. The vector size must match the src **Image** (p. 64) dimentions, excluding the color channel.
in	*interp*	InterpolationType to be used. See **InterpolationType** (p. 18).

Definition at line 50 of file imgproc.cpp.

7.1.3.29 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 )
```

Rotates an **Image** (p. 64).

The Rotate2D procedure rotates an **Image** (p. 64) of a given angle (expressed in degrees) in a clockwise manner, with respect to a given center. The value of unknown pixels in the output **Image** (p. 64) are set to 0. The output **Image** (p. 64) is guaranteed to have the same dimensions as the input one. An optional scale parameter can be provided: this won't change the output **Image** (p. 64) size, but the image is scaled during rotation. Different interpolation types are available, see **InterpolationType** (p. 18).

Parameters

Parameters

in	*src*	The input **Image** (p. 64).
out	*dst*	The output rotated **Image** (p. 64).
in	*angle*	The rotation angle in degrees.
in	*center*	A std::vector<double> representing the coordinates of the rotation center. If empty, the center of the image is used.
in	*scale*	Optional scaling factor.
in	*interp*	Interpolation type used. Default is **InterpolationType::linear** (p. 18).

Definition at line 105 of file imgproc.cpp.

7.1.3.30 RotateFullImage2D()

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

Rotates an **Image** (p. 64) resizing the output accordingly.

The RotateFullImage2D procedure rotates an **Image** (p. 64) of a given angle (expressed in degrees) in a clockwise manner. The value of unknown pixels in the output **Image** (p. 64) are set to 0. The output **Image** (p. 64) is guaranteed to contain all the pixels of the rotated image. Thus, its dimensions can be different from those of the input. An optional scale parameter can be provided. Different interpolation types are available, see **InterpolationType** (p. 18).

Parameters

Parameters

in	*src*	The input **Image** (p. 64).
out	*dst*	The rotated output **Image** (p. 64).
in	*angle*	The rotation angle in degrees.
in	*scale*	Optional scaling factor.
in	*interp*	Interpolation type used. Default is **InterpolationType::linear** (p. 18).

Definition at line 134 of file imgproc.cpp.

7.1.3.31 saturate_cast() [1/2]

```
template<DataType ODT, typename IDT >
TypeInfo<ODT>::basetype ecvl::saturate_cast (
            IDT v )
```

Saturate a value (of any type) to the specified type.

Given an input of any type the saturate_cast function provide an output return value of the specified type applying saturation. When the input value in greater than the maximum possible value (max) for the output type, the max value is returned. When the input value in lower than the minimum possible value (min) for the output type, the min value is returned.

Parameters

Parameters

in	*v*	Input value (of any type).

Returns

Input value after cast and saturation.

Definition at line 27 of file arithmetic.h.

7.1.3.32 saturate_cast() [2/2]

```
template<typename ODT , typename IDT >
ODT ecvl::saturate_cast (
            const IDT & v )
```

Saturate a value (of any type) to the specified type.

Given an input of any type the saturate_cast function provide an output return value of the specified type applying saturation. When the input value in greater than the maximum possible value (max) for the output type, the max value is returned. When the input value in lower than the minimum possible value (min) for the output type, the min value is returned.

Parameters

Parameters

in	*v*	Input value (of any type).

Returns

Input value after cast and saturation.

Definition at line 54 of file arithmetic.h.

7.1.3.33 Sub() [1/2]

```
void ecvl::Sub (
            const Image & src1,
            const Image & src2,
            Image & dst,
            DataType dst_type,
            bool saturate = true )
```

Subtracts two Image(s) and stores the result in a third **Image** (p. 64).

This procedure subtracts the src2 **Image** (p. 64) from the src1 **Image** (p. 64) (src1 - src2) and stores the result in the dst **Image** (p. 64) that will have the specified DataType. By default a saturation will be applied. If it is not the desired behavior change the "saturate" parameter to false.

Parameters

Parameters

in	*src1*	Minuend **Image** (p. 64).
in	*src2*	Subtrahend **Image** (p. 64).
out	*dst*	**Image** (p. 64) into which save the result of the division.
in	*dst_type*	DataType that destination **Image** (p. 64) must have at the end of the operation.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

7.1.3.34 Sub() [2/2]

```
template<typename ST1 , typename ST2 >
void ecvl::Sub (
            const ST1 & src1,
            const ST2 & src2,
             Image & dst,
            bool saturate = true )
```

Subtracts two objects that could be either a scalar value or an **Image** (p. 64), storing the result into a destination **Image** (p. 64). The procedure does not perform any type promotion.

The procedure takes two input values (src1 and src2) and subtracts the second from the first, storing the result into the destination image. If src1 is an **Image** (p. 64) and src2 is a scalar value, src2 is subtracted from all the pixels inside src1 and the result is stored into dst. If src1 is a scalar value and src2 is an **Image** (p. 64), the opposite happens: src1 is diminished by each pixel value of src2, and the result is stored into dst. If src1 and src2 are both Image(s) the pixel-wise subtraction is applied and, again, the result is stored into dst.

Saturation is applied by default. If it is not the desired behavior change the saturate parameter to false.

In any case, the operation performed is dst = src1 - src2.

Parameters

Parameters

in	*src1*	Minuend operand. Could be either a scalar or an **Image** (p. 64).
in	*src2*	Subtrahend operand. Could be either a scalar or an **Image** (p. 64).
out	*dst*	Destination **Image** (p. 64). It will store the final result. If dst is not empty, its DataType will be preserved. Otherwise, it will have the same Data↩ Type as src1 if it is an **Image** (p. 64), src2 otherwise.
in	*saturate*	Whether to apply saturation or not. Default is true.

Returns

.

Definition at line 397 of file arithmetic.h.

7.1.3.35 TensorToImage()

```
 Image ecvl::TensorToImage (
            tensor & t )
```

Convert a EDDLL Tensor into an ECVL **Image** (p. 64).

Parameters

Parameters

in	*t*	Input EDDLL Tensor.

Returns

ECVL **Image** (p. 64).

Definition at line 8 of file support_eddll.cpp.

7.1.3.36 Threshold()

```
void ecvl::Threshold (
            const Image & src,
            Image & dst,
            double thresh,
            double maxval,
            ThresholdingType thresh_type = ThresholdingType::BINARY )
```

Applies a fixed threshold to an input **Image** (p. 64).

The Threshold function applies a fixed thresholding to an input **Image** (p. 64). The function is useful to get a binary image out of a grayscale (**ColorType::GRAY** (p. 17)) **Image** (p. 64) or to remove noise filtering out pixels with too small or too large values. Anyway, the function can be applied to any input **Image** (p. 64). The pixels up to "thresh" value will be set to 0, the pixels above this value will be set to "maxvalue" if "thresh_type" is **ThresholdingType::↩ BINARY** (p. 18) (default). The opposite will happen if "thresh_type" is **ThresholdingType::BINARY_INV** (p. 18).

Bug Input and output Images may have different color spaces.

Parameters

Parameters

in	*src*	Input **Image** (p. 64) on which to apply the threshold.
out	*dst*	The output thresholded **Image** (p. 64).
in	*thresh*	Threshold value.
in	*maxval*	The maximum values in the thresholded **Image** (p. 64).
in	*thresh_type*	Type of threshold to be applied, see **ThresholdingType** (p. 18). The default value is **ThresholdingType::BINARY** (p. 18).

Definition at line 293 of file imgproc.cpp.

7.1.3.37 WxFromImg()

```
wxImage ecvl::WxFromImg (
            Image & img )
```

Convert an ECVL **Image** (p. 64) into a wxImage.

Parameters

Parameters

in	*img*	Input ECVL **Image** (p. 64).

Returns

wxImage.

Definition at line 71 of file gui.cpp.

7.2 filesystem Namespace Reference

Classes

- class **path**

Functions

- **path** **operator/** (const **path** &lhs, const **path** &rhs)
- bool **exists** (const **path** &p)
- bool **exists** (const **path** &p, std::error_code &ec)
- bool **create_directories** (const **path** &p)
- bool **create_directories** (const **path** &p, std::error_code &ec)
- void **copy** (const **path** &from, const **path** &to)
- void **copy** (const **path** &from, const **path** &to, std::error_code &ec)
- bool **exists** (const **path** &p, error_code &ec)
- bool **create_directories** (const **path** &p, error_code &ec)
- void **copy** (const **path** &from, const **path** &to, error_code &ec)

7.2.1 Function Documentation

7.2.1.1 copy() [1/3]

```
void filesystem::copy (
            const path & from,
            const path & to,
            error_code & ec )
```

Definition at line 93 of file filesystem.cc.

7.2.1.2 copy() [2/3]

```
void filesystem::copy (
            const path & from,
            const path & to )
```

Definition at line 77 of file filesystem.cc.

7.2.1.3 copy() [3/3]

```
void filesystem::copy (
            const path & from,
            const path & to,
            std::error_code & ec )
```

7.2.1.4 create_directories() [1/3]

```
bool filesystem::create_directories (
            const path & p,
            error_code & ec )
```

Definition at line 61 of file filesystem.cc.

7.2.1.5 create_directories() [2/3]

```
bool filesystem::create_directories (
            const path & p )
```

Definition at line 43 of file filesystem.cc.

7.2.1.6 create_directories() [3/3]

```
bool filesystem::create_directories (
            const path & p,
            std::error_code & ec )
```

7.2.1.7 exists() [1/3]

```
bool filesystem::exists (
            const path & p,
            error_code & ec )
```

Definition at line 38 of file filesystem.cc.

7.2.1.8 exists() [2/3]

```
bool filesystem::exists (
            const path & p )
```

Definition at line 20 of file filesystem.cc.

7.2.1.9 exists() [3/3]

```
bool filesystem::exists (
            const path & p,
            std::error_code & ec )
```

7.2.1.10 operator/()

```
path filesystem::operator/ (
            const path & lhs,
            const path & rhs ) [inline]
```

Definition at line 109 of file filesystem.h.

Chapter 8

Class Documentation

8.1 ecvl::AddImpl< ST1, ST2 > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const ST1 &src1, const ST2 &src2, **Image** &dst, bool saturate)

8.1.1 Detailed Description

template<typename ST1, typename ST2>
struct ecvl::AddImpl< ST1, ST2 >

Definition at line 183 of file arithmetic.h.

8.1.2 Member Function Documentation

8.1.2.1 _()

```
template<typename ST1 , typename ST2 >
static void  ecvl::AddImpl< ST1, ST2 >::_ (
              const ST1 & src1,
              const ST2 & src2,
               Image & dst,
              bool saturate )  [inline], [static]
```

Definition at line 184 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.2 ecvl::AddImpl< Image, Image > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const **Image** &src1, const **Image** &src2, **Image** &dst, bool saturate)

8.2.1 Detailed Description

template<>
struct ecvl::AddImpl< Image, Image >

Definition at line 217 of file arithmetic.h.

8.2.2 Member Function Documentation

8.2.2.1 _()

```
static void ecvl::AddImpl< Image, Image >::_ (
            const Image & src1,
            const Image & src2,
            Image & dst,
            bool saturate ) [inline], [static]
```

Definition at line 218 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.3 ecvl::AddImpl< Image, ST2 > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const **Image** &src1, const ST2 &src2, **Image** &dst, bool saturate)

8.3.1 Detailed Description

template<typename ST2>
struct ecvl::AddImpl< Image, ST2 >

Definition at line 194 of file arithmetic.h.

8.3.2 Member Function Documentation

8.3.2.1 _()

```
template<typename ST2 >
static void  ecvl::AddImpl<  Image, ST2 >::_ (
             const  Image & src1,
             const ST2 & src2,
              Image & dst,
             bool saturate )  [inline], [static]
```

Definition at line 195 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.4 ecvl::AddImpl< ST1, Image > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const ST1 &src1, const **Image** &src2, **Image** &dst, bool saturate)

8.4.1 Detailed Description

template<typename ST1>
struct ecvl::AddImpl< ST1, Image >

Definition at line 208 of file arithmetic.h.

8.4.2 Member Function Documentation

8.4.2.1 _()

```
template<typename ST1 >
static void  ecvl::AddImpl< ST1,  Image >::_ (
             const ST1 & src1,
             const  Image & src2,
              Image & dst,
             bool saturate )  [inline], [static]
```

Definition at line 209 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.5 ecvl::arithmetic_superior_type< T, U > Struct Template Reference

```
#include <type_promotion.h>
```

Public Types

- using **type** = typename std::conditional_t< std::is_floating_point< T >::value &&std::is_floating_point< U >::value, **larger_arithmetic_type_t**< T, U >, std::conditional_t< std::is_floating_point< T >::value, T, std::conditional_t< std::is_floating_point< U >::value, U, **larger_arithmetic_type_t**< T, U > >> >

8.5.1 Detailed Description

template<typename T, typename U>
struct ecvl::arithmetic_superior_type< T, U >

Definition at line 23 of file type_promotion.h.

8.5.2 Member Typedef Documentation

8.5.2.1 type

```
template<typename T, typename U>
using  ecvl::arithmetic_superior_type< T, U >::  type = typename std::conditional_t<std::is_floating_point<T>::value && std::is_floating_point<U>::value,  larger_arithmetic_type_t<T, U>, std::conditional_t<std::is_floating_point<T>::value, T, std::conditional_t<std::is_floating_point<U>::value, U,  larger_arithmetic_type_t<T, U> >> >
```

Definition at line 28 of file type_promotion.h.

The documentation for this struct was generated from the following file:

- **type_promotion.h**

8.6 ecvl::ConstContiguousIterator< T > Struct Template Reference

```
#include <iterators.h>
```

Public Member Functions

- **ConstContiguousIterator** (const **Image** &img, std::vector< int > pos={})
- **ConstContiguousIterator** & **operator++** ()
- const T & **operator** * () const
- const T * **operator->** () const
- bool **operator==** (const **ConstContiguousIterator** &rhs) const
- bool **operator!=** (const **ConstContiguousIterator** &rhs) const

Public Attributes

- uint8_t * **ptr_**
- const **Image** * **img_**

8.6.1 Detailed Description

template<typename T>
struct ecvl::ConstContiguousIterator< T >

Definition at line 68 of file iterators.h.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 ConstContiguousIterator()

```
template<typename T >
ConstContiguousIterator::ConstContiguousIterator (
            const Image & img,
            std::vector< int > pos = {} )
```

Definition at line 78 of file image.h.

8.6.3 Member Function Documentation

8.6.3.1 operator *()

```
template<typename T >
const T& ecvl::ConstContiguousIterator< T >::operator * ( ) const [inline]
```

Definition at line 74 of file iterators.h.

8.6.3.2 operator"!=()

```
template<typename T >
bool ecvl::ConstContiguousIterator< T >::operator!= (
            const ConstContiguousIterator< T > & rhs ) const [inline]
```

Definition at line 77 of file iterators.h.

8.6.3.3 operator++()

```
template<typename T >
ConstContiguousIterator& ecvl::ConstContiguousIterator< T >::operator++ ( ) [inline]
```

Definition at line 73 of file iterators.h.

8.6.3.4 operator->()

```
template<typename T >
const T* ecvl::ConstContiguousIterator< T >::operator-> ( ) const [inline]
```

Definition at line 75 of file iterators.h.

8.6.3.5 operator==()

```
template<typename T >
bool ecvl::ConstContiguousIterator< T >::operator== (
            const ConstContiguousIterator< T > & rhs ) const [inline]
```

Definition at line 76 of file iterators.h.

8.6.4 Member Data Documentation

8.6.4.1 img_

```
template<typename T >
const Image* ecvl::ConstContiguousIterator< T >::img_
```

Definition at line 70 of file iterators.h.

8.6.4.2 ptr_

```
template<typename T >
uint8_t* ecvl::ConstContiguousIterator< T >::ptr_
```

Definition at line 69 of file iterators.h.

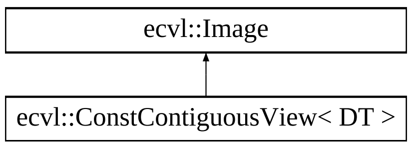
The documentation for this struct was generated from the following files:

- **iterators.h**
- **image.h**
- **iterators_impl.inc.h**

8.7 ecvl::ConstContiguousView< DT > Class Template Reference

```
#include <image.h>
```

Inheritance diagram for ecvl::ConstContiguousView< DT >:



Public Types

- using **basetype** = typename TypeInfo< DT >:: **basetype**

Public Member Functions

- **ConstContiguousView** (**Image** &img)
- const **basetype** & **operator()** (const std::vector< int > &coords)
- **ConstContiguousIterator**< **basetype** > **Begin** ()
- **ConstContiguousIterator**< **basetype** > **End** ()

Additional Inherited Members

8.7.1 Detailed Description

template< DataType DT >
class ecvl::ConstContiguousView< DT >

Definition at line 473 of file image.h.

8.7.2 Member Typedef Documentation

8.7.2.1 basetype

```
template<DataType DT>
using ecvl::ConstContiguousView< DT >:: basetype = typename TypeInfo<DT>:: basetype
```

Definition at line 475 of file image.h.

8.7.3 Constructor & Destructor Documentation

8.7.3.1 ConstContiguousView()

```
template<DataType DT>
ecvl::ConstContiguousView< DT >:: ConstContiguousView (
            Image & img ) [inline]
```

Definition at line 477 of file image.h.

8.7.4 Member Function Documentation

8.7.4.1 Begin()

```
template<DataType DT>
ConstContiguousIterator< basetype> ecvl::ConstContiguousView< DT >::Begin ( ) [inline]
```

Definition at line 495 of file image.h.

8.7.4.2 End()

```
template<DataType DT>
ConstContiguousIterator< basetype> ecvl::ConstContiguousView< DT >::End ( ) [inline]
```

Definition at line 496 of file image.h.

8.7.4.3 operator()()

```
template<DataType DT>
const basetype& ecvl::ConstContiguousView< DT >::operator() (
            const std::vector< int > & coords ) [inline]
```

Definition at line 491 of file image.h.

The documentation for this class was generated from the following file:

- **image.h**

8.8 ecvl::ConstIterator< T > Struct Template Reference

`#include <iterators.h>`

Public Types

- typedef **ConstIterator** &(ConstIterator::∗ **IncrementMemFn**) ()

Public Member Functions

- **ConstIterator** (const **Image** &img, std::vector< int > pos={})
- **ConstIterator** & **operator++** ()
- const T & **operator** ∗ () const
- const T ∗ **operator->** () const
- bool **operator==** (const **ConstIterator** &rhs) const
- bool **operator!=** (const **ConstIterator** &rhs) const

Public Attributes

- std::vector< int > **pos_**
- const uint8_t ∗ **ptr_**
- const **Image** ∗ **img_**
- **IncrementMemFn** **incrementor** = & **ConstIterator**<T>::IncrementPos

8.8.1 Detailed Description

template<typename T>
struct ecvl::ConstIterator< T >

Definition at line 32 of file iterators.h.

8.8.2 Member Typedef Documentation

8.8.2.1 IncrementMemFn

```
template<typename T >
typedef ConstIterator&(ConstIterator::* ecvl::ConstIterator< T >::IncrementMemFn) ()
```

Definition at line 37 of file iterators.h.

8.8.3 Constructor & Destructor Documentation

8.8.3.1 ConstIterator()

```
template<typename T >
ConstIterator::ConstIterator (
            const Image & img,
            std::vector< int > pos = {} )
```

Definition at line 30 of file image.h.

8.8.4 Member Function Documentation

8.8.4.1 operator *()

```
template<typename T >
const T& ecvl::ConstIterator< T >::operator * ( ) const [inline]
```

Definition at line 42 of file iterators.h.

8.8.4.2 operator"!=()

```
template<typename T >
bool ecvl::ConstIterator< T >::operator!= (
            const ConstIterator< T > & rhs ) const [inline]
```

Definition at line 45 of file iterators.h.

8.8.4.3 operator++()

```
template<typename T >
ConstIterator& ecvl::ConstIterator< T >::operator++ ( ) [inline]
```

Definition at line 41 of file iterators.h.

8.8.4.4 operator->()

```
template<typename T >
const T* ecvl::ConstIterator< T >::operator-> ( ) const [inline]
```

Definition at line 43 of file iterators.h.

8.8.4.5 operator==()

```
template<typename T >
bool ecvl::ConstIterator< T >::operator== (
            const ConstIterator< T > & rhs ) const [inline]
```

Definition at line 44 of file iterators.h.

8.8.5 Member Data Documentation

8.8.5.1 img_

```
template<typename T >
const Image* ecvl::ConstIterator< T >::img_
```

Definition at line 35 of file iterators.h.

8.8.5.2 incrementor

```
template<typename T >
IncrementMemFn ecvl::ConstIterator< T >::incrementor = & ConstIterator<T>::IncrementPos
```

Definition at line 38 of file iterators.h.

8.8.5.3 pos_

```
template<typename T >
std::vector<int> ecvl::ConstIterator< T >::pos_
```

Definition at line 33 of file iterators.h.

8.8.5.4 ptr_

```
template<typename T >
const uint8_t* ecvl::ConstIterator< T >::ptr_
```

Definition at line 34 of file iterators.h.

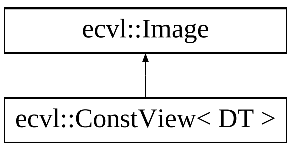
The documentation for this struct was generated from the following files:

- **iterators.h**
- **image.h**
- **iterators_impl.inc.h**

8.9 ecvl::ConstView< DT > Class Template Reference

```
#include <image.h>
```

Inheritance diagram for ecvl::ConstView< DT >:



Public Types

- using **basetype** = typename TypeInfo< DT >:: **basetype**

Public Member Functions

- **ConstView** (const **Image** &img)
- const **basetype** & **operator()** (const std::vector< int > &coords)
- **ConstIterator**< **basetype** > **Begin** ()
- **ConstIterator**< **basetype** > **End** ()

Additional Inherited Members

8.9.1 Detailed Description

template<DataType DT>
class ecvl::ConstView< DT >

Definition at line 419 of file image.h.

8.9.2 Member Typedef Documentation

8.9.2.1 basetype

```
template<DataType DT>
using ecvl::ConstView< DT >:: basetype = typename TypeInfo<DT>:: basetype
```

Definition at line 421 of file image.h.

8.9.3 Constructor & Destructor Documentation

8.9.3.1 ConstView()

```
template<DataType DT>
ecvl::ConstView< DT >:: ConstView (
            const Image & img ) [inline]
```

Definition at line 423 of file image.h.

8.9.4 Member Function Documentation

8.9.4.1 Begin()

```
template<DataType DT>
ConstIterator< basetype> ecvl::ConstView< DT >::Begin ( ) [inline]
```

Definition at line 441 of file image.h.

8.9.4.2 End()

```
template<DataType DT>
ConstIterator< basetype> ecvl::ConstView< DT >::End ( ) [inline]
```

Definition at line 442 of file image.h.

8.9.4.3 operator()()

```
template<DataType DT>
const basetype& ecvl::ConstView< DT >::operator() (
            const std::vector< int > & coords ) [inline]
```

Definition at line 437 of file image.h.

The documentation for this class was generated from the following file:

- **image.h**

8.10 ecvl::ContiguousIterator< T > Struct Template Reference

```
#include <iterators.h>
```

Public Member Functions

- **ContiguousIterator** (**Image** &img, std::vector< int > pos={})
- **ContiguousIterator** & **operator++** ()
- T & **operator** * () const
- T * **operator->** () const
- bool **operator==** (const **ContiguousIterator** &rhs) const
- bool **operator!=** (const **ContiguousIterator** &rhs) const

Public Attributes

- uint8_t * **ptr_**
- **Image** * **img_**

8.10.1 Detailed Description

template<typename T>
struct ecvl::ContiguousIterator< T >

Definition at line 53 of file iterators.h.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 ContiguousIterator()

```
template<typename T >
ContiguousIterator::ContiguousIterator (
              Image & img,
             std::vector< int > pos = {} )
```

Definition at line 56 of file image.h.

8.10.3 Member Function Documentation

8.10.3.1 operator *()

```
template<typename T >
T& ecvl::ContiguousIterator< T >::operator * ( ) const [inline]
```

Definition at line 59 of file iterators.h.

8.10.3.2 operator"!=()

```
template<typename T >
bool ecvl::ContiguousIterator< T >::operator!= (
             const ContiguousIterator< T > & rhs ) const [inline]
```

Definition at line 62 of file iterators.h.

8.10.3.3 operator++()

```
template<typename T >
ContiguousIterator& ecvl::ContiguousIterator< T >::operator++ ( ) [inline]
```

Definition at line 58 of file iterators.h.

8.10.3.4 operator->()

```
template<typename T >
T* ecvl::ContiguousIterator< T >::operator-> ( ) const [inline]
```

Definition at line 60 of file iterators.h.

8.10.3.5 operator==()

```
template<typename T >
bool ecvl::ContiguousIterator< T >::operator== (
            const ContiguousIterator< T > & rhs ) const [inline]
```

Definition at line 61 of file iterators.h.

8.10.4 Member Data Documentation

8.10.4.1 img_

```
template<typename T >
Image* ecvl::ContiguousIterator< T >::img_
```

Definition at line 55 of file iterators.h.

8.10.4.2 ptr_

```
template<typename T >
uint8_t* ecvl::ContiguousIterator< T >::ptr_
```

Definition at line 54 of file iterators.h.

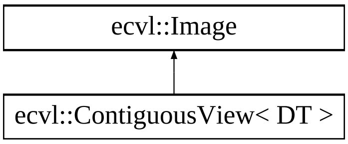
The documentation for this struct was generated from the following files:

- **iterators.h**
- **image.h**
- **iterators_impl.inc.h**

8.11 ecvl::ContiguousView< DT > Class Template Reference

```
#include <image.h>
```

Inheritance diagram for ecvl::ContiguousView< DT >:



Public Types

- using **basetype** = typename TypeInfo< DT >:: **basetype**

Public Member Functions

- **ContiguousView** (**Image** &img)
- **basetype** & **operator()** (const std::vector< int > &coords)
- **ContiguousIterator**< **basetype** > **Begin** ()
- **ContiguousIterator**< **basetype** > **End** ()

Additional Inherited Members

8.11.1 Detailed Description

template<DataType DT>
class ecvl::ContiguousView< DT >

Definition at line 446 of file image.h.

8.11.2 Member Typedef Documentation

8.11.2.1 basetype

```
template<DataType DT>
using ecvl::ContiguousView< DT >:: basetype = typename TypeInfo<DT>:: basetype
```

Definition at line 448 of file image.h.

8.11.3 Constructor & Destructor Documentation

8.11.3.1 ContiguousView()

```
template<DataType DT>
ecvl::ContiguousView< DT >:: ContiguousView (
            Image & img ) [inline]
```

Definition at line 450 of file image.h.

8.11.4 Member Function Documentation

8.11.4.1 Begin()

```
template<DataType DT>
ContiguousIterator< basetype> ecvl::ContiguousView< DT >::Begin ( ) [inline]
```

Definition at line 468 of file image.h.

8.11.4.2 End()

```
template<DataType DT>
ContiguousIterator< basetype> ecvl::ContiguousView< DT >::End ( ) [inline]
```

Definition at line 469 of file image.h.

8.11.4.3 operator()()

```
template<DataType DT>
basetype& ecvl::ContiguousView< DT >::operator() (
            const std::vector< int > & coords ) [inline]
```

Definition at line 464 of file image.h.

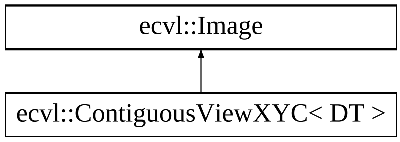
The documentation for this class was generated from the following file:

- **image.h**

8.12 ecvl::ContiguousViewXYC< DT > Class Template Reference

```
#include <image.h>
```

Inheritance diagram for ecvl::ContiguousViewXYC< DT >:



Public Types

- using **basetype** = typename TypeInfo< DT >:: **basetype**

Public Member Functions

- **ContiguousViewXYC** (**Image** &img)
- int **width** () const
- int **height** () const
- int **channels** () const
- **basetype** & **operator()** (int x, int y, int c)
- **ContiguousIterator**< **basetype** > **Begin** ()
- **ContiguousIterator**< **basetype** > **End** ()

Additional Inherited Members

8.12.1 Detailed Description

template<DataType DT>
class ecvl::ContiguousViewXYC< DT >

Definition at line 500 of file image.h.

8.12.2 Member Typedef Documentation

8.12.2.1 basetype

```
template<DataType DT>
using ecvl::ContiguousViewXYC< DT >:: basetype = typename TypeInfo<DT>:: basetype
```

Definition at line 502 of file image.h.

8.12.3 Constructor & Destructor Documentation

8.12.3.1 ContiguousViewXYC()

```
template<DataType DT>
ecvl::ContiguousViewXYC< DT >:: ContiguousViewXYC (
              Image & img ) [inline]
```

Definition at line 504 of file image.h.

8.12.4 Member Function Documentation

8.12.4.1 Begin()

```
template<DataType DT>
ContiguousIterator< basetype> ecvl::ContiguousViewXYC< DT >::Begin ( ) [inline]
```

Definition at line 530 of file image.h.

8.12.4.2 channels()

```
template<DataType DT>
int ecvl::ContiguousViewXYC< DT >::channels ( ) const [inline]
```

Definition at line 524 of file image.h.

8.12.4.3 End()

```
template<DataType DT>
ContiguousIterator< basetype> ecvl::ContiguousViewXYC< DT >::End ( ) [inline]
```

Definition at line 531 of file image.h.

8.12.4.4 height()

```
template<DataType DT>
int ecvl::ContiguousViewXYC< DT >::height ( ) const [inline]
```

Definition at line 523 of file image.h.

8.12.4.5 operator()()

```
template<DataType DT>
basetype& ecvl::ContiguousViewXYC< DT >::operator() (
            int x,
            int y,
            int c ) [inline]
```

Definition at line 526 of file image.h.

8.12.4.6 width()

```
template<DataType DT>
int ecvl::ContiguousViewXYC< DT >::width ( ) const [inline]
```

Definition at line 522 of file image.h.

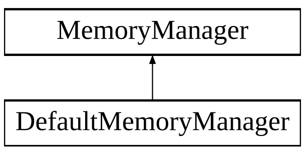
The documentation for this class was generated from the following file:

- **image.h**

8.13 DefaultMemoryManager Class Reference

```
#include <memorymanager.h>
```

Inheritance diagram for DefaultMemoryManager:



Public Member Functions

- virtual uint8_t * **Allocate** (size_t nbytes) override
- virtual void **Deallocate** (uint8_t *data) override
- virtual uint8_t * **AllocateAndCopy** (size_t nbytes, uint8_t *src) override

Static Public Member Functions

- static **DefaultMemoryManager** ∗ **GetInstance** ()

8.13.1 Detailed Description

Definition at line 16 of file memorymanager.h.

8.13.2 Member Function Documentation

8.13.2.1 Allocate()

```
virtual uint8_t* DefaultMemoryManager::Allocate (
             size_t nbytes )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 83).

Definition at line 18 of file memorymanager.h.

8.13.2.2 AllocateAndCopy()

```
virtual uint8_t* DefaultMemoryManager::AllocateAndCopy (
             size_t nbytes,
             uint8_t * src )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 83).

Definition at line 24 of file memorymanager.h.

8.13.2.3 Deallocate()

```
virtual void DefaultMemoryManager::Deallocate (
             uint8_t * data )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 83).

Definition at line 21 of file memorymanager.h.

8.13.2.4 GetInstance()

```
DefaultMemoryManager * DefaultMemoryManager::GetInstance ( ) [static]
```

Definition at line 3 of file memorymanager.cpp.

The documentation for this class was generated from the following files:

- **memorymanager.h**
- **memorymanager.cpp**

8.14 ecvl::DivImpl< ST1, ST2, ET > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const ST1 &src1, const ST2 &src2, **Image** &dst, bool saturate, ET epsilon)

8.14.1 Detailed Description

template<typename ST1, typename ST2, typename ET>
struct ecvl::DivImpl< ST1, ST2, ET >

Definition at line 590 of file arithmetic.h.

8.14.2 Member Function Documentation

8.14.2.1 _()

```
template<typename ST1 , typename ST2 , typename ET >
static void ecvl::DivImpl< ST1, ST2, ET >::_ (
            const ST1 & src1,
            const ST2 & src2,
            Image & dst,
            bool saturate,
            ET epsilon ) [inline], [static]
```

Definition at line 591 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.15 ecvl::DivImpl< Image, Image, ET > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const **Image** &src1, const **Image** &src2, **Image** &dst, bool saturate, ET epsilon)

8.15.1 Detailed Description

template<typename ET>
struct ecvl::DivImpl< Image, Image, ET >

Definition at line 634 of file arithmetic.h.

8.15.2 Member Function Documentation

8.15.2.1 _()

```
template<typename ET >
static void ecvl::DivImpl< Image, Image, ET >::_ (
            const Image & src1,
            const Image & src2,
            Image & dst,
            bool saturate,
            ET epsilon ) [inline], [static]
```

Definition at line 635 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.16 ecvl::DivImpl< Image, ST2, ET > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const **Image** &src1, const ST2 &src2, **Image** &dst, bool saturate, ET epsilon)

8.16.1 Detailed Description

template<typename ST2, typename ET>
struct ecvl::DivImpl< Image, ST2, ET >

Definition at line 601 of file arithmetic.h.

8.16.2 Member Function Documentation

8.16.2.1 _()

```
template<typename ST2 , typename ET >
static void ecvl::DivImpl< Image, ST2, ET >::_ (
             const Image & src1,
             const ST2 & src2,
             Image & dst,
             bool saturate,
             ET epsilon ) [inline], [static]
```

Definition at line 602 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.17 ecvl::DivImpl< ST1, Image, ET > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const ST1 &src1, const **Image** &src2, **Image** &dst, bool saturate, ET epsilon)

8.17.1 Detailed Description

template<typename ST1, typename ET>
struct ecvl::DivImpl< ST1, Image, ET >

Definition at line 620 of file arithmetic.h.

8.17.2 Member Function Documentation

8.17.2.1 _()

```
template<typename ST1 , typename ET >
static void  ecvl::DivImpl< ST1,  Image, ET >::_ (
             const ST1 & src1,
             const  Image & src2,
              Image & dst,
             bool saturate,
             ET epsilon )  [inline], [static]
```

Definition at line 621 of file arithmetic.h.

The documentation for this struct was generated from the following file:

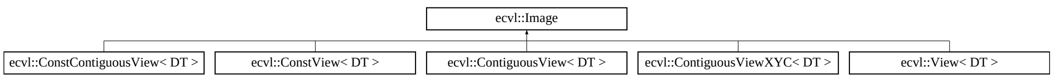
- **arithmetic.h**

8.18 ecvl::Image Class Reference

Image (p. 64) class.

`#include <image.h>`

Inheritance diagram for ecvl::Image:



Public Member Functions

- template<typename T >
  Iterator< T > **Begin** ()
  *Generic non-const Begin **Iterator** (p. 78).*
- template<typename T >
  Iterator< T > **End** ()
  *Generic non-const End **Iterator** (p. 78).*
- template<typename T >
  ConstIterator< T > **Begin** () const
  *Generic const Begin **Iterator** (p. 78).*
- template<typename T >
  ConstIterator< T > **End** () const
  *Generic const End **Iterator** (p. 78).*
- template<typename T >
  ContiguousIterator< T > **ContiguousBegin** ()
  *Contiguous non-const Begin **Iterator** (p. 78).*
- template<typename T >
  ContiguousIterator< T > **ContiguousEnd** ()
  *Contiguous non-const End **Iterator** (p. 78).*
- template<typename T >
  ConstContiguousIterator< T > **ContiguousBegin** () const
  *Contiguous const Begin **Iterator** (p. 78).*

- template< typename T >
  ConstContiguousIterator< T > **ContiguousEnd** () const

  *Contiguous const End **Iterator** (p. 78).*
- **Image** ()

  Default constructor.
- **Image** (const std::vector< int > &dims, DataType elemtype, std::string channels, **ColorType** colortype)

  Initializing constructor.
- **Image** (const **Image** &img)

  Copy constructor.
- **Image** (**Image** &&img)

  Move constructor.
- **Image** & **operator=** (**Image** rhs)
- void **Create** (const std::vector< int > &dims, DataType elemtype, std::string channels, **ColorType** colortype)

  Allocates new contiguous data if needed.
- ∼**Image** ()

  Destructor.
- bool **IsEmpty** () const

  *To check whether the **Image** (p. 64) contains or not data, regardless the owning status.*
- bool **IsOwner** () const

  *To check whether the **Image** (p. 64) is owner of the data.*
- uint8_t ∗ **Ptr** (const std::vector< int > &coords)

  Returns a non-const pointer to data at given coordinates.
- const uint8_t ∗ **Ptr** (const std::vector< int > &coords) const

  Returns a const pointer to data at given coordinates.

Public Attributes

- DataType **elemtype_**

  *Type of **Image** (p. 64) pixels, must be one of the values available in DataType.*
- uint8_t **elemsize_**

  *Size (in bytes) of **Image** (p. 64) pixels.*
- std::vector< int > **dims_**

  *Vector of **Image** (p. 64) dimensions. Each dimension is given in pixels/voxels.*
- std::vector< int > **strides_**

  *Vector of **Image** (p. 64) strides.*
- std::string **channels_**

  *String which describes how **Image** (p. 64) planes are organized.*
- **ColorType** **colortype_**

  ***Image** (p. 64) ColorType.*
- uint8_t ∗ **data_**

  *Pointer to **Image** (p. 64) data.*
- size_t **datasize_**

  *Size of **Image** (p. 64) data in bytes.*
- bool **contiguous_**

  Whether the image is stored contiguously or not in memory.
- **MetaData** ∗ **meta_**

  *Pointer to **Image** (p. 64) **MetaData** (p. 83).*
- **MemoryManager** ∗ **mem_**

  *Pointer to the **MemoryManager** (p. 82) employed by the **Image** (p. 64).*

Friends

- void **swap** (**Image** &lhs, **Image** &rhs)

8.18.1 Detailed Description

Image (p. 64) class.

Definition at line 39 of file image.h.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 Image() [1/4]

```
ecvl::Image::Image ( )  [inline]
```

Default constructor.

The default constructor creates an empty image without any data.

Definition at line 172 of file image.h.

8.18.2.2 Image() [2/4]

```
ecvl::Image::Image (
            const std::vector< int > & dims,
            DataType elemtype,
            std::string channels,
             ColorType colortype )  [inline]
```

Initializing constructor.

The initializing constructor creates a proper image and allocates the data.

Definition at line 191 of file image.h.

8.18.2.3 Image() [3/4]

```
ecvl::Image::Image (
            const  Image & img )  [inline]
```

Copy constructor.

The copy constructor creates an new **Image** (p. 64) copying (Deep Copy) the input one. The new **Image** (p. 64) will be contiguous regardless of the contiguity of the to be copied **Image** (p. 64).

Definition at line 222 of file image.h.

8.18.2.4 Image() [4/4]

```
ecvl::Image::Image (
        Image && img ) [inline]
```

Move constructor.

Move constructor

Definition at line 272 of file image.h.

8.18.2.5 ∼Image()

```
ecvl::Image::∼Image ( ) [inline]
```

Destructor.

If the **Image** (p. 64) is the owner of data they will be deallocate. Otherwise nothing will happen.

Definition at line 327 of file image.h.

8.18.3 Member Function Documentation

8.18.3.1 Begin() [1/2]

```
template<typename T >
Iterator<T> ecvl::Image::Begin ( ) [inline]
```

Generic non-const Begin **Iterator** (p. 78).

This function gives you a non-const generic Begin **Iterator** (p. 78) that can be used both for contiguous and non-contiguous non-const Images. It is useful to iterate over a non-const **Image** (p. 64). If the **Image** (p. 64) is contiguous prefer the use of ContiguousIterato which in most cases improve the performance.

Definition at line 108 of file image.h.

8.18.3.2 Begin() [2/2]

```
template<typename T >
ConstIterator<T> ecvl::Image::Begin ( ) const [inline]
```

Generic const Begin **Iterator** (p. 78).

This function gives you a const generic Begin **Iterator** (p. 78) that can be used both for contiguous and non-contiguous const Images. It is useful to iterate over a const **Image** (p. 64). If the **Image** (p. 64) is contiguous prefer the use of **ConstContiguousIterator** (p. 43) which in most cases improve the performance.

Definition at line 125 of file image.h.

8.18.3.3 ContiguousBegin() [1/2]

```
template<typename T >
ContiguousIterator<T> ecvl::Image::ContiguousBegin ( )  [inline]
```

Contiguous non-const Begin **Iterator** (p. 78).

This function gives you a contiguous non-const Begin **Iterator** (p. 78) that can be used only for contiguous Images. If the **Image** (p. 64) is contiguous it is preferable to the non-contiguous iterator since it has usually better performance.

Definition at line 142 of file image.h.

8.18.3.4 ContiguousBegin() [2/2]

```
template<typename T >
ConstContiguousIterator<T> ecvl::Image::ContiguousBegin ( ) const  [inline]
```

Contiguous const Begin **Iterator** (p. 78).

This function gives you a contiguous const Begin **Iterator** (p. 78) that can be used only for contiguous Images. If the **Image** (p. 64) is contiguous it is preferable to the non-contiguous iterator since it has usually better performance.

Definition at line 159 of file image.h.

8.18.3.5 ContiguousEnd() [1/2]

```
template<typename T >
ContiguousIterator<T> ecvl::Image::ContiguousEnd ( )  [inline]
```

Contiguous non-const End **Iterator** (p. 78).

This function gives you a contiguous non-const End **Iterator** (p. 78) that can be used only for contiguous Images.

Definition at line 150 of file image.h.

8.18.3.6 ContiguousEnd() [2/2]

```
template<typename T >
ConstContiguousIterator<T> ecvl::Image::ContiguousEnd ( ) const  [inline]
```

Contiguous const End **Iterator** (p. 78).

This function gives you a contiguous const End **Iterator** (p. 78) that can be used only for contiguous Images.

Definition at line 166 of file image.h.

8.18.3.7 Create()

```
void ecvl::Image::Create (
            const std::vector< int > & dims,
            DataType elemtype,
            std::string channels,
             ColorType colortype )
```

Allocates new contiguous data if needed.

The Create method allocates **Image** (p. 64) data as specified by the input parameters. The procedures tries to avoid the allocation of new memory when possible. The resulting image will be contiguous in any case. Calling this method on an **Image** (p. 64) that does not own data will always cause a new allocation, and the **Image** (p. 64) will become the owner of the data.

Parameters

Parameters

in	*dims*	New **Image** (p. 64) dimensions.
in	*elemtype*	New **Image** (p. 64) DataType.
in	*channels*	New **Image** (p. 64) channels.
in	*colortype*	New **Image** (p. 64) colortype.

Definition at line 8 of file image.cpp.

8.18.3.8 End() [1/2]

```
template<typename T >
Iterator<T> ecvl::Image::End ( )  [inline]
```

Generic non-const End **Iterator** (p. 78).

This function gives you a non-const generic End **Iterator** (p. 78) that can be used both for contiguous and non-contiguous non-const Images. It is useful to iterate over over a non-const **Image** (p. 64).

Definition at line 116 of file image.h.

8.18.3.9 End() [2/2]

```
template<typename T >
ConstIterator<T> ecvl::Image::End ( ) const  [inline]
```

Generic const End **Iterator** (p. 78).

This function gives you a const generic End **Iterator** (p. 78) that can be used both for contiguous and non-contiguous const Images. It is useful to iterate over a const **Image** (p. 64).

Definition at line 133 of file image.h.

8.18.3.10 IsEmpty()

```
bool ecvl::Image::IsEmpty ( ) const [inline]
```

To check whether the **Image** (p. 64) contains or not data, regardless the owning status.

Definition at line 333 of file image.h.

8.18.3.11 IsOwner()

```
bool ecvl::Image::IsOwner ( ) const [inline]
```

To check whether the **Image** (p. 64) is owner of the data.

Definition at line 336 of file image.h.

8.18.3.12 operator=()

```
Image& ecvl::Image::operator= (
            Image rhs ) [inline]
```

Definition at line 303 of file image.h.

8.18.3.13 Ptr() [1/2]

```
uint8_t* ecvl::Image::Ptr (
            const std::vector< int > & coords ) [inline]
```

Returns a non-const pointer to data at given coordinates.

Definition at line 339 of file image.h.

8.18.3.14 Ptr() [2/2]

```
const uint8_t* ecvl::Image::Ptr (
            const std::vector< int > & coords ) const [inline]
```

Returns a const pointer to data at given coordinates.

Definition at line 344 of file image.h.

8.18.4 Friends And Related Function Documentation

8.18.4.1 swap

```
void swap (
            Image & lhs,
            Image & rhs ) [friend]
```

Definition at line 288 of file image.h.

8.18.5 Member Data Documentation

8.18.5.1 channels_

```
std::string ecvl::Image::channels_
```

String which describes how **Image** (p. 64) planes are organized.

A single character provides the information related to the corresponding channel. The possible values are:

- 'x': horizontal spatial dimension
- 'y': vertical spatial dimension
- 'z': depth spatial dimension
- 'c': color dimension
- 't': temporal dimension
- 'o': any other dimension For example, "xyc" describes a 2-dimensional **Image** (p. 64) structured in color planes. This could be for example a **ColorType::GRAY** (p. 17) **Image** (p. 64) with dims_[2] = 1 or a **Color↩ Type::RGB** (p. 17) **Image** (p. 64) with dims_[2] = 3 an so on. The ColorType constrains the value of the dimension corresponding to the color channel. Another example is "cxy" with dims_[0] = 3 and **ColorType↩ ::BGR** (p. 17). In this case the color dimension is the one which changes faster as it is done in other libraries such as OpenCV.

Definition at line 52 of file image.h.

8.18.5.2 colortype_

```
ColorType ecvl::Image::colortype_
```

Image (p. 64) ColorType.

If this is different from **ColorType::none** (p. 17) the channels_ string must contain a 'c' and the corresponding dimension must have the appropriate value. See **ColorType** (p. 17) for the possible values.

Definition at line 75 of file image.h.

8.18.5.3 contiguous_

```
bool ecvl::Image::contiguous_
```

Whether the image is stored contiguously or not in memory.

Definition at line 89 of file image.h.

8.18.5.4 data_

```
uint8_t* ecvl::Image::data_
```

Pointer to **Image** (p. 64) data.

If the **Image** (p. 64) is not the owner of data, for example when using **Image** (p. 64) views, this attribute will point to the data of another **Image** (p. 64). The possession or not of the data depends on the **MemoryManager** (p. 82).

Definition at line 81 of file image.h.

8.18.5.5 datasize_

```
size_t ecvl::Image::datasize_
```

Size of **Image** (p. 64) data in bytes.

Definition at line 88 of file image.h.

8.18.5.6 dims_

```
std::vector<int> ecvl::Image::dims_
```

Vector of **Image** (p. 64) dimensions. Each dimension is given in pixels/voxels.

Definition at line 44 of file image.h.

8.18.5.7 elemsize_

```
uint8_t ecvl::Image::elemsize_
```

Size (in bytes) of **Image** (p. 64) pixels.

Definition at line 43 of file image.h.

8.18.5.8 elemtype_

```
DataType ecvl::Image::elemtype_
```

Type of **Image** (p. 64) pixels, must be one of the values available in DataType.

Definition at line 41 of file image.h.

8.18.5.9 mem_

```
MemoryManager* ecvl::Image::mem_
```

Pointer to the **MemoryManager** (p. 82) employed by the **Image** (p. 64).

It can be **DefaultMemoryManager** (p. 59) or **ShallowMemoryManager** (p. 92). The former is responsible for allocating and deallocating data, when using the **DefaultMemoryManager** (p. 59) the **Image** (p. 64) is the owner of data. When **ShallowMemoryManager** (p. 92) is employed the **Image** (p. 64) does not own data and operations on memory are not allowed or does not produce any effect.

Definition at line 92 of file image.h.

8.18.5.10 meta_

```
MetaData* ecvl::Image::meta_
```

Pointer to **Image** (p. 64) **MetaData** (p. 83).

Definition at line 91 of file image.h.

8.18.5.11 strides_

```
std::vector<int> ecvl::Image::strides_
```

Vector of **Image** (p. 64) strides.

```
Strides represent
the number of bytes the pointer on data
has to move to reach the next pixel/voxel
on the correspondent size.
```

Definition at line 46 of file image.h.

The documentation for this class was generated from the following files:

- **image.h**
- **image.cpp**

8.19 ecvl::ImageScalarAddImpl< DT, T > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (**Image** &img, T value, bool saturate)

8.19.1 Detailed Description

template<DataType DT, typename T>
struct ecvl::ImageScalarAddImpl< DT, T >

Definition at line 164 of file arithmetic.h.

8.19.2 Member Function Documentation

8.19.2.1 _()

```
template<DataType DT, typename T >
static void  ecvl::ImageScalarAddImpl< DT, T >::_ (
              Image & img,
             T value,
             bool saturate )  [inline], [static]
```

Definition at line 165 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.20 ecvl::ImageScalarDivImpl< DT, T > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (**Image** &img, T value, bool saturate)

8.20.1 Detailed Description

template<DataType DT, typename T>
struct ecvl::ImageScalarDivImpl< DT, T >

Definition at line 552 of file arithmetic.h.

8.20.2 Member Function Documentation

8.20.2.1 _()

```
template<DataType DT, typename T >
static void  ecvl::ImageScalarDivImpl< DT, T >::_ (
              Image & img,
             T value,
             bool saturate )  [inline], [static]
```

Definition at line 553 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.21 ecvl::ImageScalarMulImpl< DT, T > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (**Image** &img, T value, bool saturate)

8.21.1 Detailed Description

template<DataType DT, typename T>
struct ecvl::ImageScalarMulImpl< DT, T >

Definition at line 431 of file arithmetic.h.

8.21.2 Member Function Documentation

8.21.2.1 _()

```
template<DataType DT, typename T >
static void ecvl::ImageScalarMulImpl< DT, T >::_ (
              Image & img,
             T value,
             bool saturate )  [inline], [static]
```

Definition at line 432 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.22 ecvl::ImageScalarSubImpl< DT, T > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void **_** (**Image** &img, T value, bool saturate)

8.22.1 Detailed Description

template<DataType DT, typename T>
struct ecvl::ImageScalarSubImpl< DT, T >

Definition at line 285 of file arithmetic.h.

8.22.2 Member Function Documentation

8.22.2.1 _()

```
template<DataType DT, typename T >
static void ecvl::ImageScalarSubImpl< DT, T >::_ (
              Image & img,
             T value,
             bool saturate )  [inline], [static]
```

Definition at line 286 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.23 ecvl::Table2D< _StructFun, Args >::integer< i > Struct Template Reference

```
#include <datatype_matrix.h>
```

8.23.1 Detailed Description

template<template< DataType, DataType, typename ... >class _StructFun, typename ... Args>
template<int i>
struct ecvl::Table2D< _StructFun, Args >::integer< i >

Definition at line 83 of file datatype_matrix.h.

The documentation for this struct was generated from the following file:

- **datatype_matrix.h**

8.24 ecvl::SignedTable2D< _StructFun, Args >::integer< i > Struct Template Reference

```
#include <datatype_matrix.h>
```

8.24.1 Detailed Description

template<template< DataType, DataType, typename ... >class _StructFun, typename ... Args>
template<int i>
struct ecvl::SignedTable2D< _StructFun, Args >::integer< i >

Definition at line 120 of file datatype_matrix.h.

The documentation for this struct was generated from the following file:

- **datatype_matrix.h**

8.25 ecvl::Table1D< _StructFun, Args >::integer< i > Struct Template Reference

```
#include <datatype_matrix.h>
```

8.25.1 Detailed Description

template<template< DataType DT, typename ... > class _StructFun, typename ... Args>
template<int i>
struct ecvl::Table1D< _StructFun, Args >::integer< i >

Definition at line 18 of file datatype_matrix.h.

The documentation for this struct was generated from the following file:

- **datatype_matrix.h**

8.26 ecvl::SignedTable1D< _StructFun, Args >::integer< i > Struct Template Reference

```
#include <datatype_matrix.h>
```

8.26.1 Detailed Description

template<template< DataType, typename ... >class _StructFun, typename ... Args>
template<int i>
struct ecvl::SignedTable1D< _StructFun, Args >::integer< i >

Definition at line 50 of file datatype_matrix.h.

The documentation for this struct was generated from the following file:

- **datatype_matrix.h**

8.27 ecvl::Iterator< T > Struct Template Reference

```
#include <iterators.h>
```

Public Types

- typedef **Iterator** &(Iterator::∗ **IncrementMemFn**) ()

Public Member Functions

- **Iterator** (**Image** &img, std::vector< int > pos={})
- **Iterator** & **operator++** ()
- T & **operator** ∗ () const
- T ∗ **operator->** () const
- bool **operator==** (const **Iterator** &rhs) const
- bool **operator!=** (const **Iterator** &rhs) const

Public Attributes

- std::vector< int > **pos_**
- uint8_t ∗ **ptr_**
- **Image** ∗ **img_**
- **IncrementMemFn** **incrementor** = & **Iterator**<T>::IncrementPos

8.27.1 Detailed Description

template<typename T>
struct ecvl::Iterator< T >

Definition at line 12 of file iterators.h.

8.27.2 Member Typedef Documentation

8.27.2.1 IncrementMemFn

```
template<typename T >
typedef Iterator&(Iterator::* ecvl::Iterator< T >::IncrementMemFn) ()
```

Definition at line 17 of file iterators.h.

8.27.3 Constructor & Destructor Documentation

8.27.3.1 Iterator()

```
template<typename T >
Iterator::Iterator (
             Image & img,
             std::vector< int > pos = {} )
```

Definition at line 4 of file image.h.

8.27.4 Member Function Documentation

8.27.4.1 operator *()

```
template<typename T >
T& ecvl::Iterator< T >::operator * ( ) const [inline]
```

Definition at line 22 of file iterators.h.

8.27.4.2 operator"!=()

```
template<typename T >
bool ecvl::Iterator< T >::operator!= (
             const Iterator< T > & rhs ) const [inline]
```

Definition at line 25 of file iterators.h.

8.27.4.3 operator++()

```
template<typename T >
Iterator& ecvl::Iterator< T >::operator++ ( ) [inline]
```

Definition at line 21 of file iterators.h.

8.27.4.4 operator->()

```
template<typename T >
T* ecvl::Iterator< T >::operator-> ( ) const [inline]
```

Definition at line 23 of file iterators.h.

8.27.4.5 operator==()

```
template<typename T >
bool ecvl::Iterator< T >::operator== (
            const Iterator< T > & rhs ) const [inline]
```

Definition at line 24 of file iterators.h.

8.27.5 Member Data Documentation

8.27.5.1 img_

```
template<typename T >
Image* ecvl::Iterator< T >::img_
```

Definition at line 15 of file iterators.h.

8.27.5.2 incrementor

```
template<typename T >
IncrementMemFn ecvl::Iterator< T >::incrementor = & Iterator<T>::IncrementPos
```

Definition at line 18 of file iterators.h.

8.27.5.3 pos_

```
template<typename T >
std::vector<int> ecvl::Iterator< T >::pos_
```

Definition at line 13 of file iterators.h.

8.27.5.4 ptr_

```
template<typename T >
uint8_t* ecvl::Iterator< T >::ptr_
```

Definition at line 14 of file iterators.h.

The documentation for this struct was generated from the following files:

- **iterators.h**
- **image.h**
- **iterators_impl.inc.h**

8.28 ecvl::larger_arithmetic_type< T, U > Struct Template Reference

```
#include <type_promotion.h>
```

Public Types

- using **type** = typename std::conditional_t<(std::numeric_limits< T >::digits< std::numeric_limits< U >↩ ::digits), U, T >

8.28.1 Detailed Description

template<typename T, typename U>
struct ecvl::larger_arithmetic_type< T, U >

Definition at line 12 of file type_promotion.h.

8.28.2 Member Typedef Documentation

8.28.2.1 type

```
template<typename T, typename U>
using ecvl::larger_arithmetic_type< T, U >:: type = typename std::conditional_t<(std↩
::numeric_limits<T>::digits < std::numeric_limits<U>::digits), U, T>
```

Definition at line 15 of file type_promotion.h.

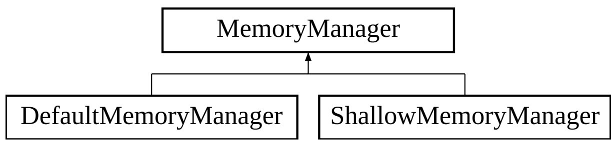
The documentation for this struct was generated from the following file:

- **type_promotion.h**

8.29 MemoryManager Class Reference

```
#include <memorymanager.h>
```

Inheritance diagram for MemoryManager:



Public Member Functions

- virtual uint8_t ∗ **Allocate** (size_t nbytes)=0
- virtual void **Deallocate** (uint8_t ∗data)=0
- virtual uint8_t ∗ **AllocateAndCopy** (size_t nbytes, uint8_t ∗src)=0
- virtual ∼**MemoryManager** ()

8.29.1 Detailed Description

Definition at line 8 of file memorymanager.h.

8.29.2 Constructor & Destructor Documentation

8.29.2.1 ∼MemoryManager()

```
virtual MemoryManager::∼MemoryManager ( ) [inline], [virtual]
```

Definition at line 13 of file memorymanager.h.

8.29.3 Member Function Documentation

8.29.3.1 Allocate()

```
virtual uint8_t* MemoryManager::Allocate (
            size_t nbytes )  [pure virtual]
```

Implemented in **ShallowMemoryManager** (p. 92), and **DefaultMemoryManager** (p. 60).

8.29.3.2 AllocateAndCopy()

```
virtual uint8_t* MemoryManager::AllocateAndCopy (
            size_t nbytes,
            uint8_t * src )  [pure virtual]
```

Implemented in **ShallowMemoryManager** (p. 92), and **DefaultMemoryManager** (p. 60).

8.29.3.3 Deallocate()

```
virtual void MemoryManager::Deallocate (
            uint8_t * data )  [pure virtual]
```

Implemented in **ShallowMemoryManager** (p. 92), and **DefaultMemoryManager** (p. 60).

The documentation for this class was generated from the following file:

- **memorymanager.h**

8.30 ecvl::MetaData Class Reference

```
#include <image.h>
```

Public Member Functions

- virtual bool **Query** (const std::string &name, std::string &value) const =0
- virtual ∼**MetaData** ()

8.30.1 Detailed Description

Definition at line 17 of file image.h.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 ∼MetaData()

```
virtual ecvl::MetaData::∼MetaData ( )  [inline], [virtual]
```

Definition at line 20 of file image.h.

8.30.3 Member Function Documentation

8.30.3.1 Query()

```
virtual bool ecvl::MetaData::Query (
             const std::string & name,
             std::string & value ) const  [pure virtual]
```

The documentation for this class was generated from the following file:

- **image.h**

8.31 ecvl::MulImpl< ST1, ST2 > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const ST1 &src1, const ST2 &src2, **Image** &dst, bool saturate)

8.31.1 Detailed Description

template<typename ST1, typename ST2>
struct ecvl::MulImpl< ST1, ST2 >

Definition at line 450 of file arithmetic.h.

8.31.2 Member Function Documentation

8.31.2.1 _()

```
template<typename ST1 , typename ST2 >
static void  ecvl::MulImpl< ST1, ST2 >::_ (
             const ST1 & src1,
             const ST2 & src2,
              Image & dst,
             bool saturate )  [inline], [static]
```

Definition at line 451 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.32 ecvl::MulImpl< Image, Image > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const **Image** &src1, const **Image** &src2, **Image** &dst, bool saturate)

8.32.1 Detailed Description

template<>
struct ecvl::MulImpl< Image, Image >

Definition at line 483 of file arithmetic.h.

8.32.2 Member Function Documentation

8.32.2.1 _()

```
static void  ecvl::MulImpl<  Image,  Image >::_ (
             const  Image & src1,
             const  Image & src2,
              Image & dst,
             bool saturate )  [inline], [static]
```

Definition at line 484 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.33 ecvl::MulImpl< Image, ST2 > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const **Image** &src1, const ST2 &src2, **Image** &dst, bool saturate)

8.33.1 Detailed Description

template<typename ST2>
struct ecvl::MulImpl< Image, ST2 >

Definition at line 461 of file arithmetic.h.

8.33.2 Member Function Documentation

8.33.2.1 _()

```
template<typename ST2 >
static void ecvl::MulImpl< Image, ST2 >::_ (
             const Image & src1,
             const ST2 & src2,
             Image & dst,
             bool saturate ) [inline], [static]
```

Definition at line 462 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.34 ecvl::MulImpl< ST1, Image > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const ST1 &src1, const **Image** &src2, **Image** &dst, bool saturate)

8.34.1 Detailed Description

template<typename ST1>
struct ecvl::MulImpl< ST1, Image >

Definition at line 474 of file arithmetic.h.

8.34.2 Member Function Documentation

8.34.2.1 _()

```
template<typename ST1 >
static void  ecvl::MulImpl< ST1,  Image >::_ (
             const ST1 & src1,
             const  Image & src2,
              Image & dst,
             bool saturate )  [inline], [static]
```

Definition at line 475 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.35 filesystem::path Class Reference

```
#include <filesystem.h>
```

Public Member Functions

- **path** ()
- **path** (const std::string &p)
- **path** & **operator/=** (const **path** &p)
- **path** & **operator=** (const std::string &s)
- **path** & **operator=** (const **path** &p)
- std::string **string** () const
- **path parent_path** () const
- **path stem** () const

8.35.1 Detailed Description

Definition at line 9 of file filesystem.h.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 path() [1/2]

```
filesystem::path::path ( ) [inline]
```

Definition at line 12 of file filesystem.h.

8.35.2.2 path() [2/2]

```
filesystem::path::path (
            const std::string & p ) [inline], [explicit]
```

Definition at line 14 of file filesystem.h.

8.35.3 Member Function Documentation

8.35.3.1 operator/=()

```
path& filesystem::path::operator/= (
            const path & p ) [inline]
```

Definition at line 20 of file filesystem.h.

8.35.3.2 operator=() [1/2]

```
path& filesystem::path::operator= (
            const std::string & s ) [inline]
```

Definition at line 48 of file filesystem.h.

8.35.3.3 operator=() [2/2]

```
path& filesystem::path::operator= (
            const path & p ) [inline]
```

Definition at line 55 of file filesystem.h.

8.35.3.4 parent_path()

```
path filesystem::path::parent_path ( ) const [inline]
```

Definition at line 66 of file filesystem.h.

8.35.3.5 stem()

```
path filesystem::path::stem ( ) const [inline]
```

Definition at line 80 of file filesystem.h.

8.35.3.6 string()

```
std::string filesystem::path::string ( ) const [inline]
```

Definition at line 61 of file filesystem.h.

The documentation for this class was generated from the following files:

- **filesystem.h**
- **filesystem.cc**

8.36 ecvl::promote_superior_type< T, U > Struct Template Reference

```
#include <type_promotion.h>
```

Public Types

- using **superior_type** = **arithmetic_superior_type_t**< T, U >
- using **type** = typename std::conditional_t<(sizeof(T)==8u||sizeof(U)==8u), double, std::conditional_t< std::is_floating_point< superior_type >::value, superior_type, std::conditional_t<(std::numeric_limits< superior_type >::digits< std::numeric_limits< std::int16_t >::digits), std::int16_t, std::conditional_t<(std::numeric_limits< superior_type >::digits< std::numeric_limits< std::int32_t >::digits), std::int32_t, std::conditional_t<(std::numeric_limits< superior_type >::digits< std::numeric_limits< std::int64_t >::digits), std::int64_t, double > >> >>

8.36.1 Detailed Description

template<typename T, typename U>
struct ecvl::promote_superior_type< T, U >

Definition at line 36 of file type_promotion.h.

8.36.2 Member Typedef Documentation

8.36.2.1 superior_type

```
template<typename T, typename U>
using ecvl::promote_superior_type< T, U >:: superior_type = arithmetic_superior_type_t<T,
U>
```

Definition at line 37 of file type_promotion.h.

8.36.2.2 type

```
template<typename T, typename U>
using ecvl::promote_superior_type< T, U >:: type = typename std::conditional_t<(sizeof(T)
== 8u || sizeof(U) == 8u), double, std::conditional_t<std::is_floating_point<superior_type>↩
::value, superior_type, std::conditional_t<(std::numeric_limits<superior_type>::digits <
std::numeric_limits<std::int16_t>::digits), std::int16_t, std::conditional_t<(std::numeric↩
_limits<superior_type>::digits < std::numeric_limits<std::int32_t>::digits), std::int32_t,
std::conditional_t<(std::numeric_limits<superior_type>::digits < std::numeric_limits<std↩
::int64_t>::digits), std::int64_t, double> >> >>
```

Definition at line 44 of file type_promotion.h.

The documentation for this struct was generated from the following file:

- **type_promotion.h**

8.37 ecvl::ScalarImageDivImpl< DT, T, ET > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (T value, **Image** &img, bool saturate, ET epsilon)

8.37.1 Detailed Description

template<DataType DT, typename T, typename ET>
struct ecvl::ScalarImageDivImpl< DT, T, ET >

Definition at line 571 of file arithmetic.h.

8.37.2 Member Function Documentation

8.37.2.1 _()

```
template<DataType DT, typename T , typename ET >
static void  ecvl::ScalarImageDivImpl< DT, T, ET >::_ (
             T value,
              Image & img,
             bool saturate,
             ET epsilon )  [inline], [static]
```

Definition at line 572 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.38 ecvl::ScalarImageSubImpl< DT, T > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (T value, **Image** &img, bool saturate)

8.38.1 Detailed Description

template<DataType DT, typename T>
struct ecvl::ScalarImageSubImpl< DT, T >

Definition at line 304 of file arithmetic.h.

8.38.2 Member Function Documentation

8.38.2.1 _()

```
template<DataType DT, typename T >
static void  ecvl::ScalarImageSubImpl< DT, T >::_ (
             T value,
              Image & img,
             bool saturate )  [inline], [static]
```

Definition at line 305 of file arithmetic.h.

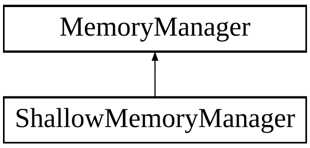
The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.39 ShallowMemoryManager Class Reference

```
#include <memorymanager.h>
```

Inheritance diagram for ShallowMemoryManager:



Public Member Functions

- virtual uint8_t * **Allocate** (size_t nbytes) override
- virtual void **Deallocate** (uint8_t *data) override
- virtual uint8_t * **AllocateAndCopy** (size_t nbytes, uint8_t *src) override

Static Public Member Functions

- static **ShallowMemoryManager** * **GetInstance** ()

8.39.1 Detailed Description

Definition at line 31 of file memorymanager.h.

8.39.2 Member Function Documentation

8.39.2.1 Allocate()

```
virtual uint8_t* ShallowMemoryManager::Allocate (
            size_t nbytes )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 83).

Definition at line 33 of file memorymanager.h.

8.39.2.2 AllocateAndCopy()

```
virtual uint8_t* ShallowMemoryManager::AllocateAndCopy (
            size_t nbytes,
            uint8_t * src )  [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 83).

Definition at line 37 of file memorymanager.h.

8.39.2.3 Deallocate()

```
virtual void ShallowMemoryManager::Deallocate (
            uint8_t * data ) [inline], [override], [virtual]
```

Implements **MemoryManager** (p. 83).

Definition at line 36 of file memorymanager.h.

8.39.2.4 GetInstance()

```
ShallowMemoryManager * ShallowMemoryManager::GetInstance ( ) [static]
```

Definition at line 10 of file memorymanager.cpp.

The documentation for this class was generated from the following files:

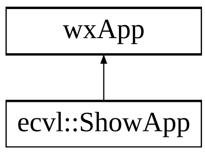
- **memorymanager.h**
- **memorymanager.cpp**

8.40 ecvl::ShowApp Class Reference

ShowApp (p. 93) is a custom wxApp which allows you to visualize an ECVL **Image** (p. 64).

```
#include <gui.h>
```

Inheritance diagram for ecvl::ShowApp:



Public Member Functions

- bool **OnInit** ()
  Initialization function. Starts the main loop of the application.
- **ShowApp** (const **Image** &img)
  Constructor.

8.40.1 Detailed Description

ShowApp (p. 93) is a custom wxApp which allows you to visualize an ECVL **Image** (p. 64).

Definition at line 34 of file gui.h.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 ShowApp()

```
ecvl::ShowApp::ShowApp (
            const  Image & img )  [inline]
```

Constructor.

The constructor creates a **ShowApp** (p. 93) initializing its **Image** (p. 64) with the given input **Image** (p. 64).

Definition at line 52 of file gui.h.

8.40.3 Member Function Documentation

8.40.3.1 OnInit()

```
bool ecvl::ShowApp::OnInit ( )
```

Initialization function. Starts the main loop of the application.

The **OnInit()** (p. 94) function creates a wxFrame which has the width and the height of the **Image** (p. 64) that has to be shown. It also creates the **wxImagePanel** (p. 113) which contains the frame and employs the conversion from **Image** (p. 64) to wxImage. It set the wxImage in the frame and starts the main loop of the **ShowApp** (p. 93).

Definition at line 47 of file gui.cpp.

The documentation for this class was generated from the following files:

- **gui.h**
- **gui.cpp**

8.41 ecvl::SignedTable1D< _StructFun, Args > Struct Template Reference

```
#include <datatype_matrix.h>
```

Classes

- struct **integer**

Public Types

- using **fun_type** = decltype(&_StructFun< static_cast< DataType >(0), Args... >::_)

Public Member Functions

- template< int i >
  constexpr void **FillData** (**integer**< i >)
- constexpr void **FillData** (**integer**< DataTypeSignedSize()>)
- constexpr **SignedTable1D** ()
- **fun_type** **operator()** (DataType dt) const

Public Attributes

- **fun_type** **data** [DataTypeSignedSize()]

8.41.1 Detailed Description

template<template< DataType, typename ... >class _StructFun, typename ... Args>
struct ecvl::SignedTable1D< _StructFun, Args >

Definition at line 45 of file datatype_matrix.h.

8.41.2 Member Typedef Documentation

8.41.2.1 fun_type

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
using  ecvl::SignedTable1D< _StructFun, Args >::  fun_type = decltype(&_StructFun<static_↩
cast<DataType>(0), Args...>::_)
```

Definition at line 47 of file datatype_matrix.h.

8.41.3 Constructor & Destructor Documentation

8.41.3.1 SignedTable1D()

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
constexpr  ecvl::SignedTable1D< _StructFun, Args >::  SignedTable1D ( )  [inline]
```

Definition at line 61 of file datatype_matrix.h.

8.41.4 Member Function Documentation

8.41.4.1 FillData() [1/2]

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
template<int i>
constexpr void  ecvl::SignedTable1D< _StructFun, Args >::FillData (
             integer< i >  )  [inline]
```

Definition at line 53 of file datatype_matrix.h.

8.41.4.2 FillData() [2/2]

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
constexpr void  ecvl::SignedTable1D< _StructFun, Args >::FillData (
             integer< DataTypeSignedSize()>  )  [inline]
```

Definition at line 59 of file datatype_matrix.h.

8.41.4.3 operator()()

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::SignedTable1D< _StructFun, Args >::operator() (
             DataType dt ) const  [inline]
```

Definition at line 65 of file datatype_matrix.h.

8.41.5 Member Data Documentation

8.41.5.1 data

```
template<template< DataType, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::SignedTable1D< _StructFun, Args >::data[DataTypeSignedSize()]
```

Definition at line 69 of file datatype_matrix.h.

The documentation for this struct was generated from the following file:

- **datatype_matrix.h**

8.42 ecvl::SignedTable2D< _StructFun, Args > Struct Template Reference

```
#include <datatype_matrix.h>
```

Classes

- struct **integer**

Public Types

- using **fun_type** = decltype(&_StructFun< static_cast< DataType >(0), static_cast< DataType >(0), Args... >::_)

Public Member Functions

- template<int i>
  constexpr void **FillData** (**integer**< i >)
- constexpr void **FillData** (**integer**< DataTypeSignedSize() ∗DataTypeSignedSize() >)
- constexpr **SignedTable2D** ()
- **fun_type** **operator()** (DataType src, DataType dst) const

Public Attributes

- **fun_type** **data** [DataTypeSignedSize() ∗DataTypeSignedSize()]

8.42.1 Detailed Description

template<template< DataType, DataType, typename ... >class _StructFun, typename ... Args>
struct ecvl::SignedTable2D< _StructFun, Args >

Definition at line 115 of file datatype_matrix.h.

8.42.2 Member Typedef Documentation

8.42.2.1 fun_type

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
using  ecvl::SignedTable2D< _StructFun, Args >::  fun_type = decltype(&_StructFun<static_↩
cast<DataType>(0), static_cast<DataType>(0), Args...>::_)
```

Definition at line 117 of file datatype_matrix.h.

8.42.3 Constructor & Destructor Documentation

8.42.3.1 SignedTable2D()

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
constexpr  ecvl::SignedTable2D< _StructFun, Args >::  SignedTable2D ( )  [inline]
```

Definition at line 133 of file datatype_matrix.h.

8.42.4 Member Function Documentation

8.42.4.1 FillData() [1/2]

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
template<int i>
constexpr void  ecvl::SignedTable2D< _StructFun, Args >::FillData (
             integer< i >  )  [inline]
```

Definition at line 123 of file datatype_matrix.h.

8.42.4.2 FillData() [2/2]

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
constexpr void  ecvl::SignedTable2D< _StructFun, Args >::FillData (
             integer< DataTypeSignedSize() *DataTypeSignedSize() >  )  [inline]
```

Definition at line 131 of file datatype_matrix.h.

8.42.4.3 operator()()

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::SignedTable2D< _StructFun, Args >::operator() (
             DataType src,
             DataType dst ) const  [inline]
```

Definition at line 137 of file datatype_matrix.h.

8.42.5 Member Data Documentation

8.42.5.1 data

```
template<template< DataType, DataType, typename ... >class _StructFun, typename ... Args>
fun_type ecvl::SignedTable2D< _StructFun, Args >::data[DataTypeSignedSize() *DataTypeSigned←
Size()]
```

Definition at line 143 of file datatype_matrix.h.

The documentation for this struct was generated from the following file:

- **datatype_matrix.h**

8.43 ecvl::StructAdd< DT1, DT2 > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (**Image** &src1, const **Image** &src2, bool saturate)

8.43.1 Detailed Description

template<DataType DT1, DataType DT2>
struct ecvl::StructAdd< DT1, DT2 >

Definition at line 144 of file arithmetic.h.

8.43.2 Member Function Documentation

8.43.2.1 _()

```
template<DataType DT1, DataType DT2>
static void ecvl::StructAdd< DT1, DT2 >::_ (
             Image & src1,
             const Image & src2,
             bool saturate ) [inline], [static]
```

Definition at line 145 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.44 ecvl::StructCopyImage< SDT, DDT > Struct Template Reference

```
#include <image.h>
```

Static Public Member Functions

- static void _ (const **Image** &src, **Image** &dst)

8.44.1 Detailed Description

template<DataType SDT, DataType DDT>
struct ecvl::StructCopyImage< SDT, DDT >

Definition at line 551 of file image.h.

8.44.2 Member Function Documentation

8.44.2.1 _()

```
template<DataType SDT, DataType DDT>
static void  ecvl::StructCopyImage< SDT, DDT >::_ (
             const  Image & src,
              Image & dst )  [inline], [static]
```

Definition at line 552 of file image.h.

The documentation for this struct was generated from the following file:

- **image.h**

8.45 ecvl::StructDiv< DT1, DT2, ET > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (**Image** &src1, const **Image** &src2, bool saturate, ET epsilon)

8.45.1 Detailed Description

template<DataType DT1, DataType DT2, typename ET>
struct ecvl::StructDiv< DT1, DT2, ET >

Definition at line 531 of file arithmetic.h.

8.45.2 Member Function Documentation

8.45.2.1 _()

```
template<DataType DT1, DataType DT2, typename ET >
static void  ecvl::StructDiv< DT1, DT2, ET >::_ (
              Image & src1,
             const  Image & src2,
             bool saturate,
             ET epsilon )  [inline], [static]
```

Definition at line 532 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.46 ecvl::StructMul< DT1, DT2 > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (**Image** &src1, const **Image** &src2, bool saturate)

8.46.1 Detailed Description

template<DataType DT1, DataType DT2>
struct ecvl::StructMul< DT1, DT2 >

Definition at line 410 of file arithmetic.h.

8.46.2 Member Function Documentation

8.46.2.1 _()

```
template<DataType DT1, DataType DT2>
static void ecvl::StructMul< DT1, DT2 >::_ (
              Image & src1,
             const Image & src2,
             bool saturate ) [inline], [static]
```

Definition at line 411 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.47 ecvl::StructScalarNeg< DT > Struct Template Reference

Static Public Member Functions

- static **Image** & _ (**Image** &img)

8.47.1 Detailed Description

template<DataType DT>
struct ecvl::StructScalarNeg< DT >

Definition at line 27 of file arithmetic.cpp.

8.47.2 Member Function Documentation

8.47.2.1 _()

```
template<DataType DT>
static Image& ecvl::StructScalarNeg< DT >::_ (
              Image & img ) [inline], [static]
```

Definition at line 28 of file arithmetic.cpp.

The documentation for this struct was generated from the following file:

- **arithmetic.cpp**

8.48 ecvl::StructSub< DT1, DT2 > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (**Image** &src1, const **Image** &src2, bool saturate)

8.48.1 Detailed Description

template<DataType DT1, DataType DT2>
struct ecvl::StructSub< DT1, DT2 >

Definition at line 264 of file arithmetic.h.

8.48.2 Member Function Documentation

8.48.2.1 _()

```
template<DataType DT1, DataType DT2>
static void  ecvl::StructSub< DT1, DT2 >::_ (
               Image & src1,
              const  Image & src2,
              bool saturate )  [inline], [static]
```

Definition at line 265 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.49 ecvl::SubImpl< ST1, ST2 > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const ST1 &src1, const ST2 &src2, **Image** &dst, bool saturate)

8.49.1 Detailed Description

template<typename ST1, typename ST2>
struct ecvl::SubImpl< ST1, ST2 >

Definition at line 323 of file arithmetic.h.

8.49.2 Member Function Documentation

8.49.2.1 _()

```
template<typename ST1 , typename ST2 >
static void ecvl::SubImpl< ST1, ST2 >::_ (
            const ST1 & src1,
            const ST2 & src2,
            Image & dst,
            bool saturate ) [inline], [static]
```

Definition at line 324 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.50 ecvl::SubImpl< Image, Image > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const **Image** &src1, const **Image** &src2, **Image** &dst, bool saturate)

8.50.1 Detailed Description

template< >
struct ecvl::SubImpl< Image, Image >

Definition at line 360 of file arithmetic.h.

8.50.2 Member Function Documentation

8.50.2.1 _()

```
static void ecvl::SubImpl< Image, Image >::_ (
            const Image & src1,
            const Image & src2,
            Image & dst,
            bool saturate ) [inline], [static]
```

Definition at line 361 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.51 ecvl::SubImpl< Image, ST2 > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const **Image** &src1, const ST2 &src2, **Image** &dst, bool saturate)

8.51.1 Detailed Description

template<typename ST2>
struct ecvl::SubImpl< Image, ST2 >

Definition at line 334 of file arithmetic.h.

8.51.2 Member Function Documentation

8.51.2.1 _()

```
template<typename ST2 >
static void  ecvl::SubImpl<  Image, ST2 >::_ (
             const  Image & src1,
             const ST2 & src2,
              Image & dst,
             bool saturate )  [inline], [static]
```

Definition at line 335 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.52 ecvl::SubImpl< ST1, Image > Struct Template Reference

```
#include <arithmetic.h>
```

Static Public Member Functions

- static void _ (const ST1 &src1, const **Image** &src2, **Image** &dst, bool saturate)

8.52.1 Detailed Description

template<typename ST1>
struct ecvl::SubImpl< ST1, Image >

Definition at line 347 of file arithmetic.h.

8.52.2 Member Function Documentation

8.52.2.1 _()

```
template<typename ST1 >
static void  ecvl::SubImpl< ST1,  Image >::_ (
             const ST1 & src1,
             const  Image & src2,
              Image & dst,
             bool saturate )  [inline], [static]
```

Definition at line 348 of file arithmetic.h.

The documentation for this struct was generated from the following file:

- **arithmetic.h**

8.53 ecvl::Table1D< _StructFun, Args > Struct Template Reference

```
#include <datatype_matrix.h>
```

Classes

- struct **integer**

Public Types

- using **fun_type** = decltype(&_StructFun< static_cast< DataType >(0), Args... >::_)

Public Member Functions

- template<int i>
  constexpr void **FillData** (**integer**< i >)
- constexpr void **FillData** (**integer**< DataTypeSize()>)
- constexpr **Table1D** ()
- **fun_type** **operator()** (DataType dt) const

Public Attributes

- **fun_type data** [DataTypeSize()]

8.53.1 Detailed Description

template<template< DataType DT, typename ... > class _StructFun, typename ... Args>
struct ecvl::Table1D< _StructFun, Args >

Definition at line 13 of file datatype_matrix.h.

8.53.2 Member Typedef Documentation

8.53.2.1 fun_type

```
template<template< DataType DT, typename ...  > class _StructFun, typename ...  Args>
using  ecvl::Table1D< _StructFun, Args >::  fun_type = decltype(&_StructFun<static_cast<Data↩
Type>(0), Args...>::_)
```

Definition at line 15 of file datatype_matrix.h.

8.53.3 Constructor & Destructor Documentation

8.53.3.1 Table1D()

```
template<template< DataType DT, typename ...  > class _StructFun, typename ...  Args>
constexpr  ecvl::Table1D< _StructFun, Args >::  Table1D ( )  [inline]
```

Definition at line 29 of file datatype_matrix.h.

8.53.4 Member Function Documentation

8.53.4.1 FillData() [1/2]

```
template<template< DataType DT, typename ...  > class _StructFun, typename ...  Args>
template<int i>
constexpr void  ecvl::Table1D< _StructFun, Args >::FillData (
              integer< i >  )  [inline]
```

Definition at line 21 of file datatype_matrix.h.

8.53.4.2 FillData() [2/2]

```
template<template< DataType DT, typename ...  > class _StructFun, typename ...  Args>
constexpr void  ecvl::Table1D< _StructFun, Args >::FillData (
            integer< DataTypeSize()>  )  [inline]
```

Definition at line 27 of file datatype_matrix.h.

8.53.4.3 operator()()

```
template<template< DataType DT, typename ...  > class _StructFun, typename ...  Args>
fun_type  ecvl::Table1D< _StructFun, Args >::operator() (
            DataType dt ) const  [inline]
```

Definition at line 33 of file datatype_matrix.h.

8.53.5 Member Data Documentation

8.53.5.1 data

```
template<template< DataType DT, typename ...  > class _StructFun, typename ...  Args>
fun_type  ecvl::Table1D< _StructFun, Args >::data[DataTypeSize()]
```

Definition at line 37 of file datatype_matrix.h.

The documentation for this struct was generated from the following file:

- **datatype_matrix.h**

8.54 ecvl::Table2D< _StructFun, Args > Struct Template Reference

`#include <datatype_matrix.h>`

Classes

- struct **integer**

Public Types

- using **fun_type** = decltype(&_StructFun< static_cast< DataType >(0), static_cast< DataType >(0), Args... >::_)

Public Member Functions

- template<int i>
  constexpr void **FillData** (**integer**< i >)
- constexpr void **FillData** (**integer**< DataTypeSize() *DataTypeSize() >)
- constexpr **Table2D** ()
- **fun_type** **operator()** (DataType src, DataType dst) const

Public Attributes

- **fun_type** **data** [DataTypeSize() *DataTypeSize()]

8.54.1 Detailed Description

template<template< DataType, DataType, typename ... >class _StructFun, typename ... Args>
struct ecvl::Table2D< _StructFun, Args >

Definition at line 78 of file datatype_matrix.h.

8.54.2 Member Typedef Documentation

8.54.2.1 fun_type

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
using  ecvl::Table2D< _StructFun, Args >::  fun_type = decltype(&_StructFun<static_cast<Data↩
Type>(0), static_cast<DataType>(0), Args...>::_)
```

Definition at line 80 of file datatype_matrix.h.

8.54.3 Constructor & Destructor Documentation

8.54.3.1 Table2D()

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
constexpr  ecvl::Table2D< _StructFun, Args >::  Table2D ( )  [inline]
```

Definition at line 96 of file datatype_matrix.h.

8.54.4 Member Function Documentation

8.54.4.1 FillData() [1/2]

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
template<int i>
constexpr void  ecvl::Table2D< _StructFun, Args >::FillData (
             integer< i >  )  [inline]
```

Definition at line 86 of file datatype_matrix.h.

8.54.4.2 FillData() [2/2]

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
constexpr void  ecvl::Table2D< _StructFun, Args >::FillData (
             integer< DataTypeSize() *DataTypeSize() >  )  [inline]
```

Definition at line 94 of file datatype_matrix.h.

8.54.4.3 operator()()

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::Table2D< _StructFun, Args >::operator() (
            DataType src,
            DataType dst ) const  [inline]
```

Definition at line 100 of file datatype_matrix.h.

8.54.5 Member Data Documentation

8.54.5.1 data

```
template<template< DataType, DataType, typename ...  >class _StructFun, typename ...  Args>
fun_type  ecvl::Table2D< _StructFun, Args >::data[DataTypeSize() *DataTypeSize()]
```

Definition at line 106 of file datatype_matrix.h.

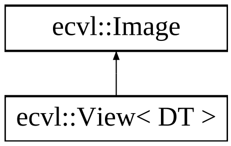
The documentation for this struct was generated from the following file:

- **datatype_matrix.h**

8.55 ecvl::View< DT > Class Template Reference

```
#include <image.h>
```

Inheritance diagram for ecvl::View< DT >:



Public Types

- using **basetype** = typename TypeInfo< DT >:: **basetype**

Public Member Functions

- **View** (**Image** &img)
- **View** (**Image** &img, const std::vector< int > &start, const std::vector< int > &size)
- **basetype** & **operator()** (const std::vector< int > &coords)
- **Iterator**< **basetype** > **Begin** ()
- **Iterator**< **basetype** > **End** ()

Additional Inherited Members

8.55.1 Detailed Description

template<DataType DT>
class ecvl::View< DT >

Definition at line 353 of file image.h.

8.55.2 Member Typedef Documentation

8.55.2.1 basetype

```
template<DataType DT>
using ecvl::View< DT >:: basetype = typename TypeInfo<DT>:: basetype
```

Definition at line 355 of file image.h.

8.55.3 Constructor & Destructor Documentation

8.55.3.1 View() [1/2]

```
template<DataType DT>
ecvl::View< DT >:: View (
              Image & img )  [inline]
```

Definition at line 357 of file image.h.

8.55.3.2 View() [2/2]

```
template<DataType DT>
ecvl::View< DT >:: View (
              Image & img,
             const std::vector< int > & start,
             const std::vector< int > & size )  [inline]
```

Definition at line 374 of file image.h.

8.55.4 Member Function Documentation

8.55.4.1 Begin()

```
template<DataType DT>
Iterator< basetype>  ecvl::View< DT >::Begin ( )  [inline]
```

Definition at line 414 of file image.h.

8.55.4.2 End()

```
template<DataType DT>
Iterator< basetype>  ecvl::View< DT >::End ( )  [inline]
```

Definition at line 415 of file image.h.

8.55.4.3 operator()()

```
template<DataType DT>
basetype& ecvl::View< DT >::operator() (
            const std::vector< int > & coords ) [inline]
```

Definition at line 410 of file image.h.

The documentation for this class was generated from the following file:

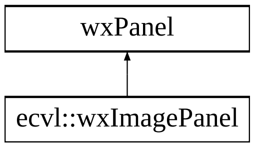
- **image.h**

8.56 ecvl::wxImagePanel Class Reference

wxImagePanel (p. 113) creates a wxPanel to contain an **Image** (p. 64).

```
#include <gui.h>
```

Inheritance diagram for ecvl::wxImagePanel:



Public Member Functions

- **wxImagePanel** (wxFrame ∗parent)
- void **SetImage** (const wxImage &img)

8.56.1 Detailed Description

wxImagePanel (p. 113) creates a wxPanel to contain an **Image** (p. 64).

Definition at line 15 of file gui.h.

8.56.2 Constructor & Destructor Documentation

8.56.2.1 wxImagePanel()

```
ecvl::wxImagePanel::wxImagePanel (
            wxFrame * parent ) [inline]
```

Definition at line 25 of file gui.h.

8.56.3 Member Function Documentation

8.56.3.1 SetImage()

```
void ecvl::wxImagePanel::SetImage (
            const wxImage & img )
```

Definition at line 12 of file gui.cpp.

The documentation for this class was generated from the following files:

- **gui.h**
- **gui.cpp**

Chapter 9

File Documentation

9.1 arithmetic.cpp File Reference

```
#include "ecvl/core/arithmetic.h"
```

Classes

- struct **ecvl::StructScalarNeg**< **DT** >

Namespaces

- **ecvl**

Macros

- #define **STANDARD_INPLACE_OPERATION**(Function, TemplateImplementation)
- #define **STANDARD_NON_INPLACE_OPERATION**(Function)

Functions

- Image & **ecvl::Neg** (Image &img)

  In-place negation of an ***Image*** *(p. 64).*

9.1.1 Macro Definition Documentation

9.1.1.1 STANDARD_INPLACE_OPERATION

```
#define STANDARD_INPLACE_OPERATION(
              Function,
              TemplateImplementation )
```

Value:
```
void Function(Image& src1_dst, const Image& src2)                      \
{                                                                      \
    static constexpr Table2D<TemplateImplementation> table;            \
    table(src1_dst.elemtype_, src2.elemtype_)(src1_dst, src2);         \
}
```

Definition at line 14 of file arithmetic.cpp.

9.1.1.2 STANDARD_NON_INPLACE_OPERATION

```
#define STANDARD_NON_INPLACE_OPERATION(
              Function )
```

Value:
```
void Function(const Image& src1, const Image& src2, Image& dst, DataType dst_type, bool saturate) \
{                                                                                                 \
    if (src1.dims_ != src2.dims_ || src1.channels_ != src2.channels_) {                           \
        throw std::runtime_error("Source images must have the same dimensions and channels.");    \
    }                                                                                             \
                                                                                                  \
    if (!dst.IsOwner()) {                                                                         \
        if (src1.dims_ != dst.dims_ || src1.channels_ != dst.channels_) {                         \
            throw std::runtime_error("Non-owning data destination image must have the             \
                 same dimensions and channels as the sources.");                               \
        }                                                                                         \
    }                                                                                             \
                                                                                                  \
    CopyImage(src1, dst, dst_type);                                                               \
    Function(dst, src2);                                                                          \
}
```

Definition at line 56 of file arithmetic.cpp.

9.2 arithmetic.h File Reference

```
#include <type_traits>
#include "ecvl/core/datatype_matrix.h"
#include "ecvl/core/image.h"
#include "ecvl/core/type_promotion.h"
#include "ecvl/core/standard_errors.h"
```

Classes

- struct **ecvl::StructAdd**< **DT1, DT2** >
- struct **ecvl::ImageScalarAddImpl**< **DT, T** >
- struct **ecvl::AddImpl**< **ST1, ST2** >
- struct **ecvl::AddImpl**< **Image, ST2** >
- struct **ecvl::AddImpl**< **ST1, Image** >
- struct **ecvl::AddImpl**< **Image, Image** >
- struct **ecvl::StructSub**< **DT1, DT2** >
- struct **ecvl::ImageScalarSubImpl**< **DT, T** >
- struct **ecvl::ScalarImageSubImpl**< **DT, T** >
- struct **ecvl::SubImpl**< **ST1, ST2** >
- struct **ecvl::SubImpl**< **Image, ST2** >
- struct **ecvl::SubImpl**< **ST1, Image** >
- struct **ecvl::SubImpl**< **Image, Image** >
- struct **ecvl::StructMul**< **DT1, DT2** >
- struct **ecvl::ImageScalarMulImpl**< **DT, T** >
- struct **ecvl::MulImpl**< **ST1, ST2** >
- struct **ecvl::MulImpl**< **Image, ST2** >
- struct **ecvl::MulImpl**< **ST1, Image** >
- struct **ecvl::MulImpl**< **Image, Image** >
- struct **ecvl::StructDiv**< **DT1, DT2, ET** >
- struct **ecvl::ImageScalarDivImpl**< **DT, T** >
- struct **ecvl::ScalarImageDivImpl**< **DT, T, ET** >
- struct **ecvl::DivImpl**< **ST1, ST2, ET** >
- struct **ecvl::DivImpl**< **Image, ST2, ET** >
- struct **ecvl::DivImpl**< **ST1, Image, ET** >
- struct **ecvl::DivImpl**< **Image, Image, ET** >

Namespaces

- **ecvl**

Functions

- template<DataType ODT, typename IDT >
  TypeInfo< ODT >::basetype **ecvl::saturate_cast** (IDT v)

  Saturate a value (of any type) to the specified type.
- template<typename ODT , typename IDT >
  ODT **ecvl::saturate_cast** (const IDT &v)

  Saturate a value (of any type) to the specified type.
- Image & **ecvl::Neg** (Image &img)

  *In-place negation of an **Image** (p. 64).*
- void **ecvl::Mul** (const Image &src1, const Image &src2, Image &dst, DataType dst_type, bool saturate=true)

  *Multiplies two Image(s) and stores the result in a third **Image** (p. 64).*
- void **ecvl::Sub** (const Image &src1, const Image &src2, Image &dst, DataType dst_type, bool saturate=true)

  *Subtracts two Image(s) and stores the result in a third **Image** (p. 64).*
- void **ecvl::Add** (const Image &src1, const Image &src2, Image &dst, DataType dst_type, bool saturate=true)

  *Adds two Image(s) and stores the result in a third **Image** (p. 64).*
- template<typename ST1 , typename ST2 >
  void **ecvl::Add** (const ST1 &src1, const ST2 &src2, Image &dst, bool saturate=true)

Adds two objects that could be either a scalar value or an ***Image*** *(p. 64), storing the result into a destination* ***Image*** *(p. 64). The procedure does not perform any type promotion.*

- template<typename ST1 , typename ST2 >
  void **ecvl::Sub** (const ST1 &src1, const ST2 &src2, Image &dst, bool saturate=true)

  Subtracts two objects that could be either a scalar value or an ***Image*** *(p. 64), storing the result into a destination* ***Image*** *(p. 64). The procedure does not perform any type promotion.*
- template<typename ST1 , typename ST2 >
  void **ecvl::Mul** (const ST1 &src1, const ST2 &src2, Image &dst, bool saturate=true)

  Multiplies two objects that could be either a scalar value or an ***Image*** *(p. 64), storing the result into a destination* ***Image*** *(p. 64). The procedure does not perform any type promotion.*
- template<typename ST1 , typename ST2 , typename ET = double>
  void **ecvl::Div** (const ST1 &src1, const ST2 &src2, Image &dst, bool saturate=true, ET epsilon=std↩ ::numeric_limits< double >::min())

  Divides two objects that could be either a scalar value or an ***Image*** *(p. 64), storing the result into a destination* ***Image*** *(p. 64). The procedure does not perform any type promotion.*

9.3 core.cpp File Reference

```
#include "ecvl/core/image.h"
```

9.4 core.h File Reference

```
#include "core/arithmetic.h"
#include "core/datatype.h"
#include "core/filesystem.h"
#include "core/image.h"
#include "core/imgcodecs.h"
#include "core/imgproc.h"
#include "core/iterators.h"
#include "core/memorymanager.h"
#include "core/support_opencv.h"
```

9.5 datatype.cpp File Reference

```
#include "ecvl/core/datatype.h"
#include "ecvl/core/datatype_tuples.inc.h"
```

Macros

- #define **ECVL_TUPLE**(name, size, ...) size,

9.5.1 Macro Definition Documentation

9.5.1.1 ECVL_TUPLE

```
#define ECVL_TUPLE(
              name,
              size,
              ...  ) size,
```

9.6 datatype.h File Reference

```
#include <cstdint>
#include <limits>
#include <array>
#include "datatype_tuples.inc.h"
#include "datatype_existing_tuples.inc.h"
#include "datatype_existing_tuples_signed.inc.h"
```

Macros

- #define **ECVL_TUPLE**(name, ...) name,
- #define **ECVL_TUPLE**(name, size, type, ...) template<> struct TypeInfo<ecvl::DataType::name> { using basetype = type; };
- #define **ECVL_TUPLE**(name, ...) + 1
- #define **ECVL_TUPLE**(name, ...) + 1

9.6.1 Macro Definition Documentation

9.6.1.1 ECVL_TUPLE [1/4]

```
#define ECVL_TUPLE(
              name,
              ...  ) name,
```

9.6.1.2 ECVL_TUPLE [2/4]

```
#define ECVL_TUPLE(
              name,
              size,
              type,
              ...  ) template<> struct TypeInfo<ecvl::DataType::name> { using basetype =
type; };
```

9.6.1.3 ECVL_TUPLE [3/4]

```
#define ECVL_TUPLE(
             name,
             ... ) + 1
```

9.6.1.4 ECVL_TUPLE [4/4]

```
#define ECVL_TUPLE(
             name,
             ... ) + 1
```

9.7 datatype_existing_tuples.inc.h File Reference

```
#include "datatype_existing_tuples_signed.inc.h"
#include "datatype_existing_tuples_unsigned.inc.h"
```

9.8 datatype_existing_tuples_signed.inc.h File Reference

9.9 datatype_existing_tuples_unsigned.inc.h File Reference

9.10 datatype_matrix.h File Reference

```
#include "datatype.h"
```

Classes

- struct **ecvl::Table1D< _StructFun, Args >**
- struct **ecvl::Table1D< _StructFun, Args >::integer< i >**
- struct **ecvl::SignedTable1D< _StructFun, Args >**
- struct **ecvl::SignedTable1D< _StructFun, Args >::integer< i >**
- struct **ecvl::Table2D< _StructFun, Args >**
- struct **ecvl::Table2D< _StructFun, Args >::integer< i >**
- struct **ecvl::SignedTable2D< _StructFun, Args >**
- struct **ecvl::SignedTable2D< _StructFun, Args >::integer< i >**

Namespaces

- **ecvl**

9.11 datatype_tuples.inc.h File Reference

```
#include "datatype_existing_tuples.inc.h"
```

9.12 eddll.h File Reference

```
#include <eddll/apis/eddl.h>
#include "ecvl/core/image.h"
```

Namespaces

- **ecvl**

Functions

- Image **ecvl::TensorToImage** (tensor &t)

  *Convert a EDDLL Tensor into an ECVL **Image** (p. 64).*
- tensor **ecvl::ImageToTensor** (const Image &img)

  *Convert an ECVL **Image** (p. 64) into EDDLL Tensor.*
- tensor **ecvl::DatasetToTensor** (vector< string > dataset, const std::vector< int > &dims)

  Convert a set of images into a single EDDLL Tensor.

9.13 filesystem.cc File Reference

```
#include "ecvl/core/filesystem.h"
#include <algorithm>
#include <fstream>
#include <string>
#include <sys/stat.h>
#include <sys/types.h>
```

Namespaces

- **filesystem**

Functions

- bool **filesystem::exists** (const path &p)
- bool **filesystem::exists** (const path &p, error_code &ec)
- bool **filesystem::create_directories** (const path &p)
- bool **filesystem::create_directories** (const path &p, error_code &ec)
- void **filesystem::copy** (const path &from, const path &to)
- void **filesystem::copy** (const path &from, const path &to, error_code &ec)

9.14 filesystem.h File Reference

```
#include <string>
#include <system_error>
```

Classes

- class **filesystem::path**

Namespaces

- **filesystem**

Functions

- path **filesystem::operator/** (const path &lhs, const path &rhs)
- bool **filesystem::exists** (const path &p)
- bool **filesystem::exists** (const path &p, std::error_code &ec)
- bool **filesystem::create_directories** (const path &p)
- bool **filesystem::create_directories** (const path &p, std::error_code &ec)
- void **filesystem::copy** (const path &from, const path &to)
- void **filesystem::copy** (const path &from, const path &to, std::error_code &ec)

9.15 gui.cpp File Reference

```
#include "ecvl/gui.h"
#include "ecvl/core/imgproc.h"
```

Namespaces

- **ecvl**

Functions

- void **ecvl::ImShow** (const Image &img)

    *Displays an **Image** (p. 64).*
- wxImage **ecvl::WxFromImg** (Image &img)

    *Convert an ECVL **Image** (p. 64) into a wxImage.*
- Image **ecvl::ImgFromWx** (const wxImage &wx)

    *Convert a wxImage into an ECVL **Image** (p. 64).*

9.16 gui.h File Reference

```
#include <wx/wx.h>
#include "ecvl/core/image.h"
```

Classes

- class **ecvl::wxImagePanel**

    wxImagePanel *(p. 113) creates a wxPanel to contain an* ***Image*** *(p. 64).*
- class **ecvl::ShowApp**

    ShowApp *(p. 93) is a custom wxApp which allows you to visualize an ECVL* ***Image*** *(p. 64).*

Namespaces

- **ecvl**

Functions

- void **ecvl::ImShow** (const Image &img)

    Displays an ***Image*** *(p. 64).*
- wxImage **ecvl::WxFromImg** (Image &img)

    Convert an ECVL ***Image*** *(p. 64) into a wxImage.*
- Image **ecvl::ImgFromWx** (const wxImage &wx)

    Convert a wxImage into an ECVL ***Image*** *(p. 64).*

9.17 home.h File Reference

9.18 image.cpp File Reference

```
#include "ecvl/core/image.h"
#include "ecvl/core/datatype_matrix.h"
#include "ecvl/core/standard_errors.h"
```

Namespaces

- **ecvl**

Functions

- void **ecvl::RearrangeChannels** (const Image &src, Image &dst, const std::string &channels)

    Changes the order of the ***Image*** *(p. 64) dimensions.*
- void **ecvl::CopyImage** (const Image &src, Image &dst, DataType new_type=DataType::none)

    Copies the source ***Image*** *(p. 64) into the destination* ***Image*** *(p. 64).*

9.19 image.h File Reference

```
#include <algorithm>
#include <numeric>
#include <stdexcept>
#include <vector>
#include <opencv2/core.hpp>
#include "datatype.h"
#include "memorymanager.h"
#include "iterators.h"
#include "iterators_impl.inc.h"
```

Classes

- class **ecvl::MetaData**
- class **ecvl::Image**
  Image *(p. 64) class.*
- class **ecvl::View**< **DT** >
- class **ecvl::ConstView**< **DT** >
- class **ecvl::ContiguousView**< **DT** >
- class **ecvl::ConstContiguousView**< **DT** >
- class **ecvl::ContiguousViewXYC**< **DT** >
- struct **ecvl::StructCopyImage**< **SDT, DDT** >

Namespaces

- **ecvl**

Enumerations

- enum **ecvl::ColorType** {
  ecvl::ColorType::none, **ecvl::ColorType::GRAY**, **ecvl::ColorType::RGB**, **ecvl::ColorType::BGR**, **ecvl::ColorType::HSV**, **ecvl::ColorType::YCbCr** }
  Enum class representing the ECVL supported color spaces.

Functions

- void **ecvl::RearrangeChannels** (const Image &src, Image &dst, const std::string &channels)
  Changes the order of the ***Image*** *(p. 64) dimensions.*
- void **ecvl::CopyImage** (const Image &src, Image &dst, DataType new_type=DataType::none)
  Copies the source ***Image*** *(p. 64) into the destination* ***Image*** *(p. 64).*

9.20 imgcodecs.cpp File Reference

```
#include "ecvl/core/imgcodecs.h"
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include "ecvl/core/support_opencv.h"
```

Namespaces

- **ecvl**

Functions

- bool **ecvl::ImRead** (const std::string &filename, Image &dst)
  Loads an image from a file.
- bool **ecvl::ImRead** (const **filesystem::path** &filename, Image &dst)
- bool **ecvl::ImWrite** (const std::string &filename, const Image &src)
  Saves an image into a specified file.
- bool **ecvl::ImWrite** (const **filesystem::path** &filename, const Image &src)

9.21 imgcodecs.h File Reference

```
#include <string>
#include "image.h"
#include "filesystem.h"
```

Namespaces

- **ecvl**

Functions

- bool **ecvl::ImRead** (const std::string &filename, Image &dst)
  Loads an image from a file.
- bool **ecvl::ImRead** (const **filesystem::path** &filename, Image &dst)
- bool **ecvl::ImWrite** (const std::string &filename, const Image &src)
  Saves an image into a specified file.
- bool **ecvl::ImWrite** (const **filesystem::path** &filename, const Image &src)

9.22 imgproc.cpp File Reference

```
#include "ecvl/core/imgproc.h"
#include <stdexcept>
#include <opencv2/imgproc.hpp>
#include "ecvl/core/datatype_matrix.h"
#include "ecvl/core/standard_errors.h"
```

Namespaces

- **ecvl**

Functions

- void **ecvl::ResizeDim** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< int > &newdims, InterpolationType interp=InterpolationType::linear)

  *Resizes an **Image** (p. 64) to the specified dimensions.*
- void **ecvl::ResizeScale** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< double > &scales, InterpolationType interp=InterpolationType::linear)

  *Resizes an **Image** (p. 64) by scaling the dimentions to a given scale factor.*
- void **ecvl::Flip2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

  *Flips an **Image** (p. 64).*
- void **ecvl::Mirror2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

  *Mirrors an **Image** (p. 64).*
- 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)

  *Rotates an **Image** (p. 64).*
- void **ecvl::RotateFullImage2D** (const **ecvl::Image** &src, **ecvl::Image** &dst, double angle, double scale=1.0, InterpolationType interp=InterpolationType::linear)

  *Rotates an **Image** (p. 64) resizing the output accordingly.*
- void **ecvl::ChangeColorSpace** (const Image &src, Image &dst, ColorType new_type)

  *Copies the source **Image** (p. 64) into destination **Image** (p. 64) changing the color space.*
- void **ecvl::Threshold** (const Image &src, Image &dst, double thresh, double maxval, ThresholdingType thresh_type=ThresholdingType::BINARY)

  *Applies a fixed threshold to an input **Image** (p. 64).*
- double **ecvl::OtsuThreshold** (const Image &src)

  Calculates the Otsu thresholding value.

9.23 imgproc.h File Reference

```
#include "image.h"
#include "support_opencv.h"
```

Namespaces

- **ecvl**

Enumerations

- enum **ecvl::ThresholdingType** { **ecvl::ThresholdingType::BINARY**, **ecvl::ThresholdingType::BINARY_INV** }

  Enum class representing the ECVL threhsolding types.
- enum **ecvl::InterpolationType** {
  ecvl::InterpolationType::nearest, **ecvl::InterpolationType::linear**, **ecvl::InterpolationType::area**, **ecvl::InterpolationType::cubic**,
  ecvl::InterpolationType::lanczos4 }

  Enum class representing the ECVL interpolation types.

Functions

- void **ecvl::ResizeDim** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< int > &newdims, InterpolationType interp=InterpolationType::linear)

    *Resizes an **Image** (p. 64) to the specified dimensions.*
- void **ecvl::ResizeScale** (const **ecvl::Image** &src, **ecvl::Image** &dst, const std::vector< double > &scales, InterpolationType interp=InterpolationType::linear)

    *Resizes an **Image** (p. 64) by scaling the dimentions to a given scale factor.*
- void **ecvl::Flip2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

    *Flips an **Image** (p. 64).*
- void **ecvl::Mirror2D** (const **ecvl::Image** &src, **ecvl::Image** &dst)

    *Mirrors an **Image** (p. 64).*
- 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)

    *Rotates an **Image** (p. 64).*
- void **ecvl::RotateFullImage2D** (const **ecvl::Image** &src, **ecvl::Image** &dst, double angle, double scale=1.0, InterpolationType interp=InterpolationType::linear)

    *Rotates an **Image** (p. 64) resizing the output accordingly.*
- void **ecvl::ChangeColorSpace** (const Image &src, Image &dst, ColorType new_type)

    *Copies the source **Image** (p. 64) into destination **Image** (p. 64) changing the color space.*
- void **ecvl::Threshold** (const Image &src, Image &dst, double thresh, double maxval, ThresholdingType thresh_type=ThresholdingType::BINARY)

    *Applies a fixed threshold to an input **Image** (p. 64).*
- double **ecvl::OtsuThreshold** (const Image &src)

    Calculates the Otsu thresholding value.

9.24 iterators.h File Reference

```
#include <vector>
#include <cstdint>
```

Classes

- struct **ecvl::Iterator**< **T** >
- struct **ecvl::ConstIterator**< **T** >
- struct **ecvl::ContiguousIterator**< **T** >
- struct **ecvl::ConstContiguousIterator**< **T** >

Namespaces

- **ecvl**

9.25 iterators_impl.inc.h File Reference

9.26 memorymanager.cpp File Reference

```
#include "ecvl/core/memorymanager.h"
```

9.27 memorymanager.h File Reference

```
#include <cstdint>
#include <cstring>
#include <stdexcept>
```

Classes

- class **MemoryManager**
- class **DefaultMemoryManager**
- class **ShallowMemoryManager**

9.28 standard_errors.h File Reference

Macros

- #define **ECVL_ERROR_MSG** "[Error]: "
- #define **ECVL_WARNING_MSG** "[Warning]: "
- #define **ECVL_ERROR_NOT_IMPLEMENTED** throw std::runtime_error(**ECVL_ERROR_MSG** "Not implemented");
- #define **ECVL_ERROR_NOT_REACHABLE_CODE** throw std::runtime_error(**ECVL_ERROR_MSG** "How did you get here?");
- #define **ECVL_ERROR_WRONG_PARAMS**(...) throw std::runtime_error(**ECVL_ERROR_MSG** "Wrong parameters - " __VA_ARGS__);
- #define **ECVL_ERROR_NOT_ALLOWED_ON_NON_OWING_IMAGE**(...) throw std::runtime_error(**ECVL_ERROR_MSG** "Operation not allowed on non-owning Image" __VA_ARGS__);
- #define **ECVL_ERROR_UNSUPPORTED_OPENCV_DEPTH** throw std::runtime_error(**ECVL_ERROR_MSG** "Unsupported OpenCV depth");
- #define **ECVL_ERROR_UNSUPPORTED_OPENCV_DIMS** throw std::runtime_error(**ECVL_ERROR_MSG** "Unsupported OpenCV dimensions");
- #define **ECVL_ERROR_EMPTY_IMAGE** throw std::runtime_error(**ECVL_ERROR_MSG** "Empty image provided");
- #define **ECVL_ERROR_NOT_ALLOWED_ON_UNSIGNED_IMG** throw std::runtime_error(**ECVL_ERROR_MSG** "Operation not allowed on unsigned Image");
- #define **ECVL_ERROR_DIVISION_BY_ZERO** throw std::runtime_error(**ECVL_ERROR_MSG** "Division by zero is not allowed.");

9.28.1 Macro Definition Documentation

9.28.1.1 ECVL_ERROR_DIVISION_BY_ZERO

```
#define ECVL_ERROR_DIVISION_BY_ZERO throw std::runtime_error( ECVL_ERROR_MSG "Division by zero
is not allowed.");
```

Definition at line 16 of file standard_errors.h.

9.28.1.2 ECVL_ERROR_EMPTY_IMAGE

```
#define ECVL_ERROR_EMPTY_IMAGE throw std::runtime_error( ECVL_ERROR_MSG "Empty image provided");
```

Definition at line 13 of file standard_errors.h.

9.28.1.3 ECVL_ERROR_MSG

```
#define ECVL_ERROR_MSG "[Error]:  "
```

Definition at line 4 of file standard_errors.h.

9.28.1.4 ECVL_ERROR_NOT_ALLOWED_ON_NON_OWING_IMAGE

```
#define ECVL_ERROR_NOT_ALLOWED_ON_NON_OWING_IMAGE(
              ...  ) throw std::runtime_error( ECVL_ERROR_MSG "Operation not allowed on non-owning
Image" __VA_ARGS__);
```

Definition at line 10 of file standard_errors.h.

9.28.1.5 ECVL_ERROR_NOT_ALLOWED_ON_UNSIGNED_IMG

```
#define ECVL_ERROR_NOT_ALLOWED_ON_UNSIGNED_IMG throw std::runtime_error( ECVL_ERROR_MSG "Operation
not allowed on unsigned Image");
```

Definition at line 14 of file standard_errors.h.

9.28.1.6 ECVL_ERROR_NOT_IMPLEMENTED

```
#define ECVL_ERROR_NOT_IMPLEMENTED throw std::runtime_error( ECVL_ERROR_MSG "Not implemented");
```

Definition at line 7 of file standard_errors.h.

9.28.1.7 ECVL_ERROR_NOT_REACHABLE_CODE

```
#define ECVL_ERROR_NOT_REACHABLE_CODE throw std::runtime_error( ECVL_ERROR_MSG "How did you
get here?");
```

Definition at line 8 of file standard_errors.h.

9.28.1.8 ECVL_ERROR_UNSUPPORTED_OPENCV_DEPTH

```
#define ECVL_ERROR_UNSUPPORTED_OPENCV_DEPTH throw std::runtime_error( ECVL_ERROR_MSG "Unsupported
OpenCV depth");
```

Definition at line 11 of file standard_errors.h.

9.28.1.9 ECVL_ERROR_UNSUPPORTED_OPENCV_DIMS

```
#define ECVL_ERROR_UNSUPPORTED_OPENCV_DIMS throw std::runtime_error( ECVL_ERROR_MSG "Unsupported
OpenCV dimensions");
```

Definition at line 12 of file standard_errors.h.

9.28.1.10 ECVL_ERROR_WRONG_PARAMS

```
#define ECVL_ERROR_WRONG_PARAMS(
              ... ) throw std::runtime_error( ECVL_ERROR_MSG "Wrong parameters - " __VA_ARGS←
__);
```

Definition at line 9 of file standard_errors.h.

9.28.1.11 ECVL_WARNING_MSG

```
#define ECVL_WARNING_MSG "[Warning]:  "
```

Definition at line 5 of file standard_errors.h.

9.29 support_eddll.cpp File Reference

```
#include <ecvl/eddll.h>
#include "ecvl/core/imgproc.h"
#include "ecvl/core/imgcodecs.h"
```

Namespaces

- **ecvl**

Functions

- Image **ecvl::TensorToImage** (tensor &t)

  Convert a EDDLL Tensor into an ECVL ***Image*** *(p. 64).*
- tensor **ecvl::ImageToTensor** (const Image &img)

  Convert an ECVL ***Image*** *(p. 64) into EDDLL Tensor.*
- tensor **ecvl::DatasetToTensor** (vector< string > dataset, const std::vector< int > &dims)

  Convert a set of images into a single EDDLL Tensor.

9.30 support_opencv.cpp File Reference

```
#include "ecvl/core/support_opencv.h"
#include "ecvl/core/standard_errors.h"
```

Namespaces

- **ecvl**

Functions

- **ecvl::Image ecvl::MatToImage** (const cv::Mat &m)

  Convert a cv::Mat into an ***ecvl::Image*** *(p. 64).*
- cv::Mat **ecvl::ImageToMat** (const Image &img)

  Convert an ECVL ***Image*** *(p. 64) into OpenCV Mat.*

9.31 support_opencv.h File Reference

```
#include "image.h"
```

Namespaces

- **ecvl**

Functions

- **ecvl::Image ecvl::MatToImage** (const cv::Mat &m)

  Convert a cv::Mat into an ***ecvl::Image*** *(p. 64).*
- cv::Mat **ecvl::ImageToMat** (const Image &img)

  Convert an ECVL ***Image*** *(p. 64) into OpenCV Mat.*

9.32 test_core.cpp File Reference

```
#include <gtest/gtest.h>
#include "ecvl/core.h"
```

Functions

- **TEST** (Core, CreateEmptyImage)
- **TEST** (Core, CreateImageWithFiveDims)
- **TEST** (ArithmeticNeg, WorksWithInt8)

9.32.1 Function Documentation

9.32.1.1 TEST() [1/3]

```
TEST (
            Core ,
            CreateEmptyImage  )
```

Definition at line 7 of file test_core.cpp.

9.32.1.2 TEST() [2/3]

```
TEST (
            Core ,
            CreateImageWithFiveDims  )
```

Definition at line 14 of file test_core.cpp.

9.32.1.3 TEST() [3/3]

```
TEST (
            ArithmeticNeg ,
            WorksWithInt8  )
```

Definition at line 26 of file test_core.cpp.

9.33 test_eddll.cpp File Reference

```
#include <gtest/gtest.h>
#include "ecvl/eddll.h"
```

9.34 type_promotion.h File Reference

```
#include <limits>
#include <type_traits>
#include "ecvl/core/datatype.h"
```

Classes

- struct **ecvl::larger_arithmetic_type< T, U >**
- struct **ecvl::arithmetic_superior_type< T, U >**
- struct **ecvl::promote_superior_type< T, U >**

Namespaces

- **ecvl**

Macros

- #define **PROMOTE_OPERATION**(op_name, op_symbol)

Typedefs

- template<typename T , typename U >
  using **ecvl::larger_arithmetic_type_t** = typename larger_arithmetic_type< T, U >::type
- template<typename T , typename U >
  using **ecvl::arithmetic_superior_type_t** = typename arithmetic_superior_type< T, U >::type
- template<typename T , typename U >
  using **ecvl::promote_superior_type_t** = typename promote_superior_type< T, U >::type
- template<DataType DT, DataType DU>
  using **ecvl::promote_superior_type_dt** = promote_superior_type_t< TypeInfo_t< DT >, TypeInfo_t< DU > >

Functions

- template<typename T , typename U >
  promote_superior_type_t< T, U > **ecvl::PromoteAdd** (T rhs, U lhs)
- template<typename T , typename U >
  promote_superior_type_t< T, U > **ecvl::PromoteSub** (T rhs, U lhs)
- template<typename T , typename U >
  promote_superior_type_t< T, U > **ecvl::PromoteMul** (T rhs, U lhs)
- template<typename T , typename U >
  promote_superior_type_t< T, U > **ecvl::PromoteDiv** (T rhs, U lhs)

9.34.1 Macro Definition Documentation

9.34.1.1 PROMOTE_OPERATION

```
#define PROMOTE_OPERATION(
            op_name,
            op_symbol )
```

Value:

```
template<typename T, typename U>                                            \
promote_superior_type_t<T, U> Promote ## op_name(T rhs, U lhs) {            \
    using dsttype = promote_superior_type_t<T, U>;                          \
    return static_cast<dsttype>(rhs) op_symbol static_cast<dsttype>(lhs);   \
}
```

Definition at line 53 of file type_promotion.h.

Index